Х.Ш.МУСАЕВ, А.М.ҚАЮМОВ

PYTHON
ДАСТУРЛАШ ТИЛИ

python

ЎЗБЕКИСТОН РЕСПУБЛИКАСИ АХБОРОТ
ТЕХНОЛОГИЯЛАРИ ВА КОММУНИКАЦИЯЛАРИНИ
РИВОЖЛАНТИРИШ ВАЗИРЛИГИ

МУҲАММАД АЛ-ХОРАЗМИЙ НОМИДАГИ
ТОШКЕНТ АХБОРОТ ТЕХНОЛОГИЯЛАРИ УНИВЕРСИТЕТИ

Х.Ш.МУСАЕВ, А.М.ҚАЮМОВ

PYTHON ДАСТУРЛАШ ТИЛИ

(Ўқув қўлланма)

ТОШКЕНТ – 2020

УДК: 004.43(075.8)
ББК: 32.973.26-018.1

Х.Ш.Мусаев, А.М.Қаюмов. Python дастурлаш тили. (Ўқув қўлланма). – Т.: «Aloqachi», 2020. – 144 с.
ISBN 978–9943–5807–5–6

Ушбу ўқув қўлланма ҳозирги кундаги замонавий дастурлаш тилларидан бири бўлган Python дастурлаш тилига бағишланган. Python дастурлаш тили энг машҳур дастурлаш тилларидан бири ҳисобланади. Python дастурлаш тили машҳурлигига асосий сабаблардан бири бу веб иловалар ва веб сайтлар ярата олиш қобилияти ҳисобланади. Python дастурлаш тилига асосланган ҳолда Django фреймворки ишлаб чиқилган. Python дастурлаш тили ёрдамида сунъий ақл билан боғлиқ турли дастурлар яратиш мумкин

Python дастурлаш тили платформа танламайдиган тиллар сирасига киради. Python дастурлаш тилида ёзилган буйруқлар кетма-кетлиги Windows операцион тизимида, Linux операцион тизимида ёки MacOS каби машҳур операцион тизимларда ишлаши мумкин.

Python дастурлаш тили юқори даражали интерактив ва объектга йўналтирилган дастурлаш тилларидан бири ҳисобланади. Ушбу қўлланма Python дастурлаш тилини бошловчилари учун кенг қўлланиладиган содда маълумотлардан ташкил топган.

УДК: 004.43(075.8)
ББК: 32.973.26-018.1

Такризчилар – Д.С.Тухтаназаров, Муҳаммад ал–Хоразмий номидаги ТАТУ "АТДТ" кафедраси доценти, PhD.

ISBN 978–9943–5807–5–6

I. Python дастурлаш тилига кириш

1.1. Python дастурлаш тили

Python дастурлаш тили оммабоп, юқори самарадорликка эга дастурлаш тили ҳисобланади ва у турли хилдаги қизиқарли дастурларни яратиш имконини беради. **Python** дастурлаш тили ёрдамида веб иловалар, ўйинлар, амалий дастурлаш ва маълумотлар базаси устида турли амаллар бажарувчи дастурлар яратиш мумкин. **Python** дастурлаш тили бошқа дастурлаш тилларига нисбатан сунъий интеллект соҳасида сезиларли даражада кенг имкониятга эга.

Дастлаб **Python** дастурлаш тили 1991 йилда Голландиялик дастурчи Гвидо ван Россум томонидан яратилган. Дастурлаш тили яратилганидан буён ривожланиб келмоқда. Ривожланишлар ўзининг самарасини 2000 йилга келиб берди ва дастурлаш тилининг 2.0 версияси оммага тақдим этилди. 2008 йилда эса 3.0 версияси яратилди. Версиялар орасидаги бунчалик узоқ вақт узулишларга қарамай, Pythonнинг янги версиялари чиқишда давом этмоқда. Ҳозирги вақтда дастурлаш тилининг 3.7 версиясидан бутун жаҳонда фойдаланиб келинмоқда.

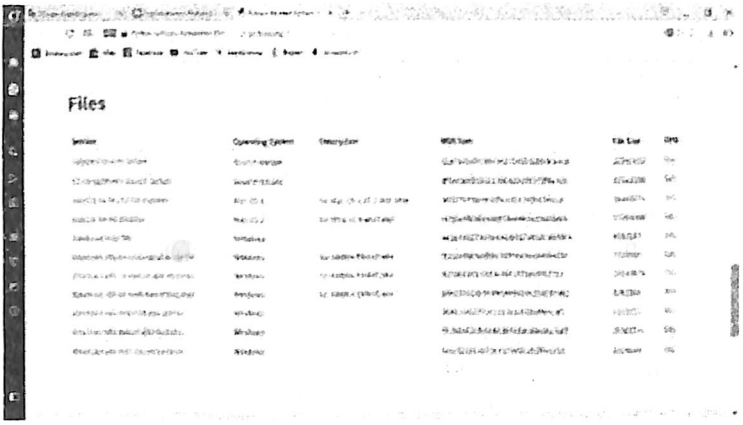
Python дастурлаш тили жуда содда дастурлаш тили ҳисобланади. Шу билан бирга бу дастурлаш тилини ўрганиш осонлиги билан ажралиб туради, бу хусусияти дастурлаш тилини тобора кенг тарқалишига сабаб бўлмоқда ва жуда кўп мамлакатларда кенг ўрганилмоқда. **Python** дастурлаш тили 2014 йилда АҚШда йилнинг энг машҳур (оммабоп) дастурлаш тили деб эътироф этилган. Бу дастурлаш тили нафақат кенг ўрганилмоқда, шу билан бирга савдо соҳасида ҳам олдинлаб бормоқда.

1.2. Python дастурлаш тилини ўрнатиш

Python дастурлаш тилида дастур яратиш учун бизга интерпретатор керак бўлади. Бунинг учун **https://www.python.org/** сайтига кириб, тилнинг энг сўнгги версиясини компьютер хотирасига юклаб оламиз.

1.1–расм: Python дастурлаш тилининг интерпретаторини олиш

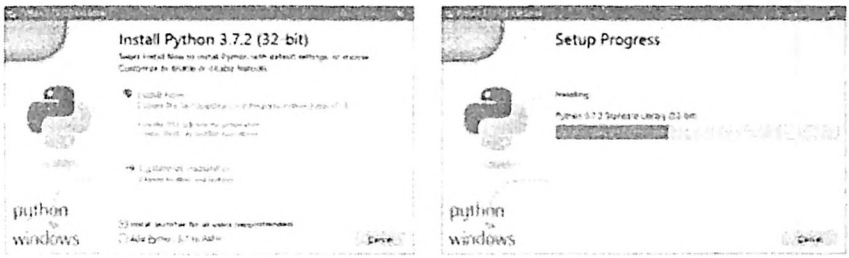
Ойнадаги **Download** бўлими орқали дастурлаш тилининг охирги версияси компьютер хотирасига юкланади. У ерда турли хилдаги операцион тизимлар учун дастурлаш тилининг транслятори мавжуд. Керак бўлган трансляторни компьютер хотирасига юкланади. Юкланган файлни компьютерга ўрнатилади.



1.2–расм: Операцион тизимлар учун дастурлаш тилининг кўринишлари

Windows операцион тизимида ишлаётган фойдаланувчилар учун ўрнатиш қуйидагича амалга оширилади:

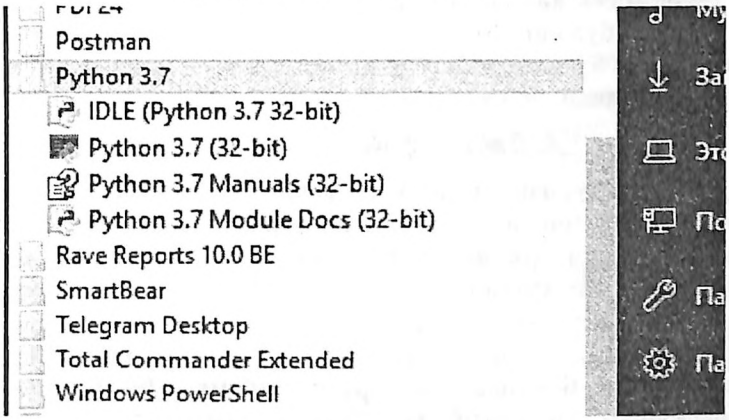
Юклаб олинган файлнинг устида сичқончанинг чап тугмасини икки марта чертилади. Ҳосил бўлган ойнадаги Install Now бўлими танланади.



1.3–расм: Python 3.7.2 ўрнатиш ойнаси

Ойнанинг остки қисмида жойлашган “Add Python 3.7 to PATH” бандига байроқча ўрнатиш керак. Дастур ўрнатилгандан сўнг Пуск менюси ёрдамида дастурлаш тилининг турли муҳитларига кириш мумкин.

Пуск → Всё программы → Python 3.7 → Python 3.7 (32-bit)



1.4–расм: Дастурлаш тилининг муҳитини ишга тушириш

1.3. Дастурлаш тили ёрдамида Hello World дастурини яратиш

Дастур ўрнатилганидан сўнг, дастурлаш муҳитини ишга тушириш керак бўлади (олдинги мавзуда ишга тушириш ҳақида маълумот берилган). Дастур ишга тушганда қуйидаги ойна ҳосил бўлади.

1.5–расм: Дастурлаш муҳитининг консол ойнаси

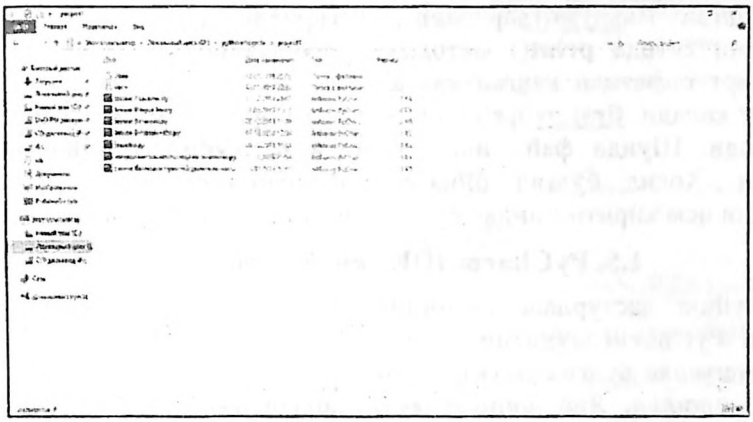
Ойнага қуйидаги кодни ёзамиз:

```
print("Hello World");
```

Бу кодни ёзиб <Enter> тугмасини босиш билан консол ёзилган кодни трансляция қилишни бошлайди ва консол ойнасида "Hello World" ҳосил бўлади. Бу содда дастур учун **print()** методидан фойдаланилди. Бу метод берилган маълумотни экранга чиқариш учун хизмат қилади.

1.4. Дастур файлини яратиш

Python дастурлаш тилининг файлини яратиш учун IDLE (Python 3.7) файлини ишга тушириш керак бўлади. Бу файл ҳам Python дастурлаш тилининг яна бир муҳити ҳисобланиб, у ёрдамида модуллар яратиш ва уларни алоҳида - алоҳида файлларда сақлаш имконияти яратилади. Барча дастурлаш тиллари каби Python дастурлаш тилининг ҳам алоҳида кенгайтмаси мавжуд. Бу кенгайтма ***.py** деб номланади. ***.py** кенгайтмаси билан сақланган файллар Python дастурлаш тилининг кодлари ёки модули деб юритилади.

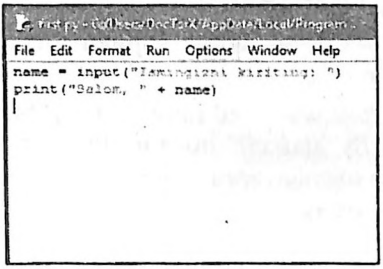


1.6 – расм: Python дастурлаш тилининг модуллари

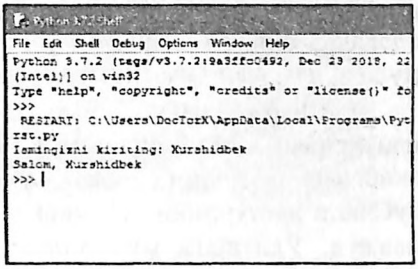
Python дастурлаш тилида ёзилган кодларни исталган матн муҳаррири ёрдамида очиб таҳрирлаш мумкин. Python дастурлаш муҳитида дастур ишга туширилганидан сўнг фойдаланувчидан исмини киритишни сўрасин ва унга дастур ишлагани ҳақида ахборот берсин.

Бундай дастур тузиш учун IDLE муҳитида Ctrl+N тугмаларни босамиз. Бу тугмалар бирикмаси янги модул ҳосил қилиш учун ойна яратади. Ҳосил бўлган ойнага қуйидаги кодлар кетма-келтилиги ёзилади.

```
name = input("Ismingizni kiriting:")
print("Salom, " + name)
```



1.7–расм: first.py файли



1.8–расм: Модул ишга туширилгандаги натижа

Дастурнинг коди икки сатрдан иборат бўлиб, биринчи сатрида **input()** методидан фойдаланилган. Бу метод маълумотларни клавиатурадан киритиш учун қўлланилади. Бу метод ёрдамида

киритилган маълумотлар **name** ўзгарувчисига ўзлаштирилади. Иккинчи сатрда **print()** методидан фойдаланилган, ушбу метод парамерт сифатида келган катталикларни экранга чиқариш учун хизмат қилади. **first.py** файлини ишга тушириш учун **F5** тугмасини босилади. Шунда файл ишга тушади ва қуйидаги ойна ҳосил бўлади. Ҳосил бўлган ойнада фойдаланувчи исми сўралади. Керакли исм киритилгандан сўнг натижани кўриш мумкин.

1.5. PyCharm IDE дан фойдаланиш

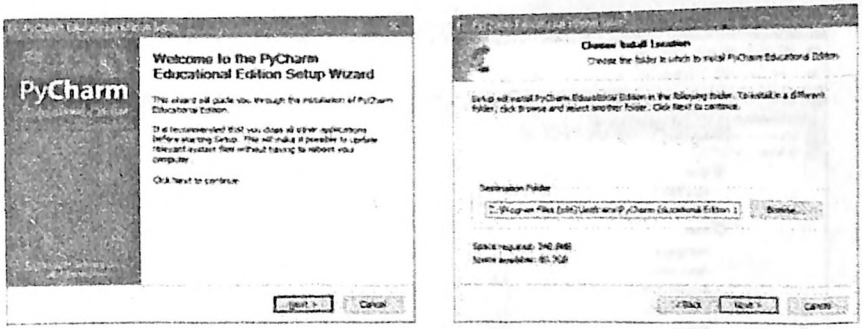
Python дастурлаш тилининг яна бир муҳитларидан бири бўлган **PyCharm** муҳитини кўриб чиқамиз. Дастурлаш тилининг ўзига тегишли бўлган муҳитларидан фойдаланиб модуллар яратиш кўриб чиқилди. Яна шундай муҳитлардан бири бўлган PyCharm муҳити билан танишамиз.

PyCharm IDE модуллар яратишда катта имкониятларга эга бўлган муҳитдир. PyCharm IDE модуллар яратишда мисоллардаги хатоликларини автоматик тарзда аниқлайди ва хатолик ҳақида маълумотларни ўз вақтида турли рангларда дастурчига кўрсатиб беради.

PyCharm IDE энг таниқли бўлган JetBrains компанияси томонидан яратилган. Ушбу муҳит бир хилда ривожланиб бормоқда ва доимо янгиланиб келмоқда. Бу муҳит ишлаб чиқарувчилари барча операцион тизимлар учун версияларни тақдим этган.

Бу муҳит ишлаб чиқарувчилар томонидан икки хил версияда тақдим этилган, яъни пуллик ва бепул. Пуллик вариантда веб иловалар билан ишлаш имконияти мавжуд. Бепул вариантида ҳам жуда кўп базавий имкониятлар ишлатилади.

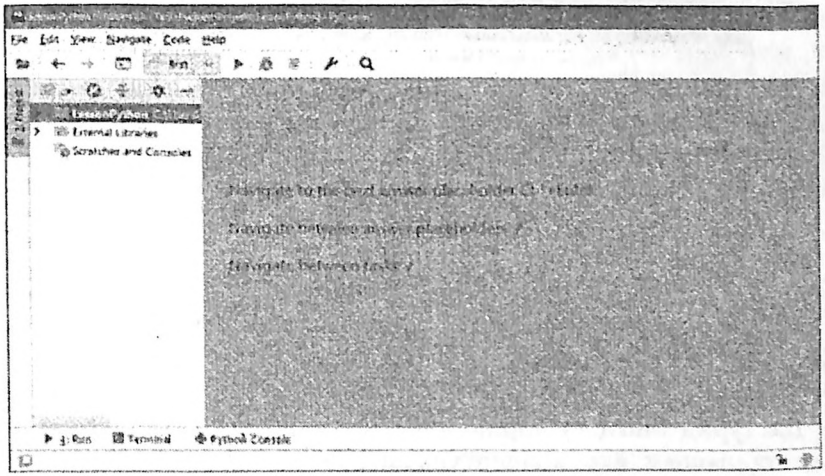
PyCharm IDE ни https://www.jetbrains.com/pycharm/ манзилидан юклаб олиш мумкин. Бу манзил орқали PyCharmни охирги версиясини юклаб, уни компьютерга ўрнатиш лозим. PyCharm дастурининг ўқитиш учун мўлжалланган версиялари ҳам мавжуд. Ўқитишга мўлжалланган версияси PyCharm Educational Edition деб номланади. Бу дастурни ўрнатиш кетма-кетлиги қуйида келтирилган:



1.9–расм: PyCharm Educational Edition муҳитини ўрнатиш

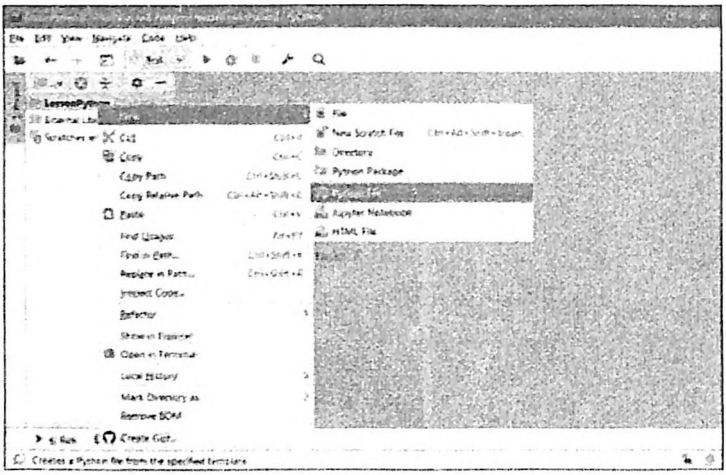
Янги лойиҳа яратиш учун Create New Project танланади. Сўнгра созлаш ойнаси ҳосил бўлади. У ойнада лойиҳа жойлашадиган йўлни кўрсатиш ва интерпретатор файлига йўл кўрсатиш керак бўлади.

Янги лойиҳанинг номини HelloApp деб номланади. Барча созлашлар тугатилганидан сўнг Create тугмаси босилади. Лойиҳанинг номи билан папка яратиб, унинг ичига лойиҳанинг файллари яратилади. Шунда бўш лойиҳа ҳосил бўлади:



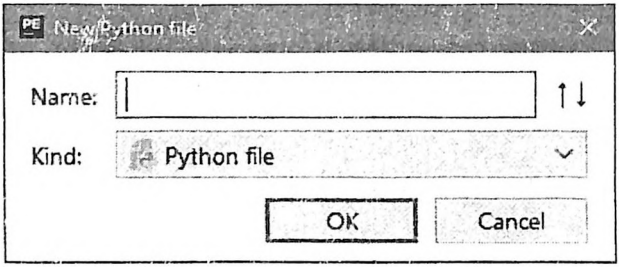
1.10–расм: PyCharm дастури ёрдамида лойиҳа яратиш

Лойиҳанинг устида сичқончанинг ўнг тугмасини чертиб янги файл яратилади. Бунинг учун ҳосил бўлган контекст менюдан **New** банди танланади ва ундан **Python File** банди танланади.



1.11–расм: PyCharm EE да янги файл яратиш

Бандлар танланганидан сўнг янги файлнинг номини киритиш учун ойна ҳосил бўлади. Файл номини масалан Hello деб номлаш мумкин.

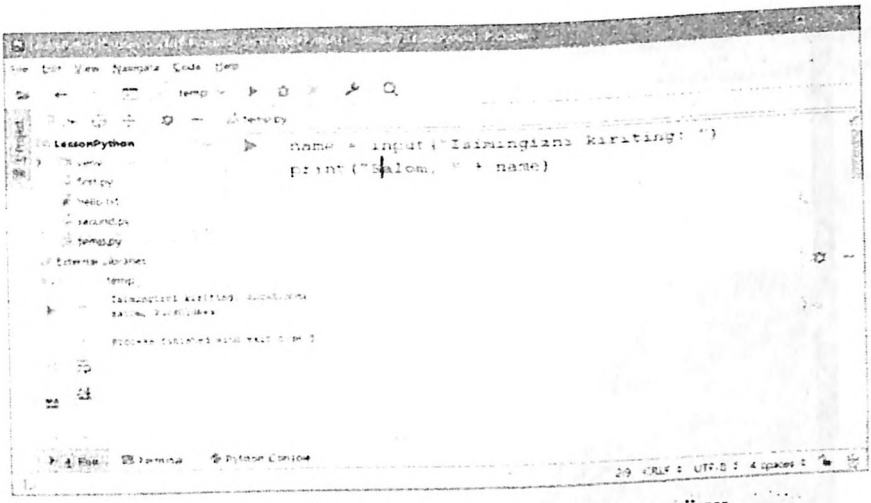


1.12–расм: PyCharm EE да янги файлни номлаш ойнаси

Яратилган файлга қуйидаги сатрларни киритилади:

```
name = input("Isimingizni kiriting:")
print = ("salom, " + name)
```

Дастурни ишга тушириш учун Run менюлар гуруҳидан Run банди танланади ёки клавиатурадан Shift+F10 тугмалари қўшиб босилади. Шунда лойиҳа ишга тушади, яъни комплияция бўлади. Дастурнинг ишга тушганини муҳитнинг остки қисмида жойлашган Run <лойиҳа номи> дан билиб олса бўлади.



1.13–расм: Лойиҳа ишга тушириш жараёни

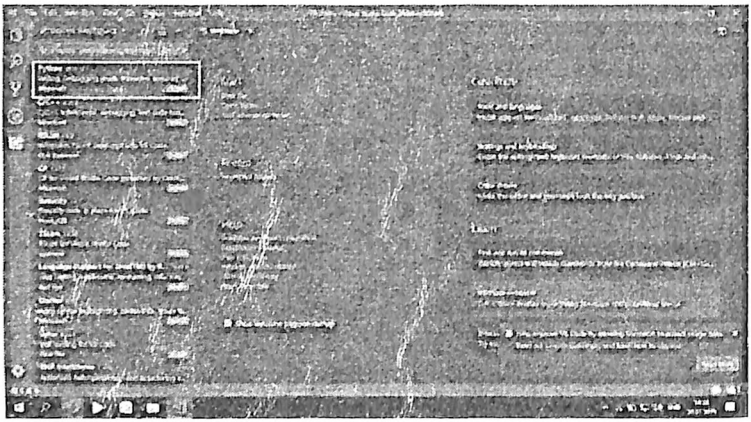
1.6. Visual Studio да Python дастурлаш тили

Visual Studio – Python дастурлаш тилида ишлаш имкониятини берувчи муҳитлардан бири ҳисобланади. Ушбу IDE нинг PyCharm IDE сидан устун томонлари мавжуд. Масалан, Веб иловалар яратиш учун ишлатиладаган қўшимча хизматлар ўрнатиш мумкин. Ҳозирги кунда **Visual Studio** ўзининг соддалаштирилган **Visual Studio Core** дастурини фойдаланувчиларга тақдим этган. Бу дастур ёрдамида фойдаланувчи ўзига тегишли бўлган хизматларни бутун жаҳон тармоғидан юклаб олиб, бу дастур орқали ишлатиши мумкин, яъни **VS Core** дастури **IDE** вазифисини бажариб беради.

VS Core дастурини https://www.viualstudio.com/ru/thank-you-downloading-visual-studio/ манзили орқали юклаб олинади. Юкланган файлни ишга тушириб, дастурни компьютерга ўрнатилади.

1.14–расм: VS Core дастури учун хизматлар

Visual Studio Core ишга тушгач, **Python** дастурлаш тилининг хизматини ўрнатилади. Бунинг учун дастурнинг чап томонида жойлашган **Extensions** банди танланади. Хизматни қидириш ёрдамида топиб ўрнатилади. Хизматнинг номи **Python** 2018.12.1 деб номланган. Бу хизмат **Visual Studio Core** дастурига ўрнатилганидан сўнг **Python** дастурлаш тилининг кодлари ёзилади. Янги файл яратиш учун **File** (Файл)→ **New** ("Новая файл") бандлари танланади.

