«ТҰРАН» УНИВЕРСИТЕТІ

КИМ Е.Р.,
СЫДЫБАЕВА М.А.,
МОЛДАКАЛЫКОВА Б.Ж.

АЛГОРИТМДЕР, ДЕРЕКТЕР ҚҰРЫЛЫМЫ ЖӘНЕ PYTHON ТІЛІНДЕ БАҒДАРЛАМАЛАУ

Оқу-әдістемелік құрал

Алматы
2020

УДК 004.438
КБЖ 32.973-018.1
К40

Баспаға «Тұран» университетінің Ғылыми кеңесі ұсынған
(09.04.2020 ж. № 11 хаттама)

Ким Е.Р., Сыдыбаева М.А., Молдакалыкова Б.Ж.
К40 Алгоритмдер, деректер құрылымы және Python тілінде бағдарламалау: Оқу-әдістемелік құрал/ Е.Р. Ким, М.А. Сыдыбаева, Б.Ж. Молдакалыкова – Алматы: «Туран» университеті, 2020. – 110 с.

Оқу-әдістемелік құрал 6В06101 – Ақпараттық жүйелер мамандығының студенттеріне арналған "Алгоритмдер, деректер құрылымы және Python тілінде бағдарламалау" пәні бойынша дәріс материалынан тұрады. Ол осы мамандық бойынша модульдік білім беру бағдарламасында қарастырылған дәрістік сабақтардың барлық тақырыптарынан тұрады. Бағдарламалау тілі ретінде атап айтқанда, қазіргі уақытта танымал тілдердің бірі Python тілі таңдалды.

Оқу-әдістемелік құрал техникалық мамандықтар студенттеріне, магистранттарға, инженерлерге, сондай-ақ Python тілінде бағдарламалаудың тәжірибелік дағдыларын өз бетінше әзірлегісі келетіндерге арналған.

Сурет – 111, кесте – 3, Әдебиеттер тізімі – 26

КІРІСПЕ

Төртінші өнеркәсіптік революцияға кіру дәуірінде адам өмірінің барлық салаларына медицина, food-нарық, банк саласы, телекоммуникация, білім беру және т. б. болсын, Ақпараттық технологиялар (ат) енгізілді немесе енгізіледі. Сондықтан қазіргі уақытта "Ақпараттық жүйелер" және "сіз сандық техника және бағдарламалық қамтамасыз ету" мамандықтары бойынша оқитын студент жоғары оқу орнын бітіргеннен кейін еңбек нарығында сұранысқа ие мамандар болып табылады.

Жоғары оқу орнында оқу процесінде, әсіресе бірінші курста қалыптасатын және алынатын дағдылардың бірі Алгоритмдеу және бағдарламалау дағдысы болып табылады. Алгоритмдеу және бағдарламалау негіздерін білу, сондай-ақ кез келген күрделілік деңгейіндегі есептерді сенімді түрде шеше білу осы пән бойынша емтихан тапсыруға ғана емес, сонымен қатар келесі курстарда да проблемасыз кез келген бағдарламалау тілін меңгеруге көмектеседі.

Ұсынылған оқу құралында 6В06101 – Ақпараттық жүйелер мамандығының студенттеріне арналған "Алгоритмдер, деректер құрылымы және Python тілінде бағдарламалау" пәні бойынша дәріс курсы келтірілген.

Дәріс курсы 6В06101 – Ақпараттық жүйелер мамандығы бойынша "Алгоритмдер, деректер құрылымы және бағдарламалау" компоненті пәні бойынша модульдік білім беру бағдарламасында қарастырылған дәрістік сабақтардың барлық тақырыптарынан тұрады.

Бағдарламалау тілі ретінде қазіргі уақытта танымал тілдердің бірі таңдалды, атап айтқанда Python тілі. Бағдарламалық жасақтама ортасы ретінде Visual Studio 2019 Community ортасы таңдалған, оны Microsoft ресми сайтынан тегін жүктеуге болады.

Оқу-әдістемелік құралы Python бағдарламалау тілін үйрену үшін жеткілікті барлық қажетті тақырыптарды қамтиды және үш тараудан тұрады.

Бірінші тарауда Python тілінің негізгі синтаксисі қарастырылған, атап айтқанда: бағдарламаның құрылымы, деректер типтері, сандарға операциялар, тілдің тармақтаушы және қайталаушы құрылымы, бағдарламаларда функциялар мен модульдерді пайдалану, ерекшеліктерді өңдеу.

Екінші тарау Python бағдарламалау тілінде қолданылатын негізгі деректер құрылымдарына арналған. Тізімдер, салынған тізімдер, кортеждер, сөздіктер, жиындар, жолдар мен файлдар сияқты деректер түрлері қарастырылған.

Үшінші тарау объектілі-бағытталған бағдарламалаудың негізгі ұғымдарын қамтиды. Сондай-ақ, онда Python Tkinter бағдарламалау тілінің кіріктірілген кітапханасы көмегімен бағдарламалардың графикалық интерфейсін құру қарастырылған.

Оқу-әдістемелік құралы арнайы техникалық студенттерге, магистранттарға және ақпараттық технологияларға қызығушылық танытатын мамандарға арналған.

1 – ТАРАУ. PYTHON ТІЛІНІҢ СИНТАКСИС НЕГІЗДЕРІ

1.1 Python тіліндегі бағдарламаның құрылымы

1.1.1 Бағдарламаның құрылымы

Python тіліндегі бағдарлама нұсқаулар жиынтығынан тұрады. Әрбір нұсқаулық жаңа жолға қойылады. Мысалы [1-5]:

```
print(2 + 3)
print("Hello")
```

Python-да шегініс үлкен рөл атқарады. Қате қойылған шегініс шын мәнінде қате болып табылады. Мысалы, келесі жағдайда біз қате деп есептейміз, бірақ код жоғарыда келтірілгенге ұқсас болсада:

```
print(2 + 3)
    print("Hello")
```

Сондықтан жаңа нұсқауларды алдымен жолдарға қою керек. Бұл C# немесе Java сияқты басқа бағдарламалау тілдерінен Пайтонның маңызды айырмашылықтарының бірі.

Дегенмен, тілдің кейбір конструкциялары бірнеше жолдан тұруы мүмкін екенін ескеру керек. Мысалы, if шартты құрылымы:

```
if 1 < 2:
  print("Hello")
```

Бұл жағдайда 1 саны 2-ден кем болса, "Hello" жолы шығады. Және мұнда шегініс болуы керек, өйткені print("Hello") нұсқаулығы өзі емес, if шартты конструкциясының бір бөлігі ретінде пайдаланылады. Сонымен қатар, кодты рәсімдеу жөніндегі басшылыққа сәйкес, 4 (яғни 4, 8, 16 және т.б.) еселік бос орындардан, егер шегіністер 4 ЕМЕС, 5 болса да, бағдарлама да жұмыс істейтін болады.

Мұндай құрылымдар соншалықты көп емес, сондықтан қай жерде қажет, ал бос орындарды қою қажет емес деген ерекше шатасулар туындауы тиіс емес.

Тіркелімге тәуелділік. Python-тіркелімге тәуелді тіл, сондықтан print және Print немесе PRINT өрнектері әртүрлі өрнектерді ұсынады. Егер консольге шығару үшін print әдісінің орнына Print әдісін қолдануға тырыссаңыз:

```
Print("Hello World")
```

онда бізде ештеңе шықпайды.

1.1.2 Түсініктеме

Кодтың белгілі бір бөлігін не істеу керектігін белгілеу үшін пікірлер қолданылады. Бағдарламаны трансляциялау және орындау кезінде интерпретатор түсіндірмелерді елемейді, сондықтан олар бағдарламаның жұмысына ешқандай әсер етпейді.

Python-да пікірлер блоктық және жол. Олардың бәрі тор (#) белгісімен белгіленеді.

Блоктық пікірлер жолдың басында қойылады:

```
# Хабарламаны консольге шығару
print("Hello World")
```

Жолдық пікірлер тіл нұсқаулықтары сияқты жолда орналасады:

```
print("Hello World")  # Хабарламаны консольге шығару
```

Бағдарламалаудың жақсы тәжірибесі арнайы нұсқаудың бірінші жолында орналастыру болып табылады – shebang (shebang), ол леп белгісі бар тор – бұл скрипт не істеп жатқанын көрсете отырып, ерекше пікірдің түрі [26].

```
#! Python бірінші бағдарламасы
print("Hello World")       # Хабарламаны консольге шығару
```

1.1.3 Айнымалыларды сипаттау

Айнымалы белгілі деректерді сақтайды. Python-дағы айнымалының атауы алфавиттік таңбадан немесе астын сызу белгісінен басталуы тиіс және алфавиттік-сандық таңбалар мен астын сызу белгісін қамтуы мүмкін. Сонымен қатар, айнымалы атауы Python тілінің кілт сөздерінің атауымен сәйкес келмеуі тиіс. Негізгі сөздер көп емес, оларды есте сақтау оңай [1]:

and	def	finally	in	or	while
as	del	for	is	pass	with
assert	elif	from	lambda	raise	yield
break	else	global	None	return	
class	except	if	nonlocal	True	
continue	False	import	not	try	

Мысалға, айнымалыны құрайық:

```
name = "Tom"
```

Мұнда "Tom" жолын сақтайтын name айнымалысы бар.

Пайтонда айнымалылардың екі түрі қолданылады: camel case и underscore notation.

Camel case айнымалының атауындағы әрбір жаңа сөз үлкен әріппен басталады дегенді білдіреді. Мысалы:

```
userName = "Tom"
```

Underscore notation айнымалының атауында астын сызу белгісімен бөлінетінін білдіреді. Мысалы:

```
user_name = "Tom"
```

Сонымен қатар, тіркелім тәуелділігін ескеру қажет, сондықтан Name және Name айнымалылары әр түрлі объектілерді көрсетеді [6-10].

1.1.4 Мәліметтер типтері

Айнымалы деректер түрлерінің бірінің деректерін сақтайды. Python-да сан, бірізділік, сөздіктер, жинақтар категорияларына бөлінетін әртүрлі деректер түрлері бар [1, 11-15]:

boolean – True немесе False логикалық мәні;

int – компьютер жадында 4 байтты пайдаланатын бүтін сан;

float – 8 байт, мысалы, 1.2 немесе 34.76 байт пайдаланылатын қалқымалы нүктесі бар санды білдіреді;

complex – кешенді сандар;

str – жолдар, мысалы, "hello". Python 3-Де.х жолдар Unicode кодтамасындағы таңбалар жиынтығын ұсынады;

bytes – 0-255 диапазонындағы сандар тізбегі;

byte array – байттар массиві, bytes сияқты өзгеруі мүмкін;

list – тізім;

tuple – кортеж;

set – бірегей нысандар коллекциясы;

frozen set – set сияқты, тек өзгеруі мүмкін емес (immutable);

dict – әр элементтің кілті мен мағынасы бар сөздік.

Python – динамикалық типтеу тілі. Ол оған берілген мәннен айнымалы деректер түрін анықтайды. Мысалы, Қос немесе бір тырнақшаларда жол берілген кезде Айнымалы str түрі бар. Python бүтін санды иеленгенде, автоматты түрде айнымалы түрін int ретінде анықтайды. Float объектісі ретінде айнымалыны анықтау үшін оған бүтін бөлгіш және бөлшек бөлгіш нүкте болып табылатын бөлшек сан беріледі. Қалқымалы нүктелі санды экспоненциалды жазбада анықтауға болады:

```
x = 3.9e3
print(x)  # 3900.0

x = 3.9e-3
print(x)  # 0.0039
```

Float саны тек 18 маңызды таңбаға ие болуы мүмкін. Осылайша, бұл жағдайда тек екі таңба қолданылады - 3.9. Егер Сан тым үлкен немесе тым аз болса, онда біз санды экспонентті пайдалана отырып, осындай нотацияда жаза аламыз. Экспоненттен кейінгі сан 10 санын көрсетеді, онда негізгі Сан - 3.9 көбейту керек.

Сонымен қатар, Бағдарлама жұмыс істеу барысында біз айнымалы түрін өзгерте аламыз, оған басқа түрдегі мән бере аламыз:

```
user_id = "12tomsmith438"  # тип str
print(user_id)

user_id = 234  # тип int
print(user_id)
```

Type() функциясының көмегімен ағымдағы айнымалы түрін динамикалық түрде білуге болады [16-20]:

```
user_id = "12tomsmith438"
print(type(user_id))  # <class 'str'>

user_id = 234
print(type(user_id))  # <class 'int'>
```

1.2 Деректерді енгізу және шығару

Біз print() функциясымен кездестік. Ол экранға әдепкі деректерді шығаруға жауап береді. Егер код файлда болса, онсыз болмайды. Интерактивті режимде кейбір жағдайларда онсыз жұмыс істеуге болады.

Деректерді бағдарламаға енгізу және оларды шығару бағдарламалауда маңызды. Бағдарламаны енгізбестен, онда кездейсоқ мәндер пайда болған жағдайларды қоспағанда, бірдей болар еді. Қорытынды бағдарлама жұмысының нәтижесін көруге, пайдалануға, әрі қарай жеткізуге мүмкіндік береді.

Әдетте, бағдарлама сыртқы көздерден түсетін әр түрлі кіріс мәліметтерінің кейбір ауқымын өңдеу қажет. Файлдарды, пернетақтаны, желіні, басқа программадан шығу деректерін соңғылары ретінде көрсете алады. Бірақ көптеген жағдайларда бұл монитордың экранына шығады.

Бағдарлама-ол үшін сыртқы ортамен бірдеңе алмасатын ашық жүйе. Егер тірі организм негізінен зат пен энергия алмасса, онда бағдарлама – Деректер, ақпарат [21-26].

1.2.1 Деректерді енгізу. Input() функциясы

Python деректер бағдарламасына енгізу үшін input() функциясы жауап береді. Бұл функция пайда болғанда, Бағдарлама орындалуын тоқтатады және пайдаланушы мәтінді енгізгенде күтеді. Содан кейін, ол Enter басқанда, input() функциясы енгізілген мәтінді алып, оны өзінің алгоритмдеріне сәйкес өңделетін бағдарламаға жібереді.

Егер интерактивті режимде input() командасын енгізсеңіз, онда қызықты ештеңе көре алмайсыз. Компьютер сіз Enter-ді енгізгенде немесе жай ғана Enter-ді басқанда күте алады. Егер сіз бір нәрсе енгізсеңіз, ол бірден экранда көрсетіледі [17]:

```
>>> input()
Yes!
'Yes!'
```

Input() функциясы енгізілген деректерді бағдарламаға жібереді. Оларды айнымалыға беруге болады. Бұл жағдайда интерпретатор жолды бірден шығармайды:

```
>>> answer = input()
No, it is not.
```

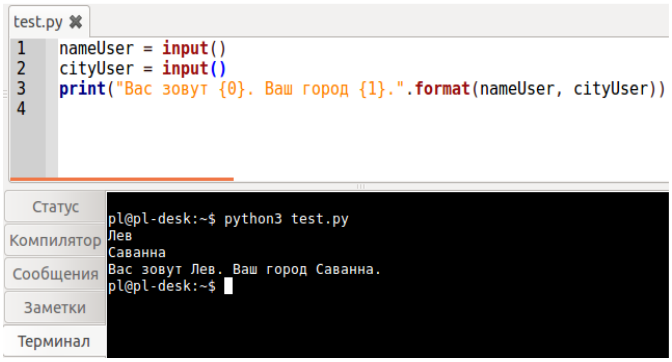
Бұл жағдайда жол answer айнымалысында сақталады және қаласаңыз, оның мәнін экранға шығара аламыз:

```
>>> answer
'No, it is not.'
```

Print() тырнақшалар функциясын пайдаланғанда шығыста төмен түсіріледі:

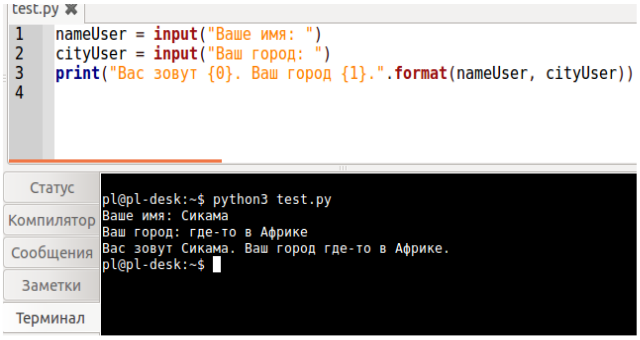
```
>>> print(answer)
No, it is not.
```

Код бар скриптер файлдарында input() функциясын қайда пайдалану қызықты. Мұндай бағдарламаны қарастырайық:

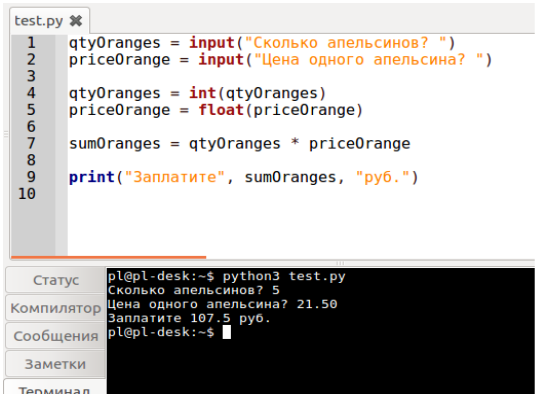


Бағдарламаны іске қосқан кезде компьютер алдымен бір жол, содан кейін екінші жол қашан енгізілгенін күтеді. Олар nameUser және cityUser айнымалы болады. Осы айнымалылардың мәндерін пішімделген шығару арқылы экранға шығарады.

Жоғарыда келтірілген скрипт кемелденуден алыс. Пайдаланушыға бағдарлама одан не қалайтынын қайдан білуге болады? Адамды шатастырмау үшін input() функциясы үшін арнайы шақыру параметрі қарастырылған. Бұл шақыру input() қоңырауы кезінде экранға шығады. Жетілдірілген бағдарлама осылай көрінуі мүмкін:



Бағдарламаға жол түсетініне назар аударыңыз. Егер Сан енгізілсе де, input() функциясы оның жол көрінісін қайтарады. Бірақ, егер санды алу керек болса, не істеу керек? Жауап: түрлерді түрлендіру функциясын пайдалану.



Бұл жағдайда int() және float() функцияларымен qtyOranges және priceOrange айнымалыларының жол мәндері тиісінше бүтін санға және заттай санға түрлендіріледі. Осыдан кейін жаңа сандық мәндер сол айнымалыға беріледі.

Input() функциясы пайда болатын сол код жолдарындағы түрлерді түрлендіру болса, бағдарламалық кодты қысқартуға болады [23]:

qtyOranges = int(input("Қанша апельсин? "))

```
priceOrange = float(input("Бір апельсин бағасы? "))
sumOranges = qtyOranges * priceOrange
print("Заплатите", sumOranges, "руб.")
```

Алдымен input() функциясы орындалады. Ол int() немесе float() функциясы бірден санға түрлендіретін жолды қайтарады. Тек осыдан кейін айнымалыны иелену жүреді, яғни ол бірден сандық мәнді алады.

1.2.2 Қорытынды деректер. Print() функциясы

Бағдарламалау функциясы дегеніміз не? Print () - бұл Python тілінің командасы, ол оның жақшасында экранды шығарады [10].

```
>>> print(1032)
1032
>>> print(2.34)
2.34
>>> print("Hello")
Hello
```

Жақшаларда деректердің кез келген түрі болуы мүмкін. Сонымен қатар, деректер саны әртүрлі болуы мүмкін:

```
>>> print("a:", 1)
a: 1
>>> one = 1
>>> two = 2
>>> three = 3
>>> print(one, two, three)
1 2 3
```

Print() функциясына тікелей литералдарды (бұл жағдайда "a:" және 1), сондай-ақ олардың мәндері шығарылатын айнымалыларды да жіберуге болады. Функцияның дәлелдері (жақшада) өзара үтірмен бөлінеді. Шығыста үтірдің орнына мәндер бос орынмен бөлінген.

Егер жақшада өрнек болса, алдымен ол орындалады, содан кейін print() осы өрнектің нәтижесін шығарады:

```
>>> print("hello" + " " + "world")
hello world
>>> print(10 - 2.5/2)
8.75
```

Print() бағдарламасында қосымша параметрлер қарастырылған. Мысалы, sep параметрі арқылы бос емес жол бөлгішін көрсетуге болады:

```
>>> print("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun", sep="-")
Mon-Tue-Wed-Thu-Fri-Sat-Sun
>>> print(1, 2, 3, sep="//")
1//2//3
```

End параметрі жол шыққаннан кейін не істеу керектігін көрсетеді. Әдепкі бойынша жаңа жолға ауысады. Алайда, бұл әрекетті кез келген басқа таңбаны немесе жолды көрсету арқылы жоюға болады:

```
>>> print(10, end="")
10
```

Әдетте, егер end қолданылса, онда интерактивті режимде емес, скриптерде, бірнеше қорытындыларды қатар жаңа жолға өтпей, үтірмен бөлу керек. Жаңа жолға ауысудың өзі "\n" таңбалар комбинациясымен белгіленеді. Егер бұл мәнді end параметріне қойсаңыз, print() функциясында ешқандай өзгерістер болмайды, себебі бұл мән және әдепкі бойынша берілген:

```
>>> print(10, end='\n')
10
```

Алайда, егер шығарғаннан кейін бір қосымша жолға шегіну керек болса, сіз осылай жасай аласыз:

```
>>> print(10, end='\n\n')
10
```

Print() функциясы туралы не айту керек – бұл жолдарды пішімдеуді пайдалану. Шын мәнінде, бұл print() ешқандай қатысы жоқ, ал жолдарда қолданылады. Бірақ әдетте print() функциясымен бірге қолданылады [12, 17-22].

1.3 Сандармен операциялар

1.3.1 Арифметикалық операциялар

Python барлық жалпы арифметикалық операцияларды қолдайды [1]:

```
+
Екі сандарды қосу:
print(6 + 2)  # 8
-
Екі сандарды азайту:
print(6 - 2)  # 4
*
Екі сандарды көбейту:
print(6 * 2)  # 12
/
Екі сандарды бөлу:
print(6 / 2)  # 3.0
//
Екі санның бүтін бөлінуі:
print(7 / 2)  # 3.5
print(7 // 2)  # 3
```

Бұл операция бөлшек бөлігін алып тастай отырып, бөлудің бүтін нәтижесін қайтарады

```
**
Дәрежеге тұрғызу:
print(6 ** 2)  # Возводим число 6 в степень 2. Результат – 36
%
```

Бөлуден қалдықты алу:

```
print(7 % 2)  # Получение остатка от деления числа 7 на 2. Результат -1
```

Бұл жағдайда 7-ге жақын сандар, ол 2-ге бөлінеді, бұл 6. Сондықтан бөлуден қалған қалдық 7-6 = 1 тең

Бірнеше арифметикалық операцияларды жүйелі түрде пайдаланған кезде оларды орындау олардың басымдығына сәйкес жүргізіледі. Басында үлкен басымдықпен операциялар орындалады. Кему тәртібінде операциялардың басымдықтары келесі кестеде келтірілген [1, 17].

Операциялар	Жолдау
**	Оң жақта солға
* / // %	Солдан оңға қарай
+ -	Солдан оңға қарай

Бізде келесі өрнек орындалсын:

```
number = 3 + 4 * 5 ** 2 + 7
print(number)  # 110
```

Мұнда басында үлкен басымдықпен операция ретінде (5 ** 2) дәрежеге тұрғызу орындалады, одан әрі нәтиже 4-ке (25 * 4) көбейтіледі, содан кейін қосу (3 + 100) болады және одан әрі қайтадан қосу (103 + 7) жүргізіледі.

Операция тәртібін қайта анықтау үшін жақшаларды қолдануға болады:

```
number = (3 + 4) * (5 ** 2 + 7)
print(number)  # 224
```

Арифметикалық операцияларға бүтін және Бөлшек сандар да қатыса алады. Егер бір операцияға бүтін сан (int) және қалқымалы нүкте (float) бар сан қатысса, онда бүтін сан float түріне келтіріледі.

1.3.2 Берілген арифметикалық операциялар

Бірқатар арнайы операциялар операция нәтижесін бірінші операндқа беруге мүмкіндік береді [1, 5-10, 17]:

+= Қосу нәтижесін беру
-= Шегеру нәтижесін беру
*= Көбейту нәтижесін беру
/= Бөлуден нәтиже беру
//= Бүтін бөлу нәтижесін беру
**= Дәрежесін беру санының
%= Бөлуден қалдықты беру

Операциялар мысалдары:

```
number = 10
number += 5
print(number)  # 15
number -= 3
print(number)  # 12
number *= 4
print(number)  # 48
```

1.3.3 Салыстыру операциялары

Тағайындау операторлары мәнді біртіндеп арттыруға немесе азайтуға, сондай-ақ кейбір есептеулерді автоматтандыруға мүмкіндік береді.

Бірқатар операциялар шартты өрнектерді ұсынады. Барлық осы операциялар екі операндты қабылдайды және Python boolean түрін білдіретін логикалық мәнді қайтарады. Тек екі логикалық мән бар – True (өрнек шынайы) және False (өрнек жалған).

Қарапайым шартты өрнектер салыстыру операцияларын ұсынады, олар екі мәнді салыстырады. Python келесі салыстыру операцияларын қолдайды [1, 6-10, 17, 25]:

==

Егер екі операнда тең болса, True қайтарады. Әйтпесе False қайтарады.

!=

Егер екі операнда тең болмаса, True қайтарады. Әйтпесе False қайтарады.

> (артық)

Егер бірінші операндадан екіншісінен артық болса, True қайтарады.

< (кем)

Бірінші операндасы екіншісінен аз болса, True қайтарады.

> = (көп немесе тең)

Бірінші операндасы екіншісінен артық немесе тең болса, True қайтарады.

< = (аз немесе тең)

Бірінші операндасы екіншіге аз немесе тең болса, True қайтарады.

Салыстыру операцияларының мысалдары [17]:

```
a = 5
b = 6
result = 5 == 6  # сохраняем результат операции в переменную
print(result)  # False - 5 не равно 6
print(a != b)  # True
print(a > b)  # False - 5 меньше 6
print(a < b)  # True

bool1 = True
bool2 = False
print(bool1 == bool2)  # False - bool1 не равно bool2
```

Салыстыру операциялары әр түрлі объектілерді салыстыра алады - жолдар, сандар, логикалық мәндер, алайда екі операнда да операциялар бір типін ұсынуы тиіс.

1.3.4 Логикалық операциялар

Құрама шартты өрнектерді жасау үшін логикалық операциялар қолданылады. Python келесі логикалық операторлар бар [17]:

and (логикалық көбейту)

Егер Екі өрнектің екеуі де True тең болса, True қайтарады

```
age = 22
```

```
weight = 58
result = age > 21 and weight == 58
print(result)  # True
```

Бұл жағдайда оператор and Екі өрнектің нәтижелерін салыстырады: age > 21 weight = = 58. Егер осы екі өрнектер де True қайтарады, онда and операторы да True қайтарады. Бұл дегеніміз, ол басқа логикалық операция немесе True немесе False сақтайтын boolean типті айнымалы болуы мүмкін.

```
age = 22
weight = 58
isMarried = False
result = age > 21 and weight == 58 and isMarried
print(result)  # False, так как isMarried = False
```

or (логикалық қосу)

True тең болса, True қайтарады

```
age = 22
isMarried = False
result = age > 21 or isMarried
print(result)  # True, так как выражение age > 21 равно True
```

not (логикалық терістеу)

Өрнек False тең болса, True қайтарады

```
age = 22
isMarried = False
print(not age > 21)  # False
print(not isMarried)  # True
```

Егер оператор операндаларының бірі and False-ді қайтарса, онда басқа операндалар бағаланбайды, өйткені оператор кез келген жағдайда False-ді қайтарады. Мұндай мінез-құлық өнімділікті сәл арттыруға мүмкіндік береді, өйткені екінші операнды бағалауға ресурстарды жұмсауға тура келмейді.

Егер or операторы операндаларының бірі True-ді қайтарса, онда екінші операндалар бағаланбайды, өйткені оператор кез келген жағдайда True-ді қайтарады.

1.3.5 Сандарды түрлендіру функциялары

Python-да енгізілген функциялар қатары сандарға жұмыс істеуге мүмкіндік береді. Атап айтқанда, int() және float() функциялары тиісінше int және float түріне мән беруге мүмкіндік береді.

Мысалы, бізде келесі код болсын [13]:

```
first_number = "2"
second_number = 3
third_number = first_number + second_number
```

Біз "2" + 3 5 тең болады деп күтеміз. Алайда, бұл код ерекшелік жасайды, өйткені бірінші Сан шын мәнінде жолды білдіреді. Және бәрі қажет болса, жолды Int() функциясы арқылы санға келтіру керек:

```
first_number = "2"
```

```
second_number = 3
third_number = int(first_number) + second_number
print(third_number) # 5
```

Ұқсас түрде float() функциясы әрекет етеді, ол құбылмалы нүктемен санға түрлендіреді. Бірақ, жалпы, бөлшек сандармен, олармен жасалатын операциялардың нәтижесі дәл болмауы мүмкін екенін ескеру керек. Мысалы:

```
first_number = 2.0001
second_number = 5
third_number = first_number / second_number
print(third_number) # 0.40002000000000004
```

Бұл жағдайда біз 0.40002 санын алуды күтеміз, бірақ соңында бірқатар нөлдер арқылы тағы бір төрттік пайда болады. Немесе тағы бір өрнек:

```
print(2.0001 + 0.1)  # 2.1001000000000003
```

Бұл жағдайда нәтижені дөңгелектеу үшін round() функциясын пайдалана аламыз:

```
first_number = 2.0001
second_number = 0.1
third_number = first_number + second_number
print(round(third_number, 4))  # 2.1001
```

Функцияның бірінші параметрі-дөңгелектелетін Сан, ал екіншісі-үтірден кейін қанша белгі алатын сан болуы керек.

1.3.6 Сандарды ұсыну

Сандық айнымалыны әдеттегі анықтау кезінде ол ондық жүйеде мәнді алады. Бірақ ондықтан басқа, біз екілік, сегіздік және он алтылық жүйелерді пайдалана аламыз.

Екілік жүйеде санды анықтау үшін оның мәні 0 және префикс b қойылады [1, 17]:

```
x = 0b101   # 101 екілік санау жүйесінде 5 ке тең
```

Сегіздік жүйеде санды анықтау үшін оның мәні 0 және O префиксі қойылады:

```
a = 0o11    # 11 в восьмеричной системе равно 9
```

Он алтылық жүйеде санды анықтау үшін оның мәні 0 және X префиксі қойылады:

```
y = 0x0a # a он алтылық жүйеде 10 тең
```

Және басқа өлшеу жүйелеріндегі сандармен де арифметикалық операцияларды жүргізуге болады:

```
x = 0b101   # 5
y = 0x0a    # 10
z = x + y   # 15
print("{0} in binary {0:08b}   in hex {0:02x} in octal {0:02o}".format(z))
```

Әр түрлі есептеу жүйелерінде сандарды шығару үшін format функциясы қолданылады, ол жолда пайда болады. Бұл жолға әртүрлі пішімдер беріледі. "{0:08b}" екілік жүйесі үшін, онда 8 саны Сан жазбасында қанша белгі болуы керек екенін көрсетеді. Егер таңбалар Сан үшін талап етілгеннен артық көрсетілсе, онда қажетсіз позициялар нөлдермен толтырылады. Он алтылық жүйе үшін "{0:02x}" пішімі қолданылады. Мұнда бәрі ұқсас - Сан жазбасы екі белгіден тұрады, егер бір белгі қажет болмаса, оның орнына нөл қойылады. Ал сегіздік жүйеде жазу үшін "{0:02o}" пішімі пайдаланылады.

Скрипт жұмысының нәтижесі:

15 in binary 00001111 in hex 0f in octal 17

1.3.7 Math модулінің негізгі функциялары

Python-да кірістірілген math модулі математикалық, тригонометриялық және логарифмдік операцияларды орындау үшін функциялар жиынтығын ұсынады. Модульдің негізгі функцияларының кейбірі 1.1-кестеде келтірілген [8, 11, 19-20, 26-30].

Кесте 1.1 – Негізгі математикалық функциялар

Функция атауы	Функцияның сипаттамасы
pow(num, power)	Num санын power дәрежесіне тұрғызу
sqrt(num)	Num санының шаршы түбірі
ceil(num)	Санды ең жақын бүтін санға дейін дөңгелектеу
floor(num)	Санды ең жақын бүтін санға дейін дөңгелектеу
factorial(num)	Факториал саны
degrees(rad)	Радианнан градустарға аудару
radians(grad)	Градустан радианға ауыстыру
cos(rad)	Радиандағы бұрыш косинусы
sin(rad)	Радиандағы бұрыштың синусы
tan(rad)	Радиандағы бұрыш тангенсі
acos(rad)	Радиандағы бұрыш арккосинусы
asin(rad)	Радиандағы бұрыш арксинусы
atan(rad)	Радиандағы бұрыш арктангенсі
log(n, base)	Base негізі бойынша N сан логарифмі
log10(n)	N санның ондық логарифмі

Математикалық функцияларды пайдаланбас бұрын, бағдарламаның басында import math нұсқауын жазу қажет, бірақ сонда әрбір функцияны айтпас бұрын модуль – math атауын қосу қажет болады, мысалы, y = math.sin(x). Math модулін бірнеше рет шақыру болдырмау үшін басқа жолы-бағдарламаның басында келесі жазба жасау: from math import *.

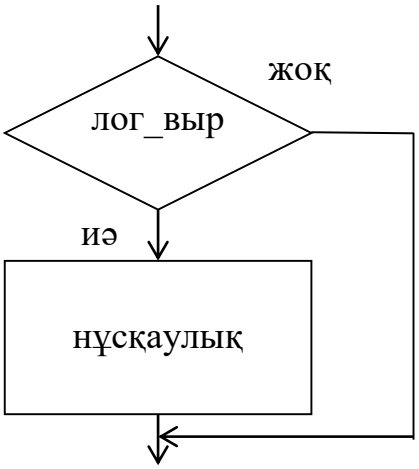
1.4 Тармақтаушы операторлар

Python тілі таңдау құрылымының екі түрін ұсынады. If таңдау құрылымында, егер шарт шынайы болса, немесе жалған болса, кейбір әрекет орындалады (таңдалады). If/else таңдау құрылымында, егер шарт шынайы болса, кейбір әрекет орындалады және бұл шарт жалған болса, басқа әрекет орындалады [1-5].

If құрылымы жеке таңдау бар құрылым деп аталады, өйткені онда бір әрекет таңдалады немесе елемейді. If/else құрылымы екі таңдау бар құрылым деп аталады, себебі таңдау екі балама әрекеттер арасында жүреді.

1.4.1 If жеке таңдау құрылымы.

Жалпы блок-схемалар түрі және if жазбасы құрылымының жалпы түрін келтірейік (сурет 1.1):



Сурет 1.1 – Дара таңдауы бар құрылымның блок-схемасы

```
if лог_выр:
нұсқаулық
```

Сурет 1.2 – If операторы жазудың жалпы түрі

Шартты конструкциялар шартты өрнектерді пайдаланады және олардың мәніне байланысты бағдарламаның орындалуын жолдың бірі бойынша жібереді. Осындай конструкциялардың бірі-if конструкциясы. Оның келесі формальды анықтамасы бар:

If кілт сөзінен кейін ең қарапайым түрде логикалық өрнек келеді. Егер бұл логикалық өрнек True қайтарады болса, онда келесі Нұсқаулық блогы орындалады, олардың әрқайсысы жаңа жолдан басталуы тиіс және жолдың басынан шегініс болуы тиіс (1.3-сурет).

