

Сәлем, әлем!

Python – бағдарламалау әліппесі



Сәлем, әлем!

Python – бағдарламалау әліппесі

Бұл кітап – қазақстандық оқушыларға бағдарламалаудың алғашқы қадамдарын қазақ тілінде үйретуге арналған. Мақсатымыз – ауыл мен қала балаларына тең мүмкіндік беру, жастарды IT саласына қызықтыру және болашақ мамандардың жаңа буынын тәрбиелеу.

Автор: **Қожахмет Нұрәділ.**

Кітаптың дайындалуына атсалысып, қолдау көрсеткен барша команда мүшелеріне алғыс білдіреміз: **Исмаил Әбдірахман, Итеғұлова Сая, Смағұлов Темірлан, Зият Ерқали, Гинет Аян, Мамбетниязов Аймұрат, Нұрғалиева Дәнекер, Батырхан Хафиз, Алдажаров Дулат, Дарбаева Айя, Қасенов Айсұлтан.** Олар халықаралық IT компанияларында еңбек етіп жүрген қазақтың жас мамандары, өз тәжірибелері мен білімін қарапайым әрі түсінікті тілде оқырманмен бөлісуді мақсат етті.

Бұл басылым Kaspi.kz қазақстандық инновациялық компаниясының қолдауының арқасында жарық көрді.

Кітаптың басылуын ұйымдастырған – **Қазақстанның спорттық бағдарламалау Федерациясы.**

** Коммерциялық мақсатта пайдалануға арналмаған. Көшіріп басуға тыйым салынады.*

Мазмұны

Алғысөз	7
Менің жолым	8
Бағдарламалауды неліктен үйрену керек?	9
IT маманы болғысы келетін жастарға кеңес:	12
Бағдарламалау дегеніміз не?	12
Python - бағдарламалау әліппесі	13
Python бағдарламасын компьютерге қалай жүктеуге және орнатуға болады?	14
Python бағдарламау тілін орнату	17
Windows-қа Python орнату нұсқаулығы	17
Орнатуды тексеру	19
Қосымша: "pip"-тің орнатылуын тексеру	19
Алғашқы бағдарлама	20
Python негіздері: айнымалылар, деректер типтері, операторлар және негізгі бағдарламалау түсініктері.	21
Айнымалылар	21
Деректер типтері	21
Операторлар:	23
Шартты операторлар	28
Нақты өмірден мысал: Саябаққа саяхат	28
Логикалық операторлар	29
Табуляция (ендірме) туралы	29
Қорытынды	30
Функциялар: Python тілінде функцияларды қалай жазуға және пайдалануға болады.	30
Функция дегеніміз не?	30
Функцияның атауы қандай бола алады?	32
Функцияның денесі	33
Функцияның нәтижесі	33
Циклдер: қайталанатын тапсырмаларды автоматтандыру үшін циклдарды қалай пайдалануға болады?	34
"For" циклі	34
range	35

“While” циклі	37
Циклді бұзу және жалғастыру	37
Кірістірілген циклдер	38
Қорытынды	39

Деректер құрылымы 41

Тізімдер: Тізімдерге кіріспе және олармен қалай жұмыс істеу қажет?	41
<i>Тізім деген не?</i>	41
<i>Тізімді құрастыру</i>	41
<i>Тізім элементтеріне қол жеткізу</i>	42
<i>Тізімдерді өзгерту</i>	43
<i>Элементтерді қосу</i>	43
<i>Элементтерді жою</i>	44
<i>Элементтерді өзгерту</i>	44
<i>Тізім арқылы айналым</i>	44
<i>Тізім операциялары</i>	45
<i>Тізім ұзындығы</i>	45
<i>Тізімдерді біріктіру</i>	45
<i>Тізімдерді кесу</i>	46
<i>Элементтің бар-жоғын тексеру</i>	46
<i>Тізімді сұрыптау</i>	46
<i>Тізімді кері айналдыру</i>	46
<i>Қорытынды: тізімдер – сіздің достарыңыз</i>	47
Сөздіктер: кіріспе және оларды пайдалану жолдары	47
<i>Сөздіктің жасалуы:</i>	47
<i>Мәндерге қол жеткізу:</i>	47
<i>Мәндерді өзгерту:</i>	47
<i>Жаңадан кілт-мән жұбын енгізу:</i>	48
<i>Кілт-мән жұптарын жою:</i>	48
<i>Кілттің бар-жоғын тексеру:</i>	48
<i>Сөздік әдістері:</i>	48
<i>Сөздік бойынша өтіп шығу:</i>	49
<i>Сөздік жасаудың тағы бір жолы:</i>	50
Кортеж (Tuple)	50
<i>Мысалдар:</i>	51
<i>Қарапайым кортеж:</i>	51
<i>Кортеждегі элементтерге қол жеткізу:</i>	51
<i>Tuple және List деректер типтерің айырмашылықтары</i>	51
<i>Тізім (list) қашан қолдану керек?</i>	51
<i>Кортежді (tuple) қашан қолдану керек?</i>	52
<i>Қорытынды:</i>	53
Set	53
<i>Set-тің негізгі ерекшеліктері:</i>	53
<i>Set-ті құру:</i>	53

<i>Set-пен жасалатын негізгі операциялар:</i>	54
<i>Set-ті қашан қолдану керек?</i>	55
Deque дегеніміз не?	55
<i>Deque-нің негізгі сипаттамалары:</i>	56
<i>deque-ді қолдану мысалдары:</i>	56
<i>Deque-нің негізгі әдістері:</i>	57
<i>Deque не себепті пайдалы?</i>	57
Файлдармен жұмыс	59
Файлдардың маңызы неде?	59
Файлдар қолдану негіздері	59
Файлдарды ашу және жабу	59
Файлдарды оқу	60
Файлдарды жазу	60
Файлдарды өзгерту	61
“for” циклы арқылы файлдарды оқу	61
Қорытынды	62
Модуль: Python модульдерін қалай пайдалануға болады?	62
Модуль деген не?	63
Модульдерді енгізу	63
Жалпы модульдер	64
Өзіңіздің модульдеріңізді жасау	65
1-қадам: жаңа Python файлын жасаңыз	65
2-қадам: Модульдегі функцияларды анықтаңыз	65
3-қадам: Пайдаланушы модулін қолдану	65
Модульдерді ұйымдастыру	66
Қорытынды	66
Кластар: объектіге бағытталған бағдарламалауға кіріспе және кластарды пайдалану	67
Класс дегеніміз не?	67
Кластар не үшін қажет?	67
Класс құрылымы	68
Кластан объектілерді жасау	69
Мұрагерлік	74
Полиморфизм	75
Қорытынды	78
Қорытынды жоба: Жылан ойыны	79
Жылан ойыны дегеніміз не?	79
Енді “Жылан” ойынын неліктен жасайтынымызды түсіндірейін.	79
Алғышарттары:	79
Difficulty және Direction кластары	79

Block класы	82
<i>convert_to_block_pos</i>	83
<i>create_random_point</i>	83
__init__ конструкторы	84
__eq__ функциясы	84
Window класы	85
__init__ конструкторы	86
<i>is_in_bound</i> әдісі	86
<i>get_top_location</i> әдісі	87
<i>get_top_left_location</i> әдісі	87
<i>get_bottom_location</i> әдісі	88
Snake класы	88
__init__ конструкторы	88
<i>get_head_pos</i> әдісі	89
<i>change_direction</i> әдісі	89
Move әдісі	90
<i>is_touching_itself</i> әдісі	90
Game класы	91
Кластың тұрақтылары (константалары)	91
__init__ конструкторы	91
<i>show_score</i> әдісі	93
<i>game_over</i> әдісі	94
<i>paint_snake</i> и <i>paint_apple</i> әдістері	95
<i>handle_key_press</i> әдістері	96
<i>start</i> әдісі	96
Негізгі ойын циклы:	97
Main функциясы	98
Pudame инициализациясы және қателерді тексеру	99
Қиындық деңгейін таңдау	100
Қиындық деңгейін орнату	101
Ойынның құрылуы және іске қосылуы	101
main функциясын іске қосу	101
Жалпы қорытынды	102
Ойынды іске қосу	102
Кодтын толық нұсқасы	102
Қорытынды сөз	112
Болашақ IT маманына бірнеше кеңес:	113
Бағдарламалаудағы келесі қадам	114
Мұның маңызы неде?	114
Бағдарламалаудағы келесі қадам қандай?	115

Сәлем, әлем!

Python – бағдарламалау әліппесі

Алғысөз

Қымбатты оқырман! Ең алдымен сіздің қолыңыздағы кітапты не үшін жазғанымды түсіндіре кеткім келеді. Бұл кітапты жазуға білек сыбана кірісуіме бірнеше жағдай түрткі болды. Біріншіден, өзге мамандықтарға қарағанда IT жаңа сала. Елімізде енді-енді етек жая бастағандықтан, бұл саланың қыр-сыры жайлы қазақ тілінде жазылған кітаптар жоқтың қасы десек, артық айтқандық болмас. Мектеп қабырғасында, әсіресе ауыл мектептерінде оқушыларға бұл сала туралы мол мағлұмат берілмейтінін байқап жүрміз. Сәйкесінше, ауыл балалары арасында “Ақпараттық технология” мамандығына түсем деушілер де аз ұшырасады. Ал қала мектептерін бітірген түлектердің тіршілігі бейтаныс ауылдық жерлерге барғысы келмейтіні анық. Осыдан келіп, ауыл балаларының жаһандық деңгейде анағұрлым қарқынды дамып жатқан сала туралы танымы шектеулі болып қала береді.

Кітапты жазуымның келесі себебіне көпшілік арасында бұл саланың қиындығы туралы қисынсыз түсініктің қалыптасуын жатқызар едім. Жеткіншектермен сөйлесу барысында “бағдарламалауды меңгеру қиын”, “код жазу қолымнан келмейді”, “IT маманы болу үшін ерекше дарынды болу керек шығар” деген пікірлерді жиі естимін. Шын мәнінде IT маманы болу үшін асқан дарынды болудың қажеті жоқ. Бағдарламалау – бұл ядролық физика сияқты тым күрделі ғылым саласы емес. Тіпті “Ақпараттық технологиялар” мамандығы бойынша академиялық білім алғандығы туралы дипломның да қажеті шамалы. Бәрі зеректігіңіз бен ұғымталдығыңызға, қабілетіңізді қаншалықты жетілдіргеніңізге байланысты. Талантты дамытпаған жағдайда таланттан гөрі еңбекқорлығы мол табанды адамның басым түсіп жататындығын естен шығармаған жөн.

Бастапқыда бағдарламалаумен айналысу қиын әрі түсініксіз болып көрінуі мүмкін. Алайда кез келген нәрсе басында оңай соқпайды. Мәселен, жаңа аспапты меңгеруге, жаңа тілді үйренуге, қарапайым ғана жаңа ойынның шартын түсінудің өзіне біраз уақыт жұмсаймыз ғой. Сол сияқты бағдарламалауды да табандылық танытсаңыз, араға аз-кем уақыт салып, үйреніп алуға болады. Маған сеніңіз!

Бағдарламалауды үйренудің мүмкіндіктері көп. Өзіңіздің қалаған ойыныңызды, анимацияларды, веб-сайттарды, тіпті, дербес жүргізілетін автокөлікті жасай аласыз. Бастысы, осы саладағы ізденісіңізді тоқтатпасаңыз, келешекте үлкен жобаларға жетекшілік етіп, әлемдік нарыққа қажетті бағдарламалардың авторына айналуыңыз әбден мүмкін. Қазір әлем сапалы қызмет жасайтын, қабілетті IT мамандарына мұқтаж. Көптеген қызмет түрлері электронды нұсқада және онлайн режимде көрсетілуіне байланысты біздің елімізге де білікті бағдарламалаушылар қажет. Байқасаңыздар киім-кешек, тұрмыстық заттар, косметика, кітап, тіпті азық-түліктің өзіне қосымшалардың көмегімен онлайн тапсырыс беру күннен күнге дамып келеді. Осындай қосымшалар мен бағдарламаларды, әрине, IT мамандары құрастырады. Бұған қоса, өз ісін жақсы меңгерген мамандар үшін әлемнің кез келген елінің есігі ашық екенін ұмытпаңыз. Мәселен, Google, Yandex, Facebook, Twitter, Booking, Opera сынды алпауыт компанияларда жұмыс жасап жүрген қазақтың қыз-жігіттері жетерлік.

Осы кітап арқылы қазақ жастарына IT саласының мүмкіндіктерін насихаттап, бағдарламалаудың алғашқы кезеңдері мен құрылымдық сатысын үйреткім келеді.

Бұл саланың қыр-сырын меңгерген маман ретінде ұсынған еңбегімді қабыл алыңыз, қадірлі досым!

Менің жолым

Мектепте оқу үлгерімім орташа болды. Түрлі байқауларды, олимпиадаларды ұнатпайтынмын. “Бәрібір үлкен нәтижеге жете алмаймын, қолымнан келмейді” деген суық ой жеңіп кете беретін. Мұндай сезім өзін үздік деп санамайтын оқушылардың көбіне таныс-ау деген ойдамын. Ең қызығы, информатика пәнінен беретін мұғалім мені тақтаға шығарып: “ақымақсың, түк білмейсің” - деп ұрысушы еді. Мектептегі ұстаздарым: “түк білмейсің” - деп қанша ұрысса да, сөздерін жүрегіме жақын қабылдамауға тырыстым. “Түптің түбінде IT маманы боламын” деген арманымнан бас тартпадым. Тынбай жұмыс жасап, үздіктердің қатарында болуға талпына бердім. 2022 жылы Назарбаев Университетін “Компьютер ғылымдары” мамандығы бойынша тәмамдадым. Оқуды бітірген соң Лондонға қоныс аударып, сондағы WhatsApp компаниясына IT маманы болып жұмысқа орналастым. Бұл – менің орындалған арманым.

Құрметті оқырман, өз оқиғамды айтудағы мақсатым мақтану емес, сіздерге шабыт сыйлау. Естеріңізде болсын, барлық адам арманына жете алады. Тек берілмеу керек. Басқа адамдардың теріс пікірінің мақсатыңызға бөгет болуына жол бермеңіз. Үнсіз ғана арманыңызға қарай жылжи беріңіз. “Көрінген таудың алыстығы жоқ” деген халық даналығын жадыңызда ұстаңыз.

Бағдарламалауды неліктен үйрену керек?

Бағдарламалауды үйрену – өте құнды әрі қажетті дағды. Бұл дағдыны қалыптастыру арқылы үлкен мүмкіндікке ие боласыз әрі шығармашылық тұрғыдан жетістіктерге жетесіз. Мәселен, жұмыспен қамтылу жағынан бағдарламашылар жоғары сұранысқа ие екені белгілі. Өйткені, жоғарыда айтып өткенімдей, IT саласы еңбек нарығындағы басқа да салалармен біте қайнасып жатыр. Сауданың барлық түрі интернет жүйесіне еніп, онлайн жүргізіле бастады десек те болады. Атап айтсақ, веб-сайттар, мобильді қосымшалар, танымдық ойындар мен бағдарламалар, бағдарламалық жасақтама әзірлеу, түрлі ғылыми деректер мен жасанды интеллект өнімдері онлайн режимде қолжетімді. Ендеше бағдарламалауды үйрену арқылы сіз бір емес, бірнеше функцияны атқару мүмкіндігіне ие боласыз.

Бағдарламалаудың тағы бір артықшылығы бар. Ол логиканы дамытуға, есептерді құрылымдық жолмен шешуге үйретеді. Бұл дағды сіздің жақсы IT маманы болуыңызға ғана емес, өмірдегі басқа да жағдаяттарды тиімді шешуіңізге көп көмегін тигізеді. Ал шығармашылық тұрғыдан келер болсақ, бағдарламалау өз идеяңызды жүзеге асыруға себепші болады. Мысалы, сіз ҰБТ-ға дайындық қосымшасын жасағыңыз келсе немесе онлайн сауда орталығын ашқыңыз келсе, жобаңызды өз бетіңізше жасай аласыз. Бұл сөзіме әлемге әйгілі IT феномендері – Марк Цукерберг, Илон Маск, Билл Гейтс сынды тұлғалардың тәжірибесі дәлел.

Қазір бар әлем техникаға телміріп, технология сәт сайын жаңарған заман. Ал технология дамыған сайын IT саласының маңыздылығы арта түседі. Бағдарламалау – бұл код арқылы компьютерге пәрмен беріп, қалаған жобаны жүзеге асыру өнері. Мысалға мен жұмыс жасайтын WhatsApp әлеуметтік желісін алайық. Жобаны алғаш Брайан Эктон мен Ян Кум ойлап тапқан. Бүгінде 180-нен астам мемлекет WhatsApp-ты қолданады. Әлемнің кез келген нүктесінен досыңызбен, жақыныңызбен ақысыз хат

алысып, сөйлесе аласыз. Бұрын алыстағы ағайынмен хабар алмасу үшін хат жазып, оны пошта арқылы жолдайтын. Ұялы телефон пайда болған бастапқы жылдары да шет елмен байланысу оңай емес еді. Телефон байланысы қызметтерінің тарифтері қымбат болатын. Ал қазір интернетіңіз болса жеткілікті, WhatsApp арқылы оп-оңай, ешбір төлемақысыз хабарласа беруге болады. Бір сөзбен айтқанда WhatsApp бүкіл әлемнің байланыс мәселесін шешіп-ақ тастады. 2014 жылдан бері WhatsApp желісі белгілі IT маманы, кәсіпкер, инвестор Марк Цукербергтің иелігінде.

Бағдарламалауды меңгеріп, білімін жетілдіруді тоқтатпаған әр азамат үшін мүмкіндіктер көкжиегі кең. Жоғарыда айтқанымдай, қазір ақпараттық технологиялар дәуірі. Ақпаратты терең меңгерген адам әлемді билейді десек, артық айтқандық болмас. Ал сол ақпаратты жан-жаққа жеткізу үшін де бағдарламалаушылар ауадай қажет. Қазір медиа саласының өзі IT саласымен тұтастай сабақтасып келеді. Телеарналар, радио, газет-журналдардың өз арнайы сайттары жұмыс жасайды. Бұл ақпараттық сайттардың БАҚ-тың негізгі бөлігіне айналғанындығының дәлелі.

Сіздерді бүгінде жиі қолданылатын ең танымал бағдарламалау тілі – Python-мен таныстырғым келеді. Оны меңгеру аса қиын емес. Жақсылап ден қойсаңыз, жылдам үйреніп аласыз. **Бұл тіл аз синтаксисті талап етіп (яғни кодты жазуда көп ережелерді талап етпейді), оқуға, жазуға және түсінуге өте оңай болып келеді.** Python веб-сайт әзірлеу, деректер ғылымы, жасанды интеллект және басқа да көптеген салаларда қолданылады.

Тақырыпқа терең енбес бұрын сіздерге бағдарламалау негіздері жайлы қысқаша ақпарат беріп өткім келеді. Содан кейін Python-ды компьютерге қалай жүктеп, қалай орнататынымыз туралы ақпаратпен бөлісемін.

Осы жерде сіздерге замандасымыз **Мамбетниязов Аймұраттың** IT саласын қалай меңгергені туралы қызықты мәліметтерін жеткізгім келеді. Ақпарат түсінікті болу үшін, Аймұраттың айтуы бойынша, бірінші жақтан бергенді жөн көрдім.

“Абай облысына қарасты Қалбатау ауылының тумасымын. Әл-Фараби атындағы ҚазҰУ-де “Автоматизация және басқару” мамандығы бойынша білім алдым. Мәскеуге барып, Яндекс компаниясында тағылымдамадан өткеннен кейін, сол жерге жұмысқа орналастым. Мәскеуде жүргенімде Meta-дан (Facebook) шақырту алып, 2 жылдан бері сонда еңбек етіп жүрмін.

IT саласын Майнкрафт ойыны үшін таңдаған едім. Ойынның барлық артықшылықтарын пайдаланғым келетін, бірақ кейбір

серверлер ойын файлдарын қорғап, оған мүмкіндік бермейтін. Интернеттен іске қосу құралын декомпиляциялау, файлдың тұтастығын тексеруді жою және іске қосу құралын қайта жинау әдісін тауып алдым. Осылайша бағдарламалау әлемімен тереңірек таныстым.

Бағдарламалауды алғаш үйрене бастаған кезде қателесу – қалыпты жағдай. Қайта-қайта қателесуден қорықпаңыз. Ол – үйренудің негізі. Бастапқыда ешкімнің көмегіңізді шешу мүмкін еместей көрінеді. Қарапайым амалдарды шығарудың өзі қиындық тудырады. Мәселенің түйінін түсінуге қанша сағат жұмсасақ та жауабын оңайлықпен таба алмаймыз. Бұл кезеңнен мен де өттім. Кейде жауапты интернеттен іздеуіме тура келетін. Қажет ақпаратты еш жерден таба алмаған кезде тығырыққа тірелгендей болатынымын. Сондай қиын кездерде қолымды бір сілтеп, бағдарламалауды тастап кетуіме де болар еді, бірақ мен адам мәселеден қашпай, оны шешуге тырысқан сайын шыңдала түсетінін ұқтым. Табандылық танытсам, мықты бағдарламалаушы болатынымды түсіндім” - дейді Аймұрат досымыз.

Өзіне қиын сәттерде көмектескен кеңестерді еске алатырып, Аймұрат өз тарапынан мыналарды ұсынады:

- Жәрдем сұраудан, ізденуден қорықпаңыз. Интернеттің мүмкіндігі үлкен, арнайы форумдар, блогтар, онлайн курстар мен видеосабақтар тәрізді ресурстар көп. Түсінбеген сұрағыңызды форумда қоюыңызға немесе бағдарламалаушылар қауымдастығына жазуыңызға болады. Көмектесуге қашанда дайын адамдар бар.
- Қателіктерден сабақ алыңыз. Қателескен уақытта нені дұрыс жасамай жатқаныңызды түсінуге тырысыңыз. Бұл дәл сол қателікті болашақта қайталамауға және кодтың қалай жұмыс жасайтынын жақсылап түсініп алуға көмектеседі.
- Берілмеңіз. Бағдарламалау сүрлеуі мен соқпағы мол жолдағы қызықты саяхат сияқты. Кейде шешім бірден табылмайды. Бірақ жасаған әр қадамыңыз мақсатыңызға жақындатады. Шыдамдылық пен табандылық – бағдарламалаушыға қажетті негізгі қабілеттер.
- Үнемі жаттығыңыз. Көп код жазған сайын өзіңіздің іскерлік пен дағдыңызды қалыптастырып, машықтана түсесіз.

Осы ретте өзім үшін аса құнды бір тәжірибеммен бөлісе кетейін. Сабақ аяқталып, жазғы демалыс басталған кез еді. Менің алдымда екі таңдау тұрды: біріншісі, құрылыс алаңына барып, ағама көмектесу. Екіншісі, бағдарламалауды өз бетімше

үйрену. Ағамның жанында болып, құрылыста жұмыс жасасам, азын-аулақ табыс табар едім. Ал код жазуды үйрену дәл сол уақытта қолыма ақша ұстатпағанымен, болашақта үлкен жемісін берері сөзсіз. Әрине мен ол кезде бағдарламалауды үйренудің құрылыста жұмыс істеуден тиімдірек екенін жете түсінген жоқпын. Бірақ код жазу өзіме ұнайтын. Соңында дұрыс таңдау жасағаным анық көзім жетті.

IT маманы болғысы келетін жастарға кеңес:

Алдыңызға биік мақсаттар қойыңыз. Мақсатыңыз мүмкіндігінше ұзақ мерзімді болсын. Мәселен, грант жеңіп алу оқушы кезімдегі ең негізгі мақсатым еді. Грантқа оқуға түскеннен кейінгі мақсатым университетті жақсы бітіру болды. Университетті бітіргеннен кейін жалақысы қомақты жұмысқа орналасуды мақсат еттім. Осылай жалғасып кете берді. Кейіннен мақсат алыс мерзімді болашаққа бағыттталып, кешенді түрде жоспарлануы керек екенін түсіндім. Десек те, мақсатыңызды қайта қарап үйреніңіз. Уақыт өте келе адамның пікірі, құндылықтары өзгеріп отырады. Бір жыл бұрын алға қойған мақсатыңызға сай жұмыс жасап жүрсеңіз, бірақ қазір сіз үшін оның маңызы жойылса, ары қарай сол бағытта жұмыс жасаудың қаншалықты мәні бар? Алға қойған мақсатыңыздың шын қалауыңыз болуы аса маңызды. Армандар оны іске асыруға дайын болған кезде мақсатқа айналады. Ұзақ мерзімді мақсат қою үшін алдымен қысқа мерзімді мақсатыңыз бен қалауыңызды орындаңыз. Арманыңыздың шындыққа айналғанын қаласаңыз, сол бағытта еңбек ете беріңіз.

Бұл саланың қызығы мен қиыншылығы төңірегіндегі замандастарымыздың басынан өткерген бірқатар мәселелерге алдағы уақытта да тоқталатын боламыз. Енді басты тақырыбымызға көшейік.

Бағдарламалау дегеніміз не?

Бағдарламалау – бұл компьютер түсінетін және орындай алатын нұсқауларды құрастыру процесі. Бұл нұсқаулар код деп аталады және олар бағдарламалау тілінде жазылады. Компьютер бұл кодты тапсырмаларды орындау немесе есептерді шешу үшін пайдаланады.

Бағдарламалауды мобильді телефондағы түрлі қосымшаларды, компьютерлік ойындар немесе веб-сайт сияқты күрделі жүйелерді жасау үшін қолданады. Бағдарламалауды үйрену арқылы қалаған технологиялық құрылымды құрастыра аласыз.

Python - бағдарламалау әліппесі

Python тілін бағдарламалаудың әліппесі десек, қателеспейміз. Python - пайдаланушыға ыңғайлы және танымал тіл. Саланы жаңадан игере бастаған мамандар үшін одан тиімді бағдарламалау тілі жоқ. Енді осы тілді меңгеру тетіктеріне тоқталайын. Алдымен Python бағдарламалау ортасын компьютерге орнатасыз. Орта дайын болған соң айнымалылар мен дерек түрлері туралы мәліметке көшеміз. Айнымалылар бағдарламаларда ақпаратты сақтау үшін қолданылады. Айнымалыға мән бере аламыз және сол мәнді бағдарламада пайдаланамыз. Мысалы, "name" деп аталатын айнымалыны жасап, оған "Аружан" мәнін тағайындауға болады. Содан кейін бүкіл бағдарламамызда "name" айнымалысын қолдана аламыз және ол әрқашан "Аружан" мәніне сілтеме жасап отырады.