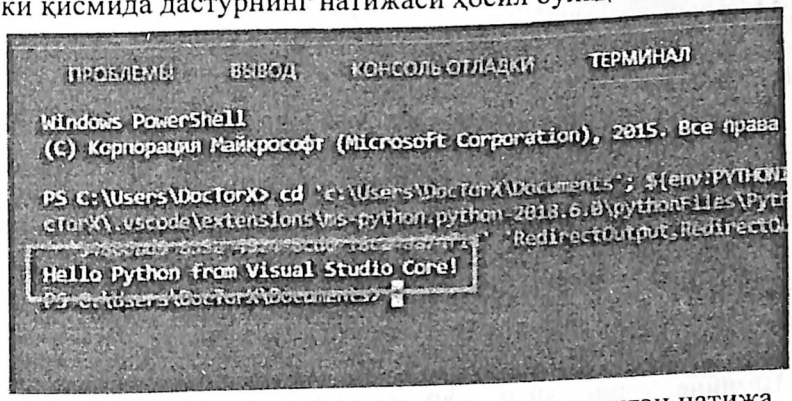


1.15–расм: Visual Studio Core дастурига Python хизматини ўрнатиш

Янги яратилган файл Hello.py номи билан сақланади. Сақланган файл бўш бўлгани учун унинг ичига қуйидаги Python дастурлаш тилида ёзилган код ёзилади:

print ("Hello Python from Visual Studio Core!")

Visual Studio Core дастурида ёзилган коднинг ишлашини текшириш учун F5 функционал тугмаси босилади. Дастурнинг остки қисмида дастурнинг натижаси ҳосил бўлади.



1.16–расм: Visual Studio Core дастурида олинган натижа

Назарий саволлар:

1. Python дастурлаш тили ким томонидан ва қачон яратилган?
2. PyCharm IDE қайси компания томонидан ишлаб чиқилган?
3. Дастурлаш тилининг расмий сайтни айтиб беринг.
4. Visual Studio Core муҳитидан дастурлаш тилининг муҳити сифатида фойдаланиш мумкинми?
5. Visual Studio Core муҳитига Python дастурлаш тилини қандай қўшиш мумкин?
6. IDE деганда нимани тушунасиз?

II. Python дастурлаш тили асослари

2.1. Pythonда дастур ёзиш қоидалари

Барча дастурлаш тилларида бўлгани каби **Python** дастурлаш тилининг ҳам ўзига мос алифбоси ва дастур ёзиш қоидалари мавжуд. **Python** дастурлаш тилида ҳар бир буйруқ янги сатрда жойлаштирилади.

Масалан:

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

Python дастурлаш тилида бўш жой ва табуляция жуда катта аҳамиятга эга. **Python** дастурлаш тилида операторлар блокини ташкил этиш бўш жой ва табуляция орқали амалга оширилади. Нотўғри қўйилган бўш жой ёки табуляция хатолик келтиради. Масалан, юқоридаги кодни ўзгартириб ёзиладиган бўлса, хатолик кўрсатади.

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

Шунинг учун янги кўрсатмаларни сатр бошидан ёзиш мақсадга муофиқ бўлади. Бу хусусият **Python** дастурлаш тилини Java, C# каби бошқа дастурлаш тилларидан ажратиб туради. Яна бир эътиборли томони шундаки, тилнинг баъзи бир операторлари бир нечта сатрлардан иборат бўлиши мумкин. Масалан:, **if** шарт оператори,

```
if a1 < a2:
    print("Hello")
```

Бу ҳолатда a1 ўзгарувчиси a2 ўзгарувчисидан кичик бўлса экранда **"Hello"** ёзуви ҳосил бўлади. Бу ерда бўш жой ёки табуляция орқали ёзилган print("Hello") буйруғи **if** операторининг бир қисми сифатида келмоқда. Python дастурлаш тилида кодларни ёзишда бўш жойларни 4 сонига каррали қилиб ташлаш мақсадга мувофиқ бўлади. 5 ёки 6 та бўш жойлар ташланса ҳам дастур хатосиз ишлайди. Бундай операторлар кўп эмас, шунинг учун бўш жой ташлаш борасида қийинчиликлар бўлмайди. **Python** дастурлаш тили C дастурлаш оиласидагидек регистр танлайдиган дастурлаш тили ҳисобланади, яъни бу дастурлаш тилида катта ва кичик ҳарфлар билан ёзилган ўзгарувчилар турли хил ўзгарувчилар деб каралади. Шунинг учун дастурлаш тилидаги хизматчи сўзлар ҳам регистрга боғлиқ бўлади.

Изоҳлар (комментариялар).

Python дастурлаш тили ёзилган кодларга қандай вазифа бажаришини изоҳлар ёрдамида ёритиб бориш мумкин. Дастур ишга тушаётган вақтда интерпретатор изоҳни танийди ва бу дастурни ишлашига ҳеч қандай нокулайлик туғдирмайди. **Python** дастурлаш тили изоҳлар блокли ёки сатрли кўринишда бўлади. **Python** дастурлаш тилида изоҳларни (# – фунта) белгиси билан ҳосил қилиш мумкин. Блокли изоҳлар қолдиришда фунта белгисини сатрнинг бошига қўйилади.

```
# Xabarni konsol oynasiga chiqarish
print("Hello World")
```

Сатрли изоҳларда эса буйруқлар ёзилган қатор охирига ёзилади.

```
print("Hello World") # Xabarni konsol oynasiga chiqarish
```

Shebang махсус буйруқ бўлиб, дастурнинг биринчи сатрида изоҳ сифатида киритиш учун ишлатилади. **Shebang** (#!) тарзда ёзилади ва дастур ҳақида маълумот беради.

```
#! Pythondagi birinchi dastur
print("Hello World") #Xabarni konsol oynasiga chiqarish
```

Асосий функциялар

Python дастурлаш тили бир қаторга киритиладиган (стандарт) функциялардан иборат. Улардан баъзилари дастур ёзишда кўп марта ишлатилади, айниқса, маълумотларни экранга чиқариш ва маълумотларни киритиш функциялари, лекин улар билан танишиб чиқамиз.

Маълумотларни чиқариш учун ишлатиладиган асосий функциялардан бири "**print()**" ҳисобланади:

```
print("Hello Python")
```

Агарда бирданига бир нечта маълумотни чиқариш керак бўлса, у ҳолда print() функциясига маълумотларни ёки ўзгарувчиларни, вергул ёки қўшиш (, ёки +) билан ажратиб ёзиш мумкин:

```
print("To'liq ismim: " + "Xurshidbek " + "Musayev")
```

Юқорида келтирилган дастурнинг қисми экранда барча маълумотларни бир сатрда ҳосил қилади:

```
To'liq ismim: Xurshidbek Musayev
```

Маълумотларни клавиатурадан киритиш ҳам мумкин. Бу функция **input()** функцияси ҳисобланади. Бу функция маълумотларни киритиш учун изоҳ ёзиш хусусиятига эга.

```
name = input("Ismingizni kiriting: ")
print("Salom, " + name)
```

Натижа:

```
Ismingizni kiriting: Xurshidbek
Salom, Xurshidbek
```

2.2. Ўзгарувчилар, маълумотлар турлари

Ўзгарувчилар маълум бир қийматни ўзида сақлайди. Python дастурлаш тилида бошқа дастурлаш тилларидаги каби лотин ҳарфи ёки остки чизиқча билан бошланади. Ўзгарувчилар ҳарфлардан ёки рақамлардан ташкил топади. Бундан ташқари, ўзгарувчиларнинг номи Python дастурлаш тили хизматчи сўзлари билан устма-уст тушмаслиги керак. Калит сўзлар дастурлаш тилида кўп эмас, уларни эслаб қолиш осон: **and, as, assert, break, class, continue, def, del, elif, else, expect, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield.**

Масалан: оддий ўзгарувчи эълон қилиш қуйидагича амалга оширилади:

```
name = "Temur"
```

Бу ерда **Temur** қийматини ўз ичига олувчи **name** номли ўзгарувчи аниқланган. Python дастурлаш тилида ўзгарувчилар икки хил йўл билан номланади: **camel case (туясимон)** ва **underscore notation (остки чизиқ билан ажратилган).**

Camel case (туясимон) да ҳар бир ўзгарувчини ташкил этувчи сўзлар катта ҳарфлардан бошланади. Ўзгарувчининг номи икки сўзнинг бирикмасидан ташкил топган бўлса, масалан,

```
userName = "Temur"
```

Underscore notation (остки чизиқ билан ажратилган) да эса қўшимча номлар остки чизиқ билан ажратилган бўлади, масалан,

```
user_name = "Temur"
```

Python дастурлаш тилида турли хилдаги маълумот турлари мавжуд, бутун сонли, ҳақиқий сонли, комплекс сонлар сатрли катталиклар, кетма-кетликлар, тўпламлар ва ҳоказоларга бўлинади.

– **boolean** – мантиқий тип. Қиймати **True** ёки **False**.

- **int** – хотирадан 4 байт жойни эгаллайдиган ихтиёрий бутун сон.
- **float** – хотирадан 8 байт жойни эгаллайдиган ихтиёрий ҳақиқий сон.
- **complex** – комплекс сон.
- **str** – сатрлар. Масалан: “Hello”. Python дастурлаш тилининг 3.x версиясидан бошлаб сатрли катталиклар Unicode белгиларини ўз ичига олган.
- **bytes** – номанфий бутун сон. 0 дан 255 гача бўлган сонларни ўз ичига олади.
- **list** – рўйхат.
- **tuple** – кортеж.
- **set** – объектларнинг тартибсиз тўплами.
- **frozen set** – set типи билан бир хил. Лекин бу ўзгармас ҳисобланади.
- **dict** – ҳар бир элементи калит ва қийматга эга бўлган тўплам.

Python дастурлаш тилида ўзгарувчилар маълумотлар турини ўзлаштириладиган қийматлардан келиб чиққан ҳолда фарқлайди. Сатрларда қўштирноқ (“ ”) ёки апостроф (‘ ’) лар ичига ёзилган маълумотлар **str** типига тегишли деб қаралади. Агарда ўзгарувчининг қиймати бутун сон бўлса, Python дастурлаш тили автоматик тарзда ўзгарувчи **int** типида эълон қилинган ҳисобланади. Агар ўзгарувчининг қиймати ҳақиқий сонга тегишли бўлса, ўзгарувчи **float** типида эълон қилинган деб қаралади.

```
x = 3.9e3
print(x)          # 3900.0
x = 3.9e-3
print(x)          # 0.0039
```

Ҳақиқий сон кўпи билан 18 та рақамдан ташкил топиши мумкин. Агарда рақамлар сони 18 та рақамдан кўп бўлса, юқорида кўрсатилганидек ёзиш мақсадга мувофиқ бўлади. Шунинг билан бирга битта дастур яратилаётган вақтда ўзгарувчиларга бир нечта турдаги қийматларни бирин-кетин бериш мумкин:

```
user_id = “12 Temur smith 438”
print(user_id)
user_id = 234
print(user_id)
```

type() функцияси орқали ўзгарувчилар қайси типга тегишли эканлигини билиш мумкин:

```
user_id = "12 Temur smith 438"
print(user_id)          # <class 'str'>
user_id = 234
print(user_id)          # <class 'int'>
```

2.3. Арифметик амаллар

Python дастурлаш тили барча арифметик амаллар билан ишлай олади.

(+) Икки сонни қўшиш:

```
print(6 + 2)        # 8
```

(–) Икки сонни айриш:

```
print(6 - 2)        # 4
```

(*) Икки сонни кўпайтириш:

```
print(6 * 2)        #12
```

(/) Икки сонни бўлиш:

```
print(6 / 2)        # 3
```

(//) Икки сонни бўлгандан ҳосил бўладиган соннинг бутун қисми:

```
print(7 // 2)       # 3
```

(**) Даражага кўтариш:

```
print(6 ** 2)           # 36
```

(%) Икки сонни бўлгандан ҳосил бўлган қолдиқ қисми:

```
print(7 % 2)            # 1
```

Арифметик амалларни кетма-кет қўлланилганда устунлик даражасига қараб ишлатилади. Устунлик даражалари қуйидаги рўйхатда келтирилган.

1. ** Чапдан ўнгга томон
2. *, /, //, % Чапдан ўнгга томон
3. +, - Чапдан ўнгга томон

Масалан: бизда қуйидаги амал бажарилмоқда:

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

Бу ерда аввал даражага кўтариш амали бажарилади (5 ** 2), сўнгра кўпайтириш амали (25 * 4), кейин эса чап томондаги сон қўшилади (100 + 3) ва ниҳоят, йиғиндига 7 сони қўшилади (103 + 7) = 110. Амаллар кетма-кетлигини ўзгартириш учун қавслардан фойдаланилади:

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

```
print(number)     # 224
```

Бундай ифодада биринчи ўринда қавсдаги кетма-кетликлар бажарилади, аввал биринчи қавснинг ичи ҳисобланади, сўнг иккинчи ва охирида кўпайтириш амаллари бажарилади. Эслатиб ўтиш керакки, арифметик амалларни бажараётганда (**int** ва **float**) типларидан фойдаланиш мумкин, агарда **int** типдаги сон **float** типидаги сонга қўшилса, **int** типидаги сон **float** типига ўгирилади.

Қиймат беришли арифметик амаллар

Арифметик амалларни бажаришда ўзгарувчининг қийматини қандайдир сонга орттириш ёки камайтириш мумкин. Бу турдаги амалларни қиймат беришли арифметик амаллар деб аталади. Қиймат беришли арифметик амаллар қуйидагилар:

(+=) Қиймат беришли қўшиш амали;

(–=) Қиймат беришли айириш амали;

(*=) Қиймат беришли кўпайтириш амали;

(/=) Қиймат беришли бўлиш амали;

(//=) Қиймат беришли бўлинмани бутун қисмини топиш амали;

(**=) Қиймат беришли даражага кўтариш амали;

(%=) Қиймат беришли қолдиқ топиш амали;

```
number = 10
number += 5
print(number)     # 15
number -= 3
print(number)     # 12
number //= 5
print(number)     # 2
```

Тип ўзгартирувчи функциялар

Python дастурлаш тилида сатр кўринишда берилган сонларни оддий сонларга айлантирувчи функция имконият яратади, хусусан, бу функциялар **int()** ва **float()** функциялари ҳисобланади. Бу функциялардан фойдаланиш қуйида берилган. Масалан:

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

Юқоридаги дастур натижаси 2 + 3 = 5 бўлиши мумкин. Лекин дастурда хатолик мавжудлигини кўрсатади, чунки дастурда

first_number номли ўзгарувчи сатр кўринишда берилган. Буни тўғрилаш учун дастурга қуйидагича ўзгартириш киритилади.

```
first_number = "2"
second_number = 3
third_number = int(first_number) + second_number
print(third_number)          # 5
```

float() типи ҳам шу кўринишда ишлайди. Лекин сонлар билан ишлаганда бир нарсани ҳисобга олиш керак бўлади, натижа кутилганидек аниқ бўлмаслиги мумкин. Масалан:

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

Натижа аниқ тўғри бўлмаслиги мумкин. Бу ҳолатда натижа аниқ бўлиши учун, сонни яхлитлаш керак бўлади. Яхлитлаш функция **round()** деб номланади. Бу функциядан фойдаланишда аниқлилик ҳам киритилади.

```
first_number = 2.0001
second_number = 0.1
third_number = first_number + second_number

print(round(third_number, 4)        # 2.1001
```

Функциянинг биринчи параметри яхлитлаш керак бўлган сонни билдиради, иккинчи параметри эса аниқлилик хонасини билдиради. Аниқлилик даражасини бутун сонларда киритиш лозим.

Сонларни саноқ системасида тақдим этиш

Дастурлаш тилларида сонлар 10 лик саноқ системасида ишлатилади. Python дастурлаш тилида сонларни 2, 8 ва 16 лик саноқ системаларида ҳам ишлатилса бўлади. Сонни иккилик саноқ системасида ёзиш учун соннинг олдига 0 ва b (**binary**) префикси ёзилади. 2 лик саноқ системасида 0 ва 1 рақамлари ишлатилади. Масалан:

```
x = 0b101   # 101 ikkilik sanoq sistemasida berilgan
```

Сонни 8 лик саноқ системасида аниқлаш учун унинг олдига 2 лик саноқ системасидагидек префикс ишлатилади. 8 лик саноқ системаси учун префекслар 0 ва o (**octal**):

```
a = 0o11          # 11 8 lik sanoq sistemasida berilgan
```

Бошқа саноқ системаларига ўхшаб 16 лик саноқ системасида ҳам ёзиш учун унинг олдига 0 ва x префекси ёзилади:

```
y = 0x0a          # a 16 lik sanoq sistamasida berilgan
```

Шунингдек, бошқа асоси бир хил бўлмаган саноқ системаларида арифметик амаллар бажариш мумкин:

```
x = ob101         # 5
y = 0x0a          # 10
z = x + y         # 15
print("{0} in binary{0:08b} in hex{0:02x} in octal{0:02o}".format(z))
```

Сонни бошқа саноқ системаларида чиқаришда форматлаб чиқариш керак бўлади ва **format** функциясидан фойдаланилади. Форматлаб чиқаришда формат буйруқларидан фойдаланилади. Формат буйруқларида сонни қайси саноқ системасида чиқариш кўрсатиб ўтилади. 2 лик саноқ системасида учун {0:08b} формати ишлатилади, бу ерда 8 – рақами сонни чиқаришда хоналар сонини билдиради, 8 рақамининг олдидаги 0 рақами эса 8 хонали бўлмаган сонларнинг олдига 0 рақамини жойлаштиришни билдиради. 16 лик саноқ системаси учун {0:02x} формати ишлатилади, бу ерда ҳам сонни чиқариш учун хоналар сони берилган. 8 лик саноқ системасида ёзиш учун {0:02o} форматда ёзилади.

Юқорида ёзилган дастурнинг натижаси қуйидагича:

```
15 in binary 00001111    in hex 0f    in octal 17
```

2.4. Мантиқий тип, мантиқий ифодалар ва солиштириш белгилари

Мантиқий ифодалар бир қатор амалларни ўз ичига олади. Барча амаллар мантиқий типда қиймат қайтаради. Python дастурлаш тилида мантиқий тип **boolean** деб номланган. Мантиқий типнинг қийматлари иккита бўлади – **True** (рост) ва **False** (ёлғон).

Солиштириш белгилари (Таққослаш амаллари)

Python дастурлаш тилида солиштириш белгилари (таққослаш белгилари) қуйидагича ёзилади:

(==) – агар иккита операнд бир-бирига тенг бўлса, рост (**True**), акс ҳолда, ёлғон (**False**) қиймат қайтаради.

(!=) – агар икки операнд бир-бирига тенг бўлмаса, рост (**True**), акс ҳолда, ёлғон (**False**) қиймат қайтаради.

(>) – агар биринчи операнд иккинчи операнддан катта бўлса, рост (**True**), акс ҳолда, ёлғон (**False**) қиймат қайтаради.

(<) – агар биринчи операнд иккинчи операнддан кичик бўлса, рост (**True**), акс ҳолда, ёлғон (**False**) қиймат қайтаради.

(>=) – агар биринчи операнд иккинчи операнддан кичик бўлмаса, рост (**True**), акс ҳолда, ёлғон (**False**) қиймат қайтаради.

(<=) – агар биринчи операнд иккинчи операнддан катта бўлмаса, рост (**True**), акс ҳолда, ёлғон (**False**) қиймат қайтаради.

Масалан,

```
a = 5
b = 6
result = (5 == 6)

print(result)       # False - 5 teng emas 6 ga
print(a != b)             # True
print(a > b)        # False - 5 kichik 6 dan
print(a < b)        # True
bool1 = True
bool2 = False

print(bool1==bool2)     # False – bool1 teng emas bool2
```

Таққослаш амали – сатрлар, сонлар, мантиқий қийматлар каби кўплаб объектларни таққослаш учун ишлатилади. Лекин таққосланаётган объектлар бир типда бўлиши керак.

Мантиқий амаллар

Python дастурлаш тилида мантиқий амаллар ҳам ишлатилади. Мантиқий амаллар 3 та бўлиб, улар қуйида келтирилган.

and (мантиқий кўпайтириш амали)

Икки ифоданинг қиймати рост бўлганда рост, қолган ҳолларда ёлғон қиймат қайтарувчи мантиқий амал ҳисобланади.

```
age = 22
weight = 58
result = age > 21 and weight == 58
print(result)       # True
```

Бу ҳолатда **and** оператори икки мантиқий ифоданинг натижасини мантиқий кўпайтиради, бу икки мантиқий ифода **age > 21** ва **weight == 58.** Агар икки мантиқий ифода рост бўлса умумий натижа рост (**True**) қиймат қайтаради, агар мантиқий ифодалардан бири ёлғон бўлса, умумий натижа ҳам ёлғон (**False**) бўлади. Масалан:

```
age = 22
```

```
weight = 58
isMarried = False
result = age > 21 and weight == 58 and isMarried
print(result)
```

or (мантиқий қўшиш амали)

Мантиқий қўшиш амали – икки операндниng қиймати ёлғон бўлса, ёлғон, қолган ҳолларда, рост қиймат қайтарувчи мантиқий амал.

```
age = 22
isMarried = False
result = age > 21 or isMarried
print(result)
```

not (мантиқий инкор амали)

Мантиқий инкор амали – операндниng қиймати рост бўлса ёлғон, ёлғон бўлса, рост қиймат қайтарувчи мантиқий амалдир.

```
age = 22
weight = 58
isMarried = False
print(not age > 21)          # False
print(not isMarried)         # True
```

2.5. Сатрлар билан ишлаш

Сатр – Python дастурлаш тилида қўштирноқ ёки апостроф ичида келган **UNICODE** кодировкаси мос келувчи белгилар кетма-кетлигидир. Масалан,

```
name = "Xurshidbek"
surname = 'Musayev'
print(name, surname)
```

Сатрлар устида бирлаштириш амали бажарилади. Бирлаштириш амали арифметик амаллардан бири бўлган қўшиш амали (+) фойдаланилади:

```
name = "Xurshidbek"
surname = 'Musayev'
fullname = name + " " + surname
print(fullname)
```

Сатрларни бирлаштириш қийинчилик туғдирмайди, лекин сатрни бирон сонга ёки рақамга бирлаштиришга тўғри келганда бирлаштириш амали фойда бермайди. Бундай ҳолларда, сонни ёки рақамни сатр кўринишга ўгириш керак бўлади. Бу ишни **str()** функцияси амалга оширади:

```
name = "Temur"
age = 33
info = "Name: " + name + "Age: " + str(age)
print(info)          # Name: Temur Age: 33
```

Eskeyp белгилари

Сатрлар билан ишлашда стандарт белгилардан ташқари, ўзгача киёфадаги махсус белгилардан фойдаланиш мумкин. Бундай белгиларни **Eskeyp белгилари** деб номланади. Eskeyp белгилари сатрнинг ичида **Enter** ва **Tab** га ўхшаш махсус буйруқларни ўз ичига олади. Масалан: \n кетма-кетлиги белгилар кетма-кетлигида Enter жойлаштириш учун хизмат қилади:

```
print("Messages:\nMessage1: Salom dasturchi " )
```

Юқорида келтирилган дастурнинг коди консол ойнасида матнни қуйидагича чиқаришда хизмат қилади:

```
Messages:
Message1: Salom dasturchi
```

Яна бир **\t** кетма-кетлиги сатрда табуляция жойлаштириш учун хизмат қилади. Умуман олганда, \ (слеш) белгиси билан келган белгилар кетма-кетлиги Eskeyp белгилари деб юритилади.

Сатрларни таққослаш

Python дастурлаш тилида сатрларни таққослаш мумкин. Сатрларни таққослашда сатрларнинг белгилари ва сатрлар қатнашган белгиларнинг регистрлари ҳисобга олинади. Рақамли белгилар алфавитдаги ҳарфлардан кичик ҳисобланади. Юкори регистрдаги ҳарфлар (бош ҳарфлар) қуйи регистрдаги ҳарфлардан (кичик ҳарфлар) кичик ҳисобланади. Масалан:

```
Str1 = "2a"
Str2 = "aa"
Str3 = "Aa"
print(str1 > str2)  # False, chunki str1 da birinchi belgi raqam hisoblanadi.
print(str2 > str3)  # True, chunki str2 ning birinchi belgisi quyi registrda.
```

Юқоридаги дастурда "2a" мантиқан "aa" дан кичик ҳисобланади. Сатрларни таққослашда сатрнинг аввал биринчи белгилари таққосланади. Агар биринчи белгилар бир хил бўлса, сатрнинг кейинги белгилари таққосланади.