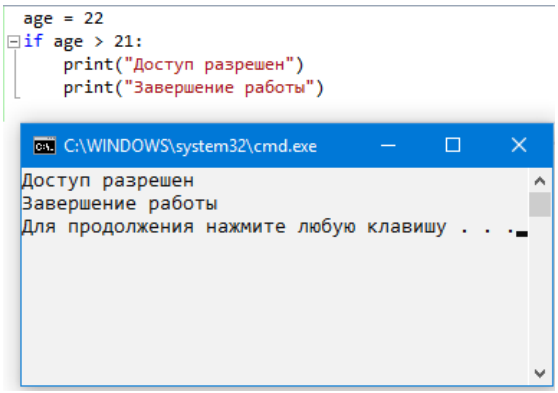
Бұл жағдайда age айнымалы мәні 21-ден көп болғандықтан, if блогы орындалады, ал консоль 1.3 суретте бейнеленген жолдарды шығарады.

Шегініс 4 бос орын немесе 4 есе бос орын санын жасаған жөн.

"Аяқтау" хабарын шығаратын соңғы стокқа кодқа назар аударыңыз. Ол жолдың басынан шегініс жоқ, сондықтан ол if блогына тиесілі емес және if конструкциясындағы өрнек False қайтаратын болса да, кез келген жағдайда орындалады.

Бірақ егер біз шегініс болса, онда ол сондай-ақ if конструкциясына жатады:

```
age = 22
if age > 21:
   print("Доступ разрешен")
   print("Завершение работы")
```



Сурет 1.3 – Листинг және жеке таңдауы бар құрылымның орындалу нәтижесі

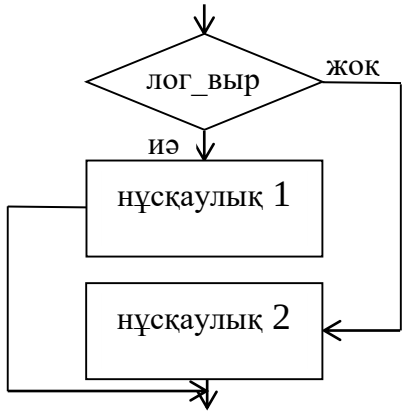
1.4.2 Қос таңдаулы if құрылымы

Егер шартты өрнек False қайтарса, онда біз else блогын пайдалана аламыз [1, 17, 20].

1.4-1.5 суретте блок-схеманың жалпы түрі және if/else операторы жазуының жалпы түрі көрсетілген.

Суретте 1.6 келтірілген листинг және коды орындалу құрылымын қос таңдау.

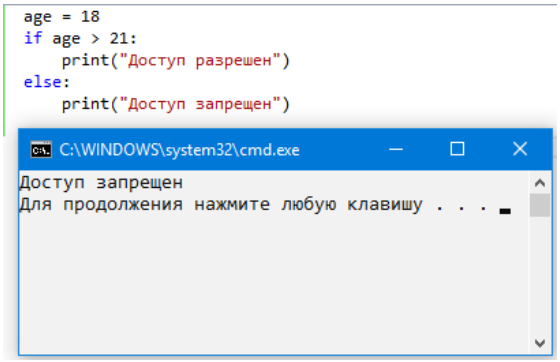
Егер age > 21 өрнегі True болса, if блогы орындалады, әйтпесе else блогы орындалады.



Сурет 1.4 – Қос таңдауы бар құрылымның блок-схемасы

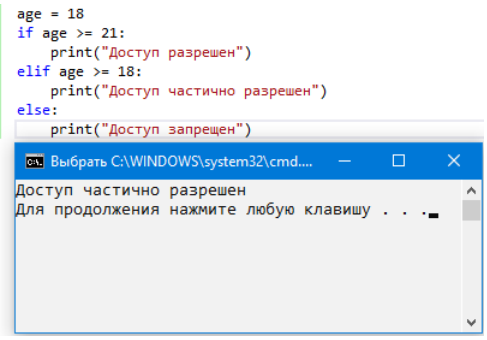
```
if лог_выр:
    нұсқаулық 1
else:
    нұсқаулық 2
```

Сурет 1.5 – If/else операторының жалпы жазба түрі



Сурет 1.6 – Листинг және екі таңдау бар құрылымның орындалу нәтижесі

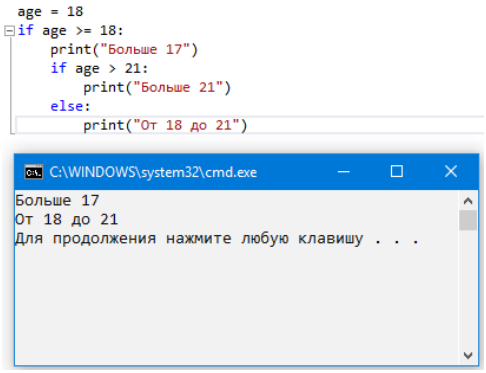
Егер бірнеше балама шарттарды енгізу қажет болса, онда қосымша elif блоктарын пайдалануға болады, содан кейін Нұсқаулық блогы өтеді (1.7 сурет).



Сурет 1.7 – Листинг және баламалы шарттармен құрылымның орындалу нәтижесі

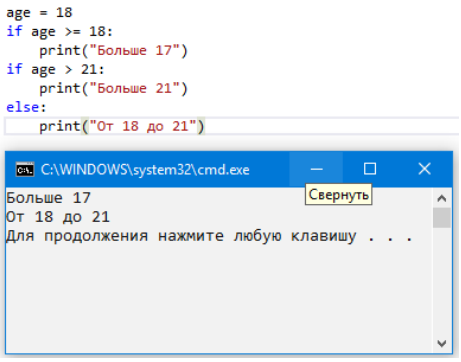
1.4.3 If салынған конструкциялары

If конструкциясы өз кезегінде if салынған конструкциялары болуы мүмкін (1.8 сурет).



Сурет 1.8 – Листинг және салынған құрылымдардың орындалу нәтижесі

If салынған өрнектері де шегіністерден басталуы тиіс, ал салынған конструкциялардағы нұсқаулықтар да шегініс болуы тиіс. Тиісті емес түрде қойылған шегіністер бағдарламаның логикасын өзгерте алады. Мысалы, алдыңғы мысал келесіге ұқсас емес (1.9 сурет).



Сурет 1.9

Енді шартты конструкцияларды пайдаланатын шағын бағдарламаны жазамыз. Бұл бағдарлама айырбастау пункті болып табылады (1.10 сурет).

```
#! Программа Обменный пункт
usd = 380
euro = 427
money = int(input("Введите сумму, которую вы хотите обменять: "))
currency = int(input("Укажите код валюты (доллары - 400, евро - 401): "))
if currency == 400:
    cash = round(money / usd, 2)
    print("Валюта: доллары США")
elif currency == 401:
    cash = round(money / euro, 2)
    print("Валюта: евро")
else:
    cash = 0
    print("Неизвестная валюта")
print("К получению:", cash)
```

```
C:\WINDOWS\system32\cmd.exe
Введите сумму, которую вы хотите обменять: 3000
Укажите код валюты (доллары - 400, евро - 401): 400
Валюта: доллары США
К получению: 7.89
Для продолжения нажмите любую клавишу . . .
```

Сурет 1.10 – Листинг және бағдарламаны орындау нәтижесі "Айырбастау пункті»

Input() функциясының көмегімен пайдаланушы енгізген деректерді консольге аламыз. Бұл функция деректерді жол түрінде қайтарады, сондықтан енгізілген деректерді арифметикалық операцияларда пайдалану үшін int() функциясы арқылы бүтін санға келтіру керек.

Бағдарлама пайдаланушы айырбастау қажет қаражат санын және айырбастау қажет валюта кодын енгізетінін білдіреді. Валюта кодтары жеткілікті шартты: доллар үшін 400 және еуро үшін 401.

If конструкциясының көмегімен валюта кодын тексереміз және тиісті валюта бағамына бөлеміз. Бөлу процесінде құбылмалы нүктелі өте ұзын Сан пайда болғандықтан, онда үтірден кейін көптеген таңбалар болуы мүмкін, онда ол round() функциясы арқылы үтірден кейін екі санға дейін дөңгелектенеді [1, 17].

Соңында консольге алынған мән шығарылады. Мысалы, Бағдарламаны іске қосып, қандай да бір мәліметтерді енгіземіз (сурет 1.10).

1.5 Қайталанатын құрылымдар

Python тілі бағдарламашыға жиі кездесетін типтік тапсырмаларды шешу үшін конструкция салынған арқасында нақты бағдарламалардың прототиптерін тез жасауға мүмкіндік береді.

Sum функциясын шақыру арқылы тізім элементтерінің сомасын есептеу міндетін қалай шешкенімізді еске түсіріңіз ([1, 4, 5, 6, 7.0, 3, 2.0]) – функцияның тек бір ғана шақыруы!

Бұл тарауда біз Python тілінде әзірлеушінің өмірін айтарлықтай жеңілдететін бірнеше ұқсас тәсілдерді қарастырамыз [1-10].

1.5.1 For цикл

Мысалы, бізде num тізімі бар және біз оның элементтерінің әрқайсысын экранға әдемі шығарғымыз келеді:

num = [0.8, 7.0, 6.8, -6]

num
[0.8, 7.0, 6.8, -6]

print(num[0],'- number') 0.8 - number
print(num[1],'- number') 7.0 - number
AAAAAAAAAAAAAAAaaaaaaaaaaaaaaaaaa!

Егер тізімде бес жүз элемент болса?! Мұндай жағдайлар үшін Python циклдары бар. Циклдар бағдарламалауда қозғаушы күш болып табылады. Оларды түсіну шынайы тірі және пайдалы бағдарламаларды жазуға мүмкіндік береді!

Бұл мысал for циклін пайдалана отырып жазамыз:

```
num = [0.8, 7.0, 6.8, -6]
for i in num:
print(i, '- number')
```

0.8 - number
7.0 - number

6.8 - number
-6 - number

For циклі көрсетілген тізімнің барлық элементтерін таңдауға мүмкіндік береді. Цикл тізімдегі элементтердің саны қанша рет жұмыс істейді. Әрбір қадамда тізім элементі орналастырылатын айнымалы аты бағдарламашыны таңдайды. Біздің мысалда бұл i атымен айнымалы.

I айнымалысының бірінші қадамында 0.8-ге тең num тізімінің бірінші элементі беріледі. Содан кейін бағдарлама шегіністермен бөлінген for циклінің денесіне өтеді (төрт бос орын немесе бір табуляция). Цикл денесінде I айнымалысы берілетін print функциясы бар.

Келесі I айнымалы қадамында 7.0 тең тізімнің екінші элементі беріледі. Тізімдегі элементтер аяқталғанша, I және т. б. айнымалының мазмұнын көрсету үшін print функциясына қоңырау қойылады!

Көрсетілген тізімнің барлық элементтерін таңдау үшін for циклінің жалпы түрі келесідей:

```
for << переменная >> in << список >>:
    << тело цикла >>
```

Мысалы:

```
for i in [1, 2, 'hi']: print(i)
```

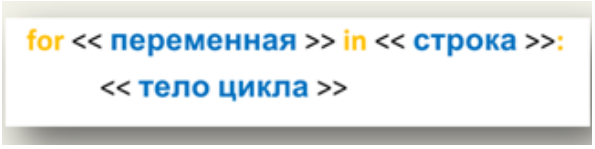
1
2
hi

Шын мәнінде, цикл for жұмыс істейді және жолдар үшін!

```
for i in 'hello': print(i)
```

h
e
l
l
o

Тізіммен ұқсас жолдар үшін барлық жол таңбалары іріктеледі.
Берілген жолға арналған for циклын жазу жалпы түрінде:



For циклы жол немесе Тізім элементтерін экранға шығаруға ғана емес, сонымен қатар белгілі бір операцияларды жасауға мүмкіндік береді:

```
num = [0.8, 7.0, 6.8, -6]
for i in num:
if i == 7.0:
print(i, '- число 7.0')
7.0 - число 7.0
```

Мысалы, әрбір цикл қадамында салыстыру арқылы тек белгіленген мәнді тізімнен шығара аламыз.
Осыған ұқсас циклде жол әдісін шақыру арқылы жолда қажетті таңбаны іздейді:

```
country = "Russia"

for ch in country: if ch.isupper():
print(ch)

R
```

Isupper жол әдісі таңбаның жоғарғы регистрін тексереді (ол үлкен әріптермен ма?), True немесе False қайтарады. Циклде жолдың әрбір символы тексеріледі. Егер таңба жоғарғы регистрде болса, онда ол экранға шығады.

1.5.2 Range функциясы

Бағдарламаларды әзірлеу кезінде жиі бүтін сандар тізбегін (ауқымын) алу қажет [17, 22]:

Бұл тапсырманы шешу үшін Python бағдарламасында Сан реттілігін (ауқымын) жасайтын range функциясы бар. Аргументтер ретінде функция қабылдайды: диапазонның бастапқы мәні (әдепкі 0), соңғы мәні (қоса саналмайды) және қадам (әдепкі 1). Егер функцияны шақырса, нәтиже көрмейміз:

```
range(0, 10, 1)
range(0, 10)
range(10)
range(0, 10)
```

Бұл сандар ауқымын құру үшін for циклін пайдалану қажет:

```
for i in range(0, 10, 1): print(i, end=' ')
0123456789
for i in range(10): print(i, end=' ')
0123456789
for i in range(2, 20, 2): print(i, end=' ')
24681012141618
```

Осылайша, циклдың әрбір қадамында I айнымалысына range функциясымен жасалатын ауқымның мәні жазылады.

Қаласаңыз, range функциясының аргументтеріне назар аударыңыз):

```
for i in range(20, 2, -2): print(i, end=' ')
201816141210864
```

Енді диапазонның көмегімен 1-ден 100-ге дейінгі аралықтағы сандар сомасын табамыз:

```
total = 0
for i in range(1, 101):
        total = total + i # total += i
total
5050
```

Циклдың әрбір қадамында 1-ден 100-ге дейінгі диапазон мән беріледі (шеткі мәнді қоспаймыз). Циклде біз есептегішті жинаймыз. Бұл нені білдіреді? Бірінші қадамда total айнымалысы 0-ге тең(цикл басталар алдында оған 0 мәнін берді), бірінші қадамда I айнымалысы 1 мәнін қамтиды (диапазоннан бірінші мән), осылайша, оң бөлік 1 мәніне тең болады және бұл мән өрнектің сол бөлігіне, яғни total айнымалысына беріледі.

Екінші қадамда total 1, і мәніне тең болады-2 мәні, яғни өрнектің оң бөлігі 3 тең болады, бұл мән қайтадан total және T .d беріледі. Нәтижесінде, total циклден шыққаннан кейін ізделетін сома болады!

Python-да осы тапсырманың әдемі шешімі бар:

```
sum(list(range(1, 101))) # sum(range(1, 101))
5050
```

Бұл шешім шағын түсіндіруді талап етеді. Тізімдерді жасау кезінде ауқымдарды пайдалануға болады:

```
list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list(range(2, 10, 2)) [2, 4, 6, 8]
```

Тізім үшін sum функциясын дәлел ретінде шақыру тізімнің барлық элементтерінің сомасын есептеуге әкеледі-бұл бізге қажет нәрсе!

Range функциясымен жасалған ауқым индекстер үшін жиі қолданылады. Мысалы, бар тізімді өзгерту қажет болса, әрбір элементті 2-ге көбейту:

```
lst = [4, 10, 5, -1.9]
print(lst)
for i in range(len(lst)):
lst[i] = lst[i] * 2
print(lst)
```

Бағдарламаны орындау нәтижесінде:

```
>>>
======== RESTART: C:/Python35-32/myprog.py ======
[4, 10, 5, -1.9]
 [8, 20, 10, -3.8]
>>>
```

Lst тізіміндегі барлық элементтер бойынша циклде өту қажет,ол үшін тізімнің элементтері олардың индексін көрсету арқылы біртіндеп іріктеледі және өзгертіледі. Range аргументі ретінде Тізім ұзындығы көрсетіледі. Бұл жағдайда жасалған диапазон 0-ден len(lst)-1-ге дейін болады. Python диапазонның шеткі элементін қамтымайды, өйткені Тізім ұзындығы әрқашан оның соңғы элементінің индексінен 1 артық, өйткені индекстеу нөлден басталады [11-15].

Тізімді жасау тәсілдері.

Тізімдерді жасаудың әртүрлі тәсілдерін қарастырайық. Ең айқын әдіс:

```
a = []
for i in range(1,15):
a.append(i)
a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

1-ден 14-ке дейінгі диапазон циклде сандарды таңдаймыз және append тізімдік әдісінің көмегімен оларды a тізіміне қосамыз.

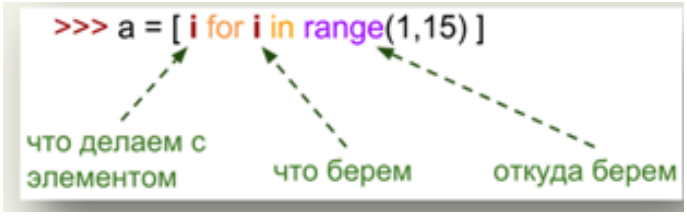
Ауқымнан тізімді жасау арқылы біз кездестік:

```
a = list(range(1, 15))
a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Сондай-ақ, "тізімді қосу" (кейде оны " тізім генераторы деп атайды»):

```
a = [i for i in range(1, 15)]
a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Тізімді қосу үшін жұмыс ережелері:



келесі мысал 1-ден 14-ке дейінгі барлық сандарды таңдап, оларды квадратқа салып, бірден жаңа тізімді қалыптастырамыз:

```
a = [i**2 for i in range(1,15)]
a
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196]
```

Тізімді қосу ауқымынан мәнді таңдау үшін шарт қоюға мүмкіндік береді (4 мәнін жоққа шығарды):

```
a = [i**2 for i in range(1,15) if i!=4]
```

```
a
[1, 4, 9, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196]
```

Тізімді қосу ауқымының орнына бар тізімді көрсетеді:

```
a = [2, -2, 4, -4, 7, 5]
b = [i**2 for i in a]
b
[4, 4, 16, 16, 49, 25]
```

мысалда біз А тізімінен бірізді мәндерді таңдаймыз, оның әрбір элементін квадратқа салып, алынған мәндерді жаңа тізімге бірден қосамыз.

Ұқсас таңбаларды жолдан таңдап, тізімді құрастыруға болады:

```
c = [c*3 for c in 'list' if c != 'i']
c
['lll', 'sss', 'ttt']
```

Python бар тізім негізінде жаңа тізімді жасауға мүмкіндік беретін қызықты map функциясы бар:

```
def f(x):
    return x + 5

list(map(f, [1, 3, 4]))
[6, 8, 9]
```

Map функциясы функция атауын және тізімін (немесе жол) аргументтер ретінде қабылдайды. Тізімнің (немесе жолдың) әр элементі функцияның кіруіне беріледі және функцияның жұмыс нәтижесі жаңа тізімнің элементі ретінде қосылады. Map қызметінің қоңырау нәтижесін for циклы немесе list функциясы арқылы алуға болады. Басқа функцияларға кіретін функциялар жоғары тәртіптің функциялары деп аталады.

Жол үшін map шақыру үлгісі:

```
def f(s):
    return s * 2

list(map(f, "hello"))
['hh', 'ee', 'll', 'll', 'oo']
```

Кездейсоқ бүтін сандардан тұратын тізімді қалай алуға болатынын қарастырайық:

```
from random import randint
A = [randint(1, 9) for i in range(5)]
A
[2, 1, 1, 7, 8]
```

бұл мысалда range функциясы қайталау санауышы ретінде әрекет етеді (for циклы дәл 5 рет жұмыс істейді). Жаңа тізімді құру кезінде і айнымалысы пайдаланылмайтынын ескеріңіз. Нәтижесінде бес рет randint функциясына қоңырау соғылады, ол интервалдан бүтін кездейсоқ санды түзеді және бұл сан жаңа тізімге қосылады.

Тізім үшін мәндерді қолмен енгізуге өтіңіз. Тізімнің ұзындығын орнатамыз және пернетақтадан оның барлық мәндерін енгіземіз [25]:

```
a = [] # бос тізімді жариялаймыз
```

N = int(input () # for i in range тізіміндегі элементтің санын санаймыз(n):
new_element = int (input ()) # келесі элементті оқыңыз a.append (new_element) # оны тізімге қосыңыз

соңғы екі жолды бір жолмен ауыстыруға болады:

```
a.append(int(input()))
print(a)
```

Бағдарламаны іске қосу нәтижесінде:

```
>>>
======== RESTART: C:\Python35-32\myprog.py ========
3
4
2
1
[4, 2, 1]
>>>
```

бұл range мысалында қайталау санауышы ретінде әрекет етеді, атап айтқанда, тізім ұзындығын анықтайды.

Енді бір жолға тізімді қосу арқылы осы мәселені жазыңыз:

```
A = [int(input()) for i in range(int(input()))]
3
4
2
1
A
[4, 2, 1]
>>>
```

1.5.3 Салынған циклдер

Циклдарды бір-біріне салуға болады.

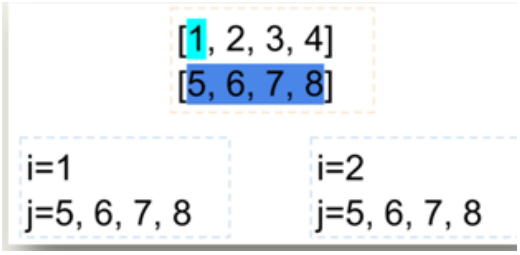
```
outer = [1, 2, 3, 4] # внешний цикл
inner = [5, 6, 7, 8] # вложенный (внутренний) цикл
for i in outer:
        for j in inner:
            print('i=', i, 'j=', j)
```

Бағдарлама жұмысының нәтижесі:

```
>>>
======= RESTART: C:\Python35-32\myprog.py ======
i= 1 j= 5
i= 1 j= 6
i= 1 j= 7
i= 1 j= 8
i= 2 j= 5
i= 2 j= 6
i= 2 j= 7
i= 2 j= 8
```

```
i= 3 j= 5
i= 3 j= 6
i= 3 j= 7
i= 3 j= 8
i= 4 j= 5
i= 4 j= 6
i= 4 j= 7
i= 4 j= 8
>>>
```

for циклі мысалында алдымен сыртқы циклдің барлық элементтері бойынша жылжиды (i = 1 тіркейміз), содан кейін ішкі циклге (J айнымалысы) ауысады және салынған тізімнің барлық элементтері бойынша өтеміз. Одан әрі сыртқы циклге ораламыз (келесі i = 2 мәнін белгілейміз) және салынған тізімнің барлық элементтері бойынша қайта өтеміз. Осылайша, сыртқы тізімде элементтерді аяқтағанға дейін қайталаймыз:



Бұл тәсіл ішкі тізімдермен жұмыс істегенде белсенді қолданылады.
Алдымен бір for циклімен мысал:

```
lst = [[1, 2, 3],
[4, 5, 6]]

for i in lst:
print(i)
```

Бағдарламаны орындау нәтижесі:

```
>>>
====== RESTART: C:\Python35-32\myprog.py =======
[1, 2, 3]
[4, 5, 6]
>>>
```

мысал ретінде for циклі арқылы тізімнің барлық элементтерін таңдайды.

Егер біз ішкі тізімдер элементтеріне жеткіміз келсе, онда сіз ішкі циклді пайдалану керек:

```
lst = [[1, 2, 3],
[4, 5, 6]]
for      i in lst: # цикл по элементам внешнего списка print()
   for j in i: # цикл по элементам элементов внешнего списка
       print(j, end="")
```

Бағдарламаны орындау нәтижесі:

```
>>>
```

```
======= RESTART: C:\Python35-32\myprog.py =======
123
456
>>>
```

1.5.4 While циклы

Бұрын болжағанындай, for циклы, Егер алдын ала қанша қайталауды орындау қажет екендігі белгілі болса (range функциясының аргументі арқылы көрсетіледі немесе тізім/жол аяқталғанша) қолданылады [15-20].

Егер алдын ала цикл қайталаудың саны белгісіз болса, онда while циклы деп аталатын басқа дизайн қолданылады:

Қояндардың санын анықтайық:

```
rabbits = 3
while rabbits > 0:
        print(rabbits)
        rabbits = rabbits – 1
```

Бағдарламаны орындау нәтижесінде:

```
>>>
========= RESTART: C:\Python35-32\myprog.py ==========
3
2
1
>>>
```

мысал while циклі қояндардың саны оң болғанша орындалады. Циклдың әрбір қадамында біз rabbits айнымалысы шарт әрқашан шынайы болған кезде шексіз циклге кетпес үшін 1-ге азайтамыз.

Бағдарламаны орындау барысын қарастырайық.

rabbits айнымалысы 3-ке тең, содан кейін while циклына түсеміз, өйткені rabbits > 0 шарты шынайы болып табылады (True мәнін қайтарады). Цикл денесінде rabbits айнымалысының ағымдағы мәнін экранда көрсететін print функциясы пайда болады. Содан кейін айнымалы 1-ге азаяды және тағы да while шарты тексеріледі, яғни 2 > 0 (True қайтарады). Циклге түсеміз және әрекеттер 0 > 0-ге дейін жеткенше қайталанады. Бұл жағдайда false логикалық мәні қайтарылады және while циклі жұмыс істемейді.

Келесі мысал қарастырайық:

```
while True:
        text = input("Введите число или стоп для выхода: ")
```

```
if text == "стоп":
    print("Выход из программы! До встречи!")
    break      # инструкция выхода из цикла
elif text == '1':
    print("Число 1")
else:
    print("Что это?!")
```

Бағдарлама жұмысының нәтижесінде:

```
>>>
============ RESTART: C:\Python35-32\myprog.py ===========
Шығу үшін сан немесе тоқта енгізіңіз: 4 Бұл не?!
Шығу үшін сан немесе тоқта енгізіңіз: 1
1 Саны
Шығу үшін сан немесе тоқта енгізіңіз: тоқта
Бағдарламадан шығу! Кездесуге дейін!
>>>
```

Бағдарлама шексіз циклде орындалады, өйткені True әрқашан шындық. Цикл ішінде пернетақтадан мәнді енгізу және енгізілген мәнді тексеру жүргізіледі. Break нұсқаулығы циклден шығады.

Мұндай бағдарламалар деректер түрлерінің түрленуін мұқият қадағалау қажет.

1.6 Функциялар

1.6.1 Функцияларды анықтау

Функциялар белгілі бір тапсырманы орындайтын және бағдарламаның басқа бөліктерінде қайта пайдалануға болатын код блогын білдіреді. Функцияның формальды анықтамасы [8, 26]:

```
def имя_функции ([параметры]):
   инструкции
```

Функцияны анықтау функцияның атауынан, параметрлері бар жақшалардың жиынынан және қос нүктеден тұратын def өрнегінен басталады. Жақшадағы параметрлер міндетті емес. Келесі жолдан функцияны орындайтын Нұсқаулық блогы бар. Барлық нұсқаулар Жол басынан шегініс бар.

Мысалы, қарапайым функцияны анықтау:

```
def say_hello():
   print("Hello")
```

Функция say_hello деп аталады. Оның параметрлері жоқ және консольге "Hello" жолын шығаратын жалғыз нұсқауы бар.

Функцияны шақыру үшін функцияның атауы көрсетіледі, одан кейін жақшада оның барлық параметрлері үшін мәндер беріледі (1.11-сурет):

```
def say_hello():
    print("Hello")

say_hello()
say_hello()
say_hello()
```

```
C:\Windows\system32\cmd.exe
Hello
Hello
Hello
Для продолжения нажмите любую клавишу .
```

Сурет 1.11. Параметрлерсіз функцияны анықтау

Мұнда үш рет қатарынан say_hello функциясы бар.
Енді параметрлермен функцияны анықтаймыз және қолданамыз (Сурет 1.12).

```
def say_hello(name):
    print("Hello,", name)

say_hello("Tom")
say_hello("Bob")
say_hello("Alice")
```

```
C:\Windows\system32\cmd.exe
Hello, Tom
Hello, Bob
Hello, Alice
Для продолжения нажмите любую клавишу .
```

Сурет 1.12. Параметрлермен функцияны анықтау

Функция name параметрін қабылдайды және функцияны шақырғанда, қандай да бір мәнді орнына бере аламыз.

1.6.2 Бастапқы мәндер

Кейбір функция опцияларын функцияны анықтау кезінде әдепкі мәндерді көрсету арқылы міндетті түрде жасай аламыз (Сурет 1.13) [26].

Мұнда name параметрі міндетті емес. Егер қоңырау шалу кезінде оған мәнді жібермесеңіз, әдепкі мән, яғни "Tom" жолы қолданылады.

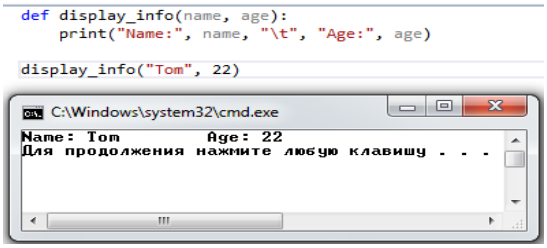
```
def say_hello(name="Tom"):
    print("Hello,", name)

say_hello()
say_hello("Bob")
```

```
C:\Windows\system32\cmd.exe
Hello, Tom
Hello, Bob
Для продолжения нажмите любую клавишу .
```

Сурет 1.13. Қосымша параметрлермен функцияны анықтау

1.6.3. Атаулы параметрлер

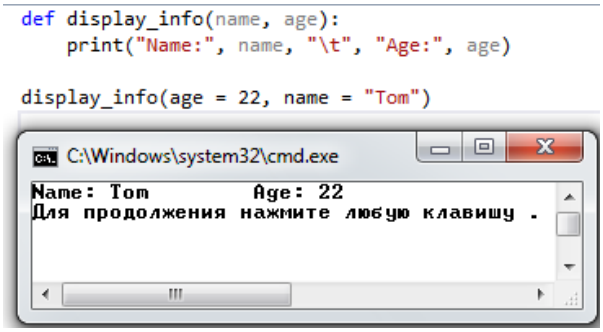
Мәндер берілгенде, функция оларды берілген тәртіппен параметрлермен салыстырады. Мысалы, келесі функция болсын (Сурет 1.14) [17, 26].



Сурет 1.14. Параметрлермен функцияны анықтау

Функцияны шақырған кезде "Tom" бірінші параметрге – name параметріне беріледі, екінші параметр – 22 Сан екінші параметрге – age беріледі. Және т.б. тәртіп бойынша. Атаулы параметрлерді пайдалану тасымалдау тәртібін қайта анықтайды (Сурет 1.15).

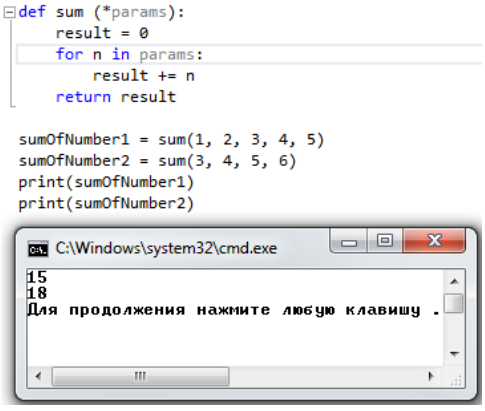
Аталған параметрлер функцияны шақыру кезінде оған мән берілетін параметр атауын көрсетеді.



Сурет 1.15. Аты берілген параметрлермен функцияны анықтау

1.6.4 Параметрлердің анықталмаған саны

Жұлдыз белгісі арқылы параметрлердің белгісіз санын анықтауға болады (Сурет 1.16) [17, 26].

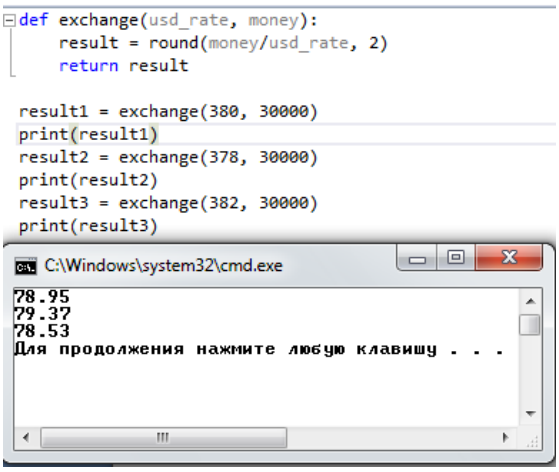


Сурет 1.16. Параметрлердің белгісіз санымен функцияны анықтау

Бұл жағдайда sum функциясы бір параметрді қабылдайды *params, бірақ жұлдызша осы параметрдің атауынан бұрын белгісіз мәндер санын немесе мәндер жиынтығын бере аламыз. Функцияның өзінде for циклі арқылы осы жиынтыққа барып, берілген мәндермен әр түрлі әрекеттер жасауға болады. Мысалы, бұл жағдайда сандар сомасы қайтарылады.

1.6.5 Нәтижені қайтару

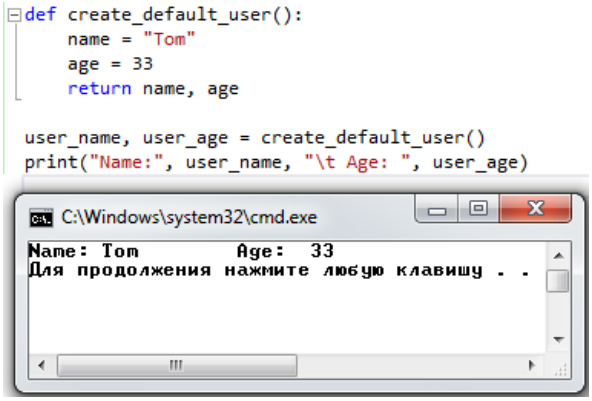
Функция нәтижені қайтаруы мүмкін. Бұл үшін функцияда return операторы қолданылады, одан кейін қайтарылатын мән көрсетіледі (сурет 1.17) [19-20, 26].



Сурет 1.17. Мәнді қайтаратын функцияны анықтау

Функция мәнді қайтарған кезде, біз бұл мәнді қандай да бір айнымалы етіп тағайындап, оны пайдалана аламыз: result2 = exchange(56, 30000).

Python функциясы бірден бірнеше мәндерді қайтаруы мүмкін (сурет 1.18) [26].



Сурет 1.18. Бірнеше мәндерді қайтаратын функцияны анықтау

Create_default_user функциясы екі мәнді қайтарады: name және age. Функцияны шақырғанда, бұл мәндер user_name және user_age айнымалыларымен беріледі және біз оларды пайдалана аламыз.

1.6.6 Main функциясы

Бағдарламада көптеген функциялар белгіленуі мүмкін. Және оларды ретке келтіру үшін, жақсы тәжірибе-арнайы main функциясын қосу, содан кейін басқа функциялар пайда болады (сурет 1.19).

