

CHAMPAN & HALL/CRC
TEXTBOOKS IN COMPUTING

КОМПЬЮТЕР ҒЫЛЫМДАРЫН АНЫҚТАУ

Пәнаралық міндеттер,
принциптер және Питон
бағдарламалау тілі

Джессен Хэвилл



«СиЭрСи» баспасы

«Тэйлор энд Фрэнсис Груп»

6000 Брокен Саунд Паркуэй Солтүстік-Батыс, 300 кеңсе, Бока-Ратон,
Флорида 33487-2742

© 2016 ООО «Тэйлор энд Фрэнсис Груп»

«СиЭрСи» баспасы «Тэйлор энд Фрэнсис Груп»

Материалдарын коммерциялық емес қолдану үшін басып шығарды және
түпнұсқалық шығармаларға ешқандай наразылық танытпайды

АҚШ үкіметі

Құрамында қышқылдар жоқ қағазға басып шығарылған. Соңғы редак-
циялау күні:

18.03.2015

Стандартты халықаралық кітап нөмірі 13: 978-1-4822-5414-3 (Мазмұны
– Кітап және Электронды кітап)

Бұл кітап сенімді және білікті мәлімет көздерінен алынған ақпаратты
құрайды. Басылымда берілген мәліметтер мен ақпараттарды сертифика-
циялау мақсатында қажет жүйелі амалдар қолданылды, бірақ автор мен
баспагер барлық материалдар шынайылығы үшін немесе оларды әрі қарай
пайдалануда орын алатын салдар үшін жауапкершілік тарта алмайды.
Автор мен баспашылар осы басылымда пайдаланылған материалдардың
барлық авторлық құқық иегерлерін анықтауға тырысты, және егер басып
шығару үшін рұқсат қажетті формада алынбаған болса, барлық авторлық
құқық иегерлерінен кешірім сұрайды. Егер қандай да бір құқық ныса-
ны бізбен анықталмаған болса, ол туралы бізге хабарласыңыз, өйткені біз
орын алған дәлсіздікті келесі редакцияда түзетуіміз керек.

АҚШ-тың авторлық құқықтар туралы Заңында рұқсат етілген
жағдайлардан басқалары баспагерлердің жазбаша берілген рұқсатынсыз
бұл кітаптың ешбір бөлігін қайта басып шығаруға, көшіруге, қолдан
қолға беруге немесе кез келген электронды, механикалық немесе қазір
белгілі, не болашақта ойлап табылатын фотокөшірме, микрофиль-
мдеу, сондай-ақ, видео және аудио тасымалдаушы сияқты құралдарды
қоса басқа құралдар арқылы немесе кез келген сақтау, не ақпарат іздеу
жүйелерін пайдалану арқылы қолдануға тыйым салынады.

Осы жұмыстың материалдарын электронды түрде қолдануға немесе көшіруге рұқсат алу үшін www.copyright.com (HTTP: // www.copyright.com) сайтына кіріңіз немесе авторлық құқықтарды тексеру Орталығымен хабарласыңыз, 222 Розевуд-Драйв, Дэнверз, Массачусетс 01923, 978-750-8400. Авторлық құқықтарды тексеру – қолданушылардың кең шеңбері үшін тіркеулер мен патенттер беруін қамтамасыз ететін коммерциялық емес ұйым. Авторлық құқықтарды тексеру Орталығымен фотокошірме жасауға рұқсаттық құқық берілген ұйымдар үшін бөлек төлеу жүйесі қарастырылған.

Тауарлық белгі: Тауарлар немесе компаниялар атаулары, тауарлық белгілер не тіркелген тауарлық маркалар болулары мүмкін және авторлық құқықтарды бұзуға қол сұғусыз тек түсіндіру мен теңдестіру мақсаттарында қолданулары тиіс.

Конгресс кітапханасы. Каталогты мәліметтер

Хэвилл, Джессен.

Информатика үйрену: Python бойынша пәнаралық есептер, негіздер мен бағдарламалау/ автор Джессен Хэвилл.

беттер – (информатика бойынша оқулық «ЧЭПМЭН ЭНД Холл»/»СиЭрСи»; 15), кітапханалық сілтемелер мен алфавиттік сілтеуіштерді қоса.

ISBN 978-1-4822-5414-3 (қышқылсыз қағаз)

1. Информатика--Оқулықтар. 2. Python (бағдарламалау тілі) Оқулықтар. I. Тақырыбы.

QA76.H3735 2015

005.13'3--dc23 2015004805

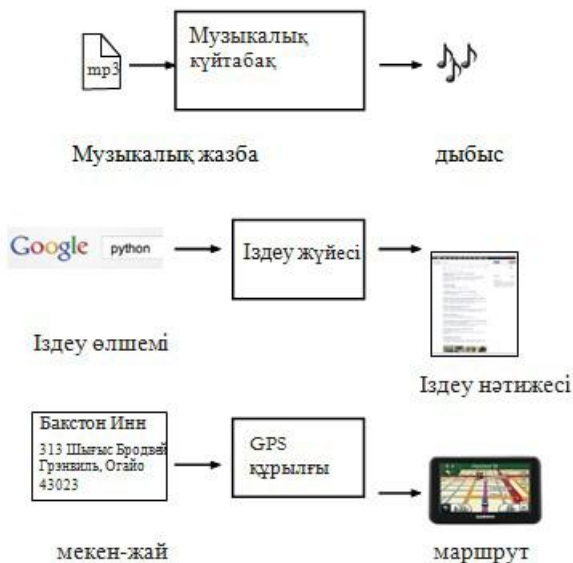
«Тэйлор энд Фрэнсис» <http://www.taylorandfrancis.com>
және «Си Эр Си» баспасының <http://www.crcpress.com> сайтына кіріңіз

Есептеу деген не?

Компьютерлер – қашанда ойлап табылған құрылғылардың ең қуаттысы, әрине тек өздерінің әмбебаптылығы мен тез әрекеттігі үшін ғана емес. Компьютерлер – өте маңызды, себебі олар бізге жаңалықтар енгізуге және теңдесі жоқ ашылулар жасауға мүмкіндік береді.

Компьютер – бұл қарайпайым қадамдар реттілігін есептейтін машина, ол белгілі бір бастапқы ақпаратты, кіру мәліметтерін, (кіру) нәтижесін белгілі бір нәтижеге(шығу) түрлендіреді. Бағдарламашылар компьютерлер мүмкіндіктерін алгоритмдер жасап шығару арқылы күрделі мәселелерді шешу үшін қолданады, ол алгоритмдер есептеулер түрінде берілуі мүмкін. Есептеу нәтижесі ретінде ғарыш кемесінің тиімді маршруты, эпидемияны басқарудың аса тиімді протоколы немесе сандық фотосуретте жасырылған жасырын хабарлама болуы мүмкін. Информатика әрқашан пәнаралық болған, себебі есептерді шығару көзге елестететін кез келген салада орын алады. Әлеуметтанушылар есептеулік үлгілерді, әлеуметтік желілерді, эпидемияларды, демографиялық динамиканы, нарықтарды және аукциондарды жақсырақ түсіну үшін қолданады. Сандық гуманитарлық ғылымдарда жұмыс жасайтын филологтар есептік құралдарды классикалық әдебиетті талдау мен сараптау үшін пайдаланады. Суретшілер өз композициялары мен көріністеріне өте жиі сандық технологияларды қоса бастады. Климат, геномика, қарапайым бөлшектер физикасы, неврология, жаңа дәрілік препараттарды ойлап шығаратын салалардағы есептеу технологиялары бойынша мамандар да сандық технологияларды пайдаланады.

Осы кітапта біз информатикалық есептерді шығарудың терең тәсілдерін зерттейміз, оларды көптеген пәнаралық мәселелерді шешуде және үлгілеуде қалай қолдануға болатындығын анықтаймыз. Осы бірінші тарауда, кітаптың басқа бөлігі құрастырылатын бағыттарды береміз және негіздерді қараймыз. Әрі қарай біз, бұл идеялардың мағынасын кеңінен ашамыз, сондықтан алғашқы кезде барлығы толықтай түсініксіз болса уайымдамаңыз.



1.1 сурет Есептеулердің кейбір мысалдары.

1.1 ЕСЕПТЕР МЕН АБСТРАКЦИЯ

Әр пайда алып келетін есептеу қандай да бір мәселені шешеді. 1.1. суретте көрсетілгендей мәселенің өзара байланысы үшін оның кіріс мәліметтері мен нәтижелеріне сапалы анықтама беруге болады. Әр есеп үшін, бізде сол жақтан енгізетін кіріс мәліметтері және оң жақтан шығатын сәйкес нәтижелер бар. Есептеу арасында кіріс мәліметтері дұрыс нәтижеге түрленеді. Сіз ән тыңдаған кезде, музыкалық күйтабақ сандық музыкалық файлды (кіріс мәліметтерін) дыбыстық образға (нәтижеге) түрлендіреді. Веб-іздеу және Ғаламтор не басқа құралдар арқылы сұраныс (кіріс мәліметтері) жіберуде, сіз сұраған ақпаратты (нәтижені) табу үшін есептеу жасалады. Сіз GPS навигацияны қолданғанда, құрылғы сіздің ағымдағы орналасқан орныңызды қолдана отырып, маршрутты (нәтиже) есептеп шығарады және құрылғыда орнатылған картаға сақтайды (кіріс мәліметтер).

Кіріс мәліметтері мен нәтижелер, мүмкін сізге жоғарғы алгебра курсынан мәлім.

сізге келесідей есеп берілгенде, мысалы

$$y = 18x + 31$$

немесе

$$f(x) = 18x + 31,$$

Сіз, мүмкін, ауыспалы x кіріс мәліметтерді сипаттау үшін, ал y немесе $f(x)$ нәтижені сипаттау үшін қолданған шығарсыз. Бұл мысалда, егер $x = 2$ шығыстағы нәтиже $y = 67$ немесе $f(x) = 67$

болады. X -ті y -ке түрлендіретін арифметикалық әрекет есептеудің өте оңай (және қызықсыз) мысалы 1.1 суретте көрсетілген техниканы қолдану үшін, бізге есептеудің негізін қалаушы кіріс мәліметтер нәтижеге қалай түрленетінін түсіну қажет емес; біз есептеуді «қара жәшік» ретінде көзге елестете аламыз және бұл техниканы әлі күнге дейін қолдана аламыз. Бұл идеяны біз функционалды абстракция деп атаймыз, біз бұл түсінікті өте жиі өзімізбен өзіміз болады деп қабылдаймыз. Басқаша айтқанда, функционалды абстракция құрал немесе технологияны қалай қолдану керек деген білімге ие болмай-ақ, қолдануға мүмкіндік береді.

1.1. Ойлануға арналған сұрақ. *Сізді қандай есептер түрлері қызықтырады? Олардың кіріс мәліметтері мен нәтижелері қандай? Есепті толығымен анықтау үшін, сіз бекіткен кіріс мәліметтер мен нәтижелер жеткілікті ме?*

Біз абстракциялар әлемінде өмір сүреміз; біз оларсыз әрекет ете алмас едік. Біз тіпті өз денелеріміз жайлы абстракциялық терминдерде ойлаймыз. Саусақтарыңызды қозғалтып көріңіз. Сізге осы әрекетті іске асыру үшін жүйке және сүйекті-бұлшық еті жүйелері қалай жұмыс жасайтындығын түсіну қажет пе? Көпшілік үшін автокөлік те абстракция. Автокөлікті жүргізу үшін рөлдің бұрылуы машинаны қалай ұратындығын немесе газ педалін басу оны қалай тезірек жүргізетіндігін сізге түсіну қажет пе? Біз бұл әрекеттерді жасағанда не болатындығын түсінеміз, бірақ олар шын мәнінде қалай іске асатындығы біз үшін қажет емес. Абстракциясыз біз майда жайттар астында қалар едік.

1.2 Ойлануға арналған сұрақ. *Көзге елестетіп көріңізші, GPS құрылғыны қолдану үшін оның қалай жұмыс жасайтындығын түсіну қажет болса. Немесе музыкалық күйтабақ. Немесе компьютер. Осындай жағдай сіздің бұл технологияларды қолдануыңызға қалай ықпал етер еді?*

Жаңа технологиялар мен автоматтандыру күнделікті өмірге жаңа функционалды абстракцияларды енгізді. Біздің азық-түлікті

жеткізулеріміз – осыған жақсы мысал. Небәрі бірнеше ғасырлар бұрын біздің ата-бабаларымыз тамақтың қай жерден келгенін нақты біліп отырған. Ауыр жұмыс пен сәйкес ауа-райы сияқты кіріс мәліметтер жанұяны керегімен қамтамасыз ету үшін астық және үй жануарлары түрінде нәтижелерді өндіреді. Заманауи уақытта кіріс мәліметтері ретінде ақшаны ала отырып, біз қаптамадағы тамақты аламыз; біздің тамақтың өндірілуі біз үшін аса абстракты болып кетті.

1.3 Ойлануға арналған сұрақ. *Сіз үнемі пайдаланатын қарапайым функционалды абстракция жайлы ойлаңыз, мысалы, сіздің телефоныңыз немесе несиелік картаңыз. Осы функционалды абстракция уақыт өте келе қалай өзгерді? Сіз функционалды абстракцияны жақсарту, бізге технологияларды пайдалану қабілеттілігімізді жоғарлататын жағдайларды ойлап таба аласыз ба?*

Сонымен қатар, біз функционалды абстракциялар қабаттарын аса тиімді жұмыс жасау үшін пайдаланамыз. Мысалы, көзіңізге елестетіңіз, с із – бес бөлімнен және 5 000 қызметкерлерден тұратын үлкен ұйымның (университет сияқты) президентісіз.

Сіз осындай үлкен ұйымның барлық элементтерін тиімді басқара алмайтындықтан, с із әр бөлімді бақылау үшін вице-президентті тағайындайсыз. Сіз, әр вице-президент өзінің бөлімінің ортақ қызметі мен тиімділігі жайлы с ізге хабарлай отырып, сі зді күнделікті егжей-текейлерден оқшаулайды деп күтесіз. Осы құрылымда әр бөлім с ізге деген функционалды абстракция болады: Сіз әр бөлім өз жұмысын істеп жатқанын білесіз, бірақ олар қалай жұмыс істейтіндігі маңызды емес. Осы абстракциялардан табыс таба ала отырып, с із енді осы ұйым деңгейінде аса маңызды жұмыстарды атқару үшін өз күш-жігеріңізді шоғырландырып уақыт босатыңыз. Әр вице-президент осыған ұқсас құрылымды өзінің бөлімінде пайдалана алады. Шынымен де, ұйымдар өте жиі өз бөлімдерін майда құрылымдық бірліктерге бөледі, әрине осындай бірліктер бір менеджермен басқаруға мүмкін болғанға дейін.

Сондай-ақ, көптеген функционалды абстракциялар қабаттарда негізделген. Сіз компьютерді қолданғанда, Сіз жұмысты атқару үшін файлдарды сақтап және үстемелерді (яғни бағдарламалар) пайдалануға болатын «үстелдік» абстракцияны аласыз. Көптеген үстемелердің бір уақытта жұмыс жасауы да абстракция болып табылады. Шын мәнінде кейбір үстемелер параллельді орындалуы мүмкін, сол уақытта басқалары – кезекпен, бірақ олар аса тез орындалғандықтан олар параллельді жұмыс

жасап жатқандай сезім пайда болады. Компьютердің барлық қорлық интерфейстері мен негізгі басқарулары (мысалы, қатқыл дисктер, желі, жад, қауіпсіздік) операциялық жүйемен өңделеді. Операциялық жүйе – «әрқашан компьютердің бір шетінде жұмыс жасайтын бағдарлама» ретінде жиі қате сипатталатын *қиын аң*. Шын мәнінде, операциялық жүйенің ядросы компьютерді аса тиімді пайдалануға мүмкіншілік беретін функционалды абстракцияның бірнеше деңгейі болып табылады.

Бағдарлаушылар есептемелік барыстарды құрастырады (мысалы, іздеу жүйелері, GPS бағдарламалық қамсыздандырулары, сонымен қатар операциялық жүйелер), олар сосын басқаларға арналып, функционалды абстракция ретінде қолдану үшін қапталады. Бірақ, біз уақыт өте келе көретіндей, олар сонымен қатар, шын мәнінде бар мәселелерді дұрыс және тиімді шешу үшін абстракцияларды пайдаланады. Бұл мәселелердің қиын болғандығы соншалық, олар адамдар түсініп, шеше алуы үшін кіші мәселелерге бөлінулері тиіс. Әр осындай кіші есептер шешілген бола отырып, бастапқы есепті шешу үшін қолдануға мүмкін болатын функционалды абстракция болып табылады.

Абстракты түсініктерде ойлау қабілеттілігі, сонымен қатар, түрлі салалардағы есептердің ортақ белгілерін көруге мүмкіндік береді. Мысалы, жыныстық көбеюдің (яғни, мутация және рекомбинация) және табиғи сұрыптаудың негізгі операцияларын айқындай отырып, эволюция барысын радномизирлік (*кездейсоқ*) анықтау. Осы түсініктен «эволюциялық есептеу» қабылданған техника мыңдаған есептерге шешім тапқан. Барабар түрде, *босаңдату ұқсастығы* ретінде белгілі әдіс өте күрделі есептер шешімін тиімді іздеу үшін металлдарды баяу суыту барысы жайлы түсінікті қолданады. Басқа мысалдар жәндіктер құжынауы мен тобырының мінез-құлқына, жасушалық мембраналардың жұмыс жасауына және нейрондар жүйесі қалай шешім қабылдайтындығына негізделген әдістерден тұрады

1.2 АЛГОРИТМДЕР ЖӘНЕ БАҒДАРЛАМАЛАР

Әр пайда алып келетін есептеу, кіріс мәліметтерін дұрыс нәтижелерге түрлендіретін реттік қадамдар болып табылады. Бұл реттік қадамдар алгоритм деп аталады. Бұл жағдайды көрнекті көрсету үшін қарапайым есепті қарастырайық: шар көлемін есептеу. Осы мысал үшін кіріс мәліметі шар радиусы r , ал нәтижесі шар радиусының көлемі болады V . Кіріс мәліметтерінен нәтижені есептеп шығару үшін біз жай ғана белгілі

формуланы қолданамыз $V = (4/3)\pi r^3$. Бұл келесідей болады.

радиус $r \rightarrow \boxed{(4/3)\pi r^3} \rightarrow r$ радиусты шар көлемі

Тура төрт бұрышта алгоритм берілген: $4/3$ -ті π және кубтегі радиуска көбейтсек нәтижені аламыз. Дегенмен, формула алгоритм емес, себебі, бұл формуланы есептеп шығару үшін бірнеше баламалық реттік қадамдарды қолданамыз. Мысалы, төменде көрсетілген формулалардың әрқайсысы шар көлемін есептейді.

1. 4-ті 3-ке бөліңіз. 2. Бастапқы нәтижені π көбейтіңіз. 3. Келесі әрекеттерді үш рет қайталаңыз: бастапқы нәтижені r көбейтіңіз.	немесе	1. $r \times r \times r$ есептеңіз. 2. Бастапқы нәтижені π көбейтіңіз. 3. Бастапқы нәтижені 4 көбейтіңіз. 4. Бастапқы нәтижені 3 бөліңіз.

Осы екі алгоритмдер бір формуланы қолданады, бірақ олар қадамдарды басқаша іске асырады. Бұл ерекшеліктер сізге ешқандай мағынасы жоқ болып көрінуі мүмкін, бірақ абстракцияның деңгейіне қарай, компьютермен «сұхбаттасу» кезінде бізге барлығын анық түрде көрсетуге қажет болуы мүмкін.

1.4 Ойлануға арналған сұрақ. *Көлем формуласы туындайтын тағы бір алгоритм жазыңыз.*

Компьютерде есептеу ретінде алгоритмді жазу үшін, біз алгоритмді компьютер «түсінетіндей» тілде жасауымыз қажет. Бұндай компьютерлік тілдер *бағдарламалау тілдері* деп аталады, бағдарламалау тілінде алгоритм жазу *бағдарлама* деп аталады. Бағдарлама толығымен немесе оның бір бөлігін жиі *бастапқы код* немесе жай ғана *код* деп атайды, сондықтан компьютерлік бағдарламалау *кодтау* деген терминмен белгілі. Пакеттік бағдарламалар, мысалы сіз компьютеріңіздің жұмыс үстелінде көріп отырған бағдарламалар да *үстемелер* немесе *apps* деп аталады, ал олардың жиынтығы *бағдарламалық қамсыздандыру* деп аталады (физикалық компьютерлік құрылғыға жататын *аппараттық қамсыздандыруға* қарама-қарсы).

Түрлі бағдарламалау тілдері пайдаланылады, олардың әрқайсысы

өзіндік артықшылықтарға және кемшіліктерге ие. Бұл кітапта *Python* бағдарламалау тілін қолданамыз.

Сізге бұл қандай бағдарлама екендігін көрсету үшін, *Python* бағдарламасын ұсынамыз, ол шар көлемін есептеу алгоритмін жазып көрсетеді:

```
def sphereVolume(radius):  
    volume = (4 / 3) * 3.14159 * (radius **  
    3) return volume  
result =  
sphereVolume(10)  
print(result)
```

Бағдарламадағы әр сызық *нұсқама* деп аталады. Бұл бағдарламадағы бірінші нұсқама *def*-тен басталады және жаңа *функцияны* анықтайды. Алгоритм сияқты функция ол кіріс мәліметтерін нәтижеге түрлендіретін қадамдар реттілігін ұсынады. Бұл жағдайда функция *sphereVolume* деп аталады, ол *radius* деп аталған, функция атауынан кейін домалақ жақшаларда орналасқан кіріс мәліметтері ретінде жалғыз көлемін алады. Екінші жол *sphereVolume* функциясының бір бөлігі екендігін көрсету үшін шегініспен жүреді, ол шар көлемін есептеу үшін формуланы қолданады, сосын осы мәнді *volume* атауына ауыстырады. Үшінші жол *volume* ауысуымен берілген мән функция нәтижесі ретінде «қайтарылған» болуы тиіс. Бұл алғашқы үш жолдар тек *sphereVolume* функциясы не істеу *керектігін* анықтайды; ал төртінші жол шын мәнінде кіріс мәліметтің *sphereVolume* жандандырады, бұл жерде *radius* 10-ға тең, және нәтижені (41.88787) *result* деп аталатын ауыспалыға ауыстырады. Бесінші жол, сіз біліп тұрғандай *result* берген мәнді басып шығарады.

1.2-сурет Есептеулердегі функционалды абстракциялар деңгейлері



Егер бұрын соңды Сіз ешқашан компьютерлік бағдарламаны көрмесеңіз, бұның бәрі мүмкін Сізге «грек» тілі болып көрінер. Бірақ, өміріңіздің бір кезінде $(4/3)\pi r^3$ де сұрақ туындатқан. (π символы дегеніміз не? Арифметикалық әрекеттерді қалайша әріптермен іске асыруға болады? Жоғарыда тұрған кішкене 3 қандай мәнді білдіреді? Бұл жерде «қосу» немесе «көбейту» амалдары жоқ; Бұл негізінде математика ма?) Осының бәрін H_2O немесе 19° жайлы айтуға болады. Бірақ, енді с із осындай айтулармен таныссыз, сондықтан с із « $4/3$ -ті π -ге және кубтағы радиуске көбейту» немесе «екі сутек атомын бір оттегі атомына» деген сияқты айтылуларға қарағанда түсініктірек және сәйкес екендігін бағалай аласыз. Сіз бағдарламалау тілін әдетті арифметикалық жазбаларды жай ғана кеңейту ретінде көзге елестетуіңіз керек. Бірақ, бағдарламалау тілдері алгоритмдерді есептер үшін қарапайым арифметиканың шегінен едәуір алыстауға мүмкіндік береді. Біз жоғарыда сипаттап өткен барыс 1.2 суретінде көрсетілген. Біз нақты белгіленген кіріс мәліметтері және нәтижелері бар есептерден бастаймыз. Сосын алгоритмді жасай отырып есепті шығарамыз. Әрі қарай, біз бағдарламалау тілінде алгоритмді бағдарлама ретінде көрсетіп, оны іске асырамыз. Python сияқты бағдарламалау тілдерін жиі жоғары деңгейлі деп атайды, себебі, «қайталау» және «басып шығару» сияқты таныс сөздерді қолданады,

сонымен қатар, адамға тән есеп шығару тәсілі бар абстракция деңгейін қарастырады. (1.4 Бөлімінде көретіндей, шын мәнінде компьютерлер жоғары деңгейлі бағдарламалық тілді «түсінбейді»). Соңында, компьютерде бағдарламаны *орындау* бізге нәтиже беретін есептеуді бастамалайды. (Бағдарламаны орындау, сонымен қатар, оны «қосу» деп те аталады.) Біз жақын уақытта көретіндей, бұл суретте кейбір бөлшектер жасырылған, бірақ ағымдағы жағдайда бұл сурет бізге жеткілікті.

Енді бұдан да қиын есепті талдауға көшейік. Көзге елестетейік, климаты ағымдағы зерттеу шеңберінде, біз Атлант мұхитында қалқып жүрген бұймен жазылатын судың максималды температурасын күнделікті бақылап отырамыз. Біз жылдық орташа температураны (орташа мән) есептеп шығарғымыз келеді. Елестетейікші, біздің қолда бар максималды температура көрсеткіштері (Цельсия градустарында) келесідей басталады:

18.9, 18.9, 19.0, 19.2, 19.3, 19.3, 19.2, 19.1, 19.4, 19.3, ...

Ойлануға арналған сұрақ 1.5 *Беріліп отырылған есептің кіріс мәліметтері мен нәтижелері қандай? Кіріс мәліметтер үшін қажетті ақпаратты мұқият ойластырыңыз. Өзіңіздің нәтижелеріңізді кіріс мәліметтер терминдерінде жазыңыз.*

Ортақ температура есебі үшін кіріс мәліметтер, шын мәнінде, температура көрсеткіштері жазылған тізімнен тұрады. Сонымен қатар, бізге қолда бар температуралар мәндерінің саны қажет, себебі біз сол санға бөлуіміз керек. Нәтиже – температура көрсеткіштер тізімінің орташа температурасы.

Ойлануға арналған сұрақ 1.6 *Орташа тәуліктік температураны табу үшін біз қандай алгоритмді қолдануымыз қажет? Шар көлемін есептеп шығару үшін қолданған алгоритмдегі қадамдардың реттілігі жағынан қарастырып көріңіз.*

Әрине біз білеміз, бізге барлық температура көрсеткіштерін қосып оларды күн сандарына бөлу қажет. Бірақ, тіпті осындай «барлық температураларды қосу» сияқты қайрапайым бұйрық компьютермен тура орындау үшін аса қиын болып табылады. 1.4 Бөлімінде көретіндей, компьютерлер өз болмысына қарай тек қарапайым арифметикалық бұйрықтарды және бір уақытта екі сандарды стандартты түрде орындай

алады. Біз компьютерге жатқызатын барлық көп күрделілік немесе «интеллект» шын мәнінде адамдарға тиесілі, себебі олар осындай қарапайым бұйрықтар жиынтығына шығармашылық қолдану тапқан. Шыныменде, осы мысалдан компьютерлік алгоритмге екі қажетті ерекшеліктер шығады: олардың қадамдары компьютер үшін *біркелкі* және *орындалмалы* болулары тиіс. Басқаша айтқанда, компьютерлік алгоритмдерден құралған қадамдар, компьютер шын мәнінде іске асыра алатын заттармен ара-қатынасын орнатып, адамнан шығармашылық талдау беруді талап етпеуі тиіс.

Алгоритм, мәселені кірістік мәліметтерді дұрыс нәтижеге түрлендіру жолымен шешетін, орындауға болатын, нақты орындауға болатын ретті нұсқамалар жинағы болып табылады.

Осы екі талаптар, шын мәнінде тек компьютерлік алгоритмдер үшін қойылатын талап емес. Мысалы, біз жаңа хирургиялық әдістер тек нақтылы анатомиялық мәліметтер мен шынайы бар хирургиялық құралдарды қолданады және түсіндіреді деп сенеміз. Дәл осы сияқты сәулетші ғимаратты жобалайды, ол материалдардың қол жетімдігін ескере отырып, оларды орнатуда аса назарлы болуы тиіс. Автор роман жазуда өз аудиториясына назар аударып, сәйкес өзіндік тілді және жергілікті мәдени ерекшеліктерді қолдануы қажет.

«Барлық температура көрсеткіштерін қосу» деген анықтаманы компьютер тілі түсінетін күйге қалайша келтіру болатындығын талдау үшін, біз мұны компьютер не калькулятор жоқ кезде қалай орындайтынымыз жайлы ойлап көрейік: біз өсіп отырған соммаға бір уақытта екі санды қосып отырар едік. Ең бірінші біздің өсетін соммаға бастапқы мән орнатайық, ол 0 тең. Сосын ағымдағы соммаға бірінші температура мәнін қосамыз: $0 + 18.9 = 18.9$. Әрі қарай осы өсіп келе жатқан соммаға екінші мәнді қосамыз: $18.9 + 18.9 = 37.8$. Сосын – температураның үшінші мәнін қосамыз: $37.8 + 19.0 = 56.8$. Және температураның барлық мәндерін қосып болғанша осылай жалғастыра береміз. Елестетейікші, барлық қосылған сомма 8,696.8 тең және біз 365 күнді қосып шықтық (толы жыл үшін емес). Енді біз сомманы күндер санына бөлу арқылы нақты орташа мәнді есептеп шығара аламыз: $8,696.8 / 365 \approx 23.8$. Осы алгоритм үшін бұйрықтар реттік қадамдары келесідей болады:

Кіріс мәліметтер: 365 күндік тәуліктік орташа температуралар көрсеткіш тізімі

1. 0 тең өспелі сомма мәнін бастамалау.
2. Өспелі соммаға температураның бірінші көрсеткішін қосу және өспелі сомманың жаңа мәнін нәтижеге беру.
3. Өспелі соммаға температураның екінші көрсеткішін қосу және өспелі сомманың жаңа мәнін нәтижеге беру.
4. Өспелі соммаға температураның үшінші көрсеткішін қосу және өспелі сомманың жаңа мәнін нәтижеге беру.
- ...
366. Өспелі соммаға 365-ші температура көрсеткішін қосу және өспелі сомманың жаңа мәнін нәтижеге беру.
367. Өспелі сомманы күндер шамасына, 365, бөлу және нәтижеге температураның орташа мәнін беру.

Нәтиже: орташа температура

Бұл 367 қадамнан тұратын алгоритм жазу үшін аса көлемді, ал 2-366 қадамдары болмысы бойынша бірдей болғандықтан, біз алгоритм жазуды 2-266 қадамын алмастыра аламыз.

Біздің тізімізде бар әр температура t үшін келесі әрекеттерді қайталау қажет:

t өспелі соммаға қосып және нәтижеге өспелі сомманың жаңа мәнін беру.

Кезең деп аталатын бұл қысқартылған жазбада, t әр температура көрсеткіші үшін бекітіледі. Алдымен сәйкес нұсқамаөспелісоммағақосатын18.9тізімдегітемператураның бірінші мәніне t беріледі. Сосын өспелі соммаға сәйкес нұсқама қосатын 18.9 температураның екінші мәніне t беріледі. Әрі қарай t температураның үшінші мәніне t беріледі. Және әрі қарай. Осы алмастырып қою (қызылмен ерекшеленген) арқылы біздің алгоритм келесідей болады:

Температураның орташа мән алгоритмі 2

Кіріс мәліметтер: 365 күндік тәуліктік температуралар көрсеткіш тізімі

1. 0 тең өспелі сомма мәнін бастамалау.
2. Біздің тізімнің әр температурасы t үшін келесі әрекеттерді қайталау:
 t өспелі соммаға қосу және өспелі сомманың жаңа мәнін нәтижеге қосу.
3. Өспелі сомманы күндер санына, 365, бөлу және нәтижеге орташа температура мәнін беру.
4. **Нәтиже:** орташа температура

Біз бұл кезеңнің орындалуын оны нұсқаманың теңдей реттілікке аса айқын «ашу» формасында нақты көрсете аламыз. 2 қадаммен анықталатын нұсқама тізімдегі t келесі мәні үшін бір рет іске асырылады. Әр кезде өспелі сомма жаңарады. Сонымен, егер біздің температура тізіміміз өзгерусіз болса, бұл кезең келесі реттік нұсқамаға тең:

- (a) 0 ге қосу және 18.9 тең өспелі сомманың жаңа мәнін беру.

18.9

Υ өспелі сомма ∇ нәтиже ∇

- (ә) -18,9 - га қосу және 37.8 тең өспелі сомманың жаңа мәнін беру.

-18,9 -

Υ өспелі сомма ∇ нәтиже ∇

- (б) 37,8 ге қосу және 56.8 тең өспелі сомманың жаңа мәнін беру.

19.0

Ү өспелі сомма V нәтиже V

Кезеңдік мәннің басқа маңызды артықшылығы, ол температура тізімінің ұзындығына алгоритм мәнін тәуелсіз етуінде. Алгоритмде ортақ температура 1-ге тең екендігін біз алдын ала білуіміз керек болған, себебі, әр температура үшін бізге бір нұсқама қажет. Бірақ, алгоритм үшін орташа температура 2 үшін 365 саны берілмейді, соңғы нұсқаманы есепке алмағанда; кезең температура мәні бар болғанша қайталана береді. Осылайша, біз ортақ температура 2 алгоритмін кез келген температура көрсеткіштерін өңдеу үшін ортақтастыра аламыз. Шын мәнінде, алгоритмде берілген мән температура болып табылуы, не табылмауы маңызды емес, сондықтан біз оны кез келген сандар жинағына қолдану үшін ортақтастырамыз. Егер біз n белгісімен тізім ұзындығын белгілесек, онда біздің ортақтастырылған алгоритміміз келесідей кейіпке енеді:

1. 0-ге тең өспелі сомма мәнін көрсету.
2. Біздің тізімнің әр t саны үшін келесі әрекеттерді қайталау:
 t өспелі соммаға қосу және өспелі сомманың жаңа мәнін нәтижеге қосу.
3. Өспелі сомманы n –ге бөлу және нәтижеге орташа мәнді беру.

Нәтиже: орташа мән **Алгоритм** Орташа мән

Кіріс мәліметтер: n санынан құралған тізім

Біз көріп отырғандай, көп жағдайларда алгоритмді түрлі, бірақ бір мағыналы етіп көрсету тәсілдері бар. Кейде оны таңдау жеке қалаушылықтарға, сондай-ақ, ол соңында өз көрінісін табатын бағдарламалау тіліне тәуелді. Бұл қалай болатындығын көрсету


```
def mean(values):
    n = len(values)
    sum = 0
    for number in values:
        sum = sum +
        number
    mean = sum / n
    return mean
```

Орташа мән алгоритмінің нұсқамасын бағдарлама нұсқамасында жобалауға тырысыңыз. Осылайша жасап, с із бағдарламаның әр бөлігі қандай қызмет атқаратындығы жайлы көрініс ала аласыз. Кейін, әрине біз мұның бәрін нақтылап өтеміз; осыған ұқсас бағдарламалар тараулары үйреншікті жұмыс болып кетеді. Жоғарыда аталып өтілгендей, осындай жазбалар түрімен жұмыс жасап үйренгеннен кейін с ізге, мүмкін, ағылшын тілінде тән нақтылықсыздары бар барлық операторлардың толық жазбаларын жүргізуге қарай бұрынғыдай *аса* күрделі есеп *айтарлықтай* анық болып көрінер.

Жаттығулар

1.2.1. Өзіңіздің күнделікті өміріңізде қашан абстракцияны қолдану тиімді болатын үш мысал келтіріңіз. Абстракциясыз өмірдің қалай болатындығын салыстырмалы түрде сипаттаңыз.

1.2.2. Минималды мән іздеу алгоритмін сандар жинағында жазыңыз. Орташа мән алгоритміндегідей сіз тек бір мәнді бір рет тексере аласыз. Бірақ, ағымдағы өспелі сомманы жаттау орнына ағымдағы орташа мәнді есте сақтаңыз.

1.2.3. Сіз туған күн үшін кеш ұйымдастыратын болдыңыз, ол жерде тәтті тоқаштар беріледі. Шақырылған қонақтар тізімінде бірнеше кішкентай балалар және жастары үлкенірек бірнеше балалар бар. Шақырылған қонақтар тізімі қысқартылған әріптер ретінде берілген. М кішірек жастағы және с үлкенірек жастағы балалар үшін. Әр жасы үлкенірек бала үш тәтті тоқаш жейді, ал әр жасы кіші бала екі тәтті тоқаш жейді.

1.2.4. Тізімге қарай отырып, қанша тәтті тоқаш керектігін санап, алгоритмді жазыңыз. Мысалы, егер кіріс мәліметтер м, м, с, м, с болса, онда нәтиже 12 тәтті тоқашқа тең болар еді.

1.2.5. Қарпайым карталық ойын ойнайтын әр ойынгер үшін алгоритм жазыңыз, мысалы ;Блэк джек немесе Балық аулан. Карталар кез

келген тәртіпте таратылатынын елестетейік.

1.2.6. Осу ретімен құндылығына қарай 5 картаны үймеден реттеу үшін алгоритм жазыңыз. Әр қадамда сіздің алгоритміңіз кез келген екі картаны салыстырып және орын ауыстыра алады.

1.2.7. Екі көршілес орындар арасына маршрут алгоритмін жазыңыз, тек «s қадам арта жасау» және «d градусқа солға бұру» жалғыз ғана командалар бола алады деп жорамалдаңыз, бұл жерде s және d толық сандар.

1.3 ТИІМДІ АЛГОРИТМДЕР

Әр есеп үшін дұрыс шешімді алгоритмдер көп болуы мүмкін. Алайда, барлық алгоритмдер бірдей жақсы бола бермейді. Біз бұл тезистерді екі есептер арқылы көрсетеміз: телефондық құрылымды ұйымдастыру және «шулы» мәлімет көзінен алынған ақпараттарды «реттеу».

«Телефондық ағашты» ұйымдастыру

Елестетіп көріңізші, сіз апатты жағдай орын алған кезде барлығын телефон қоңырауы арқылы хабарлау мүмкіншілігіне ие болу мақсатында өзіңіздің ұйымыңыз үшін, ескі үлгілі «телефонды ағаш» ұйымдастыруды тапсырдыңыз. Сіз бірінші болып қоңырау шалдыңыз, бірақ осыдан кейін басқаларға қоңырау шалуға рұқсат бере аласыз. Мұндай қоңырауларды кім және қандай тәртіпте іске асыруы алгоритм болып табылады. Мысалы, елестетіп көрейік, ұйымда Анна, Борис, Владимир, Геннадий, Дарья, Елена, Жанна, Зинаида атты адамдар саны бар болғаны сегіз. Осындай жағдайда алгоритм қарапайым:

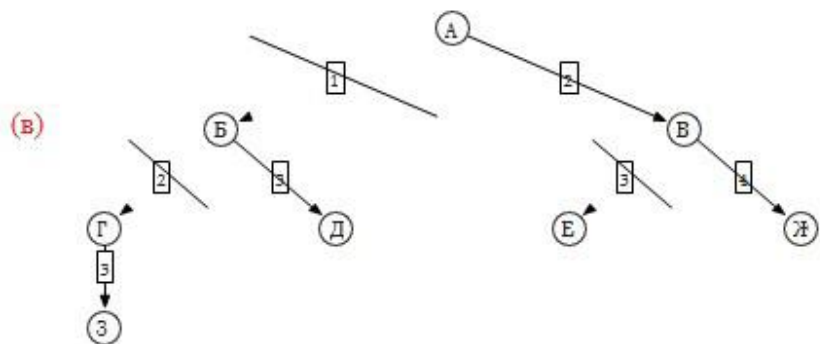
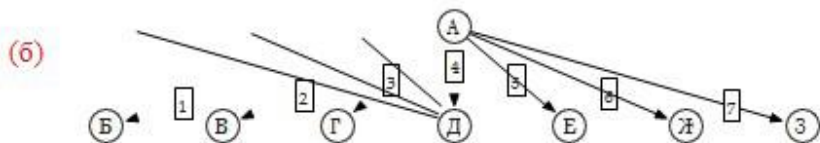
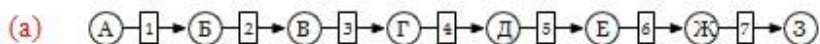
(Қарапайымдылық үшін әр адам телефон қоңырауынан лезде жауап береді және әр телефондық қоңырау бір уақыт көлемін құрайды деп санайық.) Бұл алгоритм есепті шығарудың ең жақсысы болып табыла ма? «Жақсысы» деген сөз не мағына береді?

Алфавитті тәртіптегі телефонды ағаш алгоритмі

1. Анна Бориске қоңырау шалады.
2. Борис Владимирге қоңырау шалады.
3. Владимир Геннадийге қоңырау шалады.
- ...
7. Жанна Зинаидаға қоңырау шалады.

1.7 Ойлануға арналған сұрақ «Телефонды ағаш» есебі үшін басқаға қарағанда қай алгоритм көрсеткіштері жақсы болар еді?

Сұрақ көрсетіп тұрғандай, бұл жерде бір алгоритм басқа алгоритмнен жақсы болуы жайлы шешім қабылдауда.



1.3.- суретінде телефонды ағаш алгоритмдерінің үш мүмкін нұсқасы көрсетілген. Әр есім оның бірінші әріпімен көрсетілген, қоңыраулар сілтемелермен берілген, ал сандар қабылданған қоңыраулар тәртібін меңзейді.

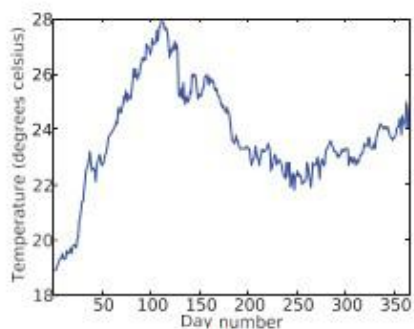
«Телефонды ағаш» есебі үшін, сіздің негізгі ұстанымыңыз ең аз қоңырау шалып, қоңырау шалуларды басқаларға тапсыру болып табылар. Алфавитті тәртіптегі «Телефонды ағаш» алгоритмі 1.3 (а) суретте графикалық түрде берілген, бұл сызба сіздің ұстанымыңызды қанағаттандыруға бағытталған. Керісінше, сіз кінә сезімін сезініп басқаларға қандай да бір жұмысты істетіп, сондықтан қалған қоңырауларды өз еркіңізбен шалуыңыз мүмкін 1. 3 (б) суретінде келтірілген. Бұған қарамастан, ортақ қауіпсіздік шарттарында біз апат-

ты жағдай туралы хабарды соңғы адам тезірек алуы үшін, телефонды ағаш алгоритмін барынша тезірек іске асыруымыз керек. Бұл жағдайда алғашқы екі алгоритм жұмыс жасамады. Шын мәнінде олар ең көп уақыт алатын алгоритмдер болып табылады! Ең жақсы алгоритм бірінші екі адамға қоңырау шалу болып табылады., сосын ол екеулері де сіз сияқты екі адамға қоңырау шалады және басқалары осы сияқты әрі қарай жалғастырады. Бұл 1.3 (в) суретінде келтірілген. Назар аударыңыз, бұл алгоритм бар болғаны төрт қадам арқылы барлығын хабарландырады, себебі бір уақытта бір телефон шалудан артық қоңырау шалынады; А және Б екеуіде екінші кезең кезінде қоңырау шалады; ал Б, В және Г үшінші кезеңде қоңырау шалады.

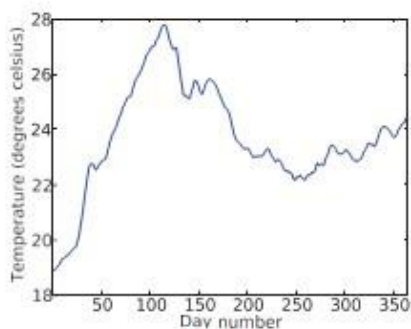
Егер барлық қоңырау шалулар ортақ пайдаланатын ресурс құралы арқылы жасалса (мысалы, желісіз жүйе), бізге мүмкін, уақыт бойынша бір уақытта шалынатын қоңыраулар санын үйлестіру қажет болар еді.

Егер біздің телефонды ағашымызда сегіз адамнан артық болмаған жағдайда, бұл мүмкін аса қиындықтар тудырмас еді, бірақ егер телефонды ағашта сегіз адамнан артық болса, бұл қиындықтарға апарып соғар еді. Мысалы, 1.3 (в)- суретінде келтірілген алгоритмді қолдануда мың адамға бір уақытта мың қоңырау шалулар болар еді.

Бұл сұрақтар ортақ жағдайлардағы алгоритмдерде қалай жұмыс жасайтындығын қарастырып көрейік.



(а)



(б)

1.4- сурет (а) графигінде жылдық температураның күнделікті максималдық көрсеткіштері берілген, ал (б) графигінде бес күндік орташаланған температура көрсеткіші берілген.

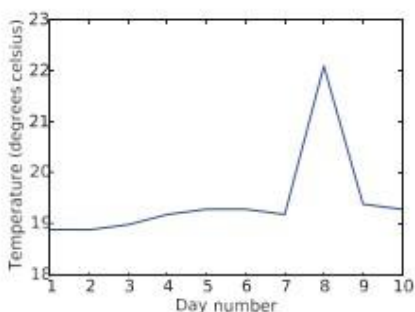
1.8 Ойлануға арналған сұрақ Қандай ортақ көрсеткіштер нақты

мәселе үшін бір алгоритмді, басқа мәселеге қарағанда жақсырақ іске асыра еді?

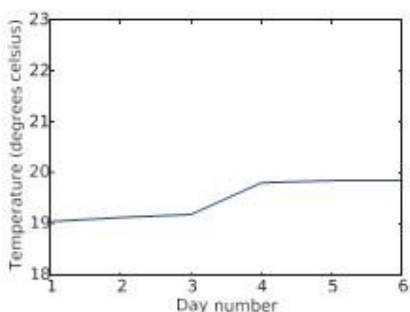
Телефондағаш есебінде берілгендей, жақсы алгоритмді ерекшелейтін көрсеткіш оның жылдамдығы болып табылады; таңдау жасауда біз әрдайым алгоритм орындауға ең аз уақытты қажет етуін қалаймыз. (Веб-іздеуде Сіз не қалар едіңіз: бес минут немесе жарты секунд күту?) Бірақ, осыған қоса, бір алгоритмді екіншіден ерекшелейтін басқа да жақтар бар. Мысалы, біз 1.2 бөлімінде көргендей, шұбалаңқы алгоритмді кезең қолдану жолымен едәуір қысқартуға болады; шоғырландырылған алгоритм түрін жазуға және бағдарламаға аудару оңай. Шоғырландырылған алгоритм түрін компьютер жадының кеңістігінде сақтау үшін аз орын керек. Компьютердегі дискілік кеңістік көлемі шектеулі болғандықтан, біз осы ресурсты анағұрлым аз пайдаланатын алгоритмдер құрастыруға тырысамыз, себебі үлкен көлемді алгоритм алатын орынға тура бұйрықтарды және пайдаланылатын мәліметтерді сақтауға болады. Ресурстарды тиімді пайдалануға желінің өткізу қабілеттілігі жатады, мысалы, желісіз жүйедегі телефонды ағаш есебі. Сонымен қатар, жазушылар мен дизайнерлер сырбаз шығарамаларды жасауға тырысатын сияқты, бағдарлаушылар да түсінуге оңай және қажет емес қадамдардан босатылған сырбаз алгоритмдерді жазуға тырысады. Ал кейбір озат алгоритмдер аса маңызды болып саналады, себебі олар басқа күрделі есептерді шешу үшін жаңа тәсілдерді жасап шығарады.

Тегістеу алгоритмі

Алгоритмді іске асыру үшін қажет уақытты формалды түрде бағалай отыра, алғашқы бөлімде берілген температура реттілігіне қайтадан назар аударайық. Үлкен көлемді мәліметтермен (келтіріп отырылған мысалдағы мәліметтер үлкен) жұмыс жасауда, бізде өте жиі құрылғының сезімталдылығымен, адами факторлармен немесе зертханаға нәтижелер жіберу үшін қолданатын желімен немесе басқа мәліметтерді жинаумен байланысты қателіктермен бетпе-бет келеміз. Біз бұл мәліметтерді «тегістеу» қате өзгерулер арқылы, әр осындай мәнді «терезе» бойынша орташа мәнге ауыстыра отыра жасыра аламыз. Осындай әдістеме, сонымен қатар, шектен шығып кеткен «бүдірлерді» алып тастай отырып, мәліметтердегі ортақ заңдылықтарды алу үшін қолданылады. Мысалы, 1.4- суретінде мұхитта қалқып жүрген буйден алынған өңделмеген жылдық температура мәліметтері көрсетілген, ал жанында бес күндік терезе бойынша тегістелген мәліметтер берілген.



(a)



(б)

1.5- сурет (а) он максималды температура көрсеткіштер графигі және (б) бес күндік терезе бойынша тегістелген температура көрсеткіштері.

Осындай есеп үшін алгоритм жасап шығарайық. Қолданатын есеп пен әдіснаманы жақсырақ түсіну үшін алғашқы бөлімде берілген бірінші он температура көрсеткіштеріне, ауытқулы көрсеткіштерді қоса отырып кішірек мысалды қарастырайық. Сонымен қатар, біз сегізінші орындағы 19.1 °C көрсеткішін 22.1 °C қате температураға ауыстырамыз.

18.9, 18.9, 19.0, 19.2, 19.3, 19.3, 19.2, 22.1, 19.4, 19.3

1.5 (а)- суретінде берілген мәліметтер графигі бұл қате «бұдырлылықты» көрсетеді.

Енді бұл мәліметтерді бес күндік терезе бойынша тегістейік. Бастапқы тізімдегі әр мән үшін оның терезесі элементтің өзін және оның алдында тұрған төрт мәнді өзіне қосады. Біздің алгоритмге осы әр терезенің орташа мәнін есептеуге тура келеді, ал сосын әр осы орташа мәнді жаңа тегістелген тізімге қосу қажет. Соңғы төрт мәнде реттік төрт мәнсіз.

Сондықтан, біздің тегістелген тізім бастапқы тізімге қарағанда төрт мәнге кем болады. Бірінші терезе келесідей болады:

$$\underbrace{18.9, 18.9, 19.0, 19.2, 19.3}_{\text{mean} = 95.3 / 5 = 19.06} \quad 19.3, 19.2, 22.1, 19.4, 19.3$$

Температура терезесінің орташа мәнін табу үшін, бес мағынаны қосып, оны 5-ке бөлеміз. 19.06 нәтижесі бұл терезені тегістелген күйде көрсетеді. Келесі бес терезелер осындай әдіспен есептеледі:

$$18.9, 18.9, \underbrace{19.0, 19.2, 19.3, 19.3, 19.2}_{\text{mean} = 96.0 / 5 = 19.20}, 22.1, 19.4, 19.3$$

$$18.9, 18.9, 19.0, \underbrace{19.2, 19.3, 19.3, 19.2, 22.1}_{\text{mean} = 99.1 / 5 = 19.82}, 19.4, 19.3$$

$$18.9, 18.9, 19.0, 19.2, \underbrace{19.3, 19.3, 19.2, 22.1, 19.4}_{\text{mean} = 99.3 / 5 = 19.86}, 19.3$$

$$18.9, 18.9, 19.0, 19.2, 19.3, \underbrace{19.3, 19.2, 22.1, 19.4, 19.3}_{\text{mean} = 99.3 / 5 = 19.86}$$

Біздің тегістелген температура көрсеткіштерінің толық тізімі келесідей болады:

19.06, 19.14, 19.20, 19.82, 19.86, 19.86

1.5 (б) суретінің тегістелген тізім графигінде көріп отырғандай «бұдыр» шын мәнінде шамалы тегістелген.

Біз өз тегістеу алгоритмімізді алғашқы алгоритмімізге ұқсас белгілеулерді қолдана отырып жаза аламыз. Біздің тегістеу алгоритміміз үшін, терезе бойынша орташа мәнді табу үшін, біз 1.2 бөлімінен орташа мән алгоритмін шақыратынымызға назар аударыңыз.

Тегістеу алгоритмі

Кіріс мәліметтері: n санынан құралған тізім және w терезе көлемі

1. Орташа мән бос тізімін құру.
2. Әр d орын үшін, 1 ден $n - w + 1$ дейін, келесінші қайталау:
 - (a) d және $d + w - 1$ орындары арасында орналасқан сандар тізімі үшін орташа мән алгоритмін шақыру;
 - (b) Нәтижені орташа мәндер тізіміне кіргізу.

Нәтиже: орташа мәндер тізімі

1 -қадам терезелердің орташа мәндерін сақтау үшін бос тізімді қалыптастырады. Есептеуден кейін әр терезе үшін орташа мәнді осы тізім соңында қосамыз. 2- қадам әр терезелер үшін орташа мәнді есептеу үшін кезеңді қолданады. Бұл кезең орташа мән алгоритмінде қолданылатын кезеңге ұқсас: орташа мән алгоритмінің барлық мәндер тізімінде қолданылатын t сияқты d ауыспалысы 1-ден $n - w + 1$ дейін барлық мәндерді қабылдайды. Алдымен d -ға 1 мәні беріледі, 1 және $1 + w - 1 = w$ орындар арасындағы терезе орташа мәні орташа мәндер тізіміне қосылады. Сосын, d -ға 2 мән беріледі, ал 2 және $2 + w - 1 = w + 1$ орындар арасындағы терезенің орташа мәні орташа мәндер тізіміне кіреді. d мәні $n - w + 1$ -ге тең болмағанша, ал $n - w + 1$ және $(n - w + 1) + w - 1 = n$ орындар арасындағы терезенің орташа мәні орташа мәндер тізіміне қосылмағанша осылай әрі қарай жалғаса береді.

1.9 Ойлануға арналған сұрақ Тегістеу алгоритмінің әр қадамын $w = 5$ және біздің берілген мысал тізімінің он элементін орындаңыз.

18.9, 18.9, 19.0, 19.2, 19.3, 19.3, 19.2, 22.1, 19.4, 19.3

Алгоритмнің дұрыс жұмыс жасауын тексеру үшін, әр қадамды біз жоғарыда жасаған қадамдармен салыстырыңыз.

Бұл қанша уақыт алды?

Тізім бойынша мәндерді тегістеу алгоритмі қанша уақыт алатындығын анықтау үшін, біз оны бағдарлама ретінде іске асырып, компьютерге қосуымыз керек. Бірақ, мұндай әдіс кейбір қиындықтарға алып келеді. Біріншіден, уақыт есептеу үшін біз өз экспертменттеріміз үшін қандай тізім(дер)ді қодануымыз керек? Екіншіден, біз w -дің қандай мағынасын қолдануымыз керек? Үшіншіден, біз қандай компьютер(лер)ді пайдалануымыз керек? Біз таңдаған нұсқалар алгоритмнің толық тиімді көрінісін бере ме? Олар бізге, басқа компьютерлердегі басқа тізімдерді тегістеу үшін қажет уақытты болжамдауға мүмкіндік бере ме? Бұл болжамдар шын мәнінде он жылдан кейін әрекетті бола ма?

Алгоритмді орындауға қажет уақытты болжамдаудың ең жақсы тәсілі – қажетті қарапайым қадамдар санын есептеу. Қарапайым қадамдарды іске асыру үшін әрқашан кіріс мәліметтерге тәуелді бірдей уақыт көлемі қажет. Қандай да бір компьютерде алгоритмді орындау қанша уақыт алатындығын қарапайым қадамдар санын есептеу жолымен,осындай

есеп үшін басқа алгоритм қанша уақыт алатындығын бағалай аламыз. Мысалы, А алгоритмі үшін 100 қарапайым қадам қажет болса, ал Б алгоритмі үшін 300 қарапайым қадам қажет болса, біз А алгоритмі, Б алгоритміне қарағанда шамамен үш есе тезірек болатындығын білеміз, оларды қандай компьютерде қосқанға тәуелсіз.

Тегістеу алгоритмінде қандай қадамдарды қарапайым деп атауға болатындығын анықтайық.

1- қадам шарттарды қанағаттандырады, себебі бос тізімді жасау кіріс мәліметтерге назар аударуды талап етпейді. Ал 2- қадам талаптарға сәйкес келмейді, себебі, оның ұзақтылығы кіріс мәліметтерінің бір бөлшегі болып табылатын n және w мәндеріне тәуелді. Осыған ұқсас 2- қадам (а) қарапайым қадамға қойылатын талаптарға сәйкес келмейді, себебі оның ұзақтылығы, өз алдына, w мәніне тәуелді орташа мән алгоритміне қажетті уақытқа тәуелді. Терезе үшін n және w орындар арасында орташа температура мәнінің орналасуы w -ға тең болғанға қарағанда w 10-ға тең келсе көп уақыт алады. Осылайша, орташа мәнді есептеу үшін қажет әр қосу, не бөлу операциясы қарапайым қадам ретінде айқындалады, себебі арифметикалық операцияға қажет уақыт операндтарға тәуелсіз (белгілі бір деңгейде). 2 (б)- қадам да қарапайым деп саналады, себебі тізім соңына бір нәрсені қосу тізімде қанша элементтер бар болуына тәуелсіз болуы тиіс (тізімнің соңына жеткенше дейін).

Ойлануға арналған сұрақ 1.10 *Осындай талдау негізінде бұл алгоритмнің қарапайым қадамдары қандай? Олардың әрқайсысы қанша рет қайталанады?*

Біз анықтаған жалғыз қарапайым қадамдар, - ол тізім қалыптастыру, тізімге қосу және арифметикалық операциялар. Бізге қажетті арифметикалық операциялар санын санайық. Егер әр терезе 5 көлемді болса, онда біз бес қосу операциясын және бір бөлу операциясын әр кез орташа мән алгоритмін шақыра отыра іске асырамыз, яғни, барлығы бір терезеде алты арифметикалық операциялар іске асырылады. Осылайша, арифметикалық операцияларды ортақ саны алтыны терезелер санына көбейткенде шығады. Ортақ жағдайларда, w терезе көлемін білдіреді, ал терезелер саны $n - w + 1$ құрайды. Сондықтан, алгоритм w қосуын және бір терезеге бөлуін іске асырады, яғни ортақ жиынтықта $(w + 1) \cdot (n - w + 1)$ арифметикалық операциялар. Мысалы, бес күндік терезелер көлемді 365 температура көрсеткіштерінен құрастырылған тізімді тегістеу $(w + 1) \cdot (n - w + 1) = 6 \cdot (365 - 5 + 1) = 2,166$ арифметикалық операцияларды

талап етеді. Арифметикалық операцияларға қосымша, біз тізімді бір рет құрастырамыз және бір рет әр терезе үшін тізімге мән қосамыз, ортақ жиынтықта $n - w + 1$ рет. Осылайша, қарапайым қадамдар ортақ саны келесідей:

$$\underbrace{(w + 1) \cdot (n - w + 1)}_{\text{Арифметикалық операциялар}} + \underbrace{1}_{\text{Тізім жасау}} + \underbrace{(n - w + 1)}_{\text{Қосу}}$$

Бұл түсінікті $(w + 2) \cdot (n - w + 1) + 1$ -ге дейін кішкене жеңілдетуге болады.

Бұл қарапайым қадамдар саны алгоритмнің уақытша қиындығы деп аталады. Алгоритмнің уақытша қиындығы әрқашан кіріс мәліметтерінің көлеміне қарай өлшеніп отырады. Берілген жағдайда кіріс мәліметтерінің көлемі n тізімінің ұзындығына және w терезесінің көлеміне тәуелді. Біз кейінірек талқылайтындай, біз бұл уақытша қиындық анықтамасын кіріс мәліметтері аса көп бола бастағанда жеңілдете аламыз. Бірақ, қазір біз бәрін өзгертусіз қалдырамыз.

1.11 Ойлануға арналған сұрақ. *Сіз тегістеу есебін аз көлемді арифметикалық операциялар көмегімен шешетін тәсіл ойлап таба аласыз ба?*

Тегістеудің ең жақсы алгоритмі

Біз аз көлемді қарапайым қадамдарды талап ететін аса тиімді тегістеу алгоритмін жай ғана келесі бақылауларды қолдана отырып жасай аламыз. Тегістеу алгоритмінде көршілес терезелер соммасын таба отырып, біз қажеттіліксіз кейбір қосымша операцияларды бірнеше рет орындадық. Мысалы, біз төртінші және бесінші температура көрсеткіштерін төрт рет қостық. Біз екі көршілес терезелер арасындағы екі соммалар қатынасын қолдана отырып, бұл қосымша жұмысты алып тастай аламыз. Мысалы, бірінші терезеге қолданбалы:

$$\boxed{18.9, 18.9, 19.0, 19.2, 19.3} \quad 19.3, 19.2, 22.1, 19.4, 19.3$$

sum = 95.3

Екінші терезе элементтерінің сомасы бірінші терезедегідей болуы тиіс, себебі олар ортақ төрт элементке ие. Екінші терезе үшін жалғыз ерекшелік, ол 18.9 бірінші элементті жоғалтады және 19.3 ала-

ды. Сондықтан, бізде бірінші терезе элементтерінің соммасы (95,3) болғандықтан, біз екінші терезе элементтер соммасына тек ғана екі қосымша арифметикалық операциялар көмегімен қол жеткіземіз: $95.3 - 18.9 + 19.3 = 95.7$.

18.9, 18.9, 19.0, 19.2, 19.3, 19.3, 19.2, 22.1, 19.4, 19.3
} sum = 95.7

Біз бұл барысты келесі терезелерге де қолдана аламыз. Біз барлық терезелер соммасын алғаннан кейін, алгоритм соңында, біз жай ғана әрқайсысын оның терезе ұзындығын бөле аламыз, бұл орташа мәндердің аяқталған тізімін алу үшін жасалады. Алдыңғы алгоритмге ұқсатып жазылған біздің жаңа тегістеу алгоритміміз келесідей болады:

Тегістеу алгоритмі 2

Кіріс мәліметтер: n санынан алынған тізім және w терезе көлемі

1. Терезелер соммасы бос тізімін жасау.
2. w санының бастапқы соммаларын есептеу және нәтижені терезелер соммасы тізіміне қосу.
3. d әр орны үшін, 2-ден $n - w + 1$ -ге дейін келесіні қайталау:
 - (a) алғашқы терезелер соммасына $d - 1$ орын санын алып тастап, сосын $d + w - 1$ орын санын қосу;
 - (b) нәтижені терезелер соммасы тізіміне қосу.
4. d , 1-ден $n - w + 1$ -ге дейін әр орын үшін келесіні қайталау:
 - (a) d -ны терезе соммасына w -ға бөлу.

Нәтиже: орташа мәндер тізімі

2- қадам бірінші терезе үшін сомманы есептеп, оны терезелер соммасы тізіміне қосады. Сосын, 3- қадам әр келесі реттік терезе үшін алғашқы терезе жоғалтатын сомманы санап, және жаңадан алынатын сомманы қосу арқылы ортақ сомманы есептейді. Соңында, 4- қадам орташа мәндер тізімін алу үшін барлық терезелер соммасын w бөледі.

Ойлануға арналған сұрақ 1.12 Тегістеу- 2 алгоритмінің әр қадамын $w = 5$ үшін орындаңыз және біздің мысалдан он элементті орындаңыз:

18.9, 18.9, 19.0, 19.2, 19.3, 19.3, 19.2, 22.1, 19.4, 19.3

Тегістеу 2 алгоритмін орындау бірінші орындауға кеткен қарапайым қадамдар санына қарағанда шын мәнінде екі есе аз қадамдар санын талап етті ме? Әр қадамды жеке-жеке қарастырайық. Бұрынғыдай, 1- қадам бір қарапайым қадам деп саналады. 2- қадам үшін $w - 1$ қосу операциясы және ортақ шамамен w қарапайым қадамдарын бір рет қосу. 3- қадам қосу, алу және бірінші терезеден басқалары үшін, орта шамамен $3(n - w)$ қарапайым қадамдар қосуды іске асырады. Ең соңында, 4- қадам әр терезе үшін бір бөлу операциясын іске асырады, орта шамамен $n - w + 1$ арифметикалық операциялар.

$$\underbrace{1}_{\text{тізімді кұру}} + \underbrace{w}_{\text{бірінші тере}} + \underbrace{3(n - w)}_{\text{Қалған терезелер}} + \underbrace{n - w + 1}_{\text{деление}}$$

Бұл бізге қарапайым қадамдардың ортақ соммасын береді. Осы шарттардың бәрін ескере отырып, біз $4n - 3w + 2$ тең Тегістеу алгоритмінің уақытша қиындығын табамыз. Осылайша, біздің ескі алгоритм

Бұл бізге қарапайым қадамдардың ортақ соммасын береді. Осы шарттардың бәрін ескере отырып, біз $4n - 3w + 2$ тең Тегістеу алгоритмінің уақытша қиындығын табамыз. Осылайша, біздің ескі алгоритм

$$\frac{(w + 2) \cdot (n - w + 1) + 1}{4n - 3w + 2}$$

Жаңа алгоритмге қарағанда екі есе операциларды талап етеді. Бұл бөлшектен біздің жаңа алгоритм ескі алгоритмге қарағанда шамамен $w/4$ есе тезірек екендігін анықтау қиын. Бұған көз жеткізу үшін, елестетіп көрейік, біздің тізім $n = 10,000$ температура көрсеткіштерінен тұрады. Келесі кестеде w терезе көлемін көбейтуде мәнінің өлшемі көрсетілген.

w	Өсу реті
5	1,7
10	3.0
20	5,5
100	25,4
500	123,9
1,000	243,7

w аз болған жағдайда, жаңа алгоритм ескі алгоритмнен артықшылығы жоқ, бірақ егер w өссе айырмашылық аса маңызды болады. Тегістеудің шынайы шарттарында миллиардты немесе триллионды элементтерден тұратын аса үлкен мәліметтер жазуда терезелер көлемі $w = 100,000$ дейін жетуі мүмкін. Мысалы, тегістеуді көбінесе ұзындығы миллиардты элементтен тұратын ДНҚ реттілік статистикалық мәліметтерін визуалдау қолданады. Осылайша, біздің нақтыланған алгоритм елеулі ықпал ете алады! Шыныменде, біз әрі қарай 6.4 бөлімінде қарастыратындай, аса тез алгоритм есептеу уақытын аппараттық қамсыздандыруға қарағанда едәуір қысқарта алады.

Жаттығу

1.3.1. 1.3 (в)- суретінде көрсетілген телефонды ағаш алгоритмі, барлық қоңырау шалуларды үш қадамда жасауға жақын. Шын мәнінде әрекетті үш қадаммен аяқтауға бола ма? Қалайша?

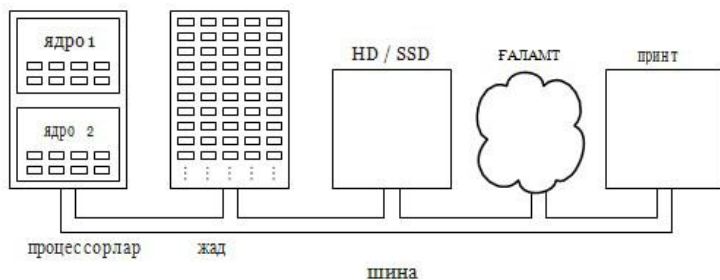
1.3.2. Болжап көрейік, 1.3 (в)- суретінде көрсетілген телефонды ағаш алгоритмі көп адам санымен іске қосылды. Бір қоңырау шалу бірінші уақыт аралығында, екі қоңырау шалу екінші және үшінші қоңырау шалу үшінші уақыт аралықтарында іске асырылғанын көріп отырмыз.

(а) 4,5 және 6 уақыт аралықтарында қанша бір уақытты қоңырау шалулар болады?

(б) Ортақ жағдайда, сіз, кез келген уақыт аралықтарында шалынған қоңыраулар санын анықтауға қажет формуланы (немесе алгоритмді) көзіңізге елестете аласыз ба?

1.3.3. Алгоритм қаншалықты жақсы екендігін бағалау үшін ең маңызды өлшемді сипаттаңыз. Сосын, жоқ дегенде бір өлшем қосыңыз және кейбір күнделікті мәселелерге арналған алгоритмдерге осы өлшемдерді қалай қолдануға болатындығын сипаттаңыз.

1.3.4. n (кіріс мәліметтер тізім көлемі) арқылы көрсетілген, 10 бетте берілген Орташа мән алгоритмінің уақытша қиындығы қандай?



1.6- сурет. Компьютердің тұрлаулы жеңілдетілген сызбасы.

1.4 АҚЫМАҚ КОМПЬЮТЕРЛЕР

Компьютерлер ақылды алгоритмдерді өте тез атқаратындықтан, кейде олар анайы ақыл көрсететін сияқты болып көрінуі мүмкін. Тарихи мысал ретінде, 1997 жылы IBM Deep Blue компьютері алты ойыннан тұратын шахмат жарысында әлем чемпионы Гарри Каспаровты жеңді. Кейірінек, 2011 жылы, IBM Watson компьютері Jeopardy теле ойынында екі чемпионды жеңді.

Бірақ, біздің компьютер-ақыл деп есептеуіміз, шын мәнінде, түпкі түрде, өз көрінісін алгоритм формасында, ал сосын бағдарлама түрінде тапқан адам ақыл-ойы. Олар компьютер ең басында атқара алатын аса жеңіл қалдықты командалар жинағына негізделген, *машиналық тіл* деп аталатын, жоғары деңгейлі бағдарламалау нұсқаулықтары шын мәнінде тек абстракты ыңғайлылықтар болып табылады.

Python бағдарламасында барлық нұсқамалар компьютерде орындалар алдында, машиналық тілде, баламалы реттілікке командаға аударылулары тиіс. Бұл барыс автоматты түрде іске асырылады, және белгілі бір деңгейде айқын, *интерпретатор* немесе *компилятор* деп аталатын бағдарламамен. Интерпретатор жоғары деңгейлі бағдарламаның бір тармағын машиналық тілге аударып орындайды, сосын келесі тармақты аударып қарайды және әрі қарай. Компилятор бұл әрекеттердің орнына бағдарламаны жоғары деңгейлік тілден машиналық тілге толығымен аударды. Осыдан кейін машиналық тілмен компилденген бағдарламалар басынан аяғына дейін қосымша аудармасыз жасалады. Сондықтан, компилденген бағдарламалар түсіндіргіш бағдарламаларға қарағанда тезірек. Алайда, түсіндіру тілдері бағдарламамен жұмыс жасауда

олармен тығыз әрекет етуге мүмкіндік береді. 2 тарауда көрсетілгендей, Python бағдарламалаудың түсіндіруші тілі болып табылады.

Компьютер ішінде

Машиналық тілден тұратын командалар түрлері заманауи компьютер ішкі құрылымына негізделген. 1.6.- суретте көрсетілгендей, шын мәнінде, компьютер *жадпен* байланысты бір немесе бірнеше процессордан тұрады. Жиі RAM (қысқаша *random access memory* – оперативтік жад) деп аталатын компьютерлік жад тұжырымдамалы күйде ұзын ретті *ұяшықтар* түрінде жасалады, ол ұяшықтың әрқайсысы бір бірлік ақпаратты құрайды.

Әр ұяшыққа ерекше *жад мекен-жайы* беріледі, бұл процессордың осы мекен-жайға сілтеме жасауға мүмкіндік береді. Осылайша, компьютер оперативтік жадын бірдей көлемді үлкен пошталық жәшіктермен салыстыруға болады, олардың әрқайсысында бір хаттан сақтауға болады. Әр пошта жәшігі жад мекен-жайына барабар, ал хат – ақпарат бірлігіне барабар. Әр ұяшықтағы ақпарат бір командадан, не мәліметтер бірлігінен тұрады¹. Осылайша, компьютер оперативтік жады мәліметтермен жұмыс жасайтын бағдарлама сияқты.

Жиі *CPU* (қысқартылғанда *central processing unit* – орталық процессор) деп аталатын процессор немесе *ядро*, командаларды орындауға арналған және ағымдағы командаларды орындауға қажетті мәліметтер мәндерін уақытша сақтауға арналған *тіркеуіш* деп аталатын кішігірім жад ұяшықтар жиынтығынан тұратын механизмді құрайды. Егер, көптеген заманауи компьютерлердегідей, компьютерде бір ядродан артық ядро болса, онда ол бір уақытта бір командадан артық командалар орындауға қабілетті. Бұл командалар бір бағдарламадан және түрлі бағдарламалардан тұруы мүмкін. Бұл біздің 7 беттегі *реттілік* түрдегі командалар күйінде берілген алгоритм анықтамасы қатаң дұрыс болып саналмайтындығын білдіреді. Шын мәнінде, алгоритм (немесе бағдарлама) ағымдар деп аталатын бірнеше жартылай автономды реттілік қадамдарды құрауы мүмкін, бұл ағымдар есепті қосыла шешеді.

1.6- суретте көрсетілгендей, процессор мен жад *шина* деп аталатын арнамен жалғанған. Процессорге жадтан мән қажет болғанда, ол шина бойынша сұраныс жібереді, ал жад сосын сұраныс жасалған тәсілмен

¹ Шын мәнінде, әр команда немесе мәліметтер шығыры бірнеше көріндіс ұяшықтар. Шина, сондай-ақ, процессор мен жадты кейбір

басқа құрамдаушылармен жалғайды, бұл әрекет аппараттық тиімділікті немесе оның ыңғайлығын жоғарлатады, мысалы, ғаламтор немесе қатқыл диск, қатты жинақтаушылар мен флэш-жад сияқты екінші сақтау жүйелері. Сізге мәлім болғандай, компьютер қуаттануы тоқтатылғанда, компьютер оперативтік жадының құрамы жойылады, сондықтан, ұзақ уақыт аралығында мәліметтерді сақтау үшін екінші сақтау жүйелерін пайдаланыңыз. Біз бұл құрылғылармен «файлдық жүйелер» - абстракциясы арқылы әрекет етеміз, олар біздің қатқыл дискілеріміз шын мәнінде – құжаттар сақтауға арналған сөрелер деген жалған көрініс береді. Сіз бағдарламаны атқаруда немесе ашқанда, ол алдымен процессор рұқсат алатын жадтағы екінші сақтау жүйесінен көшіреді.

Машиналық тіл

Процессор атқаратын машиналық тіл командалары өте оңай. Мысалы, $x = 2 + 5$ есебі сияқты оңай бір әрекетті қарастырайық. Бағдарламада бұл нұсқама 2 және 5 қосып, нәтижені ауыспалы x сілтеме жасайтын ұяшық жадына сақтайды. Бірақ бұның өзі де, машиналық тіл бір командасына тым қиын болады. Компьютер түріне қарай машиналық тілдегі балама, мүмкін, төрт командадан тұрады. Бірінші команда 2 мәнін процессор тіркеуішіне жүктейді. Екінші команда 5 мәнін процессордың басқа тіркеуішіне жүктейді. Үшінші команда осы екі тіркеуіш мәндерін қосып, нәтижені үшінші тіркеуішке сақтайды.

Ескертпе 1.1: Жоғары өндірістік есептеулер

Заманауи компьютерлер аса жылдам болғанымен, қосымша қуаттылықты қажет ететін кейбір өте үлкен көлемді есептер бар. Олар ауа-райын болжаудан, молекулярлы үлгілеуден, емдік препараттарды жасап шығарудан, аэродинамикалық есептемелерден және шифрламаларды шифрін анықтаудан тұрады. Осындай есептер үшін ғылымдар *суперкомпьютерлерді* пайдаланады. Суперкомпьютер немесе аса жоғары өндірістік компьютер аппарат топтарын тораптар арасында жалғайтын өте жылдам желіден тұратын мыңдаған *тораптар* жинағы. Торап, негізінде, бөлек тұратын компьютер, ол бірнеше ядролық процессорлардан құрылуы мүмкін. Бүгінгі таңда жылдам суперкомпьютерлер миллион ядролы және миллион гигабайт жадқа ие.

Бұл компьютерлер қуаттылығын іске асыру үшін бағдарламашылар осындай компьютерлер ядроларына үнемі тоқтаусыз мәліметтер мен командаларды бағыттап отырулары тиіс. Бөлек ядролармен есептелінген нәтижелер толық қамды шешімге біріктіріледі. *Параллельді бағдарламалау* деген әлпін мәлім алгоритм үлгісі мен бағдарламалау тәсілдері осы кітап шеңберінен шығады, бірақ егер сіз олар туралы көбірек білгіңіз келсе, біз тарау соңында қосымша ақпарат береміз.

Қорытындай келе, төртінші команда үшінші тіркеуіш мәнін x мекен-жайлы жад ұяшығына сақтайды.

Компьютер қосылғаннан бастап, оның процессорлары *сауалнама мен кезең орындау* (немесе *машиналық кезең*) деп аталатын үздіксіз тұйық барыс ішінде жұмыс жасайды. Әр кезең үшін процессор жадтан машиналық тілде бір команда алып, оны орындайды. Бұл кезең компьютер сөндірілгенше немесе қуатын жоғалтқанша жалғасы береді. Қысқаша айтқанда, компьютермен жасалатын әрекеттердің бәрі осы. Компьютер атқаратын сұраныс пен орындайтын кезең шамалы жылдамдығы ішкі сағаттың «тықылдау» (процессордың *ырғақтық жиілігімен*) жиілігімен анықталады. Бұл сағаттардың тықылдауы машиналық операциялардың реттік үйлесімділігін қолдайды. Заманауи дербес компьютерлер сағаттары секундына бірнеше миллиард рет тықылдайды; 3 гигагерцтік процессор (GHz) секундына 3 миллиард рет тықылдайды; («гига» - «миллиард», ал «герц» - секундтағы бірлікті білдіреді).

Осылайша, шын мәнінде өздерінің базалық деңгейінде компьютерлер айтарлықтай ақымақ; процессор ұсыныс артынан ілесе береді және біз оған тапсырған кез келген қарапайым командалар реттілігін орындай отырып кезеңді іске асырады. Компьютердің бізді тітіркендіретін қателіктері шын мәнінде адаммен жасалған. Компьютерлердің керемет қабілеті олардың ақылдығында емес, біздің командаларды тез орындауында; олар қысқа мерзімде өте үлкен жұмыс көлемін атқаруда.

Барлығы - биттерде

Біздің талқылауымыз әлі ең маңызды сұрақты қарастырмады: компьютер қандай формада мәліметтер мен бағдарламаларды сақтайды? Машиналық тілдегі командаларға қоса біз сандарды, құжаттарды, карталарды, суреттерді, дыбыстарды, таныстыру рәсімдерді, электронды кестелерді және көптеген басқа заттарды сақтауымыз қажет. Әр ақпарат түрі үшін түрлі ақпарат тасушыларды пайдалану аса күрделі, әрі тиімді емес болар еді.

Оның орнына бізге кез келген мәліметтерді сақтауға бейімделінген қарапайым сақтау пішімі керек. Бұған жауап – биттер. *Бит* – екі мәннің тек біреуін қабылдай алатын ақпараттық ең кіші бірлігі: 0 немесе 1 (немесе өшіру/қосу, із/жоқ не өтірік/шын баламалары). Бұл қарапайымдылық ақпаратты сақтау (яғни жад) мен оны есептеуді (мысалы процессор) біршама оңай және тиімді етеді.

Биттер ол ауыстырып-қосқыштар

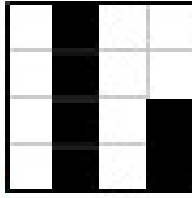
Бит мәнін сақтау өте қарапайым: бит – бұл қосулы/өшірулі ауыстырып-қосқыш баламасы, ал бит мәні ауыстырып-қосқыш күйінің баламасы: $\text{ӨШІРУЛІ} = 0$ және $\text{ҚОСУЛЫ} = 1$. Негізінде, компьютер жады, жад ұяшықтарына біріктірілген миллиардты микроскопиялық ауыстырып-қосқыштардан тұрады. Әр жад ұяшығы 8 ауыстырып-қосқыштан тұрады, сондықтан әр ұяшық *байт* деп аталатын 8 битті сақтай алады. Біз байтты жай ғана 8 биттен тұратын реттілік ретінде қабылдаймыз, мысалы 01101001. 8 гигабайтты (ГБ) компьютер жады шамамен 8 миллиард жад ұяшықтарынан тұрады, олардың әрқайсысы бір байтты сақтайды. (Дәл осы сияқты, килобайт (КБ) шамамен бір мың байттан, мегабайт (МБ) бір миллион байттан, ал терабайт (ТБ) – шамамен триллион байттан тұрады).

Биттер кез келген ақпаратты сақтай алады

Ақпараттарды биттермен сақтаудың екінші артықшылығы – олар «ең кем ортақ бөлгіш» ретінде, ең қарапайым мүмкін ақпарат шығыры болып табылады: барлық қалған ақпараттарды биттерге түрлендіру өте қарапайым түрмен іске асырылады. Машиналық тілдегі сандар, сөздер, суреттер, музыка мен командалар – сақталар алдында биттер ретілігі ретінде кодталады. Сонымен қатар, биттер түрінде сақталатын ақпаратты екінші жазба деп те атайды. Мысалы, 16 биттен (немесе 2 байттан) тұратын келесі реттілікті қарастырып көрейік: 0100010001010101

Бұл биттік реттілік қалай түсіндірілуіне қарай келесі мағыналардың кез келгенін білдіруі мүмкін:

- (a) Мақсатты мән 17,493;
- (b) Ондық мән 2,166015625;
- (c) “DU” екі символы;
- (d) команду `inc x` командасын Intel машиналық тілінде (`inc` – «өсіру» сөзінің қысқартпасы); немесе
- (e) көрсету ерітіндісі деп аталатын 4×4 келесі ақ-қара сурет (0- ол ақ шаршы, ал 1- қара шаршы).



Ал енді нақтылап қарастырайық (а). Компьютерде толық сандар *бинарлы санау жүйесі* көмегімен берілген, оны ондық есептеу жүйесімен ұйқастыра отырып жақсырақ түсінуге болады. Ондық жүйеде әр реттегі мән, онның деңгейі болып табылатын мәнге ие: оңнан солға дейін $10^0 = 1$, $10^1 = 10$, $10^2 = 100$ және тағы басқа мәндер. Ондық сан мәні алдымен олардың мәндеріне көбейтілген сандарды қосу арқылы алынады. Осылайша, 1,831 ондық саны келесі мәнді береді

$$1 \times 10^3 + 8 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 = 1000 + 800 + 30 + 1.$$

Екілік санау жүйесінің ерекшелігі әр позиция екінің деңгейін білдіреді және онның позициясынан екі сан ғана бар. Оңнан солға қарай, екілік жүйе санының позициясы $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$ және тағы басқа мәнге ие. Осылайша, мысалы, онды есептеу жүйесіндегі 110011 екілік саны келесі мәнді

$$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 32 + 16 + 2 + 1 = 51$$

16 биттік сан жоғары, 0100010001010101, ондық есептілік жүйеде $2^{14} + 2^{10} + 2^6 + 2^4 + 2^2 + 2^0 = 16,384 + 1,024 + 64 + 16 + 4 + 1 = 17,493$ баламалы.

1.13 Ойлануға арналған сұрақ. *Өзіңіздің қалай түсінгеніңізді тексеру 1001000 екілік саны неге ондық 72 санына баламалық екендігін дәлелдеңіз.*

Бұл идея, сондай-ақ бөлшектік санға таралуы мүмкін. Ондық жүйеде ондық нүктеден таралатын оң позициялардың 10 теріс деңгейі болып табылатын: ондық позиция $10^{-1} = 0.1$ мәніне ие, жүздік позиция $10^{-2} = 0.01$ мәніне ие және сол сияқты. Осылайша, 18.31 ондық саны келесі мәнді береді

$$1 \times 10^1 + 8 \times 10^0 + 3 \times 10^{-1} + 1 \times 10^{-2} = 10 + 8 + 0.3 + 0.01.$$

Тура осылайша, екілік жүйеде «екілік нүктеден» сол жаққа тараған позициялар 2 теріс деңгейі болып табылатын мәнге ие.

Мысалы, ондық есептеу жүйесіндегі 11.0011 екілік саны келесі мәнді береді

$$1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} = 2 + 1 + 0 + 0 + \frac{1}{8} + \frac{1}{16}$$

Бинарлық есептеулер – бұл оңай

Екілік жүйенің артықшылығы екілік мәннен есептеу өте оңай болуында. Шын мәнінде тек үш негізгі операторлар бар, **ЖӘНЕ** (and), **НЕМЕСЕ** (or) и **ЖОҚ** (not). Олар 9-шы ғасыр математигі Джордж Бульдің арқасына *бульдік операторлар* ретінде әйгілі болды, ол енді *бульдік логика* немесе *бульдік алгебра* деп аталатын заманауи математикалық логиканың негізін қалаушы болып саналады.

Логикалық операторлар мағынасын ашып көрсету үшін ауыспалы a және b 0 немесе 1 мәнді биталар болып табылатындығын елестетейік. Егер a және b 1-ге тең болған жағдайда ғана a **ЖӘНЕ** b 1-ге тең; кері жағдайда a **ЖӘНЕ** b 0.² тең. Мұны төменде берілген *шындықтар кестесі* түрінде қарастыруға ыңғайлы:

a	b	a и b
0	0	0
0	1	0
1	0	0
1	1	1

Шындықтар кестесінің әр жолы a және b мәндердің түрлі үйлесімдерін көрсетеді. Бұл үйлесімдер бірінші екі бағанада берілген. Шындықтар кестесінің соңғы бағанасы a **ЖӘНЕ** b мәндеріне сәйкес келеді. Біз көріп отырғандай екеуі a және b 0-ге тең болған жағдайларды қоспағанда a **ЖӘНЕ** b барлық жағдайлар үшін 0-ге тең болады. Егер біз 1-ді «шындық», ал 0-ді «өтірік» ретінде қабылдасақ, мұны біздің «**ЖӘНЕ**» операторын интуитивті к түсінігімізбен салыстыруға ыңғайлы болады. «Қамба қызыл және ак» деген пайымдау егер онда: қызыл және ак екі түстің болуында ғана дұрыс.

a немесе b ауыспалыларының біреуі 1-ге тең болған жағдайда, екінші

оператор a **НЕМЕСЕ** b 1-ге тең болады; кері жағдайда a **НЕМЕСЕ** b 0-ге тең болады. Бұл келесі шындықтар кестесінде көрсетіледі:

a	b	a
0	0	0
0	1	1
1	0	1
1	1	1

a және b ауыспалылары 1-ге тең болған жағдайда да, a **НЕМЕСЕ** b бәрібір 1-ге тең болатындығын айта өту керек. Бұл біздің «НЕМЕСЕ» операторын әдетті түсінігімізден айрықшаланады. Егер, біз «қамба қызыл немесе ақ түсті» десек, ол екі түсті емес, яғни не қызыл не ақ түсті екендігін болжамдаймыз. Бірақ логикалық оператор екі ауыспалы да «шындық» екен деп санайды. («Немесені жоққа шығару» деп аталатын тағы да бір логикалық оператор бар, ол «немесе / немесені» білдіреді, бірақ біз оны қарастырмаймыз.)

Қорытындылай келе, **ЖОҚ** операторы 0-ді 1-геал, 1-ді 0-ге ауыстыра отырып битті өзгертеді. Осылайша, егер ауыспалы a 0-ге немесе 0-ге тең, егер a 1-ге тең болса **ЖОҚ** a 1-ге тең. **ЖОҚ** операторы үшін шындықтар кестесі өте оңай:

a	ЖОҚ a
0	1
1	0

Осы негізгі операторлар көмегімен біз аса күрделі есептерді құрастыра аламыз. Мысалы, біз a **ЖӘНЕ** b **ЖОҚ** мәнін есептеп тапқымыз келеді. (Назар аударыңыз, **ЖОҚ** операторы ауыспалыдан кейін қолданылады, осы нақты жағдайда a **ЖӘНЕ** b барлық мәніне емес, тек a ғана. **ЖОҚ**-ты мәнге толығымен қолдану үшін біз қайталанба жақшаларды пайдалануымыз керек болар еді: **ЖОҚ** (a **ЖӘНЕ** b)). Біз a **ЖӘНЕ** b **ЖОҚ** мәнін ол үшін шындықтар кестесін жасау арқылы анықтай аламыз. a және b барлық комбинацияларын анықтаудан бастайық, сосын

Бульдік алгебра ережелеріне сәйкес a **ЖӘНЕ** b $a \wedge b$ деп белгілейді, a **НЕМЕСЕ** b $a \vee b$ деп, ал a **ЖОҚ** a $\neg a$ деп белгіленеді

a **ЖОҒЫ** үшін баған жасаймыз, себебі бізге оның қорытынды мәні қажет:

a	b	ЖОҚ
0	0	1
0	1	1
1	0	0
1	1	0

Осыдан кейін, **ЖӘНЕ** операторын үшінші бағанға қолдана отырып, a **ЖӘНЕ** b үшін **ЖОҚ** мағына бағанын жасаймыз.

a	b	ЖО	a ЖӘНЕ b ЖОҚ
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

Осылайша, a **ЖӘНЕ** b **ЖОҚ** мәні тек 0 және b 1-ге тең болған кезде ғана 1-ге тең екендігін анықтадық. Немесе, басқаша айтсақ, a – бұл өтірік, ал b – бұл шындық болған жағдайда ғана a **ЖӘНЕ** b **ЖОҚ** мәні шындық болып табылады. (Осы туралы ойланып көріңіз; мағынасы бар, солай емес не?)

Әмбебап машина

Бинарлы есептеу жүйесіндегі есептеудің артықшылықтары – ауыстырып-қосқыштар көмегімен қарапайым іске асырылатын әрекеттер, егер біз шын мәнінде қалаған затты бинарлы жүйеде есептей алмаған жағдайда, қарапайым логикалық операторлар мен кез келген заттарды кодтау қабылеттігі қажет болмай қалушы еді. Басқалай айтқанда, *әмбебап* екілік есептеу қажет (яғни бульдік алгебра): біз тек үш логикалық операторларды қолдана отырып, кез келген есептелетін есепті шығару мүмкіндігіне ие болуымыз керек. Бірақ, бұл шын мәнінде мүмкін бе? Және *кез келген есептелінетін есеп* деп нені атауға болады? Біз шын мәнінде барлығы үшін – веб-браузер, шахматтық бағдарлама, марса жүру бағдарламалық қамсыздандыру үшін оларды екілік форматқа үйлестіре отырып және жауап алу үшін тек үш ғана

қарапайым логикалық операторларды қолдана отырып кез келген есептеулерді жасай аламыз ба?

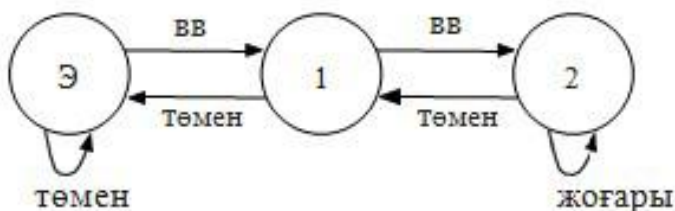
Бақытына орай, жауап – иә, егер біз логикалық сызбаларды жеткілікті жад көлемімен және трафикті өңдеу үшін қажет *қорытынды сандар қалты бар автомат* деп аталатын қарапайым контроллермен бірлетірсек.

Қорытынды автомат көптеген қалыптар қорытындыларынан, сондай-ақ қалыптар арасындағы өткелдерден тұрады. Қалып кейбір нысандар ағымдағы мәніне немесе мақсатқа жетудегі ілгерілеумен беріледі.



Сурет-1.7 Тьюринг машинасын сызбамен көрсету

Мысалы, қабаттар қалпы берілген қарапайым лифт - бұл қорытынды автомат, төмендегі суретте көрсетілгендей.



Қалыптар домалақтармен, ал өткелдер – домалақтар арасындағы бағыттамаалармен берілген. Бұл лифте тек жоғары және төмен деген батырмалар бар (кіруде өз қабатын таңдауға мүмкіндік жоқ). Әр өткелдегі белгі өткелдің болуына себеп болған батырма басуды меңзейді. Мысалы, біз бірінші қабатта бола тұра төменге апаратын батырманы бассaq, біз бәрібір бірінші қабатта қала береміз. Бірақ, жоғарыға апаратын батырманы басуда, біз екінші қабатқа ауысамыз. Тауарлық автомат, метро есіктері, бағдаршам немесе жиналу орны сияқты көптеген қарапайым жүйелер қорытынды автомат түрінде берілуі мүмкін. Компьютерде

қорытынды автоматы, іске асыру сұраныстар мен кезең орындауды, сондай-ақ, машиналық тілде командалар орындаумен байланысты аралық қадамдарды бағдарлайды.

Есептемелік жүйе әмбебап па деген сұрақ өз тарихын ақпараттың қалыптасуынан жалғастырып келеді. 1936 жылы Алан Тьюринг қазір *Тьюринг машинасы* деп аталатын абстракті есептеу үлгісін ұсынды, бұл

Черч-Тьюринг тезисы былай дейді: егер есепті Тьюринг машинасымен есептеп шығаруға болса, сол кезде ғана есеп шығарылды деп есептелінеді. Тьюринг машинасының есептеу қабілеттілігіне сәйкестік көрсеткен, кез келген механизм есептемелік әмбебап немесе Тьюринг бойынша толық деп саналады. Буьдік логикаға және қорытынды қалыпты автоматқа негізделген кез келген заманауи компьютер осы категорияға кіреді.

үлгі математикалық жолмен шешілетін, кез келген есепті есептеп шығаруға болатындығын дәлелдеді. 1.7- суретінде көрсетілгендей Тьюринг машинасы ақпаратты шексіз ұзын таспаға жазып, оқи алатын қорытынды автоматты құрайтын басқару шығырынан тұрады. Таспа «ұяшықтар» реттілігінен тұрады, ол ұяшықтардың әрқайсысы бір символдан тұрады.

Лентаға ең алдымен кіріс мәліметтер кейбір реттілігі жазылған, ал басқару шығырына қосылған бағдарлаушы енгізудің бастапқы мәніне тұрықтандырылған. Әр қадам сайын Тьюринг машинасы ағымдағы ұяшықтан ақпарат оқып тұрады. Сосын оқыған ағымдағы жағдайға негізделе отырып, қорытынды автомат символды ұяшыққа жазу керектігі және оның бағдарлауышын бір ұяшыққа оңға не солға жылжыту керектігі жайлы өз шешімін қабылдайды. Заманауи компьютердің сұраныс жасау мен кезеңді орындауынан аз айырмашылығы бар. Тьюринг машинасы компьютер жасайтын осы қарапайым барысты бекітілген қорытынды қалыпқа қол жеткізгенше қайталай береді. Нәтиже – таспадағы символдардың қорытынды реттілігі.

Тьюринг машинасы әлі күнге дейін біздің заманауи есептеудің

бекітілген түсінігімізді көрсетеді:

Жаттығулар

1.4.1. 1101 екілік санын ондық жүйеге қалай келтіруге болатындығын көрсетіңіз.

1.4.2. 1111101000 екілік санын ондық жүйеге қалай келтіруге болатындығын көрсетіңіз.

1.4.3. 11.0011 екілік санын ондық жүйеге қалай келтіруге болатындығын көрсетіңіз.

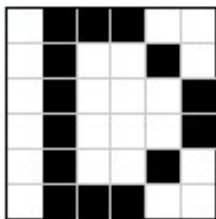
1.4.4. 11.110001 екілік санын ондық жүйеге қалай келтіруге болатындығын көрсетіңіз.

1.4.5. 22 ондық санын екілік жүйеге қалай келтіруге болатындығын көрсетіңіз.

1.4.6. 222 ондық санын екілік жүйеге қалай келтіруге болатындығын көрсетіңіз.

1.4.7. Екілік нүктеден алты оң орындарды пайдалана отырып, 0.1 Сіздің жақындауыңыздың нақты мәні қаншаға тең?

1.4.8. Келесі 6×6 ақ-қара суретті қарастырайық. Реттілік биттерді суреттеп көрсететін екі мүмкін тәсілді сипаттаңыз.



1.1.1. $(a \text{ ЖӘНЕ } b) \text{ ЖОҚ}$ мәнін логикалық сипаттау үшін шындықтар кестесін толтырыңыз.

1.1.2. **ЖОҚ** a **НЕМЕСЕ** **ЖОҚ** b мәндері үшін логикалық сипаттау үшін шындықтар кестесін толтырыңыз. Нәтижені **ЖОҚ** $(a \text{ ЖӘНЕ } b)$ мәндер шындық кестесімен салыстырыңыз. Сіз не байқадыңыз? Осы екі логикалық түсініктер арасындағы байланыс *де Морган заңдарының* бірі болып табылады.

1.1.3. **ЖОҚ** a **ЖӘНЕ** **ЖОҚ** b мәндерін логикалық сипаттау үшін шындықтар кестесін толтырыңыз. Нәтижені **ЖОҚ** $(a \text{ ЖӘНЕ } b)$ мәндер шындық кестесімен салыстырыңыз. Сіз не байқадыңыз? Осы екі логикалық түсініктер арасындағы байланыс *де Морган заңының* басқасы

болып табылады.

1.1.4. Жалғыз шлагбаумды басқаратын магистральді жиналу пункты болып табылатын қорытынды автоматты жасап шығарыңыз.

1.1.5. Сауда автомат болып табылатын қорытынды автоматты жасап шығарыңыз. Ойлап көрейік, сауда автоматы тек 25 центтік тиындарды қабылдайды және тек 75 цент тұратын бір ғана сусын түрін сатады. Ең алдымен қандай қалыптар болатындығын ойластырып көріңіз. Қалыптар арасындағы транзакцияларды жасап шығарыңыз.



1.8- сурет. Компьютердегі функционалды абстракциялар қабаттарын кеңейтіп көрсету

1.4 ТҮЙІНДЕМЕ

Бірінші тарауда абстракцияның түрлі деңгейінде есептеудің барлық тәсілдерін басынан аяғына дейін талдап шықтық. 1.8- суретте көрсетілгендей, біз нақты анықталған есептен бастап, оны алгоритм жасау жолымен шешеміз. Шын мәнінде, бұл қадам үлкен басымдылықты ең қиын қадам болып табылады. Мамандандырылған алгоритмдер жасаушылар жобалаудың ортақ әдістемелік арсеналындағы жасампаздық пен хабардарлыққа, тәжірбиелікке негізделе отырып, тиімі және дұрыс алгоритмдер жасауда өздерінің шеберлігін көрсетеді. Бұл кітапта біз осы саланың тек аз ғана бөлігін талқылаймыз. Сосын, алгоритм Python сияқты жоғары деңгейлі бағдарламалау тілінде бағдарламалануы тиіс, бұл бағдарлама компьютер қосыларға дейін машиналық тілге аударылады. Машиналық тілдегі бұл бағдарлама командалары тікелей процессорда орындалады немесе операциялық жүйе олардың атынан қандайда бір әрекетті жасауға сұраныс жасайды (мысалы, файлды сақтап немесе көбірек жад бөлу). Қорытынды есептеудегі әр команда соңғы автоматтың басқару қол астында бульдік логиканы пайдаланумен іске асырылған.

Кітаппен жұмыс жасауда осы ортақ суретті жадыңызда сақтауға тырысыңыз, әсіресе, есептерді тиімді шешудегі абстракция рөлін. Біз шешетін есептер аса қарапайым, бірақ олар көлемді есептер мен оларды шешуде бағдарлаушылармен пайдаланатын әдіснамалар туралы ортақ түсінікті қалыптастырады, сондай-ақ, с із күтетін кейбір сала аралық мүмкіншіліктер мәнін ашады.

1.4 БОЛАШАҚ АШЫЛУЛАР

Осы тарау басындағы алғы сөз Майкл Феллоус және Иэна Парберри бағдарламаушылар мақалаларынан алынған [13]. Осыған тектес цитата жиі өмірден өткен голландтық Эдсгер Дейкстреге тиесілі.

Информатика және компьютерлер қалай жұмыс жасайтындығы жайлы ақпарат беретін, оңай қол жетімді бірнеше керемет кітаптар атауын ұсынамыз. Нақтылай келе, Дэнни Хиллис *Тастағы қашау* [18], Дэвид Харель *Алгоритмдер* [17], Питер Бентли *Сандықталған* [5], және Чарльз Петцольд *КОД: компьютерлік құрылғы мен бағдарламалық қамсыздандыру жасасын тілі* [40].

Әлем мұхиттарында климат өзгеруін бақылау үшін, шын мәнінде әрдайым буилар қалқып жүреді. Мысалы мына сілтемеден қараңыз

<http://www.coriolis.eu.org>.

Әлемдегі ең жылдам суперкомпьютерлер тізімі
<http://top500.org>. сайтында жүргізіледі

IBM Deep Blue суперкомпьютері жайлы толығырақ келесі сілтеме бойынша

<http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/>

ал IBM Watson суперкомпьютері жайлы
<http://www.ibm.com/smarterplanet/us/en/ibmwatson/> сайтында
білуге болады.

Жоғары өндірістік есептеулер туралы жалпы ақпарат алу үшін, біз Иллинойс штаты университетіндегі суперкомпьютерлерді қолдану (NCSA) бойынша, келесі мекен-жайда орналасқан, Ұлттық орталық сайтына кіріп көруді ұсынамыз

<http://www.ncsa.illinois.edu/about/faq>

бұл орталық Ұлттық ғылыми қормен қаржыландырылатын суперкомпьютерлер орталықтарының бірі

Информатика әкесі болып табылатын Алан Тьюринг өмірі туралы бірнеше жақсы кітаптар бар. Алан Тьюринг толық өмірбаяны: *Энигма*, Эндрю Ходжес [20].

2 бөлім

Қарапайым есептеулер

Сандар біз есептей алатын ең қарапайым және ең кең таралған нысандардың бірі болып табылады. Сонымен қатар, сандар өте әмбебап; олар көлемді, өзгерулерді, ұқсастырғыштарды білдіруі мүмкін, сондай-ақ, олар реттелген элементтер болып табылады. Осы тарауда біз Python бағдаламалау тілінде сандармен есептер шығарудың негізгі принциптерін көрсетеміз және компьютерде сандар математикалық сандардан, неліктен өзгеше болатындығына мысал келтіре отырып түсіндіреміз. Келесі тарауларда осы идеяларға негізделе отырып, біз күрделірек есептерді шығаруға мүмкіндік беретін алгоритмдерді жасап шығарамыз, мысалы мәтін, тізімдер, кестелер мен желілер сияқты нысандарды.

2.1 ЦИРККЕ ҚОШ КЕЛДІҢІЗ

Бағдарламалар (немесе «бағдарламалау») жазу тәжірбиелік әрекет болып табылады, ол біздің алгоритмдердің нәтижелерін қажеттіліктерге сәйкес келетін тәсілмен тәжірибеде қолдануға және идеяларды тексеруге мүмкіндік береді. Есептерді шешу мен бағдарламаларды жазу қызықты және шығармашылықты барыс болуы тиіс. Python бағдарламасын жасаған Гидо ван Россум «аса көтеріңкі көңіл күйде бола тұрып» бағдарламалау тілін «Монти Пайтон ұшқын циркі» британдық комедиялық сериал атымен атаған!

Бағдарлама біздің идеяларымызды компьютер «түсіну» үшін қысқа формаға келтіруді талап ететіндіктен, бұл біздің есептің өзін түсінуімізді сөзсіз жоғарлатады. Жоғарыда аталып өтілген заманауи информатика негізін қалаушысының бірі Дональд Кнут цитатасы осы пікірді керемет білдіреді. Сол логикаға сүйене отыра, шын мәнінде алгоритмдерді жасауды үйренудің және бағдарламалар жазудың тәсілі – оларды жасай бастау!

Жұмысқа кірісу үшін, әр Python дистрибутив жиынтығымен жеткізілетін IDLE атты үстемені (немесе с іздің оқытушыңыз кеңес берген басқа, ұсқас пайдалану ортасын) іске қосыңыз. Сіз, пайда болған терезенің жоғары жағында келесіге ұқсасты көруіңіз керек:

Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 5 2014, 20:42:22)

[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type “copyright”, “credits” or “license()” for more information.
>>>

Осы терезеде атқарылатын бағдарлама интерпретатор немесе *Python* (shell) *қабықшасы* деп аталады. Бірінші жол с ізге Python-ның қандай нұсқасы қолданылып отырғандығы жайлы ақпарат береді (қазіргі жағдайда 3.4.2). Осы кітапта келтірілген мысалдар Python 3 нұсқасына негізделген, сондықтан, нұсқа нөмірі 3-тен басталатындығына көз жеткізіңіз. Егер с ізге жаңартылған нұсқаны орнату қажет болса, с із оны python.org сайтынан таба аласыз. IDLE төртінші нұсқама терезесіндегі

>>>

белгісін *шақыру* деп аталады, себебі ол Сізге Python-ге нұсқамалар енгізуді бастауға шақырады. Бастау үшін шақыру жолына `print('Сәлем әлем!')` деп жазыңыз және не болатындығын қараңыз.

>>> `print('Сәлем әлем!')` Сәлем әлем!

Құттықтаймыз, с із қазір ғана алғашқы дәстүрлі бағдарлама жаздыңыз! Бұл бір жолды бағдарлама *символдар жолы* немесе жай ғана *жол* деп аталатын дара жақшалардан символдарды экранға шығарады. Енді с із әрі қарай жылжуға дайынсыз!

Сізге , бұл кітапты оқығанда, осы әрекеттерді компьютерде жасауға кеңес береміз. Шақыруға алдын ала болған Python нұсқаулығы жоғарыда аталып өтілгендей оқырманмен Python қабықшасына енгізу үшін арналған. Әр мысалды өз бетіңізше орындап көріңіз. Сосын мысалдар шектеулерінен шығып, өзіңіздің идеяларыңызбен тәжірибелер жасап көріңіз. «Егер мен мына әрекетті істесем не болады?»- деген сұрақтарға жауап іздеудің орнына, ол әрекетті компьютерге енгізіп не болатындығын байқап көріңіз! Сіз дәл «ойлану үшін» сұрақты оқығанда бір секундке тоқтап, жауап туралы ойланыңыз немесе жауап табу үшін тәжірибе жасап көріңіз.

2.2 АРИФМЕТИКАЛЫҚ ӘРЕКЕТТЕР

Біз бір нәрседен бастауымыз керек, сондықтан, бастауыш мектептен бастасақ болады, бастауыш мектепте с із алғашқы арифметикалық есептеулерді жасай бастадыңыз. Python әрине, арифметикалық әрекеттерді

жасай алады. Шақыру жолына $8 * 9$ деп жазып, не болатындығын байқап көріңіз.

```
>>> 8 * 9
```

```
72
```

```
>>>
```

* операторы жанындағы аралықтар опционалды және елемеушілік. Тұтас алғанда, Python-ға сіз 3 өрнекке аралық қосасыз ба, жоқ па бәрібір, бірақ егер сіз өз бағдарламаларыңыздың оңай оқылуына дағдылансаңыз, онда аралықтар жақсы көмек болады.

2.1 ескертпе: Қалқыма нүктелі жазба

Python қалқыматы нүктелі сандар әдетте IEEE 754 екілік нақтылықты қалқыма нүкте деп мәлім 64 битті жад форматында сақталады. Бір бит сандарды белгілеу үшін, 11 бит деңгей көрсеткіші үшін және 52 бит нүкте оң жағынан бөлшек *мантиссені* бөліп көрсетеді. Сан екілік файл түрленгеннен кейін (1.4 Бөліміндегідей), екілік нүкте мен деңгей көрсеткіші екілік нүктенің сол жағында бір 1 болу үшін осылайша түзетіледі. Сосын көрсеткіш деңгейін және 52 биттік слотқа сақтауға болатын мансисса бөлігін 11-биттік слотқа сақтайды. Егер мантисса 5 биттен ұзын болса, оны жай ғана қиып тастайды, алақпарат жоғалады. Бұл егер біз $2.0 * 100$ есептеген жағдайда болатын әрекет. Компьютер жады шектеулі болғандықтан нақтылық пен орын алатын көлем арасындағы келісім аса кең тараған, сондықтан, қажетікерді болдырмау үшін осындай шектеулерді түсініп отыру өте маңызды. 4- тарауда бұл аса маңызды болатын ортақ жағдайды қарастырамыз.

Есіңізде болсын, қабықша біздің енгізуімізге есептеу нәтижесімен жауап берді, және бізге жаңа шақыру жіберді. Қабықша бұл шақыру-есептеу-жауап беру кезеңін біз шығып кеткенше қайталай береді (quit() командасын тергенде). Python - бұл түсіндіру тілі екендігін 1.4 бөлімінен еске түсіріңіз; қабықша түсіндіргішті әрекетте көрсетеді. Біз әр жолы шақыруға нұсқама енгізуде Python түсіндіргіші бұл нұсқаманы машиналық тіл командасын орындап, сосын нәтижені шығара отырып, машиналық тілге түрлендіреді.

Енді көптеген калькуляторлар жасай алмайтын бір әрекетті жасап көрейік:

```
>>> 2 ** 100
```

1267650600228229401496703205376

** символы – бұл кезіндегі әсер ету операторы; осылайша, бұл өрнек 2100 есептеп, 31-разрядты санды шығарады. Python бұдан да ұзын толық санды мәндерді өңдей алады; бір (10100) гуголды және 2500 есептеп шығаруға тырысыңыз. Шын мәнінде, Python-дағы толық санды мәндер *шексіз нақтылыққа* ие, бұл олар кез келген ұзындықты бола алатындығын еңзейді.

Енді, бастапқы бекітілулермен салыстырғанда басқа кішірек өзгертулер енгізіп көрейік:

```
>>> 2.0 ** 100 1.2676506002282294e+30
```

Бұл нәтиже басқаша болып көрінеді, себебі 2-ге ондық нүктені қоса отырып, біз Python-ге бұл үлкендікті толық сан орнына қалқымалы нүктесі бар сан ретінде түсіндіруге негіз береміз. Қалқымалы нүктесі бар сан – бұл ондық нүктесі бар кез келген сан. Қалқымалы нүкте өз атауын сандар жадта сақталу әдісінен алған. Оның ондық нүктесіндегі форматына «жүзу», экспоненциалды жазу формасында рұқсат етіледі. Мысалы, $2.345 \times 10^4 = 23.45 \times 10^3 = 0.02345 \times 10^7$. Қалқымалы нүкте жазу формасы туралы көбірек білу үшін 2.1 Ескертпесіне назар аударыңыз.

Қалқымалы нүктелі сан әр жолы есептеуге қатысқанда, есептеу нәтижесі әрқашан қалқымалы нүкте саны болады. $1.2676506002282294e+30$ сияқты қалқымалы нүкте өте үлкен саны экспоненциалды көрсетуде беріледі (е «экспоненциалдық» деген мағынаны білдіреді).

Бұл сан келесідей болады

$1.2676506002282294 \times 10^{30} = 12676506002282294000000000000000$.

Сондай-ақ, өте жиі сандарды экспоненциалды көріністе енгізу ыңғайлы. Мысалы, жер жасы шамамен 4.54 миллиард жыл екендігін біле отырып, біз жердің жасын күн шамасында есептеп шығара аламыз

```
>>> 4.54e9 * 365.25 1658235000000,0
```


Тура осылай сияқты, егер АҚШ-та туу деңгейі 1, 000 адамға 13.42 көрсеткіште болса, онда 2012 жылы халық саны шамамен бағалағанда 313.9 миллионды құраса, туылған нәрестелер саны

```
>>> 13.42e-3 * 313.9e6
4212538.0
```

Шектелген нақтылық

Қарапайым арифметикада 2^{100} және 2.0^{100} бірдей сандар. Бірақ, өткен бетте $2 ** 100$ және $2.0 ** 100$ есептеу нәтижелері әртүрлі болды. Бірінші жауап нақты болды, ал екінші жауап – шамамен 1.5 триллионға кесілген! Қалқымалы нүктелі сандарды сақтау қиындығы толық сандардан ерекшеленеді және шектелген нақтылыққа ие, бұл берілетін сандар диапазоны шектеулі деген мағынаны біліреді.

Бұл жағдайда шектеулі нақтылық, біз мәнге жақын нәтижелерді алатынымызды немесе мәнді жақсырақ жақындату үшін ол тым үлкен не тым кіші екендігін білдіреді. Мысалы, байқап көрейік:

```
>>> 10.0 ** 500
```

```
OverflowError: (34, 'Result too large')
```

Overflow қатесі (толып кету) дұрыс мәнді беру үшін, компьютерге жад көлемі жеткіліксіз екендігін білдіреді. Егер біз осы сияқты қандай да бір келеңсіздікті жіберсек, осындай қате орын алады, мысалы, нөлге бөлу:

```
>>> 10.0 / 0
```

```
ZeroDivisionError: division by zero
```

Бөлу

Python екі түрлі бөлу түрін қарастырады: «дұрыс бөлу» және «қалдықсыз бөлу». Дұрыс бөлу операторы – бұл слэш (/) символы, қалдықсыз бөлу операторы – екі слэш (/). Дұрыс бөлу сіз күткен нәтижені береді. Мысалы, $14/3$ 4.666666666666667 нәтижесін береді. Қалдықсыз бөлуге келсек, бөлу нәтижесі ең жақын толық санға дейін төмен жұмырланады. Мысалы, $14//3$ 4 деген нәтижені береді. (Ең жақын

толық төмен санға дейін дөңгелектеу математикада «қалдықсыз бөлу» деп аталады – осыдан оператор аты тараған.)

```
>>> 14 / 3
4,666666666666667
>>> 14 // 3
4
```

Екі санда оң және толық болғанда, сіз бастауыш мектепте оқыған «бағаналық бөлудіі» еске түсіре аласыз: қалдықсыз бөлу, сондай-ақ, бөлу нәтижесінің толық бөлігін береді, ал қалдық ескерілмейді. Біздің мысалымызда, 14-і 2-ге бөлу 4-ің толық бөлігі мен 2 қалдығын береді, себебі 14 нәтижесі $4 \cdot 3 + 2$ тең. Қалдық алу командасын «модуль» (“modulo”) бойынша оператор деп атайды. Математикада бұл оператор mod деп белгіленеді, мысалы $14 \bmod 3 = 2$; Python бағдарламасында ол $\%$ деп белгіленеді. Мысалы:

```
>>> 14%3
2
```

// және $\%$ операторлар не үшін керектігін түсіну үшін, сан жұп не тақ екендігін сіз қалай анықтар едіңіз, осы туралы ойланып көріңіз. Егер сан 2-ге бөлінетін болса, ол жұп сан деп аталады, яғни, сіз ол санды 2-ге бөлдіңіз, бөлу қалдығы 0-ге тең болады.

Осылайша, толық сан жұп сан екендігін тексеру үшін, Біз 2-ге “mod” санын қолданып, жауапты тексере аламыз.

```
>>> 14 % 2
0
>>> 15 % 2
1
```

2.1 Ойлануға арналған сұрақтар. Сан бөлу операторлары теріс санды modulo-сыз және қалдықсыз жұмыс істей ме? Бірнеше мысалдарды тексеріңіз, не боларын байқап көріңіз. Қандай ережелер нәтижені анықтайды?

Кейбір толық санды x мәндерді басқа толық санды y мәндерге бөлуді қарастырып көріңіз. Қалдықсыз және modulo-сыз сандар бөлу операторы әрқашан келесі ережелерге бағынады:

1. $x \% y$ бар белгілер y де бар, және
2. $(x // y) * y + (x \% y)$ x тең. Бұл қағиданы келесі есептерді шығарып растаңыз
 - $-14 // 3$ және $-14 \% 3$
 - $14 // -3$ және $14 \% -3$
 - $-14 // -3$ және $-14 \% -3$

Операциялар реттілігі

Енді кішкене қызығырақ нәрсе істеп көрейік: 10 радиус шеңберінің ауданын есептеу. ($A = \pi r^2$ формуласын есіңізге түсіріңіз)

```
>>> 3.14159 * 10 ** 2
```

```
314,159
```

Операторла		Сипаттамал
1	**	Денгеймен шығару (экспонент)
2	+, -	Унарлы қосу мен алу, мысалы, $-(4 * 3)$
3	*, /	Көбейту мен бөлу
4	+, -	Қосу мен алу

2.1 кестесі Үлкеннен кішіге қарай арифметикалық операторлар басымдылығы. Бірдей басымдылыққа ие операторлар солдан оңға қарай есептелінеді.

Назар аударыңыз, Python стандартты арифметикадағыдай көбейту алдында денгеймен шығаруды дұрыс жасайды. Бұл стандартты арифметикаға толығымен сәйкес келетін операторлар реттілігі ережесі 2.1 кестесінде берілген. Болжамдағандай, сіз бекітілген басымдылықты

жақшалар қолдану арқылы жеңе аласыз. Мысалы, біз жылына \$18,000 жалақы алатын 100 жалдамалы сатушы және \$18 миллион жалақы алатын бас директоры бар бөлшек сауда жасайтын компанияның орташа жалақысын есептеп шығарғымыз келеді:

$$>>> ((100 * 18000) + 18e6) / 101$$

$$196039.60396039605$$

Бұл жағдайда бізге қор жалақысын қызметкерлер санына бөлерге дейін есептеу үшін $(100 * 18000) + 18e6$ айналасында домалақ жақшалар қажет болған. Бірақ, біз көрнекі болу үшін $100 * 18000$ айналасына домалақ жақшаларды қостық.

Айта өту қажет, бастапқы мәндері толық сандар болғанымен, екі жоғарыда көрсетіліп кеткен мысалдар жауаптары қалқыма нүктелі сандармен болды. Python толық сандармен және қалқымалы нүктесі бар сандармен арифметикалық әрекеттерді жасауда алдымен, толық санды, қалқымалы нүктесі бар санға түрлендіреді. Мысалы, соңғы өрнекте $(100 * 18000)$ 1800000 толық мәні есептеліп шығарылады. Сосын, 1800000 18e6 қалқыма нүктесімен қосылады. Бұл екі операнд түрлі үлгілі болғандықтан, 1800000 қосу операциясына дейін қалқыма нүктесі бар 1800000.0 эквивалентке алып келеді. Осы жағдайда нәтиже қалқыма нүктесі бар 19800000.0 мәнге тең болады. Енді қорытындылай келе, 19800000.0 толық санды 101 бөлінеді. Тағы да 101 бірінші болып қалқыма нүктесі бар санға түрленеді, сондықтан, шынында бөлу операциясы $19800000.0 / 101.0$ деп беріледі, бұл қорытынды жауап береді. Бұл үдеріс төменде көрсетілген.

$$\begin{array}{c} \underbrace{((100 * 18000) + 18e6)}_{1800000} / 101 \\ \underbrace{1800000}_{19800000.0} \\ \underbrace{19800000.0}_{196039.60396039605} \end{array}$$

Көп жағдайларда, барлығы біз ойлағандай жұмыс жасайды, және автоматты түрде іске асып отыратын әрекеттер жайлы ойланудың қажеті жоқ.

Кешенді сандар

Біз бұл сандарды осы кітапта қолданбайтынымызға қарамастан, Python кешенді сандармен жұмыс жасай алатындығымыз жайлы айта өту керек. Кешенді сан $i^2 = -1$ қабілетіне ие i жалған бөлігін қоса, шынайы және жалған бөлігіне ие.

В Python-де $3.2 + 2i$ сияқты кешенді сандар $3.2 + 2j$ деп жазылады. (j әріпі i орнына қолданылады, себебі электроника сияқты салаларда i басқа тұрақты мәндерге ие, бұл екі мәнділікке алып келуі мүмкін).

Көптеген қарапайым арифметикалық операторлар да кешенді сандармен жұмыс жасайды. Мысалы,

```
>>> (5 + 4j) + (-4 + -3.1j) (1+0.8999999999999999j)
>>> (23 + 6j) / (1 + 2j) (7-8j)
>>> (1 + 2j) ** 2 (-3+4j)
>>> 1j ** 2 (-1+0j)
```

Соңғы мысал i анықтамасын көрсетеді: $i^2 = -1$.

Жаттығулар

Келесі сұрақтарға жауап беру үшін Python түсіндіргішін қолданыңыз. Қажет болған жағдайда, жауап алу үшін қолданған Python жауабы мен өрнегін көрсетіңіз.

2.2.1. АҚШ-та Конгресс кітапханасы өз архивтерін 838 мильді сөрелерде сақтайды. Бір кітаптің орташа қалыңдығы бір дюйм деп болжамдағанда, кітапхана қанша кітап сыйдыра алады? (Бұл жерде бір мильде 5,280 фут және футте 12 дюйм бар).