Сатрларни таққослашда регистрга боғлиқлик бир қанча нокулайликлар туғдириши мумкин. Бу нокулайликларни бартараф этиш учун сатрларни регистр бўйича бир хил белгилар кетма-кетлигига ўтказиш керак бўлади. Бунинг учун регистр

алмаштирувчи методлардан фойдаланилади. Бу методлар **lower()** ва **upper()** деб номланади. **lower()** методи сатрдаги барча белгиларни қуйи регистрга ўтказади, **upper()** методи эса сатрдаги белгиларни юқори регистрга ўтказади. Масалан,

```
str1 = "Temur"
str2 = "Temur"
print(str1 == str2)          # False – satrlar teng emas
print(str1.lower() == str2.lower())          # True
```

2.6. "if" шарт оператори

Шарт оператори бу қандайдир бир шартга асосан икки ҳолатдан бири бажарувчи тармоқланувчи жараён ҳисобланади. Тармоқланувчи жараёнларнинг оператори If шарт оператори деб юритилади. Бу операторнинг кўриниши қуйидагича:

```
if мантиқий ифода (шарт):
        буйруқлар кетма-кетлиги
[elif мантиқий ифода (шарт):
        буйруқлар кетма-кетлиги]
[else:
        буйруқлар кетма-кетлиги]
```

Энг оддий кўринишларида **if** калит сўзидан кейин мантиқий ифода ёзилади. Агарда ушбу мантиқий ифода **True** қиймат қайтарса, кейинги қаторда жойлашган буйруқлар кетма-кетлиги бажарилади. Буйруқлар кетма-кетлиги бир ёки бир нечта бўлиши мумкин:

```
age = 22
if age > 21:
        print("Kirishga ruxsat bor")
print("Dastur yakunlandi")
```

Юқоридаги дастурда **age** ўзгарувчиси 21 дан катта бўлгани учун, **if** шарт оператори консол ойнасида қуйидагича натижа қайтаради:

```
Kirishga ruxsat bor
Dastur yakunlandi
```

Агар age ўзгарувчи 21 дан катта бўлмаганда, дастурнинг натижаси бошқача бўларди. Яъни, **Dastur yakunlandi** деган маълумотни ўзи экранда намоён бўларди. Чунки **if** шарт оператори **False** қиймати учун ҳеч қандай буйруқлар кетма-кетлиги мавжуд эмас. Дастурда шарт операторининг **else** қисми ишлатилмаса, бу операторни **чала шартли оператор** деб юритилади. Агар дастурда

шарт операторининг **else** қисми ишлатилса, бу операторни **тўла шартли оператор** деб юритилади. Қуйида тўла шартли операторга мисол келтирилган:

```
age = 18
if age > 21:
    print ("Kirishga ruxsat bor")
else:
    print ("Kirishga ruxsat yo'q")
```

Дастурда икки ва ундан ортиқ шартларни текширишга тўғри келиб қолганда Python дастурлаш тилида шарт операторининг қўшимча имкониятидан фойдаланилади. Шарт операторининг қўшимча имкониятли қисми **elif** деб номланиб, **else if** буйруқларининг умумлашмасидир. Қуйида қўшимча имкониятга мисол келтирилган:

```
age = 18
if age >= 21:
    print("Kirishga ruxsat bor")
elif age >=18:
    print("Kirishga qisman ruxsat bor")
else:
    print("Kirishga ruxsat yo'q")
```

Юқоридаги дастурдан кўриниб турибдики, қўшимча имконият **if** ва **else** буйруқларининг орасида ёзилади.

Ичма-ич жойлашган if шарт операторлари.

if шарт операторини ичма-ич ишлатиш мумкин. Масалан:

```
if age = 18:
    print("18 dan katta son")
    if age > 21:
        print("21 dan katta son")
    else:
        print("18 dan katta, 21 dan kichik son")
```

Ичма-ич жойлашган шарт операторларини ишлатишда бўш жойларга эътибор бериш керак бўлади. Юқоридаги мавзуларда айтилганидек, дастур ёзиш қоидасига риоя қилиш керак.

Мисол: Киритилган суммани фойдаланувчи кўрсатган валюта суммасига ўтказиш.

```
#!Konvertor dasturi
money = int(input("Almashtirilishi kerak bo'lgan summani kiriting: "))
type_money = int(input("Valyuta kodi: USD - 100, Rubl - 200 : \n"))
if type_money == 100:
```

```
        print("Valuta: AQSH Dollari")
        usd = money / 8450
        print(usd)
elif type_money == 200:
        print("Valuta: Rossiya Rubli")
        rubl = money / 125
        print(rubl)
else:
        print("Valyuta kodi noto'g'ri!")
```

Юқоридаги дастурнинг таҳлили

Дастурда **input()** функцияси орқали консол ойнасига фойдаланувчи томонидан қийматлар киритилади. **input()** функцияси маълумотларни консол ойнасидан сатр кўринишда ўқийди. Сатр кўринишда ўқилган қийматни сонли кўринишга **int()** функцияси ёрдамида ўтказилади. Ўтказилган қиймат **money** ўзгарувчисига қиймат сифатида узатилади. **type_money** ўзгарувчисига ҳам юқоридаги кетма-кетлик тарзда қийматлар киритилади ва ўзгартирилади.

if шарт оператори ёрдамида type_money ўзгарувчисининг қиймати текширилади, агар киритилган сон 100 га тенг бўлса, фойдаланувчи киритган суммасини долларга, агар киритилган валюта коди 200 га тенг бўлса, фойдаланувчи киритган суммани рублга ўтказади. Агар фойдаланувчи томонидан валюта коди тўғри киритилмаса, у ҳолда, фойдаланувчига Валюта коди нотўғри киритилганлиги ҳақида ахборот берилади.

Дастур ишлаши учун мисоллар: ***Сўмдан долларга ўгириш***

```
Almashtirilishi kerak bo'lgan summani kiriting: 1690000
Valyuta kodi: USD - 100, Rubl - 200 :
100
Valuta: AQSH Dollari
200.0
```

Дастур ишлаши учун мисоллар: ***Сўмдан рублга ўгириш***

```
Almashtirilishi kerak bo'lgan summani kiriting: 125000
Valyuta kodi: USD - 100, Rubl - 200 :
200
Valuta: Rossiya Rubli
1000.0
```

Дастур ишлаши учун мисоллар: ***Валюта кодини 300 киритиш***

```
Almashtirilishi kerak bo'lgan summani kiriting: 125000
Valyuta kodi: USD - 100, Rubl - 200 :
300
Valyuta kodi noto'g'ri!
```

2.7. Такрорланувчи жараёнлар

Такрорланувчи жараён бу – дастур кодининг маълум бир шартга асосан бир неча марта такрорланишидан келиб чиқади. Барча дастурлаш тилларида 3 хилдаги такрорланиш операторлари мавжуд. Бу такрорланиш опеторлари қуйидагилар:

- ✓ Параметрли такрорланиш оператори;
- ✓ Шарти олдин берилган такрорланиш оператори (шартдан кейин бажараладиган такрорланиш оператори);
- ✓ Шарти кейин берилган такрорланиш оператори (шартдан олдин бажараладиган такрорланиш оператори).

Python дастурлаш тилида юқорида келтирилган такрорланиш операторларидан 2 таси ишлатилади, яъни параметрли такрорланиш оператори ва шарти олдин берилган такрорланиш оператори (шартдан кейин бажараладиган такрорланиш оператори) дир. Бу икки оператор куйида таништирилган.

while цикли

Шарти олдин берилган такрорланиш оператори (шартдан кейин бажараладиган такрорланиш оператори) **while** хизматчи сўзи ёрдамида тасвирланади. Бу операторнинг кўриниши қуйидагича:

```
while шартли_ифода:
    <такрорланиш танаси>
```

while калит сўзидан сўнг шартли ифода жойлаштирилиб, шу шарт ифоданинг қийматига қараб такрорланиш оператори иш бажариши ёки бажармаслиги белгиланади. Агар шарт **True** қиймат қайтарса, такрорланиш давом этади, шарт **False** қиймат қайтарган ҳолда такрорланиш оператори иш жараёнини якунлайди. **while** такрорланиш операторига таалуқли бўлган барча буйруқлар кетма-кетлиги сатрма-сатр ва бўш жой ёки табуляциялар билан ёзилиши керак бўлади.

```
choice = "y"
while choise.lower() == "y":
    print("Salom")
    choice = input("Davom ettirish uchun Y ni bosing, chiqish uchun ixtiyoriy
tugmani: ")
```

```
print("Dastur yakunlandi")
```

Бу ҳолатда **while** такрорланиш оператори **choice** ўзгарувчисига лотин алифбосидаги "Y" ёки "y" киритилаётган бўлса, такрорланиш давом эттиради. Такрорланиш операторининг танаси иккита буйруқлар кетма-кетлигидан ташкил топган. Цикл танасининг биринчи сатри "Salom" хабарини чиқаради, иккинчи сатри эса **choise** ўзгарувчиси учун қиймат қабул қилади. Агар фойдаланувчи томонидан "y ёки Y" ҳарфларидан бошқа белги киритилган бўлса, цикл якунланади, чунки **choice.lower() == "y"** шарти **False** қиймат қайтаради. Ҳар бир циклнинг қайтарилиши **итерация** деб номланади. Дастурнинг охирги сатрдаги **print("Dastur yakunlandi")** буйруқлар кетма-кетлиги цикл танасига кирмади, сабаби унинг олдида бўш жой ёки табуляция қўйилмаган. Қуйида такрорланувчи жараёнга мисол сифатида факториални ҳисобловчи дастур келтириб ўтилган:

```
#! Faktorialni hisoblovchi dastur
number = int(input("Sonni kiriting: "))
i = 1
fact = 1
while i <= number:
    fact = fact * i
    i = i + 1
print(str(number) + "soninig faktoriali " + str(fact) + " ga teng")
```

Konsoldagi natija:

```
Sonni kiriting:    6
6 soninig faktoriali 720 ga teng
```

for такрорланиш оператори

for такрорланиш оператори цикл ташкил этувчи операторилар нинг бири ҳисобланади. **for** цикли берилган сонлар кетма-кетлигининг ҳар бир аъзоси учун ишлатилади. Сонлар кетма-кетлигини, яъни оралиқ **range()** функцияси ёрдамида яратилади. **for** такрорланиш операторининг кўриниши қуйидагича ёзилади:

```
for int_var in range():
    instruksiyalar
```

for калит сўзидан сўнг ўзида бутун сонларни сақловчи **int_var** ўзгарувчи келади, сўнгра **in** калит сўзи ва **range()** функцияси чақирилиб, икки нуқта қўйилади. Кейинги сатрдан бошлаб циклнинг танасига тегишли бўлган буйруқлар кетма-кетлиги дастур ёзиш қоидаларига риоя қилган ҳолда ёзилади.

Цикл бажарилиши давомида Python дастурлаш тили **range()** функцияси яратган сонлар кетма-кетлигини бирин-кетинликда циклнинг ўзгарувчиси қиймат сифатида олиб циклни такрорлаб чиқади. Сонлар тугаганидан сўнг **for** такрорланиш оператори ўз ишини якунлайди.

Қуйида мисол тариқасида юқорида кўриб ўтилган факториалнинг ҳисоблаш дастурини for оператори ёрдамида кўриб ўтилган:

```
#! Factorial ni hisoblaydigon dastur
number =int(input("Sonni kiriting: "))
fact = 1
for i in range(1, number + 1):
    fact * = i
print(str(number) + " soninig faktoriali " + str(fact) + " ga teng")
```

range() функцияси икки аргументни ўз ичига оладиган – бошланғич сон (юқоридаги дастурлар 1) ва киритилган сонгача бўлган сонларни кўрсатувчи сон (яъни **number + 1**). Агар киритилган сон 6 деб фараз қилинса, range() функциясининг кўриниш қуйидагича бўлади:

range(1, 6 + 1)

Ушбу функция 1 дан бошлаб 7 гача бўлган сонлар кетма-кетлигидаги сонларни ўз ичига олади. Яъни [1,2,3,4,5,6].

Цикл бажарилиши давомида сонлар кетма-кетлиги сонлар бирма-бир **i** ўзгарувчисига узатилади, циклнинг танасида **i** ўзгарувчиси **fact** ўзгарувчисига кўпайтирилади ва якунда киритилган соннинг факторияли ҳосил бўлади.

Дастурнинг консолдаги натижаси:

```
Sonni kiriting:    6
6 soninig faktoroali 720 ga teng
```

range() функцияси

range функциясининг кўринишлари қуйидагича:

- **range**(stop): 0 дан **stop** гача бўлган барча сонларни қайтаради;
- **range**(start, stop): **start** (ҳам киради) дан бошлаб **stop** (кирмайди) гача бўлган оралиқдаги сонларни қайтаради.
- **range**(start, stop, step): **step** қадам ташлаган ҳолда, **start** дан бошлаб **stop** гача бўлган сонларни қайтаради.

range функцияси чақирилишига доир мисоллар.

range(5) **# 0, 1, 2, 3, 4**

range(1, 5) **# 1, 2, 3, 4**
range(2, 10, 2) **# 2, 4, 6, 8**
range(5, 0, -1) **# 5, 4, 3, 2, 1**

Ичма-ич жойлаштирилган цикллар

Дастурлаш тилларида бир цикл бошқа циклни ўз ичига олиши мумкин. Мисол учун Пифагор кўпайтириш жадвалини тасвирловчи дастурни олайлик:

```
for i in range(1, 10):
    for j in range(1, 10):
        print(str(i * j), end="\t")
    print("\n")
```

Ташқи цикл **for i in range(1, 10)** буйруғи 9 марта такрорланади. Ички циклдаги **for j in range(1, 10)** буйруғи 9 марта такрорланади. Ички цикл ташқи циклдаги ҳар бир такрорланиш учун 9 мартадан такрорланади. Дастурдаги умумий такрорланишни оладиган бўлсак, 81 га тенг. Дастурнинг натижаси қуйида берилган.

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Циклдан чиқиш. break ва continue операторлари

Цикллар билан ишлаш давомида циклни синдириш ёки навбатдаги қадамга ўтишга тўғри келади. Бу амалларни бажаришга **break** ва **continue** операторларидан фойдаланиш мумкин. **break** оператори циклни синдириш (яъни циклдан чиқиш, такрорланишни якунлаш) учун ишлатилади. **continue** оператори эса амалдаги циклнинг кейинги қадамига сакраб ўтишни амалга оширади. Икки оператор ҳам ичма-ич жойлаштирилган цикларда энг ички цикл учун иш бажаради. Ташқи цикллар учун таъсир кўрсата олмайди. Масалан:

```
#! Almashtirish punkti
print("Chiqish uchun Y ni bosing")
while True:
    data = input("Almashtirish uchun summani kiriting: ")
    if data.lower() == "y":
```

```
        break       #sikldan chiqish
    money = int(data)
    cache = round(money / 8450, 2)
    print("siznig mablag'ingiz " + str(cache) + " dollar ")
print("Almashtirish yakunlandi")
```

Бу дастур тузишда такрорланиш операторидан фойдаланилган, такрорланиш операторининг шарти **True** ҳолида чексиз цикл сифатида қараш мумкин. Бу такрорланиш аниқ бўлмаган дастур ҳисобланиб, дастур осилиб қолишдан, яъни тугамаслигининг олдини олиш учун break оператори қўлланилади. Маълумотларни киритиш цикл бажариш давомида амалга оширилади. Фойдаланувчидан сон киритиш ёки "Y" ҳарфи киритиш сўралади. Агар фойдаланувчи сон киритадиган бўлса, киритилган сонни валютага ўгириб фойдаланувчига тақдим этилади ва яна сон киритиш ёки "Y" ҳарфи киритиш сўралади. Бу жараён фойдаланувчи томонидан "Y" ҳарфи киритилмагунча давом этади. Агар фойдаланувчи "Y" ҳарфини киритадиган бўлса, дастур якунланганлиги ҳақида хабар берилади.

Дастурнинг натижаси қуйида келтирилган:

```
Chiqish uchun Y ni bosing
Almashtirish uchun summani kiriting: 845000
Sizning mablag'ingiz 100.0 dollar
Almashtirish uchun summani kiriting: Y
Almashtirish yakunlandi.
```

Юқорида келтирилган дастурда хатоликлар мавжуд. Бу хатоликлар мантиқий хатоликлар бўлиб, дастур ишлаши давомида келиб чиқади. Юқоридаги дастурга манфий сон бериб кўрилса, бу хатолик ҳосил бўлади. Яъни дастур манфий сон қайтаради. Бундай вазиятларда киритилган сонни манфийликка текшириш лозим, чунки шунда мантиқий хатолик юзага келмайди. Бу ҳолатда дастурга қуйидагича ўзгартириш киритилади, яъни дастур **continue** оператори ёрдамида хатоликни бартараф этилади:

```
#! Pul almashtirish punkti
print("Chiqish uchun Y ni bosing")
while True:
    data = input("Almashtirish uchun summani kiriting: ")
    if data.lower() == "y":
        break       # sikldan chiqish
    money = int(data)
    if money < 0:
        print("Kiritilgan summa musbat bo'lishi kerak!")
```

```
            continue
        cache = round(money / 8450, 2)
        print("siznig mablag'ingiz " + str(cache) + " dallar")
print("Almashtirish jarayoni yakunlandi")
```

Дастурнинг натижаси билан қуйидагича:

```
Chiqish uchun Y ni bosing
Almashtirish uchun summani kiriting: -845000
Kiritilgan summa musbat bo'lishi kerak!
Almashtirish uchun summani kiriting: 845000
Sizning mablag'ingiz 100.0 dollar
Almashtirish uchun summani kiriting: Y
Almashtirish jarayoni yakunlandi
```

2.8. Функциялар

Функция – содда қилиб айтганда бу қисм дастур. Маълум бир вазифани бажарувчи мустақил равишда бажарилиши мумкин бўлган дастур ҳисобланади. Функцияларни эълон қилиш қуйида кўрсатилган.

```
def funksiyaning_nomi ([parametrlari]):
    buyruqlar ketma-ketligi
```

Функциянинг аниқланиши **def** иборасидан бошланади. Ҳар бир функциянинг номи мавжуд ва парамертларга эга ёки параметрсиз бўлади. Қавслар функциянинг параметрли ёки парамертсизлигига қарамайди, яъни қавслар мавжуд бўлиши керак. Функциянинг номи ва параметрлари берилгандан сўнг икки нуқта қўйилиб, кейинги сатрдан функциянинг тана қисми жойлаштирилади. Функциянинг танасига кирувчи барча буйруқлар сатр бошидан ва бўш жой ёки табуляция жойлаштирилган ҳолда ёзилади. Масалан: содда бир функциянинг аниқланиши қуйида берилган:

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

Ушбу функциянинг номи **say_hello** бўлиб, параметрсиз функция эълон қилинган. Функциянинг тана қисмида битта буйрук мавжуд. Бу буйрук ёрдамида консолга "Hello" чиқарилади. Функцияни чақиришда функция эълон қилинган жойдан кейинги қаторларда функциянинг номи ёзилиб чақирилади.

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

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

Функциялар билан ишлашда парамертли функциялар билан ишлашга тўғри келади. Бундай функцияларни биринчи навбатда парамертли қилиб эълон қилиш лозим. Параметрли функциялар қуйидагича эълон қилинади.

```
def say_hello(name):
    print("hello " + name)
say_hello("Temur")
say_hello("Bobur")
say_hello("Aziza")
```

Юқоридаги дастурдан натижа қуйида кўрсатилган:

Hello, Temur

Hello, Bobur

Hello, Aziza

Стандарт қийматлар

Параметрли функциялар билан ишлашда параметрнинг бошланғич қиймати мавжуд ҳолда эълон қилиш мумкин. Бошланғич қиймат билан эълон қилинган параметр **стандарт қийматга эга параметр** деб юритилади. Масалан:

```
def say_hello(name = "Temur"):
    print("Hello" + name)
say_hello()
say_hello("Bobur")
```

Юқорида кўрсатилган дастурда **name** параметри мажбурий қиймат бериш шарт эмас ҳисобланади. Агар функциянинг параметри сифатида бирон-бир қиймат берилмаса ва функция чақирилган ҳолда, функция параметри стандарт қиймат устида амаллар бажаради.

Номланган параметрлар

Функциялар билан ишлашда параметрлар қандай тартибда жойлаштирилган бўлса, шу тартибда параметрга қиймат қабул қилинади. Масалан:

```
def display_info(name, age):
    print("Name: " + name + "\t" + "Age: " + str(age))
display_info("Temur", 22)
```

Функция чақирилган вақтда параметр сифатида берилган қийматларнинг биринчиси **name** параметрга, иккинчи қиймат эса

age параметрга узатилади, агар бошқа функцияларда параметрлар 2 ва ундан ортиқ бўлса, шу тартибда қийматлар параметрларга узатилади. Номланган параметрларни қўллаш функциядан фойдаланишда кенг имкониятлар беради. Яъни, параметрлар тартибини алмаштириб қиймат бериш имконини беради.

```
def display_info (name, age):
    print("Name: " + name + "Age: " + str(age))
display_onfo(age = 22, name = "Temur")
```

Сони ноаниқ параметрлар

Функцияларни эълон қилишда параметрлар сони ноаниқ функциялар билан ишлашга тўғри келади. Бундай функцияларни эълон қилиш учун юлдузча (*) белгиси ёрдам беради. Битта параметр эълон қилинган ҳолда, параметрнинг олдига (*) юлдузча белгиси жойлаштирилса, сони ноаниқ параметрли функция эълон қилинган ҳисобланади.

```
def sum(*params):
    result = 0
    for n in params:
        result = result + n
    return result
sum0fNumbers1 = sum(1, 2, 3, 4, 5)
sum0fNumbers2 = sum(3, 4, 5, 6)
print(sum0fNumbers1)
print(sum0fNumbers2)
```

Бу ҳолатда **sum** функцияси ягона – ***params** параметрини қабул қилади, лекин параметр номининг олдидаги юлдузча (*) шуни билдирадики, ушбу параметр ўрнига исталганча параметрлар тўпламини жойлаштириш мумкин. Функцияда for цикли орқали ушбу параметрлар тўпламининг барчасини кўриб чиқса бўлади ва улар устида турли амаллар бажариш мумкин. Масалан, юқорида рақамлар йиғиндиси қайтарадиган функция эълон қилинган.

Қиймат қайтариш

Функция, шунингдек, қиймат қайтара олади, бунинг учун функцияда **return** хизматчи сўзи ишлатилади:

```
def exchange(usd_rate, money):
    result = round(money / usd_rate, 2)
    return result
result1 = exchange(8010, 300 000)
print(result1)
```

```
result2 = exchange(8000, 300 000)
print(result2)
result3 = exchange(8020, 300 000)
print(result3)
```

Функция қиймат қайтаргани учун олинадиган қийматни бирон-бир ўзгарувчига ўзлаштириб олиш мумкин. Ўзлаштирилган қийматни ихтиёрий жойда ишлатиш мумкин.

result2 = exchange(8000, 300 000).

Python дастурлаш тилидаги функцияларда бир вақтнинг ўзида бир нечта қиймат қайтариш имконияти мавжуд:

```
def create_deafult_user():
        name = "Temur"
        age = 33
        return name, age
user_name, user_age = create_deafult_user( )
print("Name: " + user_name, "\tAge: " + str(user_age))
```

Юқоридаги дастурда функция иккита қиймат қайтаради: name ва age. Функция чақирилган вақтда навбат билан функциядан қайтарилган қийматлар user_name ва user_age ўзгарувчиларига ўзлаштирилади. Ўзлаштирилган қийматлардан дастурнинг ихтиёрий қисмида фойдаланиш мумкин.

main функцияси

Дастурда жуда кўплаб функциялар аниқланиши мумкин. **main** функциясини аниқланган функцияларни тартибга солиш учун ишлатилади. main функциясини дастурнинг бош қисмида ёзиш мумкин. Қолган аниқланадиган функцияларни main функциясидан кейин ёзилса, хатолик юз бермайди. main функциясини дастурнинг асосий жойида чақириш билан дастурни ишга тушириш мумкин.

```
def main():
        say_hello("Temur")
        usd_rate = 8010
        money = 300 000
        result = exchange(usd_rate, money)
        print("Mablag'ingiz " + result + "dollar")

def say_hello(name)
        print("Hello, " + name)

def exchange(usd_rate, money)
        result = round(money / usd_rate, 2)
        return result
```

```
# main finksiyasini chaqirish
main()
```

2.9. Локал ва глобал ўзгарувчилар

Барча дастурлаш тилида локал ва глобал ўзгарувчи тушунчалари мавжуд. Python дастурлаш тилида ҳам локал ва глобал ўзгарувчилари мавжуд. Python дастурлаш тилида ўзгарувчиларнинг ишлаш доирасига қараб ўзгарувчи ишлаш доираси деб аталади.

Глобал ўзгарувчилар дастурда ишлатиладиган барча аниқланган функциялар учун ишлатилиши мумкин. Бу ўзгарувчидан дастурнинг ихтиёрий жойида фойдаланишни англатади. Масалан:

```
name = "Temur"
def say_hi():
    print("Hello, " + name)
def say_bye():
    print("Good bye" + name)
say_hi()
say_bye()
```

Юқорида келтирилган дастурда **name** ўзгарувчиси глобал ўзгарувчи хисобланади. Дастурда берилган иккала функция ҳам уни аниқлаш ва ишлатиш имконига эга.

Глобал ўзгарувчидан фарқли равишда локал ўзгарувчи ҳам ишлатилади. Локал ўзгарувчи фақат аниқланган функция учун ишлатилиб бошқа функциялар учун ёпиқ ҳисобланади. Масалан:

```
def say_hi():
    name = "Samad"
    surname = "Johnson"
    print("Hello, " + name + " " + surname)
def say_bye():
    name = "Temur"
print("Good bye, " + name)
say_hi()
say_bye()
```

Берилган дастурда ҳар бир функцияда **name** номли локал ўзгарувчи аниқланган. Ўзгарувчилар номлари бир хил бўлишига қарамай, бу ўзгарувчилар ўзи аниқланган функция доирасида ишлатилади. Шунингдек, **say_hi** функциясида **surname** ўзгарувчиси аниқланган, лекин локал ўзгарувчи бўлгани учун **say_bye** функциясида уни қўллаш имконияти йўқ.

Ўзгарувчиларни аниқлашнинг яна бир йўли мавжуд бўлиб, унда локал ўзгарувчи глобал ўзгарувчини ёпиб қўяди:

```
name = “Temur”
def say_hi( ):
    print(“Hello, ” + name)
def say_bye( ):
    name = “Bobur”
    print(“Good bye, ” + name)
say_hi()    # Hello Temur
say_bye()   # Good bye Bobur
```

Бу ерда **name** глобал ўзгарувчи сифатида аниқланган. Лекин **say_bye** функциячида ҳудди шундай ўзгарувчи қайта аниқланган. Агар **say_hi** функциясида глобал ўзгарувчи ишлатилса, **say_bye** функциясида эса глобал ўзгарувчини ёпиб қўювчи локал ўзгарувчини ишлатади.