```
def main():
    say_hello("Tom")
    usd_rate = 380
    money = 30000
    result = exchange(usd_rate, money)
    print("К выдаче", result, "долларов")
def say_hello(name):
    print("Hello,", name)
def exchange(usd_rate, money):
    result = round(money/usd_rate, 2)
    return result
# Вызов функции main
main()
```

```
C:\Windows\system32\cmd.exe
Hello, Tom
К выдаче 78.95 долларов
Для продолжения нажмите любую клавишу . . .
```

Сурет 1.19. Main басты функциясын анықтау

1.6.7 Көріну аймағы

Көріну аймағы немесе scope оны пайдалануға болатын айнымалы контекстті анықтайды. Python-да контекстің екі түрі бар: жаһандық және жергілікті.

Жаһандық контекст айнымалы жаһандық болып табылады, ол кез келген функциядан тыс анықталған және бағдарламада кез келген функцияға қол жетімді (сурет 1.20).

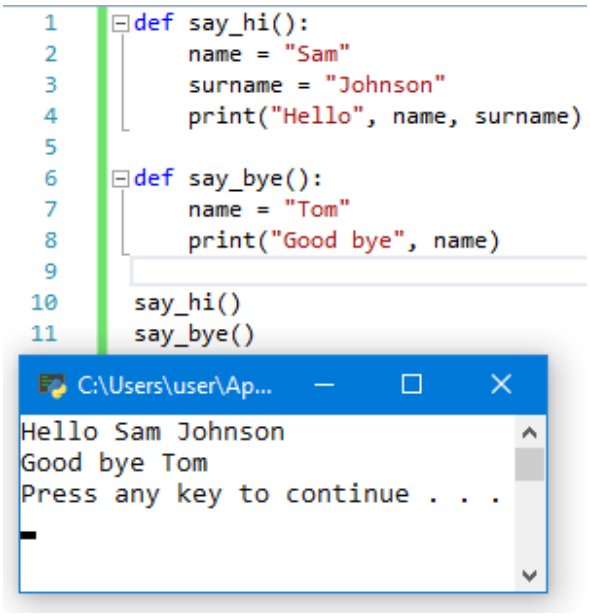
```
1  name = "Tom"
2
3  def say_hi():
4      print("Hello", name)
5
6  def say_bye():
7      print("Good bye", name)
8
9  say_hi()
10 say_bye()
11
12
```

```
C:\Users\user\...
Hello Tom
Good bye Tom
Press any key to continue . . .
```

Сурет 1.20. Ғаламдық айнымалыны анықтау

Мұнда айнымалы name жаһандық және жаһандық көріну аймағы бар. Және мұнда белгіленген екі функция оны еркін пайдалана алады.

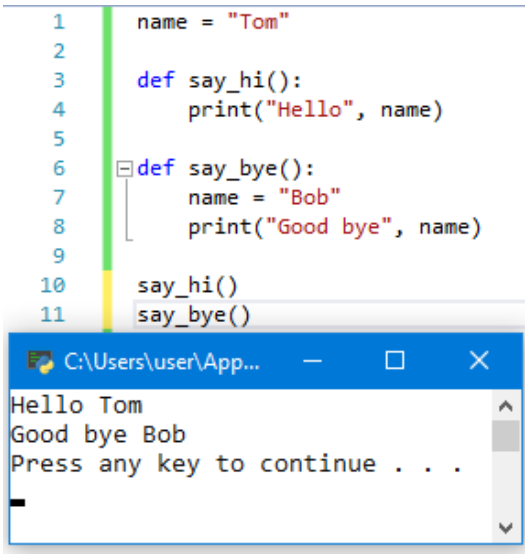
Жаһандық айнымалылардан айырмашылығы жергілікті айнымалы функцияның ішінде анықталады және тек осы функциядан қол жетімді, яғни жергілікті көріну аймағы бар (сурет 1.21).



Сурет 1.21. Жергілікті айнымалыны анықтау

Бұл жағдайда екі функцияның әрқайсысында жергілікті name айнымалысы анықталады. Бұл айнымалылар бірдей деп аталса да, бірақ бұл екі түрлі айнымалылар, олардың әрқайсысы өз функциясының шеңберінде ғана қол жетімді. Сондай-ақ, say_hi функциясында жергілікті surname айнымалысы анықталды, сондықтан say_bye функциясында біз оны пайдалана алмаймыз.

Жергілікті айнымалы сол атаумен жаһандық жасырын кезде Айнымалы анықтаудың тағы бір нұсқасы бар (сурет 1.22).



Сурет 1.22. Жаһандық айнымалыны жасыру

Мұнда ғаламдық айнымалы name анықталды. Дегенмен, say_bye функциясында name атымен жергілікті айнымалы анықталды. Егер say_hi функциясы жаһандық айнымалыны қолданса, онда say_bye функциясы жергілікті айнымалыны қолданады.

Егер біз жергілікті функцияда жаһандық айнымалыны емес, жергілікті функцияны өзгерткіміз келсе, онда global негізгі сөзін пайдалану қажет [1, 17]:

```
def say_bye():
    global name
    name = "Bob"
    print("Good bye", name)
```

Python-да, көптеген басқа бағдарламалау тілдерінде сияқты, жаһандық айнымалыларды пайдалану ұсынылмайды. Бағдарламаның жұмыс процесінде өзгермейтін жаһандық константалардың шағын санын анықтау жалғыз рұқсат етілген тәжірибе болып табылады.

```
PI = 3.14

 # шеңбер алаңын есептеу
def get_circle_square(radius):
print("Площадь круга с радиусом", radius, "равна", PI * radius * radius)

 get_circle_square(50)
```

Бұл жағдайда 3.14 саны константамен ұсынылған Pi. Бұл мағынада өзгермейді, сондықтан оны функциялардан алып, константа түрінде анықтауға болады. Әдетте, константаның аты бас әріптермен анықталады.

1.7 Модульдер

1.7.1 Модульдерді құру

Python тіліндегі Модуль басқа бағдарламаларда қайта пайдалануға болатын коды бар жеке файлды ұсынады [1, 17, 26].

Модульді құру үшін модульді ұсынатын *.py кеңейтуімен файлды жасау қажет. Файл атауы модульдің атауын көрсетеді. Содан кейін осы файлда бір немесе бірнеше функцияларды анықтау керек.

Бағдарламаның негізгі файлы деп аталсын hello.py біз оған сыртқы модульдерді қосқымыз келеді.

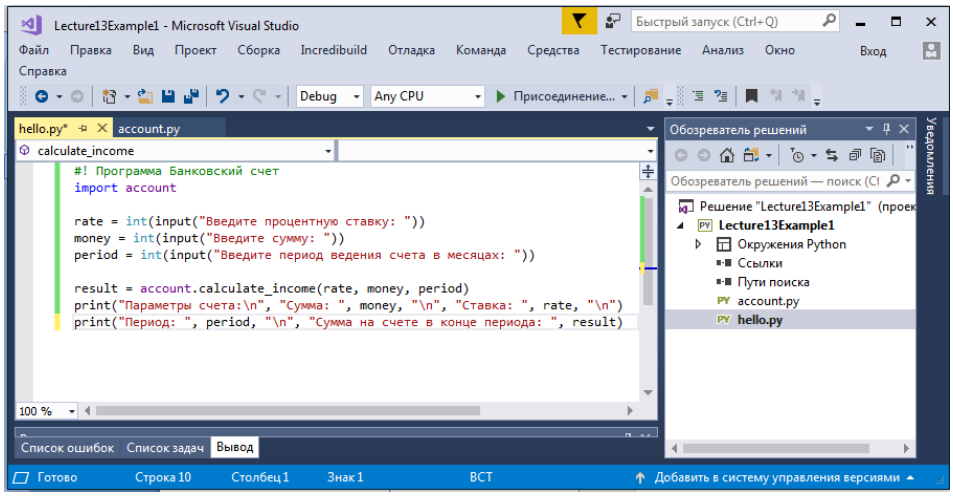
Ол үшін алдымен жаңа модульді анықтаймыз: жаңа файлды жасаймыз account.py, сол папкада орналасқан hello.py. Егер Microsoft Visual Studio немесе басқа IDE пайдаланылса, екі файл да бір жобаға орналастырылады (сурет 1.23).

Тиісінше, модуль account деп аталады, келесі кодты анықтаймыз:

```
def calculate_income(rate, money, month):
    if money <= 0:
        return 0
    for i in range(1, month+1):
        money = round(money + money * rate / 100 / 12, 2)
    return money
```

Мұнда параметрлер ретінде салымның пайыздық ставкасын, салым сомасын және салым жасалатын кезеңді алатын calculate_income функциясы анықталған және осы кезеңнің соңында алынатын соманы есептейді.

Файлда hello.py бұл модульді қолданамыз (сурет 1.23).



Сурет 1.23. Негізгі бағдарламаның листингі

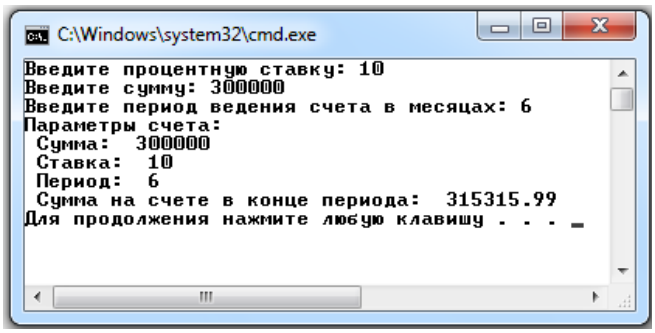
Модульді пайдалану үшін оны import операторының көмегімен импорттау керек, одан кейін Import account Модулінің атауы көрсетіледі.

Модульді пайдалану үшін, біз оның атау кеңістігін алу керек. Әдепкі бойынша ол модульдің атымен сәйкес келеді, яғни біздің жағдайда account деп аталады.

Модуль аттарының кеңістігін алып, біз схема бойынша оның функцияларына жүгіне аламыз пространство_имен.функция:

account.calculate_income(rate, money, period)

Осыдан кейін бас скриптті іске қосуға болады hello.py және ол модульді іске қосады account.py. атап айтқанда, консольдік қорытынды мынадай болуы мүмкін (сурет 1.24).



Сурет 1.24. Сыртқы модульді бағдарламаны орындау нәтижесі

1.7.2 Атау кеңістігін теңшеу

Әдепкі бойынша, модульді импорттаған кезде, ол аттардың аттас кеңістігі арқылы қол жетімді. Алайда, біз бұл мінез-құлықты қайта анықтай аламыз. Осылайша, as кілт сөзі модульді басқа аттар кеңістігімен салыстыруға мүмкіндік береді. Мысалы [1, 17, 22-26]:

```
import account as acc
#.............
result = acc.calculate_income(rate, money, period)
```

Бұл жағдайда атау кеңістігі acc деп аталады.

Параметрдің басқа опциясы from кілт сөзімен ағымдағы модуль атауларының жаһандық кеңістігіне модулдің функционалдығын импорттауды көздейді:

```
from account import calculate_income
#.............
result = calculate_income(rate, money, period)
```

Бұл жағдайда біз account модулінен calculate_income функциясына импорттаймыз. Сондықтан біз оны осы файлда анықталғандай, модуль атаулары кеңістігін көрсетпей пайдалана аламыз.

Егер account модулінде бірнеше функция болса, онда оларды бір өрнектермен Ғаламдық аттар кеңістігіне импорттай алады:

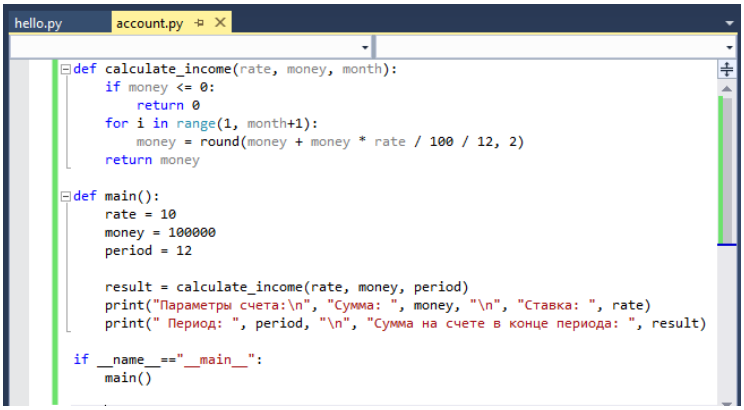
```
from account import *
#.............
result = calculate_income(rate, money, period)
```

Айта кету керек, жаһандық кеңістікке атау импорты функциялар атауларының коллизияларын бұзады. Мысалы, сол файлда бірдей аты бар функция анықталған болса, онда функцияны шақырған кезде, біз қатені ала аламыз. Сондықтан жаһандық атау кеңістігіне импортты пайдаланудан аулақ болу керек.

1.7.3 Модуль атауы

Мысалы, жоғарыда модуль hello.py, басты, модульді пайдаланады account.py. модульді іске қосу кезінде hello.py бағдарлама барлық қажетті жұмысты орындайды. Дегенмен, егер біз жеке модульді іске қосамыз account.py өзі, консольде ештеңе көрмейді. Модуль тек функцияны анықтайды және басқа әрекеттерді орындамайды. Бірақ біз модульді жасай аламыз account.py өздігінен де, басқа модульдерге де қосыла алады [22].

Модульді орындау кезінде Орта оның атын анықтайды және оның жаһандық айнымалысын ____ _ name (екі жағынан екі астын сызу) береді. Егер модуль іске қосылатын болса, онда оның аты _ _ _ main тең (сондай-ақ әр жағынан екі астын сызу). Егер модуль басқа модульде қолданылса, онда оның аты Ру кеңейтусіз файл атауына ұқсас. Және біз оны пайдалана аламыз. Осылайша, файл мазмұнын өзгертіңіз account.py (сурет 1.25).



Сурет 1.25. Модуль атауын тексерумен бағдарлама листингі

Сонымен қатар, тестілеу үшін main басты функциясы анықталған. Және біз бірден файлды жібере аламыз account.py барлығы бөлек және кодты сынау.

Main функциясына назар аудару керек:

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

Name айнымалысы модуль атауын көрсетеді. Тікелей іске қосылатын негізгі модуль үшін Бұл айнымалы файл атауына қарамастан ___ main мәні болады.

Сондықтан, егер біз іске скрипт account.py жеке өзі болса, Python присвоит ауыспалы __name__ мәні __main__, бұдан әрі мәнде if функциясын тудырады бастапқы бет келген осы файл.

Дегенмен, егер біз басқа скриптті іске қоссақ, ал бұл – account.py - көмекші ретінде қосамыз, account.py _name айнымалысы account мәні болады. Және тиісінше файлдағы main әдісі account.py жұмыс істемейді.

Модуль атауын тексерумен бұл тәсіл main әдісін шақырудан гөрі ұсынылған тәсіл болып табылады.

Файлда hello.py сондай-ақ, модульдің басты болып табылатындығын тексеруге болады (бірақ бұл міндетті емес):

```
#! Программа Банктік шот
import account
def main():
    rate = int(input("Пайыздық ставканы енгізіңіз: "))
    money = int(input("Соманы енгізіңіз: "))
    period = int(input("Айларда шот жүргізу кезеңін енгізіңіз: "))
    result = account.calculate_income(rate, money, period)
    print("Шот параметрлері:\n", "Сома: ", money, "\n", "Ставка: ", rate, "\n", "Аралық: ", period, "\n", " Кезең соңындағы шоттағы сома: ", result)

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

1.8 Ерекшеліктерді өңдеу

1.8.1 Ерекшелік түрлері

Python бағдарламалау кезінде біз екі қате түріне тап бола аламыз. Бірінші түрі синтаксистік қателер (syntax error). Олар бастапқы кодты жазу кезінде бағдарламалау тілінің синтаксисінің бұзылуы нәтижесінде пайда болады. Мұндай қателер болған жағдайда бағдарлама компиляциялануы мүмкін емес. Мысалы, Microsoft Visual Studio, IDE синтаксистік қателерді өзі бақылай алады және оларды бөліп алады.

Қателердің екінші түрі-орындау қателері (runtime error). Олар оны орындау барысында компиляцияланған бағдарламада пайда болады. Мұндай қателер әлі де ерекшеліктер деп аталады. Мысалы, санды жолға түрлендіруді қарастырайық [1-8, 15, 26]:

```
string = "5"
number = int(string)
print(number)
```

Бұл скрипт сәтті құрастырылады және орындалады, өйткені "5" жолы санға айырбасталуы мүмкін. Дегенмен, басқа мысал алыңыз:

```
string = "hello"
```

```
number = int(string)
print(number)
```

Бұл скриптті орындау кезінде ValueError-ді алып тастайды, себебі "hello" жолы санға түрлендірілмейді. Бір жағынан, мұнда жолдың саны емес, анық, бірақ біз сондай-ақ біз күтетін нәрсе емес, енгізуге мүмкін пайдаланушы енгізу іс болуы мүмкін:

```
string = input("Санды енгізіңіз: ")
number = int(string)
print(number)
```

Try..except құрылымы.

Мұндай жағдайда, бағдарлама жұмысы үзіледі және мұндай мінез-құлықты болдырмау үшін және Python-да ерекшеліктерді өңдеу үшін try конструкциясын пайдалану қажет..келесі ресми анықтамасы бар except [8, 15, 26-30]:

```
try:
   инструкции
except [Тип_исключения]:
   инструкции
```

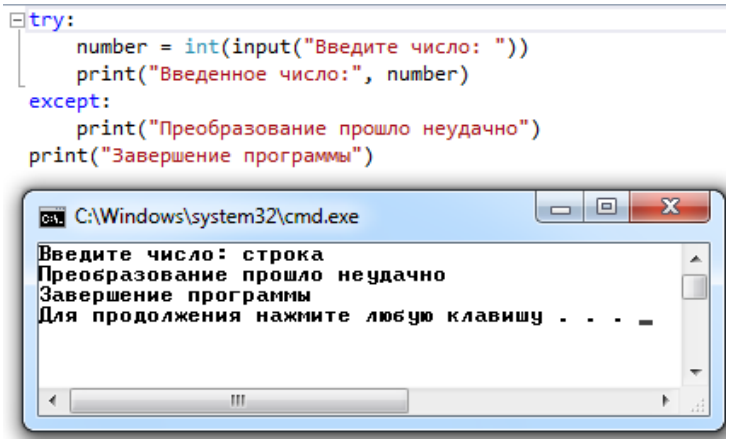
Ықтимал ерекшелік туындауы мүмкін барлық негізгі код try кілт сөзінен кейін орналастырылады. Егер бұл кодта ерекшелік пайда болса, онда try блогындағы код жұмысы үзіледі және орындау except блогына ауысады.

Except кілтсөзінен кейін қандай ерекшелікті (мысалы, ValueError немесе KeyError) өңдеуге болады. Except сөзінен кейін келесі жолда ерекшелік пайда болған кезде орындалатын except блогының нұсқаулары бар.

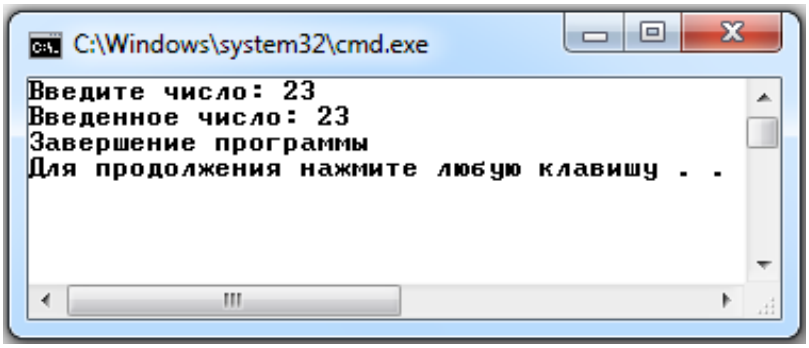
Жолды санға түрлендіру мысалында ерекшелікті өңдеуді қарастырайық (суреттер 1.26-1.27) [8, 15-17, 26].

Консоль шығысынан көрініп тұрғандай, жолды енгізгенде, сандарды консольге шығару болмайды, ал бағдарламаны орындау except блогына өтеді.

Енді барлығы қалыпты орындалады, ерекшелік болмайды және тиісінше except блогы орындалмайды.



Сурет 1.26. Ерекшелікті өңдеу (сәтсіз түрлендіру)

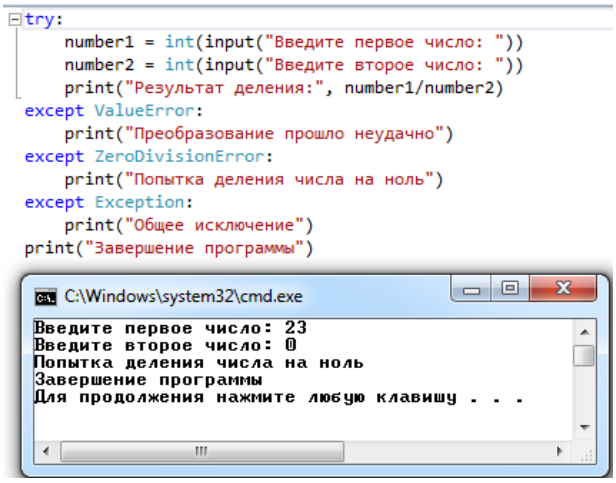


Сурет 1.27. Ерекшелікті өңдеу (дұрыс түрлендіру)

Мысалда, кодта туындауы мүмкін барлық ерекшеліктер бірден өңделді. Дегенмен, біз except сөзінен кейін оны көрсете отырып, өңделетін ерекшелік түрін нақтылай аламыз [1, 8, 15-17, 26]:

```
try:
    number = int(input("Санды енгізіңіз: "))
    print("Енгізілген сан:", number)
except ValueError:
    print("Түрлендіру сәтсіз өтті ")
print("Бағдарламаны аяқтау ")
```

Егер жағдай бағдарламада әртүрлі ерекшеліктердің түрлері түзілуі мүмкін болса, онда біз except қосымша өрнектерін пайдалана отырып, оларды бөлек өңдей аламыз [1, 8, 15-17, 26]:



Сурет 1.28. Әртүрлі ерекшелік түрлерін өңдеу

Егер жолды санға түрлендіру нәтижесінде ерекшелік пайда болса, онда ол except ValueError блогымен өңделеді. Егер екінші сан нөлге тең болса, онда нөлге бөлу болады, онда zerodivisionerror ерекшелік пайда болады және ол except ZeroDivisionError блогымен өңделеді.

Exception түрі барлық ерекше жағдайлар түсетін жалпы ерекшелік болып табылады. Сондықтан бұл жағдайда ValueError немесе ZeroDivisionError түріндегі кез келген ерекшелік except Exception блогында өңделеді:.

1.8.2 Finally блогы

Ерекшеліктерді өңдеу кезінде қосымша емес finally блогын пайдалануға болады. Бұл блоктың ерекше ерекшелігі-бұл ерекшелік пайда болды ма [1, 8, 15-17, 26]:

```
try:
    number = int(input("Санды енгізіңіз: "))
    print("Енгізілген сан:", number)
except ValueError:
    print("Санды түрлендіру мүмкін емес ")
finally:
   print("TRY блогы аяқталды ")
print("Бағдарламаны аяқтау ")
```

Әдетте, finally блогы пайдаланылатын ресурстарды босату үшін қолданылады, мысалы, файлдарды жабу үшін.

1.8.3 Ерекшеліктер генерациясы

Кейде кез келген ерекшелікті қолмен түзету қажет. Ол үшін raise операторы қолданылады [1, 8, 15-17, 26]:

```
try:
    number1 = int(input("Бірінші санды енгізіңіз: "))
    number2 = int(input("Екінші санды енгізіңіз: "))
    if number2 == 0:
        raise Exception("Екінші сан тең болмауы керек 0")
    print("Екі санды бөлу нәтижесі:", number1/number2)
except ValueError:
    print("Қате деректер енгізілді ")
except Exception as e:
    print(e)
print("Бағдарламаны аяқтау")
```

Кейде кез келген ерекшелікті қолмен түзету керек. Ол үшін raise операторы қолданылады:

```
Бірінші санды енгізіңіз: 1
Екінші санды енгізіңіз: 0
Екінші сан 0 тең болмауы керек
Бағдарламаны аяқтау
```

2 – ТАРАУ. PYTHON ТІЛІНІҢ ДЕРЕКТЕР ҚҰРЫЛЫМЫ

2.1 Тізімдер

2.1.1 Тізімді құру

Python деректер жинағымен жұмыс істеу үшін тізімдер, кортеждер және сөздіктер сияқты кірістірілген түрлерін ұсынады [1, 8, 11-17, 26].

Тізім (list) элементтердің жиынтығын немесе бірізділігін сақтайтын деректер түрін ұсынады. Шаршы жақшаларда ([]) тізімді жасау үшін үтір арқылы оның барлық элементтері (2.1-сурет, 1-жол) көрсетіледі. Көптеген бағдарламалау тілдерінде массив деп аталатын ұқсас деректер құрылымы бар.

Сондай-ақ, тізімді жасау үшін list() конструкторын пайдалануға болады (сурет 2.1, жолдар 4, 6).

Осы тізімнің екі анықтамасы да ұқсас - олар бос тізім құрады.

List конструкторы тізімді жасау үшін басқа тізімді қабылдай алады (сурет 2.1, жол 8) [15, 26].

Тізімнің элементтеріне кіру үшін тізімдегі элементтің нөмірін көрсететін индекстерді пайдалану керек. Индекстер нөлден басталады. Яғни, екінші элемент 1 индексі болады. Элементтерді соңына қарай пайдалану үшін 1 бастап теріс индекстерді пайдалануға болады. Яғни, соңғы элементтің индексі -1, соңғысының алдындағы - -2 және т.б. болады (2.1 сурет, 10-14 жолдар).

Егер бір мәнді бірнеше рет қайталайтын тізімді жасау қажет болса, жұлдызша таңбасын * пайдалануға болады (2.1-сурет, 15-жол).

Тізімдерді жасау бағдарламасын орындау нәтижелері және тізімнің элементтеріне қол жеткізу 2.2-суретте көрсетілген.

```
1  #Работа со списками
2  numbers = [1, 2, 3, 4, 5]
3  print("[Элементы]:", numbers)
4  numbers1 = []
5  print("[]:", numbers1)
6  numbers2 = list()
7  print("list():", numbers2)
8  numbers3 = list(numbers)
9  print("list(numbers):", numbers3)
10 print("numbers[0]:", numbers[0])
11 print("numbers[2]:", numbers[2])
12 print("numbers[-3]: ",numbers[-3])
13 numbers[0] = 125
14 print("numbers[0] = 125: ", numbers[0])
15 numbers4 = [2] * 6
16 print("numbers4 = [2] * 6: ", numbers4)
```

Сурет 2.1. Тізімдерді жасау және тізім элементтеріне кіру

```
C:\Users\user\AppData\Local\...
[Элементы]: [1, 2, 3, 4, 5]
[]: []
list(): []
list(numbers): [1, 2, 3, 4, 5]
numbers[0]: 1
numbers[2]: 3
numbers[-3]:  3
numbers[0] = 125:  125
numbers4 = [2] * 6:  [2, 2, 2, 2, 2, 2]
Press any key to continue . . .
```

Сурет 2.2. Тізімдерді жасау бағдарламасын орындау және тізім элементтеріне кіру нәтижесі

Сонымен қатар, сандардың тізбекті тізімі қажет болса, оны жасау үшін үш формадағы range функциясын пайдалану ыңғайлы [19-20, 25-26]:

range(end): 0-ден end санына дейінгі сандар жиынтығы жасалады;

range(start, end): start санынан end санына дейінгі сандар жиынтығы жасалады;

range(start, end, step): step қадамымен start санынан end санына дейінгі сандар жиынтығы жасалады.

2.3-суретте range функциясы арқылы тізімдерді жасау бағдарламасының листингі келтірілген.

```
1  # Создание списка функцией range
2  numbers = list(range(10))
3  print("numbers = list(range(10)): ", numbers)
4  numbers = list(range(2, 10))
5  print("numbers = list(range(2, 10)): ",numbers)
6  numbers = list(range(10, 2, -2))
7  print("numbers = list(range(10, 2, -2)): ",numbers)
8
```

```
C:\Users\user\AppData\Local\Programs\Python\Python37\...
numbers = list(range(10)):  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers = list(range(2, 10)):  [2, 3, 4, 5, 6, 7, 8, 9]
numbers = list(range(10, 2, -2)):  [10, 8, 6, 4]
Press any key to continue . . .
```

Сурет 2.3. Range функциясы арқылы тізімдерді жасау

Мысалы, тізімнің келесі екі анықтамасы ұқсас болады, бірақ range функциясы арқылы код көлемін қысқартамыз:

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

numbers2 = list(range(1, 10))

Тізімде тек бір типті нысандар болуы міндетті емес. Біз бір уақытта жолдар, сандар, басқа деректер түрлерінің нысандарын бір тізімге қоя аламыз:

objects = [1, 2.6, "Hello", True]

2.1.2 Элементтерді іріктеу және тізімдерді салыстыру

Элементтерді таңдау үшін for циклі мен while циклі пайдалануға болады.

For циклі арқылы аралықтар (сурет 2.4, жолдар 4-5) [15-17, 21-26].

Мұнда range функциясының орнына бар companies тізімін бірден қоя аламыз.

While циклының көмегімен аралықтар (сурет 2.4, жолдар 8-11).

```
1  # Перебор элементов списка
2  companies = ["Microsoft", "Google", "Oracle", "Apple"]
3  print("For:")
4  for item in companies:
5      print(item)
6
7  print("\nWhile:")
8  i = 0
9  while i < len(companies):
10     print(companies[i])
11     i += 1
```

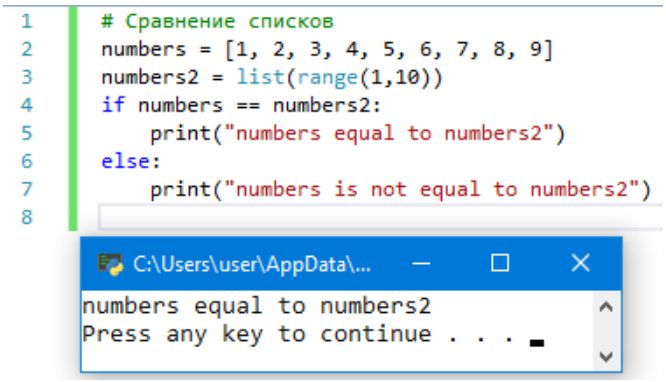
Сурет 2.4. Тізім элементтерін таңдау

```
C:\Users\...
For:
Microsoft
Google
Oracle
Apple

While:
Microsoft
Google
Oracle
Apple
Press any key to continue
```

Сурет 2.5. Тізім элементтерін таңдау (бағдарламаның нәтижесі)

Len() функциясы арқылы таңдау үшін тізімнің ұзындығын аламыз. Санауыштың көмегімен I санауыштың мәні тізімнің ұзындығына тең болмайынша элемент бойынша шығарады.

Екі тізім бірдей элементтердің жиынтығы болса, тең деп саналады (сурет 2.6) [26].



Сурет 2.6. Тізімдерді салыстыру

Бұл жағдайда екі тізім тең болады.

2.1.3 Тізімдермен жұмыс істеу әдістері мен функциялары

Элементтерді басқару үшін тізімдер бірқатар әдістерге ие. Олардың кейбіреулері 2.1-кестеде келтірілген [8, 11, 19-20].

2.1 – кесте. Тізімдермен жұмыс істеудің негізгі әдістері

Әдіс атауы	**Әдісті сипаттау**
append(item)	Тізімнің соңына item элементін қосады
insert(index, item)	Item элементін index индексі бойынша тізімге қосады
remove(item)	Item элементін жояды. Тек элементтің Бірінші кіруі жойылады. Егер элемент табылмаса, ValueError ерекшелігін жасайды
clear()	Тізімнен барлық элементтерді жояды
index(item)	Item элементінің индексін қайтарады. Егер элемент табылмаса, ValueError ерекшелігін жасайды
pop([index])	Index индексі бойынша элементті жояды және қайтарады. Индекс берілмесе, соңғы элементті жояды
count(item)	Item элементінің тізімдегі сәйкессіздік санын қайтарады
sort([key])	Элементтерді сұрыптайды. Әдепкі бойынша өсу бойынша сұрыптайды. Бірақ key параметрімен біз сұрыптау мүмкіндігін бере аламыз
reverse()	Тізімдегі барлық элементтерді кері тәртіппен орналастырады

Сонымен қатар, Python тізімдермен жұмыс істеу үшін бірқатар кіріктірілген функцияларды ұсынады [15-17, 21-26]:

len(list): тізім ұзындығын қайтарады;
sorted(list, [key]): сұрыпталған тізімді қайтарады;
min(list): ең аз тізім элементін қайтарады;
max(list): тізімнің ең үлкен элементін қайтарады.

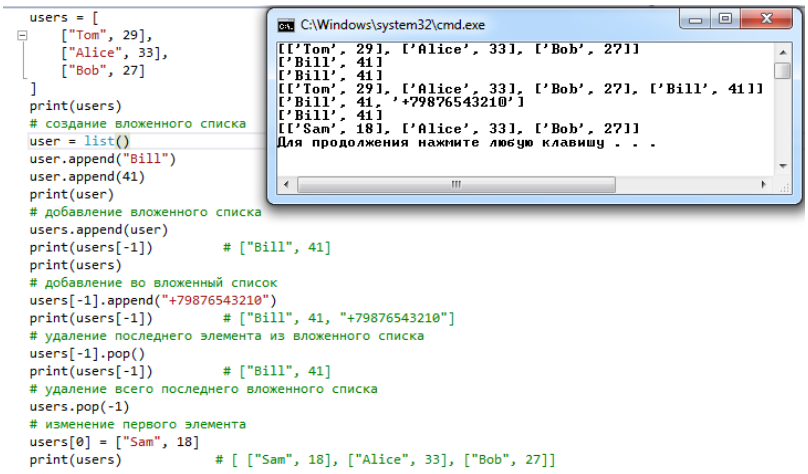
2.2 Салынған тізімдер

2.2.1 Салынған тізімдерді жасау және сипаттау

Тізімдер жолдар, сандар түріндегі стандартты деректерден басқа, сондай-ақ басқа тізімдер болуы мүмкін. Мұндай тізімдерді ішкі тізімдер жолдар рөлін атқаратын кестелермен байланыстыруға болады (сурет 2.7).

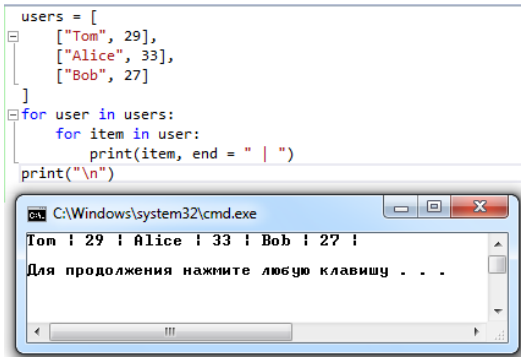
Ішкі тізім элементіне қарап шығу үшін, users[0][1] – бірінші ішкі тізімнің екінші элементіне хабарласу.

Жалпы тізімді, сондай-ақ ішкі тізімдерді қосу, жою және өзгерту әдеттегі (бір өлшемді) тізімдермен жасалатын сияқты (сурет 2.7) [26-30].