Дерек типтері - айнымалыларда сақтауға болатын ақпараттың әркелкі түрлері. Python-да жолдар, бүтін сандар және қалқымалы мәндерді қоса алғандағы бірнеше деректер түрлері бар. Мәтінді көрсету үшін жолдар, бүтін сандарды көрсету үшін бүтінсандық айнымалылар, ал бөлшек сандарды көрсету үшін қалқымалы сандар қолданылады. Сонымен қатар шартты операторлар мен циклдар сияқты негізгі бағдарламалау тұжырымдамаларын да қарастырамыз. Шартты операторлар белгілі бір шарттар негізінде әртүрлі әрекеттерді орындауға мүмкіндік береді. Мысалы, егер айнымалы мән 10-нан үлкен болса, бір әрекетті орындау үшін "if" операторын, ал 10-нан аз немесе оған тең болса, басқа әрекетті қолдануымыз мүмкін. Циклдар әрекеттерді бірнеше рет қайталауға мүмкіндік береді. Мұның белгілі бір жағдайларда пайдалы болатынын да ескерте кетейін.

Кітапты оқып шығып, Python арқылы бағдарлама жазу жолдарын меңгересіз. Айнымалылар, деректер түрлері және бағдарламалау тұжырымдамалары туралы негізгі түсінікке ие боласыз. Берілген теориялық ақпараттарды жақсылап түсініп, тәжірибе барысында қолдану арқылы білікті бағдарламашы болу мүмкіндігіне бір қадам жақындайтыныңызға сенімім мол.

Python бағдарламасын компьютерге қалай жүктеуге және орнатуға болады?

Python-мен жұмыс істеуді бастау үшін алдымен бағдарламаны компьютерге жүктеп алып, орнатуыңыз керек. Python – тегін орнатылатын және ашық қолданыстағы бастапқы тіл, яғни оны интернеттен ешбір ақы төлемей-ақ жүктеп алуға болады. Python-ді жүктеуге ұсынатын көптеген веб-сайттар бар, бірақ бағдарламаны Python ресми веб-сайтынан жүктеп алған дұрыс.

Енді бағдарламаны компьютерге жүктеу барысына тоқталайық. Алдымен Python-ның ресми веб-сайтына (python.org) өтіңіз. “Жүктеулер” сілтемесін басыңыз. Ол жерден Python нұсқасы (Python 3 – соңғы нұсқасы) мен оны пайдаланушы операциялық жүйені (Windows, Mac немесе Linux) таңдауға болады. Қажетті нұсқаны жүктеп алғаннан кейін оны компьютеріңізге орнатасыз. Орнату процесі де өте қарапайым. Нұсқаулықта көрсетілген қадамдарды орындау арқылы компьютеріңізге оңай қондырасыз.

Осыдан кейін Python бағдарламалау негіздерімен мұқият танысып шығамыз. Айнымалылар, деректер түрлері және негізгі бағдарламалау тұжырымдамалары туралы ақпарат аламыз. Ендеше, іске сәт! Бірінші бағдарламамызды жазып, бағдарламалау әлемімен іс жүзінде қауышайық.

Осы жерде сіздерді жігерлендіру мақсатында тағы бір замандасымыз - **Қасенов Айсұлтанның** естелігін келтіргім келеді:

“Мен 7 сыныпта оқып жүрген кезімде спорттық бағдарламалауды зерттеп, олимпиадаларға қатыса бастадым. Алайда 10 сыныптың соңына қарай қызығушылығым азайды. Себебі ол уақытта мен білетін IT мамандарының айтарлықтай жетістіктері жоқ еді. Жоғары оқу орнында “Electrical and Electronic Engineer” мамандығына түстім. 1 курсты бітірген соң “nFactorial Incubator”-ға түсіп, “React-Native”-тағы кросс-платформалық қосымшаларды бағдарламалауды үйрендім. Сол жылы жазда Google компаниясының инженер-бағдарламалаушысы **Қасенов Бекжанмен** таныстым. Бекжанмен сөйлескеннен кейін маған қосымшалар жасау ұнайтынын және компьютер ғылымын (Computer Science) тереңдетіп оқығаным дұрыс болатынын түсіндім. Көп ойланбай “Computer Science” факультетіне ауыстым. Ол жақтан көзқарасымыз, мақсатымыз бір достар таптым. Болашақ әріптестерімізбен бірге дамыдық, мақсатымыз да биіктей берді. Жақсы дайындалсам Big Tech (IT саласындағы ірі

компаниялар), FAANG (Facebook, Amazon, Apple, Netflix, Google) сынды компанияларда жұмыс жасау мүмкіндігімнің пайда болатынына көзім жетті. Жаңадан бағдарламалауды үйренгісі келетін адамдарды дайындайтын жерден, яғни инкубатордан өткеннен кейін менің IT маманы ретінде мансабым басталды. Содан бері инженер-бағдарламалаушы ретінде IT әлеміндегі жолымды жалғастырып келемін.

Жоғары оқу орнында білім ала жүріп, жұмыс жасап, Big Tech-ке қосымша дайындалуға уақыт бөле бастадым. Осылайша 2 курста Яндекске жұмысқа орналасу үшін әңгімелесуден өттім. Алайда олар маған Frontend әзірлеу базасы мен JavaScript негіздерін әлі де жақсы меңгермегенімді түсіндірді. Сынды дұрыс қабылдап, сол жылғы жаз айларын салалық әдебиеттерді оқуға арнадым, LeetCode арқылы біраз дайындалдым. Көбіне C++ тілінде тапсырмалар орындадым.

Күзде бірнеше компанияларға түйіндемемен жіберіп, нәтижесінде Facebook (қазіргі Meta) компаниясынан шақырту алдым. Бұл менің үлкен мүмкіндігім еді. Ең қызығы, Software Engineer ретінде өтініш жіберсем де, C++ тіліндегі тапсырмаларды орындағым келсе де, мені әңгімелесуге Frontend Engineer ретінде шақырыпты. Сондықтан дайындық стратегиясын өзгертіп, JavaScript бойынша терең талдаулар жасауыма тура келді. Әңгімелесу уақыты таяған сайын мендегі қобалжу сезімі де үдей түсті. Уайымнан қара терге түстім. Алайда әңгімелесу бастала салысымен уайымым сейілді. Алғашқы тапсырманы сәтті орындадым.

2 аптадан соң әңгімелесудің екінші кезеңі болды. Қобалжудың көкесін сол кезде таныдым. Берілген екі тапсырманы орындап, нәтижесін күттім. Жауабы шамамен 2-4 аптаға созылды. Уақыт өткен сайын мені жұмысқа шақырмайтын шығар деп ойладым. 2019 жылы желтоқсанда кадрлар бөлімінен хабарласып, әлемдік деңгейдегі компаниядан қызмет ұсынатындарын айтып, құттықтады. Бір өкініштісі, коронавирус індетінің тарауына байланысты ол жұмысқа бара алмадым. Есесіне Азимутқа жұмысқа орналасып, көп білім мен тәжірибе жинақтадым.

2021 жылы Facebook компаниясынан қайта шақырту келді. Мұндай мүмкіндіктен құр қалған болмас деп, келістім. Ол жақтағы жұмысымның басталуына әлі 9 ай бар болғандықтан, шетелдік Localize стартап жобасының IT бөліміне орналастым, JavaScript-пен өмірімде алғаш рет Ruby on Rails-та жұмыс

жасадым. Бұл мен үшін үлкен тәжірибе алаңы болды. Backend пен frontend әзірлеуші ретінде жан-жақты дамыдым.

Күзде Facebook-тегі жұмысыма жол тарттым. Ол жаққа алғаш барғандағы әсерім, сезімім тым ерекше болды. Маған берілген тапсырмаларды уақытылы, тіпті артығымен орындағандықтан басшылық тарапынан компанияда қалуға ұсыныс түсті. Әрине келістім. Келесі өндірістік тәжірибеге дейін әлі 8 айым бар еді. Осыны білген Localize өздеріне қайта жұмысқа шақырып, әлемдік компаниядан игерген тәжірибеммен бөлісуімді сұрады. Localize-да жұмыс жасау өзіме ұнайтын, әріптестермен де жақсы араласып кеткен едім. Жылы орнымды суытқым келмеген, дегенмен алда менің де, басқа IT мамандарының да арманы болған Big Tech әлемі күтіп тұр еді. Осындай қызық та қиын жолдар мені әлемнің үздік компанияларының бірі Meta-ға алып келді. Meta-ның қандай алпауыт екенін мен айтпасам да білесіздер.

Мықтылардың қатарынан табылуым үшін үнемі оқып, үздіксіз ізденіп, тәжірибе жинақтауым қажет. Өмір қызыққа толы ғой. Осы саланың арқасында әлемге саяхаттау, шетелде жұмыс жасау деген армандарым орындалуда. Жұмыс орным Лондонда. Ал Лондоннан кез келген жерге қатынау жеңілірек. Шынымен де армандар орындалады. Тек сеніңіздер!" - дейді Айсұлтан.

Айсұлтан өз тарапынан сіздерге келесі кеңесін айтады:

1. Ізденуді тоқтатпаңыз. Сынды жүрекке жақын қабылдамаңыз, сағыңыз сынбасын. Керісінше, сынға "тағы бір сатыға өсу мүмкіндігі" деп қараңыз.

2. "Маған досыңды көрсет, мен сенің кім екеніңді айтамын" деген нақыл сөз дәл айтылған. Себебі айналаңыздағы адамдарға қарап, бара жатқан бағытыңызды айқындайсыз, яғни жаныңызда жүрген жандардың талабы биік болса, сіздің де ұмтылысыңыз күннен күнге арта береді.

3. Дос демекші, жаныңызға адамгершілігі мол, өзіңізбен ниеттес адамдарды жиюға тырысыңыз.

4. Бағдарламалауда, жалпы өзіңе ұнайтын кез келген салада дамып отыру – қызықты әрі пайдалы процесс. Өмірден, ондағы әр құбылыстан ләззат алыңыз.

Міне, бізге Айсұлтан замандасымыз өз ойын осылай жеткізуде. Енді қайтадан басты тақырыбы-мызға оралайық.

Python бағдарламау тілін орнату

Windows-қа Python орнату нұсқаулығы

1. Орнатушыны жүктеп алу

- Python-ның ресми сайтына кіріңіз
<https://www.python.org/downloads/windows/>
- Жүктеулер бөліміне өтіңіз
- "Download Python 3.x.x" (соңғы нұсқа) түймесін басыңыз

Python >>> Downloads >>> Windows

Python Releases for Windows

- [Latest Python 3 Release - Python 3.11.3](#)

Stable Releases

- [Python 3.10.11 - April 5, 2023](#)
Note that Python 3.10.11 *cannot* be used on Windows 7 or earlier.
 - Download [Windows embeddable package \(32-bit\)](#)
 - Download [Windows embeddable package \(64-bit\)](#)
 - Download [Windows help file](#)
 - Download [Windows installer \(32-bit\)](#)
 - Download [Windows installer \(64-bit\)](#)

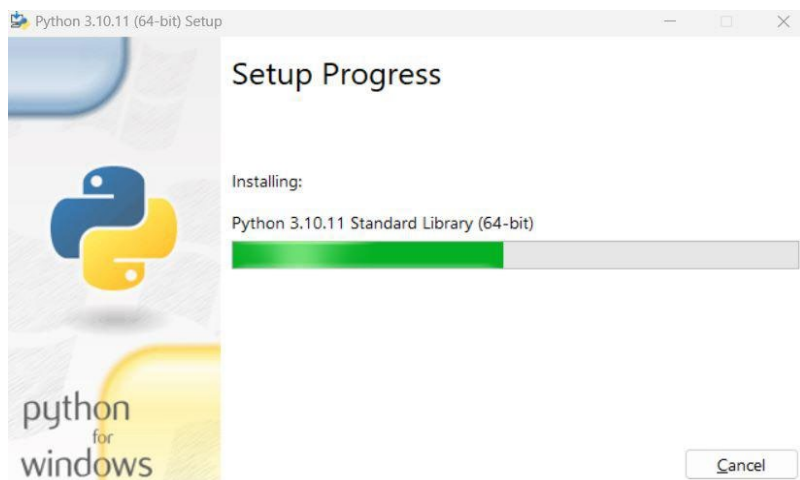
2. Орнатушыны іске қосу

- Жүктелген .exe файлын табыңыз (мысалы, python-3.x.x.exe)
- Оны екі рет басып, іске қосыңыз



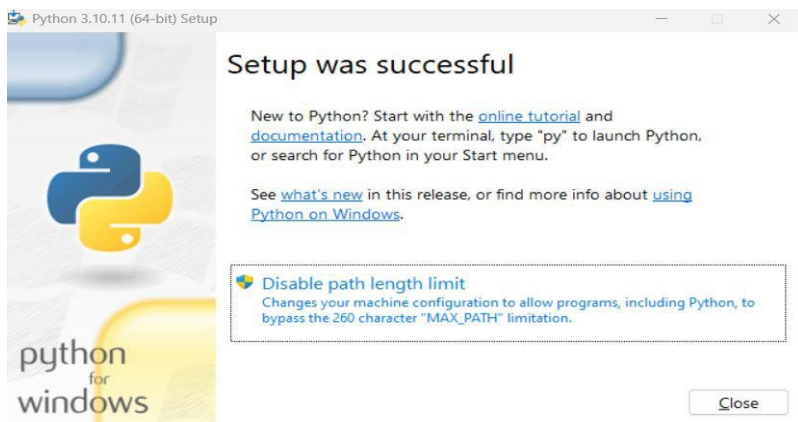
3. Орнатуды баптау

- Орнатушы терезесінің төменгі жағындағы "Add Python to PATH" қанатшаны (птичка) белгілеңіз
- Стандартты орнату үшін "Install Now" түймесін (кнопка) басыңыз
- Немесе қосымша опциялар үшін "Customize Installation" түймесін таңдаңыз



4. Орнатуды аяқтау

- Орнатушының файлдарды көшіріп, Python-ды жүктеп болуын күтіңіз
- Аяқталған соң орнатушыны жабыңыз



Орнатуды тексеру

1. Пәрмен жолды (командная строка) ашыңыз ("Win + R"-ді басып, "cmd" теріңіз, "Enter"-ді басыңыз).
2. **"python --version"** деп теріп, "Enter"-ді басыңыз.
3. Орнатылған Python нұсқасын тексеріп көруіңізге болады (мысалы, Python 3.10.11).

Қосымша: "pip"-тің орнатылуын тексеру

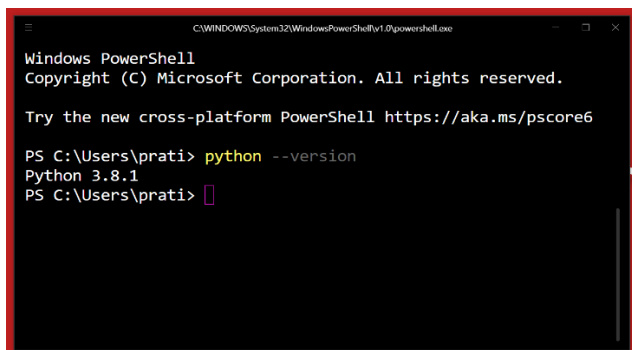
Пәрмен жолға (командная строка) **"pip --version"** деп теріп, "Enter"-ді басыңыз. Бұл "pip"-тің (Python пакеттерін орнатушы) дұрыс орнатылғанын тексереді.

Осы қадамдарды орындау арқылы сіз компьютері-ңіздегі Windows жүйесіне Python-ды сәтті орнатып, бағдарламалауға дайын боласыз.

Орнату аяқталған соң, пәрмен жолды ашып ("Windows + R" батырмаларын басып, содан кейін ұяшыққа "cmd" теріп ашуға) алуға болады. Python орнатылғанын тексеру үшін төмендегіні жазыңыз:

```
Python  
python --version
```

Егер Python нұсқасы көрсетілсе, онда бәрі дұрыс орнатылған.



```
C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\prati> python --version
Python 3.8.1
PS C:\Users\prati> 
```

Алғашқы бағдарлама

Python-ды үйрену үшін алғашқы кодты терминалда жазамыз. Пәрмен жол интерфейсі (CLI) немесе “shell” деп аталатын терминал – бұл компьютердің операциялық жүйесімен мәтін арқылы өзара әрекеттесуге арналған интерфейс. Ол графикалық пайдаланушы интерфейсін (GUI) қолданудың орнына мәтіндік пәрмендерді теру арқылы пәрмендерді орындауға, бағдарламаларды іске қосуға, файлдық жүйені шарлауға және әртүрлі тапсырмаларды орындауға мүмкіндік береді. Терминалдар әдетте “Linux” және “macOS” сияқты “Unix” қолданылады. Біз қазір терминалды ашып, “Сәлем, әлем!” Python бағдарламасын жазып көрейік.

1. **Терминалды ашыңыз:** Windows-та пәрмен жолын “Іске қосу” (Пуск) мәзірінде “Пәрмен жолы” (Командная строка) деген мәтінді іздеу арқылы ашуға болады. Linux/macOS-та терминал қолданбасын әдетте қолданбаларыңыздың “Утилиттер” немесе “Қосымша” (Приложения) папкасынан таба аласыз.

2. **Python нұсқасын тексеріңіз:** Жоғарыдағы нұсқауларды оқығаннан кейін Python-ды орнатқан болсаңыз, оның нұсқасын тексеріңіз. Python орнатылғанын көру үшін “python --version” немесе “python3 --version” деп теруге болады. Егер орнатылмаған болса, оны қайта орнату қажет болады.

3. **Python-ды ашу:** Python пәрмен жолын іске қосу үшін “python” немесе “python3” деп теріп, “Enter” батырмасын басыңыз. Нәтижесінде “Python” пәрмен жолын көре аласыз `>>>`.

4. **“Сәлем, әлем!” бағдарламасын жазыңыз:** Пәрмен жолға келесі Python кодын теріңіз:

```
Python
>>> print("Сәлем, әлем!")
"Сәлем, әлем!"
```

5. **Бағдарламаны іске қосыңыз:** кодты орындау үшін “Enter” пернесін басыңыз. Сіз “Сәлем, әлем!” деген сөздің терминалға басып шығарылғанын көруіңіз керек.

Міне, осылайша алғашқы бағдарламамызды жазып бітіреміз. Алда бізді бұдан да күрделі әрі қызықты бағдарламалар күтіп тұр.

Python негіздері: айнымалылар, деректер типтері, операторлар және негізгі бағдарламалау түсініктері.

Айнымалылар

Python-дағы айнымалы – деректерді сақтау үшін жадыдағы орынға тағайындайтын атау. Сіз мұны мәндерді сақтауға арналған контейнер ретінде қарастыра аласыз. Айнымалы атаулар әріптерден, сандардан және астыңғы сызықтан тұруы мүмкін, бірақ олар саннан басталмайды. Мысалы:

```
Python
>>> x = 5
>>> x
5
>>> name = "Айжан"
>>> name
"Айжан"
```

Бұл мысалдағы “x” және “name” – айнымалылар. Мұндағы “x” бүтін санды, ал “name” жолды сақтайды.

Деректер типтері

Python – динамикалық түрде типтендірген бағдарламалау тілі, яғни мұнда айнымалының деректер типін анық жариялаудың қажеті жоқ. Python оған тағайындалған мән

негізінде деректер типін автоматты түрде анықтайды. Python тіліндегі ең жиі қолданылатын деректер типі мыналар:

- **int**: Бүтін сандар. Мысалы: 5, -10, 42.
- **float**: Бөлшек сандар: Мысалы: 3.14, -0.5, 2.0.
- **bool**: Логикалық мәндер. Мысалы: True немесе False.

Олар ағылшын тілінен аударғанда “Дұрыс” және “Бұрыс” деген мағынаны білдіреді.

- **str**: Жолдар. Мысалы: "Сәлем, әлем!".
- **list**: Мәндердің реттелген жинағы. Мысалы:

[1, 2, 3, 1].

- **tuple**: Мәндердің өзгермейтін реттелген жинағы.

Мысалы: (1, 2, 3, 1).

- **dict**: Кілт-мән жұптары. Мысалы: {"аты": "Әлия", "жас": 30}.
- **set**: Қайталанбайтын элементтерді сақтайтын деректер

құрылымы. Мысалы: {1, 2, 3}

Осы деректер типінің ішіндегі **int**, **float**, және **bool** – қарадүрсін (примитив) деректер. Бағдарламалау тілінде ең қарапайым және негізгі дерек түрлерін қарадүрсін (примитивті) деректер типі деп атаймыз. Нақ осы қарадүрсін деректер типі Python тіліндегі күрделі деректер құрылымдарының негізін құрайды: тізімдер (**list**), кортеждер (**tuple**), сөздіктер (**dict**) және жиындар (**set**). Бірақ олар қарадүрсін деректер типіне жатпайды.

Қарадүрсін деректер типі бағдарламалау кезінде негізгі деректерді сақтау және өңдеу үшін қолданылады.

Айнымалы деректер типін “**type()**” функциясы арқылы тексеруге болады:

Python

```
>>> x = 5
>>> print(type(x)) # Шығару: <class 'int'>
<class 'int'>
```

Жоғарыдағы мысалдың шығарылуы – **<class 'int'>** болады. Сондай-ақ, **#** таңбасы мен одан кейінгі мәтін түсініктемені білдіреді.

Python-да түсініктемелер сіздің кодыңызға түсіндірме жазбалар немесе ескертулер қосу үшін қолданылады. Түсініктемелер бағдарламаның бөлігі ретінде орындалмайды және кодты жақсы түсінуге арналады. Python түсініктемелерінің екі түрі бар:

1. Бір жолды түсініктемелер: бір жолды түсініктеме жасау үшін “#” таңбасын пайдалануға болады. Сол жолдағы “#” таңбасынан кейінгі барлық нәрсе код ретінде емес, түсініктеме ретінде қарастырылады.

Python

```
>>> # Бұл бір жолды түсініктеме
>>> print(type(x)) # Бұл түсініктеме код жолының
соңында орналасқан
```

2. Көп жолды түсініктемелер: ұзағырақ түсініктеме-лер немесе құжаттама үшін Python үштік тырнақшаға (""" немесе """) салынған көп жолды мәтіндерді пайдаланады.

Python

```
'''
Бұл - көп жолды түсініктеме.
Ол бірнеше жолды қамтуы мүмкін және жиі құжат тізбегі
ретінде пайдаланылады.
'''

'''
Көп жолды түсініктеме жасаудың тағы бір жолы.
'''
```

Операторлар:

Python-дағы операторлар айнымалылар мен мәндермен операцияларды орындау үшін қолданылады. Операторлардың бірнеше түрі бар:

1. **Арифметикалық операторлар:** Python тілінде арифметикалық операторлар сандар (бүтін сандармен де, ондық бөлшектермен де) мен негізгі математикалық операцияларды орындау үшін қолданылады. Мұнда бес негізгі арифметикалық оператордың түсіндірмесі берілген. Олар:

- **Қосу (+)**
 - + операторы екі санды қосу үшін қолданылады.
 - Мысал: $5 + 3$ операциясы 8 -ге тең.
- **Азайту (-)**
 - - операторы екі санды азайту үшін қолданылады.
 - Мысал: $10 - 4$ операциясы 6 -ға тең.
- **Көбейту (*)**

- * операторы екі санды көбейту үшін қолданылады.
- Мысал: $6 * 5$ операциясы 30-ға тең.
- **Бөлу (/)**
 - / операторы екі санды көбейту үшін қолданылады.
 - Мысал: $15 / 3$ операциясы 5.0-ға тең. Бөлудің нәтижесі дәл болса, нәтиже ондық бөлшекке тең болмақ.
- **Дәреже (**)**
 - ** операторы екінші санның дәрежесіне шығыру үшін қолданылады.
 - Мысал: $2 ** 4$ операциясы 16-ға тең.

Бірнеше қосымша ескертулер мен мысалдар келтіре кетейік:

- Егер / операндтарының екеуі де бүтін сандар болса және бөлу дәл болмаса, Python бөлімнің бөлшек бөлігін сақтау үшін ондық бөлшекті қайтарады. Мысалы, $7 / 3$ операциясы 2.3333...-ке тең...
- Бөлшек бөлігін қысқартатын және бүтін нәтижені қайтаратын бүтін бөлуді орындағыңыз келсе, // операторын пайдалануға болады. Мысалы, $7 // 3$ операциясы 2-ге тең.
- Операциялардың орындалу ретін көрсету үшін жақшаларды қолдануға болады. Мысалы, $(5 + 3) * 2$ операциясы 16-ға тең, өйткені жақшаның ішіндегі қосу амалы бірінші орындалады.

Python тілінде арифметикалық операторларды айнымалыларға да пайдалануға болады. Мысалы:

Python

```
>>> x = 7
>>> y = 3
>>> нәтиже = x + y # нәтиже айнымалысында 10 саны бар
```

2. Салыстыру операторлары: Python-дағы салыстыру операторлары мәндерді салыстыру және олардың арасындағы байланысты анықтау үшін қолданылады. Бұл операторлар салыстыру нәтижесіне негізделген логикалық мәндерді (“True” немесе “False”) қайтарады. Python-дағы негізгі салыстыру операторларының түсіндірмесі:

- **Тең (==):**
 - Синтаксис: $a == b$
 - Сипаттама: a мәні b мәніне тең екенін тексереді.
 - Мысал: $5 == 5$ операциясы “True”, ал $5 == 10$ “False” мәнін қайтарады.