Агарда функциянинг ичида глобал ўзгарувчидан фойдаланиш керак бўлиб қолса, у ҳолда **global** калит сўзидан фойдаланишга тўғри келади. Масалан:

```
def say_bye( ):
    global name
    name = “Bobur”
    print(“Good bye, ” + name)
```

2.10. Модуллар

Модуллар Python дастурлаш тилида ишлатиладиган файллар ҳисобланади. Бошқа дастурлаш тилида ҳам модулга ўхшаган файллар мавжуд. Модул қайдай файл? Модул бу файл бўлиб, ўзининг ичида олдиндан аниқланган функциялар ва махсус кодлар мажмуасидир. Модул яратиш учун, хусусан, ***.py** кенгаймага эга бўлган файл яратилиши керак бўлади. Файлнинг номи модулнинг номи деб қабул қилинади. Сўнг ушбу модулда бир ёки бир нечта функциялар аниқланиши керак бўлади. Модулни бошқа модулга улаш учун **import** хизматчи сўзидан фойдаланилади.

2.11. Истисноларни бошқариш

Python дастулаш тилида икки хил хатоликларга дуч келиш мумкин. Бу икки хатолик қуйидагилар: синтаксик ва мантиқий хато. Биринчи турдаги хатоликлар дастур тузиш жараёнида юзага келади. Яъни дастурчи томонидан бирон-бир махсус белги қолиб кетиши, ўзгарувчи, ўзгармас ёки хизматчи сўзнинг нотўғри ёзилишидан келиб чиқади. Бу жараёнда хатоликни транслаятор

аниқлаб дастурчига хабар беради. Ҳозирги дастурлаш муҳитлари автоматик тарзда дастур тузиш жараёнида синтактик хатоликларни ўз вақтида аниқлаш хусусиятига эга. Масалан, бундай муҳитларга PyCharm IDE мисол бўлади.

Хатоликнинг иккинчи тури **Runtime error** (Дастур ишлаётган вақтдаги хатолик ёки мантиқий хатолик) деб юритилади. Бу хатолик дастурчи томонидан атайлаб қилинган хатоликка ўхшамайди. Мантиқий хатоликлар истиснолар деб юритилади. Истиснолар қандай юзага келади? Дастур коди тўғри бўлса? деган савол туғилиши табиий. Истиснога мисол сифатида ихтиёрий сонни 0 сонига бўлиш ёки тип ўзгартираётган вақтдаги хатоликларни (белгили катталикларни сонли катталикларга ўгириш) айтиш мумкин. Қуйидаги икки мисолни кўриб ўтайлик:

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

Юқорида келтирилган дастурда дастур тўғри иш бажариб, ҳеч қандай истисноли вазиятга дуч келади. Яъни сатрли кўринишда берилган "5" белгиси муваққиятли равишда сонли катталикка ўгирилади.

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

Ушбу дастур бажарилиш давомида истисноли вазиятга дуч келади. Сабаби бу дастурда "salom" белгилар кетма-кетлигини сонли катталикка ўгиришга уриниш амалга оширилмоқда. Бу ерда ValueError истисноси юзага келади, чунки "salom" белгилар кетма-кетлиги сонли катталикка айлантирилмайди. Бундай хатоликлар юзага келмаслиги учун дастурчи дастур тузиш давомида дастурни фойдаланувчи билан мулоқот интерфейсини ўрнатган ҳолда тузиши керак бўлади. Масалан:

```
string = input("Son kiriting: ")
number = int(string)
print(number)
```

Ушбу дастурда фойдаланувчидан сон киритиш талаб қилинмоқда. Математик хатолик, яъни 0 га бўлинишни ҳеч бир дастурчи мулоқот интерфейси орқали айланиб ўта олмайди. Сабаби шундан иборатки, фойдаланувчи киритган қийматлардан ҳам 0 га бўлиниш хатолиги келиб чиқиши мумкин. Бундай вақтларда

истисноларни бошқариш жараёнига мурожаат қилинади. Python дастурлаш тилида истисноларни бошқариш учун **try..except** блокидан фойдаланилади. Бу блок ёрдамида истисноларни бошқариш, таҳлил қилиш ва фойдаланувчига хабар бериш мумкин:

```
try:
    буйруқлар кетма-кетлиги
except [истисно тури]:
    буйруқлар кетма-кетлиги
```

Истисно юзага келиши мумкин бўлган кодлар кетма-кетлиги **try** хизматчи сўзидан кейин ёзилади. Истисно юзага келса, уни бошқариш учун алоҳида блок фойдаланувчига ёрдам беради. Бу блок **except** блоки бўлиб, таҳлил қилиш ва фойдаланувчига хабар бериш учун ишлатилади. **except** блоки бир ёки бир нечта бўлиши мумкин. Бунга сабаб истисно ҳосил бўладиган қисмда бир ёки бир нечта эҳтимолли хатоликлар мавжуд бўлиши мумкин.

```
try:
    number = int(input("Son kiriting: "))
    print("Kirilgan son:", number)
except:
    print("Kiritilgan belgilar ketma-ketligini songa aylantirishda xatolik")
print("Dastur yakunlandi")
```

Юқорида келтиришган дастурни консолдаги натижаси:

```
Son kiriting: salom
Kiritilgan belgilar ketma-ketligini songa aylantirishda xatolik
Dastur yakunlandi
```

Дастур натижасидан кўриниб турибдики, фойдаланувчи соннинг ўрнига ҳарфлардан ташкил топган белгилар кетма-кетлигини киритди. Дастурда истисно ҳосил бўлди, дастур белгилар кетма-кетлигини сонга айлантира олмаганлиги сабабли бошқа блокка дастур кўчиб ўтди. Фойдаланувчи тўғри маълумот киритган жараён қуйидаги натижада берилган.

```
Son kiriting: 22
Kiritilgan raqam: 22
Dastur yakunlandi
```

Ушбу натижада фойдаланувчи томонидан тўғри маълумот киритилганлиги сабабли ҳеч қандай истисноли вазият ҳосил бўлмади ва дастур кутилган натижани берди. Агар истисно ҳосил бўлмаса, **except** блокидаги буйруқлар кетма-кетлиги бажарилмайди.

Бир вақтнинг ўзида юзага келиши мумкин бўлган бир нечта истисноли вазиятларни кўриб чиқамиз. Мисол учун фойдаланувчи томонидан киритилган иккита сон берилган. Биринчи сонни иккинчи сонга бўлинишини кўриб чиқайлик.

```
try:
    number1 = int(input("Birinchi sonni kiriting: "))
    number2 = int(input("Ikkinchi sonni kiriting: "))
    print("Bo‘linma: ", number1/number2)
except ValueError:
    print("Kiritilgan belgilar ketma-ketligini songa aylantirishda xatolik")
except ZeroDivisionError:
    print("Sonni 0 ga bo‘lish mumkin emas")
except Exception:
    print("Umumiy xatolik")
print("Dastur yakunlandi")
```

Ушбу дастурда фойдаланувчи томонидан иккинчи сон 0 киритилса, ZeroDivisionError хатолиги юзага келади ва фойдаланувчига хабар берилади.

Истисноларни бошқаришда finally блоки

Истисноли вазиятларни бошқаришда барча берилган буйруқлар кетма-кетлигини бажариб бўлинганидан сўнг бажариладиган буйруқлар кетма-кетлигини кўрсатиш мумкин. Бу буйруқлар кетма-кетлиги **finally** блокида ёзилади. Ушбу блок истисноли вазият юзага келиши ёки келмаслигидан қатъий назар бажариладиган блок ҳисобланади:

```
try:
    number = int(input("Son kiriting: "))
    print("Kiritilgan son: ", number)
except ValueError:
    print("Kiritilgan belgilar ketma-ketligini songa aylantirishda xatolik ")
finally:
    print("try bloki ishni yakunladi")
print("Dastur yakunlandi")
```

Одатда, бу блок файллар билан ишлаган вақтда барча буйруқлар бажариб бўлингандан сўнг очилган файлларни ёпиш учун ишлатилади.

Истиснони ўзгарувчилар ёрдамида белгилаш

Буйруқлар ва типлар ёрдамида истисноли вазиятларни бошқаришда истисно ҳосил бўлган вақтда истиснонинг хабарини

олиш мумкин. Бунинг учун истиснони ўзгарувчи ёки объект сифатида белгилаш лозим.

```
try:
    number = int(input("Son kiriting: "))
    print("Kiritilgan son:", number)
except ValueError as e:
    print("Xatolik:", e)
print("Dastur yakunlandi")
```

Дастурга нотўғри маълумот киритилганда қуйидагича ахборот чиқади:

```
Son kiriting: asd
Xatolik: invalid literal for int() with base 10: 'asd'
Dastur yakunlandi
```

Истисноли вазиятларни яратиш

Баъзан истисноли вазиятларни қўлда яратишга, яъни дастурчи томонидан яратилишига тўғри келади. Истисноли вазиятларни яратиш учун **raise** буйруғидан фойдаланилади.

```
try:
    number1 = int(input("Birinchi sonni kiriting: "))
    number2 = int(input("Ikkinchi sonni kiriting: "))
    if number2 == 0:
        raise Exception("Ikkinchi son 0 ga teng bo'lishi mumkin emas")
        print("Natija: ", number1/number2)
except ValueError:
    print("Ma'lumot noto'g'ri kiritildi")
except Exception as e:
    print(e)
print("Dasturni yakunlandi")
```

Юқоридаги дастур натижасини кўриб ўтамиз:

```
Birinchi sonni kiriting: 15
Ikkinchi sonni kiriting: 0
Ikkinchi son 0 ga teng bo'lishi mumkin emas
Dastur yakunlandi
```

Назарий саволлар:

1. Python дастурлаш тилида дастур ёзиш қоидасини тушунтириб беринг.
2. Ўзгарувчи ва ўзгармаснинг фарқи нимада?
3. Маълумотлар турлари деганда нимани тушунасиз?
4. Арифметик амалларни санаб беринг.?

5. Мантиқий амаллар ва солиштириш белгиларини тушунтириб беринг?
6. Битли амаллар деганда нимани тушунасиз?
7. Мантиқий ифодаларни ташкил этишда мантиқий амалларнинг вазифасини тушунтиринг?
8. Тармоқланувчи жараёнларни тушунтиринг?
9. Такрорланувчи жараён операториларни тушунтириб беринг?
10. Функциялар қайси хизматчи сўз ёрдамида эълон қилинади?
11. Локал ва глобал ўзгарувчиларни тушунтириб беринг?
12. Модуллар деганда нимани тушунасиз?
13. Истисноли вазиятларни тушунтириб беринг?
14. Хатоликлар турлари ва уларни бартараф этиш йўлларини тушунтиринг?
15. Хатоликни бошқариш блокиларини тушунтириб беринг?

III. Рўйхатлар, туркумлар ва луғатлар

3.1. Рўйхатлар

Python дастурлаш тилида маълумотларни массивлар ёрдамида тасвирлаш каби маълумотлар мажмуаси билан ишлаш учун рўйхатлар (**list**), туркумлар (**tuple**) ва луғатлар (**dictionary**) каби олдиндан аниқланган турлар мавжуд.

Рўйхат (**list**) маълумотлар тўпламини ёки кетма-кетлигини сақлайдиган маълумотлар тури ҳисобланади. Рўйхатларни ташкил этиш квадрат қавслар ([]) билан ифодаланади. Рўйхатнинг элементлари вергуллар билан ажратилган бўлиши шарт. Кўплаб дастурлаш тилларида бир қаторли ва бир турли маълумотлар мажмуаси мавжуд. Масалан: сонлардан ташкил топган тўплам:

```
numbers = [1, 2, 3, 4, 5]
```

Рўйхатни яратиш учун **list**() типидан ҳам фойдаланиш мумкин:

```
numbers1 = []
numbers2 = list()
```

Юқорида келтирилган рўйхатларни ташкил этиш буйруқлари ўхшаш ва улар бўш рўйхат яратади. Рўйхатни тузишда янги рўйхат тузувчи буйруқ бошқа бир рўйхатни қабул қилиши мумкин:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers2 = list(numbers)
```

Рўйхат элементларига мурожаат қилиш ҳудди массивлардагидек индекслар орқали амалга оширилади. Индекслар нол сонидан бошланади. Python дастурлаш тилининг афзалликларидан яна бири бу рўйхатнинг элементларига манфий сонлар ёрдамида мурожаат қилишдир. Бу рўйхатнинг охирги элементини индексини билмай туриб ҳам унга мурожаатни таъминлайди. Бошқача қилиб айтганда, рўйхатнинг охирги элементи –1 (минус бир) индексига эга бўлади, ундан олдинги элементи эса –2 (минус икки) индексини қабул қилади ва ҳоказолар.

```
numbers = [1, 2, 3, 4, 5]
print(numbers[0]) # 1
print(numbers[2]) # 3
print(numbers[-3]) # 3
numbers[0] = 125 # биринчи элементни ўзгартириш
print(numbers[0]) # 125
```

Агар рўйхатни бир хил қийматлар билан тўлдириш керак бўлиб қолса, унда кўпайтириш белгисидан фойдаланиш мумкин. Масалан: олтита бешдан иборат рўйхат ташкил этиш учун қуйидаги кодни ёзиш керак бўлади:

```
numbers = [5] * 6 #[5, 5, 5, 5, 5, 5]
print(numbers)
```

Рўйхатларни яратишда кетма-кетлик яратувчи функциядан фойдаланиб рўйхатни ташкил этиш мумкин. Кетма-кетлик яратиш функцияси юқоридаги мавзуларда келтирилган **range()** функциясидир. **range()** функцияси қуйидаги кўринишларга эга:

- **range**(stop): 0 дан **stop** гача бўлган барча сонларни қайтаради;
- **range**(start, stop): **start** (ҳам киради) дан бошлаб **stop** (кирмайди) гача бўлган оралиқдаги сонларни қайтаради.
- **range**(start, stop, step): **step** қадам ташлаган ҳолда, **start** дан бошлаб **stop** гача бўлган сонларни қайтаради.

```
numbers = list(range(10))
print(numbers) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers = list(range(2, 10))
print(numbers) # [2, 3, 4, 5, 6, 7, 8, 9]
numbers = list(range(10, 2, -2))
print(numbers) # [10, 8, 6, 4]
```

Масалан: қуйидаги иккита рўйхатнинг таърифлари ўхшаш, аммо **range()** функцияси ишлатилганлиги учун дастур коди камаяди:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers2 = list(range(1, 10))
```

Python дастурлаш тилида рўйхат фақат бир турдаги қийматлардан ташкил топиши шарт эмас. Бир рўйхатлар турли типдаги маълумотлар бўлиши мумкин. Масалан:

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

Рўйхат элементларини танлаш

Рўйхат элементларига **for** ва **while** такрорланувчи операторлар ёрдамида мурожаат қилиш мумкин. **for** такрорланиш операторида ишлатиш қуйидагича:

```
companies = ["Microsoft", "Google", "Oracle", "Apple"]
for item in companies:
print(item)
```

Рўйхат элементларини танлаш **while** такрорланиш оператори ёрдамида қуйидагича ишлатилади.

```
companies = ["Microsoft", "Google", "Oracle", "Apple"]
i = 0
while i < len(companies):
    print(companies[i])
    i += 1
```

Ушбу дастурда ишлатилган **len()** функцияси ёрдамида рўйхатнинг узунлиги топилади. Рўйхатнинг охирги элементигача ҳар бир элементига мурожаат ҳудди массивлардагидек амалга оширилади.

Рўйхатларни таққослаш

Икки рўйхат элементлари бир хил ва устма-уст тушса, бу рўйхатлар тенг бўлади.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers2 = list(range(1,10))
if numbers == numbers2:
    print("numbers equal to numbers2")
else:
    print("numbers is not equal to numbers2")
```

Юқорида келтирилган дастурда икки рўйхат тенг.

Рўйхатлар билан ишловчи методлар ва уларнинг вазифаси

Рўйхат элементларни бошқариш ва уларни устида турли амалларни бажарувчи методлар қуйида берилган:

- **append(item)**: рўйхатнинг охирига **item** элементини қўшади;
- **insert(index, item)**: рўйхатнинг **index** индекси сифатида **item** элементни жойлаштиради;
- **remove(item)**: рўйхатдаги **item** элементини ўчиради. **item** элементлари кўп бўлса, биринчи келган элементни ўчиради, агар **item** элементи топилмаса, **ValueError** истисноси юзага келади;
- **clear()**: рўйхатнинг барча элементини ўчиради, яъни рўйхатни тозалайди;
- **index(item)**: item элементининг индексини қайтаради. Агар **item** элементи мавжуд бўлмаса **ValueError** хатолиги юзага келади;
- **pop([index])**: индекси **index** га тенг бўлган элементни қайтаради ва рўйхатдан ўчиради. Агар рўйхатда индекси **index** га тенг бўлган элемент бўлмаса, рўйхатнинг охирги элементини қайтаради ва ўчиради.

- **count(item)**: рўйхатнинг элементлар сонини қайтаради;
- **sort([key])**: рўйхат элементларни саралайди. Саралаш доимо ўсиб бориш тарзида амалга оширилади. Агар ёрдамчи параметр мавжуд бўлса, метод параметрда кўрсатилган кўринишда саралайди.
- **reverse()**: рўйхатнинг барча элементларни тескари тартибга келтиради.

Юқорида келтирилган методлар рўйхатнинг методлари ҳисобланади. Рўйхатлар билан ишлайдиган баъзи функциялар қуйида келтирилган.

- **len(list)**: рўйхатнинг узунлигини қайтарувчи функция;
- **sorted(list, [key])**: тартибланган рўйхат қайтарувчи функция;
- **min(list)**: рўйхатдаги энг кичик элементини қайтариш;
- **max(list)**: рўйхатдаги энг катта элементини қайтариш.

Рўйхатга элемент қўшиш ва ўчириш

Рўйхатларда элементларни қўшиш учун юқоридаги келтирилган методлардан **insert()** ва **append()** методларидан фойдаланилади. Элементларни ўчириш учун эса **remove()**, **pop()** ва **clear()** методларидан фойдаланиш мумкин.

```
users = ["Temur", "Bobur"]
# ro‘yxatning oxirgi elementi sifatida qo‘shish
users.append("Aziza") # ["Temur", "Bobur", "Aziza"]
# ro‘yxatni 1 indeksi sifatida qo‘shish
users.insert(1, "Bilol") # ["Temur", "Bilol", "Bobur", "Aziza"]
# elementning indeksini aniqlash
i = users.index("Temur")
# indeksi i ga teng bo‘lgan elementni o‘chirish
removed_item = users.pop(i) # ["Bilol", "Bobur", "Aziza"]

last_user = users[-1]
# oxirgi elementni o‘chirish
users.remove(last_user) # ["Bilol", "Bobur"]
print(users)
# ro‘yxatni tozalash
users.clear()
```

Элементнинг мавжудлигини текшириш

Рўйхатда элемент мавжудлигини текшириш учун **in** хизматчи сўзидан фойдаланилади. Агар берилган қиймат рўйхатда мавжуд бўлса, **рост**, акс ҳолда, **ёлғон** қиймат қайтаради. Масалан:

```
companies = ["Microsoft", "Google", "Oracle", "Apple"]
item = "Oracle"                    # ўчириладиган элемент
if item in companies:
    companies.remove(item)
print(companies)
```

Юқоридаги дастурда киритилаётган элемент рўйхатда мавжудлиги сабабли, у элемент ўчириб ташланади.

Мавжуд элементнинг сонини аниқлаш

Берилган қийматни рўйхатда неча марта қатнашганини топиш учун **count()** методидан фойдаланилади.

```
users = ["Temur", "Bobur", "Aziza", "Temur", "Bilol", "Temur"]

users_count = users.count("Temur")
print(users_count)                 #3
```

Юқоридаги мисолда "Temur" қиймати рўйхатдаги сонини топиш кўрсатилган.

Рўйхат элементларини саралаш

Рўйхат элементларини саралаш учун **sort()** методидан фойдаланилади. Бу метод ёрдамида рўйхат элементларини ўсиб бориш тартибида саралаш мумкин. Масалан:

```
users = ["Temur", "Bobur", "Aziza", "Samad", "Bilol"]
users.sort()
print(users) # ["Aziza", "Bilol", "Bobur", "Samad", "Temur"]
```

Агар рўйхат элементларини камайиб бориш тартибида саралаш керак бўлса, унда рўйхат элементларини саралаб, рўйхат элементларни тескарилаш методидан фойдаланилади. Масалан:

```
users = ["Temur", "Bobur", "Aziza", "Samad", "Bilol"]
users.sort()
users.reverse()
print(users) # ["Temur", "Samad", "Bobur", "Bilol", "Aziza"]
```

Рўйхатларни саралашда **sort()** методидан ташқари, **sorted()** функцияси мавжуд. Бу функциянинг икки хил формаси мавжуд:

- **sorted(list): list** рўйхатини саралаш;
- **sorted(list, key): list** рўйхатини **key** функциясини қўллаган ҳолда саралаш.

```
users = ["Temur", "Bobur", "Aziza", "Samad", "Bilol"]
sorted_users = sorted(users, key=str.lower)
print(sorted_users)                #["Aziza","Bilol","Bobur","Samad", "Temur"]
```

Юқоридаги дастур мавжуд бўлган рўйхатни сараланган ҳолда бошқа бир сараланган рўйхат ҳосил қилади.

Рўйхатнинг энг катта ва энг кичик қийматлари

Python дастурлаш тилида **max()** ва **min()** функциялари ёрдамида рўйхат элементларининг энг катта ва энг кичик элементларини топиш мумкин.

```
numbers = [9, 21, 12, 1, 3, 15, 18]
print(min(numbers))          # 1
print(max(numbers))          # 21
```

Рўйхатни нусхалаш

Рўйхатларни нусхалашда рўйхатни ўзлаштириб турадиган турини яратиш керак. Бунда нусхаси олинаётган рўйхатни бошқа бир ўзгарувчига ўзлаштирилади. Бу ўзлаштирилишда агар бир рўйхатдаги элементлари ўзгартирилса, иккичи рўйхатда автоматик ўзгариш амалга оширилади:

```
users1 = ["Temur", "Bobur", "Aziza"]
users2 = users1
users2.append("Samad")
print(users1) # ["Temur", "Bobur", "Aziza", "Samad"]
print(users2) # ["Temur", "Bobur", "Aziza", "Samad"]
```

Бу ҳолат **shallow copy** (ясси нусхалаш) деб юритилади. Ясси нусхалаш, одатда, нотўғри амаллардан. Элементларни нусхалаш учун **deepcopy()** (чуқур нусхалаш) деб юритилувчи функциядан фойдаланилади. Бу функция **copy** модулида жойлашган.

```
import copy
users1 = ["Temur", "Bobur", "Aziza"]
users2 = copy.deepcopy(users1)
users2.append("Samad")
print(users1) # ["Temur", "Bobur", "Aziza"]
print(users2) # ["Temur", "Bobur", "Aziza", "Samad"]
```

Рўйхатнинг бир қисмини нусхалаш

Python дастурлаш тилида бутун рўйхатни эмас, балки рўйхатнинг айрим қисмини нусхалашга тўғри келади. Бундай ҳолатларда дастурлаш тилининг синтаксисидан фойдаланилади. Бу қуйидагича амалга оширилади:

- **list[:end]** – рўйхатни бошидан **end** (кирмайди) индексли элементгача нусхалайди;
- **list[start:end]** – рўйхатни **start** (киради) индексли элементидан **end** (кирмайди) индексли элементгача нусхалайди;

– **list[start:end:step]** – рўйхатни **start** (киради) индексли элементидан **end** (кирмайди) индексли элементгача **step** қадам билан нусхалайди. **step** – одатда 1 га тенг;

```
users = ["Temur", "Bobur", "Aziza", "Samad", "Tim", "Bilol"]
slice_users1 = users[:3] # с 0 по 3
print(slice_users1) # ["Temur", "Bobur", "Aziza"]
slice_users2 = users[1:3] # с 1 по 3
print(slice_users2) # ["Bobur", "Aziza"]
slice_users3 = users[1:6:2] # с 1 по 6 с шагом 2
print(slice_users3) # ["Bobur", "Samad", "Bilol"]
```

Рўйхатларни қўшиш

Рўйхатларни қўшиш учун (+) белгисини ишлатиш керак.

```
users1 = ["Temur", "Bobur", "Aziza"]
users2 = ["Temur", "Samad", "Tim", "Bilol"]
users3 = users1 + users2
print(users3) # ["Temur", "Bobur", "Aziza", "Temur", "Samad", "Tim", "Bilol"]
```

Рўйхатдан ташкил топган рўйхат

Рўйхатлар сатрлар, сонлар ва стандарт маълумотли қийматлардан ташкил топган бўлиши керак. Бундан ташқари, рўйхатнинг элементлари рўйхатлардан ташкил топиши мумкин. Масалан:

```
users = [["Temur", 29], ["Aziza", 33], ["Bobur", 27]]
print(users[0]) # ["Temur", 29]
print(users[0][0]) # Temur
print(users[0][1]) # 29
```

Ички рўйхат элементига мурожаат қилишда қўш индексдан фойдаланилади. Масалан: ички биринчи рўйхатнинг иккинчи элементига мурожаат қуйидагича амалга оширилади: **users[0][1]**.

Рўйхатда бажариладиган барча буйруқлар ички рўйхатли рўйхатларга ҳам амал қилади.