Сурет 2.7. Листинг және ішкі тізімдердегі элементтерді қосу, жою және өзгерту әдістерімен бағдарламаны орындау нәтижесі

Салынған тізімдерді іріктеу (сурет 2.8).



Сурет 2.8. Салынған тізімдерді іріктеу

2.2.2 Салынған тізімдерді енгізу және шығару

Python матрицалармен жұмыс істеу үшін тізімдер де қолданылады. Әрбір элемент-матрицалар ішкі тізімді қамтиды [8, 26].

Осылайша, тіркелген тізімдердің құрылымы алынады, олардың саны матрица бағандарының санын анықтайды, ал әрбір салынған тізімнің ішіндегі элементтер саны бастапқы матрицадағы жолдар санын көрсетеді.

4 × 3 өлшем матрицасының мысалын қарастырайық:

```
matrix = [[-1, 0, 1],
      [-1, 0, 1],
      [0, 1, -1],
      [1, 1, -1]]
```

Бұл оператор бір жолға жазуға болады:

```
matrix = [[-1, 0, 1], [-1, 0, 1], [0, 1, -1], [1, 1, -1]]
```

Матрицаны шығаруды бір оператор жүзеге асыруға болады, бірақ мұндай қарапайым әдіс элементтерді алдын ала өңдеуге мүмкіндік бермейді:

```
print(matrix)
```

Матрицаны кесте түрінде шығару үшін осы үшін арнайы дайындалған функцияны қолдануға болады [17, 26]:

1-тәсіл:

```
def printMatrix(a):
    for i in range(len(a)):
        for j in range(len(a[i])):
            print("{:4d}".format(a[i][j]), end = "")
        print()
```

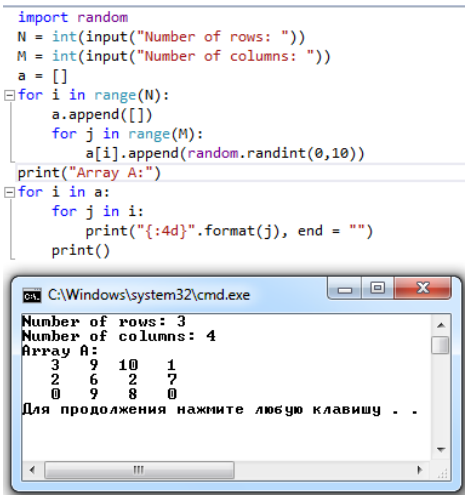
I *мысалында-жол нөмірі, ал* j-*баған нөмірі;* len(a) – *матрицадағы жолдар саны.*

2-тәсіл:

```
def printMatrix(a):
    for i in a:
        for j in i:
            print("{:4d}".format(j), end = "" )
        print()
```

Сыртқы цикл матрица (i) *жолдары бойынша өтеді, ал ішкі цикл әрбір жолдың элементтері* (j) *бойынша өтеді.*

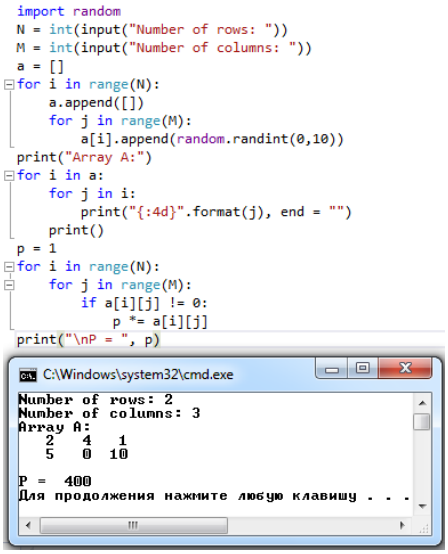
Кездейсоқ сандармен матрица элементтерін инициализациялау үшін 2.9-суретте көрсетілген алгоритм қолданылады [26].



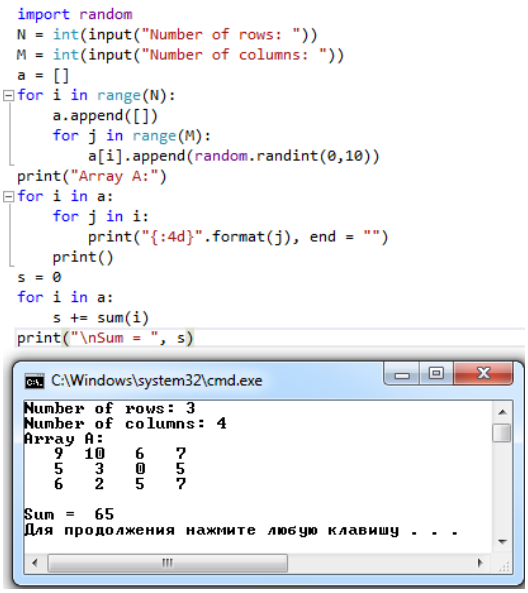
Сурет 2.9. Кездейсоқ сандармен массивті инициализациялау және матрица түрінде кестені шығару

2.2.3 Екі өлшемді массив элементтерін өңдеу

Екі өлшемді массив элементтерінің туындысы 2.10-суретте, ал 2.11-суретте массив элементтерінің қосындысы берілген [21-26].



Сурет 2.10. Массив элементтерінің жұмысын табу



Сурет 2.11. Массив элементтерінің қосындысын табу

2.3 Жиындар

2.3.1 Жиын анықтамасы

Көптеген (set) элементтер жиынтығының тағы бір түрін ұсынады. Жиындарды анықтау үшін элементтер тізілетін фигуралық жақшалар қолданылады [8, 15, 26]:

```
users = {"Tom","Bob","Alice", "Tom"}
print(users)
```

Print функциясы "Tom" элементін бір рет шығарғанына қарамастан, бұл элементтің жиыны екі рет болса да ескеріңіз. Барлығы тек бірегей мәндерді қамтиды.

Сондай-ақ, жиындарды анықтау үшін тізім немесе элементтер кортежі берілетін set() функциясы қолданылуы мүмкін:

```
users3 = set(["Mike", "Bill", "Ted"])
```

Set функциясын бос жиын жасау үшін қолдануға ыңғайлы:

```
users = set()
```

Жиын ұзындығын алу үшін кірістірілген len() функциясы қолданылады:

```
users = {"Tom","Bob","Alice"}
print(len(users))
```

Листинг және жиын құру бағдарламасын орындау нәтижелері 2.12-суретте келтірілген.

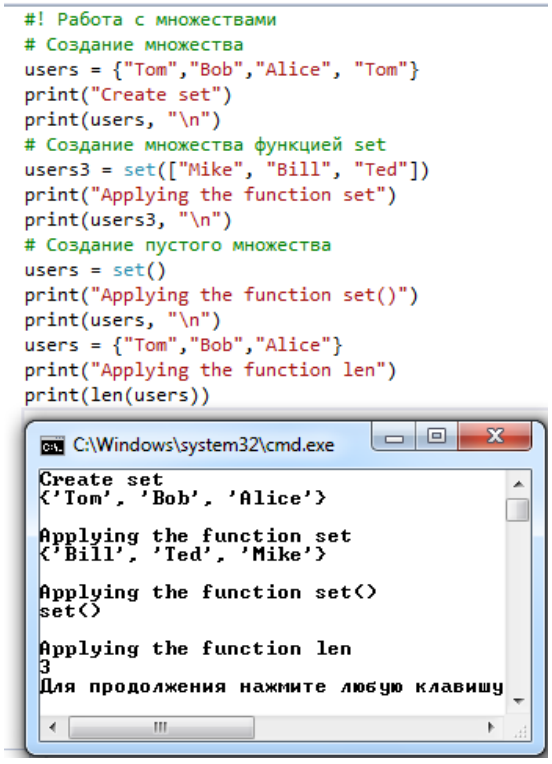
2.3.2 Элементтерді қосу, жою және іріктеу

Бір элементті қосу үшін add() әдісі пайда болады, ал бір элементті жою үшін жойылатын элемент берілетін remove() әдісі пайда болады. Листинг және әдістерді қолдану нәтижесі 2.13-суретте келтірілген [17-20, 25-26].

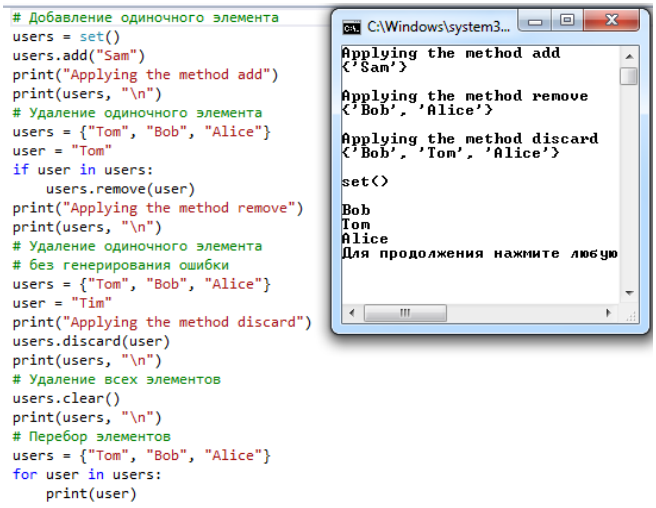
Бірақ, егер мұндай элемент жиынында болмаса, қате пайда болады. Сондықтан жою алдында элементтің болуын in (сурет 2.13) операторының көмегімен тексеру керек.

Сондай-ақ, жою үшін элемент болмаған кезде (сурет 2.13) ерекшеліктерді тудырмайтын discard() әдісін пайдалануға болады.

Барлық элементтерді жою үшін clear() (сурет 2.13) әдісі шақырылады.



Сурет 2.12. Жиын құру



Сурет 2.13. Элементтерді қосу және жою

Элементтерді таңдау үшін for (сурет 2.13) циклын қолдануға болады.
Аралықтар кезінде әрбір элемент user айнымалысына орналастырылады.

2.3.3 Жиындармен операциялар

Copy() әдісі арқылы бір жиын мазмұнын басқа айнымалыға көшіруге болады ю (сурет 2.14).

Copy() әдісі арқылы бір жиын мазмұнын басқа айнымалыға көшіруге болады (сурет 2.14) [26].

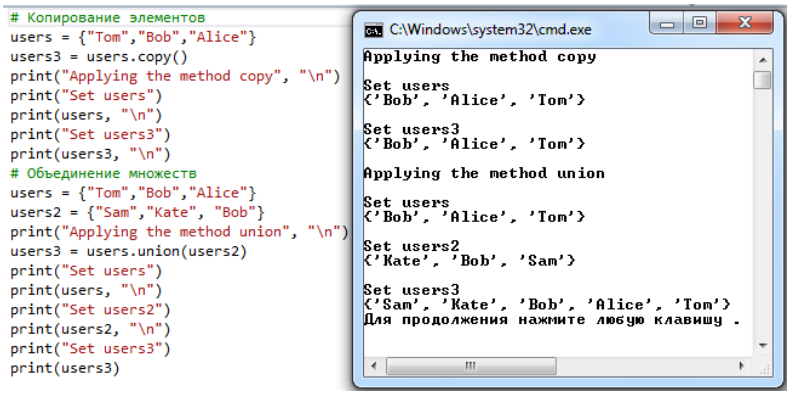
Union() әдісі екі жиындарды біріктіреді және жаңа жиындарды қайтарады (сурет 2.14).

Жиындардың қиылысуы екі жиында бір уақытта бар элементтерді ғана алуға мүмкіндік береді. Intersection() әдісі жиындардың қиылысу әрекетін жасайды және жаңа жиындарды қайтарады (сурет 2.15).

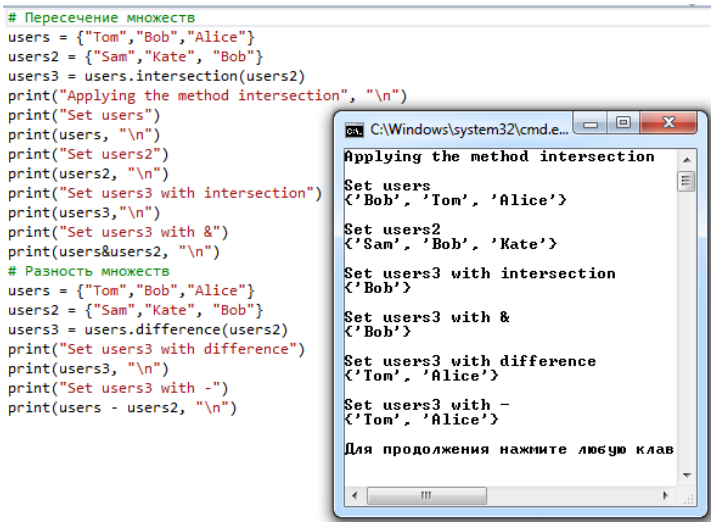
Intersection әдісінің орнына біз логикалық көбейту амалын пайдалана аламыз (сурет 2.15).

Бұл жағдайда біз сол нәтижеге қол жеткізер едік.

Тағы бір операция – жиын айырмашылығы бірінші жиында бар, бірақ екіншісінде жоқ элементтерді қайтарады. Әртүрлі жиындарды алу үшін difference әдісін немесе азайту әрекетін қолдануға болады (сурет 2.15).



Сурет 2.14. Жиындарды көшіру және біріктіру

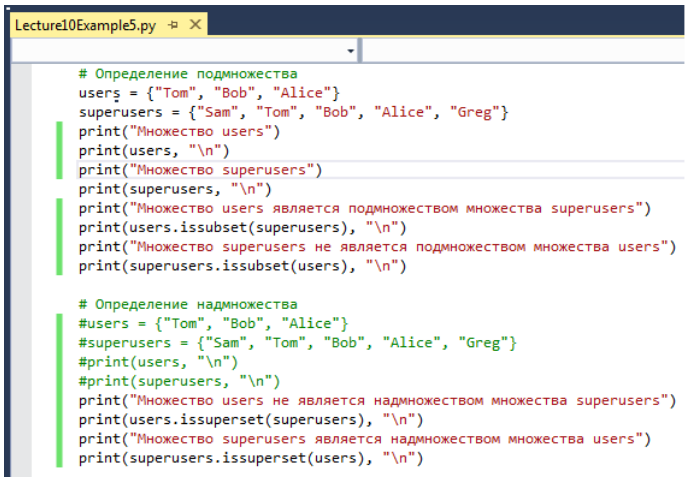


Сурет 2.15. Жиындардың қиылысуы мен айырмасы

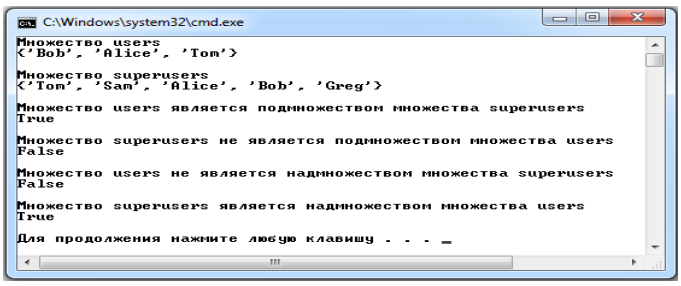
2.3.4 Жиындар арасындағы қатынастар

Issubset әдісі ағымдағы жиын (яғни бөлігі) басқа жиын екенін анықтайды (2.16-2.17-суреттер) [15, 26].

Issuperset әдісі, керісінше, егер ағымдағы жиын болса, басқа жиын үшін True қайтарады (2.16-2.17-суреттер).



Сурет 2.16. Жиын және жиын анықтамасы



Сурет 2.17. Жиын және жиындарды анықтау бағдарламасын орындау нәтижесі

2.4 Сөздіктер

2.4.1 Сөздік анықтамасы

Python тізімдері мен кортеждерімен қатар, сөздік (dictionary) деп аталатын тағы бір кірістірілген деректер құрылымы бар. Бағдарламалау тілдерінің қатарында ұқсас құрылымдар бар (C# сөздігі, PHP ассоциативті массиві).

Тізім сияқты, сөздік элементтер жинағын сақтайды. Сөздікте әрбір элементтің кейбір мәні бар бірегей кілті бар.

Сөздіктің анықтамасы келесі синтаксиске ие [1, 8, 17-20]:

dictionary = {ключ1:значение1, ключ2:значение2, ...}

Бірнеше сөздіктерді анықтайық:

users = {1: "Tom", 2: "Bob", 3: "Bill"}

elements = {"Au": "Золото", "Fe": "Железо", "H": "Водород", "O": "Кислород"}

Users сөздігінде кілттер ретінде сандар, ал мән ретінде жолдар қолданылады. Element сөздігінде кілттер ретінде жолдар пайдаланылады.

Бірақ кілттер мен жолдар бірдей болуы керек. Олар әртүрлі түрлерін ұсына алады:

objects = {1: "Tom", "2": True, 3: 100.6}

Біз сондай-ақ элементтерсіз пустой сөздігін анықтай аламыз [1, 8, 17-20]:

objects = {}

немесе:

objects = dict()

2.4.2 Элементтерді қосу және жою

Сөздік элементін қосу үшін жаңа кілт пен мәнді көрсету керек (2.18-2.19-суреттер) [1, 8, 17-20]:

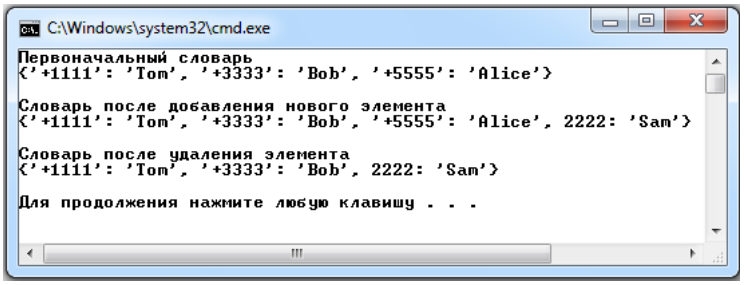
dictionary[новый_ключ] = новое значение

Кілтті жою үшін del операторы қолданылады (2.18-2.19-суреттер) [8, 19-20, 26-30]:

del dictionary[ключ]

```
users = {
    "+1111": "Tom",
    "+3333": "Bob",
    "+5555": "Alice"
}
print("Первоначальный словарь")
print(users, "\n")

print("Словарь после добавления нового элемента")
users[+2222] = "Sam"
print(users, "\n")

print("Словарь после удаления элемента")
del users["+5555"]
print(users, "\n")
```

Сурет 2.18. Сөздіктен элементті қосу және жою



Сурет 2.19. Қосу және пайдалану нәтижесі
сөздіктен элементті жою

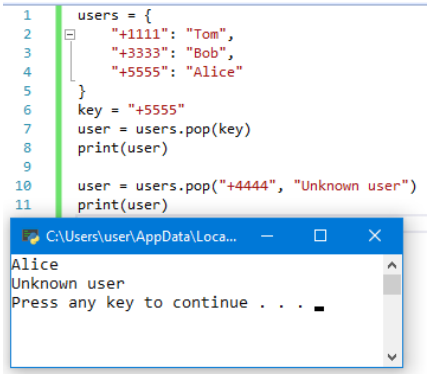
Бірақ, егер мұндай кілт сөздікте болмаса, онда KeyError-дің ерекшелігі жойылады. Сондықтан жою алдында осы кілтпен элементтің болуын тексеру қажет.

```
key = "+5555"
if key in users:
    user = users[key]
  del users[key]
    print(user, "удален")
else:
    print("Элемент не найден")
```

Жоюдың басқа тәсілі pop() әдісін ұсынады. Ол екі нысаны бар (сурет 2.20):

pop(key): key элементін жояды және жойылған элементті қайтарады. Егер бұл кілтпен элемент жоқ болса, KeyError;

pop(key, default): key кілтінің элементін жояды және жойылған элементті қайтарады. Егер бұл кілтпен элемент болмаса, default мәнін қайтарады.



Сурет 2.20. Pop арқылы сөздіктен элементті жою

Егер барлық элементтерді жойғыңыз келсе, clear() әдісін пайдалануға болады:

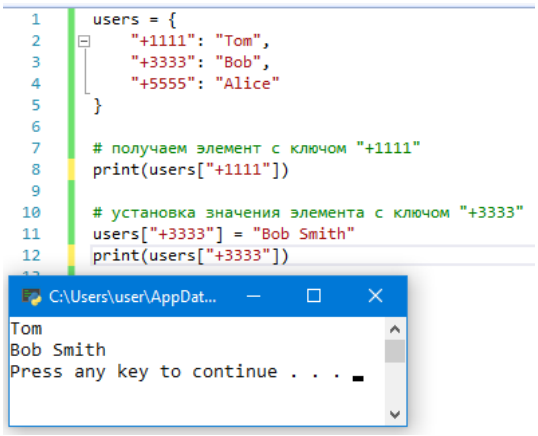
```
users.clear()
```

2.4.3 Элементтерді алу және өзгерту

Сөздік элементтеріне кіру үшін кілт қажет [1, 8, 17-20]:

dictionary[ключ]

Мысалы, сөздікте элементтерді алып, өзгертеміз (сурет 2.21).



Сурет 2.21. Сөздікте элементтерді алу және өзгерту

Егер сөздікте осы кілтпен элементтің мәнін орнатқанда, онда оны қосу болады:

users["+4444"] = "Sam"

Бірақ сөздікте жоқ кілтпен мәнді алуға тырысамыз, онда Python KeyError қатесін жасайды:

user = users["+4444"] # KeyError

Және бұл жағдайды ескерту үшін элементке Жүгінер алдында сөздікте кілттің болуын in сөздік кілті арқылы тексере аламыз. Егер кілт сөздікте болса, онда бұл өрнек True қайтарады [1, 8, 17-20]:

```
key = "+4444"
if key in users:
    user = users[key]
    print(user)
else:
    print("Элемент не найден")
```

Сондай-ақ, элементтерді алу үшін екі нысаны бар get әдісін қолдануға болады:

get(key): сөздіктен key кілті бар элементті қайтарады. Егер бұл кілтпен элемент болмаса, None мәнін қайтарады;

get (key, default): сөздіктен key кілті бар элементті қайтарады. Егер бұл кілтпен элемент болмаса, әдепкі параметр мәнін қайтарады.

```
key = "+5555"
user = users.get(key)
user = users.get(key, "Unknown user")
```

2.4.4 Сөздіктерді көшіру және біріктіру

Copy() әдісі сөздіктің мазмұнын көшіреді, жаңа сөздікті қайтару [1, 8, 17-20]. Update() әдісі екі сөздікті біріктіреді (сурет 2.22).

```
users = {"+1111": "Tom","+3333": "Bob","+5555": "Alice"}
users2 = users.copy()

users = {"+1111": "Tom","+3333": "Bob","+5555": "Alice"}
users2 = {"+2222": "Sam","+6666": "Kate"}
users.update(users2)

print(users)
print(users2)
```

```
C:\Users\user\AppData\Local\Programs\Python\Python37\python.exe
{'+1111': 'Tom', '+3333': 'Bob', '+5555': 'Alice', '+2222': 'Sam', '+6666': 'Kate'}
{'+2222': 'Sam', '+6666': 'Kate'}
Press any key to continue . . .
```

Сурет 2.22. Сөздіктерді көшіру және біріктіру

Users2 сөздігі өзгеріссіз қалады. Басқа сөздіктің элементтері қосылатын users сөздігі өзгереді. Бірақ егер бастапқы сөздіктің екеуі де өзгеріссіз, ал біріктіру нәтижесі үшінші сөздіктің болуы қажет болса, онда бір сөздікті екінші сөздікке көшіруге болады:

users3 = users.copy()
users3.update(users2)

2.4.5 Сөздік аралығы

Сөздікті таңдау үшін for циклын қолдануға болады (2.23-2.24 суреттер, 7-8 жолдар).

Элементтерді ауыстырғанда, біз ағымдағы элементтің кілтін аламыз және ол арқылы элементтің өзі ала аламыз.

Элементтерді іріктеудің басқа тәсілі items() (2.23-2.24-суреттер, 11-12-жолдар) әдісін қолдануды ұсынады.

Items() әдісі кортеждер жиынтығын қайтарады. Әрбір кортеж key және value айнымалыларын бірден алуға болатын кілт пен элемент мәнін қамтиды.

Сондай-ақ, кілттерді таңдау және мәндерді таңдау бөлек мүмкіндіктері бар. Кілттерді таңдау үшін біз сөздікте keys() әдісін шақыра аламыз (2.23-2.24 суреттер, 15-16 жолдар).

Рас, бұл таңдау тәсілі мағынасы жоқ, себебі keys() әдісін шақырусыз біз жоғарыда көрсетілгендей кілттерді таңдай аламыз.

Тек мәндерді таңдау үшін сөздікте values() әдісі шақыруы мүмкін (2.23-2.24 суреттер, 19-20 жолдар).

```
users = {
    "+1111": "Tom",
    "+3333": "Bob",
    "+5555": "Alice"
}
print("Users: ")
for key in users:
    print(key, " - ", users[key])

print("\nItems: ")
for key, value in users.items():
    print(key, " - ", value)

print("\nKeys:")
for key in users.keys():
    print(key)

print("\nValues:")
for value in users.values():
    print(value)
```

Сурет 2.23. Сөздікті әртүрлі әдістермен араластыру

```
C:\Users...
Users:
+1111  -  Tom
+3333  -  Bob
+5555  -  Alice

Items:
+1111  -  Tom
+3333  -  Bob
+5555  -  Alice

Keys:
+1111
+3333
+5555

Values:
Tom
Bob
Alice
Press any key to continue
 . . .
```

Сурет 2.24. Сөздікке арналған бағдарламаны орындау нәтижесі

2.5 Кортеждер

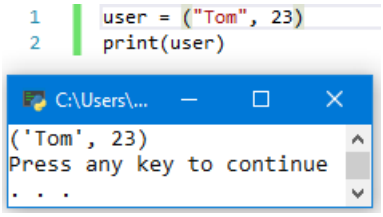
2.5.1 Кортежді анықтау

Кортеж (tuple) кортеж өзгермейтін (immutable) түрін қоспағанда, тізімге ұқсас элементтер тізбегі болып табылады. Сондықтан кортежде элементтерді қосу немесе жою, оны өзгерту мүмкін емес.

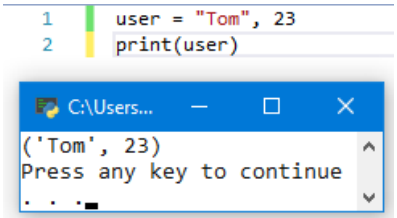
Кортеж сақталған деректерді әдейі емес өзгерістерден қорғайды және сөздіктерде кілт ретінде пайдаланылуы мүмкін. Кортеж жасау үшін, үтірмен бөлінген (сурет 2.25) мәндерін қамтитын дөңгелек жақшалар қолданылады.

Сондай-ақ, кортежді анықтау үшін, біз жай ғана жақшаларды қолданбай үтір арқылы (сурет 2.26) мәндерді айта аламыз.

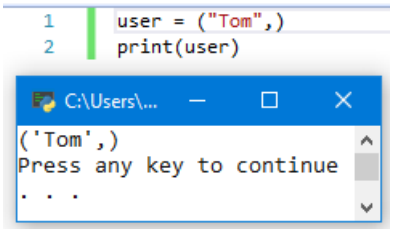
Егер кенеттен кортеж бір элементтен тұрса, онда жалғыз кортеж элементінен кейін үтір (сурет 2. 27) [1, 8, 17-20].



Сурет 2.25. Жақша арқылы кортеж жасау

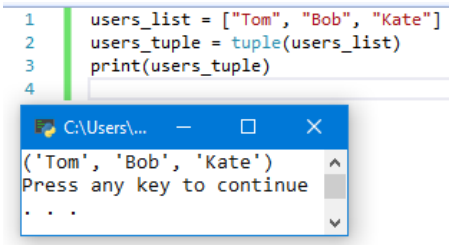


Сурет 2.26. Жақшасыз кортеж жасау



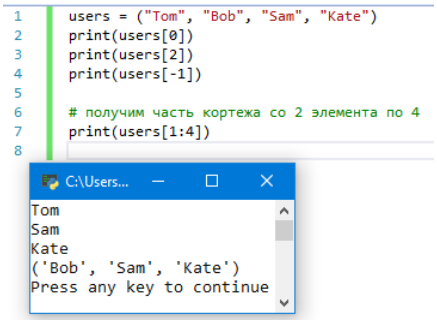
Сурет 2.27. Бір элементтен кортеж жасау

Тізімдегі кортеж жасау үшін, тізімді кортежді қайтаратын tuple() функциясына жіберуге болады (сурет 2.28).



Сурет 2.28. Тізімнен кортеж жасау

Кортежде элементтерді қолдану индекс бойынша тізімдегі сияқты болады. Индекстеу тізімнің басынан элементтерді алған кезде нөлден басталады және тізімнің соңынан элементтерді алған кезде -1 (сурет 2.29).



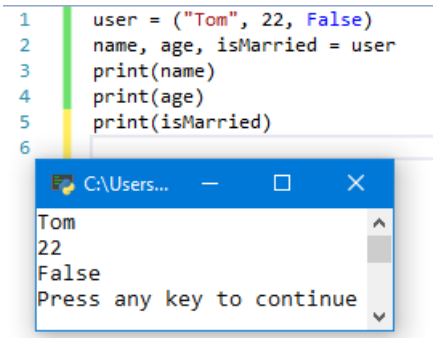
Сурет 2.29. Кортежде элементтерге қарау

Бірақ кортеж – өзгермейтін түрі (immutable) болғандықтан, біз оның элементтерін өзгерте алмаймыз. Яғни келесі жазба жұмыс істемейді:

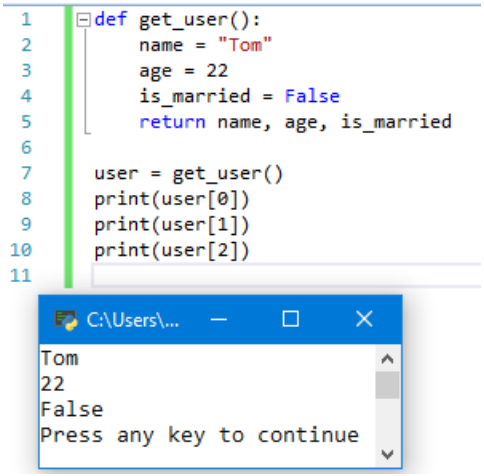
users[1] = "Tim"

Қажет болса, біз жеке айнымалыларға кортежді аша аламыз (сурет 2.30).

Функциядан бірден бірнеше мәндерді қайтару қажет болғанда, кортеждерді пайдалану әсіресе ыңғайлы. Функция бірнеше мәндерді қайтарғанда, шын мәнінде кортежге қайтарады (сурет 2.31).



Сурет 2.30. Кортеждің жекелеген айнымалыларға ыдырауы



Сурет 2.31. Функцияда кортежді пайдалану

Кірістірілген len() функциясы арқылы кортеж ұзындығын алуға болады [1, 8, 17-20]:

user = ("Tom", 22, False)
print(len(user)) # 3

2.5.2 Кортеждерді қайта реттеу

Кортежді таңдау үшін стандартты for және while циклдерін пайдалануға болады (сурет 2.32).

Тізімдегі элементтің бар болуын қалай тексеруге болады (2.33-2.36 суреттер).

```
user = ("Tom", 22, False)
print("For:")
for item in user:
    print(item)

print("\nWhile:")
i = 0
while i < len(user):
    print(user[i])
    i += 1
```

```
C:\Users...
For:
Tom
22
False

While:
Tom
22
False
Press any key to continue
```

Сурет 2.32. Кортеж элементтерін іріктеу

```
user = ("Tom", 22, False)
name = "Tom"
if name in user:
    print("Пользователя зовут Tom")
else:
    print("Пользователь имеет другое имя")
```

Сурет 2.33. Кортежде элементтің болуын тексеру (элемент бар)

```
C:\Users...
Пользователя зовут Tom
Press any key to continue
. . .
```

Сурет 2.34. Кортежде элементтің болуын тексеру бағдарламасының листингі (элемент бар)

```
user = ("Bob", 22, False)
name = "Tom"
if name in user:
    print("Пользователя зовут Tom")
else:
    print("Пользователь имеет другое имя")
```

Сурет 2.35. Кортежде элементтің болуын тексеру (элемент жоқ)

```
C:\Users\user\...
Пользователь имеет другое имя
Press any key to continue . .
```

Сурет 2.36. Кортежде элементтің болуын тексеру бағдарламасының листингі (элемент жоқ)

2.5.3 Күрделі кортеждер

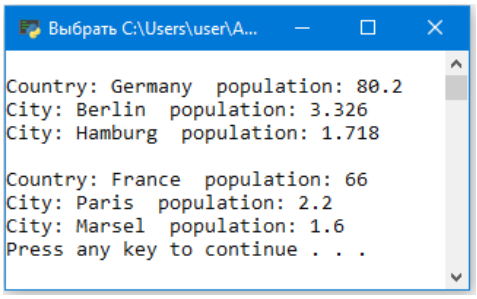
Бір кортежде элементтер түріндегі басқа кортеждер болуы мүмкін (2.37-2.38 суреттер).

Мұнда елді білдіретін кортеж countries, кортеждерден тұрады, олардың әрқайсысы жеке ел [1, 8, 17-20].

Салынған кортеждердің үш элементі бар: елдің атауы, оның халқының және қаланың саны. Әрбір жеке Қала-қаланың атауы мен оның халқының саны бар салынған кортеж.

```
countries = (
    ("Germany", 80.2, (("Berlin",3.326), ("Hamburg", 1.718))),
    ("France", 66, (("Paris", 2.2),("Marsel", 1.6)))
)

for country in countries:
    countryName, countryPopulation, cities = country
    print("\nCountry: {}  population: {}".format(countryName, countryPopulation))
    for city in cities:
        cityName, cityPopulation = city
        print("City: {}  population: {}".format(cityName, cityPopulation))
```

Сурет 2.37. Күрделі кортеждер



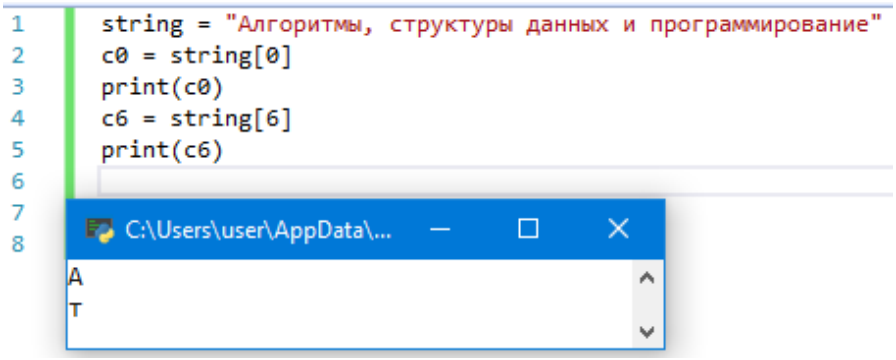
Сурет 2.38. Күрделі кортеждерге арналған бағдарламаны орындау нәтижесі

2.6 Жолдар

2.6.1 Жол таңбаларына кіру

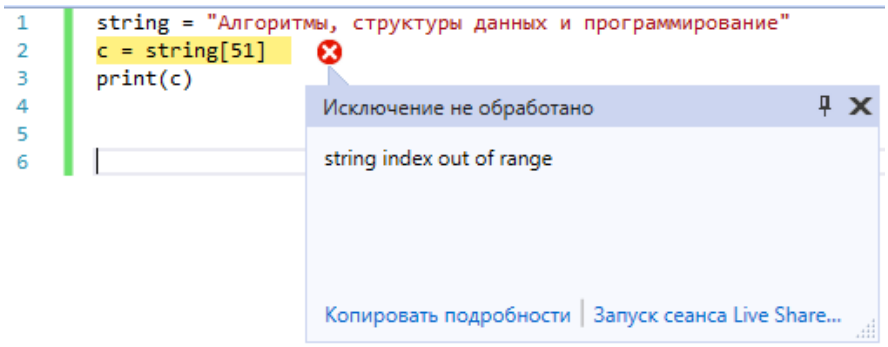
Жол Unicode кодтамасындағы таңбалар ретін білдіреді. Біз шаршы жақшадағы индексі бойынша жолдың жеке символдарына жүгіне аламыз (сурет 2.39).