2.2.2. Егер мен сізге никель (5 цент) беріп, сізде бар сомманы келесі 24 сағат ішінде әр сағат сайын көбейтіп тұрамын деп уәде бермес, соңына таман сізде қанша ақша болады? Ал егер мен әр сағат сайын сомманы 50% ұлғайтып отырсам? Сізде қанша ақша болар еді? Осы көлемдерді есептеу үшін дәрежелерге шығаруды қолданыңыз.

2.2.3. АҚШ-тың Конгресс кітапханасы өз архивтерін 838 мильді сөрелерде сақтайды. Бұл Грэнвил, Огайо мен Колумбус, Огайо арасында әрі бері қанша саяхаттау? (Қалалар арасындағы ара-қашықтық

33,3 мильді құрайды).

2.2.4. Жер жасы 4,54 миллиард жылға тең. Анатомиялық заманауи адамдар бізге мәлім ең ескі қаңқасына (тасқа айналған) шамамен 200,000 жыл. Жердің бар болу мерзімінің қай бөлігінде адам пайда болды? Осыны есептеу үшін Python экспоненциалды жазу формасын пайдаланыңыз.

2.2.5. Егер Сіз секундына орташа шамамен бір рет санасаңыз, 4,54 миллиардқа жету үшін сізге қанша уақыт қажет болар еді? Осыны есептеу үшін Python экспоненциалды жазу формасын пайдаланыңыз.

2.2.6. Болжап көрейік, заманауи компьютер ішкі сағаты секундына шамамен 2,8 миллиард жылдамдықпен «санай» алады. Осындай компьютерге 4,54 миллиард рет санау үшін қанша уақыт қажет?

2.2.7. Компьютердегі қатқыл дискі шамамен бір терабайт (240 байт) ақпарат сақтай алады. Орта көлемді ән сақтау үшін шамамен 8 мегабайт (8×220 байт) қажет. Қатқыл дискіде қаншалықты көп әндер сақтауға болады?

2.2.8. Келесі Python өрнектерінің әр мәні қандай? Әр жеке жағдай бойынша жауапты түсінетініізге көз жеткізіңіз.

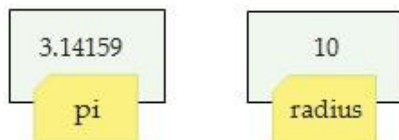
- (a) $15 * 3 - 2$
- (б) $15 - 3 * 2$
- (в) $15 * 3 // 2$
- (г) $15 * 3 / 2$
- (д) $15 * 3 \ 2$
- (е) $15 * 3 / 2e0$

2.3 АТАУМЕН НЕ БАЙЛАНЫСТЫ?

Мәнмен байланысты алгоритмдердегі және бағдарламалардағы атау ауыспалы деп аталады. Мысалы, шар көлемін есептеуде ($V = (4/3)\pi r^3$ формуласы бойынша) бізге сфера радиусын белгілеу үшін ауыспалыны және π мәніне сілтеме беретін басқа ауыспалыны (шамамен) пайдалануға болар еді:

```
>>> pi = 3.14159
>>> radius = 10
>>> (4 / 3) * pi * (radius ** 3) 4188.786666666666
```

Теңдік белгісі (=) *үлестіру операторы* деп аталады, себебі, ол атауларға мән үлестіру үшін пайдаланылады. Бұл мысалда біз `pi` атауына 3.14159 мәнін және `radius` атауына 10 мәнін бердік. Компьютер жадындағы мекен-жайларға ауыспалылар атауын «стикерлер»^е ретінде қыстырып қою аса ыңғайлы.

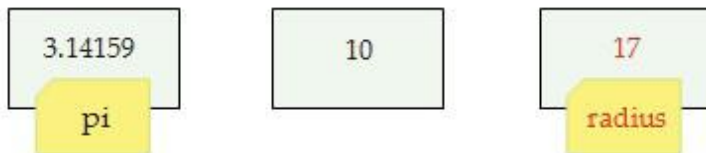


1.4 бөлімінен еске салайық, сандар компьютер жадындағы ұяшықтарда сақталады. Бұл ұяшықтар пошталық жәшіктерге ұқсас; ұяшық мекен-жайы – пошталық жәшік мекен жайы сияқты, ұяшықта сақталатын мән – жәшік мазмұны. Жоғарыда көрсетілген суреттегі сандар жазылған әр түк төртұрыш жад ұяшығы болып табылады. Ауыспалы атауы пошталық жәшік атауы бар стикерге сәйкес нақты ұяшыққа сілтеме болып табылады. Бұл суретте `pi` атауы мәнден тұратын ұяшыққа жатады, ал `radius` 0 мәнінен тұратын ұяшыққа жатады. Шын мәнінде, біз бұл жад ұяшықтардың әр мекен-жайын білмейміз, себебі, біз әрқашан оларға ауыспалы атауы арқылы жол табамыз. Стикер сияқты, уақыт өте келе мән басқа атауға берілуі мүмкін. Мысалы, егер енді біз `radius` атауына басқа мән берсек, онда суреттегі `radius` стикеры басқа мәнге ауысып кетеді.

<code>and</code>	<code>contin</code>	<code>except</code>	<code>glob</code>	<code>lambda</code>	<code>pass</code>	<code>wh</code>
<code>as</code>	<code>def</code>	<code>False</code>	<code>if</code>	<code>None</code>	<code>raise</code>	<code>wi</code>
<code>asser</code>	<code>del</code>	<code>final</code>	<code>impo</code>	<code>nonloc</code>	<code>retur</code>	<code>yie</code>
<code>break</code>	<code>elif</code>	<code>for</code>	<code>в</code>	<code>not</code>	<code>True</code>	
<code>class</code>	<code>else</code>	<code>from</code>	<code>is</code>	<code>или</code>	<code>try</code>	

2.2 кесте. Python-ның 33 басты сөздері.

^е“Sticky note” (стикер) – бұл BIC Corporation тіркелген тауар маркасы.



```
>>> radius = 17
```

Қайта бағыттаудан кейін, 17 мәнін құрайтын басқа жад ұяшыққа бағыттайтын сілтеме `radius`-қа беріледі. 10 мәні белгілі бір уақыт аралығында жадта атаусыз қала беруі мүмкін, бірақ мұндай жағдайда, оған апарар сілтеменің болмаған жағдайында, сіз бұл мәнге қол еткізе алмайсыз, бұл жағдайда Python «қоқыс жинау» механизмі жақын арада ол алып отырған жадты босатады.

Мәнге атау беру үш маңызды мақсатқа қызмет етеді. Біріншіден, жиі пайдаланылатын мәндерге сипаттаушылық атау беру, сіздің алгоритміңізді аса түсінікті етеді. Мысалы, егер біз қасқыр және бұлан (4.4 Жобадағыдай) сияқты екі аңдар арасындағы туу көрсеткіші бойынша бәсекелестікті үлгілеу алгоритмін жазу үшін 0,5 және 0,005 сияқты атаусыз белгілер орнына, мысалы, `birthRateMoose` және `birthRateWolves` атаулар беретін болсақ, біздің алгоритм аса оңай оқылатын болар еді. Екіншіден, 3.5 бөлімінде көрсетілгендей, кіріс мәндеріне атаулар беру алгоритмдерді жалпылауға мүмкіндік береді, осылайша, нақты бір кіріс мәндер жинағына байланудың орнына олар барлық кіріс мәліметтері үшін жұмыс жасайды. Үшіншіден, атаулар, біз әрі қарай пайдаланғымыз келетін мәндерді есептеу үшін белгілер ретінде қызмет ететін болады

Назар аударыңыз, біз алгебрадағы сияқты жеке әріптерді емес, толық сипаттаушы атауларды қолданамыз. Жоғарыда аталып өтілгендей, сіз бағдарламалар жазуда сипаттаушы атауларды қолдану әрдайым өте маңызды, себебі бұл сіздің жұмысыңызды басқалар үшін, түсінікті әрі қолжетімді етеді. «Шынайы әлемде» бағдарламалау өте жиі ортақ жұмыс болып табылады, сондықтан, басқалары сіздің мақсаттарыңызды түсінуі аса маңызды. Кейде, тіпті, қажет болмаған жағдайда да, аралық есептеуге сипаттаушы атау беру жеңіл оқылуды жоғарлатуы мүмкін

Python-дағы атаулар әріпті, санды және астын сызу (`_`) символдарының кез келген реттілігіне ие болуы мүмкін, бірақ атаулар сандардан бастала алмайды. Сіз 2.5 кестесінде келтірілген кез келген Python басты сөздерін қолдана аласыз. Басты сөздер алдын ала анықталған мәндерге ие Python элементтері болып табылады. Біз олардың кейбіреулерін 1 тараудағы

(def, for, in, и return) Python кодтар үзіндісінде көрдік және басқалардың көбірек бөлігімен кітап барысында кездестіреміз.

Әрі қарай не болатындығын білу үшін осы атаулар беру ережелерінің кейбірін бұзып көрейік.

```
>>> 49ers = 50
```

```
SyntaxError: invalid token
```

Синтаксистік қате Python грамматикасының немесе синтаксисінің бұзылуын меңзейді. Бастаушы бағдарламалаушылар үшін көптеген синтаксистік қателерге тап болуы қалыпты жағдай; бұл оқыту барысының бір бөлігі болып табылады. Көп жағдайларда сіз қандай қате жібергеніңіз де мәлім болады, сіз қателікті түзеп, әрі қарай жылжисыз. Бірақ кейде қателікті табу үшін аса ынталы іздеуге тура келеді, бірақ уақыт өте келе, тәжірибе жинай отырып, мұндай қателерді табу оңай болады. Беріліп отырылған мысалда, “invalid token” (жол берілмейтін лексема) біз саннан басталатын атауды қателікпен қолдандық дегенді білдіреді. Token (лексема) – бұл түсіндіргіш маңызды деп санайтын, кез келген топ символына жататын техникалық термин; бұл жағдайда ол біз таңдаған атауға қатысты дегенді білдіреді.

Енді мынаны байқап көрейік.

```
>>> my-age = 21
```

```
SyntaxError: can't assign to operator
```

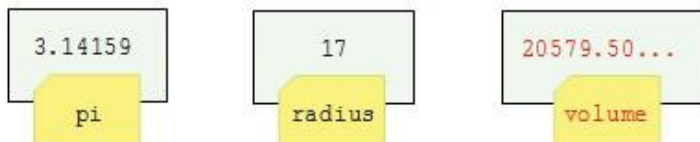
Бұл синтаксистік қате біздің атауымызда бар сызықша / дефис / алу белгісі (-) символының барына меңзейді. Python символды алу белгісі операторы ретінде түсінеді, сондықтан оны атауда қолдануға болмайды. Оның орнына астын сызу () (яғни, my_age) символын қолдануға немесе атаудағы екі сөзді бөлу үшін тіркеуішті (т. е. myAge) өзгертуге болады.

Тұрақты мәндерге атау берумен қоса біз бүтін бір есептеу нәтижесіне атау бере аламыз.

```
>>> volume = (4 / 3) * pi * (radius ** 3)
```

Python қабықшасындағы мәндерге атау беруде қабықша қайтармалы байланысты болмайды. Бірақ, ауыспалыға берілген мән атауын қарауға болады, ол үшін атауды теру жеткілікті.

```
>>> volume  
20579.50889333333
```



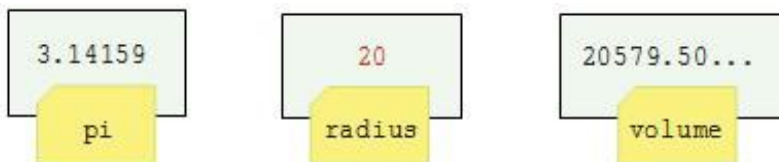
Сонымен, төменде көрсетілгендей, біз 20579.50889333333 мәніне volume атауын бердік. (pi және radius берілген мән өзгерусіз қалды.)

Нұсқа ретінде print қолдана отырып біз көргіміз келетін көлемді домалақ жақшалармен жазуға болады. Көп кешікпей, біз бағдарламаларды қабықша сыртында жаза бастағаннан кейін, print көлем атауының жалғыз тәсілі болып қалады.

```
>>> print(volume) 20579.50889333333
```

Енді radius мәнін тағы да өзгертіп көрейік:

```
>>> radius = 20
```



Басқа мысалды қарастырып көрейік. Желді ескере отырып, солтүстік америка температурасын есептеу формуласы, Цельсия градустарында:

$$W = 13.12 + 0.6215 t + (0.3965 t - 11.37) v0.16$$

Бұл жерде t – Цельсия градустарындағы қоршаған орта температурасы, және v – км/сағ.² берілген жел жылдамдығы. -3 °C желді ескере отырып температура мен 13 км/сағ жел жылдамдығы үшін температураны есептеу үшін біз ең алдымен екі ауыспалыға температураны және желді үлестіреміз:

```
>>> temp = -3
>>> wind = 13
```

Сосын нәтижеге `chill` атауын беріп, Python-ға формуланы енгіземіз.

```
>>> chill = 13.12 + 0.6215 * temp + (0.3965 * temp - 11.37) * wind**0.16
>>> chill
-7.676796032159553
```

Температура тағы өзгеруде `chill` өзгермейтіндігіне назар аударыңыз:

```
>>> temp = 4.0
>>> chill
-7.676796032159553
```

`chill` мәнін жаңа температурамен есептеу үшін, бізге `temp`-ға берілген жаңа мәнді қолданатын жоғарыда аталып өтілген формуланы қайталап жазу қажет.

```
>>> chill = 13.12 + 0.6215 * temp + (0.3965 * temp - 11.37) * wind**0.16
>>> chill 0.8575160333891443
```

Әрине, бұл барыс өте шаршатқыш және қателік ықпалды. Бірақ, біз 3.5 бөлімінде көретіндей, осындай есептеулерді анықтаудың аса жақсы тәсілі бар.

² желді ескере отырып температураны техникалық түрде тек 10 °C төмен температура мен 4.8 км/сағ жел жылдамдығы бойынша анықталады.

Теңдік белгісін қолдануға қарамастан, бекіту – теңдікпен бірдей еместігін түсіну өте маңызды. Басқаша айтқанда, біз үлестіру операторын іске асырғанда, мысалы `temp = 4.0` сияқты, біз `temp` және `4.0` теңдестірулі деп айта алмаймыз. Бұның орнына мән атауға екі кезеңмен беріледі:

1. Үлестіру операторының оң жақ бөлігіндегі мән есептелінеді.
2. Алынған мән атауы үлестіру операторының сол жағына беріледі.

Мысалы, келесі үлестіру операторларын қарастырайық:

```
>>> radius = 20
>>> radius = radius + 1
```

Егер теңдік белгісі теңдестіруді білдіретін болса, онда екінші нұсқаулық өз мағынасын жоғалтар еді! Бірақ, егер біз екі қадамдық барысты қолдана отырып нұсқаулықты түсіндірсек, ол абсолютті түрде мақсатты болар еді. Алдымен, біз сол жақты толығымен есепке алмай, оң жақ мәнін есептейміз. Қазіргі кезде `radius`-ке `20` мәні берілгендіктен, оң жақ `21` мәнін қабылдайды. Әрі қарай, біз `21` мәнін `radius` атауына береміз. Осылайша, оператор ағымдағы `radius` мәніне `1`-ді қосты, бұл операцияны біз өсіру деп атаймыз. Дәп осы әрекет іске асқанына көз жеткізу үшін, радиус мәнін тексеріңіз:

```
>>> radius 21
```

Егер біз `radius`-қа мән бермесек және оны үлкейтуге тырыссақ не болар еді? Бұны анықтау үшін, біз ешқандай мән берілмеген атауды қолдануымыз қажет.

```
>>> trythis = trythis + 1
```

```
NameError: name 'trythis' is not defined
```

Бұл атау қателігі Python түсіндіргіші оператордың оң жағын есептеуге тырысқанда `trythis`-қа мән берілмегендіктен орын алды, яғни, ауыспалы анықталмаған болған. Осылайша, біз қандайда бір ауыспалыны өсірер алдында оны анықтағанымызға көз жеткізуіміз керек. Бұл бәріне мәлім болып көрінеді, бірақ уақыт өте келе, сізге жолығатын басқа аса үлкен

бағдарламаларда жұмыс жасауда бұл жайлы ұмытып кету оңай.

Тағы бір мысалды қарастырып көрейік. Біз сандық сағаттар көрсететін, минуттар санын үлкейткіміз келеді деп болжамдайық. Минуттарды санау үшін және мәнді өсіру үшін бізге, бар болғаны, жоғарыда жасалған әрекеттерді қайталау қажет.

```
>>> minutes = 0
>>> minutes = minutes + 1
>>> minutes 1
>>> minutes = minutes + 1
>>> minutes 2
```

Бірақ minutes 59 мәніне жеткенде өсіру дұрыс жұмыс жасамайды, және бізге minutes мәнін тоқтатып, оны қайтадан 0-ге қайтару қажет. Есеп modulo операторын қолдану арқылы шығарылады: егер біз әр жолы өсірілген мәнді 60-қа бөлетін болсақ, біз қалаған мәнімізді нақты аламыз. Біз minutes-ті 58 мәнін қойып, оны бірнеше рет өсірсек не болатындығын көре аламыз:

```
>>> minutes = 58
>>> minutes = (minutes + 1) % 60
>>> minutes 59
>>> minutes = (minutes + 1) % 60
>>> minutes 0
>>> minutes = (minutes + 1) % 60
>>> minutes 1
```

Minutes 0- ден 58 дейін (minutes + 1) % 60 аралықта болғанда minutes + 1 беретін нәтижені береді, себебі minutes + 1 % 60 қарағанда аз. Бірақ minutes 59 тең болғанда, (minutes + 1) % 60 60 сәйкес келеді, яғни 0-ге тең. Бұл мысал modulo операторын қолдану арқылы арифметикалық әрекеттер, ресми үлгілеу арифметикасы деп аталатын, неліктен жиі «арифметикалық сағаттар» деп аталатындығын көрсетеді.

Жаттығулар

Келесі сұрақтарға жауап беру үшін Python түсіндіргішін пайдаланыңыз. Қажет болған жағдайда, жауап алу үшін пайдаланған Python өрнегі мен жауабын беріңіз.

2.3.1. Адам ағзасындағы әр жасауша 6 миллиард жұп негіздеу ДНҚ-сын құрайды (23 хромосомадан құрылған әр жинақта 3 миллиардтан). Әр негіздеу жұбы арасындағы қашықтық шамамен 3.4 ангстремді (3.4×10^{-10} метрді) құрайды. Егер оны шешіп, созса, онда адамның бір жасушасының ДНҚ-сы қандай ұзындықты құрайды? Адам ағзасында шамамен 50 триллион жасуша бар. Егер сіз барлық ДНҚ-ларды созып ұштарын біріктірсеңіз, олардың ұзындығы қандай болар еді?

Бұл арақашықтық күннен жерге дейін әрі-бері қандай бола еді? Күнге дейінгі арақашықтық шамамен 149,598,000 километр. Осы әр үш сұраққа жауап бері үшін Python-да өрнек жазыңыз. Келесі сұраққа жауап бері үшін әр мәнді сақтау үшін ауыспалыны бекітіңіз.

2.3.2. Radius атаулы және мәні 10-ға тең ауыспалыны енгізіңіз. Шеңбер ауданын есептеу ($A = \pi r^2$) үшін формуланы қолдана отырып, сіздің ауыспалы radius-қа тең area атаулы жаңа ауыспалыны табыңыз. (10 саны формулада болмауы керек).

2.3.3. Енді radius мәнін 15-ке дейін өзгертіңіз. Енді Area мәні қандай? Неліктен?

2.3.4. Мысал, біз x және y-ке берілген екі ауыспалыны орын ауыстырғымыз келеді. Неліктен келесі өрнек жұмыс істемейді? Жұмыс жасайтын тәсілді ұсыныңыз.

$$x = y$$

$$y = x$$

2.3.5. Келесі әрекеттерді іске асырғанан кейін x және y мәндері қандай? Бұл неліктен осылай екендігін түсінетіндігіңізге көз жеткізіңіз.

$$x = 12.0$$

$$y = 2 * x * x = 6$$

2.3.6. Келесі әрекеттерді іске асырғанан кейін x мәні қандай болады? Неліктен осылай екендігін түсінетінін көз жеткізіңіз.

$$x = 0$$

$$x = x + 1 \quad x = x + 1 \quad x = x + 1$$

2.3.7. Келесі әрекеттерді іске асырғанан кейін x және y мәндері

қандай? Бұл неліктен осылай екендігін түсінетіндігіңізге көз жеткізіңіз.

$$x = 12.0$$

$$y = 6$$

$$y = y * x$$

2.3.8. Егер толық санды түрге жататын x ауыспалы мәні берілсе, бірлік, ондық және жүздік орындарында тұрған сандарды мәннен алып шығу тәсілін ұсыныңыз. (// және . операторларын қолданыңыз)

2.4 ФУНКЦИЯЛАРДЫ ПАЙДАЛАНУ

Python тәріздес бағдарламалау тілі бай жиынтықты функциялық абстракцияларға ие, олар бізге кең қызықты есептерді шешуге мүмкіндік береді. ** сияқты ауыспалылар атаулары мен арифметикалық операторлар оның қарапайым мысалдары болып табылады. Ауыспалылар, атаулар, есептеулер нәтижесіне олардың нақты мәндерін (мысалы, volume орнына 33510.29333333333), жадқа сақтаусыз сілтеме жасауға, ал ** сияқты операторлар аса қиын операцияларды айқын детализациялаусыз іске асыруға мүмкіндік береді (мысалы, бірнеше реттік көбейту алгоритмі орнына деңгейлер жазу).

Стандартты арифметикалық операцияларға қоса Python әрқайсысы өз алдына функционалды абстракция болып табылатын бірнеше күрделі математикалық функцияларды ұсынады. Функция шығысында бір немесе тәуелді айналмалы шама деп аталатын бірнеше баптауларға ие және оларға негізделе отырып нәтиже береді.

Мүмкін Сіз $f(x) = x^2 + 3$ сияқты математикалық функцияларды бұрында көрген шығарсыз.

Бұл жерде, f - бұл функциялар, x – жалғыз тәуелді айналмалы шама, ал $f(x)$ нәтиже болып табылды.

Нәтиже теңдік белгісінің оң жағында берілген қарапайым алгоритм көмегімен кіріс мәліметтер (яғни тәуелді айналмалы шама) негізінде есептелінеді. Бұл жерде алгоритм «кірісті шаршыға келтіріп оған 3 қосыңыз» болып табылады. Егер x -ке кіріс баптауының белгілі бір мәнін берсек, онда алгоритм нәтиженің белгілі бір мәнін береді. Мысалы, егер x 4-ке тең болса, онда төменде көрсетілгендей нәтиже $f(4) = 4^2 + 3 = 19$ сәйкес болады.



Бұл таныс болып көріне ме? 1.1-суретіндегі мысал есепте көрсетілгендей (1 тарау басында), математикалық функция кіріс мәліметтеріне, нәтижеге және кіріс мәліметтерінен нәтижені есептейтін алгоритмге ие. Осылайша, математикалық функция – функционалды абстракцияның тағы бір түрі болып табылады.

Кіріктірілген функциялар

Біз жоғарыда болжамдағандай, бағдарламалау тілінде функционалды абстракция функция түрінде де беріледі. 3 тарауда біз, түрлі функционалды абстракцияларды іске асыру үшін өзіміздің жеке функцияларды қалай анықтауды үйренеміз, олардың көбісі қарапайым математикалық функцияларға қарағанда аса кең болып табылады. Осы бөлімде біз Python-ның кей бір кіріктірілген функцияларын қалай қолдануға болатындығын бақылап, негіз қалаймыз. Мүмкін, олардың ең қарапайымы Python-дағы өз тәуелді айналмалы шамаға абсолюттік мән беретін (яғни үлгі) `abs` функциясы болып табылар.

```
>>> result2 = abs(-5.0)
>>> result2 5.0
```

Атау беру операторының оң жағында функция шақыру функциясы бар, сондай-ақ ол функция бастамалауы деп те аталады. Бұл нақты жағдайда, `abs` функциясы `-5.0`. тәуелді айналмалы шамамен шақырылады.

Функция шақыруды `abs` функциясымен анықталған алгоритм (жасырын) атқарады, және `abs(-5,0)` функция шығысын (нәтиже) есептейді, бұл жағдайда `5.0`. Сондай-ақ, функция нәтижесі функцияның қайтарылма мәні деп аталады. Осыған ұқсас, функция нәтиже мәнін қайтарады деп айта аламыз. Функция шақыру қайтарылма мәні, бұл жағдайда `5.0`, сосын оған `result2` ауыспалы атауы беріледі.

Басқа екі пайдалы функциялар – бұл float және int. Float функциясы өз тәуелді шама айналмасын қалқымалы нүктелі санға түрлендіреді, егер ондай болмаған жағдайда

```
>>> float(3) 3.0
>>> float(2.718) 2.718
```

Int функциясы өз тәуелді шама айналмасын оны кесу арқылы толық санға түрлендіреді, яғни ондық нүктеден бөлшекті бөлігін жоя отырып. Мысалы,

```
>>> int(2.718) 2
>>> int(-1.618)
-1
```

2.3 Бөліміндегі жел ескерумен температура есептеуді еске түсірейік:

```
>>> temp = -3
>>> wind = 13
>>> chill = 13.12 + 0.6215 * temp + (0.3965 * temp - 11.37) * wind**0.16
>>> chill
-7.676796032159553
```

Әдеттегідей, біз жел жылдамдығы ескерілген қорытынды температураны толық санға дейін жуықтағымыз немесе қысқартқымыз келеді. («Бүгін жел жылдамдығын ескергендегі температура шамамен -7.676796032159553 °C» деп айтып тұрған жерлігікті телеарна метеорологты есту таңғаларлық болар еді.) Біз температура мәнін қысқартуды келесідей оңай есептей аламыз:

```
>>> truncChill = int(chill)
>>> truncChill
-7
```

Сондай-ақ, температураны жуықтау үшін біз round функциясын қолдана аламыз.

```
>>> roundChill = round(chill)
>>> roundChill
-8
```

Нұсқа ретінде, егер сізде chill бастапқы мәнін сақтауда қажеттілік болмаса, біз жай ғана өзгерілген мәнге chill атауын бере аламыз:

```
>>> chill = round(chill)
>>> chill
-8
```

Функциялар аргументтері қарапайым бөлек тұрақты шама мен ауыспалыға қарағанда аса күрделі болулары мүмкін; олар есептемелік мәнге ие болулары мүмкін. Мысалы, біз жел жылдамдығы ескерілген температураны Фаренгейт градустарына түлендіріп, сосын нәтижені жуықтағымыз келсе, онда біз мұны келесідей іске асыра аламыз:

```
>>> round(chill * 9/5 + 32) 18
```

Ең алдымен жақшалардағы өрнектер есептелінеді, сосын осы өрнек нәтижесі round функциясының тәуелді шама айналымы ретінде пайдаланылады.

Барлық функциялар қандай да бір нәтижеге ие бола бермейді. Мысалы, print функциясы, ол өз тәуелді шама айналымын экранға шығарады, бірақ жасамайды. Мысалы, келесіні байқап көрейік:

```
>>> x = print(chill)
-8
>>> print(x) None
```

X ауыспалысына print функциясы қайтарған мән берілген, бірақ бұл

жазылғаннан айрықша басып шығару. Егер біз `x` деп жазсақ, онда `None` атаулы бір нәрсе бекітілгенін көреміз. `None` – Python басты сөзінің бірі, ол өзімен «бір нәрсе» деген мағынаны білдіреді. Кез келген қайтарылатын мәнді анықтамайтын функция `None`-ды үнсіз келісім бойынша қайтарады. Біз бұған өзіміздің жеке функцияларымызды анықтағанда жақын арада қайта ораламыз.

Жолдар

«Сәлем әлем!» бірінші бағдарламасындағыдай, біз `print` функциясын мәтін басып шығару үшін пайдаланамыз:

```
>>> print(Маған «Жасыл жұмыртқалар мен ветчина» кітабі ұнамайды)
Маған «Жасыл жұмыртқалар мен ветчина» кітабі ұнамайды.
```

Жақша ішіндегі сөйлем жол деп аталады. Жолдар – бұл бір реттік (‘) немесе (“) екі реттік жақшалармен қоршалған символдар реттілігі, не қарапайым мәтін болып табылады. `Print` функциясы жақшалар ішіндегі символдарды экранға шығарады. Шын мәнінде `print` функциясы үтірлермен бөлінген бірнеше тәуелді шама айналымдарға ие болуы мүмкін:

```
>>> print('Жел жылдамдығы ескерілген температура chill, Цельсия
градусын құрайды.') Жел жылдамдығы ескерілген температура -8
Цельсия градусын құрайды.
```

Бірінші және соңғы тәуелді шама айналымдары- жолдар, ал екінші – біз жоғарыда анықтаған ауыспалы болып табылады. Тәуелді шама айналымдар арасында бос орын және `chill` ауыспалы атау маңайында жақшалар жоқтығына назар аударыңыз.

2.2- ойлануға арналған сұрақ. Қалай ойлайсыз, жолдар айналасында жақшалар не үшін қажет? Қалай ойлайсыз, жақшалар орнына келесіні тергенде сіз қандай қатеге тап боласыз? (Байқап көріңіз).

```
>>> print (Жел ескерілген болжамалы температура, chill, Цельсия
градусын құрайды).
```

Жақшалар қажет, себебі олар болмаған жағдайда Python мәтінді ауыспалыдан немесе функциядан ажырата алмайды. Жақшаларсыз Python түсіндірушісі жақша ішіндегі әр тәуелді шама айналым мәнін түсінуге тырысады, себебі ол әр сөз ауыспалы атауы немесе функция деп, не Python тіліндегі резервтелген сөз деп түсінуі мүмкін. Бұл сөздер реттілігі тіл синтаксисіне сәйкес келмейтіндіктен және берілген атаулар көбісі анықталмағандықтан түсіндіргіш оларды қате деп көрсетеді.

Сонымен қатар, жолдық мәндер ауыспалыларға берілген және + мен * басқарылатын болуы мүмкін, бірақ операторларда басқа мәндер болады. + операторы екі жолды бір ұзын жолға жалғастырады.

Мысалы,

```
>>> first = 'Monty'
>>> last = 'Python'
>>> name = first + last
>>> print(name) MontyPython
```

Бірінші және соңғы атаулар арасына бос орын қалдыру үшін біз тағы бір + операторын қолдана аламыз:

```
>>> name = first + ' ' + last
>>> print(name) Monty Python
```

Нұсқа ретінде, егер біз жай ғана толық атауды басып жазғымыз келсе, біз name ауыспалысын мүлдем ескермейтін едік.

```
>>> print(first + ' ' + last) Monty Python
```

2.3-ойлануға арналған сұрақ. Неге біз жоғарыда көрсетілген first және last өрнектеріне жақшаларды қолдануды қаламадық? Егер біз жақшаларды қолдансақ не болар еді?

First және last айналасында жақшалар қолдануда Python оларды ауыспалылар емес жолдар ретінде түсіндіреді:

```
>>> print('first' + ' ' + 'last') first last
```

Жолдарға қолданарлықтай, * операторы қайталау операторына айналады, ол жолды бірнеше рет қайталайды. Қайталау операторының сол жақтағы операнды - бұл жол, ал оң жақтағы операнд толық санды болып табылады, себебі ол жолды қанша рет қайталау керектігін көрсетіп тұрады.

```
>>> last * 4 'PythonPythonPythonPython'
>>> print(first + ' ' * 10 + last) Monty Python
```

Сондай-ақ, біз интерактивті тәртіпте өзіміздің бағдарламаларымызда input функциясы арқылы жолдық енгізуге тапсырыс бере аламыз. Input функциясы сұрақ жолын тәуелді шама айналмасы ретінде алып, жауапта басып жазылған жолдық мәнді қайтарады. Мысалы, нұсқама сіздің атыңызды сұрап, сәлемдемені шығарады.

```
>>> name = input('Сіздің атыңыз қандай?') Сіздің атыңыз қандай?
Геооргий
>>> print('Сәлем,', name, '!') Сәлем, Геооргий !
```

Input функциясын жоғары шақыру 'What is your name? ' жолын шығарып, сосын күтеді. Сіз бір нәрсе басып жазып (жоғарыда біз George деп жаздық, ол жазу қызылмен көрсетілген), енгізуді басқаннан кейін, экранға шыққан мәтін input функциясымен қайтарылады және name атаулы ауыспалыға беріледі. Сосын name print функциясында қолданылады.

2.4 Ойлануға арналған сұрақ Жоғары леп белгісі алдында бос орын қалдыруды болдырмау үшін біз + операторын қалай қолдана аламыз?

Бос орынды болдырмау үшін, біз жолды құрастыра аламыз, print функциясымен тәуелді шама айналымы арасында бос орын қалдыруға рұқсат еткенше, ол жолды байланыс операторы көмегімен басып жаза аламыз:

```
>>> print('Сәлем, ' + name + '!') Сәлем, Геооргий!
```

Егер біз сан мәнін енгізгіміз келсе, ол санды сан ретінде қолданар алдында, біз input функциясымен қайтарылатын жолды толық санға немесе int не float функцияларын қолдана отырып қалқымалы нүктесі бар санға түрлендіре аламыз. Мысалы, келесі нұсқаулықтар сіздің жасыңызды сұрайды, сосын сіз өмір сүрген сандар көлемін басып шығарады.

```
>>> text = input('Сіз қанша жастасыз?') Сіз қанша жастасыз? 18
>>> age = int(text)
>>> print('Сіз өмір сүресіз', age * 365, 'күн!') Сіз 6570 жыл өмір сүрдіңіз!
```

Жоғарыдағы сұраққа жауап беру үшін біз 18 санын тердік (қызылмен көрсетілген). Әрі қарай, input функциясы біз енгізген мәнді text ауыспалы жол түрде, яғни «18» ретінде берді. Сосын, int функциясымен жол 18 (жақшасыз) толық мәніне түрленді және оған age ауыспалысы берілді. Енді, age – бұл сандық мән болғанды, оны age * 365 арифметикалық өрнек түрінде қолдануға болады.

2.5- ойлануға арналған сұрақ age = int(text) ескермей жоғарыда берілген нұсқаулықты енгізіңіз.

```
>>> text = input('Сіз қанша жастасыз?') Сіз қанша жастасыз? 18
>>> print('Сіз өмір сүрдіңіз', text * 365, 'күн!')
```

Қандай өзгерістер болды? Неліктен?

Text мәні толық сан емес жол болып табылатындықтан, * операторы қайталау операторы деп түсіндірілген!

Кейде бізге түрлендіруді кері бағытта атқару қажет болады, сандық мәннен жолға қарай. Мысалы, бізге келесі формада күндер санын жазу орнына, күндер саны мен нүкте арасындағы бос орынды жою орынды болар еді.

```
>>> print('Сіздің өміріңіздегі күндер саны', age * 365, '.') Сіздің өміріңіздегі күндер саны 6570 тең.
```

Бірақ келесі әрекеттер жұмыс жасамайды, себебі біз сандық мәнді байланыстыра алмаймыз.

```
age * 365 жолмен '.'.
```

```
>>> print('Сіздің өміріңіздегі күндер саны', age * 365 + '.') TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Бұл есепті шығару үшін, алдымен бізге age * 365-ті str функциялы жолға түрлендіру қажет:

```
>>> print('Сіздің өміріңіздегі күндер саны', str(age * 365) + '.') Сіздің өміріңіздегі күндер саны 6570 тең.
```

Болжап көрейік, age-ке 18 мәні берілді, бұл жағдайда str функциясы 6570 толық мәнді '.' белгісімен байланыстырап алдында '6570' жолға түрлендіреді.

Қорытындылай келе, жел жылдамдығын ескере отырып ағымдағы температураны есептеу үшін температура мен жел жылдамдығын алу үшін қалай сұраныс жасау күрделірек мысалын қарастырайық.

```
>>> text = input('Температура: ') Температура: 2.3
```

```
>>> temp = float(text)
```

```
>>> text = input('Жел жылдамдығы: ') Жел жылдамдығы: 17.5
```

```
>>> wind = float(text)
```

```
>>> chill = 13.12 + 0.6215 * temp + (0.3965 * temp - 11.37) * wind**0.16
```

```
>>> chill = round(chill)
```

```
>>> print('Жел жылдамдығын ескерілген температура', chill, 'Цельсия градусын құрайды.') Жел жылдамдығын ескерілген температура -2 Цельсия градусын құрайды
```

2.5 Ойлануға арналған сұрақ. Жоғарыда біз неліктен *int* орнына *float* қолдандық? *Float* функциясын шақырудың бірін *int* шақырумен алмастырыңыз, сосын қалқымалы нүктесі бар мәнді енгізіз (мысалы 2.3) Не өзгерді және неге?

```
>>> text = input('Температура: ') Температура: 2.3
```

```
>>> temp = int(text)
```

Үлгілер

Python-де math деп аталатын үлгіде қолжетімді көптеген математикалық функциялар бар. Үлгі – Python-ның жасаған бағдарламасы, ол қолдануға болатын алдын ала анықталған мәндер мен функцияларды құрайды. Үлгі құрамына қол жеткізу үшін, біз import басты сөзін қолданамыз. Math үлгісіне қол жеткізу үшін келесіні басып жазыңыз:

```
>>> import math
```

Үлгі импорттаудан кейін, біз, қолданғымыз келетін функция атауының алдына нүктемен бөлінген (.) үлгі атауын жазу арқылы, үлгі функцияларына қол жеткізе аламыз. Мысалы, 5-ң шаршы түбірін алу үшін, біз math.sqrt шаршы түбірі функциясын қолдана аламыз:

```
>>> math.sqrt(5) 2.23606797749979
```

Math үлгісінің жиі қолданылатын басқа функциялары Б.1 қосымшасында берілген. Сондай-ақ, math үлгісі жиі қолданылатын екі константтардан тұрады; pi және e. Біз өзіміздің алдыңғы көлем мәнін келесі түрмен есептеу арқылы анықтай аламыз:

```
>>> radius = 20
>>> volume = (4 / 3) * math.pi * (radius ** 3)
>>> volume 33510.32163829113
```

Pi және e – функция емес, ауыспалы атауы екендігіне назар аударыңыз, себебі атаудан кейін домалақ жақшалар жоқ.

Функция шақыру, сонымен қатар, аса ұзын өрнектерде және басқа функциялар үшін тәуелді шама айналымы ретінде қолданылады. Бұл жағдайда, функция шақыруды ол қайтаратын мән баламасы ретінде түсінген жөн. Мысалы, $V = h^3 / (6\sqrt{2})$ формуласын қолдана отырып, қабырға ұзындығы $h = 7.5$ болатын тетраэдра көлемін есептеуде math.sqrt функциясын қолдануға болады.

```
>>> h = 7.5
>>> volume = h ** 3 / (6 * math.sqrt(2))
>>> volume 49.71844555217912
```

Ең алдымен math.sqrt (2) жақшалар ішіндегі мән есептелінеді, сосын

ол мән 6 көбейтіледі. Және қорытындылай келе, $h ** 3$ сол нәтижеге бөлінеді, ол жауабы `volume`-ге беріледі. Егер біз жуықталған нәтиже алғымыз келсе, біз барлық көлем есептеуді `round` функциясының тәуелді шама айналымы ретінде қолдана аламыз:

```
>>> volume = round(h ** 3 / (6 * math.sqrt(2)))
>>> volume 50
```

Осы есептеудегі барлық әрекеттер реттілігін көрсетейік:

$$\underbrace{\underbrace{h ** 3}_{421.875} / \underbrace{(6 * \text{math.sqrt}(2))}_{1.4142\dots}}_{8.4852\dots} = \underbrace{\quad}_{49.7184\dots} = 50$$

Енді біз 52° бұрышындағы косинусын тапқымыз келеді деп болжамдайық. Косинусты есептеу үшін `math.cos` функциясын қолдануға болады, бірақ Python тригонометриялық функциялары олардың тәуелді шама айналымы градууста емес, радианда болады деп күтеді. (360 градус 2π радианға тең). Ыңғайына қарай, `math` үлгісі `radians` атаулы функцияға ие, бұл функция градусты радианға түрлендіреді. Осылайшв, біз 52° бұрышының косинусын таба аламыз:

```
>>> math.cos(math.radians(52)) 0.6156614753256583
```

Алдымен, 52° радианды балама бере отырып `math.radians(52)` функциясын шақырылады, ал бұл нәтиже `math.cos` функциясының тәуелді шама айналымы ретінде қолданылады:

$$\underbrace{\text{math.cos}(\underbrace{\text{math.radians}(52)}_{0.9075\dots})}_{0.6156\dots}$$

Қорытындылай келе, егер сізге есептеуді кешенді сандармен іске асыруға тура келетін болса, `math` белгісі орнына `cmath` үлгісін қолдануға болады. Қос үлгідегі функциялар атауы, көп жағдайда, бірдей болып

келеді, бірақ `cmath` үлгісіндегі нұсқалар кешенді сандарды түсінеді. Мысалы, `math.sqrt` қолдана отырып, теріс саның шаршы түбірін табу әрекеті қатеге алып келеді:

```
>>> math.sqrt(-1)
ValueError: math domain error
```

Бұл *мән саласының қатесі* -1 мәні `math.sqrt` функциясы үшін мән саласының шегінде екендігін көрсетеді. Басқа жағынан, 1 табу үшін `cmath.sqrt` функциясын шақыру і мәнін қайтарады:

```
>>> import cmath
>>> cmath.sqrt(-1) 1j
```

Жаттығулар

Келесі сұрақтарға жауап беру үшін Python түсіндіргішін қолданыңыз. Қажет болған жағдайда, жауап алу үшін қолданған Python жауабы мен өрнегін көрсетіңіз.

2.4.1. -8 бен -8.0 үшін абсолюттік мәнді табуға тырысыңыз. Сіз нені байқадыңыз?

2.4.2. Радиусы 10 болатын шеңбер ауданын табыңыз. `Math` үлгісінің π мәнін қолданыңыз.

2.4.3. Екі санның орташа геометриясы – бұл олар шығармасының шаршы түбірі. 18 бен 31 орташа геометриясын табыңыз.

2.4.4. Болжамдап көрейік, сіздің жинақы шотыңызда r пайыз мөлшерлемелі P долларыңыз бар, оның жылдық жинақталуы n тең. t жылдан кейін сіздің шотыңызда

$$P \left(1 + \frac{r}{n} \right)^{nt}$$

доллар болады. Егер пайыздар тұрақты жиналып тұрса (яғни, n шексіздікке ұмтылуда), Сізде оның орнына

$$Pe^{rt}$$

t жылдан кейін доллар болар еді, бұл жерде e – Эйлер саны, натуралды логарифма негізі.

Болжап көрейік, сіздің шотыңызда жылдық 1% ($r = 0.01$) төлейтін

айлық жинақтаумен $P = \$10,000$ бар. $t = 10$ жылдан кейін сізде қанша ақша болады? Егер бұның орнына пайыздар үздіксіз жинала беретін болса, 10 жылдан кейін сіз қанша ақша алар едіңіз? (Math.exp функциясын қолданыңыз.)

2.4.5. X қалқымалы нүктесі бар кейбір оң сандарды, ондық нүктенің оң жағындағы екі белгіге қысқарту үшін, `int` функциясын қалай пайдалауға болатындығын болжап көріңіз. Басқа сөздермен айтқанда, сіз 3.1415926 сияқты санды 3.14 дейін қысқартқыңыз келеді. Сіздің өрнегіңіз X -ің кез келген мәнінде жұмыс істеуі керек.

2.4.6. Қалқымалы нүктесі бар кез келген, оң санның бөлшекті бөлігін табу үшін `int` функциясын қалай қолануға болатындығын болжамдап көріңіз. Мысалы, егер 3.14 мәні x берілсе, Сіз шығыста 0.14 мәнін аласыз. Сіздің өрнегіңіз X -ің кез келген мәнінде жұмыс істеуі керек.

2.4.1. Төменде көрсетілген шаршылы деңгей түбірін табуға арналған бәріне мәлім формула $ax^2 + bx + c = 0$. $ax^2 + bx + c = 0$ шаршылы өрнекті шешуге мүмкіндік береді.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$3x^2 + 4x - 5 = 0$ өрнектің екі түбірін табу үшін бұл формуланы қалай қолдануға болатындығын көрсетіңіз.

2.4.2. Байқап көрейік, сізде (x_1, y_1) и (x_2, y_2) екі нүкте бар. Олардың арасындағы қашықтық

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Оны Python-да қалай есептеуге болатындығын болжап көріңіз. Өз есептемелеріңізді шынайы мәндерде тексеріңіз.

2.4.3. Параллелепипед алты параллелограммалы жақтары бар үш өлшемді қорап болып табылады. Параллелепипед көлемі

$$V = abc\sqrt{1 + 2\cos(x)\cos(y)\cos(z) - \cos(x)^2 - \cos(y)^2 - \cos(z)^2}$$

Бұл жерде, a , b және c – қабырғалар ұзындығы, ал x , y және z – радианда берілген қабырғалар арасындағы бұрыштар. Оны Python-да қалай есептеуге болатындығын болжамдаңыз. Өз есептемелеріңізді шынайы мәндерде тексеріңіз.

2.4.4. Бұрыштар сізге градууста берілгендігіне көз жеткізе оты-

рып, алдыңғы жаттығуды қайталаңыз.

2.4.5. 2.4.4 Жаттығуын қайталаңыз, бірақ, алдымен input функциясын пайдалана отырып, әр төрт мәнге сұраныс жасаңыз, сосын нәтижені экранға шығарыңыз.

2.4.6. 2.4.7 Жаттығуын қайталаңыз, бірақ алдымен input функциясын пайдалана отырып, b, және c үшін әр мәнге сұраныс жасаңыз, сосын нәтижені экранға шығарыңыз.

2.4.7. Келесі бағдарлама ессіз кітапхананы іске асырады.

```
adj1 = input('Сіз хайуанаттар бағына бара жатырсыз. Қоршағандар  
көзінде Сіз қандайсыз? : ')\n noun1 = input('Сіз хайуанаттар бағында кімге бірінші болып  
жақындайсыз? : ')\n noun2 = input('Кімнің торы Сіз үшін бой түршіктіретін? : ')\n adj2 = input('Жолбарыс бүгін қандай көңілде? : ')\n noun3 = input('Хайуанаттар бағынан кейін Сіз қайда барасыз?: ')\n print('Қызықты кешті қалай ұйымдастыруға болады')\n print()\n print('Егер сіз іздесеңіз', adj1, ' тәсіл')\n print('Өз қалауыңызды белгілеңіз', noun1 + ', костюмдік қызықты  
кешке қалай қарайсыз')\n print(noun2 + '? Келесіден бастаңыз')\n print('жасырылған шақыруларды жіберу', adj2, ' форматта,')\n print(' бақыттаушы ', noun3 + '.')
```

Қою үшін, жоқ дегенде бес бөлікті талап ететін өзіңіздің жеке ессіз кітапхана бағдарламаңызды жазыңыз. (Ең алдымен, сіз бағдарламаны кітап веб-сайтынан жүктей аласыз).

*2.5 БИНАРЛЫ АРИФМЕТИКА

1.4 Бөлімінен еске салайық, компьютер жадында сандар екілік жазуда сақталады. Осылайша, компьютерлер арифметикалық операцияларды екілік жүйеде іске асырулары керек. Осы бөлімде біз бинарлы қосылулар қалай жұмыс жасайтындығын қысқаша қарастырамыз.

Бұл таныс емес болып көрінгенімен, бинарлы қосылулар шын мәнінде ондық қосылуларға қарағанда өте оңай, себебі бинарлы қосылулардың тек үш негізгі пайымдаулары бар: $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$ и $1 + 1 = 10$. (екілік жүйедегі 10 ондық жүйеде 2 мәніне ие екендігіне назар аударыңыз). Осы негізгі пайымдауларды қолдана отырып, біз бастауыш

мектепте оқыған кез келген еркін ұзындықты екілік сандарды қосып алгоритмін солдан оңға қарай пайдалана аламыз.

Мысалы, 1110 және 0111 екілік сандарын қосып көрейік. Оң жақтан бастап біз соңғы бағанды бастаймыз: $0 + 1 = 1$.

$$\begin{array}{r} 1110 \\ + 0111 \\ \hline \end{array}$$

Келесі бағанда біз мынадай мәнге иеміз $1 + 1 = 10$. $1 + 1 = 10$. Жауап бір биттен артық болғандықтан, біз 1-ді ауыстырамыз.

$$\begin{array}{r} 1 \\ 1110 \\ + 0111 \\ \hline 01 \end{array}$$

Келесі бағанда бізде тағы да $1 + 1 = 10$, бірақ енді ауыстырылған битті ескере отырып, ауыстыруды қосқанда, біз мына мәнді аламыз $10 + 1 = 11$ (немесе ондық мағынада $2 + 1 = 3$). Осылайша, бағандағы жауап – 1, 1 тең ауыстырумен.

$$\begin{array}{r} 11 \\ 1110 \\ + 0111 \\ \hline 101 \end{array}$$

Қорытындылай келе, ауыстыруды ескере отырып, шеткі сол бағанда мағына $1 + 0 + 1 = 10$ тең. 0 жазамыз, 1 ауыстырамыз, осымен дайын.

$$\begin{array}{r} 111 \\ 1110 \\ + 0111 \\ \hline 10101 \end{array}$$

Біз барлығын ондық жүйеге түрлендіре отырып нәтижені оңай тексере аламыз. Ондық жүйедегі жоғарғы сан $8 + 4 + 2 + 0 = 14$ тең, ал ондық жүйедегі төменгі сан $0 + 4 + 2 + 1 = 7$ тең болады.

Ондық жүйедегі біздің жауабымыз $16 + 4 + 1 = 21$. Әрине, $14 + 7 = 21$.

Шектелген нақтылық

Python толық санда кез келген көлемді мәндерді сақтай алғанымен, бұл

машиналық тіл деңгейіндегідей емес. Басқаша айтқанда, Python толық сандары компьютердің аңғал мүмкіндігіне негізделіп жасалғанымен, олар тағы да бір абстракция болып табылады. Машиналық тіл деңгейінде (және көптеген басқа бағдарламалау тілдерінде), әр сан шектелген жад көлемінде сақталады, көбінесе төрт байттықта (32 бит). Бұл есептеудің қорытынды нақтылығының тағы бір мысалы, және бұл арифметикалық операциялар нәтижелері жадқа сыю үшін, кейде аса көлемді болып табылатындығын меңзейді. Біз алдыңғы есепті қайта қарастыра отырып мұны көрсете аламыз, бірақ, әр толық санды сақтау үшін тек төрт битке ие бола отырып, біз 1110 және 0111 сияқты төрт разрядты толық сандарды қосқанда, біз 10101 соммасын аламыз, бұл сомманы көрсету үшін бес бит керек. Компьютер осындай жағдайға тап болған кезде, ол шеткі сол битті шығарып тастайды. Біздің мысалда, бұл ондық есептеу жүйесінде 5 тең 0101 дұрыс емес нәтижеге алып келді.

Біңғайына қарай, бұл қалай болатындығын анықтайтын тәсіл бар, бірақ оны біз сізге жаттығу ретінде зерттеу үшін қалдырамыз.

Толық теріс сандар

Біз жоғарыда айтып өткендей, толық сандарды қосуда олар оң немесе бағдарламалау терминіне сәйкес белгілерсіз болады. Бірақ, әрине компьютерлер оң және теріс белгілері бар сандарға арифметикалық операциялар жасау мүмкіншілігіне ие болулары тиіс.

Барлығы, тіпті теріс белгілер де, компьютерде екілік жазуда сақталулары керек. Терісі сандарды ұсыну нұсқасының бірі – белгілер ұсыну үшін бір битті санды жай ғана кейінге сақтау, мысалы оң сандар үшін 0 және теріс сандар үшін 1. Мысалы, егер біз санды сегіз битте сақтап, бірінші битті белгі үшін қалдырып қойсақ, онда, 00110011 мәні 51, ал 10110011 мәні -51 көрсетеді. Бұл тәсіл сандарды берудің тура коды деп мәлім. Бұл тәсілдің қиындығы сан теріс пе, жоқ па екендігін компьютер анықтап, оны арифметикалық әрекеттер жасау уақытында өңдеуі қажет.

Мысалы, болжап көрейік, біз -51 және 102 мәнін сандар беру тура кодында қосқыңыз келеді. Бұл жазу формасында -51 – бұл 10110011, ал 102 – бұл 01100110. Ең алдымен атап өтейік, 10110011 теріс болып табылады, себебі оның шеткі сол биті 1-ге ие, ал 01100110 оң болып табылады, себебі оның шеткі сол биті 0-ге ие. Осылайша, біз 102-ден 51-дің оңын алып тастауымыз керек:

			0	10	10		0	10	10		
	0	~~1~~	~~1~~	$\emptyset$	0	~~1~~	~~1~~	$\emptyset$		← 102	
-	0	0	1	1	0	0	1	1		← 51	
	0	0	1	1	0	0	1	1		← 51	

Екілік жүйедегі алыс-беріс тура ондық жүйедегідей жұмыс істейді, бірақ біз 10 орнына 2-ні қарызға бердік (екілік жүйедегі 10). Қорытындылай келе, нәтиже белгісін оң қалыпта қалдырамыз, себебі ең үлкен операнд оң болған.

Осындай қиындықтарды болдырмау үшін, компьютер қосымша жүріс деп аталатын аса қу тәсілді пайдаланады. Қосымша код көрінісінде сақталатын толық сандар олардың белгілеріне тәуелсіз тікелей қосыла алады. Сондай-ақ, шеткі сол бит қосымша код көрінісіндегі белгі үшін бит болып табылады, және толық санға берілген биттерді толтыру қажет болса, оң сандар басында нөлдері бар қарапайым ретте сақталады. Оң сандарды, теріс баламаға түрлендіру үшін шеткі оң 1-ден әр битті терістейміз. Мысалы, 51 сегіз бит 00110011 ретінде берілгендіктен, -51 мәні 11001101 ретінде беріледі. 4 сегіз бит 00000100 ретінде берілгендіктен, -4 бит 11111100 ретінде беріледі.

Қосымша код көрінісінде қосу қалай жұмыс жасайтындығын көрсету үшін, -51 және 102-ні тағы бір рет қосып көрейік:

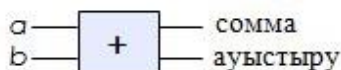
	1	1			1	1				
		1	1	0	0	1	1	0	1	← -51
+	0	1	1	0	0	1	1	0		← 102
	~~1~~	0	0	1	1	0	0	1	1	← 51

Қосу алгоритміндегі соңғы қадам ретінде, біз әрқашан ауыстырудың қосымша битін ескеріп отырамыз. Сондықтан, шын мәнінде қосымша кодта мән келесідей болады, $-51 + 102 = 51$.

Егер жауап берілген биттер санына сыймаса, қосымша кодта дұрыс емес қате алу бұрынғыдай мүмкін екендігіне назар аударыңыз. Бірнеше жаттығулар сізге әрі қарай зерттеуге талпыныс береді.

Сумматорды жобалау

Нәтиже ретінде кірісінде екі бөлек бит алатын және шығысында екі бит беретін *сумматорды* қарастырып көрейік. Біз жауапта шеткі оң битті «сомма», ал шеткі сол битті «ауыстыру» деп атаймыз. Осылайша, біздің абстракты сумматорымыз келесі қалыпта болуын қалаймыз:



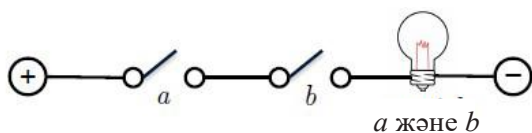
Екі бөлек биттер сол жағына кіріп, және екі нәтиже оң жағынан шығады. Біздің мақсатымыз, екі кірістен, екі шығысты есептейтін бұл «қара жәшік» ішкі бөлігін шынайы логикалық сызбаға ауыстыруда.

Бірінші қадам, суммалар мен ауыстырулар мәндерін көрсететін шындықтар кестесін жасау болып табылады, бұл кестеде барлық мүмкін кіріс мәліметтер мәндері болулары тиіс:

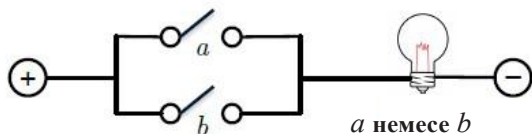
<i>a</i>	<i>b</i>	ауыстыру	сом
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

A және *b* 1-ге тең кезде, яғни біз $1 + 1$ есептегендей, ауыстыру мәні 0-ге тең екендігіне назар аударыңыз. Бұдан басқа, осы тәртіпте аталған (ауыстыру, сумма) екі кіріс биттері де, екі бит суммасы ретінде түсіндірілетіндігін ескере отырыңыз: $0 + 0 = 00$, $0 + 1 = 1$, $1 + 0 = 1$ және $1 + 1 = 10$. (Ондық жүйедегідей, алдында тұрған 0 сан мәніне ештеңе алып келмейді.)

Әрі қарай бізге осы шындықтар кестесіндегі екі шығыстың әрқайсысы үшін, теңдей бульдік өрнек жасау қажет. Сумма бағанынан бастайық. Осы бағананы логикалық өрнекке түрлендіру үшін, шығыс 1 тең бағандарға қараймыз. Осы жағдайда, бұл екінші және үшінші жолдар. Егер *a* мәні 0 тең болса, ал *b* мәні 1 тең болса, 1 тең сумманы аламыз деген ақпаратты бізге екінші жол береді. Бұл ұсыныста *және* маңызды мәнге ие; **ЖӘНЕ** логикалық өрнегі 1-ге тең болу үшін екі кірісте 1-ге тең болулары тиіс. Бірақ, бұл жағдайда *a* нөлге тең, сондықтан біз **ЖОҚ** *a* қолдана отырып, оны аударусыз керек. *B* кірісі 1 тең, сондықтан біз оны бар қалпында қалдыра аламыз. Осы екі жартыларды біріктіре отырып, бізде **ЖОҚ** *a* **ЖӘНЕ** *b* болады. Енді үшінші жол, егер *a* мәні 1-ге және *b* мәні 0-ге тең болса, біз 1 тең сумманы аламыз дейді. Тура осылайша біз осыны **a ЖӘНЕ ЖОҚ b** логикалық өрнекке түрлендіре аламыз.



(а)



2.1- сурет. Қарапайым электрлік бекітуді іске асыру (а) **ЖӘНЕ** и (б) **ЖОҚ**.

Қорытындылай келе, баған соммасы үшін осы екі өрнекті біріктірейік: бірге алынған осы екі жолдар егер a мәні 0-ге тең болғанда және b мәні 1-ге тең болғанда, *немесе*, егер a мәні 1-ге тең болғанда және b мәні 0-ге тең болғанда 1-ге тең болады деп тұжырымдайды. Басқаша айтқанда, бізге жоқ дегенде осы екі жағдайдың бірі 1-ге тең болуы қажет, яғни сома бағаны 1-ге тең болуы керек. Бұл (**ЖОҚ a ЖӘНЕ b**) **НЕМЕСЕ** (**a ЖӘНЕ $ЖОҚ b$**) мәндеріне тең. Сонымен, бұл баған соммасы үшін қорытынды логикалық өрнек.

Енді ауыстыру бағанына назар аударайық. Ауыстыру биті 1-ге тең жол, егер a мәні 1-ге және b мәні 1-ге тең болса біз 1 тең мағына аламыз деген тұжырымдама береді. Басқаша айтқанда, бұл жай ғана **a ЖӘНЕ b** болып табылады. Шын мәнінде, егер сіз ауыстыру бағанын толығымен қарасаңыз, бұл баған **a ЖӘНЕ b** үшін шындықтар кестесіндей екендігін байқайсыз. Сондықтан, ауыстыруды есептеу үшін, жай ғана **a ЖӘНЕ b** есептеу керек.

a	b	ауысты	сомма
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

a ЖӘНЕ b (ЖОҚ a ЖӘНЕ b) НЕМЕСЕ (a ЖӘНЕ $ЖОҚ b$)

Сумматорды іске асыру

Біздің сумматорды іске асыру үшін бинарлы оператордың әрқайсысын іске асыратын физикалық құрылғы қажет. 2.1- суретінде **ЖӘНЕ** операторын іске асыратын, қарапайым электрлік іске асырушы берілген. Көзіңізге елестетіп көріңіз, электрлік тоқ сол жақтағы оң клеммнен, оң жақтағы теріс клеммге жалғасуға тырысады, және табысты жағдайда шам жанады. A және b бинарлы кірістер қарапайым ауыстырып-қосқыш көмегімен іске асырылады. Ауыстырып-қосқыш ашық кезінде, ол 0-ге тең болады, ал ауыстырып-қосқыш тұйықталған кезде ол 1-ге тең болады. Нүкте қалпы шығыс ретінде болады (өшіру = 0 және қосу = 1).

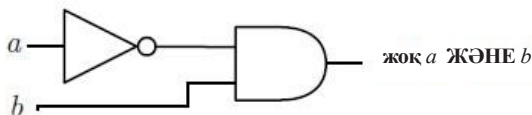
Шам тек екі ауыстырып-қосқыш тұйықталған кезде ғана жанатындығына назар аударыңыз (яғни, екі кірісте 1 тең). **НЕМЕСЕ** операторын 2.1- суретінде көрсетілген тәсілмен іске асыруға болады. (б). Бұл жағдайда, егер жоқ дегенде бір ауыстырып-қосқыш тұйықталған кезде шам жанады (яғни, егер жоқ дегенде бір кіріс 1 тең болса).



2.2-с урет. Логикалық бекітпелердің сызбалық көрінісі

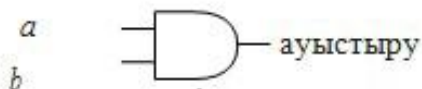
Бинарлы операторларды физикалық түрде іске асыру *логикалық бекітпелер* деп аталады. Заманауи бекітпелер электрлік сызбалармен іске асырылғанымен, олар басқа да тәсілдермен іске аса алатындығын айта өту қажет. Тапқыр ойлап шығарушылар гидравликалық және пневматикалық бекітпелерді, құрылыс ағаштары мен шығырларынан механикалық бекітпелерді, оптикалық бекітпелерді, ал жақында ДНҚ молекуласынан жасалған бекітпелерді жасап шығарды.

Логикалық бекітпелер стандарты, іске асырудан тәуелсіз сызбалық көрініске ие, бұл 2.2- с уретінде берілген. Бұл символдарды пайдалану – кез келген логикалық өрнекке сәйкес келетін *логикалық сызбаларды* жасау үшін бекітпелерді тұтастырудың ең қарапайым тәсілі. Мысалы, **ЖОҚ a ЖӘНЕ b** өрнегі келесідей көріністе болады:

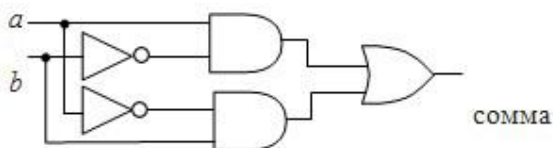


A және b екі кірісі де сол жақтан енгізіледі. A кірісі **ЖӘНЕ** бекітпесі алдында **ЖОҚ** бекітпесіне енгізіледі, бұл жағдайда **ЖӘНЕ** бекітпесінің жоғарғы кірісі бұл – **ЖОҚ** a , ал төменгі кірісі – жай ғана b . Тізбектегі жалғыз шығыстан **ЖОҚ** a **ЖӘНЕ** b мәнді **ЖӘНЕ** бекітпесін сол жағынан тастап кетеді. Осындай жолмен тиімді функцияларды атқару үшін ретті қиындық деңгейлі логикалық сызбалар құрастырылуы мүмкін.

Шығыс үшін біздің сумматордағы ауыстыру – жай ғана **ЖӘНЕ** бекітпесі:



Сомма шығуға арналған тізбек күрделірек:



«Сымшоғырлар» арасындағы жалғасуларды толығымен қара домалақтармен белгілеу қажет; егер қиылысында толығымен қара домалақтар жоқ болса, бұл бір сым, екінші сым үстінен жанасусыз шығады дегенді білдіреді. Осы жағдайда, жалғасу негізінде a кірісі бекітпесі жоғарғы **ЖӘНЕ** және төменгі **ЖОҚ** ретінде жалғанады, ал b бекітпесі жоғарғы **ЖӘНЕ**-ге және төменгі **ЖОҚ**-қа жалғасады. Жоғарғы **ЖӘНЕ** бекітпесінен a **ЖӘНЕ** **ЖОҚ** мәні шығады, ал төменгі **ЖӘНЕ** бекітпесінен **ЖОҚ** a **ЖӘНЕ** b мәні шығады. **НЕМЕСЕ** бекітпесінен осы екі мәnniң дизъюнкция (**НЕМЕСЕ**) нәтижесі шығады.

Қорытындылай келе, біз бастаған «қара жәшік» сумматорын ауыстыру үшін екі кірісімен екі шығысы бар, бұл екі сызбаны бір үлкен сумматор тізбегіне біріктіре аламыз.



a және b бастапқы мәндерінің әрқайсысы енді үш түрлі бекітпеге жалғасатынына, ал екі шығыс параллельді түрде есептелетіндігіне назар аударыңыз. Мысалы, болжамдап көрейік, a мәні 0-ге тең, ал b мәні 1-ге тең. Қорытынды шығыс мәндерін алу үшін, бұл ақпарат қалайша сумматор арқылы өтетіндігі төмендегі суретте көрсетілген.



Бұл жағдайда сумматор 0 ауыстыра отырып, $0 + 1 = 1$ деп санайды.

Жаттығулар

2.5.1. 001001 және 001101 екілік сандарын қалай белгілерсіз қосуға болатындығын көрсетіңіз.

2.5.2. 0001010 және 0101101 екілік сандарын қалай белгілерсіз қосуға болатындығын көрсетіңіз.

2.5.3. Барлық толық сандар төрт битте сақталу қажет

шартында 1001 және 1101 екілік сандарын қалай белгілерсіз қосуға болатындығын көрсетіңіз. Сіз жауапты дұрыс тапқаныңызға көз жеткізу үшін екілік сандарды ондық жүйеге түрлендіріңіз.

2.5.4. Барлық толық сандар алты битте сақталу қажет шартында 001010 және 101101 екілік сандарын қалай белгілерсіз қосуға болатындығын көрсетіңіз. Сіз жауапты дұрыс тапқаныңызға көз жеткізу үшін, екілік сандарды ондық жүйеге түрлендіріңіз.

2.5.5. Болжамдап көрейік, сізде толық сандарды белгілерсіз шектелген биттер көлемінде сақтайтын компьютер бар. Егер сіз компьютерде белгілерсіз екі толық санды қоссаңыз, сіз, алынған жауап дұрыс па жоқ па екендігін қалай тексере аласыз (дұрыс жауап алу үшін қандай да бір мәліметтер көмегіне жүгінбеңіз)? (Белгілерсіз қосу мысалын жоғарыда берілген мәтіннен қараңыз.)

2.5.6. Барлық толық сандар төрт биттерде сақталу қажет шартында қосымша кодта 0101 және 1101 екілік сандарын қалай қосуға болатындығын көрсетіңіз. Сіз жауапты дұрыс тапқаныңызға көз жеткізу үшін екілік сандарды ондық жүйеге түрлендіріңіз.

2.5.7. Төрт биттік қосымша кодта толық оң сандардың қандай ең көп көлемі берілуі мүмкін? Ең аз теріс сан қандай? (Екінші сұрақты аса назармен ойластырыңыз).

2.5.8. Барлық толық сандар төрт битте сақталу қажет шартында қосымша кодты 1001 және 1101 екілік сандарын қалай белгілерсіз қосуға болатындығын көрсетіңіз. Сіз жауапты дұрыс тапқаныңызға көз жеткізу үшін екілік сандарды ондық жүйеге түрлендіріңіз.

2.5.9. Барлық толық сандар алты битте сақталу қажет шартында қосымша кодты 001010 және 101101 екілік сандарын қалай белгілерсіз қосуға болатындығын көрсетіңіз. Сіз жауапты дұрыс тапқаныңызға көз жеткізу үшін екілік сандарды ондық жүйеге түрлендіріңіз.

2.5.10. Болжамдап көрейік, сізде толық сандарды

белгілерсіз шектелген биттер көлемінде сақтайтын компьютер бар. Егер сіз компьютерде қосымша кодты екі толық санды қоссаңыз, сіз, алынған жауап дұрыс па жоқ па екендігін қалай тексере аласыз (дұрыс жауап алу үшін қандай да бір мәліметтер көмегіне жүгінбеңіз)?

2.5.1. Екінші операнд қосымша кодына бірінші операндты қосу арқылы алып тастау амалын іске асыруға болады. Осы алгоритмді пайдалана отырып, қосымша кодтағы 1101 екілік санынан 0101 екілік санын қалай алып тастауға болатындығын көрсетіңіз. Сіз жауапты дұрыс тапқаныңызға көз жеткізу үшін екілік мәнді ондық жүйеге түрлендіріңіз.

2.5.2. Қосымша кодта 0110 екілік санынан 0011 екілік санын қалай алып тастауды көрсетіңіз. Сіз жауапты дұрыс тапқаныңызға көз жеткізу үшін, екілік мәнді ондық жүйеге түрлендіріңіз.

2.5.3. Сумматордың қорытынды сызбасын көшіріп, біз жоғарыда жасағандай a кірісі 1-ге тең болғанда, ал b кірісі 0-ге тең болғанда екі кіріс (ауыстыру мен сома) өздерінің қорытынды мәндерін қалайша алатындығын көрсетіңіз.

2.5.4. ЖОҚ (a ЖӘНЕ b) логикалық өрнектерін логикалық сызбаға келтіріңіз.

2.5.5. ЖОҚ a ЖӘНЕ ЖОҚ b логикалық өрнектерін логикалық сызбаға келтіріңіз.

2.5.6. Бір ЖӘНЕ-ЖОҚ (nand) логикалық операторы барлық үш дәстүрлі логикалық операторларды алмастыра алады. ЖӘНЕ-ЖОҚ шындықтар кестесі мен логикалық бекітпе символы төменде көрсетілген:

a	b	$a \text{ nand } b$
0	0	1
0	1	1
1	0	1
1	1	0



Тек ғана ЖӘНЕ-ЖОҚ бекітпелерін қолдана отырып қалай үш логикалық сызба жасауға болатынын көрсетіңіз, олардың әрқайсысы ЖӘНЕ, НЕМЕСЕ және ЖОҚ бекітпелерінің біреуіне балама болулары тиіс. (Көмек: Сіз тұрақты кірістерді қолдана аласыз, мысалы, әрқашан 0-ге немесе 1-ге тең, немесе бір шығыс бекітпеге бірдей кіріс бекітпелер бере аласыз).

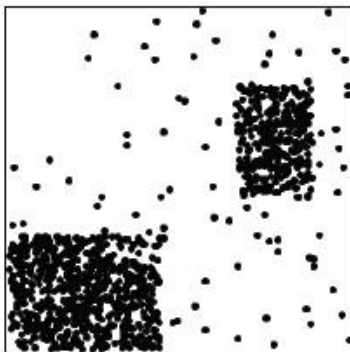
3 БӨЛІМ

Визуалды абстракция

Болжамдап көрейік, бізде қазбалар жұмысын жасауда табылған ежелгі жасандылар орны болып табылатын 1,000 екі өлшемді нүктелі тізім бар, және біз бұл жаңа ашылуларда қандай да бір қағида бар, жоқтығын білгіміз келеді. Төменде көрсетілген өңделмеген мәліметтер тізіміне қарай отырып, сіз қандай да бір тұжырымдама жасай аласыз ба?

Site 1:	(5.346, 3.537)
Site 2:	(5.249, 8.418)
Site 3:	(7.177, 1.937)
:	:
Site 1000:	(1.933, 8.547)

Бәлкім жоқ. Бірақ, картаға қарапайым нүктелер түсіру лезде бәрін түсіндіреді:



Сурет шын мәнінде мың сөздер орнын алады, әсіресе, біз көп көлемді мәліметтермен бетпе-бет келгенде. Осы сияқты мәліметтерді визуалдау үшін, біз өте жиі тасбақаша графикаға жүгінеміз. Тасбақаша графикада сурет салу үшін, біз терезеде

«тасбақа» деп аталатын абстракцияны жасап, оны бағыттаушы командаларға орын ауыстырамыз. Тасбақа қозғалғанда, оның «құйрығы» 3.1- суретінде көрсетілгендей өз артынан іздер қалдырады. Егер тасбақа құрығын жоғары көтерсек, ол іздер қалдырмай қозғала алады. Бұл тарауда, тасбақалық графиканы зерттеуде, сондай-ақ, біз есептер шығару үшін абстракциялар өзара қалай жасалатындығын, қолданатындығын және үйлесетіндігін зерттейміз.

3.1 Абстракт деректер

Тасбақалық графикада тасбақа сипаттамасы – мәліметтер абстракты түрінің (МАТ) мысалы. Мәліметтердің, абстракты түрлері – кейбір заттар категориясын сипаттауды іске асырудан тәуелсіз; ол қандай да бір затты қолдану үшін қажетті ақпаратты береді, бірақ бұл қалай жұмыс жасайтындығын анықтамайды. МАТ екі құрылымнан тұрады:

(а) Атрибуттар (ерекшеліктер) деп аталатын ақпараттар түрлері біз қолдауға тиісті заттар туралы білім, және

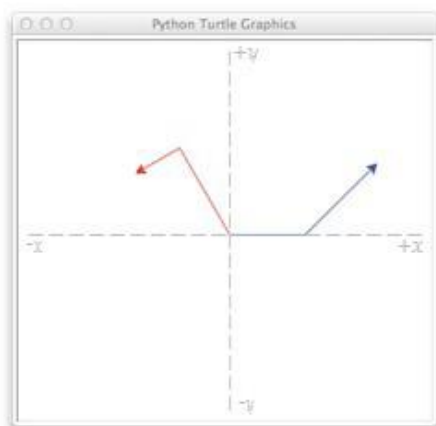
(б) операции, ақпаратты өзгерту немесе өңдеу үшін бізге қолдануға рұқсат етілетін операциялар.

Мысалы, тасбақа мәліметтерінің абстракты түрі тасбақа өз ағымдағы жағдайының келесі атрибуттарынан тұру керектігін анықтайды:

Тасбақа МАТ атрибуттары

Атауы	Сипаты
қалпы	тасбақаның ағымдағы қалпы (x, y)
бағыты	тасбақаның ағымдағы бағыты (градустарда)
түсі	тасбақа ізінің ағымдағы түсі
ені	тасбақаның ағымдағы ені
құйрық қалпы	тасбақа құйрығы түсірілген немесе көтерілген

3.1- сурет. Екі тасбақадан тұратын тасбақалық графика терезесі.
Көк тасбақа



алдыға қарай қозғалып, 45° градусқа солға бұрылып, тағы алдыға қозғала бастады. Қызыл тасбақа 120° градусқа солға бұрылып, алдыға қозғалды, сосын тағыда 90° градусқа солға бұрылып, алдыға қозғала бастады.

Тасбақа МАТ тасбақа жағдай жайлы хабарлап немесе өзгеріп отыратын операцияларды сипаттайды. Егер операция үшін тәуелсіз айнымалы шама ретінде кіріс мәліметтері қажет болса, олар екінші бағанда көрсетілген.

		Тасбақа МАТ операциялары
Атауы	Тәуелсіз айнымалы	Сипаты
create (жасау)	—	(0, 0) нүктесінде орналасқан жаңа сурет салатын, шығысқа бағытталған тасбақа жасау
forward (алдыға)	distance (қашықтық)	Тасбақаны оның ағымдағы бағытында алдыға жылжыту
backward (артқа)	distance (қашықтық)	Тасбақаны оның ағымдағы бағытында артқа жылжыту
right	бұрыш	Тасбақаны сағат бағыты бойынша аудару
left (солға)	бұрыш	Тасбақаны сағат бағытына теріс аудару
setheading	бұрыш	Тасбақа бағытын бекіту
goto	(x, y)	Тасбақаны берілген бағытта жылжыту
up, down	—	Тасбақа құрығын жоғары не төмен жылжыту
pensize	ені	Тасбақа енін орнату
pencolor (түсі)	түсі	Тасбақа түсін бекіту
xcor, ycor (координаттары)	—	Тасбақа x немесе y координаты жайлы ақпаратты қайтару
бағыты	—	Тасбақа бағыты туралы ақпаратты қайтару

Python-да мәліметтердің абстракты түрі топ түрінде берілген. Топта атрибуттар нұсқалар, ауыспалылар жинағынан және операциялардан тұрады, олар тәсілдер деп аталатын арнайы функциялы осы атрибуттарға жүгіне алады. Тасбақа МАТ іске асыратын Python тобы Turtle деп аталады. Turtle тобы жоғарыда берілген бес бөлікті ақпаратты сақтауға арналған нұсқалар ауыспалыдан тұрады (және кейбір басқа қосымшадан). Бірақ біз бұл нұсқалар ауыспалыларын тура нақтылы көре алмаймыз. Оның орнына біз left, right, forward және pencolor сияқты Turtle тәсілдерін шақыра отырып, олардың мәндеріне жанама жүгінеміз және/немесе өзгертеміз. Turtle көтеген тәсілдері Б.2 қосымшасында берілген.

Turtle тобы turtle атаулы модуль ішінде анықталған (бас әріптердің ерекшеліктеріне назар аударыңыз). Сондықтан, Turtle тобына рұқсат алу үшін, бізге алдымен бұл үлгіні импорттау қажет:

```
>>> import turtle
```

Turtle тоб шын мәнінде бар екендігіне көз жеткізу үшін келесіні байқап көріңіз:

```
>>> turtle.Turtle  
<class 'turtle.Turtle'>
```

Үй құрылысын сипаттайтын, бірақ шын мәнінде үй болып табылмайтын сызба сияқты, Turtle тобы немесе аса абстракты Turtle MAT, тасбақалық графиканың құрылымын сипаттайды (яғни, атрибуттары мен тәсілдерін), бірақ шын мәнінде сурет салатын тасбақа болып табылмайды. 3.1- суретінде көрсетілген, тасбақалық графикадағы нақты тасбақалар, нысан-тасбақалар деп аталады. Біз Turtle тобына жататын жаңа тасбақа нысанын жасауда, тасбақа нысаны, топтарды анықтауда жазылғандай, өзіндік жеке тәуелсіз бағыт, қалып, түс және сол сияқты мәндерге ие. Сол себептен 3.1- суретінде көрсетілгендей тасбақа нысаны бірден көп болуы мүмкін.

Топ пен нысан арасындағы ерекшеліктер жануарлар топталуына ұқсас ортақ ерекшеліктермен салыстырып көрсетуге болады. Түрлілік топ ретінде бірдей ортақ ерекшеліктерге ие жануарлар категориясын сипаттайды (морфологиялық және/немесе генетикалық). Нақты тірі ағза нысан топ бір түрі болып табылатындай түр тобын айқындайды. Мысалы, галопагоссты алып тасбақа (*Chelonoidis nigra*) – топ аналогиясы, ал 2012 жылы өлген әйгілі галапагоссты алып тасбақа, жалғыз Джордж бұл топтың ұқсастық нысаны болып табылады. Басқа галапагоссты алып тасбақа Супер Диего дәл сол топқа жатады, бірақ, сол топтың басқа нысаны бола тұрып, өзіне тән жеке ерекше нышанына ие индивидуум болып табылады.

3.1 Ойлануға арналған сұрақтар. *Сіз сәйкес нысанды топ үшін басқа ұқсастықты ұсына аласыз ба?*

Іс жүзінде кез келген пайдалану өнімін, азық-түлік тобына жататын нысан ретінде қарастыруға болады. Мысалы, құлаққап сыңары нақты маркалы және нақты үлгілі құлаққаптар тобына жататын нысан болып табылады. Топ ерекшелігін немесе MAT пайдаланушы нұсқаулығымен салыстыруға болады, себебі пайдаланушы нұсқаулығы өнім қалай жұмыс жасайтындығын немесе ол қалай жасалғандығын нақтыламай оны қалай іске пайдалануға болатындығын айтып береді. Курстық жұмыс тапсырмасы д MAT ұқсас, себебі ол жұмысқа деген талаптарды сипаттайды. Студент тапсырманы орындауда ол осы талаптарды ұстанатын нысанды тапсырады (осылай деп сенеміз).

Ескертке 3.1: Он алты еселік есептеу жүйесі

Он алты еселік – 16 –позициялық есептеу жүйесі.

Ондық жүйеде позициялық мәндер 10 деңгейі және екілік жүйедегі позициялық мәндер 2 деңгейі болып табылатындықтан, он алты еселік жүйедегі позициялық мәндер 16 деңгейі болып табылады.

Ол үшін он алты еселік жүйеге 16 сан қажет болады: атынан а-дан f-ке дейін әріптер жүріп отыратын 0 ден 9 дейінгі сандар. А әріпі 10 деген мәнге ие, b әріпі 11 деген мәнге ие,....., f әріпі 15 деген мәнге ие. Мысалы, 51ed он алты еселік сан келесі ондық мәнге ие

$$5 \cdot 16^3 + 1 \cdot 16^2 + 14 \cdot 16^1 + 13 \cdot 16^0 = 5 \cdot 4096 + 1 \cdot 256 + 14 \cdot 16 + 13 \cdot 1 = 20, 973.$$

Он алты еселік жүйе бинарлы жүйені ыңғайлы қысқарту үшін қолданылады. Кез келген 4 екілік разряд Оден 15 дейінгі мәндермен берілуі мүмкін болғандықтан, оларды бір он алты еселік санмен оңай атмастыруға болады. Осылайша, 100522f10 он алты еселік сан екілік жүйеде 000100000000010100100010111100010000 мәніне төменде көрсетілгендей баламалы:

1 0 0 5 2 2 e 1 0
0001 0000 0000 0101 0010 0010 1111 0001 0000

Бұл 36-битті екі еселік санды көрсетудің орынан он алты есептікке баламалы 9 санды көрсету ыңғайлы.

Python-да тасбақа нысанын жасау үшін біз топ атауы бар функцияны шақырамыз, оған үлгі атауы бірінші болып жазылады және ол жерде топ атауы келесідей.

```
>>> george = turtle.Turtle()
```

Бос жақшалар біз функцияны тәуелсіз айнаымалы шамасыз шақыратынымызды меңзейді. Turtle() функциясы Turtle нысанына сілтемені қайтарады, сосын оған george атауын береді. Айта өту керек, экранда, ортасында «тасбақа» формалы кішкене бағыт сызықшасы бар терезе пайда болады. Терезе орталығы (0, 0) координаттарына ие және координат басы деп аталады. 3.1- суретінде өзектер терезеге ашық сұр түсте берілген, бұл сізді координаттар жүйесінде оңай бағдарлау үшін жасалған. Біз, george – бұл Turtle нысаны екендігіне нысан мәнін басып жасай отырып көз жеткізе аламыз.

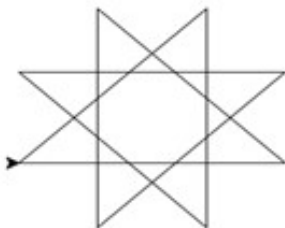
```
>>> george  
<turtle.Turtle object at 0x100522f10>
```

«0x100522f10» түсініксіз түрдегі жазба Turtle нысаны орналасқан жад мекен-жайы болып табылады. Мекен-жай он алты еселік түрде немесе 16-позициялық есептеу жүйесінде көрсетіледі. Басында жазылған 0x он алты еселік жазу формасына меңзейтін префикс болып табылады; он

алты еселік жадтың нақтылы мекен-жайы– 100522f10. Он алты еселік жүйе қалай жұмыс жасайтыны туралы толығырақ ақпарат алу үшін 3.1 ескертпесіне назар аударыңыз.

Нысанға жататын тәсілді шақыру үшін тәсіл атауының алдына нысан атауын нүктемен бөліп жазамыз. Мысалы, George-ке 200 бірлікке алдыға жылжуға бұйрық беру үшін, келесідей жазу керек

```
>>> george.forward(200)
```



3.2- сурет. Тасбақалық графикадағы қарапайым «гүл» геометриялық суреті.

Координат басы (0, 0) болғандықтан, ал george басынан шығысқа бағытталғандықтан (х оң мәніне қарай), ал george позицияға орын ауыстырды (200, 0); осыған сәйкес forward тәсілі George позициясының жасырын атрибутын үнсіз өзгертті, бұған көз жетізу үшін George-қа арналған position тәсілін шақыру қажет.

```
>>> george.position() (200.00,0.00)
```

Біз нысан атрибуттар орналасуын ауыстырмағандығымызға назар аударыңыз. Шын мәнінде біз бұл атрибуттың атауын білмейміз, себебі оның сипаты біз үшін жасырын болып қала береді. Осылай болуы қарастырылған. Нысандармен, олардың атрибуттарымен жұмыс жасамай, тек олардың тәсілдері арқылы байланыса отырып, біз МАТ баптаулары мен негізін қалаушы іске асыру арасындағы нақты бөлулерді қолдаймыз. Бұл негіз қалаушы іске асыруды аса тиімді ету үшін оны өзгертуге болады деп болжамдайды, бұрақ бұл өзгертулер қолданылатын бағдарламаларға ықпал етпеуі тиіс. Біз бұл сұрақтарды 13 тарауда толық қамды талқылаймыз.

Жаттығулар

3.1.1. Мәліметтер абстрактты түрлері (МАТ) мен топтар арасындағы ерекшелікті түсіндіріңіз.

3.1.2. Нысан мен топ арасындағы өзгешеліктерді көрсету үшін тағы бір ұқсастықты көрсетіңіз. Түсіндіріңіз.

3.1.3. Неліктен біз Turtle атрибуттар мәнін тура өзгертудің орнына нысан жағдайын өзгеру үшін тәсілдер пайдаланымыз?

3.2 ТАСБАҚАЛАР КӨМЕГІМЕН ВИЗУАЛДАУ

Енді 3.2- суретте көрсетілгендей гүл шоғырына ұқсас бір нәрсе болжамдап, суретін салайық. Біз дәл қазір төменгі сегменті суреттегідей көлденең түрде салдық. Келесі сегменті саларға дейін, біз George-ті 135 градусқа бұрылуды сұраймыз.

```
>>> george.left(135)
```

Осы тәсілді шақыру көмегімен біз george үшін жасырын атрибут бағытын өзгерттік, мұны heading тәсілін шақыра отырып тексере аламыз.

```
>>> george.heading() 135.0
```

Суретті аяқтау үшін біз жай ғана жоғарыда аталып өтілген forward және left командаларын тағы жеті рет қайталауымыз керек! (Көмек: алдыңғы нұсқамаларды шақыру үшін IDLE анықтамасыз қараңыз).

```
>>> george.forward(200)
```

```
>>> george.left(135)
```

```
>>> george.forward(200)
```

```
>>> george.left(135)
```

```
>>> george.forward(200)
```

```
>>> george.left(135)
```

```
>>> george.forward(200)
```

```
>>> george.left(135)
```

```
>>> george.forward(200)
```

```
>>> george.left(135)
```

```
>>> george.forward(200)
```

```
>>> george.left(135)
```

```
>>> george.forward(200)
```

```
>>> george.left(135)
```

Бұл шаршаттырады. Осындай зеріктірер әрекеттерді болашақта қалай айналып өтуді қарастырағға дейін, біз Python сыртқы қабатын қалдырамыз. Бұл бізге бағдарламаларымызды сақтауға мүмкіндік береді, сонда біз болашақта қателігімізді оңай өзгертіп, не оларды түзей аламыз, ал сосын қажеттіліксіз бәрін қайтадан енгізуден боса-

тыламыз. IDLE-да біз жаңа бос бағдарлама файлын жасай аламыз, ол үшін File¹ мәзірінен New Window таңдау қажет. Жаңа терезеде осыған дейін жасаған жұмысыңызды қайтадан басып жазыңыз (немесе көшіріп, қойыңыз), қызылмен белгіленген төрт қосымша жолдар қосыңыз. (Егер сіз көшіріп қоятын болсаңыз, әр жол алдында шақыру белгілерін жоюды ұмытпаңыз.)

```
import turtle
george = turtle.Turtle()
george.hideturtle()
george.speed(6)
george.forward(200)
george.left(135)
george.forward(200)
george.left(135)
george.forward(200)
george.left(135)
george.forward(200)
george.left(135)
george.forward(200)
george.left(135)
george.forward(200)
george.left(135)
george.forward(200)
george.left(135)
george.forward(200)
george.left(135)

screen = george.getscreen()
screen.exitonclick()
```

Бірінші атау беру нұсқамасыннан кейінгі екі нұсқама george нысанын жасырады және сурет салу барысын аздап жылдамдатады. (speed тәсілінің аргументі 0-ден 10-ға дейінгі санды білдіреді, бұл жерде 1 – баяу, ал 0 ең жылдам темп). Соңғының алдындағы оператор screen ау

¹ Егер сіз басқа мәтіндік редакторды пайдаланатын болсаңыз, қадамдар бәлкім өте ұқсас болар.

ыспалысына Screen топты нысанын үлестіреді, ол George өмір сүретін сурет аумағы болып табылады. Соңғы нұсқама Screen топты exitonclick тәсілін шақырады, егер біз оған бассaq ол терезені жабады.

Басып жазуды аяқтағаннан кейін File мәзірінен Save As. . . таңдап файлды сақтаңыз.

Python бағдарлама файлдарының атауы әрқашан .py кеңейтуімен аяқталуы тиіс, мысалы george.py. Өзіңіздің жаңа бағдарламаңызды IDLE-де орындау үшін Run мәзірінен Run Module таңдаңыз. Егер IDLE Сізге файлді қайтадан сақтауға ұсынса, жай ғана ОК-ді басыңыз. Бағдарлама қайтадан «гүл» мүсінін салуы тиіс. Аяқтай келе терезені жабу үшін тасбақа графигіне басыңыз.

Итерация көмегімен сурет салу

1 тараудан сіздің есіңізде болу керек, біз қайталанатын есептеулермен бетпе-бет келгенде біз оларды кезең деп аталатын аса кешенді формада (мысалы, 365 санына сомма есептеуде) қайта жаздық. Тура осылайша, біз өзіміздің сегіз бірдей жұптарымызды forward және left тәсілдерімен ауыстырғымыз келеді, бұл тәсілдер жұптарды сегіз рет қайталайды. Бұл үшін 16 сурет салу нұсқамаларын келесі for кезеңімен алмастырайық:

```
for segment in range(8): george.forward(200) george.left(135)
```

Кезең – бұл есептеулерді бірнеше рет қайталайтын құрылым. Мысалы кезеңмен берілетін қайталанба есептеулер итерация деп аталады. for кезеңінің бірінші жолы екі нүктемен (:) аяқталуы тиіс, ал кезеңмен қайталанатын нұсқама оның астында бос орын қалдыра отырып орналасуы қажет. Бос орны бар нұсқама жиі кезең денесі деп аталады. Осы for кезең бос орны бар осы екі нұсқаманы сегіз рет қайталайды, бұған дәлел 8 домалақ жақшалар. Кезең аяқталғаннан кейін, Python келесі нұсқаманы бос орынсыз қайталайды (осы нақты жағдайда бос орын жоқ).

3.2 Ойлануға арналған сұрақ *Егер сіз екі нүктені ұмытып кетсеңіз не болады?*

Екі нүктені ұмытып кету оңай. Егер, сіз шын мәнінде екі нүктені ұмытып кетсеңіз, Ссз синтаксистік қажет жайды хабарлама аласыз, ол хабарламада for басты сөзі бар жол аяқталуына меңзейтін мәлімет бар.

```
for segment in range(8)
```

```
^ SyntaxError: invalid syntax
```

For кезеңді синтаксисте: for және in – Python басты көздері, ал segment индексті ауыспалы деп аталады. Ауыспалы индексінің атауы біздің қалауымызша болуы мүмкін, бірақ олардың бағдарламада рөлі мағыналық мәнге ие болуы керек. Дәл осы жағдайда біз segment атауын таңдадық, ол кезең денесі салатын сурет мүсіндегі бір желілік сегментті болжамдайды. range(8) 0 ден 7 дейінгі толық сандар диапазонын білдіреді. Осылайша, for кезеңі шын мәнінде былай дейді: «0-ден 7-ге дейін толық сандардың әрқайсысы үшін кезең денесін бір рет орындау». Диапазонды 8 толық сан болғандықтан кезең денесі 8 рет жасалады.

3.3 Ойлануға арналған сұрақ *segment атауын басқа атауға өзгертін көріңіз. Атау өзгерту бағдарлама жұмыс жасауына әсер етті ме?*

Осы кезеңде не болып жатқанына зейінмен назар аударайық. Әр кезең итерациясы алдында 0-ден 7-ге дейінгі диапазонынан келесі мән segment индексті ауыспалға беріледі. Сосын, кезең денесіндегі нұсқама орындалады. Осылайша, тура мағынада, бұл кезең келесі 24 нұсқамаға баламалы:

```
segment = 0 v1
george.forward(200) итерация 1
george.left(135) иr
segment = 1 v1
george.forward(200) итерация 2
george.left(135) иr
segment = 2 v1
george.forward(200) итерация 3
george.left(135) иr
:
segment = 6 v1
george.forward(200) итерация 7
george.left(135) иr
segment = 7 v1
george.forward(200) итерация 8
george.left(135) иr
```

3.4 Ойлануға арналған сұрақ *range(8) ішіне 8 орнына 1ден 10 дейін түрлі мәндерді қойып көріңіз. Сіз мән және сурет арасындағы байланы-*

сты байқай аласыз ба?

Енді, бізде гүлдің негізгі мүсіні қолда бар кезде кішкене түс қосайық. Тасбақа салатын түсті орнату үшін `pencolor` тәсілін қолданыңыз. `for` кезеңі алдына қойып, бағдарламаны қайтадан іске қосыңыз.

```
george.pencolor('red')
```

Ескертпе 3.2: Түстерді анықтау

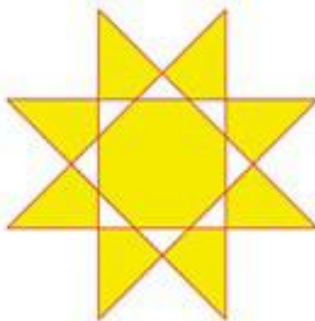
Түсті анықтаудың ең кең таралған тәсілі – оның қызыл, жасыл және көк (RGB) құрамдауыштарын бөлек-бөлек анықтау. Бұл құрамдауыштардың әрқайсысы жиі 0-ден 255 дейін толық санмен сипатталады. (Бұл 8 биттермен берілетін мәндер. Сол кезде біріккен түс 24 битпен анықталады. Егер сіз бұрын «24-битті түс» жайлы естісеңіз, енді сіз оның қайдан пайда болғанын білесіз). Басқа тәсіл де бар, ол тәсіл бойынша әр құрамдауышты 0-ден 10-дейін нақты санмен сипатталады. Python тасбақалық графикасында қажет керіністі таңдау үшін `screen.colormode(255)` немесе `screen.color(1.0)` функцияларын шақыру қажет.

Нақты құрамдауыш үшін аса жоғарғы мән – аса жарық түс. Осылайша, шеткі нүктелер (0, 0, 0) кара түсті, ал (255, 255, 255) және (1.0, 1.0, 1.0) екуіде ақ түсті болып келеді. Басқа жиі кездесетін түстер: (255, 255, 0) – сары түс үшін, (127, 0, 127) – кегілдір түс үшін және (153, 102, 51) – қоныр түс үшін. Енді, `george` – бұл Turtle нысаны деп санасак, онда а `screen` мәні `george` үшін `Screen` нысанына беріледі,

```
screen.colormode(255) george.pencolor((127, 0, 127))
```

осындай жағдайда бұл `George`-ті кегілдір түске бояйды. Немесе былай

```
screen.colormode(1.0) george.pencolor((0.5, 0, 0.5))
```



3.3- сурет. Шеттері қызыл және сары бояулы қарапайым геометриялық «гүл».

Түсті екі тәсілдің бірімен анықтауға болады. Оның біріншісі – кең тараған түс, мысалы қызыл ('red'), көк ('blue') және сары ('yellow'), мәтіндік жолдармен берілуі мүмкін. Біз алдыңғы тарауда көргендей жол-

ды ауыспалыдан немесе функциядан ажырата алу үшін, ол жақшалар ішінде болуы тиіс екендігіне назар аударыңыз. Сонымен қатар, 3.2 ескертпеде сипатталғандай, түс оның қызыл, жасыл және көк (RGB) құрамдауыштарына тура мензеу жолымен анықталуы мүмкін.

Қорытындылай келе, біз «гүл» мүсіні боялатын түсті анықтаймыз. Бояу түсі fillcolor тәсілімен орнатылады. Белгілі бір аумақ суретін бояуға тиесілі нұсқамалар begin_fill және end_fill тәсілдер арасындағы шақыруларда болулары керек. Біздің гүліміз сары болу үшін for кезеңі алдына

```
george.fillcolor('yellow') george.begin_fill()
қою қажет, ал сосын for кезеңінен кейін
george.end_fill()
қою қажет.
```

Міндетті түрде george.end_fill for кезең денесін шақырар алдында бос орын жоқтығын тексеріңіз, себебі біз бұл нұсқама кезең аяқталғаннан кейін, бір рет ғана орындалғанын қалаймыз. Енді сіздің гүліңіз 3.3-суретіндегідей болуы тиіс, ал гүлдің қорытынды бағдарламасы келесідей болуы керек:

```
import turtle
george = turtle.Turtle()
george.hideturtle()
george.speed(6)

george.pencolor('red')
george.fillcolor('yellow')
george.begin_fill()
for segment in
    range(8):
        george.forward(2
        00)
        george.left(135)
george.end_fill()

screen = george.getscreen()
screen.exitonclick()
```

3.5 Ойлануға арналған сұрақ. Сіз мүсін неліктен дәл осылай боялғанын түсінесіз бе?

Жаттығулар

Келесі сұрақтардың әрқайсысына жауап беру үшін қысқаша бағдарлама жазыңыз. Әрқайсысын .py кеңейтуі бар Python бағдарламасының бөлек-бөлек файлына салыңыз (мысалы, picture.py).

3.2.1. Екі тасбақа жасап 3.1- суретін салу үшін оларды қолданыңыз.

3.2.2. Гүл бағдарламасының өзгертілген нұсқасын жазып шығарыңыз, ол бағдарлама 100° бұрышты қолдана отырып, гүлді 18 жақты етіп салуды тиіс.

3.2.3. Ұзын жағы 200 тең шаршы сызу үшін for кезеңін қолданатын бағдарлама жазыңыз.

3.2.4. Ұзындығы 200 және ені 100 болатын тік төртбұрыш суретін салу үшін for кезеңін қолданатын бағдарлама жазыңыз.

3.2.5. Тасбақалық графика көмегімен қызықты сурет салыңыз. ақпарат алу үшін Б.2 және Б.3 қосымшаларына жүгінізіз, мүмкін сіз ең алдымен өз суретіңізді миллиметрлік қағазда салуды қалайтын шығарсыз.

3.3 ФУНКЦИОНАЛДЫ АБСТРАКЦИЯ

Егер біз түрлі түсті және түрлі көлемді гүлдерге толы, бақ салғымыз келсе ше? Біз алдыңғы бөлімдегі кодты пайдалана отырып, гүл шоғырын сала аламыз. Қосымша гүлді салу үшін, біз гүл шоғырымыздың кодын көшіріп, оны бастапқы код астына қойып, сосын көлем мен түстерді өзгерту үшін, кейбір тәсіл аргументтерін өзгерте аламыз. Мысалы, келесі бағдарлама екі гүл салады. Қызылмен белгіленген алғашқы үш нұсқамалар George-ты жаңа орынға ауыстырады, бұл бірінші гүлді жауып қалмау үшін жасалады. Қызылмен белгіленген кодтың басқа бөлігі – үш өзгертулер енгізілген жоғарғы кодтың көшірмесі.

```
import turtle
george = turtle.Turtle()
george.hideturtle()
george.speed(6)
george.pencolor('red')
george.fillcolor('yellow')
```

```

george.begin_fill()
for segment in
    range(8):
        george.forward(200
        ) george.left(135)
george.end_fill()
george.up()
george.goto(200, 200)
george.down()
george.pencolor('yellow')
george.fillcolor('purple')
george.begin_fill()
for segment in
    range(8):
        george.forward(150
        ) george.left(135)
george.end_fill()
screen = george.getscreen()
screen.exitonclick()

```

3.6 Ойлануға арналған сұрақ `george.up()` және `george.down()` нұсқамаларының мақсаттары қандай? Екінші гүл біріншіден несімен ерекшеленеді?

Бірақ, бұндай стратегия – өте нашар идея. Біріншіден, ол өте көп уақыт алады және қатеге ықпалды; сізге көп рет көшіріп қоюда қате жіберу немесе сәйкес өзгертулер енгізуді ұмытып кету өте оңай. Екіншіден, бұл сіздің кодыңызды оқу үшін тым ұзын және қиын етеді. Осындай әрекеттерді бірнеше рет орындау ондаған немесе жүздеген желілі тығыз кодқа тез апарып соқтыруы мүмкін. Үшіншіден, оған өзгертулер енгізу қиын болады. Мысалы, сіз жиырма гүл жасауға жеткілікті кодты көшіріп алдыңыз, ал сосын сіз сегіз жапырақты гүл орнына алты жапырақты гүл жасауды қаладыңыз?

Оның орнына біз жаңа гүл салу үшін жаңа функция жасай аламыз. Сонда біз түрлі гүлдерді салу үшін бір түрлі аргументтері бар функцияны бірнеше рет шақыра аламыз. Python-да функция жасау үшін, біз артынан функция атауы еріп жүретін `def` басты сөзін және қазіргі таңда қолданылатын домалақ жақшаларды (біз оларға жақын арада қайта ораламыз) қолданамыз. `for` кезеңіндегідей, `def`-тен басталатын жол екі

нүктемен (:) аяқталады.

```
def bloom():
```

Функция денесі def жолына қарағанда алшақтайды. Біздің жаңа функциямыздың денесі гүл шоғырының қолынан тұратын болады. Осы жаңа функцияны import нұсқамасынан кейін қойыңыз:

```
import turtle
```

```
def bloom():
```

```
    george.pencolor('red')
    george.fillcolor('yellow')
    george.begin_fill()
    for segment in
        range(8):
            george.forward(2
                00)
            george.left(135)
    george.end_fill()
```

```
george = turtle.Turtle()
```

```
george.hideturtle()
```

```
george.speed(6)
```

```
bloom()
```

```
screen = george.getscreen()
```

```
screen.exitonclick()
```

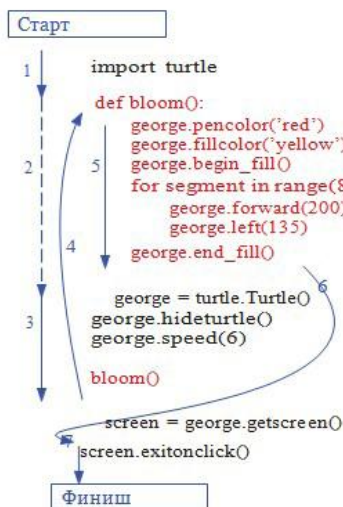
def құрылымы тек жаңа функцияны ғана анықтайды; ол оны орындамайды. Оны орындау үшін біз функцияны шақыруымыз керек. Біз жоғарыда көргендей, функция шақыру аргументтер тізімі артынан еріп жүретін функция атауынан тұрады. Бұл функцияда аргументтер жоқ болғандықтан (қазірге жоқ) және ол мәндерді қайтармайтындықтан, біз

```
bloom()
```

қоя отырып, бұрын гүл шоғыры коды болған ең сыртқы алыстау деңгейіне оны шақыра аламыз (жоғарыда көрсетілгендей).

3.7 Ойлануға арналған сұрақ. Бағдарламаны *bloom()* функциясын және олсыз шақырып, не боларын байқап көріңіз.

Әрі қарай жалғастыра алдында, уақытты пайдаланып бағдарлама не істейтінін талдап көрейік. Төменде көрсетілгендей, жұмыс орындау «Старт» жазуынан жоғары жақтан басталады, бағдарлама ең алдымен turtle үлгісін импорттайды.



Әрі қарай, Далее, bloom функциясы анықталған, бірақ орындалмаған, бұл 2 нөмірлі пунктирлі сызықпен белгіленген. Келесі үш нұсқама george деп аталатын жаңа Turtle нысанын анықтайды және тасбақаны жасырып, оны бір шама тездетеді. Әрі қарай, bloom () шақыру 4 нөмірлі бағыттау сызықшасымен белгіленген функция басына ауысуды шарттылайды. Со-сын, функциядағы нұсқамалар гүлдің суретін салады (5 қадам). Функция аяқталған кезде, 6 нөмірлі бағыттау сызықшасымен белгіленген функция шақырудан кейін, нұсқама орындау бағдарлама аяқталғанға дейін жалғаса береді.

Функциялар баптаулары

Біз жазған bloom функциясы аса тиімді емес, себебі ол әрқашан бір түрлі гүл суретін сала береді: 200 сегмент ұзындықты сары гүл. Функцияна аса ортақтастыру үшін, біз аргументтер мәні ретінде мысалы, бояу түсі немесе сегмент ұзындығына ие болуды қалаймыз. Біз мұны, функция анықтау үшін бір немесе бірнеше баптаулар қосу арқылы іске асырамыз. Баптау – бұл функция шақыруда аргументке мән берілетін ауыспалы болып табылады. (Баптаулар мен аргументтер, сондай-ақ, сәйкес формал-

ды баптаулар мен нақтылы баптаулар деп аталады). Төменде көрсетілген жаңа нұсқада, біз сегмент ұзындығы мен түс бояуын ұсыну үшін екі формалды баптауды (қызылмен белгіленген) анықтаймыз. Сондай-ақ, біз 'yellow' және 200 ескі константтарын осы жаңа баптаулар атауларына алмастырдық.

```
def bloom(fcolor, length):  
    george.pencolor('red')  
    george.fillcolor(fcolor)  
    george.begin_fill()  
    for segment in range(8):  
        george.forward(length)  
        george.left(135)  
    george.end_fill()
```

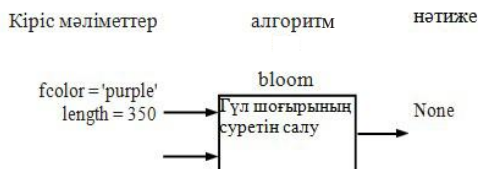
Енді, алдыңғы әрекетті қайталау үшін,
`bloom('yellow', 200)`

баптауы бар функцияны шақырайық.

Осы функцияны шақыруда 'yellow' аргументінің бірінші мәні fcolor бірінші баптауына беріледі, ал 200 аргументінің екінші мәні ength екінші баптауына бекітіледі. Функция денесі іске асырылады. Fcolor әр кез аталған сайын ол 'yellow' ауыстырылады, және length әр кез аталған сайын ол 200 ауыстырылады.

3.8 Ойлануға арналған сұрақ. *Осы өзгертулерді іске асырғаннан кейін, бағдарламаны қайтадан іске қосыңыз. Сосын, бағдарламаны bloom функциясын шақырумен берілген түрлі аргументтер көмегімен бірнеше рет іске қосып көріңіз. Мысалы, ('orange', 50) және bloom('purple', 350) байқап көріңіз. Осы функциялар шақырудағы аргументтер тәртібін өзгертсе не болады?*

Біз қазір ғана функционалды абстракция жасағанымызға назар аударыңыз! Біз гүл шоғырының суретін салғымыз келгенде, біз функция үшін кіріс мәліметтер ретінде екі аргумент береміз, ал функция бізден сурет қалай салынатын туралы түсінікті талап етпей-ақ суретті сала береді.



(Әрине, біз функцияны өзіміз жазғандықтан, ол сурет қалай салынғандығын түсінеміз. Бірақ функция шақыру үшін, бізге ол қалай жұмыс жасайтындығын түсіну қажет емес).

Біз, бұл функцияда тағы бір өзгерту жасамақпыз, әрі қарай жылжу алдында келесі сұрақтарды басшылыққа аламыз.

3.8 Ойлануға арналған сұрақ. *bloom* функциясы ішінде қолданылатын *george* атаулы ауыспалыға назар аударыңыз. Ол қай жерде анықталған?

Bloom функциясы іске асып жатқан кезде, Python түсіндіргіші *george* ауыспалы атауын бірінші жолда кездестіреді, бірақ *George* бұл функцияда анықталмаған. Осыны түсіне отырып, Python *george* атауын функция ішінде іздей бастайды. Мұдай тәртіп бақылау мәліметтер ережелері деп аталады.

Ауыспалы атауын *бақылау* бағдарламаның бір бөлігі болып табылады, бұл жерде атау анықталған, яғни, атауды қолдануға болады.

Bloom функциясында *segment* сияқты функция ішінде анықталған ауыспалу атауын *бақылау* осы функциямен шектеледі. Осындай ауыспалы *жергілікті ауыспалы* деп аталады. Егер біз *bloom* функциясы шеңберінен тыс *segment*-ке жүгінсек, біз қате жайлы хабарлама аламыз. Жергілікті ауыспалыларды біз 3.6 бөлімінде түбегейлі талдаймыз.

Ең сыртқы бос орын деңгейінде анықталған ауыспалы атауы, бағдарламаның кез келген орнынан қол жетімді бола алады, және ол *жаһандық ауыспалы* деп аталады. Біздің бағдарламамызда *george* және *screen* бұл жаһандық ауыспалылар атаулары. Әдеттегідей, бағдарламада қандайда бір жаһандық ауыспалыларға ие болу – жалпылап алғанда нашар идея, бұл тақырыпты біз 3.4 және 3.5 бөлімдерінде әрі қарай талқылаймыз. Бірақ, тіпті бұл сұрақтан алшақтай отырып, біздің функция шеңберінен тыс функциямен анықталатын *george* атаулы нақты бір тасбақаға байлаулы екендігі бізді алаңдатуы керек. Тасбақаны функция баптауы ету аса тиімді болар еді, сонда біз оны қалаған тасбақамен

шақыра алар едік. George-ті tortoise атаулы баптауға алмастыру бізге келесі өзгертілген функцияны береді:

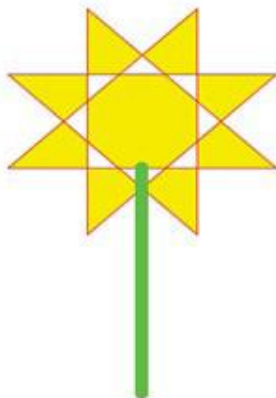
```
def bloom(tortoise, fcolor, length):  
    tortoise.pencolor('red')  
    tortoise.fillcolor(fcolor)  
    tortoise.begin_fill()  
    for segment in range(8):  
        tortoise.forward(length)  
        tortoise.left(135)  
    tortoise.end_fill()
```

Сондай-ақ, бізге tortoise бірінші баптауына бекіту керекті george-ті бірінші аргумент ретінде беріп функция шақыруды жаңарту қажет.

```
bloom(george, 'yellow', 200)
```

Біздің гүлімізді аяқтау үшін сабақша суретін салатын тағы да бір функцияны жасайық. Біздің сабақша суретін салу функциямыз екі баптауды қабылайды: тасбақа нысанының атауы болып табылатын tortoise, және сабақша ұзындығы length. Өз бағдарламаңызға осы функцияны bloom функциясы анықталған жерге лезде қосыңыз.

```
def stem(tortoise, length):  
    tortoise.pencolor('green')  
    tortoise.pensize(length / 20)  
    tortoise.up()  
    tortoise.forward(length / 2)  
    tortoise.down()  
    tortoise.right(90)  
    tortoise.forward(length)
```



3.4- сурет. Сабақшасы бар қарапайым геометриялық гүл

Bloom функциясы шығысқа бағытталып тұрған тасбақаны координаттар басына қайтаратындықтан, tortoise функциясы stem шақырылатын кездегі қалыпта деп санайық. Pensee тәсілін шақыру арқылы тасбақа құйрығын қалыңдаудан және түс түрін жасылға орнату функциясынан бастайық. Түс түрінің көлемі length баптауына тәуелді екендігіне назар аударыңыз, сондықтан ол гүлдердің түрлі көлеміне қарай сәйкес өзгертіледі. Әрі қарай, сабақша суретін салуды бастау үшін, бізге гүл жолының жартысына дейін жылжып, сосын оңтүстікке бұрылу қажет. Алдында салынған гүл суретінің үстіне тағы да бір гүл суретін салып қоюды болдырмау үшін, біз жұмысты аяқтарға дейін алдыға жылжуды тасбақа құйрығын up тәсілімен көтеріп, оны бастапқы орнына down функциясы көмегімен қалпына келтіреміз. Қорытындылай келе, қалың жасыл сабақша суретін салу үшін оңға бұрылып, тасбақаны алдыға жылжытамыз. Біздің сары гүліміз үшін сабақша суретін салу үшін, біз bloom функциясын шақырғаннан кейін

stem(george, 200)

функциясын шақыруды қоя аламыз. Бағдарлама іске қосылғаннан кейін, гүл 3.4- суретіндегідей болуы тиіс.

Енді бізде екі функция бар – нақтылай келе, екі функцияналды абстракция бар, олар гүл шоғырымен және онымен байланысты сабақша

салатын функциялар. Енді біз, гүл суретін салу функциясы әрі қарай қалай жұмыс істейтіндігін ойламай-ақ, виртуалды сабақ жасау аса кең мәселесіне назар аудара аламыз.

3.10 Ойлануға арналған сұрақ. *Сіз тағы да бір функционалды абстракция үшін бағдарламада басқа мүмкіншіліктер көріп тұрсыз ба?*

-Келесі екі функциялар нені бірігіп орындайды?

```
bloom(george, 'yellow', 200)  
stem(george, 200)
```

Дұрыс, ол гүл суретін салады! Осылайша, біз бұл екі басқа функция жолдарын гүл суретін салатын басқа екі жолдарға алмастыра аламыз (басқа функционалды абстракцияға):

```

import turtle

def bloom(tortoise, fcolor, length):
    tortoise.pencolor('red')
    tortoise.fillcolor(fcolor)
    tortoise.begin_fill()
    for segment in range(8):
        tortoise.forward(length)
        tortoise.left(135)
    tortoise.end_fill()

def stem(tortoise, length):
    tortoise.pencolor('green')
    tortoise.pensize(length / 20)
    tortoise.up()
    tortoise.forward(length / 2)
    tortoise.down()
    tortoise.right(90)
    tortoise.forward(length)

def flower(tortoise, fcolor, length):
    bloom(tortoise, fcolor, length)
    stem(tortoise, length)

def growFlower(x, y):
    span = random.randrange(20, 200)
    fill = random.choice(['yellow',
                          'pink', 'red', 'purple'])
    tortoise = turtle.Turtle()
    tortoise.hideturtle()
    tortoise.speed(6)
    tortoise.up()
    tortoise.goto(x, y)
    tortoise.setheading(0)
    tortoise.pensize(1)
    tortoise.down()
    flower(tortoise, fill, span)

george = turtle.Turtle()
george.hideturtle()
george.speed(6)
flower(george, 'yellow', 200)
screen = george.getscreen()
screen.exitonclick()

```

```

import turtle

def bloom(tortoise, fcolor, length):
    tortoise.pencolor('red')
    tortoise.fillcolor(fcolor)
    tortoise.begin_fill()
    for segment in range(8):
        tortoise.forward(length)
        tortoise.left(135)
    tortoise.end_fill()

def stem(tortoise, length):
    tortoise.pencolor('green')
    tortoise.pensize(length / 20)
    tortoise.up()
    tortoise.forward(length / 2)
    tortoise.down()
    tortoise.right(90)
    tortoise.forward(length)

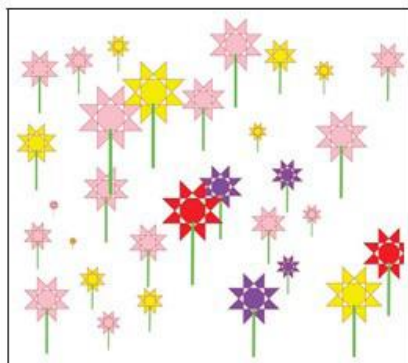
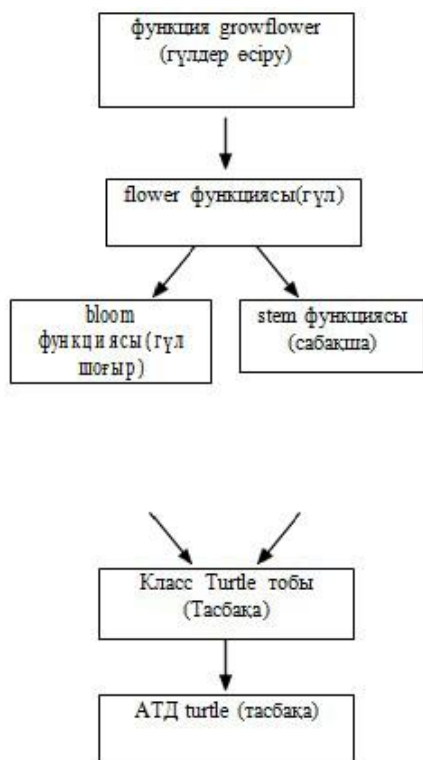
def flower(tortoise, fcolor, length):
    bloom(tortoise, fcolor, length)
    stem(tortoise, length)

def growFlower(x, y):
    span = random.randrange(20, 200)
    fill = random.choice(['yellow',
                          'pink', 'red', 'purple'])
    tortoise = turtle.Turtle()
    tortoise.hideturtle()
    tortoise.speed(6)
    tortoise.up()
    tortoise.goto(x, y)
    tortoise.setheading(0)
    tortoise.pensize(1)
    tortoise.down()
    flower(tortoise, fill, span)

george = turtle.Turtle()
george.hideturtle()
screen = george.getscreen()
screen.onclick(growFlower)
screen.mainloop()

```

3.5 сурет Гүл отырғызу кодын қойғанға дейінгі гүл бағдарламасы. бар қорытынды гүл бағдарламасы.



3.7- сурет Гүлдер суретін салу бағдарламасындағы абстракциялар деңгейлері

3.8- сурет Кездейсоқ таңдалған «Бағы»

```

def flower(tortoise, fcolor, length):
    bloom(tortoise, fcolor, length)
    stem(tortoise, length)
  
```

Bloom және stem екі функцияларына тасбақа, бояу түсі мен ұзындық қажет болғандықтан, біз гүліміз бұл үш баптаулары күйге келтіргіміз келеді, себебі олар біздің flower функциясына баптаулар болады. Біз бұл барлық үш баптаулары bloom функциясына тура береміз, сосын, олардың

екеуін stem функциясына береміз. Қорытындылай келе, келесі баптаулы flower функциясын шақырамыз:

```
flower(george, 'yellow', 200)
```

Бұл функция жоғарыда көрсетіліп кеткен 2 нұскамалар жасайтынды іске асырады. Бағдарлама 3.5 суретінде көрсетілген. Әрі қарай жылжу үшін, сіз жоғарыда аталып өтілгеннің бәрін енгізгеніңізге көз жеткізіңіз.

Осы үш бөлімде біз тек тасбақалық графика суретін салуды ғана үйренген жоқпыз, сонымен қатар, ең маңыздысы абстракцияны қалай анықтап, жасауды үйрендік. Turtle тобы мәліметтердің абстракциялық түрін іске асырады. Осы топты қолдана отырып, біз Python bloom және stem функциялы екі абстракциялық функцияларды іске асырдық. Біз оларды, өз алдына, flower тағы бір функционалды абстракциясын құрастыру үшін қолдандық. Функционалды абстракциялар иерархиясын жасау бізге аса көп көлемді мәселелерді шешуге мүмкіндік береді. Әрі қарай құрастырылатын бұл тағы да бір қосымша деңгейлі абстракция иерархиясы 3.7 суретінде берілген.

Бақ отырғызайық

Енді біз виртуалды гүл бағын отырғызуға дайынбыз. Жай ғана қызығушылық үшін кездейсоқ көлемді және түсті гүлді (flower функциясын қолдана отырып) сурет терезесінің тышқан басқан, кез келген жеріне сала береміз. Осыны іске асыру үшін код біз әлі қолданбаған Python кейбір функциялары мен тәсілдерін пайдаланады. Анықтама үшін тасбақалық графиканың барлық тәсілдері Б.2 және Б.3 қосымшаларында келтірілген.

Осындай кездейсоқ тәріпті іске асыру үшін random үлгісін қолданамыз, бұл үлгі жайлы әңгіме толығымен 6 тарауда талқыланады. Дәл қазір бізге тек екі функция қажет. Біріншісі, random.randrange функциясы кездейсоқ таңдалған толық санды оның екі аргументі арасындағы аралықтан қайтарады. Гүл көлемі үшін 20-дан 199 дейін кездейсоқ санды таңдау үшін біз келесіні қолданамыз

```
span = random.randrange(20, 200)
```

Екіншісі, random.choice кездейсоқ түрде тізімнен таңдалған элементті қайтарады. Біз кейінірек мәлім болатындай, Python тізімі төртбұрыш жақшалармен ([]) қоршалады. Гүл үшін кездейсоқ түсті таңдау

үшін түстер атауы бар жолдар тізімінен `random.choice` функциясын шақырайық:

```
fill = random.choice(['yellow', 'pink', 'red', 'purple'])
```

Осы функцияны шақырудан кейін тізімдегі жолдардың біріне шп орнатылады.

Келесі функция координатты (x, y) нүктеде кездейсоқ гүл суретін салу үшін осы екі функцияның шақыруларын біріктіреді. Оны `flower` функциясы анықталғаннан кейін өзіңіздің бағдарламаңызға қойыңыз, ал `import random` функциясын ең жоғарғы жағына қойыңыз. (3.6 суретіне назар аударыңыз). Әрі қарай жалғастырап алдында бұл функция жұмыс істейтіндігіне сенімді болыңыз.