- **Тең емес (!=):**
 - Синтаксис: $a \neq b$
 - Сипаттама: a мәні b мәніне тең емес екенін тексереді.
 - Мысал: $5 \neq 5$ операциясы “False”, ал $5 \neq 10$ “True” мәнін қайтарады.
 - **Кіші (<):**
 - Синтаксис: $a < b$
 - Сипаттама: a мәні b мәнінен кіші екенін тексереді.
 - Мысал: $5 < 10$ операциясы “True”, ал $10 < 5$ “False” мәнін қайтарады.
 - **Үлкен (>):**
 - Синтаксис: $a > b$
 - Сипаттама: a мәні b мәнінен үлкен екенін тексереді.
 - Мысал: $10 > 5$ операциясы “True”, ал $5 > 10$ “False” мәнін қайтарады.
 - **Кіші немесе тең (<=):**
 - Синтаксис: $a \leq b$
 - Сипаттама: a мәні b мәнінен кіші немесе оған тең екенін тексереді.
 - Мысал: $5 \leq 10$ операциясы “True”, $10 \leq 10$ “True”, ал $10 \leq 5$ “False” мәнін қайтарады.
 - **Үлкен немесе тең (>=):**
 - Синтаксис: $a \geq b$
 - Сипаттама: a мәні b мәнінен үлкен немесе оған тең екенін тексереді.
 - Мысал: $10 \geq 5$ операциясы “True”, $10 \geq 10$ “True”, ал $5 \geq 10$ “False” мәнін қайтарады.
3. **Логикалық операторлар:** Python тіліндегі логикалық операторлар логикалық мәндерді (“True” немесе “False”) біріктіру немесе өңдеу үшін күрделі шешімдер немесе өрнектерді қабылдау үшін пайдаланылады. Python үш негізгі логикалық операторды қолдайды: **and**, **or**, және **not**. Әрқайсысының түсіндірмесі:
- **and:**
 - **and** операторы егер оның операндтарының екеуі де “True” болса, “True” мәнін, ал кері жағдайда “False” мәнін қайтарады.
 - Мысал: “True and True” мәні “True” ретінде бағаланады, ал “True and False” мәні немесе “False and True” мәні “False” ретінде бағаланады.

- Мысал: “True and True and False” мәні “False” ретінде бағаланады. Біз білетіндей “True and True” операциясы “True”-ге тең, яғни біз бұл операцияны жай “True” деген мәнге қысқарта аламыз. Қысқартқаннан кейін, біздің өрнегіміз “True and False”-қа тең болады. True and False алдында айтылып кеткендей False-қа тең.

Python

```
>>> x = True
>>> y = False
>>> нәтиже = x and y # нәтиже False-қа тең
```

- **or:**
 - “or” оның операндтарының кем дегенде біреуі “True” болса, “True” мәнін, ал егер екі операнд да “False” болса, “False” мәнін қайтарады.
 - Мысал: “True or True, True or False” және “False or True” мәндері “True” ретінде бағаланады, ал “False or False” мәні “False” ретінде бағаланады.
 - Мысал: True and False or False өрнегі False мәніне тең. True and False бөлімінде айтылғандай False мәніне тең. Осы себеппен біз True and False-ты False-қа қысқартамыз. False or False өрнек False-қа тең.

Python

```
>>> x = True
>>> y = False
>>> нәтиже = x or y # нәтиже True-ға тең
```

- **not:**
 - not операторы - бұл операндтың қарама-қарсы логикалық мәнін қайтаратын унарлы оператор. Егер операнд “True” болса, not операциясы False-ты қайтарады, ал егер операнд False болса, not операциясы True-ды қайтарады.
 - Мысал: not False = True, ал not True = False

Python

```
>>> x = True
>>> нәтиже = not x # нәтиже False-қа тең
```

4. Python-да тағайындау операторлары айнымалыға жаңа мән берген кезде оның мәнін өзгерту үшін қолданылады. “=”, “+=” және “-=” операторлары жиі қолданылатын үш тағайындау операторы болып табылады.

- **+= (Қосу тағайындауы):**

- += операторы айнымалының ағымдағы мәніне басқа мәнді қосып және нәтижені айнымалыға қайта тағайындау үшін қолданылады.
- Бұл `variable = variable + value` деп жазудың қысқартылған жолы.

Python

```
>>> x = 5
>>> x += 3 # x = x + 3 деп жазумен бірдей
>>> print(x)
8
```

- **-= (Азайту тағайындауы):**

- -= операторы айнымалының ағымдағы мәнінен басқа мәнді азайтып және нәтижені айнымалыға қайта тағайындау үшін қолданылады.
- Бұл `variable = variable - value` деп жазудың қысқартылған жолы.

Python

```
>>> y = 10
>>> y -= 4 # y = y - 4 деп жазумен бірдей
>>> print(y)
6
```

Python-да көптеген күрделі операторлар бар. Мысалы, сәйкестендіру операторлары (`is`, `is not`), тиесілік операторлары (`in`, `not in`). Біз оларды келесі тарауларда қарастырамыз.

Шартты операторлар

Шартты операторлар — бұл бағдарламалауда компьютерге белгілі бір шарттар немесе критерийлерге сүйене отырып шешім қабылдауға мүмкіндік беретін негізгі құрал. Яғни, олар қандай да бір шарт дұрыс (True) болса, бір әрекетті, ал шарт бұрыс (False) болса, басқа әрекетті орындауға мүмкіндік береді.

Мұндай операторларсыз бағдарлама әрқашан бір бағытта орындалар еді, яғни ешқандай шешім қабылдау мүмкіндігі болмас еді. Ал шартты операторлар арқылы бағдарлама түрлі жағдайларға икемделіп, «ойлай» алады.

Нақты өмірден мысал: Саябаққа саяхат

“Саябаққа бару керек пе, жоқ па?” деген мәселені дұрыс шешкіңіз келетін жағдайды елестетіп көріңіз. Сіз екі шартты қарастыра аласыз:

- Ауа-райы: егер күн ашық болса, саябаққа барасыз, ал жаңбыр жауып тұрса немесе жаңбыр жауатыны туралы білсеңіз, үйде қаласыз.
- Үй тапсырмасы: егер сіз үй тапсырмасын аяқтасаңыз, саябаққа барасыз. Керісінше, үй тапсырмасын аяқтамаасаңыз, үйде қаласыз.

Енді осы шарттарды қарапайым тілде бағдарламамен байланыстырып көрейік:

Егер осы шарт дұрыс болса, онда осы әрекетті жасаңыз, әйтпесе осыны жасаңыз. Біздің мысалда бұл: "егер күн ашық болса және үй тапсырма орындалса (шарт дұрыс болса), саябаққа барамын, әйтпесе (шарт бұрыс болса), үйде қаламын".

Логикалық операторлар – шарттарды салыстыруға көмектесетін сөздер. Мысалы, сіз шешім қабылдау үшін үй тапсырмасының күйін салыстырасыз. Сіз "егер мен үй тапсырмасын аяқтасам (1-шарт) және күн ашық болса (2-шарт), онда саябаққа барамын" деген шарт қоясыз. Саябаққа бару үшін екі шарт та орындалуы керек. Егер шарттардың бірі дұрыс болмаса (сіз үй тапсырмасын аяқтай алмадыңыз немесе сыртта күн бұлтты), сіз үйде қаласыз.

Python

```
>>> if үй_тапсырмасы_аяқталды and күн_ашық:  
>>>     print("саябаққа бару")  
>>> else:  
>>>     print("үйде қалу")
```

Мұндағы:

if – “егер” деген мағынаны білдіреді;

and – екі шарттың да дұрыс болуын талап етеді;

else – “әйтпесе” дегенді білдіреді.

Яғни, егер екі шарт та дұрыс болса (үй тапсырмасы орындалған және күн ашық), онда “саябаққа бару” деп экранға шығарылады. Ал шарттардың бірі дұрыс болмаса, “Үйде қалу” деп шығады.

Логикалық операторлар

Шартты операторлармен бірге жиі қолданылатын арнайы логикалық операторлар бар:

Оператор	Мағынасы	Мысал
and	Екі шарт та дұрыс болса ғана орындалады	if a > 5 and b < 10:
or	Екі шарттың кем дегенде біреуі дұрыс болса орындалады	if a > 5 or b < 10:
not	Шартты терістейді (яғни, керісінше етеді)	if not күн_ашық:

Табуляция (ендірме) туралы

Бағдарламалауда табуляция (ендірме) – кодтың құрылымын көрсету үшін қолданылады.

Python тілінде бұл өте маңызды, себебі шарттың ішіндегі әрекеттер дәл осы ендірмелер арқылы анықталады.

Мысалы:

Python

```
if күн_ашық:
    print("Саябаққа барамын") # Бұл жол if-ке тиесілі
    print("Достармен кездесу") # Бұл да if блогының бөлігі
else:
    print("Үйде қаламын") # Бұл жол else блогына тиесілі
```

Мұнда **print()** қатарлары ендірімемен (tab) басталады. Егер табуляция дұрыс қойылмаса, бағдарлама қате береді. Осылайша, табуляция – бағдарлама логикасының дұрыс жұмыс істеуін қамтамасыз ететін құрылымдық белгі.

Қорытынды

Қорыта айтқанда, бағдарламадағы шартты операторлар компьютерге берілетін нұсқаулардың қызметін атқарады. Сіз белгілі бір шарттарға немесе ережелерге сүйене отырып, компьютерге не істеу керектігін айтасыз. Бұл нұсқаулар компьютерге шешім қабылдауға және шарттар мен ережелерге сүйене отырып, күнделікті өмірде шешім қабылдағандай әртүрлі әрекеттерді орындауға көмектеседі..

Функциялар: Python тілінде функцияларды қалай жазуға және пайдалануға болады.

Функция дегеніміз не?

Функциялар – белгілі бір тапсырманы бағдарламалау тілінде орындайтын код блогы. Функциялар арқылы жиі қолданатын немесе көп қайталанатын тапсырмаларды біріктіруге болады. Функциялардың маңызын түсіну үшін келесі мысалдарға назар аударыңыз.

Тапсырма: сізге екі санды қосып, нәтижесін онға көбейтіп шығару қажет. Алдымен операторлар мен айнымалыларды қолданып жазамыз:

Python

```
>>> a = 2
>>> b = 4
>>> қосынды = (a + b)
>>> нәтиже = қосынды * 10
>>> print(нәтиже)
```

Ал енді тек 2 мен 4 емес, осы операцияда 3 пен 5-ті орындау қажет болса:

Python

```
>>> a = 2
>>> b = 4
>>> қосынды = (a + b)
>>> нәтиже = қосынды * 10
>>> print(нәтиже)
60
>>> a = 3
>>> b = 5
>>> қосынды = (a + b)
>>> нәтиже = қосынды * 10
>>> print(нәтиже)
80
```

Кодтың бірнешеуі қайталанғанын байқадыңыз ба? Енді бар операциялармен бірге 6 мен 7-ні орындау қажет болса:

Python

```
>>> a = 2
>>> b = 4
>>> қосынды = (a + b)
>>> нәтиже = қосынды * 10

>>> print(нәтиже)
60
>>> a = 3
>>> b = 5

>>> қосынды = (a + b)
>>> нәтиже = қосынды * 10
>>> print(нәтиже)
80
>>> a = 6
>>> b = 7
>>> қосынды = (a + b)
>>> нәтиже = қосынды * 10
>>> print(нәтиже)
130
```

Байқасаңыз, кодтың біраз бөлігі қайталанып кетті. Егер осылай әрбір екі сан сайын қайталай берсек, көп қате жібере аламыз.

Енді қате жібермеу үшін және кодты біршама қысқарту мақсатында келесі функцияны қолданып көрейік. Мысалы:

Python

```
>>> def operation(a, b):
>>>     қосынды = (a + b)
>>>     нәтиже = қосынды * 10
>>>     print(нәтиже)

>>> operation(2, 4)
60
>>> operation(3, 5)
80
>>> operation(6, 7)
130
```

Кодтың жинақы бола түскенін байқадыңыз ба? Сондай-ақ қателерді табу да жеңілірек болады.

Осы мысал арқылы Python-да функция неден тұратынын қарап көрейік:

- **def:** функцияның басын белгілейтін арнайы сөз. Кез келген функция әрдайым осы сөзбен белгіленеді.
- **operation:** латын қарпінде жазылған функцияның атауы. Осы мысалда оның атауын “operation” деп қойдық, бірақ басқа да атаулар қоюға болады. Мысалы “my_operation”, “my_function”, тіпті “operation1”, “operation2” деп сандарды да қосуға болады.
- **a, b:** функцияның аргументтері. Оларды тек сол функцияда белгіленген айнымалылар ретінде қарастыруға болады. Біздің тапсырма үшін кез келген екі санды қосып, онға көбейту қажет болды. Мұндағы екі санды белгілейтін екі айнымалы **a** және **b** – функцияның аргументтері.
- функцияның денесі: 2-4 жолдар “функцияның денесі” деп есептеледі. Мұнда бізге қажет операцияларды белгілейміз.

Функцияны жазғаннан кейін оны аты арқылы шақыруға болады. Жоғарыдағы мысалда функцияны **operation(2, 4)**, **operation(3, 5)**, **operation(6, 7)** деп қолданамыз, яғни атынан кейін екі жақшада аргументтерді белгілейміз.

Функцияның атауы қандай бола алады?

Сізді жоғарыда **print()** функциясымен таныстырған болатынбыз. **operation()**-ның функциядан айырмашылығы

мынада: `print()` Python тіліне енгізілген, ал `operation()` -ды өзіңіз қостыңыз. Осы тұста есте сақтайтын бір мәселе бар: әр функцияның атауы басқа болуы тиіс. Мысалы, “`print`” деген атаумен тағы бір функция құрастыруға болмайды:

Python

```
>>> def print():
>>>     # функцияның денесі
```

Осы мысалды Python бағдарламалау тіліне енгізіп көрсеңіз, қате туралы хабарды байқайсыз.

Функцияның денесі

Мысал ретінде келесі функцияны қарастырып көрейік:

Python

```
>>> def my_function(num_1):
>>>     num_2 = num_1 + 10
>>>     нәтиже = num_1 * 10
```

`my_function(num_1)` функциясы белгіленген, оның денесіне “`num_2 = num_1 + 10`” жолы кіретіні анық. Бірақ “`нәтиже = num_1 * 10`” функцияның денесіне жатпайды, себебі ол жолда табуляция сақталмаған. Функцияның денесі “`def`” сөзі арқылы белгіленген келесі тізбектен басталады және бірдей шегіністе үзілмей жазылған жолдардың барлығы сол функцияның денесі болып саналады.

Функцияның нәтижесі

Функцияда біршама операцияларды қолданып, нәтижесін қайтаруға болады. Функцияның нәтижесін қайтаруға арнайы “`return`” сөзі қолданылады. Мысалы:

Python

```
>>> def my_function(a, b):
>>>     қосынды = (a + b)
>>>     нәтиже = қосынды * 10
>>>     return нәтиже
>>> нәтиже = my_function(2, 3)
>>> print(нәтиже)
50
```

Осы мысалда функция екі санды қосып, оны онға көбейтеді де нәтижесін **“return”** арқылы қайтарады. Ал кодтың негізгі бөлімінде функцияға 2 және 3 аргументтері жіберіліп, нәтижесі **“нәтиже”** айнымалысында сақталған.

Циклдер: қайталанатын тапсырмаларды автоматтандыру үшін циклдарды қалай пайдалануға болады?

Бұл тарауда біз цикл конструкциясын меңгереміз. Циклдер – Python тіліндегі қайталанатын тапсырмаларды автоматтандыруға арналған бағдарламалау нұсқасы. Олар бір кодты қайта-қайта жазбай-ақ, компьютерге бұйрық беру арқылы белгілі бір тапсырманы бірнеше рет орындатуға мүмкіндік береді. Мысалы, сізге «Сәлем!» деп бес рет жазу тапсырылды дейік. Циклдерсіз бір код жолын бес рет теруге тура келер еді. Бұл – әрине ұзақ. Бірақ қорықпаңыз. Бір әрекетті қалағаныңызша қайталауға мүмкіндік беретін ілмектер сіздің уақытыңызды үнемдеуге көмектеседі.

“For” циклі

Тапсырманы қанша рет қайталау қажетін алдын ала білетін болсаңыз, **«for»** циклі сіздің негізгі қолданатын құралыңыз болады. Енді “Сәлем!” деп бес рет басып шығарғыңыз келді дейік.

Python

```
>>> for _ in range(5):  
...     print("Сәлем!")  
...  
Сәлем!  
Сәлем!  
Сәлем!  
Сәлем!  
Сәлем!
```

Мұнда **range(5)** функциясы 0-ден 4-ке дейінгі сандардың тізімін құру үшін қолданылады және **«for»** циклі «Сәлем!» жолын әрбір итерацияда терминалға шығарады. Цикл құрастырудың маңызды ережесі – табуляция сақтау. Циклдің денесіне функция секілді бірдей шегіністе үзілмей жазылу керек. Мәселен, төменде берілген функцияда “Сау бол!” жолы шегінуді сақтамағаннан кейін **for** цикл денесіне кірмей қалады.

Python

```
>>> def амандасу():
...     for _ in range(5):
...         print("Сәлем!")
...         print("Сая бол!")
...
>>> амандасу()
Сәлем!
Сәлем!
Сәлем!
Сәлем!
Сәлем!
Сая бол!
```

range

range функциясын жақынырақ талқылайық. **range** функциясы үш негізгі параметрді қабылдай алады:

- Басы (start) – диапазон басталатын сан. Егер басқаша көрсетілмесе, әдепкі мәні 0 болады.
- Соңы (stop) – диапазон аяқталатын сан. Бұл сан диапазонға кірмейді.
- Қадам (step) – әр санның қаншаға өсетінін немесе кемітетінін көрсететін сан. Әдепкі мәні 1-ге тең.

Мысалдар:

Жай қолдану: **range(5)** дегеніміз 0-ден 4-ке дейінгі сандарды береді. Бұл 0, 1, 2, 3, 4 деген сандарды білдіреді.

Python

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

Бұл цикл 0-ден 4-ке дейінгі сандарды басып шығарады. Басы және соңы: **range(2, 6)** дегеніміз 2-ден 5-ке дейінгі сандарды береді. Бұл 2, 3, 4, 5 деген сандарды білдіреді.

Python

```
>>> for i in range(2, 6):  
...     print(i)  
...  
2  
3  
4  
5
```

Қадамды қолдану: `range(1, 10, 2)` дегеніміз 1-ден 9-ға дейінгі сандарды 2 қадаммен береді. Бұл 1, 3, 5, 7, 9 деген сандарды білдіреді.

Python

```
>>> for i in range(1, 10, 2):  
...     print(i)  
...  
1  
3  
5  
7  
9
```

Кему: егер қадам теріс мән болса, range кері бағытта жұмыс істейді. Мысалы, `range(10, 0, -1)` дегеніміз 10-нан 1-ге дейінгі сандарды береді.

Python

```
>>> for i in range(10, 0, -1):  
...     print(i)  
...  
10  
9  
8  
7  
6  
5  
4  
3  
2  
1
```

“While” циклі

«while» циклі ол белгілі бір шарт орындалғанша дейін тапсырманы қайталай береді. Мысалы, 1-ден 5-ке дейін санау үшін «while» циклін пайдалануға болады:

Python

```
>>> санауыш = 1
>>> while санауыш <= 5:
...     print(санауыш)
...     санауыш += 1
..
1
2
3
4
5
```

Бұл мысалда цикл **санауыш** 5-тен аз немесе оған тең болғанша жалғасады. **санауыш += 1** операторы әрбір итерацияда санауыштың ұлғаюын қамтамасыз етеді.

Циклді бұзу және жалғастыру

Циклдерде олардың әрекетін басқаруға арналған қосымша құрал бар:

- **break** – бұл қалған итерацияларды өткізіп жіберіп, циклден бірден шығады.

- **continue** – бұл ағымдағы итерация үшін цикл ішіндегі кодтың қалған бөлігін өткізіп жібереді және келесі итерацияға көшеді.

Python

```
>>> for сан in range(10):
...     if сан == 5:
...         break # Сан 5 болғанда циклді үзіңіз
...     print(сан)
...
0
1
2
3
4
```

Бұл мысалда цикл «**сан**» 5 болғанда үзіліп, сол нүктеден кейінгі сандарды басып шығаруға жол бермейді.

Python

```
>>> for сан in range(10):
...     if сан == 5:
...         continue # Сан 5 болғанда, келесі
итерацияға көшеді
...     print(сан)
...
0
1
2
3
4
6
7
8
9
```

Бұл мысалда цикл «**сан**» 5 болғанда, цикл ішіндегі кодтың қалған бөлігін өткізіп жіберіп, келесі итерацияға көшеді.

Кірістірілген циклдер

Циклдерді кірістіруге болады, бұл қайталанудың күр-делі үлгілерін жасауға мүмкіндік береді. Мысалы, кірістірілген циклді көбейту кестесін құру үшін пайдалана аламыз:

Python

```
>>> for i in range(1, 6):
...     for j in range(1, 6):
...         print(i * j, end="\t")
...     print()
...
1 2      3      4      5
2 4      6      8      10
3 6      9      12     15
4 8      12     16     20
5 10     15     20     25
```

Мұнда сыртқы цикл алғашқы бес санды қайталайды, ал ішкі цикл осы сандардың әрқайсысын алғашқы бес санға көбейтіп, тор тәрізді көбейту кестесін жасайды.

Қорытынды

Қайталанатын тапсырмаларды автоматтандыруға қатысты циклдерді көмекшіңіз деп есептеуіңізге болады. 'for' және 'while' циклдерімен, сондай-ақ 'break' және 'continue' сияқты қосымша кілт сөздердің көмегімен сізде компьютерге әрекеттерді бірнеше рет орындауға пәрмен беру мүмкіндігі бар. Циклдерді қолданып машықтаныңыз және Python кодыңызға автоматтандыру мен тиімділік әкелетінін қараңыз. Келесі тарауда біз функциялар әлемі – код блоктарын пакеттеуге және қайта пайдалануға мүмкіндік беретін тағы бір қуатты құралды қарастырамыз.

Келесі тарауға көшпес бұрын Лондондағы Meta компаниясында бағдарламалаушы болып еңбек етіп жүрген **Дәнекер Нұрғалиева** атты замандасымыздың төмендегі пікірін назарларыңызға ұсынғым келеді:

“Назарбаев Университетін “Компьютерлік ғылымдар” мамандығы бойынша бітірдім. Ал бағдарламалау тілімен алғаш рет Қарағандыдағы Қазақ – түрік лицейінің (қазіргі Білім-инновация лицейі - ред.) 10 сыныбында таныстым. Лицейдегі информатика сабағында бірнеше апта бойы C++ бағдарламалау тілінің кіріспе тақырыбын өттік. Неліктен мұғалімдер дәл осы тілді бағдарламалаумен танысу үшін таңдағанын білмеймін. Алайда ол, әсіресе біз сияқты жаңадан бастағандар үшін меңгеру тұрғысынан ең күрделі тілдердің бірі болғаны анық. Код жазудың қиындығы менің бағдарламалауға деген қызығушылығымды арттырып, Назарбаев Университетінің “Компьютерлік ғылымдар” мамандығына түсуіме себеп болған шығар деп ойлаймын.

Алғашқы оқу жылы мен үшін қиын болды. Себебі негізгі құрылымды білгеніммен, кодты өз бетімше жазуды түсінбей жүрдім. Бір лабораториялық жұмыста ұстазымыздың көмекшісі үй тапсырмасын сұрап, мен шешімін түсіндіріп бере алмадым. Сонда ол мамандығымды ауыстыруға кеңес беріп, менің бұл саланы меңгеріп кете алмайтынымды айтты. Сол кісінің айтқан сын-ескертпесі санамда қатты сақталып қалып, менің көбірек оқып, алға қарай талпынуыма ықпалын тигізді.

Әсіресе STEM саласындағы қыздарға өмірде осындай жағдайлардың жиі кездесетінін айтқым келеді. Сіздің

тәжірибеңіздің аздығын “ақымақтық” деп қабылдауы мүмкін немесе басқалар “бұл сала сізге арналмаған” деп сендіргісі келуі мүмкін. Бірақ сіз басқалардың пікірі шын мәнінде маңызды емес екенін, бәрі өзіңізге байланысты екенін естен шығармаңыз. Егер STEM саласында, басқа да кез келген бағытта білім алғыңыз келсе және әзірге оңай болмай жатса, берілмеңіз. Басқа адамдардың теріс пікірлерін тыңдамаңыз. Ауыр еңбек әрқашан өз нәтижесін береді. Кезінде берілмей, оқуды, ізденуді, үйренуді тоқтатпауымының арқасында қазір маған кез келген жобаны, тапсырмаларды орындау қатты қиынға соқпайды.

Университетте оқып жүрген жылдары негізгі пәндерден бөлек ML, мобильді әзірлеу, фронтенд/бэккенд, блокчейн және басқа да бағыттармен таныстым. Сондай-ақ тәжірибе жинау үшін жергілікті және халықаралық компанияларға тағылымдамадан өтуге өтініш бердім. Өз оқырмандарыма да бағдарламалаудың түрлі бағыттарын байқап көруге кеңес берер едім. Өйткені бір бағытты меңгеруде қиындық туындағанымен, басқа бағыты оңай меңгеруіңіз мүмкін. Кейін бұл бағытта жұмыс жасамасаңыз да жинаған тәжірибеңіз пайдаңызға түбі бір асары анық. Мысалы, Meta компаниясында жұмыс істеп, негізінен бэккенд әзірлеуге назар аударған кезімде бірнеше рет фронтенд немесе мобильді әзірлеу саласындағы білімді қажет ететін тапсырмаларды орындауыма тура келді.

Сөз соңында “білмейтініңізді сұраудан қорықпаңыздар” дегім келеді. Университетте ұстаздың көмекшісінің сөздерінен кейін менің бойымда өзіме деген сенімсіздік қалыптасып, профессорлардың көзінше ақымақ болып көрінбеу үшін түсінбеген тақырыптарға байланысты сұрақтарымды аз қоюға тырысатын едім. Осының салдарынан тақырыпты түсіну үшін үш есе көп уақыт жұмсауыма тура келетін. Егер түсінбеген жерлерімді дер кезінде сұрағанымда, қанша уақытты үнемдер едім?! Өзгелердің көз алдында тәжірибесіз маман болып қалудан қорықпаңыз және “бәрін білемін” деген ойдан айырылыңыз. Бұл сіздің уақытыңыз бен энергияңызды үнемдеуге жәрдемдеседі”.

Көріп отырғанымыздай, Дәнекер де еңбек жолында өз даңғылын бірден тапқан жоқ. Талай қиындықтардан өтті. Қиналған сәттеріңізде осындай замандастарыңыздың лебіздері медеу болады деген ойдамын. Ендеше келесі тақырыпты меңгеруге көшейік.

Деректер құрылымы

Тізімдер: Тізімдерге кіріспе және олармен қалай жұмыс істеу қажет?