Индекстеу нөлден басталады, сондықтан жолдың бірінші символы 0 индексі болады [1, 8, 17-20].



Сурет 2.39. Жол таңбаларына кіру

Егер Біз жолда жоқ индекске жүгінсек, онда біз IndexError ерекшелігіне ие боламыз. Мысалы, жоғарыда көрсетілген жағдайда 51 индексі бойынша өтініш қате береді.

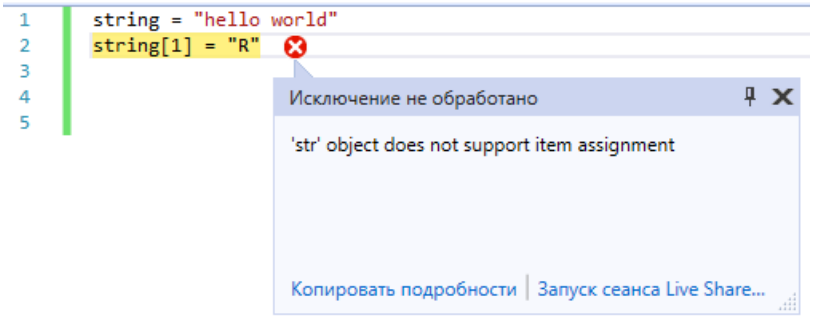


Сурет 2.40. Жол таңбасына оның шегінен тыс кіру

Жолдың соңынан бастап таңбаларға кіру үшін теріс индекстерді пайдалануға болады. Мәселен, индексі -1 соңғы таңба, ал -2 - соңғы таңба және т.б. [1, 8, 17-20]:

```
string = "hello world"
c1 = string[-1]  # d
print(c1)
c5 = string[-5]  # w
print(c5)
```

Таңбалармен жұмыс істегенде, жолдың өзгермейтін (immutable) түрі екенін ескеру керек, сондықтан егер біз жолдың кейбір жеке таңбасын өзгертуге тырыссақ, келесі жағдайда қатені аламыз (сурет 2.41).



Сурет 2.41. Жол таңбасын өзгерту әрекеті

Біз басқа мәнді бере отырып, жол мәнін толық қайта орната аламыз.

2.6.2 Ішкі жолды алу

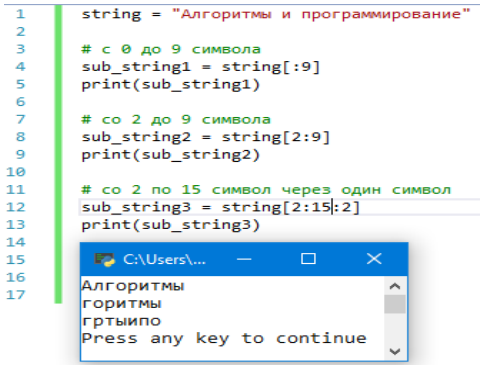
Қажет болған жағдайда біз жолдан тек жеке таңбаларды ғана емес, сондай-ақ қосымша жолды да ала аламыз. Бұл үшін келесі синтаксис қолданылады [1, 8, 17-20]:

string[:end]: end индексі бойынша 0-ші индекстен бастап таңбалар тізбегі алынады.

string[start:end]: end индексі бойынша start индексінен бастап таңбалар тізбегі алынады.

string[start:end:step]: step қадамы арқылы end индексі бойынша start индексінен бастап таңбалар тізбегі алынады.

Барлық опцияларды қолданамыз (сурет 2.42).



Сурет 2.42. Ішкі жолды алу

Ord және len функциялары. Жол Unicode таңбаларын қамтығандықтан, ord() функциясы арқылы біз Unicode кодтамасындағы таңбаның сандық мәнін ала аламыз:

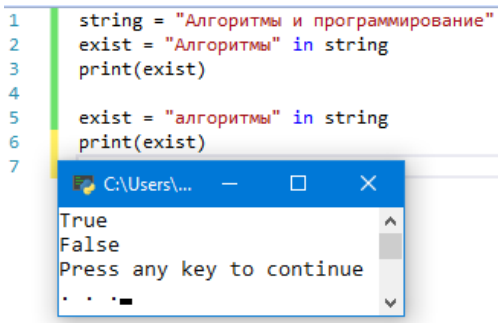
```
print(ord("A"))     # 65
```

Жолдың ұзындығын алу үшін len() функциясын пайдалануға болады:

```
string = "hello world"
length = len(string)
print(length)    # 11
```

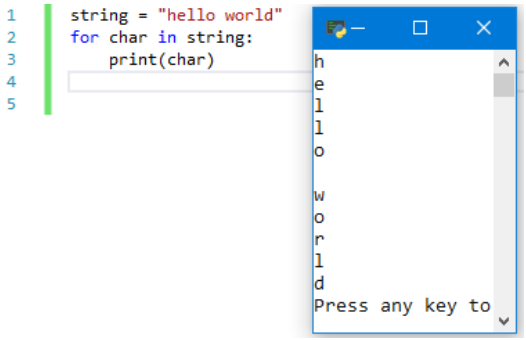
2.6.3 Жолда іздеу және жолды таңдау

Term in string өрнегінің көмегімен string жолында term қосымша жолын табуға болады. Егер қосымша орын табылса, өрнек True мәнін қайтарады, әйтпесе False мәнін қайтарады [1, 8, 17-20]:



Сурет 2.43. Ішкі жолды алу

For циклының көмегімен барлық жол таңбаларын таңдауға болады (сурет 2.44).



Сурет 2.44. Жол таңбаларын таңдау

2.6.4 Жолдың негізгі әдістері

Қосымшаларда қолдануға болатын жолдардың негізгі әдістерін қарастырайық (2.2-кесте).

Мысалы, егер Сан клавиатурасынан енгізуді күтсек, енгізілген жолды санға түрлендіру алдында isnumeric() әдісі арқылы дұрыс Сан енгізілгенін тексеруге болады және егер осылай болса, түрлендіру әрекетін орындауға болады [1, 8, 17-20]:

```
string = input("Введите число: ")
if string.isnumeric():
    number = int(string)
```

print(number)

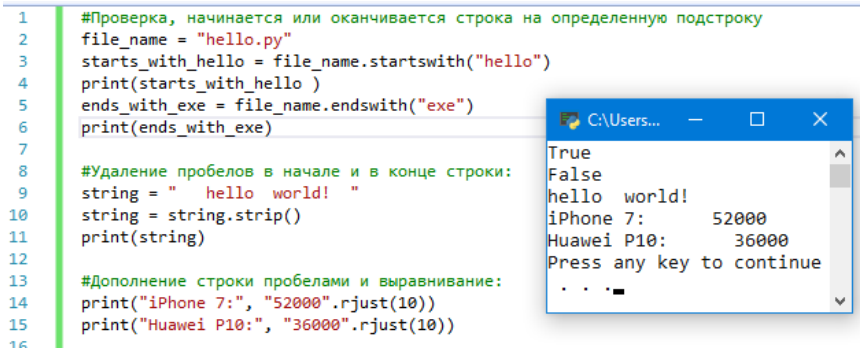
Тексеру белгілі бір қосымша жолға жол басталады немесе аяқталады (2.45 сурет, 2-6 жолдар).

Жолдың басында және соңында бос орындарды жою (2.45 сурет, 9-11 жолдар).

2.2-кесте. Жолдардың негізгі әдістері

Әдіс атауы	**Әдісті тағайындау**
1	**2**
isalpha(str)	Егер жол тек алфавиттік таңбалардан тұрса, True қайтарады
islower(str)	Жол тек төменгі регистрдегі таңбалардан тұратын болса, True қайтарады
isupper(str)	Жоғарғы регистрдегі барлық жол таңбалары болса, True қайтарады
isdigit(str)	Барлық жол таңбалары – сандар болса, True қайтарады
isnumeric(str)	Егер жол Сан болса, True қайтарады
startwith(str)	Жол str ішкі жолынан басталса, True қайтарады
endwith(str)	Жол str ішкі жолында аяқталса, True қайтарады
lower()	Жолды төменгі регистрге ауыстырады
upper()	Жолды жоғарғы регистрге ауыстырады
title()	Жолда барлық сөздердің бастапқы символдары жоғарғы регистрге ауыстырылады
capitalize()	Жоғарғы регистрге бірінші әріпті тек жолдың бірінші сөзін аударады
lstrip()	Жолдан бастапқы бос орындарды жояды
rstrip()	Жолдың соңғы бос жерлерін жояды
strip()	Жолдан бастапқы және соңғы бос орындарды жояды
ljust(width)	Егер жолдың ұзындығы width параметрінен аз болса, жолдың оң жағында Бос орындар қосылады, ал жолдың өзі сол жақ жиегіне тураланады
rjust(width)	Егер жолдың ұзындығы width параметрінен аз болса, жолдың сол жағында Бос орындар қосылады, ал жолдың өзі оң жақ жиегіне тураланады
center(width)	Егер жолдың ұзындығы width параметрінен аз болса, жолдың сол және оң жақтарына Бос орындар тең қосылады, ал жолдың өзі ортаға тураланады
find(str[, start [, end])	Жолда қосымша жол индексін қайтарады. Егер қосымша орын табылмаса, 1 Саны қайтарылады
replace(old, new[, num])	Жолда бір қосымша бағаны екіншісіне ауыстырады
split([delimeter[, num]])	Бөлгішке байланысты жолды кіші бағандарға бөледі
join(strs)	Жолдарды бір жолға біріктіріп, олардың арасында белгілі бір бөлгішті кірістіреді

Жолды бос орындармен толықтыру және туралау (сурет 2.45, 14-15 жолдар).



Сурет 2.45. Бос орындарды жою және толықтыру

2.6.5 Жолда іздеу және ауыстыру

Python жолында қосымша жолды іздеу үшін қосымша жолдың бірінші кіру индексін қайтарады және үш нысаны бар find() әдісі қолданылады [1, 8, 17-20]:

find(str): str қосымша жолын іздеу жолдың басынан оның соңына дейін жүргізіледі.

find(str, start): start параметрі іздеу жүргізілетін бастапқы индексті анықтайды.

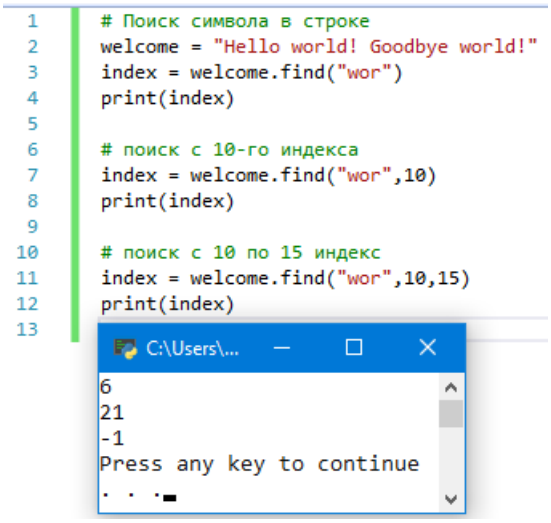
find(str, start, end): end параметрі іздеу жүргізілетін соңғы индексті анықтайды.

Егер қосымша орын табылмаса, әдіс қайтарады -1 (сурет 2.46).

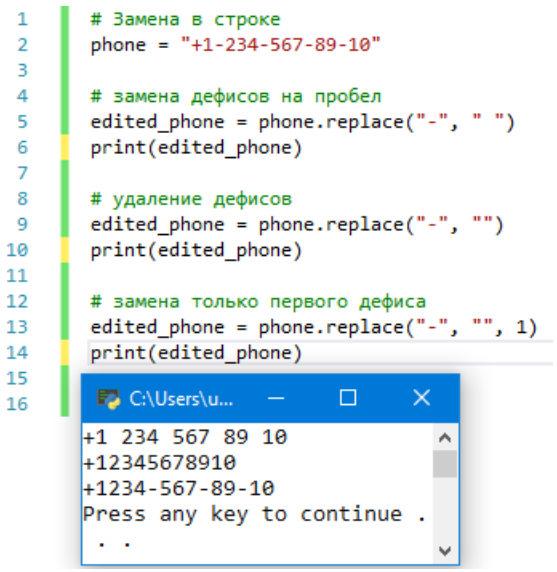
Бір қосалқы жолдың екінші жолына ауыстыру үшін replace() (сурет 2.47) әдісі қолданылады.

replace (old, new): old-ның жаңа қосымша жолағын ауыстырады.

replace(old, new, num): num параметрі old-нің қосымша қондырмасын new-ге ауыстыру қажет екенін көрсетеді.



Сурет 2.46. Жолда іздеу



Сурет 2.47. Жолда ауыстыру

2.6.6 Қосалқы жолға бөлу және жолдарды қосу

Split() әдісі бөлгішке байланысты қосымша жол тізіміне жолды бөледі. Бөлгіш ретінде кез келген таңба немесе таңбалар тізбегі болуы мүмкін. Бұл әдіс мынадай формаларға ие (2.48-2.49 суреттер):

split(): бөлгіш ретінде бос орын пайдаланылады.

split(delimeter): бөлгіш ретінде delimeter қолданылады.

split(delimeter, num): num параметрі delimeter бөлу үшін қанша айырмашылық қолданылатынын көрсетеді. Жолдың қалған бөлігі тізімге қосымша бағандарға бөлінбей қосылады.

```
1  text = "Это был огромный, в два обхвата дуб, с обломанными ветвями и с обломанной корой"
2  # разделение по пробелам
3  splitted_text = text.split()
4  print(splitted_text)
5  print(splitted_text[6])
6
7  # разбиение по запятым
8  splitted_text = text.split(",")
9  print(splitted_text)
10 print(splitted_text[1])
11
12 # разбиение по первым пяти пробелам
13 splitted_text = text.split(" ", 5)
14 print(splitted_text)
15 print(splitted_text[5])
```

Сурет 2.48. Қосымша жолға бөлу (бағдарлама листингі)

```
C:\Users\user\AppData\Local\Programs\Python\Python37\python.exe
['Это', 'был', 'огромный,', 'в', 'два', 'обхвата', 'дуб,', 'с', 'обло
манными', 'ветвями', 'и', 'с', 'обломанной', 'корой']
дуб,

 ['Это был огромный', ' в два обхвата дуб', ' с обломанными ветвями и
 с обломанной корой']
 в два обхвата дуб

 ['Это', 'был', 'огромный,', 'в', 'два', 'обхвата дуб, с обломанными
ветвями и с обломанной корой']
обхвата дуб, с обломанными ветвями и с обломанной корой
Press any key to continue . . .
```

Сурет 2.49. Қосымша жолға бөлу (орындалу нәтижесі)

Қатармен қарапайым операцияларды қарастыру кезінде қатарларды қосу операциясы арқылы қалай біріктіру керектігі көрсетілді. Жолдарды қосу үшін басқа мүмкіндік join() әдісін ұсынады: ол жолдар тізімін біріктіреді. Бұл әдіс туындайтын ағымдағы жол бөлгіш ретінде пайдаланылады (сурет 2.50).

```
1  words = ["Let", "me", "speak", "from", "my", "heart", "in", "English"]
2  # разделитель - пробел
3  sentence = " ".join(words)
4  print(sentence)
5
6  # разделитель - вертикальная черта
7  sentence = " | ".join(words)
8  print(sentence)
9
10 word = "hello"
11 joined_word = "|".join(word)
12 print(joined_word)
13
```

```
C:\Users\user\AppData\Local\Programs\Python\Pytho...
Let me speak from my heart in English
Let | me | speak | from | my | heart | in | English
h|e|l|l|o
Press any key to continue . . .
```

Сурет 2.50. Жолдарды қосу

Тізімнің орнына join әдісіне қарапайым жолды беруге болады, содан кейін бөлгіш осы жолдың символдары арасында кірістіріледі (сурет 2.50, 10-12-жолдар).

2.6.7 Пішімдеу

Жолдардағы анықталған format() әдісі белгілі бір мәндерді плейсхолдерлер орнына енгізе отырып, жолды пішімдеуге мүмкіндік береді [1, 8, 17-20].

Жолға қою үшін фигуралық жақшалармен ({}) жиектелетін арнайы параметрлер қолданылады.

Атаулы параметрлер. Пішімделетін жолда біз параметрлерді анықтай аламыз, format() әдісі осы параметрлер үшін мәндерді бере аламыз (сурет 2.51).

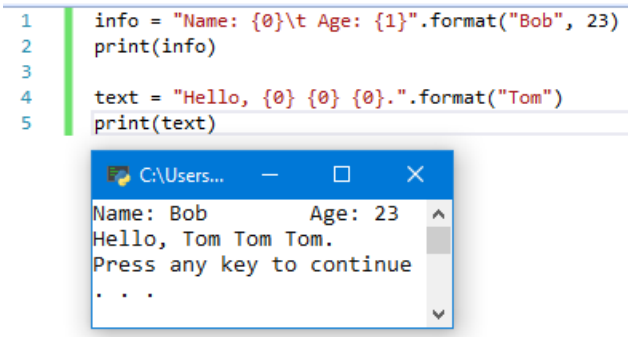
```
1  text = "Hello, {first_name}.".format(first_name="Tom")
2  print(text)
3
4  info = "Name: {name}\t Age: {age}".format(name="Bob", age=23)
5  print(info)
6
7
```

```
C:\Users\...
Hello, Tom.
Name: Bob        Age: 23
Press any key to continue
```

Сурет 2.51. Жолды пішімдеу (атаулы параметрлер)

Сонымен қатар, әдісіне дәлелдер пішімі жол параметрлері сияқты атауымен анықталады. Егер параметр first_name деп аталатын болса, онда мән берілетін дәлел де first_name деп аталады.

Позиция параметрлері. Біз сондай-ақ format әдісіне аргументтер жинағын жүйелі түрде бере аламыз, ал форматталатын жолда фигуралық жақшаларда олардың нөмірін көрсете отырып, осы аргументтерді енгізе аламыз (нөмірлеу нөлден басталады). Бұл ретте аргументтерді жолға бірнеше рет қоюға болады (сурет 2.52).



Сурет 2.52. Жолды пішімдеу (позиция бойынша параметрлер)

Айырбастау. Форматталатын мәндерді жолға берудің тағы бір тәсілі белгілі бір мәндер енгізілетін орындықтарды немесе арнайы плейсхолдерлерді пайдалануды білдіреді. Пішімдеу үшін келесі плейсхолдерлерді пайдалана аламыз [1, 8, 17-20]:

s: жолдарды енгізу үшін.

d: бүтін сандарды енгізу үшін.

f: бөлшек сандарды қою үшін. Бұл түрді нүкте арқылы бөлшек бөліктегі белгілер санын анықтауға болады.

%: 100 мәнін көбейтіп, пайыз белгісін қосады.

e: санды экспоненциалды жазбаға шығарады.

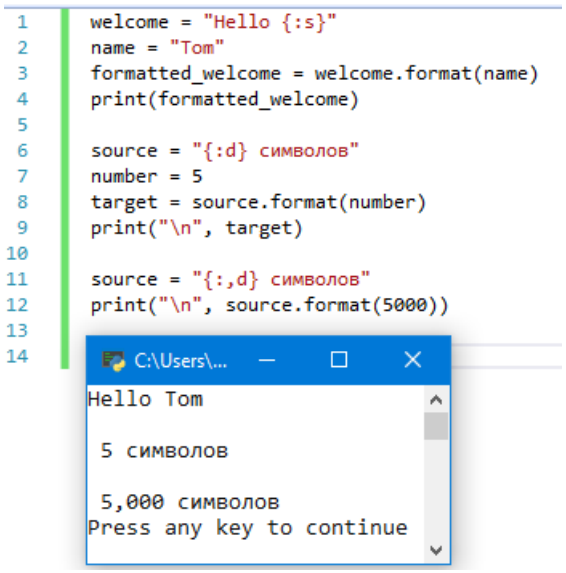
Плейсхолдердің жалпы синтаксисі келесі:

{:плейсхолдер}

Плейсхолдерге байланысты қосымша параметрлерді қосуға болады. Мысалы, float сандарды пішімдеу үшін келесі параметрлерді пайдалануға болады

{:[количество_символов][запятая][.число_знаков_в_дробной_части] плейсхолдер}

Format әдісін шақыру кезінде оған дәлел ретінде плейсхолдерлер орнына салынатын мәндер беріледі (сурет 2.53, 1-4-жолдар).



Сурет 2.53. Жолды пішімдеу

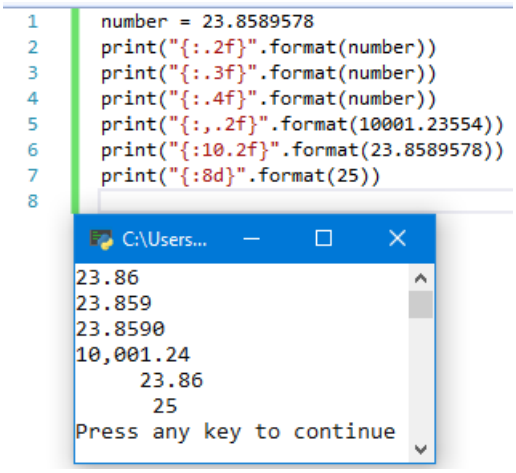
Нәтиже ретінде format() әдісі жаңа пішімделген жолды қайтарады.

Бүтін сандарды пішімдеу (сурет 2.53, 6-9 жолдар).

Егер пішімделетін Сан 999-дан артық болса, онда біз плейсхолдерді анықтауда үтірлерді разряд бөлгіш ретінде пайдаланғымыз келетінін көрсете аламыз (сурет 2.53, 11-12 жолдар).

Бөлшек сандар үшін, яғни float түрін білдіретін сандар үшін, плейсхолдер кодының алдында нүктеден кейін бөлшек бөлікте қанша белгі шығарғымыз келетінін көрсетуге болады (сурет 2.54, 1-5 жолдар).

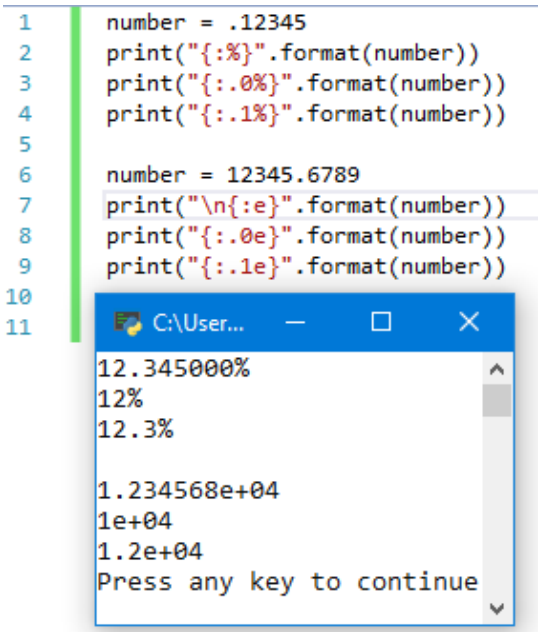
Тағы бір параметр форматталатын мәннің ең аз енін символдарда (сурет 2.54, 6-7 жолдар) орнатуға мүмкіндік береді.



Сурет 2.54. Жолды пішімдеу (қою, бөлшек сандарды шығару)

Пайыздарды шығару үшін "%" кодын пайдалану жақсы (2.55 сурет, 1-4 жолдар).

Сандарды экспоненциалды жазбаға шығару үшін "e" плейсхолдері қолданылады (сурет 2.55, 6-9 жолдар).



Сурет 2.55. Жолды пішімдеу (қою, шығару %)

2.6.8 Format әдісі жоқ пішімдеу

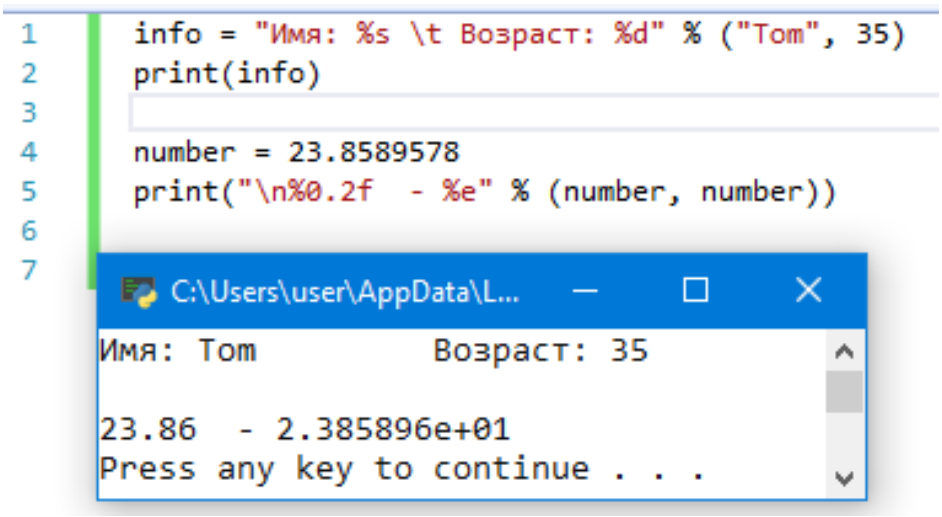
Негізінде, мәндерді пішімдеу үшін келесі синтаксисті қолдана отырып, format әдісі жоқ [1, 8, 17-20]:

строка%(параметр1, параметр2,..параметрN)

Яғни, басында жоғарыда қарастырылған (%плейсхолдерді қоспағанда) сол плейсхолдерлерді қамтитын жол жүреді, жолдан кейін % пайыз белгісі, содан кейін жолға салынатын мәндердің тізімі қойылады. Шын мәнінде пайыз белгісі операцияны ұсынады, нәтижесінде жаңа жол пайда болады (сурет 2.56, 1-2-жолдар).

Плейсхолдердің жанында пайыз белгісі көрсетіледі және format функциясына қарағанда мұнда фигуралық жақшалар қажет емес.

Мұнда сандарды пішімдеу тәсілдері де қолданылады (сурет 2.56, 4-5-жолдар).



Сурет 2.56. Format әдісі жоқ жолды пішімдеу

3 – ТАРАУ. ОБЪЕКТІЛІ-БАҒЫТТАЛҒАН БАҒДАРЛАМАЛАУ

3.1 Объектілі-бағытталған бағдарламалаудың негізгі ұғымдары

Объектілі-бағытталған бағдарламалау (ООП) – бұл бағдарламалау парадигмасы, онда компьютерлік бағдарламаның әр түрлі компоненттері нақты объектілер негізінде модельделеді. Объект-қандай да бір сипаттамалары бар және қандай да бір функцияны орындай алатын нәрсе.

OOP Python артықшылықтары мен кемшіліктері [1, 8, 17-20].

Объектілі-бағытталған бағдарламалаудың бірнеше негізгі артықшылықтарын қарастырайық:

1. Объектілі-бағытталған бағдарламалау қайта пайдалануды білдіреді. Нысандар мен сыныптар түрінде жазылған компьютерлік бағдарлама кодты қайталамай басқа жобаларда қайта пайдаланылуы мүмкін.
2. Объектілі-бағытталған бағдарламалауда модулярлық тәсілді қолдану оқылатын және икемді кодты алуға мүмкіндік береді.
3. Объектілі-бағдарлы бағдарламалауда әр сыныптың нақты міндеті бар. Егер қате кодтың бір бөлігінде болса, кодтың басқа бөлігіне араласудың қажеті жоқ, оны жергілікті түзетуге болады.
4. Деректер инкапсуляциясы (мақалада бұдан әрі қарастырамыз) объектілі-бағытталған тәсілді пайдалана отырып, әзірленетін бағдарламаға қосымша қауіпсіздік деңгейін енгізеді.

Объектілі-бағытталған бағдарламалау бірқатар артықшылықтарға ие болса да, ол белгілі бір кемшіліктерден тұрады, олардың кейбіреулері төменде:

1. Объектілерді құру үшін әзірленетін бағдарламалық қамтамасыз ету туралы егжей-тегжейлі түсінік болу қажет.
2. Емес әрбір аспект бағдарламалық қамтамасыз ету-бұл ең жақсы шешім ретінде іске асыру үшін объект. Жаңадан келгендер үшін алтын ортасында сызық сызып қиын болуы мүмкін.
3. Сіз барлық жаңа және жаңа сыныптарды кодқа енгізгеннен кейін, Бағдарламаның мөлшері мен күрделілігі геометриялық прогрессияда өсуде.

Келесі бөлімде біз объектілі-бағытталған бағдарламалаудың бірқатар маңызды концепцияларын қарастырамыз.

Атаудағыдай, объектілі-бағытталған бағдарламалау-бұл объектілер туралы сөз. Алайда, нысанды құру алдында біз оның класын анықтауымыз керек.

3.1.1 Кластар мен объектілер

Класстар. Объектілі-бағытталған бағдарламалаудағы сынып объект үшін сызба рөлінде болады. Сыныпты үйдің картасы ретінде қарастыруға болады. Сіз қалай көрінеді үй, жай қарап, оның картасына.

Сынып өзі ештеңе білдірмейді. Мысалы, карта үй деп айтуға болмайды, ол тек нақты үй қалай көрінуі керек екенін түсіндіреді.

Сынып пен объект арасындағы қарым-қатынасты машина мен Audi арасындағы қарым-қатынасқа қарап, көрнекі түрде көрсетуге болады. Иә, Audi-бұл машина. Дегенмен, машина сияқты нәрсе жоқ. Машина-бұл Toyota, Honda, Ferrari және басқа да компанияларда жүзеге асырылатын дерексіз тұжырымдама.

Class кілт сөз Python класты жасау үшін қолданылады. Сынып Атауы class кілтінен кейін, одан кейін қос нүкте қажет. Сынып денесі жаңа жолдан басталады, бір қойындыға солға қарай шегініп.

Біз Python ең қарапайым сынып жасай алатынымызды қарастырайық. Келесі кодты қараңыз [1, 8, 17-20]:

```
# Создаем класс Car
class Car:
    # создаем атрибуты класса
    name = "c200"
    make = "mercedez"
    model = 2008

    # создаем методы класса
    def start(self):
        print ("Заводим двигатель")

  def stop(self):
      print ("Отключаем двигатель")
```

Жоғарыда мысалда, біз үш атрибуты бар Car деп аталатын сыныпты құрдық: name, make маркасы және model моделі. Біздің сыныпта екі әдіс бар: start() и stop().

Объектілер. Бұрын біз сынып объектінің сызбасын ұсынатынын түсіндік. Алайда, шын мәнінде сынып нысандары мен әдістерін пайдалану үшін, сіз осы сыныптың нысанын жасау керек. Объектіден тыс пайдалануға болатын бірнеше әдістер мен класс атрибуттары бар, біз оларды келесі бөлімде қарастырамыз.

Қазір әдетті есте сақтаңыз, біз оның әдістері мен атрибуттарын пайдалана бастамас бұрын сынып нысанын жасауымыз керек.

Объект де дана деп аталады. Дегенмен, класс объектісін құру процесі инициализация деп аталады. Python-да класс нысанын жасау үшін, біз сынып атауын енгізіп, Келесі ашылатын және жабылатын жақшалармен.

Алдыңғы бөлімде құрылған Car сынып нысанын құрайық.

```
# Создаем объект класса Car под названием car_a
car_a = Car()

# Создаем объект класса Car под названием car_b
car_b = Car()
```

Біз жасаған нысандардың түрін білу үшін, біз type әдісін қолдана аламыз және оған Нысандар атауларын бере аламыз. Келесі кодты орындаңыз:

```
print(type(car_b))
```

Нәтижесінде көретініміз:

```
<class '__main__.Car'>
```

Бұл car_b-car класы нысанының түрі екенін білдіреді.

Қазіргі уақытта біз сынып пен оған сәйкес нысандарды құрдық. Енді сынып атрибуттарына қол жеткізу және сынып нысанының көмегімен сынып әдісін шақыру уақыты келді. Мұны істеу үшін, Сіз тек нүкте операторы қажет нысанның атын жазу керек . қол жеткізу немесе қоңырау шалу келетін атрибуттың немесе әдістің атауы. Келесі мысал қарастырайық:

```
car_b.start()
```

Бұл скриптте біз car_b нысаны арқылы start() әдісін шақырамыз:

Қозғалтқышты ашамыз

Сол сияқты, келесі синтаксисті пайдалана отырып, атрибутқа қол жеткізе аласыз:

```
print(car_b.model)
```

Төменде көрсетілгендей модель төлсипатының мәнін көресіз:

2008

Сынып атрибуттары. Алдыңғы секцияда сынып нысандарын қалай құру және сынып атрибуттарына кіру үшін осы нысандарды қалай пайдалана аламыз

Python-да, әрбір нысан белгілі бір әдепкі атрибуттарды және белгілі бір пайдаланушы атрибуттарына қосымша әдістерді қамтиды. Нысанның барлық атрибуттары мен әдістерін көру үшін dir() деп аталатын кірістірілген функцияны пайдаланыңыз. Алдыңғы бөлімде жасалған car_b нысанының барлық атрибуттарын қараңыз. Келесі скриптті орындаңыз:

```
print(dir(car_b))
```

Берілісте келесі атрибуттарды көресіз [1, 8, 17-20]:

```
['__class__',
'__delattr__',
'__dict__',
'__dir__',
'__doc__',
'__eq__',
'__format__',
'__ge__',
'__getattribute__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'make',
'model',
'name',
'start',
'stop']
```

Бұл кірістірілген функция элементтің атрибуттары мен функцияларын, әсіресе REPL арқылы пайдаланғанда өте пайдалы.

Дана атрибуттарына қарсы сынып атрибуттары.

Атрибуттар екі түрге көрнекі жатқызылуы мүмкін:

– сынып атрибуттары;
– дана атрибуттары.

Сынып атрибуттары барлық сынып объектілері арасында бөлінеді, ал даналардың атрибуттары дананың меншігі болып табылады.

Есте сақтаңыз, дана-бұл объектінің баламалы атауы.

Дананың атрибуттары кез келген әдістің ішінде жарияланады, ал сынып атрибуттары кез келген әдістен тыс жарияланады.