```
def growFlower(x, y):
    span = random.randrange(20, 200)
    fill = random.choice(['yellow', 'pink', 'red', 'purple'])
    tortoise = turtle.Turtle()
    tortoise.hideturtle()
    tortoise.speed(6)
    tortoise.up()
    tortoise.goto(x, y)
    tortoise.setheading(0)
    tortoise.pensize(1)
    tortoise.down()
    flower(tortoise, fill, span)
```

Әр шақыруда (бір гүлге бір рет) бұл функция `Turtle` жаңа нысанын құрастыратындығына назар аударыңыз.

Әрі қарай, біздің бағдарлама, тышқан координаттарына тең аргументтері бар `growFlower` функциясын шақырғанда тышқан шертулеріне әрекет етуін қалаймыз. Бұл `Screen` топты `onclick` тәсілі көмегімен жүзеге асырылады. Шын мәнінде, келесі нұсқама Python түсіндіргішіне, аргумент ретінде берілетін функция (нақты осы жағдайда `growFlower` функциясы) әр кез тышқанмен терезені шертуде шақырылуы керек дейді.

```
screen.onclick(growFlower)
```

`OnClick` функциясы тышқан шерту (x, y) координаттарын `growFlower` функция баптаулары ретінде береді. (Біз `onclick` функциясына беретін

функция тек осы екі баптауларды қабылдай алады – сондықтан біз жоғарыда Turtle нысанын growFlower функциясына баптау ретінде қоспадық). Жоғарыда берілген нұсқаманы george.getscreen () шақырғаннан кейін қойыңыз. Қорытындылай келе, screen.exitonclick() бағдарламаның соңғы жолында

```
screen.mainloop()
```

ауыстыру қажет. Mainloop тәсілі тышқан шертулерінің болуын мерзімді тексеріп отырады және егер тышқан шертулері орын алған жағдайда сәйкес функцияны шақырады (ол бізден берілген growFlower функциясы болуы тиіс). Бұл қорытынды бағдарлама 3.6 суретінде және мүмкін гүл бақшасы 3.8 суретінде көрсетілген.

Жаттығулар

Келесі сұрақтардың әрқайсысына жауап беру үшін қысқаша бағдарлама жазыңыз. Әрқайсысын .py кеңейтуі бар Python бағдарламасының бөлек-бөлек файлына салыңыз (мысалы, picture.py).

3.3.1. Түрлі көлемді гүл жапырақшалар суретін салуға мүмкіндік алу үшін flower функциясын түрлендіріңіз. Түрі өзгертілген функция қосымша баптауларды талап етеді:

```
flower(tortoise, fcolor, length, petals)
```

Бастапқы функцияда тасбақа орта шамамен $8 \cdot 135 = 1080$ градус саяхаттайтындығына назар аударыңыз. Гүл жапырақшалар көлемін функцияға енгізуде градустардың ортақ соммасы әлі де 360 градусқа тең екендігіне көз жеткізіңіз.

3.3.2. Нарцисс гүліне ұқсас екі гүл шоғырын жасайтындай етіп, төменде көрсетілгендей Flower функциясын өзгертіңіз. Түрі өзгертілген функцияға екі бояу баптаулары қажет болады:

```
flower(tortoise, fcolor1, fcolor2, length)
```

Гүл шоғырының кез келген екі қарама-қарсы нүктелер арасындағы қашықтық сегменттің ұзындығынан шамамен 1.08 есе үлкен деген ақпарат сізге көмектесуі керек.



3.3.3. Төменде көрсетілгендей, «CODE» көзін жазатын бағдарлама жасаңыз. «C» және «D» доғалар суретін салу үшін circle тәсілін қолданыңыз. Circle тәсілі екі аргумент алады: айналым радиусы мен градуста ұзындығын. Мысалы, `george.circle (100, 180)` радиусы 100 тең доға суретінің жартысын салады. Егер теріс ұзындықты дұрыс берсек, онда доға суреті кері бақытта салынады. Сонымен қатар, әріптер арасында із қалдырмай тасбақаны орын ауыстыру үшін `up` тәсілін, ал сурет салуды жаңарту үшін `down` тәсілін қолданыңыз.

3.3.4. Әр әріп суретін өзінің жеке функциясымен салу үшін 3.3.3 Жаттығуынан бағдарламаны көшіріп жазыңыз. Со-сын, «DECODE» суретін салу үшін өзіңіздің функцияңызды пайдаланыңыз. (Өзіңіздің «D» және «E» функцияларыңызды екі реттен шықырыңыз).

3.3.5. Width берілген жағымен шаршы суретін салу үшін `tortoise` атаулы тасбақаны қолданатын `drawSquare(tortoise, width)`

функциясын жазыңыз. Бұл функция сіз 3.2.3 жаттығуы үшін жазған кез келген енді шаршы суретін салу үшін кодты жалпыландырады. `For` кезеңін пайдаланыңыз.

3.3.6. Берілген `width` енді және `length` ұзындықты тік төртбұрыш суретін салу үшін `tortoise` атаулы тасбақаны пайдала-

натын

`drawRectangle(tortoise, length, width)`

функциясын жазыңыз. Бұл функция сіз 3.2.4 жаттығуы үшін жазған кез келген көлемді тік төртбұрыш суретін салу үшін кодты жалпыландырады. For кезеңін пайдаланыңыз.

3.3.7. Берілген жақтар сандары мен жақтар ұзындығы бар дұрыс тік төртбұрыш суретін салу үшін `tortoise` атаулы тасбақаны пайдаланатын

`drawPolygon(tortoise, sideLength, numSides)`

функциясын жазыңыз. Бұл функция сіздің 3.3.5 жаттығудағы `drawSquare` функцияңызды жалпыландырады. Бұрылыс бұрышы үшін жаңа ауыспалы жасаңыз және сіздің for кезеңі үшін `numSides` мәнін пайдаланыңыз.

3.3.8. Берілген радиус айналасын аппроксимациялау үшін 3.3.7 жаттығуынан өзіңіздің `drawPolygon` функцияңызды шақыратын

`drawCircle(tortoise, radius)`

функциясын жазыңыз.

3.3.1. Сурет терезесінде көлденең орналасқан, әрқайсысы 50 радиус болатын әрқайсысы он айналым суретін салатын

`horizontalCircles(tortoise)`

функцияны жазыңыз. For кезеңін пайдаланыңыз.

3.3.2. Сурет терезесінің жоғарғы сол жақ бұрышынан төменгі оң жағына диагональ орналасқан, әрқайсысы 50 радиус болатын, әрқайсысы он айналым суретін салатын

`diagonalCircles(tortoise)`

функцияны жазыңыз. For кезеңін пайдаланыңыз.

3.3.3. 8×8 қызыл-қара шахмат тақтасының бір ретінің суретін салатын

`drawRow(tortoise)`

функциясын жазыңыз. For кезеңі мен Сіз 3.3.5 Жаттығуында жазған `drawSquare` функциясын пайдаланыңыз.

3.3.4. `Color1` және `color2` түстер ұяшықтарын алмастырып

8×8 қызыл-қара шахмат тақтасының бір ретінің суретін салатын
`drawRow(tortoise, color1, color2)`

функциясын жазыңыз. `color1` және `color2` екі баптаулары түстер атауы бар жолдармен берілген. Мысалы, `drawRow(george, 'red', 'black')` функциясын шақыру ауыспалы қызыл және қара торлардың бір реттік суретін салуы тиіс.

3.3.5. `For` кезеңі мен Сіз 3.3.12 Жаттығында жазған функцияны пайдаланып 8×8 қызыл-қара шахматтық тор тақтасы суретін салатын

`checkerBoard(tortoise)`

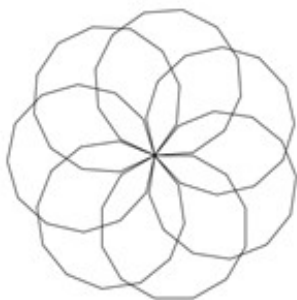
функцияны жазыңыз.

3.3.6. Қызықты, гүлге ұқсас мүсіндердің, әр кез бірнеше градусқа айналып тұратын көп бұрыштарды көп реттік жолмен суретін салу арқылы жасауға болады. Ерекше дизайнды гүл суретін салу үшін 3.3.7 Жаттығуында `drawPolygon` функциясын шақыратын

`polyFlower(tortoise, sideLength, numSides, numPolygons)`

функцияны жазыңыз. Гүлді нақыш жасау үшін функция `drawPolygon`-ды `numPolygons` баптауымен бекітілген рет қайталап, әр кез тасбақаны айналдыра отырып шақыра береді. Сізге салынатын полигондар суретінің негізінде бұрылу бұрышын есептеуге тура келеді.

Мысалы, келесі өрнек суреті `drawFlower (george, 40, 12, 7)` функциясын шақыру арқылы салынды.



3.3.7. Кездейсоқ шарлау бөлшектің , немесе адамның екі еселік кеңістікте кездейсоқ қозғалыстағы кезуіне еліктейді. Әр қадам сайын бөлшек кездейсоқ бағытты таңдап, ағымдағы бағытта шектелген арақашықтықты (мысалы 10 бірлікті) өтеді. Егер бұл қадам көп рет қайталанса, онда ол броундық қозғалысқа немесе азық іздеудегі жауарлар мінез-құлқын еліктей адалы.

Бекітілген қадамдар саны үшін кездейсоқ кезу көрсететін

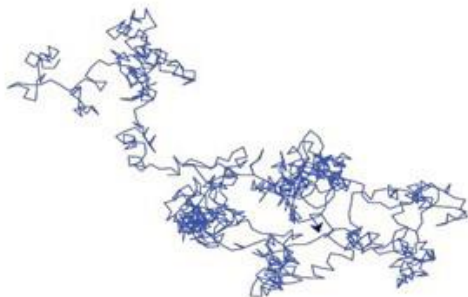
`randomWalk(steps)`

функцияны жазыңыз. Әр қадам сайын кездейсоқ түрде бұрышты таңдау үшін

`angle = random.randrange(360)`

функциясын пайдаланыңыз.

Бұл функцияны пайдалану үшін Сізге `random` үлгісін импорттауға тура келеді. (Кездейсоқ сандар мен кездейсоқ шарлау жайлы толығырақ 5 Тарауда әңгіме қозғаймыз). 1000-қадамдық тентіреудің жеке жағдайы төменде көрсетілген.



Төменде қосымша жаттығулар берілген, олар Сізден тасбақалық графикамен байланысты емес функцияларды жазуды сұрайды. Олардың әрқайсысын үшін сәйкес аргументтер шақыра отырып міндетті түрде тексеріңіз.

3.3.8. Егер екі ұпайлық және үш ұпайлық доп лақтырулар саны `fieldgoal` және `threePointers` баптауларымен берілетін, Сіздің баскетбол командаңыздың шотын экранға шығаратын

`basketball(fieldGoals, threePointers)`

функцияны жазыңыз.

3.3.9. Параметр ретінде адам туған жылын жазуда оның жас мөлшерін экранға шығаратын

`age(birthYear)`

функцияны жазыңыз. Сіз, функция тек осы жылда жұмыс жасайды және осы жылы ол адам әрі туған күнін атаап өткен жоқ деген болжамдама жасай аласыз.

3.3.10. Параметр ретінде команда атауын алып және команда атауы артынан ілесіп отыратын «Go» жазуын басып шығаратын `cheer(teamName)` функцияны жазыңыз. Мысалы, егер функция `cheer('Buzzards')` ретінде шақырылса, онда ол экранға 'Go Buzzards' жазу жолын шығару тиіс.

3.3.11. `Number1` және `number2` сандар соммасын экранға шығаратын

```
sum(number1, number2)
```

функцияны жазыңыз.

3.3.12. Экранға өз параметрін екі рет екі бөлек жолдармен шығаратын

```
printTwice(word)
```

функцияны жазыңыз.

3.3.13. Өз атыңызды экранға 100 рет шығару үшін `for` кезеңін пайдаланатын

```
printMyName()
```

функцияны жазыңыз.

3.4 КӨРКЕМ БАҒДАРЛАМАЛАУ ҚАБІЛЕТІ

Бағдарламалау стилі мен хат жазу стилі көп жағдайда ұқсас қиындықтарға ие. Біз эссе жазуда оқырман біздің тезистеріміз болып оларды дәлелдейтін аргументтерді нақты түсінгенін қалаймыз. Біз, ол нақты және қысқаша, басынан аяғына дейін логикалық реттілік ұстанымына сәйкес болғанды қалаймыз. Тура осылайша, бағдарлама жазуда біз әріптестерімізге бағдарламамыздың мақсатын түсінуге көмектестіміз келеді, осындай жолмен олар бағдарламаны басынан аяғына дейін орындауда осы мақсатқа жетеді. Егер Сіз өзіңіздің кодыңызды оқыған жалғыз адам болсаңыз, жақсы стиль тек есепті құрастыру барысында ғана емес, сонымен қатар, болашақта Сіз өз жұмысыңызды қайтадан зеттеуге кіріскенде көмектеседі. Біз өзіміздің бағдарламаламызда ауыспалылар мен функциялар сипаттаушылық атауларын қолдана отырып, әдейлі емес обфунцияларды (кодтар шатасу) болдырмай кодтар жасай отырып логикалық және абайлы ұйымдастыру арқылы, сондай-ақ, бағдарлама шеңберінде өзіміздің ниеттерімізді құжаттамалай отырып бұл мақсаттарға жете аламыз.

Бағдарлама құрылымы

Біз алдыңғы бөлімде жазған бағдарламаға (3.5 Суреті) қайта оралайық және бағдарламалаудың жоғары стандарттарына сәйкес келу үшін оны аздап түрлендірейік. 3.9 Суретте көрсетілгендей, әр бағдарлама, авторы мен оның мақсаты бекітетін құжаттамадан жасалуы тиіс. Құжаттаманың мұндай түрі құжаттамалық жол деп аталады; келесі бөлімде біз құжаттама жолдары мен құжаттамалардың басқа түрлерін қарастырамыз.

Өрі қарай біз import нұсқамаларымызды қоямыз. Олардың біздің бағдарламаның жоғарғы жағына қою екі мәселені шешеді: біздің бағдарламаны аса ретті етеді, сондай-ақ, импортталған модульдер кез келген орында қол жетімді болатындығына кепіл береді.

Сосын біздің барлық функцияларымызды анықтаймыз. Бағдарлама түсіндіргішпен жоғарыдан төмен қарай оқылатындықтан, біз өз функцияларымызды әрқашан оларды шақыру орынынан жоғары анықтауымыз керек. Мысалы, біздің гүл бағдарламамызда flower функциясы шақырылатын оның белгілі бір функцияларын жол үстіне қоюымыз қажет болған. Егер біз flower функциясы анықталарға дейін оны бағдарламаның жоғарғы жағына шақыруға тырыссақ, онда қате жайлы хабарлама алатын едік.

Бағдарлама
құжаттамасының
жолы

```
"""  
Мақсат: Гүл суретін салу  
Гүл авторы: Мән Студент  
Мерзімі: 15 қыркүйек, 2020  
Информатика 111, 2020 күз  
"""
```

импорт
нұсқамалары

```
import turtle
```

Функциялар
анықтау

```
def bloom(turtle, fcolor, length):  
    """Геометриялық гүл шоғыры суретін салады.  
    Баптаулар:  
    turtle: гүл шоғыры салынатын Turtle нысаны,  
    fcolor: гүл шоғырын бояу үшін қолданылатын гүл атау жолы. length:  
    гүл шоғыры әр сегменттің ұзындығы.  
  
    Мән қайтару:  
    None  
    """  
  
    turtle.pencolor("red")           #тасбақа түс түрі қызылға және бояу түсі  
    turtle.fillcolor(fcolor)         #fcolor анықталады  
    turtle.begin_fill()              # 8 жақты бояу түсімен сурет  
    for segment in range(8):         салады  
        turtle.forward(length)       #turtle.left(135)           гүлдің геометриялық шоғыры  
        turtle.left(135)  
  
    # басқа функциялар қосылмаған..
```


негізгі функция

```
def main():
    """Сегмент ұзындығы 200 болатын сары гүл суретін салады және шығу
    үшін тышқан шертпесін күтеді.
    """
    george = turtle.Turtle()
    george.hideturtle()
    george.speed(6)
    flower(george, 'yellow',
    200)
    screen =
    george.getscreen()
    screen.exitonclick()
```

негізгі функцияны
шақыру

```
main()
```

3.9 суреті Бағдарлама құрылымы туралы ортақ түсінік.

Бағдарлама соңында 3.5 суретіндегі гүл суретінде ең сыртқы бос орын деңгейінде алты нұсқама беріледі. Ескертеміз, бірінші және нұсқамалар бағдарламаның кез келген жерінде көрінетін және өзгертуге болатын жаһандық ауыспалының атауын анықтайды. Жаһандық ауыспалыға мән беру функция ішінде өзгертілген жағдайда, бұл қосалқы әсер деп аталады. Жаһанды ауыспалылар сансыз көр түрлі жерлерде өзгертуге ұшырайтын көптеген бағдарламаларда оларды пайдалану қателіктерін анықтау іс жүзінде анықтау мүмкін емес.

Сондықтан, біз оралды ешқашан қолданбауды әдетке алуымыз тиіс, әрине егер бізде оларды қолдану үшін өте орынды себеп болған жағдайда және оларсыз әрі қарай жұмысты жалғастыру мүмкін болмайтын жағдайда. Python бағдарламасында жаһанды атаулар қалай өңделетіндігі жайлы қосымша ақпарат алу үшін 3.3 Ескертпесіне назар аударыңыз.

Жаһанды ауыспалыларды іске пайдалануды болдырмау үшін және бағдарламаны оқу үшін аса тиімді ету үшін, біз бағдарламамыздағы жаһанды деңгейдегі нұсқамаларды main атаулы функцияға орын ауыстырамыз, сосын, 3.9 Суретінде көрсетілгендей бағдарлама аяқталуында, бағдарламадағы соңғы нұсқама ретінде main атауын шақырамыз. Осындай өзгертулер барысында, main функциясын шақырумен осы бағдарламада әрекеттер жұмыс жасай бастайды.

Python түсіндіргіші функция үшіндегі жаһанды атауларды мән атаулары оқытуына немесе атауға қандай мән берілетіндігіне қарай өңдейді. Python түсіндіргіші есептеуге тура келетін атаумен бетпе-бет келгенде, ол ең алдымен атау функция шеңберінде анықталған ба жоқ па соны табуға тырысады (мысалы, оператордың оң жағынан беріледі). Егер солай болса, онда атау жергілікті салада пайдаланылады. Оған кері жағдайда, түсіндіргіш, атау табылмағанға дейін, сыртқы салаларды ретті түрде қарастырады. Егер ол жаһанды деңгейге жетсе, ал атау әлі табылмаған болса, онда біз «атау қатесін» аламыз.

Басқа жағынан, егер біз атауға мән берсек, жаһанды нұсқаманы пайдалана отырып басқасын белгілемесек, ондай бұл атау жергілікті деп саналады. Мысалы, келесі бағдарламаны қарастырайық:

```
spam = 13
def func1():
    spam = 100
def func2():
    global spam
    spam = 200
func1()
print(spam)
func2()
print(spam)
```

Бірінші print экранға 13 санын шығарды, себебі func1 іске асырылатын атау бері нұсқамасы жаңа жаһандық ауыспалыны анықтайды; ол тура осындай атаулы жаһанды ауыспалыны өзгертпейді. Ал екінші print 200 санын экранға шығарады, себебі, func2 жаһанды нұсқама spam –ды тура осындай атаулы жаһанды ауыспалыға жатқызу керектігін көздейді, осының нәтижесінде әрі қарай атау беру нұсқамасы жаһанды ауыспалыға берілген атауды өзгертпеді. Мұндай ереже кездейсоқ жағымсыз әсерлерге жол бермейді, себебі, жаһанды ауыспалыны өзгерту қажет па жоқ па жайлы шешім қабылдауға мәжбүрлейді. Қалай болғанда да, жаһанды ауыспалыларды іске пайдалану ұсынылмайды.

(Жоғарыда аталып өтілген функциялар анықтаулар функцияларды тек ғана анықтайды, бірақ оларды атқармайды). Қорытындылай келе main функциясы біздің flower функциямызды шақырады, ал бұл функция сосын bloom және stem функцияларын шақырады. Бағдарламалаудың осындай стиліне әдеттену қосымша артықшылықтарды береді: ол басқа кең таралған бағдарламалау тілдерінің стиліне өте ұқсас (мысалы, C, C++, Java), сондықтан, егер Сіз болашақта олардың біреуін қолданатын болсаңыз, олар Сізге салыстырмалы түрде таныс болып көрінеді.

Құжаттау

Python бағдарламаларын құжаттау екі тәсілмен іске асырылады: *құжаттау жолдары* мен *пікірлер*. Құжаттау жолы оны қалыптастыру үшін қажет, ал ол үшін бағдарламаны, үлгілеуді және функция шақыруды қалай пайдалану туралы білу қажет.

Басқа жағынан, пікірлер бөлек бағдарламалық нұсқамаларды немесе нұсқамалар блогін құжаттау үшін іске пайдаланылады. Басқалай айтқанда, құжаттау жолы бағдарлама немесе функция не істейтіндігін, ал дәл осы уақытта пікірлер код қалай жұмыс істейтіндігін түсіндіреді; құжаттау жолы абстракцияны сипаттайды, ал пікірлер қара жәшік ішінде не болы жатқанын сипаттайды. Python түсіндіргіші бағдарлама орындау барысында құжаттау жолын да пікірлерді де елемейді; олар тек адам қарау үшін арналған.

Жұжаттау жолы

Құжаттау жолы үш екі еселі жақшалар жұбының («») ішінде орналасқан және бірнеше жолдарды алады. 3.9² суретінің жоғарғы жағында көрсетілгендей біз әр бағдарлама басында құжаттау жолын бағдарлама авторы мен оның мақсаттылығын анықтау үшін пайдаланамыз. Сонымен қатар, біз құжаттау жолын оқырман қандай әрекет жасап отырғанын нақты түсіну үшін әр жазып отырған функцияны құжаттау үшін қолданамыз. Функцияны құжаттау жолы функцияны шақыру үшін барлық қажет әрекеттерді білуі тиіс: функцияның жалпы мақсаты, баптаулар функциясы мен қайтарылатын мәндер сипаттамасын.

Функция құжаттау жолының басы `def` нұсқамасынан кейін келесі жолға бос орын арқылы қойылады. Бағдарламалаушы құжаттау жолы үшін түрлі стильдері таңдайды; біз Python стилі бойынша Google ресми нұсқаулығында бар стильге өте жақын стилін пайдаланамыз. Төменде осы құжаттама жолының 3.5 суретіндегі үш функцияға қолданылатын стилі көрсетілген (Денелер төмен қарап тұр).

```
def bloom(tortoise, fcolor, length):
    """Гүлдің геометриялық шоғыр
    суретін салады.
    Баптаулар:
        tortoise: fcolor гүл шоғыры суреті салынатын Turtle
        нысаны: length гүл шоғырын бояуға үшін
        қолданылатын гүл атауы бар жол: әр гүл
        шоғырының ұзындығы
    Мән қайтару: None
    """

def stem(tortoise, length):
    """Гүл сабақшасының
    суретін салады.
    Баптаулар:
        tortoise: length гүл шоғырының бастапқы қалпында
        орналасқан Turtle нысаны: сабақша мен гүл шоғырының
        әр сегментінің ұзындығы.
    Мән қайтару:
    None """
```

² Сіздің мұғаліміңіздің стильге деген талаптары басқа болуы мүмкін, сондықтан стиль жайлы сұрастыруды ұмытпаңыз.

```
def flower(tortoise, fcolor, length):
```

```
    """Гүл суретін салады.
```

```
    Баптаулар:
```

```
        tortoise: fcolor гүл шоғыры суреті салынатын
        Turtle нысаны: length гүл шоғырын бояу үшін
        қолданылатын гүл атауы бар жол: әр гүл
        шоғырының ұзындығы
```

```
    Мән қайтару:
```

```
    None """
```

Құжаттаманың бірінші жолына біз бұл функция не істейтіндігін қысқаша түсіндіріп өтеміз. Әрі қарай әр баптауды мақсаты мен тобын атап өтетін баптаулар бөлімі болады. Егер параметр мәніне шектеуліктер бар болса, онда олар да көрсетілуі керек. Мысалы, `stem` функциясындағы `turtle` баптауы гүл шоғырығын басында орнатылады деп болжамдайды. Қорытындылай келе, біз функциялар мәндерін сипаттаймыз. Функциялардың ешқайсысы ештеңе қайтармады, сондықтан, қайтаралытын мән `None` болады. 3.5 Бөлімінде қайтарылатын мәндерді қайтаратын функция қалай жазылатындығын қарастырамыз.

Құжаттау жолдарын жазудың тағы да бір артықшылығы Python оларды `help` (анықтамалар) функцияларын шақыруға жауап ретінде құжаттамаларды автоматы түрде жасай алатындықта. Мысалы, Python сыртқы қабатындағы мына қысқаша мысалды байқап көріңіз:

```
>>> def spam(x):
```

```
    """x-ті қарапайым қайтарады.
```

```
    Баптаулар:
```

```
        x: кейбіреулері, қандай мән екендігі маңызды емес
```

```
    Қайтарылатын мән:
```

```
        x: кейбіреулері, қандай мән екендігі маңызды емес
```

```
    """
```

```
    return x
```

```
>>> help(spam)
```

```
Help on function spam in module __main__:
```

```
spam(x)
```

```
    x-ті қарапайым қайтарады.
```

```
    Баптаулар:
```

```
        x: кейбіреулері, қандай мән екендігі маңызды емес
```

```
    Қайтарылатын мән:
```

```
        x: кейбіреулері, қандай мән екендігі маңызды емес
```

```
>>> import turtle
>>> help(turtle)
>>> help(turtle.color)
```

Пайда болған құжаттауда парақты алдына айналдыру үшін бос орын тетікшелерін, ал шығу үшін Q тетікшелерін басыңыз.

Пікірлер

Хэш (#) символы мен жол соңы арасындағы орналасатын функция пікірлер деп аталады. Құжаттау жолдарындағыдай Python түсіндіргіші пікірлерді елемейді. Ереже ретінде пікірлер Сіздің кодыңыздың оң жағында байыппен орнатылуы қажет. Бірақ, күрделі код аумағын түсіндіру үшін аса ұзын пікірлер де болады. Мұндай жағдайларда, біз мүмкін, бұл бөлімді бір немесе бірнеше жолдармен түсіндіріп өтуді дұрыс көретін шығармыз.

Жеткіліксіз және тым көп пікірлер жазу арасында өте жұқа шекара бар. Нақтылай келе, Сіз код не үшін арналғанына жоғары деңгейлі сипаттама жасаңыз келеді. Сіз, әр жол, тура мағынада, қандай әрекеттерді іске асыратындығын қайталағыңыз келмейді, себебі, бұл кодты оқуға мүлдем көмектеспейді. Сіздің бағдарламаңызды оқитын әр адам – бағдарламалаушы, сондықтан, Сізге Python әр нұсқамасы қандай қызмет атқаратындығын түсіндіру қажет емес екендігін есте сақтаңыз. Мұндай әрекеттер Сіздің кодыңызды артық ақпаратмен тым көп жүктейді және оны оқу үшін аса қиын етеді! Альберт Энштейн тұспалап айтқандай,

Егер сіз оны жай сөздермен түсіндіре алмасаңыз, яғни сіз оны аяғына дейін түсінбедіңіз.

Bloom функция денесіне сәйкес келетін пікірлер мысалдары төменде.

```
tortoise.pencolor('red') # тасбақа түс түрін қызылға орнатамыз tortoise.
fillcolor(fcolor)       # және бояу түсін заливки на fcolor
tortoise.begin_fill() орнатамыз
for segment in range(8):#
tortoise.forward(length) # бояулы 8 жақтының суретін саламыз
геометриялық гүл шоғыры
tortoise.left(135)
tortoise.end_fill()
```

Гүл шоғыры суретін салатын бес жолдар бірге пікірленген, бұл жай ғана бағдарламашы мақсатын белгілеу үшін жасалған. Салыстыру үшін келесі пікірлер қалай жасамау керектігін түсіндіру үшін берілген. Келесі пікірлер күрделі оқылады және өзімен ешқандай ақпарат алып келмейді.

```
tortoise.pencolor('red')      # тасбақа түс түрін қызылға орнатамыз
tortoise.fillcolor(fcolor)    # fcolor
tortoise.begin_fill() тасбақа түс бояуын орнатамыз    # өрнекті толтыра
бастаймыз
for segment in range(8):      # 0, 1, 2, ..., 7 тең segment үшін tortoise.
forward(length) # length tortoise.left(135)    # тасбақаны алдыға жыл-
жытамыз
Тасбақаны 135 градусқа солға бұрамыз 135 градусов
tortoise.end_fill()          # өрнек бояуды тоқтатамыз
```

Біз бұл бағдарламадығы басқа функцияларға пікірлер жазу мәселесін

Сипаттаушы атаулар мен ғажайып сандар

2.3. тарауында талқыланғандай, сипаттаушы атауларды пайдалану Сіздің бағдарламаңыздың мақсатын түсіндіруде өте маңызды қадам болып табылады. Гүл бағдарламасындағы ауыспалылар атаулары толық қалыпта, сондықтан басқа мысалды қарастырып көрейік. Келесі нұсқамаларға назар аударайық.

$$x = 462$$
$$y = (3.95 - 1.85) * x - 140$$

Қандай да бір басқапқы ақпаратсыз, бұл не екендігіне тұжырымдама жасау өте күрделі. Бірақ, егер біз екі ауыспалы атауын келесідей өзгертсек, онда мағынасы айқындала түседі.

$$\text{cupsSold} = 462$$
$$\text{profit} = (3.95 - 1.85) * \text{cupsSold} - 140$$

Бұл код қандайда бір сусын сатудан түскен пайданы есептейтіндігі айқын болды. Бірақ сандар мағыналары әлі де жасырын болып қала береді. Олар, бағдарламалаушылар жаргонды тілінде «ғажайып сандар» деп аталатын мысалдар болып табылады, себебі олар ешқайдан пайда болатын сияқты. «Ғажайып сандардан» алшақ жүрудің жоқ дегенде екі

себебі бар. Біріншіден, олар Сіздің кодыңызды оқылуға күрделі етеді және оның мағынасын боямалайды. Екіншіден, кодты тым күрделі және код өзгеруде қателерге аса ықпалды етеді, әсіресе, егер бір мән бірнеше рет пайдаланатын болса. Егер бұл сандарды ауыспалыларды сипаттайтын сипаттаушылармен бекітсе, онда код түсініктірек болады.

```
cupsSold = 462  
pricePerCup = 3.95  
costPerCup = 1.85  
fixedCost = 140  
profit = (pricePerCup - costPerCup) * cupsSold - fixedCost
```

Бұл жағдайда, біз «өзіндік құжаттау» деген түсінікті аламыз. Біз өзіміздің барлық ауыспалыларымызды және мәндерімізді сипаттаушы атаулармен атағандықтан, код мақсатын түсіну үшін оны жай ғана оқып шығу жеткілікті. Әрине тура сол ережелер функциялар атаулары мен баптауларына қолдануға болады. Өзіміздің функцияларымызды сипаттаушы атаулармен атай отырып, біз олардың мақсаттарын түсіндіреміз және шақыратын функциялар реттілігіне себеп боламыз. Бұл тәжірибені іске пайдалануды келесі баптарда көрсетіп өтеміз. Бұл кітапта, біз, кейде «бүкірТіркеуіш» («camelCase») деп аталатын атаулар беру келісімін пайдаландық, бұл тіркеуіште бірінші әріп төмен тіркеуішке ие, ал сөздің келесі бөліктері жоғарғы тіркеуіштен басталады. Бірақ түрлі бағдарламалаушылар түрі стильдер түрін таңдайды. Мысалы, кейбір бағдарламалаушылар «жылан тіркеуішті» («snake_case») таңдайды, мұндай тіркеуіште астын сызу белгісі сөздер арасында (cupsSold сөзі cups_sold сөзі болар еді) орналасады. Егер сіз стиль бойынша белгілі бір қағидалар ортасында жұмыс жасамасаңыз, таңдау сіздің қарамағыңызда, әрине, егер, нәтиже өзіндік құжатталған болса.

Жаттығулар

3.4.1. Біз осы тарауда талқылаған барлық өзгертулерді Өзіңіздің гүл суретін салу бағдарламаңызға енгізіп, қалған функциялар денелерін пікірлеуді аяқтаңыз.

3.4.2. 2.4.13 жаттығуынан Сіздің Ессіз кітапхана бағдарламаңызды іске асыратын функцияны жазыңыз, сосын оны шақыратын бағдарламаны толығымен (main функциясымен) жазып шығыңыз. Сіздің Ессіз кітапхана бағдарламаңыз бос орындарды толтыруға қажет сөздерді баптаулар ретінде алуы қажет. Сіздің main

функцияңыз бұл мәндерді input функциясын шақыру жолымен алып, оларды әрі қарай Сіздің функцияңызға жіберуі керек. Өз бағдарламаңызға құжаттау жолдары мен пікірлерді қосыңыз. Төменде 2.4.13 Жаттығуынан (құжаттау жолдары мен пікірлерсіз) функция мысалы берілген.

```
def party(adj1, noun1, noun2, adj2, noun3):
    print('Кешті қалай ұйымдастыруға болады')
    print()
    print('Егер сіз издесеңіз', adj1, 'тәсіл')
    print('Сіздің махаббатыңызды белгілеу', noun1 + ', костюмду кешке қалай қарайсыз')
    print(noun2 + '? Келесіден бастаңыз ')
    print('жасырылған шақыруларды жіберіңіз ', adj2, 'форматта,')
    print('бағыттаушы', noun3 + '.')
```

```
def main():
    firstAdj = input('Сіз хайуанаттар бағына бара жатырсыз ба? Қоршағандар сөзінде Сіз қандай болып көрінесіз? : ')
    firstNoun = input('Хайуанаттар бағында Сіз кімге бірінші болып жақындайсыз? : ')
    secondNoun = input('Қай топ Сіз үшін бой түршіктіреп? : ')
    secondAdj = input('Жолбарыс бүгін қандай көңіл күйде? : ')
    thirdNoun = input('Хайуанаттар бағынан кейін Сіз қай жерге барасыз?: ')
    party(firstAdj, firstNoun, secondNoun, secondAdj, thirdNoun)
main()
```

3.4.3. Келесі бағдарламаны зерттеңіз, сосын, бір немесе бірнеше басқа функцияларды шақыратын main функциясымен оны түрлендіріп көріңіз. Сіздің main функцияңыз өте қысқа болуы тиіс: тасбақа жасау, Өзіңіздің функцияларыңызды шақырып, тышқан шертуін күту. Өзіңіздің бағдарламаңызды сәйкес пікірлермен және құжаттау жолдарымен құжаттаңыз.

```
import turtle
george = turtle.Turtle()
george.setposition(0, 100)
george.pencolor('red')
george.fillcolor('red')
```



```

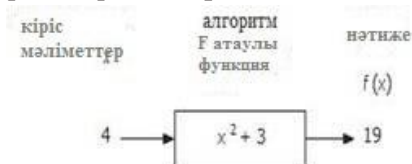
george.begin_fill()
george.circle(-100, 180)
george.right(90)
george.forward(200)
george.end_fill()
george.up()
george.right(90)
george.forward(25)
george.right(90)
george.forward(50)
george.left(90)
george.down()
george.pencolor('white')
george.fillcolor('white')
george.begin_fill()
george.circle(-50, 180)
george.right(90)
george.forward(100)
george.end_fill()
screen = george.getscreen()
screen.exitonclick()

```

3.5 ФУНКЦИЯЛАРҒА ҚАЙТА ОРАЛУ

Осыған дейін біз функцияны баптаулар деп аталатын бір немесе бірнеше кіріс мәліметтерін алып, қайтарылатын мән деп аталатын нәтижені қайтаратын есептеу ретінде сипаттап келдік. Бірақ, біз осы уақытқа дейін жазып келген функция қайтарылатын мәнге ие болмағын. Осы бөлімде біз бұны түзейміз.

$f(x) = x^2 + 3$ қарапайым математикалық функцияға қайта оралайық. Біз бұл функция кіріс барысында x алып, $x^2 + 3$ нәтижесін есептеп шығаратындығын түсінеміз. Мысалы, $f(4) = 4^2 + 3 = 19$; егер 4 кіріс болса, онда шығыста мән 19 тең болады. Кіріс мәліметтерді нәтижеге түрлендіретін алгоритм, әрине – бұл «3 санын x шаршысына қосу».



Біздің f математикалық функциямыздың Python баламасы келесідей күйде болады

```
def f(x):  
    return x ** 2 + 3
```

Return нұсқамасы функция нәтижесін анықтайды. Функцияны ақытау өз алдына еш нәрсені есептемейтіндігін есте сақтау қажет. Біз функцияны оны орындау үшін шақыруымыз керек. Мысалы, бағдарламада біз, өзіміздің f функциясызды келесідей шақыра аламыз:

```
def main():  
    y = f(4)  
    print(y)  
main()
```

Main-де үлестіру нұсқамасын іске асырғаннан кейін, оң жақ 4 аргументі f функциясын шақырады. Осылайша, f функциясы x баптауына 4 мәнін бергеннен кейін іске асырылады. Мұны x функциясын 4 мәніне алмастыра отырып, келесідей көрсету ыңғайлы:

$$y = (3 * \underbrace{f(4)}_{19} + 2) ** 2$$
$$\underbrace{\hspace{1.5cm}}_{59}$$
$$\underbrace{\hspace{2.5cm}}_{3481}$$

Сосын $4 ** 2 + 3 = 19$ мәні есептеліп, функцияға қайтарылады. Бұл қайтарылатын мән $f(4)$ функциясын шақырумен байланысты мән болады, яғни, нәтижелей келе, main-де y берілетін мән болады. $F(4)$ функциясы 19 мәнін қабылдайтындықтан, біз main-дегі екі жолды нұсқа ретінде

```
def main():  
    print(f(4))
```

ауыстырар едік. Осыған ұқсас түрде, біз қайтарылатын мәнді функцияларды шақыруды аса ұзын өрнекке қосуға мүмкіндік алар едік, мысалы мынадай

$$y = (3 * f(4) + 2) ** 2$$

Бұл мысалда $f(4)$ функциясы 19 мәнін қабылайды, сосын, өрнектік

қалған бөлігін есептейді:

2.3 Бөлімінде, біз көлем және үлестіру операторының оң жағынан жел жылдамдығы ескерілген температура сияқты үлкендіктерді есептегенімізді еске саламыз.

$$\text{volume} = (4 / 3) * \pi * (\text{radius} ** 3)$$

Сосын, radius басқа мәнді көлемді немесе wind басқа мәнді жел жылдамдығы ескерілген температураны есептеу үшін біз дәл сол формуланы қайталап енгізуіміз керек еді. Бұл шын мәнінде, шаршатарлық және қатеге душар болғыш болатын, мысалы, егер біз бір бума попкорн жасау үшін микротолқынды пешті қайтадан жасауға тура келетіндей. Бізге соншалықты жетіспейтін – бұл функция. Әр осы есептеулер үшін функциялар жасағаннан кейін, түрлі кіріс мәліметті нәтижелерді есептеу тек функцияны қайтадан шақырумен шектеледі. Мысалы, шар ауданын келесі функция көмегімен есептеп шығарға болады:

```
import math
def volumeSphere(radius):
    """Шар көлемін есептейді.
    Баптау:
        radius: шар радиусы
    Қайтарылатын мән: радиусы берілген шар
    көлемі"""
    return (4 / 3) * math.pi * (radius ** 3)
def main():
    radius1 = float(input('Radius of first sphere: '))
    radius2 = float(input('Radius of second sphere: '))
    volume1 = volumeSphere(radius1)
    volume2 = volumeSphere(radius2)
    print(Шар көлемі құрайды: ', volume1, ' и ', volume2)
main()
```

Функциядағы return нұсқамасы берілген radius шар көлемін беріп, оның мәнін қайтарады. Енді, Теперь, main функциясы үшін жоғарыда көрсетілгендей, екі түрлі шарлар ауданын есептеу қарапайым екі реттік функцияны шақыруда. Осыған ұқсас, бізде 20 радиусты сфера ішінде 10 радиусты қатты шар бар деп болжамдайық. Сфера ішінде қалған бос кеңістікті есептеу үшін біз келесідей есептейміз

$$\text{emptyVolume} = \underbrace{\text{volumeSphere}(20)}_{33510.3216\dots} - \underbrace{\text{volumeSphere}(10)}_{4188.7902\dots}$$

$$\underbrace{\hspace{10em}}_{29321.5314\dots}$$

Функцияның қайтарылатын мән анықтауына қосымша return нұсқасы, осыған қоса, функция орындалуын аяқтап, алынған мәндерді шақырылған функцияға қайтадан қайтарады. Осылайша, шын мәнінде return нұсқамасы екі әрекетті іске асырады. Return нұсқамасы

1. Функцияның қайтарылатын мәнін анықтайды, және
2. Функцияның аяқталуына алып келеді.

Сондықтан, егер біз функцияға нұсқаманы return нұсқамасынан кейін қоссақ олар ешқашан орындалмайды. Мысалы, функцияны басып шығару қызылмен белгіленген шақыру келесі бағдарламада жұмыс жасамайды.

```
def volumeSphere(radius):
    """Шар ауданын есептейді.

    Баптау:
        radius: шар радиусы

    Қайтарылатын мән:
        Радиусы берілген шар ауданы
        """

    return (4 / 3) * math.pi * (radius ** 3)
    print('Бұл ешқашан басылып
    шығарымайды.')
```

«Return» «Print»-ке қарсы

Жаңадан үйреніп келе жатқандардың ең кең тараған жасайтын қатесі функцияны return нұсқамасын шақытып жабудың орнына print функциясын шақыру. Мысалы, болжамдап көрейік, біз volumeSphere функциясында return-ді print-ке алмастырдық делік:

```
def volumeSphere(radius):
    """Шар ауданын есептейді.

    Баптау:
        radius: шар радиусы

    Қайтарылатын мән:
        Берліген радиусты шар ауданы
    """

    print((4 / 3) * math.pi * (radius ** 3))
    # ДҰРЫС ЕМЕС
```

Күтілетін жауап басылып жазылатын болады, бірақ біз мәнді қайтармадық, ондықтан функцияның қайтарылатын мәні None болып қала береді. Сондықтан, егер біз есептеп шығаруға тырыссақ, біз қатеге тап боламыз, себебі emptyVolume = None – None-ге үлестіруге тырысамыз, ал мұның, өз алдына мағынасы жоқ.

$$\text{emptyVolume} = \underbrace{\text{volumeSphere}(20)}_{\text{None}} - \underbrace{\text{volumeSphere}(10)}_{\text{None}}$$

None - None ???

Жаттығулар

Келесі есептердің әрқайсысы үшін функция жазыңыз. Әр жеке жағдай бойынша функция берліген мәнді қайтаратындығына (тек ғана экранға шығарып қоймай) сенімді болыңыз.

Number1 және number2 сандар соммасын қайтаратын

sum(number1, number2)

функциясын жазыңыз. Сондай-ақ, осы екі мәндерді input функциясы көмегімен алып, оларды Сіздің sum функцияңызға беріп, сосын sum функциясының қайтарылатын мәнін экранға шығаратын бағдарламаны толығымен (с функцией main функциясымен) жазып шығыңыз.

3.5.1. baseexponent мәнін қайтаратын

power(base, exponent)

функциясын жазыңыз. Осы екі мәндерді input функциясы көмегімен толықтай (main функциясымен) қайтарып, оларды Сіздің power функцияңызға беретін, ал сосын baseexponent қайтарылатын мәнін экранға шығаратын бағдарлама жазыңыз.

3.5.2. Сіздің америкалық футбол командасының шотын қайтаратын

football(touchdowns, fieldGoals, safeties)

Функция жазыңыз, әрине егер тачдаунлар саны (7 ұпай), қақпаға соғылған голдар (3 ұпай) саны, сондай-ақ сейфти (2 ұпай) саны баптау ретінде берілетін болса. Осы үш мәндерді input функциясы көмегімен толықтай алып (main функциясымен), Сіздің football функцияңызға беріп, сосын экранға шотты шығаратын бағдарламаны жазыңыз.

3.5.3. 2.4.8 Жаттығуында екі нүктелер арасындағы қашықтықты қалай есептеуге болады деген сұрақ туындаған еді. Енді (x1, y1) и (x2, y2) нүктелері арасындағы қашықтықты қайтаратын

distance(x1, y1, x2, y2)

функциясын жазыңыз.

3.5.4. Мінсіз газ заңы $PV = nRT$ деген қағиданы білдіреді, бұл жерде

- P = атмосферадағы қысым (Атм)
- V = литрлердегі көлем (л)
- n = мольдердегі газ бөлшектерінің саны (моль)
- R = әмбебап газдық тұрақтылық = 0.08 л Атм / моль К
- T = Кельвиндердегі абсолюттік газ температурасы (К)

Цельсия градустарындағы T тепмературада және P қысымда болатын V литрлер құрамды мінсіз газ бөлшектер моль санын қайтаратын moles(V, P, T)

функцияны жазыңыз. (Цельсия градустарындағы функцияны Кельвин градустарына түрлендіруді ұмытпаңыз). Осы үш мәнді input функциясы көмегімен алып, оларды Сіздің moles функцияңызға қайтарып, сосын экранға мінсіз газ моль бөлшектер санын шығаратын бағдарламаны толықтай (main функциясымен) жазыңыз.

3.5.5. Сізде мінсіз газды екі контейнер бар деп болжамдайық. Біріншісінде 1.5 Атм қысымды және 20 Цельсия градус температуралы 10 л газ бар. Екіншісінде 2 Атм қысымды және 30 Цельсия градус температуралы 25 л газ бар. Екі контейнерлердегі мінсіз газ бөлшекті мольдер ортақ санын есептеу үшін алдыңғы жаттығудан екі функция шақыруды қалай қолдануға болатындығын ойластырып көріңіз.

Енді, Өзіңіздің moles функцияңызда return нұсқамасын print шақыру үшін алмастырыңыз. (Яғни, Сіздің функцияңызда eturn нұсқамасы болмайды). Сіз мольдердің ортақ санын бұрынғыдай тәсілмен есептеп шығара аламыз ба? Егер солай болса, із жүзінде дәлелдеп көрсетіңіз. Егер олай болмаса, неліктен екендігін түсіндіріңіз.

3.5.6. Біздің планетамыздың өмір сүруі көп жағдайда жер асты суларына тәуелді. Сондықтан, ластанудың болуы мүмкін қатерлерін анықтау үшін жер асты сулар ағысын бақылап отыру мүмкіншілігіне ие болу өте маңызды. Дарси заңы (ламинарлы сүзгілеу заңы) ол туралы былай дейді: борпылдақ орта (мысалы, құм, қара жер) арқылы өтетін сұйықтық ағысы (мысалы, су) ортаның ағысты өткізу (сүзгілеу) мен ағыс гидравликалық градиенті мүмкіншілігіне тікелей байланысты:

$$dh Q = K dl$$

бұл жерде

- K – ортаның су өткіздігіштігі, қалыңдығы l м аумақ/уақыт бірліктерінде өлшенетін орта қабатынан сұйықтықтың өту жылдамдығы

- dh/dl – бұл гидравликалық градиент
- dh – ағынның (биіктіктің) азаюы (азаюшы ағын үшін теріс)
- dl – сүзгілеу жолының ұзындығы (көлденең жазықтықта)

Берілген параметрлермен су ағысының ең көп шығынын есептейтін darcy(K, dh, dl)

функцияны жазыңыз.

130 м²/тәу су өткіздіккішке ие төбе ішінде жер асыты суының ең көп шығынын және 1 км қашықтықта 50 метрлік су ағының азаюын есептейтін функцияңызды қолданыңыз.

3.5.7. Адам денесі масса индексі (ДМИ немесе BMI) келісі формула бойынша есептелінеді:

w

$$BMI = h^2 \cdot 703$$

102 □ Визуалды абстракция

Бұл жерде w - футтардағы алам салмағы, h – дюймдардағы адам бойы.

Сәйкес ДМИ-ін қайтару үшін осы формуланы пайдаланатын

bmi(weight, height)

функциясын жазыңыз. Сіз (немесе Сіздің ата-анаңыз) компакт-дисктен әндер көшіруде, файл дыбысты уақыт ішінде белгілі бір жылдамдықпен дискредиттеу қолымен жасалы. Дикредитациялаудың стандарты мәндері 128 Кбит/с (секундына 128×210 бит), 192 Кбит/с, және 256 Кбит/с.

Біреу өз iPod-на орната алатын ұзындығы 4 минутты құрайтын әндер санын қайтаратын

songs(capacity, bitrate)

функцияны жазыңыз. Функцияның екі баптауы ол гигабайттағы (ГБ) iPod жады мен Кбит/с-ғы дискредитация жылдамдығы болып табылады. Гигабайт бұл 230 байт, ал байт 8 биттентұрады. Осы екі мәндерді input функциясы көмегімен толығымен алып, оларды Сіздің

songs функцияңызға беріп, сосын экранға әндер санын шығаратын бағдарламаны толығымен (main функцисымен) жазыңыз.

3.5.8. Компьютер жыдамдығы жиі гигагерцтерде (ГГц) беріледі (оңайлатып), олар компьютер ішкі сағатының миллиард рет «тықылдауымен» есептеледі. Мысалы, 2 ГГцті компьютерде сағат секундына 2 миллиард рет «тықылдайды». Бір нұсқаманы орындау үшін компьютерге 3 «тықылдау» қажет. Гигагерцті сағаттық жылдамдықпен компьютерде берілген нұсқамаларды орындауға кеткен уақытты секунта қайта қайтаратын

time(instructions, gigahertz)

функцияны жазыңыз. Мысалы, time(1e9, 3) 1 (секундты) қайтаруы тиіс.

3.5.9. 2.3.4 Жаттығуында екі ауыспалыларды қалай орын ауыстыруға болатындығы жайлы сұрақ қозғалған. Екі баптаулар мәндерін орын ауыстыру үшін функция жазуға бола ма? Басқа сөздермен айтқанда, біз мына функцияны шақырып,

swap(a, b)

функцияны шақырғаннан кейін x мәні 1ге, ал y мәні 10-ға тең болу үшін оны былайша атасак?

x = 10

y = 1 swap(x, y)

(Функция ештеңені қайтаруға тиісті емес) Егер солай болса, онда бұл функцияны жазыңыз. Егер олай болмаса, неліктен екендігін түсіндіріңіз.

3.5.10. Бағыт бағалау үшін 0ден 99 дейін толық сан берілген, біз ол санды келесі шкала бойынша баламалы бағалауға түрлендіреміз: 90–99: 4, 80–89: 3, 70–79: 2, 60–69: 1, < 60:0.

Алынған бағалауға балама ұпайды қайтаратын (яғни, орташа академиялық ұпайды)

gradePoint(score)

функциясын жазыңыз. (Көмек: қалдықсыз блу мен екі немесе одан көп санның ең үлкенін қайтаратын енгізілген max функциясын пайдаланыңыз.)

3.5.11. Time.time () (үлгілеуде time деп аталады) 1 қантар 1970 жылдан бастап ағымдағы ауқытты секундтарда қайтарады.

3.5.12. Ағымдағы жылды толық мән ретінде қайтару үшін осы функцияны пайдаланып

year()

функциясын жазыңыз.

3.6 ҚОЛДАНУ САЛАСЫ МЕН АТАУЛАР КЕҢІСТІГІ

Бұл тарауда біз, функция ішіндегі ауыспалылар атауын пайдалануды толығырақ талқылаймыз. Мысал ретінде 2.3 бөліміндегі main функциясынан шақырылатын жел жылдамдығы ескерілген температураны есептеу түрінде іске асырылған функцияны байқап көрейік.

```
def windChill(temp, wind):  
    """Берілген температура мен жел жылдығы үшін жел жылдамдығы  
    ескерілген солтүстік америкалық температурасының метрикалық  
    баламасы.
```

Баптаулар:

temp: Цельсия градустарында берілген температура

wind: км/ч-та берілген 10 м-ік биіктіктегі жел жылдамдығы

Қайтарылатын мәндер:

Ең жақын толық санға дейін жуықталған Цельсия градусындағы жел жылдамдығы ескерілген температура
"""

```
chill = 13.12 + 0.6215*temp + (0.3965*temp - 11.37) *  
wind**0.16 return round(chill)
```

def main():

```
chilly = windChill(-3, 13) print('Температура с учетом ветра – ',  
chilly)
```

```
main()
```

Біз windChill функциясы ішіне есептеуді аздап бөлшектеу үшін chill атаулы ауыспалыны енгізгенімізге назар аударыңыз. Біз temp және wind баптаулары қолдана отырып жел жылдамдығы ескерілген температура деңгейін есептеу нәтижесіне chill нәтижесін береміз, сосын осы мәнді ең жақын толық санға дейін жуықтап қайтарамыз. Біз windChill функциясы ішінде chill анықтағандықтан, оның қолдану аумағы функция үшін жергілікті болып табылады. Басқаша айтқанды, егер біз windChill функциясы шеңбері сыртында chill-ға беп бұруға тырыссақ (мысалы, main функциясында) біз қате жайлы келесі хабарлама аламыз:

NameError: name 'chill' is not defined

Chill жерлілікті аумақтық қолдануға ие болғандықтан, ол жергілікті

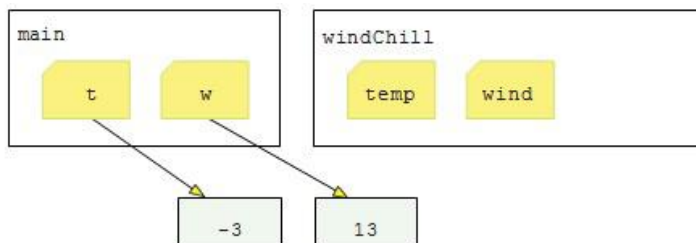
ауыспалы деп аталады.

Атаулардың жергілікті кеңістігі

Ауыспалылар мен баптаулар жергілікті атаулары қолдану саласының ережелері қалай жұмыс жасайтындығын анық түсіну үшін Python бағдарламасында атауларды басқару қалай іске асатындығын толығырақ қарастырайық. Бұны аса қызықты ету үшін, төменде қызылмен белгіленген өзіміздің жел ескерілген температура бағдарламасын аздап өзгертейік:

```
def windChill(temp, wind):  
    """ (құжаттау жолдары берілмеген) """  
  
    chill = 13.12 + 0.6215*temp + (0.3965*temp - 11.37) *  
    wind**0.16 temp = round(chill)  
    return temp  
def main():  
    t = -3  
    w = 13  
    chilly = windChill(t, w) print('Жел ескерілген температура  
    - ', chilly)  
  
main()
```

WindChill функциясын шақырғаннан кейін, бірақ temp және wind баптауларына t мен w аргумент мәндері берілгенге дейін, осы бағдарлама үшін біз жағдайды келесідей көрсете аламыз:



T мен w айналасындағы тік төртбұрыш main функциясын пайдалану аумағы болып табылады, ал temp және wind айналасындағы тік төртбұрыш - windChill функциясын пайдалану аумағы болып табылады. Әр бөлек жағдай, пайдалану аумағында осы функция ішінде қандай атаулар анықталғаны немесе мәндерге ие екендігін анықталады. Сурет-

те біз мәнді белгілеу үшін бағыт сызықшалары орнына «стикерлерді» пайдаланамыз, бұл олардың пайдалану аумағына мәндер емес атаулар жататындығын түсіндіру үшін жасалады. Атаулар – бұл жадта орналасқан мәндер элементтеріне жасалатын сілтеме. Қолдану аумағы Python-дағы функция, ол *атаулар кеңістігімен* басқарылады.

Атаулар кеңістігінің функциясы – осы функцияда анықталған мәндерге сілтеме беретін қарапайым атаулардың тізімі. Біз кез келген функцияның атаулар кеңістігін оның ішінде locals функциясын шақыра отырып қарай аламыз. Мысалы, windChill функциясын шақырар алдында main функциясы ішіне келесі нұсқаманы қойыңыз:

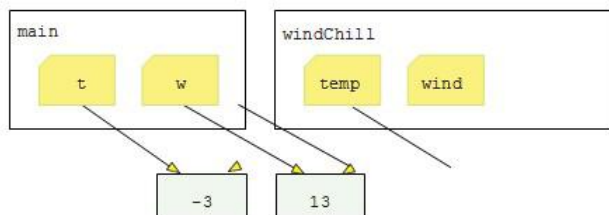
```
print(' windChill функциясын шақырар алдындағы main атаулар кеңістігі windChill: ', locals())
```

Біз бағдарламаны іске қосқанда келесіні көреміз:

windChill функциясын шақырар алдында main атаулар кеңістігі windChill: {'t': -3, 'w': 13} Жел ескерілген температура деңгейі – -8

Атаулар жергілікті кеңістік бағдарламасының осы жерінде main функциясы екі атаудан тұратындығы байқалып тұр: -3 мәні берілген t, және 13 мәні берілген w (дәл осылай біз жоғарыда көрсетіп өттік). Атаулар кеңістігін көрсету айналасындағы мүсінді жақшалар ({}), Python мәліметтерінің басқа түрі болып табылатын кеңістіктер атауының сөздігіне меңзейді. Біз сөздіктерді 8 Тарауда толығырақ талқылаймыз.

Бағдарламаға қайта оралып, сәйкес аргументтерді мәнге атау бері windChill баптауын қашыруда, біз temp = t және wind = w атауын береміз, сондықтан, сурет келесіге ауысады:



Біз бұны windChill функциясының бірінші жолына locals басқа шақыруын қою арқылы көре аламыз:

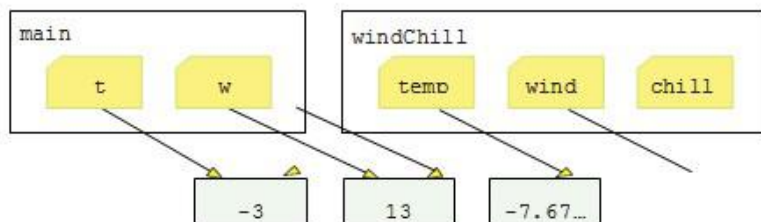
```
print(' windChill стартындағы атаулардың жерлілікті кеңістігі: ', locals())
```

Енді біз бағдарламаны іске қосқаннан кейін келесіні көреміз

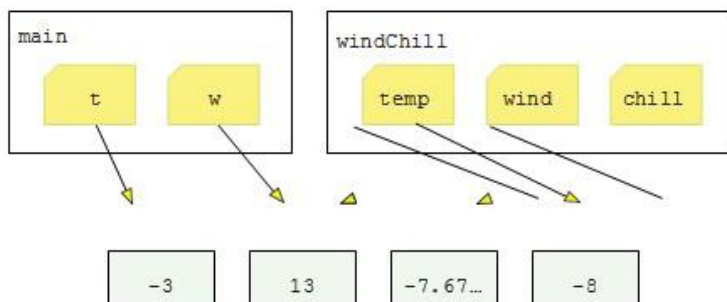
windChill шақыру алдындағы main атауының кеңістігі: {'w': 13, 't': -3} windChill бастауында атау жерлігікті кеңістігі : {'temp': -3, 'wind': 13} Жел ескерілген температура – -8

Бұл бізге, windChill функциясының басында жалғыз көрінетін атаулар – бұл t мен w мәні берілген temp және wind атаулары. Бірақ, t мен w мәндері windChill функциясының ішінде жоқ және t мен temp немесе w мен wind арасында ешқандай тура байланыс жоқ екендігін атап өту керек; олар өздеріне мәндер арқылы жай ғана жанама байналысты болып келеді.

Өрі қарай атау бері нұсқамасындағы функция ішіндегі нұсқама chill жерлгілікті ауыспалысын пайдаланып, windChill функциясының атаулар кеңістігін қолдана отырып оған chill жаңа атауын үлестіреді және оған жел ескерілген температура есептеу нәтижесін береді.



Қорытындылай келе, функция жел ескерілген жуықталған температура мәнін temp жергілікті баптауына береді:



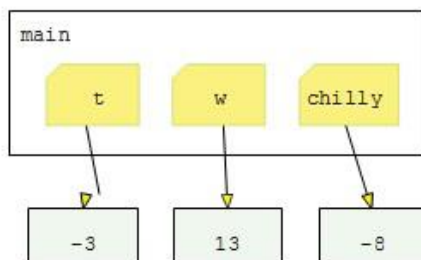
Бұл жағдайлардың реттілігін бағдарламада көру үшін windChill функциясында return нұсқамасының дәл алдына басқа locals шақыруын қояйық:

```
print(' windChill соңындағы жерлігікті атау кеңістік: ', locals())
```

Енді,біз бағдарламаны іске қосқаннан кейін келесі жағдайды байқаймыз:

```
windChill қашыру алдындағы атау жергілікті кеңістігі: {'t': -3, 'w': 13}
windChill бастауда атау жергілікті кеңістігі: {'wind': 13, 'temp': -3}
windChill соңында атау жергілікті кеңістігі: {'wind': 13, 'chill': -7.676796032159553, 'temp': -8}
Жел ескерілген температура – -8
```

windChill функциясы –8 жел ескертілген температураны қайтарғаннан кейін, windChill атау кеңістігі, сондай-ақ, осы атаулар кеңістігіндегі барлық жергілікті атаулар жоққа шығады, олар main атаулар кеңістігінде тек t және w атауларын өзгертпейді. Бұдан басқа, chilly жаңа атауы main атаулар кеңістігінде жасалады және оған windChill функциясының қайтарылатын мәні беріледі:



Пайымдауды тексеру үшін main мәнінің соңына тағы да бір locals шақыруын қосайық:

```
print(' windChill шақырудан кейін main атау кеңістігі: ', locals())
```

Осындай жағдайда 3.10 суретінде көрсетілген қорытынды бағдарлама келесіні экранға шығарады:

```
windChill шақырудан кейін main атау кеңістігі: {'t': -3, 'w': 13}
windChill бастау алдында атаудың жергілікті кеңістігі: {'wind': 13, 'temp': -3}
windChill соңындағы атаудың жергілікті кеңістігі: {'wind': 13, 'chill': -7.676796032159553, 'temp': -8}
Жер ескерілген температура – -8
```

```
windChill шақырудан кейін main атау кеңістігі: {'t': -3, 'w': 13, 'chilly': -8}
```

Бұл мысалдың ең маңызды тұжырымдамасы - біз баптаулар болып табылатын сандармен жұмыс жасауда баптау мен оған сәйкес аргумент өзара байланысты емес; біз temp баптауымен жасағандай, баптауға берілген мәнді өзгертсеңіз бұл жағдай сәйкес аргументті өзгертпейді,

яғни t аргументін өзгерпейді.

Атаудың жаһандық кеңістігі

3.3 Бөлімінде біз ауыспалылардың жаһанды атауларын және оларды неліктен қолдануға болмайтындығы жайлы өз қозғағанымыз жайлы еске сала өтейік. Ауыспалылардың жаһанды атаулары жататын атаулар кеістігі *атаулардың жаһанды* кеңістігі деп аталады. Біз globals функциясын шақыра отырып атаулардың жаһанды кеңістік құрамын қарай аламыз. Мысалы, жоғарыда берілген біздің бағдарламаға main соңына келесі globals шақыруын қосып келесі әрекетке назар аударайық:

```
print('Атаудың жаһандық кеңістігі: ', globals())
```

Нәтиже келесіге ұқсас болуы мүмкін (кейбір атаулар көрсетілмеген):

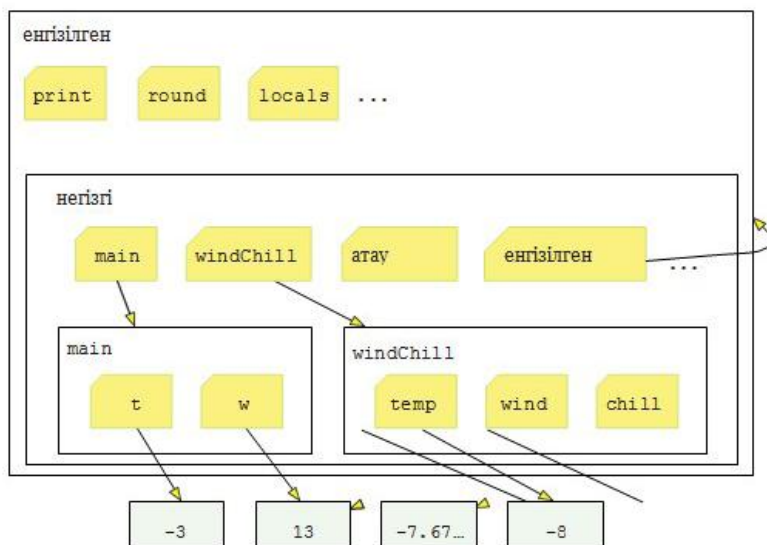
```
Атаулардың жаһандық кеңістігі: {'main': <function main at
0x10065e290>, 'windChill': <function windChill at 0x10065e440>, '__
name__': '__main__',
'__builtins__': <module 'builtins' (built-in)>, ... }
```

Біз жасаған жалғыз жаһанды атаулар main және windChill екі функциямызға жататындығына назар аударыңыз. Бұл атаулардың әрқайсысын төмендегі суретте көрсетілгендей функция атаулар кеңістігіне сілтеме ретінде бере аламыз (кейбір атауларға сілтеме берілмеген):

```
def windChill(temp, wind):
    """ (құжаттау жолдары берілмеген) """
    print(' windChill іске қосудағы атаулардың жергілікті кеңістігі: ',
    locals()) chill = 13.12 + 0.6215*temp + (0.3965*temp - 11.37) *
    wind**0.16 temp = round(chill)
    print(' windChill соңындағы атаулардың жергілікті кеңістігі: ',
    locals()) return temp

def main():
    t = -3
    w = 13
    print(' windChill шақырар алдында main атаулар кеңістігі: ', locals())
    chilly = windChill(t, w)
    print('Жел ескерілген температура – ', chilly)
    print(' windChill шақырылғаннан кейін main атау кеңістігі: ',
    locals()) main()
```

3.10 сурет locals функциясын толықтай шақырумен жел ескерілген температура есептеу бағдарламасы.



Жаһанды атаулар кеңістігінде анықталған басқа атаулар – бұл Python әр бағдарламасында анықталған стандарты атаулар. Атау ат беруі ағымдағы үлгі атауына жатады, яғни ‘__негізгі__’ (main негізгі функциясының атаулар кеңістігімен шатыстырмаңыз), себебі біз ең басынан Python түсіндіргішінде жасап келген файл ішіндеміз.

Енгізілген__ атауы айқын түсіндірілетін үлгіге жатпайды, бұл атау Python бағдарламасының барлық енгізілген функцияларынан құралады.

Суретте көрсетілгендей, біз бұл атаулар кеңістігін бір біріне енгізіп көрсете аламыз, себебі жергілікті кеңістікте орналаспаған атаулар сыртқы кеңістіктерде ізделінеді. Мысалы, егер біз main функциясы ішіне бола тұра print функциясын шақырсак, онда ең алдымен Python түсіндіргіші бұл функция атауын атаулар жергілікті кеңістігінде іздейді. Оны ол ерде таппағаннан соң, ол келесі сыртқы __негізгі__ деп аталатын атау кеңістігіне көшеді.

Тағы да оны ол жерде таппағаннан соң, ол __енгізілген__ деп аталатын атаулар кеңістігіне ауысады. Біз импорттайтын әр үлгі өзінің жеке кеңістік атауын анықтайды. Мысалы, біз import math функциясын пайдалана отырып math үлгісін импорттағанда атаудың жаңа кеңістігі

төменде көрсетілген суреттегідей main атауы кеңістігі енгізілгеннің ішінде жасалады.

Біз math-қа math үлгісіндегі (мысалы, math.sqrt (7)) әр функция алауын берген кезде біз Python түсіндіргішіне функцияны math атау кеңістігінде іздеу керектігін хабарлаймыз.

Бізге беріліп отырған үлгіге ұқсас менталды үлгіні колдау. Сіз өз бағдарламаңызда пайдаланатын атауларды басқаруға көмектесуі тиіс, әсіресе олар ұзарған кезде.

Жаттығулар

3.6.1. 3.5.4 жаттығуындағы функцияның ауыспалылар барлық жергілікті атауларын экранға шығару үшін locals функциясын қалай қолдану керектігін көрсетіңіз. (3, 7.25) және (9.5, 1) нүктелері арасындағы арақашықтықты есептеп шығару үшін функцияның өзгертілген нұсқасын пайдаланыңыз. Функция орындауда атаулардың жергілікті кеңістігі қандай күйде болады?

```
import turtle
```

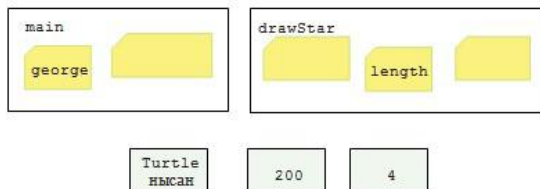
```
def drawStar(tortoise, length): for segment in range(5):  
    tortoise.forward(length) tortoise.left(144)
```

```
def main():
```

```
    george = turtle.Turtle() sideLength = 200 drawStar(george, sideLength)  
    screen = george.getscreen() screen.exitonclick()
```

```
main()
```

106 беттегі суретке ұқсас сурет салыңыз, ол drawStar функциясын қайтарар алдында осы бағдарламада атаулар кеңістігін көрсететін сурет болуы тиіс. Келесі сурет бастау үшін берілген:



3.6.3. Экономикада сұраныс функциясы тұтынушы тауар үшін төлеуге дайын бағаны береді, бұл тауардың тек белгілі бір көлемі қол жетімді екендігін ескере отырып. Мысалы, кофе дәндері нарығында

сұраныс функциясы келесі өрнекпен есептелінеді

$$2.3Q \text{ D}(Q) = 45 - 1000 ,$$

Бұл жерде Q – қол жеткімді кофе көлемі, ол киллограммен өлшенеді және қайтарылатын баға да – 1 кг құрайды. Мысалы, егер 5000 кг кофе дәндері қолда бар болса, онда 1 кг тауар бағасы $45 - (2.3)(5000)/1000 = 33.50$ доллар болады. Келесі бағдарлама осы мәнді есептейді.

```
def demand(quantity):  
    quantity = quantity / 1000 return 45 - 2.3 * quantity  
def main():  
    coffee = 5000  
    price = demand(coffee) print(price)  
main()
```

106 беттегі суретке ұқсас сурет салыңыз, ол demand функциясын, сондай-ақ main функциясын қайтарар алдында бұл бағдарламадағы атаулар кеңістігін көрсететін сурет болуы тиіс.

4 бөлім

Өсу мен кему

Жаратылыстану мен қоғамдық ғылымдар саласындағы аса маңызды сұрақтардың бірі келешек уақытта популяция мен басқа өлшемдердің динамикалық санына жатады. Мысалы, бізді агрессиялық ықпал астында орналасқан өсімдіктердің популяция көлемі, халық санына әсер ететін инфекцияның таралу жиілігі, қоймада сақталатын радиоактивті материалдар көлемі, тауардың әлем нарығына енуі немесе динамикалық әлеуметтік желінің даму мінездемелері қызықтыруы мүмкін.

Мүмкіндіктер шексіз.

Осыдай жағдайдарды зерттеу үшін ғалымдар жеңілдетілген үлгіні ойлап тапқан, осындай үлгі шынайы жағдайдың негізгі нышандарын оқшаулатып, жағдайды оңай қабылап, зерттеуге мүмкіндік береді. Бұл мағынада үлгі абстракцияның тағы да бір мысалы болып табылады. Мәселені сипаттайтын үлгіні тапқан сәтте біз үліні қолдануды ұзақ уақыт бойы имитациялай аламыз. Үлгілеу мен имитациялау күші не өткен бақылаулар үшін теориялық негіз беру қабілетінде, не болашақта жүрістұрысты болжамдау қабілетінде. Ғалымдар өте жиі үлгілерді дәстүрлі эксперименттермен бірге пайдаланады, бұл олардың бақылауларын болжамды теориялық базамен салыстыру үшін жасалады. Бұл параллельді ғылыми жұмыс барыстары 4.1 суретінде көрсетілген. Сол жағында есептеу жұмыс барысы орналасқан. Үлгі алгоритмдікке қарағанда көбірек математикалық болатындығына назар аудару үшін біз «алгоритм» орнына «үлгі» қолданамыз. (біз бұл барыс туралы 7.1 Бөлімінде толығырақ сөйесеміз). Оң жақта ғылыми тәсілмен жүргізілген параллельді эксперименттік барыс орналасқан. Есептеушілік және эксперименттік барыстар нәтижелерін салыстыруға болады, бұл үлгіні нақтылауға немесе нәтижелер сапасын жоғарлату үшін жаңа эксперименттерге алып келуі мүмкін.

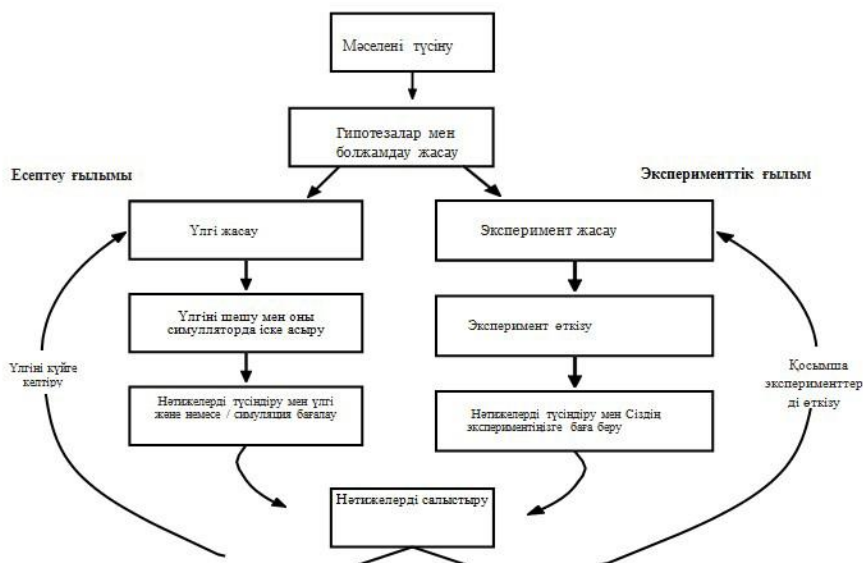


Рисунок 4.1 Параллельді эксперименталды және есептеу барыстары.

Біз популяция дамуының динамикалық үлгілеуін жасауда уақыт дискретті қадамдармен жүреді деп болжамдаймыз және кез келген уақыт аралығында популяцияның көлемі алдыңғы уақыт шеңберіндегі популяция көлеміне негізделген деп санаймыз. Нақты есепке қарай уақыт бөлігі наносекундтан жүздеген жылға дейін өлшенуі мүкін. Тұтас алғанда, уақыттың жаңа жиілігі туылумен және иммиграциямен байланысты халық санының өсуін және өлім мен жер аударумен байланысты халық санының төмендеуін алып келуі мүмкін. Бұл тарауда біз аккумулятор деп аталатын фундаменталды алгоритмдік тәсілді қарастырамыз, бұл тәсілді біз динамикалық барыстарды үлгілеуде қолданатын боламыз. Аккумуляторлар барлық есептер түрлерінде берілген және көптеген түрлі алгоритмдік тәсілдер негізінде жатыр. Біз мысалдарды қарастыруды осы кітап бойы жалғастыра береміз.

4.1 ДИСКРЕТТІ ҮЛГІЛЕР

Балық аулау шаруашылығын басқару

Болжамдап көрейік, біз 1 қаңтарға алабұға популяциясы 12, 000 дананы құрайтын балық аулау шаруашылығын басқарамыз. Балық аулау

болмаған жағдайда алабұға популяциясы, туылу мен өлім көрсеткішін ескере отырып, күтілгендей жылына 8% көбейіп отырады.

Жылына балық аулауға рұқсат етілген көлем 1,500 дана алабұғаны құрайды. Балық аулау орны балықшылар арасында жақсы мәлім болғандықтан мұндай жылдық балық аулау деңгейіне оңайлықпен қол жеткізуге болады. Біздің жылдық балық аулауымыз ұзақ уақыттық көлемде экологиялық қауіпсіз болып табыла ма? Егер ондай болмаса, популяцияның толығымен жойылуы үшін қанша уақыт өтеді? Біз жылдық балық аулау деңгейін қысқартуымыз қажет па? Егер мұндай болса, балық аулау деңгейі қаншаға дейін төмендетілуі тиіс?

Біз берілген жыл үшін популяция болжамалы санын таба аламыз, бұл үшін популяцияның бастапқы санынан бастап, алдыңғы жылы ауланған балық санының негізінде әр келесі жыл үшін популяция санын есептей аламыз. Мысалы, біз төрт жылдан кейінгі болжамалы популяция санын білгіміз келеді деп болжамдайық. Population0 ауыспалы атауы берілген үлкен ауызды алабұға бастапқы популяциясынан бастайық:

$$\text{population0} = 12000$$

Енді біз бірінші жыл соңына популяция көлемін орнатқымыз келеді, яғни бастапқы мәннен популяция санының мәні 8% және минус 1,500 дананы құрайды. Басқаша айтқанда, егер population1 ретінде бірінші жыл соңында популяция санын алсақ, онда

$$\text{population1} = \text{population0} + 0.08 * \text{population0} - 1500 \quad \# 11,460$$

немесе, дәл сол сияқты,

$$\text{population1} = 1.08 * \text{population0} - 1500 \quad \# 11,460.0$$

Екінші жыл соңындағы популяция көлемі дәл біріншідегідей population1 мәніне негізделіп отырып есептелінеді:

$$\text{population2} = 1.08 * \text{population1} - 1500 \quad \# 10,876.8$$

Жалғастыра келе,

$$\begin{array}{llllll} \text{population3} = & 1.08 & * & \text{population2} & - 1500 & \# & 10,246.94 \\ \text{population4} = & 1.08 & * & \text{population3} & - 1500 & \# & 9,566.6952 \end{array}$$

Осылайша, бұл үлгі келешек төрт жылғы популяция көлемін 9,566 деп болжамдайды (ондық нүктенің оң жағында сандар түрінде берілген «балықтың пропорциялық бөліктеріне» назар аудармағанда).

Біз қазір ғана аяқтаған рәсім толығымен қайталандалық (немесе итерациялық) мінезде болды, сондықтан for кезеңін пайдалану идеясы ойға келді. Біз сегіз жапырақшасы бар геометриялық гүл шоғыры суретін салу үшін 3.2 Бөлімінде келесі for кезеңін қолданғанымызды еске сала өтейік:

```
for segment in range(8):  
    tortoise.forward(length)  
    tortoise.left(135)
```

For және in басты көздер – синтаксистің қажетті элементтері, ал қызыл түспен белгіленген көздер – бағдарлаушы қажеттігіне қарай. Бұл жағдайдағы segment ауыспалы атауы *индексті ауыспалы* деп аталады. In артынан ілесетін сегмент әр итерацияға бір мәннен беріп отыратын индексті ауыспалылар мәндерінің тізімінен тұрады.

Range(8) 0ден 7 дейінгі толық сандар тізіміне сәйкес келетіндіктен, бұл кезең нұсқаманың келесі реттілігіне баламалы:

```
segment = 0  
tortoise.forward(length)  
tortoise.left(135)  
segment = 1  
tortoise.forward(length)  
tortoise.left(135)  
segment = 2  
tortoise.forward(length)  
tortoise.left(135)  
:  
segment = 7  
tortoise.forward(length)  
tortoise.left(135)
```

Сегіз түрлі мәндер segment-пен берілгендіктен сурет салудың екі нұсқамасы кезеңі денесінде сегіз рет атқарылады.

Біздің балық аулау шаруашылығы есебімізде төртінші жыл соңындағы популяция көлемін есептеу үшін біз есептеудің төрт итерациясын жасадық, яғни:

```

population0 = 12000
population1 = 1.08 * population0 - 1500    # 11460.0
population2 = 1.08 * population1 - 1500    # 10876.8
population3 = 1.08 * population2 - 1500    # 10246.94
population4 = 1.08 * population3 - 1500    # 9566.6952

```

Яғни бізде итерацияны төрт рет өткізетін for кезең қажет:
for year in range(4):

алдыңғы мән негізінде популяция көлемін есептеу

Әр кезең итерациясында біз алдыңғы жылғы популяция көлемін негізге ала отырып, ағымдағы жылдың популяция көлемін есептеп шығарғымыз келеді. Бірінші итерацияда біз population1-ге берген мәнді есептеп шығарғымыз келеді. Екінші, үшінші және төртінші итерацияларда біз population2, population3 мен population4-ке берген мәндерді есептеп шығарғымыз келеді.

Бірақ осы төрт рет қайталауды қажет ететін төрт нұскаманы қалайша бір

нұсқамаға жүктеуге болады? Ол үшін соңғы жылғы популяцияны есептеу үшін қолданылған ауыспалы бұдан әрі қолданбайтындығын уақыты байқау қажет. Осылайша, шын мәнінде бізде бұл барлық ауыспалыларды қолдануда қажеттілік жоқ. Осы барлық әрекеттердің орнына біз сумматор ауыспалысы (немесе жай ғана сумматор) деп аталатын бір ғана population ауыспалысын қолданып, оны 1.08 көбейтіп, сосын әр жыл үшін 1500 алып тастау қажет. Біз population үшін бастапқы мән береміз = 12000, сосын әр келесі жыл үшін

$population = 1.08 * population - 1500$

мәндерін беріп отырамыз. Үлестіру нұскамасы ең алдымен оң жақты есептейтіндігін есте сақтаңыз.

Сондықтан, тең белгісінің оң жағында орналасқан population мәні алдыңғы жылғы популяция көлемін білдереді, бұл мән нәтижесіне сол жақтағы population мәні берледі және ол ағымдағы жыл популяция көлемін есептеу үшін қолданылады.

```

population=12000
for year in
range(4):
    population = 1.08 * population - 1500

```

For кезеңі үшін бұл келесі нұсқамалармен баламалы:

```

population = 12000

year = 0
population = 1.08 * population - 1500    # new population = 11460.0
                        12000

year = 1
population = 1.08 * population - 1500    # new population = 10876.8
                        11460.0

year = 2
population = 1.08 * population - 1500    # new population = 10246.94
                        10876.8

year = 3
population = 1.08 * population - 1500    # new population = 9566.6952
                        10246.94

```

For кезеңінің бірінші итерациясында year 0 мәнін береді, ал population – 1.08 көбейтіп, минус 1500 алдыңғы population (12000) мәнін береді, яғни шешім 11460.0 тең болады. Осыдан кейін, екінші итерацияда year 1 мәнін береді, ал population тағы да 1.08 көбейтіп, минус 1500 алдыңғы population (12000) мәнін береді, яғни шешім 10876.8 тең болады. Бұл for кезеңі аяқталғанша тағы да екі рет жалғаса береді. Population қорытынды мәні, біз осыған дейін есептегендей – 9566.6952 тең болады.

4.1 Ойлануға арналған сұрақтар Жоғарыда берілген нұсқамаларды for кезең денесіне population үлестіру нұсқамасынан кейін енгізіңіз:

```
print(year + 1, int(population))
```

Бағдарламаны іске қосыңыз. Экранда не пайда болды? Неге пайда болғанын түсінесіз бе?

Осы мысалдан көріп отырғандай, year индексті ауыспалысын кез келген басқа ауыспалы ретінде қолдануға болады. Year нөлден басталатындықтан, ал кезеңнің бірінші итерациясында 1 жылдық популяция көлемі есептелетіндіктен, біз экранға year орнына year + 1

шығарамыз.

4.2 Ойлануға арналған сұрақтар *Бес жылдан кейін балық популяциясын есептеу үшін сіз бұл кезеңді қалай өзгерттіңіз? Ал 10 жылдан кейін?*

Есептілік жыл көлемін өзгерту оңай. Бізге керегі бар болғаны - range мәнін қалауымызша өзгерту керек: (5), range(10) және т.б. Егер біз бұл есептеуді функцияға енгізсек, онда баптау ретінде келесі қажетті жылдар санын орнату қажет:

```
def pond(years):
    """Simulates a fish population in a fishing pond, and
    prints annual population size. The population
    grows 8% per year with an annual harvest of 1500.

    Parameter:
        years: number of years to simulate

    Return value: the final population size
    """

    population = 12000
    for year in range(years):
        population = 1.08 * population - 1500
        print(year + 1, int(population))

    return population

def main():
    finalPopulation = pond(10)
    print('The final population is', finalPopulation)

main()
```

Ойлануға берілетін сұрақ 4.3 Егер population = 12000 цикл денесінің алдына емес, ал ішіне қойсақ не болар еді? Егер population = 12000 нұсқаулығын мүлде алып тастаса не болар еді?

Циклдің алдындағы сумматор айнымалылардың инициализациялары негізгі мәнге ие. Егер population үшін циклдің алдында бастапқы шама берілмегенде, онда for циклінің бірінші итерациясында қателік болар еді,

себебі меншіктеу нұсқаулығының оң бөлігі қисынсыз болып қалар еді!

Ойлануға берілетін сұрақ 4.4 Бастапқы сұрақтарға жауап беру үшін *pond* функциясын пайдаланыңыз: Максималды жылдық балық аулау ұзақ мерзімге созылса экологияға зиян келтіре ме? Егер зиян келтірмесе, онда популяцияның толық жойылып кетуіне қанша уақыт кетеді? Балық шаруашылығының басқарушысы максималды өндірісті азайту қажет пе? Егер азайту керек болса, онда ол қандай деңгейге дейін азайтылуы тиіс?

Бұл функцияны көп жылдық шамамен шақыру, балық популяциясының 14 жылы нөлден төмендейтіндігін көрсетеді (әлбетте, шындығына келгенде, бұл мүмкін емес):

```
1 11460
2 10876
3 10246
:
13 392
14 -1076
:
```

Әлбетте мұндай балық аулау жоспары экологияға зиян болып табылады, сондықтан балық шаруашылығының басшысы оны оңтайлы деңгейге дейін қысқарту қажет. Бұл жағдайда экологиялық қауіпсіз деңгейді анықтау оңай: популяция жылына 8% өсетіндіктен, ал тоғанда 12,000 балық түрі бар, біз балық аулау көлемінің $0.08 \% 12000 = 960$ -тан асып кетіп, популяцияның үздіксіз азаюына жол бере алмаймыз.

Ойлануға берілетін сұрақ 4.5 *Екі қосымша параметрлер арқылы *pond* функциясын жалпылаңыз: популяцияның бастапқы саны және жылдық өндіріс. Өзгертілген функцияңызды пайдаланыңыз, егер біз жылдық өндірісті 800 балық түріне дейін өзгертетін болсақ, 15 жылдан кейін тоғанда болатын балық санын есептеңіз.*

Осы өзгерістерді ескергенде, сіздің функцияңыз шамамен төмендегідей болмақ:

```
def pond(years, initialPopulation,
        harvest): """ (құжаттау жолдары
        түсірілген) """

    population =
    initialPopulation for year
    in range(years):
        population = 1.08 * population - harvest
        print(year + 1, int(population))

    return population
```

initialPopulation параметр мәні біздің жоғарыда айтылған популяцияның– 12000 орнында тұр, ал harvest атты параметрі біздің алғашқы өндірісіміздің 1500 орнында. Жоғарыда берілген сұраққа жауап беру үшін, main-нен pond функциясын шақырудың орнына мынаны шақыра аламыз:

```
finalPopulation = pond(15, 12000, 800)
```

экранның мына қорытындылар шығады:

```
1   12160
2   12332
3   12519
4   12720
:
13  15439
14  15874
15  16344
```

Популяцияның ақырғы саны – 16344.338228396558

Ойлануға берілетін сұрақ 4.6 Жылдық өндірісі 1500 көрсеткішпен бастапқы мәнді қайталау үшін pond функциясының жаңа нұсқасын қалай шақыратын едіңіз?

Әрі қарай бармастан бұрын, алдымен пішімдеу жолы деп аталатын пайдалы Python аяласын қарастырып көрелік, ол жыл сайынғы популяциялар кестесін көркемдеуге көмектеседі. Пішімдеу жолының пайдаланылуын суреттеу үшін, әуелгі функцияның келесі модификацияланған түрін қарастырып көрейік.

```
def pond(years, initialPopulation,
harvest): """ (құжаттау жолдары
түсірілген) """

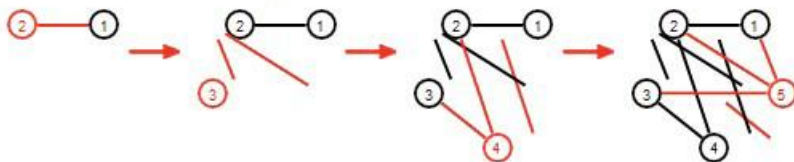
population =
initialPopulation print('
Популяция'|ЖЫЛЫ)
print('-----|-----')
for year in range(years):
    population = 1.08 * population - harvest
    print('{0:^4} | {1:>9.2f}'.format(year + 1,
population)) return population
```

Функция бағаналарды белгілеу үшін экранға кестенің тақырыбын шығара бастайды. Содан кейін, for циклінің ішінде print функциясын шақырған кезде, әрбір жолда екі мән ұйымдастыру үшін біз пішіндеу жолын қолданамыз. Пішіндеу жолының синтаксисі келесі

'<ауыстырылатын өрістер>'.format(<пішіндеуге арналған мәндер>)

(Жоғарыда қызылмен белгіленген сегменттер синтаксис бөлігі емес, сипаттаушы болып табылады.) Жол мен format арасындағы нүкте format жолдар қатарының әдісі болып табылатындығын көрсетеді; біз жолдар қатары жөнінде 6 тарауда толығырақ айтамыз. format әдісінің параметрлері пішінделетін мәндер болып табылады. Әрбір мәнге арналған пішін ауыстырылатын өрісте пішін жолында фигуралық жақша ({})ішінде бекітіліп көрсетіледі.

Жоғарыда берілген for цикліне арналған мысалда, ауыстырылатын өрістегі {0: ^ 4} біріншісі (негізінде «нөлдік»; программистерге санақты 0-ден бастаған ұнамдырақ) format-тағы аргумент, бұл жағдайда year + 1, өрісте ені 4 болып ортасында (^) орналасуы қажет. Ауыстырылатын өрістегі {1:> 9.2f} format –тың екінші аргументі ретінде population ондық нүктенің (.2f) оң жағында екі белгілі жылжымалы нүктесі бар сан секілді, ені 9-ға тең болып өрістің оң жағында (>) туралануы қажет. Жылжымалы нүктелі сандарды пішіндеу кезінде (f қойылады), ауыстырылатын өрістегі ондық нүктеге дейінгі сан ондық нүктені қоса минималды ені болып табылады. Ауыстырылатын өрістегі ондық нүктеден кейінгі сан дегеніміз – сандағы ондық нүктенің оң жағындағы цифрлердің мини-



Әр қадамда қызыл торап желіге қосылады. Бұл кезеңдердегі қызыл сызықтар жаңа мүшені қосу нәтижесінде пайда болуы ықтимал жаңа байланыстардың барлығы болып табылады.

Ойлануға берілетін сұрақ 4.7 2, 3, 4 және 5 тораптарының әрбірін қосқандағы пайда болуы ықтимал жаңа байланыстардың максималды саны қанша? Жалпы жағдай үшін, номер n торабын қосқан кезде пайда болуы ықтимал жаңа байланыстардың максималды саны қанша?

Торап 2 ең көп дегенде 1 жаңа байланысты, торап 3 ең көп дегенде 2 жаңа байланысты, торап 4 – 3 жаңа байланысты қосады және т.с.с. Жалпы алғанда, максимум $n - 1$ жаңа байланыс номер n торабын қосқанда пайда болуы мүмкін. Бұл үлгі төмендегі кестеде келтірілген.

Торап нөмірі:	2	3	4	5	...	n
Байланыс санының максималды өсуі:	1	2	3	4	...	$n - 1$

Осылайша, соңғы жолда көрсетілгендей, n торапты желідегі байланыстардың максималды саны– екінші қатардағы сандар сомасы:

$$1 + 2 + 3 + \dots + n - 1.$$

Біз бұл соманы желінің әлеуметті құндылығын беру үшін қолданамыз.

Желіге жаңа тораптардың қосылуына қарай байланыстардың максималды санын және байланыстардың жалпы максималды санын есептейтін алдыңғы функцияға ұқсас функция жазайық.

Өсу және кему

Бұл жағдайда бізге байланыстардың жалпы санын есептеу үшін сумматор қажет болады. Біздің pond функциямыздың бұл жаңа тапсырмаға бейімделуі келесіні береді:

```

def countLinks(totalNodes):
    totalNodes тораптары арасында жұптық
    байланыстардың жалпы максималды саны
    берілген кестені шығарады.

    Параметр:
        totalNodes: желідегі тораптардың жалпы саны

    Қайтарылатын мәндер:
        totalNodes тораптарынан желідегі байланыстардың
        максималды саны
    """

    totalLinks = 0
    for node in
        range(totalNodes):
            newLinks = ???
            totalLinks = totalLinks +
            newLinks print(node, newLinks,
            totalLinks) print(node, newLinks,
            totalLinks)

    return totalLinks

```

Бұл функцияда бізге сумматордың тұрақсыз шамасы байланыстардың жалпы санын есептеуі қажет, сондықтан біз оның атауын `population` – нен `totalLinks` –ке өзгертіп, нөлмен белгіледік. Сонымен қатар біз итерациялар санын анықтайтын параметрдің атын өзгертіп, `years` орнына `totalNodes` дедік, ал сондай-ақ циклдің `for` индекстік ауыспалы шамасын `year` –ден `node` деп өзгерттік, себебі ол енді біз әр қадамда қосатын торап санын есептеп отырады. Циклдің `for` денесінде біз сумматорға ағымдағы тораппен (бұған жуырда қайта ораламыз) желіге қосылатын жаңа байланыстардың максималды санын қосамыз, содан кейін экранға торап нөмірі, жаңа байланыстардың максималды саны және қазіргі уақыттағы желіде бар байланыстардың максималды саны көрсетілген жолды шығарамыз.

`newLinks` мәнінің қандай болуы керектігін анықтаудан бұрын, біз бір мәселені шешуіміз қажет. Жоғарыда берілген кестеде тораптар нөмірлері 2-ден бастап желідегі торап санына дейінгі диапазонда жатыр, алайда біздің `for` циклінде, `node` 0-ден бастап `totalNodes – 1` дейінгі диа-

пазонда жатыр. Бұл мәнді түзету оңай екен, себебі range функциясы біз осы уақытқа дейін көріп келген шамаға қарағанда күрделірек келетін көптеген сандық диапазондар түрлендіре алады. Егер біз range-ке бір аргументтің орнына екеуін беретін болсақ, мысалы, range(start, stop), бірінші аргумент бастапқы мән, ал екіншісі – ақырғы мән болып интерпретацияланады да, start –пен басталып, stop-пен аяқталатын, алайда оны қоспайтын мәндер диапазоны жасалады. Мысалы, range(-5, 10) –5, –4, –3, . . . , 8, 9 секілді бүтін сандар жасайды.

Бұған өзіміз көз жеткізу үшін, бағдарламадағы Python қабығында range(-5, 10) теріңіз(немесе бағдарламада print орындаңыз).

```
>>> range(-5, 10)
range(-5, 10)
```

Өкінішке орай, сіз пайдасы жоқ нәтиже аласыз, алайда оны оңай жолмен түзетуге болады, ол үшін диапазонды нөмірлердің тізіміне түрлендіру керек. Шаршы жақшада ([]) бекітіліген тізім – біз келесі тарауларда кеңінен қолданатын мәліметтердің тағы бір абстрактілі түрі. Диапазонды тізімге түрлендіру үшін, біз оны list функциясының аргументі ретінде бере аламыз:

```
>>> list(range(-5, 10))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ойлануға берілетін сұрақ 4.8 *Қандай нөмірлер тізімі range(1, 10) жасайды?*

range(10, 1) туралы не дейсіз? Әрбір жағдай үшін берілген нәтижені түсіндіре аласыз ба?

Ойлануға берілетін сұрақ 4.9 *Бағдарламамызға оралайық, for цикліміздің түрі қандай болғандығын қалаймыз?*

2-ден басталып, totalNodes аяқталатын node үшін, for цикліміз келесідей болуы қажет

```
for node in range(2, totalNodes + 1):
```

Ал енді бағдарламамыздағы newLinks мәні қандай болуы тиіс? Бұған жауап жоғарыда біз келтірген кестенің ішінде; n нөмірлі торап

арқылы желіге қосылатын жаңа байланыстардың максималды саны $n - 1$ құрайды. Циклімізде торап нөмірі `node` атына беріледі, сондықтан бізге әрбір қадамда $node - 1$ байланыстарын қосып отыру керек:

`newLinks = node - 1`

Көрсетілген өзгерістерді ескерсек, функциямыздың түрі келесідей болмақ:

```
def countLinks(totalNodes):
    """ (құжаттау жолдары түсірілген) """

    totalLinks = 0
    for node in range(2, totalNodes +
        1): newLinks = node - 1
        totalLinks = totalLinks +
            newLinks print(node, newLinks,
                totalLinks)

    return

totalLinks def

main():
    links = countLinks(10)
    print('Байланыстардың жалпы саны

- құрайды', links) main()
```

Алдыңғы `for` цикліміздегі жағдай тәрізді, бұл циклдің не істейтіндігін біз нұсқаулардың баламалы бірізділігіне қарап жақсырақ түсіне аламыз. Өзгертілетін `node` мәні қызылмен белгіленген.


```
totalLinks = 0
```

```
node = 2
```

```
newLinks = node - 1          # newLinks 2 - 1 = 1 totalLinks =  
totalLinks + newLinks мәні беріледі # totalLinks 0 + 1 = 1  
print(node, newLinks, totalLinks)мәні беріледі# 2 1 1 экранға  
шығарады
```

```
node = 3
```

```
newLinks = node - 1          # newLinks 3 - 1 = 2 totalLinks  
= totalLinks + newLinks мәні беріледі # 1 + 2 = 3 print(node,  
newLinks, totalLinks) мәні беріледі # 3 2 3 экранға шығарады
```

```
node = 4
```

```
newLinks = node - 1          # newLinks 4 - 1 = 3 totalLinks  
= totalLinks + newLinks мәні беріледі # totalLinks 3 + 3 = 6  
print(node, newLinks, totalLinks)мәні беріледі# 4 3 6 экранға  
шығарады
```

```
:
```

```
node = 10
```

```
newLinks = node - 1          # newLinks 10 - 1 = 9 totalLinks =  
totalLinks + newLinks мәні беріледі # totalLinks 36 + 9 = 45  
print(node, newLinks, totalLinks) мәні беріледі # 10 9 45  
экранға шығарады
```

Жоғарда main функциясынан countLinks(10) шақырғанда экранға келесі шығады

```
2 1 1  
3 2 3  
4 3 6  
5 4 1  
6 5 1  
7 6 2  
8 7 2  
9 8 3  
10 9 45
```

Байланыстардың жалпы саны 45 құрайды

Түзу бағаналарды пішіндеу жолын қолдану арқылы жасауды жаттығу ретінде қалдырамыз.

Ойлануға берілетін сұрақ 4.10 *countLinks(100) не қайтарады? Ол не білдіреді?*

Концерт ұйымдастыру

Тағы бір мысал қарастырып көрелік. Мәселен, Сіз концерт ұйымдастырудасыз делік, сізге билет құнының қандай болатындығын анықтау керек. Топ пен іс-шара өткізетін алаңға кететін қаражаттың жалпы сомасы \$ 8000 құрайды. Алаңға ең көп дегенде 2,000 адам сияды, және Сіз нарықтық зерттеулер арқылы сіз сатуға ұйғаратын билеттердің саны билеттің құнына келесі ара қатынасқа байланысты екенін анықтадыңыз:

$$\text{sales} = 2500 - 80 * \text{price}$$

Бұл ара- қатынасқа сәйкес, сіз билеттерді тегін таратасыз, ал алаңға адам сыймай қалады. Екінші жағынан, егер сіз тым жоғары баға белгілесеніз, онда бір де бір билет сата алмайсыз. Сіз билет бағасының максималды пайда түсіретіндей орташа деңгейде белгіленуін қалайсыз. Билет сатылымынан келетін жалпы түсім $\text{sales} * \text{price}$ тең болады, сондықтан сіздің түсіміңіз \$ 8000 шегергенде осы соманы құрайды.

Билеттің ең тиімді бағасын анықтау үшін алдыңғы екі есепке ұқсас for Циклін қолданып кесте құрай аламыз. Бұл жағдайда билетер бағасының диапазонын сұрыптап, әрбір таңдау нәтижесіндегі түсімді баспаға шығаруды қалаймыз. Келесі функцияда, for циклі бір долларлық билет бағасынан басталады және әрбір итерацияда доллардың maxPrice –қа жеткенше бағаға бір доллардан тағы қосады.

```
def profitTable(maxPrice):
    """Билет бағасына қарай концерттен түсетін пайда
    кестесін экранға шығарады.
```

Параметрлер:

maxPrice: қарастырылатын максималды баға

Қайтарылатын

мәндер: None"""

```
print(Баға    Түсім    Пайда)
```

```
print('----- -----')
print('')
```

```
for price in range(1, maxPrice +
```

```
1): sales = 2500 - 80 * price
```

```
income = sales * price
```

```
profit = income - 8000
```

```
formatString = '${0:>5.2f}
```

```
                ${1:>8.2f}
```

```
                ${2:8.2f}'
```

```
print(formatString.format(price, income,
profit))
```

```
def main():
```

```
    profitTable(25)
```

```
main()
```

Ықтимал сатылымдар саны әрбір итерацияда жоғарыда келтірілген ара қатынасқа сәйкес price индекстік тұрақсыз шамасынан есептеліп шығады. Содан кейін біз экранға пішіндеу жолымен тиянақты етіліп пішінделген бағаны, есептік кіріс пен түсімді шығарамыз. Бұрын жасағанымыздай, әрбір цикл интерациясында не болатындыған көрелік:

price = 1

sales = 2500 - 80 * price # sales берілетін мән - 2500 - 80 *
1 = 2420 income = sales * price # income берілетін мән -
2420 * 1 = 2420 profit = income - 8000 # profit берілетін
мән - 2420 - 8000 = -5580 print(price, income, profit) #
экранға шығаратыны - \$ 1.00 \$ 2420.00 \$-5580.00

price = 2

sales = 2500 - 80 * price # sales берілетін мән - 2500 - 80 *
2 = 2340 income = sales * price # income берілетін мән -
2340 * 2 = 4680 profit = income - 8000 # profit берілетін
мән - 4680 - 8000 = -3320 print(price, income, profit) #
экранға шығаратыны - \$ 2.00 \$ 4680.00 \$-3320.00

price = 3

sales = 2500 - 80 * price # sales берілетін мән - 2500 - 80 *
3 = 2260 income = sales * price # income берілетін мән -
2260 * 3 = 6780 profit = income - 8000 # profit берілетін
мән - 6780 - 8000 = -1220 print(price, income, profit) #
Экранға шығаратыны - \$ 3.00 \$ 6780.00 \$-1220.00

:

Ойлануға берілетін сұрақ 4.11 Бағдарламаны қосып, билеттің ең тиімді бағасын анықтаңыз.

Бағдарлама экранға келесі кестені шығарады:

Баға	Түсім	Пайда
\$ 1.00	\$ 2420.00	\$-5580.00
\$ 2.00	\$ 4680.00	\$-3320.00
\$ 3.00	\$ 6780.00	\$-1220.00
\$ 4.00	\$ 8720.00	\$ 720.00
:		
\$15.00	\$19500.00	\$11500.00
\$16.00	\$19520.00	\$11520.00
\$17.00	\$19380.00	\$11380.00
:		
\$24.00	\$13920.00	\$ 5920.00
\$25.00	\$12500.00	\$ 4500.00

Пайда үшінші бағанада билет бағасы \$ 16 болса, \$ 11,520.00 жеткенше артады, содан кейін кемиді. Сонымен, билеттің ең тиімді бағасы \$ 16 секілді.

Ойлануға берілетін сұрақ 4.12 Біздің бағдарлама бүтін долларға есепті билеттерді ғана қарастырады. Оның орнына әрбір итерацияда билет бағасының өсімі елу центті құрайтындай етіп оны қалай өзгертуге болады?

range функциясы тек бүтін сандардың диапазонын жасай алатындықтан, біз өсімді 1-ден 0,5 –ке ауыстыру үшін range функциясын бере алмаймыз. Алайда біз іріктеуге берілген сандар диапазонын екі еселеп, содан кейін әр итерацияда индекстік тұрақсыз шамасының екіге бөлінген көлемінде баға белгілеу арқылы өз мақсатымызға жете аламыз.

```
def profitTable(maxPrice):
    """ (құжаттау жолдары түсірілген) """

    print(Баға    Түсім    Пайда)
    print('-----')
    for price in range(1, 2 * maxPrice
        + 1):
        realPrice = price / 2
        sales = 2500 - 80 *
            realPrice
        income = sales *
            realPrice
        profit = income -
            8000
        formatString = '$ {0:>5.2f}          $ {1:>8.2f}
                        $ {2:8.2f}'
        print(formatString.format(realPrice, income,
            profit))
```

Енді, price 1-ге тең болғанда, пайданы есептеу үшін пайдаланылатын «шынайы баға» (realPrice), 0.5 құрайды. Ал price 2-ге тең болғанда, «шынайы баға» 1-ді құрайды және т.с.с.

Ойлануға берілетін сұрақ 4.13 Біздің функция билет үшін \$ 16 тиімдірек бағаны таба ала ма?

Біздің жаңа функция экранға келесі кестені шығарады

Баға	Түсім	Пайда
\$ 0.50	\$ 1230.00	\$-6770.00
\$ 1.00	\$ 2420.00	\$-5580.00
\$ 1.50	\$ 3570.00	\$-4430.00
\$ 2.00	\$ 4680.00	\$-3320.00
:		
\$15.50	\$19530.00	\$11530.00
\$16.00	\$19520.00	\$11520.00
\$16.50	\$19470.00	\$11470.00
:		
\$24.50	\$13230.00	\$ 5230.00
\$25.00	\$12500.00	\$ 4500.00

Егер біз \$ 16 жанындағы билет бағасына қарайтын болсақ, онда \$ 15.50 шын мәнінде \$ 10 көбірек пайда әкелетінін көреміз.

Кестеге қараған кезде билет бағасы мен пайданың ара- қатынасы графикте берілетін мәліметтер секілді онша түсінікті болып көрінбейді. Мысалы, пайда максимумге дейін сызықты түрде өсіп, содан кейін дәл солай кеме ме? Әлде ол біртіндеп түзеле ме? Біз бұл сұрақтарға тасбақалы кескіндеу арқылы, тасбақаны бір нүктеден екіншіге жылжыту үшін goto әдісін қолдана отырып, тәуелділікті суреттеп жауап бере аламыз.

```
import turtle

def profitPlot(tortoise,
               maxPrice): """ (құжаттау
                               жолдары түсірілген) """

    for price in range(1, 2 * maxPrice
                       + 1): realPrice = price / 2
                               sales = 2500 - 80 * realPrice
                               income = sales * realPrice
                               profit = income - 8000
                               tortoise.goto(realPrice,
                                              profit)

def main():
    george = turtle.Turtle()
    screen = george.getscreen()
    screen.setworldcoordinates(0, -15000, 25, 15000)
    profitPlot(george, 25)
    screen.exitonclick()

main()
```

Біздің жаңа main функциямыз тасбақа береді, содан кейін setworldcoordinates функциясын пайдаланып, біз көрсетуіміз ықтимал нүктелерді орналастыру үшін график терезесіндегі координаттар жүйесін өзгертеді. setworldcoordinates алғашқы екі аргументті терезенің төменгі сол жақ бұрышының координаталарын белгілейді, біздің жағдайда (0, -15, 000). Осылайша, терезеде ең минималды көрінетін мән x (баға) нөлге тең болады, ал минималды көрінетін мән y (пайда) -15,000 болады. Екінші екі аргумент терезенің жоғарғы оң жақ бұрышының координаталарын белгілейді, біздің жағдайда (25, 15,000). Осылайша, терезедегі максималды көрінетін мән x (баға) 25-ке тең болады, ал максималды көрінетін мән y (пайда) 15,000 болады. for циклінде profitPlot функциясында realPrice бірінші мәні 0.5 құрайды, сондықтан бірінші goto – бұл

```
george.goto(0.5, -6770)
```

ол бастапқы (0, 0) нүктесінен (0.5, -6770) нүктесіне сызық салады. Келесі итерацияда, realPrice мәні 1.0 болады, сәйкесінше, цикл келесіде мынаны орындайды

```
george.goto(1.0, -5580)
```

ол бастапқы позициядағы (0.5, -6770) нүктеден (1.0, -5580) нүктесіне сызық салады. realPrice келесі мәні 1.5, цикл мынаны орындайды

```
george.goto(1.5, -4430)
```

ол (1.0, -5580) нүктесінен (1.5, -4430) нүктесіне сызық салады. Тағы солай сияқты, realPrice өзінің ақырғы 25 мәніне жеткенше болады, ал біз бастапқы позициядағы (24.5, 5230) нүктесінен (25, 4500) нүктесіне сызық салмаймыз.

Ойлануға берілетін сұрақ 4.14 *Графиктің пішіні қандай? Неліктен екенін түсінесіздер ме?*

Ойлануға берілетін сұрақ 4.15 *Бұл график жасайтын бағдарламаны іске қосқанда сіз координаталардан бастап графиктің бірінші нүктесіне дейінгі түрі нашар сызықты байқайсыз. Оны қалай жоя аласыз? (Жауапты жаттығу ретінде қалдырамыз.)*

Жаттығулар

Келесі тапсырмалардың әрқайсысына функция жазыңыз. Егер мүмкін болса, функция талап етілетін мәнді қайтаратындығын (оны экранға жай шығармай) тексеріңіз. Функцияларыңызды міндетті түрде сәйкес шолулармен және құжаттау жолдарымен құжаттаңыз.

4.1.1. Функцияны жазыңыз

```
print100()
```

ол, for циклін қолдана отырып, экранға 0-ден 100-ге дейінгі бүтін сандарды шығарады.

4.1.2. Функцияны жазыңыз

```
triangle1()
```

ол, for циклін қолдана отырып, экранға келесіні шығарады:

```
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
```

4.1.1. Функцияны жазыңыз

```
square(letter, width)
```

ол экранға letter әріптерінің ішінен жолды қолданып, width жағы белгіленген шаршы шығарады. Мысалы,

square('Q', 5) экранға келесіні шығаруы керек:

```
QQQQQ
QQQQQ
QQQQQ
QQQQQ
QQQQQ
```

4.1.3. Экранға -50-ден 50-ге дейінгі диапазондағы бүтін сандарды шығаратын for циклін жазыңыз.

4.1.4. Экранға 1-ден 100-ге дейінгі диапазондағы тақ бүтін сандарды шығаратын for циклін жазыңыз. (Сыбырсөз: range(50))

қолданыңыз.)

4.1.5. 43-бетте біз модульді арифметиканы пайдалану арқылы цифрлық сағаттардағы минуттардың тықылын қалай келтіруге болатыны жөнінде талқылаған болатынбыз. Функцияны жазыңыз

`clock(ticks)`

Ол баспаға сағаттың `ticks`, түнгі он екіден басталатын, бір минуттық кадаммен жүретін тықылын келтіруі керек. Оңайлату үшін түнгі 12 орнына 0 деп белгілеуге болады. Мысалы, `clock(100)` экранға мынаны шығаруы керек

0:00

0:01

0:02

:

0:59

1:00

1:01

:

1:38

1:39

Сағаттағы бағаналарды түзету үшін және бірінші позициядағы нөлдерді минуттарға қою үшін келесі пішіндейтін жолды пайдаланыңыз:

`print('{0:>2}:{1:0>2}'.format(hours, minutes))`

4.1.6. Негізінде `range` функциясының үш формасы бар:

- 1 параметр: `range(stop)`
- 2 параметр: `range(start, stop)`
- 3 параметр: `range(start, stop, skip)`

Үш аргументпен `range start` мәнінен басталатын және `stop – 1` мәнінде аяқталатын немесе оған дейінгі бүтін сандардың диапазонын шығарады, әр кезде `skip` қосады. Мысалы,

`range(5, 15, 2)`

5, 7, 9, 11, 13 сандар диапазонын береді

`range(-5, -15, -2)`

-5, -7, -9, -11, -13 сандар диапазонын береді. Бұл сандарды жолға бір-бірден шығару үшін `for` циклін қолдануға болады:

`for number in range(-5, -15, -2): print(number)`

(a) Экранға 2-ден 100-ге дейінгі бүтін жұп сандарды шығаратын `for` циклін жазыңыз, `range` функциясының үшінші формасын қолданыңыз.

(b) Экранға 1-ден 100-ге дейінгі бүтін тақ сандарды шығаратын `for`

циклін жазыңыз, range функциясының үшінші формасын қолданыңыз.

(с) Экранға 1-ден 100-ге дейінгі бүтін сандарды кему тәртібінде шығаратын for циклін жазыңыз.

(d) Экранға 7, 11, 15, 19 мәндерін шығаратын for циклін жазыңыз.

(е) Экранға 2, 1, 0, -1, -2 мәндерін шығаратын for циклін жазыңыз.

(f) Экранға -7, -11, -15, -19 мәндерін шығаратын for циклін жазыңыз

4.1.7. Функцияны жазыңыз

multiples(n)

ол экранға 0-ден 100-ге дейінгі диапазондағы барлық еселі n сандарын шығаруы керек. Мысалы, егер n 4-ке тең болса, функция экранға 0, 4, 8, 12, ... мәндерін шығаратын еді.

4.1.8. Функцияны жазыңыз

countdown(n)

ол экранға 0-ден n –ге дейінгі сандарды кему тәртібінде шығаруы керек. Мысалы, егер n 5- ке

тең болғанда, онда функция экранға 5, 4, 3, 2, 1, 0 мәндерін шығаратын еді.

4.1.9. Функцияны жазыңыз

triangle2()

ол for циклін қолданып экранға келесіні шығарады:

```
*****
****
***
**
*
```

4.1.10. Функцияны жазыңыз

triangle3()

ол for циклін қолданып экранға келесіні шығарады:

```
*****
****
***
**
*

```

4.1.11. Функцияны жазыңыз

`diamond()`

ол for циклін қолданып экранға келесіні шығарады:

```
*****
*****
****
***
**
*
*
**
***
****
*****
```

4.1.12. Функцияны жазыңыз

`circles(tortoise)`

ол тасбақа графикасын және for циклін қолданып, радиусы 10, 20, 30, . . . , 100-ге тең шоғырлас шеңберлер салады. (Шеңбер салу үшін Сіз circle тасбақа графикасын немесе 3.3.8 жаттығуда жазылған DrawCircle функциясын қолдана аласыз.)

4.1.13. Мәтінде profitPlot функциясында 4.15 ойлануға арналған

сұрағында айтылған қатені түзеңіз.

4.1.14. Функцияны жазыңыз

`plotSine(tortoise, n)`

ол тасбақа графикасын $x = 0$ -ден градусқа $x = n$ дейінгі аргументтер диапазоны үшін $\sin x$ салу үшін қолданады. X терезесінің координаталарын 0-ден 1080-ге дейінгі диапазонда, ал y координаталарын -1-ден 1-ге дейінгі диапазонда белгілеу үшін setworldcoordinates қолданыңыз.

4.1.15. Python сонымен бірге функциялардың атауларын параметрлер ретінде бере алады. Сондықтан біз өзіміз қалаған кез-келген функцияның графигін салу үшін 4.1.15 Жаттығуынан функцияны жалпылай аламыз. Келесі функцияны жазайық,

`plot(tortoise, n, f)`

мұндағы f – бір сандық аргументті алатын және санды қайтаратын кез келген функцияның аты. for циклінің ішінде plot функциясында біз f функциясын x индексті тұрақсыз шамасына келесідегідей қолдана аламыз

`tortoise.goto(x, f(x))`

Plot функциясын шақыру үшін бізге аргумент ретінде беру үшін бір немесе бірнеше функцияларды анықтау қажет. Мысалы, $x2$ салу үшін біз мынаны бере аламыз

```
def square(x): return x * x
```

Және plot функциясын

```
plot(george, 20, square) көмегімен шақыра аламыз
```

Немесе, созылған синусоида $\sin x$ құру үшін біз

```
def sin(x):
```

```
return 10 * math.sin(x) бере аламыз
```

Және plot функциясын

```
plot(george, 20, sin) арқылы шақыра аламыз
```

Plot жаңа нұсқасын жасаған соң, кем дегенде f параметрі ретінде plot-қа беру үшін жаңа бір функцияны жасаңыз. Сіз беретін функцияңызға қарай, сізге setworldcoordinates арқылы координаталар жүйесінің терезесін икемдеу қажет болуы мүмкін.

4.1.16. profitTable функциясын билеттердің барлық бағаларын төрттен бірге еселі етіп қарастыратындай өзгертіңіз.

4.1.17. pond функциясын жылдық өсім пайызын параметр ретінде қабылдайтындай етіп жалпылаңыз.

4.1.18. Pond функциясын тоғанды қосымша балық санымен жыл сайын толтыру мүмкіндігін қамтамасыз ететіндей әрі қарай жалпылаңыз.

4.1.19. countLinks функциясын экранға кестені төмендегідей шығаратындай етіп өзгертіңіз:

Тораптар	Байланыстар саны	
	Жаңа	Ортақ
2	1	1
3	2	3
4	3	6
5	4	10
6	5	15
7	6	21
8	7	28
9	8	36
10	9	45

4.1.20. Функцияны жазыңыз

```
growth1(totalDays)
```

Ол популяцияның күніне 3 түрге өсетіндігін үлгілейді. Әр күн үшін экранға күннің реттік нөмірі мен популяцияның жалпы көлемін

шығарыңыз

4.1.21. Функцияны жазыңыз

growth2(totalDays)

Ол популяцияның күніне 3 түрге өсуін, сонымен қатар орташа алғанда 2 күнде 1 түрге қысқаруын үлгілейді. Әр күн үшін экранға күннің реттік нөмірі мен популяцияның жалпы көлемін шығарыңыз

4.1.22. Функцияны жазыңыз

growth3(totalDays)

Ол популяцияның күн сайын 110% өсуін үлгілейді. Популяцияның бастапқы саны 10-ға тең делік. Әр күн үшін экранға күннің реттік нөмірі мен популяцияның жалпы көлемін шығарыңыз

4.1.23. Функцияны жазыңыз

growth4(totalDays)

Ол популяцияның бірінші күні 2-ге, екінші күні 4-ке, үшінші күні 8-ге, төртінші күні 16-ға және т.с.с. өсуін үлгілейді. Популяцияның бастапқы саны 10-ға тең делік. Әр күн үшін экранға күннің реттік нөмірі мен популяцияның жалпы көлемін шығарыңыз

4.1.24. Функцияны жазыңыз

sum(n)

Ол бүтін сандардың сомасын 1-ден n-ге дейін, оны қоса алғанда қайтарады. Мысалы, $\text{sum}(4) = 1 + 2 + 3 + 4 = 10$ қайтарады. (for циклін қолданыңыз; егер оңайтылған формуланы білсеңіз, оны қолданбаңыз.)