Бағдарламалау әлеміндегі сіздің қолыңыздағы ең қуатты құралдардың бірі – тізім. Тізімдерді сандар, сөздер немесе басқа тізімдер сияқты әртүрлі деректерді сақтай алатын контейнер теңеуге болады. Олар деректерді сақтау мен ұйымдастырудың сансыз тәсілдерінде қолданылуы мүмкін. Бұл тарауда біз тізімдер әлеміне терең еніп, олардың не екенін, қалай құрастыруға болатынын және оларды қолдана отырып, түрлі операцияларды қалай атқару керектігін үйренеміз.

Тізім деген не?

Python тіліндегі тізім – элементтердің реттелген жинағы және әрбір элемент сандар, сөздер немесе одан да күрделі объектілер сипатындағы кез келген деректер түрі болуы мүмкін. Тізімдер тік жақшаға `[]` алынады және элементтер үтірмен бөлінеді. Мысалы:

Python

```
>>> жемістер = ["алма", "банан", "шие"]
>>> сандар = [1, 2, 3, 4, 5]
>>> аралас = [42, "сәлем", 3.14, True]
```

Тізімдер мектепте ұстаған сөмкелеріміз тәрізді. Сөмкеміздің ішіне кітап, қарындаш секілді мектеп құралдарымен бірге қосымша керек-жарақтарымызды да салушы ек қой. Сол сияқты тізімдер де әртүрлі деректерді сақтай алады, бұл оларды көптеген бағдарламалау тапсырмалары үшін пайдалы етеді.

Тізімді құрастыру

Python тілінде тізім жасау оп-оңай. Сіз `[]` бос тізімнен бастайсыз және оған келесідей тік жақшалар мен үтірлерді пайдалану арқылы элементтерді қосасыз:

Python

```
>>> менің_тізімім = [] # бос тізім
>>> жемістер = ["алма", "банан", "шие"]
>>> сандар = [1, 2, 3, 4, 5]
```

Сондай-ақ тізімді **list()** конструкторымен жасауға болады:

Python

```
>>> жемістер = list(("алма", "банан", "шие")) #
list() функциясын қолдану арқылы тізім құрастыра аламыз
>>> жемістер
['алма', 'банан', 'шие']
```

Тізім элементтеріне қол жеткізу

Тізіміңіз бар, енді оны қалай қолдануға болатынын үйренейік. Тізімдегі әрбір элементке **индекс (тізімдегі орны)** тағайындалады. Бірінші элемент 0-ден басталады, екіншісі 1 деп ары қарай кете береді. Мысалы, суреттен алдында құрастырған тізімдердің бейнелеуін көре аласыз. Тізімнің үстінде тұрған сандар тізім элементтерінің индекстері.

0	1	2	0	1	2	3	4
алма	банан	шие	1	2	3	4	5
жемістер			сандар				

0	1	2	3
42	сәлем	3.14	True
аралас			

Тізімдегі элементке қол жеткізу үшін оның тік жақшаға алынған индексін пайдаланыңыз:

Python

```
>>> жемістер = ["алма", "банан", "шие"]
>>> бірінші_жеміс = жемістер[0] # Бірінші элементті алу
>>> екінші_жеміс = жемістер[1] # Екінші элементті алу
>>> соңғы_жеміс = жемістер[-1] # Соңғы элементті алу
>>> ең_соңғы_жемістің_алдындағы_жеміс = жемістер[-2]
# Ең соңғы элементтің алдындағы элементті алу
>>> print(бірінші_жеміс)

"алма"
>>> print(екінші_жеміс)
"банан"
>>> print(соңғы_жеміс)
"шие"
>>> print(ең_соңғы_жемістің_алдындағы_жеміс)
"банан"
```

Тізімдерді өзгерту

Тізімдердің бір қасиеті – олардың өзгергіштігі, яғни оларды жасағаннан кейін де оңай өзгерте аламыз. Тізімдегі элементтерді қосуға, жоюға немесе өзгертуге болады.

Элементтерді қосу

Элементті тізімнің соңына қосу үшін **append()** әдісін қолдануға болады:

Python

```
>>> жемістер = ["алма", "банан", "шие"]
>>> жемістер.append("апельсин") # "апельсин" сөзін
тізімнің соңына қосамыз
>>> жемістер
["алма", "банан", "шие", "апельсин"]
```

Белгілі бір орынға элемент қосу үшін **insert()** әдісін де пайдалануға болады:

Python

```
>>> жемістер.insert(1, "жүзім") # "жүзім" сөзін
бірінші индекске саламыз
>>> жемістер
["алма", "жүзім", "банан", "шие", "апельсин"]
```

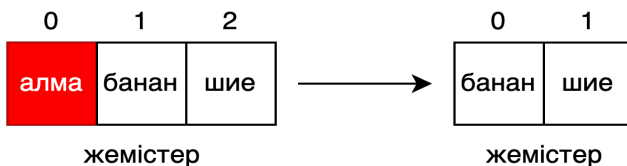
Элементтерді жою

Элементті тізімнен жою үшін **remove()** немесе **pop()** сияқты әдістерді қолдануға болады:

Python

```
>>> жемістер = ["алма", "банан", "шие"]
>>> жемістер.remove("банан") # "банан" сөзін тізімнен
алып тастаймыз

>>> жемістер.pop(0) # 0 индексте тұрған жемісті, яғни
"алма" сөзін алып тастаймыз
"алма"
```

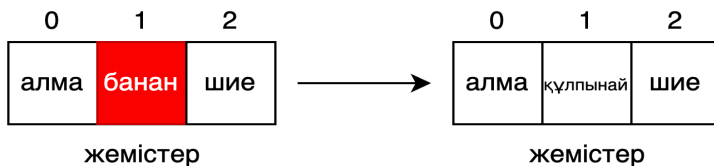


Элементтерді өзгерту

Белгілі бір элементтің мәнін оған жаңа мән тағайындау арқылы өзгертуге болады:

Python

```
>>> жемістер = ["алма", "банан", "шие"]
>>> жемістер[1] = "құлпынай" # Екінші элементін
"құлпынай" жолына ауыстырдық
>>> жемістер[1]
"құлпынай"
```



Тізім арқылы айналым

Циклдар арқылы тізімдерді өңдеуге не басқа да әрекеттер жасауға болады. Мысалы, сізде төрт түліктің тізімі бар делік:

Python

```
>>> төрт_түлік = ["құлыншақ", "бұзау", "қошақан", "лақ"]
>>>
>>> for жануар in төрт_түлік:
...     print("Шақыру", жануар)
...
Шақыру құлыншақ
Шақыру бұзау
Шақыру қошақан
Шақыру лақ
```

Бұл «**for**» циклі тізімдегі әрбір үй жануарын қайталап, әрқайсысы үшін шақыру хабарын басып шығарады.

Тізім операциялары

Тізімдер деректерді өңдеу және олармен тиімді жұмыс істеу үшін әртүрлі операцияларды қолдайды.

Тізім ұзындығы

len() функциясы арқылы тізімде қанша элемент бар екенін білуге болады:

Python

```
>>> жемістер = ["алма", "банан", "шие"]
>>> ұзындығы = len(жемістер) # Тізімнің ұзындығы 3-ке тең
>>> print(ұзындығы)
3
```

Тізімдерді біріктіру

+ операторын пайдалану арқылы екі немесе одан да көп тізімдерді біріктіруге болады:

Python

```
>>> тізім1 = [1, 2, 3]
>>> тізім2 = [4, 5, 6]
>>> біктірілген_тізім = тізім1 + тізім2 #
біктірілген_тізім [1, 2, 3, 4, 5, 6]-ке тең
>>> print(біктірілген_тізім)
[1, 2, 3, 4, 5, 6]
```

Тізімдерді кесу

Кесу – бар тізімнің бір бөлігін шығарып алу арқылы жаңа тізім жасауға мүмкіндік береді:

Python

```
>>> жемістер = ["алма", "банан", "шие", "құрма", "шабдалы"]
>>> кесілген_жемістер = жемістер[1:4] #
кесілген_жемістер ["банан", "шие", "құрма"]-ке тең
>>> print(кесілген_жемістер)
['банан', 'шие', 'құрма']
```

Элементтің бар-жоғын тексеру

Элементтің тізімде бар-жоғын **in** кілт сөзі арқылы тексеруге болады:

Python

```
>>> жемістер = ["алма", "банан", "шие"]
>>> if "банан" in жемістер:
...     print("Иә, 'банан' сөзі тізімнің ішінде.")
```

Тізімді сұрыптау

Тізімді **sort()** әдісі арқылы сұрыптауға болады:

Python

```
>>> сандар = [5, 1, 3, 2, 4]
>>> сандар.sort()
>>> print(сандар)
[1, 2, 3, 4, 5]
```

Тізімді кері айналдыру

reverse() әдісін пайдаланып тізім ретін кері айналдыруға болады:

Python

```
>>> жемістер = ["алма", "банан", "шие"]
>>> жемістер.reverse()
>>> print(жемістер)
["шие", "банан", "алма"]
```

💡 Қорытынды: тізімдер – сіздің достарыңыз

Тізімдер – бағдарламалау әлеміндегі сіздің динамикалық деректер атты достарыңыз. Олар деректерді реттеп және өңдей алады. Бұл оларды бағдарламалаудың маңызды құралына айналдырады.

Бұл тарауда сіз тізімдерді жасауды, олардың элементтерін игеруді, оларды өзгертуді және олармен түрлі әрекеттерді орындауды үйрендіңіз. Тізімдердің жаныңызда болуы сіздің бағдарламалау тапсырмаларының кең ауқымын шешуге жақсы жабдықталғаныңызды байқатады. Олай болса, осы саланы зерттеуді жалғастырыңыз, код жазудан жалықпаңыз және тізімдеріңіз таңғажайып дүниелер жасауыңызға көмектессін!

Сөздіктер: кіріспе және оларды пайдалану жолдары

Python тіліндегі сөздік (dictionary немесе dict) – кілт-мән (key-value) жұптарының жинақтарын сақтауға және алуға мүмкіндік беретін деректер құрылымы. Мұнда әрбір мән бірегей кілтке сәйкес келеді. Сөздіктер – Python-дағы негізгі дерек түрлерінің бірі. Оларды басқа бағдарламалау тілдерінде хэш-кестелер немесе ассоциативті массивтер деп те атайды.

Енді Python тіліндегі сөздіктерге шолу жасайық:

1. Сөздіктің жасалуы:

Сіз `{}` өрнек жақшалар ішіндегі кілт-мән жұптарының үтірмен бөлінген тізімін қосу арқылы сөздікті жасай аласыз. Әрбір кілт өзінің сәйкес мәнінен қос нүкте арқылы `:` бөлінеді.

Python

```
сөздік = {'кілт1': 'мән1', 'кілт2': 'мән2', 'кілт3': 'мән3'}
```

2. Мәндерге қол жеткізу:

Сөздіктегі белгілі бір мәнге сәйкес кілтті тік жақшаның `[]` ішіне қою арқылы қол жеткізуге болады.

Python

```
мән = сөздік['кілт1'] # мән айнымалысында “мән1” жолы бар
```

3. Мәндерді өзгерту:

Белгілі бір кілтке сәйкес мәнді оған жаңа мән беру арқылы өзгертуге болады.

```
Python
сөздік['кілт1'] = 'жаңа мән'
```

4. Жаңадан кілт-мән жұбын енгізу:

Сөздікке жаңа кілт-мән жұбын қосу үшін жаңа кілтке мән беру керек.

```
Python
сөздік['жаңа кілт'] = 'жаңа мән'
```

5. Кілт-мән жұптарын жою:

del операторын пайдаланып, сөздіктен кілт-мән жұбын жоюға болады.

```
Python
del сөздік['кілт1']
```

6. Кілттің бар-жоғын тексеру:

Сөздікте кілттің бар-жоғын **in** операторы арқылы тексеруге болады.

```
Python
if 'кілт1' in сөздік:
    # сөздік['key1']-мен өңдеу
```

7. Сөздік әдістері:

Python сөздіктерінде кілттерді, мәндерді немесе элементтерді алу, сөздіктерді көшіру, оларды тазарту және т.б. әрекеттерді орындаудың бірнеше әдіс-тәсілдері бар.

- **keys():**

Сөздіктегі кілттерді көрсетеді.

```
Python
сөздік = {'кілт1': 'мән1', 'кілт2': 'мән2', 'кілт3': 'мән3'}
кілттер = сөздік.keys() # dict_keys(['кілт1', 'кілт2', 'кілт3'])
кілттер_тізімі = list(keys) # ['кілт1', 'кілт2', 'кілт3']
```

- **values():**

Сөздіктегі мәндерді көрсетеді.

Python

```
мәндер = сөздік.values() # dict_values(['мән1', 'мән2', 'мән3'])
мәндер_тізімі = list(мәндер) # ['мән1', 'мән2', 'мән3']
```

- **items():**

Сөздіктегі кілт-мән жұптарын көрсетеді.

Python

```
кілт_мән_жұптар = сөздік.items() # dict_items([('кілт1', 'мән1'), ('кілт2', 'мән2'), ('кілт3', 'мән3')])
кілт_мән_жұптар_тізімі = list(кілт_мән_жұптар) # [('кілт1', 'мән1'), ('кілт2', 'мән2'), ('кілт3', 'мән3')]
```

- **copy():**

Сөздіктің көшірмесін қайтарады.

Python

```
сөздік_көшірмесі = сөздік.copy() # dict_copy
айнымалысында сөздіктің көшірмесі сақталған
```

- **clear():**

Сөздіктегі барлық кілт-мән жұптарды жояды.

Python

```
сөздік.clear()
```

8. Сөздік бойынша өтіп шығу:

Циклдарды қолдана отырып, кілт-мән сөздігінің кілттері, мәндері немесе жұптары бойынша өтіп шыға аласыз.

Python

```
for кілт in сөздік: # бұл цикл сөздіктің кілттері бойынша өтіп шығады
    print(кілт, сөздік[кілт])
for кілт, мән in сөздік.items(): # бұл цикл сөздіктің кілт-мән жұптары бойынша өтіп шығады
    print(кілт, мән)
```

9. Сөздік жасаудың тағы бір жолы:

Python

```
квадраттар = {x: x**2 for x in range(5)}  
...
```

Бұл мысалда біз 0-ден 4-ке дейінгі сандарды кілт ретінде, ал олардың квадратын мән ретінде сақтап отырмыз.

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}  
...
```

Сөздік – бағдарламалауда ең көп қолданылатын деректер құрылымының бірі. Мысалы, мектепте оқушылардың бағаларын сақтау үшін сөздікті қолдануға болады. Сөздікті пайдалану арқылы оқушының деректерін сақтап, оны басқара аласыз. Мұнда әрбір оқушы өзінің бірегей идентификаторымен (кілтімен) анықталады және сәйкес мән олардың белгілі бір пән бойынша алған бағасы болып табылады.

Python

```
журнал = {  
    'Әлия': 90,  
    'Нұрәділ': 85,  
    'Алмат': 78,  
    'Сая': 92,  
    'Әлішер': 88  
}
```

Бұл мысалда кілттер оқушы аттары, ал мәндер олардың сәйкес бағаларын көрсетеді. Жоғарыда көрсеткен сөздіктің тәсілдерін қолданып, оқушылардың аттары арқылы оқушының бағасын оңай жаңартуға, басқа оқушыларды қосуға немесе жоюға, оқушыларға баға қоюға болады.

Tuple

Кортеж (tuple) – бұл Python тіліндегі деректер құрылымы, бір рет жасалғаннан кейін оның элементтерін өзгертуге болмайды. Кортеж – тұрақты тізім сияқты, бірақ оны өзгерте алмайсыз.

Мысалы, тізімде (list) біз элементтерді қоса, өзгерте немесе алып тастай аламыз, ал кортежде ондай мүмкіндік жоқ. Оның басты

артықшылығы – тұрақтылық пен жылдамдығында, өйткені кортеждер өзгермейтін болғандықтан, оларды өңдеуге де аз уақыт кетеді.

Мысалдар:

Қарапайым кортеж:

Python

```
>>> a = (1, 2, 3) # үш саннан тұратын кортеж
>>> print(a)
(1, 2, 3)
```

Бір элементтен тұратын кортеж жасау үшін элементтен кейін міндетті түрде үтір қою керек:

Python

```
>>> b = (5,) # бір элементті кортеж
>>> print(b)
(5,)
```

Кортеждегі элементтерге қол жеткізу:

Python

```
c = ('алма', 'алмұрт', 'өрік')
print(c[1]) # "алмұрт"-ты шығарады
'алмұрт'
```

Tuple және List деректер типтерің айырмашылықтары

Қорыта айтсақ, кортеж – ішіндегі мәндері өзгермейтін, реттелген мәліметтер жиынтығы.

Тізім (list) мен кортежді (tuple) қолдану жағдайлары олардың ерекшеліктеріне байланысты өзгереді.

Тізім (list) қашан қолдану керек?

Мәліметтер өзгертін кезде, нақтырақ айтқанда мәліметтерді қосу, өзгерту немесе жою қажет болғанда тізімдерді қолданған аса қолайлы.

Мысалы:

Python

```
>>> fruits = ['алма', 'алмұрт', 'өрік']  
>>> fruits.append('шабдалы') # Жаңа элемент қосу  
>>> fruits[0] = 'банан' # Бірінші элементті алмастыру  
>>> print(fruits)  
['банан', 'алмұрт', 'өрік', 'шабдалы']
```

Құрылымы өзгеріп тұратын мәліметтер үшін, яғни элементтер саны өзгеріп тұратын болса (жаңа элементтер қосылып, ескілері алынып тасталса), тізімдер ыңғайлырақ.

Қайталау немесе өңдеуді жүзеге асыру үшін тізімдерді қолданған тиімді. Өйткені тізімдер бірінен соң бірі келетін, реттелген объектілер жиынтығы болғандықтан түрлі мәліметтерді қайта өңдеуге, есеп жүргізуге қолайлы болып келеді.

Кортежді (tuple) қашан қолдану керек?

Тұрақты мәліметтер керек болғанда, нақтырақ айтқанда деректерді өзгертпей, сол күйінше сақтап қалу қажеттілігі туындағанда (мысалы, күндер, координаттар, конфигурациялар) кортежді қолданған дұрыс. Ол мәліметтерді өзгермейтіндей етіп бекітеді.

Мысал:

Python

```
>>> координаталар = (50.4501, 30.5234) #  
координаттар, өзгермейді  
>>> print(координаталар)  
(50.4501, 30.5234)
```

Жылдамдық қажет болғанда, әсіресе үлкен көлемді мәліметтермен жұмыс істегенде кортеждер ыңғайлырақ.

Құрылымды мәліметтерде, яғни айқын құрылымдалған мәліметтерді, нақты кластар немесе тұрақты деректерді (мезгілдер, апта күндері) пайдалану барысында кортежді қолданған қолайлы.

Мәліметтерді қорғауда, белгілі бір деректерді өзгеріссіз сақтау қажет болғанда (қорғану мақсатында), кортежді таңдау пайдалы.

💡 **Қысқаша қорытынды:**

Тізім (list): мәліметтер өзгеріп, жаңартылуы мүмкін кезде, әрі оларды өңдеу қажет болғанда қолданылады;

Кортеж (tuple): мәліметтер тұрақты және өзгеріссіз қалуы керек болса немесе жылдамдық маңызды болғанда қолданылады;

Set

Set (жиын) – Python тілінде қолданылатын деректер құрылымы, онда әрбір элемент бірегей, яғни қайталанатын элементтер болмайды. Алайда set ішіндегі элементтер реттелмегендіктен оларға индекстер арқылы қол жеткізе алмаймыз.

Set-тің негізгі ерекшеліктері:

Элементтердің қайталанбауы, яғни бір элемент жиын ішінде тек бір рет кездеседі. Егер дәл сондай элементті жиынға қоссаңыз, ол есепке алынбайды.

Ретсіздік, яғни жиындағы элементтердің реті болмайды, сондықтан оларды тізімдегі сияқты индекспен шақыру мүмкін емес.

Өзгергіштігі: set-тің элементтерін қосуға және жоюға болады, бірақ set-тің өзі өзгермейтін мәліметтерді (мысалы, кортеждер) ғана қабылдайды.

Set-ті құру:

Set-ті құру үшін жай ғана өрнек жақшаларды (`{}`) қолданамыз немесе `set()` функциясын пайдаланамыз.

Мысалы:

Python

```
>>> менің_жиыным = {1, 2, 3, 4}
>>> print(менің_жиыным)
{1, 2, 3, 4}
```

Егер бірдей элементтерді қосуға ұмтылсаңыз:

Python

```
>>> менің_жиыным = {1, 2, 2, 3, 4}
>>> print(менің_жиыным) # Қайталанып тұрған элемент
есепке алынбайды
{1, 2, 3, 4}
```

Set-пен жасалатын негізгі операциялар:

Элементті қосу: `add()` арқылы бір элемент қосуға болады.

Python

```
>>> менің_жиыным = {1, 2, 3}
>>> менің_жиыным.add(4)
>>> print(менің_жиыным)
{1, 2, 3, 4}
```

Элементті жою: `remove()` немесе `discard()` әдістерімен элементті жоюға болады. Егер `remove()` әдісімен жоқ элементті жойсаңыз, қате шығады, ал `discard()` әдісімен жойсаңыз, қателік болмайды.

Python

```
>>> менің_жиыным = {1, 2, 3}
>>> менің_жиыным.remove(2)
>>> print(менің_жиыным)
{1, 3}
```

Жиындарды біріктіру (union): екі немесе одан көп жиындарды біріктіру үшін `|` операторын немесе `union()` әдісін қолдануға болады.

Python

```
>>> жиын1 = {1, 2, 3}
>>> жиын2 = {3, 4, 5}
>>> нәтиже = жиын1 | жиын2
>>> print(нәтиже)
{1, 2, 3, 4, 5}
```

Элементтердің қиылысуын табу (intersection): жиындардың ортақ элементтерін табу үшін `&` операторын немесе `intersection()` әдісін қолдануға болады.

Python

```
>>> жиын1 = {1, 2, 3}
>>> жиын2 = {3, 4, 5}
>>> нәтиже = жиын1 & жиын2
>>> print(нәтиже)
{3}
```

Айырмашылық (difference): бір жиында бар, бірақ басқа жиында жоқ элементтерді табу үшін - операторын немесе **difference()** әдісін қолдануға болады.

Python

```
>>> жиын1 = {1, 2, 3}
>>> жиын2 = {3, 4, 5}
>>> нәтиже = жиын1 - жиын2
>>> print(нәтиже)
{1, 2}
```

Set-ті қашан қолдану керек?

Қайталанбайтын элементтерді сақтау барысында, яғни адамдардың бірегей идентификаторлары немесе бірегей элементтер керек болғанда қолданған ыңғайлы.

Жылдам тексеру қажет болғанда пайдаланған тиімді, өйткені set ішінен бір элементтің бар-жоғын аз уақытта табуға болады.

Жиындарға тән операцияларда ортақ элементтерді табу, екі жиынды біріктіру сияқты математикалық амалдар қажет болғанда қолданған тиімді.

Қысқаша қорытынды: set – қайталанбайтын, реттелмеген элементтерден тұратын жиын. Оны қайталанатын элементтерді автоматты түрде алып тастау қажет болғанда немесе математикалық жиын операцияларын орындау керек кезде қолдануға болады.

Deque дегеніміз не?

deque (double-ended queue) – бұл Python-ның **collections** модулі арқылы ұсынылатын деректер құрылымы. Бұл "double-ended queue" сөз тіркесінің қысқартылған түрі, ол "екі шеті бар кезек" деп аударылады. Бұл құрылымның ерекшелігі – элементтерді кезектің басына да, соңына да қосуға және алып тастауға болатынында. **deque** құрылымын түсіну үшін кеңірек қарастырамыз.

Deque-нің негізгі сипаттамалары:

1. Элементтерді жылдам кірістіру және жою:

а. Қарапайым тізімде (list) тізімнің басына элемент қосу немесе элементті жою баяу жүзеге асырылады (себебі осыдан кейін барлық элементтерді жылжыту қажет), ал deque-де бұл операциялар өте жылдам орындалады.

ә. Deque құрылымында элементтерді екі жағынан қосу және жою тұрақты уақытта жүзеге асады.

2. Әмбебаптығы:

а. deque құрылымын стек ретінде (элементтерді бір жақтан қосу және жою) немесе кезек ретінде (бір жақтан қосу және басқа жақтан жою) қолдануға болады.

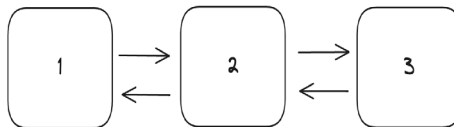
ә. Бұл deque-ді түрлі тапсырмалар үшін өте икемді деректер құрылымы етеді.

deque-ді қолдану мысалдары:

deque-ді құру

Python

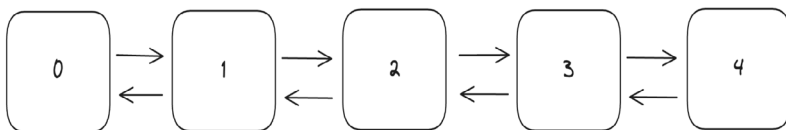
```
>>> from collections import deque
>>> # Бос deque құру
>>> d = deque()
>>> d
deque([])
>>> # Бастапқы элементтермен deque құру
>>> d = deque([1, 2, 3])
>>> d
deque([1, 2, 3])
```



Элементтерді қосу

Python

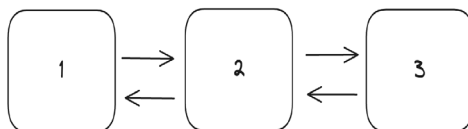
```
>>> d.append(4) # элементті соңына қосу
>>> d.appendleft(0) # элементті басына қосу
>>> d
deque([0, 1, 2, 3, 4])
```



Элементтерді өшіру

Python