```
users = [["Temur", 29], [Aziza", 33], ["Bobur", 27]]
user = list()
user.append("Bilol")
user.append(41)
users.append(user)
print(users[-1]) # ["Bilol", 41]
users[-1].append("+998999999999")
print(users[-1]) # ["Bilol", 41, "+998999999999"]
users[-1].pop()
print(users[-1]) # ["Bilol", 41]
users.pop(-1)
```

```
users[0] = ["Samad", 18]
print(users) # [["Samad", 18], ["Aziza", 33], ["Bobur", 27]]
```

Ичма-ич жойлашган рўйхатларни қуйидагича консол ойнасига чиқарилади:

```
users = [[Temur", 29], ["Aziza", 33], ["Bobur", 27]]
for user in users:
    for item in user:
        print(item, end=" | ")
```

Консолда қуйидагича маълумот ҳосил бўлади:

```
Temur | 29 | Aziza | 33 | Bobur | 27 |
```

3.2. Туркумлар (Кортежлар)

Туркумлар рўйхатга ўхшаш тур бўлиб, туркумнинг рўйхатдан фарқи шундаки, рўйхатдаги элементлар устида турли амалларни бажариш мумкин. Туркумда эса бундай амалларни бажариш имкони мавжуд эмас. Туркумларни эълон қилиш учун оддий қавслардан фойдаланиш мумкин. Масалан:

```
user = ("Temur", 23)
print(user)
```

Бундан ташқари, доиравий қавслардан фойдаланмасдан, вергуллар билан ажратилган ҳолда, қийматлар тўпламини бериш билан ҳам туркумларни яратиш мумкин.

```
user = "Temur", 23
print(user)
```

Агар туркум битта элементдан ташкил топган бўлса, у ҳолда биринчи элементдан сўнг вергул қўйилади.

```
user = ("Temur",)
```

Рўйхатлардан туркумлар яратиш учун **tuple()** функциясидан фойдаланиш мумкин.

```
users_list = ["Temur", "Bobur", "Kate"]
users_tuple = tuple(users_list)
print(users_tuple)        # ("Temur", "Bobur", "Kate")
```

Туркумларда ҳам элементларга мурожаат қилиш ҳудди рўйхатлардаги каби бўлади. Туркумнинг бошидан мурожаат қилиш мусбат сонлар ёрдамида, охиридан мурожаат қилиш эса манфий сонлар ёрдамида амалга оширилади.

```
users = ("Temur", "Bobur", "Samad", "Kate")
print(users[0])                # Temur
print(users[2])                # Samad
print(users[-1])               # Kate
```

```
print(users[1:4])          # ("Bobur", "Samad", "Kate")
```

Туркумлар ўзгармас тур бўлганлиги сабабли, туркумнинг элементларини ўзгартириб бўлмайди. Туркумнинг элементини ўзгартириш учун, элементлар қийматларини оддий ўзгарувчиларга олиш керак. Ҳосил бўлган элементларнинг кераклигини ўзгартириб, кейин яна уни туркумга ўтказиш керак бўлади. Масалан:

```
user = "Musayev", "Xurshidbek", 34
print(user)
sname, gname, age = user
gname = "Shokirjon"
age = 31
user = sname, gname, age
print(user)
```

Python дастурлаш тилида функция бир вақтнинг ўзида бир нечта қиймат қайтара олади. Бунда функциядан қайтган қийматни туркум сифатида қабул қилиш мумкин.

```
def get_user():
    name = "Temur"
    age = 22
    is_married = False
    return name, age, is_married
user = get_user()
print(user[0])          # Temur
print(user[1])          # 22
print(user[2])          # False
```

len() функциясидан фойдаланиб, туркумнинг узунлигини топилади.

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

Туркум элементларига мурожаат

Туркум элементларга мурожаат қилиш учун **for** ва **while** такрорланиш операториларидан фойдаланиш мумкин. Қуйида **for** оператори ёрдамида туркумнинг элементларига мурожаат кўрсатилган:

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

while такрорланиш оператори ёрдамида мурожаат қуйидагича амалга оширилади.

```
user = ("Temur", 22, False)
i = 0
while i < len(user):
    print(user[i])
    i += 1
```

Туркум элементларининг ичида берилган қийматнинг мавжудлигини текшириш учун **in** хизматчи сўзидан фойдаланилади.

```
user = ("Temur", 22, False)
name = "Temur"

if name in user:
    print("Foydalanuvchi ismi Temur.")
else:
    print("Foydalanuvchining ismi Temur emas")
```

Мураккаб туркумлар

Мураккаб туркумлар деганда туркумнинг ўзи ички туркумлардан иборат бўлишига айтилади. Яъни, туркумнинг ҳар бир элементи туркумлардан ташкил топади.

```
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))
```

Юқоридаги дастурда мамлакатлар ва уларнинг аҳолиси, мамлакатларнинг шаҳарлари ва бу шаҳарда яшовчи аҳоли сони мураккаб туркумлар ёрдамида ифодаланган.

3.3. Луғатлар

Рўйхатлар ва туркумлар билан бир қаторда, Python дастурлаш тилида **луғат (dictionary)** деб аталувчи қўшимча маълумотлар тузилмаси мавжуд. Рўйхатлар каби луғат ҳам ўзида элементлар коллекциясини сақлайди. Луғатдаги элементлар иккига бўлинади,

яъни калит ва қиймат кўринишга эга (бу маълумотлар тузилмаси PHPда ассоциатив массий, C# дастурлаш тилида луғат).

Луғатлар қуйидагича ифодаланади:

```
dictionary = { калит1:қиймат1, калит2:қиймат2,....}
```

Масалан:

```
users = {1: "Temur", 2: "Bobur", 3: "Bilol"}
elements = {"Au":"Oltin","Fe":"Temir","H":"Vodorod","O":"Kislorod"}
```

Юқоридаги дастур кодида **users** номли луғатда калитлар сифатида рақамлар, қийматлар сифатида эса сатрли маълумотлар келтирилган. Луғатларда калит ва қийматлар бир хил типда бўлиши шарт эмас. Луғатларнинг яна бир имконияти борки, бу имконият ёрдамида калитлар турли хилдаги турларда бўлиши мумкин. Масалан:

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

Элементларга эга бўлмаган, яъни бўш тўплам қуйидагича ёзилади.

```
objects = {}
```

ёки қуйидагича

```
objects = dict()
```

Рўйхатларни луғатга айлантириш

Луғатлар ва рўйхатларнинг тузилиши турлича бўлса-да, бир нечта алоҳида рўйхатларни dict() функцияси ёрдамида луғат кўринишга келтириш мумкин. Бунинг учун рўйхат ичма-ич жойлашган кўринишда бўлиши керак ва ичма – ич жойлашган рўйхатларда иккитадан элементга эга бўлиши талаб этилади. Луғатга ўтказилаётган рўйхат элементларидан биринчи келган элемент луғатнинг калити, иккичи келган элемент эса калитнинг қиймати сифатида қабул қилади.

```
users_list = [
        [“+111123455”, “Temur”],
        [“+384767557”, “Bobur”],
        [“+958758767”, “Aziza”]
]
users_dict = dict(users_list)
print(users_dict)
# {"+111123455": "Temur", "+384767557": "Bobur", "+958758767": "Aziza"}
```

Шу йўл билан икки ўлчовли туркумларни ҳам луғат кўринишига ўтказиш мумкин бўлади:

```
users_tuple = (
```

```
        ("+111123455", "Temur"),
        ("+384767557", "Bobur"),
        ("+958758767", "Aziza")
)
users_dict = dict(users_tuple)
print(users_dict)
```

Элементларга мурожаат қилиш ва уларни ўзгартириш

Луғат элементларига мурожаат қилиш калит ёрдамида амалга оширилади.

```
dictonary[kalit]
```

Масалан: луғат элементига мурожаат қилиш:

```
users = {
        "+11111111": "Temur",
        "+33333333": "Bobur",
        "+55555555": "Aziza"
}
print(users["+11111111"]) # Temur
users["+33333333"] = "Bobur Smith"
print(users["+33333333"]) # Bobur Smith
```

Луғатда йўқ калит ёрдамида луғатга элементлар қўшиш мумкин эмас.

```
users["+4444444"] = "Samad"
```

Луғатда мавжуд бўлмаган калитга мурожаат қилинса, Python дастурлаш тили KeyError юзага келтиради.

```
user = users["+4444444"] # KeyError
```

Бу ҳолатнинг олдини олиш учун элементларга мурожаат қилишдан олдин, изланаётган элемент бор ёки йўқлигини текшириш керак. Элемент бор ёки йўқлигини текшириш учун **in** хизматчи сўзидан фойдаланилади ва қуйидагича амалга оширилади:

```
key = "+4444444"
if key in users:
        user = users[key]
        print(user)
else:
        print("Element mavjud emas.")
```

Луғатдаги элементлар қийматини олиш учун **get()** методидан фойдаланилади. Бу методнинг икки хил кўриниши мавжуд бўлиб, улар қуйида келтирилган:

- **get(key):** Луғат элементларидан калити **key** га тенг бўлган элементни қайтаради, агарда кўрсатилган калитда қиймат мавжуд бўлмаса, **None** ни қайтаради.
- **get(key, default):** Луғат элементларидан **key** га тенг бўлган элементни қайтаради, агарда кўрсатилган калитда қиймат мавжуд бўлмаса, **default** ни қайтаради.

```
key = "+55555555"
user = users.get(key)
user = users.get(key, "Unknown user")
```

Элементни ўчириш

Луғатдаги элементларни ўчириш учун **del** операторидан фойдаланилади. Элементларни ўчиришда калитдан фойдаланиш зарур

```
users = {
        "+11111111":"Temur",
        "+33333333":"Bobur",
        "+55555555":"Aziza"
}
del users["+55555555"]
print(users)
```

Яна бир жиҳатни эътиборга олиш керакки, агарда кўрсатилган калит луғатда мавжуд бўлмаса, **KeyError** истисноси юзага келади. Шу сабабли калитни мавжуд ёки мавжуд эмаслигини текшириш тавсия этилади. Масалан:

```
key = "+55555555"
if key in users:
        user = users[key]
                del users[key]
        print(user, "o'chirilgan")
else:
        print("Element topilmadi")
```

Элементларни ўчиришнинг яна бир йўли **pop()** методидир. Бу метод ёрдамида ўчирилаётган элемент қийматини аввал ўзгарувчига узатади ва уни кейин ўчиради.

- **pop(key):** key калитли элемент ўчирилади. Агарда берилган калит мавжуд бўлмаса, **KeyError** истисноси юзага келади.

– **pop(key, default):** key калитли элемент ўчирилади. Агарда берилган калитдаги элемент мавжуд бўлмаса, **default** қиймати қайтарилади.

```
users = {
        "+11111111":"Temur",
        "+33333333":"Bobur",
        "+55555555":"Aziza"
}
key = "+55555555"
user = users.pop(key)
print(user)
user = users.pop("+4444444", "Unknown user")
print(user)
```

Агарда луғатнинг барча элементларини ўчиришга тўғри келса, у ҳолда **clear()** методидан фойдаланилади.

```
users.clear()
```

Луғатларни бирлаштириш ва нусхалаш

Луғатдан нусха олиш учун **copy()** методидан фойдаланилади. Бу метод ёрдамида янги луғат яратиш мумкин.

```
users = {
        "+1111111": "Temur",
        "+33333333": "Bobur",
        "+55555555": "Aziza"
}

users2 = users.copy()
```

Луғатларни бирлаштириш учун **update()** методидан фойдаланилади. Бу метод ёрдамида биринчи методга параметр сифатида келган луғат бирлаштирилади.

```
users = {
        "+1111111": "Temur",
        "+33333333": "Bobur",
        "+55555555": "Aziza"
}
users2 = {
        "+22222222": "Samad",
        "+66666666": "Kate"
}
users.update(users2)
print(users)
print(users2)
```

Юқоридаги дастурда **users2** луғати ўзгармасдан қолади. **users** луғати эса ўзгаради. Агар бирлаштирилаётган луғатларни ўзгартиришсиз қолдириш керак бўлса, у ҳолда қуйидагича бажарилади:

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

Луғат элементларига мурожаат қилиш

Луғат элементларига мурожаат қилиш учун луғатнинг калити ёрдамида ва цикл операторлари ёрдамида амалга оширилади. Масалан,

```
users = {
    "+11111111": "Temur",
    "+33333333": "Bobur",
    "+55555555": "Aziza"
}
for key in users:
    print(key, " - ", users[key])
```

Юқоридаги дастурда луғатнинг калити олиниб, шу калит ёрдамида луғатнинг элементига мурожаат қилинади. Луғатларда калит ва шу калитга тегишли бўлган қийматларни олиш ҳам мумкин. Бунинг учун **items()** методидан фойдаланилади.

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

items() методи луғатнинг элементларини туркумлар сифатида қайтаради. Масалан:

```
for k in users.items():
    print(k)
```

Луғат элементлари калитларининг ўзини ҳам олиш мумкин. Бунинг учун **keys()** методидан фойдаланилади.

```
for key in users.keys():
    print(key)
```

Луғат элементларининг қийматларини калит каби олиш мумкин. Қийматларини олиш учун **value()** методи ёрдам беради.

```
for value in users.value( ):
    print(value)
```

Комплекс луғатлар

Луғатлар сонлар сатрлардан ташқари, мураккаб бўлган объектларни ўзида сақлаш имкониятига эга. Унда бир калит ўзида луғат сақлаши мумкин. Масалан:

```
users = {
        "Temur":{
                "phone": "+971478745"
                "email": "Temur12@gmail.com"
        },
        "Bobur":{
                "phone": "+876390444"
                "email": "Bobur@gmail.com"
                "skype": "Bobur123"
        }
}
```

Ичма-ич жойлашган луғатларнинг элементларига мурожаат қилиш учун мос равишда иккиталик калит ишлатиш керак бўлади:

```
old_email = users["Temur"]["email"]
users["Temur"]["email"] = "superTemur@gmail.com"
```

Агар луғатларга ва мавжуд бўлмаган калитга мурожаат қилинса, **KeyError** истисноси юзага келади.

```
Temur_skype = user["Temur"]["skype"]                    #KeyError
```

Хатоликни чеклаб ўтиш учун луғатда калит мавжуд ёки йўқлигини текшириб олиш керак.

```
key = "skype"
if key in users["Temur"]:
        print(users["Top"]["key"])
else:
        print("skype kaliti topilmadi")
```

3.4. Тўпламлар

Python дастурлаш тилидаги маълумотлар мажмуасидан яна бири бу **тўплам (set)** дир. Тўпламларни аниқлаш учун фигурали қавслардан фойдаланилади:

```
users = {"Temur", "Bobur", "Aziza", "Temur"}
print(users)#{"Temur","Bobur","Aziza"}
```

Эътибор бериб қаралса, берилган тўпламда “Temur” элементи иккита бўлишига қарамай, чиқариш функцияси (**print**) уни бир марта чиқаради. Чунки, тўпламларда бир хил элементлар фақатгина бир марта қабул қилинади. Тўпламларини аниқлашнинг иккинчи усули бу функция орқали аниқлаш ҳисобланиб, бу функция **set()** деб номланади:

```
users3 = set([“Mike”, “Bilol”, “Ted”])
```

Асосан, **set()** функцияси бўш тўпламларни яратиш учун қўлланилади:

```
users = set()
```

Тўпламларнинг узунлигини аниқлаш учун **len()** функциясидан фойдаланилади:

```
users = ("Temur", "Bobur", "Aziza")
print(len(users))          # 3
```

Элемент кўшиш

Тўпламга элемент қўшиш учун **add()** методидан фойдаланилади:

```
users = set()
users.add("Samad")
print(users)
```

Элементларни ўчириш

Элементларни ўчириш учун ўчирилиши керак бўлган элементга мурожаат сифатида **remove()** методидан фойдаланилади. Агар кўрсатилган элемент мавжуд бўлмаса, дастур хатолик кўрсатади. Шунинг учун дастурда ўчирилиши керак бўлган элемент тўпламда мавжудлигини текшириш керак бўлади. Масалан:

```
users = {"Temur", "Bobur", "Aziza"}
user = "Temur"
if user in users:
    users.remove(user)
print(users)          # {"Bobur", "Aziza"}
```

remove() методидан ташқари тўплам элементларини ўчириш учун **discard( )** методидан фойдаланиш ҳам мумкин, бу методнинг афзаллик томони агар ўчирилаётган элемент топилмаса, истисно рўй беради:

```
user = "Tim"
user.discard(user)
```

Тўпламлардан элементларни ўчиришдан ташқари, тўпламни тозалаш буйруғи ҳам мавжуд. Бу буйруқни ишлатиш учун **clear()** методидан фойдаланилади:

```
users.clear()
```

Тўплам элементларига мурожаат

Тўплам элементларига мурожаат қилиш учун **for** операторидан фойдаланиш мумкин. **for** оператори тўпламнинг ҳар

бир элементига мурожаат қилиш учун ишлатилади. Бунда тўпламнинг ҳар бир элементига мурожат қилиш учун **in** хизматчи сўзи ишлатилади. Масалан

```
users = {"Temur", "Bobur", "Aziza"}
for user in users:
    print(user)
```

Тўпламлар устида амаллар

copy() методи ёрдамида берилган тўплам таркибидаги элементларнинг нусхасини олиб бошқа тўплам яратиш мумкин бўлади:

```
users = {"Temur", "Bobur", "Aziza"}
users3 = users.copy()
```

union() методи икки тўпламни бирлаштириш учун хизмат қилади. Бу метод объект сифатида ишлатилган тўпламга параметр сифатида келган тўпламни қўшади. Масалан:

```
users = {"Temur", "Bobur", "Aziza"}
users2 = {"Samad", "Kate", "Bobur"}
users3 = users.union(users2)
print(users3)              # {"Bobur", "Aziza", "Samad", "Kate", "Temur"}
```

Тўпламнинг устида кесишиш амалини бажариш мумкин. Бу амал бажарилганда икки тўпламда мавжуд бўлган элементларнинг фақатгина бири сақланиб қолади. **intersection()** методи ёрдамида иккита тўплам устида кесишиш амалини бажариш мумкин:

```
users = {"Temur", "Bobur", "Aziza"}
users2 = {"Samad", "Kate", "Bobur"}
users3 = users.intersection(users2)
print(users3)      # {"Bobur"}
```

intersection() методининг ўрнига мантиқий кўпайтириш амалидан фойдаланиш мумкин. Масалан:

```
users = {"Temur", "Bobur", "Aziza"}
users2 = {"Samad", "Kate", "Bobur"}
print(users & users2)          # {"Bobur"}
```

Тўпламлар устида бажариладиган амаллардан бири – бу айириш амали ҳисобланади. Айириш амалини ишлаш тартиби қуйидагича биринчи тўпламда мавжуд, лекин иккинчи тўпламда мавжуд бўлмаган элементларни қайтаради. Тўпламларнинг айирмасини ҳосил қилиш учун **difference()** методи ёки арифметик айириш амалидан фойдаланиш мумкин:

```
users = {"Temur", "Bobur", "Aziza"}
users2 = {"Samad", "Kate", "Bobur"}
users3 = users.difference(users2)
print(user3)                # {"Temur", "Aziza"}
print(users – users2)              # {"Temur", "Aziza"}
```

Тўпламлар орасидаги муносабат

issubset() методи тўпламнинг бошқа бир тўпламнинг бир қисми ёки бир қисми эмаслигини аниқлайди:

```
users = {"Temur", "Bobur", "Aziza"}
superusers = {"Samad", "Temur", "Bobur", "Aziza", "Greg"}
print(users.issubset(superusers))          # True
print(superusers.issubset(users))          # False
```

issuperset() методи **issubset()** методига тескари метод бўлиб, тўпламга бошқа бир тўплам тегишли ёки тегишли эмаслигини аниқлайди:

```
users = {"Temur", "Bobur", "Aziza"}
superusers = {"Samad", "Temur", "Bobur", "Aziza", "Greg"}
print(users.issuperset(superusers))        # False
print(superusers.issuperset(users))        # True
```

Ўзгармас тўпламлар

Тўпламлар ҳам ўзгармаслар сингари ўзининг элементларини ўзгартирмаслиги мумкин. Бунинг учун тўпламни эълон қилинаётганда ёки бундай тўплам яратилаётганда **frozenset** функциясидан фойдаланиш мумкин:

```
users = frozenset({"Temur", "Bobur", "Aziza"})
```

Бундай тўпламларда ҳеч қандай элемент қўшиш ёки ўчириш мумкин эмас. Шуни инобатга олган ҳолда, **frozenset** кўринишдаги тўпламлар қуйидаги функциялар ва методлар билан ишлай олади:

- **len(s):** тўпламнинг элементлар сонини қайтаради;
- **x in s:** x элементи s тўпламда мавжуд бўлса рост (**True**), акс ҳолда, ёлғон (**False**) қиймат қайтаради;
- **x not in s:** x элементи s тўпламда мавжуд бўлмаса, рост (**True**), акс ҳолда, ёлғон (**False**) қиймат қайтаради;
- **s.issubset(t):** агар t тўплами s тўпламининг элементлардан ташкил топган бўлса, **True** қиймат қайтаради;

- **s.issuperset(t):** агар s тўплам элементларидан t тўплам элементларини ҳосил қилиш мумкин бўлса, **True** қиймат қайтаради;
- **s.union(t):** s ва t тўплам элементлари бирлашмасини қайтаради;
- **s.intersection(t):** s ва t тўплам элементлари кесишмасини қайтаради;
- **s.difference(t):** s ва t тўплам айирмасини қайтаради;
- **s.copy():** s тўплам нусхасини қайтаради.

Назорат саволлари:

1. Рўйхатларни ташкил этишни тушунтириб беринг.
2. Рўйхатнинг узунлигини топиш мумкинми?
3. Тўпламнинг узунлиги қайси функция ёрдамида топилади?
4. Луғатлар қайси хизматчи сўз ёрдамида эълон қилинади?
5. **tuple** хизматчи сўзи ёрдамида қайси маълумотлар мажмуаси эълон қилинади?
6. Ўзгармас тўпламларни эълон қилишни тушунтириб беринг.
7. Ўзгармас тўпламлар қайси функция орқали эълон қилинади?
8. Қайси маълумотлар мажмуаси калит ва қийматлардан ташкил топади?

IV. Файллар билан ишлаш

4.1. Файлларни очиш ва ёпиш операторлари

Python дастурлаш тили замонавий дастурлаш тиллари каби турли хилдаги файллар билан ишлаш хусусиятига эга. Замонавий дастурлаш тилларидаги файлларни мантиқий икки турга бўлиш мумкин: матнли ва бинар. Матнли файллар ўз номидан келиб чиққан ҳолда, матнлардан ташкил топган бўлади. Бу турга csv, txt, html кенгайтмали файлларни мисол қилиш мумкин. Бинар файлларга – аудио, видео, расм ва бошқа шу турдаги файлларни мисол қилиш мумкин. Файлларнинг турига қараб улар билан ишлаш фарқ қилади. Файллар билан ишлаётганда бир нечта амаллар кетма-кетлигини амалга ошириш керак бўлади:

- Файлни **open()** методи ёрдамида очиш;
- **read()** методи ёрдамида файлни ўқиш ёки **write()** методи ёрдамида файлга ёзиш;
- **close()** методи ёрдамида файлни ёпиш

Файллар билан ишлаш учун аввал файлни **open()** функцияси ёрдамида очиш керак бўлади, бу функциянинг тўлиқ кўриниши қуйида кўрсатилган:

open(file, mode)

Функциянинг биринчи параметри файлга йўл кўрсатади. Файлга йўл ўзгармас бўлиши мумкин, яъни сақланаётган дискнинг номидан бошланиши мумкин, Масалан, **C://somedir/somefile.txt**. ёки нисбий жойлашган ҳам бўлиши мумкин, **somedir/somefile.txt** – бу ҳолатда файл излаш ёки жойлаштириш **Python** коди жойлашган жойга нисбатан амалга оширилади.

Иккинчи ўринда жойлашган параметр **mode** файл устида бажариладиган амални билдиради. Бу амални файл билан ишлаш режими ҳам деб юритилади. Умуман олганда, файллар устида бажариладиган режимлар 4 та ҳисобланади:

- **r (Read),** файлни ўқиш учун очилади. Агар файл мавжуд бўлмаса, **FileNotFoundError** истисноси юзага келади.
- **w (Write),** файлга маълумотларни ёзиш учун очади. Агар файл мавжуд бўлмаса, у ҳолда файл яратилади. Агарда кўрсатилган номдаги файл мавжуд бўлса, файл қайтадан яратилади ва олдинги файлнинг маълумотлари ўчиб кетади.

- **a (Append),** файл тўлдириш учун очилади (якунига етказиш) учун очилади. Агарда файл мавжуд бўлмаса, у ҳолда файл яратилади. Агар кўрсатилган номдаги файл мавжуд бўлса, янги киритилган маълумотлар файлнинг охиридан бошлаб жойлаштирилади.
- **b (Binary),** бинар файллар билан ишлаш учун фойдаланилади. **w** ёки **r** режимлари билан бирга қўлланилади.

Файллар устида амаллар бажариб бўлинганидан сўнг **close()** методи ёрдамида жорий файл ёпилиши талаб қилинади. Масалан: ёзиш учун "hello.txt" файлини очиб кўрамиз:

```
myfile = open("hello.txt", "w")
myfile.close()
```

Файллар билан ишлаётган вақтда турли хилдаги истисноли вазиятларга дуч келиш мумкин. Масалан, файлга кириш ҳуқуқи йўқ бўлса ва ҳоказолар. Бу ҳолатда дастур хатоликларга дуч келади ва дастурнинг бажарилиши **close()** методигача бормаслиги мумкин. Бундай вазиятларда истисноларни бошқаришдан фойдаланиш мумкин бўлади:

```
try:
    somefile = open("hello.txt", "w")
    try:
        somefile.write("hello world")
    except Exception as e:
        print(e)
    finally:
        somefile.close( )
except Exception as ex:
    print(ex)
```

Ушбу ҳолатда файл устида амаллар ичма-ич жойлаштирилган **try** блоки ичида амалга оширилади. Агарда бирон-бир истисно юзага келган тақдирда ҳам **finally** блокида файл ёпилади. Файллар билан ишлашда янада қулайроқ кўринишга эга бўлган **with** конструкциясидан фойдаланиш мумкин.