4.1.25. Функцияны жазыңыз

sumEven(n)

Ол бүтін жұп сандарды 2-ден n-ге дейін, оны қоса алғанда қайтарады. Мысалы,

$\text{sumEven}(6) = 2 + 4 + 6 = 12$ қайтарады. (for циклін қолданыңыз.)

4.1.26. Қыз балалар үштен он үш жасқа дейінгі аралықта жылына орташа алғанда алты сантиметрге өседі. Функцияны жазыңыз

growth(finalAge)

Ол берілген мәліметке негізделе отырып, экранға қарапайым бой өлшегіш шығарады: Үш жас мөлшеріндегі орташа қыз баланың бойы 95 сантиметр (37 дюйм) болады деген есеппен әр жас мөлшеріне бір жазбадан.

4.1.27. Функцияны жазыңыз

average(low, high)

Ол low-ден high-ге дейінгі, оны қоса алғандағы аралықта бүтін сандардың орта шамасын қайтарады. Мысалы, $\text{average}(3, 6) = (3 + 4 + 5 + 6) / 4 = 4.5$ қайтарады.

4.1.28. Функцияны жазыңыз

factorial(n)

Ол n мәнін қайтарады! $= 1 \times 2 \times 3 \times \dots \times (n - 1) \times n$. (Назар аударыңыз; сумматорға қандай бастапқы мән береміз?)

4.1.29. Функцияны жазыңыз

power(base, exponent)

Ол операторды ** exponent дәрежесінде base мәнін қайтарады.

. exponent оң бүтін сан деп есептейік.

4.1.30. Сандардың ішіндегі орташа геометриялық шамасы n сандар көбейтіндісінің n-ші түбірі ретінде анықталады. (n-ші түбір – $1/n$ дәрежесімен бірдей) Функцияны жазыңыз

geoMean(high)

Ол 1-ден high-ге дейін, оны қоса алғандағы геометриялық санның орташа шамасын қайтарады.

4.1.31. Функцияны жазыңыз

sumDigits(number, numDigits)

Ол numDigits белгілері бар number параметрінде дара цифрлар сомасын қайтарады. Мысалы, sumDigits(1234, 4) - $1 + 2 + 3 + 4 = 10$ шамасын қайтаруы тиіс (Сыбырсөз: for циклін және қалдықсыз бөлуді (// и %) қолданыңыз.)

4.1.32. Келесі қызық ойынға көз салайық. 100-ден төмен кез келген оң бүтін санды таңдаңыз да,

Оның шаршыларын цифрға қосыңыз. Мысалы, егер сіз 25-ті таңдасаңыз, оның цифрларының шаршыларының сомасы $22 + 52 = 29$. Енді жауап сіздің жаңа саныңыз болсын, әрекетті қайталаңыз. Мысалы, егер біз бұл әрекеттерді жалғастыра беретін болсақ, 25-тен бастап, біз келесіні аламыз: 25, 29,

85, 89, 145, 42 және т.с.с.

Функцияны жазыңыз

fun(number, iterations)

Ол экранға осы ойынмен түрлендірілетін сандардың бірізділігін, екі белгілі саннан бастап, iterations қадамдарының берілген сандары бойы жалғастыра отырып, экранға шығарады. Бұл бірізділіктегі бір де бір сан үштен артық цифрді иеленбейтінін білген жөн.

15-Тен 25-ке дейінгі әр бүтін санмен, iterations 30-дан кем болмайтын мөлшерде функцияны орындаңыз. Нені байқадыңыз? Алынған нәтижелерге негізделіп, екі топтың біріне осы сандардың әрбіріні жіктеуге бола ма?

4.1.33. Сізде инвестициялауға \$ 1,000 қаражат бар, Сізге екі жинақ

шоттардың бірін таңдау керек. Бірінші шот күн сайынғы капиталдандырумен бірге жылдық мөлшерлемесі 1% пайыз төлейді, бұл күн сайын $(1/365)$ % мөлшерлеме бойынша пайыздар есептелетіндігін білдіреді. Екінші шот 1.25 % мөлшерлемеде пайыз төлейді, ал капиталдандыру ай сайын болады. Функцияны жазыңыз

`interest(originalAmount, rate, periods)`

Ол берілген кезең (periods) саны бойы жинақталатын белгіленген жылдық пайыздық мөлшерлемені (rate) төлейтін шоттағы долларлардың originalAmount-нан бір жылда алынған пайыздарды есептейді. Пайыздық мөлшерлеме бөліктерде емес (яғни, 0,0125 емес 1,25) пайызда көрсетіледі делік. Бастапқы сұраққа жауап қайтару үшін функцияны қолданыңыз.

4.1.34. Сіз белгілі бір соманы ай сайын үнемдеп, оны ай сайынғы капиталдануы бар инвестициялық шотқа салуды қалайсыз делік. Болашақта қанша ақша алатыныңызды болжап анықтау үшін, келесі функцияны жазыңыз

`invest(investment, rate, years)`

Ол ай сайынғы капиталдандырумен $(rate / 12 \text{ \% ай сайын})$ белгіленген табыстылық мөлшерлемесін төлейтін, ай сайын долларлардың investment-не салым төлеуден келген табысты қайтарады.

4.1.35. Ипотекалық несие ай сайын несие бойынша ағымдағы қалдық негізінде біршама пайыздық төлем есептейді. Егер жылдық пайыздық мөлшерлеме ипотека бойынша $г \text{ \% құраса}$, онда ағымдағы теңгерімнен $г / 12 \text{ \% тең пайыз ай сайын берешек сомасына қосылады}$. Сонымен бірге ай сайын қарыз алушы, күтілгендей, төлем жасайды, бұл берешек сомасын азайтады.

Функцияны жазыңыз

`mortgage(principal, rate, years, payment)`

Ол экранға ипотека бойынша төлемдер мен несие мерзіміне ай сайын қалған теңгерім кестесін шығарады. Соңғы төлемге кез келген қалған теңгерім кіруі керек. Мысалы, 30 жылға берілген 4.5 % (rate) мөлшерлемедегі \$ 200,000 (principal) көлемдегі несие үшін айына \$ 1,000 (payment) көлемдегі төлемдер үшін келесі кесте жасалады:

Ай	Төлем	Тенгері
1	1000.0	199750.
2	1000.0	199499.
3	1000.0	199247.
...		
359	1000.0	11111.7
360	11153.	0.00
	46	

4.1.36. Мәселен, бактериялар саны сағатына 10% жылдамдықпен өседі делік, ал бастапқы саны 100 бактерия болсын. Функцияны жазыңыз

`bacteria(days)`

Ол бактерии санын берілген күн– `days` ішінде колонияларға қайтарады. Бір аптадан кейін колонияда қанша бактерия болады?

4.1.37. Алдыңғы жаттығуға жазған функцияны популяцияның бастапқы саны мен өсу жылдамдығының параметрлері ретінде алатындай етіп жалпылаңыз. Сол колонияда бір аптадан кейін 15% өсу жылдамдықта қанша бактерия болады?

4.2 ПОПУЛЯЦИЯ ӨЗГЕРІСТЕРІН ВИЗУАЛДАУ

Популяция санының уақыт аралығында өзгеруін визуалды түрде келтіру популяцияның даму үлгісін тереңірек түсінуге мүмкіндік береді. Біз популяцияның өзгеруін тасбақа графикасы арқылы 4.1 Тараудағыдай суреттей алар едік, алайда оның орнына біз графиктер құратын арнайы `matplotlib` модулін пайдаланамыз, оның солай аталу себебі ол `MATLAB`¹ бағдарламалау тілінің техникалық мүмкіндіктерін эмуляциялайды. Егер `matplotlib` сіздің компьютеріңізге әлі орнатылмаған болса, нұсқама алу үшін А қосымшасын қараңыз.

`matplotlib` қолдану үшін біз алдымен модульді

`import matplotlib.pyplot as pyplot` көмегімен импорттаймыз.

`matplotlib.pyplot` – дегеніміз модуль атауы; “`as pyplot`” ұзақ атауының орнына `pyplot` аббревиатурасын қолданып, бағдарламадағы модульге сілтенуге мүмкіндік береді. `Matplotlib`-те графиктерді құрудың негізгі функциялары екі аргументті қабылдайды: `X` мәндерінің тізімі және онымен байланысты `y` мәндерінің тізімі. `Python`-дағы тізім `list` тобының

¹ `MATLAB` - The MathWorks, Inc - тіркелген тауарлық белгісі болып табылады.

нысаны болып табылады, және шаршы жақшаға алынған элементтер бірізділігі үтірлермен бөлінген түрде ұсынылған, мысалы,
[4, 7, 2, 3.1, 12, 2.1]

Біз тізімдерге 4.1 Тарауда range мәндерін визуалдау үшін list функциясын қолданған кезде қарап өткенбіз; біз тізімдерді белсенді түрде 8 Тарауда қолданатын боламыз. Дәл қазір бізге for циклімізде популяция көлемдерінің тізімін жасап, оларды графикке құруды біліп алу қажет. Оны 119 Беттегі төменде келтірілген балық аулау шаруашылығына арналған функцияда қалай жасау керектігін қарап көрейік.

```
def pond(years, initialPopulation,
        harvest): """ (құжаттау жолдары
        түсірілген) """

    population =
    initialPopulation
    print('Год Population')
    for year in range(years):
        population = 1.08 * population - harvest
        print(' {0:^4} {1:>9.2f}'.format(year + 1,
        population))
    return population
```

Біз циклдың алдындағы популяцияның жылдық көлемдерінің бос тізімін жасаудан бастаймыз:

```
populationList = [ ]
```

Бос тізім екі шаршы жақшаға алынып ортасында бос орынмен белгіленетінін байқайсыз. Тізімнің соғына популяцияның жылдық санын қосу үшін list тобының append әдісін қолданамыз. Алдымен бос тізімнің соғына популяцияның бастапқы көлемін

```
populationList.append(initialPopulation) көмегімен қосамыз.
```

Егер біз популяцияның бастапқы көлемінің параметрі ретінде 12000 берсек, онда нәтижесінде бір элементтен [12000] тұратын populationList тізімін аламыз. Циклдың ішіне біз үдемелі тізімнің соғына population-ның әрбір мәнін

populationList.append(population) көмегімен қосқымыз келеді.

Бұл кодті pond функциясына қосып, print шақыруларын жойып, келесіні аламыз:

```
def pond(years, initialPopulation, harvest):  
    """Балық аулау шаруашылығындағы балық  
    популяциясын үлгілейді және  
    популяцияның жылдық санының графигін  
    құрайды. Жыл сайынғы  
    аулауды есептегінде популяция жылына  
    8% өседі.
```

Параметрлер:

years: initialPopulation болжауға арналған
жыл саны: Harvest популяцияның
бастапқы саны: Жыл сайынғы аулау
көлемі

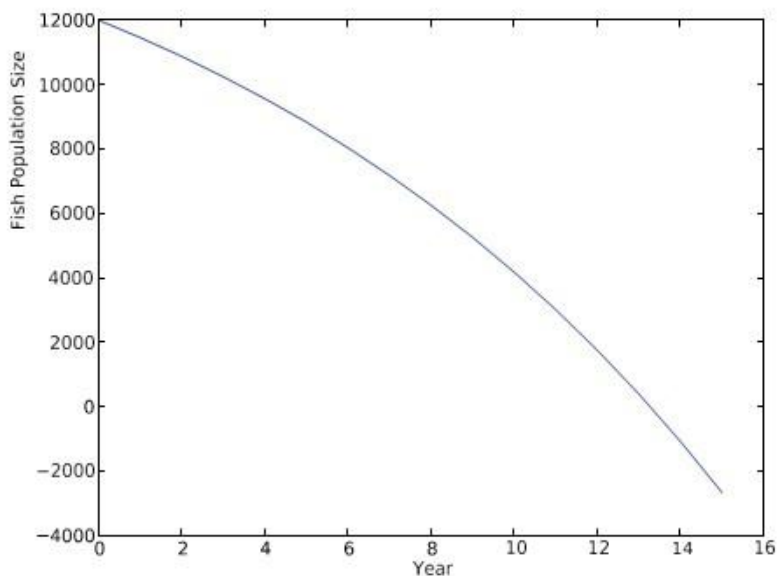
Қайтарылатын мән:

Популяцияның ақырғы саны"""

```
population = initialPopulation  
populationList = []  
populationList.append(initialPopul  
ation) for year in range(years):  
    population = 1.08 * population – harvest  
    populationList.append(population)
```

```
return population
```

Төменде келтірілген кестеде болжамға негізделіп популяцияның бастапқы саны 12,000 құраған болса, population ағымдағы мәнінің соңына қосу арқылы populationList әр итерацияда қалай ұлғайғандығы көрсетілген. Циклдің аяқталуына қарай біз көлемі year + 1 тең тізім аламыз.



Сурет 4.2 years = 15 үшін біздің балық шаруашылығы үлгімізге арналған популяция санының графигі.

year	population	populationList
--	12000	[12000]
1	11460.0	[12000, 11460.0]
2	10876.8	[12000, 11460.0, 10876.8]
⋮	⋮	⋮
13	392.53984839	[12000, 11460.0, ..., 392.53984839]
14	-1076.05696374	[12000, 11460.0, ..., 392.53984839, -1076.05696374]

Тізімге элементтерді қосу тәсілі мен біз осы тарауда талқылаған сумматорлардың әрекет ету қағидасының ұқсастықтары көп. Сумматорда біз мәндерді сомаға жинақтаймыз да, бірізді түрде үдемелі сомаға жаңа мәндерді қосамыз. Циклдің әр итерациясында үдемелі сома өзгереді (себебі өседі) . for цикліндегі тізімнің көмегімен біз жоғарыда басқа тәсілмен мәндерді - тізім көлемінде өсетін мәnniң соңына бірнеше рет қосу арқылы жинақтаймыз. Сондықтан біз осы әдісті жинақтаушы тізім деп атаймыз.

Енді біз бұл популяция көлемдерінің тізімін matplotlib графигінде у

мәндері ретінде қолдануды қалаймыз. x мәндері үшін бізге `range(years + 1)` арқылы алуға болатын сәйкес жыл саны қажет. Екі тізімге қол жеткізген соң, біз графикті `plot` функциясын шақырып жасай аламыз, содан кейін `show` функциясын шақырып графикті көрсете аламыз:

```
pyplot.plot(range(years + 1),  
populationList) pyplot.show()
```

`plot` функциясының бірінші аргументі - x мәндер тізімі, ал екіншісі – y мәндер тізімі. `matplotlib` модулі - `show` шақыруға дейін графигімізді икемдеу бойынша көптеген опциялардан тұрады. Ең қарапайымдарының бірі – x пен y осьтерін жазу функциялары:

```
pyplot.xlabel('Год') pyplot.ylabel('Численность популяции рыб')
```

График салу кодін қосып, 4.2. Суретте нәтижесі көрсетілген келесі функцияны аламыз:

```
import matplotlib.pyplot as pyplot көмегімен импорттаймыз.
```

```
def pond(years, initialPopulation,  
harvest): """ (құжаттау жолдары  
түсірілген) """  
  
population = initialPopulation  
populationList = []  
populationList.append(initialPopul  
ation) for year in range(years):  
    population = 1.08 * population – harvest  
    populationList.append(population)  
  
pyplot.plot(range(years + 1), populationList)  
pyplot.xlabel('Год')  
pyplot.ylabel('Численность  
популяции рыб') pyplot.show()  
  
return population
```

Күрделірек графиктерді салу үшін біз осьтер масштабын, айнымалылардың түсі мен стилін өзгерте аламыз, сонымен бірге бір графикте бірнеше айнымалыны белгілей аламыз. Бар опциялардың мысалдары берілген Б.4 қосымшасын қараңыз. Опционалдық мәндердің кейбіреулері атау= мәні түрінде негізгі аргументтер ретінде көрсетілуі

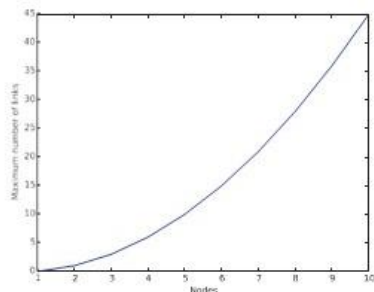
керек. Мысалы, графикте айнымалының түсін қызылға бояп, графиктің түсіндірме сөзіне орын көрсету үшін төмендегіге ұқсас функцияны шақыруға болады:

```
pyplot.plot(x, y, color = 'red', label = 'алабұға популяциясы')
pyplot.legend()      # белгіленген жолдардан түсіндірме сөз жасайды
```

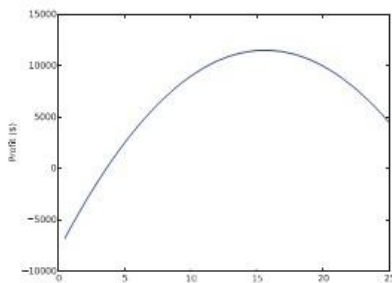
Жаттығулар

4.2.1. 123 Беттегі функциясын x осімен торап санын және y осі бойымен максималды байланыс санының графигін құру үшін matplotlib-ті қолданатындай етіп өзгертіңіз. Нәтижесінде шыққан график 4.3. Суреттегідей болуы керек.

4.2.2. 128 Беттегі profitPlot функциясын билет бағасын x осімен және табысты y осімен жүретін график салу үшін matplotlib-ті пайдаланатындай етіп өзгертіңіз. (Tortoisetortoise параметрін жойыңыз.) Нәтижесінде шыққан график 4.4. Суреттегідей болуы керек. X осінде дұрыс бағаларды (жарты долларлық қадаммен) алу үшін, Сізгех -тің екінші мәндер тізімін жасап, оған әрбір итерацияда realPrice қосып отыру керек.



Сурет 4.3 4.2.1 жаттығуға арналған график.



Сурет 4.4 4.2.2 жаттығуға арналған график.

4.2.3. 4.1.21 Жаттығуындағы growth1 функциясын ол x осімен күннің реттік нөмірін және y осімен популяцияның жалпы санын көрсететін график салу үшін matplotlib-ті қолданатындай етіп өзгертіңіз. 30 күн бойы популяцияның өсуін көрсететін график жасаңыз.

4.2.4. 4.1.23 Жаттығуындағы growth3 функциясын ол x осімен күннің реттік нөмірін және y осімен популяцияның жалпы санын көрсететін график салу үшін matplotlib-ті қолданатындай етіп өзгертіңіз. 30 күн бойы популяцияның өсуін көрсететін график жасаңыз.

4.2.5. 4.1.35 Жаттығуындағы invest функциясын x осімен

айдың реттік нөмірін және у осімен салымдағы жинақтың жалпы сомасын көрсететін графикті салу үшін matplotlib-ті қолданатындай етіп өзгертiңiз. Жылдық пайыздық мөлшерлемесі 8% 10 жыл бойы

айына \$ 50 пайда түскен салымның өсуін көрсететін графикті құрыңыз.

4.3 ШАРТТЫ ИТЕРАЦИЯ

Біздің балық аулау шаруашылығы үлгімізде популяция санының теріс мәнге ие болатын кездерін анықтау үшін популяцияның көлемін кем дегенде 14 жылға дейінгі жыл сайынғы көлемін экранға шығарып, нәтижесін көру жеткілікті болатын. Алайда, егер популяция саны мың жылдан кейін нөлден төмендейтіндігін ескерсек, онда шығыс мәндеріне қарау өте ыңғайсыз болатын еді. Оның орнына бізге population нөлге дейін төмендеген кездегі итерацияның дәл тоқтаған жылын көрсетіп, содан кейін ол орын алған жылды қайтаратын бағдарлама қажет. Бұл тапсырманың басқа түрі, себебі циклдің басталуына дейін біз қанша итерация керек болатынын нақты білмейміз. Басқа сөзбен айтар болсақ, бізде for циклінде қандай мәнде range-ге беру керек екендігін білетін мүмкіндік жоқ.

Оның орнына бізге кейбір шарт орындалғанша жүзеге асатын циклдің жалпы түрі қажет. Мұндай цикл әдетте while цикл деп немесе шартты тексеретін цикл деп аталады. Python-да while цикл мынадай болады:

```
while <шарт>:  
<Цикл денесі>
```

<Шарт> логикалық сөзге ауысады, ол Ақиқат немесе

Жалған деген мәнге ие болады, ал <цикл денесі> цикл денесін құрайтын нұсқаулыққа ауысады. Цикл әрбір итерация алдында шарттардың мәнін тексереді. Егер шарт - ақиқат болса, ол цикл денесінде нұсқаулықтарды орындайды, содан кейін шартты қайта тексереді. Егер шарт жалған болса, цикл денесі жіберіледі де, цикл тәмамдалады.

Біз логикалық сөздер туралы толығырақ келесі тарауда айтамыз; қазір бізге тек қана population > 0 секілді өте қарапайымдары керек. Егер population мағынасы оң болса, бұл логикалық сөз ақиқат, ал керісінше жағдайда жалған болады. Осы логикалық сөздің көмегімен while циклінде келесі функцияда біз балық популяция нөлге дейін төмендейтін жылды таба аламыз.

```
def yearsUntilZero(initialPopulation, harvest):
    """Анықтайды# жыл, балық популяциясы нөлге
    жеткенше.
    Жыл сайынғы аулауды қосқанда популяция жылына 8%
    өседі.
```

Параметрлер:
 initialPopulation: Harvest популяцияның
 бастапқы саны. Жыл сайынғы аулау
 көлемі

Қайтарылатын мән: Популяция нөлге жететін
 жыл"""

```
population =
initialPopulation
year = 0
while population > 0:
    population = 1.08 * population -
    harvest
    year = year + 1
return year
```

4.1. Тараудағы бастапқы `pond` функциясындағыдай `initialPopulation` -12000 -ға тең және `harvest` - 1500-ге тең делік. Онда циклдің алдына `population - 12000` және `year - 0` мәндерін берейік `population > 0` ақиқат болғандықтан, цикл денесі орындалып, `population` мәндерін 11460 және `year` мәндерін 1-ге өзгертеді. (Мүмкін Сіз 118 Бетке оралып популяцияның жылдық санының функциясын еске түсіргіңіз келеді.) Содан кейін шартты қайта тексеру үшін циклдің басына қайтамыз. `population > 0` әлі де ақиқат болғандықтан, цикл денесі қайта орындалып, `population - 10876` және `year -`

2-ге өзгереді. Итерациялар `year 14`-ке жеткенше жалғасады. Осы жылы, `population`

-1076.06 жетеді. Енді шартты тексерген кезде біз `population > 0` жалған екендігін байқаймыз, сондықтан цикл аяқталады және функция 14 мәнін қайтарады.

`while` циклдерін қолдану екі себепті қиындықтар тудыруы мүмкін. Біріншісі – `while` цикл мүлдем басталмауы мүмкін. Мысалы, егер `population` бастапқы мәні нөлге тең болса, онда `while` циклінің шарты бірінші итерацияға дейін жалған болады да, цикл басталмай ақ тоқтап қалады.

Ойлануға берілетін сұрақ 4.16 Бұл жағдайда функция нені қайтарады?

Кейде мүлдем қайталанбайтын цикл, әдеттегідей – өте жаман нәрсе емес, керісінше біз оны өз мақсатымызда қолдана аламыз. Бұл жағдайда, егер population басынан бастап нөлге тең болса, онда функция да нөлді қайтарады, себебі year мәні циклде бір рет те ұлғаймайды. Және бұл дұрыс; популяция бастапқы санынан кейін санақ басталғанға дейін нөлінші жылы нөлге түсті. Алайда while циклдерін қосатын алгоритмдерін жасау кезінде осыны есте сақтау керек.

Екінші себеп – while цикл шексіз циклге айналып кетуі мүмкін. Мысалы, initialPopulation - 12000 және harvest - 1500 орнына 800-ге тең делік. Бұл жағдайда, 119 бетте көргеніміздей, популяция саны, керісінше жыл сайын өседі. Осылайша популяция көлемі ешқашан нөлге түспейді және циклдің шарты ешқашан жалған болмайды, ал шексіз қайталана береді. Осы себептен біз міндетті түрде while циклінің денесі ақырында циклдің жалған шартына әкелетіндігіне көз жеткізуіміз қажет.

Тағы бір мысал қарастырайық. Бізде инвестициялауға \$ 1000 бар делік, енді біз бұл соманы жылына 5 % өсіммен екі еселеу үшін салымда ақшаны қанша уақыт сақтау керектігін білгіміз келеді. Бұл сұраққа жауап беру үшін, біз келесі циклге ұқсас бірін жасай аламыз, ол жыл сайын 5% жылдық өсім есептейді:

```
amount = 1000
while ???:
    amount = 1.05 * amount
print(amount)
```

Ойлануға берілетін сұрақ 4.17 *while циклінде не нәрсе шарт болу керек?*

Біз циклдің amount 2000 мәніне жеткен кезде қайталануын тоқтағанын қалаймыз. Сәйкесінше, біз amount < 2000 болғанша, циклдің қайталануын жалғастыруды қалаймыз.

```
amount = 1000
while amount < 2000:
    amount = 1.05 *
    amount
print(amount)
```


Ойлануға берілетін сұрақ 4.18 *Мына код сегменті экранға нені шығарады? Нәтиже бізге не туралы айтады?*

Цикл орындалған соң, экранға ақырғы сома (шамамен \$ 2078.93) шығарылады, алайда біз сұрағымызға жауап алмадық.

Ойлануға берілетін сұрақ 4.19 *\$ 1000 екі еселеу үшін қанша жыл кететіндігін қалай есептеуге болады?*

Сұрағыңызға жауап беру үшін, while циклі неше рет қайталанатындығын есептеу керек, бұл yearsUntilZero функциясында істелген әрекетке өте ұқсайды. Әр итерацияда бірлікке артылатын тағы бір ауыспалыны енгізіп, оның мәнін amount ақырғы мәнмен бірге циклдің орындалғанынан кейін экранға шығаруға болады:

```
amount = 1000
while amount < 2000:
    amount = 1.05 *
    amount
    year = year +
    1
print(year, amount)
```

Ойлануға берілетін сұрақ 4.20 *Мына өзгерістерді енгізіп, кодты қайта іске қосыңыз. Енді экранға не шығарылады?*

Ой, year анықталмады деген қателік туралы хабарлама шықты.

Ойлануға берілетін сұрақ 4.21 Қателікті қалай жоюға болады?

Проблема циклдің алдында year мәнін инициализацияламағанымызда жатыр. Шын мәнінде, year = year + 1 бірінші рет орындалғанда меншіктеу нұсқауының оң жағынан year анықталмаған болатын. Циклдің алдында нұсқау қосу бұл қателікті жояды:

```
amount = 1000
year = 0
while amount < 2000:
    amount = 1.05 *
    amount
    year = year +
```

1

```
print(year, amount)
```

Ойлануға берілетін сұрақ 4.22 Бұл код сегменті экранға енді не шығарады? Басқа сөзбен айтқанда, \$ 1000 екі еселеу үшін қанша жыл кететін болды?

Біз осы тарауда және тағы 5.5 Тарауда кейінірек while циклдерінің тағы бірнеше мысалын қарастырамыз.

Жаттығулар

4.3.1. Сіз банкке жыл сайынғы капиталданумен 3 % жылдық өсім беретін салымға \$ 1000 салдыңыз делік. while циклін шотыңызда кем едегенде \$ 1200 болатын мерзімді анықтау үшін қалай қолданар едіңіз?

4.3.2. Алдыңғы сұрақты қайталайық, бірақ осы жолы функцияны жазыңыз

```
interest(amount, rate, target)
```

Ол бастапқы соманы, пайыздық мөлшерлемені және мақсат сомасын параметрлер ретінде алады. Функция мақсат сомасына жету үшін кететін жыл санын қайтаруы керек.

4.3.3. while циклдері for циклдеріне қарағанда жалпы сипатқа ие болғандықтан, біз for циклінің тәртібін while циклімен эмуляциялай аламыз. Мысалы, for циклінің мына тәртібін эмуляциялауға болады

```
for counter in range(10): print(counter)
```

while циклінің көмегімен

```
counter = 0
```

```
while counter < 10:
```

```
    print(counter)
```

```
    counter = counter + 1
```

Екі циклді де «қолмен» орындаймыз, бұл олардың неліктен эквивалентті екендігін түсініп алу үшін қажет.

(а) While циклінің алдындағы counter = 0 инициализациясын ескермесек не болады? Неліктен олай болады?

(b) Егер циклінің денесіндегі `counter = counter + 1` ескермесек не болады? Цикл экранға не шығарады?

(c) Келесі `for` циклін `while` циклімен қалай эмуляциялауға болатынын ұсыныңыз:

```
for index in range(3, 2): print(index)
```

(d) Келесі `for` циклін `while` циклімен қалай эмуляциялауға болатынын ұсыныңыз:

```
for index in range(12, 3, -1): print(index)
```

4.3.4. 127 беттегі `profitTable` функциясында біз жарты долларға еселенген билеттердің барлық бағаларын жанама түрде қарастыру үшін `for` циклін қолдандық. Осы функцияны көшіріп алыңыз, оның орнына ол `while` циклін қолданды, ол `price` -ты әр итерацияда \$ 0.50 ұлғайтады.

4.3.5. Зомби екі адамды зомбиге күн сайын айналдыра алады. Бір зомбиден бастасақ, планетамыздың барлық халқын ((7 миллиард адам) зомбиге айналдыруға қанша уақыт кетеді? Функцияны жазыңыз `zombieApocalypse()`

Ол осы сұраққа жауап қайтарады.

4.3.6. Трибблдер сағатына 50 % жылдамдықпен көбейеді (төмен қарай дөңгелектенеді, егер олардың саны тақ болса). 10 трибблге популяция саны 1 миллионға жету үшін қанша уақыт керек болады? Функцияны жазыңыз `tribbleApocalypse()`

Ол осы сұраққа жауап қайтарады.

4.3.7. Әрбір вампир `v` адамды бір күнде вампирге айналдыра алады. Алайда күніне `k` вампир өлтіре алатын вампирлер аулайтын аңшылар тобы бар. Егер вампирлер қауымы өз бастамасын `vampires` мүшелерінен алатын болса, `people` адамы бар қала бір де бір адамы қалмайтын қалаға айналғанша қанша уақыт кетеді? Функцияны жазыңыз `vampireApocalypse(v, k, vampires, people)`

Ол осы сұраққа жауап қайтарады.

4.3.8. Амеба `h` сағатта екіге бөліне алады. Бір амебаға `target` амебаға айналу үшін қанша сағат керек. Функцияны жазыңыз `amoebaGrowth(h, target)`

Ол осы сұраққа жауап қайтарады.

Ол осы сұраққа жауап қайтарады.

*4.4 ҮЗДІКСІЗ ҮЛГІЛЕР

Егер біз балық аулау шаруашылығындағы жағдайды нақты үлгілегіміз келсе, біз балық популяциясының шын мәнінде жылына бірнеше рет өзгеретіндігін білуіміз қажет. Іс жүзінде барлық табиғи үдерістердегідей, өзгерістер үнемі немесе математикалық түрде, уақыт аралығында үздіксіз жүреді. Үздіксіз табиғи үдерістерді нақтырақ үлгілеу үшін, бізге аз уақыт аралықтарын пайдаланып, популяция санын жиі жаңартып отыру қажет. Мысалы, біз балық популяциясының санын жыл сайын емес, ай сайын жаңартып отыра алар едік

$$\text{population} = \text{population} + 0.08 * \text{population} - 1500$$

Он бір жылды он екі айға ауыстыру керек:

for month in range(12):

$$\text{population} = \text{population} + (0.08 / 12) * \text{population} - (1500 / 12)$$

Өсу жылдамдығы 0.08 және жылына аулау 1500 деп берілгендіктен, біз бұл екі мәнде 12-ге бөлеміз.

Ойлануға берілетін сұрақ 4.23 Екі жағдай үшін population ақырғы мәні бірдей ме?

Егер population бастапқы мәні 12000 болса, population мәні бір жылдық жаңартудан кейін 11460.0 құрайды, ал 12 айлық жаңартудан кейін ақырғы мән – 11439.753329049303 құрайды. Пайыз ай сайын «капиталданғандықтан», популяция жылдам қысқарады.

Дәл осылай банк несиелері де жұмыс істейді. Мәселен, банк жылдық пайыздық мөлшерлемені 6% (немесе 0.06) деп белгілейді, ал содан кейін ай сайын пайыздарды $6 / 12 \% = 0,5 \%$ (немесе 0.005) мөлшерде капиталдайды, бұл Сізге аударылатын тиімді пайыздық мөлшерлеме деп аталатын 12 іс жүзіндегі жылдық пайыздық мөлшерлеме $(1+0.005)^{12} - 1 \approx$

$0.0617 = 6.17\%$ құрайды. Іс жүзінде ай сайынғы мөлшерлеме деп анықталатын, 12-ге көбейтілген жылдық пайыздық мөлшерлемені

кейде сонымен қатар “атаулы мөлшерлеме” деп атайды. Осылайша, біз балық популяциясы атаулы мөлшерлеме бойынша 8% өседі, бірақ ай сайын жаңартылып отырады деп айта аламыз.

Айырымдық теңдеулер

Бұндай популяция үлгісі нақтырақ айырымдық теңдеу арқылы

көрсетіледі, ол сондай ақ рекуррентті ара қатынас деп аталады. Егер $P(t)$ орнына t жылдың соңындағы балық популяциясын алатын болсақ, онда алғашқы үлгімізді анықтайтын айырымдық теңдеу төмендегідей

$$P(t) = \underbrace{P(t-1)}_{\text{Немесе, дәл сондай болмақ}} + \underbrace{0.08 \cdot P(t-1)}_{\text{Алдыңғы жылдағы популяция}} - \underbrace{1500}_{\text{Мөлшерлес ұлғаю}}$$

$$P(t) = 1.08 \cdot P(t-1) - 1500.$$

Басқа сөзбен айтқанда, популяция саны t жылдың соңында алдыңғы жылдың соңындағы $(t-1)$ минус 1500 популяция санына қарағанда 1.08 есе көбірек. Бастапқы популяция, немесе формалды түрде, бастапқы шарт – бұл $P(0) = 12,000$. Біз популяцияның болжалды санын кез келген $P(0)$ бастап, $P(t)$ мәндерін бірізді түрде есептеу үшін айырымды теңдеуді қолданып отырып, нақты бір жыл үшін таба аламыз. Мысалы, $P(4)$ деп ұсынылған популяцияның төрт жылдан кейінгі болжалды санын білгіміз келеді делік. Бастапқы шарттан бастаймыз: $P(0) = 12,000$. Со-сын, $t = 1$ үшін айырымдық теңдеуді қолданамыз :

$$P(1) = 1.08 \cdot P(0) - 1500 = 1.08 \cdot 12,000 - 1500 = 11,460.$$

Енді, бізде $P(1)$ бар болса, $P(2)$ есептей аламыз:

$$P(2) = 1.08 \cdot P(1) - 1500 = 1.08 \cdot 11,460 - 1500 = 10,876.8.$$

Жалғастыра отырып,

$$P(3) = 1.08 \cdot P(2) - 1500 = 1.08 \cdot 10,876.8 - 1500 = 10,246.94$$

және

$$P(4) = 1.08 \cdot P(3) - 1500 = 1.08 \cdot 10,246.94 - 1500 = 9,566.6952.$$

Осылайша бұл үлгі алабұға популяциясы 4 жылдан кейін 9,566 құрайтынын болжайды Біз дәл осы үдерісті 4.1. Тараудағы for циклінде қолдандық.

Бұл дискретті үлгіні үздіксіз ету үшін, жаңартудың кішігірім интерва-

лын анықтамыз, әдетте ол Δt деп аталады (Δ дегеніміз өзгеріс», яғни Δt дегеніміз «уақыттағы өзгеріс»).

Егер біз мысал ретінде популяция санын ай сайын жаңартқымыз келсе онда, $\Delta t = 1/12$ қолданамыз. Енді айырымдық теңдеуімізді мына түрде көрсетейік

$$P(t) = P(t - \Delta t) + (0.08 \cdot P(t - \Delta t) - 1500) \cdot \Delta t$$

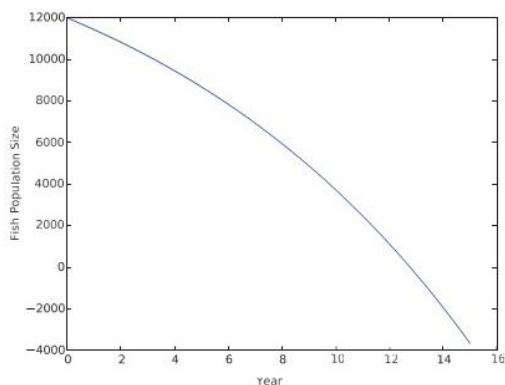
Бұл айырымдық теңдеу t жылының соңындағы популяция санын бір Δt бұрынғы аралық бұрынғы популяция санына сүйеніп анықтайды. Мысалы, егер $t = 3$ -ке тең болса, онда $P(t)$ дегеніміз 3 жылдың соңындағы популяция саны, ал

$P(t - \Delta t)$ дегеніміз “2¹¹ жылдың” соңындағы популяция саны, бұл

Бір ай бұрынғы аралыққа тең. Өсу темпі мен аулау саны

145 бетте циклінде жасалғандай Δt рет дифференциалданғанына назар аударыңыз.

Бұл үлгіні іске асыру үшін, 119 беттегі алгоритмге бірнеше аналогтық өзгерістер енгізу керек. Біріншіден, Δt мәнін параметрретінде көрсетіп, жақындау нақтылығын бақылау мүмкіндігін алу керек. Екіншіден, біз жыл сайын $1/\Delta t$ есе азайған құбылыстарды көрсету үшін цикліміздегі итерациялар санын өзгертуіміз керек; итерациялар саны $\text{years} \cdot (1/\Delta t) = \text{years}/\Delta t$ болады. Үшіншіден, біз бұл айырымдық теңдеудің жаңа түрін көрсету үшін циклдағы сумматордың жаңару үдерісін өзгертуіміз керек. Бұл өзгерістер төменде көрсетілген. Біз Δt белгілеу үшін dt қолданамыз.



Сурет 4.5 pond(15, 12000, 1500, 0.01)шақыру арқылы жасалған график.

```
def pond(years, initialPopulation, harvest,
dt): """ (құжаттау жолдары
түсірілген) """

population = initialPopulation
for step in range(1, int(years / dt) + 1):
    population = population + (0.08 * population -
harvest) * dt
return population
```

Ойлануға берілетін сұрақ 4.24. Неліктен біз for циклінде `range(int(years / dt))` орнына `range(1, int(years / dt) + 1)` қолдандық?

Біз for циклін нөлдің орнына бірден бастаймыз, себебі циклдің бірінші итерациясы дегеніміз үлгідегі уақыттың бірінші қадамы болып табылады. Циклдің алдында `population` иемденетін популяцияның бастапқы саны нөлдік уақыттағы популяцияны білдіреді.

Бұл үлгілеу нәтижелерінің графигін салу үшін 4.2. Тарауындағы әдісті қолданамыз. Алайда біз сондай ақ x осі үшін уақыт мәндерінің тізімін жасау үшін жинақтаушы тізімді қолданамыз, себебі `step` индекстік ауыспалы шамасының мәндері енді жыл нөмірін көрсетпейді. Келесі функцияда `step * dt` мәніне t ауыспалы шамасы беріледі, сосын `timeList` атымен тізімге қосылады.

```
import matplotlib.pyplot as plt көмегімен импорттаймыз.
```

```
def pond(years, initialPopulation, harvest, dt):  
    """Балық аулау шаруашылығындағы балық  
    популяциясын үлгілейді және популяция санының  
    жылдық графигін құрайды. Популяция 8%  
    жылдық атаулы мөлшерлемеді және жыл сайынғы  
    аулаумен бірге өседі.
```

Параметрлер:

```
    years: initialPopulation болжауға арналған  
    жыл саны: Harvest популяцияның  
    бастапқы саны: Жыл сайынғы аулау  
    көлемі  
    dt: үлгідегі "Delta t" мәні (жыл бөлігі)
```

Қайтарылатын мән:

```
    Популяцияның ақырғы саны"""
```

```
    population = initialPopulation  
    populationList =  
    [initialPopulation] timeList = [0]  
    t = 0
```

```
    for step in range(1, int(years / dt) + 1):  
        population = population + (0.08 * population -  
        harvest) * dt populationList.append(population)  
        t = step * dt  
        timeList.ap  
        pend(t)
```

```
    plt.plot(timeList,  
    populationList)  
    plt.xlabel('Жыл')  
    plt.ylabel('Балық  
    популяциясының саны')  
    plt.show()  
    return population
```

$\Delta t = 0.01$ мәніндегі бұл функцияны шақыру арқылы жасалған график 4.5. Суретте көрсетілген. Бұл графикті 4.2 суретімен салыстырыңыз. Шынай үздіксіз үдерістің аппроксимациясына расында жету үшін біз Δt -нің өте кішкентай мәндерін қолдануымыз керек. Алайда бұндай әрекет шарасыз ымыраға байланысты, ол жөнінде біз осы тарауда толығырақ айтамыз.

Радиокөміртекті талдау

Археологтар органикалық қалдықтардың жасын анықтағысы келгенде, олар жиі радиокөміртекті талдауды қолданады. -12 (12C) көміртегі мен -14 (немесе 14C радиокөміртегі) көміртегі де көміртектің изотоптары болып табылады, олар біздің атмосферада салыстырмалы тұрақты пропорцияда болады. -12 көміртегі тұрақты изотоп болса, -14 көміртек тұрақсыз болып табылады және белгілі жылдамдықта ыдырайды. Барлық тізі ағзалар екі изотопты да сіңіреді, өлген кезде олардағы пропорция атмосферадағымен тең болады. Одан кейін ағзаға сіңірілген -14 көміртек белгілі жылдамдықпен ыдырайды, ал -12 көміртек толық сақталады. Органикалық қалдықтардағы (шамамен 60 000 жылға дейін) -12 көміртегінің -14 көміртегіне ағымдағы ара қатынасын талдасақ және бұл ара қатынасты атмосферадағы белгілі ара қатынаспен салыстыратын болсақ, ағзаның шамамен қай уақытта өлгендігін анықтауға болады.

-14 көміртегінің жылдық ыдырау жылдамдығы (нақтырақайтқанда, тұрақты ыдырау²) шамамен

$$k = -0.000121.$$

Радиобелсенді ыдырау дискретті интервал ішінде болмайды, ол үздіксіз үдеріс болып табылады. Сәйкесінше, t жылы басындағы -14 көміртегінің көлемін $Q(t)$ $Q(t - \Delta t)$ шамасы арқылы, бір жыл бұрынғы Δt шағын бөлігіндегі -14 көміртегінің көлемін анықтау керек. Соныдқтан -14 көміртегінің ыдырау үлгісінің айырымды теңдеуі мынадай болады

$$Q(t) = Q(t - \Delta t) + k \cdot Q(t - \Delta t) \cdot \Delta t.$$

k тұрақты ыдырау шамасы бір жылға келтірілгендіктен, біз оны Δt көбейтіп, Δt ұзақтықтағы интервалға шамалас етіп азайтуымыз керек. $Q(0) = q$ бастапқы шартын анықтайық, мұндағы q дегеніміз -14 көміртегінің бастапқы саны болып табылады.

²-14 көміртегінің молекулярлы ыдырау ықтималдығы Δt жылының өте аз бөлігі үшін $k\Delta t$ құрайды.

Мүлдем әртүрлі қолданыстарға қарамастан, біз үлгіні дәл балық популяциясының үздіксіз үлгісіндей етіп іске асыра аламыз.

```
import matplotlib.pyplot as pyplot көмегімен импорттаймыз.
```

```
def decayC14(originalAmount, years, dt): ""-14  
    көміртегінің үздіксіз ыдырауын  
    аппроксимациялайды.
```

Параметрлер:

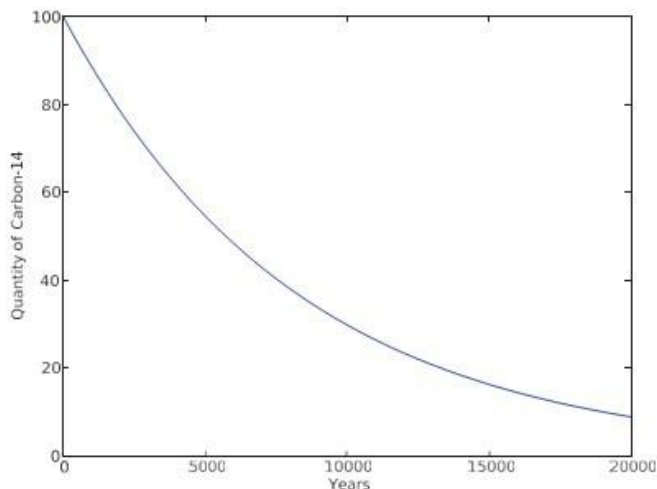
originalAmount: -14 көміртегінің (ж) years
бастапқы мөлшері: Болжау үшін берілетін жыл
саны
dt: үлгідегі "Delta t" мәні (жыл бөлігі)

Қайтарылатын мән: -14 (ж)
көміртегінің ақырғы мөлшері ""

```
k = -0.000121  
amount =  
originalAmount  
t = 0  
timeList = [0] # x мәні график құру  
үшін  
amountList = [amount] # y мәні график құру үшін
```

```
for step in range(1, int(years/dt) + 1):  
    amount = amount + k * amount *  
    dt  
    t = step * dt  
    timeList.append  
    (t)  
    amountList.append  
    amount  
    t
```

```
pyplot.plot(timeList, amountList)
```



4.6-сурет `decayC14(100, 20000, 0.01)` функциясы арқылы құрылған көміртегі-14-тің ыдырау графигі.

```
pyplot.xlabel('Лет')
pyplot.ylabel('көміртегі саны - 14')
pyplot.show()
```

```
return amount
```

Біздің бастапқы сумматорларымыз секілді, бұл функция цикл алдындағы `amount` деп аталатын сумматорымыздың ауыспалы шамасын инициализациялайды. Содан кейін сумматор жоғардағы айырымдық теңдеуге сәйкес цикл денесінде жаңарады. 4.6-суретте бұл функция бөлігінің мысалы көрсетілген.

Ойлануға берілетін сұрақ 4.25 *5,000 жылдық ыдыраудан кейін 100 г көміртектен қанша қалады? Δt -ның әртүрлі мәндерін 1 -ден 0.001 дейін қолданып көріңіз. Нені байқадыңыз?*

Нақтылық пен уақыт арасындағы ымыра

Үздіксіз үлгілердің аппроксимациясы Δt мәні қаншалықты аз болса соншалықты нақты болады. Алайда нақтылықтың өз құны бар. `decayC14` функциясының уақытша қиындығы $y/\Delta t$ пропорционал, мұндағы y –

үлгіленетін жыл саны, сондай ақ for циклі қайталанатын сан болып табылады. Осылайша, егер біз нақтылықты Δt -ні екіге бөліп арттырғымыз келсе, онда біздің алгоритмді орындауға қажетті уақыт екіге көбейеді. Егер біз Δt ағымдағы мәnnің оннан бір бөлігін құрағанын қаласақ, біздің алгоритмді орындауға он есе уақыт керек болады.

Ескертпе 4.1: Дифференциалды теңдеулер

Егер Сіз математикалық талдау күрсын өткен болсаңыз, онда берілген үздіксіз үлгілердегі *Дифференциалды теңдеудің* аппроксимациясын біте аласыз. Дифференциалды теңдеу дегеніміз - туынды деп аталатын осы функцияның өзгеру жылдамдығымен функцияны байланыстыратын теңдеу. Мысалы, -14 көміртегінің ыдырауы туралы тапсырмаға сәйкес дифференциалды теңдеу дегеніміз -

$$\frac{dQ}{dt} = -kQ$$

Мұндағы Q - -14 көміртегінің мөлшері, dQ/dt - -14 көміртегі мөлшерінің уақыт аралығындағы өзгеру жылдамдығы, (яғни Q -дің t бойынша туындысы) ал k - ыдыраудың тұрақты шамасы. Бұл дифференциалды теңдеуді математикалық талдаудың негізгі әдістемелерін пайдалана отырып шешетін болсақ мынаны аламыз

$$Q(t) = Q(0) e^{-kt}$$

Мұндағы $Q(0)$ - -14 көміртегінің бастапқы мөлшері. Біз бұл теңдеуді 200 жылдан кейін 1000 г -14 көміртегіден қанша көміртегі мөлшері қалатынын бірден білу үшін қолдана аламыз:

$$Q(2000) = 1000 e^{-2000k} \approx 785.0562.$$

Осыған ұқсас қарапайым дифференциалды теңдеулер ондай шешілетіндігіне қарамастан, егер Сізге математикалық талдау таныс болса, ғылыми ортадағы тәжірибеде кезігетін дифференциалды теңдеулердің көбісі ондай ондай емес, бұл жақындатылған итерациялық шешу әдістерін түбегейлі маңызды етеді. Осы тарауда біз пайдаланатын *Эйлер әдісі* деп аталатын аппроксимация әдісі - негіз қалаушы болып табылады және Δt мөлшерлес дәлсіздік береді. Бұдан озық әдістері аппроксимация дәлсіздігінің кейін азаюына ұмтылады.

Δt азаятын мәндері нәтижеге қалай әсер ететіндігін түсіну үшін, originalAmount = 1000 және years=2000 параметрлі decayC14 функциясында не болатынын байқап көрелік. Төмендегі кестеде осы нәтижелер келтірілген және соңғы жолда теориялық алынған жауап берілген (Ескертулер 4.1). Қателік бағанасы әрбір Δt мәні үшін нәтиже мен осы мән арасындағы айырмашылықты көрсетеді. Барлық мәндер үш мәнді цифрға дейін дөңгелектелген, бұл ыдыраудың тұрақты шамасының аппроксима-

ция сипатын көрсету үшін жасалған. Кестеден dt кішкентай мәндері шынында нақтырақ жуықтауды қамтамасыз ететіндігін байқаймыз.

dt	Ақырғы нәтиже	қателік	уақыт
1.0	785.0447	0.0115	500 мкс
0.1	785.0550	0.0012	5 мс
0.01	785.0561	0.0001	50 мс
0.001	785.0562	0.0	500 мс
0.0001	785.0562	0.0	5 с
—	785.0562	—	—

Ойлануға берілетін сұрақ 4.26 *dt мәні мен қателік арасындағы өзара байланыс қандай? dt мәні мен орындау уақыты арасындағы байланыс қандай?*

Кестедегі әр жол алдыңғы жолдағы уақытқа қарағанда он есе көп уақытты алатын есепті білдіреді, себебі dt мәні он есе азаяды. Алайда дәлсіздік те он есе азаяды. Мұндай ымыра тиімді ме? Жауап жағдайға байланысты. Әрине, $dt = 0.0001$ қолдану дәлелденбеді, себебі ол $dt = 0.001$ сияқы (үш мәнді цифрға дейін) жауапты береді, алайда он есе артық уақытты алады.

Бұл ымыра түрлері – баға/сапа – барлық сала үшін ортақ, және компьютерлік ғылымдар ерекшелік емес. Жылдамырақ жадыны жасау қымбатырақ технологияны қажет етеді. Желілер арқылы нақтырақ мәліметтерді жіберу үлкен үстеме шығындарды қажет етеді. Ал күрделі тапсырмалар үшін үздік аппроксимациялы шешімдерді іздеу көп уақытты талап етеді.

Қателіктерді тарату

Осы тараудың екі `pond` және `desayC14` функциясында да біз сандық компьютерлік алгоритмдерде кеңінен тараған байқалмайтын қателікте тоқтамай өтіп кеттік. 2.2. Тараудан компьютерлер сандарды екілік кодта ақырғы нақтылықпен сақтайтынын еске түсірейік, бұл көзге көрінбейтін қателіктерге әкеледі, әсіресе жылжымалы нүктелі өте кішкентай сандар жағдайында. Алайда кішкентай қателік итеративті есептеуде қатты ұлғаюы мүмкін. `for` циклдеріне арналған `pond` және `desayC14` функция-

ларында егер біз t -ді dt -ді $step$ -ке көбейтіп алудың орнына, әр итерацияда dt -ні

```
t = t + dt көмегімен қосып,
```

t мәнін жинақтағанда осылай болар еді. Егер dt өте аз болса, онда dt -ді t -ге қосқан кезде ылғи кішкентай қателік шығып отыратын еді. Алайда егер цикл ұзақ уақыт ішінде қайталанған болса, онда t мәні одан сайын дәл болмайтын еді.

Бұл проблеманы суреттеу үшін, `decayC14` функциясында не болатынын

```
келесі код сегменті арқылы көрсетейік.
```

```
dt = 0.0001
```

```
iterations = 1000000
```

```
t = 0
```

```
for index in
```

```
    range(iterations): t = t
```

```
        + dt көмегімен қосып,
```

```
correct = dt * iterations      # дұрыс мән t
```

```
print(correct, t)
```

Цикл 0.0001 мәнін миллион рет жинақтайды, себебі дұрыс мән $t = 0.0001 \cdot 1,000,000 = 100$. Алайда, кодты іске қосқанда біз t ақырғы мәні 100.00000000219612 болатынын байқаймыз, дұрыс жауап алдындағы кішігірім бөлшек. Кейбір жерлерде тіпті осындай қателіктердің өзі едәуір ықпалы болуы мүмкін. Ал теріс ықпал итерацияның үлкен санымен ұлғая алады. Ғылыми есептеулер күндерге немесе апталарға созылады, ал қолданылатын итерациялар саны қателіктерді едәуір мөлшерге дейін ұлғайтуы мүмкін.

Ойлануға берілетін сұрақ 4.27 *Кодты 10 миллион және 100 миллион итерация үшін жоғарырақ іске қосыңыз. Қателік туралы не айтасыз?*

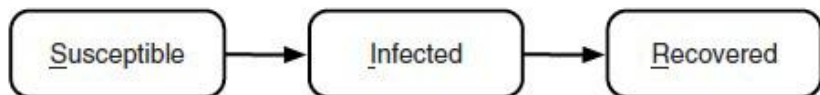
Осы сияқты қателікті болдырмау үшін, біз оның орнына $t = t + dt \cdot step$ туындысы мен ағымдағы $step$ итерациясының нөмірлерін бердік:

```
t = step * dt
```

Бұл жағдайда, t мәні бір арифметикалық операцияның жолымен шығарылады, бұл потенциалды қателікті азайтады.

Индетті үлгілеу

Шынайы популяциялардың өзара және қоршаған ортамен қарым-қатынасының үлгісі күрделі болады. Сондықтан, оларды нақты үлгілеу үшін айырымдық теңдеулердің өзара байланысқан жиынтығы қажет, олар айырымдық теңдеулер жүйесі деп аталады. 1927 жылы Кермак пен МакКендрик [23] SIR үлгі деп аталатын жұқпалардың таралу үлгісін ұсынды. «S» - «шалдыққыш» (susceptible) халық дегенді білдіреді – ауруға шалдығуы мүмкін; «I» - «жұқтырған» (infected) халық, ал «R» – «сауыққан» (recovered) халықты білдіреді. Бұл үлгіде біз сауыққаннан кейін тұлға ауруға иммунитет түзіп, ауруды қайта жұқтырмайды деп болжаймыз. Сонымен қатар, біз халықтың жалпы саны тұрақты және ешкім аурудан өлмейді деп есептейміз. Осылайша, тұлға «шалдыққыш» күйде «жұқтырған» күйде өтіп, одан кейін «сауыққан» күйге өтеді де төмендегі суретте көрсетілгендей сонда қалады.



Бұл мысалдар тұмау секілді кең тараған вирустық жұқпаларға жақсы келеді.

Тұмауға ұқсас вирус шалдыққыш адам қаншалықты көп жұқтырған адамдармен байланысқа түссе, соншалықты халыққа тез тарайды. Басқа сөзбен айтатын болсақ, ауру жұқтырған адамның саны қаншалықты көп болса шалдыққыш адам соншалықты ықтималдықпен вирусты жұқтырып алады. Сонымен қатар, популяцияның сомалық көлемі өзгермейтіндіктен, ауру жұқтырған адамдардың санының артуы шалдыққыштар санының оған тең болып азаюын білдіреді. Осыған ұқсас, жұқтырған адам санының қысқаруы сауыққандар санының оған тең болып ұлғаюын білдіреді. Көптеген табиғи үдерістер секілді аурудың тарауы бірқалыпты немесе үздіксіз сипатқа ие, сондықтан біз оның өзгерістерін радиобелсенді ыдырау үлгісінде жасағанымыздай, Δt кішігірім интервалдарында үлгілеуіміз қажет болады.

Бізге үш топ мөлшерінің өзгеруін сипаттайтын үш айырымдық теңдеу құру қажет. $S(t)$ ретінде t күніндегі шалдыққыш адам санын алайық, $I(t)$ ретінде – t күніндегі ауру жұқтырған адам санын, ал $R(t)$ ретінде – t күніндегі сауыққан адам санын алайық.

Сауыққан халық үшін айырымдық теңдеу өте аз қиындық тудырады. Сауыққан топ саны тек өседі; ауру жұқтырған адамдар сауыққан кезде, олар ауру жұқтырған топтан сауыққандар тобына ауысады. Әр қадамда

сауыққан адамның саны қазіргі уақытта ауру жұқтырғандарының санына және r сауығу жылдамдығына тәуелді: Күн сайын сауығатын адамдардың орташа үлесі.

Ойлануға берілетін сұрақ 4.28 *Шынайы ауру тараған кезде сауығу жылдамдығына қандай факторлар әсер етуі мүмкін?*

Біз әр күнді ұзақтығы Δt интервалдарға бөлетініміздіктен, онда әр интервал үшін шамалас азайтылған жылдамдық $r \cdot \Delta t$ қолдану керек болады. Осылайша, t күніндегі сауыққан топтың санын суреттейтін айырымдық теңдеу мынадай болмақ

0 күнінде бір де бір сауыққан адам жоқ болғандықтан, біз $R(0) = 0$ бастапқы шартты орнатамыз. Енді $S(t)$ үшін айырымдық теңдеуді қарастырайық. Шалдыққыш халықтың саны қайта ауру жұқтырған адамдардың санына ғана азаяды. Бұл азаю шалдыққыш адамдардың санына, олар байланысқа түсуі мүмкін ауру жұқтырған адамдардың санына, және бұл ықтимал өзара әрекеттесулер жаңа жұқтыруға әкелетін жылдамдығына тәуелді. Шалдыққыштар мен ауру жұқтырған тұлғалар арасындағы ықтимал өзара әрекеттесулер саны $t - \Delta t$ сәтінде – бұл олардың тек туындысы ғана: $S(t - \Delta t) \cdot I(t - \Delta t)$.

Егер біз d -ні *осы әрекеттесулерді жұқтыруға әкелетін жылдамдық* десек, онда біздің айырымдық теңдеу мынау болады

$$S(t) = S(t - \Delta t) - \underbrace{d \cdot S(t - \Delta t) \cdot I(t - \Delta t) \cdot \Delta t}_{\text{newly infected, to } I}.$$

Ойлануға берілетін сұрақ 4.29 Шынайы ауру тараған жағдайда аурудың тарау жылдамдығына қандай факторлар әсер етуі мүмкін?

Егер N – халқымыздың жалпы саны болса, онда бастапқы шарт $S(0) = N - 1$ болады, себебі біз бір ауру адамнан бастаймыз, бұл $N - 1$ шалдыққышты құрайды. Жұқтырылған топ үшін айырымдық теңдеу қазір ғана ауру жұқтырған адам (жаңа жұқтырылғандар) санына және жаңа ғана сауыққан адамдар (жаңа сауыққандар) санына тәуелді. Бұл сәйкесінше шалдыққыш топтан шыққан және сауыққандар тобына кірген адамдардың нақты саны. Біз оларды жоғарыда келтірілген теңдеулерден көшіріп аламыз.

$$I(t) = I(t - \Delta t) + \underbrace{d \cdot S(t - \Delta t) \cdot I(t - \Delta t) \cdot \Delta t}_{\text{newly infected, from } S} - \underbrace{r \cdot I(t - \Delta t) \cdot \Delta t}_{\text{newly recovered, to } R}.$$

Біз бір ауру жұқтырған адамнан бастайтын болғандықтан, $I(0) = 1$ ор-

натамыз.

Төмендегі бағдарлама - $r = 0.25$ қалпына келу жылдамдығында және $d = 0.0004$ аурудың тарау жылдамдығында осы үлгіні іске асыратын «қаңқа» болып табылады. Бұл көрсеткіштер орташа жұқтыру $1/r = 4$ күн жүретіндігін білдіреді, және шалдыққыш адам мен ауру жұқтырғанның кездесуі жаңа жұқтыруға әкелу ықтималдығы 0.04% тең. Бағдарлама сонымен бірге, бір суретте бірнеше айнымалыларды қалай құруға және белгілеуге болатынын және түсіндірме сөзді қалай қою керектігін көрсетеді. Біз циклдегі айырымдық теңдеулерді шешуді жаттығу ретінде қалдырамыз.

```
import matplotlib.pyplot as pyplot көмегімен импорттаймыз.
```

```
def SIR(population, days, dt):  
    """жұқпалы аурулардың SIR үлгісін құрайды  
    және уақыт ағынымен халықты топтарға бөлу  
    графигін жасайды.
```

Параметрлер:

population: Халық саны days:

Болжау үшін күн саны

dt: Үлгідегі "Delta t" мәні (күн бөлігі)

Қайтарылатын
мәндер: None"""

```
susceptible = population - 1 # шалдыққыштар есебі  
S(t) infected = 1.0          # жұқтырылғандар  
есебі I(t)  
recovered = 0.0               # сауыққандар есебі R(t)  
recRate = 0.25                # қалпына келу жылдамдығы r  
infRate = 0.0004              # аурудың тарау  
жылдамдығы d SList = [susceptible]  
IList = [infected]  
RList =  
[recovered]  
timeList = [0]
```

Мұнда айырымдық теңдеулер колданылатын цикл жүруде.

```
pyplot.plot(timeList, SList, label =
```

```

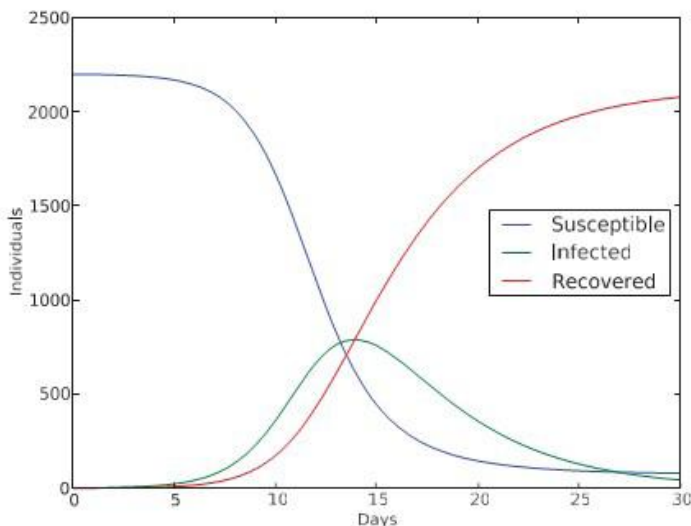
'Шалдыққыштар') pyplot.plot(timeList,
PList, label = 'Жұқтырғандар')
pyplot.plot(timeList, RList, label =
'Сауыққандар') pyplot.legend(loc = 'center
right') pyplot.xlabel('Күн')
pyplot.ylabel('Адам саны')
pyplot.show()

```

4.7-суретте 2,200 тұлға үшін (мүмкін шағын колледж студенттері шығар?) 30 күн ішіндегі үлгілеу нәтижесі көрсетілген. Біз аурудың өз шыңына шамамен 2 аптадан кейін жетіп, содан кейін біртіндеп азаятындығын байқаймыз. 30 күннен кейін вирус іс жүзінде жойылады да, шамамен тек 40 адам ғана жұқтырылған күйде қалады. Сонымен қатар 30 күннен кейін барлығының ауру жұқтырмағандығын ескерейік; 80 адамға жуығы сау қалды.

Ойлануға берілетін сұрақ 4.30 *4.7-суреттегі үш топтың саны мен уақыт ағынының өзара ара қатынасына қараңыз. Шалдыққыш және сауыққан топтарының саны екінші аптада жұқтырған топтың азаюына қалай әкеледі? Тағы қандай заңдылықтарды байқадыңыз?*

Ішкі өзара әрекеттесулерді ескеріп үлгіні іске асырғанда біз бастапқы итерациядағы топ санының негізінде әр топтың санының өзгеруін есептеу үшін өте мұқият болуымыз қажет,



4.7-сурет Ағымдағы итерацияда бұрынырақ есептелген топ сандары үшін емес, $\Delta t = 0.01$ мәніндегі 30 күн ішіндегі 2,200 тұлға үшін SIR үлгілеуінің нәтижесі. Сондықтан біз $\Delta R(t)$, $\Delta S(t)$ және $\Delta I(t)$ деп белгіленген t уақыты бойынша топ санындағы барлық өзгерістерді әрбір итерациядағы топ санын жаңартудан бұрын алдыңғы топ сандарына негізделе отырып есептеуіміз қажет. Басқа сөзбен айтқанда, әрбір итерацияда, біз алдымен мынаны есептейміз

$$\begin{aligned}\Delta R(t) &= r \cdot I(t - \Delta t) \cdot \Delta t, \\ \Delta S(t) &= d \cdot S(t - \Delta t) \cdot I(t - \Delta t) \cdot \Delta t \text{ және} \\ \Delta I(t) &= \Delta S(t) - \Delta R(t).\end{aligned}$$

Сосын бұл мәндерді топтың жаңа сандарын алу үшін қолданамыз:

$$\begin{aligned}R(t) &= R(t - \Delta t) + \Delta R(t), \\ S(t) &= S(t - \Delta t) - \Delta S(t) \text{ және} \\ I(t) &= I(t - \Delta t) + \Delta I(t).\end{aligned}$$

SIR үлгісін сонымен бірге идеяны, сән бағытын немесе халық арасына мемдерді таратуды үлгілеуге қолдануға болатынын есте сақтау қажет. «Жұқпалы» идея өз бастауын кішігірім адам тобынан алады да, уақыт өте келе басқаларға «жұғуы» мүмкін. Бұл идеяны қаншалықты адам қолдаса, соншалықты ол тез тарайды, себебі идеяны қабылдағандар мен қабылдамағандар арасында әлеуетті өзара әрекеттесулер көбейе бастайды. Алайда ақырында адамдар басқа нәрсеге назарын аударып, идеяны ұмыта бастайды да, осыдан «сауығу» орын алады.

Жаттығулар

Келесі тапсырмалардың әрқайсысына функцияны жазыңыз.

4.4.1. Бізде ұрғашы және аталығы бар екі қоян бар делік. Қояндардың ұрғашылары ересек жасқа жетіп, бір айға тола салысымен бірінші рет шағылысуға дайын болады, сондай ақ олар бір айлық мерзім бойы буаз болады. Ересек ұрғашы қоян ай сайын шағылысады және бір айдан кейін бір аталық және бір ұрғашыдан тұқым береді делік. Бір де бір қоян ешқашан өлмейді.

(а) Әр айдың соңындағы ұрғашы қояндардың санын беретін $R(m)$ айырымдық теңдеуін жазыңыз. (Мұнда Δt -ге тәуелділік жоқ, дискретті үлгі екендігіне назар аударыңыз.) Бірінші айдан бастаңыз да, заңдылықты көргенше бірнеше айды қатарынан есептеңіз. Бастапқы шарт(тар) туралы ұмытпаңыз. Жұмыс басы үшін:

- $R(0) = 1$, бастапқы жұп
- $R(1) = 1$, бір айдан кейінгі бастапқы жұп
- $R(2) = 2$, бастапқы жұп қосу жаңа туылған жұп
- $R(3) = 3$, бастапқы жұп қосу жаңа туылған жұп қосу 1 айлық жұп
- $R(4) = 5$, бастапқы буынның жұптары қосу екі жаңа туылған жұптар
- $R(5) = 8$, бастапқы жұп қосу жаңа туылған жұп қосу 2 айлық жұп
- $R(6) = 13$, бастапқы жұп қосу жаңа туылған жұп қосу 3 айлық жұп
- $R(7) = 21$, бастапқы жұп қосу жаңа туылған жұп қосу 4 айлық жұп
- $R(8) = 34$, бастапқы жұп қосу жаңа туылған жұп қосу 5 айлық жұп
- $R(9) = 55$, бастапқы жұп қосу жаңа туылған жұп қосу 6 айлық жұп
- $R(10) = 89$, бастапқы жұп қосу жаңа туылған жұп қосу 7 айлық жұп
- $R(11) = 144$, бастапқы жұп қосу жаңа туылған жұп қосу 8 айлық жұп
- $R(12) = 233$, бастапқы жұп қосу жаңа туылған жұп қосу 9 айлық жұп
- $R(13) = 377$, бастапқы жұп қосу жаңа туылған жұп қосу 10 айлық жұп
- $R(14) = 610$, бастапқы жұп қосу жаңа туылған жұп қосу 11 айлық жұп
- $R(15) = 987$, бастапқы жұп қосу жаңа туылған жұп қосу 12 айлық жұп
- $R(16) = 1597$, бастапқы жұп қосу жаңа туылған жұп қосу 13 айлық жұп
- $R(17) = 2584$, бастапқы жұп қосу жаңа туылған жұп қосу 14 айлық жұп
- $R(18) = 4181$, бастапқы жұп қосу жаңа туылған жұп қосу 15 айлық жұп
- $R(19) = 6765$, бастапқы жұп қосу жаңа туылған жұп қосу 16 айлық жұп
- $R(20) = 10946$, бастапқы жұп қосу жаңа туылған жұп қосу 17 айлық жұп
- $R(21) = 17711$, бастапқы жұп қосу жаңа туылған жұп қосу 18 айлық жұп
- $R(22) = 28657$, бастапқы жұп қосу жаңа туылған жұп қосу 19 айлық жұп
- $R(23) = 46368$, бастапқы жұп қосу жаңа туылған жұп қосу 20 айлық жұп
- $R(24) = 75025$, бастапқы жұп қосу жаңа туылған жұп қосу 21 айлық жұп
- $R(25) = 121393$, бастапқы жұп қосу жаңа туылған жұп қосу 22 айлық жұп
- $R(26) = 196418$, бастапқы жұп қосу жаңа туылған жұп қосу 23 айлық жұп
- $R(27) = 317811$, бастапқы жұп қосу жаңа туылған жұп қосу 24 айлық жұп
- $R(28) = 514229$, бастапқы жұп қосу жаңа туылған жұп қосу 25 айлық жұп
- $R(29) = 821440$, бастапқы жұп қосу жаңа туылған жұп қосу 26 айлық жұп
- $R(30) = 1317601$, бастапқы жұп қосу жаңа туылған жұп қосу 27 айлық жұп
- $R(31) = 2107522$, бастапқы жұп қосу жаңа туылған жұп қосу 28 айлық жұп
- $R(32) = 3395468$, бастапқы жұп қосу жаңа туылған жұп қосу 29 айлық жұп
- $R(33) = 5497559$, бастапқы жұп қосу жаңа туылған жұп қосу 30 айлық жұп
- $R(34) = 8814736$, бастапқы жұп қосу жаңа туылған жұп қосу 31 айлық жұп
- $R(35) = 14049695$, бастапқы жұп қосу жаңа туылған жұп қосу 32 айлық жұп
- $R(36) = 22585219$, бастапқы жұп қосу жаңа туылған жұп қосу 33 айлық жұп
- $R(37) = 36413918$, бастапқы жұп қосу жаңа туылған жұп қосу 34 айлық жұп
- $R(38) = 58998147$, бастапқы жұп қосу жаңа туылған жұп қосу 35 айлық жұп
- $R(39) = 94406166$, бастапқы жұп қосу жаңа туылған жұп қосу 36 айлық жұп
- $R(40) = 151416075$, бастапқы жұп қосу жаңа туылған жұп қосу 37 айлық жұп
- $R(41) = 244832140$, бастапқы жұп қосу жаңа туылған жұп қосу 38 айлық жұп
- $R(42) = 392248215$, бастапқы жұп қосу жаңа туылған жұп қосу 39 айлық жұп
- $R(43) = 627080355$, бастапқы жұп қосу жаңа туылған жұп қосу 40 айлық жұп
- $R(44) = 1006633070$, бастапқы жұп қосу жаңа туылған жұп қосу 41 айлық жұп
- $R(45) = 1614005715$, бастапқы жұп қосу жаңа туылған жұп қосу 42 айлық жұп
- $R(46) = 2590638785$, бастапқы жұп қосу жаңа туылған жұп қосу 43 айлық жұп
- $R(47) = 4104644500$, бастапқы жұп қосу жаңа туылған жұп қосу 44 айлық жұп
- $R(48) = 6559283285$, бастапқы жұп қосу жаңа туылған жұп қосу 45 айлық жұп
- $R(49) = 10463927790$, бастапқы жұп қосу жаңа туылған жұп қосу 46 айлық жұп
- $R(50) = 16640609075$, бастапқы жұп қосу жаңа туылған жұп қосу 47 айлық жұп
- $R(51) = 26550526865$, бастапқы жұп қосу жаңа туылған жұп қосу 48 айлық жұп
- $R(52) = 41491135940$, бастапқы жұп қосу жаңа туылған жұп қосу 49 айлық жұп
- $R(53) = 65541662805$, бастапқы жұп қосу жаңа туылған жұп қосу 50 айлық жұп
- $R(54) = 102573800745$, бастапқы жұп қосу жаңа туылған жұп қосу 51 айлық жұп
- $R(55) = 160665463550$, бастапқы жұп қосу жаңа туылған жұп қосу 52 айлық жұп
- $R(56) = 251814064295$, бастапқы жұп қосу жаңа туылған жұп қосу 53 айлық жұп
- $R(57) = 394389527845$, бастапқы жұп қосу жаңа туылған жұп қосу 54 айлық жұп
- $R(58) = 606203592140$, бастапқы жұп қосу жаңа туылған жұп қосу 55 айлық жұп
- $R(59) = 930593119985$, бастапқы жұп қосу жаңа туылған жұп қосу 56 айлық жұп
- $R(60) = 1436796712125$, бастапқы жұп қосу жаңа туылған жұп қосу 57 айлық жұп
- $R(61) = 2217390832110$, бастапқы жұп қосу жаңа туылған жұп қосу 58 айлық жұп
- $R(62) = 3434187544235$, бастапқы жұп қосу жаңа туылған жұп қосу 59 айлық жұп
- $R(63) = 5251578376355$, бастапқы жұп қосу жаңа туылған жұп қосу 60 айлық жұп
- $R(64) = 8085766020590$, бастапқы жұп қосу жаңа туылған жұп қосу 61 айлық жұп
- $R(65) = 12337344396945$, бастапқы жұп қосу жаңа туылған жұп қосу 62 айлық жұп
- $R(66) = 18723110417535$, бастапқы жұп қосу жаңа туылған жұп қосу 63 айлық жұп
- $R(67) = 28360454814480$, бастапқы жұп қосу жаңа туылған жұп қосу 64 айлық жұп
- $R(68) = 43083565232015$, бастапқы жұп қосу жаңа туылған жұп қосу 65 айлық жұп
- $R(69) = 65444019946495$, бастапқы жұп қосу жаңа туылған жұп қосу 66 айлық жұп
- $R(70) = 99527585178510$, бастапқы жұп қосу жаңа туылған жұп қосу 67 айлық жұп
- $R(71) = 150971605124995$, бастапқы жұп қосу жаңа туылған жұп қосу 68 айлық жұп
- $R(72) = 228499190303505$, бастапқы жұп қосу жаңа туылған жұп қосу 69 айлық жұп
- $R(73) = 348470795428495$, бастапқы жұп қосу жаңа туылған жұп қосу 70 айлық жұп
- $R(74) = 530969985732000$, бастапқы жұп қосу жаңа туылған жұп қосу 71 айлық жұп
- $R(75) = 804440781160495$, бастапқы жұп қосу жаңа туылған жұп қосу 72 айлық жұп
- $R(76) = 1215410766892500$, бастапқы жұп қосу жаңа туылған жұп қосу 73 айлық жұп
- $R(77) = 1836370748052995$, бастапқы жұп қосу жаңа туылған жұп қосу 74 айлық жұп
- $R(78) = 2771781514945495$, бастапқы жұп қосу жаңа туылған жұп қосу 75 айлық жұп
- $R(79) = 4178152263008490$, бастапқы жұп қосу жаңа туылған жұп қосу 76 айлық жұп
- $R(80) = 6324933780953985$, бастапқы жұп қосу жаңа туылған жұп қосу 77 айлық жұп
- $R(81) = 9503086043962475$, бастапқы жұп қосу жаңа туылған жұп қосу 78 айлық жұп
- $R(82) = 14228019824916460$, бастапқы жұп қосу жаңа туылған жұп қосу 79 айлық жұп
- $R(83) = 21631105868878935$, бастапқы жұп қосу жаңа туылған жұп қосу 80 айлық жұп
- $R(84) = 32859125693795400$, бастапқы жұп қосу жаңа туылған жұп қосу 81 айлық жұп
- $R(85) = 50090231562674335$, бастапқы жұп қосу жаңа туылған жұп қосу 82 айлық жұп
- $R(86) = 75149357256470740$, бастапқы жұп қосу жаңа туылған жұп қосу 83 айлық жұп
- $R(87) = 112239588819145075$, бастапқы жұп қосу жаңа туылған жұп қосу 84 айлық жұп
- $R(88) = 170388946075615815$, бастапқы жұп қосу жаңа туылған жұп қосу 85 айлық жұп
- $R(89) = 257628534894760890$, бастапқы жұп қосу жаңа туылған жұп қосу 86 айлық жұп
- $R(90) = 390017480970376705$, бастапқы жұп қосу жаңа туылған жұп қосу 87 айлық жұп
- $R(91) = 586645915865137595$, бастапқы жұп қосу жаңа туылған жұп қосу 88 айлық жұп
- $R(92) = 881663406835514300$, бастапқы жұп қосу жаңа туылған жұп қосу 89 айлық жұп
- $R(93) = 1321680322700651895$, бастапқы жұп қосу жаңа туылған жұп қосу 90 айлық жұп
- $R(94) = 2003343729536166200$, бастапқы жұп қосу жаңа туылған жұп қосу 91 айлық жұп
- $R(95) = 3005024052236818095$, бастапқы жұп қосу жаңа туылған жұп қосу 92 айлық жұп
- $R(96) = 4508367781772984300$, бастапқы жұп қосу жаңа туылған жұп қосу 93 айлық жұп
- $R(97) = 6763391834009792395$, бастапқы жұп қосу жаңа туылған жұп қосу 94 айлық жұп
- $R(98) = 10171759615782776700$, бастапқы жұп қосу жаңа туылған жұп қосу 95 айлық жұп
- $R(99) = 15335151450792569095$, бастапқы жұп қосу жаңа туылған жұп қосу 96 айлық жұп
- $R(100) = 23006911066575345800$, бастапқы жұп қосу жаңа туылған жұп қосу 97 айлық жұп
- $R(101) = 34642062517367914895$, бастапқы жұп қосу жаңа туылған жұп қосу 98 айлық жұп
- $R(102) = 52148973583943260700$, бастапқы жұп қосу жаңа туылған жұп қосу 99 айлық жұп
- $R(103) = 78391036101311175595$, бастапқы жұп қосу жаңа туылған жұп қосу 100 айлық жұп
- $R(104) = 117539009685254436300$, бастапқы жұп қосу жаңа туылған жұп қосу 101 айлық жұп
- $R(105) = 176687982786565611895$, бастапқы жұп қосу жаңа туылған жұп қосу 102 айлық жұп
- $R(106) = 266226992471820048200$, бастапқы жұп қосу жаңа туылған жұп қосу 103 айлық жұп
- $R(107) = 400914975258385659995$, бастапқы жұп қосу жаңа туылған жұп қосу 104 айлық жұп
- $R(108) = 601141967730205708200$, бастапқы жұп қосу жаңа туылған жұп қосу 105 айлық жұп
- $R(109) = 897056942988591368195$, бастапқы жұп қосу жаңа туылған жұп қосу 106 айлық жұп
- $R(110) = 1348198910718797076400$, бастапқы жұп қосу жаңа туылған жұп қосу 107 айлық жұп
- $R(111) = 2025255853707388444595$, бастапқы жұп қосу жаңа туылған жұп қосу 108 айлық жұп
- $R(112) = 3023454764426185521000$, бастапқы жұп қосу жаңа туылған жұп қосу 109 айлық жұп
- $R(113) = 4521709618133573965595$, бастапқы жұп қосу жаңа туылған жұп қосу 110 айлық жұп
- $R(114) = 6765164382560759486600$, бастапқы жұп қосу жаңа туылған жұп қосу 111 айлық жұп
- $R(115) = 10136874000694333452195$, бастапқы жұп қосу жаңа туылған жұп қосу 112 айлық жұп
- $R(116) = 15102038383255092938800$, бастапқы жұп қосу жаңа туылған жұп қосу 113 айлық жұп
- $R(117) = 22538912383949426390995$, бастапқы жұп қосу жаңа туылған жұп қосу 114 айлық жұп
- $R(118) = 34040950767204519329800$, бастапқы жұп қосу жаңа туылған жұп қосу 115 айлық жұп
- $R(119) = 51079863150454045720795$, бастапқы жұп қосу жаңа туылған жұп қосу 116 айлық жұп
- $R(120) = 76120813917658565050600$, бастапқы жұп қосу жаңа туылған жұп қосу 117 айлық жұп
- $R(121) = 113200677068112610771395$, бастапқы жұп қосу жаңа туылған жұп қосу 118 айлық жұп
- $R(122) = 170321490985771175822000$, бастапқы жұп қосу жаңа туылған жұп қосу 119 айлық жұп
- $R(123) = 256522168053883786593395$, бастапқы жұп қосу жаңа туылған жұп қосу 120 айлық жұп
- $R(124) = 386843659039654962415400$, бастапқы жұп қосу жаңа туылған жұп қосу 121 айлық жұп
- $R(125) = 580365827093538749008795$, бастапқы жұп қосу жаңа туылған жұп қосу 122 айлық жұп
- $R(126) = 867209486133293711424200$, бастапқы жұп қосу жаңа туылған жұп қосу 123 айлық жұп
- $R(127) = 1290575313226832460432995$, бастапқы жұп қосу жаңа туылған жұп қосу 124 айлық жұп
- $R(128) = 1937784799360126171857200$, бастапқы жұп қосу жаңа туылған жұп қосу 125 айлық жұп
- $R(129) = 2878360112586958632289195$, бастапқы жұп қосу жаңа туылған жұп қосу 126 айлық жұп
- $R(130) = 4316144911947084804146400$, бастапқы жұп қосу жаңа туылған жұп қосу 127 айлық жұп
- $R(131) = 6454505024534043436435595$, бастапқы жұп қосу жаңа туылған жұп қосу 128 айлық жұп
- $R(132) = 9602649936481128240582000$, бастапқы жұп қосу жаңа туылған жұп қосу 129 айлық жұп
- $R(133) = 14257154961015171676017995$, бастапқы жұп қосу жаңа туылған жұп қосу 130 айлық жұп
- $R(134) = 21459804897506300916600000$, бастапқы жұп қосу жаңа туылған жұп қосу 131 айлық жұп
- $R(135) = 32316959858521472592617995$, бастапқы жұп қосу жаңа туылған жұп қосу 132 айлық жұп
- $R(136) = 48476764756027773509218000$, бастапқы жұп қосу жаңа туылған жұп қосу 133 айлық жұп
- $R(137) = 72193724614549246001835995$, бастапқы жұп қосу жаңа туылған жұп қосу 134 айлық жұп
- $R(138) = 107670489370577019511054000$, бастапқы жұп қосу жаңа туылған жұп қосу 135 айлық жұп
- $R(139) = 160864213985126265512889995$, бастапқы жұп қосу жаңа туылған жұп қосу 136 айлық жұп
- $R(140) = 240534703355703285023944000$, бастапқы жұп қосу жаңа туылған жұп қосу 137 айлық жұп
- $R(141) = 357398917340829550536833995$, бастапқы жұп қосу жаңа туылған жұп қосу 138 айлық жұп
- $R(142) = 531933620696532835560778000$, бастапқы жұп қосу жаңа туылған жұп қосу 139 айлық жұп
- $R(143) = 794332538042362386097611995$, бастапқы жұп қосу жаңа туылған жұп қосу 140 айлық жұп
- $R(144) = 1175666158738895221658392000$, бастапқы жұп қосу жаңа туылған жұп қосу 141 айлық жұп
- $R(145) = 1740002316781257607756003995$, бастапқы жұп қосу жаңа туылған жұп қосу 142 айлық жұп
- $R(146) = 2585668475519152829414404000$, бастапқы жұп қосу жаңа туылған жұп қосу 143 айлық жұп
- $R(147) = 3855670792300410437170403995$, бастапқы жұп қосу жаңа туылған жұп қосу 144 айлық жұп
- $R(148) = 5741339267819563266584804000$, бастапқы жұп қосу жаңа туылған жұп қосу 145 айлық жұп
- $R(149) = 8556909060120973703755203995$, бастапқы жұп қосу жаңа туылған жұп қосу 146 айлық жұп
- $R(150) = 12703248327940537069339604000$, бастапқы жұп қосу жаңа туылған жұп қосу 147 айлық жұп
- $R(151) = 18859157388061510773094803995$, бастапқы жұп қосу жаңа туылған жұп қосу 148 айлық жұп
- $R(152) = 28012405716002047842434404000$, бастапқы жұп қосу жаңа туылған жұп қосу 149 айлық жұп
- $R(153) = 41624563032063558615529203995$, бастапқы жұп қосу жаңа туылған жұп қосу 150 айлық жұп
- $R(154) = 61636968748125606457963604000$, бастапқы жұп қосу жаңа туылған жұп қосу 151 айлық жұп
- $R(155) = 91261531780189165073492803995$, бастапқы жұп қосу жаңа туылған жұп қосу 152 айлық жұп
- $R(156) = 135898490528314771531456404000$, бастапқы жұп қосу жаңа туылған жұп қосу 153 айлық жұп
- $R(157) = 203160022308493936604949203995$, бастапқы жұп қосу жаңа туылған жұп қосу 154 айлық жұп
- $R(158) = 300058512836808708136405604000$, бастапқы жұп қосу жаңа туылған жұп қосу 155 айлық жұп
- $R(159) = 443218535145202644741354803995$, бастапқы жұп қосу жаңа туылған жұп қосу 156 айлық жұп
- $R(160) = 661277047982011352877760404000$, бастапқы жұп қосу жаңа туылған жұп қосу 157 айлық жұп
- $R(161) = 982495583127214097619115203995$, бастапқы жұп қосу жаңа туылған жұп қосу 158 айлық жұп
- $R(162) = 1443772630109225450496875604000$, бастапқы жұп қосу жаңа туылған жұп қосу 159 айлық жұп
- $R(163) = 2156268213236439548115990603995$, бастапқы жұп қосу жаңа туылған жұп қосу 160 айлық жұп
- $R(164) = 3220040843345664998612865604000$, бастапқы жұп қосу жаңа туылған жұп қосу 161 айлық жұп
- $R(165) = 4816309056582104546728855603995$, бастапқы жұп қосу жаңа туылған жұп қосу 162 айлық жұп
- $R(166) = 7136349900927769545341720604000$, бастапқы жұп қосу жаңа туылған жұп қосу 163 айлық жұп
- $R(167) = 10602658957510874092060575603995$, бастапқы жұп қосу жаңа туылған жұп қосу 164 айлық жұп
- $R(168) = 15839008858438643637402300604000$, бастапқы жұп қосу жаңа туылған жұп қосу 165 айлық жұп
- $R(169) = 23441667815949517729462875603995$, бастапқы жұп қосу жаңа туылған жұп қосу 166 айлық жұп
- $R(170) = 34680676674388161366865175604000$, бастапқы жұп қосу жаңа туылған жұп қосу 167 айлық жұп
- $R(171) = 51562344490337679096327925603995$, бастапқы жұп қосу жаңа туылған жұп қосу 168 айлық жұп
- $R(172) = 76243021164725840463193000604000$, бастапқы жұп қосу жаңа туылған жұп қосу 169 айлық жұп
- $R(173) = 112705365655063519559520925603995$, бастапқы жұп қосу жаңа туылған жұп қосу 170 айлық жұп
- $R(174) = 168948386819789360022713925604000$, бастапқы жұп қосу жаңа туылған жұп қосу 171 айлық жұп
- $R(175) = 250893752474852879582234875603995$, бастапқы жұп қосу жаңа туылған жұп қосу 172 айлық жұп
- $R(176) = 374842139294642240604948875604000$, бастапқы жұп қосу жаңа туылған жұп қосу 173 айлық жұп
- $R(177) = 55473589176949512018718375603995$, бастапқы жұп қосу жаңа туылған жұп қосу 174 айлық жұп
- $R(178) = 82557803106413736079213260604000$, бастапқы жұп қосу жаңа туылған жұп қосу 175 айлық жұп
- $R(179) = 122531392283363248097931635603995$, бастапқы жұп қосу жаңа туылған жұп қосу 176 айлық жұп
- $R(180) = 181589195389777084177144895604000$, бастапқы жұп қосу жаңа туылған жұп қосу 177 айлық жұп
- $R(181) = 268120587673140332275076495603995$, бастапқы жұп қосу жаңа туылған жұп қосу 178 айлық жұп
- $R(182) = 395240783062917416452221395604000$, бастапқы жұп қосу жаңа туылған жұп қосу 179 айлық жұп
- $R(183) = 584961370736057748727297895603995$, бастапқы жұп қосу жаңа туылған жұп қосу 180 айлық жұп
- $R(184) = 867201553809075165179519395604000$, бастапқы жұп қосу жаңа туылған жұп қосу 181 айлық жұп
-

4.4.2. 4.1.37 жаттығуын қарайық, бірақ енді бактерия колониясы Δt сағатының азғантай бөлігінде $0.1\Delta t$ жылдамдықпен үздіксіз өседі делік.

(a) Бактерия колониясының көлемі $B(t)$ үшін

Үздіксіз өсу үлгісінің аппроксимациясы үшін Δt қолданып айырымдық теңдеу құрыңыз.

(b) Функцияны жазыңыз

bacteria(population, dt, days)

Ол (a) бөлігіндегі сіздің айырымдық теңдеуіңізді белгіленген күндер санынан days кейін колониядағы бактериялар санын есептеу және қайтару үшін қолданады. Қалған параметрлер, population және dt – колонияның бастапқы көлемі және сәйкесінше (Δt) қадам көлемі. Функцияңыз сонымен бірге matplotlib-ті қолданып популяцияның өсу графигін құруы керек.

4.4.3. Көміртек-14 секілді радиобелсенді элементтер, әдеттегідей, элемент бастапқы мөлшерінен жартысына дейін ыдырайтын уақыт - жартылай ыдырау кезеңімен сипатталады.

Функцияны жазыңыз

halflifeC14(originalAmount, dt)

Ол көміртек-14-тің жартылай ыдырау кезеңін анықтайды. Сіздің функцияңыз радиобелсенді ыдырауды мөлшер originalAmount жартысына дейін жеткенше келтіруі керек.

Көміртек-14-тің жартылай ыдырау кезеңі $5,730 \pm 40$ жылды құрайтыны белгілі. Бұл аппроксимация қаншалықты жақын? dt әртүрлі мәндері үшін байқап көріңіз.

4.4.4. Көміртек-14-тің жартылай ыдырау кезеңін анықтау - радиобелсенді талдаудың жеке жағдайы болып табылады. Органикалық артефактінің жасын радиокөміртекті талдаған кезде, біздің заманымызға дейін сақталған көміртек-14-тің үлесі атмосферадағы құрамына сәйкес екендігі анықталады. Көне пергамент үзіндісінде 70% мөлшерде үлес табылды делік. Онда артефактінің жасын анықтау үшін halflifeC14 функциясының жалпыламасын қолдануға болады, ол $amount > originalAmount * 0.7$ болғанша қайталанады. Жана функция құрып, алдыңғы жаттығудан halflifeC14 функциясын қалай жалпылауға болатынын ұсыныңыз.

carbonDate(originalAmount, fractionRemaining, dt)

Ол қалған көміртек-14-тің белгіленген үлесімен бірге артефактінің жасын қайтарады. Бұл функцияны пергаменттің болжалды жасын анықтау үшін қолданыңыз.

4.4.5. SIR үлгілеуін іске асыруды тоқтатыңыз. Нәтижелеріңізді 4.7-суретпен салыстырып, жұмысыңызды тексеріңіз.

4.4.6. 4.7-суретте көрсетілгеннен ұзағырақ уақыт кезеңі үшін SIR үлгілерін іске асыруды қосыңыз. Жеткілікті уақыт болса барлығы ауру жұқтыра ма?

4.4.7. Адамдардың көп саны екпе алып, аурудың тарау жылдамдығы екі есе азайды делік. Бұндай екпенің әсері қандай болады?

4.4.8. SIS үлгісі иммунитет түзбейтін үлгі болып табылады. Басқа сөзбен айтқанда, «сауыққан» халық жоқ; сауыққан адамдар қайта «шалдыққыш» топқа өтеді.

(a) Бұндай үлгіге арнап айырымдық теңдеу құрыңыз.

(b) SIR функциясын (SIS-ке ауыстырып) көшіріп алып, сіздің айырымдық теңдеулеріңізді орындайтындай етіп өзгертіңіз.

(c) SIR үлгісіне қолданған параметрлері бар функцияны іске қосыңыз. Нені байқадыңыз? Нәтижелерді түсіндіріңіз.

4.4.9. Бір олжа үшін бәсекелесетін, алайда бір біріне тікелей зиян келтірмейтін екі жыртқыш түрі бар делік. Егер олжа қоры төмен болып, ал бір түрдің мүшелері мықтырақ аңшы болғанда, бұл екінші түрдің денсаулығына үлкен зиянын келтіретін еді деп болжай аламыз. Бұны келесі айырымдық теңдеулер жұбы арқылы үлгілеуге болады. $P(t)$ және $Q(t)$ t уақытында сәйкесінше 1 және 2 жыртқыш түрлерінің популяциясы болып табылады.

$$\begin{aligned}P(t) &= P(t - \Delta t) + bP \cdot P(t - \Delta t) \cdot \Delta t - dP \cdot P(t - \Delta t) \cdot Q(t - \Delta t) \cdot \Delta t \\Q(t) &= Q(t - \Delta t) + bQ \cdot Q(t - \Delta t) \cdot \Delta t - dQ \cdot P(t - \Delta t) \cdot Q(t - \Delta t) \cdot \Delta t\end{aligned}$$

$P(0) = p$ және $Q(0) = q$ бастапқы шарттар болады, мұндағы p және q - сәйкесінше бірінші және екінші жыртқыш түрлерінің популяция саны болып табылады. bP және dP мәндері бірінші жыртқыш түрінің туу және өлім көрсеткіштері (немесе, ресми тілде, пропорционалдық коэффициент), ал bQ және dQ мәндері екінші жыртқыш түрінің туу және өлім көрсеткіштері болып табылады. Бірінші теңдеудегі $bP \cdot P(t - \Delta t)$ құрамдас бөлігі жыртқыштың бірінші түрінің бір айдағы таза туу санын, Ал $dP \cdot P(t - \Delta t) \cdot Q(t - \Delta t)$ құрамдас бөлігі бір айдағы өлім санын білдіреді. Бұл құрамдас бөлік екі популяцияның да санына байланысты екеніне көңіл бөліңіз: $P(t - \Delta t) \cdot Q(t - \Delta t)$ - дегеніміз жеке аңдардың тағам үшін ықтимал (тікелей емес) күрестерінің саны, ал dP - осы күрестің бірі бірінші жыртқыш түрінің өліміне (аштықтан) әкелетін ықтималдық.

Бұндай үлгі түрі жанама, бейтарап бәсекелестік ретінде мәлім, себебі

бұл екі жыртқыш түрі бір бірімен тікелей түрде жарыспайды (яғни бір бірін жемейді), алайда олар ортақ тағам қоры үшін бәсекелеседі.

Функцияны жазыңыз

```
compete(pop1, pop2, birth1, birth2, death1, death2, years, dt)
```

Ол бұл үлгіні уақыт ағынымен популяция санының өзгеру графигін құрып, жанама бәсекелестіктің жалпыланған сценарии үшін іске асырады. Бағдарламаңызды

- $p = 21$ және $q = 26$

- $bP = 1.0$ және $dP = 0.2$; $bQ = 1.02$ және $dQ = 0.25$

- $dt = 0.001$; 6 жыл мәндерін қолданып іске қосыңыз,

және нәтижелерді түсіндіріңіз. Мұнда жұмысты бастауға арналған функцияның "қаңқасы" берілген.

```
def compete(pop1, pop2, birth1, birth2, death1, death2, years, dt): """"  
МҰНДА СІЗДІҢ ҚҰЖАТТАУ ЖОЛДАРЫҢЫЗ ОРНАЛАСҚАН""""
```

```
pop1List = [pop1]
```

```
pop2List = [pop2] timeList = [0]
```

```
for step in range(1, int(years / dt) + 1):
```

```
# МҰНДА СІЗДІҢ КОДЫҢЫЗ ЖҮРЕДІ
```

```
pyplot.plot(timeList, pop1List, label = 'Популяция 1')
```

```
pyplot.plot(timeList, pop2List, label = 'Популяция 2')
```

```
pyplot.legend()
```

```
pyplot.xlabel('Лет')
```

```
pyplot.ylabel('Түрлер')
```

```
pyplot.show()
```

*4.5 ТАЛДАУДЫҢ САНДЫҚ ӘДІСТЕРІ

Жинақтаушы алгоритмдер сонымен қатар жаратылыстану және қоғамдық ғылым салаларында қолданылады, бұл математикамен шешіле алмайтын жалпы математикалық константа мәндерін аппроксимациялау және мәндерді сандық түрде есептеуге және күрделі функциялардың түбірлерін шешуге қажет. Осы тарауда біз сандық талдау деп аталатын осы математика және информатика саласынан бірнеше салыстырмалы түрде қарапайым мысалдарын талқылаймыз.

Үйлесімді қатар

Бізде 1-метрлік резеңке арқанның бір ұшынан жүріп бастайтын құмырсқа бар делік. Бірінші минуттың ішінде құмырсқа 10 см жүріп өтеді. Бірінші минут өткен соң біз резеңке арқанды біркелкі етіп 1 метрге созамыз, осылайша ол 2 метрге ұзартылады. Келесі минуттың ішінде құмырсқа тағы 10 см жүріп өтеді, ал содан кейін біз арқанды тағы 1 метрге созамыз. Егер біз бұл үдерісті шексіз қайталайтын болсақ, құмырсқа арқанның соңына жете ала ма? Егер жететін болса, бұған қанша уақыт кетеді?

Жауабы құмырсқа минут сайын арқанның қай бөлігін жүріп өтетіндігінде жатыр. Бірінші минут ішінде құмырсқа арқанның соңына дейінгі $1/10$ аралықты жүріп өтеді. Созғаннан кейін құмырсқа бұрынғыдай қашықтықтың $1/10$ өтті, себебі құмырсқа жүріп өткен арқан бөлігі басқа бөлігімен бірге екі еселенді. Алайда екінші минутта құмырсқаның 10 см серуені жалпы қашықтықтың

тек $10/200 = 1/20$ қамтиды. Осылайша 2 минуттан кейін құмырсқа арқан бөлігінің

$1/10 + 1/20$ өтті. Үшінші минутта, арқанның ұзындығы 3 м, сондықтан құмырсқа қашықтықтың

тек $10/300 = 1/30$ бөлігін жүріп өтеді. Бұл үрдіс жалғасатындықтан, біздің тапсырма келесі сома 1 жете ме екендігіне саяды.

$$\frac{1}{10} + \frac{1}{20} + \frac{1}{30} + \dots = \frac{1}{10} \left(1 + \frac{1}{2} + \frac{1}{3} + \dots \right)$$

Әрине, біз бұл сұраққа сумматорды қолданып жауап бере аламыз. Бірақ бұл бөлшекенген мүшелерді қалай қосамыз? 4.1.25 жаттығуында сіз $1 + 2 + 3 + \dots + n$ есептедіңіз:

```
def sum(n):  
    total = 0  
    for number in range(1, n  
        + 1): total = total +  
        number  
    return total
```

Бұл for циклінің әр итерациясында біз total сумматорының ауыспалы шамасына number мәнін қосамыз. Себебі number -ге 1, 2, 3, ... мәндері

беріледі, n, total циклінің орындалғанынан кейін осы мәндер сомасын білдіреді. Құмырсқа жүріп өтетін арқанның бөлігін есептеу үшін , біз функцияны, оның орнына әр итерацияда $1 / \text{number}$ қосылатындай етіп өзгерте аламыз, содан кейін нәтижені $1/10$ көбейтеміз:

```
def ant(n):
```

```
    """резенке арқандағы құмырсқа” тапсырмасын  
    үлгілейді.
```

```
  
    Арқанның бастапқы ұзындығы 1 м және құмырсқа әр  
    минутта 10 см жүріп өтеді.
```

```
    Параметр:
```

```
        n: Құмырсқа жол жүретін минуттардағы уақыт
```

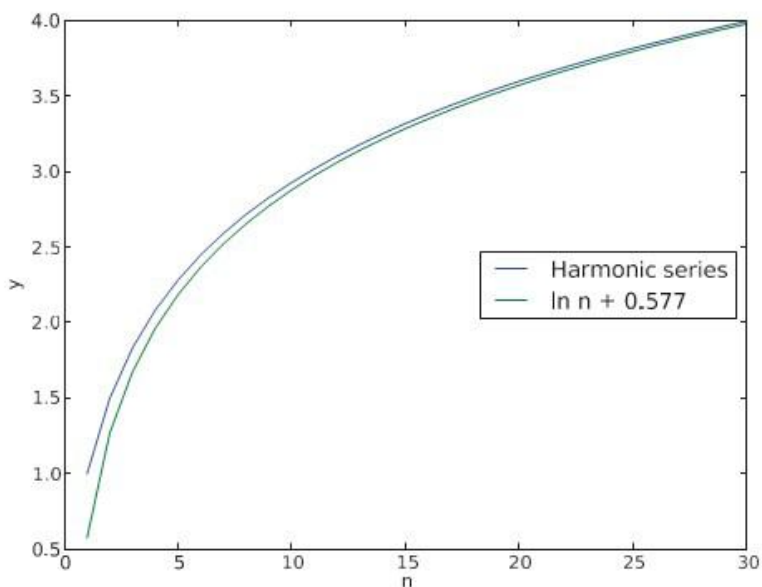
```
    Қайтарылатын мәндер:
```

```
        Құмырсқа n минутта жүріп өтетін арқан бөлігі  
        """
```

```
    total = 0
```

```
    for number in range(1, n +  
        1): total = total + (1 /  
            number)
```

```
    return total * 0.1
```



4.8-сурет Натуралды логарифмнің үйлесімді қатарымен аппроксимациялау ($\ln$).

Осы функция арқылы сұрағымызға жауап беру үшін, бізге сома 1-ден асатын өте жоғары мәнді таба аламыз ба екендігін көру үшін бірнеше n мәндерін байқап көру керек. Егер біз мұндай мәнді тапсақ, онда бізге сома 1-ге алғашқы рет жететін немесе асатын n шамасын тапқанша аз мәндерді тексеруіміз керек.

Ойлануға берілетін сұрақ 4.31 *ant* функциясын тек 5 рет шақырып шартты қанағаттандыратын минут диапазонын табыңыз.

Мысалы, $\text{ant}(100)$ шамамен 0.52 қайтарады, $\text{ant}(1000)$ шамамен 0.75 қайтарады, $\text{ant}(10000)$ шамамен 0.98 және $\text{ant}(15000)$ шамамен 1.02 қайтарады. Осылайша құмырсқа арқанның соңына 10,000–15,000 минуттан кейін жетеді. Минуттың нақты санын дәл сол әдіспен іздеуді жалғастыру өте жағымсыз болар еді.

Ойлануға берілетін сұрақ 4.32 *Құмырсқа арқанның соңына бірінші*

рет жететін немесе өтетін кездің мәнін іздеу үшін функцияны қалай құратын едіңіз? (Сыбырсөз: Бұл -14 көміртегінің жартылай ыдырау проблемасына ұқсайды.) Біз жауапты табуды жаттығу ретінде қалдырамыз.

Шексіз келтіруде циклімізде есептеген сома,

$$1 + \frac{1}{2} + \frac{1}{3} + \dots,$$

Үйлесімді қатар деп аталады, ал әрбір ақырғы сома

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

n ым үйлесімді сан деп аталады. (Математикалық түрде, құмырсқаның арқанның басқа басына жететін себебі үйлесімді қатарың ажырайтындығында, яғни оның бөлшектік сомалары ылғи өседі.) Үйлесімді қатарды натуралды логарифм ($\ln$) функциясының аппроксимациясы үшін қолдануға болады. Өте үлкен мәндерде n ,

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \approx \ln n + 0.577.$$

Бұл n тек кішкентай мәндері үшін 4.8. Суретте көрсетілген. Аппроксимацияның партылғанында қалай жақсаратындығына назар аударыңыз.

Ойлануға берілетін сұрақ 4.33 $H_n \approx \ln n + 0.577$ біле отырып, егер құмырсқа минут сайын тек 1 ғана см өтетін болса, ол арқан соңына жеткенше қанша уақыт кететініне қалай шамамен жауап беруге болады?

Жауабын табу үшін бізге

$$100 = \ln n + 0.577 \text{ болатындай } n \text{ қажет,}$$

себебі құмырсқаның бірінші үзіндісі жалпы қашықтықтың 1/100 құрайды. Бұл

$$e^{100-0.577} = n. \text{ бірдей}$$

Python-да біз жауап іздеу үшін `math.exp(100 - 0.577)` қолдана аламыз, ол бізге 1.5×10^{43} минут береді, бұл шындығында өте ұзақ.

π Аппроксимациясы π мәні – ең танымал математикалық константа шығар. Соңғы 500 жылда таппроксимациясы үшін қолданыла алатын көптеген шексіз қатарлар табылды. Олардың ішіндегі ең танымалы - Лейбниц қатары болып табылады, атауы есептің қосалқы авторы Готфрид Лейбництің құрметіне берілген:

$$\pi = 4 \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots \right)$$

Натуралды логарифимнің үйлесімді қатарының аппроксимациясы жағдайындағыдай, бұл қатардың қаншалықты көп мүшесін есептесек, шынайы π мәніне соншалықты жақындай түсеміз. Бұл соманы табу үшін, біз мүшелердегі заңдылықты анықтап, оларды for цикліндегі индекстік ауыспалы шамаларымен салыстыруымыз қажет. Сонда біз төменде қызылмен белгіленген бос жолға i -ң индекстік ауыспалы шамасынан i -ші мүшені анықтайтын мәнді қоя аламыз.

`defleibniz(terms):`

"""Лейбниц қатарының бөлшектік сомасын есептейді.

Параметр:

`terms`: Қосылатын мүшелер саны

Қайтарылатын мәндер:

Белгіленген мүшелер санының сомасы

"""

`sum = 0`

`for i in range(terms):`

`sum = sum +`

`pi = sum *`

`4 return pi`

Үрдісті түсіну үшін, біз индекстік ауыспалы шамасын қатар мәнінің жанына жазып, өзара байланысты орната аламыз:

i	0	1	2	3	4	...
i^{th} term	1	$-\frac{1}{3}$	$\frac{1}{5}$	$-\frac{1}{7}$	$\frac{1}{9}$	...

Заңдылыққа әлі назар аудармай, біз мүшенің i^{th} абсолютті мәні - 1 екендігін байқай аламыз

$$\frac{1}{2i + 1}$$

Белгілерді кезектестіру үшін, біз жұп дәрежесіне көтерілген -1 1-ге тең, ал тақ дәрежеге көтерілген -1 мәні -1-ге тең деген фактіні қолданамыз. Барлық жұп мүшелер оң, ал тақ мүшелер теріс болғандықтан, ақырында i мүшені ретінде көрсетуге болады.

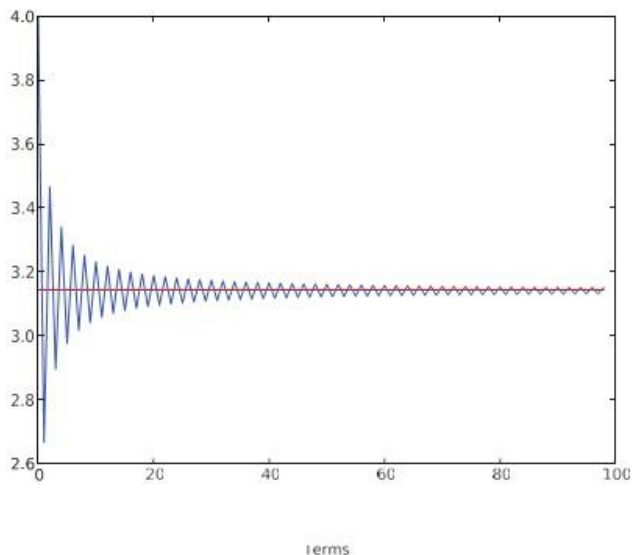
$$(-1)^i \cdot \frac{1}{2i + 1}$$

Осылайша, қызылмен белгіленген меншіктеу функциясының орнында leibniz функциясында мынау болады

$$\text{sum} = \text{sum} + (-1)^i / (2 * i + 1)$$

Ойлануға берілетін сұрақ 4.34 *Өсетін аргументтер топтамасымен толтырылған leibniz функциясын шақырыңыз. Мәндер π -ге қалай келетіндігі туралы не айта аласыз?*

Функцияның бірнеше мәндерін қарап шығып, сіз π -дің іс жүзіндегі мәндерінен көбірек және азырақтарының арасында кезектесіп жүретіндігін байқаған боларсыз. 4.9-сурет осыны көрсетеді.



4.9-сурет Лейбниц қатарының π -ге қосылуы.

Шаршы түбір аппроксимациясы

Шаршы түбір функциясы ($\sqrt{n}$), жалпы жағдайда, тікелей есептеліне алмайды. Алайда ол *вавилон әдісі* (немесе *Ньютон әдісі*) деп аталатын келесі айырымдық теңдеудің көптеген итерациялары арқылы жақсы аппроксимацияланады:

$$X(k) = \begin{cases} 1 & \text{if } k = 0 \\ \frac{1}{2} \left(X(k-1) + \frac{n}{X(k-1)} \right) & \text{otherwise} \end{cases}$$

Мұндағы n біз шаршы түбір алғымыз келетін мән болып табылады. $\sqrt{n}$ аппроксимациясы k үлкен мәндері үшін жақсырақ болады; $X(20)$ шаршы түбірдің шынайы мәніне $X(10)$ қарағанда жақынырақ болады.

Алдыңғы мысалдарымызға ұқсас, біз айырымдық теңдеудің бірізділік мәндерін итерациялар арқылы есептей аламыз. Бұл жағдайда әрбір мән алдыңғысынан жоғарда келтірілген формулаға сәйкес есептеледі. Егер біз айырымдық теңдеудегі мүшені x атты ауыспалы шаманың орнына алатын болсақ, онда келесі қарапайым функция арқылы k -ші мүшені есептей аламыз:

```
def sqrt(n, k):
    """n -нен вавилон әдісінің k итерациялары
    арқылы шаршы түбірді
    аппроксимациялайды.
```

Параметрлер:

n: k шаршы түбірі алынатын сан:

Итерациялар саны

Қайтарылатын мәндер:

n-дегі шаршы түбірдің болжалды мәні
"""

x = 1.0

for index in range(k):

 x = 0.5 * (x + n / x)

return x

Ойлануға берілетін сұрақ 4.35 *k* Әртүрлі мәндері бар $\sqrt{10}$ болжалды мәнін табу үшін жоғардағы функцияны шақырыңыз. Нәтижесі *math.sqrt* функциясынан алынған мәнге сәйкес болу үшін *k*-дің қандай мәнін алу керек?

Жаттығулар

4.5.1. Ойлануға арналған 4.32-сұрақты еске салайық: Құмырсқа бірінші рет арқанның соңына жеткендегі немесе өткендегі мәнді іздестіру үшін қандай функцияны жаза алатын едік? (Сыбырсөз: Бұл көміртек -14-тің жартылай ыдырау проблемасына ұқсайды.)

- (a) Бұл сұраққа жауап беру үшін функцияны жазыңыз.
- (b) Құмырсқаға арқанды толық жүріп өту үшін қанша уақыт керек?
- (c) Егер құмырсқа әр минутта 5 см жүріп өтетін болса, оған екінші соңына жету үшін қанша уақыт керек болады?

4.5.2. *ant* функциясын 4.8.-суреттегідей график салатындай етіп толықтырыңыз.

4.5.3. *e* шамасы (Эйлер саны, натуралды логарифм негізі) шексіз сомаға тең

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

Функцияны жазыңыз
 $e(n)$

Ол n мүшелерінің сомаларын есептеп және қайтарып емәнін аппроксимациялайды. Мысалы, $e(4)$ шақыру $1+1/1+1/2+1/6 \approx 2.667$ мәнін қайтаруы керек. Функцияңыз сіз 4.1.29 жаттығуында жазған factorial функциясын *шақыруы* керек, бұл есепке көмектеседі.

4.5.4. Алдыңғы есепке жазылған функциядағы factorial функциясы өте тиімсіз, себебі көптеген бірдей арифметикалық операциялар қайталана береді. Бұл қайда болатынын түсіндіріңіз.

4.5.5. Алдыңғы жаттығуда айтылған проблемаларды болдырмау үшін, функцияны 4.5.3 жаттығуынан factorial функциясын шақырмай көшіріп жазыңыз.

4.5.6. Алдын ала итерациялар санын көрсетудің орнына, сандық алгоритмдер әдетте ағымдағы мүшенің абсолютті шамасы өте аз болғанша қайталай береді. Сондай ақ біз аппроксимация "өте жақсы" деп болжаймыз. Leibniz функциясын ағымдағы мүшенің абсолюттік шамасы 10–6 артық болып тұрғанда қайталана беретіндей етіп көшіріңіз.

4.5.7. Алдыңғы жаттығуға ұқсас, sqrt функциясын x ағымдағы және алдыңғы мәндері арасындағы айырманың абсолютті шамасы 10–15 көбірек болып тұрғанда қайталана беретіндей етіп көшіріңіз.

4.5.8. Келесісі 14-ғасырда үнді математигі Мадхава Сангама-грама ашқан өрнек π есептеудің тағы бір әдісі болып табылады.

$$\pi = \sqrt{12} \left(1 - \frac{1}{3 \cdot 3} + \frac{1}{5 \cdot 3^2} - \frac{1}{7 \cdot 3^3} + \dots \right)$$

Функцияны жазыңыз
approxPi(n)

Ол π аппроксимациясы үшін осы өрнектің n мүшелерін анықтайды. Мысалы,
approxPi(3)
мәнін қайтаруы қажет

$$\sqrt{12} \left(1 - \frac{1}{3 \cdot 3} + \frac{1}{5 \cdot 3^2} \right) \approx \sqrt{12} (1 - 0.111 + 0.022) \approx 3.156.$$

Бұл мүшелердің бірізділігін қағидасын түсіну үшін, мына кестені

қолданыңыз:

Мүше нөмірі	Мүше
0	$1/(3^0 \cdot 1)$
1	$1/(3^1 \cdot 3)$
2	$1/(3^2 \cdot 5)$
3	$1/(3^3 \cdot 7)$
$\vdots$	$\vdots$
i	?

i ортақ мәні үшін мүше қандай болмақ?

4.5.9. 17-ші ғасырдағы ағылшын математигі Джон Валлистің құрметіне аталған Валлис туындысы - π -ге жетуге ұмтылатын шексіз туынды болып табылады

$$\pi = 2 \left(\frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \dots \right)$$

Функцияны жазыңыз

wallis(terms)

ол Валлис туындысы мүшелерінің берілген санын анықтайды. Сыбырсөз: Мүшелерді екі деп қарастырып, әрбір жұп үшін өрнек табыңыз. Сосын қажетті мүше сандарын көрсетуге керекті жұп санын қайталаңыз. terms мүшелерінің саны жұп деп есептеуге болады.

4.5.10. 15-ғасырдағы үнді математигі Нилаканта Сомаяджидің құрметіне аталған Нилаканта қатары - π үшін тағы бір шексіз қатар болып табылады:

$$\pi = 3 + \frac{4}{2 \cdot 3 \cdot 4} - \frac{4}{4 \cdot 5 \cdot 6} + \frac{4}{6 \cdot 7 \cdot 8} - \frac{4}{8 \cdot 9 \cdot 10} + \dots$$

Функцияны жазыңыз

nilakantha(terms)

Ол Нилаканта қатарының берілген мүшелер санын есептеп

анықтайды.

4.5.11. Келесі шексіз туындыны 16-ғасырдағы француз математигі Франсуа Виет ашқан болатын:

$$\pi = 2 \cdot \frac{2}{\sqrt{2}} \cdot \frac{2}{\sqrt{2+\sqrt{2}}} \cdot \frac{2}{\sqrt{2+\sqrt{2+\sqrt{2}}}} \cdot \frac{2}{\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}}} \dots$$

Функцияны жазыңыз
viete(terms)

ол Виет туындысының берілген мүшелер санын есептейді. (Зандылыққа мұқият қарап шығыңыз; егер сіз әрбір мүшедегі бөлгішті табу үшін алдыңғы мүше бөлгішінің мәнін алатын болсаңыз барлығы көзге көрінгеннен оңайырақ болмақ.)

4.6 ТҮЙІНДЕМЕ

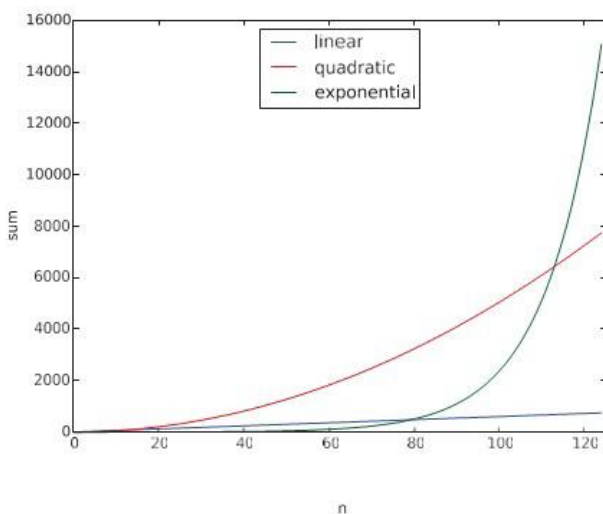
Осы тарауда көптеген әртүрлі есептерді шешсек те, біз әзірлеген көптеген функцилардың бірдей негіздік форматы болды:

```
def accumulator( ):
    sum =                # for index in range( )
    сумматорының        # бірнеше сан рет қайталануы
    инициализациясы:
        sum = sum +      # returnsum сумматорына
    бірдеңе қосу        # сумматордың ақырғы мәнін
    қайтару
```

Біз жасаған функциялардың негізгі айырмашылығы - циклдің әр итерациясында сумматорға (қызылмен белгіленген нұқсау) қосылатындығында. Осы функцияның үшеуіне, әсіресе: 119 беттегі `pond` функциясы, 123 беттегі `countLinks` функциясы, және төменде келтірілген 134 беттегі 4.1.27 жаттығуның шешіміне қарайық.

```
def growth(finalAge):
    height = 95
    for age in range(4, finalAge + 1):
        height = height + 6
    return height
```

Әрбір итерацияда `growth` функциясында сумматорға тұрақты мән қосылып отырады:



4.10-сурет Сызықтық, шаршылық және экспоненциальді өсудің кескіні. Әр итерацияда сумматормен құрылған 6 қосатын, индекстік ауыспалы шамасын қосатын және сәйкесінше сумматормен 1,08-ге көбейтетін айнымалылар.

$$\text{height} = \text{height} + 6$$

countLinks функциясында сумматорға индекстік ауыспалы шаманың минус бір мәні қосылады:

$$\text{newLinks} = \text{node} - 1$$

$$\text{totalLinks} = \text{totalLinks} + \text{newLinks}$$

Ал pond функциясында сумматордың өзінің факторы әр итерацияда қосылады:

$$\text{population} = \text{population} + 0.08 * \text{population} \quad \# \text{ "1500" есептемегенде}$$

(Оңайлату үшін біз бастапқы бағдарламадағы азайтуды ескермейміз.) Бұл үш түрлі сумматор үш түрлі әдіспен өседі. Әр итерацияда сумматорға тұрақты шаманы қосу, growth функциясындағыдай, тұрақты шамаға көбейтілген итерациялар санына тең соманың ақырғы мәніне әкеледі.