```
>>> d.pop()          # элементті соңынан өшіру
4
>>> d.popleft()      # элементті басынан өшіру
0
>>> d
deque([1, 2, 3])
```



Deque-нің негізгі әдістері:

- **append(x)**: элементті x кезектің соңына қосады.
- **appendleft(x)**: элементті x кезектің басына қосады.
- **pop()**: кезектің соңындағы элементті жояды және қайтарады.
- **popleft()**: кезектің басындағы элементті жояды және қайтарады.

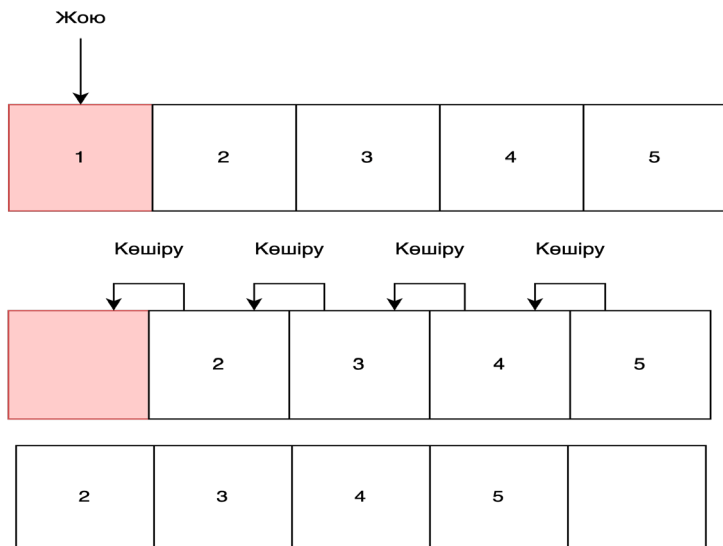
Deque не себепті пайдалы?

Deque тізімнің екі шетінен элементтерді жиі қосу немесе жою қажет болған жағдайларда пайдалы. Мысалы, бұл әртүрлі алгоритмдерді, симуляцияларды жүзеге асыруда немесе деректерді тиімді басқаруда пайдалы болуы мүмкін.

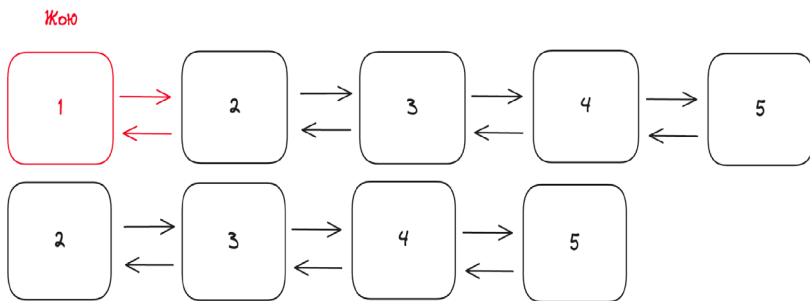
Сіз "deque орнына неге list пайдалана алмаймыз?" деп сұрауыңыз мүмкін. Себебі list-те элементті басына қосу немесе өшіру барлық қалған элементтерді жылжытуды қажет етеді. Бұл әсіресе тізім үлкен болғанда көп уақыт алады. Ал deque-де элементтерді екі жақтан қосу мен өшіру жылдам жұмыс істейді, өйткені deque барлық элементтерді жылжытуды қажет етпейді.

Мысалы, біз `list` жиымынан `pop()` әдісін қолдану арқылы бірінші элементті қалай жоюға болатынын қарастырайық. Бізде `[1, 2, 3, 4, 5]` жиымы бар, және біз жиымның 1-элементін жойғымыз келеді. Енді 0 индексі үшін `pop()` әдісін шақырғанда не болатынын қарастырайық.

`pop(0)` 0-позициясындағы жиым элементін жояды және барлық элементтерді солға жылжытады.



Бұған дейін айтып өткенімдей, бұл әсіресе үлкен жиымдар үшін өте тиімсіз. Ал енді `deque`-де бірінші элементтің қалай жойылатынын қарайық.



Бірінші элементті `deque`-ден жою үшін бізге `list`-тен бір ғана операция қажет болды.

Енді сізде deque не екені және оны қалай қолдану керектігі туралы түсінік бар. Бұл қуатты және икемді деректер құрылымы, ол бағдарламалаудағы көптеген мәселелердің шешімін айтарлықтай жеңілдетуге мүмкіндік береді.

Файлдармен жұмыс

Бұл тарауда біз Python көмегімен файлдарды оқу мен жазудың қызықты әлеміне енеміз. Файлдарды ақпаратты сақтауға болатын сандық жазу кітапшалары деп, ал Python-ды сол жазу кітапшаларына арналған қалам деп елестетіңіз. Ал, бастайық!

Файлдардың маңызы неде?

Файлдар компьютерлік бағдарламаларға арналған қазына жәшіктері сияқты. Олар бізге күнделік, ас мәзірі, кітап сияқты деректерді сақтауға мүмкіндік береді. Файлдармен жұмыс істеуді үйрену өте маңызды, себебі бағдарламалар файлдар арқылы пайдалы ақпаратты сақтау және оқу қабілетін қалыптастырады.

Файлдар қолдану негіздері

Код жазуды бастамас бұрын файлдармен орындауға болатын екі негізгі операцияны түсіндірейін:

1. **Файлдарды оқу:** бұл әрекет кітапты ашып, оқыған сияқты файлды ашуға және оның мазмұнын көруге мүмкіндік береді.
2. **Файлдарды жазу:** бұл әрекет жаңа файл жасауға немесе бар файлды өзгертуге мүмкіндік береді.

Файлдарды ашу және жабу

Python-да файлды оқу немесе жазу үшін оны ашу керек. Ал файлды ашу үшін `open()` функциясын қолданасыз. Мұны іс жүзінде жүзеге асыру үшін төмендегі бағдарламаға қараңыз:

Python

```
>>> файл = open("my_file.txt", "r") # Файлды оқу үшін ашу
>>> файл = open("my_file.txt", "w") # Файлды жазу үшін
ашу (егер ол жоқ болса, жаңа файл ашады)
```

Мұнда «**r**» «оқу» режимін білдіреді, ал «**w**» «жазу» режимін білдіреді. Файлды өңдегеннен кейін оны `файл.close()` арқылы жабуды ұмытпаңыз.

Файлдарды оқу

Алдымен файлды оқудан бастайық. Сізде төмендегідей мазмұны бар `my_file.txt` деп аталатын файл бар делік:

Python

```
Сәлем, жас бағдарламалаушы!  
Python әлеміне қош келдіңіз!
```

Бұл файлдың мазмұнын оқу және басып шығару жолы төмендегідей:

Python

```
>>> файл = open("my_file.txt", "r") # Оқу үшін файлды  
ашыңыз  
>>> файл_мазмұны = файл.read() # Файлды толығымен оқып  
шығыңыз және оны айнымалыда сақтаңыз  
>>> файл.close() # Файлды жабыңыз  
>>> print(файл_мазмұны) # Мазмұнды басып шығарыңыз  
Сәлем, жас бағдарламалаушы!  
Python әлеміне қош келдіңіз!
```

Осы кодты іске қосқанда экраннан `my_file.txt` мазмұнын көре аласыз.

Файлдарды жазу

Енді файлға кейбір деректерді жазып үйренейік. Ол үшін `my_notes.txt` деп аталатын файлды ашып, оған істер тізімін енгізейік:

Python

```
>>> файл = open("my_notes.txt", "w") # Файлды жазу  
үшін ашыңыз (егер ол жоқ болса, жаңа файл жасайды)  
>>> файл.write("Істер тізімі:\n") # Файлға мәтін жазыңыз  
14  
>>> файл.write("1. Python тарауын аяқтау\n")  
25  
>>> файл.write("2. Кодты жазу 30 минут жаттығыңыз\n")  
32  
>>> файл.write("3. Кітап оқу\n")  
13  
>>> файл.close() # Файлды жабыңыз
```

Бұл кодты іске қосқаннан кейін сізде "my_notes.txt" деп аталатын жаңа файл пайда болады, оның ішінде істер тізімін таба аласыз. Осы жерде жазу режимі атауы деректеріңізбен бірдей файлды өшіріп, қайта жазатынын ескеріңіз. **write** файл функциясын қолданған кезде сан қайтарылатынын байқаған шығарсыз. Бұл қанша таңба жазылғанын көрсетеді. Мысалы: **"Істер тізімі:\n"** жолы 14 таңбадан (\n – жаңа жол таңбасы, компьютерде бір таңба болып келеді) құрастырылған.

Файлдарды өзгерту

Егер сіз файлды өшірмей жазғыңыз келсе, **"a"** режимін қолданыңыз. Бұл режимде сіздің деректеріңіз файлдың соңына қосылып жазылады. Түсінікті болу үшін алдында жазылған **"my_notes.txt"** істер тізіміне тағы екі іс жазайық:

Python

```
>>> файл = open("my_notes.txt", "a")
>>> файл.write("4. Үйді жинау\n") # Файлға мәтін
жазыңыз
14
>>> файл.write("5. Ертеңгі сабаққа дайындалу\n")
29
>>> файл.close() # Файлды жабыңыз
```

"for" циклы арқылы файлдарды оқу

"for" циклін пайдаланып, файлдан жолдарды оқу – файл мазмұнын жол бойынша өңдеудің кең тараған әрі тиімді жолы. Алдыңғы тараудағы `my_file.txt` мысалын жалғастыру арқылы `for` циклінің көмегімен жолдарды қалай оқу керектігін анықтаймыз:

"my_file.txt" файлында келесі жолдар бар дейік:

Python

```
Сәлем, жас бағдарламалаушы!
Python әлеміне қош келдіңіз!
```

Файлдағы әрбір жолды келесідей оқуға және басып шығаруға болады:

Python

```
>>> файл = open("my_file.txt", "r") # Оқу үшін  
файлды ашыңыз  
>>> for жол in файл: # Әрбір жолды оқу және басып  
шығару үшін for циклін пайдаланыңыз  
...     print(жол)  
Сәлем, жас бағдарламалаушы!  
  
Python әлеміне қош келдіңіз!  
>>> файл.close() # Файлды жабыңыз
```

Бұл келесі реттілікпен жұмыс істейді:

1. Файлды оқу үшін "r" режимі арқылы ашамыз.
2. Файлдағы әрбір жолды қайталау үшін «**for**» циклін қолданамыз. Әрбір итерациядағы «**жол**» ағымдағы жолдың мәтінін қамтиды.
3. Әрбір жолды **print(жол)** көмегімен басып шығарамыз.
4. Оқып болғаннан кейін файлды жабамыз.

Жолдарды оқу үшін «for» циклін пайдалану үлкен файлдармен жұмыс істегенде ыңғайлы. Себебі ол бүкіл файлды жадқа жүктемей-ақ оларды бір жолда өңдеуге мүмкіндік береді.

Қорытынды

Бұл тарауда сіз Python-да файлдармен жұмыс істеуді үйрендіңіз. Файлдарды ашуға, мазмұнын оқуға және деректер жазуға болатынын білдіңіз. Жоғарыда айтылған ақпараттарды мұқият түсініп алыңыз. Себебі сол арқылы бағдарламаларда ақпаратты сақтауға және алуға көмектесетін негізгі дағды қалыптасады.

Модуль: Python модульдерін қалай пайдалануға болады?

Ақпараттық технология саласының майталманы болуды көздеген талапкер, Python модульдерінің әлеміне қош келдіңіз! Бұл тарауда модульдер Python мүмкіндіктерін кеңейтуге және кодты күштірек етуге қалай көмектесетінін қарастырамыз. Модульдерді бағдарламаларыңызда қолдануға дайын, алдын ала жазылған кодты қамтитын емле кітаптары ретінде қарастырыңыз.

Модуль деген не?

Модульдер – бұл алдын-ала әзірленген құралдармен толтырылған жәшіктер сияқты. Олар – нақты тапсырмаларды орындау үшін қолданылуы мүмкін айнымалы мәндерді, функцияларды және жіктелімдерді қамтитын Python кодының жинақтары. Python әртүрлі мақсаттарға арналған көптеген модульдерді қамтитын кең стандартты кітапханамен келеді, бірақ сіз өз модульдеріңізді де жасай аласыз. Python тіліндегі модуль – **<файлдың_аты>.ру** файлы ретінде сақталған Python скрипті. Онда әртүрлі бағдарламаларда қайта пайдаланғыңыз келетін функциялар, кластар және айнымалылар болуы мүмкін. Осы функциялар, кластар және айнымалыларды жаңа бағдарламалауда модуль енгізу арқылы қолдана аласыз.

Модульдерді енгізу

Модульді кодта пайдалану үшін ең алдымен оны енгізу керек. Python модульдерді енгізудің бірнеше жолын ұсынады:

1. **Толық модульді енгізу:** сіз толық модульді енгізе аласыз және нүкте белгісін пайдалану арқылы функциялары мен кластарына қол жеткізесіз.

Python

```
import math
нәтиже = math.sqrt(25) # math модульінен sqrt
функциясын пайдалану
```

2. **Арнайы элементтерді енгізу:** модульден нақты функцияларды, айнымалы мәндерді немесе кластарды (class) енгізе аласыз.

Python

```
from math import sqrt # Тек sqrt функциясын енгізу
нәтиже = sqrt(25)
```

3. **Қысқарымдар:** кодты қысқа ету үшін модульдерге қысқартылған аттар беруге болады.

Python

```
import math as m
нәтиже = m.sqrt(25) # math модуль деген сөз үшін 'm'
қысқарған атауды пайдалану
```

Жалпы модульдер

Кейбір жалпы модульдерді және олардың не істей алатынын қарастырайық:

1. «math» модулі

«math» модулі әртүрлі математикалық функциялар мен тұрақтыларды қамтамасыз етеді. Бұл арифметикалық амалдар, тригонометрия және т.б. қатысты тапсырмалар үшін өте пайдалы.

Python

```
import math

# Санның квадрат түбірін есептеу
нәтиже = math.sqrt(25)

# Бұрыштың синусын радиан арқылы есептеу
sine_value = math.sin(math.pi / 2)
```

2. «random» модулі

«random» модулі кездейсоқ сандарды жасауға арналған.

Python

```
import random

# 1 мен 100 арасындағы кездейсоқ бүтін санды жасау
random_number = random.randint(1, 100)

# Тізімнен кездейсоқ элементті таңдау
my_list = [1, 2, 3, 4, 5]
random_element = random.choice(my_list)
```

3. «datetime» модулі

«datetime» модулі күндер және уақыттармен жұмыс істеуге арналған негізгі құрал болып табылады. Бұл құрал арқылы тапсырмаларды жоспарлау, оқиғаларды басқару ыңғайлы.

Python

```
import datetime

# Қазіргі күн мен уақытты алу
current_time = datetime.datetime.now()

# Нақты күнді жасау
my_birthday = datetime.datetime(2000, 5, 15)
```

Өзіңіздің модульдеріңізді жасау

Python бағдарламалаушысы ретінде тәжірибеңіз артқан сайын, кодты қайта пайдалануға болатын бөліктерге бөлгіңіз келеді. Өз модуліңізді жасау баға жетпес құндылыққа айналады. Бөлек Python файлдарында функцияларды, кластар мен айнымалы мәндерді анықтап, қажет болған жағдайда оларды бағдарламаңызға импорттай аласыз. Сізбен бірге қарапайым модуль құру жолдарын қарастырайық. Сізде негізгі арифметикалық амалдарды орындайтын бірнеше функциялар бар делік және сіз оларды **my_math** деп аталатын модульде сақтағыңыз келеді.

1-қадам: жаңа Python файлын жасаңыз

Жаңа мәтіндік файл жасаңыз және оны **my_math.py** ретінде сақтаңыз.

2-қадам: Модульдегі функцияларды анықтаңыз

my_math.py ішінде өзіңізге керек келесі функцияларды анықтаңыз:

Python

```
# my_math.py

def қосы(x, y):
    return x + y

def алу(x, y):
    return x - y

def көбейту(x, y):
    return x * y

def бөлу(x, y):
    if y == 0:
        return "Нөлге бөлуге болмайды"
    return x / y
```

3-қадам: Пайдаланушы модулін қолдану

Енді сіз кез келген Python бағдарламасында **my_math** модулін қолдана аласыз. Мәселен **main.py** жаңа файлының ішінде **my_math** модулін пайдалануға болады:

Python

```
# main.py
import my_math

нәтиже = my_math.қосу(5, 3)
print(нәтиже)
```

“main.py” іске қосу “my_math” модулін импорттайды және нәтижені есептеу, басып шығару үшін оның “қосу” функциясын пайдаланады.

Модульдерді ұйымдастыру

Кейде сіз бірнеше модульдерді топтастырғыңыз келеді. Мұндай жағдайда жай ғана модульдерден және `\_\_init\_\_.py` файлынан тұратын тізілім түріндегі пакетті жасай аламыз. “Модульді қалай жасауға болады?” деген сұрақ туындайды:

1. Жаңа папка ашыңыз және оның мазмұнына сәйкес атау беріңіз (мысалы, “my_package”). Ол пакеттің атауы болады.
2. Папканың ішінде бос `\_\_init\_\_.py` файлын жасаңыз. Бұл файл қажеттілігіне қарай конфигурацияларды немесе инициализация кодын қамтуы мүмкін.
3. Пакет ішіндегі жеке модуль файлдарын жасаңыз (мысалы, `module1.py`, `module2.py`). Бұл модульдерге кез келген басқа модуль сияқты нүктелік белгілер арқылы қол жеткізуге болады.
4. Пакет атауын пайдаланып, Python кодында папканың ішіндегі модульдерді импорттай аласыз.

Қорытынды

Қорыта айтсақ, модульдер Python мүмкіндіктерін кеңейту үшін импорттауға болатын код жиынтығы сияқты. Модульдер математикамен, кездейсоқтықпен, күндермен немесе өз кодыңызбен жұмыс жасау барысында бағдарламалаудағы саяхатыңызды қызықты әрі тиімді етеді. Ал келесі тарауда өз модуліңізді қалай құру керек екенін және оқуға, қайта пайдалануға ыңғайлы болу үшін кодты қалай ұйымдастыруға болатынын қарастырамыз. Оқыған ақпараттарды іс жүзінде орындап, жаттыға беріңіз. Кәсіби маман болу үшін қателесуден қорықпау керектігін және нәтижеге жетпейінше қайталай беру қажеттігін ұмытпаңыз.

Кластар: объектіге бағытталған бағдарламалауға кіріспе және кластарды пайдалану

Бұл тарауда Python тіліндегі негізгі ұғым – кластармен танысасыз.

Класс дегеніміз не?

Класс – бұл объектілерді құруға арналған сызба немесе үлгі. Ал объект дегеніміз – компьютердің RAM жадында сақталатын контейнер. Осы контейнер ішінде деректермен орындалатын әрекеттер (функциялар) сақталады. Мәселен, объект ретінде автокөлікті елестетіп көріңіз. Автокөліктің түсі, маркасы және жылдамдығы сияқты әртүрлі қасиеттері бар және ол жылдам жүру, тоқтау және сигнал беру сияқты әрекеттерді орындай алады. Бұл үлгідегі автокөлік – объект, ал оның қасиеттері мен әрекеттері объект ішіндегі деректер мен функциялар сияқты сипатталады.

Бағдарламалауда кластан объекті туындайды. Класс объектінің қандай қасиеттері мен әрекеттері барын анықтайтын сызба сияқты. Демек, объектіні осы класс негізінде жасап, нақты қолдануға болады.

Кластар не үшін қажет?

Бағдарламалауда кластар мынадай маңызды рөл атқарады:

1. Ұйымдастыру: бағдарламалар күрделене түскен сайын, барлық айнымалылар мен функцияларды басқару қиынға соғады. Кластар бізге деректер мен функцияларды топтастыруға көмектеседі. Бұл кодты ұйымдастырады әрі түсінуді жеңілдетеді.

2. Қайта пайдалану мүмкіндігі: класты анықтағаннан кейін одан бірнеше объектілер жасауға болады. Бұл объектіге бағытталған бағдарламалаудың (ОББ) негізгі принципі болып табылады және қайталанатын кодты болдырмауға мүмкіндік береді.

3. Абстракция: кластар жоғары деңгейлі ұғымдардың күрделі бөлшектерін абстракциялайды. Мысалы, «Автокөлік» класын жасап, оның қозғалтқышы мен дөңгелектерінің ішкі жұмысын жасыра аламыз. Бұл біздің бағдарламамызда автомобильдермен жұмыс істеуді жеңілдетеді.

Кластарды құру және пайдалану

Бұл бөлімде класты анықтау негіздерін зерттейміз.

Класс құрылымы

Python тіліндегі класс келесі элементтерден тұрады:

- класс атауы: мағыналы болуы керек және Python атау конвенцияларына сәйкес келуі тиіс.
- атрибуттар: объектінің сипаттамалары немесе қасиеттері.
- класс әдістері: кластың әрекет тәртібін анықтайтын функциялар.

Қарапайым класс құру

“Адам” деп аталатын, сондай-ақ “есім” мен “жас” параметрлері және “танысу” әдісі бар класс құрастырайық. Әдіс дегеніміз – объектімен немесе класпен байланысты функция. Әдіс және функция терминдері бірінің орнына бірі жиі қолданылады.

Python

```
>>> class Адам:
...     def __init__(self, есім, жас):
...         self.есім = есім
...         self.жас = жас
...         self.тіршілік_түр = "адам"
...     def танысу(self):
...         return f"Сәлем, менің есімім {self.есім}, {self.жас} жастамын және мен {self.тіршілік_түр}мын."
```

Кластың барлық әдістерінің бірінші параметрі **self** айнымалысы болатынын байқай аламыз. Бұл параметр кластың объектісін сақтайды және барлық объектілер үшін автоматты түрде жіберіледі. (Объект туралы келесі тарауда нақтырақ талқылаймыз). Ал объектіде атрибуттарды жасау үшін **self.<параметрдің_аты> = <мәні>** форматын қолдану керек. Мысалы, Адам класының **__init__(self, есім, жас)** әдісінде **есім** және **жас** параметрлері объектінің параметрлеріне көшіріледі. Дегенмен **__init__()** функциясында **тіршілік_түр** параметрі тұрақты "адам" мәніне тең болады.

__init__() әдісі – кластың арнайы функцияларының бірі. Python тілінде көптеген арнайы функциялар бар, солардың қатарына **__init__()** әдісі кіреді. **__init__()** әдісі «инстанция» кезінде шақыралады («инстанция» туралы келесі тарауда талқылаймыз). Адам класын мысал ретінде алайық, **__init__()**

әдісі **Адам(«Әлия», 15)** инстанциясы кезінде шақырылады. Бұл мысалда «Әлия» мәні **есім** параметрі, ал **15** болса **жас** параметрі. Жоғарыда айтып өткеніміздей, **self** параметрі барлық объектілердің функцияларына автоматты түрде жіберіледі. Сол себепті **self** параметрін инстанцияда белгілемейміз.

Кластан объектілерді жасау

Класты пайдалану үшін одан объектілер жасауымыз керек. Бұл процесс «инстанция» деп аталады. Басқаша айтқанда, инстанция – класс объектіні компьютердің жады жазу процесі.

Компьютердің жады мен қоймасы жеке файлдарды сақтау үшін ғана емес, компьютерді қосу барысында күнделікті орындалатын операциялар үшін де қолданылатынын естеріңізге сала кеткім келеді. Мәселен, сіз бір веб-сайтқа кірсеңіз, ондағы веб-парақшаларды, суреттерді, бейнелерді компьютер жадында сақтап қалады. Сәйкесінше, бұл сізге сол ақпараттарды тез көруге мүмкіндік береді. Компьютердің ақпаратты жадында жедел сақтап қалуы ағылшын тілінде Random Access Memory (RAM) деп аталады. Жедел жады құрылғысы уақытша жұмыс кеңістігі сияқты екенін ұмытпаңыз. Ол пайдаланушыға ыңғайлы болу үшін жиі қолданатын ақпаратты сақтап отырады. Сіз құрылғыны өшіргеннен кейін жедел жадыға жазылған ақпараттар да өшеді. Егер ол ақпараттарды жойып алғыңыз келмесе, деректерді диск қоймасына сақтаңыз.

Компьютердің қоймасына келер болсақ, онда ұзақ мерзімге сақталуы тиіс заттар жинақталады. Мәселен, сіздің отбасылық фотосуреттеріңіз, бейнежазбалар, фильмдер немесе әндер болуы мүмкін. Ағылшынша Hard-Disk-Drive (HDD), яғни қатқыл диск – компьютердің қоймасы.

Кластар тақырыбына қайта оралайық. Жоғарыда «Адам» класын жасау бойынша код мысал келтірдік. Енді бірнеше «Адам» объектілерін жасайық:

Python

```
>>> адам1 = Адам('Алия', 15)
>>> адам2 = Адам('Арман', 16)
```

адам1 және адам2 объектілер инстациясынан кейін компьютердің жадында сақталады. Мұны төменде келтірілген код арқылы көре аламыз:

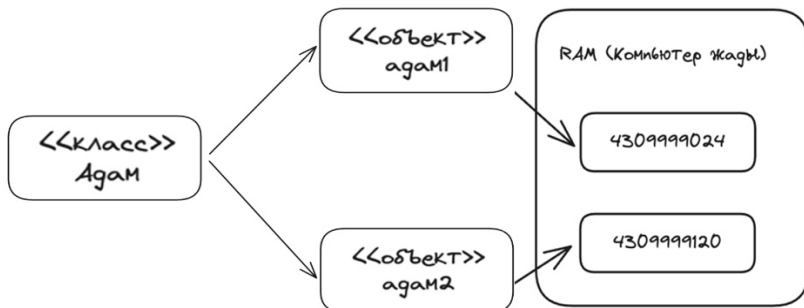
Python

```
>>> print(id(адам1))
4309999024
>>> print(id(адам2))
4309999120
```

`print()` функциясына объектіні `id` функциясының аргументі ретінде қойсаңыз, сол объектінің жадыдағы қай орында тұрғанын көрсетеді. Мысалы, адам1 және адам2 жадының 4309999024 және 4309999120 орындарында орналасқан. Бұл кодтың компьютер түсінетін төмен деңгейінде есептеледі. Сол себепті де сізге бұл сандардың қайдан шыққаны, қалай саналғаны туралы білудің аса қажеті жоқ. адам1 және адам2 екі түрлі инстацияда құрылғандықтан, олар жадының екі түрлі жерінде сақталған әрі екеуі бір-біріне әсер ете алмайды. Екі объект бір екенін тексеру үшін сіз тең операторын қолдана аласыз:

Python