Келесі мысал осы айырмашылықты анықтайды [1, 8, 17-20]:

```
class Car:

    # создаем атрибуты класса
    car_count = 0

    # создаем методы класса
    def start(self, name, make, model):
        print("Двигатель заведен")
        self.name = name
        self.make = make
        self.model = model
        Car.car_count += 1
```

Жоғарыда көрсетілген скриптте біз car класын car_count деп аталатын класс атрибуты және name, make және model деп аталатын үш дана төлсипаты бар жасайды. Класс біздің үш дана атрибуты бар бір start() әдісін қамтиды. Даналардың атрибуттарының мәндері start() әдісіне дәлел ретінде берілген. Start әдісінің ішінде car_count атрибуты бір-біріне ұлғайтылды.

Бұл әдіс ішінде, дананың атрибуттары self кілт сөзі арқылы сілтеме жасайды, ал сынып атрибуттары сынып атауының көмегімен сілтеме жасайды.

Car сынып нысанын құрайық және start() әдісті шақырайық.

```
car_a = Car()
car_a.start("Corrola", "Toyota", 2015)
print(car_a.name)
print(car_a.car_count)
```

Жоғарыда скриптте біз дананың атрибуты мен car_count класының атрибуты атауын шығардық. Ұстап беру кезінде сіз car_count атрибуты төменде көрсетілгендей 1 мәні болады:

```
Қозғалтқыш ашылды
Corrola
1
```

Енді Car класының тағы бір нысанын жасап, start() әдісін шақырамыз.

```
car_b = Car()
car_b.start("City", "Honda", 2013)
print(car_b.name)
print(car_b.car_count)
```

Қазір сіз car_count төлсипатының мәнін шығарсаңыз, сіз 2 ұстап беру көресіз. Бұл ciscount атрибуты класс атрибуты болып табылады және осылайша ол даналар арасында бөлінеді. Car_a нысаны өз мәнін 1-ге дейін арттырды, ал car_b өз мәнін тағы бір рет арттырды, сондықтан қорытынды мәні 2 тең. Беру келесідей:

```
Қозғалтқыш ашылды
City
2
```

3.1.2 Әдістері

Бұдан бұрын анықталғандай, объектілі-бағытталған бағдарламалауда әдістер объектінің функционалдарын жүзеге асыру үшін пайдаланылады. Алдыңғы бөлімде біз car класы үшін start() және stop() әдістерін құрдық. Осы уақытқа дейін, біз әдістерді шақыру үшін сынып нысандарын пайдаландық. Дегенмен, сынып атауының көмегімен тікелей туындауы мүмкін әдістердің түрі бар. Бұл әдіс статикалық әдіс деп аталады.

Статичные әдістері. Статикалық әдісті жариялау үшін, төмендегі әдіс атауынан бұрын @staticmethod дискрипторын көрсету керек:

```
class Car:

    @staticmethod
    def get_class_details():
    print ("Это класс Car")
```

Жоғары кодта біз бір статикалық get_class_details() әдісімен Car класын құрдық. Сынып атауын пайдалана отырып, осы әдісті шақырайық.

```
Car.get_class_details()
```

Сіз бізге get_class_details() әдісін шақыру үшін Car класының данасын жасаудың қажеті жоқ екенін көре аласыз, оның орнына біз сынып атауын пайдаландық. Статикалық әдістер тек Python класс атрибуттарына ғана қол жеткізе алады, сіз self арқылы әдістерге жүгіне алмайсыз.

Әдістен көпше мәндерді қайтару. Python тілінің ең жақсы ерекшеліктерінің бірі-сынып әдістері көпше мәндерді қайтаруы мүмкін. Келесі мысал қараңыз:

```
class Square:

@staticmethod
def get_squares(a, b):
    return a*a, b*b

print(Square.get_squares(3, 5))
```

Жоғарыда скриптада біз square деп аталатын сынып get_squares() статикалық әдісімен құрдық. Әдіс екі параметрді қабылдайды. Ол әр параметрді көбейтеді және return операторының көмегімен екі нәтижені қайтарады. Жоғарыда көрсетілген скриптті беруде сіз 3 және 5 квадраттарын көресіз [1, 8, 17-20].

Str әдісі. Осы уақытқа дейін біз атрибуттарды print() әдісі арқылы шығардық. Егер біз сынып нысанын шығарсақ, не болатынын көрейік.

Ол үшін біз бір әдіспен қарапайым Car класын жасап, класс объектісін консольге шығаруға тырыстық. Келесі скриптті орындаңыз:

```
class Car:

    # класс әдістерін құру
    def start(self):
        print ("Қозғалтқыш ашылды")

car_a = Car()
print(car_a)
```

Жоғарыда скриптте біз car класындағы car_a нысанын құрдық және оның мәнін экранға шығардық. Шын мәнінде, біз car_a нысанына жол ретінде қараймыз. Беру келесідей:

```
<__main__.Car object at 0x000001CCCF4335C0>
```

Ұстап беру біздің нысан сақталатын жад орнын көрсетеді. Әрбір Python нысаны әдепкі бойынша __ _ str әдісін қамтиды . Нысанды жол ретінде пайдаланғанда, ____ str _ әдісі шақырылады, ол әдепкі бойынша объектінің жады орнын шығарады. Дегенмен, сіз сондай-ақ ___ str әдісінің жеке анықтамасын ұсына аласыз. Мысалы, келесі мысал ретінде:

```
# car класын құру
class Car:
    # класс әдістерін құру
    def __str__(self):
        return "Car class Object"
    def start(self):
        print ("Қозғалтқыш ашылды")
car_a = Car()
print(car_a)
```

Бұл скрипте жоғарыда біз переопределили әдісі __str__ ұсына отырып, біздің өз анықтау әдісі. Енді сіз car_a нысанын шығарсаңыз, консольде "Car class Object" хабарын көресіз. Бұл біздің пайдаланушылық әдісіне __str_ енгізген хабарлама.

Бұл әдісті пайдалану объект шыққан кезде пайдаланушылық және мағыналы сипаттамаларды жасауға мүмкіндік береді. Сіз тіпті сынып ішінде қандай деректерді көрсете аласыз, мысалы car сынып Атауы.

3.1.3 Конструкторлар

Конструктор-класс нысанын жасаған кезде әдепкі бойынша туындайтын арнайы әдіс [1, 8, 17-20].

Конструктор жасау үшін Сіз __init__ негізгі сөзімен әдісті жасау керек. Келесі мысал қараңыз:

```
class Car:
    # класс атрибуттарын құру
    car_count = 0
    # класс әдістерін құру
    def __init__(self):
        Car.car_count +=1
```

```
print(Car.car_count)
```

Скриптте біз car класын car_count класының бір атрибутымен құрдық. Класс құрамында car_count мәнін арттыратын және экранға қорытынды мәнді шығаратын конструктор бар.

Енді Car класы нысаны құрылғанда, конструктор да пайда болады, car_count мәні ұлғаяды және экранда көрсетіледі. Жасаймыз қарапайым объект және көреміз, не шығатынын:

```
car_a = Car()
car_b = Car()
car_c = Car()
```

1, 2 және 3 мәнін көресіз, өйткені әрбір нысан үшін car_count айнымалы мәні ұлғаяды және экранда көрсетіледі.

Атауын қоспағанда, конструктор әдеттегі әдіс ретінде пайдаланылуы мүмкін. Конструктордан мәндерді беруге және алуға болады. Ол әдетте сынып данасын жасау кезінде атрибуттың мәнін инициализациялау қажет болған кезде қолданылады.

Жаһандық өзгерістерге қарсы жергілікті айнымалылар.

Біз Python атрибуттарының екі түрі бар екенін білеміз: дана атрибуттары және сынып атрибуттары.

Сынып атрибуттары да айнымалы деп аталады. Көріну аймағына байланысты, айнымалылар екі түрге де қатысты болуы мүмкін: жергілікті айнымалылар және жаһандық айнымалылар.

Жергілікті айнымалылар. Кластағы жергілікті айнымалы-ол анықталған код блогының ішінде ғана қолжетімді болатын айнымалы. Мысалы, егер сіз әдіс ішінде айнымалыны анықтасаңыз, оған әдістен тыс жерден кіруге болмайды. Келесі скриптке қараймыз [1, 8, 17-20]:

```
# Car класын құрамыз
class Car:
    def start(self):
        message = " Қозғалтқыш ашылды"
        return message
```

Жоғарыда скриптте біз start() car класының ішінде жергілікті message айнымалысын құрдық. Енді Car класы нысанын жасап, төменде көрсетілгендей жергілікті message айнымалысына қол жеткізуге тырысамыз:

```
car_a = Car()
print(car_a.message)
```

Жоғарыда Скрипт келесі AttributeError қатесіне әкеледі:

```
AttributeError: 'Car' object has no attribute 'message'
```

Бұл жергілікті айнымалы анықталған блоктан тыс жергілікті айнымалыға қол жеткізе алмауымызға байланысты.

Жаһандық айнымалы. Ғаламдық айнымалы кез келген блоктан тыс, яғни әдіс, if операторлары және т.б. анықталады. Жаһандық айнымалыға қол жеткізу сыныпта кез келген жерде алынуы мүмкін. Келесі мысал қарастырайық [1, 8, 17-20].

```
# создаем класс Car
class Car:
    message1 = " Қозғалтқыш ашылды"
```

```
        def start(self):
            message2 = "Автокөлік қосылды"
            return message2
car_a = Car()
print(car_a.message1)
```

Бұл скриптте біз жаһандық message1 айнымалысын құрдық және оның мәнін экранға шығардық. Берілісте сіз қатесіз шығарылған message1 айнымалы мәнін көресіз.

Сынып атрибуттары мен даналарының, сондай-ақ жаһандық және жергілікті айнымалылардың арасындағы айырмашылық бар екеніне назар аударыңыз.

Дананың және сыныптың атрибуттары оларға қол жеткізу тәсілімен ерекшеленеді. Басқаша айтқанда, сынып атауын пайдалану және дананың атауын пайдалану туралы. Екінші жағынан, жаһандық және жергілікті айнымалылар өз көрініу салаларымен, басқа сөзбен, оларға қол жеткізуге болатын жерлермен ерекшеленеді.

Жергілікті айнымалыға кіру тек әдісті ішінде ғана алынуы мүмкін. Бұл мақалада жергілікті айнымалылар мен даналардың атрибуттары әдіс ішінде анықталса да, жергілікті айнымалылар өз кілт сөзімен анықталады.

3.1.4 Қол жеткізу модификаторлары

Python қатынау модификаторлары әдепкі айнымалы көрініу аймағын түрлендіру үшін қолданылады. Python ООР үш түрі бар [1, 8, 17-20]:

- көпшілік – public;
- жеке – private;
- қорғалған – protected.

Ашық қол жетімділік түрлендіргіштері бар айнымалыларға қол жетімділік кластан тыс кез келген нүктеден ашық, жеке айнымалыларға қол жетімділік тек сынып ішінде ғана ашық және қорғалған айнымалылармен болған жағдайда, қол жетімділік тек сол пакеттің ішінде ғана ашық.

Жеке айнымалы құру үшін, Сіз айнымалы атауы бар қос астын сызу префиксін қою керек.

Қорғалған айнымалы құру үшін, Сіз бір төменгі астын сызу префиксін айнымалы деп атау керек. Көпшілік айнымалылары үшін префикстерді мүлдем қою қажет емес.

Жария, жеке және қорғалған айнымалыларды қарап көрейік. Келесі скриптті орындаңыз [1, 8, 17-20]:

```
class Car:
    def __init__(self):
        print ("Қозғалтқыш ашылды")
        self.name = "corolla"
        self.__make = "toyota"
        self._model = 1999
```

Мұнда біз конструктор және үш айнымалысы бар қарапайым Car класын құрдық: name, make, және model (атауы, маркасы және моделі). Name айнымалысы ашық, ал make және model айнымалылар тиісінше жеке және қорғалған.

Car класының нысанын жасайық және name айнымалысына қол жеткізуге тырысамыз. Келесі скриптті орындаңыз:

```
car_a = Car()
print(car_a.name)
```

Name ашық айнымалы болғандықтан, біз оған сыныптан тыс қол жеткізе аламыз. Берілісте консольде шығарылған name айнымалы мәнін көресіз.

Енді make айнымалы мәнін шығаруға тырысамыз. Келесі скриптті орындаңыз:

```
print(car_a.make)
```

Беруден біз аламыз келесі қате туралы хабарлама:

```
AttributeError: 'Car' object has no attribute 'make'
```

Біз алдыңғы екі секцияда объектілі-бағытталған бағдарламалаудың негізгі тұжырымдамаларының көп бөлігін қарастырдық. Енді, объектілі-бағытталған бағдарламалау бағандары туралы сөйлесейік:

- Полиморфизм;
- Мұрагерлік;
- Инкапсуляция

3.1.5 Мұрагерлену

Объектілі-бағдарлы бағдарламалауда мұрагерлік бала өзінің сипаттарына қосымша оның ата-анасының қандай да бір сипаттамаларын мұра ететін нақты өмірде мұраға ұқсайды.

Мысалы, болид-бұл көлік. Мұрагерлік-бұл объектілі-бағытталған бағдарламалау ең ғажайып тұжырымдамаларының бірі, өйткені ол қайта пайдалануды білдіреді.

Объектілі-бағдарлы бағдарламалауда мұрагерлік негізгі идеясы класс басқа класс сипаттамаларына ие бола алады. Басқа сыныпқа жататын сынып еншілес сынып немесе туынды сынып деп аталады және мұра беретін сынып ата-ана немесе негізгі деп аталады.

Мұраны алудың өте қарапайым үлгісін қарастырайық.

Келесі скриптті орындаңыз [1, 8, 17-20]:

```
# Создание класса Vehicle
class Vehicle:
    def vehicle_method(self):
        print("Это родительский метод из класса Vehicle")

# Создание класса Car, который наследует Vehicle
class Car(Vehicle):
    def car_method(self):
        print("Это метод из дочернего класса")
```

Жоғарыда скриптте біз екі класс жасаймыз: Vehicle және Car, ол Vehicle класын мұра. Класты иелену үшін, Сіз тек қана жақшаның ішінде ата-ана класының атауын жазу керек, ол еншілес сынып атауынан кейін. Vehicle класы vehicle_method() әдісін қамтиды, ал еншілес класы car_method() әдісін қамтиды. Дегенмен, Car класы Vehicle класын иеленеді, ол сондай-ақ vehicle_method() әдісін де иеленеді.

Мұны іс жүзінде қарастырайық және келесі скриптті орындаңыз:

```
car_a = Car()
car_a.vehicle_method() # Вызываем метод родительского класса
```

Бұл скриптте біз Car класы нысанын құрдық Vehicle_method() әдісін Car класы нысанының көмегімен шақырдық. Сіз Car класы бір де vehicle_method() әдісін

қамтымайтынына назар аудара аласыз, бірақ ол vehicle_method() бар Vehicle класы мұра болғандықтан, Car класы, сондай-ақ оны пайдалануға болады. Беру келесідей:

Это родительский метод из класса Vehicle

3.1.6 Көптеген Python мұрагерлік

Python-да, ата-ана класы бірнеше еншілес болуы мүмкін, және ұқсас, еншілес сынып бірнеше ата-аналық сынып болуы мүмкін. Бірінші сценарийді қарастырайық. Келесі скриптті орындаңыз [1, 8, 17-20]:

```
# создаем класс Vehicle
class Vehicle:
    def vehicle_method(self):
        print("Это родительский метод из класса Vehicle")

# создаем класс Car, который наследует Vehicle
class Car(Vehicle):
    def car_method(self):
        print("Это дочерний метод из класса Car")

# создаем класс Cycle, который наследует Vehicle
class Cycle(Vehicle):
    def cycleMethod(self):
        print("Это дочерний метод из класса Cycle")
```

Бұл скриптте, Vehicle ата – аналар класы екі қызы – Car және Cycle. Екі еншілес сынып vehicle_method() ата-аналық сынып қол жетімді болады. Оны жеке көру үшін келесі сценарийді іске қосыңыз:

```
car_a = Car()
car_a.vehicle_method() # вызов метода родительского класса
car_b = Cycle()
car_b.vehicle_method() # вызов метода родительского класса
```

Ұстап беру кезінде сіз vehicle_method() әдісін беруді төменде көрсетілгендей екі рет көресіз:

Это родительский метод из класса Vehicle
Это родительский метод из класса Vehicle

Сіз ата-ана класы екі еншілес сыныптармен мұраға қалай отыратынын көре аласыз. Осылайша, еншілес сынып бірнеше ата-ана болуы мүмкін. Мысал қарайық [1, 8, 17-20]:

```
class Camera:
    def camera_method(self):
        print("Это родительский метод из класса Camera")
class Radio:
    def radio_method(self):
        print("Это родительский метод из класса Radio")
class CellPhone(Camera, Radio):
    def cell_phone_method(self):
        print("Это дочерний метод из класса CellPhone")
```

Скриптте біз үш класты құрдық: Camera, Radio, және CellPhone. Camera және Radio класстары CellPhone класымен мұраға қалады. Бұл дегеніміз, CellPhone класы Camera және Radio сыныптарының әдістеріне қол жеткізуге болады. Келесі скрипт растайды:

```
cell_phone_a = CellPhone()
cell_phone_a.camera_method()
cell_phone_a.radio_method()
```

Беру келесідей болады:

```
Это родительский метод из класса Camera
Это родительский метод из класса Radio
```

3.1.7 Полиморфизм

Полиморфизм термині бірнеше форманың болуын білдіреді. Объектілі-бағытталған бағдарламалау контекстінде полиморфизм объектінің өзін әртүрлі ұстау қабілетін білдіреді.

Программалауда Полиморфизм әдісті қайта жүктеу арқылы немесе оны қайта анықтау арқылы іске асырылады.

Әдісті қайта жүктеу. Әдісті қайта жүктеу параметрлердің санына немесе түріне байланысты өзін әртүрлі ұстау әдісінің қасиетіне жатады. Әдіс жүктемесінің өте қарапайым мысалына қарайық. Келесі скриптті орындаңыз:

```
# создаем класс Car
class Car:
    def start(self, a, b=None):
        if b is not None:
            print (a + b)
        else:
            print (a)
```

Скриптте жоғары, егер start() әдісі бір Аргументтің берілуімен байланысты болса, параметр экранға шығады. Алайда, егер біз start әдісіне 2 дәлел () берсе, ол екі дәлел енгізеді және сома нәтижесін шығарады.

Бастау үшін бір аргументпен көріңіз:

```
car_a = Car()
car_a.start(10)
```

Ұстап беруде біз 10 көре аламыз. Енді екі дәлелді беруге тырысамыз:

```
car_a.start(10, 20)
```

Соңында шығатыны 30.

Әдісті қайта анықтау. Әдісті қайта анықтау қызы және ата-ана сыныптарында бірдей атауы бар әдістің болуына жатады. Әдіс анықтамасы ата-ана мен қызы сыныптарында ерекшеленеді, бірақ атауы бірдей болып қалады. Python әдісін қайта анықтаудың қарапайым мысалын көрейік.

```
# создание класса Vehicle
class Vehicle:
    def print_details(self):
```

```
            print("Это родительский метод из класса Vehicle")

# создание класса, который наследует Vehicle
class Car(Vehicle):
    def print_details(self):
        print("Это дочерний метод из класса Car")

# создание класса Cycle, который наследует Vehicle
class Cycle(Vehicle):
    def print_details(self):
        print("Это дочерний метод из класса Cycle")
```

Жоғарыда скриптте, cycle және Car кластары vehicle класын мұра. Vehicle класы print_details() әдісін қамтиды. Енді сіз print_details() әдісін шақырған болсаңыз, беру әдісі пайда болатын нысанға байланысты болады. Іс мәнін түсіну үшін келесі скриптті орындаңыз [1, 8, 17-20]::

```
car_a = Vehicle()
car_a. print_details()
car_b = Car()
car_b.print_details()
car_c = Cycle()
car_c.print_details()
```

Беру осылай көрінеді:

```
Это родительский метод из класса Vehicle
Это дочерний метод из класса Car
Это дочерний метод из класса Cycle
```

Сіз көріп отырғаныңыздай, беру әр түрлі, сол print_details() әдісі бір базалық сыныптың туынды сыныптары арқылы шақырылады. Алайда, еншілес сыныптар ата-ана класы әдісімен қайта анықталған, әдістер әртүрлі.

3.1.8 Инкапсуляция

Инкапсуляция – объектілі-бағытталған бағдарламалаудың үшінші бағанасы. Инкапсуляция деректерді жасыру дегенді білдіреді. Әдетте, объектілі-бағытталған бағдарламалауда бір сынып басқа сынып деректеріне тікелей қол жеткізбеуі тиіс. Оның орнына, кіру сынып әдістері арқылы бақылануы тиіс [1, 8, 17-20]:.

Python-да сынып деректеріне бақыланатын қатынауды қамтамасыз ету үшін кіру модификаторлары мен қасиеттері қолданылады. Біз қол жеткізудің модификаторларының қалай жұмыс істейтінімен таныстық. Бұл бөлімде сипаттардың қалай жұмыс істейтінін көреміз.

Біз автомобиль моделі 2000 және 2018 жыл арасында даталануға тиіс екеніне көз жеткізуіміз керек деп ойлаймын. Егер пайдаланушы автомобиль моделі үшін 2000-дан аз мәнді енгізуге тырысса, мәні автоматты түрде 2000 ретінде орнатылады және 2018-дан жоғары болса, ол 2018-ға орнатылуы тиіс. Егер мән 2000 мен 2018 арасында болса-ол өзгеріссіз қалады. Біз осы логиканы жүзеге асыратын модель төлсипатының қасиетін жасай аламыз. Біз мысалға қараймыз:

```
# создаем класс Car
class Car:
```

```
 # создаем конструктор класса Car
 def __init__(self, model):
     # Инициализация свойств.
     self.model = model

  # создаем свойство модели.
 @property
 def model(self):
     return self.__model

# Сеттер для создания свойств.
 @model.setter
 def model(self, model):
     if model < 2000:
         self.__model = 2000
     elif model > 2018:
         self.__model = 2018
     else:
         self.__model = model
 def getCarModel(self):
     return "Год выпуска модели " + str(self.model)
```

Қасиеттің үш бөлігі бар. Сіз жоғарыда скрипт үлгісі болып табылатын төлсипатты анықтау керек. Содан кейін, @property декораторымен атрибуттың қасиетін анықтау керек. Ақыр соңында, @model дескриптор болып табылатын сипатты орнатушыны жасау керек.жоғарыда мысалында setter.

Енді сіз модель төлсипатында 2018 Жоғары мәнін енгізуге тырыссаңыз, сіз мәні орнатылған деп көресіз 2018. Оны тексерейік. Келесі скриптті орындаңыз:

```
car_a = Car(2088)
print(car_a.get_car_model())
```

Мұнда біз модель үшін мән ретінде 2088 береді, алайда, егер сіз get_car_model() арқылы модель төлсипаты үшін мәнді енгізсеңіз, Сіз 2018.

Бұл мақалада біз объектілі-бағытталған бағдарламалаудың маңызды негіздерінің бір бөлігін меңгердік. Бұл бағдарлама түрі-ең танымал және қолданылатын парадигмалардың бірі.

Объектілі-бағытталған бағдарламалаудың маңыздылығы қазіргі заманғы бағдарламалау тілдерінің көп бөлігі объектілі-бағытталған немесе кем дегенде объектілі-бағытталған бағдарламалауды қолдайтын факт болып табылады.

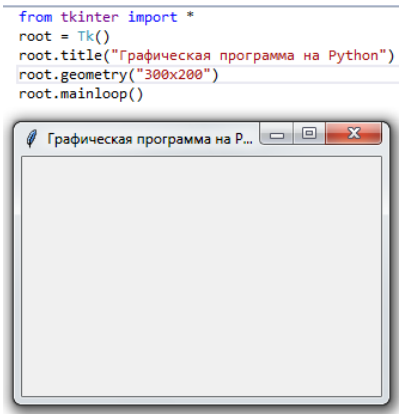
3.2 Tkinter графикалық интерфейсін құру

3.2.1 Қолданба терезесін жасау

Көптеген бағдарламалар бүгінгі таңда консольден гөрі пайдаланушы үшін интуитивен және ыңғайлы графикалық интерфейс пайдаланады. Python бағдарламалау тілінің көмегімен графикалық бағдарламаларды жасауға болады. Бұл үшін Python әдепкі бойынша tkinter деп аталатын компоненттер жиынтығы арнайы тулкит қолданылады [1, 8, 17-20].

Tkinter тулкиті барлық қажетті графикалық компоненттерді - батырмалерді, мәтіндік өрістерді және т. б. қамтитын жеке кіріктірілген модуль түрінде қол жетімді.

Графикалық бағдарламаларды құрудың негізгі сәті терезе жасау болып табылады. Содан кейін терезеге Графикалық интерфейстің барлық қалған компоненттері қосылады. Сондықтан алдымен қарапайым терезе жасаймыз. Бұл үшін келесі скриптті анықтаймыз (сурет 3.1):



Сурет 3.1. Бос терезе құру

Графикалық терезені құру үшін tkinter модулінде анықталған Tk() конструкторы қолданылады. Жасалған терезе root айнымалысына беріледі және осы айнымалы арқылы біз терезе төлсипаттарын басқара аламыз. Атап айтқанда, title() әдісі арқылы терезенің тақырыбын орнатуға болады.

Geometry() әдісі арқылы – терезе өлшемі. Өлшемдерді geometry() әдісіне орнату үшін "ені х биіктік" пішіміндегі жол беріледі. Егер қолданба терезесін жасау кезінде geometry() әдісі шақырылмаса, терезе ішкі мазмұнды орналастыру үшін қажетті кеңістікті алады.

Терезені көрсету үшін, оны пайдаланушымен өзара іс-қимыл жасау үшін терезе оқиғаларын өңдеу циклін іске қосатын mainloop() әдісін шақыру керек.

Нәтижесінде скриптті іске қосу кезінде біз осындай бос терезені көреміз (сурет 3.1).

Терезенің бастапқы орны. Әдепкі бойынша, терезе экранның жоғарғы сол жақ бұрышына орналастырылады. Бірақ біз оны өзгерту керек мәндерді geometry() әдісіне бере аламыз:

```
from tkinter import *
root = Tk()
root.title("Графическая программа на Python")
root.geometry("400x300+300+250")
root.mainloop()
```

Енді geometry әдісіндегі жол келесі форматқа ие: "ені х биіктік + координаттар + координатаУ". Яғни, іске қосылған кезде, терезе оңға 300 пиксельде және экранның жоғарғы сол бұрышынан төмен 250 пиксельде болады.

3.2.2 Батырмалар

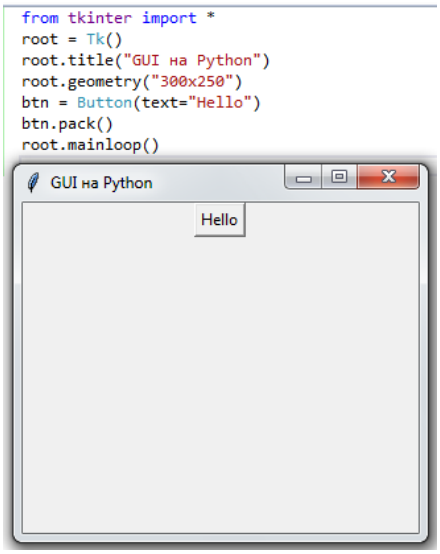
Tkinter тулкиті компоненттер немесе виджеттер жиынтығын қамтиды, олардың бірі батырма болып табылады. Терезеге батырманы қосыңыз [1, 8, 17-20]:

```
from tkinter import *
root = Tk()
```

```
root.title("GUI на Python")
root.geometry("300x250")
btn = Button(text="Hello")
btn.pack()
root.mainloop()
```

Батырмані жасау үшін button() конструкторы қолданылады. Осы конструкторда text параметрімен батырма мәтінін орнатуға болады.

Элементті көру үшін pack() әдісі бар. Нәтижесінде терезенің жоғарғы жағында батырма болады (сурет 3.2).

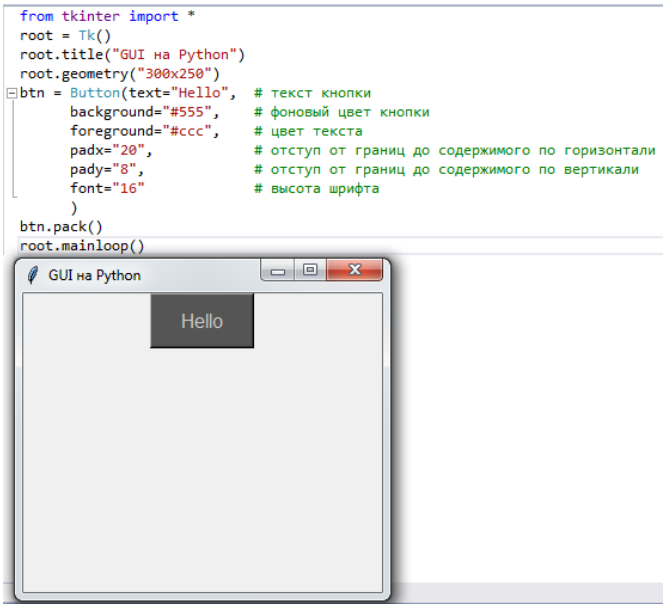


Сурет 3.2. Батырманы құру

Әрбір виджет, оның ішінде батырма, оның визуализациясына әсер ететін және конструктор арқылы теңшей алатын бірқатар атрибуттар бар:

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
btn = Button(text="Hello",      # батырма мәтіні
        background="#555",      # батырманың фондық түсі
        foreground="#ccc",      # мәтін түсі
        padx="20",              # шектерден бастап  ішіндегіге дейін көлденеңінен шегініс
        pady="8",               # шектерден тік ішіндегіге дейінгі шегініс
        font="16"               # қаріп биіктігі
        )
btn.pack()
root.mainloop()
```

Pady, padx, font параметрлері сандық мәнді қабылдайды, ал background және foreground параметрлері он алтылық түс мәнін алады. Font параметрінде қаріп анықтамасы бар (сурет 3.3).



Сурет 3.3. Түрлі атрибуттары бар батырманы құру

Button конструкторы келесі параметрлерді қабылдай алады:

Button (master, options)

Master параметрі ата-аналық контейнер сілтемесін ұсынады. Жоғарыда жағдайда бұл графикалық терезе болуы мүмкін және біз жаза алар едік:

```
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
btn = Button(root, text="Hello")
btn.pack()
```

Егер кодта бір терезе құрылса, онда батырма және кез келген басқа элемент әдепкі бойынша осы терезеде орналастырылады. Сондықтан бірінші параметрді мысал ретінде жоғары түсіре аламыз. Егер кодта бірнеше терезе пайда болса, онда біз Button конструкторына қажетті терезеге сілтемені жібере аламыз.

Options екінші параметрі шын мәнінде біз олардың атынан орната алатын параметрлер жиынтығын ұсынады [1, 8, 17-20]:

activebackground: батырма түсі түрту күйінде болған кезде activeforeground: батырма мәтінінің түсі, ол басылған күйінде;

bd: шекара қалыңдығы (әдепкі 2);

bg/background: батырманың фондық түсі;

fg/foreground: батырма мәтінің түсі;

font: мәтін қарпі, мысалы, font="Arial 14" – қаріп Arial биіктігі 14px, немесе font=("Verdana", 13, "bold") – қаріп Verdana биіктігі 13px қоюланған;

height: батырма биіктігі;

highlightcolor: батырма түсі, ол фокуста;

image: батырмадағы сурет;

justify: мәтінді туралауды орнатады. LEFT мәні мәтіннің сол жақ шетінде, CENTER – ортасында, RIGHT-оң жақ шетінде;

padx: батырманің шекарасынан оң және сол жақтағы мәтінге дейінгі шегініс;

pady: батырманың шегінен оның мәтініне дейін жоғарғы және төменгі шегініс;

relief: шекара түрін анықтайды, SUNKEN, RAISED, GROOVE, RIDGE мәндерін қабылдай алады;

state: батырма күйін орнатады, DISABLED, ACTIVE, NORMAL (әдепкі);

text: батырманың мәтінін орнатады;

textvariable: StringVar элементіне байлау орнатады;

underline: астын сызылатын батырманың мәтініндегі таңбаның нөмірін көрсетеді. Әдепкі мән -1, яғни ешқандай таңба сызылмайды;

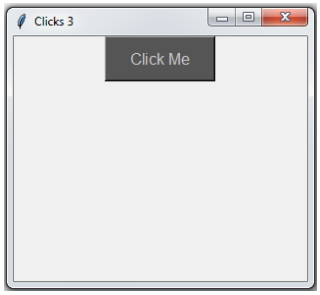
width: батырманың ені;

wraplength: мәтін жолының оң мәні болса, батырманың кеңістігіне орналастыру үшін ауыстырылады.

*Батырмаға басуды өңдеу.*Батырмані басуды өңдеу үшін, басқанда іске қосылатын функцияға сілтеме бере отырып, конструкторда command параметрін орнату қажет:

```
from tkinter import *
clicks = 0
def click_button():
   global clicks
   clicks += 1
   root.title("Clicks {}".format(clicks))
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
btn = Button(text="Click Me", background="#555",
                        foreground="#ccc", padx="20", pady="8",
                        font="16", command=click_button)
btn.pack()
root.mainloop()
```

Мұнда басыңыз өңдеуші ретінде click_button функциясы орнатылады. Бұл функцияда жаһандық clicks айнымалысы өзгереді, ол басу санын сақтайды және оның мәні терезе айдарына шығады. Осылайша, батырманы әрбір басқанда click_button функциясы іске қосылады және Клик саны артады (сурет 3.4).



Сурет 3.4. Батырманы басуды өңдеу

3.2.3 Элементтердің қасиеттерін өзгерту

Кейде мәтін сияқты батырмалер төлсипаттарын өзгерту қажет болуы мүмкін. Дегенмен, сол батырмада мәтінді өзгерту үшін ешқандай әдіс жоқ. Сұрақ туындайды, оны қалай өзгертуге болады?

Мәтінді өзгерту үшін StringVar аралық компонентін пайдалана аламыз. Бұл компонент жолға байланыстыруды жасауға мүмкіндік береді.

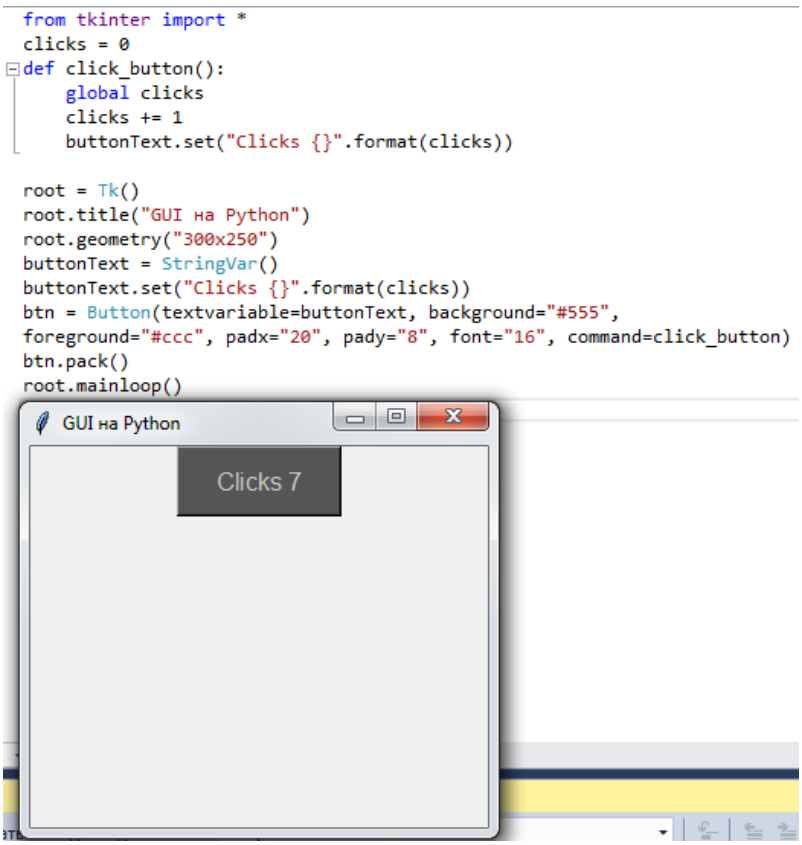
Ол екі әдісі бар:

get(): StringVar жолын қайтарады

set(str): StringVar жолын орнатады

StringVar нысанының визуалды элементтің мәтінімен конструктордағы байланысы үшін textvariable параметрін орнату керек. Мысалы, батырманы басу арқылы оның мәтіні өзгертілсін (сурет 3.5).