Басқа сөзбен айтқанда, егер бастапқы мәні – a болса, тұрақты қосылатын шама – бұл c , ал итерациялар саны n -ге тең болса, онда сумматордың ақырғы мәні $a+cn$ болады. (growth функциясы үшін $a=95$ және $c=6$, сондықтан ақырғы сома $95 + 6n$.) n өскен сайын, cn тұрақты шамаға артады. Бұл *сызықты* өсу деп аталады және ол көк сызықпен

4.10. Суретте көрсетілген.

Сумматорға индекстік ауыспалы шаманы қосу, countLinks функциясы жағдайындағыдай өте жылдам өсуге әкеледі. countLinks функциясында ақырғы шама

Ескерту 4.2: Үшбұрышты сандар

Кез келген n оң мәнінің

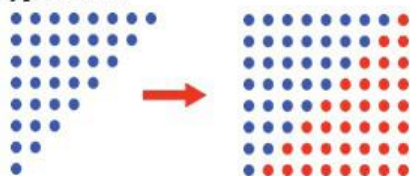
$$1 + 2 + 3 + \dots + (n - 2) + (n - 1) + n \text{ сомасын}$$

алудың бірнеше жақсы әдісі бар. Бірінші тәсіл сомада сыртқы жақтан сандарды қосу арқылы істеледі. Бірінші және соңғы сандардың сомасы $n + 1$ тең екендігіне назар аударыңыз. Енді біз тағы бір позициядан сол және оң жағынан алатын болсақ, онда да $(n - 1) + 2 = n + 1$ аламыз. Келесі жұп, $(n - 2) + 3 = n + 1$. Бұл заңдылық жалғасатын секілді, себебі біз сол жақ саннан 1 алып отырып, оң жақ санға 1 қосып отырамыз. Жалпы алғанда, сомадағы әрбір екі мүшеге бір $n + 1$ бар, жиынтығы $n/2$ рет $n + 1$ -ден. Сәйкесінше, сома келесідей болады

$$1 + 2 + 3 + \dots + (n - 2) + (n - 1) + n = \frac{n(n + 1)}{2}$$

Мысалы, $1 + 2 + 3 + \dots + 8 = (8 \cdot 9) / 2 = 36$, ал $1 + 2 + 3 + \dots + 1000 = (1000 \cdot 1001) / 2 = 500,500$.

Бұл тәсілді алудың екінші әдісі көрнекірек болып табылады. Әр санды сомада $n = 8$ үшін төменде суретте көрсетілгендей етіп дөңгелектер бағанасы ретінде суреттейік.



Бірінші бағанада $n = 8$ дөңгелек, екіншіде $n - 1 = 7$ және т.с.с. Осылайша бұл үшбұрыштағы дөңгелектердің жалпы саны ізделіп отырған мәнге тең. Енді осы үшбұрыштың нақты көшірмесін алып оны жоғарда оң жақтағы суретте көрсетілгендей бастапқы үшбұрыштың оң жағына оның айна-қатесіз

суретін қояйық. Алынған үшбұрыштың n жолы және $n + 1$ бағаналары бар, осыдан жалпы алғанда $n(n + 1)$ дөңгелектер шығады. Бұл үшбұрыштарда дөңгелектердің саны бастапқы үшбұрыштыкіне қарағанда екі есе көбірек болғандықтан, бастапқы үшбұрыштағы дөңгелектер саны $n(n + 1) / 2$ тең. Бұл шамаға сәйкес, осындай форманың сомасы болып табылатын 36 және 500,500 секілді сандар *үшбұрышты сандар* деп аталады.

Сумматордың мәні - болып табылады

$$1 + 2 + 3 + \dots + (n - 1)$$

Ол мынаған тең

$$\frac{1}{2} \cdot n \cdot (n - 1) = \frac{n^2 - n}{2}.$$

4.2 ескерту бұл нәтижені алудың екі жылжитпа амалын түсіндіреді. Бұл сома n^2 шамалас болғандықтан, 4.10-суретте қызыл айнымалымен көрсетілгендей біз оны *шаршылы* өсім көрсететіндігін айтамыз. Шын мәнінде бұл соманы білген өте пайдалы, бұл 11-тарауда байқалады

Ақырында, әр итерацияда сумматор факторына өзіне қосып отырып, `pond` функциясындағыдай, жылдамырақ өсімді аламыз. `pond` функциясында егер біз

`0.08 * population`-ді әр итерацияда `population`-ге қосатын болсақ, сумматордың ауыспалы шамасы циклдің n итерацияларының соңына қарай `population` бастапқы мәнінің $1.08n$ көбейтілген шамасына тең болады. Сондықтан біз оны *экспоненциалды* өсу дейміз, ол 4.10-суретте жасыл айнымалымен көрсетілген. Назар аударыңыз, n қаншалықты көп болса, экспоненциалды өсімнің айнымалысы басқа екі айнымалының өсуінен соншалықты озады, егер тіпті n дәрежесіне дейін көтерілетін сан 1.08 секілді аз болса да.

Осылайша, барлық жинақтаушы алгоритмдер бір біріне азды көпті ұқсас болса да, сумматорлардың әрекеттерінің нәтижелері ғажап түрде ерекшеленеді. Бұл әртүрлі өсу түрлерінің салыстырмалы көрсеткіштерін түсіну - тек информатикада ғана емес, сонымен бірге басқа да әртүрлі салаларда өте маңызды болып табылады. Мысалы, экспоненциалды түрде өсетін індетті қателесіп сызықты деп алатын болсақ, бұндай қателік тіпті біреудің өліміне әкелуі мүмкін!

Бұл өсу түрлері сондай ақ алгоритмдердің тиімділігіне қолданылуы мүмкін, оны келесі тарауларды көреміз. Бұл жағдайда n кіріс мәліметтердің көлемін білдіретін болады, ал y осі сәйкесінше нәтижені алгоритм арқылы есептеу үшін қажет бастауыш қадамдардың санын білдіретін болады.

Жаттығулар

4.6.1. Келесі сумматорлардың әрқайсысы сызықтық, шаршылық

немесе экспоненциалдық өсімге әкелетіндігін анықтаңыз.

- (a)

```
sum = 0
for index in range(n):
    sum = sum + index * 2
```
- (b)

```
sum = 10
for index in range(n):
    sum = sum + index / 2
```
- (c)

```
sum = 1
for index in range(n):
    sum = sum + sum
```
- (d)

```
sum = 0
for index in range(n):
    sum = sum + 1.2 * sum
```
- (e)

```
sum = 0
for index in range(n):
    sum = sum + 0.01
```
- (f)

```
sum = 10
for index in range(n):
    sum = 1.2 * sum
```

4.6.2. 4.10-суретіне қараңыз. n мәндері үшін шамамен 80 аз тез өсетін экспоненциалды айнымалылар қалған екеуінен төмен орналасады. Неліктен екенін түсіндіріңіз.

4.6.3. 4.10-суретін елестету үшін бағдарлама құрыңыз.

4.7 АЛДАҒЫ ЗЕРДЕЛЕУ

Бұл тараудың эпиграфы TED пен Стивен Хокингтің арасындағы 2008 жылғы сұхбаттарының бірі болмақ.

Сіз оны мына сайтта көре аласыз

www.ted.com/talks/stephen_hawking_asks_big_questions_about_the_universe.

Егер сіз демографиялық динамика үлгілері мен жалпы есептік үлгілеу жөнінде толығырақ білгіңіз келсе, тамаша дереккөз – Анджела Шифлет және Джордж Шифлеттің *Информатикаға кіріспе* [52] еңбегі болып табылады.

4.8 ТӘЖІРИБЕ ТАПСЫРМАЛАРЫ

Жоба 4.1 Паразиттік қарым-қатынас үлгісі

Бұл жобаны іске асыру үшін сіз 4.4. Тарауындағы 145-146 бетте берілген дискретті айырымдық теңдеулер туралы материалды оқып шығуыңыз керек.

Паразит дегеніміз – басқа ағзада немесе оның ішінде, оның өмірінің бір бөлігі болып өмір сүретін ағза болып табылады. Паразитоид дегеніміз – өз қожайынын ақырында өлтіретін паразиттік ағза болып табылады. Паразитоид-бунақденелі қожайынын жұқтырып, оған жұмыртқа салады, жұмыртқадан жарылып шыққан дернәсілдер содан кейін қожайынның тірі ағзасымен қоректенеді. (Расында ғажайып емес пе?) Қожайын ақырында өледі, ал ересек паразитоидтар қожайынының денесінен сыртқа шығады.

Алғаш рет 1935 [36] жылы Александр Николсон және Виктор Бэйли ұсынған Николсон-Бейлидің үлгісі, - паразитоид және олардың қожайындарының салыстырмалы көлемдерін шығаруға ұмтылатын екі айырымдық теңдеу болып табылады. Біз қожайындар популяциясының көлемін t жылы $H(t)$ деп, ал паразитоидтар популяциясының көлемін t жылы $P(t)$ деп аламыз. Сонда бұл үлгіні суреттейтін айырымдық теңдеулер мыналар болады

$$H(t) = r \cdot H(t-1) \cdot e^{-aP(t-1)}$$

$$P(t) = c \cdot H(t-1) \cdot (1 - e^{-aP(t-1)})$$

мұндағы

- r – паразитоид жұқтырмаған қожайындардың тірі қалған ұрпақтарының орташа саны,
- c – бір қожайынның ішінде салынатын жұмыртқаның орташа саны, ал
- a – түзету коэффициенті, паразитоидтармен іздестіру немесе іздестіру саласының тиімділігін сипаттайды (қаншалықты жоғары болса, соншалықты тиімді).

$(1 - e^{-aP(t-1)})$ шамасы – бұл қожайынның $P(t-1)$ паразитоид болса жұқтырылады деген ықтималдық, мұндағы e – Эйлер саны (натуралды логарифмнің негізі). Осылайша,

$$H(t-1) \cdot (1 - e^{-aP(t-1)})$$

бұл $t-1$ жыл ішінде жұқтырылған қожайындар саны. Бұл шаманы c -ке көбейтсек $P(t)$ аламыз, бұл t жыл ішінде туылатын жаңа пара-

зитоидтар саны. Назар аударыңыз, індеттің ықтималдығы паразитоид популяциясының көлемінің өсуіне қарай экспоненциалды түрде өседі. a шамасының үлкенірек болуы да жұғу ықтималдығын арттырады.

Тапсырма 4.1.1 Осыған ұқсас әдіспен $H(t)$ үшін айырымдық теңдеудің мәнін түсіндіріңіз ($e^{-aP(t-1)}$ – қожайын жұқтырылмаған деген ықтималдық.)

I-бөлім: Үлгіні іске асыру

Бұл үлгіні іске асыру үшін функцияны жазыңыз

NB(hostPop, paraPop, r, c, a, years)

Ол мына айырымдық теңдеулерді уақыт өтуімен екі популяция көлемінің өзгеру графиктерін құру үшін қолданады. Функцияңыз бұл мәндерді екі түрлі әдіспен құруы қажет болады (сәйкесінше, екі түрлі графикте). Біріншіден, x осімен қожайын популяциясының көлемін және y осімен паразитоид популяциясының көлемін көрсетіп график құрыңыз. Осылайша, графиктің әр нүктесі екі популяцияның нақты бір жылдағы көлемдері болып табылады. Екіншіден, y осінде екі популяцияның өзгеруі ал x осінде уақыты көрсетілген графикті құрыңыз. Бір графикте екі популяция көлемін көрсету үшін, `pyplot.show`шықырудан бұрын популяцияның әрқайсысының тізімі үшін `pyplot.plot` функциясын шақырыңыз. 4.2-тараудың соңында графиктің әр сызыған қалай белгілеп түсінік сөзді қалай қосу керектігін қараңыз.

Тапсырма 4.1.2 35 жылға 24 қожайын мен 12 паразитоид популяцияларының бастапқы көлемдерінде үлгілеу үшін NB функцияңызды шақыратын `main` функциясын жазыңыз. $r = 2$, $c = 1$ және $a = 0.056$ мәндерін қолданыңыз. Алынған нәтижелерді суреттеп түсіндіріңіз.

Тапсырма 4.1.3 Сол параметрлермен үлгілеуді қайта іске қосыңыз, бірақ енді a -ға $-\text{math.log}(0.5) / \text{paraPop}$ мәнін беріңіз. (Бұл шамамен $a = -\ln 0.5 / 12 \approx 0.058$, бастапқы a мәнінен жоғарырақ.) Нені байқадыңыз?

Тапсырма 4.1.4 Сол параметрлермен үлгілеуді қайта іске қосыңыз, бірақ енді $a = 0.06$ мәнін беріңіз. Енді нені байқадыңыз?

Тапсырма 4.1.5 Осы үш үлгілеудің негізінде сіз аталмыш үлгі туралы және оның а мәніне деген сезімталдығы жөнінде не дейсіз?

2-бөлім : Шектеулі өсім

Николсон-Бейлидің жаңартылған үлгісіне экожүйенің сыйымдылығы кіреді, ол қожайынның популяциясын бақылап отырады. Экожүйе сыйымдылығы – экожүйе кез келген нақты уақыт сәтінде қолдай алатын ағзалардың максималды саны. Егер популяция саны экожүйенің сыйымдылынан асып кетсе, бүкіл популяцияны қолдайтын қор жеткіліксіз болып, түрлердің бір бөлігі тіршілігін тоқтатады.

Қайта қаралған бұл үлгіде $P(t)$ өзгермей өз күйінде қалып, ал $H(t)$ келесіге өзгерді

$$H(t) = H(t-1) \cdot e^{-aP(t-1)} \cdot e^r(1 - H(t-1) / K)$$

Мұндағы K – экожүйе сыйымдылығы. Бұл жаңа айырымдық теңдеуде қожайынның тірі қалған ұрпағының орташа саны, бұрын r болса, енді келесідей берілген

$$e^r(1 - H(t-1) / K)$$

Назар аударыңыз, қожайындар саны $H(t-1)$ экожүйе сыйымдылығы K -ге тең болғанда дәреже көрсеткіші нөлге тең болады. Осылайша қожайынның тірі қалған ұрпағының саны $e^0 = 1$ тең. Жалпы, қожайын саны $H(t-1)$ экожүйе сыйымдылығы K -ге жеткенде дәреже көрсеткіші азая түседі, ал жоғардағы өрнек мәні 1-ге жақындайды. Басқа жағынан, $H(t-1) - 0$ -ге ұмтылғанда, өрнек e^r -ге ұмтылады. Осылайша, жалпы алғанда, тірі қалған ұрпақ саны 1 -ден e^r -ге дейінгі аралықта, $H(t-1)$ экожүйе сыйымдылығына қаншалықты жақын келетіндігіне қарай түрленеді.

Функцияны жазыңыз

$$NB\_CC(hostPop, paraPop, r, c, a, K, years)$$

ол осы нақтыланған үлгіні іске асырады және алдыңғы функция секілді сондай графиктерді құрайды.

Тапсырма 4.1.6 Өзіңіздің NB_CC функцияңызды 35 жылға 24 қожайын мен 12 паразитоидтан тұратын популяциялардың бастапқы көлемдерінде үлгілеу үшін шақырыңыз. $r = 1.5$, $c = 1$, $a = 0.056$ және $K = 40$ мәндерін қолданыңыз Алынған нәтижелерді суретпен түсіндіріңіз.

Тапсырма 4.1.7 1-бөлімде қолданған *a* -ның барлық үш мәндерімен үлгіні іске қосыңыз. Бұл нәтижелер алдыңғы үлгіден қалай ерекшеленеді?

Жоба 4.2 Қаржы калькуляторлар

Бұл жобада сіз үш калькулятор жазасыз, олар біреуге (бәлкім сізге де) бюджетіңізді болшаққа жоспарлауға көмектеседі.

Бөлім 1: Білім алуға несиемді жабу үшін қанша уақыт кетеді?

Бірінші калькуляторыңыз студенттің екі түрлі ай сайынғы төлемдерде несиені төлеуге кететін уақытты салыстыратын болады. Сіздің негізгі бағдарламаңыз

- төлеудің басындағы білім алу несиесінің теңгерімі
 - жылдық атаулы пайыздық мөлшерлеме (он екіге көбейтілген ай сайынғы мөлшерлеме)
 - екі ай сайынғы төлем сомасы
- Функцияны жазыңыз
`comparePayoffs(amount, rate, monthly1, monthly2)`

ол ай сайынғы төлем сомаларының әрқайсысы үшін несиені төлеуге қажетті ай санын есептейді. Несие бойынша үстемақылар ай сайын $\text{rate} / 100 / 12$ мөлшерлеме бойынша аударылуы қажет. Сіздің функцияңыз сонымен бірге төлемнің әр сомалары үшін несие бойынша қалдық графигін екі қалдық та нөлге жеткенге дейінгі уақыт бойы құруы керек. Содан кейін, ол экранға өтеудің екі кезеңінің ұзақтығын және ай сайынғы төлемнің ең жоғарғы сомасын таңдауда несиені қаншалықты ерте өтеуге болатын уақытты шығаруы керек. Мысалы, сіздің нәтижеңіз келесідей болуы мүмкін:

Бастапқы шот теңгерімі: 60000

Жылдық атаулы пайыздық мөлшерлеме: 5

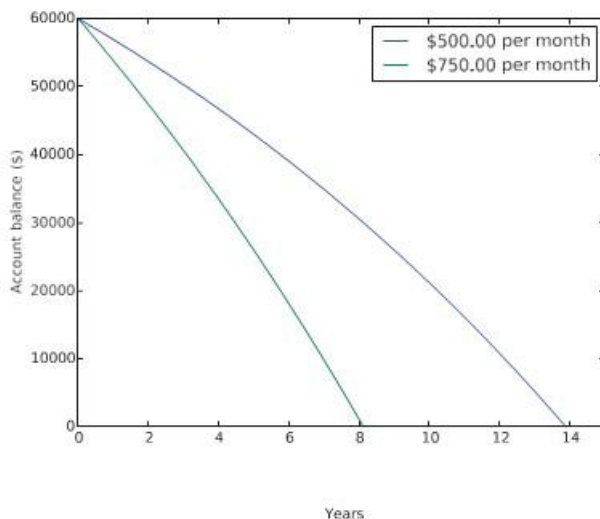
Ай сайынғы төлем 1: 500

Ай сайынғы төлем 2: 750

Егер сіз айына \$ 500.00 төлесекіз, өтеу мерзімі 13 жыл және 11 айды құрайды.

Егер сіз айына \$ 750.00 төлесекіз, өтеу мерзімі 8 жыл және 2 айды құрайды.

Егер сіз айына \$ 250.00 көбірек төлесекіз, өтеу мерзімі 5 жылға және 9 айға азаяды.



Тапсырма 4.2.1 Пайыздық мөлшерлемесі 4 % болатын \$ 20,000 білім алу несиесін төлеуге егер сіз ай сайын \$ 100 төлесеңіз қанша уақыт кетеді? Он жыл ішінде несиені өтеу үшін Сізге ай сайын шамамен қанша төлеу керек болады?

Тапсырма 4.2.2 Егер сіз дәл сол несиені төлеуге кететін уақытты анықтайтын бағдарламаны іске қосатын болсаңыз, алайды айына тек \$ 50 төлейтін болсаңыз, проблема туындайды. Қандай проблема?

2-бөлім: Сіз зейнетақыға қанша қаражат жинайсыз?

Сіздің екінші калькуляторыңыз зейнетақы қорына салынатын екі түрлі ай сайынғы салымдарды жасаудан түскен зейнетақы жинақтарын салыстыратын болады. Бағдарламаңыз келесі кірістік мәліметтерді сұрауы керек:

- Зейнетақы шотындағы бастапқы теңгерім
- Салымшының қазіргі жасы
- Салымшының зейнетке шығатын болжалды жасы
- Салымның болжалды жылдық атаулы табыстылық мөлшерлемесі
- Салымдардың екі ай сайынғы сомалары Содан кейін Функцияны жазыңыз

compareInvestments(balance, age, retireAge, rate, monthly1, monthly2)

Ол салымшы зейнеткерлік жасқа жеткендегі ай сайынғы инвестициялардың сомаларының екеуі үшін зейнетақылық шотындағы ақырғы теңгерімін есептейді. Салым бойынша пайыздар $\text{rate} / 100 / 12$ мөлшерлемеді ай сайынғы капиталданумен аударылуы керек. Функция ай сайынғы салымдардың екі сомасы үшін зейнетақылық шот теңгерімінің өсу графигін құруы керек, содан кейін екі ақырғы теңгерімді, сонымен бірге жоғарырақ ай сайынғы салымдардан келетін қосымша түсімді экранға шығаруы керек. Мысалы, сіздің нәтижеңіз келесідей болуы мүмкін:

Бастапқы шот теңгерімі: 1000

Қазіргі жасы: 20

Зейнетке шығатын жасы: 70

Табыстылықтың жылдық атаулы пайыздық мөлшерлемесі: 4.2

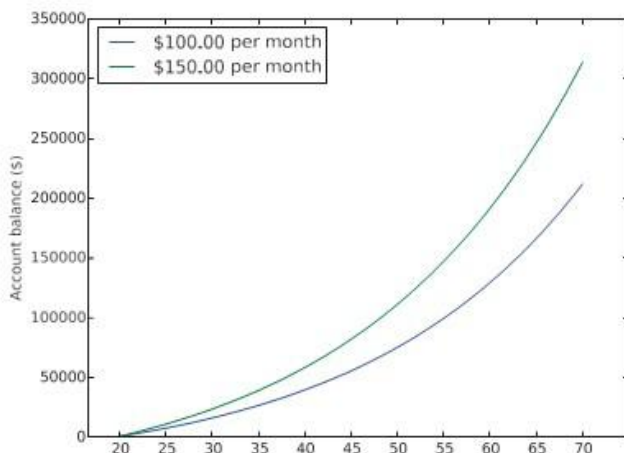
Ай сайынғы салымдар 1: 100

Ай сайынғы салымдар 2: 150

Бір айдағы \$ 100.00 инвестициялардан келетін ақырғы теңгерім:
\$ 212030.11.

Бір айдағы \$ 150.00 инвестициялардан келетін ақырғы теңгерім:
\$ 313977.02.

Егер сіз айына \$ 50.00 көбірек салатын болсаңыз, зейнетақыдық салымда \$ 101946.91 көбірек аласыз.



Тапсырма 4.2.3 *Сіздің жасыңыз 30-да, және бірнеше жұмыс жылы ішінде сіз зейнетақыға \$ 6,000 үнемдедіңіз делік. Егер сіз айына \$ 200 салым салатын болсаңыз, 72 жасыңызда зейнетке шықсаңыз салымыңыз 3% жылдық мөлшерлемеге артылса қанша қаражат жиналады? Сіз айына \$ 50 көбірек салатын болсаңыз қаншалықты көп аласыз?*

Бөлім 3: Менің зейнетақылық жинақтарым қанша уақытқа жетеді?

Сіздің үшінші калькуляторыңыз басынан бастап алдыңғы калькулятор секілді есептерді шешеді, алайда ол тек бір ғана ай сайынғы салымдар сомасын есептейді. Шоттағы ақырғы қалдықты есептеген соң ол жинақтарыңыздың зейнетақыға қаншалықты жететінін бағалайды. Сіздің бағдарламаңыз дәл жоғарыдағыдай мәндерді (бірақ тек бір ғана ай сайынғы салымдар сомасымен), сонымен қатар салымшы зейнетке шыққан соң бірінші жылы алуды жоспарлаған ақырғы теңгерім пайызын сұрауы керек. Функцияны жазыңыз

retirement(amount, age, retireAge, rate, monthly, percentWithdraw)

ол ай сайынғы салымдар сомасын салымшы зейнетке шығатын жасқа жетіп, кейін зейнетке шығып ай сайын соманы алып отырған уақытқа дейін зейнетақы шотындағы қаражат қалдығына қосады. Зейнетке шыққанан кейінгі бірінші айда , бұл сома ағымдағы теңгерімнің percentWithdraw -дың он екіден бір бөлігін құрауы керек. Әр ай сайын алу сомасы жылына мәселен 3 % инфляция деңгейіне сәйкес ұлғаюы керек. Ай сайын ағымдағы теңгерімнен пайыздар $rate / 100 / 12$ мөлшерлемесі бойынша аударылуы керек. Функция уақыт бойынша зейнетақылық шот теңгерімі графигін құрып, сосын экранға зейнетақы қоры таусылатын жасты шығаруы керек. Мысалы, сіздің бағдарламаңыздың нәтижесі келесідей болуы мүмкін:

Бастапқы шот теңгерімі: 10000

Қазіргі жасы: 20

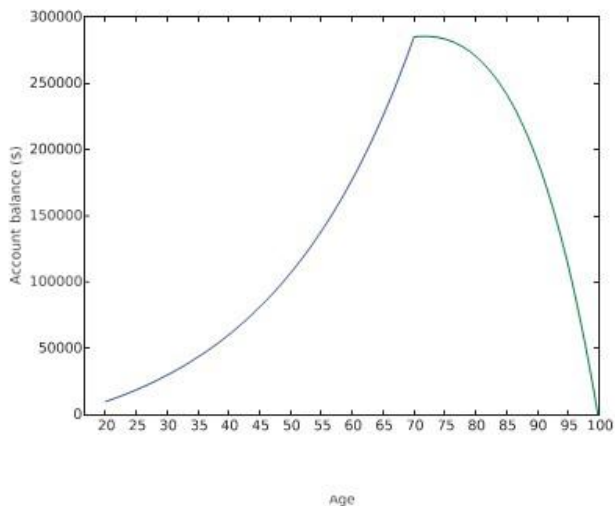
Зейнетке шығатын жасы: 70

Табыстылықтың жылдық пайыздық мөлшерлемесі: 4.2

Ай сайынғы салымдар: 100

Зейнеткерлікте алынатын жыл сайынғы пайыз : 4

Сіздің жинақтарыңыз 99 жас және 10 айға дейін жетеді.



Тапсырма 4.2.4 Егер сіз 4.2. Тапсырмасында жазылған жоспарға сәйкес жүріп (айына \$ 200 салып) және зейнетке шыққанда 4 % алсаңыз сіздің зейнетақы жинағыңыз қанша уақытқа жетеді?

*Жоба 4.3 Нарыққа кіру

Сіз бұл жобаны іске асыру үшін 4.4-тараудың материалын қарап шығуыңыз керек.

Бұл жобада сіз 1969 жылы маркетинг Франк Басс [4] әзірлеген айырымдық теңдеуді қолданып, уақыттағы жаңа өнімнің нарыққа қабылдануын (яғни, өнімнің “нарыққа кіруі”) үлгілейтін бағдарлама құрасыз. Бастың диффузиялық үлгісі - бір өнім және N әлеуетті өнім тұтынушыларынан тұратын бір тұрақты халық барлығын білдіреді. Айырымдық теңдеудегі $A(t)$ – халық (0 -ден 1-ге дейінгі) үлесі, ол өнімді нарыққа жібергеннен кейін t аптадан соң жаңа өнімді қабылдады. 4.4-тараудағы айырымдық теңдеулердегідей $A(t)$ мәні $t - \Delta t$ алдыңғы уақыт кезеңінде өнімді қабылдаған халық үлесінің $A(t - \Delta t)$ мәніне тәуелді. t аптада өлшенетіндіктен, сәйкесінше, Δt – бұл аптаның бір бөлігі. Өнімнің қабылдану жылдамдығы екі факторға тәуелді.

1. Өнімді әлі қабылдамаған халық сегменті оны g тұрақты қабылдау жылдамдығымен қабылдайды, ол тек кездейсоқтыққа тәуелді. Анықтамаға сәйкес $A(t)$ дегеніміз алдыңғы $A(t - \Delta t)$ уақыт аралығында

өнімді қабылдап қойған халық үлесі болып табылады. Осылайша, әлі де алдыңғы уақыт аралығында өнімді қабылдамаған халық үлесі бұл $1 - A(t - \Delta t)$. Онда осы тұрақты қабылдау жылдамдығына сәйкес t апта ішіндегі жаңа тұтынушылар үлесі мынаны құрайды

$$r \cdot (1 - A(t - \Delta t)) \cdot \Delta t.$$

2. Екіншіден, қабылдау жылдамдығына халық арасындағы ауыз-ба ауыз жеткізілуі әсер етеді. Өнімді қабылдаған халық мүшелері оны әлі қабылдамағандарына ықпал жасауы мүмкін. Тұтынушылар қаншалықты көп болса, тұтынушылар мен тұтынбаушылар арасындағы әлеуетті өзара әрекеттесу соншалықты көп болып, бұл қабылдану жылдамдығын арттырады. Алдыңғы уақыт аралығындағы тұтынушылар мен тұтынбаушылар арасындағы барлық әлеуетті өзара әрекеттесулер үлесі мынаны құрайды

$$\underbrace{A(t - \Delta t)}_{\text{Тұтынушылар үлесі}} \cdot \underbrace{(1 - A(t - \Delta t))}_{\text{Тұтынбаушылар үлесі}}.$$

Бір апта ішінде өнімді жаңа қабылдауға әкелетін өзара әрекеттесулер үлесі - әлеуметтік жұғу деп аталады. Әлеуметтік жұғуды s деп белгілейік. Осылайша, уақыт аралығындағы әлеуметтік жұғуды есептегендегі жаңа тұтынушылар үлесі t аптаның соңында мынаны құрайды

$$s \cdot A(t - \Delta t) \cdot (1 - A(t - \Delta t)) \cdot \Delta t.$$

Әлеуметтік жұғу тұтынушылардың тұтынбаушыларды өнімді қабылдау керектігін қаншалықты сәтті сендіре алатындығын өлшейді. Ең ақырғы жағдайда, егер $s = 1$ болса, онда тұтынбаушылар мен тұтынушылар арасындағы өзара әрекеттесу тұтынбаушының өнімді қабылдауына әкеледі. Басқа жағынан алғанда, егер $s = 0$ болса, онда ағымдағы тұтынушылар тұтынбаушылардың ешқайсысын өнімді қабылдау керектігін сендіре алмайды.

Бұл екі бөлікті қоса отырып, Басстың диффузиялық үлгісіне айырымдық теңдеуді аламыз

$$A(t) = A(t - \Delta t) + \underbrace{r \cdot (1 - A(t - \Delta t)) \cdot \Delta t}_{\text{Енгізу жылдамдығының тұрақты құрамдас бөлігінен келетін жаңа тұтынушылар үлесі}} + \underbrace{s \cdot A(t - \Delta t) \cdot (1 - A(t - \Delta t)) \cdot \Delta t}_{\text{Әлеуметтік жұғудан келетін жаңа тұтынушылар үлесі}}$$

Енгізу жылдамдығының
тұрақты құрамдас бөлігінен
келетін жаңа тұтынушылар үлесі

Әлеуметтік жұғудан
келетін жаңа
тұтынушылар үлесі

Бөлім 1: Басстың диффузиялық үлгісін іске асыру

Басстың диффузиялық үлгісін іске асыру үшін функцияны жазыңыз

productDiffusion(chanceAdoption, socialContagion, weeks, dt)

chanceAdoption мен socialContagion параметрлері - сәйкесінше r және s мәндері болып табылады. Соңғы екі параметр болжауға weeks апта санын және қолданылатын Δt мәнін береді. Сіздің функцияңыз екі айнымалы құруы керек, екеуіне де x осінде уақыт, ал y осінде тұтынушылардың арақатынасы болады. Бірінші айнымалы t уақытында өнімді қабылдаған халықтың жалпы үлесін көрсетеді. Бұл жоғарыдағы айырымдық теңдеудегі $A(t)$. Екінші айнымалы $A(t)$ өзгеру жылдамдығы болады. Оны мына формуламен есептеуге болады

$$\frac{A(t) - A(t - \Delta t)}{\Delta t}$$

Немесе, бұл өзгеріс жылдамдығын дәл солай апта сайынғы өсу жылдамдығына келтірілген кез келген уақыт сәтіндегі жаңа тұтынушылар үлесі ретінде ұсынуға болады, яғни dt -ге бөлінген халық үлесінің өсуі. $\Delta t = 0.01$ мәнінде 15 апта ішінде өнімді іске қосудың үлігісі үшін сіздің функцияңызды қолданатын бағдарламаны жазыңыз. Өнімді нарыққа кіргізу үшін біз өнімнің қабылдануы әлеуметтік ықпалсыз баяу жүреді деп болжаймыз, ал әлеуметтік жұғу ықпалы үлкен болады. Бұл болжамдарға сәйкес болу үшін $r = 0.002$ және $s = 1.03$ қолданыңыз.

Тапсырма 4.3.1 Суретті сипаттаңыз және жаңа тұтынушылардың пайда болуы мен өнімді енгізгеннен кейінгі 15 аптадан кейін тұтынушылардың ақырғы жалпы санындағы заңдылықтарды түсіндіріңіз.

Тапсырма 4.3.2 *Енді r -ге өте кішкене мән беріңіз, ал s -ті сол күйінше қалдырыңыз ($r=0.00001$, $s=1.03$) да сол сұраққа жауап беріңіз. Мұнда қандай нарық түрі келтірілген?*

Тапсырма 4.3.3 Енді r -ід оның бастапқы мәнінен 100 есе көп етіп белгілеңіз, ал s -ті нөлге тең етіп ($r=0.2, s=0$) қайтадан бірінші сұраққа жасау беріңіз. Мұнда қандай нарық түрі келтірілген?

Бөлім 2: Ықпал етушілер мен ізбасарлар

Шынайы нарықтық қарым-қатынастарда кейбір тұтынушылар басқаларға қарағанда көбірек ықпал ете алады. Сондықтан, шынайылық үшін, бүкіл халықты екі топқа бөліп, *ықпал етушілер және ізбасарлар* [56] деп атайық. Ықпал етушілерге тек қана басқа ықпал етушілер әсер ете алады, ал, ізбасарларға топтың әрқайсысы да әсер ете алады. Халық арасындағы ықпал етушілер мен ізбасарлар санын NA және сәйкесінше NB деп белгілейміз, сондықтан халықтың жалпы саны $N = NA + NB$ болады. Енді $A(t)$ деп t уақыты кезінде өнімді қабылдаған ықпал етушілер тобы үлесін есептейік, ал $B(t)$ деп – t уақыты кезінде өнімді қабылдаған ізбасарлар тобы үлесін есептейік.

Ықпал етушілердің қабылдау жылдамдығы бұрынғы айырымдық теңдеуге бағынады, алайда мұндағы айрымашылық біз қабылдау жылдамдығы мен ықпал етушілер үшін әлеуметтік жұғу мәндерін rA және sA деп аламыз.

$$A(t) = A(t - \Delta t) + rA \cdot (1 - A(t - \Delta t)) \cdot \Delta t + sA \cdot A(t - \Delta t) \cdot (1 - A(t - \Delta t)) \cdot \Delta t$$

Ізбасарлардың қабылдау жылдамдығы айрықшаланады, себебі олар ықпал етушілердің де, ізбасарлардың да пікірін бағалайды. rB мен sB деп ізбасарлар үшін қабылдау жылдамдығы мен әлеуметтік жұғуды алайық. Тағы бір параметр, w (0-ден 1-ге дейін) ізбасарлардың ықпал етушілердің пікірін басқа ізбасарлардың пікірінен қаншалықты артық көретіндігін көрсетеді. Шекті жағдайларда, $w = 1$ ізбасарларға қатты ықпал ететін ықпал етушілер екендігін, және ізбасарлардың ықпал етпейтіндігін білдіреді. Басқа жағынан, $w = 0$ олар ықпал етушілердің ықпалына мүлдем түспейтіндігін, бірақ ізбасарлардың ықпалыны түсетіндігін білдіреді. $B(t)$ үшін айырымдық теңдеуді үш бөлікке бөлеміз.

1. Біріншіден, әлі өнімді қабылдамаған ізбасарлар арасындағы қабылдау жылдамдығының тұрақты құрамдас бөлігі, $A(t)$ үшін айырымдық теңдеудің бірінші бөлігіндегідей болады:

$$rB \cdot (1 - B(t - \Delta t)) \cdot \Delta t$$

2. Екіншіден, өнімді әлі қабылдамаған, және қабылдау туралы шешімі тұтынушы-ықпал етушілердің әлеуметтік жұқтыру ықпалында болатын ізбасарлардың үлесі.

$$w \cdot s_B \cdot \underbrace{A(t - \Delta t)}_{\text{Өнімді қабылдаған ықпал етушілер үлесі}} \cdot \underbrace{(1 - B(t - \Delta t))}_{\text{Өнімді қабылдамаған ізбасарлар үлесі}} \cdot \Delta t.$$

Жоғарыда берілген w – ізбасарлар басқа ізбасарлардың ықпалына қарағанда, ықпал етушілердің ықпалына көбірек түсетіндігінің көрсеткіші екендігін еске салайық.

3. Үшіншіден, өнімді әлі қабылдамаған ізбасарлар үлесі, олардың өнімді қабылдау туралы шешіміне басқа тұтынушы-ізбасарлардың әлеуметтік жұқтыруы ықпал етеді.

$$(1 - w) \cdot s_B \cdot \underbrace{B(t - \Delta t)}_{\text{Өнімді қабылдаған ықпал етушілер үлесі}} \cdot \underbrace{(1 - B(t - \Delta t))}_{\text{Өнімді қабылдамаған ізбасарлар үлесі}} \cdot \Delta t.$$

$1 - w$ құрамдас бөлігі дегеніміз - ізбасарлар ықпал етушілерге қарағанда басқа ізбасарлардың ықпалына түсетін ықтималдықтың көрсеткіші.

Бұл үш бөлікті қосып, біз өнімді қабылдайтын ізбасарлар үлесінің өсуін үлгілеудің айырымдық тендеуін аламыз.

$$\begin{aligned} B(t) &= B(t - \Delta t) + rB \cdot 1 - B(t - \Delta t)) \cdot \Delta t \\ &+ w \cdot s_B \cdot A(t - \Delta t) \cdot (1 - B(t - \Delta t)) \cdot \Delta t \\ &+ (1 - w) \cdot s_B \cdot B(t - \Delta t) \cdot (1 - B(t - \Delta t)) \cdot \Delta t \end{aligned}$$

Енді функцияны жазыңыз

productDiffusion2(inSize, imSize, rIn, sIn, rIm, sIm, weight, weeks, dt)

ол өнімді қабылдаудың осы диффузиялық үлгісін ықпал етушілер

мен ізбасарларды қосып іске асырады. Параметрлер алдыңғы функцияға ұқсас (бірақ олардың атаулары қысқартылған). Алғашқы екі параметр – ықпал етуші халықтың және сәйкесінше ізбасарлардың саны. Үшінші және төртінші параметрлер – қабылдау жылдамдығы (rA) және сәйкесінше ықпал етушілер үшін әлеуметтік жұғу (sA). Бесінші және алтыншы параметрлер – ізбасарлар үшін (rB және sB) сол мәндер. Жетінші параметр $weight$ – бұл w мәні. Сіздің функцияңыз екі графикті беруі тиіс. Бірінші графикте әр топ үшін өнімнің жаңа тұтынушылары мен уақыт ағынымен жаңа тұтынушылардың жалпы санының пайда болуын көрсетіңіз (бұрынғыдай, dt -ге нормалы бөлу арқылы). Екінші графикте әр топ үшін тұтынушылардың жалпы санының, сонымен қатар екі топтың да тұтынушыларының уақыт ағынымен бірге жалпы санының өзгеруін көрсетіңіз. Алдыңғы функцияға қарағанда, сіз графикте әр топтағы тұтынушылар үлесін емес, ал тұтынушылар санын көрсетуіңіз керек, яғни халықтың әр тобының әртүрлі көлемдері ескерілуі керек. (Оны істеу үшін, бөлшекті жай ғана халықтың сәйкес тобының көлеміне көбейтіңіз.) Өнімді осылай енгізуді үлгілеу үшін сіздің функцияңызды қолданатын бағдарламаны жазыңыз, тек мұнда енді 600 ықпал етуші және 400 ізбасар, ал халықтың жалпы саны 1000 болады. Ықпал етушілер үшін қабылдау жылдамдығы мен әлеуметтік жұғу бұрынғыдай ($rA = 0.002$ және $sA = 1.03$) болады, ал ізбасарлар үшін $rB = 0$ және $sB = 0.8$ мәндері беріледі. $w = 0.6$ мәнін қолданыңыз, ол ізбасарлар басқа ізбасарлардың пікіріне қарағанда ықпал етушілердің пікірін жоғары бағалайтындығын білдіреді.

Тапсырма 4.3.4 *Суретті сипаттап беріңіз және жаңа тұтынушылардың пайда болуындағы және тұтынушылардың ақырғы жалпы санындағы заңдылықтарды түсіндіріңіз. Сізге қызық көрінетін кез келген заңдылықтарға сілтеңіз.*

Тапсырма 4.3.5 *Енді $w = 0.01$ қойыңыз да үлгілеуді қайталаңыз. Жаңа суретті сипаттап нәтижелердің қалай және неліктен өзгертіндігін түсіндіріңіз.*

***Жоба 4.4 Қасқырлар мен бұғылар**

Сіз бұл жобаны іске асыру үшін 4.4. Тарауының материалын қарап шығуыңыз керек.

Жыртқыш-олжа үлгілері әдетте биологияда қолданылады, себебі олар мысалы, қасқырлар мен бұғылар, коала мен эвкалипт немесе адамдар мен

тунец арасындағы экологиялық байланыстардың кең шоғырын келтіре алады. Жыртқыштар популяциясының тіршілік етуінің қарапайым жолдары тек қана олжалар популяциясының бар болуына сүйенеді. Олжалар популяциясы, өз кезегінде, жыртқыштар жағынан бақыланып отырады.

1920 жж. Альфред Лотка және Вито Вольтерра бір бірінен тәуелсіз қазіргі кезде Лотка-Вольтерра деп аталатын теңдеулерді ұсынды, олар жыртқыш-олжа қатынастарын үлгілейді. Үлгі байланысқан дифференциалды теңдеулер жұбынан тұрады, олар екі популяция көлемінің уақыт ағынымен өзгеруін сипаттайды. Біз дискретті айырымдық теңдеулермен дифференциалды теңдеулерді аппроксимациялаймыз. Жыртқыштар - қасқырлар, ал олжалар – бұғылар делік. Бұғылар мен қасқырлар популяцияларының көлемдерін t айының соңына $M(t)$ және сәйкесінше $W(t)$ деп алайық. Қасқырлар мен бұғылар популяциясын сипаттайтын айырымдық теңдеулер мыналар болады:

$$M(t) = M(t - \Delta t) + bMM(t - \Delta t)\Delta t - dMW(t - \Delta t)$$

$$M(t - \Delta t)\Delta t W(t) = W(t - \Delta t) + bWW(t - \Delta t)M(t - \Delta t)\Delta t - dWW(t - \Delta t)\Delta t$$

мұндағы

- bM – бұғылардың туу көрсеткіші (бір айдағы)
- dM – бұғылардың өлімі, немесе қасқыр бұғыларды кезіктірген жағдайда оны өлтіренін көрсеткіш (бір айдағы)
- bW – қасқырлардың туу көрсеткіші, немесе бұғылардың өлу көрсеткіші $\times$ жем болған бұғы жаңа қасқырдың туылуына қаншалықты тиімді болады (бір айдағы)
- dW – қасқырлар өлімінің көрсеткіші (бір айдағы)

Осы теңдеулерді толығырақ қарастырайық. Бірінші теңдеуде $bM M(t - \Delta t)\Delta t$ құрамдас бөлігі - қасқырлар жоқ кездегі соңғы уақыт аралығында туылған бұғылардың таза саны болса, ал $dM W(t - \Delta t) M(t - \Delta t)\Delta t$ құрамдас бөлігі - соңғы уақыт аралығында бұғылар өлімінің саны болып табылады. Назар аударыңыз, бұл құрамдас бөлік қасқырлар санына да, және бұғылар санына да тәуелді: $W(t - \Delta t) M(t - \Delta t)$ – дегеніміз қасқыр-бұғылардың ықтимал кездесу саны, ал $dM\Delta t$ – қасқыр бұғыны кездестірген сайын Δt уақыт аралығында өлтіретін ықтималдық.

Екінші теңдеуде $bW W(t - \Delta t) M(t - \Delta t)\Delta t$ құрамдас бөлігі - бір айда туылған қасқырлар саны болып табылады. Назар аударыңыз, бұл сан сонымен қатар қасқыр санына да, және бұғылар санына да шамалас болып табылады, идеяның мәнісі мынада - қасқырлар азық артылған кезде көбірек ұрпақ береді. Жоғарыда айтылғандай, bW іс жүзінде екі

шамаға негізделеді: Бұғылар өлімінің көрсеткіші (себебі қасқырлар сәтті дамып, ұрпақ қалдыру үшін бұғыны жем қылуы керек) және қасқыр желінген бұғыдан алынатын қуатты жаңа қасқырды дүниеге келтіру үшін қаншалықты тиімді пайдалана алатындығы. $dWW(t - \Delta t)\Delta t$ құрамдас бөлігі - бұғыны болмаған айдағы қасқырлар өлімінің таза саны.

Бұл жобада сіз уақыт ағынымен қасқырлар мен бұғылар популяциясының динамикалық көлемдерін үлгілеу үшін осы айырымдық теңдеулерді қолданатын бағдарлама жазасыз. Жоба үш бөлімнен тұрады. Бірінші бөлімде сіз негізгі сценарийді үлгілеуге Лотка-Вольтерраның үлгісін қолданасыз. Екінші бөлімде сіз популяциялардың көлеміне қасқырларды аулау әсерін үлгілейсіз. Соңғы үшінші бөлімде сіз шынайы үлгілеуді жасайсыз, онда популяциялар көлемдері олардың мекендеу аяларында ғана болатын табиғи қорлармен шектелетін болады.

Бөлім 1: Лотка-Вольтерра үлгісін іске асыру

Python Функциясын жазыңыз

`PP(preypop, predpop, dt, months)`

ол жоғарыда келтірілген айырымдық теңдеулер арқылы осы жыртқыш-олжа үлгісін үлгілейді. `preypop` және `predpop` параметрлері дегеніміз - олжалар мен жыртқыштар популяцияларының бастапқы көлемдері ($M(0)$ және сәйкесінше $W(0)$), `dt` (Δt) – үлгіде қолданылатын уақыт аралығы, `months`– үлгілеу іске қосылатын ай саны (t үшін максималды шама). Параметрлер санын азайту үшін, сіз функцияңыздағы жергілікті ауыспалы шама арқылы туу мен өлімнің тұрақты коэффициенттерін белгілей аласыз. Алдымен мынаны көріңіз

```
birthRateMoose = 0.5      # bM
deathRateMoose = 0.02     # dM
birthRateWolves = 0.005   # bW = dM * 0.25 көлеміндегі тиімділік
deathRateWolves = 0.75    # dW
```

Сіздің функцияңыз, `matplotlib`-ті қолдана отырып, үлгілеуді орындаған сайын уақыт ағынымен қасқырлар мен бұғылардың популяция көлемдерінің өзгеру графигін құруы керек. `dt = 0.01` шамасында 5 жыл ішіндегі 500 бұғы мен 25 қасқыр арасындағы өзара қарым-қатынасты үлгілеу үшін сіздің `PP` функцияңызды шақыратын бағдарламаны жазыңыз.

Тапсырма 4.4.1 *Уақыт өткен сайын популяция көлемдеріне не болады? Бұл өзгерістер неліктен болады?*

Бөлім 2: Мұнда аңшылар келді!

Енді қасқырлар жергілікті малшылардың үй малына қауіп төндіріп, (мысалы, Вайомингте) және малшылар қасқырларды өлтіре бастайды делік. Бұл әсерді қасқырлар өлімінің көрсеткіші dW шамасының артуы арқылы үлгіленіз.

Тапсырма 4.4.2 *Бұл бұғы популяциясына қалай әсер етеді?*

Тапсырма 4.4.3 *Қасқыр популяциясы қандай өлім дәрежесінде бес жыл ішінде жойылып кетеді? Өлім дәрежесі 1-ге жететіндігін ескеріңіз. dW шамасын біртіндеп арттырып көріңіз де, не болатынын байқаңыз. Егер сіз бүкіл қасқырды өлтіру мүмкін емес деп ойласаңыз, әрі қарай оқыңыз.*

Бүкіл қасқырдың көзін жою мүмкін емес болып көрінеді, себебі сіз қолданатын теңдеулер ешқашан мәннің нөлге жетуін болдырмайды. (Неліктен?) Оны түзету үшін, популяция көлемінің кез келген мәнін, егер ол мысалы 1.0 секілді шекті шамадан түсетін болса нөлге қоя аламыз. (Шынында да, бізде қасқырдың бөлігі болуы мүмкін емес.) Бұл үшін келесі операторларды сіздің `for` цикліңіздің денесіне жыртқыштар мен олжалар популяцияларын арттырғаннан кейін қойып, қайтадан алдыңғы сұраққа жауап беріп көріңіз.

```
if preyPop < 1.0:
    preyPop = 0.0
if predPop < 1.0:
    predPop = 0.0
```

(`preyPop` және `predPop` сіз ағымдағы популяция көлемдері үшін қолданған атауларға ауыстырыңыз). Жуырда көретініміз - алғышқы екі нұқсаулар `preyPop` үшін 0-ді меншіктенеді, егер ол 1-ден аз болса. Келесі екі нұқсау да `predPop` үшін соны жасайды.

Бөлім 3: Шектеулі өсімді үлгілеу

Үлгілеу кезінде біз осы уақытқа дейін популяция шексіз өсе алады деген болжауға сүйендік. Мысалы, егер қасқырлар популяция-

сы жойылса, онда бұғылар популяциясы геометриялық прогрессияда өседі. Шындығында, экожүйе кеңстік, азық және т.б. Шектеулі болғандықтан популяцияның шектелген көлемін ғана қолдай алады. Бұл шектеу экожүйе сыйымдылығы деп аталады. Бұғылар үшін экожүйе сыйымдылығын қарапайым әдіспен - бұғылардың туу көрсеткішін бұғылар популяциясының көлеміне шамалас етіп қысқарту арқылы үлгілей аламыз. Әсіресе, МСС – бұғылар үшін экожүйенің сыйымдылығы болсын, ол экожүйе сыйдыра алатын бұғылардың максималды саны болып табылады. Бұғылардың туу көрсеткішінің құрамдас бөлігі $bMM(t - \Delta t)\Delta t$ мәнін мынаған ауыстырайық

$$bM(1 - M(t - \Delta t) / MCC)M(t - \Delta t)\Delta t.$$

Назар аударыңыз, қазір, бұғылар саны экожүйе сыйымдылығына жақындаған сайын, туу көрсеткіші бәсеңдей түседі.

Тапсырма 4.4.4 *Неліктен бұл өзгеріс бұғылар популяциясының көлемі экожүйе сыйымдылығына жақындаған сайын бұғылардың туу деңгейінің бәсеңдеуіне әкеледі?*

Бұл өзгерісті үлгіде, бұғыларға арналған экожүйе сыйымдылығын 750-ге қойып көрсетіңіз, және оны 10 жыл ішінде 500 бұғы мен 25 қасқыр үшін туу және өлім көрсеткіштерінің бастапқы мәндерін қойып, қайта іске қосыңыз.

Тапсырма 4.4.5 *Алғашқы нәтижесімен салыстырғанда қандай? Нәтиже нені көрсетеді? Бұғылар популяциясы 750 мәнінде экожүйе сыйымдылығына жетеді ме? Егер жетпесе, онда оны болдыру үшін туу немесе өлім көрсеткіштерінің параметрлерін қалай өзгерту керек?*

Тапсырма 4.4.6 *Туу және өлім көрсеткіштерінің түпнұсқаларын қайта қалпына келтіріп, аңшылықты қайта енгізіңіз; енді қасқырлардың популяциясы бес жыл ішінде жойылу үшін қасқырлардың өлім көрсеткіші қандай болуы керек?*

5 ТАРАУ

Жол тораптары

5.1 КЕЗДЕЙСОҚ ҚОЗҒАЛУ

Жүйенің кейбір сипаттарын жақсырақ түсіну үшін ғалымдарды жиі кездейсоқ үдерістерін үлгілеу қызықтырады. Мысалы, біз қандай да бір генетикалық ауытқудың теориялық ықтималдығын немесе бөлшектің броундық қозғалыста жүріп өткен болжалды қашықтығын бағалағымыз келді. Кейбір шамаларды бағлау үшін кездейсоқ үдерістер қолданатын алгоритмдер Монакодағы танымал казиноның атына қойылған *Монте-Карло үлгісі бойынша үлгілеу* деп аталады. Монте-Карло үлгілері

Ескертпе 5.1: Интервалдарды жазу формалары

a мен b арасындағы, a мен b қосатын нақты сандардың интервалын $[a, b]$ шамамен көрсету қабылданған. Салыстыру үшін, a мен b арасындағы, a мен b қосқанда, бүтін сандар жиынтығы $[a..B]$ деп белгіленеді. Мысалы, $[3, 7]$ кез келген нақты сандарды білдіреді, олар 3-тен көп немесе оған тең және 7-ден аз немесе оған тең, ал $[3..7] - 3, 4, 5, 6, 7$ бүтін сандар болып табылады.

a немесе b ақырғы нүктесі жоқ a мен b арасындағы нақты сандар интервалын белгілеу үшін біз ақырғы нүктенің шаршы жақпасын дөңгелек жақшамен алмастырамыз. Собылай $[a, b)$ дегеніміз $-a$ -ден b -ге дейінгі нақты сандар интервалы болып табылады, оның құрамында a бар болса, b жоқ болады. Осыған ұқсас, $(a, b]$ құрамында b бар, алайда a жоқ, ал (a, b) құрамында a да b да жоқ.

кездейсоқ сынақты бірнеше рет қайталайды және бұл сынақтарды мағыналы нәтижеге келтіру үшін орталайды; үлгіні бір ретт өткізу кездейсоқ сипатқа ие, сондықтан оның өзінің ешқандай мағынасы жоқ.

1827 жылы британдық ботаник Роберт Браун, су бетінде қалқыған гүл тозаңын микроскоппен бақылап, өте қызық нәрсені байқады. Тозаң дәндері жарылған кезде, олар айналасында бей-берекет қозғалуын жалғастырған өте ұсақ бөлшектерді шығарды. Бұл ерекше құбылыс броундық қозғалыс деп аталады, ол судың қозғалатын молекулаларымен соқтығысқандықтан пайда болды. Броундық қозғалыс қазіргі кезде сұйықтықтағы кез келген ұсақ бөлшектің (немесе молекуланың) қозғалысын суреттеу үшін қолданылады.

Біз екі өлшемде кездейсоқ қозғалатын бір бөлшек үшін броундық

қозғалыстың мәнін үлгілеп бере аламыз. Бұл үдеріс кездейсоқ қозғалыс деп аталады. Кездейсоқ қозғалыстар сонымен қатар нарықтық сауда және азық іздеуде жүрген жануарлар қылығы сияқты алуан түрлі құбылыстарды үлгілеу үшін, ірі әлеуметтік желілерді жасау үшін қолданылады. Осы тарауда біз Монте-Карло әдісі арқылы кездейсоқ адасуды үлгілеуді үйренеміз, бұл белгіленген уақыт ішінде бей-берекет қозағалатын бөлшектер ықтимал қашықтауды анықтау үшін керек.

Бәлкім, сіз қарапайым кездейсоқ қозғалысты 3.3.15 жаттығуында, тасбақаны экран бойымен жылжыту арқылы және әр қадамда кездейсоқ бұрылу бұрышын таңдау арқылы үлгілеген боларсыз. Енді қосымша шектеулері бар кездейсоқ адасу нұсқасын жасайық, мұнда бөлшек екі өлшемді тор бойынша қозғалуға мәжбүр болады. Біз әр қадамда бөлшектің төрт әлем бірінің жағына әрқайсысына бірдей ықтималдықпен қозғалуын қалаймыз. Кездейсоқ үдерістерді үлгілеу үшін бізге алгоритм немесе кездейсоқ сандарды беретін құрал қажет, ол кездейсоқ сандар генераторы (КСГ) деп аталады. Компьютер, 1 тарауда суреттелгендей, шынайы КСГ-ны іске асыра алмайды, себебі оның барлық іс-әрекеттері алдын ала белгілі. Сәйкесінше, компьютердің ішіне шынайы кездейсоқ физикалық құбылыстарды (мысалы, субатомдық кванттық флуктуациялар) анықтап жібере алатын, немесе ақылды алгоритм арқылы кездейсоқтықты келтіре алатын арнайы құрал кіруі керек, мұндай алгоритмді жалған кездейсоқ сандар генераторы (ЖКСГ) дейді. ЖКСГ кездейсоқ көрінетін, алайда шын мәнінде ондай болып табылмайтын сандар тізбегін береді.

random деп аталатын Python модулі random функциясында ЖКСГ болып табылады. random функциясы нөлден бірге дейінгі, бірлікті қоспай, жалған кездейсоқ санды қайтарады. Мысалы:

```
>>> import random
>>> random.random() 0.9699738944412686
```

(Сіздің нәтижеңіз өзгеше болады.) random функциясымен қайтарылатын $[0, 1)$ секілді нақты сандар диапазонын анықтау ыңғайлы. Сол жағындағы тік жақша 0 диапазонға кіретіндігін, ал оң жағындағы доғал жақша - 1 диапазонға кірмейтіндігін білдіреді. Ескерту 5.1 интервалдарды жазу формасын жөнінде толық түсіндіреді, егер оны әлі білмейтін болсаңыз.

Біздің бөлшекті кездейсоқ түрде әлемнің төрт жағының біріне жылжыту үшін, біз алдымен random функциясын r -ге $[0, 1)$ аралығында

кездейсоқ мән беру үшін қолданамыз. Содан кейін біздің ықтимал кездейсоқ сандарымыздың кеңістігін төрт бірдей аралыққа бөліп, олардың әрқайсысымен бағытты байланыстырамыз.

1. Егер $r \in [0, 0.25)$ мәнінде болса, шығысқа қарай .
2. Егер $r \in [0.25, 0.5)$ мәнінде болса, солтүстікке қарай .
3. Егер $r \in [0.5, 0.75)$ мәнінде болса, батысқа қарай .
4. Егер $r \in [0.75, 1.0)$ мәнінде болса, оңтүстікке қарай жылжымыз.

Енді кездейсоқ адасудың бір қадамын жасайтын алгоритм жазайық. (x, y) бөлшектерінің орнын x пен y (сондай ақ x пен y бөлшегінің *координатасы* деп аталатын) екі ауыспалы шамамен белгілейміз. r орналасқан интервалдың негізінде әрбір қозғалысты шарттау үшін Python `if` нұсқауын қолданамыз. `if` нұсқауы нұсқаулардың белгілі бір тізбегін тек қана кейбір шарт шынайы болса ғана орындайды. Мысалы, келесі нұсқаулар жалған кездейсоқ r мәнін иеленеді де, сосын егер $r \in [0, 0.25)$ мәнінде орналасса, бірінші жағдайды x көбейту арқылы іске асырады:

```
x = 0
y = 0
r = random.random()
if r < 0.25:           # if r < 0.25,
    x = x + 1          # move to the east
```

Ойлануға берілетін сұрақ 5.1 *Неге r -дің нөлден кем емес екендігін тексеру қажет емес?*

`if` операторын сондай ақ *шартты оператор* деп атайды, себебі бұрын қарастырған `while` циклдері тәрізді ол логикалық өрнекке сәйкес шешім қабылдайды. (Алайда `while` циклдеріне қарағанда `if` операторы бір рет қана орындалады.) Берілген жағдай үшін логикалық өрнек $r < 0.25$ шынайы болып табылады, егер $r - 0.25$ аз болса және кері жағдайда жалған болады. Егер логикалық өрнек шынайы болса, онда шар астында шегініп орналасқан нұсқау(лар) орындалады. Керісінше, егер логикалық өрнек жалған болса,

Математикалық символ	Python Символы
$<$	<code><</code>
$\leq$	<code><=</code>
$>$	<code>></code>
$\geq$	<code>>=</code>
$=$	<code>==</code>
$\neq$	<code>!=</code>

Кесте 5.1 Python алты салыстыру операторы.

шегініспен орналасқан нұсқау(лар) өткізіліп, шегініспен орналасқан нұсқаудан кейінгі нұсқаулар орындалады.

Екінші жағдайда, $r \in [0.25, 0.5)$ аралығына кіретіндігін тексеру үшін, r -дің 0.25-тен үлкен немесе тең екенін және r -дің 0.5-тен кіші екендігін тексеру керек. Алдыңғы сөйлемдегі "және" мағыналық мәні 1.4. Тараудағы бульдік мәнге ұқсас. Python-да бұл шарт, әрине, келесі түрде беріледі:

$$r \geq 0.25 \text{ and } r < 0.5$$

Python-да оператор $\geq$ "үлкен немесе тең" ($\geq$) дегенді білдіреді. Бұл 5.1. Кестеде берілген *салыстыру операторларының* (немесе *қатынас операторларының*) бірі, олардың кейбірінің Python-да екісимволды мәндері бар. ($=$ мәнге аса назар аударыңыз, ол теңдікті тексеру үшін қолданылады. Біз бұл операторларды кейінірек 5.4. Тарауында талқылаймыз.) Осы жағдайды біріншісіне қосатын болсақ, екі if опараторын аламыз:

```
if r < 0.25:                # if r < 0.25,
    x = x + 1                #   move to the east
if r >= 0.25 and r < 0.5:   # if r is in [0.25, 0.5),
    y = y + 1                #   move to the north
```

Енді егер r -ге 0.25-тен төмен шама берілсе, онда бірінші if нұсқауындағы шарт шынайы болады да, $x = x + 1$ орындалады. Кейін екінші if нұсқауындағы шарт тексеріледі. Алайда бұл шарт жалған болғандықтан, онда y ұлғайтылмайды. Басқа жағынан, r -ге 0.25 пен 0.5 арасында орналасқан шама берілді делік. Онда бірінші if нұсқауындағы шарт жалған болады, сондықтан $x = x + 1$ шегінісі бар нұсқау өткізіледі де екінші if нұсқауымен орындалу жалғасады. Екінші if нұсқауындағы шарт шынайы болғандықтан, $y = y + 1$ орындалатын болады.

Төрт жақтың бірін таңдау арқылы есепті аяқтау үшін, тағы екі if нұсқауын қосуға болады:

```
1 | if r < 0.25:                # if r < 0.25,
2 |     x = x + 1                #   move to the east
3 | if r >= 0.25 and r < 0.5:   # if r is in [0.25, 0.5),
4 |     y = y + 1                #   move to the north
```

```

5 | if r >= 0.5 and r < 0.75:      # if r is in [0.5, 0.75),
6 |     x = x - 1                  # move to the west
7 | if r >= 0.75 and r < 1.0:      # if r is in [0.75, 1.0),
8 |     y = y - 1                  # move to the south
9 | print(x, y)                   # executed after the 4 cases

```

г орналаса алатын әр интервалға бір-бірден бұл кодты орындаудың төрт мүмкін әдісі бар. Мысалы, $r \in [0.5, 0.75)$ интервалында орналасты делік. Алдымен 1 жолындағы if нұсқауын орындаймыз. if шарты жалған болғандықтан, 2-жолдағы шегінісі бар нұсқау $x = x + 1$, қалдырылады. Сосын 3-жолдағы if нұсқауын орындап, оның шартын тексереміз. Ол да жалған болғандықтан, 4-жолдағы шегінісі бар нұсқау, $y = y + 1$, қалдырылады. 5-жолдағы үшінші if нұсқауын орындауды жалғастырамыз. Оның шарты, $r \geq 0.5$ and $r < 0.75$, шынайы, сондықтан 6-жолдағы шегініс бар нұсқау $x = x - 1$, орындалады. Одан кейін 7-жолдағы төртінші if нұсқауына өтеміз де, оның $r \geq 0.75$ and $r < 1.0$ шартын тексереміз. Бұл шарт жалған болғандықтан, экранға x пен y мәндері шығарылатын 9-жолдың барлық шартты құрылымдарынан кейін орындалу жалғасады. Қалған үш жағдайдың барлығында, r басқа үш интервалда болған кезде, басқа шарт шынайы болады да, ал қалған үшеуі жалған болады. Осылайша, шегініс бар нұсқаудың төртеуінің тек біреуі ғана r -дің кез келген мәнінде орындалатын болады.

Ойлануға берілетін сұрақ 5.2 *Осынндай қадамдар тізбегі тиімді ме? Тиімсіз болса, онда қандай қадамдарды қалдыруға болады және қандай жағдайда?*

Код әрекеті дұрыс, алайда дұрыс шартты тапқан соң кейінгі шарттарды тексеру қажет емес көрінеді. Егер 4-тен көп жағдай болатын болса, бұл қосымша жұмыс маңызды болар еді. Мұнда тиімдірек құрылым келтірілген:

```

1  if r < 0.25:      # if r < 0.25,
2      x = x + 1    #   move to the east and finish
3  elif r < 0.5:     # otherwise, if r is in [0.25, 0.5),
4      y = y + 1    #   move to the north and finish
5  elif r < 0.75:    # otherwise, if r is in [0.5, 0.75),
6      x = x - 1    #   move to the west and finish
7  elif r < 1.0:     # otherwise, if r is in [0.75, 1.0),
8      y = y - 1    #   move to the south and finish
9  print(x, y)      # executed after each of the 4 cases

```

elif кілт сөзі - "else if" қысқартылған түрі ("егер басқаша болса") - elif-тен кейін жүретін шарт, тек алдыңғы шарттардың ішінде шынайысы болмаған жағдайда ғана тексерілетіндігін білдіреді. Басқаша айтқанда, біз төрт шарттың әрқайсысын егер шынайысын тапқанда ғана бірізді тексереміз, онда онымен байланысқан шегінісі бар нұсқа(лар) орындалады, ал топтағы қалған шарттарды *қалдырып кетеміз*. Біз сонымен қатар әрбір шарттың төменгі шебін тексеруді алып тастайы алатын едік(мысалы, екінші if нұсқаудағы $r \geq 0.25$), себебі егер біз elif шартына өтсек, онда алдыңғы шарттың жалған болғандығын білеміз, және сәйкесінше, r мәні алдыңғы жағдайда тексерілетін интервалдан үлкен немесе оған тең болуы керек.

Оны суреттеу үшін, алдымен 1-жолдағы бірінші if нұсқауындағы шарт шынайы делік, және 2-жолдағы шегінісі бар нұсқау орындалады. Онда қалған elif шарттарының ешқайсысы тексерілмейді де, 9-жолға алға қарай секіріп кетеді. Басқа жағынан, егер бірінші if нұсқауындағы шарт жалған болса, біз $r - 0.25$ -тен төмен болмау керектігін білеміз. Содан кейін 3-жолдағы elif шарты тексеріледі. Егер бұл шарт шынайы болса, онда 4-жолдағы шегінісі бар нұсқау орындалады, қалған шарттардың ешқайсысы тексерілмейді де, орындалу 9-шы жолдан бастап жалғасады. Егер екінші elif нұсқауының шарты сондай жалған болса, біз $r - 0.5$ -тен кем болуы керектігін білеміз. Кейін 5-жолдағы шарт тексеріледі, $r < 0.75$. Егер ол шынайы болса, онда 6-жолдағы шегініс бар нұсқау орындалады да, орындалу 9-шы жолдан бастап жалғасады. Ақырында, егер алғашқы шарттардың бір де біреуі шынайы болмаса, біз $r - 0.75$ -тен кем болмау керектігін білеміз, сонда 7-жолдағы elif шарты тексеріледі. Егер ол дұрыс болса, онда 8-жолда шегініс бар нұсқау орындалады. Және, кез келген жағдайда орындалу 9-жолдан бастап жалғасады.

Ойлануға берілетін сұрақ 5.3*r тиесілі болатын ықтимал*

интервалдардың төртеуінің әрқайсысы үшін қанша шарт тексеріледі?

Ойлануға берілетін сұрақ 5.4 Сіз ең соңғы нұсқадағы барлық *elif-терді if-ке жоғары етіп ауыстырасыз делік. Егер $r - 0.1$ мәнін қабылдаса не болады?*

Бұл кодты тағы да оңайлатуға болады. $r - [0, 1)$ мәніне кіру керек болғандықтан, соңғы шартты $r < 1.0$ тексерудің қажеті жоқ. Егер орындалу соңғы интервалға дейін жалғасса, онда $r - [0.75, 1)$ мәніне кіруі қажет. Сондықтан біз ешқандай тексерусіз соңғы $y = y - 1$ нұсқауын орындауымыз керек. Бұған соңғы *elif*-ті *else* операторына ауыстыру арқылы жетуге болады.

```
1 | if r < 0.25:      # if r < 0.25,
2 |     x = x + 1    # move to the east and finish
3 | elif r < 0.5:    # otherwise, if r is in [0.25, 0.5),
4 |     y = y + 1    # move to the north and finish
5 | elif r < 0.75:  # otherwise, if r is in [0.5, 0.75),
6 |     x = x - 1    # move to the west and finish
7 | else:           # otherwise,
8 |     y = y - 1    # move to the south and finish
9 | print(x, y)     # executed after each of the 4 cases
```

else операторы алдыңғы шарттардың ешқайсысы шынайы болмады деп хабарласа, *else* астындағы шегінісі бар нұсқау(лар) орындалуы керек.

Ойлануға берілетін сұрақ 5.5 *Енді сіз барлық elif-ті if-ке ауыстырыңыз делік. Егер $r - 0.1$ мәнін қабылдаса не болады?*

Егер біз (қателесіп) *elif*-тің екі кірісін *if*-ке жоғары етіп ауыстырсақ, онда соңғы *else* тек соңғы *if*-пен байланысқан болатын еді. Осылайша, егер $r - 0.1$ мәнін қабылдаса, барлық үш *if* шарттары шынайы болып, шегінісі бар алғашқы үш нұсқаулардың барлығы да орындалатын еді. Шегінісі бар соңғы нұсқау орындалмайды, себебі соңғы *if* шарты шынайы болатын еді.

Ойлануға берілетін сұрақ 5.6 *Оның орнына біз бөлшекті сызық бойымен кездейсоқ жылжытқымыз келді делік. Онда бізге тек қана екі жағдайды тексеру керек: $r - 0.5$ қарағанда аз немесе көп екендігін. Егер $r - 0.5$ -тен аз болса, x -ті көбейтеміз, немесе x -ті азайтамыз. Оны іске*

асыру үшін *if/else* нұсқауын жазыңыз. (Ілгері қарауға қызықпаңыз.)

Таңдаудың тек екі ғана нұсқасы болған жағдайларда, *else* жай ғана *if*-ке ілесе алады. Мысылы, егер біз кездейсоқ түрде бөлшекті сызық бойымен жылжитқымыз келсе, біздің шартты құрылымымыз мынадай түрді қабылдайтын еді:

```
if r < 0.5:      # if r < 0.5,
    x = x + 1    #   move to the east and finish
else:           # otherwise (r >= 0.5),
    x = x - 1    #   move to the west and finish
print(x)        # executed after the if/else
```

Енді біз топ бойымен кездейсоқ адасып жүрген көптеген қадамдарды үлгілеу үшін біздің циклдегі *if/elif/else* шартты құрылымымызды қолдануға дайынбыз. Оны төменде келтірілген *randomWalk* функциясы жасайды, содан кейін координаталардың басынан бастап бөлшек жүріп өткен қашықтықты қайтарады.

```
def randomWalk(steps, tortoise):
    """Топ бойымен кездейсоқ
    адасуды көрсетеді.
```

Параметрлер:

steps: Tortoise кездейсоқ адасу
қадамдарының саны: Turtle нысаны

Қайтарылатын мән: Бастапқы қалыптан
басталатын соңғы қашықтық"""

```
x = 0                # - (x, y) = (0, 0) y = 0
бастапқы мәнін береді
moveLength = 10      # тасбақа қадамының
ұзындығы for step in range(steps):
    r = random.random() # - if r < 0.25
    бағытының кездейсоқ таңдалуы:    #
    егер r < 0.25,
        x = x + 1 #   онда шығысқа қозғаламыз
    да elif r < 0.5 аяқтаймыз:    # болмаса,
    егер r - [0.25, 0.5) мәніне кірсе,
        y = y + 1 #   онда солтүстікке
    қозғаламыз да elif r < 0.75 аяқтаймыз:    #
```



```

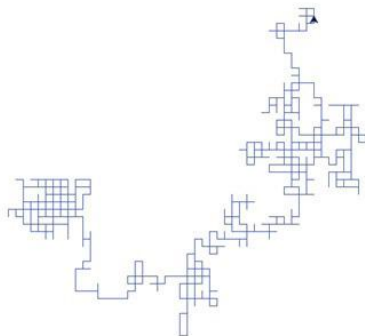
болмаса, егер r - [0.5, 0.75) мәніне кірсе,
    x = x - 1 # онда батысқа
қозғаламыз да else-ті аяқтаймыз: #
басқа жағдайларда,
    y = y - 1 # оңтүстікке жылжимыз да
tortoise.goto(x * moveLength, y * moveLength) аяқтаймыз
# бір қадамды суреттейді

```

```

return math.sqrt(x * x + y * y) # (0, 0)-ден басталатын
қашықтықты қайтарады

```



Сурет 5.1 1000 қадамнан тұратын, randomWalk функциясы арқылы алынған кездейсоқ қозғалыс.

Тор бойындағы қозғалысты көру оңай болу үшін, біз тасбақаны әрі қарай әрбір қадамда әр координатаны moveLength ауыспалы шамасына көбейту арқылы жылжытамыз. Кездейсоқ қозғалысты байқап көру үшін, main функциясын жазыңыз, ол жаңа Turtle нысанын жасап, 1000 қадам үшін randomWalk функциясын шақырады. Осындай сүлелердің бірі 5.1 суретте көрсетілген.

Монте-Карло әдісіндегі кездейсоқ адасу

Орташа алғанда, броундық бөлшек екі өлшемді кеңістікте координаталардың басынан бастап белгіленген қадам саны ішінде қаншалықты алыс жылжиды? Бұл сұраққа, мысалы, сұйықтықтың тарайтын жылдамдығын немесе жануарлар азық іздеп кететін аумақтың шақырымын келтіру арқылы жауап беруге болады.

Қандай да бір нақты кезде жүріп өтілген қашықтықтың мағыналық жүктемесі жоқ; бөлшек бастапқы қалыпқа қайтуы мүмкін, әрбір қадамда бастапқы қалыптан алшақтай алады немесе орташа бір әрекет жасайды.

Бұл нәтижелердің ешқайсысы *болжанатын*, немесе орташаланған жүйе қылығы туралы айтпайды. Керісінше, болжалды қылықты үлгілеу үшін, бізге көптеген кездейсоқ кезулердегі орташа қашықтықты есептеуіміз қажет. Мұндай үлгілеу түрі *Монте-Карло әдісі бойынша* үлгілеу деп аталады.

Біз randomWalk функциясын жиі шақыратын болғандықтан, үдерісті жылдамдатып, тасбақаның кездейсоқ кезуін визуалдауды өткізгіміз келіп отыр. Біз сурет салу үдерісін randomWalk параметрінің сапасына флагтың-айнымалы шамасын енгізу арқылы ескерте аламыз. Флаг-айнымалы шамасы логикалық мәнді қабылдайды және кейбір жұмыс режимін қосу немесе өшіру үшін қолданылады. Python-да логикалық шаманың екі ықтимал мәні True және False деп аталады (тізімдерге назар аударыңыз). randomWalk функциясында біз draw флаг-айнымалысын шақырамыз да оның мәнінің негізінде сурет салу үдерісіне тағы if нұсқауы арқылы әсер етеміз:

```
def randomWalk(steps, tortoise, draw):
    :
    if draw:
        tortoise.goto(x * moveLength, y * moveLength)
    :
```

Енді randomWalk шақырылғанда біздің үшінші аргументімізге True не False деген мағына береміз. Егер draw мағынасы True болса, онда tortoise.goto(...) орындалатын болады, ал егер draw мағынасы False болса онда қалдырылады.

Көптеген сынақтар ішіндегі орташа мәнді табу үшін циклдегі біздің randomWalk функциясын шақыруды жалғастырамыз және барлық қашықтықтарды қосу үшін сумматордың айнымалы шамасын қолданамыз.

```
def rwMonteCarlo(steps, trials):
    """Кездейсоқ кезудегі бастапқы қалыптан соңғы
    қалыпқа дейінгі болжалды қашықтықты іздеу
    үшін Монте-Карло әдісі бойынша үлгілеу.
```

Параметрлер:

steps: trials Кездейсоқ қозғалыстар

қадамының саны: Кездейсоқ қозғалыстар

саны

Қайтарылатын мән: Бастапқы қалыптан
басталатын орташа қашықтық ""

```
totalDistance = 0
for trial in range(trials):
    = randomWalk(steps, None, False)
    totalDistance = totalDistance + distance
return totalDistance / trials
```

Назар аударыңыз, біз randomWalk функциясында None мәнін (tortoise) екінші параметр үшін аргумент ретінде жібердік. False-ті draw параметрі реінде жіберген кезде tortoise-ге берілетін мән ешқашан қолданылмайды, сондықтан біз None-ді "жалған мән" бітеуіші ретінде жібереміз.

Ойлануға берілетін сұрақ 5.7 500 қадамнан тұратын кездейсоқ кездудің бес сынағында жүріп өтілген он түрлі орташа қашықтық бағасын циклде он рет rwMonteCarlo (500, 5) функциясын шақырып және экранға қайта қайта нәтижені шығара отырып алыңыз. Нені байқадыңыз? Қалай ойлайсыз, бес сынақ жеткілікті ме? Енді осы экспериментті 10, 100, 1000 және 10000 сынақпен өткізіңіз. Дұрыс нәтижеге қол жеткізу үшін сіздің ойыңызша қанша сынақ керек?

Қорытындысында біз қадам санынан функция түрінде жүріп өтілген қашықтық туралы тұжырым жасағымыз келеді. Басқа сөзбен айтқанда, егер бөлшек n қадамға жылжыса, ол орташа есеппен келесіні жүріп өте ме

$n/2$ немесе $n/25$ немесе $\sqrt{n}$ немесе $\log_2 n$ немесе ... ?

Эмпирикалық жолмен жауабын табу үшін,

бізге қадам санының әр түрлі көптеген мәндері үшін Монте-Карло әдісі бойынша үлгілеуін іске қосып, нәтижелер графигінен заңдылықты шығаруға тырысуымыз керек. Келесі функция оны 100, 200, 300 ... , maxSteps мәндеріне тең steps қадамымен жасайды, содан кейін нәтижелер графигін құрайды.

```
import matplotlib.pyplot as pyplot көмегімен импорттаймыз.
```

```
def plotDistances(maxSteps, trials):  
    """100, 200, ..., maxSteps қадамнан тұратын  
    кездейсоқ кезулерде орташа жүріп өтілген  
    қашықтықтың графигін құрады.  
  
    Параметрлер:  
    maxSteps: Trials графигін құру үшін максималды  
    қадам саны: әрбір үлгілеу кезіндегі Кездейсоқ  
    қозғалыстар саны  
  
    Қайтарылатын  
    мәндер: None"""  
  
    distances = [ ]  
    stepRange = range(100, maxSteps + 1,  
    100) for steps in stepRange:  
        distance = rwMonteCarlo(steps, trials)  
        distances.append(distance)  
  
    pyplot.plot(stepRange, distances, label =  
    'Үлгілеу') pyplot.legend(loc = 'center right')  
    pyplot.xlabel('Қадам саны')  
    pyplot.ylabel('Бастапқы қалыптан басталатын  
    қашықтық')  
    pyplot.show()
```

Ойлануға берілетін сұрақ 5.8*plotDistances(1000, 5000) іске қосыңыз. Сіздің ойыңызша, құрылған график қандай функцияны аппроксимациялайды?*

Ізделіп отырған функция шын мәнінде математикалық жолмен анықталып шамамен $\sqrt{n}$ -ге тең болды. Сіз оны эмпирикалық түрде осы функцияны

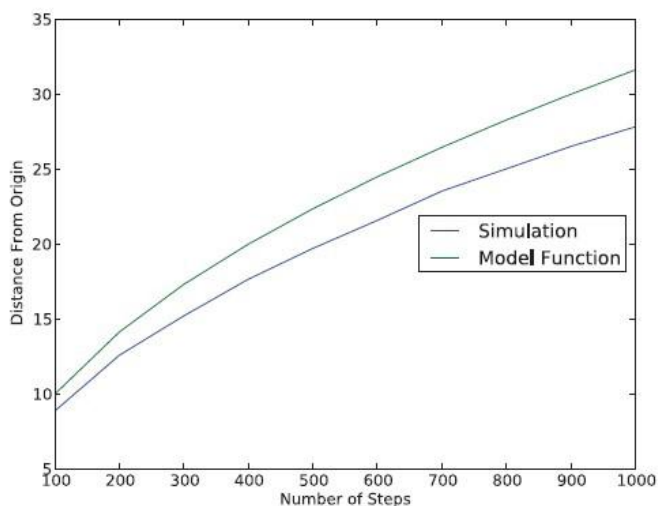
үлгіленген нәтижелер графигіне қою арқылы растай аласыз. Оны жасау үшін, келесі үш нұсқауды plotDistances функциясындағы сәйкес орындарға қойыңыз:

```
y = [ ] # before loop  
  
y.append(math.sqrt(steps)) # inside loop  
  
pyplot.plot(stepRange, y, label = 'Model Function') # after loop
```

Нәтижесі 5.2 суретте көрсетілген. Бұл нәтиже бізге бөлшектің бірлікті ұзындықтағы n қадамдарға кездейсоқ кезуінің нәтижесінде ол, орташа алғанда, басталу жерінен бастап шашамен $\sqrt{n}$ бірлікте

болтынын айтады. Алайда әрбір нақты жағдайда бөлшек едәуір жақын немесе едәуір алыс бола алады.

5.1. Ойлануға беріген сұрақтан біліп отырғаныңыздай, кез келген Монте-Карло аппроксимациясының сапасы сынақтардың санына тәуелді. Егер сіз әртүрлі сынақ санымен `plotDistances` тағы бірнеше рет шақырсаңыз, онда үлгілеу нәтижелерінің графигі



Сурет 5.2 Шаршы түбірдің функциясымен салыстырып, `plotDistances(1000, 10000)` құрған графигі.

көп сынақ санында "едәуір тегіс" болатынын байқайсыз. Алайда сынақтың көп саны әлбетте көп уақытты алады. Бұл есептерді шығару барысында бізге жиі тап болатын ымыраның тағы бір мысалы болып табылады.

Гистограммалар

Сынақ санының ұлғаюына қарай біздің орташа нәтижелеріміз біршама қисынды бола түседі, алайда жеке сынақтар неге ұқсайды? Оны анықтау үшін, дара Кездейсоқ қозғалыстар үшін қашықтықтар гистограммасын жасауға болады. Мәліметтерді жинауға арналған гистограм-

ма - дегеніміз бағаналы диаграмма болып табылады, ол элементтердің мәліметтерді жинақтау кезінде интервалдардың кейбір сандары бойымен таратылғандығын көрсетеді, оларды әдетте "қоржындар" немесе "бункерлер" деп атайды.

Келесі функция `rwMonteCarlo`-ға ұқсайды, айырмашылығы - оның цикл алдында бос тізімді инициалдап, оған әрбір итерацияда қашықтықтың бір мәнін қосып отыратындығында. Содан кейін ол бұл тізімді `hist` гистограммасының `matplotlib` функциясына жібереді. `hist` функциясының бірінші аргументі – сандар тізімі, ал екінші параметрі – біз пайдаланғымыз келетін бункерлер саны.

```
def rwHistogram(steps, trials):  
    """Белгіленген қадам санымен Кездейсоқ  
    қозғалыстар сынағының белгіленген саны  
    үшін гистограмма салыңыз.
```

Параметрлер:

```
    steps: trials Кездейсоқ қозғалыстар  
    қадамының саны: Кездейсоқ қозғалыстар  
    саны
```

Қайтарылатын
мәндер: None"""

```
    distances = [ ]  
    for trial in range(trials):  
        distance = randomWalk(steps, None, False)  
        distances.append(distance)
```

```
    pyplot.hist(distances, 75)  
    pyplot.show()
```

Ойлануға берілген сұрақ 5.9 1000 қадамнан тұратын 5000 кездейсоқ қозғалыстар сынағының гистограммасын құру үшін `rwHistogram` функциясын шақырыңыз.. Гистограмманың формасы қандай?

Жаттығулар

5.1.1. Кейде біз кездейсоқ кездудің белгілі бір бағыттағы бөлшектің қозғалу ықтималдығын бұрмалайтын жағдайларды көрсеткендігін қалаймыз (яғни, ауыртпалық күші, су ағынының немесе желдің күші). Мысалы, бізге гравитацияны ескеру қажет делік,

онда солтүстікке қозғалу ауыртпалық күшіне қарсы шынайы жоғарға қозғалуды үлгілейтін болады. Бұндай жағдайда біз солтүстікке жылжу ықтималдығын 0.15 дейін азайтып, ал оңтүстікке жылжу ықтималдығын 0.35 дейін ұлғайтқымыз келеді, ал басқа бағыттарды өзгертусіз қалдырамыз. Осындай функцияны іске асыру үшін randomWalk функци-ясын қалай өзгерту керек екендігін ұсыныңыз.

5.1.2. Мәселен, ауа-райының болжамы 70 % ықтималдықпен жауын-шашынның болатынын айтады делік. Функцияны жазыңыз

```
weather()
```

ол экранға 0.7 ықтималдықпен 'ЖАҢБЫР' (немесе соған ұқсас сөзді), болмаса 'КҮН ШУАҚТЫ!' деп шығарады.

Енді 0.66 ықтималдықпен қар жауады, 0.33 ықтималдықпен күн шуақты және 0.01 ықтималдықпен жауын болады деп көрсететін тағы бір нұсқаны жазу керек.

5.1.3. Функцияны жазыңыз

```
loaded()
```

Ол ауырлық ортасы жылжытылған ойын сүйегінің лақтырылуын келтіреді, оны лақтырған кезде 1 мен 6 жиірек түседі. 1 немес 6 түсу ықтималдығы 0.25 болуы керек. Функция random.random функциясы мен if/elif/else шартты құрылымын қолданып, roll айнымалы шамасына мән беріп, сосын roll мәнін қайтаруы керек.

5.1.4. Функцияны жазыңыз

```
roll()
```

ол бір ауырлық ортасы жылжытылған ойын сүйегіні лақтырылуын үлгілейді. Функцияны жазыңыз

```
diceHistogram(trials)
```

ол екі ойын сүйегінің лақтырылуының trials үлгілеп, экранға нәтижелер гистограммасын шығарады. (2–12 мәнді бункерлерді қолданыңыз.) Ненің түсуі көбірек ықтимал?

5.1.5. Монті Холл парадоксы – "Бәстесейік" ойын шоуы-на негізделген танымал ойжұмбақ, оны өрлену кезінде Монті Холл жүргізген. Сізге үш есіктің бірін таңдай керек. Біреуінің артында – авто-мобиль, ал қалған екеуінің артында – ешкілер. Сіз бір есікті таңдайсыз, сосын барлық үш есіктің артында не тұрғанын білетін Монті, міндетті түрде ешкі тұрған басқа бір есікті ашады. Содан кейін сіз өзіңіздің бастапқы есігіңізді қалдыра немесе ауыстыра аласыз. Сіз қайтер едіңіз (машинаны қалаған болсаңыз)?

Монте-Карло әдісі бойынша үлгілеу жазып, анықтауға болады. Алдымен, функцияны жазыңыз

montyHall(choice, switch)

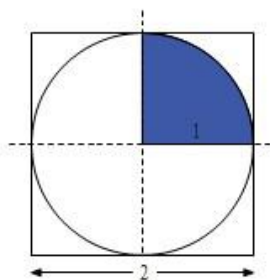
ол біздің бастапқы есікті таңдауымызға немесе оны ауыстыру шешіміне негізделіп ұтатынымызды немесе ұтылатынымызды анықтайды. Есіктер 0, 1 және 2 деп нөмірленген, ал автомобиль әрқашан 2 нөмірлі есіктің артында болады делік. Егер біз басында автомобильді таңдайтын болсақ, онда біз шешімді ауыстырамыз да ұтыламыз, егер олай істемесек ұтамыз. Керісінше жағдайда, егер біз бастапқыда көлікті таңдамай, шешімді ауыстырсақ ұтамыз, ал ауыстырмасақ ұтыламыз. Функция біз жеңсек True-ді, ал егер жеңілсек False-ты қайтару керек.

Енді функцияны жазыңыз

monteMonty(trials)

Ол есікті ауыстыру шешімінде ұту ықтималдығын табу үшін Монте-Карло әдісі бойынша сынақ саны берілген үлгілеуді орындайды. Сынақтың әрқайсысы үшін есіктің кездейсоқ санын (0-ден 2-ге дейін) таңдаңыз да өз таңдауыңызбен және switch = True-мен montyHall функциясын шақырыңыз. Неше рет ұтқаныңызды санаңыз да, осы санды сынақ санына бөліп қайтарыңыз. Алынған нәтижені түсіндіре аласыз ба?

5.1.6. π мәні Монте-Карло әдісі бойынша үлгілеу арқылы табыла алады. Сіз керегеге жағының ұзындығы 2-ге тең шаршыға іштей сызылған радиусы 1 болатын шеңбер салдыңыз делік.



Сосын көзіңізді жұмып шеңберге найза лақтырасыз. Найзаның барлығы шаршыға тиеді десек, шеңберге тиегін найзалардың үлесі шеңбер ауданы мен шаршы ауданының арақатынасын анықтайды. Біз шеңбер ауданы $C = \pi r^2 = \pi 1^2 = \pi$ құрайтынын, ал шаршы ауданы $S = 22 = 4$ тең екендігін білеміз. Сонда нақты арақатынас $-\pi/4$ болады. Найзаның жеткілікті санында, f , шеңберге тигендерінің (0-ден 1-ге дейін) үлесі, мына арақатынасты аппроксимациялайды: $f \approx \pi/4$, бұл дегеніміз $\pi \approx 4f$.

Есепті жеңілдету үшін, біз жай ғана найзаларды шеңбердің жоғарғы оң бұрышына лақтыра аламыз (жоғарыдағы боялған жер). Мұндағы

арақатынас сондай болмақ: $(\pi/4)/1 = \pi/4$. Егер шеңбердің осы төрттен бір бөлігін x және y осінде $(0, 0)$ нүктесіне шеңбер орталығын қойсақ, біздің найзаның барлығы енді 0-ден 1-ге дейінгі координаталары x пен y нүктелеріне тиетін болады.

Келесі функцияны жазу үшін осы идеяны қолданыңыз
`montePi(darts)`

ол π мәнін виртуалды найзаларды кездейсоқ тәсілмен көп рет лақтырып аппроксимациялайды, найзалар $[0, 1)$ интервалындағы координаталары x пен y нүктелеріне тиеді. Координаталардың басынан 1-ден аспайтын қашықтықта нүктеге найзаның тию санын есептеңіз және оның үлесін қайтарыңыз.

5.1.7. Жақсы, Жаман және Ашулы – үшжақты пистолеттік жарыс (кейде "трузль" деп аталады). Жақсы әрқашан өзінің нысанасына 0.8 ықтималдықпен тиеді, Жаман кез келген жағдайда нысанаға 0.7 ықтималдықпен тиеді, ал Ашулы – 0.6 ықтималдықпен тиеді. Әуелгіден, Жақсы Жаманға көзделеді, Жаман Жақсыға көзделеді ал Ашулы Жаманға көзделеді. Бандиттер бір мезгілде атады. Келесі ацналымда бандиттің әрқайсысы, егер тірі қалса, бұрынғы нысанасына көзделеді, егер ол нысана тірі болса, болмаса, басқа бандитке көзделеді, егер басқасы болса. Бұл тек қана бір бандит қалып, қалғандары өлгенше жалғасады. Олардың барлығының өлетін ықтималдығы қандай? Жақсының тірі қалатын ықтималдығы қандай? Жаман туралы не дейсіз? Ал Ашулы ше? Орташа раунд саны қандай? Функцияны жазыңыз

`goodBadUgly()`

ол пистолеттік үшжақты жарыстың бір оқиғасын үлгілейді. Сіздің функцияңыз Жақсы, Жаман, Ашулы тірі қалғандығына немесе сәйкесінше ешқайсысының тірі қалмағандығына тәуелді 1, 2, 3 немесе 0 мәндерін қайтаруы тиіс. Сосын функцияны жазыңыз

`monteGBU(trials)`

Ол Монте-Карло әдісі арқылы үлгілеуде жоғарыда берілген сұрақтарға жауап беру үшін сіздің `goodBadUgly` функцияңызды көп қайтара шақырады.

5.1.8. Төменде келтірілген жағдайлардың әрқайсысында келесі нұсқаулардың тізімімен экранға не шығарылады? Жауаптарыңызды түсіндіріңіз.

```

if votes1 >= votes2:
    print('Candidate one wins!')
elif votes1 <= votes2:
    print('Candidate two wins!')
else:
    print('There was a tie.')

```

(a) votes1 = 184 and votes2 = 206

(b) votes1 = 255 and votes2 = 135

(c) votes1 = 195 and votes2 = 195

5.1.9. Алдыңғы жаттығудың кодында проблема бар. Оны берілген есептерді басынан бастап дұрыс шешетіндей етіп түзетіңіз.

5.1.10. Айтылған жағдайлардың әрқайсысында келесі нұсқаулар тізімімен экранға не шығарылады? Жауаптарыңызды түсіндіріңіз.

```

majority = (votes1 + votes2 + votes3) / 2
if votes1 > majority:
    print('Candidate one wins!')
if votes2 > majority:
    print('Candidate two wins!')
if votes3 > majority:
    print('Candidate three wins!')
else:
    print('A runoff is required.')

```

(a) votes1 = 5 and votes2 = 5 and votes3 = 5

(b) votes1 = 9 and votes2 = 2 and votes3 = 4

(c) votes1 = 0 and votes2 = 15 and votes3 = 0

5.1.11. Алдыңғы есептегі кодты тиімдірек етіңіз де оны берілген есептерді басынан ақ дұрыс шешетіндей етіп өзгертіңіз.

5.1.12. Келесі нұсқаулар тізбегінен синтаксистік қателікті табыңыз.

```

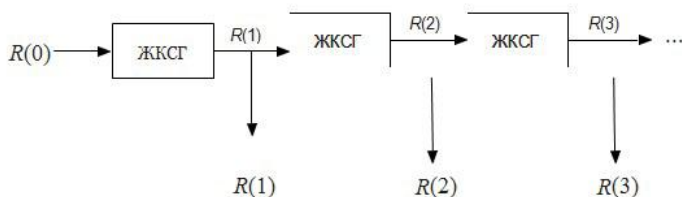
if x < 1:
    print('Something.')
else:
    print('Something else.')
elif x > 3:
    print('Another something.')

```

5.1.13. Келесі код нұсқауларының әрқайсысынан кейін result-қа берілетін мән қандай?

(a) `n = 13`
`result = n`
`if n > 12:`
`result = result + 12`
`if n < 5:`
`result = result + 5`
`else:`
`result = result + 2`

(b) `n = 13`
`result = n`
`if n > 12:`
`result = result + 12`
`elif n < 5:`
`result = result + 5`
`else:`
`result = result + 2`



Сурет 5.3 Қарапайым ЖКСГ әрекеті қағидасының суреті.

5.1.14. Функцияны жазыңыз

`computeGrades(scores)`

Ол `scores` тізіміндегі баллдармен берілген бағаларға сәйкес әріптік түрде тізімді қайтарады. Мысалы, `computeGrades([78, 91, 85])` - `['C', 'A', 'B']` тізімін қайтару керек. Функцияны тексеру үшін келесі функцияның көмегімен алынған бағалар тізімін қолданыңыз.

```
import random

def randomScores(n):
    scores = []
    for count in range(n):
        scores.append(random.gauss(80, 10))
    return scores
```

5.1.15. Гистограммада қолдануға арналған бункерлердің санын анықтау - ғылымның, өнердің бір бөлігі болып табылады. Егер сіз тым азғантай бункер қолдансаңыз, тарату формасын өткізіп алуыңыз мүмкін. Егер тым көп бункер қолдансаңыз, онда бос бункерлер саны көбейіп кетеді де, тарату формасы онша тегіс болмай қалады. Бункердің дұрыс санымен 10,000 сынақ үшін `rwHistogram` функциясында тәжірибе жасап көріңіз. Аса қажет жағдайда ғана тек 3 бункермен ғана және 1,000 бункермен гистограммалар жасаңыз. Сосын аралық мәндерді байқап көріңіз. Бункерлердің қандай саны тиімді болмақ? (Бәлкім, сіз осы сұрақты зерттегіңіз келедін шығар.)

*5.2 ЖАЛҒАН КЕЗДЕЙСОҚ САНДАР ГЕНЕРАТОРЛАРЫ

Жалған кездейсоқ сандар генераторының (ЖКСГ) бізге кездейсоқ тәрізді көрінетін сандар тізімін қалай жасайтындығына үңіліп қарайық. Ереже бойынша, ЖКСГ алгоритмі бұзғыш деп аталатын белгілі бір саннан басталады. $R(t)$ ретінде – t қадамында ЖКСГ қайтаратын санды алайық, сонда бұзғышты $R(0)$ деп белгілеуге болады. Біз қарапайым болып келетін, алайда мұқият таңдалған арифметикалық функцияны бұзғышқа қолданамыз да, біздің жалған кездейсоқ тізбектегі алғашқы $R(1)$ санын аламыз. Жалған кездейсоқ тізбектегі келесі санды алу үшін дәл сол арифметикалық функцияны $R(1)$ -ге қолданамыз да, $R(2)$ жасаймыз. Осы үдеріс $R(3)$, $R(4)$, ... Үшін бізге “кездейсоқ” сандар керек болғанша жалғаса береді, бұл 5.3 суретте көрсетілген.

Ойлануға берілген сұрақ 5.10 *Біз әрбір қадамда $R(t)$ -ді $R(t - 1)$ мәнінен есептейтінімізге назар аударыңыз. Бұл сізге таныс па?*

Егер сіз 4.4 тарауды оқыған болсаңыз, сіз бұдан айырымдық теңдеудің тағы бір мысалын байқайсыз. Айырымдық теңдеу дегеніміз - алдыңғы мәннің негізінде келесі мәннің әрқайсысын есептейтін функция болып табылады.

Лемердің жалған кездейсоқ сандарының генераторы деп атала-

тын қарапайым ЖКСГ алгоритмі, дүниеден өткен математик Деррик Лемердің құрметіне аталған . Доктор Лемер Берклиде Калифорния Университетінде мұғалім болды, және ENIAC алғашқы электронды компьютерінде бағдарламаларды іске қосатын адамдардың бірі болды. Лемердің ЖКСГ келесі айырымдық теңдеуді қолданады:

$$R(t) = a \cdot R(t - 1) \bmod m$$

мұндағы m - қарапайым сан, ал $a - 1$ және $m - 1$ арасындағы бүтін сан. Мысалы, $m = 13$, $a = 5$, және бұзғыш $R(0) = 1$ делік. Содан кейін

$$R(1) = 5 \cdot R(0) \bmod 13 = 5 \cdot 1 \bmod 13 = 5 \bmod 13 = 5$$

$$R(2) = 5 \cdot R(1) \bmod 13 = 5 \cdot 5 \bmod 13 = 25 \bmod 13 = 12$$

$$R(3) = 5 \cdot R(2) \bmod 13 = 5 \cdot 12 \bmod 13 = 60 \bmod 13 = 8$$

Сонымен бұл жалған кездейсоқ тізбек 5, 12, 8, ... мәндерімен басталады.

Ойлануға берілген сұрақ 5.11 *Келесі төрт мәнді тізбектеп есептеңіз.*

Келесі төрт мәндер мыналар $R(4) = 1$, $R(5) = 5$, $R(6) = 12$, және $R(7) = 8$. Сонымен енді тізбек - 5, 12, 8, 1, 5, 12, 8, ... болады.

Ойлануға берілген сұрақ 5.12 Осы тізбек кездейсоқ болып көріне ме?

Өкінішке орай, біз таңдаған параметрлеріміз қайта қайта 1, 5, 12, 8 тізбекшесін қайталай беретін тізбекті береді. Әрине, бұл мүлдем кездейсоқ емес, алайда біз оны m мен a өте мұқият таңдап өзгерте аламыз. Біз осыған кейін қайта ораламыз.

Іске асыру

Алдымен, Лемердің ЖКСГ айырымдық теңдеуінің негізіндегі 5.3. Суретінде берілген "қара жәшік" ЖКСГ іске асырайық. Бұл өте қарапайым функция!

```
def lehmer(r, m, a):
```

```
    """Лемердің ЖКСГ қолданып, келесі жалған кездейсоқ санды
    есептейді.
```

```
    Параметрлер:
```

```
    r: бұзғыш немесе m-нің
```

```
    алғашқы жалған кездейсоқ
```

```
    саны: a жай сан
```

```
    a: 1 және m - 1 арасындағы бүтін сан
```

Қайтарылатын мән: Келесі жалған
кездейсоқ сан""

```
return (a * r) % m
```

r параметрі - $R(t-1)$ білдіреді және функция арқылы қайтарылатын мән $R(t)$ -дің жалғыз мәні.

Енді біздің функциямыз жалған кездейсоқ сандарды жасауы үшін бізге m және a үшін мәндерін өзгертуіміз керек. Бірнеше жақсы деген нұсқаларын Стивен Парк пен Кейт Миллер 1988 жылы ұсынған болатын:

$$m = 2^{31} - 1 = 2,147,483,647 \text{ және } a = 16,807.$$

Осы параметрлер бар Лемер генераторы жиі *Парк-Миллердің жалған кездейсоқ сандар генераторы* деп аталады. Парк-Миллердің ЖКСГ жасау үшін біз lehmer функциясын бірнеше қайтара шақырып, әр кезде оған бұрын түзілген Парк-Миллердің m және a мәндері мен шамаларын келесідей жібере аламыз.

```
def randomSequence(length, seed):
```

```
    """Парк-Миллердің жалған кездейсоқ сандарының  
    тізімін қайтарады.
```

```
    Параметрлер:
```

```
    length: seed түзетін жалған кездейсоқ сандарының  
    саны: Бастапқы бұзғыш
```

```
    Қайтарылатын мән: Ұзындығы белгіленген Парк-  
        Миллердің жалған кездейсоқ  
        сандарының тізімі
```

```
    """
```

```
    r = seed
```

```
    m = 2**31 -
```

```
    1 a = 16807
```

```
    randList = [ ]
```

```
    for index in
```

```
        range(length): r =
```

```
            lehmer(r, m, a)
```

```
            randList.append(r)
```

```
    return randList
```

Әрбір итерацияда функция `randList` атты тізімнің соңына жаңа жалған кездейсоқ санды қосады. (Бұл тағы бір жинақтаушы тізім.) Сосын `randomSequence` функциясы бұл тізімді жалған кездейсоқ сандардың `length` мәнінен қайтарады. `randomSequence(100, 1)` нәтижесін экранға шығарып көріңіз.

Барлық қайтарылатын `lehmmer` мәндері m бөлшегінің қалдығы болғандықтан, олардың барлығы $[0 \dots m - 1]$ интервалында жатады. m шамасы едәуір еркін болғандықтан, ереже бойынша, кездейсоқ сандар мәндері $[0, 1)$ интервалынан қайтарылады, Python-дағы `random` функциясы дәл осылай істейді. Бұған жалған кездейсоқ санының әрқайсысын m -ге жай бөлу арқылы жетуге болады. Біздің `randomSequence` функциямызды сандар тізімін $[0 \dots m - 1]$ интервалының орнына $[0, 1)$ интервалынан қайтаруы үшін, біз r / m -ді `randList`-ге r орнына қоса аламыз.

Ойлануға берілген сұрақ 5.13 *Бұл өзгерістерді `randomSequence` функциясына енгізіп функцияны 3, 4 және 5 бұзғыштарымен шақырыңыз. Осындай бұзғыштар болса нәтижелердің бір-біріне ұқсас болатындығын болжайсыз ба? Олардың ұқсастығы бар ма?*

Шегініс ретінде, сіз бұзғышты, Python `random` функциясын қолданып, `seed` функциясын шақырып ораната аласыз. Бұзғыштың мәні шартты түрде ағымдағы уақытқа тәуелді.

Жуықтама кездейсоқ, алайда жаңғыртылатын тізбекті, бұзғыш шамасын орнатып, жасау мүмкіндігінің практикалық қолданысы едәуір көп. Мысалы, Монте-Карло әдісі бойынша жиі үлгілеу кезінде жай ғана дәл сол бұзғышты қолдану арқылы өте қызық тізбекті жасай алу мүмкіндігінің болуы пайдалы болады. Жалған кездейсоқ тізбектері сонымен қатар автомобиль есіктеріні электронды құлып тарында қолданылады. Сіз есікті ашу үшін кілтңіздегі кнопканы басқан сайын ол басқа кездейсоқ кодты жібереді. Алайда кілт пен автомобиль де бірдей бұзғышы бар КЖСГ пайдаланатындықтан, автомобиль кодтың дәлдігін анықтай алады.

Кездейсоқтықты тексеру

Сандар тізбегінің шын мәнінде қаншалықты кездейсоқ екендігін біз қалай анықтай аламыз? Шын мәнінде “кездейсоқ” болу дегеніміз не? Егер тізбек кездейсоқ болса, онда оны болжау немесе жаңғырту әдісі болмауы керек, бұл тізбектің өзінен басқа тізбекті қысқаша суреттей алатын әдістің жоқ екендігін білдіреді. Онда ЖКСГ шын мәнінде мүлдем кездейсоқ емес екендігі анықталады, себебі оны толықтай жаңғыртуға

болады және қарапайым формула арқылы суреттеуге болады. Алайда, практикалық мақсаттар үшін, біз сандарды алу үшін қолданылатын формуны білмесек, біз оларды немесе олардағы қандай да бір заңдылықтарды болжай алар ма едік деген сұрақ қоя аламыз?

Тексерудің қарапайым бір түрі - бұл мәндер тізімінен гистограмма құру.

Ойлануға берілген сұрақ .14 *Сіз $[0, 1)$ интервалындағы бір миллион кездейсоқ санның ішінен тізімге арнап гистограмма жасайсыз делік. Егер гистограмма 100 бункерден тұрса, ал ол бункерлердің әрқайсысы ұзындығы 0,01 интервалды білдіретін болса, әрбір бункерге шашамен қанша сан тағайындалуы керек?*

Егер тізбек кездейсоқ болса, бункердің әрқайсысында шамамен 1 %, немесе 10,000 сан болу керек. Келесі функция осындай гистограмманы жасайды.

```
import matplotlib.pyplot as plt
```

```
def histRandom(length):
```

```
    """Парк-Миллердің ЖКСТ көмегімен түзілген сандар  
    гистограммасын көрсетеді.
```

```
    Параметр:
```

```
    length: Түзілетін жалған кездейсоқ сандар саны
```

```
    Қайтарылатын
```

```
    мәндер: None"""
```

```
    samples = randomSequence(length,
```

```
    6) pyplot.hist(samples, 100)
```

```
    pyplot.show()
```

Ойлануға берілген сұрақ 5.15 histRandom(100000) шақырыңыз.. randomSequence функциясы кездейсоқ сандарды түзетінгіне ұқсай ма? Осындай сынақ сандардың шын мәнінде кездейсоқ екендіктеріне кепіл бере ме?

Кездейсоқтықты тексерудің көптеген әдістері бар, олардың көбісі өте күрделі болғандықтан оларды осында көрсете алмаймыз (біз бірнеше

онай сынақтарды жаттығу ретінде қалдырамыз). Бұл сынақтар біз суреттеген Лемердің ЖКСГ-ның жоғары сапалы кездейсоқ сандарды талап ететін жағдайларда орынсыз болып табылатын кейбір заңдылықтарды көрсетеді. Дегенмен, Python-дағы random функциясы танымал ЖКСГ алгоритмдері ішіндегі ең үздігі Мерсеннің ұйтқымасы атты алгоритмді қолданады, сондықтан сіз оны ойланбастан қолдана аласыз.

Жаттығулар

5.2.1. Біз ЖКСГ сапасын оны экранда кездейсоқ нүктелерді құру үшін қолданып визуалды түрде тексере аламыз. Егер ЖКСГ шын мәнінде кездейсоқ мәнге ие болса, онда нүктелер көзге көрінбейтін тәртіпте тегіс таралуы керек. Функцияны жазыңыз

testRandom(n)

Ол n кездейсоқ нүктелерін $[0, 1)$ интервалындағы x пен y координаларымен белгілеу үшін тасбақа графигін және random.random функциясын қолданады. Мұнда бірнеше тасбақа графигі орнатылған функцияның “қаңқасы” сіз үшін берілген. setworldcoordinates функциясын шақыру координаталар жүйесін төменгі сол жақ бұрыш $(0, 0)$ болатындай, ал жоғарғы оң жақ бұрыш $(1, 1)$ болатындай етіп қайта анықтайды. Тасбақа графигінің goto және dot функцияларын қолданып тасбақаны әрбір нүктеге жылжытып онда нүкте салыңыз.

```
def testRandom(n):
    """ your docstring here """

    tortoise = turtle.Turtle()
    screen = tortoise.getscreen()
    screen.setworldcoordinates(0, 0, 1, 1)
    screen.tracer(100)    # only draw every 100 updates
    tortoise.up()
    tortoise.speed(0)

    # draw the points here

    screen.update()      # ensure all updates are drawn
    screen.exitonclick()
```

5.2.2. 5.2.1 жаттығуын қайталаңыз, бірақ random.random функциясының орнына lehmer функциясын қолданыңыз. lehmer $[0 . . m - 1]$ интервалынан бүтін санды қайтаратындығын есте сақтаңыз, сіз оны $[0, 1)$ интервалындағы санға түрлендіруіңіз керек. Сонымен қатар,

сізге циклдің әр итерациясында `lehmer` функциясын екі рет шақыруыңыз қажет болады, сондықтан оған `r` мәндерін беретін әдісіңізбен абай болыңыз.

5.2.3. ЖКСГ тексерудің тағы бір сынағы - көптеген жалған кездейсоқ сандарды

$[0, 1)$ интервалында жасау және олардың орташа шамасы 0.5 болуын тексеру. Бұны тексеру үшін келесі функцияны жазыңыз

`avgTest(n)`

Ол `random` функциясы арқылы жасалған n кездейсоқ сандарының орташа шамасын қайтарады

*5.3 ЫҚТИМАЛДЫҚ ҮЛЕСТІРІМДІ ҮЛГІЛЕУ

Жалған кездейсоқ сандар генераторларын табиғи құбылыстарды үлгілеу кезінде әртүрлі ықтималдықтар үлестірімін аппроксимациялау үшін қолдануға болады. Ықтималдық үлестірімі кездейсоқ қарапайым құбылыстар жиынтығындағы $[0, 1]$ диапазонына кіретін мәндердің ықтималдығын иемденеді. Барлық қарапайым құбылыстардың сомасы мынаған тең болуы керек

1. Мысалы, 70% ықтималдықпен жауын-шашынды болжайтын ауа-райының болжамы - (болжанған) ықтималдық үлестірім болып табылады: Жауынның болу ықтималдығы - 0.7 және жауынның болмау ықтималдығы - 0.3. Математиктер ықтималдық үлестірімді табиғи құбылыстармен түсіндірмей, сандық мәндерге немесе мәндер интервалымен байланыстырып түсіндіреді, алайда біз әрқашан табиғи құбылыспен сандық мәндер арасындағы эквиваленттікті анықтай аламыз. Python `random` модулінде бірнеше ықтималдық үлестірімді қолдайды. Мұнда біз тек кең тараған екі түріне ғана көз жүгіртеміз: тегіс және нормалы таралу. Олардың екеуі де Python-ның `random` функциясына негізделген.

Тегіс үлестірім

Ықтималдық үлестірімнің ең қарапайымы және кең тараған түрі - тегіс үлестірім болып табылады, ол диапазондағы әрбір мәнге немесе диапазондағы ұзындыққа тең әрбір интервалға тең ықтималдықты береді. `random` функциясы секілді ЖКСГ мәндерді $[0, 1)$ интервалынан тегіс үлестірімге сәйкес қайтарады. Жалпыланған Python функциясы `random.uniform(a, b)` кездейсоқ мәндерді $[a, b]$ интервалынан тегіс үлестіріп қайтарады. Басқа сөзбен айтқанда, $[a, b]$ интервалындағы кез келген сан тең дәрежелі ықтималдықпен қайтарылады. Мысалы,

```
>>> random.uniform(0, 100)  
77.10524701669804
```

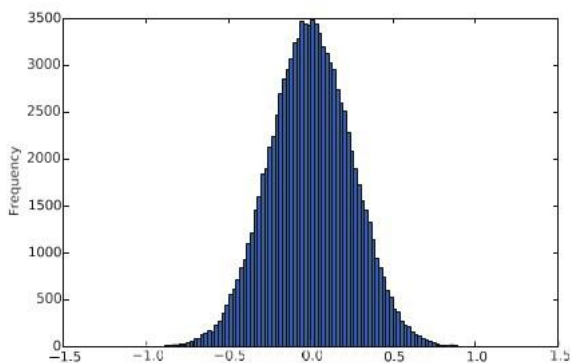
Нормалы үлестірім

Күнделікті және табиғи үдерістердің көбісін өлшеу, ереже бойынша, “қоңырау пішінді иректі”, ресми түрде нормалы немесе Гаусс үлестірімі деп аталатын үлестірімді көрсетеді. Нормалы үлестірім орташа шаманың айналасында шоғырланған мәндерді түзетін үрдісті сипаттайды. Мысалы, адамдардың бойы мен салмағын өлшеу шартты газдағы молекулалар жылдамдығы тәрізді нормалы үлестірімге келеді.

Нормалы үлестірімнің екі параметрі бар: орташа шама және стандартты ауытқу. Стандартты ауытқу - дегеніміз, интуицияға сәйкес, мәннің орташа шамасынан басталатын орташа қашықтығы болып табылады; төменгі стандартты ауытқу орташа шаманың айналасында тығызырақ шоғырлануға бейім келетін мәндерді береді, ал жоғарғы стандартты ауытқу орташа шамадан таралуға талпынатын мәндерді береді.

Нормалы үлестірім нақты сандар интервалдарына ықтималдықты орташа шаманың айналыснад шоғырланған сандардың орташа шамадан шалғай жатқан сандарға қарағанда жоғарырық ықтималдыққа ие болатындай етіп меншіктейді. Сондықтан, егер біз кездейсоқ мәндерді нормалы үлестірімге сәйкес генерацияласақ, біз орташа шамадан шалғай жатқан мәндерге қарағанда орташа мәнге жақын мәндерді алатымыздың ықтималдық үлесі зор.

Python-дағы `random.gauss` функциясы мәнді орташа мән және стандартты ауытқу белгіленген нормалы үлестірімге сәйкес қайтарады. Мысалы, келесі



Сурет 5.4 `random.gauss(0, 0.25)` функциясынан алынған 100,000

іріктемелер гистограммасы.

функция мәнді 0 мәні және 0,25 стандартты ауытқуы бар нормалы үлестірімге сәйкес қайтарады.

```
>>> random.gauss(0, 0.25)  
-0.3371607214433552
```

5.4 суретте random.gauss(0, 0.25) қайтаратын 100,000 мәннен тұратын histogram гистограммасы көрсетілген. Назар аударыңыз, пайда болу жиілігінің “қоңыраутәрізді ирегі” 0-ге тең орташа мәнге орталықтанған.

Орталық шектік теорема

Интуициялық түрде, нормалы үлестірім кең таралғандығы түсінікті болатын себебі - көптеген аддитивті кездейсоқ факторлардың жиынтығы болып табылатын құбылыстар орташалануға ұмтылады. Мысалы, адамның бойы көптеген гендердің қатары мен адамның денсаулығының нәтижесі болып табылады, сондықтан халықтың өсу үлестірімі нормалы шамаға ұмтылады. Бұл құбылысты сипаттау үшін, random функциясымен түзілген сандар сомасының гистограммасын құрайық. Төменде, sumRandom функциясы $[0, 1)$ интервалындағы n кездейсоқ сандарының сомасын қайтарады. sumRandomHist функциясы samples атты тізім құрайды, онда осы сомалардың trials болады, және содан кейін олардан гистограмма құрайды.

```
import matplotlib.pyplot as  
pyplot import random
```

```
def sumRandom(n):  
    """[0,1) интервалындағы n жалған кездейсоқ сандардың  
    сомасын қайтарады.
```

Параметр:

n: Түзілетін жалған кездейсоқ сандар саны

Қайтарылатын мән: $[0, 1)$ интервалындағы n жалған кездейсоқ сандар саны """

```

sum = 0
for index in range(n):
    sum = sum + random.random()
return sum

```

```

def sumRandomHist(n, trials):
    """n жалған кездейсоқ сандар сомасының гистограммасын
    көрсетеді.

```

Параметрлер:

n: trials сомасының әрқайсысындағы
жалған кездейсоқ сандарының саны:
түзілетін сомалар саны

Қайтарылатын
мәндер: None"""

```

samples = [ ]
for index in range(trials):
    samples.append(sumRandom(n)
    )
pyplot.hist(samples, 100)
pyplot.show()

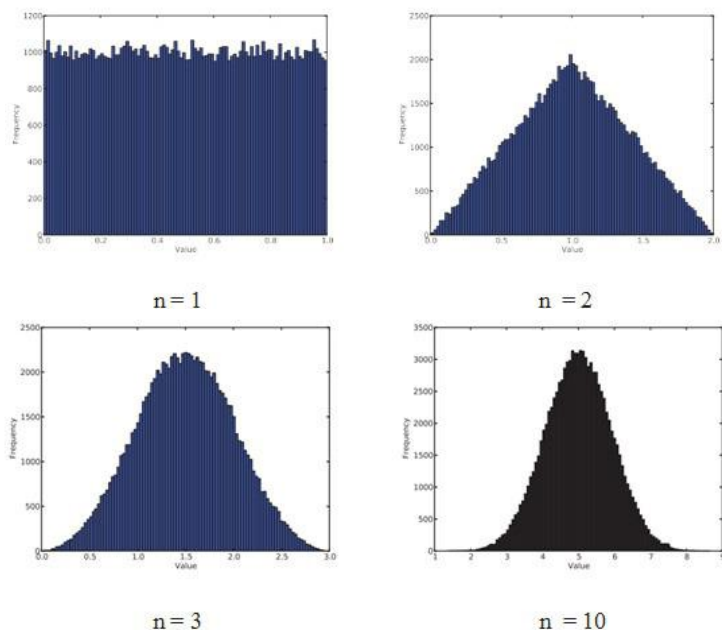
```

Ойлануға берілген сұрақ 5.16 100,000 сынақпен және 1, 2, 3, және 10-ға тең n мәндерімен sumRandomHist функциясын шақырыңыз. Нені байқадыңыз?

Біз sumRandomHist функциясын $n = 1$ мәнімен шақырған кезде, біз мәндердің болжалды тегіс үлестірімін аламыз, бұл 5.5. Суреттің жоғарғы сол жағында көрсетілген. Дегенмен, n-нің мәндерін ұлғайта отырып, біз мәндердің үлестірімінің өте жылдам түрде нормалы үлестірімге ұқсап кететіндігін байқаймыз, бұл 5.5 суретіндегі басқа кескіндерде көрсетілген. Сонымен қатар, үлестірімнің стандартты ауытқуы (интуициялық түрде, “қоңыраудың” ені) орташа шамаға қарағанда кішірейте түсетіндей.