```
with open(file, mode) as file_obj:
    buyruqlar to'plami
```

Бу конструкция очилган файлни **file_obj** ўзгарувчига узатади ва буйруқлар тўплами бажарилади. Буйруқлар кетма-кетмалиги бажарилганидан сўнг автоматик тарзда файл ёпилади. Агарда **with**

блокида бирон-бир истисноли вазият юзага келган тақдирда ҳам файл ёпилади. Юқоридаги мисол with блоки ёрдамида қуйидагича ёзилади:

```
with open("hello.txt", "w") as somefile:
    somefile.write("hello world")
```

4.2. Матнли файллар

Матнли файлларга маълумотлар ёзиш

Матнли файллар w режимидан фойдаланиб очилади. Очилган файлга маълумотларни сақлаш ёки файлга маълумотларни қўшиш учун **write(str)** методидан фойдаланилади. Шуни ёдда сақлаш керакки, файлларга маълумотларни сатр кўринишга ўтказиб ёзилади, шу сабабдан файлга бирор-бир сонни ёзишга тўғри келса, сон аввалambор сатрли кўринишга келтирилади. Масалан, файлга маълумот ёзиш:

```
with open("hello.txt", "w") as file:
    file.write("hello world")
```

Юқорида келтирилган дастурдаги файл дастур коди жойлашган каталогда ҳосил бўлади. Яратилган файлнинг номи "hello.txt" номли файл ҳисобланади. Ўзгартиришлар қилиш мақсадида бу файлни ихтиёрий файл таҳрирловчи дастурларда очиш мумкин. Юқоридаги дастурга ўзгартириш киритиб яратилган файлга қўшимча маълумот ёзиш мумкин:

```
with open("hello.txt", "a") as file:
    file.write("\ngood bye, world")
```

Файлда жойлаштирилган маълумотларнинг охирги сатрини тўлдириш ва маълумотларни ёзиш учун **file** ўзгарувчиси ташкил этилади. Дастурни иккинчи сатрда ёзилган маълумотлар кетма-кетлигида "\n" амали келтирилган. Бу амал сатрни тугатиб, янги сатрдан маълумотларни ёзишни билдиради. Бу кетма-кетликлар бажарилиб бўлинганидан сўнг "hello.txt" файли қуйидаги кўринишга келади:

```
hello world
good bye, world
```

Файлларга ёзишнинг яна бир йўли бу стандарт **print()** функциясидир. Бу функция ёрдамида қуйидагича кўринишда файлларни ёзиш мумкин:

```
with open("hello.txt", "a") as hello_file:
    print("Hello, world", file=hello_file)
```

Юқорида келтирилган дастурда маълумотларни файлга чиқариш кўрсатиб ўтилган. Бу ерда **print** функцияси **file** параметр–ўзгарувчиси орқали файлнинг номи кўрсатилмоқда. Биринчи параметр сифатида файлга чиқариладиган маълумотлар келтирилган.

Файлларни ўқиш

Файллар **r (Read)** режими ёрдамида ўқиш учун очилади. Файлни ўқиш учун очилганидан сўнг турли методлар ва функциялар ёрдамида файл маълумотларини ўқиш мумкин бўлади:

- **readline(),** файлдан бир сатрни ўқиш учун қўлланилади;
- **read(),** файлнинг барча маълумотларни бир сатрда ўқишдир;
- **readlines(),** файлнинг барча маълумотларини сатрма-сатр рўйхат тарзида ўқийди.

Масалан, файлдаги маълумотларни сатрма-сатр ўқиш қуйидагича амалга оширилади:

```
with open("hello.txt", "r") as file:
    for line in file:
        print(line, end="")
```

Одатда, ҳар бир қаторни ўқиш учун **readline()** методидан фойдаланмасдан ҳам файлни ўқиб олиш мумкин, чунки бу усулда маълумотлар сатрма-сатр ўқилади. Шунинг учун циклда **readline()** методини чақиришнинг маъноси йўқ. Сатрлар "\n" белгиси билан ажратилганлиги сабабли кейинги қаторга ортиқча маълумот ўтказишни бартараф этиш учун **print()** функциясига **end=""** қиймати узатилди.

Аниқ бир қаторларни ўқиш учун **readline()** методи қуйидагича ишлатилади:

```
with open("hello.txt", "r") as file:
    str1 = file.readline()
    print(str1, end="")
    str2 = file.readline()
    print(str2)
```

readline() методини **while** циклида файлларнинг қаторларини ўқиш учун ишлатилиши мумкин:

```
with open("hello.txt", "r") as file:
    line = file.readline()
    while line:
        print(line, end="")
        line = file.readline()
```

Агар ўқитаётган файл кичик бўлса, **read()** методи ёрдамида файлни бирданига ўқиш мумкин:

```
with open("hello.txt", "r") as file:
    content = file.read()
    print(content)
```

Файлнинг барча қаторларини бирданига ўқиш имконияти мавжуд. Бунинг учун **readlines()** методидан фойдаланилади:

```
with open("hello.txt", "r") as file:
    contents = file.readlines()
    str1 = contents[0]
    str2 = contents[1]
    print(str1, end="")
    print(str2)
```

Файлни ўқитаётганда унинг кодлаш тизими ASCII тизими билан мос келмаслиги мумкин. Бундай ҳолда, **encoding** параметри ёрдамида кодлаш тизими белгиланади:

```
filename = "hello.txt"
with open(filename, encoding="utf8") as file:
    text = file.read()
```

4.3. CSV файллари

Маълумотларни қулай шаклда сақлайдиган ва кенг тарқалган файл форматларидан яна бири – бу **CSV** форматидир. CSV файлидаги ҳар бир сатр вергуллар билан ажратилган алоҳида устунлардан иборат алоҳида ёзувни ифодалайди. Шунинг учун бу кенгайтмали файл "**Comma Separated Values**" деб номланади. CSV форматли матн кўринишдаги файл бўлса-да, Python у билан ишлашни соддалаштириш учун махсус ўрнатилган CSV модулини тақдим этади. Қуйидаги мисолда CSV кўринишдаги файлдан фойдаланиш келтириб ўтилган:

```
import csv
FILENAME = "users.csv"
users = [["Temur", 28], ["Aziza", 23], ["Bobur", 34]]
with open(FILENAME, "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(users)
with open(FILENAME, "a", newline="") as file:
    user = ["Samad", 31]
    writer = csv.writer(file)
    writer.writerow(user)
```

Файлга икки ўлчовли рўйхат ёзилади – ҳар бир сатр битта фойдаланувчини ифодалайдиган ҳақиқий жадвал ҳисобланади. Ҳар бир фойдаланувчи эса иккити майдон – исм ва ёшни ўз ичига олади. Яъни, учта сатр ва икки устундан иборат кўринишдаги жадвал. Файлни ўқиш учун **reader** объектини яратиш лозим:

```
import csv
FILENAME = "users.csv"
with open(FILENAME, "r", newline="") as file:
        reader = csv.reader(file)
        for row in reader:
                print(row[0], " - ", row[1])
```

reader объекти маълумотларни қабул қилганидан сўнг, маълумотларни консолга чиқариш учун циклдан фойдаланилган:

```
Temur - 28
Aziza - 23
Bobur - 34
Samad - 31
```

Файлларда луғатлардан фойдаланиш

Юқоридаги мисолда ҳар бир ёзув ёки сатр алоҳида рўйхатни ўз ичига олган. Бундан ташқари, CSV модуллари луғатлар билан ишлаш учун махсус қўшимча хусусиятларга эга. Хусусан, **csv.DictWriter()** функцияси файлга ёзиш имконини берувчи **writer** объектини қайтаради. **csv.DictReader()** функцияси эса файлдан ўқиш учун **reader** объектини қайтаради.

```
import csv
FILENAME = "users.csv"
users = [
        {"name": "Temur", "age": 28},
        {"name": "Aziza", "age": 23},
        {"name": "Bobur", "age": 34}
]
with open(FILENAME, "w", newline="") as file:
        columns = ["name", "age"]
        writer = csv.DictWriter(file, fieldnames=columns)
        writer.writeheader()
        # bir necha qatorni yozish
        writer.writerows(users)
        user = {"name" : "Samad", "age": 41}
        # bitta qatorni yozish
        writer.writerow(user)
with open(FILENAME, "r", newline="") as file:
        reader = csv.DictReader(file)
```

```
for row in reader:
    print(row["name"], "-", row["age"])
```

Қаторлар **writerow()** ва **writerows()** методлари ёрдамида ҳам ёзилади. Аммо энди ҳар бир сатр алоҳида луғатдир ва бундан ташқари, устун сарлавҳалари **writeheader()** усулидан фойдаланиб ёзилади ва иккинчи параметр сифатида **csv.DictWriter()** методи устунлар тўплами узатилади.

4.4. Бинар файллар

Бинар файлларни матнли файллардан фарқли ўлароқ, маълумотлар байтлар кўринишда сақланади. Pythonда улар билан ишлаш учун ўрнатилган модул **pickle** талаб қилинади. Ушбу модул иккита методни тақдим этади:

- **dump(obj, file)** – **obj** объектини **file** бинар файлга ёзилади;
- **load(file)** – бинар файлдан маълумотларни объектга кўчиради.

Бинар файлни ўқиш ва ёзишда ("w") ва ("r") режимларидан ташқари "b" режимидан фойдаланиш керак бўлади. Масалан:

```
import pickle
FILENAME = "user.dat"
name = "Temur"
age = 19
with open(FILENAME, "wb") as file:
    pickle.dump(name, file)
    pickle.dump(age, file)
with open(FILENAME, "rb") as file:
    name = pickle.load(file)
    age = pickle.load(file)
    print("Ism:", name, "\tYosh:", age)
```

Ҳудди шундай ҳолатда файлдан бир нечта объектни сақлаш ва олиш мумкин:

```
import pickle
FILENAME = "users.dat"
users = [["Temur", 28, True],["Aziza", 23, False],["Bobur", 34, False]]
with open(FILENAME, "wb") as file:
    pickle.dump(users, file)
with open(FILENAME, "rb") as file:
    users_from_file = pickle.load(file)
    for user in users_from_file:
        print("Ism:",user[0],"\tYosh:",user[1],"\tOila qurgan:", user[2])
```

dump методи ёрдамида қандай объект ёзилган бўлса, файлни ўқиётганда **load** методи ёрдамида ҳудди шундай объект қайтаради.

4.5. shelve модули

Python дастурлаш тилидаги бинар файллар билан ишлаш учун яна битта **shelve** модулидан фойдаланиш мумкин. Объектларни махсус калит билан файлга сақланади. Кейинчалик бу калит ёрдамида аввалдан сақланган объектни файлдан чиқариб олиш мумкин. **shelve** модули билан ишлаш жараёни, файлларни сақлаш ва чиқариш учун калитлардан фойдаланадиган луғатлар билан ишлашни эслатади. Файлларни очиш учун **shelve** модули **open()** функцияси ишлатилади:

```
open(faylning_manzili [,flag="c" [,protocol=None [,writeback=False]]])
```

Функциядаги **flag** параметри қуйидаги қийматларни қабул қилиши мумкин:

- **c** – файл ўқиш ва ёзиш учун очилади. Агар файл мавжуд бўлмаса, файл яратилади;
- **r** – файл фақат ўқиш учун очилади;
- **w** – файл ёзиш учун очилади;
- **n** – файл ёзиш учун очилади. Агар файл мавжуд бўлмаса, файл яратилади, мавжуд бўлса, у ҳолда қайта ёзилади.

Файлларга мурожаатни тугатиш учун **close()** методидан фойдаланилади:

```
import shelve
d = shelve.open(filename)
d.close()
```

Шу билан бир қаторда **with** оператори ёрдамида файлни очиш мумкин. Файлда бир нечта объектлар яратиш ва сақлаш қуйидаги мисолда берилган.

```
import shelve
FILENAME = "states2"
with shelve.open(FILENAME) as states:
    states["London"] = "Great Britain"
    states["Paris"] = "France"
    states["Berlin"] = "Germany"
    states["Madrid"] = "Spain"
with shelve.open(FILENAME) as states:
    print(states["London"])
    print(states["Madrid"])
```

Маълумотни ёзишда маълум бир калит учун қиймат белгилашни ўз ичига олади.

```
states["London"] = "Great Britain"
```

Файлдан қийматларни ўқиш учун калит ёрдамидан фойдаланилади.

```
print(states["London"])
```

Сатрларни калит сифатида ишлатиш

Маълумотлар ўқиш жараёнида калит мавжуд бўлмаса, истисно ҳосил бўлади. Бундай ҳолда қабул қилинган олдин калитни **in** оператори ёрдамида болигини текшириш мумкин:

```
with shelve.open(FILENAME) as states:
    key = "Brussels"
    if key in states:
        print(states[key])
```

Бундан ташқари, **get()** методидан фойдаланиш мумкин. Методнинг биринчи параметри қийматдан олинадиган калит, иккинчиси эса калит топилмаса қайтариладиган қиймат.

```
with shelve.open(FILENAME) as states:
    state = states.get("Brussels", "Undefined")
    print(state)
```

for циклидан фойдаланиб файлдан қийматларни олиш мумкин:

```
with shelve.open(FILENAME) as states:
    for key in states:
        print(key," - ", states[key])
```

keys() методи файлдаги барча калитларни олиб беради, **values()** методи эса калитга тегишли бўлган қийматларни қайтаради:

```
with shelve.open(FILENAME) as states:
    for city in states.keys():
        print(city, end=" ")
# London Paris Berlin Madrid
    print()
    for country in states.values():
        print(country, end="")
# Great Britain France Germany Spain
```

Юқорида кўриб ўтилган методлардан ташқари, яна бир метод мавжуд. Бу метод ёрдамида маълумотларни туркумлар (кортежлар) кўринишидаги тўплам сифатида олиш мумкин. Ҳар бир кортеж ўзида калит ва қийматни сақлайди.

```
with shelve.open(FILENAME) as states:
    for state in states.items():
        print(state)
```

Консол ойнасидаги натижа:

```
("London", "Great Britain")
("Paris", "France")
("Berlin", "Germany")
("Madrid", "Spain")
```

Маълумотларни янгилаш

Маълумотларни ўзгартириш учун кўрсатилган калитга янги қиймат бериш, маълумотларни киритиш учун янги калит белгилаш кифоя:

```
import shelve
FILENAME = "states2"
with shelve.open(FILENAME) as states:
    states["London"] = "Great Britain"
    states["Paris"] = "France"
    states["Berlin"] = "Germany"
    states["Madrid"] = "Spain"
with shelve.open(FILENAME) as states:
    states["London"] = "United Kingdom"
    states["Brussels"] = "Belgium"
    for key in states:
        print(key, " - ", states[key])
```

Маълумотларни ўчириш

Бир вақтнинг ўзида калит ва қийматни ўчириш учун **pop()** функциясидан фойланилади, агар калит топилмаса, кўрсатилган қиймат ўзгарувчига узатилади:

```
with shelve.open(FILENAME) as states:
    state = states.pop("London", "NotFound")
    print(state)
```

Шу билан бирга, маълумотларни ўчириш учун **del** операторидан фойдаланиш мумкин:

```
with shelve.open(FILENAME) as states:
    del states["Madrid"]            # Madrid калитли маълумотни ўчириш
```

Барча элементларни ўчириш учун **clear()** методидан фойдаланилади:

```
with shelve.open(FILENAME) as states:
    states.clear()
```

4.6. OS модули ва файл тизими билан ишлаш

Каталоглар ва файллар билан ишлаш қатор имкониятларни OS модули тақдим этади. OS модули ўз ичига кўп функцияларни олади. Қуйида модулнинг айрим функциялари берилган:

- **mkdir()** – янги папка ҳосил қилади;
- **rmdir()** – кўрсатилган папкани ўчиради;
- **rename()** – файлни қайта номлаш;
- **remove()** – файлни ўчириш.

Папкани ҳосил қилиш ва ўчириш

OS модули ёрдамида папкаларни ҳосил қилиш учун **mkdir()** функциясидан фойдаланилади. Бу функция ҳосил қилаётган папканинг ўрнига йўналиш беради.

```
import os
# Joriy skriptga nisbatan yo'l
os.mkdir("hello")
# absolyut yol
os.mkdir("c://somedir")
os.mkdir("c://somedir/hello")
```

Папкани ўчириш учун **rmdir()** функциясидан фойдаланилади. Бу функция ёрдамида папкани ўчириш учун папканинг жойлашган ўрни берилади.

```
import os
# hozirgi skript bo`yicha yol;
os.rmdir("hello")
# absolyut yol;
os.rmdir("c://somedir/hello")
```

Файлни қайта номлаш

Қайта номлаш учун **rename(source, target)** функцияси чақирилади, биринчи параметр файлга олиб борувчи йўлдир, иккинчи параметр эса файлнинг янги номи ҳисобланади. Файлнинг манзили сифатида мутлоқ ва нисбий манзиллар ишлатилиши мумкин. Мисол учун, **somefile.txt** файли **C://SomeDir/** папкасида жойлашганлигини тасаввур қилиш лозим. Уни "hello.txt" номли файлга ўзгартириш қуйида берилган:

```
import os
os.rename("C://SomeDir/somefile.txt", "C://SomeDir/hello.txt")
```

Файлни ўчириш

Файлни ўчириш учун файлнинг манзили кўрсатилган ҳолда, **remove()** функциясидан фойдаланилади.

```
import os
os.remove("C://SomeDir/hello.txt")
```

Файлни мавжудликка текшириш

Агар фойдаланувчи томонидан мавжуд бўлмаган файл очишга ҳаракат қилинса, у ҳолда дастурда **FileNotFoundError** истисноси хосил бўлади. Истисноли вазиятлардан фойдаланишда **try...except** конструкторидан фойдаланиш мумкин. Файлларни очишда os.path.exists(path) методидан файл бор ёки йўқлигини текшириш мумкин. Бу методга файлнинг манзили текшириш учун жўнатилади:

```
filename = input("Faylni adresini kiriting: ")
if os.path.exists(filename):
    print("Ko'rsatilgan fayl mavjud ")
else:
    print("Ko'rsatilgan fayl mavjud emas")
```

Назорат саволлари:

1. Файлларни очиш учун қайси буйруқдан фойдаланилади.
2. Файлларни очиш учун ишлатиладиган функцияни ишлаш принципини тушунтириб беринг.
3. Файллар билан ишлаш функцияларини санаб беринг?
4. CSV файлларнинг афзалликларини тушунтириб беринг?
5. Бинар файллар деганда нимани тушунасиз?
6. Файлнинг мавжудлигини текшириш учун қандай буйруқдан фойдаланилади?
7. Файлни ўчириш учун қайси буйруқдан фойдаланилади?

V. Сатрли катталиклар

5.1. Сатрлар билан ишлаш

Python дастурлаш тилида сатрли катталиклар **Unicode** да кодланган белгилар кетма-кетлигидан иборат. Сатрларнинг ҳар бир белгисига мурожаат қилиш учун квадрат қавс ичида белгининг сатрдаги индексини ишлатиш лозим.

```
string = "hello world"
c0 = string[0]              # h
print(c0)
c6 = string[6]              # w
print(c6)
c11 = string[11]    # хатолик IndexError: string index out of range
print(c11)
```

Индекслаш нолдан бошланади, шунинг учун сатрнинг биринчи белгисининг индекси нолга тенг бўлади. Агар сатрда мавжуд бўлмаганда индексга мурожаат қилинса, **IndexError** ноли хатолик рўй беради. Масалан, юқорида келтирилган дастурда сатрнинг узунлиги 11 га тенг, яъни 11 та белгидан иборат сатр, шунинг учун бу белгилар индекси 0 дан 10 гача ифодаланади.

Python дастурлаш тилида бошқа дастурлаш тилларида мавжуд бўлмаган имконият бор. Бу имконият сатрнинг охирги белгисидан мурожаат қилишдир. Бунда манфий сонлардан фойдаланилади. Масалан, -1 индекси бу сатрнинг охирги белгисини ифодалайди, -2 эса охирги белгидан битта олдинги белгини ва ҳоказолар.

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

Сатрдаги белгилар билан ишлашда сатрларнинг ўзгармас (**immutable**) типига киришини ҳисобга олиш керак, агар сатрнинг маълум бир белгисини ўзгартириш керак бўлса, хатолик рўй беради:

```
string = "hello world"
string[1] = "R"
```

Сатрнинг бошқа қиймат бериш учун унинг тўлиқ қийматини ўзгартиришга тўғри келади.

Сатр белгилари билан ишлаш

Сатрдаги белгиларнинг бир нечтаси билан ишлаш мумкин. Бунда сатрдаги белгилар гуруҳини олиш, бошқа сатрга қиймат сифатида узатиш мумкин. Сатрлар гуруҳи билан ишлаш учун қуйидаги синтаксисдан фойдаланилади.

- **string[:end]**: сатрнинг бошидан бошлаб **end** индексгача белгилар кетма-кетлигини ифодалаш;
- **string[start:end]**: сатрнинг **start** индексидан бошлаб, **end** индексигача бўлган белгилар гуруҳини ифодалаш;
- **string[start:end:step]**: сатрнинг **start** индексидан бошлаб, **end** индексигача бўлган сатрлар кетма-кетлигини **step** қадам билан ифодалаш.

```
string = "hello world"
# 0 dan 5 gacha element
sub_string1 = string[:5]
print(sub_string1) # hello
# 2 dan 5 gacha element
sub_string2 = string[2:5]
print(sub_string2) # llo
sub_string3 = string[2:9:2]
print(sub_string3) # lowr
```

ord ва len функцияси

Сатрдаги белгиларнинг **Unicode** даги қийматини билиш учун **ord()** функцияси ёрдамидан фойдаланиш мумкин:

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

Сатрнинг узунлигини топиш учун **len()** функциясидан фойдаланилади.

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

Сатрда қидириш амали

in буйруғи ёрдамида сатрда жойлашган белгилар кетма-кетлигини топиш мумкин. Агар бу белгилар кетма-кетлиги берилган сатрда мавжуд бўлса, рост, акс ҳолда, ёлғон қиймат қайтаради.

```
string = "hello world"
exist = "hello" in string
print(exist)        # True
exist = "sword" in string
print(exist)        # False
```

5.2. Сатрлар билан ишловчи асосий функциялар

Сатрлар билан ишлашда турли кўринишдаги методлардан фойдаланилади. Бу методлар қуйидагилар:

- **isalpha()** – агар сатр фақат ҳарфлардан ташкил топган бўлса, **true**, акс ҳолда, **false** қиймат қайтаради;
- **islower()** – агар сатр фақат кичик ҳарфлардан ташкил топган бўлса, **true**, акс ҳолда, **false** қиймат қайтаради;
- **isupper()** – агар сатр фақат катта ҳарфлардан ташкил топган бўлса, **true**, акс ҳолда, **false** қиймат қайтаради;
- **isdigit()** – агар сатр рақамлардан ташкил топган бўлса, **true**, акс ҳолда, **false** қиймат қайтаради;
- **isnumeric()** – агар берилган сатр сонни ифодаласа, **true**, акс ҳолда, **false** қиймат қайтаради;
- **startwith(str)** – агар сатр **str** белгилар кетма-кетлиги билан бошланган бўлса, **true**, акс ҳолда, **false** қийма қайтаради;
- **endwith(str)** – агар сатр **str** белгилар кетма-кетлиги билан тугаган бўлса, **true**, акс ҳолда, **false** қиймат қайтаради;
- **lower()** – сатрнинг барча белгиларини қуйи регистрга ўтказиш методи;
- **upper()** – сатрнинг барча белгиларини юқори регистрга ўтказиш методи;
- **title()** – сатрдаги барча сўзларнинг бошланғич ҳарфларини юқори регистрга ўтказиш методи;
- **capitalize()** – сатрнинг фақат биринчи сўзининг биринчи ҳарфини юқори регистрга ўтказиш методи;
- **lstrip()** – сатрнинг бошланғич пробелларини ўчириш методи;
- **rstrip()** – сатрнинг охирги пробелларини ўчириш методи;
- **strip()** – сатрдаги бошланғич ва охирги пробелларни ўчириш методи;
- **ljust(width)** – агар сатр узунлиги **width** параметридан кичик бўлса, сатрнинг узунлигини **width** қийматига тенг бўлгунга қадар сатрнинг ўнг тарафига пробел қўшади ва чап тарафга текислаш методи;
- **rjust(width)** – агар сатр узунлиги **width** параметридан кичик бўлса, сатрнинг узунлигини **width** қийматига тенг бўлгунга қадар сатрнинг чап тарафига пробел қўшади ва ўнг тарафга текислаш методи;
- **center(width)** – агар сатр узунлиги **width** параметридан кичик бўлса, сатрнинг узунлигини **width** қийматига тенг бўлгунга қадар

сатрнинг чап ва ўнг тарафига пробел қўшади ва марказ бўйича текислаш методи;

- **find(str[,start[,end])** – сатрдаги **str** белгилар кетма-кетлигининг индексини қайтаради. Агар сатрда **str** белгилар кетма-кетлиги топилмаса, -1 сонини қиймат сифатида қайтаради.
- **replace(old, new[,num])** – сатрдаги **old** белгилар кетма-кетлигини **new** белгилар кетма-кетлигига алмаштирувчи метод;
- **split([delimeter[,num]])** – сатрни маълум бир белгилар ёрдамида бир нечта сатр кўринишга ўтказиш методи;
- **join(strs)** – сатр ёки сатрдан иборат массив элементларининг орасига берилган белгини жойлаштириб, сатр ҳисоб қилиш методи;

Қуйидаги дастурда клавиатурадан киритилган белгилар кетма-кетлигини сонга ўтказиш учун текшириш ва сонга айлантириш мумкин бўлса, сонга айлантириш келтирилган.