```
from tkinter import *
clicks = 0
def click_button():
    global clicks
    clicks += 1
    buttonText.set("Clicks {}".format(clicks))

root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
buttonText = StringVar()
buttonText.set("Clicks {}".format(clicks))
btn = Button(textvariable=buttonText, background="#555",
foreground="#ccc", padx="20", pady="8", font="16", command=click_button)
btn.pack()
root.mainloop()
```



Сурет 3.5. Батырма мәтінін өзгерту

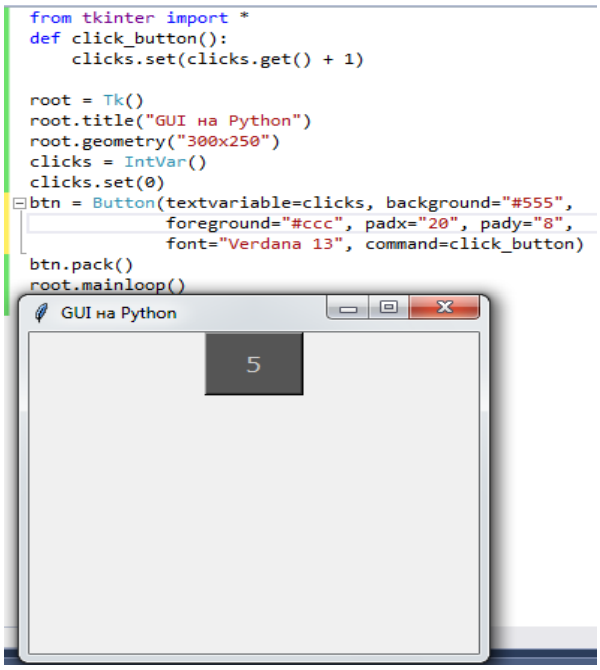
Мұнда StringVar түрі бар buttonText айнымалысы бар. Ол үшін бастапқы мән белгіленеді, содан кейін btn айнымалы textvariable параметрі арқылы осы мәтінмен байланыстырылады. Соңында батырманы басқан кезде оның мәтіні өзгереді.

StringVar құрамдастарынан басқа, басқа деректер түрлері үшін ұқсас компоненттер қатары бар:

– IntVar
– BooleanVar
– DoubleVar

Мысалы, Біз IntVar айнымалысына байланыстырғышты орната аламыз және Клик санын шығара аламыз (сурет 3.6).

```
from tkinter import *
def click_button():
    clicks.set(clicks.get() + 1)

root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
clicks = IntVar()
clicks.set(0)
btn = Button(textvariable=clicks, background="#555",
             foreground="#ccc", padx="20", pady="8",
             font="Verdana 13", command=click_button)
btn.pack()
root.mainloop()
```



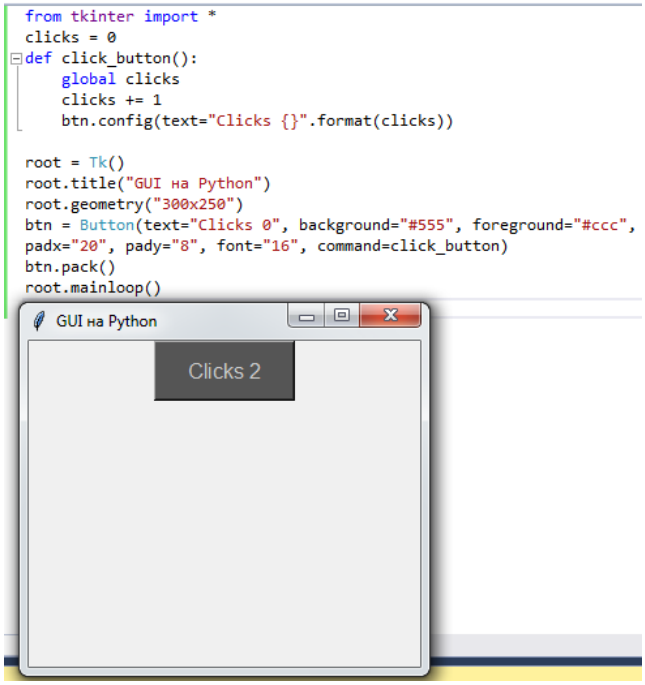
Сурет 3.6. IntVar компонентін пайдалану

Config әдісі.

Бірақ тек мәтінді ғана емес, басқа да батырма параметрлерін немесе басқа компонентті өзгерту қажет болса, әсіресе көмектесетін басқа әдіс бар. Бұл әдіс config() әдісінің элементіне қоңырау шалу болып табылады, онда қажетті параметр

орнатылады. Мысалы, скрипт кодын өзгерту арқылы config әдісін қолданамыз (сурет 3.7) [1, 8, 17-20].

```
from tkinter import *
clicks = 0
def click_button():
    global clicks
    clicks += 1
    btn.config(text="Clicks {}".format(clicks))

root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
btn = Button(text="Clicks 0", background="#555", foreground="#ccc", padx="20", pady="8", font="16", command=click_button)
btn.pack()
root.mainloop()
```



Сурет 3.7. Config әдісін пайдалану

3.2.4 Элементтерді позициялау

Pack әдісі.

Терезеде элементтерді позициялау үшін әр түрлі әдістер қолданылады және ең қарапайым әдіс pack() әдісінің элементіне қоңырау білдіреді. Бұл әдіс келесі параметрлерді қабылдайды [1, 8, 17-20]:

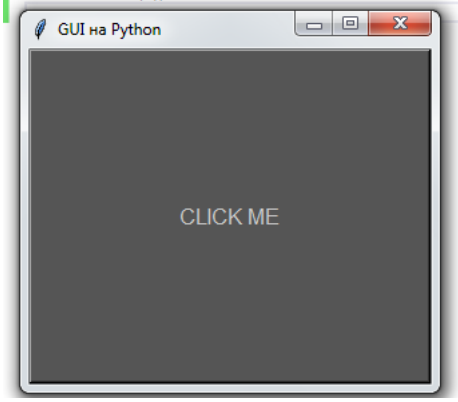
expand: егер True болса, виджет контейнердің барлық кеңістігін толтырады.

fill: виджет айналасында бос кеңістікті толтыру үшін созылатынын анықтайды. Бұл параметр келесі мәндерді қабылдай алады: NONE (әдепкі, элемент созылмайды), X (элемент тек көлденеңінен созылады), Y (элемент тек тігінен созылады) және BOTH (элемент тігінен және көлденеңінен созылады).

side: виджетті контейнердің бір жағынан тегістейді. Жоғары (әдепкі бойынша, контейнердің жоғарғы жағында туралanады), BOTTOM (төменгі жағында туралanады), LEFT (сол жағында туралanады), RIGHT (оң жағында туралanады) мәндерін қабылдай алады.

Мысалы, expand және fill параметрлерін пайдалана отырып, барлық пішіндегі батырмані созыңыз (сурет 3.8).

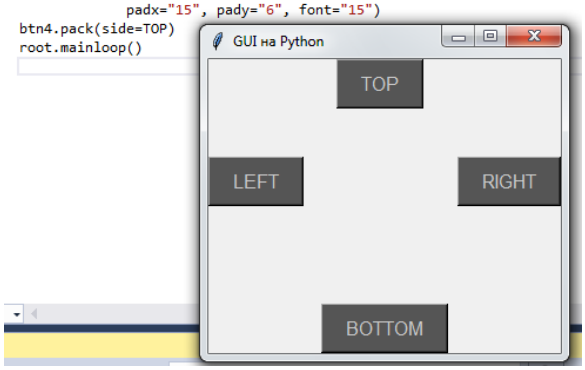
```
from tkinter import *

root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
btn1 = Button(text="CLICK ME", background="#555", foreground="#ccc",
              padx="15", pady="6", font="15")
btn1.pack(expand = True, fill=BOTH)
root.mainloop()
```



Сурет 3.8. Pack әдісін пайдалану (expand және fill параметрлері)

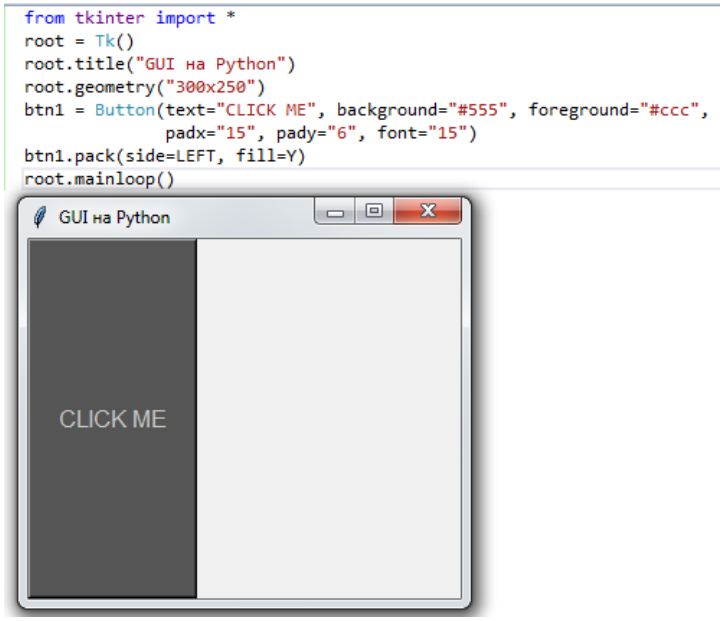
Side параметрін пайдаланамыз (сурет 3.9).

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
btn1 = Button(text="BOTTOM", background="#555", foreground="#ccc",
              padx="15", pady="6", font="15")
btn1.pack(side=BOTTOM)
btn2 = Button(text="RIGHT", background="#555", foreground="#ccc",
              padx="15", pady="6", font="15")
btn2.pack(side=RIGHT)
btn3 = Button(text="LEFT", background="#555", foreground="#ccc",
              padx="15", pady="6", font="15")
btn3.pack(side=LEFT)
btn4 = Button(text="TOP", background="#555", foreground="#ccc",
              padx="15", pady="6", font="15")
btn4.pack(side=TOP)
root.mainloop()
```



Сурет 3.9. Pack әдісін пайдалану (side параметрі)

Side және fill параметрлерін біріктіріп, элементті тігінен созуға болады (сурет 3.10).



Сурет 3.10. Pack әдісін пайдалану (side және fill параметрлері)

Place әдісі.

Place() әдісі позициялау параметрлерін дәл реттеуге мүмкіндік береді. Ол келесі параметрлерді қабылдайды [1, 8, 17-20]:

height және width: пикселдегі элементтің биіктігі мен еніне сәйкес орнатады;

relheight және relwidth: сондай-ақ элементтің биіктігі мен еніне сәйкес келеді, бірақ мән ретінде 0.0 және 1.0 арасындағы аралықтағы float саны қолданылады, ол басты контейнердің биіктігі мен енінен үлесін көрсетеді;

x және y: контейнердің жоғарғы сол бұрышына қатысты пикселде элементтің көлденең және тігінен жылжуын орнатады;

relx және rely: сондай-ақ элементтің көлденең және тігінен жылжуын орнатады, бірақ мән ретінде 0.0 және 1.0 арасындағы аралықтағы float саны қолданылады, ол басты контейнердің биіктігі мен енінен үлес көрсетеді;

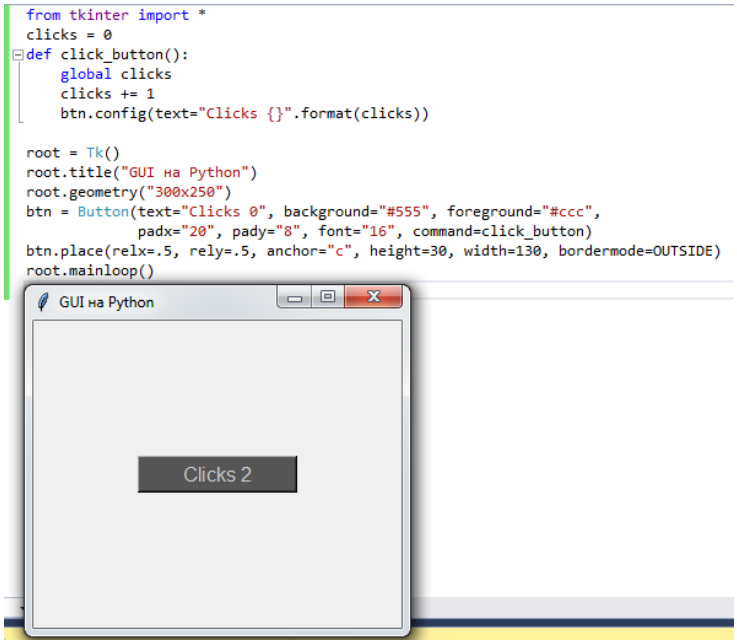
bordermode: элемент шегінің пішімін көрсетеді. INSIDE (әдепкі) және OUTSIDE мәнін қабылдай алады;

anchor: элементтің созылу опцияларын орнатады. Noth(Солтүстік – жоғары), South (Оңтүстік – төмен), East (Шығыс – оң жағы), West (Батыс – сол жағы) және Center (ортасында) қысқартулары болып табылатын N, e, s, w, ne, NW, se, sw, c мәндерін қабылдай алады. Мысалы, NW жоғарғы сол жақ бұрышын көрсетеді.

Мысалы, терезенің ортасында 130 пиксель ені мен 30 пиксель биіктігі бар батырмані орналастырамыз (сурет 3.11).

```
from tkinter import *
clicks = 0
def click_button():
    global clicks
    clicks += 1
    btn.config(text="Clicks {}".format(clicks))

root = Tk()
root.title("GUI на Python")
```

```
root.geometry("300x250")
btn = Button(text="Clicks 0", background="#555",foreground="#ccc",
             padx="20", pady="8", font="16", command=click_button)
btn.place(relx=.5,rely=.5,anchor="c", height=30, width=130, bordermode=OUTSIDE)
root.mainloop()
```



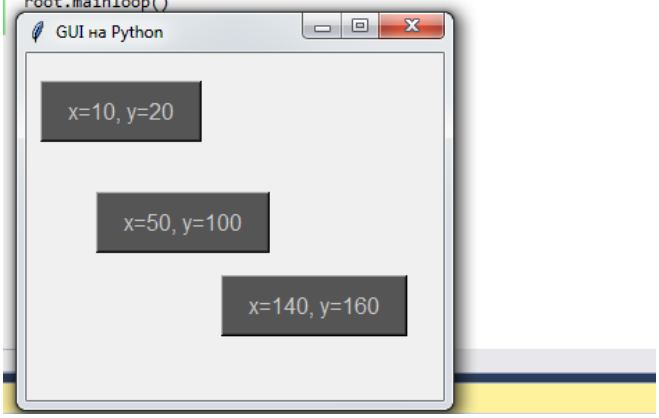
Сурет 3.11. Place әдісін пайдалану

Place() әдісін қолданғанда, элементті көрінетін ету үшін pack() әдісін пайдаланбау керек.

Немесе үш батырмані орналастырамыз (сурет 3.12).

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
btn1 = Button(text="x=10, y=20", background="#555",
              foreground="#ccc", padx="14", pady="7",
              font="13")
btn1.place(x=10, y=20)
btn2 = Button(text="x=50, y=100", background="#555",
              foreground="#ccc", padx="14", pady="7",
              font="13")
btn2.place(x=50, y=100)
btn3 = Button(text="x=140, y=160", background="#555",
              foreground="#ccc", padx="14", pady="7",
              font="13")
btn3.place(x=140, y=160)
root.mainloop()
```

```
from tkinter import *

root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
btn1 = Button(text="x=10, y=20", background="#555",
              foreground="#ccc", padx="14", pady="7", font="13")
btn1.place(x=10, y=20)
btn2 = Button(text="x=50, y=100", background="#555",
              foreground="#ccc", padx="14", pady="7", font="13")
btn2.place(x=50, y=100)
btn3 = Button(text="x=140, y=160", background="#555",
              foreground="#ccc", padx="14", pady="7", font="13")
btn3.place(x=140, y=160)
root.mainloop()
```



Сурет 3.12. Place әдісін пайдалану

Grid әдісі.

Grid әдісі элементті шартты тордың немесе гридтің белгілі бір ұяшығына орналастыруға мүмкіндік беретін place әдісіне қарағанда элементтерді жайғастыруға басқа тәсілді қолданады [1, 8, 17-20].

Grid әдісі келесі параметрлерді қолданады:

column: баған нөмірі, Сан нөлден басталады;

row: жол нөмірі, санақ нөлден басталады;

columnspan: элементті қанша баған алуы керек;

rowspan: элементті қанша жол алуы керек;

ipadx және ipady: элементтің шекарасынан оның мәтініне дейінгі көлденең және тігінен шегіністер;

padx және pady: грид ұяшығының шекарасынан элементтің шекарасына дейінгі көлденең және тігінен шегіністер;

sticky: ұяшық элементтен көп болса, ұяшықтағы элементті туралау. Тиісті туралау бағытын көрсететін N, e, s, w, NE, NW, se, sw мәндерін қабылдай алады.

Мысалы, 9 батырмадан гридты анықтаймыз (сурет 3.13):

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
for r in range(3):
    for c in range(3):
        btn = Button(text="{}-{}".format(r,c))
        btn.grid(row=r, column=c, ipadx=10, ipady=6,
                 padx=10, pady=10)
root.mainloop()
```

```
from tkinter import *

root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
for r in range(3):
    for c in range(3):
        btn = Button(text="{}-{}".format(r,c))
        btn.grid(row=r, column=c, ipadx=10, ipady=6, padx=10, pady=10)
root.mainloop()
```

Сурет 3.13. Grid әдісін пайдалану

3.2.5 Label мәтіндік белгісі және Entry енгізу өрісі

Label мәтіндік белгісі.

Python мәтіндік белгілері Label элементімен берілген. Бұл элемент өңдеу мүмкіндігінсіз статикалық мәтінді шығаруға мүмкіндік береді.

Label элементін жасау үшін екі параметр қабылдайтын конструктор қолданылады:

Label(master, options)

Master параметрі ата-аналық контейнер сілтемесін ұсынады және options параметрі келесі атаулы параметрлерді ұсынады [1, 8, 17-20]:

anchor: мәтін позициясын орнатады;

bg / background: фон түсі;

bitmap: таңбада көрсетілетін Сурет сілтемесі;

bd: белгі шегінің қалыңдығы;

fg / foreground: мәтін түсі;

font: мәтін қаріпі, мысалы, font= "Arial 14" - биіктігі 14px Arial қаріпі;

height: Элемент биіктігі;

cursor: белгіге апарғанда тінтуір көрсеткішінің меңзері;

image: белгіде көрсетілетін Сурет сілтемесі;

justify: мәтінді туралауды орнатады. LEFT мәні мәтіннің сол жақ шетінде, CENTER – ортасында, RIGHT-оң жақ шетінде;

padx: элемент шекарасынан оң және сол жақтағы мәтінге дейінгі шегініс;

pady: элемент шекарасынан оның мәтініне дейін жоғарғы және төменгі;

relief: шекара түрін анықтайды, әдетте, FLAT мәні;

text: белгі мәтінін орнатады;

textvariable: StringVar элементіне байланыстырады;

underline: сызылған батырма мәтініндегі таңбаның нөмірін көрсетеді. Әдепкі мән -1, яғни ешқандай таңба сызылмайды;

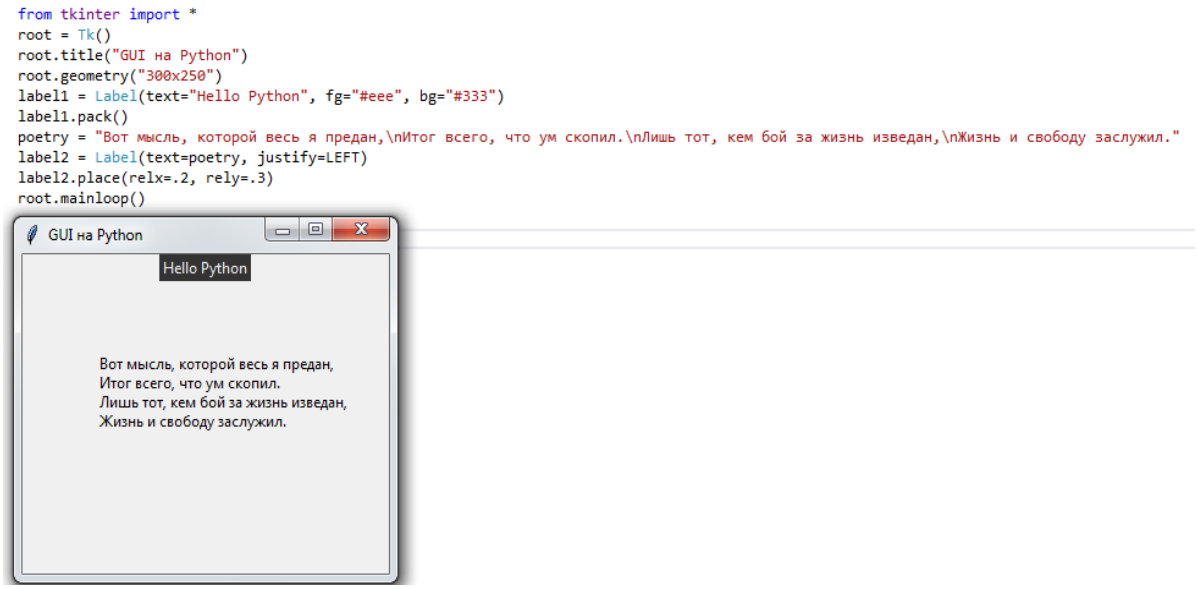
width: Элемент ені;

wraplength: мәтін жолының оң мәні болса, элемент кеңістігіне орналастыру үшін ауыстырылады.

Қолданба терезесінде Қарапайым мәтінді шығарамыз (сурет 3.14):

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
label1 = Label(text="Hello Python", fg="#eee", bg="#333")
label1.pack()
poetry = "Вот мысль, которой весь я предан,\nИтог всего,
        что ум скопил.\nЛишь тот, кем бой за жизнь изведан,\nЖизнь и
        свободу заслужил."
label2 = Label(text=poetry, justify=LEFT)
label2.place(relx=.2, rely=.3)
root.mainloop()
```

Мәтінді басқа жолға көшіру үшін эскейп-бірізділікті \n пайдалануға болады.



Сурет 3.14. Label виджетін қолдану

Entry енгізу өрісі.

Entry элементі мәтін енгізу өрісін көрсетеді. Entry конструкторы келесі параметрлерді қабылдайды:

Entry(master, options)

Мұнда master – ата-ана терезесіне сілтеме, ал options – келесі параметрлер жиынтығы [1, 8, 17-20]:

bg: фон түсі;

bd: шекара қалыңдығы;

cursor: мәтіндік өріске апарғанда тышқан көрсеткіш меңзері;

fg: мәтін түсі;

font: мәтін қаріпі;

justify: мәтінді туралауды орнатады. LEFT мәні мәтіннің сол жақ шетінде, CENTER – ортасында, RIGHT – оң жақ шетінде;

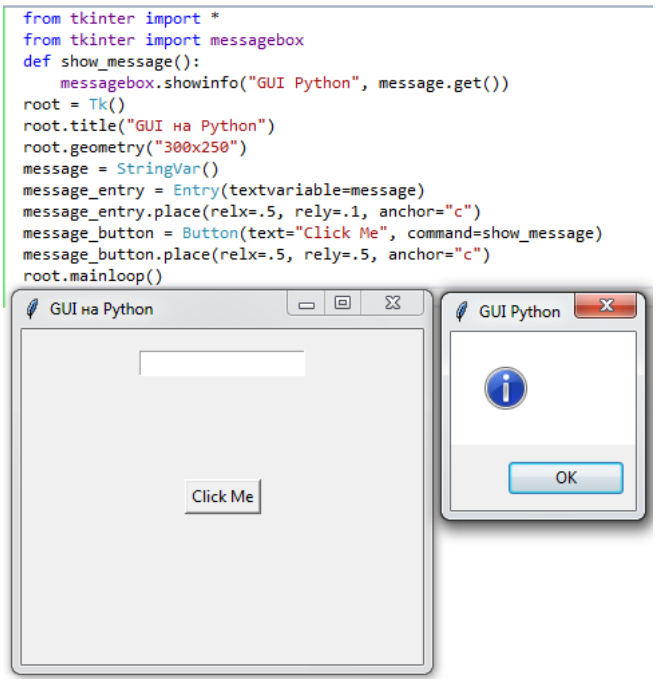
relief: шекара түрін анықтайды, әдетте, FLAT мәні;

selectbackground: таңдалған мәтін бөлігінің фондық түсі;

selectforeground: таңдалған мәтін түсі;
show: енгізілетін таңбалар үшін масканы көрсетеді;
state: элемент күйі, NORMAL (әдепкі) және DISABLED мәндерін қабылдай алады;
textvariable: StringVar элементіне байланыстырады;
width: Элемент ені.

Entry элементін анықтаймыз және батырманы басқаннан кейін оның мәтінін хабарламамен жеке терезеге шығарамыз (сурет 3.15):

```
from tkinter import *
from tkinter import messagebox
def show_message():
   messagebox.showinfo("GUI Python", message.get())
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
message = StringVar()
message_entry = Entry(textvariable=message)
message_entry.place(relx=.5, rely=.1, anchor="c")
message_button = Button(text="Click Me", command=show_message)
message_button.place(relx=.5, rely=.5, anchor="c")
root.mainloop()
```

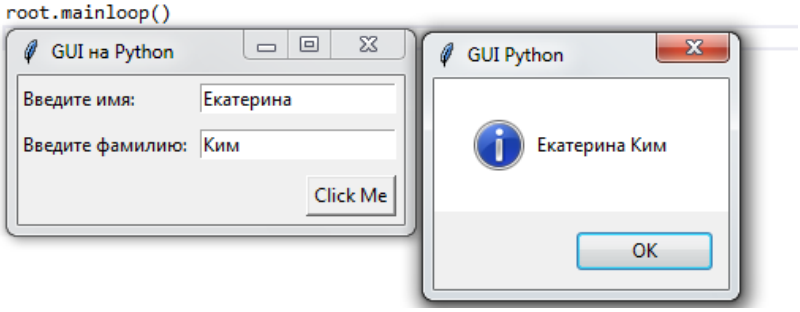


Сурет 3.15. Entry виджетін қолдану

Хабарламаны шығару үшін мұнда мәтін өрісіне енгізілген мәтінді шығаратын showinfo() функциясы бар қосымша messagebox модулі қолданылады. Енгізілген мәтінді алу үшін алдыңғы тақырыптардың бірінде сипатталғандай StringVar жиынтығы қолданылады.

Енді біз енгізу нысаны бар күрделі үлгі жасаймыз (сурет 3.16).

```
from tkinter import *
from tkinter import messagebox
def display_full_name():
    messagebox.showinfo("GUI Python", name.get() + " " + surname.get())
root = Tk()
root.title("GUI на Python")
name = StringVar()
surname = StringVar()
name_label = Label(text="Введите имя:")
surname_label = Label(text="Введите фамилию:")
name_label.grid(row=0, column=0, sticky="w")
surname_label.grid(row=1, column=0, sticky="w")
name_entry = Entry(textvariable=name)
surname_entry = Entry(textvariable=surname)
name_entry.grid(row=0,column=1, padx=5, pady=5)
surname_entry.grid(row=1,column=1, padx=5, pady=5)
message_button = Button(text="Click Me", command=display_full_name)
message_button.grid(row=2,column=1, padx=5, pady=5, sticky="e")
root.mainloop()
```



Сурет 3.16. Entry виджетін қолдану

Entry әдістері.

Entry элементінің бірнеше әдістері бар. Негізгі оның:

insert(index, str): мәтін өрісіне белгілі бір индекс бойынша жол енгізеді;

get(): мәтін өрісіне енгізілген мәтінді қайтарады;

delete(first, last=None): first индексі бойынша таңбаны жояды. Егер last параметрі көрсетілсе, жою last индексіне дейін жүргізіледі. Соңына дейін жою үшін END мәнін екінші параметр ретінде пайдалануға болады.

Бағдарламада әдістерді қолданамыз:

```
from tkinter import *
from tkinter import messagebox
def clear():
   name_entry.delete(0, END)
   surname_entry.delete(0, END)
def display():
      messagebox.showinfo("GUI Python", name_entry.get() + " " +
      surname_entry.get())
root = Tk()
root.title("GUI на Python")
name_label = Label(text="Введите имя:")
surname_label = Label(text="Введите фамилию:")
name_label.grid(row=0, column=0, sticky="w")
surname_label.grid(row=1, column=0, sticky="w")
name_entry = Entry()
surname_entry = Entry()
name_entry.grid(row=0,column=1, padx=5, pady=5)
surname_entry.grid(row=1,column=1, padx=5, pady=5)
```

```
# бастапқы деректерді кірістіру
name_entry.insert(0, "Tom")
surname_entry.insert(0, "Soyer")
display_button = Button(text="Display", command=display)
clear_button = Button(text="Clear", command=clear)
display_button.grid(row=2, column=0, padx=5, pady=5,
                                    sticky="e")
clear_button.grid(row=2, column=1, padx=5, pady=5,
                                    sticky="e")
root.mainloop()
```

Бағдарламаны іске қосқан кезде екі мәтін өрісіне әдепкі мәтін қосылады:

```
name_entry.insert(0, "Tom")
surname_entry.insert(0, "Soyer")
```

Clear батырмасы delete әдісі арқылы екі өрісті тазартады:

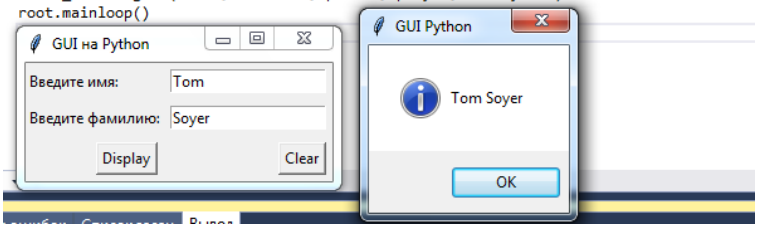
```
def clear():
   name_entry.delete(0, END)
   surname_entry.delete(0, END)
```

Екінші батырма get әдісін пайдаланып, енгізілген мәтінді алады:

```
def display():
        messagebox.showinfo("GUI Python", name_entry.get() +
                                    " " +  surname_entry.get())
```

Мысалы, Entry мәтініне StringVar түріндегі айнымалылар арқылы хабарласу міндетті емес, біз мұны тікелей get әдісі арқылы жасай аламыз (сурет 3.17).

```
from tkinter import *
from tkinter import messagebox
def clear():
    name_entry.delete(0, END)
    surname_entry.delete(0, END)
def display():
    messagebox.showinfo("GUI Python", name_entry.get() + " " + surname_entry.get())
root = Tk()
root.title("GUI на Python")
name_label = Label(text="Введите имя:")
surname_label = Label(text="Введите фамилию:")
name_label.grid(row=0, column=0, sticky="w")
surname_label.grid(row=1, column=0, sticky="w")
name_entry = Entry()
surname_entry = Entry()
name_entry.grid(row=0,column=1, padx=5, pady=5)
surname_entry.grid(row=1,column=1, padx=5, pady=5)
# вставка начальных данных
name_entry.insert(0, "Tom")
surname_entry.insert(0, "Soyer")
display_button = Button(text="Display", command=display)
clear_button = Button(text="Clear", command=clear)
display_button.grid(row=2, column=0, padx=5, pady=5, sticky="e")
clear_button.grid(row=2, column=1, padx=5, pady=5, sticky="e")
root.mainloop()
```



Сурет 3.17. Entry виджетін қолдану

3.2.6 **Checkbutton** және **Radiobutton элементтері**

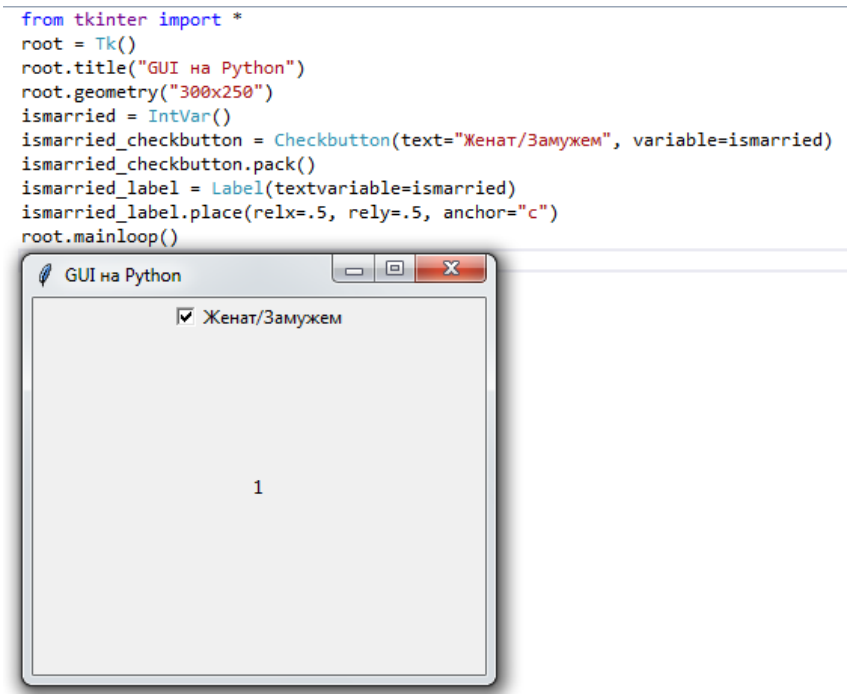
Checkbutton элементі.

Checkbutton элементі екі жағдайда болуы мүмкін құсбелгі болып табылады: белгіленген және белгіленбеген.

Қарапайым құсбелгіні жасаңыз [1, 8, 17-20]:

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
ismarried = IntVar()
ismarried_checkbutton = Checkbutton(text="Женат/Замужем", variable=ismarried)
ismarried_checkbutton.pack()
ismarried_label = Label(textvariable=ismarried)
ismarried_label.place(relx=.5, rely=.5, anchor="c")
root.mainloop()
```

Checkbutton ерекшелігі – variable параметрі арқылы IntVar компонентіне байланыстыру мүмкіндігі. Белгіленген күйде IntVar байланыстырылған компоненті 1, ал белгіленбеген компоненті 0 мәні бар. IntVar арқылы біз пайдаланушы көрсеткен мәнді ала аламыз (сурет 3.18).