Бұл құбылыстың математикалық атауы - орталық шектік теорема-сы делінеді. Интуициялық түрде, орталық шектік теорема көптеген кездейсоқ факторлардың жиынтық нәтижесі болып табылатын үдерістің

тәуелсіз өлшемдерінің кез келген үлкен жиынтығы нормалы үлестірім тәрізді көрінетінін білдіреді. Себебі нормалы үлестірім орташа шаманың айналасына шоғырланған,



Сурет 5.5 $n = 1, 2, 3, 10$ мәнді `sumRandomHist(n, 100000)` функциясының нәтижелері.

ол сондай ақ барлық өлшемдердің орташа шамасының ақиқаттық мәнге жақын болатынын білдіреді.

Ойлануға берілген сұрақ 5.17 5.9 ойлануға берілген сұрағына жауап беру үшін жүргізген экспериментті еске түсіріңіз. Гистограмманың пішіні нормалы үлестірімге ұқсас болуы керек еді. Неліктен кенін түсіндіру үшін сіз орталық шектік теореманы қолдана аласыз ба?

Жаттығулар

5.3.1. Шынайы кездейсоқ кездудің әр қадамда жылжуы нормалы үлестірімге сәйкес келеді. Әсіресе, әр қадамда, біз x пен y координаталарының екеуін де орташа шамасы 0 болатын нормалы

үлестірімге сәйкес ауыстыра аламыз. Себебі осы үлестіріммен жасалған мәндер оң да, теріс те болады, бөлшек әлеуетті түрде кез келген бағытқа жылжи алады. Қадамдардың көлемін кішірейту үшін, бізге мәселен 0,5 сияқты шағын стандартты ауытқуды пайдалану керек:

$$x = x + \text{random.gauss}(0, 0.5) \quad y = y + \text{random.gauss}(0, 0.5)$$

`randomWalk` функциясын бөлшектің қалпын бастапқының орнына осы тәсілмен жаңартатындай етіп өзгертіңіз. Сосын Monte Carlo әдісі арқылы үлгілеуді іске асыру үшін сіздің жаңа `randomWalk` функцияңызбен `rwMonteCarlo` және `plotDistances` функцияларын қолданыңыз. Бұрынырақта істегеніміздей

`plotDistances(1000, 10000)` функциясын шақырыңыз. Сіздің нәтижелеріңіз түпнұсқасынан қалай ерекшеленеді?

5.3.2. Функцияны жазыңыз

`uniform(a, b)`

ол `[a, b]` интервалындағы санды тек қана `random.random` функциясын қолданып қайтарады. (`random.uniform` функциясын қолданбаңыз.)

5.3.3. Бізге 0 мен 7 (қоса алғанда) арасындағы жалған кездейсоқ бүтін санды алу керек делік. Осы нәтижені алу үшін біз `random.random()` және `int` функцияларын қалай қолдана аламыз?

5.3.4. Функцияны жазыңыз

`randomRange(a, b)`

ол `[a . . b]` интервалындағы жалған кездейсоқ бүтін санды тек қана `random.random` функциясын пайдаланып қайтарады. (`random.randrange` немесе `random.randint` функциясын қолданбаңыз.)

5.3.5. Функцияны жазыңыз

`normalHist(mean, stdDev, trials)`

ол орташа шама мен стандартты ауытқуы белгіленген `gauss` функциясы арқылы қайтарылатын `trials` сынақ мәндерінің гистограммасын құрайды. Басқа сөзбен айтқанда, 5.4-суретті жаңғыртыңыз. (Бұл `sumRandomHist` функциясына қатты ұқсайды.)

5.3.6. Функцияны жазыңыз

`uniformHist(a, b, trials)`

ол `uniform` функциясы арқылы қайтарылатын `[a, b]` диапазонындағы `trials` сынақ мәндерінен тұратын гистограмманы жасайды. Басқа сөзбен, 5.5-суретте жоғарғы сол жағындағы гистограмманы жаңғыртыңыз. (Бұл `sumRandomHist` функциясына қатты ұқсайды.)

5.3.7. Функцияны жазыңыз

plotChiSquared(k, trials)

ол trials мәндерінің гистограммасын жасайды, ол мәндердің әрқайсысы орташа шамасы 0 және стандартты ауытқуы белгіленген random.gauss функциясы беретін мәндердің k шаршыларының сомасы болып табылады

1. (Бұл sumRandomHist функциясына қатты ұқсайды.) Нәтижесіндегі ықтималдық үлестірім - k бостандық дәрежелі хи-шаршылы үлестірім (χ^2 үлестірім)

деп аталады.

5.4 БУЛЬДІК ШАМАҒА ОРАЛУ

Тараудың басында айтқанымыздай, іс жүзінде тәжірибеде қолданылатын алгоритмдердің әрқайсысы кірістік мәліметтерге тәуелді іс әрекет ретін өзгертуге қабілетті болуы керек. Бұл шартты тәртіп 5.1.-тарауда пайдаланған дәл сол if нұсқауларының көмегімен іске асырылуы мүмкін. Осы тарауда біз сіздерге әрі қарай осы шартты нұсқауларды қолдану машығыңызды дамытамыз және осы шартты нұсқауларды басқаратын бульдік логиканы қолдануды үйретеміз. (Егер сіз бульдік логика жөнінде ұмыстып қалған болсаңыз, онда жалғастырмастан бұрын 1.4-тарауындағы жалпы ережелерді қарап шығыңыз.)

5.1-тарауда біз қарапайым

```
if r < 0.5:
    x = x + 1
else:
    x = x - 1
```

секілді if нұсқауларын r-дің кездейсоқ мәніне негізделген кездейсоқ кезуді бақылау үшін қолдандық. Осы жағдайдағы нәтиже $r < 0.5$ бульдік өрнектің мәніне негізделеді. Python-да логикалық өрнектер True немесе False мәндерінің бірін қабылдайды, олар біз 1.4. Тарауда қолданған 1 және 0 екілік шамаларына сәйкес келеді. True және False мәндерін экранға шығаруға, айнымалылар атауына меншіктеуге, және сандық мәндер тәрізді қолдануға болады. Мысалы, келесі мысалдарды байқап көріңіз де, әрбір нәтижені түсінетіндігіңізді тексеріңіз.


```

>>> print(0 < 1)
True
>>> name = 'Kermit'
>>> print(name == 'Gonzo')
False
>>> result = 0 < 1
>>> result
True
>>> result and name == 'Kermit'
True

```

“Екі рет тең” (==) операторы теңдікті тексереді; оның меншіктеу операторымен байланысы жоқ. Python интерпретаторы сізге іfnұсқауында байқаусыздан “бір рет тең” операторы басылған кезде ескертіп отырады. Мысалы, былай істеп көрелік:

```

>>> if r = 0:
    if r = 0:
        ^

```

Дегенмен, интерпретатор егер сіз меншіктеу нұсқауында қателесіп == қолдансаңыз қателікті байқамай қалады. Мысалы, былай істеп көрелік:

```

>>> r = 1
>>> r == r + 1    # increment r?
False

```

Бағдарлама экранға екінші нұсқаудың нәтижесі ретінде ештеңе шығармайды, және r ойланғанымыздай көбейтілмейді. Сондықтан абайлаңыз!

1.4-тараудан байқағанымыздай, логикалық өрнектер **және**, **немесе**, және **емес** деген *бульдік операторлармен* (немесе *логикалық операторлармен*) қосыла алады. 5.6 суретте Python белгілеріндегі үш бульдік операторға арналған ақиқаттық кестелері берілген. Кестелердегі a және b айнымалыларының атаулары кез келген логикалық айнымалы шамалар немесе өрнектер болып табылады.

Мысалы, бізге жанұяның жылдық табысының мәселен \$40,000 - \$65,000 аралығындағы шамада екендігін анықтау керек делік:

a	b	a және	a	a
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Сурет 5.6 Үш бульдик операторларға арналған аралас ақиқаттық кестесі.

```
>>> income = 53000
>>> income >= 40000 and income <= 65000
True
>>> income = 12000
>>> income >= 40000 and income <= 65000
False
```

53000 - income-ға меншіктелген кезде, income >= 40000 және income <= 65000 логикалық өрнектің екеуі де True болады, сондықтан income >= 40000 және income <= 65000 сондай ақ True болып табылады. Дегенмен, 12000 - income-ға меншіктелген кезде, income >= 40000 - False болады, ал income <= 65000 - True болады, сондықтан income >= 40000 және income <= 65000 енді False болады. Біз сонымен қатар бұл тексерісті шарттың мәнін жай ғана қайтаратын функцияға қоса аламыз:

```
>>> def middleClass(income):
    return (income >= 40000 and income <= 65000)
>>> middleClass(53000)
True
>>> middleClass(12000)
False
```

Ойлануға берілген сұрақ 5.18 Егер біз and мәнін or мәніне middleClass функциясында өзгертсек не болатын еді? Функция income-ның қандай мәндерінде False-ты қайтаратын еді?

Егер біз middleClass функциясын мынаған өзгертсек

```
def middleClass(income):  
    return (income >= 40000 or income <= 65000)    # ҚАТЕ!
```

ол ылғи True-ді қайтаратын еді! Үш ықтимал жағдайды қарастырайық:

- Егер income 40000 мен 65000 (қоса алғанда) арасында болса, ол True қайтарады, себебі or өрнегінің екеуі де True.
- Егер income < 40000, ол True қайтарады себебі income <= 65000 - True болады.
- Егер income > 65000, ол True қайтарады себебі income >= 40000 - True болады.

Қысқартылған есеп

and және or операторлары қысқартылған есеп деп аталатын іс әрекет көрсетеді, бұл кейбір жағдайларда пайдалы болады. and өрнегінің шынайы болуы үшін екі операнды да ақиқат болуына Python интерпретаторы бірінші өрнек жалған болса, and өрнегіндегі екінші операндты бағаламайды. Осыған ұқсас, өрнектің ақиқаттығы үшін or өрнегінің тек бір операнды ғана шынайы болса, Python интерпретаторы біріншісі ақиқат болса, or өрнегіндегі екінші операндты бағаламайды.

Мысалы, кез келген бір коммерциялық ұйындағы бас директордың жалақысы мен жұмыскердің орташа жалақысының арақатынасы “әділетті” шамадан аспайтындығын анықтайтын функция жазайық. Бұндай тексерістің нәтижесін қайтаратын карапайым функция төмендегіге ұқсас:

```
def fair(employee, ceo, ratio):
    """Бас директор мен жұмыскердің жалақыларының
    арақатынасы әділетті екендігін анықтайды.
```

Параметрлер:

```
    employee: ceo жұмыскердің
    орташа жалақысы: Бас
    директордың орташа
    жалақысы
    ratio: Әділетті арақатынас
```

```
    Қайтарылатын шама: ceo / employee әділетті
    екендігін көрсететін бульдік шама"""
```

```
    return (ceo / employee <= ratio)
```

Ойлануға берілген сұрақ 5.19 *Бұл функцияда проблема бар. Қандай?*

Бұл функция әрқашан да дұрыс істей бермейді, себебі жұмыскерлердің орташа жалақысы 0-ге теңейгенде, бөлу операциясы қателікке әкеледі. Сәйкесінше, біз бөлуден бұрын `employee == 0` екендігін тексеріп алуымыз қажет, егер сондай болса, `False` қайтарамыз (себебі жұмыскерлерге жалақы төлемеу ешқашан әділетті болмайды). Басқа жағдайда біз әділеттікті тексеру нәтижесін қайтарғымыз келеді. Келесі функция осы алгоритмді іске асырады.

```
def fair(employee, ceo,
    ratio): """ (құжаттау
    жолдары түсірілген)
    """

    if employee == 0:
        result = False
    else:
        result = (ceo / employee <= ratio)
    return result
```

Алайда, қысқартылған есеппен біз функцияны жалпы былай оңайлата

аламыз:

```
def fair(employee, ceo,
        ratio): """ (құжаттау
        жолдары түсірілген)
        """

    return (employee != 0) and (ceo / employee <= ratio)
```

Егер employee - 0 тең болса, онда (employee != 0) -False болады, ал функция False-ты есептеусіз қайтарады (ceo / employee <= ratio). Басқа жағынан алғанда, егер (employee != 0) ақиқат болса, онда (ceo / employee <= ratio) есептеледі, ал қайтарылатын мән осы нәтижеге тәуелді болады. Назар аударыңыз, and операторы қысқартылған есепті пайдаланбаған кезде бұлай болмас еді, шын мәнінде егер (employee != 0) жалған болып, (ceo / employee <= ratio) есептелген болса, бөлу нәтижесі “нөлге бөлу” деген қате беретін еді!

or операторына ұқсас механизмді суреттеу үшін, біз арақатынас әділетсіз болғанда True қайтаратын кері функцияны жазу керекпіз делік. Функцияның бірінші нұсқасы мынадай болмақ:

```
def unfair(employee, ceo, ratio):
    """Бас директор мен жұмыскердің жалақыларының
    арақатынасы әділетсіз екендігін анықтайды.
```

 Параметрлер:

```
        employee: ceo жұмыскердің
        орташа жалақысы: Бас
        директордың орташа
        жалақысы
        ratio: Әділетті арақатынас
```

```
    Қайтарылатын шама: ceo / employee әділетсіз екендігін
    көрсететін бульдік шама"""
```

```
Return: a Boolean indicating whether ceo / employee is not fair
"""
```

```
if employee == 0:
    result = True
else:
    result = (ceo / employee > ratio)
return result
```

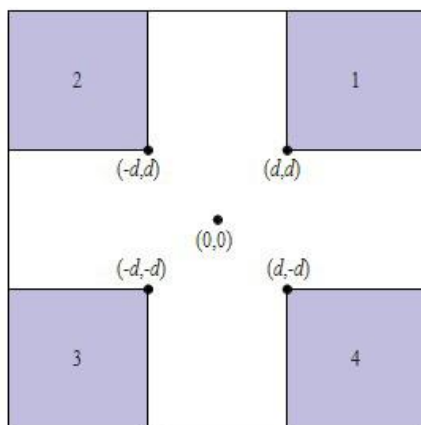
Алайда `or` операторы үшін қысқартылған есептің артықшылығын қолданып біз функцияны жалпы төмендегідей етіп оңайлата аламыз:

```
def unfair(employee, ceo,  
    ratio): """ (құжаттау  
    жолдары түсірілген) """  
  
    return (employee == 0) or (ceo / employee > ratio)
```

Бұл жағдайда егер (`employee == 0`) Ақиқат болса, өрнек түгелімен `True`-ді бөлуді тексерудің нәтижесін *бағалаусыз* қайтарады, бұл қателікті болдырмауға мүмкіндік береді. Басқа жағынан алғанда, егер (`employee == 0`) Жалған болса, бөлу тесті есептеледі де, ақырғы нәтиже осы тесттің нәтижесіне тең болады.

Күрделі өрнектер

Кейбір жағдайлар біз бұрын көрген өрнектерден анағұрлым күрделі келетін логикалық құрылымдардың қолданылуын талап етеді. Бұны суреттеу үшін, (x , y) координаталарындағы кездейсоқ кезіп жүрген бөлшектің төменде боялған аяларда көрсетілгендей экранның төрт бұрышының бірінде



екендігін қалай тексеру керектігін қарастырып көрелік.

Бұл шартты тексеруге арналған логикалық өрнекті екі түрлі (кем дегенде) әдіспен жазуға болады. Олардың бірі - бөлшектің бұрыштардың

әрқайсысында тәуелсіз болғандығын тексереді.

1. Бөлшектің 1 бұрышында болуы үшін, $x > d$ and $y > d$ Ақиқат \ болуы керек;
2. Бөлшектің 2 бұрышында болуы үшін, $x < -d$ and $y > d$ Ақиқат болуы керек;
3. Бөлшектің 3 бұрышында болуы үшін, $x < -d$ and $y < -d$ Ақиқат болуы керек; немесе
4. Бөлшектің 4 бұрышында болуы үшін, $x > d$ and $y < -d$ Ақиқат болуы керек.

Бұлардың кез келген бірі Ақиқат бола алатындықтан, біздің шарттың Ақиқат болуы үшін, функция түріндегі ақырғы тексеріс келесідей болмақ. (Төмендегі “кері слэш” (\) символы - жолдың жалғасу символы болып табылады. Ол ол аяқтайтын жолдың келесі жолда жалғасатындығын білдіреді. Бұл кейде өте ұзын болатын код жолдарын бөлуге ыңғайлы.)

```
>>> def corners(x, y, d):
    return x > d and y > d or x < -d and y > d or \
           x < -d and y < -d or x > d and y < -d
>>> corners(11, -11, 10)
True
>>> corners(11, 0, 10)
False
```

Біздің логика дұрыс болғанымен, бұл өрнек тек Python интерпретаторы and операторын or операторынан бұрын өндеген кезде ғана дұрыс болады. Болмаса, егер or операторын бірінші өндейтін болса, онда бірінші тексерілетін өрнек төменде қызылмен көрсетілген $y > d$ or $x < -d$ болады да, біз болжағанымыздай болмайды.

Операторлар		Суреттеу
1.	**	дәрежеге көтеру (экспонентті)
2.	+, -	унарлы плюс пен минус, мысалы, $-(4 * 9)$
3.	*, /, //, %	көбейту және бөлу
4.	+, -	қосу және алу
5.	<, <=, >, >=, !=, ==, in, not in	салыстыру операторлары
6.	not	ЛОГИКАЛЫҚ ЖОҚ
7.	and	ЛОГИКАЛЫҚ ЖӘНЕ
8.	or	ЛОГИКАЛЫҚ НЕМЕСЕ

Кесте 5.2 Оператор басымдылық тізімі, ең жоғарғыдан ең төменгіге дейін. Бұл 2.1 кестенің кеңейтілген нұсқасы.

$x > d$ and $y > d$ or $x < -d$ and $y > d$ or $x < -d$ and $y < -d$ or $x > d$ and $y < -d$

Осылайша, операторлардың басымдылық реті деп аталатын операторлар өңделетін тәртіпті түсіну, осындай едәуір күрделі шарттар үшін анағұрлым маңызды болады. Осы жағдайда, өрнек дұрыс, себебі and-тің орға қарағанда басымдылығы көбірек, бұл 5.2-суретте көрсетілген. Назар аударыңыз, 5.2-кестеге сәйкес салыстыру операторларының логикалық операторларға қарағанда басымдылыққа ие, бұл сондай ақ біздің өрнектің дұрыс болуына қажет шарт болып табылады. Егер күмәнді болсаңыз, сіз мүмкінді тәртіпті анық көрсету үшін жақшаны пайдалана аласыз. Жақшалар мұнда кез келген жағдайда, өрнекті түсінікте ете түсу үшін құпталады:

```
>>> def corners(x, y, d):  
    return (x > d and y > d) or (x < -d and y > d) or \  
           (x < -d and y < -d) or (x > d and y < -d)
```

Ал енді бұрышта болу шартын елестету үшін балама әдісті қарастырып көрелік: Нүкте бұрышардың бірінде болуы үшін, x -тің d -ден көп немесе $-d$ -ден аз деген екі мән де, және y -тің d -ден көп немесе $-d$ -ден аз деген мәндері дұрыс болуы қажет. Эквивалентті логикалық өрнек келесі функцияға енгізілген.

```
>>> def corners2(x, y, d):  
    return x > d or x < -d and y > d or y < -d  
>>> corners2(11, -11, 10)  
True  
>>> corners2(11, 0, 10)  
True
```

Ойлануға берілген сұрақ 5.20 *corners2* функциясын екінші рет шақыру қате нәтижені береді. Мәселені байқайсыз ба? (Бәлкім, сіз бұны сурет арқылы көрсетесіз.)

Операторлар басымдылығы тәртібін ескергенде, бұл жағдайда жақшалар қажет. Жақшаларсыз, төмендегі қызылмен боялған and өрнегі бірінші болып өңделеді, бұл біздің болжамымызға сәйкес келмейді.

$$x > d \text{ or } x < -d \text{ and } y > d \text{ or } y < -d$$

Осылайша дұрыс іске асыру мынау болмақ

```
>>> def corners2(x, y, d):
    return (x > d or x < -d) and (y > d or y < -d)
>>> corners2(11, -11, 10)
True
>>> corners2(11, 0, 10)
False
```

*Ақиқаттық кестелерін қолдану

Ойлануға берілген сұрақ 5.21 Осы өрнектің дұрыс екендігін қалай дәлелдей аламыз?

Кез келген логикалық өрнектің дұрыс екендігін дәлелдеу үшін, біз ол үшін ақиқаттық кестесін құрымыз қажет, және содан кейін әрбір жағдай үшін нәтженің болжанған нәтижеге сәйкес екендігін тексеру керек. Бұл өрнекте салыстыру операторы бар әрбір өрнек үшін бір бірден төрт дара логикалық “кіріс” бар. Ақиқаттық кестесінде біз олардың әрқайсысын орын үнемдеу үшін әріппен белгілеп көрсетеміз:

- $x > d$ - p деп беріледі,
- $x < -d$ - q деп беріледі
- $y > d$ - r деп беріледі, ал
- $y < -d$ - s деп беріледі.

Сонымен біз есептейтін соңғы өрнек $(p \text{ or } q) \text{ and } (r \text{ or } s)$ болады.

Төмендегі ақиқаттық кестеде алғашқы төрт бағана біздің төрт кірісімізді береді. Төрт кіріс үшін бізге кірістердегі мәндердің ықтималды ақиқаттық нұсқасының әрқайсысына бір бірден $2^4 = 16$ жол керек. Аққаттық мәндері жиынтығының барлығын тез жазып алудың бір айласы бар; сіз оны алғашқы төрт бағанадан анықтай аласыз ба байқап көріңіз. (Біз ақиқат және жалған мәндерін А және Ж деп қысқартып алдық. Қосымша көлденең сызықтар оқылуды жақсарту үшін ғана берілген.)

$x > r$	$x < -d$	$y > r$	$y < -d$			
p	q	r	n	$p \text{ or } q$	$r \text{ or } s$	$(p \text{ or } q) \text{ and } (r \text{ or } s)$
Ж	Ж	Ж	Ж	Ж	Ж	Ж
Ж	Ж	Ж	А	Ж	А	Ж
Ж	Ж	А	Ж	Ж	А	Ж
Ж	Ж	А	А	Ж	А	Ж
Ж	А	Ж	Ж	А	Ж	Ж
Ж	А	Ж	А	А	А	А
Ж	А	А	Ж	А	А	А
Ж	А	А	А	А	А	А
А	Ж	Ж	Ж	А	Ж	Ж
А	Ж	Ж	А	А	А	А
А	Ж	А	Ж	А	А	А
А	Ж	А	А	А	А	А
А	А	Ж	Ж	А	Ж	Ж
А	А	Ж	А	А	А	А
А	А	А	Ж	А	А	А
А	А	А	А	А	А	А

Бесінші бағананы толтыру үшін, бізге тек қана бірінші және екінші бағаналарға қарау керек. Бұл оғ өрнегі болғандықтан, егер осы жодағы бірінші және екінші бағаналарда кем дегенде бір А болса жолдың әрқайсысы үшін бесінші бағанаға біз А деп енгіземіз, ал кері жағдайда Ж деп енгіземіз. Осыған ұқсас, алтыншы бағананы толытур үшін, біз тек қана үшінші және төртінші бағаналарға қараймыз. Соңғы бағананы толтыру үшін, біз алғашқы екі бағанаға қараймыз, бесінші және алтыншыда А, немесе кері жағдайда Ж тұрған әр жолда А деп енгіземіз.

Біздің өрнектің дұрыстығын дәлелдеу үшін, біз соңғы бағанада жолының әрқайсысындағы ақиқаттық мәнінің сол жолдағы кірістік мәндерге қатысты дұрыс екенін тексеріп алуымыз қажет. Мысалы, ақиқаттық кестесінің (қызыл түсті) бесінші, алтыншы және сегізінші жолдарына қарап көрелік.

Бесінші жолдағы соңғы бағана өрнектің Жалған екенін көрсетеді. Сәйкесінше, бұл жағдай ол жолдағы алғашқы төрт мәндермен анықталған нүкте төрт бұрыштың ешқайсысында емес екендігін көрсететін тәрізді. Бұл ақиқаттық мәндері x

$> d$ - жалған, $x < -d$ - ақиқат, $y > d$ -жалған, және $y < -d$ -жалған; басқа сөзбен, x - $-d$ -ден аз және y - $-d$ мен d арасында деп көрсетеді. Бұл шекарадағы нүкте бұрыштың ешқайсысында емес, сондықтан бұл жол дұрыс.

Енді ақырғы өрнек дұрыс болып табылатын алтыншы жолға қарайық. Бұл жолдың алғашқы бағаналары $x > d$ - жалған, $x < -d$ -ақиқат, $y > d$ -жалған, және $y < -d$ - ақиқат; басқа сөзбен, x пен y екеуі де $-d$ -ден аз деп көрсетеді. Осы шекарадағы нүкте сол жақ бұрыштың төменгі бөлігінде болғандықтан, бұл жол да дұрыс.

Сегізінші жол өте қызық, себебі ол $y > d$ және $y < -d$ екеуін де ақиқат деп көрсетеді. Алайда, бұл әрине мүмкін емес. Сондықтан біз берілген мәнді *өзгешеленген ақиқат* дейміз. Тәжірибеде біз ондай нүкте ала алмаймыз, сондықтан бұл жол толығымен мағынасыз. Кестеде осындай жеті жол бар, сонда төрт бұрышқа сәйкес келетін тек төрт жол ғана қалады.

Бастапқы қалыпқа қайта оралу

Біз кейбір жадайларда шарттың нәтижесіне тәуелді әртүрлі мәндерді қайтартын алгоритмді (яғни, функцияны) қажетсінеміз. Ең қарапайым мысал - a және b екі санының максмалды шамасын табу. Егер a - b -ден кем емес болса, біз a -ді қайтарғымыз келеді; болмаса, b көбірек болуы керек, сол үшін біз b -ді қайтарамыз.

```
def max(a, b):
```

```
    """a мен b-дің максималды шамасын қайтарады.
```

```
    Параметрлер:
```

```
    a, b: Екі сан
```

```
    Қайтарылатын мән:
```

```
    Максималды шама"""
```

```
    if a >= b:
```

```
        result = a
```

```
    else:
```

```
        result = b
```

```
    return result
```

Біз бұл функцияны келесі нұсқаудың өзіне сәйкес мәнді қайтару арқылы біраз оңайлата аламыз

```
if/else:
```

Бірінші рет қараған кезде бір функцияда екі нұсқаудың болуы біртүрлі көрінуі мүмін, алайда бұл жерде барлығы қисынды. 3.5. Тарауды еске алатын болсақ, `return` функцияны аяқтайды және де функцияның

қайтарылатын мәнін меншіктейді. Бұл функцияда әрқашан да тек бір ғана return нұсқауы орындала алады дегенді білдіреді. Бұл жағдайда, егер $a \geq b$ - ақиқат болса, функция аяқталады да, a мәнін қайтарады. Болмаса, функция else шартын орындайды, ол b мәнін қайтарады.

Функцияның $a \geq b$ ақиқат болғандағы аяқталатыны біз оны одан сайын оңайлата алатындығымызды білдіреді: Егер орындалу if/else құрылымының if бөлшегінен кейін жалғасатын болса, бұл $a \geq b$ -жалған дегенді білдіруі қажет. Осылайша else артық болып табылады; функцияны келесідей оңайлатуға болады:

```
def max(a, b):  
    """ (құжаттау жолдары түсірілген) """  
  
    if a >= b:  
        return a  
    return b
```

Дәл осындай қағиданы екіден көп нұсқалы таңдау болатын жағдайларда қолданыла алады. Мәселен, біз пайыздағы бағаларды 0-4 шкаладағы баллдарға (яғни, орташа академиялық баллға) түрлендіретін функцияны жазуымыз керек делік. Оны әдеттегідей іске қосу келесідей болмақ:

```
def assignGP(score):  
    """Алынған бағаға эквивалентті баллды қайтарады.  
  
    Параметр:  
        score: 0 мен 100 аралығындағы баға  
  
    Қайтарылатын мән: Баллдың  
        эквивалентті мәні"""  
  
    if score  
        >= 90:  
        return  
        4  
    elif score >=  
        80: return 3  
    elif score >=  
        70: return 2  
    elif score >=  
        60: return 1  
    else:  
        return 0
```

Ойлануға берілген сұрақ 5.22 Неліктен біз әр жағдайда бағалардың жоғарғы шегін тексермейміз? Басқа сөзбен, $\text{score} \geq 80$ and $\text{score} < 90$ екінші шарты неліктен қажет емес?

Мәселен, $\text{score} = 92$ тең болды делік. Бұл жағдайда, бірінші шарт ақиқат болғандықтан, функция 4 мәнін қайтарады да аяқталады. Орындалу $\text{return } 4$ деген нұсқаудан әрі бармайды.

4. Сол үшін, келесі `elif`-тегі “`el`” артық болады. Басқа сөзбен, егер алдыңғы шарт ақиқат болғанында бұл жерде орындалу ешқашан іске асырылмайтын болғандықтан, егер алдыңғы шарт ақиқат болса, онда интерпретаторға бұл шартты тексеруді қалдырып кету керек деп көрсетудің қажеті жоқ.

Ал енді $\text{score} = 82$ -ге тең болды делік. Бұл жағдайда, бірінші шарт жалған болар еді, сондықтан біз алғашқы `elif` шартынан жалғастырамыз. Біз осы жерге дейін жеткендіктен, $\text{score} < 90$ (осыдан барып тексеріс жасалмайды) екендігін білеміз. Бірінші `elif` шарты ақиқат, сондықтан біз бірден 3 мәнін қайтарамыз. Енді функция аяқталған кезде, “`el`” мен екінші `elif`-тің еш қажеті жоқ болады. Басқа сөзбен айтқанда, егер алғашқы екі шарттың екеуі де ақиқат болғанында, орындалу ешқашан екінші `elif`-ке әкелмейтіндіктен, алғашқы шарт ақиқат болса бұл шартты тексеруді қалдырып кетудің қажеті жоқ болады. Іс жүзінде, біз “`el`”-дерді `elif`-тердің барлығынан және `else`-тен алып тастай аламыз да, ақырында ешқандай ешқандай кемістік болмайды.

```
def assignGP(score):
    """ (құжаттау жолдары түсірілген) """

    if score
        >= 90:
            return
            4
    if score
        >= 80:
            return 3
    if score
        >= 70:
            return 2
    if score
        >= 60:
            return 1
    return 0
```

Кейбір программисттер `elif` нұсқауларын қалдырғанды жөн көреді, оның да жөні бар. Келесі тарауларда біз екі әдісті де қолданамыз. Алайда, сіз алгоритмнің көптеген санына тап болған кезде, сіз осыған ұқсас кодты көруіңіз мүмкін, және оның неліктен дұрыс екендігін түсіну өте маңызды.

Жаттығулар

5.4.1. Функцияны жазыңыз

`password()`

ол қолданушы аты мен құпия сөзді сұрайды. Ол енгізілген қолданушы аты `alan.turing` дегенге сәйкес болса және құпия сөз `notTuring` деп енгізілсе, `True` деп қайтарады, ал керісінше болса `False` деп қайтарады.

5.4.2. Сіз ойын жасадыңыз делік, онда ойыншы баллы кем дегенде 100-ге жетсе жеңетін болады. Функцияны жазыңыз

`hasWon(score)`

ол ойыншы жеңіске жетсе `True` деп, ал жеңілсе `False` деп қайтарады.

5.4.3. Мәселен, сіз адам өз денсаулығын бақылай алатын тетік жасап шығардыңыз делік. Бұл тетіктің бір мақсаты бұл дене қызуын бақылау болады: Егер ол 97.9°F - 99.3°F диапазоны аралығынан асып кететін болса, онда адам аурып қалды дегенді білдіреді. Функцияны жазыңыз

`monitor(temperature)`

ол дене қызуын (Фаренгейтте) параметр ретінде алады да, егер дене қызуы қалыпты диапазонға жетсе `True` деп, ал керісінше жағдайда `False` деп қайтарады.

5.4.4. Функцияны жазыңыз

`winner(score1, score2)`

Ол 1 немесе 2 қайтарады, олар ойынның жеңімпазының Ойыншы 1 немесе Ойыншы

2 екендігін білдіреді. Ұпай саны көп болғаны жеңеді және сіз мұнда ешқандай тең ойын жоқ дей аласыз.

5.4.5. Алдыңғы жаттығуды қайталаңыз, бірақ бұл жолы тең ойын мүмкін деңіз. Тең ойынды көрсету үшін 0 -ді қайтарыңыз.

5.4.6. Сіздің фирмаңыз бір машинаны \$1500 бағамен сататын компьютерлер сатушысын іздейді. Сатушы 20-дан көп компьютер алатын болсаңыз сізге а 5% жеңілдік береді.

Функцияны жазыңыз

`cost(quantity)`

ол параметр ретінде сіз сатып алғыңыз келетін компьютердің са-

нын quantity алады, және оларды сатушыдан сатып алынатын бағасын қайтарады.

5.4.7. Алдыңғы жаттығуды қайталаңыз, бірақ бұл жолы үш қосымша параметр қосыңыз: Бір машинаның құны, жеңілдік алуға қажетті компьютер саны, және жеңілдік.

5.4.8. Көршілес жатқан қаладағы жылдамдықты арттыру үшін төленетін айыппұл көлемі \$50 және сағатына рұқсат етілген жылдамдықтан асырылған әр миля үшін \$5. Сонымен бірге, сағатына 90 милядан асырылатын барлық жылдамдыққа қосымша \$200 көлемінде айыппұл төленеді. Функцияны жазыңыз

fine(speedLimit, clockedSpeed)

ол айыппұл көлемін (немесе, clockedSpeed ≤ speedLimit болса 0) қайтарады.

5.4.9. Функцияны жазыңыз

gradeRemark()

Ол баллды сұрайды да, сосын төменде келтірілген кестеден сәйкес жауап (жол түрінде) қайтарады:

Балл	Жауап
96-	Өте жақсы
90-95	жақсы
80-89	жарайды
1-79	Нашар

5.4.10. Параметрлер ретінде екі бүтін санды мәндерді алатын және олар тең болмағанда олардың сомасын, және олар тең болғандағы туындысын қайтаратын функция жазыңыз.

5.4.11. Функцияны жазыңыз

amIRich(amount, rate, years)

Ол years жыл ішінде rate пайыздың жылдық мөлшерлемесі бойынша доллар amount сомасына пайыз аударарды. Егер сіздің ақырғы салымыңыз кем дегенде екі еселенген бастапқы сомаңызды құраса, True-ді қайтарыңыз; ал керісінше болса, False-ты қайтарыңыз.

5.4.12. Функцияны жазыңыз

max3(a, b, c)

ол a, b, және c параметрлерінің максималды шамасын қайтарады. Оны әртүрлі сандармен, сонымен бірге бірнеше тең сандармен міндетті түрде тексеріп көріңіз.

5.4.13. Функцияны жазыңыз

shipping(amount)

ол amount доллармен сатып алынған тауар негізінде ғаламтор-дүкенінен жеткізу құнын қайтарады. Компания \$100 дейінгі сатып алынатын тауар үшін \$6.95 көлемінде, ал \$100 асатын сомадан 5% белгіленген мөлшерлемені өндіріп алады.

5.4.14. а және b екі оң санынан бастай отырып, келесі санның әрқайсысы алдыңғы екі сан сомасының бірлік разряды болып табылатын тізбекті қарап көрелік. Мысалы, егер $a = 1$ және $b = 1$, тізбек мынадай болады

1, 1, 2, 3, 5, 8, 3, 1, 4, 5, 9, 4, 3, 7, 0, ...

Функцияны жазыңыз

mystery(a, b)

ол соңғы екі сан алғашқы рет a және b мәндерін қайталаған кезде тізбекті қайтарады. ($a = 1$ және $b = 1$ болғанда, функция 62-ні қайтаруы керек.)

5.4.15. Қытай зодиагы он екі жылдық айналымдағы әр жылды жануармен байланыстырады. Соңғы жылдардың дануарлары төменде берілген.

Жыл	Жануар	Жыл	Жануар
2004	Мешін	2010	Жолбарыс
2005	Өтеш	2011	Қоян
2006	Ит	2012	Айдаһар
2007	Қабан	2013	Жылан
2008	Тышқан	2014	Жылқы
2009	Бұқа	2015	Қой

Функцияны жазыңыз

zodiac(year)

Ол параметр ретінде төрт сандық жылды year (бұл өткендегі немесе болашақтағы кез келген жыл болуы мүмкін) алады да, жол түрінде сәйкесінше жануарды қайтарады.

5.4.16. Жыл төртке бөлінсе, онда толы жыл болады, бұған жүз жылдықты бастайтын жыл кірмейді, бұл жағдайда ол 400-ге бөлінуі керек.

Функцияны жазыңыз

zodiac(year)

Ол параметр ретінде төрт сандық жылды year (бұл өткендегі немесе болашақтағы кез келген жыл болуы мүмкін) алады да, жол түрінде сәйкесінше жануарды қайтарады.

5.4.16. Жыл төртке бөлінсе, онда толы жыл болады, бұған жүз

жылдықты бастайтын жыл кірмейді, бұл жағдайда ол 400-ге бөлінуі керек. Мысалы, 2012 және 1600 - толы жылдар болса, ал 2011 және 1800 толы жылдар емес. Функцияны жазыңыз

```
leap(year)
```

ол жылдың толы екендігін көрсететін бульдік шаманы қайтарады.

5.4.17. Функцияны жазыңыз

```
even(number)
```

ол сан жұп болса True қайтарады, керісінше болса False қайтарады.

5.4.18. Функцияны жазыңыз

```
between(number, low, high)
```

ол сан number [low, high] интервалында (төмегі low және жоғарғы high арасында (немесе тең) болса True қайтарады, ал керісінше болса False қайтарады.

5.4.19. Функцияны жазыңыз

```
justone(a, b)
```

ол a және b тек бір саны (бірақ екеуі бірей емес) 10-ға тең болса True қайтарады, ал

керісінше болса False қайтарады.

5.4.20. Функцияны жазыңыз

```
roll()
```

ол 5.1.3 жаттығуында (loaded функциясын шақыру арқылы) жасалған ауырлық ортасы жылжытылған екі ойын сүйегін лақтыруды үлгілейді, және сүйектер сомасы 7 немесе 11 құраса True қайтарады, немесе керісінше болса False қайтарады.

5.4.21. Келесі функция number бүтін санының жай санның шаршысы болып табылатындығын көрсететін бульдік шаманы қайтарады . Функцияны and өрнегінің қысқартылған есебінің басымдылығын қолдана отырып қайта жазыңыз.

```
def
```

```
    perfectSquare(numbe
```

```
    r): if number < 0:
```

```
        return False
```

```
    else:
```

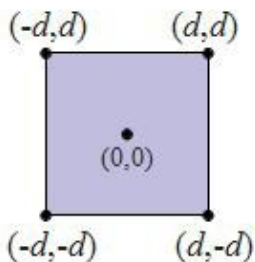
```
        return math.sqrt(number) ==
```

```
        int(math.sqrt(number))
```

5.4.22. 193 беттегі rwMonteCarlo функциясын қарастырып көрелік . Функция егер trials сынқа саны 0-ге тең болса не қайтарады?

Егер trials сынақ саны теріс болса не болады? Функцияға нұсқаулар косу арқылы осы екі сұраққа жауап беру әдісін ұсыныңыз.

5.4.23. (x, y) нүктесі төмендегі боялған өрісте (оның шегарасын қоса алғанда) болған кезде ғана True болатын логикалық өрнекті жазыңыз.



5.4.24. Ақиқаттық кестесін мына бульдік өрнектер $(x > d \text{ and } y > d) \text{ or } (x < -d \text{ and } y > d)$

және

$(y > d) \text{ and } (x > d \text{ or } x < -d)$

ұқсас екендігін көрсету үшін қолданыңыз.

5.4.25. Функцияны жазыңыз

`drawRow(tortoise, row)`

ол тасбақа графигін қолданып шахмат тақтасының бір жолын 8×8 қызылды-қара етіп салады. Егер row қатарының мәндері тақ болса, бұл қатар қызыл шаршымен басталады; керісінше жағдайда, ол қара шаршымен басталады. Сіз 3.3.5-жаттығуда жазған `drawSquare` функциясын қолданғыңыз келетін шығар. Сіздің функцияңыз үшін тек қана бір `for` цикл қажет болады және әр итерацияда тек қана бір шаршы салу керек.

5.4.26. Функцияны жазыңыз

`checkerBoard(tortoise)`

ол 8×8 қызылды-қара шахмат тақтасын салады, мұнда ол 5.4.25 жаттығуда сіз жазған функцияны қолданады.

5.5 ОЙЛАН ТАП ОЙЫНЫ

Бұл тарауда біз компьютермен “Мен бір санды ойладым” деген классикалық ойынды ойнауға мүмкіндік беретін алгоритмді құру арқылы, күрделі шартты құрылымдардан және `while` циклдерден тұратын

алгоритмдерді жазуды тәжірибе жүзінде көп қолданамыз. Компьютер (іс жүзінде ЖКСТ) 1 мен 100 арасындағы бір санды “ойлайды” да, ал біз оны ойлап табуымыз керек. Ойынды мына функциямен ойнаймыз:

```
import random
```

```
def guessingGame(maxGuesses):  
    """Ойлан тап ойынын ойнайды.      Адам ойыншы  
       компьютер ойлаған 1 - 100 аралығындағы санды  
       ойлап табуға тырысады.
```

Параметр:

maxGuesses: Рұқсат етілетін максималды жауап саны

Қайтарылатын
мәндер: None"""

```
secretNumber = random.randrange(1, 101)  
for guesses in range(maxGuesses):  
    myGuess = input('Please guess my number: ')  
    myGuess = int(myGuess)  
  
    if myGuess == secretNumber:      # win  
        print('You got it!')  
    else:                             # try again  
        print('Nope. Try again.')
```

randrange функциясы өзінің бірінші аргументінен аз болмайтын, алайда екінші аргументінен аз болатын кездейсоқ бүтін санды (range функциясы өз аргументтерін интерпретациялайтынына ұқсас) қайтарады. Функция 1-100 аралығындағы кездейсоқ бүтін санды таңдағаннан кейін, ол for циклге кіреді де, бұл бізге maxGuesses ретке дейін жауапты беруге мүмкіндік береді. Функция біздің болжамымызды input функциясы арқылы сұрайды, сосын myGuess-ке біздің жауабымызды жол түрінде меншіктейді. Себебі біз жауапты бүтін сан ретінде интерпретациялағымыз келетіндіктен, біз жолды түрлендіру үшін int функциясын қолданамыз. Функция myGuess ала салысымен, if/else нұсқауын пайдаланады да біздің жауабымыздың дұрыс немесе бұрыстығын айтады. Одан кейін, цикл бізден тағы да болжам сұрайды, бұл болжамдардың барлығын қолланып болғанша жалғасады.

Ойлануға берілген сұрақ 5.23 Ойынды guessingGame(20) функциясын шақырып ойнап көріңіз. Ол жұмыс істей ме? Біз жұмысты істеуді жалғастыратын тағы бірдеңе бар ма?

Сіз үш сұрақты байқаған боларсыз:

1. Егер біз барлық болжамымызды пайдаланбай тұрып, дұрыс жауап берген болсақ, онда цикл қайтадан қайталанады да, бізден тағы бір болжамды сұрайды. Оның орнына біз функцияның осы жерде аяқталуын қалаймыз.

2. Егер ойында біздің қате пікіріміз тым үлкен немесе тым кішкентай екендігін айтатын болса өте ыңғайлы болатын еді.

3. Егер біз maxGuesses барлығында да қате жауап берсек біз соңында 'Жоқ. Қайта байқап көріңіз.' деген жазуды функция аяқталмай тұрып көреміз. Алайда тағы байқап көретін мүмкіндік жоқ; оның орнына ол бізге ұтылдығыз деп айтуы керек.

Бұл сұрақтарға рет-ретімен жауап береміз.

1. Ұтыс болған жағдайда ойынның аяқталуы

Біздің ағымдағы for цикліміз басқа for циклдер тәрізді берілген сан рет қайталанады. Оның орнына біз циклдің бізге тағы мүмкіндік керек болғанша ғана қайталануын қайлаймыз. Сондықтан бұл while циклді шақыратын тағы бір мысал. while циклдің кейбір логикалық шарттардың дұрыс болғанында қайталанатындығын еске салайық. Бұл жағдайда, бізге циклдің біз құпия санды тапқанымызша қайталануы керек, басқа сөзбен айтқанда, myGuess != secretNumber болғанша. Алайда біз ағымдағы функциямызда айнымалы guesses санап отыратын барлық болжамымызды қолданып болғанда ойынды тоқтатуымыз керек. Сонымен бірге біз біздің while цикліміздің guesses < maxGuesses болғанша қайталануын қалаймыз. Біздің қайталанулар жалғасуы үшін бұл екі шарттың да дұрыс болуы керек болғандықтан, бізге қажетті while циклінің шарты бұл:

`while (myGuess != secretNumber) and (guesses < maxGuesses):`

Біз for цикліміздің орнына осы while циклді қоятындықтан, енді бізге индекстіка айнымалы шаманы қолымызбен басқару керек. Оны біз циклдің алдында guesses = 0 жауаптарды инициалдау арқылы және цикл денесінде guesses қосу арқылы орындаймыз. Осы өзгерістер ескерілген функция келесідей:

```

Def guessingGame(maxGuesses):
    """ (құжаттау жолдары
        түсірілген) """

    secretNumber = random.randrange(1,
    101) myGuess = 0
    guesses = 0
    while (myGuess != secretNumber) and (guesses < maxGuesses):
        myGuess = input('Менің санымды тапшы, өтінемін: ')
        myGuess = int(myGuess)
        guesses = guesses + 1          # қосу # of guesses if
        myGuess == secretNumber:      # ұтыс
            print('Дәл таптың!')
        else:                          # қайта тырысып көріңіз
            print('Жок. Қайта байқап көріңіз.')

```

Ойлануға берілген сұрақ 5.24 Назар аударыңыз, біз сондай ақ `myGuess = 0` циклден бұрын қостық.. Неліктен біз циклден бұрын `myGuess`-ке мән меншіктедік? 0 мәнінде бір ерекшелік бар ма? (Сыбырсөз: оны шолуға айналдыруға тырысыңыз.)

Егер біз `myGuess = 0` шолуға айналдырсақ, `while` циклі тұрған жолда келесі қатені байқаймыз:

UnboundLocalError: local variable 'myGuess' referenced before assignment

Бұл қате біздің `myGuess` атты белгісіз айнымалы шамасына сілтеме жасағанымызды білдіреді. Бұл атау Python интерпретаторына белгісіз, себебі біз оған `while` циклінің шартында сілтеместен бұрын оны анықтап алмадық. Сондықтан біз `myGuess`-ті `while` циклінен бұрын инициалдауымыз керек, сонда шарт бірінші тексеруде мәнге ие болады. Циклдің кем дегенде бір рет қайталанатындығына сенімді болу үшін, бізге `myGuess`-ке құпия бола алмайтын санды беріп инициалдауымыз керек. Жасырын сан 1-ден төмен болмайтындықтан, 0 осы мақсатта жарайды. Бұл логиканы екі ереженің бірімен жалпылануы керек, ол ережелерді `while` циклінің көмегімен алгоритм жасау үшін әрқашан есте сақтау керек:

1. Шартты циклдің алдына беріңіз. Әрқашан да шарттың мағыналы екенін және оның бірінші рет тексерілген кезінде ақ болжанғандай әрекет

ететініне көз жеткізіңіз.

2. Циклдің әр итерациясында, ақырында жалған болып кететін шартқа қарай жылжыңыз. Бұл ережені бұзу шексіз циклге әкеледі.

Біздің шарт ақырында жалған болатынын тексеру үшін біз оның қашан солай болатынын түсінуіміз керек. and өрнегі while циклінде жалғанға айналуы үшін, не(myGuess != secretNumber) жалған болуы керек, не (guesses < maxGuesses) жалған болуы керек.

Ойлануға берілген сұрақ 5.25 *Циклдің денесіндегі құрылымдар (myGuess != secretNumber) немесе(guesses < maxGuesses) ақырында жалған болуын қалайша қамтамасыз етеді?*

Жаңа болжамды сұрау бірінші бөлік жалған болатындай мүмкіндік жасайды, ал артылатын guesses жауаптарының саны екінші бөліктің соңында жалған болуын қамтамасыз етеді. Осылайша біз шексіз циклді ала алмаймыз.

Бұл пікір *Де Морган заңдарының* (19 ғ. Британдық математик Огастес Де Морганның құрметіне аталған) біріншісінде айтылған. Де Морганның екі заңы мыналар:

1. $\text{not } (a \text{ and } b) = \text{not } a \text{ or not } b$

2. $\text{not } (a \text{ or } b) = \text{not } a \text{ and not } b$

Сіз оларды 1.4.10 және 1.4.11 жаттығуларында көрген боларсыз. Бірінші заң *егер не a жалған, не b жалған болса, онда a and b* жалған дейді. Екінші заң *егер a мен b екеуі де жалған болса, онда a or b* жалған дейді. Біздің жағдайда қолданғанда, бірінші заң біздің while цикліміздің шартын терістеу мынадай болатындығын айтады (myGuess == secretNumber) or (guesses >= maxGuesses)

Бұл біз алған нәтижені алудың тек ресми түрі болып табылады.

2. Достық кеңестер

Қазір біз циклдің ішінде екі жағдайды іске асырудамыз: (1) біз ұтамыз және (2) біз ұтыламыз бірақ басқа жауап беруге мүмкіндік аламыз. Цикл достық реңк танытуы үшін біз екінші жағдайды екіге бөлуіміз керек: (2a) біздің болжам тым кішкентай болды, және (2b) біздің болжам тым үлкен болды. Біз оны іске асыруымыз үшін достық реңкі жоқ print('Жоқ. Қайта

байқап көріңіз.’) дегеннің орнына if/else деген сөзбен ауыстыра аламыз, ол екі жаңа шағын жағдайдың бірін таңдайды:

```
if myGuess == secretNumber:           # win
    print('You got it!')
else:                                   # try again
    if myGuess < secretNumber:         # too low
        print('Too low. Try again.')
    else:                               # too high
        print('Too high. Try again.')
```

Енді егер myGuess == secretNumber жалған болса, біз бірінші else шартты орындаймыз, оның денесіif/else құрылымының жаңа түрі болып табылады. Егер myGuess < secretNumber ақиқат болса, біз экранға сан тым аз деп; ал керісінше жағдайда, біз экранға сан тым үлкен деп шығарамыз.

Ойлануға берілген сұрақ5.26 Жоғардағы шартты құрылымыды оңайлататын тәсілді көріп тұрсыз ба?

Жоғардағы шартты құрылым тек қана үш қиылыспайтын мүмкіндіктер арасындағы таңдауға тең: (а) жауап жасырын санға тең, (б) жауап жасырын санға қарағанда аз, немесе (с) жауап жасырын санға қарағанда үлкен. Басқа сөзбен айтқанда, бұл мынаған тең:

```
if myGuess == secretNumber:           # win
    print('You got it!')
elif myGuess < secretNumber:          # too low
    print('Too low. Try again.')
else:                                   # too high
    print('Too high. Try again.')
```

3. Экранға ұтыс немесе жеңіліс туралы тиісті хабарлама шығару

Енді циклдің ішінде біз үш жағдайды қарастырамыз: (а) ұтыс, және (б) біздің жауап тым аз және біз басқа жауап беруге мүмкіндік аламыз, және (с) біздің жауап тым үлкен және біз тағы бір мүмкіндік аламыз. Алайда біз жауап беру мүмкіндігіміз таусылатын жағдайды қалдырып кеттік. Бұл жағдайда, біз “қайта байқап көріңіз” деген хабарламаның орнына ’өте нашар. Сіз жеңіліп қалдыңыз.’ деген сияқты сөздерді экранға шығаруды қалаймыз.

Ойлануға берілген сұрақ 5.27 *if/elif/else* нұсқауларын төрт жағдайдың барлығын дұрыс өңдейтіндей етіп қалай тиянақтылауға болады?

Ойында біз соңғы жауабымызды қате айтатын болсақ, онда дәл *if* шартын тексеруден бұрын екі зат ақиқат болуы керек: біріншіден, *myGuess* - *secretNumber* тең емес және, екіншіден, *guesses* - *maxGuesses* тең. Сонда біз бұл шартты *if/elif/else*-ке кірістіре аламыз:

```
if (myGuess != secretNumber) and (guesses == maxGuesses): # lose
    print('Too bad. You lose.')
elif myGuess == secretNumber: # win
    print('You got it!')
elif myGuess < secretNumber: # too low
    print('Too low. Try again.')
else: # too high
    print('Too high. Try again.')
```

Назар аударыңыз, біз бірінші *if* шартын *elif* нұсқауына берген себебіміз - біз төрт хабарламаның тек біреуі ғана экранға шығарылғанын қалаймыз. Дегенмен, бұл құрылымның анағұрлым қарапайым түрі бар:

```
if myGuess == secretNumber: # win
    print('You got it!')
elif guesses == maxGuesses: # lose
    print('Too bad. You lose.')
elif myGuess < secretNumber: # too low
    print('Too low. Try again.')
else: # too high
    print('Too high. Try again.')
```

Жаңа шартты екінші орынға ауыстырып, біз бірінші *elif*-ке жақындасақ, онда біз *myGuess != secretNumber* екенін білеміз деген ақиқатты қолдана аламыз. Сондықтан бізге оны қосудың қажеті жоқ екендігі анық.

Бұл жағдайды орындайтын үшінші бір әдіс бар, ол едәуір көрнекі. Назар аударыңыз, алғашқы екі жағдайдың екеуі де тек бір рет қана және бағдарламаның соңында болмақ. Сондықтан оларды циклден *кейін қойған жөндірек болады*. Осы арқылы біз бұл екі оқиға және *while* циклі шартының екі бөлігінің арасында нақты параллель жүргіземіз. Бұрын талқылап өткеніміздей, *while* циклінің шартын терістейтін келесі

(*myGuess* == *secretNumber*) or (*guesses* >= *maxGuesses*)

Сондықтан цикл аяқталған кезде осы жағдайдың екеуінің кем дегенде біреуі ақиқат болады. Назар аударыңыз, дәл осы екі оқиға ұтыс пен жеңілісті анықтайды: Егер бірінші бөлік дұрыс болса, біз жеңеміз; егер екінші бөлік дұрыс болса, біз жеңілеміз. Сондықтан біз ұтыс/жеңіс деген нұсқауларды циклден кейін қойып, экранға while циклі шартының қай бөлігі жалған

болғанының негізінде қайсы нұсқауды шығару керектігін шешеміз:

```
if myGuess == secretNumber:      # win
    print('You got it!')
else:                             # lose
    print('Too bad.  You lose.')
```

Енді осы жағдай өтіп кеткенде, циклдің денесінде “қайта байқап көріңіз” деген хабарламаның бірі экранға шықпастан бұрын бізге жаңа мүмкіндік (while циклі шартының көрінісі) беріле ме екендігін тексеру қажет болады:

```
if (myGuess != secretNumber) and (guesses < maxGuesses):
    if myGuess < secretNumber:      # too low
        print('Too low.  Try again.')
    else:                           # too high
        print('Too high.  Try again.')
```

Ойлануға берілген сұрақ 5.28 Жоғарыда келтірілген екі if нұсқауын төмендегідей етіп бір нұсқауға біріктіруге неліктен дұрыс болмайды?

Осы өзгерістер ойынның соңғы нұсқасына қосылған, олар төменде келтірілген.

```
import random
```

```
def
```

```
    guessingGame(maxGuesses
): """ (құжаттау жолдары
    түсірілген) """
```

```
    secretNumber=random.randrange(1,
    101) myGuess = 0
```

```
    guesses = 0
```

```
    while (myGuess != secretNumber) and (guesses < maxGuesses):
        myGuess = input('Менің санымды тапшы, өтінемін: ')
        myGuess = int(myGuess)
```

```

    guesses = guesses + 1          # пайдаланылған
    мүмкіндіктерді # қосу

    if (myGuess != secretNumber) and (guesses < maxGuesses):
        if myGuess < secretNumber:      # guess is too low
            print('Too low. Try again.') # give a hint
        else:                            # guess is too high
            print('Too high. Try again.') # give a hint

    if myGuess == secretNumber:          # win
        print('You got it!')
    else:                                # lose
        print('Too bad. You lose.')

def main():
    guessingGame(10)

main()

```

Ойын кезінде ең жақсы стратегияны ойлап табыңыз. Әртүрлі стратегияға қанша мүмкіндік керек? 5.5.7 жаттығуында сіз ойынға арналған үш түрлі стратегияны салыстыру үшін Монте-Карло әдісі бойынша үлгілеуді жазуыңыз керек.

Жаттығулар

5.5.1. Функцияны жазыңыз

ABC()

ол А, В, немесе С таңдауды сұрайды және while циклін 'А', 'В', немесе 'С' жолын алып кеткенше сұраныстарды жіберуді жалғастыру үшін қолданады.

5.5.2. Функцияны жазыңыз

numberPlease()

ол 1 - 100 (қоса алғанда) аралығындағы бүтін санды сұрастырады және while циклін осы диапазоннан санды алып кеткенше сұраныс жіберуді жалғастыру үшін пайдаланады.

5.5.3. Функцияны жазыңыз

differentNumbers()

ол екі түрлі санды сұрастырады. Функция while циклін екі сан әртүрлі болғанша сандар жұбын сұрастыруды жалғастыра беру үшін пайдаланады.

5.5.4. Функцияны жазыңыз

`rockPaperScissorsLizardSpock(player1, player2)`

ол Тас-қағаз-қайшы- кесіртке-Спок¹ ойынында кім жеңгенін анықтайды. `player1` және `player2` параметрінің әрқайсысы - 'тас', 'қағаз', 'қайшы', 'кесіртке', немесе 'Спок' мәні жазылған жол болып табылады. Функция 1 мәнін 1-ші ойыншы жеңсе қайтарады, ал 2-ші ойыншы жеңсе -1 мәнін, немесе 0 мәнін тең ойын болса қайтарады. Келесі негізгі бағдарламамен ойын ойнап өз функцияңызды тексеріңіз:

5.5.5. Функцияны жазыңыз

`yearsUntilDoubled(amount, rate)`

ол `amount` соманың жылдық пайыздық мөлшерлемеде және жыл сайынғы капиталданудан екі еселенген кезде артылғанға дейін жылдар санын қайтарады. `while` циклін қолданыңыз.

5.5.6. Бұршақ сандар - келесі жай алгоритмнен алынатын сандар тібегі болып табылады. Алдымен, кзе келген оң бүтін санды таңдаймыз. Сосын, мына ережеге қайта-қайта сүйенеміз: Егер ағымдағы сан жұп болса, оны екіге бөлеміз; керісінше болса, оны үшке көбейтеміз де бірлік қосамыз. Мысалы, біз 3 деген бастапқы бүтін санды таңдадық делік. Онда бұл алгоритм келесі тізбекті шығарады:

3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1 . . .

Сіз әлдеқашан бастап көрген кез келген бастапқы бүтін сан үшін тізбек ылғи бірлікке жетеді де, сосын шексіз 4, 2, 1 тізбегін қайталайды. Бұл заңдылықты ешкім дәлелдемеген болса да, ол кез келген бүтін сан үшін қамтитыны қызық!

Функцияны жазыңыз

`hailstone(start)`

```
def main():
```

```
    player1 = input('Player1: ')
```

```
    player2 = input('Player2: ')
```

```
    outcome = rockPaperScissorsLizardSpock(player1, player2)
```

¹ережелерді мына сайттан көруге болады [http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock for the rules](http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock_for_the_rules).

```

if outcome == 1:
    print('Player 1 wins!')
elif outcome == -1:
    print('Player 2 wins!')
else:
    print('Player 1 and player 2 tied.')

```

ол баспаға бұршақ сандар тізбегін start параметрінен бастап, мәннің бірлікке жеткенінше шығарады. Сіздің функция бүтін сандарды сіздің тізбекте қайтаруы керек. Мысалы, егер n - 3 болса, функция 8 қайтаруы керек. (while циклін қолданыңыз.)

5.5.7. Осы жаттығуда, біз ойлан тап ойынын ойнаудың үш стратегиясының тиімділігін салыстыру үшін Монте-Карло әдісі бойынша үлгілеу жасаймыз. Бұл стратегиялардың әрқайсысы осы тарауда біз құрған ойлан тап ойынының функциясына біріктіріледі, бірақ функция ойыншы ұтатынын немесе жеңілетінін тексерудің орнына дұрыс сан табылғанша жалғасады да, содан кейін пайдаланылған мүмкіндіктер санын қайтарады. Біз сонымен қатар жасырын санның максималды ықтимал параметрін береміз де жасырын сандардың әртүрлі диапазондары үшін берілген нәтижелерді салыстыра алатын боламыз.

Бірінші стратегия - әр кезде алдыңғы болжамға қарамай кездейсоқ болжам жасау болып табылады:

```

def guessingGame1(maxNumber):
    """Кездейсоқ болжамдарды айтып ойлан тап ойынын
    ойнау."""

    secretNumber = random.randrange(1, maxNumber + 1)
    myGuess = 0
    guesses = 0
    while (myGuess != secretNumber):
        myGuess = random.randrange(1, maxNumber + 1)
        guesses = guesses + 1

    return guesses

```

Екінші стратегия - кез келген қайталануды болдырмауға мүмкіндік беретін 1-100 аралығындағы барлық мүмкін деген болжамдарды тексеру

бірізділігі болып табылады:

```
def guessingGame2(maxNumber):
    """Әр қадам сайын ұлғаятын болжамдар жасап ойлан тап
    ойынын ойнау."""

    secretNumber = random.randrange(1, maxNumber + 1)
    myGuess = 0
    guesses = 0
    while (myGuess !=
           secretNumber): myGuess =
        myGuess + 1 guesses =
        guesses + 1

    return guesses
```

Соңғы, үшінші стратегия жасырын санды табу үшін алдыңғы жауаптардың нәтижелерін пайдаланады:

```
def guessingGame3(maxNumber):
    """Жасырын санды нақтылау арқылы ойлан тап
    ойынын ойласып ойнау."""

    secretNumber = random.randrange(1, maxNumber + 1)
    myGuess = 0
    low = 1
    high = maxNumber
    guesses = 0
    while (myGuess !=
           secretNumber): myGuess =
        (low + high) // 2 guesses =
        guesses + 1

        if myGuess < secretNumber:
            low = myGuess + 1
        elif myGuess > secretNumber:
            high = myGuess - 1

    return guesses
```

Берілген үш стратегияның болжанатын (яғни, орташаланған) тәртібін салыстыру үшін Монте-Карло әдісі бойынша үлгілеу жазыңыз. Нақты нәтижелерге жету үшін жеткілікті сынақ санын қолданыңыз. 5.1.-тараудағыдай, үлгілеуіңізді максималды жасырын сандар диапозонына, нақтырақ 5, 10, 15,..., 100 үшін іске қосып, максималды жасырын номердің әрқайсысы үшін стратегияның әрқайсысы қажет ететін болжамдардың орташа санының графигін жасаңыз. (Сіздің графигіңіздегі x осінде максималды жасырын сандар болады, ал y осінде жауаптардың орташа саны болады.) Нәтижелерді түсіндіріп беріңіз. Жалпы, стратегияның әрқайсысына 1 және n аралығындағы жасырын санды табу үшін қандай орташа сан қажет болады деп ойлайсыз?

Жоба 5.1 Сауалнама жүргізудің сиқыры

Соңғы сауалнамаға сәйкес, президентке сенімділік деңгейі $\pm 3\%$ дәлсіздікпен 45% құрайды, бұл демалыс күндері сұранылған шамамен 1,500 ересектердің жауабына негізделген.

Біз осындай бастақырыптарды жаңалықтар жолынан әрқашан көреміз. Алайда кездейсоқ таңдалған 1,500 адамның сауалнамасы бүкіл миллиондаған халықтың пікірін қалайша ұсына алады? Әлеуметтанушылар дәлсіздік көрсеткішіне қалай соншалықты сенімді бола алады? Осы жобада, біз кездейсоқ іріктелімнің көпсанды халықтың өзіндік ерекшелігін қаншалықты анықтай алатындығын зерттеп білеміз. Біз мұнда өзіндік ерешеліктері мен сенімдері бар жалпы халықтың шынайы пайызын білеміз делік, сосын шағын топтың сауалнамасы қаншалықты дұрыс болатындығын зерттейміз.

Мәселен, біз еліміздегі тұрғындардың 30% “Жануарларға адамдардың құқытарымен тең құқық берілу керек.” деген пікірді қолдайды делік. Интуитивті түрде, бұл тұрғындардың онын кездейсоқ түрде таңдап алатын болсақ, онда орташа есеппен алғанда олардың үшеуі осы пікірді ұстанып, қалған жетеуі оны қолдамайтындығын анықтаймыз. Бірақ бұл кездейсоқ таңдалған он адамның сауалнамасы бүкіл саны көп халық үшін пайыздық арақатынасты көрсете ала ма? Монте-Карло әдісі бойынша үлгілеумен салыстырғанда, сауалнама тобы нақты бір уақыт ішінде тек бір рет (немесе екі рет) қана таңдалады. Сауалнама нәтижелеріне сенімді болу үшін бізге басқа кездейсоқ таңдалған тұлғалар тобының сауалнамасы нәтижелерінен көп айырмашылық болмайды деген кепіл керек. Мысалы, сіз он адамға сауалнама жүргізіп олардың екеуі бұл пікірді қолдайтынын анықтадыңыз, сосын тағы он адамға сауалнама жүргісесіз де, олардың жетеуі пікірді қолдайтынын анықтайсыз, сосын тағы он адамға сауалнама жүргізіп, олардың оны да пікірді қолдайтынын анықтайсыз делік. Қандай қорытынды жасайсыз? Сауалнама нәтижелері атаулы шамадан едәуір ауыткитындықтан, ол сенімсіз. Ал егер біз сауалнаманы оннан астам адамға жүргізсек не болмақ? Ауытқу шамасы түзеліп, сенімділік те арта түсер ме еді?

Бұл жобада, сіз осындай сұрақтарды зерттеу үшін бағдарлама жазсыз және эмпирикалық жолмен көп санды халықтың көңіл-күйін дұрыс ұсынатын сыналатын топтың қаншалықты үлкен болу керектігін анықтайсыз.

1. Сауалнаманы үлгілеңіз

Бұл сауалнаманы жүргізу үшін әлеуметтанушы кездейсоқ таңдалған тұлғаның әрқайсысынан ол бұл пікірді қолдайтындығын сұрайды. Біз халықтың 30% қолдайтынын білеміз, сондықтан тұлғалардың 30% “иә” деп айтады деген ықтималдық бар. Бұл сауалнаманы үлгілеу үшін, біз жауап беруші тұлғаларды іріктеп, оларды 0,3 ықтималдықпен “иә” деп жауап бергендер сомасында ескере аламыз. Циклдің аяқталғанынан кейін ақырғы соманы сауалнамаға жауап бергендер санына бөліп сауалнама нәтижесін аламыз. Бұл үлгілеуді келесі функцияны жазу арқылы іске асырамыз

```
poll(percentage, pollSize)
```

ол көп санды халық ішінен таңдалған pollSize тұлғаларының сауалнамасын үлгілейді, ондағы белгіленген пайыз percentage (0-100 аралығында) “иә” деп жауап береді. Функция “иә” деп жауап берген сұралғандардың пайыздық арақатынасын (0-100 дейін) қайтаруы керек. Функция шақырылған сайын нәтиженің әртүрлі болатынын есте сақтаймыз. Функцияңызды сауалнама тобының әртүрлі көлемдері үшін тексеріп көріңіз.

2. Сауалнаманың шектік шамаларын табу

Сауалнама жүргізілетін топтың саны белгілі болған кездегі ауытқулардың атаулы шамадан қаншалықты үлкен болатындығын анықтау үшін келесі функцияны жазыңыз

```
pollExtremes(percentage, pollSize, trials)
```

ол сауалнама нәтижелерінің сынақтарынан trials тұратын тізім жасайды, бұл үшін poll(percentage, pollSize) функциясын trials рет шақырады. Функция осы тізімдегі пайыздық арақатынсының минималды және максималды шамасын қайтару керек. Мысалы, егер бес сынақ [28, 35, 31, 24, 31] пайыздық арақатынас беретін болса, онда функция минималды 24 және максималды 35 шамасын қайтарады. Егер сауалнама нәтижелерінің тізімі pollResults деп аталса, сіз осы екі мәнді мыналармен қайтара аласыз

```
return min(pollResults), max(pollResults)
```


Функцияңызды әртүрлі көлемдегі тәжірибелік топқа, сынақтардың әртүрлі санында тексеріп көріңіз.

3. Тәжірибелік топтың жеткілікті көлемі дегеніміз не?

Әрі қарай, біз сіздің алдыңғы функцияларыңызды қолданып сауалнамаға тартылатын топ көлемінің ұлғаюы сауалнама нәтижелерінің атаулы шамадан ауытқуына қалай әсер ететіндігін зерттейміз. Интуициялы түрде, сауалнамаға тартылатын адам саны неғұрлым көп болса, оның нәтижесі соғұрлым дәл болмақ. Функцияны жазыңыз

`plotResults(percentage, minPollSize, maxPollSize, step, trials)`

ол `pollExtremes(percentage, pollSize, trials)` функциясын шақыру арқылы `pollSize` мәндері үшін `minPollSize` - `maxPollSize` аралығындағы диапазоннан `step` қадамын ұлғайтып минималды және максималды пайыздық арақатынастар графигін құрайды. Тәжірибелік топ көлемінің әрқайсысы үшін сіздің `pollExtremes` функцияңызды төмендегі функция арқылы шақырыңыз

`low, high = pollExtremes(percentage, pollSize, trials)`

содан кейін `low` және `high` мәндерінің әрқайсысын графикті құру үшін сәйкес тізімге қосыңыз. Функцияңыз $(high - low) / 2$ деп анықталатын ең көп деген тәжірибелік топтың сауалнамасының дәлсіздігін қайтару керек. Тәжірибелік топтың көлемі графигіңіздің x-осінде ал пайыздық көрсеткіш y-осінде болуы керек. Осының екеуінде белгіленгенін тексеріңіз.

Тапсырма 5.1.1 *Сіз сауалнамаға көбірек жауап берушілер санын тартуға байланысты кететін еңбек шығындары мен төменгі дәлсіздік арасындағы тепе-теңдікті келтіруді қалайсыз делік, тәжірибелік топтың экономикалық жағынан тиімді көлемі қандай болмақ? Тәжірибелік топтың осындай көлемі қандай дәлсіздік береді?*

Осы сұраққа жауап іздеу үшін `plotResults` функцияңызды шақыратын `main` функциясын жазыңыз (егер әлі жазбасаңыз). Оны `plotResults(30, 10, 1000, 10, 100)` параметрлерде шақырудан бастауыңызға болады.

4. Дәлсіздік іс жүзіндегі пайыздық арақатынасқа тәуелді ме?

Осы сұраққа жауап іздеу үшін, келесі функцияны жазыңыз

`plotErrors(pollSize, minPercentage, maxPercentage, step, trials)`

ол step қадамының ұлғайып, minPercentage - maxPercentage диапазоны аралығындағы іс жүзіндегі пайыздық арақатынастың өзгеруінде pollSize тұлғадан құралған тәжірибелік топ үшін дәлсіздік графигін құрайды. Тәжірибелік топтың әрқайсысының дәлсіздігін табу үшін, жоғардағыдай pollExtremes функциясын шақырыңыз да, $(high - low) / 2$ есептеңіз. Графигіңізде пайыздық шама x-осінде ал дәлсіздік шама y-осінде көрсетілуі керек. Осының екеуін де белгілегеніңізді тексеріп алыңыз.

Тапсырма 5.1.2 *Алдыңғы бөлікке берген жауабыңыз халықтың іс жүзіндегі пайыздылығы өте төмен немесе өте жоғары болғанда өзгере ме?*

Бұл сұраққа жауап іздестіруді функцияны plotErrors(1500, 10, 80, 1, 100) көмегімен шақырудан бастай аласыз.

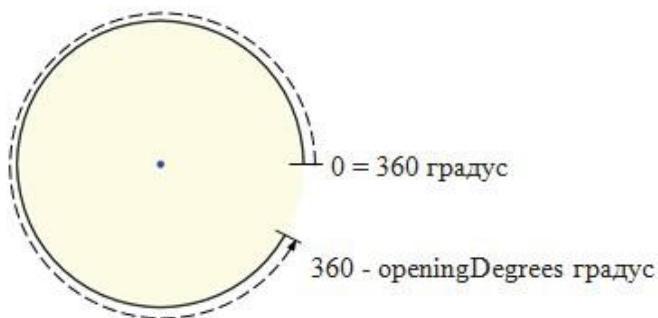
Жоба 5.2 Қашу!

Кейбір сценарийлерде, кездейсоқ кезудегі “бөлшектің” қозғалысы шектеумен қоршалады. Ал егер онда бір кішкентай ашық саңылау болып, ол жерден бөлшек шығып қашып кете алатын болса ше? Осындай саңылауды кездейсоқ кезіктіріп, қашып кету үшін бөлшекке орташа есеппен қанша қадам жасау керек? *Сіңірілу мәселесі* деп аталатын бұл үлгі жыртқыштың аймағына келіп қалған саңырауқұлақ терушіні, карантинді аймаққа кіретін кіреберіс тапқан жануарды, жасушалы мембранаға ұсақ тесіктер арқылы өтетін жол тапқан молекулаларды, немесе моншада ашық есік арқылы шығатын ауа молекулаларын келтіруге болады.

1. *Кездейсоқ кезудегі сіңірілуді үлгілеңіз*
Функцияны жазыңыз

escape(openingDegrees, tortoise, draw)

ол openingDegrees градустық ашық саңылауы бар радиусы 1 шеңбердегі кездейсоқ кезудегі сіңірілу мәселесін үлгілейді. Шеңбердегі саңылау төменде келтірілген суреттегідей 360 – openingDegrees және 360 градус аралығында болсын.



Бөлшек 5.3.1 жаттығуында сипатталғандай нормалы үлестірімге сәйкес кездейсоқ қозғалуы керек. Нормалы үлестірімнен сатндартты ауытқу шамасы бөлшектің кішкентай саңылауды табуы үшін едәуір кішкентай болуы керек. $\pi/128$ мәні [7] жұмыста берілген. Бізге бөлшектің жүріп өткен қадамдарының саны керек болғандықтан (өткендегідей ол жүріп өткен қашықтығы емес), циклдің әр итерациясында қадам саны үдетіліп отырады. Егер бөлшек қабырғаға соқтығысса, ол бастапқы қалпына қайта “секіруі” керек. Бөлшектің шеңбер ішінде қозғалғандығынан, біз оның қабырғаға соқтығысқанын оның координаталар басынан алшақтығын шеңбер радиусымен салыстырып айта аламыз. Егер бөлшек қабырғаға дейінгі қашықтықтан шығып кетсе, алайда саңылау бұрышының маңында болатын болса, цикл бөлшектің қашып кетуі туралы айтып аяқталуы керек.

Координаталардың басына қатысты бөлшектің ағымдағы бұрышын табу үшін тригонометрия ілімін білу керек. Бөлшектің x және y координаталарын білгендіктен, бұрышты y/x : $\tan^{-1}(y/x)$ арктангенсін есептеу арқылы табуға болады. Дегенмен, бұл $x = 0$ жағдайында проблема тудырады, сондықтан бізге бұл мүмкіндікті ескеріп, x мәнін ойдан шығаруымыз керек. Сондай-ақ, Python `math.atan` арктангенс ($\tan^{-1}$) функциясы, ылғи $-\pi/2$ және $\pi/2$ радиандары арасындағы (-90 және 90 градус аралығындағы) бұрышты қайтарады, сондықтан 0 және 360 градус аралығындағы мәнді қабылдау үшін нәтижені түзету керек. Оны сіз үшін келесі функция жүзеге асырады.

```

def angle(x, y):
    if x == 0: # x = 0.001
        0-ге бөлінуін
        болдырмайды
    angle = math.degrees(math.atan(y /
x)) if angle < 0:
        if y < 0:
            angle = angle + 360 # квадрант
IV else:
            angle = angle + 180 # квадрант
II elif y < 0:
            angle = angle + 180 # квадрант
III return angle

```

Төменде сіз циклі мен сурет салу коды жазулы escape функциясының “қаңқасын” көресіз . Төменде setupWalls функциясының көмегімен салынған шеңбер бөлігі берілген. Назар аударыңыз, мұнда функция итерацияларды басқарған кезде while циклін және логикалық escaped деп аталған *флаг-айнымалыны қолданады*. escaped мәні False деп инициалданған, ал сіздің алгоритміңіз бөлшектің сіңірілгенінде True деп белгілеуі керек. Қалған кодтың *едәуір бөлігін*, алайда түгелдей емес, while цикліне енгізу керек.

```

def escape(openingDegrees, tortoise, draw):
    x = y = 0 # инициализация (x, y) = (0, 0)
    radius = 1 # stepLength = math.pi /
128 бірлік радиус шеңберінде
    жылжу # әрбір қадам үшін стандартты ауытқу

    if draw:
        scale = 300 # setupWalls(tortoise,
openingDegrees, scale, radius) суретін
масштабтайды

    escaped = False # бөлшек әлі сіңірілмеді
    me? while not escaped:

```

```

# мұнда кездейсоқ қозғалыс бар, қабырға
мен саңылауды анықтай if draw:
    tortoise.goto(x * scale, y * scale) #бөлшектің
                                         |жылжуы

if draw:
    screen = tortoise.getscreen() # screen.update()
    өзгеруінен кейін экранның жаңаруы # соңғы
    итерацияны көрсету үшін

def setupWalls(tortoise, openingDegrees, scale,
radius): screen = tortoise.getscreen()
screen.mode('logo') # шығысқа қарай 0
                    градус
screen.tracer(5) # сурет салу
                 жылдамдығының артуы

tortoise.up() #
tortoise.width(0.015 * scale) көмегімен шегара салу
#
tortoise.goto(radius
* scale, 0) боялған фон
tortoise.down()
tortoise.pencolor('lightyellow')
tortoise.fillcolor('lightyellow')
tortoise.begin_fill()
tortoise.circle(radius * scale)
tortoise.end_fill()
tortoise.pencolor('black')
tortoise.circle(radius * scale, 360 -
openingDegrees) tortoise.up()
tortoise.home()

```

```

tortoise.pencolor('blue') # бөлшек көк
шеңбермен берілген tortoise.fillcolor('blue')
tortoise.shape('circle')
tortoise.shapesize(0.75, 0.75)

tortoise.width(1) #
tortoise.pencolor('green') қозғалуын
орнату
tortoise.speed(0)
tortoise.down() # ізді жасыру үшін оны шолуға
айналдырыңыз

```

2. Монте-Карло әдісі бойынша үлгілеуді жазыңыз
Функцияны жазыңыз

```
escapeMonteCarlo(openingDegrees, trials)
```

ол белгіленген сынақ саны үшін openingDegrees градусындағы саңылау арқылы сіңірілуге қажетті қадамдардың орташа санын қайтарады. Бұл 5.1-тараудағы rwMonteCarlo функциясына қатты ұқсайды.

3. Эмпирикалық әдіспен функция алу
Функцияны жазыңыз

```
plotEscapeSteps(minOpening, maxOpening, openingStep, trials)
```

ол openingStep кадамын үдетіп minOpening -maxOpening градусы диапазоны аралығында жатқан саңылау арқылы сіңірілуге қажетті берілген сандағы сынақтар үшін қадамдардың орташа санының графигін құрайды. (Графигіңіздегі x-осінің мәндері - саңылау ені, ал y-осіндегі мәндер сіңірілуге қажетті қадамдардың орташа саны.) Бұл 5.1-тарауындағы plotDistances функциясына ұқсас.

10 градустық кадаммен 10-180 градусы аралығындағы саңылаулар арқылы сіңірілуге керекті қадамдардың орташа санының өзгеру графигін құраңыз, мұнда тегіс айнамлы алу үшін кем дегенде 1,000 сынақ жүргізіңіз. Мұндай санды сынақпен үлгілеуді жүргізу бірнеше минутты алатындықтан, аз санды сынақпен бастаңыз да, Сіздің үлгілеуіңіз дұрыс

жұмыс істейтіндігін тексеріп алыңыз.

Питтсбург Университетінде Кэри Кэджинэлп өзінің дипломдық жұмысында [7] осы нәтижелерді суреттейтін функцияны математикалық жолмен шығарды. Ол, әсіресе, бөлшектің α градустық саңылаудан өту үшін керекті уақыттың келесіні құрайтынын дәлелдеді

$$T(\alpha) = \frac{1}{2} - 2 \ln \left(\sin \frac{\alpha}{4} \right).$$

Бұл функцияны сіздің эмпирикалық нәтижелеріңіз тұрған графикте құрыңыз. Сіз $T(\alpha)$ айнымалысының үлгілеу нәтижелерінен едәуір төменгі деңгейде орналасқанын байқайсыз, ол біз қолданатын қадамға сәйкес осылай болуы керек (яғни, escape функциясында stepLength мәні). Бұл қадам көлеміне сәйкестіру үшін, escapeMonteCarlo функциясымен қайтарылатын әрбір мәнді графикке түсірместен бұрын қадам көлемінің шаршысына көбейтіңіз ($(\pi/128)$). Осылай жасап болған соң, Сіздің Монте-Карло бойынша үлгілеуіңіздің нәтижелері дәл Кэджинэлп қолданған уақыт бірліктерінде берілетін болады, және олар математикалық жолмен алынған нәтижемен тығыз орналасуы керек.

6 ТАРАУ

Мәтін, құжаттар, және ДНҚ

6.1 СӨЗДЕРДЕН ТҰРАТЫН ЕСЕПТЕР

Жоғарыда көргеніміздей, біз мәтінді жол деп аталатын символдар тізбегі түрінде сақтай аламыз. Жол константасы (сондай ақ жол литералы), мысалы 'Дәл таптыңыз!', бір (') немесе екі (") тырнақшаға алынады. Сонымен қатар, Біз жол мәндерінің параметрлер ретінде жберілетін айнымалылардың атауларына меншіктеліне алатынын анықтадық, және олар келесі қарапайым функцияларда көрсетілгендей сандар тәрізді бір-бірімен салыстырыла алады.

```
def roar(n):
    return 'r' + ('o' * n) + ('a' * n) + 'r!'

def speak(animal):
    if animal == 'cat':
        word = 'meow.'
    elif animal == 'dog':
        word = 'woof.'
    else:
        word = roar(10)

    print('The', animal, 'says', word)

speak('monkey')
```

Ойлануға берілетін сұрақ 6.1 *Жоғарыда келтірілген код не басады? Неліктен екенін түсінгеніңізді тексеріңіз.*

Turtle секілді жол дегеніміз - нысан, str деп аталатын сынып нұсқасы болып табылады. Осылайша, жол дегеніміз - мәліметтердің абстрактілі түрінің бір мысалы. Мәліметтердің абстрактілі түрі оның мәліметі сақталғаны жөніндегі мағлұматтарды жасырады да, программистке әдістер деп аталатын функциялар жиынтығы арқылы онымен

әрекеттесуге мүмкіндік береді. 6.3. Тарауында тағы талқылайтындықтан, жолдар іс жүзінде екілік тізбек түрінде сақталады, алайда біз олармен табиғи түрде әрекеттесе аламыз.

Жолдар үшін бірнеше әдіс қолайлы.¹ Мысалы, `upper()` әдісі жаңа жолды қайтарады, онда алдыңғы жолдағы бүкіл кіші әріптер бас әріптерге ауыстырылған. Мысалы,

```
>>> pirate = 'Longbeard'
>>> newName = pirate.upper()
>>> newName
'LONGBEARD'
```

Әдістерді қолданған кезде, нысанның атауы, берілген жағдайда жол нысанының атауы - `pirate`, әдістің атауының алдында жүретінін есте сақтаймыз.

Үлкен жолдардың ішіндегі шағын жолдар бөлшегі тармақша деп аталады. Біз жолда тармақшалардың бар екендігін тексеру үшін `in` және `not in` операторларын қолдана аламыз. Мысалы, келесі `if` нұсқауларының әрқайсысындағы шарттар дұрыс.

```
>>> if 'bear' in pirate:
...     print('There is a bear in Longbeard.')
...
There is a bear in Longbeard.
>>> if 'beer' not in pirate:
...     print('There is no beer in Longbeard.')
...
There is no beer in Longbeard.
```

Тармақшалармен байланысты, ерекше пайдалы әдістің екеуі - `replace` және `count` болып табылады. `replace` әдісі бір тармақшаның келесі тармақшамен ауыстырылған жағдайларының барлығын алмастырып жаңа бір жол қайтарады. Мысалы,

¹Тізімді Б.6.Қосымшасынан көруге болады.

```
>>> newName = pirate.replace('Long', 'Short')
>>> newName
'Shortbeard'
>>> quote = 'Yo-ho-ho, and a bottle of rum!'
>>> quote2 = quote.replace(' ', '')
>>> quote2
'Yo-ho-ho,andabottleofrum!'
```

Екінші мысалда `replace` әдісі жолдағы бос орындарды жою үшін қолданылған. `replace` әдісіндегі екінші параметр, тырнақшадағы бос орын, бос жол деп аталады, және ноль символдан тұратын рұқсат етілетін жол болып табылады.

Ойлануға берілетін сұрақ 6.2 *Мәтіндегі барлық `brb` орнына `be right back` деп ауыстыратын функция жазыңыз.*

```
>>> txt = 'Hold on brb'
>>> txt.replace('brb', 'be right back')
'Hold on be right back'
```

`count` әдісі тармақшалардың жолға кіру санын қайтарады. Мысалы,

```
>>> pirate.count('ear')
1
>>> pirate.count('eye')
0
```

Біз мәтіндегі сөз санын “аралық” символдарды санау арқылы есептеп білеміз, себебі сөздердің барлығы дерлік қандай да бір бос орын аралығымен жүруі тиіс. Аралық символдар дегеніміз - бос орындар, табуляциялар, және жолды аудару символдары (мәтін жолының соңын көрсететін жасырын символдар) болып табылады. Табуляциялар мен жолды аудару символдары - басылмайтын басқарушы *символдар* мысалы болып табылады. Табуляция `\t` екі символмен белгіленеді, ал жолды аудару `\n` екі символмен белгіленеді. `wordCount1` функциясы төменде `count` әдісін `text` деп аталған жолдарда аралық символдар санын қайтару

үшін қолданады.

```
def wordCount1(text):
    """Бос орындар, табуляциялар мен жолдарды
    аударуды санау арқылы есептелінетін бір
    жолдағы әріптер саны.

    Параметр:
        text: Жол нысаны

    Қайтарылатын мән: Мәтіндегі бос орындар,
    табуляциялар мен жолдарды аудару саны """
    return text.count(' ') + text.count('\t') +
    text.count('\n')

def main():
    shortText = 'This is not long. But it will do. \n' + \
        'All we need is a few
    sentences.'
    wc =
    wordCount1(shortText)
    print(wc)

main()
```

Жоғарыдағы shortText жолында біз бос аралықтың символын () айқын етіп көрсеттік және оны екі бөлікке бөлдік, себебі ол бір жолға сыймады.

Ойлануға берілетін сұрақ 6.3 *Python қабығында*

```
print('Two tabs\tthere.      Two newlines\n\nhere.')
```

деп теріп, \t және \n символдарының әрекетін байқап көріңіз.

Ойлануға берілетін сұрақ 6.4 *Жоғарыда берілген мысалда wordCount1 функциясы қандай жауап қайтарады? Бұл жауап дұрыс па? Егер дұрыс болмаса, неге олай?*

wordCount1 функциясы 16 қайтарады, алайда shortText-те іс жүзінде тек 15 сөз ғана. Біз дұрыс жауап алмадық, себебі бірінші сөйлемнен кейін

екі бос орын тұр, екінші сөйлемнен кейін - бос орын мен жолды аудару символы тұр, ал соңғы сөйлемнен кейін ештеңе жоқ. Оны қалай түзетуге болады? Егер бізге мәтінде жолды аудару символы болмағандығын және ол сөздер арасында бір бос жол қойылып, сөйлемдер арасына екі бос орын қойылып, ал соңында ешқандай бос орын қойылмай керемет өңделгенін білгенде, онда біз жоғарыда берілген есепті мәтінге кірген екі бос орынның санын алып, соңғы сөзді есептеу үшін бірлік қосу арқылы түзете алатын едік:

```
def wordCount2(text):  
    """ (құжаттау жолдары түсірілген) """  
  
    return text.count(' ') - text.count('\n') + 1
```

Алайда мәтіннің көбі керемет өңделмегендіктен, бізге анағұрлым күрделірек әдіс керек.

Ойлануға берілетін сұрақ 6.5 *Егер сізге үлкен мәтін кезгішін, сіз бір ретте тек бір ақ символды тексере алатын болғаныңызда, сөздер санын анықтау үшін қандай алгоритмді қолданаар едіңіз?*

Бастама ретінде, count әдісі қалай жұмыс істеу керектігі туралы ойланайық. Жолдағы кейбір символдың кіру санын есептеу үшін, count әдісі жолдағы әрбір символды тексеріп, және әрбір сәйкестілік үшін сумматордың айнымалы шамасын ұлғайтуы керек.

Бұл әрекетті қайталау үшін, біз for циклінің көмегімен жолды сұрыптай аламыз, бұл бүтін сандар диапазонын ranges сұрыптауға ұқсайды. Мысалы, келесі for циклі shortText жолындағы символдарды сұрыптайды. Бұл циклді жоғарда берілген main функциясына енгізіп, оның не істейтінін көріңіз.

```
for character in shortText: print(character)
```

Бұл циклде жолдағы әрбір символ рет-ретімен character индекстік айнымалысына меншіктеледі. Оны суреттеу үшін, біз циклдің денесінде экранға индекстік айнымалыға меншіктелген символдардың жай шығарылуын көрсеттік. Жоғарыда main функциясында меншіктелген shortText белгіленген мәнінде цикл экранға келесіні шығарады

```
T
h
i
:   (ортасы түсірілген)
e
s
```

Аралық символдарды санау үшін, біз сол циклді пайдалана аламыз, бірақ циклдің денесіндегі есептеуішті біз character мәні аралық символға тең болған сайын ұлғайтуымыз керек болады:

```
def wordCount3(text):
    """ (құжаттау жолдары түсірілген) """

    count = 0
    for character in text:
        if character == ' ' or character == '\t' or character == '\n':
            count = count + 1
    return count
```

Ойлануға берілетін сұрақ 6.6 *Егер біз or операторларын and операторларына ауыстырсақ не болады?*

Сонымен бірге, if шартын in операторы арқылы оңайлатуға болады:
if character in '\t\n':

Ойлануға берілетін сұрақ 6.7 *wordCount3 функциясы біз жоғарыда анықтаған shortText параметрі арқылы шақырсақ қандай жауап беретін еді?*

Біз бұл функцияны shortText көмегімен шақырсақ, біз wordCount1 (16) жағдайындағыдай жауап аламыз, өйткені біз жай ғана функцияның әрекетін қайталадық.

Ойлануға берілетін сұрақ 6.8 *Енді бізде сөздерді санайтын негіздік функция болған кезде, біз оны қосымша бос орындары бар мәтінмен жұмыс істеу үшін қалай нақтылай аламыз?*

wordCount3 функциясын жақсарту үшін, бізге тек қана бірінші аралық

символды немесе осындай символдар тізбегінің әрқайсысындағы жолды аудару символын санау керек.

Ойлануға берілетін сұрақ 6.9 *Егер character мәні - бос орын, табуляция немесе жолды аудару болса, біз оның осындай символдар тізбегінде бірінші болады деп қалай айта аламыз? (Сыбырсөз: егер ол бірінші болса, онда алдыңғы символ қандай болмау керек?)*

Егер character мәні аралық символ немесе жолды аудару болса, егер бірінші символ осы символдардың бірі болмаса, ол тізбекте бірінші екенін біз білеміз. Сондықтан бізге алдыңғы символды бақылап, оның if нұсқауындағы мәнін текесру керек. Егер біз оның мәнін сақтау үшін prevCharacter атты айнымалыны қолдансақ, онда бізге if нұсқауын келесідей өзгерту керек

```
if character in ' \t\n' and prevCharacter not in ' \t\n':  
    count = count + 1
```

Ойлануға берілетін сұрақ 6.10 *prevCharacter-ге мәнді қай жерде меншіктеуіміз керек?*

Ағымдағы character мәнін prevCharacter-ге әрбір итерацияның соңында меншіктеуіміз керек, себебі character-дің ағымдағы осы мәні келесі итерация бойы алдыңғы символ болады. Осы бөліктерді қосқан соң, түзетілген wordCount4 келесідей болады:

```
def wordCount4(text):  
    """ (құжаттау жолдары түсірілген) """  
  
    count = 0  
    prevCharacter = ''  
    for character in  
text:  
        if character in ' \t\n' and prevCharacter  
            not in ' \t\n': count = count + 1  
        prevCharacter =  
character return count
```

Жоғарыда айтылғанда есептегенде, енді қалғаны - prevCharacter -ге циклдің алдында if шартының бірінші итерацияда мағынасы болтындай

бастапқы мән беру.

Ойлануға берілетін сұрақ 6.11 *Егер prevCharacter-ді циклдің алдында инициалдамасақ не болады? Неліктен біз prevCharacter-ді бос орынмен инициалдадық? Бастапқы мән маңызды ма?*

Text-тегі бірінші символдың бола алатын екі мүмкіндікті қарастырып көрейік: Не аралық символ, не ол емес. Егер text-тегі бірінші символ аралық символ немесе жолды аудару символы болмаса (әдеттегідей), онда if шартының бірінші бөлігі (characterin'\t\n') жалған болады. Сәйкесінше, prevCharacter -дің бастапқы мәні маңызсыз болады. Басқа жағынан, егер character-ге меншіктелген бірінші мән - аралық символ немесе жолды аудару символы болса, онда if шартының бірінші бөлігі ақиқат болады. Алайда біз бұл символдың сөздің соңы ретінде есептелмейтіндігін тексеріп алу керек. prevCharacter мәнін басынан бастап бос орынмен белгілеу арқылы біз if шартының (prevCharacter not in '\t\n') екінші бөлігі бастапқыда жалған екендігіне кепіл береміз де, count мәнінің үдеуін болдырмаймыз.

Ойлануға берілетін сұрақ 6.12 *prevCharacter инициалдау үшін басқа мәндер жарамды бола ма?*

Ақырында, біз мәтін аралық символмен аяқталмайтын жағдайды шешіп алуымыз керек. Бұл жағдайда, соңғы сөз саналмайтындықтан, біз count мәнін бірге арттырамыз.

Ойлануға берілетін сұрақ 6.13 *text-тегі соңғы символ аралық немесе жолды аудару символы деп қалай айта аламыз?*

Цикл аяқталған соң prevCharacter-ге text-тегі соңғы символдың мәні меншіктеледі, біз оның мәнін циклден кейін тексере аламыз. Егер ол аралық символ болса, онда соңғы сөз саналып қойғанын білдіреді; керісінше жағдайда, біз count мәнін арттыруымыз керек. Ақырында функция мынадай болады:

```
def wordCount5(text):
    """Жолдағы сөздер санын есептейді.

    Параметр:
        text: Жол нысаны

    Қайтарылатын мән: Мәтіндегі
    сөздер саны"""
    count = 0
    prevCharacter = ' '
    for character in text:
        if character in ' \t\n' and prevCharacter not in ' \t\n':
            count = count + 1
        prevCharacter = character
    if prevCharacter not in ' \t\n':
        count = count + 1
    return count
```

Біздің мысалдарда қысқа жолдар пайдаланылса да, функциямыз біз қалаған кез келген жол көлемінде жұмыс істейтін болады. Келесі тарауда біз бүкіл мәтіндік файлды немесе ғаламтор парағын бір жолда есептеуді үйренеміз, және біздің сөз санайтын функциямызды өзгертпей файлдағы сөздерді санау үшін қолданамыз. Келесі тарауларда біз толық кітаптар мен ұзақ ДНҚ тізбектерінен тұратын мәтіндік файлдардың күрделі талдауларын орындайтын ұқсас алгоритмдер жасап шығарамыз.

Жаттығулар

6.1.1. Функцияны жазыңыз

`twice(text)`

ол арасында аралық қойылған екі рет қайталанған `text` жолын қайтарады. Мысалы,

`twice('bah')` - `'bah bah'` жолын қайтару керек.

6.1.2. Функцияны жазыңыз

`repeat(text, n)`

ол `text` жолының `n` көбейтілген жолын қайтарады. Мысалы, `repeat('AB',`

3) `'ABABAB'` жолын қайтару керек.

6.1.3. Функцияны жазыңыз

vowels(word)

ол count әдісін қолданып word жолындағы дауысты дыбыс санын қайтарады. (Назар аударыңыз, word кіші және бас әріптерден де тұра алады.)

6.1.4. Функцияны жазыңыз

nospaces(sentence)

ол replace жолының әдісіні қолданып өзгертілген sentence жолын қайтарады, онда барлық бос жолдар төменгі сызық () символымен ауыстырлған.

6.1.5. Функцияны жазыңыз

txtHelp(txt)

ол txt жолының кеңейтілген нұсқасын қайтарады, онда “brb” және “lol” секілді мәтіндік аббревиатуралар болуы мүмкін. Функцияңыз кем дегенде төрт түрлі мәтіндік аббревиатураларды аша білуі керек. Мысалы, txtHelp('imo u r lolbrb')'in my opinion you are laugh out loud be right back' жолын қайтаруы керек.

6.1.6. Функцияны жазыңыз

letters(text)

ол экранға жолда бір-бірден text жолының символдарын шығарады.

Мысалы

letters('abc') экранға мыналарды шығарады

a

b

c

6.1.7. Функцияны жазыңыз

count(text, letter)

ол letter атты бір символды жолдың text жолына кіру санын count әдісін қолданбай қайтарады.

(Оның орнына for циклін қолданыңыз.)

6.1.8. Функцияны жазыңыз

vowels(word)

Ол 6.1.3 жаттығудағыдай нәтижені count әдісін қолданбай қайтарады.

(Оның орнына for циклін қолданыңыз.)

6.1.9. Функцияны жазыңыз

sentences(text)

ол text жолындағы сөйлем санын қайтарады. Сөйлем нүктемен, сұрау белгісімен немесе леп белгісімен аяқталуы мүмкін. Ол сондай-ақ біренеше тыныс белгілерімен аяқталуы мүмкін, мысалы '!!!' немесе '?!'.

6.1.10. Функцияны жазыңыз
nospaces(sentence)

ол 6.1.4 жаттығудағыдай нәтижені replace әдісін қолданбай қайтарады. (Сыбырсөз: жаңа жол жасау үшін for циклін және тіркесу операторын (+) циклдің ішінде қолданыңыз. Біз осыған ұқсас көп мысалдарды 6.3 тарауда көреміз.)

6.1.11. Желі бойынша биттерді жіберу кезінде, олар кейде кедергілермен және қателермен бұзылады. Жіберілістің соңында бірнеше артық бит қосу арқылы бұл қателіктерді табуға көбектеседі. Қателерді табатын ең қарапайым алгоритмді жұптықты тексеру дейміз. Биттердің тізбегі егер ол жұп бірлік санына тұрса, жұптық паритетке ие болады, ал керісінше жағдайда тақ паритетті болады. Жұптық паритет сұлбасында, жіберуші соңғы бит жолы жұпты бірліктер санынан тұратындай етіп бит жолының соңына бір бит қосады. Мысалы, егер біз 1101011 мәліметін жібергіміз келсе, оның орнына іс жүзінде біз 11010111 жібереміз, сондықтан биттердің тізбегі жұп сандар бірлігінен тұратын болады. Егер біз оның орнына 1101001 жібергіміз келсе, онда іс жүзінде 11010010 жібереміз. Содан кейін алушы қабылданған бит жолының жұптық паритеті бар-жоғын тексереді; егер жоқ болса, онда қателік шығып, қайта жіберуді сұрайды.

(a) Паритет тек қана өте қарапайым қателерді анықтай алады. Жұптық паритет сұлбасы арқылы табылмайтын қателіктің мысалын келтіріңіз.

(b) Жоғарыда берген мысалыңыздағы қателікті анықтайтын шешім ұсыныңыз.

(c) Келесі екі проблемада, біз биттер жолдар тәрізді жіберілді деп аламыз (бірақ олай емес; әйтпесе бұл өте тиімсіз болар еді). Функцияны жазыңыз

evenParity(bits)

ол count әдісін қолданып егер биттер жолында жұптық паритет болса True қайтарады, ал керісінше болса False қайтарады. Мысалы, evenParity('110101') - True қайтару керек және evenParity('110001') False –қайтару керек.

(d) Енді evenParity функциясын count әдісін қолданбай жазыңыз.

(e) Функцияны жазыңыз

makeEvenParity(bits)

ол bits пен қайтарылатын жол жұптық паритетті болатындай етіп тандалған бір қосымша тіркескен биттен тұратын жолды қайтарады. Сіздің функцияңыз evenParity функцияңызды шақыруы керек. Мы-

салы, `makeEvenParity('110101')` - '1101010' қайтару керек және `makeEvenParity('110001')` - '1100011' қайтару керек.

6.2 МӘТІНДІК ҚҰЖАТТАР

Шынайы қолданғырларда, біз мәтінді қатты дискте немесе флэш-жинақтауышында сақталған файлдардан оқимыз. Компьютердегі барлық нәрселер тәрізді файлдар дегеніміз – биттердің тізбегі болып табылады. Алайда біз файлдармен абстракциялар, мәтін, кестелер немесе суреттер секілді ақпараттан тұратын электронды құжаттар ретінде әрекеттесеміз. Бұл абстракциялар файлдық жүйе деп аталатын операциялық жүйенің бөлігінің жанамасы болып табылады. Файлдық жүйе файлдарды иерархиялық мұқабалар құрылымына ұйымдастырады, мысалы 6.1.-суретте көрсетілген Mac OS X файлдық жүйесінің оңайтылған түрі. Осы тәрізді иерархиялық құрылымдарды ағаштар деп атаймыз. Ағаштың басы түбір деп, ал бұтақтары туындылар деп аталады. Ағаштың туындылары жоқ элементтері жапырақтар делінеді. Файлдық жүйеде мұқабаның туындылары болуы мүмкін, олар файлдар мен олардың ішінде болатын басқа мұқабалардан тұрады, ал файлдар ылғи жапырақтар болып табылады.

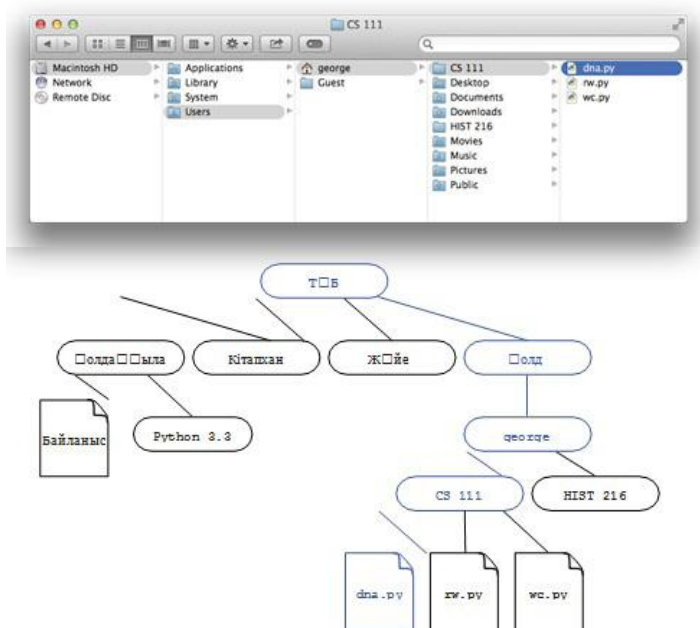
Түбір мұқабаның төменгі жағында бұл сұлбада қолданушылар деп аталған мұқаба берілген, онда әрбір қолданушы компьютердің үй мұқабасы болады, оған қолданушының аты жазылады, мысалы `george`. Жоғарыда берілген суретте бұл үй мұқабасы `CS 111` және `HIST 216` деп аталған екі тіркемеленген бумадан тұрады. Біз файлдың орналасқан жерін жол түрінде беріп, оған сәйкес жүру арқылы түбір мұқабадан файлға кіруге болады. Мысалы, `dna.py` мұқабасына апаратын жол - `/Users/george/CS 111/dna.py`. Басында тұрған слэш символы түбір мұқаба болып табылады, ал қалғаны жай ғана деңгейлерді бөліп тұрады. Бірінші слэш жоқ кез келген жол операциялық жүйеде орнатылған ағымдағы жұмыс каталогына байланысты жол деп аталады. Мысалы, егер ағымдағы жұмыс каталогы `/Users/george` болса, онда `dna.py CS 111/ dna.py` арақатысты жолымен анықталуы мүмкін.

Мәтіндік файлдардан оқу

Python-да файл файлдық нысан деп аталатын абстракция болып табылады. Біз файлды файлдық нысанмен `open` функциясын қолдану арқылы байланыстырамыз, ол файл атауын немесе жолды аргумент ретінде алады да, файл нысанын қайтарады. Мысалы, келесі нұсқау `inputFile` атты файлдық нысанды ағымдағы жұмыс каталогындағы `mobydick.txt` атты

файлдық нысанмен байланыстырады.²
`inputFile = open('mobydick.txt', 'r')`

Екінші аргумент - `open`, файлмен жұмыс жасау кезінде қолданылатын режим; `'r'` (`read`) дегеніміз - біз файлдан оқығымыз келгендігін білдіреді. Шарт бойынша, әрбір файл мәтіннен тұрады деп ұйғарылады.



Сурет 6.1 Mac OS X-тегі Finder терезесі және бөлшектік сәйкестіліктегі ағаш көрінісі. Сопақша - папка, ал тік төртбұрыш - файл болып табылады.

Файлдық нысанның `read` режимі файлдың құрамындағы мазмұнды түгелімен жолға енгізеді. Мысалы, келесі нұсқау бүкіл мәтінді файлдан `text` атты мәнге меншіктелген жолға құяды.

```
text = inputFile.read()
```

²Бұл файлды кітап сайтынан немесе Гутенберг жобасынан алуға болады <http://www.gutenberg.org/files/2701/2701.txt>

Файлмен жұмысты аяқтаған кезде оны жабу өте маңызды. Файлды жабу арқылы біз операциялық жүйені файлды пайдаланып болғанымыз жайлы хабарлаймыз, бұл бағдарлама файлға берген жады босағандығын кепілдейді. Файлды жабу үшін біз жай ғана файлдық нысанның `close` әдісін қолданамыз:

```
inputFile.close()
```

Осының барлығын біріктіретін мысалды қарастырайық. `WordCount5` атты функция белгіленген файлды оқып, файлдағы сөздер санын қайтарады.

```
def wcFile(fileName):  
    """Аты берілген файлдағы сөз санын қайтарады.
```

 Параметр:

`fileName`: Мәтіндік файлдың аты

 Қайтарылатын мән: Файлдағы сөз
 саны"""

```
    textFile = open(fileName, 'r', encoding = 'utf-  
8') text = textFile.read()  
    textFile.close()
```

```
    return wordCount5(text)
```

`Open` функциясына арналған қосымша `encoding` параметрі файлда биттердің қалай интерпретациялану керектігін көрсетеді (біз UTF-8 не екендігін 6.3 тарауда талқылаймыз).

Ойлануға берілетін сұрақ 6.14 *mobydick.txt* файлында қанша сөз бар?

Енді әрбір жолдың сол жағынан, жолдары нөмірленіп өңделген мәтіндік файлды экранға шығару керек делік. “Жол”— символдардың тізбегі ретінде анықталады да, жолды аудару (`'\n'`) символымен бітеді. Бізге жол нөмірін әрбір жолдың сол жағына шығару керек. Тапсырманы, файлды жолма-жол деп оқи алғанда, оңайлатуға болатын еді. Бақытына

орай, осылай істей аламыз. Бүтін сандар диапазонын немесе жол символдарын сұрыптайтын әдіс арқылы файлдағы жолдарды да сұрыптауға болады. Біз файлдық нысанды for циклін тізбек түрінде қолданғанда, индекстік айнымалыныға әрбір итерацияға бір мәтіндік файлдың әр жолы сәйкестіріледі. Мысалы, келесі код экранға textFile атты файлдық нысанның әрбір жолын шығарады.

```
for line in textFile: print(line)
```

Бұл циклдың әр итерациясында line индекстік айнымалысына келесі жол меншіктеледі де, экранға цикл түрінде шығарылады. Біз бұл идеяны экранға жол-жолдан шығару тапсырмасына оңай қолдана аламыз:

```
def lineNumbers(fileName):  
    """Әрбір жолға жол нөміріне сәйкес  
    файл мазмұнын экранға шығару.
```

Параметр:

fileName: Мәтіндік файлдың аты

Қайтарылатын

мәндер: None"""

```
textFile = open(fileName, 'r', encoding = 'utf-8')  
lineCount = 1  
for line in textFile:  
    print('{0:<5} {1}'.format(lineCount,  
    line[:-1])) lineCount = lineCount + 1  
textFile.close()
```

LineNumbers функциясы сумматор мен for циклдерін біріктіреді, ол мәтіндік файлды жол-жолдан оқиды. Файлды ашқан соң lineCount сумматорының айнымалысы бірлікке инициалданады. Циклдың ішінде ағымдағы мәні lineCount жолынан бұрын тұратын әрбір жол экранға пішіндеу жолын қолдану арқылы шығарылады. Циклдың соңында сумматор мәні ұлғаяды да цикл қайталанады.

Ойлануға берілетін сұрақ 6.15 *Неліктен функция әр итерацияда экранға line орнына line[:-1] деп шығарады? Егер line[:-1] орнына line қойса не болады?*

Ойлануға берілетін сұрақ 6.16 *Егер lineCount циклда print шақырудан бұрын ұлғайса нәтиже қалай өзгереді?*

Ойлануға берілетін сұрақ 6.17 *mobydick.txt* файлында неше жол бар?

Мәтіндік файлдарға жазу

Сондай ақ жаңа файлдарды жасауға немесе бар файлдарға жазбалар енгізуге болады. Мысалы, жолдары нөмірленіп өңделген файлды экранға шығарудың орнына, жол нөмірі бар жаңа файл нұсқасын жасауға болатын еді. Жаңа файлға мәтін жазу үшін, біз алдымен оны 'w' ("write" - жазу) режимі арқылы ашу керекпіз:

```
newTextFile = open('newfile.txt', 'w')
```

Осындай әдіспен файлды ашу арқылы newfile.txt атты жаңа файл жасалады, егер ондай файл жоқ болса, немесе керісінше жағдайда, ондай файл бар болса, файлды қайта жазады. (Сондықтан абай болыңыз!) Бар файлдың соңына қосу үшін, оның орнына 'a' ("append"-қосу) режимін қолданыңыз. Файл ашылған соң, біз оған write әдісі арқылы мәтін жаза аламыз:

```
newTextFile.write('Hello.\n')
```

Write әдісі әдеттегі қалпындағыдай жолдың соңына жолды аудару символын жаза алмайды, қажет болған жағдайда біз оны айқын түрде қосуымыз қажет.

LineNumbers функциясы файлдағы мәтінді жол нөмірлерімен экранға шығарудың орнына басқа файлға тікелей жазатындай етіп өзгертуге болатынын көрсетеді.

```
def lineNumbersFile(fileName, newFileName):  
    """fileName файлының құрамын newFileName  
    файлына әрбір жол алдына жол нөмірі  
    тұратындай етіп жазу.
```

Параметрлер:

```
    fileName: newFileName мәтіндік  
    файлының аты: Алынған мәтіндік  
    файлдың аты
```

```
Қайтарылатын  
мәндер: None"""
```

```
textFile = open(fileName, 'r', encoding =  
'utf-8') newTextFile = open(newFileName, 'w')  
lineCount = 1  
for line in textFile:  
    newTextFile.write('{0:<5} {1}\n'.format(lineCount,  
    line[:-1])) lineCount = lineCount + 1  
textFile.close()  
newTextFile.close()
```

Жаңа файлмен жұмыс істеп болған соң оны жабуды естен шығармауымыз керек. Жазбалар — енгізілген файлды жапқан кезінде өзгерістердің іс жүзінде дискке жазылғандығын кепілдейді. Тиімділікті арттыру мақсатында, операциялық жүйе мәтінді бірден дискіге көшірмейді. Ол әдетте жеткілікті жиынтық жиналғанша күтіп, барлығын бірден жазады. Сондықтан, егер файлды жабуды ұмытып кетіп компьютеріңіз шалыс берсе, бағдарламаңыздың соңғы жазбалары іс жүзінде жазылмай қалуы мүмкін. (Бұл компьютерлік шалыстан кейінгі, бүлінген файлдармен проблемалар тудыратын себептердің бірі болып табылады.)

Веб-сайттан оқу

Файлдарды оқу мүмкіндігі біз жасап шығарған функцияларды веб-сайттардан жүктелген шынайы мәліметтерге ыңғайлы қолдануды мүмкін етеді. Алайда, егер кейбір мәліметтерді қолмен жүктеп, сосын оны бағдарламамыз арқылы оқудың орнына веб-сайттан оқу анағұрлым ыңғайлы болатын еді. Бақытына орай, оны Python-да жүзеге асыруға болады.

Urllib.request модулі `urlopen` атты функциядан тұрады, ол веб-парағы үшін `open` функциясы арқылы қайтарылатынына ұқсас файлдық нысан абстракциясын қайтарады. `Urlopen` функциясы параметр ретінде бізге қажетті веб парақтың мекен-жайын алып кетеді. Ресми түрде веб парағын URL деп атайды, ол *Uniform Resource Locator (сәйкестендіріген ресурс көрсеткіші)* қысқартылған сөзі, ол әдетте `http://` (“http” -

“hypertext transfer protocol” (гипермәтінді жіберу хаттамасы) сөзінің қысқартылған түрі) префиксінен басталады. Мысалы, `python.org`-та бастапқы бетті ашу үшін келесіні орындаймыз:


```
>>> import urllib.request as web
>>> webpage = web.urlopen('http://python.org')
```

Бізде файлдық нысан болған кезде оны бұрын істегеніміздей оқи аламыз.

```
>>> text = webpage.read()
>>> webpage.close()
```

Кадрдан тыс Python интерпретаторы Ғаламтор арқылы веб-сервермен ақпарат аламасады да, осы веб-парақты оқиды. Алайда ғажап абстракцияның арқасында біз бұл үдерістің егжей-тегжейін ойламаймыз.

Urlopen функциясы кодтау параметрін қабылдамайтындықтан, read функциясы веб-парағының мәтіні қалай кодталғандығын хабарлай алмайды. Сәйкесінше, read - bytes нысанын (байттық массив) жолдың орнына қайтарады. bytes нысаны айқын түрде интерпретацияланбайтын өңделмеген байттар тізбегінен тұрады. Экранға шығарудан бұрын bytes нысанын жолға түрлендіру үшін келесіде көрсетілгендей decode әдісін қолданамыз:

```
>>> print(text.decode('utf-8'))
<!doctype html>
:
```

(Ал, UTF-8 не екенін 6.3 тарауда қарастырамыз.) Экранға text шығарған кезде python.org үй парағының HTML (hypertext markup language қысқартылған түрі (гипермәтіндік белгі тілі)) коды болып табылады. HTML дегеніміз - көптеген веб парақтар жазылатын тіл болып табылады.

Осы әдіспен, URL білетін болсақ, мәліметтер файлдарын веб-сайттан жүктей аламыз. Мысалы, АҚШ-тың Тамақ өнімдері және дәрі-дәрмек сапасын бақылау басқармасы (FDA) қайтарылған өнімдердің тізімін жасап, оны мына сайтта жаңартып отырады <http://www.fda.gov/Safety/Recalls/default.htm>. Бұл тізімнен тыс өңделмеген мәлімет те бізге қолжетімді. Сондықтан ұзақ уақыт ішінде жиналған пікірлер туралы статистиканы жинау үшін оны оқып талдайтын бағдарлама жаза аламыз. Алдымен мәліметтердің түріне қарап алайық. Файл өте ұзын болғандықтан, біз басында экранға тек алғашқы жиырма жолды readline әдісін шақыру арқылы шығарып аламыз, ол бір мәлімет жолын for циклінде жол айнымалысына қояды:

```
>>> url = 'http://www.fda.gov/DataSets/Recalls/RecallsDataSet.xml'
>>> webpage = web.urlopen(url)
>>> for i in range(20):
...     line = webpage.readline()
...     print(line.decode('utf-8'))
...
```

Бұл цикл экранға келесіге ұқсас нәрсені шығарады.

```
<?xml version="1.0" encoding="UTF-8" ?>

<RECALLS_DATA>
<PRODUCT>
  <DATE>Mon, 11 Aug 2014 00:00:00 -0400</DATE>
  <BRAND_NAME><![CDATA[Good Food]]></BRAND_NAME>
  <PRODUCT_DESCRIPTION><![CDATA[Carob powder]]></PRODUCT_DESCRIPTION>
  <REASON><![CDATA[Salmonella]]></REASON>
  <COMPANY><![CDATA[Goodfood Inc.]]></COMPANY>
  <COMPANY_RELEASE_LINK>http://www.fda.gov/Safety/Recalls/ucm40969.htm
    </COMPANY_RELEASE_LINK>
  <PHOTOS_LINK></PHOTOS_LINK>
</PRODUCT>
:
</RECALLS_DATA>
```

Бұл файл стандартты XML (extensible markup language (белгінің кеңейтілген тілі) сөзінің қысқартылған түрі) форматында берілген. XML-де мәлімет элементтері сәйкес келетін тег жұптарына кіргізілген. Бұл файлда, қайтарылған тауардың әрқайсысы <PRODUCT> ... </PRODUCT> тег жұбына кіргізілген. Бұл элементтің ішінде басқа элементтер бар, олар тауар туралы толық ақпарат беретін сәйкес тегтерге кіргізілген.

Ойлануға берілетін сұрақ 6.18 жоғарыда келтірілген

<PRODUCT> ... </PRODUCT> тэгтеріне алынған тауардың (жалған) мысалын қарастырыңыз. Тауарды қандай компания өндірді? Тауар қалай аталады? Неліктен ол қайтарылды? Қашан қайтарылды?

Бұл файлдағы мәліметті пайдаға асырмастан бұрын, біз қайтарылған тауардың жеке сипатын анықтауды үйреніп алуымыз керек. Мұндағы файл тақырып жолынан басталатынына назар аударыңыз, ол XML нұсқасын және мәтіннің кодтамасын сипаттайды. Бірнеше бос жолды өткізіп, біз

барлық пікірлердің сәйкес <RECALLS_DATA> ... </RECALLS_DATA> тэг жұптарына алынғанын көреміз. Сондықтан бірінші тауарды алу үшін <RECALLS_DATA> тәгіне дейін оқылуды жалғастыру керек. Оны while циклінің көмегімен жүзеге асыруға болады:

```
url = 'http://www.fda.gov/DataSets/Recalls/RecallsDataSet.xml' webpage = web.urlopen(url)
```

```
line = ''
while line[:14] != '<RECALLS_DATA>':
    line = webpage.readline()
    line = line.decode('utf-8')
```

Файлдық нысанның абстракциясында күңгірт көрсетілген файлға сілтеуші, келесі оқылатын символдың позициясын қадағалайды. Осылайша while циклі файлға сілтеушіні <RECALLS_DATA> тәгінен кейінгі тұрған жолдың басына жылжытуға жәрдемдеседі.