```
>>> print(адам1 == адам2)
False
```



Ал енді анықталған кластың атрибуттары мен әдістеріне қалай қол жеткізуге болатынын қарастырайық. Біз бұл әрекеттерді нүкте белгісін пайдалана отырып іске асыра аламыз. Мысалы, адам1 объектісінің «есім» атрибутына қол жеткізу үшін адам1.есім деп жазамыз:

Python

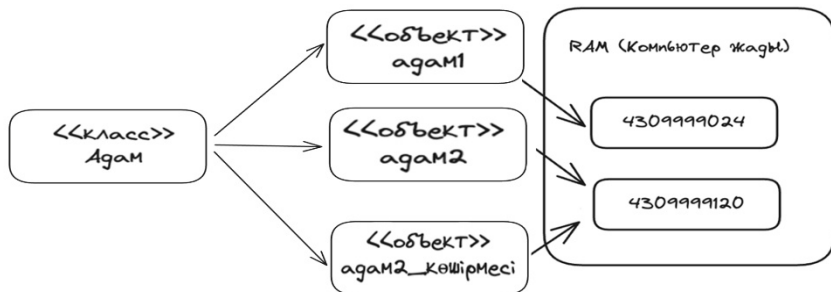
```
>>> адам1.есім
Алия
>>> адам2.есім
Арман
>>> адам2.жас
16
>>> print(адам2.танысу())
Сәлем, менің есімім Арман, мен 16 жастамын және мен адаммын.
>>> print(адам1.танысу())
Сәлем, менің есімім Алия, мен 15 жастамын және мен адаммын.
```

Енді 16 жасар Арман атты адамды адам2 айнымалыдан инстанциясыз көшіру арқылы көрейік:

Python

```
>>> адам2_көшірмесі = адам2
>>> print(id(адам2_көшірмесі))
4309999120
>>> print(id(адам2))
4309999120
>>> print(адам2_көшірмесі == адам2)
True
```

Осы кодтан қандай қызықты өзгерісті байқадыңыз? Анықтап қарасаңыз адам2_көшірмесі және адам2 жадының бір бөлігінде орналасқанын көресіз. Бұл нені білдіреді? Егер біз адам2 айнымалысымен жұмыс жасап, объектіде сақталаған параметрін өзгертсек, адам2_көшірмесі айнымалы объектісінің де параметрі өзгереді.



Мысалы, адам2 объектісінің жасын жиырмаға ауыстырайық.

Python

```
>>> адам2.жас = 20
>>> print(адам2.танысу())
Сәлем, менің есімім Арман, мен 20 жастамын және мен
адаммын.
>>> print(адам2_көшірмесі.танысу())
Сәлем, менің есімім Арман, мен 20 жастамын және мен
адаммын.
```

Жоғарыда айтқанымыздай, адам2_көшірмесі мен объектінің жасы да жиырмаға ауысты. Сол себепті де екі айнымалының бір объекіге нұсқауы өте қауіпті болмақ. Егер сіз жаңа айнымалыны жаңа объект етіп көшіргіңіз келсе, сору модуліндегі deepcopy функциясын қолданыңыз.

Python

```
>>> import copy
>>> адам2_көшірмесі = copy.deepcopy(адам2)
>>> print(id(адам2_көшірмесі))
4309996528
>>> print(id(адам2))
4309999120
```

Енді адам2 және адам2_көшірмесі айнымалылары екі түрлі объектілерге нұсқайды. Осыдан кейін ғана адам2 объектісінің жасын қанша өзгертсек те адам2_көшірмесі айнымалысының жасы өзгермейтін болады.

Python

```
>>> адам2.жас = 25
>>> print(адам2.танысу())
Сәлем, менің есімім Арман, мен 25 жастамын және мен
адаммын.
>>> print(адам2_көшірмесі.танысу())
Сәлем, менің есімім Арман, мен 20 жастамын және мен
адаммын.
```

Егер біз адам2 және адам2_көшірмесі айнымалыларын екі түрлі инстанция арқылы жасасақ, олар жадының екі түрлі жерінде сақталады.

Python

```
>>> адам2 = Адам("Арман", 16)
>>> адам2_көшірмесі = Адам("Арман", 16)
>>> print(id(адам2))
4308601296
>>> print(адам2_көшірмесі)
4309996864
```

Сіздерге беретін теориялық білімімізді тағы да естеліктермен ұштастырсам деймін. Бұл жолы сіздерге **Айя Дарбаеваны** мысалға алғым келеді. Айяның өзі:

“Мектепті бітіргеннен кейін ата-анамның кеңесімен Назарбаев Университетінің химия-биология бағытына түстім. Мақсатым дәрігер болу еді. Алайда күн өткен сайын биология мен химиядан гөрі физика мен математика көбірек қызықтыратынын түсіне бастадым. Сөйтіп таңдаған саламды өзгертуге бел будым. Алайда ата-анам таңдауымды қолдамады. Ата-анамның маған тек жақсылық тілейтінін түсінсем де, жақындарымнан бірден қолдау таппағаным біраз қиындық тудырды.

Университетке түскен алғашқы екі жыл ішінде бірнеше мәрте мамандық ауыстырдым. Биологиядан математиканы, кейіннен компьютерлік ғылымдарды таңдадым. Әсіресе бірінші курс өте қиын болды. Емтиханнан өте алмаймын, жаңа пәндерді меңгере алмаймын деп уайымдадым. Дегенмен уақыт өте келе бағаларым жақсара бастады, ең бастысы – маған бағдарламалау ұнайтынын түсіндім. Үшінші курста жақсылап дайындалып, Meta компаниясында тағылымдамадан өтуге мүмкіндік алдым. Бұл жетістікке қол жеткізуіме қоршаған ортам мол әсерін тигізді. Бірге оқитын құрбы-құрдастарым оқу барысында, сондай-ақ сұхбаттарға дайындық кезінде көмектесетін. Кейде адам өзінен гөрі көп жетістікке қол жеткізген адамды көргенде өзінен-өзі қуыстанып, іштей мүжіле бастайды. Сондай сезімге бой алдырмау үшін өзіңнен үздіктермен дос болып, олардан үйренумен бірге өзіңді бағалай білу де маңызды. Осы ретте менің сізге “өзіңізді мықты адамдармен қоршаңыз, себебі олар сізді қолдап, жаңа жетістіктерге жетуге шабыттандырып отырады” деп кеңес берер едім.

Оқуымды және тағылымдаманы аяқтағаннан кейін, Meta компаниясынан жұмысқа шақырту алып, Лондонға көшіп бардым. Алайда мұнда да бәрі ойдағыдай бола қойған жоқ. Жұмысты бастамай жатып, IT индустриясындағы жаһандық өзгерістермен

тұспа-тұс келгендіктен, мен де қысқартуға ұшырадым. Тез арада жаңа бағытқа бейімделіп, барлық мүмкіндіктерді қарастыруыма тура келді. Отбасымның қолдауы мен дайындықтың арқасында Bloomberg компаниясына жұмысқа орналастым.

Осы жерде сіздерге екінші кеңесімді бергім келеді. Кез келген уақытта күтпеген қиындықтарға дайын болыңыз. Өйткені қай жолда не күтіп тұрғанын ешқашан болжай алмайсыз. Сондықтан қандай жағдай болмасын, өзіңіздің кәсіби капиталыңызды, сондай-ақ байланыстарыңызды нығайта беріңіз. Дәл осы нәрсе қиын сәтте сізге айтарлықтай көмек көрсетуі мүмкін. Менің жүріп өткен жолым – бүгінгі керемет көрініске жетер алдында күмән мен күдікке толы жолдың болатындығына тағы бір дәлел. Бірақ табандылық пен жақын адамдарыңның қолдауы арқылы өз мақсаттарыңа қол жеткізуге болады. Бағытыңды өзгертуден және өзіңе лайықты істі іздеуден қорықпаңыз. STEM – бұл қызықты әрі болашағы зор бағыт. Ол білімге ұмтылып, еңбек етуге дайын әрбір адамға әрқашан да ашық” - деп сыр шертеді.

Ендеше Айяның кеңесіне құлақ түріп, компьютер әлеміне бірге сапарға шығайық.

Мұрагерлік

Бағдарламалаудағы мұрагерлік – бұл жаңа класс (оны "мұрагер" класы деп атайық) кодты басқа кластан ("ата-ана" класы деп атаймыз) пайдалана алатын процесс. Бұл концепт ата-анасынан қасиеттерді мұра ететін бала сияқты. Балалар класының өзіндік қосымша мүмкіндіктері немесе өзгерістері болуы мүмкін, бірақ онда ата-ана класының барлық мүмкіндіктері кездеседі. Мысалға жұмысшы класын құру үшін Адам класын ата-ана класы ретінде қолданайық.

Python

```
>>> class Жұмысшы(Адам):
...     def __init__(self, есім, жас, мамандық,
кәсіп_орны):
...         self.мамандық = мамандық
...         self.кәсіп_орны = кәсіп_орны
...         super().__init__(есім, жас)
...     def танысу(self):
...         жай_танысу = super().танысу()
...         return жай_танысу + f" Қазіргі уақытта мен
{self.мамандық} болып {self.кәсіп_орны}-да(-де, -та, -те)
жұмыс істеймін."
```

Басқа кластан мұраға қалған жаңа класты құру үшін сіз класс анықтамасындағы жақша ішіне мұрагер кластың атын жазуыңыз керек. Жұмысшы мысалында Адам класы жақша ішіне жазылған. Жұмысшы – мұрагер класының ішінде біз **super()** деген функция арқылы Адам – ата-ана класының барлық атрибуттары мен әдістерін қолдана аламыз.

Бұл функция суперкласс әдістерін суперкласты, яғни ата-ана класын тікелей атымен атаудан гөрі икемді етіп шақыруға мүмкіндік береді.

Python

```
>>> айбек = Жұмысшы('Айбек', 25, 'бағдарламалаушы',  
'Google')  
>>> айсұлтан = Жұмысшы('Айсұлтан', 30, 'шопыр',  
'Әкімшілік')  
>>>  
>>> print(айбек.танысу())  
Сәлем, менің есімім Айбек, 25 жастамын және мен  
адаммын. Қазіргі уақытта мен бағдарламалаушы болып  
Google-да(-де, -та, -те) жұмыс істеймін.  
>>> print(айсұлтан.танысу())  
Сәлем, менің есімім Айсұлтан, 30 жастамын және мен  
адаммын. Қазіргі уақытта мен шопыр болып Әкімшілік-да(-  
де, -та, -те) жұмыс істеймін.
```

Полиморфизм

Полиморфизм – бұл объектіге бағытталған бағдарламалаудағы қуатты ұғым. Ол әртүрлі кластағы объектілерді жалпы суперкласс объектілері ретінде қарастыруға мүмкіндік береді. Яғни, әртүрлі типтегі объектілерде жұмыс істеу үшін бір интерфейсті (әдіс немесе функция) қолдануға болады. Полиморфизм икемділікке, кеңейтуге және кодты қайта пайдалануға мүмкіндік береді. Бұл оны модульдік және техникалық қызмет көрсетуге болатын бағдарламалық жүйелерді жобалаудың негізгі принципіне айналдырады. Полиморфизм ұғымын жақсылап түсіну үшін Ит, Мысық және Құс кластарын құрастырып көрейік.

Python

```
>>> class Ит:
...     def __init__(self, аты):
...         self.дауыс = "аф-аф!"
...     def дауыс_шығару(self):
...         print(f"{self.аты} {self.дауыс} деген
дыбыс шығарады")
...     def ояну(self):
...         print(f"{self.аты} оянды.")
...     def ұйықтау(self):
...         print(f"{self.аты} ұйықтап жатыр.")
...
>>> class Мысық:
...     def __init__(self, аты):
...         self.дауыс = "миау-миау!"
...     def дауыс_шығару(self):
...         print(f"{self.аты} {self.дауыс} деген
дыбыс шығарады")
...     def ояну(self):
...         print(f"{self.аты} оянды.")
...     def ұйықтау(self):
...         print(f"{self.аты} ұйықтап жатыр.")
...
>>> class Құс:
...     def __init__(self, аты):
...         self.дауыс = "шық-шық!"
...     def дауыс_шығару(self):
...         print(f"{self.аты} {self.дауыс} деген
дыбыс шығарады")
...     def ояну(self):
...         print(f"{self.аты} оянды.")
...     def ұйықтау(self):
...         print(f"{self.аты} ұйықтап жатыр.")
```

Байқап қарасаңыз, `__init__`, `дауыс_шығару`, `ояну` және `ұйықтау` әдістерін көшіріп, қайта жазып шыққанымызды көресіз. Бұл кодтың дұрыс жазылмағанына мысал бола алады. Бағдарламалау кезінде бұлай көшіріп жазуға болмайды, себебі код мүмкіндігінше қайта пайдалануға жарамды болуы керек. Осы көшіріп жазылған бағдарламамыз мәселені шешу үшін біз білетін класс мұрагерлігі принципін қолданайық:

Python

```
>>> class Жануар:
...     def __init__(self, аты, дауыс):
...         self.аты = аты
...         self.дауыс = дауыс
...     def дауыс_шығару(self):
...         print(f"{self.аты} {self.дауыс} деген
дыбыс шығарады")
...     def ояну(self):
...         print(f"{self.аты} оянды.")
...     def ұйықтай(self):
...         print(f"{self.аты} ұйықтап жатыр.")
...
>>> class Ит(Жануар):
...     def __init__(self, аты):
...         super().__init__(аты, "аф-аф!")
...
>>> class Мысық(Жануар):
...     def __init__(self, аты):
...         super().__init__(аты, "миау-миау!")
...
>>> class Құс(Жануар):
...     def __init__(self, аты):
...         super().__init__(аты, "шық-шық!")
...
... 
```

Бұдан бағдарламамыз кем дегенде 3 есеге жақсарады. Өйткені Жануар класы арқылы `__init__`, `дауыс_шығару`, `ояну` және `ұйықтай` әдістерін бұрынғыдай үш рет көшіріп, қайта жазуға мәжбүр болған жоқпыз.

Енді Жануар ата-ана класы арқылы полиморфизмді тәжірибеде қолданып көрейік:

Python

```
>>> def жануарды_ояту(жануар):
...     жануар.ояну()
...     жануар.дауыс_шығару()
...

>>> # Ит, Мысық және Құс объектітерін жасау
>>> ит = Ит('Ақтөс')
>>> мысық = Мысық('Марғай')
>>> құс = Құс('Бұлбұл')
```

Python

```
>>> # Жануарлардың әртүрлі түрлерімен жануарды_ояту  
функциясын шақыру  
>>> жануарды_ояту(ит)  
Ақтөс оянды.  
Ақтөс аф-аф! деген дыбыс шығарады  
>>> жануарды_ояту(мысық)  
Марғау оянды.  
Марғау миау-миау! деген дыбыс шығарады  
>>> жануарды_ояту(құс)  
Бұлбұл оянды.  
Бұлбұл шық-шық! деген дыбыс шығарады
```

Ит, Мысық және Құс кластары Жануар класының мұрагерлері болғандықтан оларды **жануарды_ояту** деген функцияда қолдануға болады. Егер осы кластардың біреуінде **ояну** немесе **дауыс_шығару** әдістері болмағанда, бағдарламаның орындалу кезінде қате шығатын еді. Сол себепті полиморфизмді мұқият қолданыңыз.

Қорытынды

Бұл бөлімде кластар, инстанция, мұрагерлік және полиморфизм сияқты негізгі ұғымдарды зерттедік, олар туралы ақпарат білдік. Бұл концепциялардың барлығы – заманауи бағдарламалық жасақтама әзірлеу үшін әрі масштабы, сенімді кодты құру үшін қолданылатын негізгі құралдар.

Ақпараттық технология саласын игеру оңай емес. Бұрын код жазып көрмеген адам үшін бұл әлем тіпті күрделі болып көрінуі мүмкін. “ІТ мамандығын меңгергім келеді, осы салада білікті маман болғым келеді” деген табандылығыңыздан таймаңыз. Жаңа білім алуға деген құштарлық пен жігеріңіз міндетті түрде сізді мақсатыңызға жеткізеді. Бастысы аз-аздан қадам басып, алға жылжи беріңіз, талпынуды тоқтатпаңыз. Кітапты оқи отырып, берілген тапсырмаларды менімен бірге орындауды жалғастырыңыз. Осы кітапта берілген әрбір ақпарат болашақ бағдарламалаушы үшін өте маңызды. Өз класыңызды құруға, класс иерархияларын жобалауға және нақты мәселелерді шешу үшін объект бағдарламалау принциптерін қолдануға тәжірибе барысында дағдылана беріңіз. Жаттығуда тер көп төгілген сайын ІТ маманы ретінде іскерлігіңіз арта түседі.



Қорытынды жоба: Жылан ойыны

Python әлеміне саяхат сіз үшін қызықты әрі пайдалы болып жатыр деген ойдамын. Жоғарыдағы тапсырмаларды орындап, тәжірибеде қолданып көрсеңіз, менің құттықтауымды қабыл алыңыз. Осы уақытқа дейін бағдарламалау концепциялары, дерек түрлері мен функциялар жайлы біраз ақпарат алдыңыз. Енді осы алған білімді қызықты жоба жасау барысында қолданып көріңіз. Сізге жеңіл болу үшін “жылан” класикалық ойынын жасауды үйренеміз. “Жылан” ойынының өзіндік нұсқасын жасай отырып, объектіге бағытталған бағдарламалау (ОББ) әлеміне енеміз.

Объектіге бағытталған бағдарламалау – бағдарламалау парадигмасы. Ол біз құрастырып отырған бағдарламаны нақты әлем субъектілерін бейнелейтін объектілерге ұйымдастыруға көмектеседі. Бұл тәсіл біздің кодты модульді етеді, түсінуді жеңілдетеді және кодты қайта пайдалануға мүмкіндік береді.

Жылан ойыны дегеніміз не?

Бағдарламалауды бастамас бұрын “жылан ойыны” туралы қысқаша айтып өтейін. Бұл – класикалық аркада ойыны. Ойыншы жыланды басқару арқылы алмаларды жеу керек. Басты мақсаты – жыланды өз денесіне және ойын алаңының қабырғаларына соқтырмай, мүмкіндігінше көп алма жеуге бағыттау. Жылан көп алма жеген сайын ұзарып, маневр жасау мейлінше қиындай түседі.

Енді “Жылан” ойынын неліктен жасайтынымызды түсіндірейін.

Жылан ойынын жасау – осы уақытқа дейін бағдарламалау туралы үйренген дағдыңызды бекітіп, әсіресе Объектіге Бағытталған Бағдарламалауды тәжірибе жүзінде іске асыруға жәрдемдеседі. Ол төмендегі тәсілмен жүзеге асырылады:

1. Объектіге Бағытталған Бағдарламалау: біз ойынның әрбір құрамдас бөлігіне (жылан, алма және ойын тақтасы) объект ретінде ұсынылған кластарды пайдалана отырып, ойынды жобалаймыз.

2. Ойын логикасы: жыланның қалай қозғалатыны, алма жегенде қалай өсетіні және соқтығысуын қалай анықтайтыны сияқты ойын ережелерін орындаймыз.

3. Пайдаланушы енгізуін өңдеу: біз жыланның қозғалыс бағытын басқару үшін пайдаланушы батырмасын өңдейміз.

4. Графика және анимация: сәнді графика жасамасақ та, **pygame** кітапханасымен қалай жұмыс істеу керегін көрсетеміз.

Алғышарттары:

Жобаға кіріспес бұрын python тілінің негізгі концепция-ларын, әсіресе айнымалыларды, циклдарды, if шартын, функцияларды, кластарды, әсіресе объектілерді жақсы-лап түсіну аса маңызды. Сол үшін жоғарыда берілген ақпараттарды қайталап, ойыңызды жинақтап алыңыз.

Бұл жобада біз PyGames кітапханасын ойынды салу үшін қолданамыз. Осы кітапхананы қолдану үшін, оны біздің жеке компьютерге орнатуымыз қажет. Оны орнату үшін келесі команданы терминалға енгізіңіз.

Python

```
pip install pygame
```

Қанекей, ұялы телефон нарықта кең таралғаннан бері танымал болған “Жылан” ойынын өз бетіңізше жасауға дайын болсаңыз, бастайық!

Difficulty және Direction кластары

“Жылан” ойынының қиындық деңгейлерін белгілеу үшін **Difficulty** деп аталатын класты анықтайық. Бұл кішігірім класс тек қандай қиындық дәрежелері бар екенін анықтайды.

Python

```
from enum import Enum

class Difficulty(Enum):
    """
    Қиындық параметрлері
    Оңай      -> 10
    Орташа    -> 25
    Қиын      -> 40
    Өте қиын -> 60
    """
    EASY = 10
    MEDIUM = 25
    HARD = 40
    SUPER_HARD = 60
```

Difficulty класы мысалында астында жазылған қиындықтарды анықтайық:

- **"EASY"**: бұл оңай қиындық деңгейіне тең және **10** мәніне ие.
- **"MEDIUM"**: бұл орташа қиындық деңгейіне тең және **25** мәніне ие.
- **"HARD"**: бұл күрделі қиындық деңгейіне тең және **40** мәніне ие.
- **"SUPER_HARD"**: бұл өте күрделі қиындық деңгейіне тең және

60 мәніне ие.

Difficulty секілді кластарды біз ағылшынша Enum кластар деп атаймыз. Enum класы ойын кодындағы қиындық деңгейлерін көрсету мен басқарудың ыңғайлы әрі оқылатын әдісін ұсынады. Сондай-ақ бұл бағдарламада қиындық деңгейлерін көрсету үшін ерікті мәндерді қолданатын болсақ, ол қателіктердің болу мүмкіндігін азайтуға көмектеседі. **Difficulty** класын қолданбағанда 10, 25, 40 және 60 сандары оңай, орташа, күрделі және өте күрделі қиындықтарды білдіретінін есте сақтауымыз керек.

Тура осы логиканы қолданып, жыланның жылжу бағытын көрсететін **Direction** класын құрастырайық:

Python

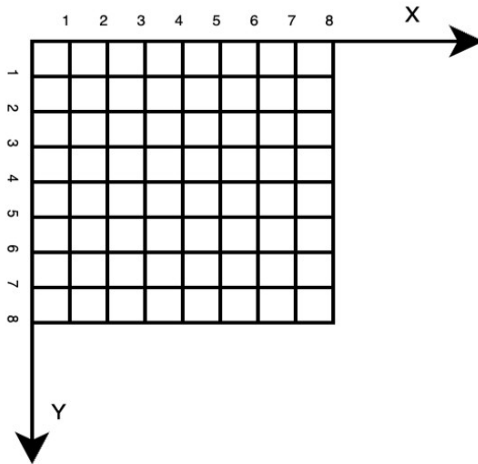
```
from enum import Enum

class Direction(Enum):
    LEFT = "Сол"
    RIGHT = "Оң"
    UP = "Жоғары"
    DOWN = "Төмен"
```

Block класы

Ал енді күрделірек кластар құрастыруға кірісейік.

Ойынды тік бұрышты торкөз координаталар жүйесі ретінде қарастырайық.



Суреттегі координаталар жүйесі тік бұрыштың әдеттегі, қалыпты координаталар (**Oy**) осыне қарама-қарсы. Бұған **pygame** кітапханасының координаталар жүйесі себепкер деп айта аламыз (**pygame**-нің не екеніне және қалай қолданылатынына төменде тоқталамыз). Ал қазір координаталар жүйесіндегі нүктені анықтайтын **Block** класын құрастырамыз:

Python

```
import random

class Block:
    BLOCK_SIZE = 10

    def __init__(self, x: int, y: int) -> None:
        self.x: int = Block.convert_to_block_pos(x)
        self.y: int = Block.convert_to_block_pos(y)

    def __str__(self):
        return f"({self.x}, {self.y})"

    def __eq__(self, value: "Block") -> bool:
        return self.x == value.x and self.y == value.y
```

Python

```
@staticmethod
def convert_to_block_pos(x: int) -> int:
    return x // Block.BLOCK_SIZE * Block.BLOCK_SIZE

@staticmethod
def create_random_block(max_x: int, max_y: int) ->
"Block":
    return Block(
        Block.convert_to_block_pos(random.randrange(1,
max_x)),
        Block.convert_to_block_pos(random.randrange(1,
max_y)),
    )
    def move_up(self) -> None:
        self.y -= self.BLOCK_SIZE
    def move_down(self) -> None:
        self.y += self.BLOCK_SIZE

    def move_left(self) -> None:
        self.x -= self.BLOCK_SIZE

    def move_right(self) -> None:
        self.x += self.BLOCK_SIZE
```

Бұл класта 9 функция анықталған. Оның ішіндегі **convert_to_block_pos** және **create_random_block** – статикалық функциялар. Статикалық функцияларды қолдану үшін бізге класс объектілері керек емес, себебі олар объект атрибуттарын қолданбайды. Енді сол функцияларды біртіндеп өтіп шығайық.

convert_to_block_pos

Block класс мысалында **convert_to_block_pos** функциясы тек жай ордината нүктеден блок позициясына көшіру үшін қолданылады. Бұл функцияны орындау үшін бізге ешқандай объект атрибуттарын қолдану қажет емес, өйткені бұл функцияның ешқандай объектіге қатысы жоқ. Сол себепті де бұл функцияны статикалық деп есептейміз.

create_random_point

Тура осы себеппен **create_random_block** функциясының да статикалық екені анықталады. Бұл функция бізге max_x және max_y

ойын аланның ішінде кездейсоқ блок объектісін қайтарады. Түсінгеніңіздей, бұл функциялар үшін ешқандай объектілерді басшылыққа алудың қажеті жоқ, себебі `max_x` және `max_y` параметрлерін пайдаланып, нөлден жаңа объект жасаймыз. Сол себепті де бұл функцияны статикалық деп есептейміз.

`__init__` конструкторы

Сондай-ақ біз `Block` класына `__init__` және `__eq__` сияқты кірістірілген функцияларды жүзеге асыруды (имплементациялау) қостық. Жоғарыда `__init__` функциясының класс объектілерінің конструкторы екенін айтқан болатынбыз. Бұл функцияда **`convert_to_block_pos`** статикалық функцияларын қолдана отырып, ордината нүктелерін блоктарға аударамыз.

`__eq__` функциясы

`__eq__` (`EQ` – ағылшын тілінен қысқартылған сөз, тең деген мағынаны білдіреді) кірістірілген функциясы екі `Block` класының объектілерін бірдей екеніне тексереді. Объектілер туралы түсіндіргенде біз әртүрлі инстанциялардың көмегімен жасалған бірдей атрибуттары бар объектілердің әрқелкі болатынына мән бердік, себебі олар жадыда әртүрлі объект ретінде сақталады. Дегенмен біз `Block` **`__eq__`** функциясын өзгерту арқылы екі объектінің теңдік шартын өзгерте аламыз. **`__eq__`** функциясын жүзеге асыруда блоктардың позицияларында бірдей `x` пен `y` бар-жоғын тексереміз. Егер олар бірдей болса, онда екі объект бірдей деп айтамыз. Егер позициялар сәйкес келмесе, объектілер әртүрлі деп айтамыз.

Python