Сурет 3.18. Checkbutton виджетін қолдану

Checkbutton конструкторы құсбелгілердің көрсетілуін теңшеуге болатын бірқатар параметрлерді қабылдайды:

Checkbutton(master, options)

Master параметрі ата-ана терезесіне сілтеме жасайды, ал options параметрі келесі параметрлер жиынтығын ұсынады [1, 8, 17-20]:

activebackground: фон түсі түрту күйіндегі құсбелгі;

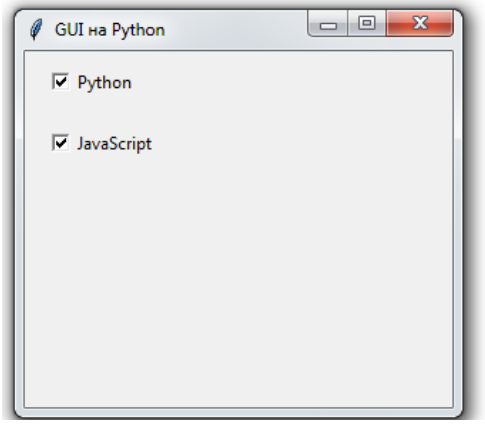
activeforeground: мәтін түсі түрту күйінде құсбелгі;

bg: фон түсі құсбелгі;
bitmap: жалаушаға арналған монохромды сурет;
bd: құсбелгінің айналасындағы шекара;
command: құсбелгіні басқан кезде туындайтын функцияға сілтеме;
cursor: элементке кіргенде курсор;
disabledforeground: DISABLED күйіндегі мәтін түсі;
font: қаріп;
fg: мәтін түсі;
height: Элемент биіктігі;
image: элементте көрсетілетін графикалық сурет;
justify: мәтінді туралау, CENTER, LEFT, RIGHT мәндерін қабылдайды;
offvalue: белгіленген күйдегі IntVar айнымалы құсбелгісімен байланысқан мәні, әдепкі 0 тең;
onvalue: белгіленген күйдегі IntVar айнымалы құсбелгісімен байланыстырылған мәні, әдепкі 1 тең;
padx: мәтіннің оң және сол жақ шегіністері құсбелгі шегіне дейін;
pady: мәтіннен жоғарғы және төменгі шегіністер құсбелгі шегіне дейін;
relief: құсбелгі стилі, әдепкі бойынша FLAT мәні бар;
selectcolor: түс шаршы жалаушасы;
selectimage: белгіленген күйде болған кезде құсбелгідегі сурет;
state: элемент күйі, NORMAL (әдепкі), DISABLED және ACTIVE;
text: элемент мәтіні;
underline: мәтіндегі асты сызылған таңбаның индексі құсбелгі;
variable: айнымалы сілтеме, әдетте, IntVar түрі, ол құсбелгі күйін сақтайды;
width: Элемент ені;
wraplength: элемент мәтініндегі басқа жолға таңбаларды тасымалдайды.

Осы параметрлердің кейбірін қолданамыз (сурет 3.19):

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
python_lang = IntVar()
python_checkbutton = Checkbutton(text="Python", variable=python_lang,
                                 onvalue=1, offvalue=0, padx=15, pady=10)
python_checkbutton.grid(row=0, column=0, sticky=W)
javascript_lang = IntVar()
javascript_checkbutton = Checkbutton(text="JavaScript", variable=javascript_lang,
                                     onvalue=1, offvalue=0, padx=15, pady=10)
javascript_checkbutton.grid(row=1, column=0, sticky=W)
root.mainloop()
```

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
python_lang = IntVar()
python_checkbutton = Checkbutton(text="Python", variable=python_lang,
                                 onvalue=1, offvalue=0, padx=15, pady=10)
python_checkbutton.grid(row=0, column=0, sticky=W)
javascript_lang = IntVar()
javascript_checkbutton = Checkbutton(text="JavaScript", variable=javascript_lang,
                                     onvalue=1, offvalue=0, padx=15, pady=10)
javascript_checkbutton.grid(row=1, column=0, sticky=W)
root.mainloop()
```



Сурет 3.19. Әртүрлі параметрлері бар Checkbutton виджетін қолдану

Radiobutton элементі.

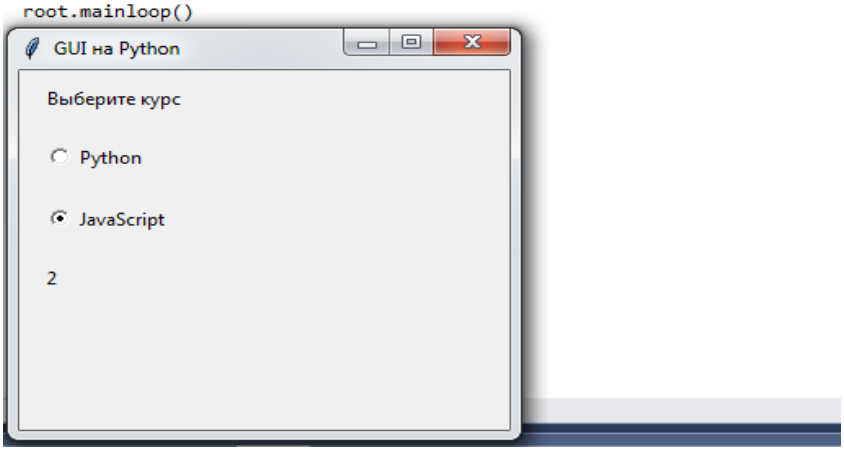
Radiobutton элементі екі жағдайда болуы мүмкін: белгіленген немесе белгіленбеген. Бірақ Checkbutton-ге қарағанда, қосқыштар бір ғана қосқышты таңдай алатын топты жасай алады.

Қосқыштарды қолданамыз (сурет 3.20):

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
header = Label(text="Выберите курс", padx=15, pady=10)
header.grid(row=0, column=0, sticky=W)
lang = IntVar()
python_checkbutton = Radiobutton(text="Python", value=1,
                                 variable=lang, padx=15, pady=10)
python_checkbutton.grid(row=1, column=0, sticky=W)
javascript_checkbutton = Radiobutton(text="JavaScript",
                                     value=2, variable=lang, padx=15,
                                     pady=10)
javascript_checkbutton.grid(row=2, column=0, sticky=W)
selection = Label(textvariable=lang, padx=15, pady=10)
selection.grid(row=3, column=0, sticky=W)
root.mainloop()
```

Мұнда екі қосқыш анықталды, бірақ екеуі де бір IntVar айнымалы. Сонымен қатар, олар value параметрі арқылы орнатылатын әр түрлі мәндерге ие. Сондықтан бір қосқышты қосқан кезде, екіншісі автоматты түрде өшірілмеген күйге ауысады.

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
header = Label(text="Выберите курс", padx=15, pady=10)
header.grid(row=0, column=0, sticky=W)
lang = IntVar()
python_checkbutton = Radiobutton(text="Python", value=1,
                                 variable=lang, padx=15, pady=10)
python_checkbutton.grid(row=1, column=0, sticky=W)
javascript_checkbutton = Radiobutton(text="JavaScript", value=2,
                                     variable=lang, padx=15, pady=10)
javascript_checkbutton.grid(row=2, column=0, sticky=W)
selection = Label(textvariable=lang, padx=15, pady=10)
selection.grid(row=3, column=0, sticky=W)
root.mainloop()
```



Сурет 3.20. Radiobutton виджетін қолдану

Басқа виджет конструкторлары сияқты Radiobutton конструкторы қосқышын баптау үшін екі параметрді қабылдайды:

Radiobutton (master, options)

Бірінші параметр – master ата-ана терезесіне сілтеме береді, ал екінші параметр келесі параметрлердің жиынтығын біріктіреді [1, 8, 17-20]:

activebackground: түрту күйіндегі қосқыштың фондық түсі;
activeforeground: түрту күйінде қосқыш мәтінінің түсі;
bg: қосқыштың фондық түсі;
bitmap: қосқыш үшін монохромды сурет;
borderwidth: қосқыштың айналасындағы шекара;
command: қосқышты басқан кезде туындайтын функцияға сілтеме;
cursor: элементке кіргенде курсор;
font: қаріп;
fg: мәтін түсі;
height: мәтін жолағындағы элементтің биіктігі. Әдепкі бойынша 1;
image: элементте көрсетілетін графикалық сурет;
justify: мәтінді туралау, CENTER, LEFT, RIGHT мәндерін қабылдайды;
padx: мәтіннің оң және сол жағында қосқыш шегіне дейінгі шегіністер;
pady: мәтіннен қосқыштың шегіне дейінгі жоғарғы және төменгі шегіністер;
relief: қосқыш стилі, әдепкі бойынша FLAT мәні бар;
selectcolor: қосқыш кружка түсі;
selectimage: ол белгіленген күйде болған кезде қосқышта сурет;
state: элемент күйі, NORMAL (әдепкі), DISABLED және ACTIVE;
text: элемент мәтіні;

textvariable: қосқыш мәтінін көрсететін StringVar айнымалысына байланыстыруды орнатады;

underline: элемент мәтініндегі астыңғы таңба индексі;

variable: айнымалы сілтеме, әдетте, IntVar түрі, ол ауыстырып-қосқыштың күйін сақтайды;

мән: қосқыштың мәні;

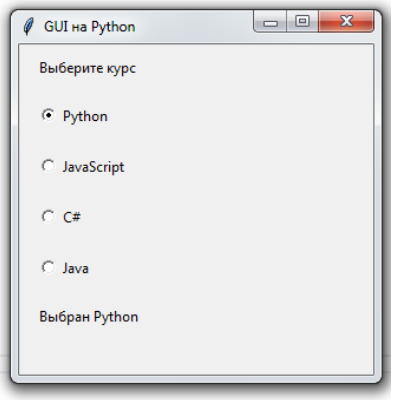
width: Элемент ені;

wraplength: элемент мәтініндегі басқа жолға таңбаларды тасымалдайды.

Пайдаланушы таңдауын өңдеу арқылы күрделі мысалда осы параметрлердің қатарын қолданамыз (сурет 3.21):

```
from tkinter import *
languages = [("Python", 1), ("JavaScript", 2), ("C#", 3),
                ("Java", 4)]
def select():
   l = language.get()
   if l == 1:
      sel.config(text="Выбран Python")
   elif l == 2:
      sel.config(text="Выбран JavaScript")
   elif l == 3:
      sel.config(text="Выбран C#")
   elif l == 4:
      sel.config(text="Выбран Java")
root = Tk()
root.title("GUI на Python")
root.geometry("300x280")
header = Label(text="Выберите курс", padx=15, pady=10)
header.grid(row=0, column=0, sticky=W)
language = IntVar()
row = 1
for txt, val in languages:
   Radiobutton(text=txt, value=val, variable=language,
                        padx=15, pady=10, command=select)\
            .grid(row=row, sticky=W)
   row += 1
sel = Label(padx=15, pady=10)
sel.grid(row=row, sticky=W)
root.mainloop()
```

```
from tkinter import *
languages = [("Python", 1), ("JavaScript", 2), ("C#", 3), ("Java", 4)]
def select():
    l = language.get()
    if l == 1:
        sel.config(text="Выбран Python")
    elif l == 2:
        sel.config(text="Выбран JavaScript")
    elif l == 3:
        sel.config(text="Выбран C#")
    elif l == 4:
        sel.config(text="Выбран Java")
root = Tk()
root.title("GUI на Python")
root.geometry("300x280")
header = Label(text="Выберите курс", padx=15, pady=10)
header.grid(row=0, column=0, sticky=W)
language = IntVar()
row = 1
for txt, val in languages:
    Radiobutton(text=txt, value=val, variable=language,
                padx=15, pady=10, command=select)\
        .grid(row=row, sticky=W)
    row += 1
sel = Label(padx=15, pady=10)
sel.grid(row=row, sticky=W)
root.mainloop()
```



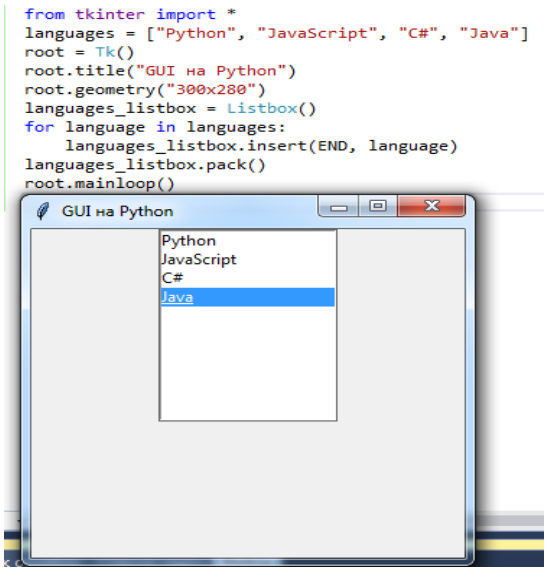
Сурет 3.21. Әртүрлі параметрлері бар Radiobutton виджетін қолдану

3.2.7 Listbox және Меню элементтері

Listbox элементі

Tkinter-дегі Listbox элементі нысандар тізімін ұсынады. Қарапайым тізімді анықтайық (сурет 3.22):

```
from tkinter import *
languages = ["Python", "JavaScript", "C#", "Java"]
root = Tk()
root.title("GUI на Python")
root.geometry("300x280")
languages_listbox = Listbox()
for language in languages:
    languages_listbox.insert(END, language)
languages_listbox.pack()
root.mainloop()
```



Сурет 3.22. Listbox виджетін қолдану

For циклындағы listboxa элементтерді толтыру үшін languages тізімінің барлық элементтеріне өтіңіз және insert() әдісі арқылы listbox әрбір элементті қосыңыз. Бірінші дәлел ретінде insert элементті кірістіру индексі беріледі. Бірақ біз элементтерді дәйекті қосқымыз келсе, онда индекстің орнына END мәнін пайдалануға болады.

Listbox виджетін теңшеу үшін, оның конструкторында келесі параметрлерді көрсете аламыз [1, 8, 17-20]:

bg: фон түсі;

bd: элемент шекарасының қалыңдығы;

cursor: ListBox-қа апару кезіндегі курсор;

font: қаріп параметрлері;

fg: мәтін түсі;

height: жолдың элементінің биіктігі. Әдепкі бойынша 10 жолды көрсетеді;

highlightcolor: ол фокус алған кезде элемент түсі;

highlightthickness: ол фокуста болған кезде элемент шегінің қалыңдығы;

relief: элемент мәнерін орнатады, әдепкі бойынша SUNKEN мәні бар;

selectbackground: таңдалған элемент үшін фон түсі;

selectmode: қанша элементтің бөлінуі мүмкін екенін анықтайды. Келесі мәндерді қабылдай алады: BROWSE, SINGLE, MULTIPLE, EXTENDED. Мысалы, көпше элементтерді қосу қажет болса, MULTIPLE немесе EXTENDED мәндерін пайдалануға болады.

width: таңбалардағы элементтің енін орнатады. Әдепкі ені-20 таңба;

xscrollcommand: көлденең жылжуды көрсетеді;

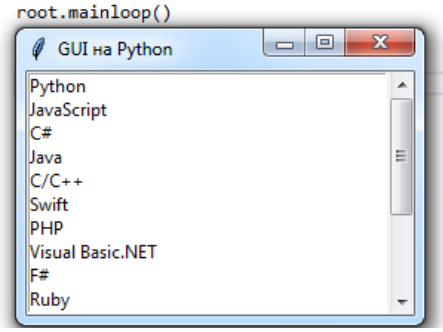
yscrollcommand: тік жылжуды орнатады.

Listbox пайдалану кезінде кейбір қиындық жылжыту жасау болып табылады. Мұны қалай қарастырайық (сурет 3.23).

Айналдыру үшін Scrollbar элементі жасалады. Және ListBox тігінен айналдыру қажет болғандықтан, ол yscrollcommand=scrollbar параметрін scrollbar.set

Scrollbar соңында айналдыру кезінде орындалатын функциямен байланысты. Бұл жағдайда бұл listbox элементінің yview әдісі. Нәтижесінде біз элементтерді тігінен айналдыра аламыз.

```
from tkinter import *
languages = ["Python", "JavaScript", "C#", "Java", "C/C++", "Swift","PHP",
             "Visual Basic.NET", "F#", "Ruby", "Rust", "R", "Go", "T-SQL",
             "PL-SQL", "Typescript"]
root = Tk()
root.title("GUI на Python")
scrollbar = Scrollbar(root)
scrollbar.pack(side=RIGHT, fill=Y)
languages_listbox = Listbox(yscrollcommand=scrollbar.set, width=40)
for language in languages:
    languages_listbox.insert(END, language)
languages_listbox.pack(side=LEFT, fill=BOTH)
scrollbar.config(command=languages_listbox.yview)
root.mainloop()
```



Сурет 3.23. Айналдыру ListBox элементі

Listbox негізгі әдістері.

Listbox элементтің мінез-құлқын және оның мазмұнын басқару үшін бірқатар әдістері бар. Олардың кейбірі:

– curselection(): таңдалған элементтердің индекстер жиынтығын қайтарады;

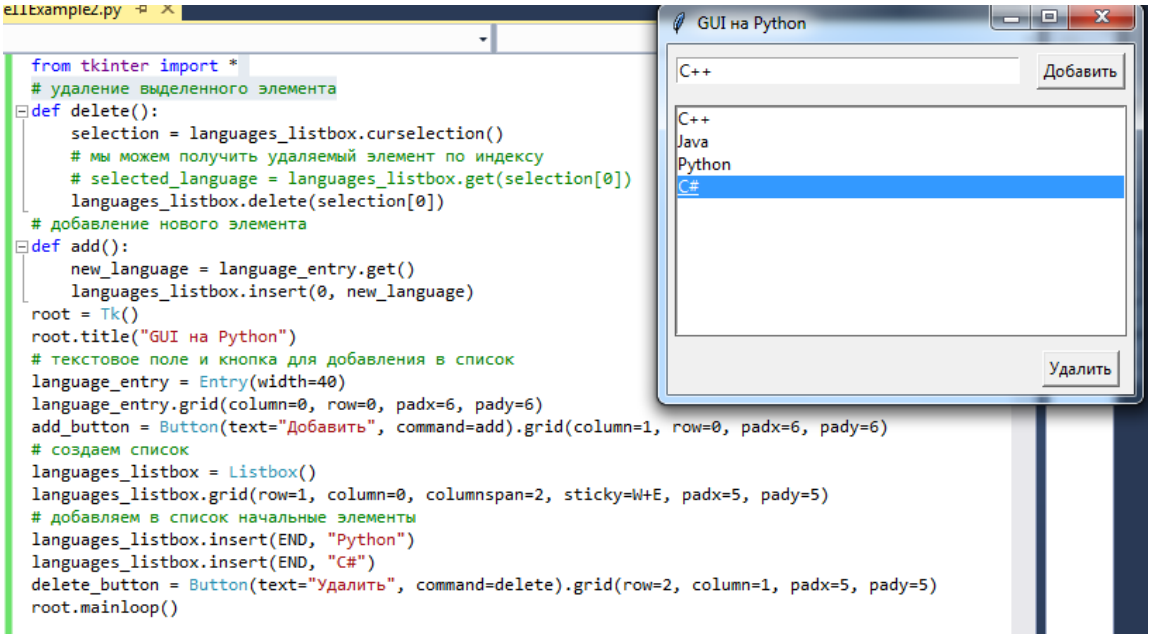
– delete(first, last = None): [first, last] ауқымынан индекстер элементтерін жояды. Егер екінші параметр төмен болса, first индексі бойынша тек бір элементті жояды;

– get(first, last = None): [first, last] индекстері бар элементтердің мәтіні бар кортеж қайтарады. Егер екінші параметр төмен болса, тек first индексі бар элементтің мәтіні қайтарылады;

– insert(index, element): белгілі бір индекс бойынша элементті кірістіреді;

– size(): элементтер санын қайтарады.

Осы әдістерді қарау үшін біз данны басқару бойынша шағын сценарийді жазамыз (сурет 3.24).




```
from tkinter import *
# удаление выделенного элемента
def delete():
    selection = languages_listbox.curselection()
    # мы можем получить удаляемый элемент по индексу
    # selected_language = languages_listbox.get(selection[0])
    languages_listbox.delete(selection[0])
# добавление нового элемента
def add():
    new_language = language_entry.get()
    languages_listbox.insert(0, new_language)
root = Tk()
root.title("GUI на Python")
# текстовое поле и кнопка для добавления в список
language_entry = Entry(width=40)
language_entry.grid(column=0, row=0, padx=6, pady=6)
add_button = Button(text="Добавить", command=add).grid(column=1, row=0, padx=6, pady=6)
# создаем список
languages_listbox = Listbox()
languages_listbox.grid(row=1, column=0, columnspan=2, sticky=W+E, padx=5, pady=5)
# добавляем в список начальные элементы
languages_listbox.insert(END, "Python")
languages_listbox.insert(END, "C#")
delete_button = Button(text="Удалить", command=delete).grid(row=2, column=1, padx=5, pady=5)
root.mainloop()
```

Сурет 3.24. Әртүрлі әдістермен Listbox виджетін қолдану

Тізім элементін манипуляциялау үшін екі батырма бар. Бірінші батырма мәтіндік өріске енгізілген мәнді алатын add() функциясын шақырады және insert() әдісі арқылы тізімдегі бірінші орынға қосады.

Екінші батырма таңдалған элементті жояды. Ол үшін алдымен curselection() әдісі арқылы таңдалған индекстерді аламыз. Біздің жағдайда тек бір элемент бөлінгендіктен, оның индексін selection[0] өрнегі арқылы аламыз. Және бұл индексті жою үшін delete() әдісіне жібереміз.

Меню.

Tkinter және Python иерархиялық мәзірді жасау үшін Menu виджеті қолданылады.

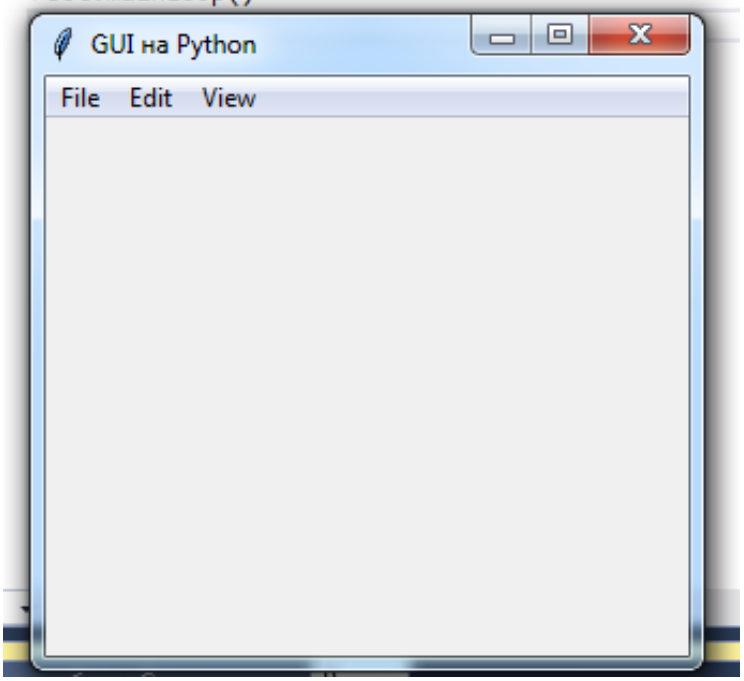
Мәзірде көптеген элементтер болуы мүмкін және бұл элементтер өздері мәзірді көрсетіп, басқа элементтерді қамтуы мүмкін. Мәзірге қосқымыз келетін элементтердің қандай түріне байланысты, оларды қосу үшін қолданылатын әдіс өзгеше болады. Атап айтқанда, келесі әдістер қол жетімді:

– add_command(options): мәзір элементін options параметрі арқылы қосады;

– add_cascade(options): мәзір элементін қосады, ол өз кезегінде ішкі мәзірді көрсетеді;
– add_separator(): шектеу сызығын қосады;
– add_radiobutton(options): мәзірге құсбелгіні қосады;
– add_checkbutton(options): мәзірге құсбелгі қосады.

Қарапайым мәзір жасаңыз (сурет 3.25):

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
main_menu = Menu()
main_menu.add_cascade(label="File")
main_menu.add_cascade(label="Edit")
main_menu.add_cascade(label="View")
root.config(menu=main_menu)
root.mainloop()
```



Сурет 3.25. Menu виджетін қолдану

Menu нысанындағы мәзір пункттерін қосу үшін add_cascade() әдісі шақырылады. Бұл әдіс мәзір тармағының параметрлері беріледі, бұл жағдайда олар label параметрі арқылы орнатылатын мәтіндік белгімен беріледі.

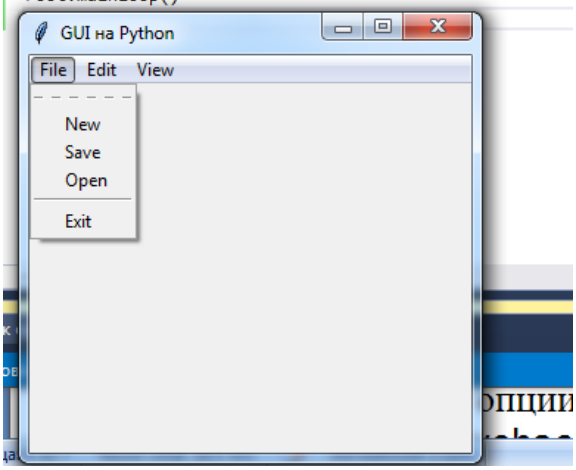
Бірақ жай ғана мәзір жасау жеткіліксіз. Оны ағымдағы терезе үшін MENU параметрімен config() әдісі арқылы орнату керек. Соңында графикалық терезе келесі мәзірге ие болады (сурет 3.25).

Енді мәзірді қосамыз (сурет 3.26).

Мұнда menu=file_menu параметрін орнату арқылы негізгі мәзірдің бірінші тармағына қосылатын file_menu ішкі мәзірі анықталады:

main_menu.add_cascade(label="File", menu=file_menu)

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
main_menu = Menu()
file_menu = Menu()
file_menu.add_command(label="New")
file_menu.add_command(label="Save")
file_menu.add_command(label="Open")
file_menu.add_separator()
file_menu.add_command(label="Exit")
main_menu.add_cascade(label="File", menu=file_menu)
main_menu.add_cascade(label="Edit")
main_menu.add_cascade(label="View")
root.config(menu=main_menu)
root.mainloop()
```



Сурет 3.26. Мәзірмен Menu виджетін қолдану

Мәзірді орнату қажет болса, Menu конструкторында келесі опцияларды орнатуға болады:

activebackground: мәзірдің белсенді тармағының түсі;

activeborderwidth: мәзірдің белсенді тармағы шегінің қалыңдығы;

activeforeground: белсенді мәзір орны мәтінінің түсі;

bg: фон түсі;

bd: шекара қалыңдығы;

cursor: мәзірге апарғанда тышқан көрсеткіш меңзері;

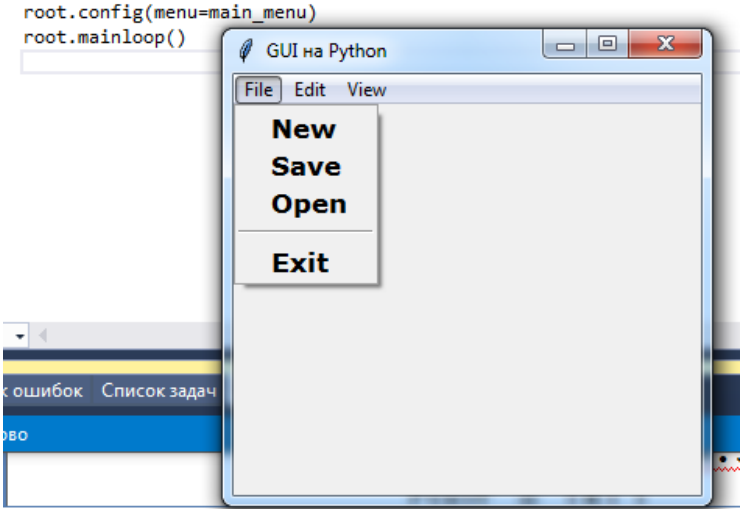
disabledforeground: мәзірі DISABLED күйіндегі түс;

font: мәтін қаріпі;

fg: мәтін түсі;

tearoff: мәзір графикалық терезеден ажыратылуы мүмкін. Атап айтқанда, А скриншотаны құру кезінде, оны ажыратуға болатын, жоғарғы мәзірдегі үзілген сызықты көруге болады. Дегенмен, tearoff=0 мәнінде ішкі мәзір ажыратыла алмайды (сурет 3.27).

```
from tkinter import *
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
main_menu = Menu()
file_menu = Menu(font=("Verdana", 13, "bold"), tearoff=0)
file_menu.add_command(label="New")
file_menu.add_command(label="Save")
file_menu.add_command(label="Open")
file_menu.add_separator()
file_menu.add_command(label="Exit")
main_menu.add_cascade(label="File", menu=file_menu)
main_menu.add_cascade(label="Edit")
main_menu.add_cascade(label="View")
root.config(menu=main_menu)
root.mainloop()
```



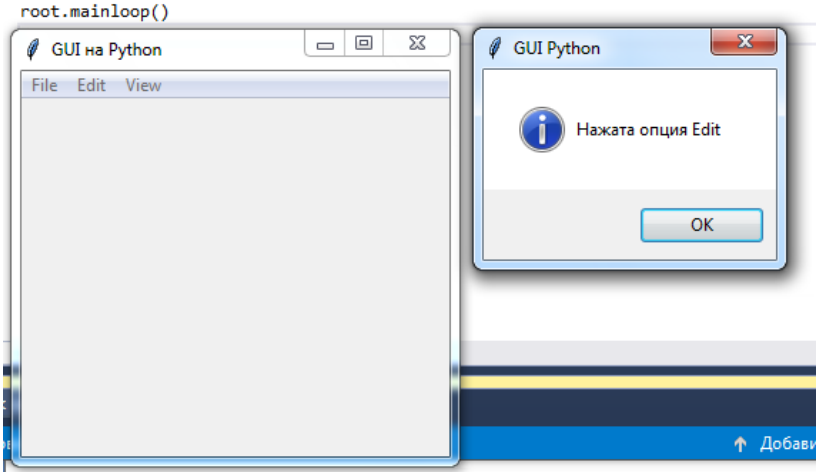
Сурет 3.27. Мәзірі бар Menu виджеті

Мәзірмен өзара әрекеттесу.

Мәзір элементтерінің ерекше ерекшелігі пайдаланушының басуына жауап беру қабілеті болып табылады. Ол үшін мәзірдің әр элементінде басқанда орындалатын функцияға сілтеме жасайтын command параметрін орнатуға болады (сурет 3.28).

```
from tkinter import *
from tkinter import messagebox
def edit_click():
    messagebox.showinfo("GUI Python", "Нажата опция Edit")
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
main_menu = Menu()
main_menu.add_cascade(label="File")
main_menu.add_cascade(label="Edit", command=edit_click)
main_menu.add_cascade(label="View")
root.config(menu=main_menu)
root.mainloop()
```

```
from tkinter import *
from tkinter import messagebox
def edit_click():
    messagebox.showinfo("GUI Python", "Нажата опция Edit")
root = Tk()
root.title("GUI на Python")
root.geometry("300x250")
main_menu = Menu()
main_menu.add_cascade(label="File")
main_menu.add_cascade(label="Edit", command=edit_click)
main_menu.add_cascade(label="View")
root.config(menu=main_menu)
root.mainloop()
```



Сурет 3.27. Мәзірмен өзара әрекеттесу

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ

1. Ким Е.Р. Программирование на Python в среде Visual Studio 2017: Уч.-мет. пос. – Алматы: КазАТК, 2019. – 140 с.
2. Гэддис Т. Начинаем программировать на Python. – 4-е изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2019. – 768 с..
3. Доусон М. Программируем на Python. – СПб.: Питер , 2014. – 400 с.
4. Лутц М. Изучаем Python (4-е издание). – СПб.: Символ Плюс, 2011. – 848 с.
5. Прохорёнок Н.А. Python 3 и PyQt. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – 704 с.
6. Саммерфилд М. Программирование на Python 3. Подробное руководство. – СПб.: Символ Плюс, 2009. – 608 с.
7. Сэнд У., Сэнд К. Hello World! Занимательное программирование. – СПб.: Питер, 2016. – 400 с.
8. Хахаев И.А. Практикум по алгоритмизации и программированию на Python: / И.А. Хахаев – М. : Альт Линукс, 2010. – 126с.
9. Онлайн-школа Фоксфорд. Официальный сайт. – URL: https://foxford.ru/wiki/informatika/ekranirovannye-simvoly-v-python/.
10. Программирование на Python. – URL: https://pythoner.name/.
11. Сузи Р. Язык программирования Python. Курс лекций на сайте НОУ ИНТУИТ – URL: https://www.intuit.ru/studies/courses/49/49/info.
12. Язык программирования Python. Официальный сайт. – URL: https://www.python.org/.
13. Python 3.0 для начинающих. – URL: https://pythonworld.ru/.
14. Python для новичков и продолжающих начинающих. – URL: https://python.ru/.
15. Python уроки. – URL: https://bunkerbook.ru.
16. Бизли Д. Python. Подробный справочник. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 864 с.
17. METANIT.COM. – URL: https://metanit.com.
18. Хахаев И.А. Практикум по алгоритмизации и программированию на Python: / И.А. Хахаев – М. : Альт Линукс, 2010. – 126с.
19. Масленникова Н. А. Сборник задач по программированию. – URL: http://shkolnie.ru/informatika/1112/index.html?page=12/.
20. Решение задач по программированию. Языки программирования Pascal, Basic, КуМир, C, Python. – URL: https://taskcode.ru/.
21. Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию. – М.: Наука, 1988.
22. Белоцерковская И.Е., Галина Н.В., Катаева Л.Ю. Алгоритмизация. Введение в язык программирования C++ : учеб. пособие. – 2-е изд. М.: ИНТУИТ, 2016. – 196 с.
23. Гуденко Д., Петроченко Д. Сборник задач по программированию. – СПб.: Питер, 2003.
24. Кнут Д. Искусство программирования для ЭВМ. Т.1.-Т.3. Основные алгоритмы. Получисленные алгоритмы. Сортировка и поиск. – М.: Мир, 1976-1978.
25. Семакин И.Г., Шестаков А.П. Основы алгоритмизации и программирования: учебник. – 3-е изд. – М.: Издательский центр «Академия», 2012. – 400 с.
26. Стивен С. Скиена. Алгоритмы. Руководство по программированию. – СПб.: БХВ-Петербург, 2011. – 700 с.

МАЗМҰНЫ

Жазбалар үшін

Ким Е.Р., Сыдыбаева М.А., Молдакалыкова Б.Ж.

Алгоритмдер, деректер құрылымы және Python тілінде бағдарламалау

Оқу-әдістемелік құрал

Пішімі 60х84, 1/16.
Тығыздығы 80 г/м^2.
Көлемі 6.51