Ойлануға берілетін сұрақ 6.19 *Неліктен циклдың алдында line-ді инициалдау керек?*

Ойлануға берілетін сұрақ 6.20 *Неліктен line != '<RECALLS_DATA>' while циклінде шарт ретінде жұмыс істемейді? (Бұл жолдың соңында қандай басқарушы символ “жасырынған”?)*

Тауарлық элементті қалай айқындау керектігін суреттеу үшін олардың тек біреуін ғана экранға шығару алгоритімін жазайық. Содан кейін біз бұл кодты барлық тауарлық элементтерді экранға шығару үшін қоя аламыз немесе мәліметті толығырақ етіп өңдей алатындай етіп кодты модификациялай аламыз. Файлға сілтеуші болатын <PRODUCT> тәгі басында деп болжай отырып, біз экранға </PRODUCT> тәгіне сәйкестікті тапқанша, оларды шығару керек болады. Оған басқа while циклінің көмегімен жетуге болады:

```
line = webpage.readline()      # <PRODUCT> line
line = line.decode('utf-8')    жолын оқиды
while line[:10] != '</PRODUCT>': # </PRODUCT>
    print(line.rstrip())        шыққанын көргенше      #
    line = webpage.readline()  жолын экранға
    шығарамыз                # келесі line =
    line.decode('utf-8')       жолын оқимыз
```

```
print(line.rstrip()) # экранға </PRODUCT>
жолын шығарады
```

```
def printProducts():
    """FDA қайтарылған тізіміндегі

    тауарлар тізімін экранға шығарады.

    Параметрлер: none

    Қайтарылатын
    мәндер: None"""

    url =
    'http://www.fda.gov/DataSets/Recalls/RecallsDataSet.xml'
    webpage = web.urlopen(url)

    line = ''

    while line[14] != '<RECALLS_DATA>': # line =
        webpage.readline() алдыңғы тақырыптарды оқиды
        line = line.decode('utf-8')

    productNum = 1
    line = webpage.readline() # бірінші
    <PRODUCT> line line = line.decode('utf-8') оқиды
    while line[15] != '<RECALLS_DATA>': #
        print(productNum) тауарлар таусылғанша
        while line[10] != '</PRODUCT>': #
            print(line.rstrip()) тауар элементін экранға
            шығару
            line = webpage.readline()
            line = line.decode('utf-
            8')
            print(line.rstrip())
            productNum = productNum
            + 1
            line = webpage.readline() # келесі <PRODUCT>
            line line = line.decode('utf-8') жолын оқу

    webpage.close()
```

Жаңа while циклі және тауар санайтын сумматорды басқаратын жаңа нұсқаулар қызылмен белгіленген. Цикл ішіндегі циклді әдетте *тіркемеленген цикл* дейді, бұл цикл күрделі болып көрінуі мүмкін, алайда алдыңғы жазылған ішкі while циклді (ортада тұрған қара түсті бес нұсқау) функционалдық абстракция түрінде беруге көмектеседі, ол экранға бір тауарды шығарады. Сыртқы цикл осы сегментті ақырғы

</RECALLS_DATA> тегі шыққанша қайталайды. Сыртқы циклдың шарты циклдың алдындағы бірінші <PRODUCT> жолының оқылуымен инициалданады, ал циклдың орындалуы нәтижесінде келесі <PRODUCT> (немесе </RECALLS_DATA>) жолы циклдың соңында оқылғанда жалған цикл шартына әкеледі.

Біздің функция экранға тек тауар жөнінде ақпаратты шығаруына қарамастан, ол мәліметтерді одан әрі қызықты етіп өңдеу үшін негіз жасайды. 6.5. Тараудағы 6.5.7 жаттығуы Сізден осы құрылымды белгілі бір жылы белгілі бір себеппен қайтарылған тауар саны жөнінде ақпарат алу үшін қолдануды үйретеді.

Жаттығулар

6.2.1. lineNumbers функциясын ол экранға тек қана әрбір оныншы жолдың нөмірін шығаратындай етіп өзгертiңiз (1, 11, 21, . . . жолдары үшін).

6.2.2. Функцияны жазыңыз

wcWeb(url)

ол URL белгіленген веб-сайттан мәтіндік файлды оқиды және файлдағы сөз санын 6.1. Тараудағы wordCount5 функциясының соңғы нұсқасын қолданып қайтарады. Сіз функцияңызды <http://www.gutenberg.org> сайтындағы Гутенберг Жобасының кітаптарына сынай аласыз. Кез келген кітапқа “Қарапайым UTF-8 мәтінін” немесе ASCII нұсқасын тандаңыз. Кез келген жағдайда да, файл .txt файлының кеңейімімен аяқталуы керек. Мысалы, wcWeb('http://www.gutenberg.org/cache/epub/98/pg98.txt')

Чарльз Диккенстің Екі қала туралы хикаятатты кітабындағы жолдар санын қайтаруы керек. Сонымен бірге Екі қала туралы хикаятатты кітабының айна көшірмесін мына сайттан алуға болады

<http://discovercs.denison.edu/chapter6/ataleoftwocities.txt>

6.2.3. Функцияны жазыңыз

wcLines(fileName)

ол 6.1. Тарауындағы wordCount5 функциясын қолданып экранға аты берілген файлдағы әрбір жолдағы сөздердің санын шығарады.

6.2.4. Функцияны жазыңыз

`pigLatinDict(fileName)`

ол аты берілген сөздік файлдағы әрбір сөздің “торай латыны” баламасын экранға шығарады. (6.3.3 жаттығуын қараңыз.) Файлдың әрбір жолында бір сөз делік. Функцияңызды өзіңіз жасаған шағын файлдарға тексеріп бастаңыз. Шынайы сөздік файлын көптеген Mac OS X және Linux компьютерлерінен

`/usr/share/dict/words` табуға болады. Сондай-ақ сөздік файлын кітаптың веб-сайтынан алуға болады.

6.2.5. Алдыңғы жаттығуды қайталаңыз, бірақ функция енді бір жолға бір “торай латыннан” нәтижелерді жаңа файлға жазуы керек. Жаңа файлға атау беру үшін екінші параметрді қосыңыз.

6.2.6. Функцияны жазыңыз

`strip(fileName, newFileName)`

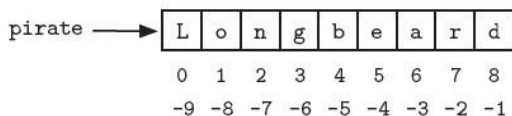
Ол `fileName` деп аталған файлдың жаңа нұсқасын жасайды, онда барлық аралық символдар (‘ ’, ‘\n’, және ‘\t’) жойылған. Екінші параметр - жаңа файлдың атауы.

6.3 ЖОЛДАРДЫ КОДТАУ

Компьютерде барлығы екілік түрде сақталатындықтан, жолдар да солай сақталуы керек. Бұл сатыда олардың іске асырылуын егжей-тегжейін білудің қажеті жоқтығына қарамастан, біраз түсінік алу пайдалы болмақ.

Индекстеу және кесінді шығару

Жол дегеніміз - іргелес жады ұяшықтарында сақталатын символдар тізбегі болып табылады. Оны көрнекі түрде келесі әдіспен суреттей аламыз:



Тоғыз символдан тұратын жолға сілтеме `pirate` атауына меншіктеледі. Жолдағы әрбір символ *индекспен* анықталады, ол оның позициясын көрсетеді. Индекстеу ыңғайы 0-ден басталады. Біз индекске тікелей рұқсатты, жол атауынан кейін тұрған оның шаршы жақшаға алынған, индексіне сілтеу арқылы алуымызға болады. Мысалы,

```
>>> pirate = 'Longbeard'
```

```
>>> pirate[0] 'L'
```

```
>>> pirate[2] 'n'
```

Мұнда әр символ тырнақшаға алынған дара белгіден тұратын жол ретінде ұсынылатынына назар аударыңыз. Жоғарыда берілген суреттегідей, біз сондай ақ *теріс индекстеуді* қолдана аламыз, ол жолдың соңынан басталады. Мысалы, суреттегі pirate[2] және pirate[-7] бірдей символға сілтенеді.

Ойлануға берілетін сұрақ 6.23 *Теріс индекстеуді қолдану арқылы, біз жолдың ұзындығына тәуелсіз және соңғы символға рұқсатты қалай алуға болады?*

Жол ұзындығы len функциясымен қайтарылады:

```
>>> len(pirate) 9
```

```
>>> pirate[len(pirate) - 1] 'd'
```

```
>>> pirate[-1] 'd'
```

```
>>> pirate[len(pirate)]
```

```
IndexError: string index out of range
```

Ойлануға берілетін сұрақ 6.24 *Жоғарыдағы соңғы нұсқау неліктен қателік берді?*

Назар аударыңыз, len - әдіс емес, ол жолдағы символ санын қайтарады, бірақ соңғы сөздің индексін қайтармайды. Жолдағы соңғы символдың оң индексі - әрқашан жолдың ұзындығы минус бір болып табылады. Жоғарыда көрсетілгендей, жоқ индекске сілтеме беру – *индекс қателігін* береді.

Егер бізге белгілі бір символдың индексін немесе жолдағы тармақшаны табу керек болса, біз find әдісін қолдана аламыз. Find әдісі тек қана *бірінші* кіріс позициясын қайтарады. Мысалы,

```
>>> pirate2 = 'Willie Stargell'
```

```
>>> pirate2.find('gel') 11
```

```
>>> pirate2.find('ll') 2
```

```
>>> pirate2.find('jelly')
```

-1

Соңғы мысалда, 'jelly' тармақшасы табылмады, сондықтан find -1 деп қайтарды.

Ойлануға берілетін сұрақ 6.25 Неліктен қайтарылатын -1 мәні, "табылмады" дегенді білдіру үшін қолданылады?

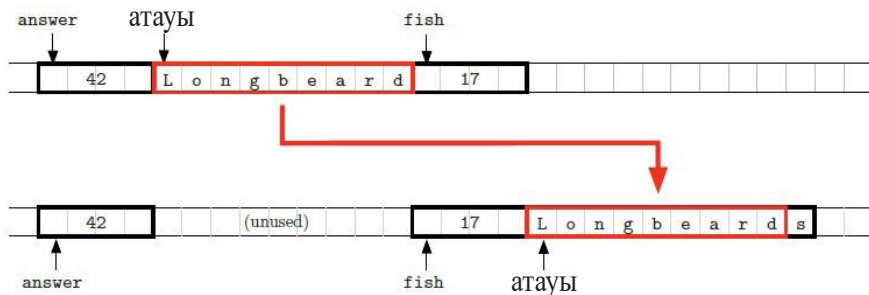
Өзгертілген жолдарды жасау

pirate (әлі де 'Longbeard') деп белгіленген жолда біз 5 индексті символды 'o'-ге өзгерткіміз келді делік. Ол үшін индекстеу мен меншіктеуді қолдану жөн көрінеді:

```
>>> pirate[5] = 'o'
```

```
TypeError: 'str' object does not support item assignment
```

Дегенмен, Python-да біз олай істей алмаймыз. Жолдар *өзгертілмейді*, яғни оларды бөлшектеп өзгерте алмаймыз. Бұл дәлелсіз (ыңғайсыз) шешім болып көрінгенімен, ол үшін дәлелді себептер бар. Олардың негізгісі - кеңістіктегі және уақыттағы тиімділік болып табылады. Жолды сақтау үшін жады жолды жасау кезінде бөлінеді. Жолдан кейін босатылған жадыны бірден басқа мәндерді сақтау үшін қолдануға болады. Егер жолдар соңына қосымша символдарды қосу арқылы ұзартыла алатын болса, онда бұндай символдарды ораналастыру үшін үлкен жады блоктары қажет және ескі жолдарды жаңа жерге көшіру керек еді. Төмендегі сурет *егер* біз 's' деп pirate соңына қоятын болсақ не болатындығын көрсетеді. Бұл мысалда, үш айнымалы аты компьютердің үш түрлі іргелес жады блоктарына сілтенеді: answer - 42 мәніне сілтенеді, pirate - 'Longbeard' жолына сілтенеді және fish - 17 мәніне сілтенеді.



17 мәні 'Longbeard' кейін бірден сақталатындықтан, біз жолды одан әрі ұзарта алмаймыз. Оның орнына 17-ден кейін үлкен жады блогі

бөлінуі тиіс. Оған ескі жол көшіріліп, 's' қосылуы керек. Бұл көп уақытты алатындықтан, жадыда кейін қолданылатын не қолданылмайтын бос кеңістік қалдырады. Символдарды тұрақты қосу осы үдерісті қайталай береді.

Жолдағы символды (немесе символдар тобын) ауыстырудың орнына біз түпнұсқа жолдың негізінде жаңа жол жасауымыз керек. Ең оңай әдіс бұл - кесінді болып табылады. Кесінді дегеніміз - жолдың жай тармақшасы. Мысалы, егер pirate - 'Longbeard' жолына сілтенсе, онда pirate[4:8] - 'bear' жолына сілтенеді. Кесіндідегі бірінші индекс - кесіндідегі бірінші символдың индексі ал екінші индекс - кесіндідегі соңғы символдан кейін бірден келетін символдың позициясы болып табылады. Ол range функциясы тәрізді жұмыс істейді.

Бізге әдетте жолдың басында немесе соңында кесінді керек болады. Осы жағдайда, біз кесінді жазбасындағы бірінші немесе соңғы индекстерді түсіре аламыз. Мысалы,

```
>>> pirate[:4] 'Long'
>>> pirate[4:] 'beard'
>>> pirate[:] 'Longbeard'
>>> pirate[:-1] 'Longbear'
```

Соңынан бастағандағы екінші өрнек pirate жолының көшірмесін жасайды. Индекс

-1 жолдағы соңғы символдың индексіне қатысты болғандықтан, соңғы мысал бізге соңғына дейінгі барлық символды (өзін есептемегенде) қосатын кесіндіні береді.

Ойлануға берілетін сұрақ 6.26 *Өзіне бірінші символдан басқасының бәрін қосыптан pirate кесіндісін қалай жасайсыз? 'Ear' мәнін алатын pirate кесіндісі туралы не айтасыз ?*

Ал енді бастапқы сұрағымызға келейік: 'Longbeard'-тағы 'e'-ді 'o'-ге қалай өзгертеміз? Оны кесіндінің көмегімен орындау үшін, бізге pirate-ке үш жолдың тіркелімін меншіктеу керек: pirate кесіндісін 'e' дейін, ал 'o' мен pirate кесіндісін 'e' кейін:

```
>>> pirate = pirate[:5] + 'o' + pirate[6:]
>>> pirate 'Longboard'
```

Жолдың көшірмесін немесе өзгертілген нұсқасын жасаудың жалпыланған тәсілі - оны әрбір символға итерациялау, символды немесе оның өзгертілген түрін жаңа жолдың соңына тіркестендіру. Келесі функция осы идеяның ең қарапайым мысалын іске асырады.

```
def copy(text):  
    """Мәтін көшірмесін қайтарады.
```

 Параметр:

 text: Жол нысаны

 Қайтарылатын мән:

 Мәтін көшірмесі """

```
newText = ""
```

```
for character in text:
```

```
    newText = newText +
```

```
    character  
return newText
```

Бұл тәсіл іс жүзінде - сумматордың тағы бір нұсқасы ғана, және біз график құруға қолданған жинақтаушы тізімдерге өте қатты ұқсайды. Оның жұмысын сипаттау үшін, text - 'abcd' болды делік. Төменде берілген кесте циклдың әр итерациясында мәннің қалай өзгертіндігін көрсетеді.

Итерация	символ	newText
1	'a'	'' + 'a' → 'a'
2	'b'	'a' + 'b' → 'ab'
3	'c'	'ab' + 'c' → 'abc'
4	'd'	'abc' + 'd' → 'abcd'

Бірінші итерацияда, text-тегі бірінші символ 'a' болып табылатын character-ге меншіктеледі. Сосын newText -ке newText пен character-дің ағымдағы мәнінің тіркелімі меншіктеледі, ол '' + 'a', немесе 'a' болып табылады. Екінші итерацияда, character-ге 'b' меншіктеледі, ал newText -ке newText пен character-дің ағымдағы мәнінің тіркелімі меншіктеледі, ол 'a' + 'b', немесе 'ab' болып табылады. Бұл тағы екі итерация бойы жалғасады, оның нәтижесінде newText мәні бастапқы text -ке ұқсас бо-

лып кетеді.

Бұл жинақтаушы жол тәсілі мәтінді түрлендіретін көптеген әдістердің негізін құрай алады. Мысалы, жолдағы бүкіл бос орындарды жоятын функция жазайық. Бұл біз жоғарыда істеген әрекетке ұқсайды, алайда мұнда біз жай ғана символды тіркегіміз келеді, егер ол аралық символы болмаса.

```
def noWhitespace(text):  
    """Мәтіннің бос орындары жойылған нұсқасын  
    қайтарады.
```

Параметр:

text: Жол нысаны

Қайтарылатын мән: Бос орындар жойылған
мәтін нұсқасы"""

```
newText = ''  
for character in text:  
    if character not in  
        '\t\n': newText =  
            newText + character  
return newText
```

Төменде келтірілген жаттығулар Сізден мәтінді әртүрлі әдістермен түзетуге осыған ұқсас функцияларды жасауды сұрайды.

Символдарды кодтау

Жоғарыда айтып өткеніміздей, компьютер жадында жолдағы символдардың әрқайсысы әлдебір әдіспен екілік жүйеге кодталуы керек. Күні кешеге дейін ағылшын тіліндегі мәтіннің көбі ASCII.³ деп аталатын форматта кодталған болатын. ASCII символдың әрқайсысына 7-биттық код меншіктейді, сәйкесінше, әртүрлі символдардың $2^7 = 128$ ұсына алады. Жадыда ASCII символдарының әрқайсысы бір байтта сақталады, ал

3ASCII - American Standard Code for Information Interchange (ақпарат алмасуға арналған американдық стандартты коды) аббревиатурасы.

байттың шеткі сол жақ биты 0 болуы керек.

Осылайша, жол байттар тізбегі түрінде сақталады, ол сондай ақ келесі түрде ұсынылады

Басқа- рушы сим- волдар	Бос орын	Тыныс белгі- лері	Сандар	Тыныс белгілері	Бас әріптер	Тыныс белгі- лері	Кіші әріптер	Тыныс белгілері	Жою
0	31 32 33	47 48	57 58	64 65	90 91	96 97	122 123	126 127	

Сурет 6.2 Ондық код диапазонды ASCII символдар жынтығының (және Unicode символдар жиынтығының негізгі латын сегменті) ұйымдасуының жалпы түрі (масштабта емес). Толық Unicode символдар жиынтығын <http://unicode.org> сілтемесінен көруге болады

0 және 127 аралығындағы сандар тізбегі. Мысалы, 'Longbeard' жолы ASCII жадында төмендегідей сақталады

```

010011000110111101101110011001110110001001100101011000010111001001100100
  76      111      110      103      98      101      97      114      100
  L       o       n       g       b       e       a       r       d

```

Ортаңғы қатар екілік кодтардың ондық баламаларынан тұрады. Сурет 6.2 ASCII символдары жиынтығының жалпы түрін көрсетеді.

ASCII символдары жиынтығы едәуір деңгейде, сондай ақ Python-да да, Unicode деп аталатын халықаралық стандартпен ығыстырылған. Тек латын әріптері үшін кодтар ұсынатын ASCII -ге қарағанда, Unicode 100 ден аса тілдердегі 100,000 астам әртүрлі символдарды кодтайды, бір символға 4 байтқа дейін қолданады. Unicode жолын үш тәсілдің бірі арқылы кодтауға болады, бірақ көбінесе UTF-8 деп аталатын айнымалы ұзындық жүйесін қолдану арқылы кодталады. UTF-8 - ASCII-мен қайта үйлесімді болғаны ыңғайлы, себебі ASCII символдары жиынтығындағы әрбір символ дәл сондай 1-байттық форматта UTF-8-де кодталған. Python-да біз Unicode кодын (ондық санау жүйесінде) кез келген символ үшін ord функциясын қолдану арқылы қарай аламыз. (ord - “ordinal” (реттік) сөзінің қысқартылған түрі). Мысалы,

```
>>> ord('L') 76
```

chr функциясы - ord-тың кері шамасы болып табылады; белгіленген Unicode коды үшін chr сәйкес символды қайтарады.

```
>>> chr(76)
'L'
```

ord және chr функцияларын әріптер мен сандар арасында түрлендіру үшін қолдануға болады. Мысалы, біз экранға print функциясы арқылы сандық мәнді шығарған кезде, санның әрбір цифрі көрсетілу үшін сәйкес символға түрленуі керек. Басқа сөзбен айтқанда, 0 мәні '0' символына түрленуі керек, 1 мәні '1' символына түрленуі керек, және т.с.с. Цифрлық символдарға арналған Unicode кодтары - ыңғайлы түрде тізбектеліп орналасқан, сондықтан кез келген цифрлық символ үшін код - '0' кодына тең, ол - ord('0') плюс цифрдің мәні болып табылады. Мысалы,

```
>>> ord('2') 50
>>> ord('0') + 2 50
```

Сондықтан кез келген бір разрядты бүтін сан үшін біз сәйкес символды оның мәнін плюс ord('0')-ді chr функциясына жіберу арқылы ала аламыз. Мысалы:

```
>>> chr(ord('0') + 2) '2'
```

Келесі функция осы идеяны digit атты кез келген бір разрядты бүтін сан үшін жалпылайды:

```
def digit2String(digit):
    """Бүтін санды цифрді оның жолдағы түріне
    түрлендіреді.
```

```
    Параметр:
```

```
        digit: 0, 1, ..., 9 ішіндегі бүтін сан
```

```
    Қайтарылатын мәндер:
```

```
        Сәйкес символ '0', '1', ..., '9' немесе
```

```
        None, егер аргумент ретінде цифр
```

```
        берілмесе
```

```
    """
```

```
    if (digit < 0) or
```

```
        (digit > 9):
```

```
        return None
```

```
    return chr(ord('0') + digit)
```

Егер digit ондық цифр болмаса, біз оны анықтай ала біліп, сәйкес әрекет етуіміз керек. Бұл жағдайда, біз тек қана None қайтарамыз.

Осыған ұқсас идеяны символды санға түрлендіру үшін қолдануға болады. Әліпбидің әрпін оның позициясын білдіретін бүтін санға түрлендіру керек делік. Басқа сөзбен айтқанда, бізге 'A' немесе 'a' символын - 1-ге, 'B' немесе 'b' - 2-ге, т.т. түрлендіру керек. Цифрлар үшін символдар секілді, бас әріптер мен кіші әріптердың кодтары тізбекетіліп орналасқан. Сондықтан бас әріптер біз 'A' кодын әріптер кодынан алып, 'A' орнына қоямыз. Дәл осылай, 'a' кодын кіші әріптер кодынан алып 'a' орнына қоя аламыз. Өзіңіз байқап көріңіз:

```
>>> ord('D') - ord('A') 3
```

Бұл бізге қажет мәнді бірлікке азайтып беретін болғандықтан, әріп үшін дұрыс позицияны алу үшін біз жай ғана бірді қоса аламыз.

```
>>> ord('D') - ord('A') + 1 4
```

Келесі функция if нұсқауларын үш ықтимал жағдайды өңдеу үшін қолданады.

```
def letter2Index(letter):
    """Әліпбидегі әріп позициясын қайтарады.

    Параметр:
        letter: Бас әріп немесе кіші әріп

    Қайтарылатын мән: Әліпбидегі әріп
    позициясы"""
    if (letter >= 'A') and (letter <= 'Z'): # upper case
        return ord(letter) - ord('A') + 1
    if (letter >= 'a') and (letter <= 'z'): # lower case
        return ord(letter) - ord('a') + 1
    return None # non-letter
```

Жоғарыда сандарды салыстырғанымыздай символдарды да дәл солай салыстыруға болатынына назар аударыңыз. Салыстырылатын мәндер - іс жүзінде символдардың Unicode кодтары болып табылады, алайда әріптер мен цифрлар тізбектеліп орналасқандықтан, салыстыру әліпби ретімен жүреді. Сонымен қатар ұзынырақ жолдарды дәл осылай салыстыруға болады.

```
>>> 'cat' < 'dog' True
>>> 'cat' < 'catastrophe' True
```

```
>>> 'Cat' < 'cat' True
>>> '1' < 'one'
True
```

Ойлануға берілетін сұрақ 6.27 Неліктен 'Cat' < 'cat' және '1' < 'one' екеуі de True? 6.2. Суретін қараңыз..

Жаттығулар

6.3.1. Сізде word атты айнымалыда сақталған жол бар делік. Экранға келесіні қалай шығаруға болатынын көрсетіңіз

- (a) Жол ұзындығы
- (b) Жодағы бірінші символ
- (c) Жолдағы үшінші символ
- (d) Жолдағы соңғы символ
- (e) Жолдағы соңғы үш символ
- (f) Екінші, үшінші және төртінші символдардан тұратын жол
- (g) Символдардың соңынан бесінші, төртінші және үшіншісінен

тұратын жол

- (h) Соңғы символдан басқа символдардың бәрінен тұратын жол

6.3.2. Функцияны жазыңыз
username(first, last)

ол адамның тегі түрінде берілген қолданушының атын қайтарады, одан кейін астын сызу символы мен адам атының бірінші әрпі жүреді. Мысалы, username('martin', 'freeman') - 'freeman_m' жолын қайтару керек.

6.3.3. Функцияны жазыңыз
piglatin(word)

ол word жолының “торай латынындағы” баламасын қайтарады. “Торай латыны” жолдың бірінші символын соңына жылжытады, және оны 'au'-мен толықтырады. Мысалы, 'python'-ның “торай латыны” - 'ythonpay'.

6.3.4. Мысалы

quote = 'Well done is better than well said.'

(Бенджамин Франклиннің дәйексөзі.) Келесі сұрақтардың әрқайсысына жауап беру үшін кесінді шығару жазбасын қолданыңыз.

- (a) quote-тің қандай кесіндісі - 'done'-ға тең?
- (b) quote-тің қандай кесіндісі 'well said'-ке тең?
- (c) quote-тің қандай кесіндісі 'one is bet'-ке тең?
- (d) quote-тің қандай кесіндісі 'Well do'-ге тең?

6.3.5. Функцияны жазыңыз

noVowels(text)

ол text жолының дауыстыларының барлығы жойылған нұсқасын қайтарады. Мысалы,

noVowels('this is an example.') 'ths s n xmpl.' жолын қайтару керек.

6.3.6. Мәселен, Сіз код жасайсыз делік, ол жолды жаңа жолмен ауыстырады делік, ондағы жұп индексті түпнұсқа символдардың барлығы тақ индексті символдардың барлығынан кейін тұрады. Мысалы, 'computers' жолы 'cmuesoptr' деп кодталады.

Функцияны жазыңыз

encode(word)

ол word деген жолдың кодталған нұсқасын қайтарады.

6.3.7. Функцияны жазыңыз

decode(codeword)

ол алдыңғы жаттығудағы encode функциясындағы үдерісті қайта қосады.

6.3.8. Функцияны жазыңыз

daffy(word)

ол жолға Даффи Дактың “шолжындығын” қосып қайтарады (Даффи 's' дыбысын соңынан 'th' естілетіндей етіп айтатын болады). Мысалы, daffy("That's despicable!") "That'sth desthpicable!" жолын қайтару керек.

6.3.9. Мәселен, Сіз машиналардың барлық нөмірлік белгілері әріп жолынан кейін сандар жолы тұратын штатта жұмыс істейсіз делік, мысалы 'ABC 123'. Функцияны жазыңыз

randomPlate(length)

ол кездейсоқ түзілген

length бас әріптіренен, және одан кейін бос орын, одан кейін length цифрларынан тұратын нөмірлік белгіні білдіретін жолды қайтарады.

6.3.10. Функцияны жазыңыз

int2String(n)

ол n оң бүтін сандарының оң мәнін оның жолдық баламасына str функциясын қолданбай түрлендіреді. Мысалы, int2String(1234) - '1234' жолын қайтару керек. (Осы тараудағы digit2String функциясын қолданыңыз.)

6.3.11. Функцияны жазыңыз

reverse(text)

ол text жолының көшірмесін кері тәртіпте қайтарады.

6.3.12. Кейбір адамдар тұрмыс құрған кезде, олар жұбайының тегін алады немесе дефис арқылы өзінің тегі мен жұбайының тегін біріктіреді. Функцияны жазыңыз

marriedName(fullName, spouseLastName, hyphenate)

ол тұлғаның жаңа тегін егер hyphenate - True болса, дефиспен біріктірілген тегін толық етіп қайтарады, , ал егер hyphenate - False болса, тұлғаның жаңа тегін жұбайының тегімен бірге қайтарады. fullName параметрі - тұлғаның 'Firstname Lastname' формасындағы қазіргі толық аты-жөні болып табылады, ал spouseLastName параметрі - жұбайының тегі болып табылады. Мысалы, marriedName('Jane Doe', 'Deer', True) 'Jane Doe-Deer' жолын қайтару керек болса, ал marriedName('Jane Doe', 'Deer', False) 'Jane Deer' жолын қайтару керек.

6.3.13. Функцияны жазыңыз

letter(n)

ол әліпбидегі n-ші бас әрпін chr және ord функциялары арқылы қайтарады.

6.3.14. Функцияны жазыңыз

value(digit)

ол digit жолының параметріне сәйкес келетін бүтін санды қайтарады. Параметр жалғыз '0', '1', ..., '9' символынан тұратын болады. ord функциясын қолданыңыз. Мысалы, value('5') - 5 бүтін санын қайтару керек.

6.3.15. chr және ord функцияларын қолдану арқылы емтихан бағасының сандық түрін әріптік бағаға аз мөлшерде if/elif/else нұсқауларын қолданып түрлендіруге болады. Іс жүзінде, 60 пен 99 арасындағы кез келген бағаны, біз chr және ord қолданып бір ғана нұсқаумен бере аламыз. Осыны келесі функцияда SOMETHING-ді алмастыру арқылы көрсетіңіз.

```
def letterGrade(grade):  
    if grade >= 100:  
        return 'A'  
    if grade > 59:  
        return SOMETHING  
    return 'F'
```

6.3.16. Функцияны жазыңыз

capitalize(word)

ол word жолының нұсқасын бірінші бас әріппен қайтарады. (Сөз бас әріптен жазылып қойған болуы мүмкін, абай болыңыз!)

6.3.17. 6.1.11 жаттығудағы жұптықты тексеру тәрізді, бақылау сомасы жолдарды жіберу кезіндегі қателіктерді анықтау үшін жалпыланған жолдарға да қолданыла алады. Жолдың бақылау сомасын есептеудің ең оңай әдісі бұл - символдың әрқайсысын оның әліпбиіндегі

позициясын білдіретін бүтін санға түрлендіру ($a = 0, b = 1, \dots, z = 25$), бұл бүтін сандардың барлығын қосып, сосын соманы қайтадан символға түрлендіру. Оңай ету үшін, жол тек кіші әріптерден тұрады делік. Соманың 25-тен асуы ықтимал болатындықтан, бізге соманы 26-ға бөлгендегі қалдықты табу арқылы 0-25 диапазоны аралығындағы санға түрлендіру керек болады. Мысалы, 'snow' жолы үшін бақылау сомасының символын табу үшін

$(18 + 13 + 14 + 22) \bmod 26$ есептейміз (себебі $s = 18, n = 13, o = 14$ және $w = 22$), ол $67 \bmod 26 = 15$ -ке тең. 15 = p болғандықтан, біз 'p' -ді

бұл символдардың тізбегін жібергенде 'snow' -дің соңына қосамыз. Соңғы символ сосын алушының жағында тексеріледі. Функцияны жазыңыз

`checksum(word)`

ол word-ті соңында сәйкес бақылау сомасының символымен қайтарады. Мысалы, `checksum('snow')` - 'snowp' қайтару керек. (Сыбырсөз: chr және ord қолданыңыз.)

6.3.18. Функцияны жазыңыз

`checksumCheck(word)`

ол word соңындағы (6.3.17 жаттығуды қараңыз) бақылау сомасының символы дұрыстығын тексереді. Мысалы, `checksumCheck('snowp')` True қайтару керек, ал `checksumCheck('snowy')` - False қайтару керек.

6.3.19. Юлий Цезарь құпия хабарды қарапайым шифрлеу сұлбасы арқылы жіберген делінеді, ол сұлба қазіргі кезде Цезарь шифрі деп аталады. Цезарь шифрінде, мәтіндегі әрбір әріп белгіленген біраз қашықтыққа жылжытылған әріппен алмастырылады, ол ілгерлеу деп аталады. Мысалы, 3-ке ілгерілегенде, А орнына D келеді, В орнына Е келеді, және т.с.с. Әліпбидің соңына келгенде кодталу Х орнына А, Y орнына В, және Z орнына С келумен шырғаланады. Функцияны жазыңыз

`encipher(text, shift)`

ол text-тің Цезарь шифрімен шифрленген нәтижесін белгіленген shift ілгерілеуімен қайтарады. Мәселен, text тек қана бас әріптерден тұрады делік.

6.3.20. Алдыңғы тапсырмадағы encipher функциясын қосымша логикалық параметрге тәуелді tech-ті шифрлейтіндей және шифрді жай жазуға айналдыратындай етіп өзгертіңіз.

6.4 СЫЗЫҚТЫҚ УАҚЫТ АЛГОРИТМДЕРІ

Біздің бір жолдық wordCount1 функциямыз 6.1 тараудың басынан бастап wordCount5 функцияның соңғы нұсқасына қарағанда әлдеқайда жылдам жұмыс істеуі керек деп ойлау жөн көрінетін тәрізді. Дегенмен, count method функциясының жұмыс істеу қағидасы жөніндегі талқылаудан екі функцияның да мәтіндегі әрбір символды тексеру керектігі анық көрінеді. Сол үшін ақиқат одан әрі күрделі сияқты.

1.3 тарауда біз кірістік мәліметтер көлеміне сәйкес қарапайым амалдардың қажетті санын есептеу арқылы алгоритмдердің уақыттық күрделілігін талдаған болатынбыз. Қарапайым амалды орындау кіріске не берілсе де әр кезде бірдей уақыт мөлшерін жұмсайды. Біз қарапайым амалдар деп - негізгі арифметикалық әрекеттерді, сандар мен дара символдарды меншіктеу және салыстыруды айта аламыз.

Ойлануға берілетін сұрақ 6.28 *word1 және word2 айнымалы атауларының екеуіне де жолдық мәндер меншіктелді делік. word1 < word2 секілді осы екі жолды салыстыру да қарапайым амал бола ма?*

Жол үсті амалдарының уақытша күрделілігі мұқият талдану қажет. word1 < word2 екендігін анықтау үшін бірінші символдарды салыстыру керек, сосын, егер бірінші символдар тең болса, екінші символдар салыстырылады, осылай екі символ әр түрлі болмағанша немесе жолдың біреуінің соңына жетпегенше жалғасады. Осылай, word1 және word2 салыстыруға қажетті, жеке алынған символдарды салыстырудың жалпы саны осы айнымалылар атауына меншіктелген мәндерге тәуелді болып табылады.

Ойлануға берілетін сұрақ 6.29 *Егер word1 = 'python' және word2 = 'rattlesnake' болса, онда word1 < word2 ақиқат деп қорытындылау үшін қанша символды салыстыру керек? Егер word1 = 'rattlesnake' ал word2 = 'rattlesnakes' болса ше?*

Бірінші жағдайда өрнектің ақиқаттығын анықтау үшін бірінші символдарды ғана салыстыру керек. Ал, екінші жағдайда болса, немесе егер екі жол бірдей болса, жауап беру үшін, бізге символдың әрқайсысын салыстыру керек. Сондықтан, егер жол бос емес десек, онда минималды салыстыру саны 1-ге тең болады, ал максималды салыстыру саны - n-ге тең, мұндағы n дегеніміз - қысқарақ жолдың ұзындығы. (6.4.1 жаттығуда жолдарды салыстыру қалай жұмыс істейтіндігі толығырақ көрсетілген.)

Басқа сөзбен айтқанда, жолдарды салыстырудың уақыттық күрделілігі ең жақсы жағдайда - константа болып табылады, себебі ол кірістік сигналдың көлеміне тәуелді емес, ал ең нашар жағдайда жолдарды салыстырудың уақыттық күрделілігі - n - қысқарақ жолдың ұзындығына тура шамалас.

Ойлануға берілетін сұрақ 6.30 *Тәжірибеде шынайы уақыттық күрделелілікті бағалауда қайсысы үлгілі: уақыттық күрделіліктің ең жақсысы ма әлде ең нашар жағдайы ма ?*

Біз, әдеттегідей, алгоритмнің уақыттық күрделілігінің ең нашар жағдайына көбірек көңіл бөлеміз, себебі ол алгоритмнің орындалып бітуінің болжалды уақытын сипаттайды. Ол сондай ақ тәжірибеде болатын нәрсеге өте ұқсас келеді. Символдарды салыстырудың орташа саны жолдарды салыстырған кезде шашамен қысқарақ жолдың ұзындығының жартысын немесе $n/2$ құрайтындығы интуициялы түрде түсінікті. $N/2/2$ - n -ге тура шамалас болғандықтан ол ең жақсы уақыттық күрделілік жағдайынан гөрі, ең нашар жағдайына көбірек ұқсас.

Ойлануға берілетін сұрақ 6.31 *Егер бір немесе екі жол да константа болса не болады? Мысалы, 'buzzards' == 'buzzword' бағалау үшін қанша символдың салыстырылуы керек?*

Мәндердің екеуі де айнымалы емес, константа болғандықтан, біз бұл салыстыру үшін ылғи да бес салыстыру символдары болуы керек екендігін білеміз (жолдардағы бесінші символдардың әртүрлі екендігін анықтау үшін). Басқа сөзбен айтқанда, осындай жолды салыстыруға қажетті символдарды салыстыру саны оның құрамындағы кез келген алгоритм үшін кірістік мәліметтерден тәуелсіз болады. Сәйкесінше, уақыт күрделілігі тұрақты болып табылады.

Ал енді wordCount1 және wordCount5 функцияларын салыстыруға қайта оралайық.

Соңғы wordCount5 функциясы төменде берілген.

```
1 def wordCount5(text):
2     """ (docstring omitted) """
3
4     count = 0
5     prevCharacter = ' '
6     for character in text:
7         if character in ' \t\n' and prevCharacter not in ' \t\n':
8             count = count + 1
9             prevCharacter = character
10    if prevCharacter not in ' \t\n':
11        count = count + 1
12    return count
```

Функциядағы 4 және 5 жолдарындағы алғашқы екі нұсқаудың әрқайсысы - қарапайым амал болып табылады, себебі оның орындалу уақыты `text` кірістік параметрінің мәніне тәуелді емес. Бұл нұсқаулардан кейін 6-9 жолдарында `for` циклі жүреді.

Ойлануға берілетін сұрақ 6.32 *Мәселен, `text` - n символдан тұрады делік (яғни, $\text{len}(\text{text}) = n$ -ге тең). n шамаларында `for` циклі неше рет қайталанады?*

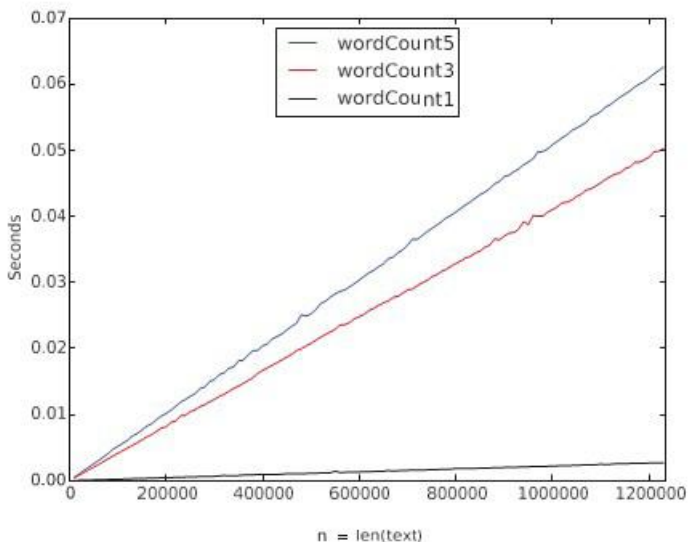
Цикл `text`-тегі әрбір символ үшін бірден n рет қайталанады. Осы итерациялардың әрқайсысында `text`-тегі жаңа символ көрінбей `character` индекстік айнымалысына меншіктеледі, оны біз бір итерация үшін бір қарапайым амал ретінде есептейміз. Сосын 7 жолында салыстыру орындалады. `character` және `prevCharacter` екеуі де бір символдан тұратындықтан, және ол символдардың әрқайсысы ' \t\n' жолының үш символымен салыстырылатындықтан, онда бұл жерде жалпы алғанда алтыдан артық емес символ салыстыру саны болады. `count` 8 жолында тек қана шарттың ақиқат болғанында ғана ұлғаяды, біз әр итерацияда осылай болады деп есептейміз, себебі бізге ең нашар уақыттық күрделіліктің ең нашар жағдайы керек. Сонда 9 жолындағы бұл еселеу мен меншіктеу цикл денесінде тағы екі қарапайым амал қосады, сонда барлығы тоғыз болады. Осы тоғыз қарапайым амалдар `text` жолындағы символдың әрқайсысы үшін бір рет орындалатындықтан, циклдегі қарапайым амалдардың жалпы саны $9n$ құрайды. Ақырында, 10 жолындағы салыстыру үш қарапайым амал деп, 11 жолындағы еселеу - бір, ал 12 жолындағы `return` нұсқауы бір қарапайым амал деп саналады.

Осының бәрін қосқанда, біз ең нашар жағдайда, жалпы алғанда, $9n + 7$ қарапайым амалын аламыз.

Ал енді wordCount1 және wordCount3 функцияларын орындауға қажетті қарапайым амалдар санын есептейік. Олар ойдағыдай жұмыс істемесе де, пайдалы салыстырумен қамтамасыз етеді. wordCount1 функциясы төменде жасалған.

```
def wordCount1(text):  
    """ (docstring omitted) """  
  
    return text.count(' ') + text.count('\t') + text.count('\n')
```

wordCount1 функциясы text параметріндегі бос орын, табуляция және жол аудару санын есептеу үшін жай count әдісін үш рет шақырады. Алайда, алдыңғы тараудан, бұл екі функцияның әрқайсысы for циклін жасыратындығын байқаймыз, ол text-тегі әрбір символды count-ке жіберілген аргумент символымен салыстырады. Біз wordCount1 уақыттық күрделілігін төмендегі wordCount3-ге қарап бағалай аламыз, оны біз wordCount1 әрекетін келтіру үшін жасап шығарған болатынбыз.



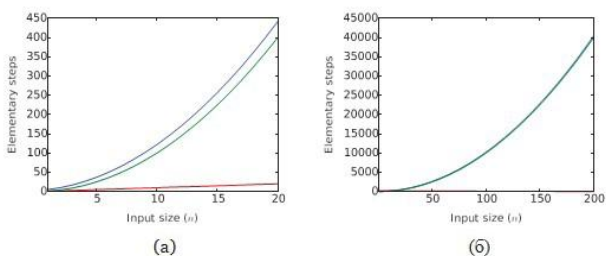
Сурет 6.3 wordCount1, wordCount3, және wordCount5 функцияларының уақыттық күрделілігін эмпирикалық салыстыру.

Екіншіден, біз әртүрлі қарапайым амалдардың әртүрлі шынайы уақыт санын алатынын ескермедік. Үшіншіден, және мұндағы ең маңыздысы, Python-да барлық кіріктірілген функциялар мен әдістерін іске асыруда жасырынған көптеген оңтайландыру түрі бар, мысалы count оларды Python кодына қарағанда жылдамырақ етеді. Біздің абстракциялық деңгейде бұл оңтайландырушылар қол жетімсіз.

Асимптоталық уақыт күрделілігі

Осы тәрізді техникалық факторлар іс жүзіндегі орындау уақытын болжауды едәуір қиындата түседі. Алайда, бақытына орай, бұл факторлар кіріс көлемі өскен сайын уақыттық күрделілік артатын жылдамдықтай өте маңызды емес. Ал бізді расында тек қана кіріс өте үлкен болғандағы алгоритмнің уақыттық күрделілігі өсетін жылдамдық қызықтырады. Бізге қызықты келесі ұғым бұл - масштабталулық; кіріс мәліметтері өте үлкен көлемге арттырылған кезде алгоритмнің олармен қалай жұмыс істейтіндігі біз үшін қызық. Интуициялық түрде, бұл алгоритмдердің түгелі дерлік кіріс мәліметінің көлемі шағын болғанда жылдам бола түсетіні түсінікті нәрсе. (Шын мәнінде, тіпті wordCount5 функциясының Моби Дик кітабының толық мәтінімен, шамамен 1,2 миллион символмен, жұмыс істеуге кеткен уақыты бірден он секундтан аз болды.) Дегенмен, кіріс мәліметінің көлемі үлкен болғанда, уақыттық күрделіліктегі айырмашылық едәуір үлкен болмақ.

Осу жылдамдығына мұқият зерттеу алгоритмдердің уақыттық



Сурет 6.4 $n^2 + 2n + 2$ (көк), n^2 (жасыл), және n (қызыл) уақыт күрделіліктерінің екі көрінісі.

күрделілігі өрнектерін кіріс мәліметтерінің көлемдерін айтарлықтай ұлғайтқан кезде жылдамдықтың өсуіне әсер ететін параметрлерді ғана қосып, оңайлатуға мүмкіндік береді. Мысалы, $9n + 7$ өрнегін жай n -ге оңайлата аламыз, себебі n үлкейгенде, $9n + 7$ және n дәл сондай

жылдамдықпен өседі. Басқа сөзбен айтқанда, $9n + 7$ өрнегіндегі 9 және 7 константалары n ұлғайған кезде өрнектің өсу жылдамдығына ықпал етпейді. Бұл келесі кестеде көрсетілген, ол әрбір өрнектік n -ді 10 есе өсірген кезде қалай өзгертіндігін көрсетеді.

n	↑ көбейт кіш	$9n + 7$	↑ көбейт кіш
10	-	97	-
100	10	907	9.35052
1,000	10	9,007	9.93054
10,000	10	90,007	9.99301
100,000	10	900,007	9.99930
1,000,000	10	9,000,007	9.99993

Әобір қатардағы екінші бағанада сол жолдағы n мәнінің алдыңғы жолдағы n мәніне әрдайым он есеге артылған арақатынасы көрсетілген. Төртінші бағанада дәл сондай арақатынастар $9n + 7$ өрнегі үшін көрсетілген, олар n үлкейген сайын 10-ға жақындай түседі. Басқа сөзбен айтқанда, n және $9n + 7$ мәндері асылында дәл n үлкейген жылдамдықпен өседі.

Бұл идеяны асимптоталық уақыт күрделілігі дейміз. Сонымен, `wordCount1`, `wordCount3`, және `wordCount 5`-тің әрқайсысының асимптоталық уақыт күрделілігі - n -ге тең. Асимптоталық кіріс мәліметінің көлемі қандай үлкен болса да (яғни, шексіз) біздің сұранысқа жауап бере алады. Асимптота - дегеніміз шексіз алыстатылған қисық сызықтың нүктесі барынша жақындай түсетін, алайда ешқашан жете алмайтын түзу, Сіз оны математикалық талдау сабағынан білетін шығарсыз. Егер біздің n көлемдегі кіріс параметріміз шексіздікке ұмтылатын болса, $9n + 7$ асимптоталық түрде n -ге тең деп айту арқылы біз $9n + 7$ деген дәл n -нің өзі деп сендіреміз. n -ге тең болатын асимптоталық уақыттық күрделілігі бар алгоритмдер жайлы оларда сызықтық уақыт күрделілігі болады делінеді. Оларды сондай ақ *сызықтық уақыт алгоритмдері* дейміз. Жалпы алғанда, тізімнің немесе жолдың әрбір элементіне тұрақты санды амалдарды орындайтын кез келген алгоритмнің сызықтық уақыт күрделілігі болады. Біз бастаған сөздерді санайтын және жолдарды салыстыратын алгоритмдердің барлығында сызықтық уақыт күрделілігі болады.

Ойлануға берілетін сұрақ 6.34 *10 беттегі Mean алгоритмінің де сызықтық уақыт күрделілігі бар. Неліктен түсіндіріп бере аласыз ба?*

Басқа мысал ретінде, кіріс мәліметінің көлемі n болғанда $n^2 + 2n + 2$ кадамды қажет ететін алгоритмді қарастырайық. $n^2 - 2n$ және 2 екеуіне қарағанда жылдамырақ өсетіндіктен, біз бұл параметрдің екеуін де ескермей, бұл алгоритмнің асимптоталық уақыттық күрделілігі n^2 құрайды деп айтамыз. Бұл 6.4 суретте көрсетілген. Егер біз $n^2 + 2n + 2$ және n^2 бірге 6.4. суреттегі n -нің кіші мәндеріне қарастыратын болсақ, $n^2 + 2n + 2$ және n^2 біріншісіне қайшы болатындай көрінеді, алайда біз “масштабты кішірейткенде,” 6.4(b) суреттегідей, біз функциялар бірдей дерлік екенін көреміз. Сондай-ақ, алгоритмдердің екеуінің де уақыттық күрделілігінің сызықтық уақыт алгоритмге қарағанда едәуір айырмашылығы бар екенін байқаймыз. Асимптоталық уақыт күрделілігі n^2 -ге тең алгоритмдердің квадраттық уақыт күрделілігі болады делінеді. Квадраттық уақыт алгоритмдері әдетте жол мен тізімнен жол мен тізімнің ұзындығына шамалас бірнеше рет өтеді. Квадраттық уақыт алгоритмінің мысалы - тізімді сұрыптау алгоритмі болып табылады, ол алдымен жол ашып ең кішкентай элементті тауып, сосын тағы жол ашып, екінші ең кішкентай элементті іздейді, . . . , осылай n -ші ең кішкентай элементті тапқанша жалғасады. Алгоритмдер тізім бойынша n жасайтындықтан, олардың әрқайсысына n қадам керек, оның n^2 асимптоталық уақыт күрделілігі болады. (Біз бұл алгоритмді 11 тарауда толығырақ қарастырамыз.)

Кейінірек алгоритмдердің әртүрлі асимптоталық уақыт күрделілігі болатынын байқайтын боламыз. Мысалы, температураны Цельсий градусынан Фаренгейт градусына түрлендіретін, сандармен қарапайым арифметикалық амалдарды орындайтын алгоритмдерде тұрақты уақыт күрделілігі болады делінеді, себебі олар кіріске тәуелсіз бірдей уақытты алады. Кересінше жағдайда, Сіз кемемен тасымалдайтын компанияда жұмыс істейсіз делік, Сізге сұрыпталған жәшіктердің көбісін екі бірдей салмақты топқа бөлу тапсырмасы берілсін. Мұнда, жалпы алғанда, жәшіктерді барлық мүмкін деген әдістермен бөліп байқап көруден артық ешқандай алгоритм жоқ екенін байқаймыз. n жәшіктерін бөлудің $2n$ мүмкін әдісі бар болғандықтан, бұл алгоритмнің экспоненциалды уақыт күрделілігі болады. Әрі қарай 11.5 тарауда біз экспоненциалды уақыт күрделілігі алгоритмдерінің қаншалықты баяу екендігін, және олардың іс жүзінде пайдасыз екендігін байқаймыз.

Жаттығулар

6.4.1. Келесі функция жолдарды салыстырудың “астыртын” жұмысын айқын көрсетеді. $steps$ айнымалысы жеке символдарды қанша рет салыстыру керектігін санайды.

```

def compare(word1, word2):
    index = 0
    steps = 3    # number of comparisons
    while index < len(word1) and \
           index < len(word2) and \
           word1[index] == word2[index]:    # character comparison
        steps = steps + 3
        index = index + 1

    if index == len(word1) and index == len(word2):    # case 1: ==
        print(word1, 'and', word2, 'are equal.')
        steps = steps + 2
    elif index == len(word1) and index < len(word2):    # case 2: <
        print(word1, 'comes before', word2)
        steps = steps + 2
    elif index == len(word2) and index < len(word1):    # case 3: >
        print(word1, 'comes after', word2)
        steps = steps + 2
    elif word1[index] < word2[index]:                    # case 4: <
        print(word1, 'comes before', word2)
        steps = steps + 1
    else:    # word2[index] < word1[index]:              # case 5: >
        print(word1, 'comes after', word2)
        steps = steps + 1

    print('Number of comparisons =', steps)

```

index айнымалысы екі жолда ағымдағы уақытта салыстырылатын позицияларды бақылайды. while циклі index-ті index позициясындағы символдар тең болғанша арттырады. while циклі келесі үш әрекеттің бірі болғанда аяқталады:

- (a) index word1 соңына жеткенде,
- (b) index word2 соңына жеткенде, немесе
- (c) word1[index] word2[index]-ке тең болмағанда.

Егер index word1-дің басынан аяғына дейін жәнe word2-нің басынан аяғына дейін өтсе, онда екі жолдың ұзындығы бірдей болады, ал олардың ұзындығындағы барлық символдар тең болады, бұл екі жолдың тең болу керектігін білдіреді (1-жағдай). Егер index word1-дің басынан аяғына дейін өтсе, алайда index әлі де word2 ұзындығынан кем болса, онда word1 - word2-дің префксі болуы керек, сонда word1 < word2 болады (2-жағдай). Егер index word2-нің басынан аяғына дейін өтсе, ал

index әлі де word1-дің ұзындығынан кем болса, онда біз кері жағдайды аламыз да word1 > word2 болады (3-жағдай). Егер index екі жолдың да аяғына жетпесе, онда символдардың арасында сәйкессіздік болады да, бізге символдарды қайта салыстырып, олардың қайсысының кіші болғандығын анықтау керек болады (4 және 5-жағдайлар).

steps айнымалысы функциядағы салыстырудың жалпы санын есептейді. Ол символдады салыстыруды да, жол ұзындығымен салыстыруды да ескереді. steps мәні while циклінің әр итерациясындағы 3 салыстырудың алдында 3-ке артылады, және циклден кейін әр жағдайда 1-ге немесе 2-ге артылады.

Осы функциямен тәжірибе жасап көріңіз де, келесі сұраққа жауап беріңіз.

(a) compare функциясында егер төмендегідей болса неше салыстыру амалы болады

- i. 'canny' мен 'candidate' салыстырылса?
- ii. 'canny' мен 'danny' салыстырылса?
- iii. 'canny' мен 'canny' салыстырылса?
- iv. 'can' мен 'canada' салыстырылса?
- v. 'canoeing' мен to 'canoe' салыстырылса?

(b) word1 және word2 екеуі де n-символдық жол болсын делік. word1 мен word2 салыстырылғанда неше салыстыру амалы болады?

(c) word1 (m символдарымен) - word2 (n символдарымен) префикстері болсын делік. word1 мен word2 салыстырылғанда неше салыстыру амалы болады?

(d) steps мәні іс жүзінде кейбір жағдайларда салыстыру саынан артылады. Бұл қай кезде орын алады?

6.4.2. Келесі код фрагменттерінің әрқайсысы қалай жұмыс істеу керек екені туралы ойланыңыз да, сосын ол сызықтық уақыт алгоритмі ме, әлде тұрақты уақыт алгоритмі ме немесе басқа алгоритм екендігін анықтаңыз. Айнымалы атауының әрқайсысы n ұзындығындағы жолдың нысанға сілтейді. Бір символмен орындалатын кез келген амал бір қарапайым амал болып табылады делік.

- (a) name = name.upper()
- (b) name = name.find('x')
- (c) name = 'accident'.find('x')
- (d) newName = name.replace('a', 'e')
- (e) newName = name + 'son'
- (f) newName = 'jack' + 'son'
- (g) index = ord('H') - ord('A') + 1

- (h) for character in name: print(character)
- (i) for character in 'hello': print(character)
- (j) if name == newName:
print('иә')
- (k) if name == 'hello':
print('иә')
- (l) if 'x' in name:
print('иә')
- (m) for character in name:
x = name.find(character)

6.4.3. Келесі қарапайым амалдар санының әрқайсысын қажет ететін алгоритмнің асимптоталық уақыт күрделілігі қандай? Мәселен, n ір жағдайдағы кірістің ұзындығы болсын.

- (a) $7n - 4$
- (b) 6
- (c) $3n^2 + 2n + 6$
- (d) $4n^3 + 5n + 2n$
- (e) $n \log_2 n + 2n$

6.4.4. Бір тапсырманың екі алгоритміне $12n$ және n^2 қарапайым амал қажет делік. Секундына 1 миллиард амал орындайтын компьютерде іріктеу алгоритмінің әрқайсысы кіріс көлемі $n = 10, 102, 104, 106$, және 109 болғанда қанша уақытқа созылады (секундта)? N^2 амал қажет ететін алгоритм $12n$ амалды қажет ететін алгоритмнен әрқашан жылдамырақ бола ма?

6.5 МӘТІНДІ ТАЛДАУ

Осы тарауда, біз үлкен мәтіндерді талдау үшін қолдана алатын көбірек алгоритмдік тәсілдерді көрсетеміз. Алдымен 6.1. Тараудағы сөз санайтын алгоритмді жалпылаймыз да, сосын жеке сөздерді қалай іздестіру керектігін және олардың мәнмәтінін конкорданс түрінде көрсету әдісін талқылаймыз.

Есептеу және іздестіру

6.1 тарауда біз мәтіндегі сөз санын жолдағы смволдарды іріктеу кезінде бос орындарды іздестіру арқылы есептеген болатынбыз. Бұл тәсілді жалпылап, оны мәтіндегі кез келген символ кірісінің санын есептей аламыз. Келесі алгоритмнің wordCount3 функциясына ұқсастығын 245 -бетте байқаймыз:

```
def count1(text, target):
    """Count the number of target characters in text.

    Parameters:
        text: a string object
        target: a single-character string object

    Return value: the number of occurrences of target in text
    """

    count = 0
    for character in text:
        if character == target:
            count = count + 1
    return count
```

Ойлануға берілетін сұрақ 6.35 Егер target әріп десек, функцияны target-тің бас және кіші кірістерін санайтындай етіп қалай өзгертуге болады? (Сыбырсөз: Б.6 қосымшасын қараңыз.)

Ойлануға берілетін сұрақ 6.36 Егер target бірден көп символдан тұрады десек, target-тегі қандай да бір символдың кірісінің санын қалай есептеуге болады? (Сыбырсөз: “-тегі” жалғауы кілт сөз.)

Мәтіндегі қандай да бір нақты сөзді немесе тармақшаның кіріс санын іздеу үшін count1-ді жалпыласақ не болады?

Ойлануға берілетін сұрақ 6.37 Бірден көп символдан тұратын жол кірістерінң санын есептеу үшін count1 функциясында қолданылған for циклін қайта қолдануға бола ма?

Жолдағы символдардысұрыптау арқылы бір уақытта тек бір ғана символды “көруге” мүмкіндік береді, сондықтан бізде белігілі бір уақытта цикл денесіндегі мақсатты жолмен салыстыру үшін тек бір ғана символ болады. Оның орнына, бізге text-тегі бірдей ұзындықтағы көпсимволды тармақшалардың барлығын мақсатты жолмен салыстыру керек болады. Мысалы, text атты үлкен жолда мақсатты 'good' жолын табу керек делік. Сосын бізге text[0:4] пен 'good' сәйкес келетіндігін, сосын text[1:5] пен 'good', сосын text[2:6] пен 'good', және т.б. сәйкес келетінін тексеру керек. Қысқасы, index-тің 0, 1, 2, ...мәндеріне тең болатын барлық мәндері үшін, бізге text[index:index + 4] пен 'good' сәйкес келетіндігін тексеру керек. Жалпы, index-тің 0, 1, 2, ...мәндеріне тең болатын барлық мәндері үшін, бізге text[index:index + len(target)] пен target сәйкес келетіндігін тексеру керек. Осы кесінділерді қарастыру үшін, бізге for циклі қажет, ол text-тегі символдардың орнына text-тегі әрбір index -ті іріктейді.

Ойлануға берілетін сұрақ 6.38 Жолдағы барлық индекстердің тізімін қалай алуға болады?

text атты жолдағы индекстер тізімі - 0, 1, 2, ..., len(text) - 1. Бұл range(len(text)) арқылы берілетін нақты бүтін сандар тізімі. Сәйкесінше, бізге қажетті for циклі келесідей болады:

```
def count(text, target):
    """Count the number of target strings in text.

    Parameters:
        text: a string object
        target: a string object

    Return value: the number of occurrences of target in text
    """

    count = 0
    for index in range(len(text)):
        if text[index:index + len(target)] == target:
            count = count + 1
    return count
```

count келесі аргументтермен бірге шақырылғандағы жұмысын көрелік:⁴

```
result = count('Diligence is the mother of good luck.', 'the')
```

Егер біз циклді “аударсақ”, онда цкл денесінде орындалатын нұсқаулардың мынаған тең екенін байқаймыз:

```
if 'Dil' == 'the':      # text[0:3] салыстырылады -
    count = count + 1    # орындалмаған; count өні - 0
if 'ili' == 'the':      # text[1:4] салыстырылады -
    count = count + 1    # орындалмаған; count өні - 0
if 'lig' == 'the':      # text[2:5] салыстырылады -
    count = count + 1    # орындалмаған; count өні - 0
:
if ' th' == 'the':      # Text[12:15]           к 'the'
    count = count + 1    # орындалмаған;       ә — 0
if 'the' == 'the':      # Text[13:16]           к 'the'
    count = count + 1    # count өнді - 1
if 'he ' == 'the':      # Text[14:17]           к 'the'
    count = count + 1    # орындалмаған;       ә — 1
```

⁴“Diligence is the mother of good luck.” (Еңбек етсең ерінбей, тояды қарның тіленбей) Бенджамин Франклиннің Байлыққа апарар жол (1758) кітабынан

```

if 'oth' == 'the': # text[18:21]
    және 'the' count = count + 1
    салыстырлады # орындалмаған;
    count әлі 1-ге тең
if 'the' == 'the': #
    text[19:22] салыстырлады -
    'the' count = count + 1 # |
    count енді - 2

if 'k.' == 'the': # text[35:38]
    салыстырлады 'the'
    count = count + 1 # орындалмаған; count әлі
- 2 if '.' == 'the': # text[36:39]
    салыстырылады - 'the'
    count = count + 1 # орындалмаған; count әлі -
2 қайтарылады count # қайтарылады - 2

```

Байқағанымыздай, соңғы екі салыстыру ешқашан ақиқат бола алмайды, себебі text[35:38] пен text[36:39]-ға сәйкес келетін жолдар тым қысқа. Сондықтан, біз len(text) - len(target) кейін келетін кесіндіге көңіл аудармадық. Осы қажетсіз салыстыруларды алып тастау үшін, индексстердің диапазонын былай өзгертуге болады

range(len(text) - len(target) + 1)

Ойлануға берілетін сұрақ 6.39 Жолдағы соңғы символдан кейін басталатын жол кесіндісімен не қайтарылады (мыс., 'good'[4:8])? Соңғы символға дейін басталып, соңғы символдан тыс шығып кететін кесіндімен не қайтарылады (мыс., 'good'[2:10])?

Жолдағы индексстерді іріктеу символдарды іріктеуге ұқсас келеді. Мысалы, count1 функциясын былай жазуға болады:

```

def count1(text, target):
    """ (құжаттау жолдары түсірілген) """

    count = 0
    for index in
        range(len(text)): if
            text[index] == target:
                count = count + 1
    return count

```

Осы функцияның екі нұсқасын салыстырыңыз да, осы екі тәсілдің де циклдерінде бірдей етіп салыстыруды орындағанын түсінгеніңізді тексеріңіз.

Кейбір қосымшаларда жол индекстерін іріктеу қажетті болып табылады. Мысалы, жолдағы белгілі бір символдың алғашқы кірісінің индексін қайтару керек функцияны қарастырайық. Егер функция жолдағы символдарды іріктейтін болса, ол келесідей болады:

```
def find1(text, target):  
    """Мәтіндегі мақсаттың ірінші кірісінің индексін табу.
```

Параметрлер:

text: Іздеу орындалатын жол нысаны

target: Іздеуге арналған бір символды жол нысаны

Қайтарылатын мән: Мәтіндегі мақсаттың бірінші кірісінің индексі"""

```
    for character in text:  
        if character == target:  
            return ???          # символ индексін  
    қайтарады? return -1
```

```
def find1(text, target):  
    """ (құжаттау жолдары түсірілген) """  
  
    for index in  
        range(len(text)): if  
            text[index] == target:  
                return index      # text[index] return -1  
    индексін қайтарады
```

Енді біз `text[index] == target` екенін анықтағанда, бізге іздестірілген символымыздың `index` позициясында екенін бідеміз де, енді біз оны қайтара аламыз.

Ойлануға берілетін сұрақ 6.40 Функцияның соңындағы `return -1` мақсаты қандай? Ол қандай жағдайда орындалады? Келесі баламалы іске асыру неге дұрыс емес?

```
def find1BAD(text, target):
    """ (құжаттау жолдары түсірілген) """

    for index in
        range(len(text)): if
            text[index] == target:
                return index
    else:
        return -1
```

БҰЛ
ДҰРЫС ЕМЕС!

Алдымен return нұсқауларын дұрыс find1 функциясында қарастырып алайық. Егер text[index] == target екендігі for цикліндегі кейбір index мәндеріне ақиқат болса, онда find1 функциясы return index нұсқауымен тоқтатылады. Осы жағдайда, цикл ешқашан өзінің “табиғи” нәтижесіне жете алмайды да, ешқашан return -1 нұсқауына жетпейді. Сәйкесінше, егер text-те target-ке сәйкестік табылмаса, find1 функциясында return -1 нұсқауы ғана орындалады.

Дұрыс емес find1BAD функциясында return -1 орынсыз, себебі ол функцияның ылғи циклдың бірінші итерациясында қайтарырауын мәжбүрлейді! 0 - index-ке меншіктелген кезде, егер text[index] == target екендігі ақиқат болған жағдайда, онда 0 мәні қайтырылады. Кері жағдайда, -1 мәні қайтарылады. Кез келген жағдайда for циклінің келесі итерациясы ешқашан орындалмайды. Функция дұрыс жұмыс істейтіндей көрінеді: егер target text-тегі бірінші символға сәйкес келсе немесе егер target

text-те табылмаса, алайда, егер target text-тегі кейбір соңындағы кейбір символға сәйкес келетін болса, ол дұрыс болмайды. Функция кез келген кіріс мәні үшін жұмыс істемейтіндіктен ол түгелімен дұрыс емес болады.

Кесінділер арқылы count1 функциясын count-ке жалпылағанымыздай, біз find1-ді кез келген тармақшадағы бірінші кірісті іздейтін find функци-ясына жалпылай аламыз.

```
def find(text, target):
    """Мәтіндегі мақсаттың ірінші кірісінің индексін табу.
```

Параметрлер:

text: Target ізделетін жол

нысаны: Іздеуге арналған жол

нысаны

Қайтарылатын мән: Мәтіндегі мақсаттың бірінші кірісінің индексі ""

```
for index in range(len(text) - len(target)
    + 1): if text[index:index + len(target)]
    == target:
        return
index return -1
```

Келесі алгоритмнің find1 алгоритміне қаншалықты ұқсастығына назар аударыңыз, жәнн len(target) - 1-ге тең болғанда, find және find1 сәйкес болады. 6.1. Тараудағы wordCount5 функциясы тәрізді count1 және find1 функциялары сызықтық уақыт алгоритмдерін жүзеге асырады. Неліктен олай екенін білу үшін, find1 функциясын толығырақ зерттейік. Алдыңғы тарауларда істегеніміздей, n деп text кірісінің ұзындығын алайық. Екінші кіріс параметрі find1, target - бірліктік ұзындыққа ие. Сонда кірістің жалпы көлемі - n + 1 болады. find1 функциясында ең жиі кездесетін қарапайым амал - if нұсқауын for циклінің ішінде салыстыру болады. Функция target-ті text жолында тапқаннан кейін аяқталатындықтан, ең нашар жағдай (яғни, максималды салыстыру саны) target табылмаған кезде болады. Бұл жағдайда, text-тегі символдардың әрқайсысы үшін бір бірден n салыстыру орындалады. Қарапайым амалдар саны асимптоталық түрде кіріс мәліметтерінің көлеміне тең болғандықтан, find1 - сызықтық уақыт алгоритмі болып табылады. Осы себептен, find1 функциясында қолданылатын алгоритмдік тәсіл, сызықтық іздеу (немесе тізбектік іздеу) деп аталады. 8 тарауда, біз айтарлықтай жылдам орындалатын ұқсас іздеу алгоритмін көреміз, алайда оны тек қана мәліметтер реттеліп сақталған кезде ғана қолдануға болады.

Ойлануға берілетін сұрақ 6.41 *Жалпырақ келетін count және find функцияларының уақыт күрделеліліктері қандай? Мәселен, text n ұзындықтан ал target m ұзындықтан тұрсын делік.. Бұлар да сызықтық уақыт алгоритмдері болып табыла ма?*

Конкорданс

Енді бізде мәтіндегі тармақшаларды санайтын және іздейтін функциялар болған кезде, біз оларды кез келген мәтінге, тіпті өте ұзын мәтіндік файлдарға қолдана аламыз. Мысалы, біз Моби Дик кітабындағы кез келген сөздің бірінші кірісін табу үшін интерактивті бағдарлама жаза аламыз:

```
def main():
    textFile = open('mobydick.txt', 'r', encoding =
'utf-8') text = textFile.read()
    textFile.close()

    word = input('іздей: ')
    while word != 'q':
        index = find(text, word)
        print(word, 'позицияда бірінші рет
шығады', index) word = input('іздей: ')

```

Алайда, find функциялық абстаркция ретінде қолдану арқылы, біз мәтіндік файл туралы анағұрлым пайдалы мәлімет жинақтау үшін алгоритмдер жасай аламыз. Оның бір мысалы - конкорданс болып табылады, бұл мәтінді бар мәтіндегі барлық сөздердің алфавиттік тізімі болып табылады, онда олардың әрқайсысы шығады. Вильям Шекспирдің шығармаларының конкордансын келесі сайттан табуға болады <http://www.opensourceshakespeare.org/concordance/>.

Ең қарапайым конкорданс мәтіндегі сөздердің әрқайсысынан тұратын барлық жолдарды жай ғана санап шығады. Біз жеке алынған сөздің конкорданстық жазбасын мәтіндік файлдағы әр жолды find функциясын қолдану арқылы іріктей отырып, жолда берілген сөздің бар-жоғын анықтаймыз. Егер болса, біз жолды экранға шығарамыз.

Ойлануға берілетін сұрақ 6.42 *Егер біз find функциясын сөзді іздеу үшін шақыратын болсақ, оның табылғандығын қалай білеміз?*

Келесі функция бір конкорданстық жазбаның экранға шығару алгоритмін жүзеге асырады:

```
def concordanceEntry(fileName, word):
    """Берілген сөзден тұратын файлдың барлық
жолдарын экранға шығарады.

```

Параметрлер:

fileName: Word жолы түріндегі мәтіндік
файлдың аты: Іздестіретін сөздер

```
Қайтарылатын  
мәндер: None""
```

```
textFile = open(fileName, 'r', encoding =  
'utf-8') for line in textFile:  
    found = find(line, word)  
    if found >= 0: #  
        print(line.rstrip()) жолында  
        сөзді іздеу  
textFile.close()
```

Біз Моби Дик кітабындағы мәтінге “lashed” сөзін іздейтін concordanceEntry функциясын шақыратын болсақ, онда 14 сәйкестік аламыз, олардың алғашқы 6-уы:

things not properly belonging to the room, there was a hammock lashed
blow her homeward; seeks all the lashed sea’s landlessness again; sailed
with. How he flashed at me!--his eyes like powder-pans! is he I was so
taken all aback with his brow, somehow. It flashed like a with storm-
lashed guns, on which the sea-salt cakes!

to the main-top and firmly lashed to the lower mast-head, the strongest

Әрбір жолда “lashed” шығатын жерін сөздерді келесідей етіп тегістей алатын болсақ оңайлықпен көретін едік:

... belonging to the room, there was a hammock lashed
blow her homeward; seeks all the lashed sea’s ... sailed with. How he
flashed at me!...

... all aback with his brow, somehow. It flashed like a
with storm-lashed guns, ... to the main-top and firmly lashed to the ...

Ойлануға берілетін сұрақ 6.43 Мәтіндік файлдағы әрбір жол тек 80 символдық ұзындықтан ғана тұрады делік. Мақсатты сөз бойынша тегістеуді орындау үшін экранға әр жолдан бұрын қанша интервал шығару керек? (Сыбырсөз: found мәнін қолданыңыз.)

Ескертпе 6.2: Табиғи тілді өңдеу

Табиғи тілді өңдеу саласындағы зерттеушілер тек қана мәтінді іздестіру мен ұйымдастыруды орындаудан бөлек, оның жазбаша және ауызша түрлерінде “түсіне” алатын және жауап бере алатын алгоритмдерді жасап шығаруға ұмтылады. Мысалы, Google Translate (<http://translate.google.com>) шынайы уақытта бір тілден екінші тілге автоматты аударма жасайды. Коммерциялық веб-сайттарда кең таралған “виртуалды көмекшілер” Сіздің сұрақтарыңызды түсініп, оларға пайдалы жауап беруге талпынады. Озат жүйелер Ғаламторда және басқа жерлерде берілген құрылымсызданған мәліметтердің үлкен көлемдерінен түсінік алуға талпынады. Егер бұл тапсырмалар Сізге қызық болса, онда мына сайттан <http://www.nltk.org> *Natural Language Toolkit (NLTK)* туралы, табиғи тілді өңдеу құралдарын қамтамасыз ететін Python модулі туралы біле аласыз. Сәйкес кітаптар мына сайтта берілген <http://nltk.org/book>.

Сөз табылған әрбір жолдың found-ке меншіктелген индекс пен табылғандығын білеміз. Осылайша, ол жолда сөзден бұрын found символдары тұрады. Біз 80 позициядағы мақсатты сөздердің жалғауларын әр жолдың алдына (80 - len(word) - found) бос жолдарды қойып туралай аламыз, ол үшін print орнына мынаны шақырамыз:

```
space = ' ' * (80 - len(word) - found) print(space + line.rstrip())
```

Нәтижесінде алынған үзіктердің мәтіннің қай бөлігінде тұратынын білмесе қажеттігі болмайды. Сондықтан біз экранға шығаратын мәтіннің әрбір жолына жол нөмірін қосуымыз керек. 6.2. Тараудағы lineNumber функциясындағыдай, оны сумматорды қосу арқылы орындаймыз, ол жолды оқыған сайын артылады. Жолды экранға шығару кезінде біз ені 6-ға тең болатын өрісте жол нөмірін жасаймыз да, алдыңғы тураланған қалпын сақтаймыз.

```
def concordanceEntry(fileName,
    word): """ (құжаттау жолдары
    түсірілген) """

    text = open(fileName, 'r', encoding = 'utf-8')
    lineCount = 1
    for line in text:
        found = find(line, word)
        if found >= 0: # space = '
            ' * (80 - len(word) -
                found) жолында сөз
```

```

іздеу
print('{0:<6}'.format(lineCount) + space +
line.rstrip()) lineCount = lineCount + 1
text.close()

```

6.7 тарауда осы алгоритмдік әдістерді геномика есептеріне, тірі жасушалардағы ДНҚ қызметі мен құрылымын зерттейтін биология ғылымында қолдануға болатындығы көрсетілген.

Жаттығулар

6.5.1. Келесі for цклдерінің әрқайсысына text жолдарының индекстерін символдардың орнына іріктейтін ұқсас цикл жазыңыз.

(a) for character in text: print(character)

(b) newText = ""

for character in text: if character != ' ':

newText = newText + character

(c) for character in text[2:10]:

if character >= 'a' and character <= 'z': print(character)

(d) for character in text[1:-1]: print(text.count(character))

6.5.2. Келесі код блоктарының әрқайсысындағы синтаксис-те ненің қате екенін түсіндіріңіз де, оны қалай түзетуге болатынын көрсетіңіз. Мәселен, text-ке бұрынырақ жолдық мән меншіктелді делік.

(a) for character in text:

caps = caps + character.upper()

(b) while answer != 'q':

answer = input('Word? ') print(len(answer))

(c) for index in range(text): if text[index] != ' ':

print(index)

6.5.3. Функцияны жазыңыз

prefixes(word)

ол берілген сөздің барлық префикстерін экранға шығарады. Мысалы, prefixes('cart')

экранға мынаны шығару керек

c sa car

cart

6.5.4. find функциясын бүтін символдар болып табылатын target -тің кірістерін ғана іздейтіндей етіп өзгертіңіз.

6.5.5. Келесі әдістердің әрқайсысымен concordanceEntry функциясын оңтайландырыңыз:

(a) Функцияны мақсатты сөздің кірістерін олардың тіркелімге сәйкестілігіне тәуелсіз белгілейтіндей етіп өзгертiңiз. Мысалы:

MOBY DICK; OR THE WHALE

The pale Usher--threadbare in coat ...

... embellished with all the gay flags of all

... in hand to school others, and to teach them by what

(b) Әрбір сәйкестілік үшін экранға шығарылған жолда word-ті бас әріптермен көрсетіңіз. Мысалы:

... any whale could so SMITE his stout sloop-of-war

... vessel, so as to SMITE down some of the spars and

(c) 6.5.4 жаттығуындағы өзгертілген find функциясын әрбір мақсатты сөз тек қана бүтін сөзге сәйкес келетіндей етіп қолданыңыз.

(d) concordanceEntry ағымдағы нұсқасы әрбір жолдағы сөздің бірінші кірісін ғана анықтайтын болады. Оны мәтіннің әр жолындағы мақсатты сөздің әр кірісі үшін жаңа мәнмәтін жолын көрсететіндей етіп өзгертiңiз. Мысалы, “ship” - “upon the ship, than to rejoice that the ship had so victoriously gained,” жолында екі рет шығады, сондықтан функция экранға келесіні шығару керек:

upon the SHIP, than to ... upon the ship, than to rejoice that the SHIP had so ...

Сіз count және find функцияларының өзгертілген нұсқаларын жасап шығара аласыз.

6.5.6. Функцияны жазыңыз

concordance(dictFileName, textFileName)

ол файл атауы белгіленген сөздік пен concordanceEntry функциясын экранға файл атауы белгіленген мәтінге арналған толық конкорданс шығару үшін қолданады. (Сөздік файлын алу үшін 6.2.4 жаттығудағы мәліметті қараңыз.) Функция мәтіндегі әрбір сөзге алфавиттік тізімде мәнмәтін келтіруі керек.

Бұл екі себеппен конкордасн құрудың ең тиімсіз әдісі болып табылады. Біріншіден, мәтіннен ғана сөздер іздеу, сөздіктен әр сөзді іздегенге қарағанда өте тиімді болар еді. Екіншіден, әрбір жаңа сөз үшін мәтіндік файлды басынан соңына дейін тексеру өте тиімсіз. (Осы себептер үшін, экранға толық конкордансты шығаруға тырыспаңыз. Сөздіктен оқитын сөздердің санын шектеңіз.) Бұл мәселеге 11 тарауда қайта келеміз, және тиімдірек алгоритм жасаймыз.

6.5.7. Бұл мысалда, Сіз 6.2. Тараудағы printProducts функциясін белгілі бір жылы белгілі бір себептермен қайтарылған тауарлар саны іздейтіндей етіп толықтырасыз.

(a) Функцияны жазыңыз

findReason(product)

Ол қайтару жылы мен берілген тауарды қайтару элементімен сипатталған қайтару себебінен тұратын жолды қайтарады. Мысалы, егер product жолға тең болғанда

```
'<PRODUCT>
<DATE>Mon, 19 Oct 2009 00:00:00 -0400</DATE>
<BRAND_NAME><![CDATA[Good food]]></BRAND_NAME>
<PRODUCT_DESCRIPTION><![CDATA[Cake]]></PRODUCT_
DESCRIPTION>
<REASON><![CDATA[Allergen]]></REASON>
<COMPANY><![CDATA[Good food Inc.]]></COMPANY>
<COMPANY_RELEASE_LINK>      ...   </COMPANY_RELEASE_
LINK>
<PHOTOS_LINK></PHOTOS_LINK>
</PRODUCT>'
```

сосын функция '2009 Allergen' жолын қайтару керек.

(b) 6.2. Тараудағы printProducts функциясын жаңа функция жазу үшін өзгертіңіз

recalls(reason, year)

ол берілген себеппен берілген жылы жарияланған қайтару элементтерінің санын қайтарады. Ол үшін жаңа функция әрбір қайтарылған тауар элементін экранға шығарудың орнына қайтарылған тауар элементінен тұратын жолды жасауы керек (жоғарда көрсетілгендей), сосын бұл жолды findReason функциясына жіберу керек. (print(line.rstrip()) нұсқауының әрқайсысын жинақтаушы жол нұсқауымен алмастырып, тауарларды нөмірлейтін нұсқауларды жойыңыз. Басқа нәрсені жоймаңыз.) findReason арқылы қайтарылатын әрбір нәтижемен есептеуіш параметрлер ретінде жіберілген себеп reason және жылдан year тұрса, оны арттырыңыз.

6.6 МӘТІНДЕРДІ САЛЫСТЫРУ

Мәтіндер арасындағы ұқсастықты талдау үшін көптеген әдістер жасалған, олардың көбісі бұл кітаппен қамтылмаған. Алайда бір ерекше әдіс бар, оны нүктелі диаграммалар әдісі дейміз, ол қолжетімді әрі өте мықты әдістің бірі болып табылады. Нүктелі диаграммада біз бір

мәтінді x-осімен, ал екінші мәтінді y-осімен байланыстырамыз. Егер бірінші мәтіндегі x индексті символ екінші мәтіндегі y индексті символмен сәйкес келетін болса, біз (x, y) координаталарына нүкте қоямыз. Осылайша, нүктелі диаграмма осы екі мәтін арасындағы ұқсастықты суреттеп көрсетеді.

Екі мәтіндегі бірдей индексті символдар сәйкес келген кезде ғана нүктелер қоятын алгоритмді жазудан бастайық. Келесі екі сөйлемді қарастырайық:

Мәтін 1: Peter Piper picked a peck of pickled peppers.

Мәтін 2: Peter Pepper picked a peck of pickled capers.

Біздің бастапқы алгоритміміз 1-мәтігіндегі P мен 2-мәтіндегі P-ді салыстырады, сосын 1-мәтіндегі e мен 2-мәтіндегі e-ді салыстырады, сосын

1-мәтіндегі t мен 2-мәтіндегі t -ді, т.с.с. салыстырып кете береді. Төменде келтірілген бұл алгоритм бір уақытта екі жолды іріктеу керек, және жолдың әрқайсысын өз позициясында салыстырады, бұған тек бір цикл ғана керек, себебі біз жолдарды ылғи бір ғана индекс бойынша салыстырамыз.

```
import matplotlib.pyplot as pyplot
```

```
def dotplot1(text1, text2):
```

```
    """бірдей ұзындықтағы екі жолды салыстырудың  
    оңайлатылған нүктелі диаграммасын көрсетеді.
```

Параметрлер:

text1: Жол нысаны

text2: Жол

нысаны

Қайтарылатын

мәндер: None"""

```
text1 =  
text1.lower() text2
```

```

= text2.lower() x
= []
y = []
for index in range(len(text1)):
    if text1[index] ==
        text2[index]:
        x.append(index)
        y.append(index)

pyplot.scatter(x, y)      # шашу диаграммасы
pyplot.xlim(0, len(text1)) # x осі text1
pyplot.ylim(0, len(text2)) толықтай қамтиды
                        # y осі text2
pyplot.xlabel(text1) толықтай қамтиды
pyplot.ylabel(text)
pyplot.show()

```

Ойлануға берілетін сұрақ 6.44 *lower әдісінің шақырылу мақсаты қандай?*

Ойлануға берілетін сұрақ 6.45 *Неліктен біз жолдағы символдарды іріктемей, жол индекстерін іріктеу керек?*

Циклдегі екі символ тең келген сайын, сәйкес келген символдардың индексі x-координатасы мен y-координатасының тізімдеріне қосылады. Осы тізімдер графикке scatter функциясының көмегімен matplotlib-тен салынады. scatter функциясы графикке тек нүктелерді оларды байланыстыратын сызықсыз құрады. 6.5 суретте жоғардағы екі тізбектің нәтижелері көрсетілген.

Ойлануға берілетін сұрақ 6.46 *6.5 суретке қараңыз. Бұл символдарға қай нүктелер сәйкес келеді? Неліктен барлық нүктелер диагональде орналасқан?*

Бұл функция тек бірдей индекстерге ғана сәйкестіктерді анықтайтыны

байқалады, ал көптеген бірдей символдар дәл бір сызық бойында орналаспайтындықтан, бұл функция олардың арасындағы ұқсастықтың шынайы дәрежесін ашпайды. Алайда егер біз жолдарға жай ғана екі бос орын қоятын болсақ, символды символмен салыстыру басқа нәтиже беретін еді:

Сурет 6.5 dotplot1 функциясының нәтижесі.

MƏTİN 1: Peter Piper picked a peck of pickled peppers.

MƏTİN 2: P e t e r P e p p e r p i c k e d a p e c k o f p i c k l e d c a p e r s.

Шынайы нүктелі диаграмма бір тізбектегі символдық әрқайсысын келесі жолдағы символдың әрқайсысымен салыстырады. Бұл біздің `text1[0]` мен `text2[0]`, сосын `text1[0]` мен `text2[1]`, содан кейін `text1[0]` мен `text2[2]`, т.с.с. салыстырғымыз келетінін білдіреді, бұл төменде көрсетілген:

text1[0] -ді text2-ге барлық символдармен салыстырған соң, бізге бұл үдерісті text1[1] үшін text1[1] пен text2[0], және text1[1] пен text2[1], одан кейін text1[1] пен text2[2], және т.т. салыстырып қайталау керек, бұл төменде көрсетілген:

```

0 1 2 3 4 5 6 7 8 9 ...
text1: Peter Piper picked a peck of pickled peppers.
      ...
text2: Peter Pepper picked a peck of pickled capers.
0 1 2 3 4 5 6 7 8 9 ...

```

Басқа сөзбен айтқанда, `index`-тің әр мәні үшін біз `text1[index]`-ті тек `text2[index]`-пен ғана емес, сонымен бірге `text2`-дегі барлық позициямен салыстыруды қалаймыз. Бұл үшін, бізге `if` нұсқауын `dotplot1`-де басқа бір `for` циклімен алмастыру керек:

```

import matplotlib.pyplot as pyplot

def dotplot(text1, text2):
    """Екі жолды салыстыратын нүктелі диаграмманы
    көрсетеді.

    Параметрлер:
        text1: Жол нысаны
        text2: Жол нысаны

    Қайтарылатын
    мәндер: None"""

    text1 = text1.lower()
    text2 = text2.lower()
    x = []
    y = []
    for index in range(len(text1)):
        for index2 in range(len(text2)):
            if text1[index] == text2[index2]:
                x.append(index)
                y.append(index2)
    pyplot.scatter(x, y)
    pyplot.xlim(0, len(text1))
    pyplot.ylim(0, len(text2))

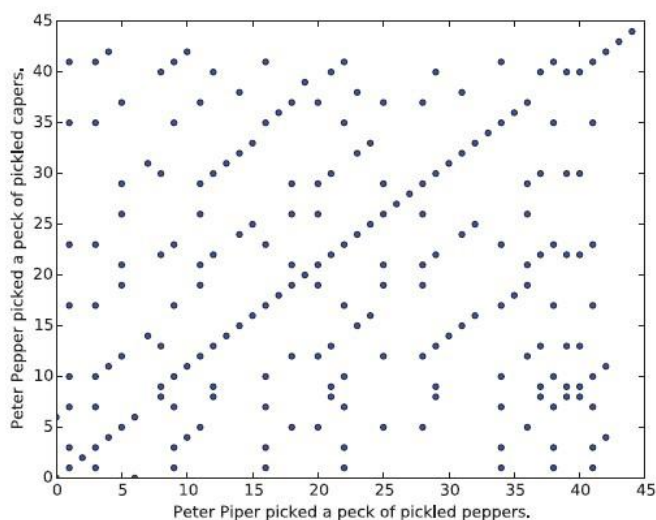
```

```
pyplot.xlabel(text1)
pyplot.ylabel(text2)
pyplot.show()
```

Осындай өзгеріспен бірінші for циклінің ішінде `text1[index]`-тің барлық символы `index2` индекстік айнымалысымен көрсетілетін `text2`-ге барлық символмен салыстырылады, бұл жоғарыда көрсетілген суреттерге ұқсайды. Сәйкестілік табылса, біз `index` -ті `x` тізіміне, ал `index2`-ді `y` тізіміне қосамыз.

Ойлануға берілетін сұрақ 6.47 мәселен Сіз *'plum'* сөзін *text1*-ге және *'pea'* сөзін *text2*-ге жібердіңіз делік. *for* циклінің денесінде орындалатын салыстырудың тізбегін жазыңыз.. Қаниша салыстыру амалы шықты?

Барлығы $4 \cdot 3 = 12$ салыстыру амалы шығады, себебі *'plum'* сөзіндегі символдардың төртеуінің бәрі



Сурет 6.6 Қайта қарастырылған `dotplot` функциясының нәтижесі.

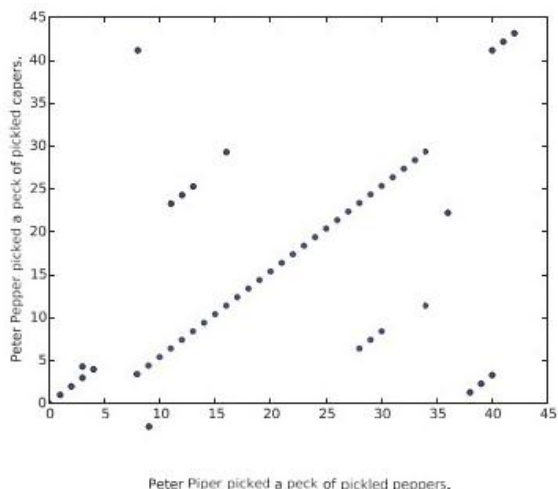
'Pea' сөзіндегі үш символдың әрқайсысымен салыстырылады. Бұл он екі салыстыру амалының реті келесідей болады:

1. text1[0] == text2[0]	немесе 'p' == 'p'
2. text1[0] == text2[1]	немесе 'p' == 'e'
3. text1[0] == text2[2]	немесе 'p' == 'a'
4. text1[1] == text2[0]	немесе 'l' == 'p'
5. text1[1] == text2[1]	немесе 'l' == 'e'
6. text1[1] == text2[2]	немесе 'l' == 'a'
7. text1[2] == text2[0]	немесе 'u' == 'p'
8. text1[2] == text2[1]	немесе 'u' == 'e'
9. text1[2] == text2[2]	немесе 'u' == 'a'
10. text1[3] == text2[0]	немесе 'm' == 'p'
11. text1[3] == text2[1]	немесе 'm' == 'e'
12. text1[3] == text2[2]	немесе 'm' == 'a'

Назар аударыңыз, index - 0-ге тең болған кезде, ішкі for циклі index2-дің барлық мәндері арқылы өтеді. Содан кейін ішкі цикл аяқталады да, сыртқы for циклін де тоқтатады, ал index - 1-ге артылады. Ішкі цикл сосын оның барлық мәндері арқылы қайта өтеді, т.с.с.

6.6 суретте қайта қаралған функцияның нұсқасының графикі көрсетілген. Бұл екі жолдың көптеген символдары болғандықтан, оларда бірнеше сәйкестік бар, олар графикті “шулы” етеді. Алайда график енді жолдардағы бас диагональдің бойында нүктелермен көрсетілген сәйкестіктерді аулайды.

Біз нүктелі диаграммадағы “шуды” жеке символдардың орнына тармақшаларды



Сурет 6.7 6.6.5 жаттығуындағы dotplot функциясының нәтижесі (3-грамм).

салыстыру арқылы азайтуға болады. Мәтіндік талдау қосымшаларындағы n ұзындықтағы тармақшалар n -грамдар деп аталады. $n > 1$ болғанда, ықтимал тармақшалар неғұрлым көп болса, сәйкестіліктердің болу үрдісі соғұрлым аз болады. 6.6.5 жаттығуы Сізден нүктелі диаграмма функциясын барлық жеке символдардың орнына екі мәтіндегі барлық n -грамдарды салыстыратындай етіп жалпылауды сұрайды. 6.7 суретте осы функцияның $n = 3$ болғандағы нәтижесі көрсетілген.

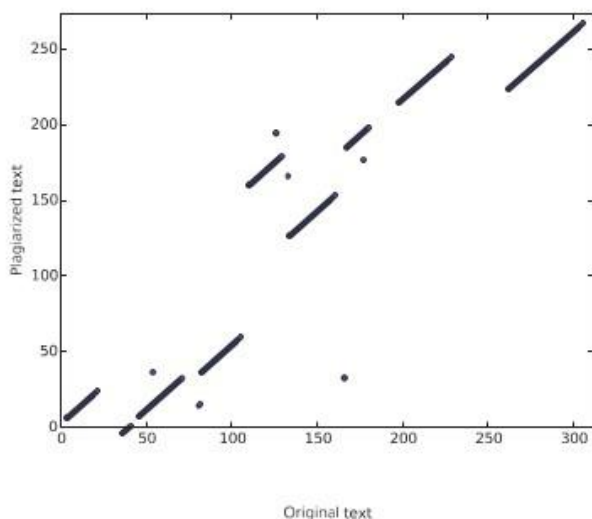
Нүктелі диаграмма плагиатты анықтауда қолданыс тапты. Мысалы, 6.8 суретте келесі екі үзіндінің нүктелі диаграмма әдісімен салыстырылған түрі көрсетілген. Біріншісі Эрик Хабсбаумның Мұқтаждық дәуірі: Дүниежүзілік тарих, 1914–1991 кітабының 23 бетінен алынған [19]:

All this changed in 1914. The First World War involved all major powers and indeed all European states except Spain, the Netherlands, the three Scandinavian countries and Switzerland. What is more, troops from the world overseas were, often for the first time, sent to fight and work outside their own regions.

Екіншісі Хабсбаумның үзіндісінің ойдан шығарып плагиатын жасау әрекеті:

All of this changed in 1914. World War I involved all major powers and all European states except the three Scandinavian countries, Spain, the Netherlands, and Switzerland. In addition, troops from the world overseas were sent to fight and work outside their own regions.

Ойлануға берілетін сұрақ 6.48 6.8 суретіне жай қарап отырып, Сіз үзіндінің плагиат екенін байқай алар ма едіңіз? (Екі кездейсоқ үзіндіні нүктелі дирамма әдісі арқылы салыстыру қалай болатын еді.)



Сурет 6.8 Түпнұсқа мен плагиат үзінділерінен нүктелі диграмма әдісімен 6-грам салыстыру.

Жаттығулар

6.6.1. Келесі кірістірілген цикл арқылы экранға не шығарылады?

```
text = 'imho'
for index1 in range(len(text)):
    for index2 in range(index1, len(text)): print(text[index1:index2 + 1])
```

6.6.2. Функцияны жазыңыз

difference(word1, word2)

ол word1 және word2 жолдары ұқсамайтын бірінші индексті қайтарады. Егер сөздердің ұзындығы әртүрлі болса, ал қысқарақ сөз ұзынырақ сөзді префиксі болса, функция қысқарақ сөздің ұқзындығын қайтару керек. Егер сөздің екеуі де

бірдей болса, функция -1 қайтарады. Осыны

word1 және word2 теңдігін тікелей салыстырмай істеңіз.

6.6.3. Екі биттық жолдар арасындағы биттардың айрықша позицияларының саны ретінде анықталатын Хэмминг қашықтығы желі арқылы мәліметті жіберу кезінде шығатын қатені өлшеу үшін

қолданылады. Мысалы, біз желі арқылы 011100110001 бит тізбегін жібердік делік, бірақ оның орнына алушы 011000110101 қабылдады. Жіберілу қателігін өлшеу үшін, біз екі тізбектің арасындағы Хемминг қашықтығын оларды келесідей туралау арқылы анықтай аламыз:

Жіберілді: 011100110001

Қабылданды: 011000110101

Бит тізбектері екі позицияда әр түрлі болғандықтан, Хемминг қашықтығы 2-ге тең.

Функцияны жазыңыз

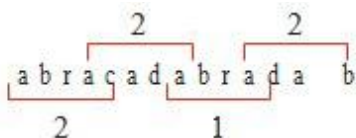
hamming(bits1, bits2)

ол берілген биттық тізбектер екеуінің арасындағы Хемминг қашықтығын қайтарады. Мұндағы екі жолдың да ұзындығы бірдей делік.

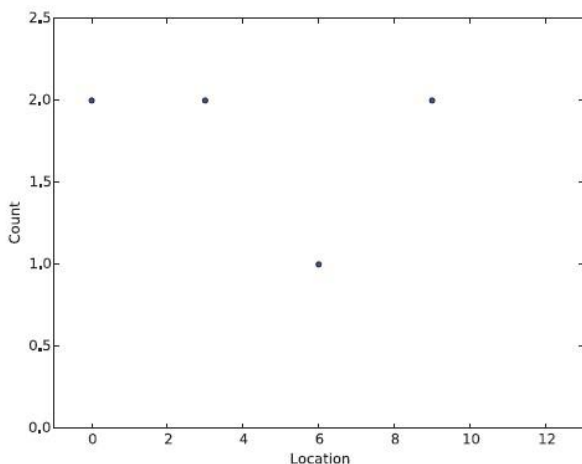
6.6.4. 6.6.3 жаттығуын қайталаңыз, бірақ функцияның екі жол ұзындығы әртүрлі болса да дұрыс жұмыс істеуін қамтамасыз етіңіз. Бұл жағдайда, қысқарақ жолдың соңындағы “жетіспейтін” биттың әрқайсысы Хемминг қашықтығына бірлік қосады. Мысалы, hamming('000', '10011') - 3 қайтару керек.

6.6.5. dotplot функциясын жеке символдар орнына n-грамдарды салыстыратындай етіп жалпылаңыз.

6.6.6. Мәтін бойында біршама ұзындықтағы “терезелерде” сөздің шығу жиілігін бақылай отырып, мәтінді тереңірек зерттеуге болатын еді. Төмендегі өте шағын мысалды қарайық, онда біз 'abracadabradab' жолындағы 4 өлшемдік терелердегі “сөздің” шығу жиілігін санаймыз:



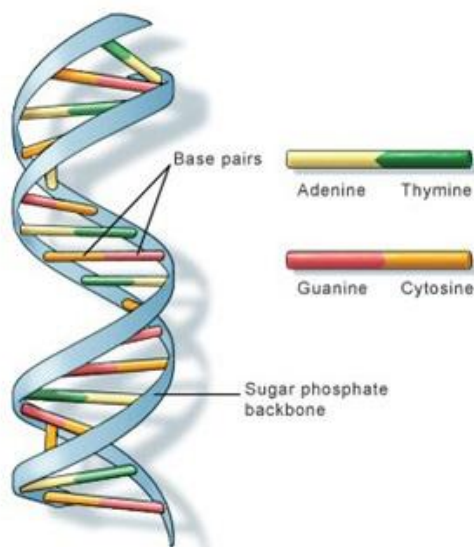
Бұл мысалда, терезе әр кезде 3 символға ілгерілейді. Сонда осы төрт терезедегі шығу жиілігі - 2, 2, 1, 2, оны графикте төмендегідей көрсетуге болады:



Бұл мысалда, терезе әр кезде 3 символға ілгерілейді. Сонда осы төрт терезедегі шығу жиілігі - 2, 2, 1, 2, оны графикте төмендегідей көрсетуге болады:

X-осіндегі-осіндегі сандар әр терезенің бастапқы позицияларының индекстері болып табылады. Функцияны жазыңыз

`wordFrequency(fileName, word, windowSize, skip)`



<http://ghr.nlm.nih.gov/handbook/basics/dna>

Сурет 6.9 ДНҚ-ның қос шиыршығының кескіні.

ол осыған ұқсайтын графикті көрсетеді, онда word жолының windowSize өлшемді терезеде қайталану жиілігі берілген, ондағы терезелер әр кезде skip индекске ілгерілейді. Талданатын мәтін белгіленген fileName атты файлдан оқылуы керек.

*6.7 ГЕНОМИКА

Кез келген тірі жасуша ДНҚ (дезоксирибонуклеин қышқылы) молекулаларынан тұрады, ол ағзаның даму және қызмет ету генетикалық бағдарламасын кодтайды. ДНҚ құрамындағы мәліметті ашу, және оның барлық тіршілік үдерістерінде қолданылу жолдарын түсіну - қазіргі таңдағы медицина мен биология ғылымындағы ең маңызды әрі ең күрделі тапсырмаларының бірі болып табылады. Өртүрлі ағзалардың ДНҚ салыстыру сондай ақ эволюция мен тіршілік ағашы жөнінде түсінік береді. ДНҚ молекулаларының ұзындығы және тірі жасушалар ДНҚ-нан мәтіндік файлдарға оқылған мәліметтер көлемінің өзі биология ғалымдарынан бұл күрделі мәселені шешу үшін есептеу әдістерін қолдануды талап етеді.

Геномика ежелері

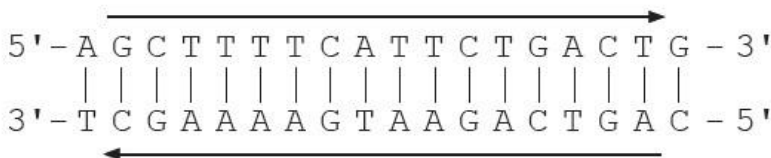
6.9 суретінде көрсетілгендей, ДНҚ - дегеніміз қос шиыршықты жіпшелер пішіндегі ұзын қос тізбекті молекула болып табылады. Әрбір қабат - нуклеотид деп аталатын шағын бірліктер тізбегі болып табылады. *Нуклеотид* қант (дезоксирибоза) фосфат тобынан және төрт азотты негіздердің бірінен тұрады: Аденин (А), тимин (Т), цитозин (С), немесе гуанин (G). Молекуладағы әрбір нуклеотид құрамындағы негізіне сәйкес әріппен белгіленуі мүмкін, ал қабаттың өзі - нуклеотидтерінің тізбегіне сәйкес келетін символдар тармағы ретінде беріледі. Мысалы, келесі жол аденин нуклеотиді соңынан гуанин нуклеотиді соңынан цитозин нуклеотидінен, және т.с.с. тізбектен тұратын ДНҚ молекуласын ұсынады.:

```
>>> dna = 'agcttttcattctgactg'
```

(Символдардың регистрі маңызды емес; кейбір тізбек жиынтығы

төменгі регистрді, ал кейбіреуі - жоғарғы регистрді қолданады.) Шынайы ДНҚ тізбектері үлкен мәтіндік файлдарға сақталады; біз оларды кейінірек қайта қарастырамыз.

Бір жіпшенің негізі екінші қабаттың комплиментарлы негізімен сутекті байланыс арқылы бірігеді. С және G, А және Т секілді комплиментарлы болып табылады. Бір жіпше негізі екіншісімен комплиментарлы байланысуы негіз жұбы деп аталады. Екі жіпше “антипараллель” құрайды, бұл ДНҚ көшіретін және оқитын жасушалы механизмнің қарама-қарсы бағыттарында қозғалатынын білдіреді. Әрбір жіпшеде, ДНҚ “жоғарыдан-төменгі” бағытта оқылады, алайда жіпшелердің “жоғарғы” және “төменгі” ұштары бір-біріне қарама-қайшы. Мұнда маңызы жоқ себептер үшін, “жоғарғы” ұшы 5’ (“бес штрих” деп оқылады) ал төменгі ұшы 3’ (“үш штрих”) деп аталады. Мысалы, төменде берілген 5’ -тен 3’ -ке дейінгі бағытта оқылатын жоғарғы тізбек - AGCTT...CTG болып табылады, ал 5’ -тен 3’-ке дейінгі бағытта оқылатын *кері комплиментарлы* деп аталатын төменгі тізбек - CAGTC...GCT болып табылады.



Ойлануға берілетін сұрақ 6.49 ДНҚ тізбектерін мәтіндік файлдарда сақтаған кезде тек бір жіпшенің тізбегін сақтайды. Неліктен олай?

РНҚ (рибонуклеин қышқылы) дегеніміз - ұқсас молекула, алайда әрбір нуклеотиді басқа қанттан (дезоксирибозаның орнына рибоза) тұрады, ал тиминнің (Т) орнында урацил (U) деп аталатын негіз орналасқан. Сондай ақ РНҚ молекулалары бір тізбектен тұрады. Нәтижесінде, РНҚ бір тізбектің комплиментарлы негіздері бір бірімен шағылысқан сәтте “иілуге” бейім келеді. Осы жиылған құрылым бірегей үштік құрылым түзеді, ол молекуланың функциясында маңызды рөл атқарады.

ДНҚ молекулаларының кейбір бөліктері *ген* деп аталады, себебі олар генетикалық мәліметті кодтайды. *РНҚ-полимераза* деп аталатын фермент РНҚ комплиментарлы молекуласын синтездеу үшін генді *транскрипциялайды*. Мысалы, 5'-GACTGAT-3' ДНҚ тізбегінің транскрипциясының нәтижесінде 3'-CUGACUA-5' РНҚ тізбегі синтезделуші еді. Егер РНҚ - мәліметтік (матрицалық) РНҚ (мРНҚ) болатын болса, онда ол *кодтаушы бөліктен* тұрады, оны кейін ақуызды

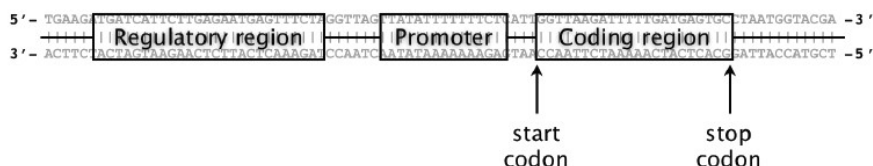
синтездеу үшін қолдануға болады. Басқа РНҚ-ның транскрипция өнімдері *РНҚ гендері* деп аталады, олар ақуызға айналмайды, оның орнына олар гендерді реттеуге қатысады, геннің ақуызға транскрипциалану немесе айналуы керектігін анықтайды. Тізбек транскрипциалануды

	U			C			A			G			
U	UUU	Phe	F	UCU	Ser	S	UAU	Tyr	Y	UGU	Cys	C	U
	UUC	Phe	F	UCC	Ser	S	UAC	Tyr	Y	UGC	Cys	C	C
	UUA	Leu	L	UCA	Ser	S	UAA	Stop	*	UGA	Stop	*	A
	UUG	Leu	L	UCG	Ser	S	UAG	Stop	*	UGG	Trp	W	G
C	CUU	Leu	L	CCU	Pro	P	CAU	His	H	CGU	Arg	R	U
	CUC	Leu	L	CCC	Pro	P	CAC	His	H	CGC	Arg	R	C
	CUA	Leu	L	CCA	Pro	P	CAA	Gln	Q	CGA	Arg	R	A
	CUG	Leu	L	CCG	Pro	P	CAG	Gln	Q	CGG	Arg	R	G
A	AUU	Ile	I	ACU	Thr	T	AAU	Asn	N	AGU	Ser	S	U
	AUC	Ile	I	ACC	Thr	T	AAC	Asn	N	AGC	Ser	S	C
	AUA	Ile	I	ACA	Thr	T	AAA	Lys	K	AGA	Arg	R	A
	AUG	Met	M	ACG	Thr	T	AAG	Lys	K	AGG	Arg	R	G
G	GUU	Val	V	GCU	Ala	A	GAU	Asp	D	GGU	Gly	G	U
	GUC	Val	V	GCC	Ala	A	GAC	Asp	D	GGC	Gly	G	C
	GUA	Val	V	GCA	Ala	A	GAA	Glu	E	GGA	Gly	G	A
	GUG	Val	V	GCG	Ala	A	GAG	Glu	E	GGG	Gly	G	G

Кесте 6.1 Кодондар мен амин қышқылдары арасында тасымалданатын стандартты генетикалық код. Әр кодон ш әріптік кодпен, ал сәйкес амин қышқылы бір әріптік кодпен берілген.

промоторлы бөліктен бастайды, ол РНҚ-полмеразаны транскрипцияны бастатқызады. Оның алдында әдетте реттеуші бөлік тұрады, ол геннің транскрипциалануына немесе транскрипциаланбауына әсер етеді.

Транскрипция бақыты ("downstream")



мРНҚ-ның кодтайтын бөлігі *рибосома* деп аталатын молекулярлық механизм арқылы ақуызға *трансляцияланады*. Рибосома кодтайтын аймақты *кодон* деп аталатын үш нуклеотидтен тұратын топтарға есептейді. Прокариоттарда (мыс., бактериялар), кодтайтын бөлік нақты *бастаушы кодонмен* басталады да, және үш *терминациялаушы кодонның* бірімен аяқталады. Рибосома бастаушы және терминациялаушы кодондардың арасындағы мРНҚ кодонының әрқайсысын *генетикалық кодқа* сәйкес амин қышқылдарға трансляциялайды, бұл 6.1 кестеде көрсетілген. Ақырғы ақуыз жәл осы амин қышқылдарының тізбегінен тұрады. Эукариоттар гені (жасушалары ядродан тұратын ағзалар) өте күрделі болып табылады, себебі олар трансляцияланбайтын аймақтармен ауысып тұрады, оларды *интрон деп* атаймыз, олар трансляция қайта орын алу ықтималдығынан бұрын сплайсингтен өтуі керек.

Ойлануға берілетін сұрақ 6.50 *мРНҚ тізбегімен қандай амин қышқылының тізбегі беріледі*
CAU UUU GAG?

Ойлануға берілетін сұрақ 6.51 *Назар аударыңыз, генетикалық кодта (6.1 кесте), амин қышқылдарының көбі бірнеше кодон аналогтарымен берілген. Уақыт өтісімен нуклеотидтер мутацияға ұшырайтындығын ескере отырып, бұл қандай эволюциялық артықшылықтан тұруы мүмкін?*

Ағзаның толық ДНҚ тізбегі оның *геномы* деп аталады. Геномның көлемі қарапайым бактерияларда 10^5 жұп негіз (жн) бен кейбір өсімдітерде 1.5×10^{11} жн диапазоны аралығында болуы мүмкін. Адам геномы шамамен 3.2×10^9 (3.2 миллиард) жн тұрады. Айта кететін жәйт, ғазаның геномының көлемі міндетті түрде оның күрделілігіне сәйкес бола бермейді;

өсімдіктердің геномының көлемі адамдардың геномының көлеміне қарағанда айтарлықтай үлкен.

Биологияның геномдардың құрылысы мен функциясын зерттейтін тармағы *геномика* деп аталады. Геномды жақсырақ түсіну үшін. Геномика саласындағы ғалымдар келесі сұрақтарды қояды:

- Геномдағы әр негіздің пайда болу жиілігі қандай? Әрбір кодонның пайда болу жиілігі қандай? Әрбір амин қышқылында белгілі бір кодонға деген бейімдігі бола ма?

- Гендер қайда орналасқан және олардың функциясы қандай?

- Екі тізбек қаншалықты ұқсас? Тізбектерді салыстыру екі геннің шығу тегі бірдей ме екенін анықтауда қолданылуы мүмкін, оны *гомология* дейміз. Гомологтар арасындағы тізбектердегі салыстыру сонымен қатар геннің белгісіз функцияларын ашуға негіз болуы мүмкін.

- Ағзалар тобы эволюциялық түрде қалай байланысады? Біз тізбектерді салыстыруды пайдаланып *филогенетикалық ағашты* жасай аламыз, ол түрлер арасындағы генетикалық қатынастарды анықтайды.

- Ғалымдар сүтқоректілердің геномдары айтарлықтай дәрежеде қайта біріктірілген ұқсас блоктар жиынтығы болып табылатындығын анықтады, олар *синтения блоктары* деп аталады. Бір геномның басқаға өзгеруіне себеп болған қайта топтастыру әрекеттерінің ең ықтимал деген тізбегі қандай?

- Қандай гендер бірлесіп реттеледі? Бірдей реттелетін гендер тобын сәйкестендіру (идентификациялау) арқылы гендердің, әсіресе аурулармен байланысты гендердің функцияларын түсінуге мүмкіндік береді.

Геномдардың көлемі едәуір үлкен болғандықтан, осындай сұрақтарға тек қана есептеу әдісі арқылы жауап беруге болады. Келесі беттердің біразында біз жоғарыда қойылған сұрақтардың кейбіреуіне жауап беру үшін алдыңғы тараулардағы тәсілдер мен әдістердің қайсысын пайдалануға болатынын қарастырамыз. Көптеген қосымша мысалдарды жаттығу ретінде қалдырамыз. Біз шағын тізбектермен жұмыс істеуді бастаймыз; сосын, файлдар мен веб-сайттардан едәуір ұзынырақ келетін тізбектерді қалай оқу керектігі туралы талқылаймыз.

ДНҚ талдауының негіздері

Жоғарыда бірінші бөлімде қойылған сұрақтарға жауап беру үшін алдыңғы тараудағы кейбір әдістерді қолдануға болады. Мысалы, бактерия геномдары кейде олардың G және C негіздерінің A және T негіздеріне қатынасына сәйкес топтастырылады. G немесе C негіздерінің үлесі геномның GC құрамы деп аталады. GC құрамын есептеудің бірнеше

тәсілі бар. Біріншіден, жай ғана count әдісін қолдануға болады:

```
>>> gc = (dna.count('c') + dna.count('g')) / len(dna)
>>> gc 0.3888888888888889
```

Немесе, алдыңғы тараудағы count1 функциясына ұқсайтын for циклін қолдануға болады:

```
def gcContent(dna):
    """ДНҚ тізбегіндегі GC құрамын есептейді.

    Параметр:
        dna: ДНҚ тізбекті білдіретін жол

    Қайтарылатын мән: ДНҚ тізбегіндегі GC
    құрамы"""

    dna = dna.lower()
    count = 0
    for nt in dna:          # nt қысқартылған
        "нуклеотид" if nt in 'cg':
            count = count + 1
    return count / len(dna)
```

Ойлануға берілетін сұрақ 6.52 *Неліктен біз dna-ді функцияның басында төменгі регистрге түрлендіреміз?*

ДНҚ тізбектері жоғарғы және төменгі регистрлерде бола алатындықтан, бізге функция жазғанда осы мүмкіндіктің екеуін де ескеру қажет. Алайда біздің функциядағы екі мүмкіндікті де тексерудің орнына - алдымен параметрді төменгі немесе жоғарғы регистрге түрлендіру оңайырақ болады.

Гендердегі кодондар статистикасын жинау үшін, белгілі бір кодонның қиылыспайтын кірістерінің санын есептеу қажет. Бұл алдыңғы тараудағы count функциясына қатты ұқсайды, алайда мұндағы айырмашылық - әр қадамда индекстік айнымалыны үшке арттырау керектігінде:

```
def countCodon(dna, codon):  
    """Кодонның ДНҚ тізбегіндегі кіріс санын іздейді.
```

Параметрлер:

dna: Codon ДНҚ тізбегін білдіретін жол:
Кодонды білдіретін жол

Қайтарылатын мән: Dna-дегі кодонның кіріс саны
"""

```
count = 0  
for index in range(0, len(dna) - 2,  
    3): if dna[index:index + 3] ==  
    codon:  
        count = count + 1  
return count
```

Ойлануға берілетін сұрақ 6.53 *Неліктен біз 2-ні жоғарыдағы `len(dna)`-ден алып тастаймыз?*

Тізбектерді түрлендіру

ДНҚ тізбектерін сақтаған кезде мәліметтер қорында тек қана бір жіпшенің тізбегін сақтайды. Алайда гендер мен басқа белгілер кез келген жіпшеде болатындықтан, біз түпнұсқа тізбектен кері комплиментарлы тізбекті шығара алуымыз керек. Алдымен, комплиментарлы тізбекті есептеу үшін, жинақтаушы жолды қолдануға болады, ал әр итерацияда әрбір негіздің комплементін қосу керек.

```
def complement(dna):  
    """ДНҚ-ның комплементарлы тізбегін қайтарады.
```

Параметр:

dna: ДНҚ тізбекті білдіретін жол

Қайтарылатын мән: Комплементарлы ДНҚ тізбегі"""

```

dna = dna.lower()
compdna = ''
for nt in dna:
    if nt == 'a':
        compdna = compdna + 't'
    elif nt == 'c':
        compdna = compdna + 'g'
    elif nt == 'g':
        compdna = compdna + 'c'
    else:
        compdna = compdna + 'a'
return compdna

```

Комплементімізді кері түрлендіру үшін, бізге ДНҚ жолын аударатын алгоритм жазу керек. Оны істеудің екі әдісін қарап көрелік. Біріншіден, бастапқы жолды кері тәртіпте қайта теріп, осы тәртіпке жаңа жолдың символдарын қосуға болатын еді.

Ойлануға берілетін сұрақ 6.54 *Индекстерді кері тәртіпте қалай қайта теруге болады?*

Индекстерді кері тәртіпте range арқылы қайта теруге болады, ол len(dna) - 1 басталады да -1-ге дейін (бірақ қоспайды) -1 қадаммен жалғасады. Келесі функция осы тәсілді пайдаланады.

```

def reverse(dna):
    """ДНҚ-ның кері тізбегін қайтарады.

```

Параметр:

dna: ДНҚ тізбекті білдіретін жол

Қайтарылатын мән: Кері ДНҚ тізбек"""

```

revdna = ''
for index in range(len(dna) - 1, -1, -
    1): revdna = revdna +
    dna[index]
return revdna

```

Жақсырақ шешімде dna символдары жай ғана әдеттегі тәртіпте қайта теріледі, алайда әр символды revdna жолының басына қосады.

```

def reverse(dna):
    """ (құжаттау жолдары түсірілген) """

    revdna = ''
    for nt in
    dna:
        revdna = nt + revdna
    return revdna

```

Оның қалай жұмыс істейтіндігін көру үшін, dna - 'agct' болған дейік. Төменде берілген кесте циклдың әр итерациясында мәннің қалай өзгеретіндігін көрсетеді.

nt итерациясы	revdna жаңа мәні
1	'a' 'a' + '' → 'a'
2	'g' 'g' + 'a' → 'ga'
3	'c' 'c' + 'ga' → 'cga'
4	't' 't' + 'cga' → 'tcga'

Ендігі жерде, кері комплементке функция жасау үшін, оіз осы екі бөлшекті бірге қоя аламыз:

```
def reverseComplement(dna):
    """ДНҚ кері комплементарлы тізбегін қайтарады.

    Параметр:
        dna: ДНҚ тізбекті білдіретін жол

    Қайтарылатын мән: ДНҚ кері комплементарлы тізбегі
    """
    return reverse(complement(dna))
```

Бұл функция алдымен dna-дан complement-ті есептейді, сосын нәтиже үшін reverse шығарады. Енді біз, мысалы, reverseComplement функциясын ДНҚ тізбегінің екі тізбектегі нақты кодонның пайда болу жиілігін санау үшін қолдана аламыз:

```
countForward = countCodon(dna, 'atg')
countBackward = countCodon(reverseComplement(dna), 'atg')
```

Тізбектерді салыстыру

Салыстырмалы геномика деп аталатын ДНҚ тізбектері арасындағы ұқсастықты өлшеу, заманауи биологияның маңызды бағытының біріне айналды. Әр түрлі түрлердің геномдарын салыстыру эволюциялық эволюциялық қарым-қатынастардың басты түсінулерін қамтамасыз етеді. Биологтар, сондай-ақ белгісіз геннің функциясын аша алады, ол үшін оны белгілі функциялары бар эволюциялық туыс түрлерінің гендерімен салыстырады. Нүктелі диаграммалар есептеу геномикасында кеңінен қолданылады, ол үлкен ДНҚ тізбектері мен амин қышқылдарының қаншалықты ұқсас екендігін визуалды түрде елестетуді қамтамасыз етеді. Келесі екі тізбекті қарап көрелік. 6.5-тарауындағы dotplot1 функциясы қарамен жазылған нуклеотидтер арасындағы жұптық сәйкестіктерді ғана табады.