```
>>> block_1 = Block(100, 100)
>>> block_2 = Block(100, 100)
>>> print(id(block_1))
4306721792
>>> print(id(block_2))
4305091408
>>> print(block_1 == block_2)
True
>>> print(block_1)
(100, 100)
>>> print(block_2)
(100, 100)
```

Жоғарыдағы мысалға назар аударсаңыз, `block_1` және `block_2` әртүрлі объектер екенін көресіз. Мұны олардың қай жадыда сақталғанынан түсінуге болады: `block_1` **4306721792**-де, ал `block_2` **4305091408**-де сақталады. Бұл екі объект әртүрлі, бірақ `__eq__` `True` мәнін қайтарғандықтан, олар біз үшін бірдей болып саналады. Егер біз `__eq__` функцияларының имплементациясын жазбасақ, `block_1 == block_2` `False` мәнін қайтарар еді.

Window класы

Енді `Window` класын құрайық. `Window` ағылшын тілінен аударғанда терезе деген мағынаны білдіреді. Бұл класс ойын шеңберін анықтауға жауапты болады, мәселен `is_in_bound` `Window` класының әдісі блоктың біздің терезеде ішінде бар жоғын білу үшін қолданылады.

Өзіңіз байқағандай, біз бұл әдісті жылан терезеден шығып кеткенін немесе шықпағанын анықтау үшін қолданатын боламыз. Бұған қоса, терезенің жоғарғы, сол жақ жоғарғы және төменгі бөліктерін анықтау үшін `get_top_location`, `get_top_left_location` және `get_bottom_location` әдістерін жасадық. Қазір бұл әдістерге мән бермеуге болады, бірақ оларды кейінірек, мәтінді рендерлеу үшін пайдаланамыз.

Python

```
from typing import Tuple

# Ойын терезесінің (экранының) параметрлері
class Window:
    def __init__(self, frame_size_x: int, frame_size_y:
int) -> None:
        # Терезе ені
        self.frame_size_x = frame_size_x
        # Терезе биіктігі
        self.frame_size_y = frame_size_y

    # Шектен шығуын анықтайтын әдіс
    def is_in_bound(self, block: Block) -> bool:
        if block.x < 0 or block.x >= self.frame_size_x:
            return False
        if block.y < 0 or block.y >= self.frame_size_y:
            return False
        return True
```

Python

```
def get_top_location(self) -> Tuple[int, int]:  
    return (self.frame_size_x / 2, self.frame_size_y  
    / 4)  
  
    def get_top_left_location(self) -> Tuple[int, int]:  
        return (self.frame_size_x / 10, self.frame_size_y  
        / 10)  
  
    def get_bottom_location(self) -> Tuple[int, int]:  
        return (self.frame_size_x / 2, self.frame_size_y  
        / 1.25)
```

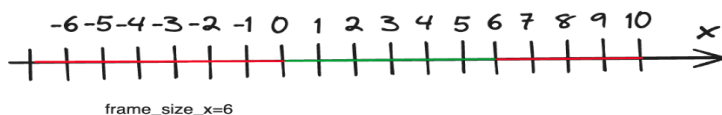
Енді Window класының функцияларын егжей-тегжейлі қарастырайық.

__init__ конструкторы

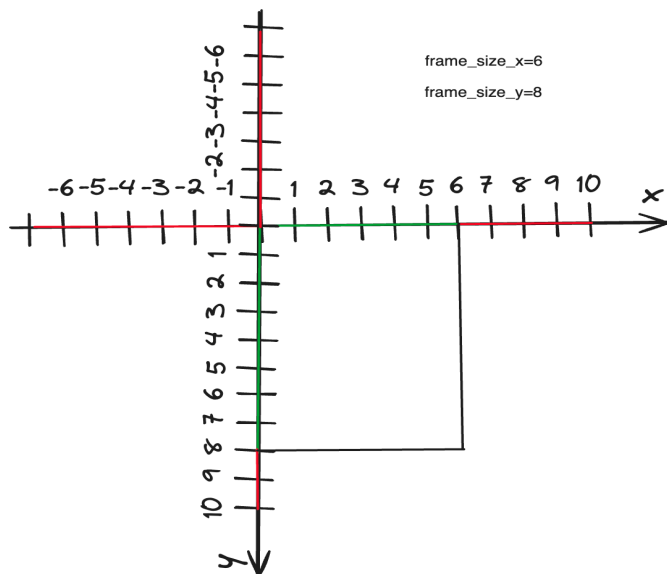
__init__ конструкторы Window класындағы конструкторлардың бірі. Конструктор объектіні Window терезесінің ені *frame_size_x* және терезенің биіктігі *frame_size_y* мәндерімен инициализациялайды.

is_in_bound әдісі

Бұл әдіс блоктың (block) терезе шегінде орналасқанын тексереді. Егер блоктың *x* немесе *y* координаттары нөлден аз болса немесе терезе шектерінен (ені мен биіктігі) асып кетсе, әдіс *False* мәнін қайтарады, кері жағдайда *True* мәнін қайтарады. Төмендегі суреттен терезенің ені 6 бірлік болатын мысалды көре аласыз. Жасыл кесінді 0 мен 6 арасында, блок терезенің ішінде орналасады, ал қызыл сәулелер блоктың терезе шегінен тыс екенін білдіреді. Блоктың жасыл кесіндіде орналасқанын тексеру үшін, біз блоктың орнын жасыл кесіндінің шеттерімен салыстыруымыз керек. Тура осылай блоктың *y* координатының жасыл кесіндісінде орналасқанын тексеруіміз қажет.



Жасыл кесінділердегі x және y нүктелерін тексеру біздің ойын терезесінде блоктың орналасқанына кепіл береді. Сіз мұны келесі графиктен көре аласыз. PyGames құрылымына сәйкес ордината мен абсцисса осының берілу тәртібі математикадағы қалпынан өзгешеленеді. Сол себепті де біздің жағдайда y осы төмен қарай үлкейсе, жоғары қарай азаяды.



get_top_location әдісі

Бұл әдіс терезенің жоғарғы бөлігінде орналасқан нүктенің координаттарын қайтарады. Бұл нүкте көлденеңінен алғанда терезенің ортасында (терезе енінің тең жартысы) және терезе биіктігінің төрттен бірінде орналасады. Біз бұл әдісті ойынды аяқтаған кезде "Ойын аяқталды" тіркесін көрсету үшін рендеринг позицияларын табу мақсатында қолданамыз.

get_top_left_location әдісі

Бұл әдіс терезенің жоғарғы сол жақ бұрышында орналасқан нүктенің координаттарын қайтарады. Біз бұл әдісті ойын барысында ойын ұпай көрсету позицияларын табу үшін қолданамыз.

get_bottom_location әдісі

Бұл әдіс терезенің төменгі бөлігінде орналасқан нүктенің координаттарын қайтарады. Бұл нүкте көлденеңінен (терезе енінің жартысында) және терезе биіктігінің 1/1.25 бөлігінде (яғни, шамамен терезе биіктігінің 80%-ы) орналасады. Біз бұл әдісті ойын соңындағы қорытынды ұпай көрсету позицияларын табу үшін қолданамыз.

Snake класы

Енді Snake (ағылшын тілінде жылан сөзін білдіреді) деген жылан класын іске асыруға көшейік.

__init__ конструкторы

Python

```
from typing import Deque
from collections import deque

class Snake:
    def __init__(self, head_x_pos: int, head_y_pos: int,
snake_length: int = 3) -> None:
        self.snake_body: Deque[Block] = deque()

        for i in range(snake_length):
            self.snake_body.append(Block(head_x_pos - i *
Block.BLOCK_SIZE, head_y_pos))
        # Жылан әрқашан ойынды оң жаққа қарап бастайды
        self.direction = Direction.RIGHT
```

Конструктор Snake объектісін инициализациялайды. Ол жылан басының бастапқы координаттарын head_x_pos және head_y_pos, сондай-ақ жыланның ұзындығын snake_length (әдетте 3) ретінде қабылдайды. Жыланның дене бөліктерін сақтау үшін deque кезегі (self.snake_body) жасалады. Алдымен жыланның басы көрсетілген координаттарда, ал денесі солға қарай жыланның ұзындығына сәйкес орналасады. Ал жыланның бастапқы қозғалыс бағыты (self.direction) оңға (Direction.RIGHT) орнатылады.

get_head_pos әдісі

Python

```
from copy import deepcopy

class Snake:
    def get_head_pos(self) -> Block:
        return deepcopy(self.snake_body[0])
```

Бұл әдіс жыланның басының ағымдағы координаталарын білдіретін **Block** объектісін қайтарады, олар әрдайым **deque** кезегінің бірінші элементінде орналасады. Бірінші элементті қайтармас бұрын **deepcopy()** функциясын қолданғанымызды байқауыңыз мүмкін. Бұл қайтарылатын объектіні өзгерту үшін емес, нақты көшірмесін беру үшін жасалады. Ішкі объектілерді пайдаланушыға беру – дұрыс емес, себебі бұл объектіге кез келген өзгеріс кластың күтпеген әрекетіне әкелуі мүмкін.

change_direction әдісі

Python

```
class Snake:
    ...
    def change_direction(self, new_direction: Direction) -> None:
        # Жылан бірден қарама-қарсы бағытта қозғала алмайды
        if new_direction == Direction.UP and
self.direction != Direction.DOWN:
            self.direction = Direction.UP
        if new_direction == Direction.DOWN and
self.direction != Direction.UP:
            self.direction = Direction.DOWN
        if new_direction == Direction.LEFT and
self.direction != Direction.RIGHT:
            self.direction = Direction.LEFT
        if new_direction == Direction.RIGHT and
self.direction != Direction.LEFT:
            self.direction = Direction.RIGHT
```

Мұндай әдіс жыланның қозғалыс бағытын өзгертеді. Ол жаңа бағытты (**new_direction**) қабылдап, оның ағымдағы бағытқа қарсы емес екенін тексереді, өйткені жылан артқа бұрыла алмайды. Егер жаңа бағытқа рұқсат етілсе, ол **direction** айнымалына бағыт ретінде орнатылады.

Move әдісі

Python

```
class Snake:
    ...
    def move(self, grow = False) -> None:
        # Жыланды жылжыту
        next_head_pos = self.get_head_pos()
        if self.direction == Direction.UP:
            next_head_pos.move_up()
        if self.direction == Direction.DOWN:
            next_head_pos.move_down()
        if self.direction == Direction.LEFT:
            next_head_pos.move_left()
        if self.direction == Direction.RIGHT:
            next_head_pos.move_right()
        if not grow:
            self.snake_body.pop()
        self.snake_body.appendleft(next_head_pos)
```

Бұл әдіс жыланның қозғалысына жауап береді. Алдымен жыланның басының жаңа позициясы ағымдағы позициясына негізделіп, қозғалыс бағыты ескеріле отырып жасалады. Ағымдағы бағытқа байланысты, бас координаталары жаңартылады:

- Жоғары: y координатасы азаяды
- Төмен: y координатасы артады
- Солға: x координатасы азаяды
- Оңға: x координатасы артады

Егер **grow** параметрі **False** болып орнатылса, жыланның денесінің соңғы элементі жойылады, сондықтан жылан ұзара алмайды. Ал егер **grow** параметрі **True** мәніне тен болса, біз жылан ұзындығын арттырамыз. Бұл жылан алма жеген кезде орындалады. Содан кейін жаңа бас позициясы deque кезегінің басына қосылады.

is_touching_itself әдісі

Python

```
class Snake:
    ...
    # Жыланның өз денесіне тиіп кетуіне тексеру
    def is_touching_itself(self) -> bool:
        match_count = 0
        for pos in self.snake_body:
            if self.get_head_pos() == pos:
                match_count += 1
        return match_count != 1
```

Бұл әдіс жыланның өзіне-өзі тиетінін немесе тимейтінін тексереді. Әдіс жылан денесінің барлық блоктары арқылы өтеді және басының басқа блок координаталарымен неше рет сәйкес келетінін есептейді. Егер тек бір рет сәйкес келсе (басы өзімен салыстырылады), әдіс False-ті, ал кері жағдайда True-ді қайтарады.

Game класы

"Жылан" ойынының логикасы мен визуалын басқаратын Game класына кірісейік. Ол ойынның параметрлерін баптауды, объектілерді сызуды, пернетақтадан енгізу барысын өңдеуді және ойынның аяқталу шарттарын тексеруді қамтиды.

Кластың тұрақтылары (константалары)

Python

```
import pygame, sys, time

class Game:
    # Жылан басының координаталары
    SNAKE_X_POS = 100
    SNAKE_Y_POS = 50
    # Ойын терезесінің ені мен биіктігі
    WINDOW_WIDTH = 720
    WINDOW_HEIGHT = 480
    # Жылан ұзындығы
    SNAKE_LENGTH = 3
```

Бұл тұрақтылар жылан басының бастапқы координаттарын, ойын терезесінің өлшемдерін және жыланның бастапқы ұзындығын белгілейді. Game класындағы барлық объектілер үшін осы атрибуттар бірдей мәнге ие болады.

__init__ конструкторы

Python

```
class Game:
    ...
    def __init__(self, difficulty: Difficulty) -> None:
        self.difficulty = difficulty
```

Python

```
        self.snake = Snake(self.SNAKE_X_POS,
self.SNAKE_Y_POS, self.SNAKE_LENGTH)

        # FPS -- 1 секундтағы кадрлар сандарының
бақылаушылары
        self.fps_controller = pygame.time.Clock()

        self.window_frame = Window(Game.WINDOW_WIDTH,
Game.WINDOW_HEIGHT)

        # Түс объектері. Color кластың конструкторына
берілетін 3 сан - қызыл, жасыл, көк (RGB color model)
түстерінің кодтары. Осылайша түстерді анықтай аламыз.
        self.black = pygame.Color(0, 0, 0)
        self.white = pygame.Color(255, 255, 255)
        self.red = pygame.Color(255, 0, 0)
        self.green = pygame.Color(0, 255, 0)
        self.blue = pygame.Color(0, 0, 255)

        self.game_window = pygame.display.set_mode(
            (self.window_frame.frame_size_x,
self.window_frame.frame_size_y)
        )

        # Жылан тамағының координаталары
        self.food_pos = Block.create_random_block(
            self.window_frame.frame_size_x,
            self.window_frame.frame_size_y,
        )

        self.score = 0
        # Тамақтың координаталары өзгерді ме
        self.food_spawn = False
```

Ойынның конструкторында негізгі объектілер инициализацияланады:

- **difficulty** - ойынның күрделілігі.
- **snake** - жылан объектісі.
- **fps_controller** - кадрлар жиілігін басқаратын бақылаушы.
- **window_frame** - ойын терезесінің шекара объектісі.
- ойында қолданылатын түстер.

- `game_window` - ойын бейнеленетін бет.
- `food_pos` - `Block` класындағы `create_random_block` статикалық функцияның көмегімен жасалған тамақтың кездейсоқ орналасқан жері.
- `score` - ойыншының ұпайы.
- `food_spawn` - жаңа тамақ жасау қажет екенін көрсететін белгі.

show_score әдісі

Python

```
class Game:
    ...
    def show_score(self, active: bool, color:
pygame.Color, size: int) -> None:
        score_font = pygame.font.SysFont("times new
roman", size)
        score_surface = score_font.render("Ұпай " +
str(self.score), True, color)
        score_rect = score_surface.get_rect()
        if active == 1:
            score_rect.midtop =
self.window_frame.get_top_left_location()
        else:
            score_rect.midtop =
self.window_frame.get_bottom_location()
        self.game_window.blit(score_surface, score_rect)
```

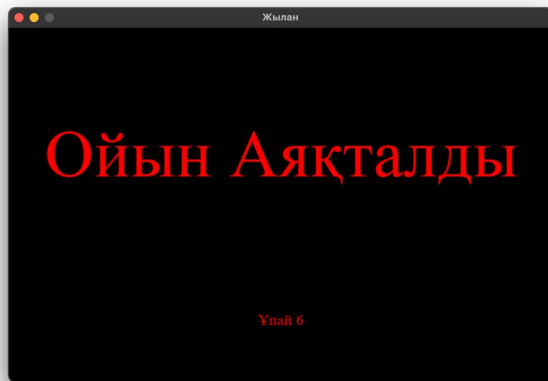
Бұл әдіс экранда ағымдағы есепті көрсетеді. Егер `active` параметрі `True` болса, есеп экранның сол жақ жоғарғы бұрышында көрсетіледі, ал егер `False` болса, экранның төменгі бөлігінде көрсетіледі. `active` параметрі ойынның статусын анықтайды. Егер `True` болса, ойын әлі аяқталмағанын, ал `False` болса, ойын аяқталғанын білдіреді. Ойын барысында есепті сол жақ жоғарғы бұрышта көрсетуіміз керек, ал ойын аяқталғанда есеп экранның төменгі бөлігінде көрсетіледі.

game_over әдісі

Python

```
class Game:
    ...
    # Ойын аяқталды
    def game_over(self) -> None:
        my_font = pygame.font.SysFont("times new roman",
90) # шрифт
        game_over_surface = my_font.render(
            "Ойын аяқталды", True, self.red
        ) # Ойын аяқталды мәтін анықтауы
        # ойын аяқталу мәтіннің локация төртбұрышы
        game_over_rect = game_over_surface.get_rect()
        game_over_rect.midtop =
self.window_frame.get_top_location()
        # түсін анықтау
        self.game_window.fill(self.black)
        self.game_window.blit(game_over_surface,
game_over_rect)
        # ойын есебінің суретін салу
        self.show_score(False, self.red, 20)
        pygame.display.flip()
        # 3 секунд күту
        time.sleep(3)
        # ойын терезесін жабу
        pygame.quit()
        # питон скрипті тоқтатады
        sys.exit()
```

Бұл әдіс “Ойын аяқталды” хабарламасын көрсетіп, ойынды аяқтайды. Мәтінді экранға шығарып, ағымдағы ұпайды көрсетеді, 3 секунд күткен соң бағдарламаны аяқтайды.



paint_snake и paint_apple әдістері

Python

```
class Game:
    ...
    def paint_snake(self) -> None:
        for pos in self.snake.snake_body:
            # Жылан денесі
            # .draw.rect(ойын_беті, түс, ху-координаталары)
            # ху-координаталары -> .Rect(x, y, x_өлшемі,
            y_өлшемі)

            pygame.draw.rect(
                self.game_window,
                self.green,
                pygame.Rect(pos.x, pos.y,
                    Block.BLOCK_SIZE, Block.BLOCK_SIZE),
            )

    def paint_apple(self) -> None:
        # Жылан тамағы
        pygame.draw.rect(
            self.game_window,
            self.white,
            pygame.Rect(
                self.food_pos.x, self.food_pos.y,
                Block.BLOCK_SIZE, Block.BLOCK_SIZE
            ),
        )
```

`paint_snake` әдісі жыланның денесінің барлық блоктарынан өтіп, оларды жасыл төртбұрыштар ретінде сызып, экранда жыланды бейнелеп шығады. Ал `paint_apple` әдісі тамақты экранда ақ төртбұрыш түрінде салады.

handle_key_press әдістері

Python

```
class Game:
    ...
    def handle_key_press(self, event: pygame.event.Event)
-> None:
        # W -> Жоғары; S -> Төмен; A -> Сол; D -> Оң
        if event.key == pygame.K_UP or event.key ==
ord("w"):
            self.snake.change_direction(Direction.UP)
        if event.key == pygame.K_DOWN or event.key ==
ord("s"):
            self.snake.change_direction(Direction.DOWN)
        if event.key == pygame.K_LEFT or event.key ==
ord("a"):
            self.snake.change_direction(Direction.LEFT)
        if event.key == pygame.K_RIGHT or event.key ==
ord("d"):
            self.snake.change_direction(Direction.RIGHT)
        # Esc -> Ойыннан шығу
        if event.key == pygame.K_ESCAPE:
            pygame.event.post(pygame.event.Event(pygame.QUIT))
```

Бұл әдіс жыланның қозғалыс бағытын өзгерту және ойыннан шығу үшін пернелердің басылуын өңдейді.

start әдісі

Python

```
class Game:
    ...
    def start(self) -> None:
        # Herizgi логика
        while True:
            for event in pygame.event.get():
```

Python

```
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        # Перне басылған сайын
        elif event.type == pygame.KEYDOWN:
            self.handle_key_press(event)

    # Жыланның алма жеу ережесі
    if self.snake.get_head_pos() == self.food_pos:
        self.score += 1
        self.food_spawn = True
        self.snake.move(True)
    else:
        self.snake.move(False)

    # Экранда жаңа тамақтың пайда болуы
    if self.food_spawn:
        self.food_pos = Block.create_random_block(
            self.window_frame.frame_size_x,
            self.window_frame.frame_size_y,
        )
        self.food_spawn = False

    self.game_window.fill(self.black)
    self.paint_snake()
    self.paint_apple()

    print(self.snake.get_head_pos())

    # Ойынның аяқталу шарттары
    if not
self.window_frame.is_in_bound(self.snake.get_head_pos()) or
self.snake.is_touching_itself():
        self.game_over()
        self.show_score(True, self.white, 20)
    # Ойын экранын жаңарту
    pygame.display.update()
    # Жаңарту жылдамдығы
    self.fps_controller.tick(self.difficulty.value)
```

Негізгі ойын циклы:

1. Оқиғаларды өңдейді (пернелердің басылуы, терезенің жабылуы).
2. Жыланның тамақты жегенін тексереді, ұпайды арттырады, жыланды өсіреді.
3. Тамақты жесе, орнына жаңа тамақ шығады.

4. Экранды тазартып, жылан мен тамақты қайта салады.
 5. Ойынды аяқтау шарттарын тексереді (терезе шегінен шығу немесе өз денесіне соқтығысу).
 6. Ағымдағы ұпайды көрсетеді.
 7. Экранды жаңартып, кадрлар жиілігін бақылайды.
- Game** класы ойынның барлық компоненттерін біріктіріп, олардың өзара әрекетін басқарады, осылайша толық функционалды "Жылан" ойынын жасайды.

Main функциясы

Python

```
def main() -> None:
    # Ойынды инициализациялау, қателердің бар-жоғын
    тексеру
    check_errors = pygame.init()
    # pygame.init() екі элементтен тұратын кортеж
    қайтарады
    # кортеждегі екінші сан - қателердің саны
    if check_errors[1] > 0:
        print(
            f"[!] Ойынды инициализациялау кезінде
            {check_errors[1]} қате табылды, ендеше ойын басталмайды."
        )
        sys.exit(-1)
    else:
        print("[+] Ойын сәтті инициализацияланды")
    # Ойын терезесіне ат қою
    pygame.display.set_caption("Жылан")

    print("Ойын қиындығын таңдаңыз")
    print("Қиындықты таңдау үшін 1-4 аралығындағы санды
    таңдаңыз")
    print("Оңай      -> 1")
    print("Орташа     -> 2")
    print("Қиын       -> 3")
    print("Өте қиын    -> 4")

    difficulty = input()
    while not difficulty in ["1", "2", "3", "4"]:
        print("1-4 аралығындағы санды таңдаңыз")
        difficulty = input()

    if difficulty == "1":
        difficulty = Difficulty.EASY
```

```

Python
if difficulty == "2":
    difficulty = Difficulty.MEDIUM
if difficulty == "3":
    difficulty = Difficulty.HARD
if difficulty == "4":
    difficulty = Difficulty.SUPER_HARD

game = Game(difficulty)

game.start()

if __name__ == "__main__":
    main()

```

main функциясы “Жылан” ойынының кіру нүктесі болып табылады. Ол ойынды инициализациялайды, қатенің бар-жоғын тексереді, ойынның қиындық деңгейін орнатады және негізгі ойын циклін іске қосады. Енді функцияның әрбір бөлігін егжей-тегжейлі қарастырайық.

Pygame инициализациясы және қателерді тексеру

```

Python
def main() -> None:
    # Ойынды инициализациялау, қателердің бар-жоғын
    тексеру
    check_errors = pygame.init()
    # pygame.init() екі элементтен тұратын кортеж
    қайтарады
    # кортеждегі екінші сан - қателердің саны
    if check_errors[1] > 0:
        print(
            f"[!] Ойынды инициализациялау кезінде
            {check_errors[1]} қате табылды, ендеше ойын басталмайды."
        )
        sys.exit(-1)
    else:
        print("[+] Ойын сәтті инициализацияланды")

```

- **pygame.init()** Pygame-ның барлық модульдерін инициализациялайды және сәтті инициализацияланған модульдердің саны мен қателіктер санын көрсететін кортежді қайтарады.

- Егер кортеждегі екінші сан (`check_errors [1]`) нөлден көп болса, қате туралы хабарлама шығады және бағдарлама тоқтатылады.
- Егер қателіктер болмаса, сәтті инициализациялау туралы хабарлама көрсетіледі.

Ойын терезесінің тақырыбын орнату

Python

```
def main() -> None:
    ...
    # Ойын терезесіне ат қою
    pygame.display.set_caption("Жылан")
```

Ойын терезесінің атауын "Жылан" деп орнатады.

Қиындық деңгейін таңдау

Python

```
def main() -> None:
    ...
    print("Ойын қиындығын таңдаңыз")
    print("Қиындық деңгейін таңдау үшін 1-4 аралығындағы санды таңдаңыз")
    print("Оңай      ->  1")
    print("Орташа   ->  2")
    print("Қиын     ->  3")
    print("Өте қиын  ->  4")

    difficulty = input()
    while not difficulty in ["1", "2", "3", "4"]:
        print("1 мен 4 арасындағы санды таңдаңыз")
        difficulty = input()
```

- Экранға қиындық деңгейін таңдау нұсқауларын шығарады.
- Пайдаланушыдан 1 мен 4 аралығындағы мәнді енгізуді күтеді. Егер дұрыс емес мән енгізілсе, қайта енгізуді сұрайды.

Қиындық деңгейін орнату

Python