```
string = input("Sonni kiriting: ")
if string.isnumeric():
    number = int(string)
    print(number)
```

Сатрнинг бошланғич ва охирги белгилар кетма-кетлигини текшириш:

```
file_name = "hello.py"
starts_with_hello = file_name.startswith("hello")  # True
ends_with_exe = file_name.endswith("exe")              # False
```

Сатрнинг бошланғич ва охиридаги пробелларни ўчириб ташлаш дастури:

```
string = " hello world! "
string = string.strip()
print(string)              # hello world!
```

Сатр маълумотларни чиқаришда пробеллар билан тўлдириб, экранга чиқариш ва текислаш дастури:

```
print("iPhone 7:", "52000".rjust(10))
print("Huawei P10:", "36000".rjust(10))
```

Сатрда белгилар кетма-кетлигини қидириш

Pythonда сатрдаги белгилар кетма-кетлигини излаш учун **find()** методидан фойдаланилади, бу методнинг 3 хил кўриниши мавжуд:

- **find(str)** – сатрдаги **str** белгилар кетма-кетлигини сатрнинг бошидан охиригача қидиради;

– **find(str, start)** – сатрдаги **str** белгилар кетма-кетлигини сатрнинг **start** индексли белгисидан охиригача қидиради;
– **find(str, start, end)** – сатрдаги **str** белгилар кетма-кетлигини сатрнинг **start** индексли белгисидан **end** белгисигача қидиради.

Методнинг юқорида келтирилган кўринишларида белгилар кетма-кетлиги топилмаса, метод -1 қийматини қайтаради:

```
welcome = "Hello world! Goodbye world!"
index = welcome.find("wor")
print(index)                    # 6
index = welcome.find("wor",10)
print(index) # 21
index = welcome.find("wor",10,15)
print(index) # -1
```

Сатрдаги белгилар кетма-кетлигини алмаштириш

Сатрларда белгилар кетма-кетилигини бошқа белгилар кетма-кетлигига алмаштириш учун **replace()** методидан фойдаланилади.

– **replace(old, new)** – сатрдаги **old** белгилар кетма-кетлигини **new** белгилар кетма-кетлигига алмаштириш;
– **replace(old, new, num)** – сатрдаги **old** белгилар кетма-кетлигининг бошланғич **num** тасини **new** белгилар кетма-кетлигига алмаштириш.

```
phone = "+998-91-234-56-78"
# chiziq (defis)larni probelga alamshtirish
edited_phone = phone.replace("-", " ")
print(edited_phone) # +998 91 234 56 78
# chiziq (defis)larni o'chirish
edited_phone = phone.replace("-", "")
print(edited_phone) # +998912345678
# faqat birinchi chiziq (defis)ni o`zgartirish
edited_phone = phone.replace("-", "", 1)
print(edited_phone) # +99891-234-56-78
```

Сатрни белгилар кетма-кетлигига бўлиш

Сатрларни маълум бир белгига қараб бир нечта қисм сатрларга бўлиш учун **split()** методидан фойдаланилади. Бўлувчи сифатида исталган белги ёки белгилар кетма-кетлигидан фойдаланиш мумкин. Бу метод қуйидаги шаклларга эга:

– **split()** – бўлувчи сифатида пробеллардан фойдаланиш;
– **split(delimeter)** – бўлувчи сифатида **delimeter** белгисидан ёки белгилар кетма-кетлигидан фойдаланиш;

– **split(delimeter, num)** – **num** параметри сатрни бўлувчиларга ажратишда **delimeter** белгисини ишлатишлар сонини белгиловчи параметр.

```
text = "Bu ulkan, ikki quchoq eman, singan shoxlari va singan qobig'i bilan"
# probellar bo`yicha bo`lish
splitted_text = text.split()
print(splitted_text)
print(splitted_text[4])
# vergullar bo`yicha bo`lish
splitted_text = text.split(",")
print(splitted_text)
print(splitted_text[1])
# birinchi beshta probel bo`yicha bo`lish
splitted_text = text.split(" ", 5)
print(splitted_text)
print(splitted_text[5])
```

Сатрларни бирлаштириш

Сатрларда оддий амаллар билан ишлаш каби сатрларни бирлаштириш қўшиш каби амалга оширилади. Сатрларни бирлаштиришнинг яна бир йўли бу **join()** методидан фойдаланишдир. Бу метод рўйхат шаклида берилган сатрларни бирлаштириш учун хизмат қилади.

```
words = ["Let", "me", "speak", "from", "my", "heart", "in", "English"]
# bo`luvchi - probel
sentence = " ".join(words)
print(sentence)
# bo`luvchi – vertikal chiziq
sentence = " | ".join(words)
print(sentence)
```

join() методида рўйхатлар ўрнига оддий сатрлардан фойдаланилса, унда бирлаштириш учун ишлатилган белги ёки белгилар кетма-кетлиги сатрнинг белгилари учун ишлатилади. Масалан:

```
word = "hello"
joined_word = "|".join(word)
print(joined_word)
```

5.3. Сатрларни форматлаш

Сатрларни форматлаш учун **format()** методидан фойдаланилади. Бу метод ёрдамида форматлаш буйруқларнинг ўрнига маълум бир қийматларни жойлаштиради. Сатрдаги форматлаш буйруқлари учун фигурали қавслар {} ишлатилади.

Номланган форматлаш буйруқлари

Форматланадиган сатрларда форматлаш буйруқларини номлаш мумкин, бунда, **format()** методида номланган форматлаш буйруғи кўрсатилган ҳолда маълумотларни ёзиш керак.

```
text = "Hello, {first_name}.".format(first_name="Islom")
print(text)
info = "Ism: {name}\t Yosh: {age}".format(name="Islom", age=23)
print(info)
```

Сатрларга форматлаш парамертлари

Сатрларда бир нечта агрументларни кетма-кет равишда форматлар методига жойлаштириш мумкин, бунда, форматланаётган сатрда бу аргументларни фигурали қавслар ва рақамлар (рақамлар 0 дан бошланади) билан кўрсатиб қўйиш керак.

```
info = "Ism: {name}\t Yosh: {age}".format(name="Islom", age=23)
print(info) # Ism: Islom Yosh: 23
```

битта аргументдан бир неча марта фойдаланиш:

```
text = "Hello. {0} {0} {0}.".format("Islom")
```

Форматлаш буйруғининг турлари

Сатрларнинг форматланган қийматларини юклаш учун буйруқлар ёки махсус белгилардан фойдаланилади.

Маълумотларни форматлаш учун қуйидаги махсус белгилардан фойдаланилади:

– s – сатрларни жойлаштириш;

– d – бутун сонларни жойлаштириш;

– f – ҳақиқий сонларни жойлаштириш;

– % – қийматларни 100 га кўпайтиради ва фоиз белгисини қўйиб натижага жойлаштиради;

– e – қийматларни экспонент белгиларда акс эттиради.

Форматлаш буйруқларининг умумий синтаксиси қуйидагича:

```
{:formatlash_buyrug'i}
```

Форматлаш буйруқлари ишлатилишига қараб қўшимча параметрлар киритиш мумкин. Масалан, ҳақиқий сонларни форматлаш учун қуйидаги параметрлардан фойдаланилади.

```
{:[belgilar_soni][vergul][.kasrdagi_belgilar_soni]formatlash_buyrug'i}
```

Формат методи чақирилганда, форматлаш буйруғи ўрнига қўйиладиган қийматлар аргументлар бўлиб юкланади.

```
welcome = "Hello {:s}"
```

```
name = "Islom"
formatted_welcome = welcome.format(name)
print(formatted_welcome)              # Hello Islom
```

format() методининг натижаси сифатида форматлаш янги қатор қайтарилади. Бутун сонларни форматлаш:

```
source = "{:d} belgilar"
number = 5
target = source.format(number)
print(target)              # 5 ta belgilar
```

Форматланаётган сон 999 дан катта бўлганда, минглик разрядларини ажратувчи сифатида вергуллардан фойдаланиш мумкин. Бунинг учун қуйидагича кўринишдан фойдаланилади:

```
source = "{:,d} belgilar"
print(source.format(5000))                    # 5,000 belgi
```

float типидаги ҳақиқий сонлар учун форматлаш буйруғи ёрдамида сонларни яхлитлаб чиқариш мумкин. Бунинг учун форматлаш буйруғидаги нуқтадан кейин каср қисмдаги сонларнинг аниқлигини кўрсатиш керак бўлади:

```
number = 23.8589578
print("{:.2f}".format(number))          # 23.86
print("{:.3f}".format(number))          # 23.859
print("{:.4f}".format(number))          # 23.8590
print("{:,.2f}".format(10001.23554)) # 10,001.24
```

Бошқа параметр ёрдамида форматланган қийматларни минимал кенглигини белгилаш имконини беради:

```
print("{:10.2f}".format(23.8589578))    # 23.86
print("{:8d}".format(25))                       # 25
```

Фоизни кўрсатиш учун "%" кодини ишлатиш яхшироқ ҳисобланади.

```
number =.12345
print("{:%}".format(number))            # 12.345000%
print("{:.0%}".format(number))                  # 12%
print("{:.1%}".format(number))                  # 12.3%
```

Экспоненциал кўринишдаги рақамларни чиқариш учун "e" форматлар буйруғидан фойдаланилади:

```
number = 12345.6789
print("{:e}".format(number))            # 1.234568e+04
print("{:.0e}".format(number))          # 1e+04
print("{:.1e}".format(number))          # 1.2e+04
```

format методисиз форматлаш

Python дастурлаш тилида **format** методисиз ҳам маълумотларни форматлаш имконияти мавжуд, бундай форматлаш учун қуйидагича синтаксисдан фойдаланилади.

satr % (parametr1, parametr2,... parameterN)

format методисиз сатрларни форматлашда биринчи бўлиб, ўзида форматлаш буйруқларини сақлайдиган сатр келади (% дан ташқари), сатрдан сўнг % белгиси, ундан кейин эса сатрга қўйиладиган қийматлар рўйхати туркумлар (кортежлар) сифатида жойлаштирилади. % белгиси янги қатор ҳосил қилувчи жараённи ифодалайди.

```
info = "Ism: %s \t Yosh: %d" % ("Temur", 35)
print(info)                    # Ism: Temur        Yosh: 35
```

Форматлаш буйруқларининг ёнида % белгиси ишлатилади, **format** методидан фарқли равишда бу ерда фигурали қавслар ишлатилмайди. Бундан ташқари, бу ерда рақамларни форматлаш усуллари ҳам қўлланилмайди:

```
number = 23.8589578
print("%0.2f - %e" % (number, number)) # 23.86 - 2.385896e+01
```

Назорат саволлари:

1. Сатр деганда нимани тушунасиз?
2. Ord функциясини тушунтириб беринг.
3. Сатрнинг узунлигини топиш учун қайси функциядан фойдаланилади?
4. Белгилар кетма-кетлиги сатрда мавжудлигини текшириш қайдай амалга оширилади?
5. Сатр билан ишлайдиган методларни тушунтириб беринг.

VI. Python дастурлаш тилининг асосий ички модуллари

6.1. Random модули

Random модули тасодифий сонларнинг генерациясини назорат қилади. Бу модулдаги функциялар қуйидагилар:

- **random()** – бу функция 0 дан 1 гача тасодифий сонни танлайди;
- **randint()** – маълум бир оралиқдаги тасодифий сонни қайтаради;
- **randrange()** – маълум бир рақамлар тўпламидан тасодифий сонни танлаб беради;
- **shuffle()** – тўпламни тасодифий аралаштириб беради;
- **choice()** – тасодифий рўйхат элементини қайтаради.

random() функцияси 0 ва 1 оралиғидаги тасодифий сузувчи вергулли (ҳақиқий) сонларни қайтаради. Агар катта ораликда сон керак бўлса, ҳосил қилинган тасодифий сонни ўша сонга кўпайтирилади.

```
import random
number = random.random() # 0.0 dan 1.0 gacha qiymat
print(number)
number = random.random() * 100 # 0.0 dan 100.0 gacha qiymat
print(number)
```

randint(min,max) функцияси **min** ва **max** қиймат орасида тасодифий сонни танлайди.

```
import random
number = random.randint(20, 35)
print(number)
```

randrange() функцияси маълум бир рақамлар тўпламидан тасодифий сонни танлайди. Бу функцияси уч шаклга эга:

- **randrange(stop)** – **0** дан **stop** сонигача бўлган тасодифий сони қайтаради;
- **randrange(start,stop)** – **start** дан **stop** сонигача бўлган тасодифий сонни қайтаради;
- **randrange(start, stop, step)** – **start** дан **stop** сонигача бўлган тасодифий сонни қайтаради. Сонларни танлашда уларнинг орасидаги абсолют қиймат **step** га тенг бўлади.

```
import random
number = random.randrange(10) # 0 dan 10 gacha qiymat
print(number)
```

```
number = random.randrange(2, 10) # 2, 3, 4, 5, 6, 7, 8, 9, 10 diapazondagi
qiymatlar
print(number)
number = random.randrange(2, 10, 2) # 2, 4, 6, 8, 10 diapazondagi qiymatlar
print(number)
```

Рўйхат билан ишлаш

Random модулида рўйхатлар билан ишлаш учун иккита функция мавжуд. Бу функциялардан бири **shuffle()** функцияси рўйхатда берилган сонларни тасодифий аралаштириб беради, **choice()** функцияси эса рўйхатдаги элементларнинг ихтиёрийсини қайтаради.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
random.shuffle(numbers)
print(numbers) # 1
random_number = random.choice(numbers)
print(random_number)
```

6.2. Math модули

Python дастурлаш тилида мавжуд бўлган **math** модули математик, тригонометрик ва логарифмик амалларни бажариш учун функциялар тўпламидан иборат:

- **pow(num, power)** – **num** сонини **power** даражага кўтаради;
- **sqrt(num)** – **num** сонини квадрат илдизини қайтаради;
- **ceil(num)** – берилган сонни юқорига яхлитлаш;
- **floor(num)** – берилган сонни қуйига яхлитлаш;
- **factorial(num)** – берилган соннинг факториалини қайтаради;
- **degrees(rad)** – радианда берилган бурчакнинг қийматини градус кўринишга келтиради;
- **radians(grad)** – градусда берилган бурчакнинг қийматини радиан кўринишга келтиради;
- **cos(rad)** – берилган соннинг косинусини қайтаради;
- **sin(rad)** – берилган соннинг синусини қайтаради;
- **tan(rad)** – берилган соннинг тангенсини қайтаради;
- **acos(rad)** – берилган соннинг арккосинусини қайтаради;
- **asin(rad)** – берилган соннинг арксинусини қайтаради;
- **atan(rad)** – берилган соннинг арктангенсини қайтаради;
- **log(n, base)** – **base** асосга кўра **n** сонининг логарифми;
- **log10(n)** – **n** сонни ўнли логарифмини қайтаради.

Юқорида кўрсатилган функциялардан фойдаланиш қуйида берилган:

```
import math
# 2 ni 3 -chi darajaga oshirishi
n1 = math.pow(2, 3)
print(n1) # 8
# shu jarayonni quyidagicha ham amalga oshirish mumkin
n2 = 2**3
print(n2)
# ildiz
print(math.sqrt(9)) # 3
# katta qiymatga yaxlitlash
print(math.ceil(4.56)) # 5
# kichik qiymatga yaxlitlash
print(math.floor(4.56)) # 4
# radiandan gradusga o`tkazish
print(math.degrees(3.14159))                    # 180
# gradusdan radianga o`tkazish
print(math.radians(180))          # 3.1415.....
# kosinus
print(math.cos(math.radians(60)))     # 0.5
# sinus
print(math.sin(math.radians(90)))     # 1.0
# tangens
print(math.tan(math.radians(0)))                    # 0.0
print(math.log(8,2))                  # 3.0
print(math.log10(100))                # 2.0
```

Math модулида қўшимча равишда ўзгармаслар берилган. Бу ўзгармаслар **PI** ва **E** сонларидир.

```
import math
radius = 30
# 30 radiusli aylana yuzasi
area = math.pi * math.pow(radius, 2)
print(area)
# 10 ning natural logarifmi
number = math.log(10, math.e)
print(number)
```

6.3. Locale модули

Рақамларни форматлашда Python дастурлаш тилида Англия–Саксон тизимидан фойдаланилади, бунда бутун соннинг рақамлар қатори бир-биридан вергул билан ажралиб туради, каср қисми эса нуқта билан ажратилади. Мисол учун, Европада бошқа тизим ишлатилади, унда рақамлар нуқта билан, соннинг каср қисми эса вергул билан ажратилади:

```
# anglosakson tizimi
1,234.567
# yevropa tizimi
1.234,567
```

Python дастурлаш тилида шу муаммони ҳал қилиш мақсадида маълум бир маъданиятга мослашган **locale** модули мавжуд. Ҳудудий локал модулини ўрнатиш учун **locale** модулида махсус **setlocale()** функцияси мавжуд. Бу функция 2 та қиймат қабул қилади.

```
setlocale(category, locale)
```

Биринчи параметр функция қўлланиладиган тоифасини – рақамлар, валюталар ёки ҳар иккала кўринишни билдиради. Параметр қиймати сифатида қуйида берилган катталиклардан фойдаланиш мумкин:

- **LC_ALL** – Барча тоифаларга ишлатиш маъносини билдиради ва қўллайди. Рақамлар, валюталар, саналар ва бошқаларни форматлаш;
- **LC_NUMERIC** – рақамларни маҳаллийлаштириш учун қўлланилади;
- **LC_MONETARY** – валюталарга маҳаллийлаштиришни қўлланилади;
- **LC_TIME** – сана ва вақт билаш ишлашда локализация қўллаш;
- **LC_CTYPE** – Белгиларни юқори ёки кичик ҳарфларга ўтказиш жараёни учун локализацияни қўллаш;

LC_COLLIATE – сатрларни солиштиришда локализацияни қўллаш.

setlocale функциясининг иккинчи параметри фойдаланиш керак бўлган локализациянинг номини билдиради. **Windows** операцион тизимида ISO бўйича иккита белгидан иборат давлат кодини ифодалайди. Масалан, АҚШ учун – "**us**", Германия учун – "**de**", Россия Федерацияси учун – "**ru**" ишлатилиши мумкин. Лекин **MacOS** да тил ва давлат кодини киритиш лозим. Масалан, АҚШда инглиз тили учун – "**en_US**", Германияда немис тили учун – "**de_DE**", Россия Федерациясида рус тили учун – "**ru_RU**" ишлатилади. Одатий ҳолатда "**en_US**" созламаси қўлланилади.

locale модулида рақамларни ва валюталарни форматлаш учун иккита функция мавжуд:

– **currency(num)** – валютани форматлайди;
– **format(str, num)** – сатр ҳосил қилишда str сатрдаги форматлаш буйруқларининг ўрнига num сонини жойлаштиради. Форматлаш буйруқлари сифатида қуйидаги буйруқлар ишлатилади;
  – **d** – бутун сонлар учун;
  – **f** – сузувчи вергулли, яъни ҳақиқий сонлар учун;
  – **e** – экспоненсиал ёзиш учун;

Ҳар бир форматлаш буйруғи олдига % фоиз белгиси қўйилади. Мисол учун "**%d**". Ҳақиқий сонларни форматлашда форматлаш буйруғи олдидан нуқта ва кейин ҳақиқий соннинг каср қисмида нечтагача рақам қатнашишини кўрсатиб қўйиш лозим. Масалан:

```
import locale
locale.setlocale(locale.LC_ALL, "de")          # Windows uchun
# locale.setlocale(locale.LC_ALL, "de_DE")     # MacOS uchun
number = 12345.6789
formatted = locale.format("%f", number)
print(formatted)                               # 12345,678900
formatted = locale.format("%.2f", number)
print(formatted)                               # 12345,68
formatted = locale.format("%d", number)
print(formatted)                               # 12345
formatted = locale.format("%e", number)
print(formatted)                               # 1,234568e+04
money = 234.678
formatted = locale.currency(money)
print(formatted) # 234,68 €
```

Агар **setlocale()** методининг иккинчи параметрига аниқ қиймат бермасдан бўш сатр берилса, Python дастурлаш тили шу компьютерда фойдаланиладиган ҳудудий локал моделни автоматик аниқлайди. **Getlocal()** функцияси орқали бу ҳудудий моделни олиш ёки аниқлаш мумкин:

```
import locale
locale.setlocale(locale.LC_ALL, "")
number = 12345.6789
formatted = locale.format("%.02f", number)
print(formatted) # 12345,68
print(locale.getlocale()) # ('Russian_Russia', '1251')
```

6.4. Decimal модули

Сузувчи вергулли сонлар билан ишлашда (яъни **float**) дастурнинг натижаси жуда аниқ бўлавермайди. Масалан:

```
number = 0.1 + 0.1 + 0.1
print(number)                    # 0.30000000000000004
```

Бу муаммони **round()** яхлитлаш функциясидан фойдаланиб ечиш мумкин. Бундан ташқари, яна битта усул мавжуд бўлиб, бу усулда **decimal** моделидан фойдаланилади.

Ушбу модулдаги сонлар билан ишловчи асосий компонент **Decimal** класси ҳисобланади. Бу компонентни ишлатиш учун конструктор ёрдамида унинг объектини яратиш лозим. Конструкторга рақамни ифодаловчи сатрли қиймат узатилади:

```
from decimal import Decimal
number = Decimal("0.1")
```

Шундан сўнг **Decimal** объектидан арифметик операцияларда фойданиш мумкин:

```
from decimal import Decimal
number = Decimal("0.1")
number = number + number + number
print(number)
```

Decimal классидан яратилган объект ишлатилган арифметик ифодаларда бутун сонлардан фойдаланиш ҳам мумкин:

```
number = Decimal("0.1")
number = number + 2
```

Ифодаларда **float** ва **Decimal** ҳақиқий сонларини аралаштириб юбормаслик керак.

```
number = Decimal("0.1")
number = number + 0.1
```

Қўшимча белгилардан фойдаланиб, соннинг каср қисмида қанча рақам мавжудлигини аниқланади. Яъни, “0.10” сон охири 0 билан тугаган бўлса ҳам каср қисмида иккита элементдан иборат деб қабул қилинади. Шунга мос равишда “0.100” сонининг каср қисми 3 та рақамдан иборат.

```
number = Decimal("0.10")
number = 3 * number
print(number)
```

Сонни яхлитлаш

Decimal классидан яратилган объектлар ўзининг қийматини яхлитлаш имкониятларини берувчи **quantize()** методига эга. Бу методда бир нечта параметр мавжуд бўлиб, биринчи параметр сифатида **Decimal** объекти қиймат сифатида юборилади:

```
from decimal import Decimal
number = Decimal("0.444")
number = number.quantize(Decimal("1.00"))
print(number)                              # 0.44

number = Decimal("0.555678")
print(number.quantize(Decimal("1.00")))    # 0.56

number = Decimal("0.999")
print(number.quantize(Decimal("1.00")))    # 1.00
```

Юқорида келтирилган дастурларда қўлланилган "1.00" сатри яхлитлаш кўринишини ифодалайди. Форматлаш учун кўрсатилган соннинг каср қисмида нечта рақам қатнашса, форматланаётган соннинг каср қисмида ҳам шунча рақам қатнашади. Одатда, яхлитлаш **ROUND_HALF_EVEN** ўзгармаси ёрдамида тавсифланади. Масалан:

```
from decimal import Decimal, ROUND_HALF_EVEN
number = Decimal("10.025")
print(number.quantize(Decimal("1.00"), ROUND_HALF_EVEN))

number = Decimal("10.035")
print(number.quantize(Decimal("1.00"), ROUND_HALF_EVEN))
```

“1.00” сатр каср қисмида иккита рақамгача яхлитлашни амалга ошириш мумкинлиги тушунилади. Биринчи ҳолатда “10.025” да каср қисмининг иккинчи рақами 2 жуфт, шунинг учун ундан кейинги рақам 5 бўлса ҳам учга яхлитланмайди. Чунки **ROUND_HALF_EVEN** ўзгармаси яхлитланаётган хонадаги рақамнинг қийматига эътибор беради. Агар бу хонадаги сон 4 ва ундан кичик жуфт сон бўлса, яхлитланиш жуфт сонга қараб юрилади. Агар бу хонадаги рақам 5 ва ундан катта бўлса, оддий яхлитланади.