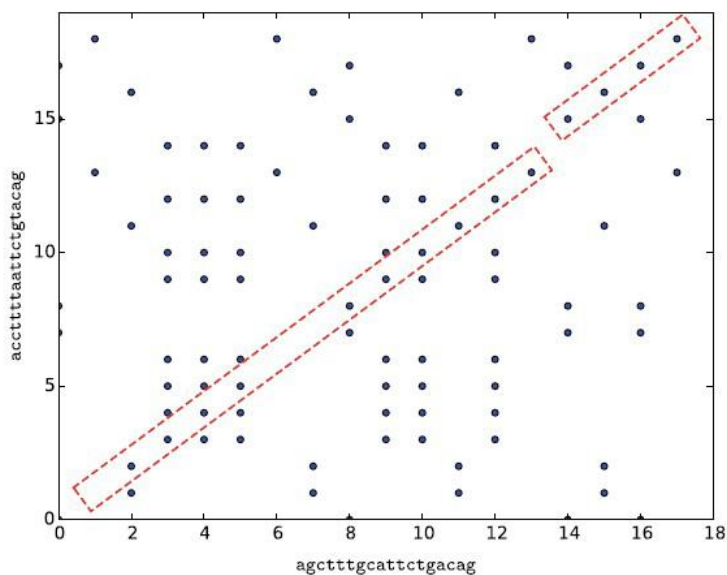
1-тізбек:	agctttgcattctgacag
2-тізбек:	acctttaattctgtacag

Бірақ, мұнда екі тізбектегі соңғы төрт негіз іс жүзінде бірдей екеніне

назар аударыңыз; егер бірінші тізбекте, екінші тізбектегі соңғы t үстінен бос орын қойсаңыз, онда әр түрлі негіздер саны сегізден төртке дейін қысқарады, ол төменде көрсетілген.

1-тізбек: agctttgcattctg- acag
2-тізбек: acctttaattctgtacag

Эволюциялық түрде, егер осы екі түрдің ең жақын ортақ атасы ДНҚ-ның құрамындағы бос орында t болған болса, онда біз бұл бос орынды бірінші тізбектегі делеция деп түсіндіреміз. Немесе, егер ең жақын ортақ атасында t болмағанда, онда



6.10-сурет agctttgcattctgtacag және acctttaattctgtacag жекелеген негіздерді салыстыратын нүктелі диаграмма. Түзетудің негізгі учаскесін білдіретін нүктелер белгіленіп көрсетілген.

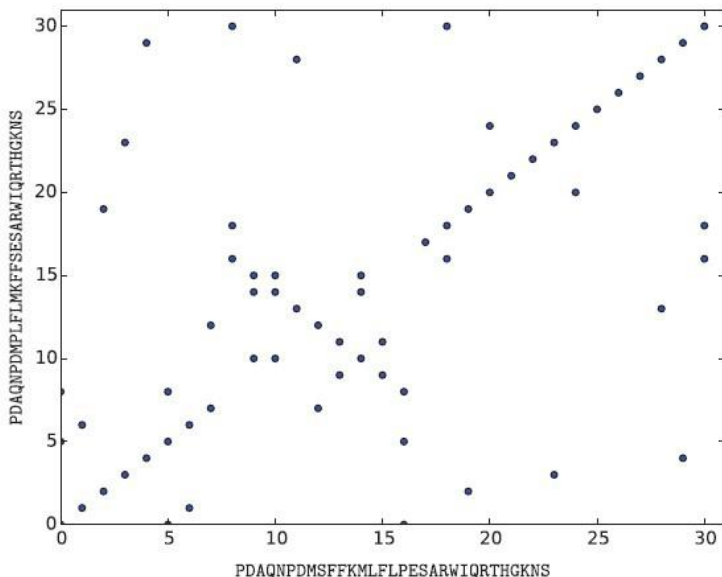
біз бос орынды екінші тізбектегі қосымша ретінде түсіндіреміз. Осы қосымшалар мен делециялар, барлығы бірге мутациялар деп ата-

лады, олар өте жиі кездеседі; сондықтан, тізбектерді салыстыратын алгоритмдер оларды ескеру қажет. Дәл dotplot1 функциясы секілді мәтіндер арасындағы сәйкестілікті лайықты түрде көрсетпейді, және ДНҚ тізбектері үшін де осылай. 6.10 суретте жоғарыдағы тізбектер үшін 6.5-тарауындағы dotplot функциясын қолданған толық нүктелі диаграммасы көрсетілген.

Алдыңғы тараудан байқағанымыздай, нүктелі диаграммалардың белгіленген, мәселен, сырғымалы терезедегі А ұзындықтағы тізбектерді салыстыру арқылы “шуды” азайту кезінде пайдасы арта түседі. А ұзындықтағы ықтимал 4А түрлі тізбектер бар болғандықтан, сәйкестіктер саны да аз болады. Нүктелі диаграммалар сондай ақ амин қышқылдарының тізбектерін салыстыруда пайдалы. 20 түрлі амин қышқылдары бар болғандықтан, біз графиктерде азырақ шу көруіміз мүмкін. Мысалы, 6.11- суретте келесі гипотезалық шағын ақуыздардың нүктелі диаграммасы көрсетілген. Әр әріп әр түрлі амин қышқылына сәйкес келеді. (Егер бұл қызықтыратын болса, әріптердің мәнін 6.1-кестеден көре аласыз.)

```
seq1 = 'PDAQNPDMSSFFKMLFLPESARWIQRTHGKNS'  
seq2 = 'PDAQNPDMPLFLMKFFSESARWIQRTHGKNS'
```

Графикте көрстелгін бір тізбекті екінші, жоғарыда қызылмен белгіленген тізбекпен салыстырудағы *инверсиясына* назар аударыңыз.



6.11-сурет Екі гипотезалық шағын ақуызды салыстыратын нүктелі диаграмма

PDAQNPDSFFKMLFLPESARWIQRTHGKNS

және PDAQNPDMPLFLMKFFSESARWIQRTHGKNS.

Тізбектер файлдарын оқу

ДНҚ тізбектері мұрағатталып, Ұлттық Денсаулық сақтау Институты (НИИ) бөлімшесі, Ұлттық биотехнологиялық ақпарат орталығы басқармасының (NCBI) мәліметтер қорында барлығына қолжетімді етіп сақталды. Мәлімет қорындағы әр тізбектің идентификаторы болады, оны есептік нөмір дейді. Мысалы, торлы питонның бірінші геном сегментінің есептік нөмірі (Python molurus bivittatus) - AEQU02000001 болып табылады. Бұл мәліметтерді NCBI веб-сайтынан көруге болады <http://www.ncbi.nlm.nih.gov>

ол үшін іздестіру өрісіне инвентарлық нөмірді енгізіп, шыққан тізімнің ішінен “нуклеотидті” таңдайсыз.

Геномдық мәлімет әдетте FASTA [38] форматында сақталады. FASTA файлы алдында үлкен (>) белгісі тұратын тақырыптан басталады, одан кейін тізбек мәліметінің бірнеше жолы жүреді. Тізбектің әр жолы жолды

аудару символымен аяқталады. Мысалы, торлы жыланның бірінші геном сегментінің FASTA файлы мынадай болады:

```
>gi|541020484|gb|AEQU02000001.1| Python bivittatus ...
AGGCCCTGGCGCAATATGGTTCATGGGGTCACGAAGAGTCGGACACGACTTAACGACTAAACAACAATTAATCTAAACCCTAGCTATCTGGGTTGCTTCCCTACATTATATTCCGTGAATAAATCTCACTAAACCTCAGAGGATTACTTTTGAGTGAACACTCATGATCAAGAACTCAGAAAACTGAAAGGATTCCGGGAATAGAAGTGTTATTTTCAGCCCTTTCCAATCTTTCTAAAAAGCTGACAGTAATACSTTTTTTTGTTTTAAAAATTTAAAAAAGAGCTCAGACTTATGCTACAGAAATGAGTCTGTAACAGGGAAGCCAGAAAGGAAAAAGACTGAAACGTAACCAAGGACATAATGCAGGGATCATAATTTCTAGTTGGCAAGACTAAAGTCTATACATGTTTTATTGAAAAACCACTGCTATTTGCTCTTACAAGAACTTATCCCTCCAGAAAAAGTAGCACTGGCTTGATACTGCCTGCTCTAACTCAACAGTTTCAGGAAGTGCTGGCAGCCCTCCTAACACCATCTCAGCTAGCAAGACCAAGTTGTGCAAGTCCACTCTATCTCATAACAAAACCTCTTAAATTAAGTTGAAAGCCTAGAGCTTTTTTTACCTTGGTCTTCAAGTCTTCGTTTGTACAATATGGGCAATGTCATAGCTTGGTGTATTAGTATTATTATTGTACTGTTGGTTACACAGTCAGTCAGATGTTTAGACTGGATTTGACATTTTACCCTATGATCGAGTGCCTTCCCAAGGACCTGGGATAGGCAGATGTTGGTGTTGATGGTGTTAAAGGT
```

Осы ДНК тізбекті бағдарламада пайдалану үшін, тақырып пен жолды аудару символдарын алып тастап, тізбекті жолдан ешбір өзгертусіз шығарып алуымыз керек. Осыны істетейін функция 6.2- тараудағы функцияға өте ұқсас келеді.

```
def readFASTA(filename):
    """Жалғыз тізбектен құралған FASTA файлы оқиды,
       және тізбекті жол түрінде қайтарады.
```

Параметр:

filename: FASTA файлының атын білдіретін жол

Қайтарылатын мән: FASTA файлындағы тізбектен тұратын жол"""

```
inputFile = open(filename, 'r', encoding = 'utf-8')
header = inputFile.readline()
dna = ''
for line in inputFile:
    dna = dna + line[:-1]
inputFile.close()
return dna
```

FASTA файлын ашқан соң, біз тақырып жолын `thereadline` функциясы арқылы оқимыз. Бізге тақырып қажет емес, сондықтан `readline` бірінші жолдан кейін файлға сілтеушіні нақты тізбектің басына жылжыту үшін ғана қолданылады. Файлдың қалған бөлігін оқу үшін, біз оны жол-жолдан қайта тереміз. Әр итерацияда соңындағы жол аудару символынсыз тағы жол қосамыз да, `dna` атты жолды арттырамыз. Кітап веб-сайтында жоғарыда берілген FASTA файлына тікелей сілтеме бар, сондықтан Сіз осы функцияны тексере аласыз.

Сондай-ақ, FASTA файлдарына тікелей URL арқылы кіруге болады, ол NCBI-ға сұрау сал жібереді. Төмендегі URL NCBI веб-сайтынан торлы питон (есептік нөмірі AEQU02000001) геномының бірінші сегментінің FASTA файлын жүктеуді сұрайды.

<http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore &id=AEQU02000001&rettype=fasta&retmode=text>

Басқа тізбекті алу үшін жай ғана басқа есептік нөмірді енгізу керек (қызыл түсті). Келесі функция барлық сегменттерді біріктіреді.

```
import urllib.request as web
```

```
def getFASTA(id):
```

```
    """NCBI қорынан идентификаторы белгіленген ДНҚ  
    тізбегін алып, оны жол түрінде қайтарады.
```

```
    Параметр:
```

```
        id: ДНҚ тізбегінің идентификаторы
```

```
    Қайтарылатын мән: Идентификаторы берілген тізбекті  
    білдіретін жол"""
```

```
    prefix1 =
```

```
    'http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi'
```

```
    prefix2 = '?db=nuccore&id='
```

```
    suffix =
```

```
    '&rettype=fasta&retmode=text' url
```

```
    = prefix1 + prefix2 + id + suffix
```

```
    readfile = web.urlopen(url)
```

```
    header = readfile.readline()
```

```

dna = ""
for line in
    readFile: line
        = line[:-1]
        dna = dna + line.decode('utf-8')
readFile.close()
return dna

```

Функция алдымен жол жасайды, ол `id` параметрінде берілген есептік нөмірге арналған URL-дан тұрады. URL-дың бірінші ортақ бөліктері `prefix` айнымалыларына берілді, ал URL-дың соңғы бөлігі `suffix`-ке берілді. Осы бөліктерді есептік нөмірмен біріктіріп жеке URL жинақтаймыз. Со-сын URL-ды ашамыз да, алдыңғы `readFASTA` функциясындағы кодты ДНҚ тізбегін алу үшін пайдаланып, оны жол түрінде қайтарамыз.

Ойлануға берілетін сұрақ 6.55 Торлы питон (есептік нөмір AEQU02000001) геномының бірінші сегментін оқу үшін, `getFASTA` функциясын пайдаланыңыз.

Біз есептеу геномикасының дамуышы сласын аз ғана қозғадық, сәл жақындадық. Келесі жаттығу бірнеше қосымша мәселелерді зерттейді және Сізге алгоритм жасап жаттығуға және бағдарламалау дағдыңызды нығайтуға мүмкіндік береді.

Жаттығулар

6.7.1. Функцияны жазыңыз

`countACG(dna)`

ол берілген ДНҚ тізбегіндегі нуклеотидтердің үлесін қайтарады, олар `T` негіз болып табылмайды. Жолдағы символдарды қайта теретін `for` циклын қолданыңыз.

6.7.2. Алдыңғы жаттығуды қайталаңыз, бірақ символ орнына

жол индекстерін теретін `for` циклын қолданыңыз.

6.7.3. Функцияны жазыңыз

`printCodons(dna)`

ол экранға кодондарды `dna` жолының сол жақ аяғынан бастап шығарады. Мысалы,

`printCodons('gggtacactgt')` экранға мынаны шағарады

```
gg
t
ac
a
ct
g
```

6.7.4. Функцияны жазыңыз

```
findCodon(dna, codon)
```

ол dna жолындағы берілген кодонның codon бірінші кірісінің индексін қайтарады. countCodon функциясымен салыстырғанда, мұнда Сіз циклдың индексін 3-тің орнына 1-ге арттырасыз да, кодонды кез келген позициядан таба аласыз.

6.7.5. Функцияны жазыңыз

```
findATG(dna)
```

ол берілген ДНҚ жолында ATG кодонының барлық орындары индекстерінің тізімін жасайды. Кіріктірілген index әдісін немесе find әдісін қолданбаңыз. Индекстердің тізімін құрастыру графиктерді құру үшін тізімдерді жасау кезіндегі бірнеше рет жасалған әрекеттерге өте ұқсас.

6.7.6. Әр кодон үш негізден тұратындықтан, транскрипция сол жақ соңынан 0, 1 және 2 ығыстырылуынан басталатын кез келген ДНҚ тізбегіндегі үш тәуелсіз оқылу аясының бірінде орын алуы мүмкін. Мысалы, егер біздің ДНҚ тізбегіміз ggtacactgtcat болса, үш оқылу шеңберінде кодондар мыналар болады:

ОА 1: ggt aca ctg tca t

ОА 2: g gta cac tgt cat

ОА 3: gg tac act gtc at

Функцияны жазыңыз

```
printCodonsAll(dna)
```

ол 6.7.3-жаттығуындағы Сіздің функцияңызды барлық үш оқылу шеңберіндегі кодондарды экранға шығару үшін қолданады.

6.7.7. Көптеген омыртқалы жануарлардың CG жұптарының (CG динуклеотидтері деп аталады) тығыздығы болжағанымыздан әлдеқайда кішкентай болады. Дегенмен, олардың гендерінің концентрациялары (кодталатын аймақтар) әдетте едәуір жоғары болады. Осы себептен, “СрG аралдары” деп аталатын осындай аймақтарды іздеу гендердің шоғырланған жерін табуға көмектесетін өте жақсы әдіс болып табылады. (С мен G арасындағы “р” С мен G екі комплементарлы

тізбектің сутектік байланысының орнындағы фисфодизфирлі байланысты білдіреді.) Кіріктірілген count әдісін қолданбай, мына функцияны жазыңыз

CpG(dna)

ол cg нуклеотидтерінің үлесін қайтарады. Мысалы, егер dna atcggtcg болса, онда функция 0.5 қайтару керек болар еді, себебі тізбектің жартысы CG динуклеотидтерінен тұрады.

6.7.8. Микросателлиттер немесе жай қайталанатын тізбектер (SSR) дегеніміз - қысқа ДНҚ тізбегінің қайталануы болып табылады. Қайталатын тізбектер, әдетте ұзындығы 2–4 негізден тұрады және 10-100 рет немесе одан да көп қайталану алады. Мысалы, сасасасаса дегеніміз - са-ның жай қысқа қайталануы, ол адамда жиі қайталану алады. Микросателлиттер көптеген ағзалардың ДНҚ-да кең тараған, алайда олардың популяциялардағы ұзындықтары көшіру үдерісіндегі “сырғанау” нәтижесінде болатын репликация қателіктері үшін әралуан болады. Популяциялар ішінде және арасында ұзындықтар әралуандығының таралуын салыстыруды - генетикалық туыстықты анықтауға және эволюциялық үдерістерді зерттеуге қолдануға болады.

Функцияны жазыңыз

ssr(dna, repeat)

ол repeat параметрін қайталайтын dna-дегі бірінші SSR ұзындығын қайтарады. Егер repeat табылмаса, 0 қайтарылады. Жолдық тобының find әдісін қолданып, repeat-тің dna-ге бірінші кірісін іздеп табыңыз.

6.7.9. ssr функциясының тағы бір нұсқасын жазыңыз, ол repeat параметрін қайталайтын dna-дегі ең ұзын SSR-дің ұзындығын табады. Функцияңыз алдыңғы тапсырмадағы ssr функцияны бірнеше қайтара шақыру керек. Сіз while циклын қолдана аласыз.

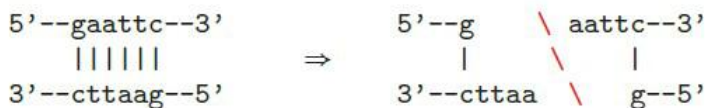
6.7.10. ssr функциясының үшінші нұсқасын жазыңыз, ол алдыңғы тапсырмадағы функцияны пайдаланып, dna-да *кез келген динуклеотидтің* (ұзындығы 2-ге тең тізбек) ең ұзын SSR-ды табады. Функцияңыз ең ұзын қайталанған динуклеотидті қайтару керек.

6.7.11. Функцияны жазыңыз

palindrome(dna)

Ол dna өзінің кері комплементімен сәйкес келсе True қайтарады, ал керісінше болса False қайтарады. (Бұл стандартты палиндром анықтамасынан айрықша екеніне назар аударыңыз.) Мысалы, gaattc - палиндрома болып табылады, себебі ол өзінің кері комплементіне сәйкес келеді. Бұл тізбектердің маңызы өте зор, себебі кейбір рестриктивті ферменттер ерекше палиндромалық тізбектерді анықтайды да, ДНҚ

молекуласын орналасқан жерінде кеседі. Мысалы, EcoR1 рестриктивті ферменті *Escherichia coli* ішек таяқшасы бактериясының *gaattc* тізбектегі ДНҚ-сын төмендегідей кеседі:



6.7.12. Функцияны жазыңыз

`dna2rna(dna)`

ол әрбір Т негіз U-мен ауыстырлған dna көшірмесін қайтарады.

6.7.13. Функцияны жазыңыз

`transcribe(dna)`

ол кері комплементарлы dna-дің РНҚ баламасын қайтарады. (Осы мақсатқа бір жолмен жету үшін алдыңғы жазылған функцияларды қолданыңыз.)

6.7.14. Функцияны жазыңыз

`clean(dna)`

ол ДНҚ жаңа жолын қайтарады, ондағы А, С, G, немесе Т болып табылмайтын әр символ N-мен ауыстырылады. Мысалы, `clean('goat')` - `'gnat'` жолын қайтару керек.

6.7.15. ДНҚ молекулаларын секвенирлеу кезінде жиі белгісіздіктер туындайды. Осындай мәселелерді шешу үшін, “көп белгілі символдар” анықталады, олар бір немесе бірнеше негіздерді кодтайды.

Символ	Ықтимал	Символ	Ықтимал
R	A G	B	C G T
Y	C T	D	A G T
K	G T	H	A C T
M	A C	V	A C G
S	C G	N	A C G T
W	A T		

Функцияны жазыңыз

`fix(dna)`

ол көп белгілі символ ықтимал негіздердің бірімен ауыстырылған ДНҚ жолын қайтарады. Мысалы, егер dna-де R бар болса, ол 1/2 ықтимал А -ге немесе 1/2 ықтимал G-ге ауыстырылады.

6.7.16. Функцияны жазыңыз

mark(dna)

ол жаңа ДНҚ жолын қайтарады, ондағы әрбір АТГ бастамашы кодон >>> белгісімен алмастырылады, ал әрбір терминациялаушы кодон (ТАА, TAG, немесе TGA) <<< белгісімен алмастырылады. Циклыңыз әр итерацияда 3 есе артылуы керек, сондықтан тек қана қиылыспайтын кодондарды қараңыз. Мысалы, mark('ttgatggagcattagaag') - 'ttg>>>gagcat<<<aag' жолын қайтару керек.

6.7.17. Гепатит А вирусының есептік нөмірі -NC_001489 болып табылады. Гепатит А вирусына ДНҚ тізбегін алу үшін getFASTA функциясын қолданатын бағдарлама жазыңыз, сосын осы тараудағы gcContent функциясын жазып А гепатитіндегі GC құрамын табыңыз.

6.7.18. С гепатиті вирусының ДНҚ бір ұзын ақуызды кодтайды, ол ақырында он шағын ақуызды алу үшін протеаза деп аталатын ферменттермен өңделеді. Осы вирустың жеті түрі болады. 1-тип пен 2-тип арқылы түзілетін ақуыздардың есептік нөмірі - NP_671491 және YP_001469630 болып табылады. getFASTA функциясы мен 6.6.5-жаттығуындағы Сіздің нүктелі диаграмма функциясын пайдаланатын бағдарлама жазып, 4 көлемдік терезені пайдаланып, осы екі ақуызды салыстыратын нүктелі диаграмма жасаңыз.

6.7.19. *Гомеозистік* гендер эмбриондық дамудың негізгі аспектілерін бақылайды. Бастапқы даму сатысында эмбрион бірнеше сегменттен тұрады, кейін олар бастан құйрыққа дейінгі дене құрылымының негізгі симметриялық осіне айналады. Гомеозистік гендер әрбір сегменттің қай дене мүшесі болатынын бұйырады. Нох А1 деп аталатын бір нақты гомеозистік ген мидың артқы жағының дамуын бақылайды. getFASTA функциясы мен 6.6.5-жаттығуындағы Сіздің нүктелі диаграмма функцияңызды пайдаланатын бағдарлама жазып, 8 көлемдік терезені пайдаланып, адам мен тышқанның Нох А1 гендерін салыстыратын нүктелі диаграмма жасаңыз. Адамның Нох А1 генінің есептік нөмірі - U10421.1 ал тышқанның Нох А1 генінің есептік нөмірі - NM_010449.4.

6.10. ТӘЖІРИБЕЛІК ТАПСЫРМАЛАР

*Жоба 6.1. Партиялы-бағдарланған саясат

Құрама Штаттар үкіметінің заң шығарушы билігі, өзінің екі партиялық жүйесімен, осы екі тараптар бір-бірімен заңдар қабылдауда тиімді

ынқтымақтасатын, сондай-ақ басқа да партиялы заң шығарауға ие сатылардан өтеді. Осы жобада, біз АҚШ Өкілдер Палатасы Спикерінің веб-сайтындағы мәліметтерді қолданып, (<http://clerk.house.gov/legislative/legvotes.aspx>) уақыт ағымындағы Өкілдер палатасының электоратының жұмысы мен бағдарын талдаймыз.

1-бөлім: Партиялық бағыттағы сайлау

Бір партияның дауыс беруші мүшелерінің ең кем деген жартысы бір тәсілмен, ал дауыс беруші мүшелердің екінші жартысы басқа тәсілмен дауыс беруді *партиялық бағыттағы* сайлау дейміз. Палатада дауыс беру кезінде әр мүше тіркелген кезде оны *аттары аталған* сайлау дейміз. Аттары аталған сайлаудың жеке нәтижелері URL мекенжайы бойынша келесі сайтта берілген:

<http://clerk.house.gov/evs/<year>/roll<number>.xml>

<year> өрісі дауыс берілген жылмен ауыстырылады ал

<number> өрісі аттары аталған сайлау нөмірімен ауыстырылады. Мысалы, 2010 жылғы 194 аты аталған сайлаудың нәтижелері (Қолжетімді медициналық қызмет көрсету туралы заңына соңғы дауыс беру) мына сайтта

<http://clerk.house.gov/evs/2010/roll194.xml>

Егер аттары аталған сайлау нөмірі үш цифрдан аз болса, ол нөлдермен толықтырылады.

Мысалы, 2010 аттары аталған 11 сайлаудың нәтижелері мына сайтта берілген

<http://clerk.house.gov/evs/2010/roll011.xml>

Мына файлдардың бірін қарау үшін мынадай бағдарламаны іске қосыңыз:

```
import urllib.request as web
```

```
def main():
```

```
    url = 'http://clerk.house.gov/evs/2010/roll194.xml'
```

```
    webpage = web.urlopen(url)
```

```
    for line in webpage:
```

```
        line =
```

```
line.decode('utf-8')
print(line.rstrip())
webpage.close()
```

```
main()
```

Сіз элементтерінің сәйкес келетін

```
<recorded-vote> ... </recorded-vote>
```

көптеген XML тэгтерін байқайсыз. Осы элементтердің әрқайсысы Палата мүшесінің дауысы болып табылады.

Алдымен, функцияны жазыңыз

```
partyLine(year, number)
```

ол Палатаның аттары аталған сайлау нөмірі number белгіленген year жылы партиялық бағытты ұстанғандығын анықтайды. Тек қана дауыстарды жазу элементтерін пайдаланып, XML файлдардың басқа бөліктеріне тиіспеңіз. Ешқандай кіріктірілген әдістерді қолданбаңыз. Кейбір дауыстар Yea/Nay (Иә/Жоқ) ал кейбірі Aye/No (Иә/Жоқ) деп жазылғанын ұмытпаңыз. Басқа кез келген тәсілмен жазылған дауыстарды (мыс., “Present” - Бар немесе “Not Voting” - шет қалды), тіпті жалпы дауыс санын есептеген кезде санамаңыз. Функцияңызды келесі қадамды жасамастан бұрын мұқият тексеріп алыңыз.

6.1.1-тапсырма *Бұл функция қандай мәлімет түрін қайтару керек? (Оны келесі функция үшін қалай оңайлатуға болатыны туралы ойланыңыз.)*

6.1.2-тапсырма *Қолжетімді медициналық қызмет көрсету туралы заңына дауыс беру партиялық бағытты ұстанды ма?*

6.1.3-тапсырма *Өзіңізге қызықты басқа дауыс беруді таңдаңыз. Ол партиялық бағыттағы сайлау болды ма?*

Екінші, басқа функция жазыңыз

```
countPartyLine(year, maxNumber)
```

ол partyLine функциясын қолданып, берілген year жылындағы

партиялық бағытты ұстанған дауыс берулер үлесін қайтарады. `maxNumber` параметрі - жылдағы соңғы аттары аталған дауыс берудің нөмірі.

Енді, мына функцияны жазыңыз
`plotPartyLine()`

ол соңғы 20 жылдағы партиялық бағытты ұстанған дауыс беру үлесін графикке түсіреді. Бұл тапсырманы оңайлату үшін, жыл сайын 450 аттары аталған дауыс беру өтеді делік. Ал егер Сіз барлық аттары аталған дауыс беру санын есептегіңіз келсе, төменде соңғы 20 жыл ішіндегі дауыс берудің саны берілген:

Жыл	Нөмір	Жыл	Нөмір
2013	641	2003	677
2012	659	2002	484
2011	949	2001	512
2010	664	2000	603
2009	991	1999	611
2008	690	1998	547
2007	1186	1997	640
2006	543	1996	455
2005	671	1995	885
2004	544	1994	507

6.1.4-тапсырма *1994 - 2013 жж. аралығындағы әр жылғы партиялық бағытты ұстанған дауыс беру санының үлесі қандай?*

Бұл мәліметті жинау үшін ұзақ уақыт кететінін ескеріңіз, сондықтан алдымен функцияңыздың жеке бір жыл үшін дұрыстығын тексеріп алыңыз.

6.1.5-тапсырма *Графигіңізді сипаттап беріңіз. Соңғы 20 жыл ішінде Американдық саясат партиялық бағытталған саясатқа айналды ма?*

6.1.6-тапсырма *Көптеген жаңалықтар партиялық бағытталу ту-*

ралы мәселені алға қояды . Желіден осы туралы жаңалықты тауып, оны алынған нәтижелермен салыстырыңыз. Жаңалық агенттіктерінің уәждемесі туралы ойлау пайдалы болады.

Бөлім 2: Демократиялық және республикандық штаттар

Аттары аталған дауыс беру файлдарын басқа статистикалық мәліметтерді шығару үшін де қолдануға болады. Осы бөлімде функция жазыңыз

stateDivide(state)

Ол соңғы 20 жылдағы әр жылдағы берілген штат үшін Демократтар мен Республикандықтар өкілдерінің санын графикке белгілейді. Графигіңізде екі түрлі ирек салу керек (біріншісі - Демократтар және екіншісі - Республикандықтар үшін). Сондай-ақ басқа партиялар үшін де иректерді қосуға болады. Функцияңызды бірнеше штат түріне тексеріп көріңіз.

stateDivide функцияңызды өзіңіздікін қоса алғанда бес штат үшін шақырыңыз.

6.1.7-тапсырма *Әр штат үшін алынған иректерді сипаттап беріңіз. Алынған нәтижелерді 1-бөлімдегі нәтижелерді ескеріп түсіндіріп беріңіз. Өкілдер Палатасының партиялық бағытталғандығы артты ма ?*

***Жоба 6.2. Гендерді іздеу**

Бұл жоба үшін 6.7-тарауындағы материалды оқу керек.

Прокариоттық ағзалардағы гендердің кодтаушы аймақтарынан бұрын әдетте ATG бастамашы кодон жүреді, олар терминациялаушы кодондардың бірімен аяқталады: TAA, TAG, және TGA. Бастамашы және терминациялаушы кодондармен межеленген, арасында тоқтату кодондары жоқ ДНҚ ұзын аймағы- ашық оқу шеңбері (немесе ORF) деп аталады. Айтарлықтай ұзын ORF-терді іздестіру - гендердің ықтимал орналасу жерін табатын тиімді (алайда, жоғарғы машықты талап ететін) әдіс болып табылады.

Мысалы, төмендегі тік төртбұрышпен нақты бір ашық оқитын меже белгіленген, ол бастамашы ATG кодонымен басталып, TAA стоп кодоны-

мен аякталады. Бұл жерде олардың арасында басқа тоқтату кодондардың жоқ екендігіне назар аударыңыз.

ggcggatgaaacgcattagcaccaccattaccaccaccatcaccattaccacaggttaagttc

ORF барлығы генге сәйкес келе бермейді. $43 = 64$ ықтимал кодондар мен 3 түрлі тоқтату кодон бар болғандықтан ДНҚ кездейсоқ аймағына шамамен әрбір $64/3 \approx 21$ кодондарға бір тоқтату -кодон табылады. Сонда, бізге іс жүзінде бұған қарағанда ұзындығы өте үлкен ORF-тер қажет екен. Мұндай ұзын ORF-тер кездейсоқ шықпайтындықтан, оларды геннің кодтаушы аймағы болып табылады деп айтуға айқын дәлел бар.

ДНҚ жіпшесіндегі ашық оқу жиектемесінің барлығы тізбектің сол жақ соңына (5') тураланбайды. Мәселен, жоғарыда берілген мысалда, егер біз сол жақ соңына қарай тураланған — ggc, gga, tga, т.б. кодондарда оқу жиектемесін іздеп бастайтын болсақ, онда тік төртбұрышпен белгіленген ашық оқылу жиектемелерін қалдырып кетер едік. ДНҚ жіпшесіндегі барлық ашық оқылу жиектемелерін табу үшін, біз сол жақ ұшынан 0, 1 және 2 ығысатын кодондарды қарауымыз керек. (Басқа мысалды 6.7.6 жаттығудан қараңыз.) Осылай ығысқан кодон тізбектері үш *тікелей оқу жиектемелері* деп аталады. Жоғарыдағы мысалда, үш тікелей оқу жиектемелері былай басталады:

Оқу жиектемесі 0: ggc gga tga aac . .
. Оқу жиектемесі 1: gcg gat gaa acg .
. . Оқу жиектемесі 2: cgg atg aaa cgc . . .

ДНҚ қос тізбекті болғандықтан, екінші тізбектің кері комплементарлы тізбегінде сондай ақ үш кері оқу жиектемесі болады. ДНҚ осы тізбегіндегі кері оқу жиектемелері мыналар болатын еді:

Кері оқу жиектемесі 0: gaa ctt acc tg. .
. Кері оқу жиектемесі 1: aac tta cct gtg .
. . Кері оқу жиектемесі 2: act tac ctg tgg . . .

Геномдық тізбектер айтарлықтай ұзын болып келетіндіктен, ORF-терді іздеу есептеу әдістері арқылы орындалуы керек. Мысалы, келесі 1,260 нуклеотидтен тұратын *Escherichia coli* (немесе *E. coli*) ішек таяқшасының геномындағы тізбекті қарап көрелік, ол бүкіл геномның тек шамамен 0.03% ғана құрайды. (Бұл сан адам геномының шамамен

0.00005% құрайды.) Бұл тізбектен ашық оқу жиектемесін таба аласыз ба?

agcttttcattctgactgcaacgggcaatatgtctctgtgtggattaaaaaagagtgtctgatagcagc tt
ctgaactgggtacctgccgtgagtaaatataattttattgacttaggtcactaaatactttaaccaa tataggca
tagcgcacagacagataaaaattacagagtacacaacatccatgaaacgcattagcaccacc attaccacc
accatcaccattaccacaggtaacgggtgcgggctgacgcgtacaggaaacacagaaaaaag ccgcgacc
tgacagtgcgggctttttttcgaccaaaggtaacgaggtaacacccatgcgagtgtgaag ttcggcggtac
atcagtggcaaatgcagaacgttttctgcgggtgccgatattctggaagcaatgccca ggcaggggaggt
ggccaccgtcctctctgccccgccaaaatcaccaaccatctggtagcgtatgattga aaaaaccattagcgg
tcaggatgctttaccaatatcagcgatgccgaacgtattttgccgaactctg acgggactcgcgcgcgc
cagccgggatttccgctggcacaattgaaaactttcgtcgaccaggaatttg cccaataaaacatgtctctgc
atggcatcagttgttggggcagtgcccgatagcatcaacgctgcgct gatttgcctggcgagaaaatgt
cgatcgccattatggccggcgtgttagaagcgcgtggtcacaacgtt accgtatcgatccggtcgaaaaac
tgctggcagtgggctcattacctgaatctaccgttatattgtg aatccaccgcgctattgcggcaagccg
cattccggctgaccacatggtgctgatggtggttctactgc cgtaaatgaaaaggcgagctggtggttct
gggacgcaacgggtccgactactccgctgcgggtgctggcg gcctgtttacgcgccgattgtgagatctg
gacggatgttgacggtgtttatacctgcgatccgcgtc aggtgcccgatgcgaggtgttgaagtcgatgtcc
tatcaggaagcgatggagctttctacttccggcg taaagtcttaccccccgcaccattacccccatgccca
gttccagatcccttgctgattaaaaatacc ggaaatcccaagcaccaggtacgctcattggtgccagccgt
gatgaagacgaattaccggtcaaggga

Бұл жұмыс адамға лайық емес көрінеді. Алайда, егер көп уақыт жұмсақ, онда 29 бен 27 позициялары арасынан 2 оқу жиектемесінен ORF -ті, 189 бен 254 позициялары арасындағы 0 оқу жиектемесінен, және 915 пен 1073 позициялары арасынан 0 оқу жиектемесінен ORF-терді табуға болады (бұл тек кейбірі ғана). Олар төменде қызылмен белгіленген.

agcttttcattctgactgcaacgggcaatATGTCTCTGTGTGGATTAAAAAAGA
GTGTCTGATAGCAGC TTCTGAACTGGTTACCTGCCGTGAGTAAatta
aaattttattgacttaggtcactaaatactttaaccaa tataggcatagcgcacagacagataaaaattacagag
tacacaacatccATGAAACGCATTAGCACCACC ATTACCACCACCATCAC
CATTACCACAGGTAACGGTGC GGGCTGAcgcgtacaggaaacacagaaaaaag
ccgcacctgacagtgcgggctttttttcgaccaaaggtaacgaggtaacacccatgcgagtgtgaag ttc
ggcggtacatcagtggaatgcagaacgttttctgcgggtgccgatattctggaagcaatgccca ggcag
gggcaggtggccaccgtcctctctgccccgccaaaatcaccaaccatctggtagcgtatgattga aaaaac
cattagcggctcaggtatgctttaccaatatcagcgatgccgaacgtattttgccgaactctg acgggactcg
ccgcgcccagccgggatttccgctggcacaattgaaaactttcgtcgaccaggaatttg cccaataaaaac
atgtcctgcagtgcatcagttgttggggcagtgcccgatagcatcaacgctgcgct gatttgcctggcgca

gaaaatgtcgatgccattatggccggcgtgtagaagcgcgtgggcacacgtt accgttatcgatccgggtc
gaaaaactgctggcagtgggcattacctgaatctaccgttgatattgctg aatccaccgcgcgtattgcgg
caagccgcattccggctgaccacatggctgctgatggctggttactgc cggtaATGAAAAAGGC
GAGCTGGTGGTTCTGGGACGCAACGGTTCGACTACTCCGCTGCG
GTGCTGGCG GCCTGTTTACGCGCCGATTGTTGCGAGATCTGGACG
GATGTTGACGGTGTTTATACCTGCGATCCGCGTC

AGGTGCCCCGATGCGAGGTTGTTGAagtcgatgtcctatcaggaagcgatggagct
ttcttactcggcgc taaagttcttcccccgaccattacccccatgccagttccagatccctgctgat
taaaaatacc ggaatccccaagcaccaggtacgctcattggcgccgctgatgaagacgaattaccgg
tcaaggcca

Берілген үшеуінің тек екіншісі ғана белгілі геном болып табылады.

Бөлім 1: ORF-терді іздеу

Осы жобаның бірінші бөлімінде, біз ұзын ДНҚ тізбегінен тұратын мәтіндік файлды оқимыз, осы тізбекті тасбақа графикасының терезесіне қояйық, сосын ашық оқу жиектемелерінің орнын көрсету үшін, тізбек бойынша түстер кезектесетін интервалдармен жолақтар салайық. Төмендегі суретте терезенің тек бір ғана бөлігі көрсетілген, үш жолақ төмендегі 0 оқу жиектемесімен үш тікелей оқу жиектемелерін білдіреді. Бастаушы және терминациялаушы кодондардағы көк жолақтардың қалай басталып аяқталатын түсіну үшін суретке (оң жағындағы әрі кетіп бітетін жоғарғы жолақтан бөлек) мұқият қараңыз. Екі қысқарақ келген көк жолақ іс жүзінде қысқа ұзындықтары үшін шынайы ген боа алмайды.



Жұмысты бастап, жобаңызды ұйымдастыру үшін Сіз кітаптың веб-сайтынан бағдарламаның “қаңқасын” жүктей аласыз. Бағдарламадағы viewer функциясы тасбақа графикасының терезесін қосады, төменгі бөлікке ДНҚ тізбегін жазады (бір символды бір x координатасына), сосын Сіз жазатын екі функцияны шақырады. main функциясы ұзын ДНҚ тізбегін файлдан жолдың айнаымалысына оқиды, сосын viewer-ді шақырады.

Ашық оқу жиектемелірін көрсету үшін Сіз функция жазасыз

orf1(dna, rf, tortoise)

ол tortoise деп аталатын тасбақаны қолданып, dna жолындағы rf (0, 1, немесе 2) ығысқан тікелей оқу жиектемесіндегі ашық оқу жиектемелерін білдіретін түрлі түсті жолақтар салады. Бұл функция rf әртүрлі мәндерге, үш оқу жиектемелерін салу үшін үш рет шақырылады. Бағдарламаның қаңқасында түсіндірілгендей, сурет салу функциясы жазылып болды; Сіз оны шақырмастан бұрын тек қана қажет болғанда түстерді өзгертуіңіз керек. Еске түсіру: Сіздің for циклыңызды inORF-тің бульдік айнымалысын пайдаланып, Сіз қазір ORF-те екендігіңізді бақылайсыз.

Сонымен бірге оқулық сайтында E. coli-дің белгілі бір штаммы геномының префикстерінің әртүрлі көлемдерінен тұратын файлдар берілген. Кішірек файлдардан бастаңыз.

6.2.1-тапсырма *Ықтимал ген ретінде қаралатындай, ашық оқу жиектемесінің ұзындығы қандай болу керек?*

6.2.2-тапсырма *E. coli геномының алғашқы 10 000 негіздерінің арасындағы ықтимал гендер қайда орналасқан?*

Бөлім 2: GC құрамы

Белгілі бір ДНҚ аймағының GC құрамы - C мен G негіздері санының жалпы негіз санына қатынасты білдіреді. Мысалы, келесі тізбекте

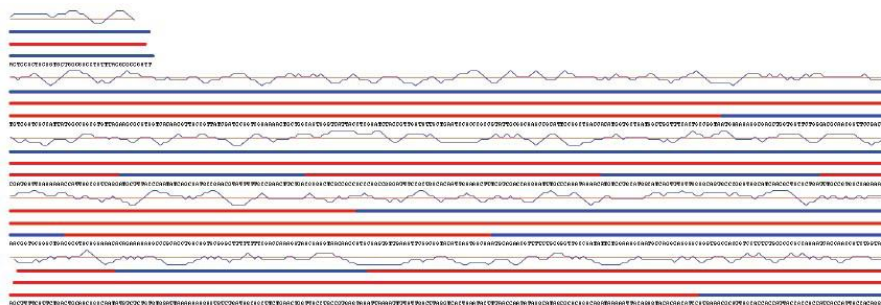
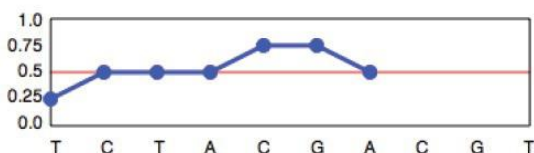
TCTACGACGT

GC құрамы - 0.5, себебі негіздің 5/10 не C, не G болып табылады. Әдетте GC құрамы кодтаушы тізбектердің аймағында жоғары болатындықтан, бұл да геннің орналасқан жеріне бағдар бере алады. (Бұл іс жүзінде E. coli.. тәрізді бактерияларынан гөрі омыртқалыларға тән). Ұзын тізбектерде GC құрамы белгіленген көлемдегі “терезелерде” өлшенеді, бұл 1.3 тарауда талқыланған мысалға ұқсас. Мәселен, жоғарыда берілген мысалда, егер GC құрамын 4 өлшемді терезеде өлшейтін болсақ, нәтижесіндегі арақатынас мынадай болады

TCTACGACGT

TCTA	0.25
CTAC	0.50
TACG	0.50
ACGA	0.50
CGAC	0.75
GACG	0.75
ACGT	0.50

Оны сосын графикке былай түсіреміз:



Сурет 6.12 Қорытынды нәтиже.

Мына функцияны аяқтаңыз
gcFreq(dna, window, tortoise)

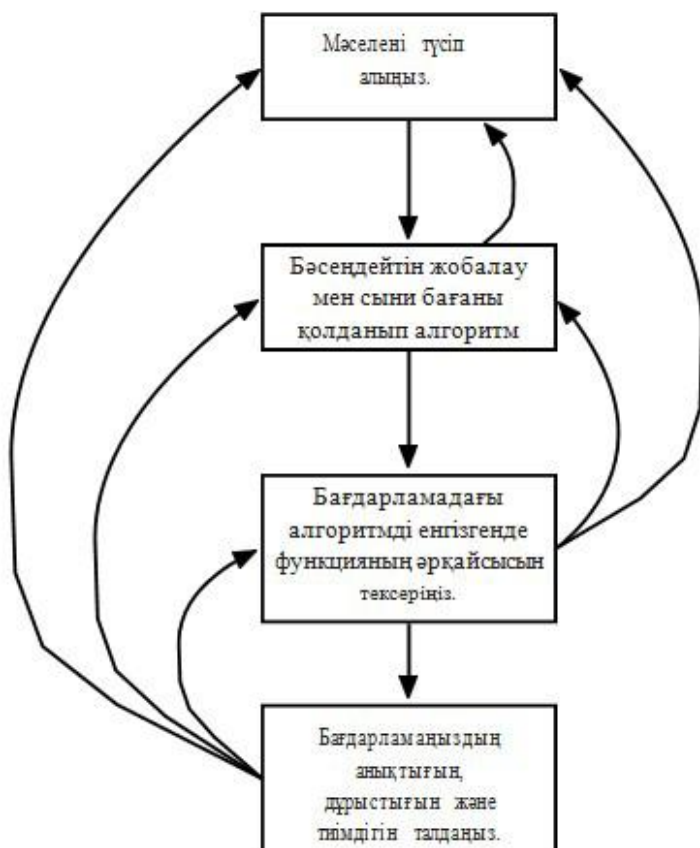
Ол dna жолында Gc-дің шығу жиілігін window көлемдегі терезелерден tortoise атты тасбақаның көмегімен құрайды. Осыны ORF жолақтары тұрған терезеге құраңыз. Қаңқа кодта түсіндірілгендей,

график құрайтын функция жазылып бітті; Сізге тек GC-дің дұрыс үлестерін есептеу керек. 1.3-тарауда талқылағанымыздай, Сіз GC құрамын әр терезе үшін қайтадан есептемейсіз. Бірінші терезе үшін G мен C негіздерін санап болған соң, Сіз бұл есептеуішті келесі терезелер үшін өзгерте аласыз. Қорытынды көрсетілетін нәтиже 6.12-суреттегідей болу керек.

7 ТАРАУ

Бағдарламаларды әзірлеу

Бағдарламамыздың дұрыс жұмыс істеуін қалайтынымыз туралы айтудың да қажеті жоқ. Яғни, біз алгоритмдеріміздің кез келген кіріс мәліметтері үшін дұрыс нәтиже бергенін, ал бағдарламаларымыз алгоритмдердің түпкі ойын дұрыс жеткізгенін қалаймыз. Дегенмен практикада, өкінішке орай, толығымен дұрыс жұмыс істейтін бағдарламалар жоқ деуге болады. Бағдарламалық жасақтаманың айтарлықтай әрбір маңызды бөлігі өзінің күрделілігіне орай қателерден немесе “бағтардарн” тұрады. Алайда біздің функцияларымыз және бағдарламалардың дұрыс жұмыс істейтіндігінің *сәтті ықтималдығын* арттыруға мүмкіндік беретін әдістер бар. Біріншіден, біз алдымызға қойылған мәселені толық түсініп алуымыз керек, сосын қандай да бір кодты енгізуден бұрын мәселенің шешімін табу арқылы уақытты пайдасымен өткізе аламыз. Екіншіден, біздің кіріс мәліметтер мен функция параметрлеріне қойылатын талаптарды мұқият қарап, сосын осы талаптарды ескеріп, кейінгі проблемаларға жол бермейміз. Үшіншіден, жанжақты тестілеу стратегиясын қолданып, бағдарламаларымыздың жұмысының дұрыстығын тексеріп біле аламыз. Осы тарауда, біз аталмыш үш практиканың негізін ұсынамыз және Сізден оларды кейін де ұстануыңызды сұраймыз. Бұл жүйеленген практикаларға сәйкес жүру арқылы қателерді болдырмай және бағдарламаңызда табылған қателерді оңай табу арқылы кейін көп уақытыңызды үнемдей аласыз.



7.1- сурет Мәселені шешу үдерісінің сұлбасы.

7.1 ЕСЕПТІ ҚАЛАЙ ШЕШУ КЕРЕК

Есепті шешу бағдарламасын әзірлеп енгізу - жоспарлау, тұтыну және қайта қараудан тұратын көп сатылы үдеріс болып табылады, бұл жазу үдерісіне ұқсайды. Бұл әсіресе көптеген адам қатысатын, әдістемелік нұсқамалар бекітілетін және үдерістер ресімделетін, *бағдарламалық инженерия* деп аталатын салаларға қатысты үлкен жобаларға тән. Осы кітапта, біз салыстырмалы шағын тапсырмаларды орындаймыз, сондықтан оларды ресімдеудің онша қажеті жоқ. Дегенмен, 7.1- суретте көрсетілген үдерістегі пайдалы дағдыларды дамыту өте маңызды.

Аталмыш үдеріс пен осы тараудың атауы математик ғалым Джордж Пойаның *Есепті қалай шешу керек* атты кітабынан алынды [41]. Олар алғаш рет 1945 жылы Пойаның есептерді шешу қағидалары уақытпен сыналды. Пойа үдерісті төрт сатыға бөлді:

1. Мәселені түсініп алыңыз. Не нәрсе белгісіз [қорытынды нәтиже]? Не берілген [кірістік мәліметтер]? Шарттар қандай [кірістік мәліметтер мен қорытынды нәтиженің байланысы]?

2. Есепті шешу жоспарын әзірлеңіз.

3. Әр қадамды тексеріп, жоспарды жүзеге асырыңыз.

4. Кейін қараңыз. Нәтижені тексеріңіз. Басқаша нәтижені алуға бола ма?

Кейбір өзгерістер енгізіген төрт қадамды есептеу тапсырмаларын шешу үшін қолдануға болады. Аталмыш тарауда, біз тек осы төрт қадамды ғана алып, үдерісті келесі есептің шешілу мысалында көрсетеміз.

Берілген мәтінде біз белгілі бір сөздің таралуын өлшеу үшін мәтінді біршама ұзындықтағы “терезелерге” бөліп, сөздің әр терезеде неше рет шығатынын есептегіміз келеді. Нәтижесінде әр терезеде шығатын осы жиіліктің, сондай ақ терезедегі шығуының орташа жиілігінің графигін аламыз.

Мәселені ұғыну

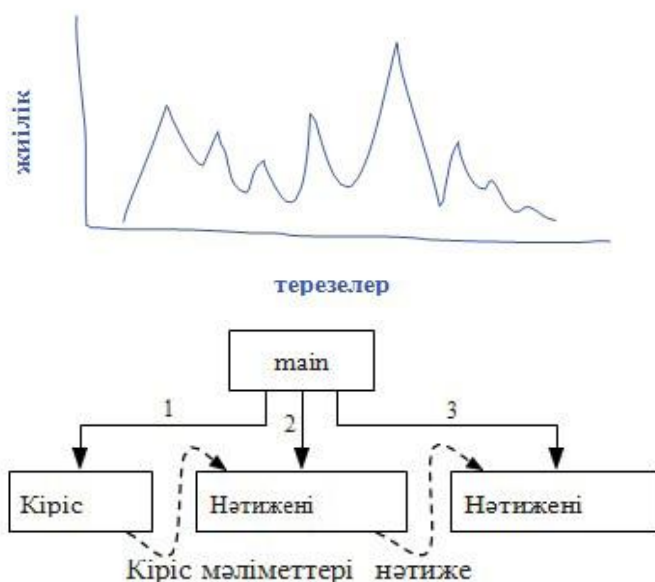
Аталмыш кітаптың алғашқы беттерінде талқылағанмыздай, пайда беретін есеп кірістік мәліметтер мен қорытынды нәтиже үшін шешім береді. сәйкесінше, бірінші қадам - мәселенің кірістік мәліметтерін, болжалды нәтижелер мен олардың арасындағы байланыстарды ұғыну. Сонымен қатар, біз кіріс мәліметтер немесе нәтижелерге қатысты қандай да бір шектеулер немесе арнайы шарттар бар ма деген сұрақ қоюымыз керек. Егер бір жерлері түсініксіз болса, бірден анықтамасын іздеңіз; қате мәселені шешіп, көп уақыт жұмсағанша, оны кідірместен бірден орындаңыз! Осындай жағдайда мәселені қателік шықпастан бұрын жазбаша не ауызша түрде қайтадан түсіндірген дұрыс. Осы жаттығудан алатын жауап, түсініспеушілік нүктелерін анықтауға көмектеседі. Тапсырманың түріне қарай суреттерді де қосуға болады. Сонымен бірге, барлық қосылған талаптар мен егжей-тегжейін түсінетіңізді тексеру үшін бірнеше мысалды қолыңызбен жасап көріңіз.

Біздің тапсырмамыздағы кіріс мәлімет - мәтін, сөз және терезе ұзындығы болып табылады. Нәтижелері - мәтіннің әр терезесіндегі

сөздің шығу жиілігі мен терезедегі орташа шығу жиілігінің графигі болып табылады.

Ойлануға берілетін сұрақ 7.1 бұл тапсырманы жақсы түсіну үшін кіріс мәліметі мен нәтиже туралы тағы не білуіміз керек?

Мәтінді қайдан алу керек? Сөз бен терезе ұзындығын сұрау керек пе? Терезелер қиылысу керек пе? Графиктің түрі қандай болу қажет? Үлкен мәтіндерді талдау үшін, мәтінді файлдан оқиық. Алайда файл атауын, сөзде және терезе ұзындығын сұрауға болады. Жеңілдету үшін, біз қиылысатын терезелерді пайдаланбаймыз. Төменде керекті графиктің үлгісі берілген:



7.2-сурет Жалпыланған бәсендеуші жобалау.

Мәтін бойындағы терезелер x-осінде, әр терезенің бастапқы индексі түрінде берілуі мүмкін. y-осі әр терезедегі сөздің кездесу санына сәйкес болады.

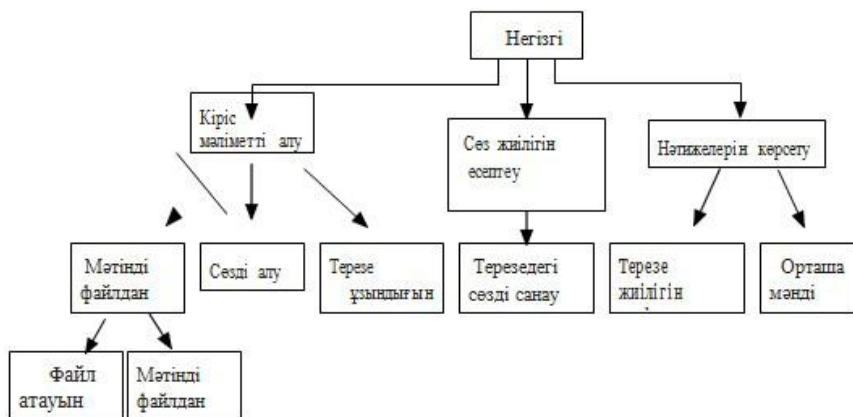
Алгоритм құрыңыз

Тапсырманы қалай шешу керектігін түсініп алған соң, келесі қадам - алгоритм әзірлеу болып табылады. Бұл үдерістегі ең маңызды бөлім - шағын тапсырма бөліктерін анықтау болып табылады, олардың әрқайсысы өз функциясымен шешіледі. Осы функцияларды жазған соң, оларды функционалдық абстракциялар ретінде тапсырманы толық шешуде қолдануға болады. Оның мысалын біз 3.7- суреттегі гүлді салатын бағдарламада көрген болатынбыз. Бұл тәсілді бәсеңдеуші жобалау дейміз, ол шешілетін тапсырмадан бастайды да, ол тапсырманы өте кішкентай бөліктерге бөледі. Осы бөліктердің әрқайсысы тағы кішірек бөліктерге бөлінеді, осылай анық шешілетін тапсырма бөліктеріне жеткенше жалғастырамыз. 7.2- суретте бәсеңдеуші жобалаудың жалпы көрінісі берілген. Жалпыланған бағдарлама үш негізгі тапсырма бөліктеріне (яғни, функцияларға) бөлінген. Бірінші, функция кіріс мәліметтерді алып кетеді. Екінші, функция кіріс мәліметтен нәтижені есептейді. Үшінші, келесі функция нәтижені көрсетеді. Нүкте сызықтар қайтарылатын мәндер мен параметрлер арқылы келетін ақпаратта білдіреді. Кіріс мәліметтерін алып кететін функция оларды негізгі бағдарламаға қайтарады, сосын ол оларды нәтижені есептейтін функцияға жібереді. Бұл функция сосын нәтижені функцияға жіберетін негізгі бағдарламаға қайтарады, сосын ол сонда көрсетіледі. Python-да бұл жалпыланған бағдарлама мынадай болады:

```
def main():  
    input = getInput()  
    output =  
    compute(input)  
    displayOutput(output)
```

```
main()
```

Осы жалпыланған құрылымнан әрі қарай жасалатын қадам - берілген үш тапсырма бөліктерін нақты бір тапсырмаға тәуелді тағы кішірек тапсырма бөліктеріне бөледі. Осы тапсырма бөліктерін/функцияларды анықтау - өнер немесе ғылым тәріздес, алайда бірнеше кеңесті еске сақтап алыңыз:



7.3-сурет Сөз жиілігі проблемасына арналған бәсеңдеуші жобалау.

1. Тапсырма бөлігі/функция салыстырмалы түрде шағын бір әрекетті орындау керек және өзінің мағынасы болу керек.

2. Функцияларды жиі және әртүрлі аргументтермен шақырылуы мүмкін тапсырма бөліктері үшін жазу керек.

3. Функция жалпы алғанда бір паракқа сыйуы керек немесе көп жағдайда одан аз болады.

4. main функциясы қысқа болу керек, әдетте ол бағдарламаны қосады да басқа функцияларды шақырады, олар болса жұмысты орындайды.

Сіз осы үдеріс барысында өзіңізді кері бірінші қадамға қайтаратын тапсырма туралы жаңа сұрақтарды байқауыңыз мүмкін. Бұл керемет: оны үдерістің соңына қалдырмай, бірден анықтап алған жақсырақ. Тапсырманы шешу үдерісі, жазу үдерісі тәрізді, сипаты жағынан циклды әрі ағымды, бұл 7.1 -суретте көрсетілген.

Ойлануға берілетін сұрақ 7.2 Қойылған тапсырманы шешу үшін алгоритм қандай қадамдар жасау керек?

Біріншіден, біз жалпыланған құрылымды жекелей аламыз, ол үшін негізгі үш тапсырма бөлігі не істеу керектігін көрсету керек.

1. Кіріс мәліметтерін алу: Файл атын, сөзді және терезе ұзындығын

сұрау. Мәтінді файлдан оқу.

2. Есептеу: Сөз жиілігінің тізімі мен сөз жиілігінің орташа саны.

3. Нәтижелерді көрсету: Жиілік графгін құру және экранға шығу жиілігінің орташа санын шығару.

Осы тапсырма бөліктерінің әрқайсысы бірнеше бөліктен тұратындықтан, біз әрбір тапсырма бөлігін әрі қарай бөлшектей аламыз:

1. Кіріс мәліметтерді алу.

(a) Мәтінді алу.

i. Файл атын сұрау және алу.

ii. Осындай файлдан мәтінді оқу.

(b) Сөзді сұрау және алу.

(c) Терезе ұзындығын сұрау және алу.

2. Сөздердің шығу жиілігінің тізімін және сөздің шығу жиілігінің орташа мәнін есептеу.

(a) Әр терезедегі сөздің шығу жиілігін есептеу.

3. Нәтижелерді көрсету:

(a) Жиіліктер графигін құрастыру.

(b) Экранға жиіліктің орташа мәнін шығару.

7.3 суретте бұл сұлба визуалды түрде көрсетілген. Біз бастапқы тапсырма бөлігін үш бөлікке бөлдік, әрқайсысы бізге қажетті кіріске арналған. Мәтінді алу бөлігі тағы файл атын алу және осы файлдан мәтінді оқу бөліктеріне бөлінген. Келесі есептеу бөлігінде бізге жасалатын терезенің әрқайсысындағы сөздерді есептеу керек, сондықтан осы терезелердің әрқайсысында сөз санайтын тағы бір функцияны жасау қисынды. Соңында, біз нәтижені шығару бөлігін екі бөлікке бөлеміз, біріншісі - графикке және екіншісі - жиіліктің орташа мәніне арналады.

Сосын бәсеңдеуші жобалауымызды бағдарлама ретінде іске қосудан бұрын бізге әрбір “бейтривияльді” тапсырма бөліктерінің әрқайсысына арнап алгоритм жасау керек. Кірістік мәліметтерді алу және нәтижені шығару тапсырма бөліктері айтарлықтай жеңіл болғанымен, негізгі тапсырма бөлігі ондай оңай емес. Осы сатыда алгоритмімізді *жалған кодта* жартылай ресми түрде жазып алған жөн. Жалған кодтың белгілі бір анықтамасы жоқ, бірақ ол ағылшын тілі мен бағдарламалау тілі ортасындағы сөз болып табылады. Біз жалған кодта алгоритм жазған кезде, бағдарламалау тілінде не орындалып, не орындалмайтыны жайлы білімімізге сүйенген жөн, алайда мұнда әрбір нәрсені жазып отырудың қажеті жоқ. Мысалы, әрбір терезедегі сөз жиілігін табатын алгоритм

үшін біз бұл терезелерді қандай да бір тәсілмен циклді қолданып іріктеу керектігін білеміз, алайда бізге дәл қазір циклдың формасы нақты қандай болатынын шешудің қажеті жоқ. Төменде Python-тәрізді жалған кодта жазылған ықтимал алгоритмдердің бірі берілген. Біз оны әдейі көк түске бояп, оның шын мәнінде Python функциясы емес екендігін ескерттік.

```
getWordCounts(мәтін, сөз, терезе_ұзындығы)
    ортақ_сөз_есептеуіші= 0
    ортақ_терезе_есептеуіші= 0
    сөз_есептеуіші_тізім
    i= []
    Әрбір қиылыспайтын терезе
        үшін: Үдемелі
            ортақ_терезе_есептеуіші
            есептеуіші= терезедегі сөз
            кірісінің саны есептеуішті
            ортақ_терезе_есептеуішіне
            қосу есептеуішті
            сөз_есептеуіші_тізіміне қосу
    Қайтару (ортақ_сөз_есептеуіші/ жалпы_терезе_есептеуіші) және
    сөз_есептеуіші_тізімі
```

Алгоритмде, біз үш негізгі мәнді қолдаймыз. Табылған жалпы сөздер кірісінің саны (жалпы_сөз_есептеуіші) және біз өңдеген терезелердің жалпы саны (жалпы_терезе_есептеуіші) терезеде табылған орташа санды есептеу үшін керек. Әр терезе үшін табылған сөздер санының мәндері сақталатын табылған сөздер саны тізімі (сөз_есептеуіші_тізімі) график құру үшін қажет. Цикл барлық терезелерді іріктейді де, олардың әрқайсысына осы терезедегі сөздердің кірісін табады да, оны қолданып, табылған сөздердің жалпы санын және табылған сөздер санының тізімін жаңартады. Соңында, біз табылған сөздердің орташа санын және табылған сөздер санының тізімін қайтарамыз.

Алгоритмді талдау Пойаның төрт сатылы үдерісінің соңғы амалы болғанымен, біз іс жүзінде оны алгоритмдеріміздің тиімділігін, анықтығы мен дұрыстығын талдау үшін күтуді қажет етпейміз. Әр сатыда, біз өз жұмысымызды сынап, оны жақсарту жолдарын іздеу керекпіз. Мысалы, егер біздің терезелеріміз қиылысқан болса, онда 1.3-тараудағыдай жоғарыда келтірілген алгоритмді әр терезедегі сөз санауды алдыңғы

терезедегі саналған мәнге негіздеп әлдеқайда тиімді етер едік. Назар аударыңыз, қандай да бір кодты іске асырмастан бұрын, осы жақсарту амалын орындауға болады.

Алгоритміңізді бағдарламада іске асырыңыз

Бағдарлама мен функционалдық жобасы аяқталғанда ғана, Сіз өз бағдарламаңызды жазуды бастай аласыз. Бұл пікірді елемеуге болмайды. Бастаушы бағдарламашылар жиі алдымен пернетақтаға отыруға асығады, сосын тек бірнеше сағат жұмыс істеп болғаннан кейін ғана олар іс жүзінде мәселені түсінбейтіндігін және /немесе оны қалай шешу керектігін түсінбейтіндігін білдіреді. Алайда бағдарламалауға неғұрлым тез кіріссең, соғұрлым уақыт үнемделетіндей көрінгенімен, ақырында оған керісінше жағдай болады. Тәжірибелі бағдарламашылардың жағдайлары қатарында алдымен өте ұзын бағдарламалардағы нақты бір код фрагментінің түпнұсқасын жасап алады да, жаңа идеялардың жүзеге асырылу мүмкіндігін тексереді. Дегенмен, осы кітаптағыдай шағын бағдарламалар жазатын бастаушы бағдарламашылар үшін, код жазудан бұрын, әрқашан шешу жобасын алдымен қағаз бетінде бастаған дұрыс.

Сіз бағдарлама жазарда алдымен функцияларды, сонымен бірге main функциясын, құжаттандыру жолдарын қоса орналастырып алуыңыз керек. Python интерпретаторы функцияда нұсқаулар болмайтын жағдайда шағымданатындықтан, алдымен өз функцияларыңыздың денесінде “жалған” pass нұсқауын орналастыруыңызға болады. Жобамызда, біз кірістіретін барлық сұраныстарды бір функцияға топтастыруды шештік, себебі бұл өте аз ғана кодты талап етеді. Сонымен қатар, біз терезедегі орташа табылған санды экранға шығаратын бөлек функция жасамадық, себебі бұл тек бір ғана жол болады.

```
def getInputs():  
    """Мәтіндік файл атауын, талданатын сөзді және терезе  
       ұзындығын сұрайды. Мәтін, сөз және терезе ұзындығын  
       оқиды және қайтарады.
```

Параметрлер: none

Қайтарылатын мән: мәтін, сөз және терезе
ұзындығы """

pass

```

def count(text, target):
    """Мәтіндегі мақсатты жолдарды санайды.

    Параметрлер:
        text: Жол нысаны
        target: Жол нысаны

    Қайтарылатын мән: Мәтіндегі мақсатты
    жолдың кіріс саны"""

    pass

def getWordCounts(text, word, windowLength):
    """Мәтіндегі әр терезедегі сөздердің кіріс санын іздейді.

    Параметрлер:
        text: Мәтіннен тұратын
        жол word: жол
        windowLength: Терезе ұзындығының бүтін мәні

    Қайтарылатын мәндер: Әр терезедегі орташа саны мен әр
    терезе үшін табылған санның тізімі"""

    pass

def plotWordCounts(wordCounts):
    """сөздің шығу жиілігі тізімінің графигін құрайды.

    Параметр:
        wordCounts: Сөздің шығу жиілігінің тізімі

    Қайтарылатын
    мәндер: None"""

    pass

def
main():
    pass

    # getInputs шақыру
    # getWordCounts шақыру
    # plotWordCounts шақыру
    # экранға терезедегі табылған санның орташа мәнін
    шығарады

main()

```

7.2-тарауда біз Сіздің параметрлеріңіз бен қайтарылатын мәндерге қойылатын талаптарды ұғыну және қолдану үшін едәуір ресми әдісті талқылайтын боламыз. Параметрлерге қойылатын талаптар алғышарттар деп, ал қайтарылатын мәндерге қойылатын талаптар соңғышарттар (немесе жағымсыз әсер) деп аталады. Мысалы, `getWordCounts` функциясында біз `text` пен `target` жолдар деп аламыз, ал `windowLength` - оң бүтін сан болады, бірақ біз оларды тексермейміз. Егер біз байқаусыздан теріс санды `windowLength`-ке жіберіп қоятын болсақ, онда қате шығуы әбден мүмкін. Мұның алғышарты - ресми түрде `windowLength` оң бүтін сан деп белгілейміз және функция олай болмаған жағдайда жұмыс істемейтіндігін тексереміз.

Функцияларды орналастырып болған соң, Сіз оларды бірінен соң бірін іске асыруға кірісе аласыз. Жұмыс барысында кодыңызға шолу жасаңыз. Функциямен жұмыс жасаған кезде оны

`Main`-нен шақыру арқылы және бағдарламаны іске қосу арқылы тексеріңіз. Осылайша, Сіз кодыңыздың жұмыс істейтіндігіне және онда синтаксис қателіктері жоқ екендігіне әрқашан сенімді бола аласыз. Егер Сіз бағдарламаны өте жиі іске қосатын болсаңыз, онда синтаксистік қателердің қай кезде туындайтындығын оңай анықтай аласыз, себебі олар соңғы жазылған бірнеше жолда болады.

Функцияңыздың жұмыс істейтіндігіне толық сенімді болсаңыз, Сіз одан әлдеқайда ресми түрдегі тестілеу үдерісіне кірісе аласыз. Бұл тақырып 7.3- тарауға қатысты. Сіз функцияның әрқайсысының жеке-кей дұрыс жұмыс істейтінін мұқият тексеріп алып қана, келесісіне өте аласыз. Ол жұмыс істейтініне көзіңіз жетсе, функция енді функциялық абстракцияға айналады да, Сіз енді ол жайлы ойламайтын боласыз, бұл жұмысыңызды оңайлата түседі! Функцияңыздың әрқайсысы дұрыс жұмыс істесе, онда бағдарламаны толық жинақтауға кірісуіңізге болады.

Мысалы, `getInputs` функциясын іске асырудан бастайық:

```
def getInputs():
    """ (құжаттау жолдары түсірілген) """

    fileName = input('мәтіндік файл атауы: ') # файл
    атауын алып кетеді

    textFile = open(fileName, 'r', encoding = 'utf-8') #
    text файлын оқиды = textFile.read()
    textFile.close()
```

```

word = input('сөз: ') # сөзді алып
кетеді windowLength = int(input('терезе ұзындығы: '))
# терезе
ұзындығын алып кетеді

return text, word, windowLength

```

Енді функцияның өзін келесіге өтпестен бұрын тексеру керек. main -нен функцияны шақырамыз да, жай ғана экранға нәтижелерді шығарамыз да, оның дұрыс жұмыс істейтінін тексереміз.

```

def main():
    testText, testWord, testLength = getInputs()
    print(testText, testWord, testLength)

main()

```

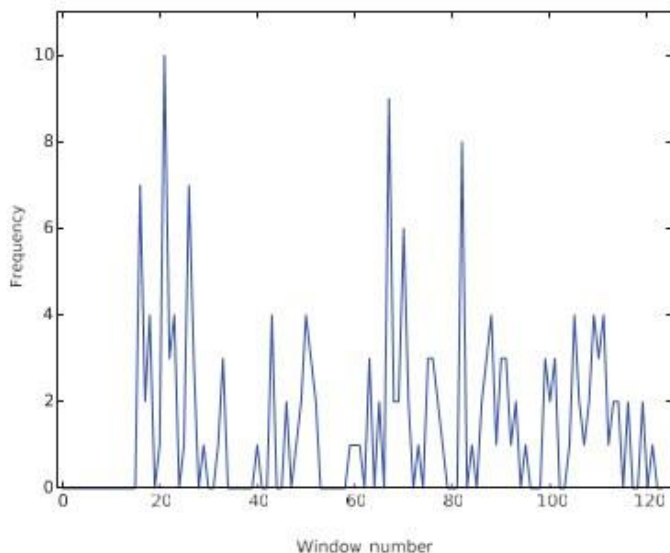
Егер синтаксистік қателер болса немесе күтпеген жайт орын алса, оны бірден түзетіп алу қажет. Бұл үдеріс қалған үш функцияның әрқайсысы үшін жалғасады. Содан кейін ақырғы main функциясы олардың барлығын байланыстырады:

```

def main():
    text, word, windowLength = getInputs()
    average, wordCounts = getWordCounts(text, word,
    windowLength) plotWordCounts(wordCounts)
    print(word, 'орташа алғандағы шығатын саны', average,
    'рет терезеде.')

```

Сіз бағдарламаның дайын нұсқасын кітап веб-сайтында таба аласыз. 7.4 -суретте түзілген бағдарламаның графигі көрсетілген.



7.4-сурет Моби Дик кітабындағы мәтінде “Requod” сөзінің шығу жиілігін тарату графигі (терезе ұзындығы = 10,000).

Бағдарламаңыздың түсініктілігін, дұрыстығын және тиімділігін талдаңыз

Бағдарламаңызды аяқтап болған соң, енді жалпы жұмысыңыз туралы ойлану уақыты келеді.

- Бағдарламаңыз дұрыс істей ме?

Әр функция үшін істегеніңіздей, бағдарламаңызды әртүрлі кіріс мәліметтерімен, әсіресе күрделі болып көрінетіндерін немесе бағдарламаның орындалуын бұзатындай етіп көрінетіндерін тексеріңіз. Мүмдіктердің барлық диапазонын білдіретін кіріс мәліметтерін тексеру маңызды. Мысалы, егер Сіздің кіріс мәліметтеріңіз сан болса, оң және теріс бүтін сандарды және де жылжымалы нүктесі бар сандар мен нөлдері бар сандарды да байқап көріңіз. Егер кейбір кіріс мәндер жарамсыз болатын болса, оларды анық етіп құжаттау жолдарында көрсетіп қою керек, біз ол туралы кейінгі тарауда талқылаймыз.

- Бағдарламаңыз шын мәнінде тиімді ме?

Бағдарламаңыз тым көп әрекет жасамай ма? Оны оңтайландыру жолы бар ма? Біздің бағдарламаларымыз айтарлықтай күрделі болып бара жатқандықтан, біз алгоритмдердің уақыт күрделілігін азайтудың

бірнеше мысалын қарайтын боламыз.

- Бағдарламаңыз расында жеңіл оқыла ма? Оған дұрыс пікір айтылды ма? Түсіріп қалдырып кететіндей артық нәрсе бар ма?

Бағдарламаны шолу мен құрылымдаудың негізгі қағидаларын 3.4- тараудан қараңыз.

- Функцияның оқылуын жақсарту үшін жасалатын код бөліктері бола ма?

Орындау барысында түзету қажеттіктері туралы шешім қабылдау үшін кері бұрынғы сатыларға қайта баруыңыз мүмкін. Тағы да қайталаймыз, онда тұрған ештеңе жоқ, керісінше бұл құпталады. Ешкім ешқашан бірінші реттен ақ керемет шешім жасап бере алмайды!

Әрі қарай бару үшін осы қадамдарды және 7.1-суретті есте сақтап алыңыз. Біз біртіндеп күрделі мәселелерді шешуге кірісеміз, ондағы жетістік осы қадамдарға тәуелді болады.

*7.2 КЕЛІСІМ-ШАРТТЫҚ БАҒДАРЛАМАЛАУ

Тапсырманы анықтаудағы кейбір тетіктерді елемей кету кейін үлкен проблема тудыруы мүмкін. Осы тарауда, біз кіріс мәліметтері мен нәтижелерге қатысты айтарлықтай қатаң әрі егжей-тегжейлі әдіс қолданып, алгоритмде барлығы теріс кеткендегі жағдайды талқылаймыз. Алдымен 6.3-тарауындағы кейбір шағын функцияларды қараудан бастаймыз, сосын алдыңғы тараудағы бағдарламаға қайта ораламыз.

Алғышарттар мен соңғы шарттар

Кіріс мәліметтерінің тапсырмасын түсіну дегеніміз - оларға қандай да бір шектеулер қойылғандығын ұғыну. Мысалы, 6.3-тарауындағы `digit2String` функциясын қарастырайық.

```
def digit2String(digit):
    """Бүтін санды цифрды сәйкес жол түріне
       түрлендіреді.

    Параметр:
        digit: 0, 1, ..., 9 ішіндегі бүтін сан

    Қайтарылатын мәндер:
        Сәйкес символ '0', '1', ..., '9' немесе
        None, егер аргумент ретінде цифр
        берілмесе
    """

    if (digit < 0) or
        (digit > 9):
        return None
    return chr(ord('0') + digit)
```

Бұл функцияның 0-9 дейінгі бүтін санды цифрді сәйкес символға түрлендіруге арналғанын еске түсіре кетейік. Мысалы, егер кіріс мәлімет - 4 болса, нәтижесінде '4' жол болу керек. Функцияны әзірлеу барысында біз 0-ден кіші немесе 9-дан үлкен мән жұмыс істемейтіндігін байқаған болатынбыз, сол үшін біз ондай жағдайларда None қайтардық. digit 0 мен 9 арасындағы бүтін сан болу керек деген талап - *алғышарт* деп аталады. Функцияның алғышарты дегеніміз - функцияны шақырғанда оның дұрыс әрекет етуіне қажетті ақиқат мән болып табылады. Басқа сөзбен айтқанда, біз алғышарт дұрыс болмаса, функция нәтижесінің дұрыс болуына кепіл бере алмаймыз.

Ал функцияның соңғы шарты дегеніміз - функция аяқталған кезде ақиқат болу керек мән болып табылады. Соңғы шарт әдетте функцияның қайтрылатын мәндерін және оның жағымсыз әсерлерін, егер бар болса, қайтарады. Жағымсыз әсер жаһандық ресурсты өзгертетін функциядағы жаһандық айнымалының немесе басқа құбылыстың өзгеруі кезінде болатын нәрсе екендігін еске сала кетейік. Мысалы, print бірнеше қайтара шақыруды және файлдарды өзгертуді - жағымсыз әсер деп есептеуге болады, себебі олар функциядан тыс аяларға ықпал етеді.

Ойлануға берілетін сұрақ 7.3 digit2String функциясының соңғы шарты қандай?

digit2String функциясының соңғы шарты функция өзінің кіріс мәліметтерінің жол түрін қайтарады деп айту керек.

Алғышарттар мен соңғы шарттарды қолдану келісім-шарттық бағдарламалау деп аталады, себебі алғышарт пен соңғы шарт әзірлеуші-функция мен клиент-функциясының арасында “келісім” орнатады. Клиент-функция алғышарттың функцияны шақырғанға дейін орындау керектігін түсінеді, ал әзірлеуші функция алғышарт орындалса, соңғы шарттың да орындалатынын кепілдейді.

Олар кіріс мәліметтері мен нәтижелерді, сәйкесінше функцияның қалай қолданылу керек екендігін суреттейтіндіктен, алғышарттар мен соңғы шарттар функцияның құжаттау жолына қосылуы керек. Егер олар параметрлер мен қайтарылатын мән туралы ақпараттан тұратын болса, оларды құжаттау жолдарының сәйкес бөліктеріне қоюға болады. Мысалы, digit2String үшін жаңа құжаттау жолы мынадай болады:

```
def digit2String(digit):  
    """Бүтін санды цифрді сәйкес жол түріне түрлендіреді.  
  
    Алғышарт: digit 0 мен 9 арасындағы бүтін сан болып  
    табылады. Соңғы шарт: Цифрдің жол түрін қайтарады,  
    немесе  
    None егер digit 0 мен 9 арасындағы бүтін сан  
    болмаса  
    """
```

Параметрлерді тексеру

digit2String функциясы егер digit 0 мен 9 арасында болмаса, алғышартты None қайтару арқылы жартылай “мәжбүрлейді”. Алайда біз жұмыс істеп көрмеген тағы бір алғышарттың бөлігі бар.

Ойлануға берілетін сұрақ 7.4 *Егер digit-ке жіберілген аргумент бүтін сан болмаса не болады?*

digit2String('cookies') шақыру келесі қателікті тудырады:
TypeError: unorderable types: str() < int()

Бұл қателік if нұсқауында, егер digit мәні берілген 'cookies' жағдайында 0-ге тең болса пайда болады. Осындай қателік туралы көмескі хабарламаны болдырмау үшін, бізге функцияның алдында digit бүтін сан екенін тексеріп алу керек. Олай instance функциясын пайдаланып істеуге болады, ол екі аргументті алып кетеді: Айнымалы (немесе мәннің) атауы және жіктелім атауы. Функция айнымалы не оның мәні жіктелім нысанына (яғни, данаға) қатысты болса True қайтарады, ал кері жағдайда False қайтарады. Мысалы,

```
>>> isinstance(5, int) True
>>> isinstance(5, float) False
>>> s = 'a string'
>>> isinstance(s, str) True
>>> isinstance(s, float) False
```

Біз if нұсқауындағы instance пайдаланып digit2String қатеге төзімдірек ете аламыз:

```
def digit2String(digit):
    """ (құжаттау жолдары түсірілген) """

    if not isinstance(digit, int) or (digit < 0) or
        (digit > 9): return None
    return chr(ord('0') + digit)
```

Ал енді осы функцияны есептеуде қалай қолдануға болатыны жөнінде ойланайық. Мысалы, 6.3.10 жаттығуы үшін шешімге digit2String қолдануға болады, ол арқылы оң бүтін санды value өзіне сәйкес жол түріне түрлендіреміз:

```
def int2String(value):
    """Бүтін санды өзіне сәйкес жол түріне түрлендіреді.
```

Параметр:

value: Бүтін сан

Қайтарылатын мән: value жол түрі

»»» int2String(1234567890)

```

intString = ''
while > 0:
    digit = value % 10
    value = value // 10
    intString = digit2String(digit) + intString
intString return intString

```

while циклдың әр итерациясында value-дің шеткі оң жақ цифры алынады да, value онға бөлінеді, цифр жолға digit2String арқылы түрленеді, ал нәтиже артылатын жол түрінің басына қосылады.

Ойлануға берілетін сұрақ 7.5 *Егер value - 123.4 тәрізді жылжыма-лы нүктесі бар сан болса не болады?*

Егер value - 123.4 болса, онда digit -тің бірінші мәні - 3.4 болады. Себебі ол бүтін сан емес, digit2String(digit) жауабы - None болады. Бұл, өз кезегінде, біз None-ді жолға қосқымыз келгенде қателік тудырады:

TypeError: unsupported operand type(s) for +: 'NoneType' and 'str'

Біз анық түрде digit2String арналған құжаттау жолында функция осылай әрекет ететіндігін көрсеткендіктен, келісім-шарттық бағдарламалау ережелеріне сәйкес, int2String функциясы, digit2String клиенті ретінде, осы мүмкіндікті аулауға және оны дұрыс орындау үшін жауапты. Әсіресе, digit2String - None-ді қайтаратындығын тексеру керек, егер қайтараса, онда сәйкес әрекет жасау керек. Мысалы, үдерісті жай ғана үзіп жіберуіп, None қайтаруға болады:

```

def int2String(value):
    """ (құжаттау жолдары түсірілген) """

    intString = ''
    while value > 0:
        digit = value % 10
        value =

```

```

value // 10
digitStr =
digit2String(digit) if
digitStr != None:
    intString = digit2Str +
intString else:
    return None
return intString

```

Ойлануға берілетін сұрақ 7.6 *value жылжымалы нүктесі бар сан болса функцияның None қайтару баламасы ретінде нені ұсынуға болады?*

Балама нұсқасы ретінде value бүтін бөлігін білдіретін жолды қайтаруға болатын еді. Басқа сөзбен айтқанда, егер value - 123.4 мәні меншіктелсе, онда '123' жолы қайтарылатын еді. Алайда бұл функцияның соңғы шартында нақты айтылуы керек. 7.2.1 және 7.2.2- жаттығуларда Сізден осы функцияға алғышарттар мен соңғы шарттар жазуыңызды, сосын функцияны барлық кіріс мәліметтерді дұрыс өңдейтіндей етіп өзгертуіңізді сұрайды.

Логикалық өрнектер

Дәлірек баламасы бұл - қателік жөнінде хабарламаны көрсетіп және егер параметр алғышартты бұзатын болса, барлық бағдарламаны тоқтату болыптабылады. Жоғарыда орын алған дәл осындай жағдай, Python интерпретаторы + операторы None және жолды тіркемекші болғанда TypeError көрсетіп, бағдарламаны үзіп жіберді. Осы үйреншікті ValueError және ValueError хабарламалар ерекшеліктер деп аталады. Кіріктірілген функция жұмыс дұрыс жүрмегенде және функция әрі жалғастырылмайтын кезде “ерекшелікті шақырады”. Егер функция ерекшелікті шақырса, ол әдеттегі қалпына қайта келмейді. Оның орнына, ерекшелік шақырылған кезде функцияның орындалуы аяқталады да, ал орындалу Python интерпретаторындағы бөлікте жалғасады, оны ерекшеліктерді өңдеуіш деп атаймыз. Шартты түрде, ерекшелікті өңдеуші қателік туралы хабарламаны шығарады да бүкіл бағдарламаны тоқтатады.

Функцияларымыз үшін сондай-ақ, TypeError мен ValueError ерекшеліктерін шақыруға болады, алайда олардың жұмыс істеу

қағидаларын зерттеу берілген кітаптың материалымен қамтылмайды. Оның орнына біз AssertionError деп аталатын тек бір ғана қарапайым ерекшелік түрін қараймыз, ол assert нұсқауы арқылы шақырылады. Assert нұсқауы логикалық шартты тексереді де, егер шарт жалған болса AssertionError шақырады. Егер шарт ақиқат болса, онда assert нұсқауы ештеңе істемейді. Мысалы,

```
>>> value = -1
>>> assert value < 0
>>> assert value > 0 AssertionError
>>> assert value > 0, 'мән оң болу керек' AssertionError: Мән оң болу
керек
```

Жоғарыдағы бірінші assert нұсқауы ештеңе істемейді, себебі шарттың мәні True болып тұр. Ал, екінші assert нұсқауындағы шарт мәні - жалған False болғандықтан, AssertionError ерекшелігі шақырылады. Үшінші assert нұсқауы сондай-ақ ілесуші AssertionError ақпараттық хабарламаны қосуға болатынын көрсетеді.

Біздің digit2String функциямыздағы if нұсқауын алдыңғы талқыланған қателік түрлерін аулау үшін басқа өрнектерге ауыстыруға болады:

```
def digit2String(digit):
    """ (құжаттау жолдары түсірілген) """

    assert isinstance(digit, int), 'digit бүтін сан болу
керек' assert (digit >= 0) and (digit <= 9), 'digit
[0..9] кіру керек'

    return chr(ord('0') + digit)
```

Егер digit бүтін сан болмаса, онда бірінші assert нұсқауы мынаны көрсетеді

AssertionError: digit бүтін сан болу керек

және бағдарламаны тоқтатып жібереді. Егер digit бүтін сан бол-
са (және, сәйкесінше, бірінші өрнек өтсе) алайда қажетті диапазонға
кірмесе, екінші assert нұсқауы мынаны көрсетеді

AssertionError: digit [0..9] кіру керек

және бағдарламаны тоқтатып жібереді.

Ойлануға берілетін сұрақ 7.7 *digit2String өзгертілген нұсқасын int2String функциясынан шақырыңыз. Егер int2String(123.4) шақырса енді не болады?*

Назар аударыңыз, assert нұсқауы бүкіл бағдарламаны тоқтатып жіберетіндіктен, оны тек қана басқа бір қолайлы әрекет тәртібі болмаған жағдайда қолдану керек. Алайда “қолайлы” сөзінің анықтамасы белгілі бір жағдайларға байланысты.

Басқа бір мысал ретінде, алдыңғы тараудағы count және getWordCounts функцияларын қарап көрейік. Біз онда бұл функцияларды жүзеге асырмадық, сондықтан олар төменде көрсетілген. Біз count функциясына алғышарт пен соңғы шарт және де алғышартты тексерін логикалық өрнек қостық.

```
def count(text, target):
    """мәтіндегі мақсатты жолдардың санын
    есептейді. Алғышарт: text пен target жол
    нысаны болып табылады
    Соңғы шарт: Мәтіндегі мақсатты жолдардың кіру
    санын қайтарады"""
    assert isinstance(text, str) and isinstance(target, str), \
        'аргументтер жол болу керек'
    count = 0
    for index in range(len(text) - len(target) +
        1): if text[index:index + len(target)]
        == target:
            count = count + 1
    return count
```

```
def getWordCounts(text, word, windowLength):  
    """Мәтіндегі әр терезедегі сөздердің кіріс санын іздейді.
```

Параметрлер:

text: Мәтіннен тұратын

жол word: жол

windowLength: Терезе ұзындығының бүтін мәні

Қайтарылатын мәндер: Әр терезедегі орташа саны мен әр терезе үшін табылған санның тізімі"""

```
wordCount = 0
```

```
windowCount =
```

```
0 wordCounts =
```

```
for index in range(0, len(text) - windowLength + 1,  
    windowLength): window = text[index:index +  
    windowLength]  
    windowCount = windowCount + 1  
    wordsInWindow = count(window,  
    word) wordCount = wordCount +  
    wordsInWindow  
    wordCounts.append(wordsInWindow)  
return wordCount / windowCount, wordCounts
```

Ойлануға берілетін сұрақ 7.8 *getWordCounts* функциясы үшін лайықты алғышарттар мен соңғы шарттар қандай? Алғышарт *text* немесе *word* мәндері үшін бірдеңе бекіту қажет пе? (Егер *text* тым қысқа болса не болады?)

Алғышарт *text* және *word* екеуі де жол деп және *wordLength* оң бүтін сан деп бекіту керек. Сонымен қатар, *text* мазмұнына ерекше назар ауда-

ру керек. Егер text ұзындығы windowLength-тен кем болса, онда циклда бір де бір итерация болмайды. Осы жағдайда, windowCount нөл мәнінде қалады, ал return нұсқауындағы бөлу амалы қателік береді. Жағдайларға тәуелді біз сонымен қатар word мәні үшін бірнеше шектеулер жаза аламыз, алайда, жалпы алғанда, бұл функцияның дұрыс істеуі үшін қажетті шарт емес. Осы талаптарды ескергендегі лайықты алғышарт мынадай болады:

Алғышарт: text және word жол нысандары болып табылады ал windowLength - оң бүтін сан және
text құрамында windowLength символдан кем емес

Алғышарт орындалды дейтін болсақ, соңғы шарт қарапайым болады:

Соңғы шарт: Сөздің шығу жиілігінің тізімін және терезедегі сөздің кірісінің орташа санын қайтарады

Алғышарттың талаптарына қандай да бір сәйкессіздіктер getWordCounts функциясын бұзады, сол үшін assert нұсқауының көмегімен талаптардың сәйкестілігін тексеріп көрейік. Алдымен, text пен word жол екендігін мына assert нұсқауының көмегімен тексереміз:

assert isinstance(text, str) and isinstance(word, str), \ 'терезе ұзындығы бүтін оң сан болу керек'

windowLength оң бүтін сан екенін мынаның көмегімен тексеруге болады:

assert isinstance(windowLength, int) and windowLength > 0, \ 'терезе ұзындығы оң бүтін сан болу керек'

windowLength оң бүтін сан екенін мынаның көмегімен тексеруге болады:

assert len(text) >= windowLength, \

'Терезе ұзындығы мәтін ұзындығынан қысқа болу керек'

Соңында, функцияның соңында дәл бөлуден бұрын қосымша өрнек қосу жөн көрінеді:

assert windowCount > 0, 'терезелер табылмады'

Алдыңғы өртнектеріміз осындай өрнектің шығуын болдырмау үшін, қауіпсіз бағдарламалау әрқашан артық көрінбейді. Осы өзгерістерді функцияға енгізу мынадай болады:


```
def getWordCounts(text, word, windowLength):
    """Мәтіндегі әр терезедегі сөздердің кіріс санын іздейді.
```

Алғышарт: text және word жол нысандары
 болып табылады ал windowLength
 - оң бүтін сан және
 text құрамында windowLength символдан
 кем емес

Соңғы шарт: Сөздің шығу жиілігінің тізімін және
 терезедегі сөздің кірісінің орташа санын
 қайтарады

```
.....
```

```
assert isinstance(text, str) and isinstance(word, str), \
    'терезе ұзындығы бүтін оң сан болу керек'
```

```
assert isinstance(windowLength, int) and windowLength
    > 0, \ 'терезе ұзындығы оң бүтін сан болу
    керек'
```

```
assert len(text) >= windowLength, \
    'Терезе ұзындығы мәтін ұзындығынан қысқа болу
    керек'
```

```
wordCount = 0
windowCount =
0 wordCounts =
:
for index in range(0, len(text) - windowLength + 1,
    windowLength): window = text[index:index +
    windowLength]
    windowCount = windowCount + 1
    wordsInWindow = count(window,
    word) wordCount = wordCount +
```



```

wordsInWindow
wordCounts.append(wordsInWindow)

assert windowCount > 0, 'терезелер
табылмады' return wordCount /

windowCount, wordCounts

```

Келесі тарауда біз осы функциялардың дұрыс деп табатын болсақ, үдерісті мұқият талқылаймыз.

Жаттығулар

7.2.1. int2String функциясына лайықты алғышарт пен соңғы шартты жазыңыз.

7.2.2. int2String функциясын if нұсқаулары арқылы Сіз алдыңғы жаттығуда жазған алғышарт пен соңғы шартқа дұрыс сәйкес келетіндей етіп және барлық ықтимал кіріс мәліметтерін дұрыс өңдейтіндей етіп өзгертіңіз.

7.2.3. int2String функциясын assert нұсқаулары арқылы Сіз 7.2.1-жаттығуда жазған алғышарт пен соңғы шартқа дұрыс сәйкес келетіндей етіп және барлық ықтимал кіріс мәліметтерін дұрыс өңдейтіндей етіп өзгертіңіз.

Алдыңғы тараулардағы келесі функциялардың әрқайсысына,

(а) лайықты алғышарт пен соңғы шарт жазыңыз, және

(б) алғышартыңыздың орындалуын қамтамасыз ету үшін өрнек нұқсалуын қосыңыз. Әрбір өрнек нұқсауын қателік туралы ақпараттық хабарламамен байланыстырыңыз.

7.2.4. def volumeSphere(radius):

return (4 / 3) * math.pi * (radius ** 3)

7.2.5. def fair(employee, ceo, ratio): return (ceo / employee <= ratio)

7.2.6. def windChill(temp, wind):

chill = 13.12 + 0.6215 * temp + \

(0.3965 * temp - 11.37) * wind ** 0.16 return round(chill)

Желді ескергендегі температура тек қана 10° С -ге тең не одан

төменгі температураларға және 4,8 км/сағ жылдамдықтағы жел үшін анықталғанына назар аударыңыз 4.8 km/h.

```
7.2.7.      def plot(tortoise, n):
    for x in range(-n, n + 1): tortoise.goto(x, x * x)
7.2.8.      def pond(years):
    population = 12000 print('Жыл      Популяция')
    print('{0:<4}      {1:>9.2f}'.format(0, population)) for year in range(1,
years + 1):
    population = 1.08 * population - 1500
    print('{0:<4}      {1:>9.2f}'.format(year, population))      return
population
7.2.9.      def decayC14(originalAmount, years, dt): amount =
originalAmount
    k = -0.00012096809434
    numIterations = int(years / dt) + 1 for i in range(1, numIterations):
    amount = amount + k * amount * dt return amount
7.2.10.     def argh(n):
    return 'A' + ('r' * n) + 'gh!'
7.2.11.     def reverse(word):
    newstring = ""
    for index in range(len(word) - 1, -1, -1): newstring = newstring +
word[index]
    return newstring
7.2.12.     def find(text, target):
    for index in range(len(text) - len(target) + 1): if text[index:index +
len(target)] == target:
    return index return -1
7.2.13.     def lineNumbers(fileName):
    textFile = open(fileName, 'r', encoding = 'utf-8') lineCount = 1
    for line in textFile:
    print('{0:<5} {1}'.format(lineCount, line[:-1])) lineCount = lineCount + 1
    textFile.close()
Файлдың бар-жоғын анықтау үшін мынаны қолданыңыз:
import os.path
assert os.path.isfile(fileName)
Сосын, файлдың оқылатынын тексеру үшін мынаны қолданыңыз:
import os
assert os.access(fileName, os.R_OK)
os.R_OK мәні функцияға файлдың дұрыс оқылатындағын (OK)
```

көрсетеді.

*7.3 ТЕСТІЛЕУ

Функцияны мұқият анықтап, жүзеге асырған соң, оны әбден тексеріп алу керек. Сіз осы уақытқа дейін функцияңызды тексерген болсарсыз, бірақ енді, біз күрделірек функцияларды жазып бастағандықтан, әлдеқайда есептірек тәсіл қолданудың уақыты келді.

Басқа функцияларға бармай тұрып, алдымен жазған функцияларымыздың әрқайсысын тексеріп алған өте маңызды. Бағдарлама жазу үдерісі көптеген итерациялардан тұру керек, яғни, әзірлеу—іске қосу—сынау, бұл 7.1-тараудағы 7.1-суретте көрсетілген. Жалпы құрылымды ойлап тауып, қандай функциялардың қажет екенін анықтап алған соң, Сіз жеке функцияның әрқайсысы үшін осы әзірлеу—іске қосу—сынау үдерісіне сәйкес әрекет етуіңіз керек. Егер осы кеңеске сәйкес жүрмей, Сіз жұмысты аяқтадым деп есептеп, оны бірден тексерсеңіз, онда қатенің қай жерде екенін анықтау өте қиын болады. Функцияның іс жүзінде дұрыс екендігін тексерудің екі әдісі бар - оны не математикалық дәлелдеу, не оны барлық мүмкін деген кіріс мәліметтері үшін (яғни, параметрдің барлық мүмкін деген мәндері үшін) тексеру. Алайда, бұл стратегияның екеуі де тәжірибе жүзінде мүмкін емес болғандықтан, түрлі жағдайларын есептемегенде, ең дұрысы бұл - функцияларымызды әртүрлі, мұқият таңдалған, барлық мүмкіндіктер диапазоны үшін репрезентативті болып табылатын кіріс мәліметтерімен сынау болып табылады. Бағдарламалық жасақтама әзірлейтін ірі компанияларда арнайы топтар болады, олардың міндеті тек қана - жеке функцияларды әзірлеу және сынау, жалпы функциялар мен бағдарламалық жасақтама арасындағы қарым-қатынастарды сынау болып табылады.

Модульдік сынау

Біз сынақтарымызды әрбір функцияға арнап топтастырамыз, ол модульдік сынақ деп аталады. Біздің жағдайымыздағы “модуль” жеке бір функция болады, ал жалпы белгілі бір мақсаты бар кез келген код блогі модуль бола алады. Модульдік сынақтың әрқайсысы `test_` деп аталатын функция болады.

Ол біз сынайтын функция алдында жүреді. Мысалы, біздің модульдік сынақ функциямыз `count` функциясы үшін `test_count` деп аталатын болады. Модульдік сынақ функциясының әрқайсысы бірнеше жеке сынақтан тұратын болады, олардың әрқайсысы `assert` болады, ол беліглі бір па-

раметрлер жиынтығы бар функцияны шақырады және дұрыс жауапты қайтарады.

Оны суреттеу үшін біз алдыңғы тараудағы төменде оңайлатылған түрде берілген count және getWordCounts функцияларына модульдік тестілер әзірлейміз. Біз құжаттау жолдарын және assert нұсқауларына ілескен хабарламаларды түсірдік, терезедегі шығау жиілігінің тізімін алып тастап getWordCounts біршама оңайлаттық және графигінің құрылуын алып тастап main функциясын оңайлаттық.

```
def getInputs():
    :   # код түсірілді

def count(text, target):
    assert isinstance(text, str) and isinstance(target, str)

    count = 0
    for index in range(len(text) - len(target) +
        1): if text[index:index + len(target)]
        == target:
            count = count + 1
    return count

def getWordCounts(text, word, windowLength):
    assert isinstance(text, str) and isinstance(word, str)
    assert isinstance(windowLength, int) and
    windowLength > 0 assert len(text) >= windowLength

    wordCount = 0
    windowCount = 0
    for index in range(0, len(text) - windowLength + 1,
        windowLength): window = text[index:index +
        windowLength]
        windowCount = windowCount + 1
        wordsInWindow = count(window,
        word) wordCount = wordCount +
        wordsInWindow

    assert windowCount > 0
    return wordCount / windowCount
```

```
def main():
    text, word, windowLength = getInputs()
    average = getWordCounts(text, word, windowLength)
    print(word, 'орташа шамамен шығады', average,
          'реттерезеде.')

main()
```

Біз бағдарламаға арналған барлық модульдік тестілерді бөлек файлға саламыз, бұл бағдарламалық файлымыздағы реттілікті сақтау үшін керек. Егер жоғарыдағы бағдарлама wordcount.py атты файлда сақталса, онда модульдік тестінің файлы test_wordcount.py деген атпен сақталады және құрылымы мынадай болады:

```
from wordcount import count, getWordCounts

def test_count():
    """count-ке арналған модульдік тест"""

    # мұнда count-ке арналған тестілер бар
    Print('count-ке арналған
барлық тестілер тапсырылды!')

def test_getWordCounts():
    """getWordCounts-ке арналған модульдік тест"""

    # getWordCounts-ке арналған модульдік тестілер осында
    print('getWordCounts-ке арналған
барлық тестілер тапсырылды!')

def test():
    test_count()
    test_getWordCount
    s()

test()
```

Бірінші жол біз сынайтын екі функцияны wordcount.py-дан тестілеу бағдарламасы атауының жаһандық кеңістігіне импорттайды. 3.6 тарау-

дан еске алатынымыз, әдеттегі `import` нұсқауы жаңа атаулар кеңістігін жасайды, онда импортталатын модульдегі барлық функциялар болады. Оған қарағанда `import` нұсқауының

```
from <module name> import <function names>
```

формасы функцияны ағымдағы атаулар кеңістігіне импорттайды. Оның басымдылығы - `count` және `getWordCounts` шақырған сайын олардың алдына модульдің атауын қою керек еместігінде. Басқа сөзбен айтқанда, біз

```
count('This is fun.', 'is') өрнегін
```

```
wordcount.count('This is fun.', 'is')
```

онына шақыра аламыз.

`import` нұсқауынан кейін бағдарламада модульдік тестілеу функциялары мен негізгі `test` функциясы жүреді, ол осы модульдік тестілердің барлығын шақырады.

Регрессивтік тестілеу

Бағдарламамызды сынау кезінде біз бөлек модульдік тестілердің функцияларының орнына `test()` шақыратын боламыз. Ыңғайлығымен қатар, бұл әдістің тағы бір жақсы жағы - Біз жаңа функцияларды тестілегенде, сондай-ақ алдында сыналған функцияларды қайта тестілей алатындығымызда. Егер біз бағдарламадағы кез келген функцияның біріне өзгеріс енгізер болсақ, онда біз бұл өзгерістердің жұмыс істейтінін *және де* бұрын жұмыс істеп тұрған нәрсені бүлдіріп алмағанымызды тексергіміз келеді. Бұл идеяны *регрессивтік тестілеу* дейміз, себебі бағдарламамыздың бұрынғы қате қалпына *регресстелмегендігін* тексереміз.

Шынайы модульдік тестілерді әзірлеместен бұрын, тағы бір техникалық мәселені талқылайық. Бұл жөнінде алдыңғы тарауларда осылай ойламасақ та, `import` нұсқауы басқа файлдан атауларды импорттап қана қоймай, *сондай -ақ* ол сол файлдың ішінен кодты да орындайды. Сондықтан да, `test_wordcount.py` функцияны `wordcount.py`-ден жаһандық атау кеңістігіне импорттағанда, `wordcount.py`-да код орындалады. Бұл `main` шақырылып, сосын `wordcount.py`-дегі барлық бағдарлама `test()` шақырылғанға дейін орындалып қоятындығын білдіреді. Алайда біз `main()`-ді, оны тестілеу үшін импорттаған кезде емес, ал тек `wordcount.py`-ді негізгі бағдарлама ретінде іске қосқанда ғана орындалғанын

қалаймыз. Бұл жағдайды түзету үшін, бізге wordcount.py-дегі main() шақырылуын келесі шартты нұсқаулардың ішінде орналастыру керек:

```
if __name__ == '__main__':  
    main()
```

3.6 тараудан еске алатынымыз, name дегеніміз - ағымдағы модульдің атауы болып табылады, егер модуль негізгі бағдарлама ретінде іске қосылатын болса, оған 'main' мәні меншіктеледі. wordcount.py басқа модульге импортталса, name - 'wordcount'-ке орнатылады. Осылайша, нұсқау main()-ді тек модульдер негізгі бағдарлама ретінде іске қосылғанда ғана орындайды.

Модульдік тестілерді әзірлеу

Ал енді іргетасын қалап қойған соң, count функциясына арнап модульдік тест әзірлейік. Бірнеше оңай тестілерден бастайық:

```
def test_count():  
    quote = 'Diligence is the mother of good luck.'  
    assert count(quote, 'the') == 2  
    assert count(quote, 'them') == 0  
  
    print('count барлық тестілері  
тапсырылды!')
```

Ойлануға берілетін сұрақ 7.9 *assert нұсқауы test_count функциясында не шығарады?*

count функциясы дұрыс істейтін болса, assert нұсқауы ештеңе экранға шығармайды. Басқа жағынан, егер count функциясы сынақтардың бірінен өтпей қалса, бағдарлама осы жерде AssertionError ерекшелігімен аяқталады. Оны көру үшін, екінші өрнекті (қате) assert count(quote, 'them') == 1 оқуға ауыстырыңыз. Сосын бағдарламаны қайта қосыңыз. Сіз мынаны көру керексіз

Ескертпе 7.1: Модульдік тестілеу құралдары

Python да модульдік тестілеуге арналған екі модульді ұсынады. doctest модулі тестілерді тікелей құжаттау жолдарына қосуға мүмкіндік береді. unittest модулі модульдік тестілеудің толық функционалды құралын ұсынады, ол модульдік және регрессивтік тестілеудің үдерістерін автоматтандырады, және ол аталмыш тарауда біз қабылдаған тәсілге көбірек сәйкес келеді. Егер бұл құралдар туралы толығырақ білгіңіз келсе, келесі сайтқа кіріңіз

<http://docs.python.org/3/library/development.html>

Traceback (most recent call last):

: (order of function calls displayed here)

```
assert count(quote, 'them') == 1
```

AssertionError

Бұл қате Сізге қандай өрнектердің орындалмайтынын көрсетеді, сол үшін Сіз проблеманы бақылай аласыз. (Берілген жағдайда, проблема жоқ; өрнекті бастапқы дұрыс нұсқасына қайта ауыстырыңыз.)

Осы екі тестінің нәтижелері өз беттерінше count функциясының дұрыс екендігін растайтындай жеткілікті дәлел бермейді. Бұрын айтқанымыздай, біз барлық мүмкіндік диапазонына репрезентативті болып табылатын көптеген тестілерді таңдау керекпіз. Белгілі бір тест үшін қолданылатын кіріс мәліметтері тестілік жағдай деп аталады. Тестілік жағдайларды жалпы үш санатқа бөлуге болады:

1. *Жалпы жағдайлар.* Алдымен біз функцияны бірнеше қарапайым кіріс мәліметтеріне тексереміз де, оның негізгі функционалдығының дұрыстығына көз жеткіземіз. Өртүрлі жауптарды беретін тестілік жағдайларды таңдауға тырысыңыз. Мысалы, count функциясы үшін біз келесі тестілерді жасай аламыз (оларға жоғарыда келтірілген екеуі кіреді):

```
quote = 'Diligence is the mother of good luck.'  
assert count(quote, 'the') == 2  
assert count(quote, 'the ') == 1  
assert count(quote, 'them') == 0  
assert count(quote, ' ') == 6
```


Назар аударыңыз, біз әртүрлі нәтижелерге, соның ішінде мақсатты жол табылмайтын жағдайға әкелетін кіріс мәліметтерін таңдадық.

2. *Шектес жағдайлар.* Шектес жағдай дегеніміз - жарамды кіріс мәліметтері аумағының шекарасында орналасқан кіріс мәліметтері болып табылады. count функциясында кіріс мәліметтері - екі жол болып табылады, сондықтан біз функцияны минималды ықтимал ұзындықтағы жолға және ұзындықтары минималдыға жақын келетін жолдарға тексергіміз келеді. (Егер максималды ұзындық болса, біз оны да тексерер едік.)

Минималды жол ұзындығы - бос жол (") болып табылады. Енді бізге тестілік жағдайларды count функциясына әзірлеу керек, онда кез келген параметр бос жол болып табылады. target мақсатты жолы бос болатын жағдайдан бастайық.

Ойлануға берілетін сұрақ 7.10 *count(quote, ") үшін қандай жауап дұрыс болады? Басқаша айтқанда, бос емес жолда бос жол неше рет шығады?*

Бұл сұрақ 10-да неше нөл бар деген сұраққа ұқсайды. Математикада, бұл сан (10/0) белгісіз, ал Python қате жауап береді. Жолдағы бос жолдарды санау да қателік бере ме? Немесе, мүмкін, жауабы 0 болу керек шығар; басқаша айтқанда, бос емес жолда бос жол болмайды.

Мәселен, функция егер target бос жол болса 0 қайтару керек делік. Сонымен біз келесі үш тестілік жағдайды test_count функциясына қосамыз:

```
assert count(quote, '') == 0
assert count('a', '') == 0
assert count('', '') == 0
```

Мұнда біз үштүрлі ұзындықтағы жолды text-ке жібереміз.

Ойлануға берілетін сұрақ 7.11 count функциясы осы тесттерді тапсыра ала ма? count(quote, ") іс жүзінде нені қайтарады? Кодқа қараңыз да неліктен олай екенін түсініп көріңіз. Салыстыру үшін, кіріктірілген жолдық quote.count(") әдісі не қайтарады?

count(quote, ") және quote.count(") екеуінің де қайтарылатын мәні 38, ол len(quote) + 1 құрайтындықтан тестіден өтпейді. Осылай target бос жоал болғанда, if шартының төмендегідей анықталатын

```
if text[index:index] == "":
```

әрбір тесті ақиқат болып табылады. Цикл `len(text) + 1` рет қайталанатындықтан, әр уақытта

`Count` артылады, снрда `len(text) + 1 - count` -ке меншіктеледі.

Бұл жағдайды тест анықтамаса, біз байқамай қалдырып кетер едік. Біз күткен нәтиже функцияның қазіргі әрекетінен айрықша болғандықтан, біз `count` функциясын `if` нұсқауын `for` циклынан бұрын қою арқылы өзгертуіміз керек:

```
def count(text, target):
    assert isinstance(text, str) and isinstance(target, str)

    if target == "":
        return 0

    count = 0
    for index in range(len(text) - len(target) + 1):
        if text[index:index + len(target)] == target:
            count = count + 1
    return count
```

Осы өзгеріспен, барлық тестілер тапсырылу керек.

Ал енді егер `text` бос жол болса ше? Егер `text` бос болса, онда `count` әрқашан нөлді қайтару керек деп ойлау жөн көрінетін секілді. Онда тағы екі тестілік жағдай қосайық, бірі ұзын `target` екіншісі қысқа `target` болсын:

```
assert count("", quote) == 0
assert count("", 'a') == 0
```

Ойлануға берілетін сұрақ 7.12 *`count("", quote)` іс жүзінде не қайтарады?*

Бұл жағдайда екі тест те тапсырылады. Егер `text` бос жол болса, ал `target` кем дегенде бір символдан тұрса, онда `for` циклінің диапазоны бос

болады да функция нөлді қайтарады.

Бос жолмен тестілерге қосымша ретінде біз тағы бірнеше тесті қосамыз, ондағы екі жол да бос жолға жақын болады:

```
assert count(quote, 'e') == 4
assert count('e', 'e') == 1
assert count('e', 'a') == 0
```

Шектес жағдайдың тағы бір түрі алгоритмнің жолдың басына не соңына (немесе, келесі тарауда, тізімде) жақын іздестіруді орындайтын жағдайларды білдіреді. Мысалы, біз функциямыздың text басында және соңында шығатын тармақшарларды дұрыс санайтындығын тексеріп алуымыз керек:

```
assert count(quote, 'D') == 1
assert count(quote, 'Di') == 1
assert count(quote, 'Dx') == 0
assert count(quote, '.') == 1
assert count(quote, 'k.') == 1
assert count(quote, 'x') == 0
```

Назар аударыңыз, мұнда біз тағы екі жағдай қостық, онда мақсатты жол басындағы не соңындағы кез келген бірнәрсеге ұқсайды, бірақ шын мәнінде ондай сәйкестік жоқ.

3. Тұйық жағдайлар. Тұйық жағдай дегеніміз - сирек кезігетін кіріс мәліметінің басқа бір түрі болып табылады, ол функцияның бұзылуына себеп болуы мүмкін. count функциясы үшін біздің тұйық жағдайларымыз олардың көбісін қарастырады. Алайда text пен target бірдей болған кезде және text target-ке қарағанда қысқарак болса тағы екі жағдай әдеттегідей болмайды:

```
assert count(quote, quote) == 1 assert count('the', quote) == 0
```

Патологиялық тұйық жағдайларды ойлап табу - тәжірибе арқылы келетін дағды болып табылады. Көптеген компаниялар бағдарламалық жасақтамасын бұзатын тұйық жағдайларды анықтаумен ғана айналысатын бағдарламашыларға қыруар қаражат төлейді!

Егер бұл тестілік жағдайлардың барлығын бірге қосатын болсақ, біздің модульдік тестілеуіміз count функциясы үшін мынадай болады:

```
def test_count():
    quote = 'Diligence is the mother of good luck.'
    assert count(quote, 'the') == 2      # жалпы
    жағдайлап assert count(quote, 'the ') == 1
    assert count(quote, 'them') == 0
    assert count(quote, ' ') == 6
    assert count(quote, '') == 0          # шектес
    жағдайлап assert count('a', '') == 0
    assert count('', '') == 0
    assert count('', quote) == 0
    assert count('', 'a') == 0
    assert count(quote, 'e') == 4
    assert count('e', 'e') == 1
    assert count('e', 'a') == 0 assert
    count(quote, 'D') == 1 assert
    count(quote, 'Di') == 1
    assert count(quote, 'Dx') == 0
    assert count(quote, '.') == 1
    assert count(quote, 'k.') == 1
    assert count(quote, '.x') == 0
    assert count(quote, quote) == 1      # тұйық
    жағдайлап assert count('the', quote) == 0
```

Print('count-ке арналған барлық тестілер тапсырылды!')

Жылжымалы нүктесі бар мәндерді тексеру
count функциясы дұрыс жұмыс істейтініне көз жеткізген соң, біз
getWordCounts функциясын іске асыруға және сынауға кірісе аламыз.
Бұл функция үшін модульдік тестілеу келесідей болуы мүмкін. (Басқа
тестілерді жаттығу ретінде қалдырамыз.)

```
def test_getWordCounts():
    text = 'Call me Ishmael. Some years ago--never mind
how long \ precisely--having little or no money in my
purse, and nothing \ particular to interest me on shore, I
thought I would sail about \
a little and see the watery part of the world. It is a way I have \
of driving off the spleen and regulating the circulation.'

    assert getWordCounts(text, 'the', 20) == 4 / 15
    assert getWordCounts(text, 'spleen', 20) == 1 / 15

    print('getWordCounts үшін барлық тестілер
тапсырылды!')
```

Бұл цикл бір онмындықты бір миллион рет қосады, сондықтан жауап assert нұсқауында көрсетілгендей жүз болу керек еді. Дегенмен, assert орындалмайды, себебі sum мәні іс жүзінде дөңгелектеу қателігінен 100-ден көп болады. Осындай проблеманы шешу үшін, бізге диапазонның ішінде жылжымалы нүктесі бар мәнді әрқашан тексеруіміз керек. Мысалы, жоғарыда берілген assert нұсқауын мынаған ауыстыру керек

```
assert sum > 99.99 and sum < 100.01
немесе
assert sum > 99.99999 and sum < 100.00001
```

Тексерілетін диапазонның көлемі нақтылыққа тәуелді, ол белгілі бір қосымшаға қажетті болып келеді.

Осы идеяны өзіміз білетін volumeSphere функциясына қолданып көрейік:

```
def volumeSphere(radius):
    return (4 / 3) * math.pi * (radius ** 3)
```

Бұл функцияға тестілік жағдайларды жасау үшін бізге әртүрлі radius мәндерінің жауаптары қандай болатынын анықтап алу керек, сосын осы

жағдайлардың әрқайсысына assert нұсқауын жазу керек. Мысалы, радиусы 10-ға тең шардың көлемі шамамен 4188.79 болады. Сонда біздің assert нұсқауымыз мынадай болуы керек

```
assert volumeSphere(10) > 4188.789 and volumeSphere(10) < 4188.791
```

Жаттығулар

7.3.1. getWordCounts функциясы үшін модульдік тестілеуді аяқтаңыз. Шектес және тұйық жағдайларды қосуды ұмытпаңыз.

7.3.2. 7.2-тараудағы digit2String функциясы үшін толық модульдік тест әзірлеңіз.

7.3.3. 7.2-тараудағы int2String функциясы үшін толық модульдік тест әзірлеңіз.

7.3.4. Төмендегі assignGP функциясы үшін толық модульдік тест әзірлеңіз.

```
def assignGP(score): if score >= 90:  
    return 4  
    if score >= 80: return 3  
    if score >= 70: return 2  
    if score >= 60: return 1  
    return 0
```

7.3.5. 7.2.4-жаттығудағы volumeSphere функциясы үшін толық модульдік тест әзірлеңіз.

7.3.6. 7.2.6-жаттығудағы windChill функциясы үшін толық модульдік тест әзірлеңіз.

7.3.7. 7.2.9-жаттығудағы decayC14 функциясы үшін толық модульдік тест әзірлеңіз.

7.3.8. 7.2.11-жаттығудағы reverse функциясы үшін толық модульдік тест әзірлеңіз.

7.3.9. 7.2.12-жаттығудағы find функциясы үшін толық модульдік тест әзірлеңіз.

МАЗМҰНЫ

1.1 ЕСЕПТЕР МЕН АБСТРАКЦИЯ	5
1.2 АЛГОРИТМДЕР ЖӘНЕ БАҒДАРЛАМАЛАР	8
1.3 ТИІМДІ АЛГОРИТМДЕР	18
1.4 АҚЫМАҚ КОМПЬЮТЕРЛЕР	30
2 ҚАРАПАЙЫМ ЕСЕПТЕУЛЕР	45
2.1 ЦИРККЕ ҚОШ КЕЛДІҢІЗ	45
2.2 АРИФМЕТИКАЛЫҚ ӘРЕКЕТТЕР	46
2.3 АТАУМЕН НЕ БАЙЛАНЫСТЫ?	54
2.4 ФУНКЦИЯЛАРДЫ ПАЙДАЛАНУ	63
2.5 БИНАРЛЫ АРИФМЕТИКА	76
3 ВИЗУАЛДЫ АБСТРАКЦИЯ	88
3.1 АБСТРАКТ ДЕРЕКТЕР	89
3.2 ТАСБАҚАЛАР КӨМЕГІМЕН ВИЗУАЛДАУ	95
3.3 ФУНКЦИОНАЛДЫ АБСТРАКЦИЯ	101
3.4 КӨРКЕМ БАҒДАРЛАМАЛАУ ҚАБІЛЕТІ	119
3.5 ФУНКЦИЯЛАРҒА ҚАЙТА ОРАЛУ	129
3.6 ҚОЛДАНУ САЛАСЫ МЕН АТАУЛАР КЕҢІСТІГІ	137
4 ӨСУ МЕН КЕМУ	146
4.1 ДИСКРЕТТІ ҮЛГІЛЕР	147
4.2 ПОПУЛЯЦИЯ ӨЗГЕРІСТЕРІН ВИЗУАЛДАУ	176
4.3 ШАРТТЫ ИТЕРАЦИЯ	182
4.4 ҮЗДІКСІЗ ҮЛГІЛЕР	188
4.5 ТАЛДАУДЫҢ САНДЫҚ ӘДІСТЕРІ	207
4.6 ТҮЙІНДЕМЕ	218
4.7 АЛДАҒЫ ЗЕРДЕЛЕУ	223
4.8 ТӘЖІРИБЕ ТАПСЫРМАЛАРЫ	224
5 ЖОЛ ТОРАПТАРЫ	241
5.1 КЕЗДЕЙСОҚ ҚОЗҒАЛУ	241
5.2 ЖАЛҒАН КЕЗДЕЙСОҚ САҢДАР ГЕНЕРАТОРЛАРЫ	260
5.3 ЫҚТИМАЛДЫҚ ҮЛЕСТІРІМДІ ҮЛГІЛЕУ	266
5.4 БУЛЬДІК ШАМАҒА ОРАЛУ	272
5.5 ОЙЛАН ТАП ОЙЫНЫ	290
6 МӘТІН, ҚҰЖАТТАР, ЖӘНЕ ДНҚ	312
6.1 СӨЗДЕРДЕН ТҰРАТЫН ЕСЕПТЕР	312
6.2 МӘТІНДІК ҚҰЖАТТАР	323
6.3 ЖОЛДАРДЫ КОДТАУ	334
6.4 СЫЗЫҚТЫҚ УАҚЫТ АЛГОРИТМДЕРІ	347
6.5 МӘТІНДІ ТАЛДАУ	356
6.6 МӘТІНДЕРДІ САЛЫСТЫРУ	368
6.7 ГЕНОМИКА	379
6.8 ТӘЖІРИБЕЛІК ТАПСЫРМАЛАР	398
7 БАҒДАРЛАМАЛАРДЫ ӘЗІРЛЕУ	409
7.1 ЕСЕПТІ ҚАЛАЙ ШЕШУ КЕРЕК	410
7.2 КЕЛІСІМ-ШАРТТЫҚ БАҒДАРЛАМАЛАУ	422
7.3 ТЕСТІЛЕУ	435