```
def main() -> None:
    ...
    if difficulty == "1":
        difficulty = Difficulty.EASY
    if difficulty == "2":
        difficulty = Difficulty.MEDIUM
    if difficulty == "3":
        difficulty = Difficulty.HARD
    if difficulty == "4":
        difficulty = Difficulty.SUPER_HARD
```

Енгізілген мәнді **Difficulty** тізіміндегі сәйкес қиындық деңгейіне түрлендіреді.

Ойынның құрылуы және іске қосылуы

Python

```
def main() -> None:
    ...
    game = Game(difficulty)
    game.start()
```

- Таңдалған қиындық деңгейімен ойынның кезекті объектісін жасайды.
- Жасалған **game** объектісінің **start** әдісін шақыра отырып, негізгі ойын циклін іске қосады.

main функциясын іске қосу

Python

```
if __name__ == "__main__":
    main()
```

- Бұл код файлдың негізгі бағдарлама ретінде іске қосылып жатқанын (модуль ретінде импортталмағанын) тексереді.
- Егер файл негізгі бағдарлама ретінде іске қосылса, **main** функциясы шақырылады.

Жалпы қорытынды

`main` функциясы келесі негізгі тапсырмаларды орындайды:

1. Pygame-ді инициализациялайды және қателердің бар-жоғын тексереді.
2. Ойын терезесінің тақырыбын орнатады.
3. Пайдаланушыдан қиындық деңгейін таңдауды сұрайды.
4. Таңдалған қиындықпен ойын объектісін жасайды.
5. Негізгі ойын циклын бастайды.

Осылайша `main` функциясы ойынды бастапқы күйге келтіріп, негізгі ойын процесін іске қосуға жауап береді.

Ойынды іске қосу

Ойынды бастау үшін терминалға `python snake.py` деп жазыңыз. Терминалда сізден ойынның қиындық деңгейін сұрайды. Ол үшін 1-ден 4-ке дейін кез келген санды енгізіп, Enter пернесін басыңыз қажет. Бұл ойынның жеке терезеде басталуына себеп болады.

Python

```
nuradilkozhakhmet@Nuradils-Laptop ~ % python snake.py
pygame 2.3.0 (SDL 2.24.2, Python 3.10.9)
Hello from the pygame community.
https://www.pygame.org/contribute.html
[+] Ойын сәтті инициализацияланды
Ойын қиындығын таңдаңыз
Қиындықты таңдау үшін 1-4 аралығындағы санды таңдаңыз
Оңай      ->  1
Орташа    ->  2
Қиын      ->  3
Өте қиын  ->  4
```

Кодтың толық нұсқасы

Python

```
import pygame, sys, time, random
from enum import Enum
from typing import Deque, Tuple
from collections import deque
from copy import deepcopy
class Difficulty(Enum):
    """
```

Python

Қиындық параметрлері

Оңай -> 10

Орташа -> 25

Қиын -> 40

Өте қиын -> 60

"""

EASY = 10

MEDIUM = 25

HARD = 40

SUPER_HARD = 60

class Direction(Enum):

LEFT = "Сол"

RIGHT = "Оң"

UP = "Жоғары"

DOWN = "Төмен"

class Block:

BLOCK_SIZE = 10

def __init__(self, x: int, y: int) -> None:

self.x: int = Block.convert_to_block_pos(x)

self.y: int = Block.convert_to_block_pos(y)

def __str__(self):

return f"({self.x}, {self.y})"

def __eq__(self, value: "Block") -> bool:

return self.x == value.x and self.y == value.y

@staticmethod

def convert_to_block_pos(x: int) -> int:

return x // Block.BLOCK_SIZE * Block.BLOCK_SIZE

@staticmethod

def create_random_block(max_x: int, max_y: int) ->

"Block":

return Block(

Block.convert_to_block_pos(random.randrange(1,

max_x)),

```

Python
        Block.convert_to_block_pos(random.randrange(1,
max_y)),
    )

    def move_up(self) -> None:
        self.y -= self.BLOCK_SIZE

    def move_down(self) -> None:
        self.y += self.BLOCK_SIZE

    def move_left(self) -> None:
        self.x -= self.BLOCK_SIZE

    def move_right(self) -> None:
        self.x += self.BLOCK_SIZE

# Ойын терезесінің (экранының) параметрлері
class Window:
    def __init__(self, frame_size_x: int, frame_size_y:
int) -> None:
        # Терезе ені
        self.frame_size_x = frame_size_x
        # Терезе биіктігі
        self.frame_size_y = frame_size_y

# Блоктың белгіленген шектен шығу-шықпауын тексеретін әдіс
def is_in_bound(self, block: Block) -> bool:
    if block.x < 0 or block.x >= self.frame_size_x:
        return False
    if block.y < 0 or block.y >= self.frame_size_y:
        return False
    return True

# Ойын терезесінің үстіңгі координаталарын қайтарады
def get_top_location(self) -> Tuple[int, int]:
    return (self.frame_size_x / 2, self.frame_size_y / 4)

# Ойын терезесінің үстіңгі сол жақ координаталарын
қайтарады
def get_top_left_location(self) -> Tuple[int, int]:

```

Python

```
        return (self.frame_size_x / 10, self.frame_size_y / 10)

# Ойын терезесінің астыңғы координаталарын қайтарады
def get_bottom_location(self) -> Tuple[int, int]:
    return (self.frame_size_x / 2, self.frame_size_y / 1.25)

class Snake:
    def __init__(self, head_x_pos: int, head_y_pos: int,
snake_length: int = 3) -> None:
        self.snake_body: Deque[Block] = deque()

        for i in range(snake_length):
            self.snake_body.append(Block(head_x_pos - i
* Block.BLOCK_SIZE, head_y_pos))

        self.direction = Direction.RIGHT

    def get_head_pos(self) -> Block:
        return deepcopy(self.snake_body[0])

    def change_direction(self, new_directon: Direction) -
> None:
        # Жылан бірден қарама-қарсы бағытта қозғала алмайды
        if new_directon == Direction.UP and
self.direction != Direction.DOWN:
            self.direction = Direction.UP
        if new_directon == Direction.DOWN and
self.direction != Direction.UP:
            self.direction = Direction.DOWN
        if new_directon == Direction.LEFT and
self.direction != Direction.RIGHT:
            self.direction = Direction.LEFT
        if new_directon == Direction.RIGHT and
self.direction != Direction.LEFT:
            self.direction = Direction.RIGHT

    def move(self, grow = False) -> None:
        # Жыланды жылжыту
        next_head_pos = self.get_head_pos()
```

Python

```
if self.direction == Direction.UP:
    next_head_pos.move_up()
if self.direction == Direction.DOWN:
    next_head_pos.move_down()
if self.direction == Direction.LEFT:
    next_head_pos.move_left()
if self.direction == Direction.RIGHT:
    next_head_pos.move_right()
if not grow:
    self.snake_body.pop()
self.snake_body.appendleft(next_head_pos)

# Жыланның өз денесіне тиіп кетуі
def is_touching_itself(self) -> bool:
    match_count = 0
    for pos in self.snake_body:
        if self.get_head_pos() == pos:
            match_count += 1
    return match_count != 1

class Game:
    # Жылан басының координаталары
    SNAKE_X_POS = 100
    SNAKE_Y_POS = 50
    # Ойын терезесінің ені мен биіктігі
    WINDOW_WIDTH = 720
    WINDOW_HEIGHT = 480
    # Жылан ұзындығы
    SNAKE_LENGTH = 3

    def __init__(self, difficulty: Difficulty) -> None:
        self.difficulty = difficulty
        self.snake = Snake(self.SNAKE_X_POS,
self.SNAKE_Y_POS, self.SNAKE_LENGTH)

    # FPS - 1 секундағы кадрлар сандарының бақылаушылары
    self.fps_controller = pygame.time.Clock()
```

Python

```
self.window_frame = Window(Game.WINDOW_WIDTH,
Game.WINDOW_HEIGHT)

# Түс объектері. Color кластың конструкторына
берілетін 3 сан - қызыл, жасыл, көк (RGB color model)
түстерінің кодтары. Осылайша түстерді анықтай аламыз.
self.black = pygame.Color(0, 0, 0)
self.white = pygame.Color(255, 255, 255)
self.red = pygame.Color(255, 0, 0)
self.green = pygame.Color(0, 255, 0)
self.blue = pygame.Color(0, 0, 255)
self.game_window = pygame.display.set_mode(
    (self.window_frame.frame_size_x,
    self.window_frame.frame_size_y)
)

# Жылан тамағының координаталары
self.food_pos = Block.create_random_block(
    self.window_frame.frame_size_x,
    self.window_frame.frame_size_y,
)

self.score = 0
# Тамақтың координаталары өзгергенін анықтайды
self.food_spawn = False

def show_score(self, active: bool, color: pygame.Color,
size: int) -> None:
    score_font = pygame.font.SysFont("times new roman", size)
    score_surface = score_font.render("Ұнай " +
str(self.score), True, color)
    score_rect = score_surface.get_rect()
    if active == 1:
        score_rect.midtop =
self.window_frame.get_top_left_location()
    else:
        score_rect.midtop =
self.window_frame.get_bottom_location()
    self.game_window.blit(score_surface, score_rect)
```

```

    # Ойын аяқталды
    def game_over(self) -> None:
        my_font = pygame.font.SysFont("times new roman", 90)
# шрифт
        game_over_surface = my_font.render(
            "Ойын аяқталды", True, self.red
        ) # Ойын аяқталды текст анықтауы
        # Ойын аяқталды мәтін төртбұрышының орны
        game_over_rect = game_over_surface.get_rect()
        game_over_rect.midtop =
self.window_frame.get_top_location()
        # Ойын терезесін қара түспен бояйды
        self.game_window.fill(self.black)
        self.game_window.blit(game_over_surface,
game_over_rect)
        # ойын ұпайының суретін салу
        self.show_score(False, self.red, 20)
        pygame.display.flip()
        # 3 секунд күту
        time.sleep(3)
        # ойын терезесін жабу
        pygame.quit()
        # питон скриптті тоқтатады
        sys.exit()

    def paint_snake(self) -> None:
        for pos in self.snake.snake_body:
            # Жылан денесі
            # pygame.draw.rect(ойын_беті, түс, (x,y-
координаталары))
            # (x,y-координаталары) -> pygame.Rect(x, y,
x_өлшемі, y_өлшемі)
            pygame.draw.rect(
                self.game_window,
                self.green,
                pygame.Rect(pos.x, pos.y, Block.BLOCK_SIZE,
Block.BLOCK_SIZE),
            )

```

Python

```
def paint_apple(self) -> None:
    # Жылан тамағы
    pygame.draw.rect(
        self.game_window,
        self.white,
        pygame.Rect(
            self.food_pos.x, self.food_pos.y,
            Block.BLOCK_SIZE, Block.BLOCK_SIZE
        ),
    )

def handle_key_press(self, event: pygame.event.Event)
-> None:
    # W -> Жоғары; S -> Төмен; A -> Сол; D -> Оң
    if event.key == pygame.K_UP or event.key ==
ord("w"):
        self.snake.change_direction(Direction.UP)
    if event.key == pygame.K_DOWN or event.key ==
ord("s"):
        self.snake.change_direction(Direction.DOWN)
    if event.key == pygame.K_LEFT or event.key ==
ord("a"):
        self.snake.change_direction(Direction.LEFT)
    if event.key == pygame.K_RIGHT or event.key ==
ord("d"):
        self.snake.change_direction(Direction.RIGHT)
    # Esc -> Ойыннан шығу
    if event.key == pygame.K_ESCAPE:
        pygame.event.post(pygame.event.Event(pygame.QUIT))

def start(self) -> None:
    # Herizri логика
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
        # Перне басылған сайын
```

Python

```
        elif event.type == pygame.KEYDOWN:
            self.handle_key_press(event)
# Жыланның тамақ жеу ережесі
if self.snake.get_head_pos() == self.food_pos:
    self.score += 1
    self.food_spawn = True
    self.snake.move(True)
else:
    self.snake.move(False)

# Экранда жаңа тамақтың пайда болуы
if self.food_spawn:
    self.food_pos = Block.create_random_block(
        self.window_frame.frame_size_x,
        self.window_frame.frame_size_y,
    )
    self.food_spawn = False
self.game_window.fill(self.black)
self.paint_snake()
self.paint_apple()
print(self.snake.get_head_pos())

# Ойынның аяқталу шарттары
if not
self.window_frame.is_in_bound(self.snake.get_head_pos()) or
self.snake.is_touching_itself():
    self.game_over()

    self.show_score(True, self.white, 20)
# Ойын экранын жаңарту
pygame.display.update()
# Жаңарту жылдамдығы
self.fps_controller.tick(self.difficulty.value)

def main() -> None:
    # Ойынды инициализациялау, қателердің бар-жоғын тексеру
    check_errors = pygame.init()
    # pygame.init() мысал шығаруы -> (6, 0)
    # tuple-дағы екінші сан - қателердің саны
```

Python

```
if check_errors[1] > 0:
    print(
        f"[!] Ойынды инициализациялау кезінде
        {check_errors[1]} қате табылды, тоқтату..."
    )
    sys.exit(-1)
else:
    print("[+] Ойын сәтті инициализацияланды")

# Ойын терезесіне ат қою
pygame.display.set_caption("Жылан")

print("Ойын қиындығын таңдаңыз")
print("Қиындық таңдау үшін 1-4 аралығындағы санды таңдаңыз")
print("Оңай      ->  1")
print("Орташа     ->  2")
print("Қиын        ->  3")
print("Өте қиын    ->  4")

difficulty = input()
while not difficulty in ["1", "2", "3", "4"]:
    print("1 мен 4 аралығындағы санды таңдаңыз")
    difficulty = input()
if difficulty == "1":
    difficulty = Difficulty.EASY
if difficulty == "2":
    difficulty = Difficulty.MEDIUM
if difficulty == "3":
    difficulty = Difficulty.HARD
if difficulty == "4":
    difficulty = Difficulty.SUPER_HARD

game = Game(difficulty)

game.start()

if __name__ == "__main__":
    main()
```

Қорытынды сөз

Құрметті оқырман! Міне, кітаптың соңына да келдік. Бағдарламалауды теориялық тұрғыда баяндау қаншалықты күрделі болғанымен, барынша қарапайым тілде түсіндіруге тырыстым. Кітапты оқу арқылы IT саласы туралы мәліметтер білдіңіз, код жазуды үйрендіңіз, алғашқы ойыныңызды жасап көрдіңіз. Бұл қадамдар сізге бағдарламалауды үйренуді жалғастыруға, ары қарай зерттеуге, осы жолда дамуға себепші болады деп сенемін. Жоғары оқу орнын бітірген азаматтар немесе тума таланттар ғана бағдарламалауды жақсы меңгереді деген қате түсініктен арылыңыз. Қызығушылық танытып, үздіксіз ізденген кез келген адам қай жасқа келгеніне қарамастан бағдарламалаушы бола алады. Оқушысыз ба, студентсіз бе әлде басқа салада жұмыс жасайтын орта жастағы азаматсыз ба, ол маңызды емес. Қазір IT саласын зерделеп жүрген жасы 50-ден асқан жандар да бар. Жаңа саланы, білімді немесе кәсіпті меңгеру ешқашан кеш емес екенін ұмытпаңыз.

Кез келген салада бір орында тұрып қалмай, дамып отыру маңызды. Сол сияқты бағдарламалаудың шебері болу, келесі деңгейлерге өту үшін код жазуды жалғастыра беріңіз. Үнемі ізденіңіз. Құлшына зерттеңіз. Үлкен мақсат қойыңыз. Бүкіл әлемді аузына қаратқан Марк Цукерберг, Стив Джобс, Билл Гейтс, Илон Маск, Джефф Безос, Сундар Пичаи, Сьюзан Войчицки, Тим Куктар да кезінде қарапайымнан бастаған.

Осындай IT алпауыттарының командасында жұмыс жасайтын қазақтың қыз-жігіттері де аз емес. Олардың бірқатарымен жоғарыда танысып та шықтық. Өмірде сәтсіздіктер, қиындықтар болады. Үздік маман болу жолында түрлі кедергілер кездесуі мүмкін. Алайда қиындықтар орта жолдан қайтуға, қолды бір сілтеуге себеп болмасын. Адам қиындықта шыңдалу арқылы биіктерге жетеді, армандары орындалады.

Бүгінде IT көптеген саланың мәселесін шешуге тікелей үлесін қосуда. Күнделікті өмірде қолданып жүрген қызмет түрлерінің барлығы веб-сайттар мен мобильді қосымшалардың арқасында жүзеге асуда. Мәселен қарапайым кофені сайт немесе мобильді қосымша арқылы алдыра аласыз. Тіпті алдын ала тапсырыс беріп, кезек күтпей алып кету мүмкіндігі де бар. Байқасаңыз, банктердің көбі онлайн қызметке көшкен. Міне, соның барлығы IT өндірісіндегі жетістіктердің жемісі. Бағдарламалауды меңгеру арқылы сіз де тың идеяларды іске асырып, қоғамға пайдалы жобаларды жасай

аласыз. Бағдарламалау – бұл жай ғана дағды емес. Ол – ақыл-ойдың жемісі, адамдардың мәселесін шешу жолындағы масштабты жобаларды жасаудың тамаша мүмкіндігі.

Бұл кітапты жазудағы мақсатым қазақ жастарына пайдамды келтіру болатын. Кітап сізді ақпараттық технология әлеміне жақындатса, осы салаға қызығушылығыңызды оятса, демек мақсатыма жеткенім. Араларыңыздан әлемдік бәсекеге қабілетті IT мамандары шықпасына кім кепіл?!

Сөз соңында тағы да “ешқашан тоқтамаңыз” деп, сізді жігерлендіргім келеді. Ақпараттық технология талантты, талапты жастарға мұқтаж!

Бағдарламалау биігінде кездесетінімізге сенімдімін!

Болашақ IT маманына бірнеше кеңес:

1. **Ағылшын тілін үйреніңіз.** Ағылшын тілі сізге жаңа дағдыларды меңгеруге, IT-дан басқа салаларда да терең білім алуға мүмкіндік береді.

2. **Танымды кеңейтіңіз.** Тек бағдарламалар жасай бермей салада орын алған басқа да жаңалықтарды біліп жүріңіз. Бұл сізге жаңа идеялардың келуіне, қоғамдағы өзекті мәселелердің шешімін табуыңызға түрткі болуы мүмкін.

3. **Нетворкинг құрыңыз.** IT мамандарының көбі табиғатынан интроверт болады, дегенмен көпшілікпен жиі араласып, пікір алмасуға тырысыңыз. Білімді дос, білікті таныстардың болуы өте маңызды. Жайлы орта сіздің адами және техникалық қасиеттеріңізді дамытып, ілгері ұмтылуыңызға ықпалын тигізеді. Өзіңізбен бір бағыттағы, тілеулес достарды табу мақсатыңызға қол жеткізуге жәрдемдеседі.

4. **Талаптана беріңіз.** Кез келген табысты, іскер адамның артында оның табандылығы, еңбекқорлығы тұрады. “Бұл іс менікі емес”, “менің қолымнан келмейді” деп айту оңай. Алайда тоқтаусыз тама берсе, тамшының да тас жаратынын ұмытпаңыз. Алға ұмтылмайтын, үйренуге талпынбайтын адамның өмірі сәтсіздікке толы болуы қалыпты жағдай. Позитивті ойлау, жаңа істен қорықпау, соңына дейін әрекет жасау жеңімпаз адамға тән қасиеттер екенін ұмытпаңыз.

5. **Өзіңізге сеніңіз.** Егер менен: “адамдардың айырмашылығы неде?” - деп сұраса, “өзін-өзі бағалауында” - дер едім. Ерекше адамдар өз күштеріне және дағдыларына әрдайым сенеді, бірақ бұл тәкаппарлық емес. Олар туындаған мәселені ерте ме, кеш пе, әйтеуір шеше алатындығына нық сенімді болады. Олар үшін бұл – уақыттың еншісіндегі іс. Сондықтан өз-өзіңізді жете бағалаңыз, дамып

отырыңыз және ешкімнің төмендетуіне жол бермеңіз. Өйткені өзіңізге деген сенім – сіздің ең құнды қазынаңыз.

6. Жан-жақты дамыңыз. Мансап жолыңызда үнемі дамып отыруыңыз қажет. Дамуды, үйренуді ақшадан биік қойыңыз. Студент кезімде тәжірибе жинау үшін тегін жұмыс жасауға дайын едім. Ол тәжірибе маған болашақта ауадай қажет екенін жақсы түсінетінмін. Егер тәжірибелі, қабілетті, мықты маман болсаңыз ақша өзі келетінін естен шығармаңыз.

Бағдарламалаудағы келесі қадам

Құттықтаймыз, қымбатты оқырман! Сіз Python негіздерін меңгердіңіз, айнымалылармен, циклдермен, функциялармен және деректер құрылымдарымен жұмыс істеп үйрендіңіз. Ал енді ары қарай не істеу керек? Алынған білімді логикалық ойлау қабілетімізді дамытатын, күрделі мәселелерді шешуге және IT саласында табысқа жетуге бағытталған құралға қалай айналдыруға болады?

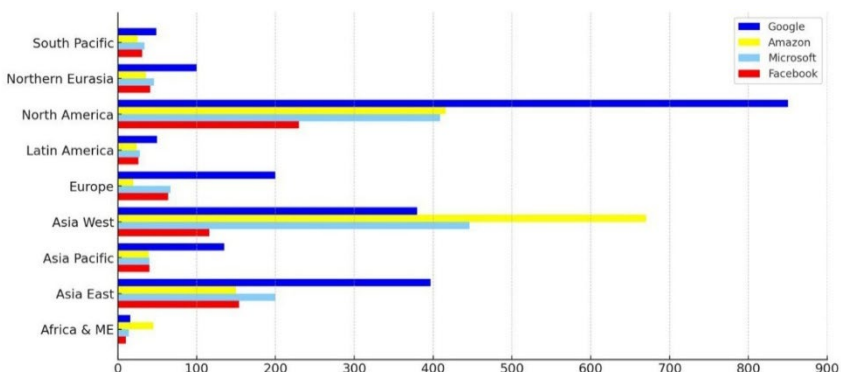
Аталған мақсаттарға жеткізетін ең тиімді әдістердің бірі – спорттық бағдарламалау. Спорттық бағдарламалау – IT, білім, ғылым және спорттың бірегей үйлесімі, онда қатысушылар алгоритмдік ойлау дағдыларын дамытып, бағдарламалау тілдерінің көмегімен стандартты емес тапсырмаларды шешеді.

Мұның маңызы неде?

ICPC (International Collegiate Programming Contest) жүргізген зерттеулердің нәтижесі (диаграмманы қараңыз) спорттық бағдарламалаудың мансаптық өсуге игі әсері барын дәлелдейді. ICPC — студенттер арасында ең ірі және беделді халықаралық бағдарламалау жарысы. Ол 1970 жылдан бері өткізіліп келеді және болашақ IT-мамандары үшін негізгі турнир саналады. Қатысушылар командалық форматта күрделі алгоритмдік тапсырмаларды шешіп, әлемдегі ең үздік бағдарламашылар атағына таласады.

ICPC зерттеу деректеріне сүйенсек, 30 000-нан астам түлектердің (спорттық бағдарламашылардың) еңбекпен қамтамасыз етілу көрсеткіші төмендегі пайыздық үлесті құрайды екен:

- 40% қатысушылар Google-де жұмыс істейді;
- 25% қатысушылар Amazon-ды таңдаған;
- 23% қатысушылар Microsoft-та еңбек етеді.
- 12% қатысушылар өз орнын Facebook (Meta)-дан тапқан.



Жоғары өнімділікке ие таланттың негізгі қайнар көзі

40 жылдан астам уақыт бойы, ICPC бағдарламалау өнеріндегі және компьютерлік ғылым оқу бағдарламасындағы алгоритмдік шеберлікті арттырды. Топ-4 компанияда жұмыс істейтін ICPC түлектерінің таралуы:

40% Google #1
25% Amazon #2
23% Microsoft #3
12% Facebook #4

ICPC түлектеріне арналған сауалнама (1999-2017 жылдары жүргізілген) – әлем бойынша 30 000 ICPC түлегінен алынған таңдама негізінде. Түлектердің 10%-ы туралы ақпарат беретін іздеу нәтижелерінен 5 514 жазба.

Global Ecosystem of Competitive Programming

Жоғарыдағы көрсеткішке қарап, жарыстарға қатысу сала мамандарының кәсіби шеберлігін қалыптастырып қана қоймай, озық технологиясы бар беделді компанияларға орналасу мүмкіндігін айтарлықтай арттыратынын аңғарамыз. Алайда спорттық бағдарламалаудың маңызы кейде IT саласының шеңберінен де шығып кетеді. Өйткені алгоритмдік тапсырмаларды шешу дағдылары ғылыми зерттеулерде, қаржыда, инженерияда, тіпті кәсіпкерлікте де қолданылады.

Бағдарламалаудағы келесі қадам қандай?

Сіз Python-ды меңгердіңіз, енді кодты сенімді түрде жаза аласыз. Бірақ бағдарламалау тек синтаксис білімінен ғана емес, күрделі тапсырмаларды шешу, алгоритмдік ойлау және деректермен тиімді жұмыс істеу қабілетінен де тұрады. Сол себепті сізді келесі қадам тосады. Ол – өзіңізді спорттық бағдарламалауда сынап көру.

Жоғарыда тоқталып өткеніміздей, спорттық бағдарламалау ойлау жылдамдығы мен аналитикалық қабілеттерді дамытып қана қоймай, әлемдегі ең ірі технологиялық компаниялардың есігін ашады. Сол себепті де осы бағытта ізденген дұрыс. Бұл ретте өз

дағдыларыңызды Қазақстанның Спорттық Бағдарламалау Федерациясымен бірге дамытуға болатынын еске саламыз. Федерация елімізде спорттық бағдарламалауды кеңінен насихаттап, барлық ниет білдірушілер үшін тегін негізде жарыстар мен білім беру бағдарламаларын ұйымдастырады. Спорттық бағдарламалау, оқу мүмкіндіктері және алдағы жарыстар туралы толығырақ ақпаратты Федерацияның ресми сайты cpfed.kz арқылы алуға болады.

Қымбатты досым, Python-ды меңгергеннен кейін, келесі қадамды жасауға толық мүмкіндігіңіз барын естен шығармаңыз. Іске сәт! Биік белестерді бірге бағындырайық!