Иккинчи ҳолатда “10.035” – касрнинг иккинчи рақами 3 – тоқ сон, шунинг учун у 4 га яхлитланади. Яхлитлашнинг яна қуйидаги кўринишлари мавжуд. Бу кўринишда барча қийматлар константалар ҳисобланади:

– **ROUND_HALF_UP** – агар ундан кейинги рақам 5 ёки 5 дан катта бўлса, юқори қийматга қараб яхлитлайди;

– **ROUND_HALF_DOWN** – агар ундан кейинги рақам 5 ёки 5 дан катта бўлса, юқори қийматга қараб яхлитлайди;

```
number = Decimal("10.026")
print(number.quantize(Decimal("1.00"), ROUND_HALF_DOWN))
number = Decimal("10.025")
print(number.quantize(Decimal("1.00"), ROUND_HALF_DOWN))
```

– **ROUND_05UP** – фақат 0 рақами учун яхлитлайди, кейинги сонни 5 деб фараз қилади.

```
number = Decimal("10.005")
print(number.quantize(Decimal("1.00"), ROUND_05UP))          # 10.01

number = Decimal("10.025")
print(number.quantize(Decimal("1.00"), ROUND_05UP))          # 10.02
```

– **ROUND_CEILING** – ўзидан сўнг қандай рақам келишидан қатъий назар юқори қийматга яхлитлайди;

```
number = Decimal("10.021")
print(number.quantize(Decimal("1.00"), ROUND_CEILING)) # 10.03

number = Decimal("10.025")
print(number.quantize(Decimal("1.00"), ROUND_CEILING)) # 10.03
```

– **ROUND_FLOOR** – ўзидан сўнг қандай рақам келишидан қатъий назар қуйига яхлитлаш.

```
number = Decimal("10.021")
print(number.quantize(Decimal("1.00"), ROUND_FLOOR))  # 10.02

number = Decimal("10.025")
print(number.quantize(Decimal("1.00"), ROUND_FLOOR))  # 10.02
```

Назарий саволлар:

1. Python дастурлаш тилидаги стандарт модулларни санаб беринг.

2. Математик модул ва унинг таркибига кирувчи функциялар ҳақида маълумотлар беринг.

3. Ҳақиқий сонлар билан ишловчи модул вазифаси нимадан иборат?

VII. Python дастурлаш тилида объектга йўналтирилган дастурлаш

7.1. Класс ва объект

Python дастурлаш тили объектга йўналтирилган дастурлаш тили таркибига кирувчи дастурлаш тили ҳисобланади. Дастурлаш тилида дастур яратишда класслардан фойдаланиш мумкин.

Класс – объектнинг эскизи ёки расмий тавсифи ҳисобланади. Класс – объектга йўналтирилган дастурлашнинг асосий ва фойдаланувчи томонидан яратиладиган типи ҳисобланади. Класс ёрдамида яратиладиган ўзгарувчилар ўзида объектларни сақлайди. Класс ўзида ўзгарувчилар ва функцияларни сақлайди. Класс ўзгарувчилари хусусиятлар, функциялари эса методлар деб юритилади. Методлар класснинг хатти–ҳаракатини белгилаб беради. Класслар **class** калит сўзи ёрдамида белгиланади.

```
class sinf_nomi:
    klass_metodlari
```

Класс ёрдамида объектлар ҳосил қилиш Python дастурлаш тилида қуйидагича амалга оширилади:

```
ob`yekt_ = klass_nomi([parametrlari])
```

Масалан, инсонни ифодаловчи **Person** номдаги класс қуйида яратилган:

```
class Person:
    name = "Ali"

    def display_info(self):
        print("Salom, mening ismim:", self.name)

person1 = Person()
person1.display_info()          # Salom, mening ismim Ali

person2 = Person()
person2.name = "Vali"

person2.display_info()          # Salom, mening ismim Vali
```

Person класси инсон исмини ўзида сақловчи **name** номли хусусиятга ва маълумотларни ойнага чиқарувчи **display_info** методларини ўз ичига олади.

Ҳар қандай класс методларини ишлатишда, методларни биринчи параметри сифатида шу объектга билдирувчи **self** хизматчи сўзи ишлатилади (бошқа дастурлаш тилларида объектни

кўрсатиш учун **this** хизматчи сўзи ишлатилади). Класс ичида ушбу буйруқ орқали класснинг методига ёки хусусиятига мурожаат қила олади. Юқорида келтирилган дастурда **self.name** буйруғи орқали фойдаланувчининг номини олиш мумкин. **Person** классини ҳосил қилгандан сўнг бир нечта унинг объектларини ўзгарувчиларга қиймат сифатида ўзлаштириш мумкин. Масалан, юқоридаги дастурда **person1** ва **person2** ўзгарувчиларида объектлар сақланади.

Конструкторлар

Конструктор класснинг методи бўлиб, класс ёрдамида объектлар ҳосил қилиш учун ишлатилади. Объектлар яратишда асосан стандарт конструктордан кенг қўлланилади.

```
person1 = Person()
person2 = Person()
```

Класс яратишда унинг конструкторини **__init__()** методи ёрдамида аниқланади. Фойдаланувчи қўшимча конструкторлар яратиш ҳуқуқига эга. Қўшимча конструкторлар стандарт конструктордан фарқли равишда объектнинг қайсидир бир хусусиятига бошланғич қиймат ўрнатишга хизмат қилади. Чунки конструктор курувчи деган маънони билдиради. Класс конструктори яратилаётган объектга бошланғич қийматлар ўрнатган ҳолда объект яратиб беради.

```
class Person:
    # konstruktor
    def __init__(self, name):
        self.name = name # ism qo`yish
    def display_info(self):
        print("Salom, mening ismim ", self.name)
person1 = Person("Ali")
person1.display_info() # Salom, mening ismim Ali

person2 = Person("Vali")
person2.display_info() # Salom, mening ismim Vali
```

Конструкторнинг биринчи параметри **self** ҳисобланади. Бу хизматчи сўз ёрдамида объект ўзидаги атрибутларга мурожаат этиш ҳуқуқига эга бўлади. Конструкторнинг иккинчи параметри сифатида фойдаланувчи номи яъни **self.name** хусусияти кўрсатилган. Юқорида кўрсатилган **Person** классида **name** хусусиятини кўрсатиш шарт эмас, чунки **self.name = name** қийматини ўрганиш бевосита **name** хусусиятини ҳосил қилади.

```
person1 = Person("Ali")
person2 = Person("Vali")
```

Деструктор

Деструктор – дастурда яратилган объектларни хотирадан ўчириш учун ишлатиладиган метод. Объектларни хотирадан ўчириш учун **del** операторидан фойдаланиш мумкин:

```
person1 = Person("Ali")
del person1 # xotiradan o`chirish
# person1.display_info() # Bu metod ishlamaydi, chunki person1 metodi allaqachon xotiradan o`chirilgan.
```

Шуни таъкидлаш керакки, дастурда деструкторни ишлатиш шарт эмас, чунки дастур коди иш фаолияти тугатганидан сўнг барча яратилган объектлар автоматик тарзда хотирадан ўчирилади. Бундан ташқари, классларда деструкторни кўрсатиш учун **__del__** функциясини ҳосил қилиш мумкин. Масалан:

```
class Person:
    # konstuktor
    def __init__(self, name):
        self.name = name # ismni o`rnatamiz
    def __del__(self):
        print(self.name,"xotiradan o`chirish")
    def display_info(self):
        print("Salom,mening ismim", self.name)

person1 = Person("Ali")
person1.display_info() # Salom, mening ismim Ali

del person1 # xotiradan o`chirish

person2 = Person("Vali")
person2.display_info() # Salom, mening ismim Vali
```

7.2. Инкапсуляция

Классдаги майдонлар класснинг объектлари учун очиқдир, яъни дастурдаги исталган жойдан объектнинг хусусиятини билиш ва ўзгартириш мумкин. Масалан:

```
class Person:
def __init__(self, name):
        self.name = name  # ism o`rnatamiz
        self.age = 1      # yosh

    def display_info(self):
        print("Ism:", self.name, "\tYosh:", self.age)

Temur = Person("Ali")
```

```
Temur.name = "O`rgimchak odam"
Temur.age = -129
Temur.display_info()
```

Юқорида келтирилган дастурда ёшга ёки инсон номига нотўғри киймат қабул қилиш мумкин. Масалан, ёшга нисбатан манфий қиймат кўрсатиш. Бундай ҳолатларни бартараф этиш, яъни "Объектнинг атрибутига қиймат беришни назорат қилиш мумкинми?" деган савол туғилади.

Инкапсуляция тушунчаси бу муаммо билан чамбарчас боғлик. Инкапсуляция объектга йўнилтирилган дастурлашнинг асосий концепциясидир. Объектнинг атрибутларига коддан тўғридан–тўғри киришни тақиклайди. Python дастурлаш тилида инкапсуляцияни ишлатилишига келсак, атрибутларни махсус ёки ёпик қилиб, махсус методлар орқали кириш ҳуқукини чеклаш орқали класс хусусиятларини яшириш мумкин.

Юқоридаги классни ўзгартириб, хусусиятарини яшириш қуйидагича амалга оширилади:

```
class Person:
    def __init__(self, name):
        self.__name = name        # ism
        self.__age = 1            # yosh

    def set_age(self, age):
        if age in range(1, 100):
            self.__age = age
        else:
            print("noto`g`ri yosh ")

    def get_age(self):
        return self.__age

    def get_name(self):
        return self.__name

    def display_info(self):
        print("Ism:", self.__name, "\tYosh:", self.__age)
Temur = Person("Ali")
Temur.__age = 43
Temur.display_info()
Temur.set_age(-3486)
Temur.set_age(25)
Temur.display_info()
```

Ҳусусий атрибутларни яратишда атрибутнинг олдига иккиланган чизиқ жойлаштирилади. Масалан, класс ҳусусий майдонининг номи **name** бўлса, **__name**. Бу атрибутга фақат шу классдан мурожаат этиш мумкин, холос. Лекин бу атрибутга классдан ташқарида тўғридан-тўғри мурожаат қилиш мумкин эмас. Масалан:

```
Temur.__age = 43          #қиймат қабул қилинмайди
```

Ҳусусий атрибутнинг қийматини олиш хатолик келтириб чиқаради:

```
print(Temur.__age)
```

Ҳусусий атрибутларга қийматларни класс ташқарисидан ўрнатиш ёки қийматни олиш керак бўлади. Бунинг учун классснинг ичида методлар яратилади. **__age** атрибутнинг қийматини олиш учун қуйидагича метод ёзиш мумкин. Бу методни **getter** методи деб ҳам юритилади:

```
def get_age(self):
    return self.__age
```

__age атрибутига қиймат ўрнатиш учун қуйидаги метод ёзилиши мумкин. Бу методни **setter** методи деб юритиш ҳам мумкин:

```
def set_age(self, value):
    if value in range(1, 100):
        self.__age = value
    else:
        print("Noto`g`ri yosh")
```

Ҳар бир махсус атрибутлар учун шу каби жуфтлик методларни яратиш керак. Класснинг айрим ҳусусий атрибутлари учун бу жуфтлик метолардан бирини ёзиш ҳам мумкин. Юқорида келтирилган класснинг **__name** атрибутига қиймат беришда конструктордан фойдаланиб, унинг қийматини олишда **get_name()** методидан фойдаланиш мумкин.

Хусусиятларни изоҳлаш

Юқорида хусусиятларни қандай яратиш ҳақида кўрсатиб ўтилган. Лекин Python дастурлаш тилида хусусиятларни аниқлашнинг яна бир ўзига хос йўли мавжуд. Бу усулда @ белгиси ёрдамида яратиладиган методлар ва уларга ишлов бериш имконини яратади. @ белгисидан сўнг ишлатиладиган изоҳ сўз **property** дир.

getter хусусияти яратиш учун хусусият устига **@property** изоҳ жойлаштирилади.

setter хусусиятини яратиш учун яратилаётган хусусиятга **@getter_hususiyati_nomi.setter** изоҳи жойлаштирилади.

Қуйида изоҳлар ёрдамида яратилган **Person** класси берилган:

```
class Person:
    def __init__(self, name):
        self.__name = name # ism
        self.__age = 1      # yosh

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, age):
        if age in range(1, 100):
            self.__age = age
        else:
            print("Noto`g`ri yosh")

    @property
    def name(self):
        return self.__name

    def display_info(self):
        print("Ism:", self.__name, "\tYosh:", self.__age)

Temur = Person("Ali")
Temur.display_info()          # Ism: Ali Yosh: 1
Temur.age = -3486                  # Noto`g`ri yosh

print(Temur.age)              # 1

Temur.age = 36
Temur.display_info()          # Имя: Ali Yosh: 36
```

Биринчидан, **setter** методи **getter** методидан кейин аниқланади. Иккинчидан, ҳар иккала методларга мурожаат қилиш **age** хусусияти ёрдамида амалга оширилади. **getter** методи **age** деб номланганлиги учун, **setter** методининг устига **@age.setter** изоҳи жойлаштирилади.

7.3. Ворислик

Объектга йўналтирилган дастурлашда ворислик ёрдамида мавжуд классга асосланган ҳолда янги класс яратишдан иборат. Инкапсуляция билан бир қаторда ворислик объектга йўналтирилган дастурлашнинг асосий тамойилларидан бири ҳисобланади.

Ворисликнинг асосий тушунчалари субкласс ва суперкласс ҳисобланади. Яратилаётган класс барча атрибутлари ва методларини суперклассдан мерос олади. Суперкласс асосий класс **(base class)** ёки асос класс **(parent class)** деб ҳам юритилиши мумкин. Субкласс эса ҳосил қилинган класс **(derived class)** ёки бола **(child class)** деб юритилади.

Ворислик ёрдамида яратилаётган класснинг синтаксиси қуйидагича:

```
class klass_nomi (superklass):
    subklass_metodlari
```

Ворислик ёрдамида корхонада ишлайдиган ходимнинг классини яратиш керак бўлсин. Бунинг учун олдин яратилган **Person** классидан фойдаланилади. Ходим классини **Employee** деб номланади ва унинг **Person** классидан мерос олинган ҳолда яратилади. Ҳодим инсон бўлганлиги сабабли **Person** класси билан бир хил атрибут ва методларга эга бўлади. Шунинг учун **Employee** классида ҳудди шундай атрибут ва методларни яратиш мантиқсизлик ҳисобланади. Бундай ҳолатда, ворислик энг катта вазифани бажариб беради. **Person** классидан **Employee** классини яратиш қуйидаги дастурда келтирилган:

```
class Person:
    def __init__(self, name, age):
        self.__name = name # ism
        self.__age = age # yosh
    @property
    def age(self):
        return self.__age
    @age.setter
    def age(self, age):
        if age in range(1, 100):
            self.__age = age
        else:
            print("Noto`g`ri yosh")

    @property
    def name(self):
        return self.__name

    def display_info(self):
        print("Ism:", self.__name, "\tYosh:", self.__age)

class Employee(Person):
    def details(self, company):
```

```
            # print(self.__name, "korxonanada ishlaydi", company)
            # так нельзя, self.__name – xususiy atribut
            print(self.name, " korxonada ishlaydi ", company)
Temur = Employee("Ali", 23)
Temur.details("Google")
Temur.age = 33
Temur.display_info()
```

Employee класси **Person** классининг функционал атрибутларини тўлиқ қабул қилади ва унга қўшимча равишда **details()** методини қўшади. Ҳусусий **__name** ёки **__age** методлардан ташқари барча метод ва атрибутлар **self** калит сўзи орқали **Employee** класси учун очиқ ҳисобланади.

Employee объектини яратишда **Person** классини конструктори ишлатилади. Бундан ташқари бу объектда **Person** классининг барча методларини чақириш мумкин.

7.4. Полиморфизм

Полиморфизм – объектга йўналтирилган дастурлашнинг яна бир асосий жиҳати ва асосий классдан меърос бўлиб ўтган методларни ўзгартиш имконини яратувчи хусусиятидир. Қуйида ворислик ёрдамида яратилган классларда полиморфизм кўрсатиб ўтилган:

```
class Person:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age
    @property
    def name(self):
        return self.__name
    @property
    def age(self):
        return self.__age
    @age.setter
    def age(self, age):
        if age in range(1, 100):
            self.__age = age
        else:
            print("Noto`g`ri yosh")
    def display_info(self):
        print("Ism:", self.__name, "\tYosh:", self.__age)
class Employee(Person):
    def __init__(self, name, age, company):
```

```
            Person.__init__(self, name, age)
            self.company = company
        def display_info(self):
            Person.display_info(self)
            print("Korxona:", self.company)

class Student(Person):
        def __init__(self, name, age, university):
            Person.__init__(self, name, age)
            self.university = university
        def display_info(self):
            print("Talaba", self.name, self.university , "universitetda o`qiydi")
people = [Person("Ali", 23), Student("Ali", 19, "TATUFF"), Employee("Vali", 35,
"Google")]
for person in people:
        person.display_info()
        print()
```

Ходимни тавсифлайдиган **Employee** – синфида унинг конструктори аниқланади. Объектни яратишда, шунингдек, ходим ишлайдиган компанияни ўрнатиш мумкин. Бу класснинг конструктори тўрт параметрни, яъни стандарт **self** параметр, **name**, **age** параметрлари ва **company** параметрларини ўз ичига олади. **Employee** классининг конструктори унинг базавий класси бўлган **Person** класси конструкторини чақиради. Базавий класснинг методларига мурожаат қуйидаги синтаксис кўринишга эга бўлади:

```
superklass.metod_nomi(self [, parametr])
```

Шунинг учун субкласс **age** ва **name** базавий класс конструкторига ўтказади. **Employee** класси **Person** классидан барча хусусиятларни олган ҳолда, **self.company** хусусиятини қўшади.

Бундан ташқари, **Employee** класси **Person** классининг **display_info()** методини қайта белгилайди, чунки **name** ва **age**га қўшимча равишда ходимнинг ишлаётган компанияси ҳақида маълумот бериши керак. **name** ва **age** чиқариш учун кодни қайта ёзмаслик учун, асосий класс методига – **get_info** методига мурожаат қиланади: **Person.display_info(self)**.

Ҳудди шандай равишда талаба учун Student класси ҳам яратилади. Яратилган класс учун базавий классда мавжуд бўлган **display_info()** методини қайта ёзилади.

Дастурнинг асосий қисмида **Person** классининг 3 та объектидан иборат рўйхат тузилади. Базавий классдан фойдаланилган ҳолда, фарзанд класслар билан ишлаш мумкин.

Дастурдаги рўйхатнинг икки элементи **Employee** ва **Student** класслари томонидан яратилган объектлардан иборат. Цикл оператори ёрдамида рўйхатнинг ҳар бир элементининг **display_info()** методи чақирилади. Дастур бажарилиш давомида рўйхатдаги элементнинг турига қараб **display_info()** методи чақирилади. Юқоридаги дастурнинг натижаси қуйидагича:

```
Ism: Ali      Yosh: 23
Ali talaba Garvard universitetida o`qiydi
Ism: Vali     Yosh: 35
Korxona: Google
```

Объектлар турини текшириш

Объектлар билан ишлашда уларнинг турига қараб муайян операцияларни бажариш керак бўлади. **isinstance()** функцияси орқали объектнинг турини текшириш мумкин. Бу функция икки параметрли бўлиб, қуйидагича кўринишга эга:

```
isinstance(object, type)
```

Биринчи параметр текширилаётган объектни ифодалайди. Иккинчи параметр эса тахмин қилинаётган класс номи. Агар объект тахмин қилинаётган классга мансуб бўлса, бу функция **True** қиймат қайтаради, акс ҳолда, **False** қиймат қайтаради. Мисол учун, қуйида юқорида тавсифланган класслар учун ўзига хос бўлган хусусиятнинг қийматини олиш кўрсатилган:

```
for person in people:
    if isinstance(person, Student):
        print(person.university)
    elif isinstance(person, Employee):
        print(person.company)
    else:
        print(person.name)
    print()
```

7.5. Object класси. Объектни сатр кўринишга ўтказиш

Python дастурлаш тилининг 3 версиясидан бошлаб, барча класслар учун бир умумий базавий класс – **object** класси яратилди. Бу базавий класс барча яратиладиган класслар автоматик равишда унинг методларидан мерос олади.

Object классининг энг кўп ишлатиладиган методи бу **__str__()** методидир. Объектларни сатрли кўринишда чақириш ёки сатрли кўринишини олиш учун Python дастурлаш тили шу методни чақиради. Янги яратилган классларда бу методни қайтадан ёзиш яхши натижалар

беради. Масалан, қуйидаги дастурда **Person** дастурлаш тилида яратилган класс учун **__str__()** методини қайта юклаши кўрсатиб ўтилган:

```
class Person:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    @property
    def name(self):
        return self.__name

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, age):
        if age in range(1, 100):
            self.__age = age
        else:
            print("No`to`g`ri yssh")

    def display_info(self):
        print(self.__str__())

    def __str__(self):
        return "Ism: {} \t Yosh: {}".format(self.__name, self.__age)

Temur = Person("Temur", 23)
print(Temur)
```

__str__() методи сатрли қиймат қайтариши керак. Бу ҳолатда класснинг хусусиятларидан ташкил топган сатр ҳосил қилишда тўғри келади. Юқорида келтирилган дастурнинг натижаси қуйидагича:

```
Ism: Ali        Yosh: 23
```

Назорат саволлари:

1. Класс нима?
2. Объект яратиш учун қандай катталик керак?
3. ОЙДнинг тамойилларини тушунтиринг.
4. Ворислик деганда нимани тушунасиз?
5. Объектларни сатрга ўтказиш функциясини тушунтиринг.

VIII. Сана ва вақт билан ишлаш

8.1. Datetime модули

Python дастурлаш тилида вақт ва саналар билан ишлаш учун модулдан фойдаланилади. Бу функционал модул **datetime** деб аталади. Бу модул қуйидаги класслардан ташкил топган:

- **date** – санадан ташкил топган объект яратувчи класс;
- **time** – вақтдан ташкил топган объект яратувчи класс;
- **datetime** – сана ва вақтдан ташкил топган объект яратувчи класс.

Date класси

Саналар билан ишлаш учун **datetime** модулида кўрсатилган **date** классидан фойдаланилади. **date** ёрдамида объект яратиш учун 3 та йил, ой, кун параметрларини кетма-кет киритишдан иборат конструктордан фойдаланилади.

```
date(year, month, day)
```

Масалан, бирор-бир санани киритиш қуйидагича амалга оширилади:

```
import datetime
yesterday = datetime.date(2017,5, 2)
print(yesterday)     # 2017-05-02
```

Жорий санани олиш учун **today()** методидан фойдаланиш мумкин:

```
from datetime import date
today = date.today()
print(today)     # 2017-05-03
print("{}.{}.{}".format(today.day, today.month, today.year))
```

day, month, year хусусиятларидан фойдаланиб, кун, ой ва йилнинг қийматларини олиш мумкин.

Time klassi

Вақтлар билан ишлаш учун **time** классидан фойдаланилади. Бу класснинг конструкторидан фойдаланиб, вақт объектини яратиш мумкин:

```
time([hour] [, min] [, sec] [, microsec])
```

Конструктор кетма-кет равишда соат, дақиқа, сония ва микросонияларни кўрсатиш мумкин. Конструктордаги барча параметрларни киритиш шарт эмас, агар бирон-бир параметр киритилмаса, унинг ўрни мос равишда 0 билан тўлдирилади.

```
from datetime import time
```

```
current_time = time()
print(current_time)    # 00:00:00

current_time = time(16, 25)
print(current_time)    # 16:25:00

current_time = time(16, 25, 45)
print(current_time)    # 16:25:45
```

Datetime klassi

datetime класси шу номли модулдаги класс бўлиб, бу класс ёрдамида бир вақтнинг ўзида сана ва вақт билан ишлаш имкониятини яратади. **datetime** объектини яратиш учун қуйида кўрсатилган конструктордан фойдаланиш мумкин:

```
datetime(year, month, day [, hour] [, min] [, sec] [, microsec])
```

Конструктордаги **year**, **month** ва **day** параметрларни киритиш мажбурдир. Қолган параметрларни киритиш ихтиёрий ҳисобланади, агар уларнинг қийматлари кўрсатилмаса, уларнинг ўрнига автоматик тарзда 0 қиймати қабул қилинади.

```
from datetime import datetime
deadline = datetime(2017, 5, 10)
print(deadline)
deadline = datetime(2017, 5, 10, 4, 30)
print(deadline)
```

Жорий сана ва вақтни олиш учун **now()** методини чақириш мумкин:

```
from datetime import datetime
now = datetime.now()
print(now)
print("{}.{}.{} {}:{}".format(now.day, now.month, now.year, now.hour,
now.minute))
print(now.date())
print(now.time())
```

day, month, year, hour, minute, second хусусиятларидан фойдаланиб, вақт ва сананинг алоҳида қийматларини олиш мумкин. **date()** ва **time()** методи билан эса мос равишда вақт ва санани алоҳида олишимиз мумкин.

Сатрдан вақтга ўтказиш

Сатрли кўринишдаги маълумотдан санали ва вақтли кўринишдаги ўтказиш учун **datetime** классининг **strptime()** методларидан фойдаланиш мумкин.

strptime(str, format)

Биринчи параметр сатрли кўринишдаги сана ва вақтдан иборат қийматлар, иккинчи параметр сатр кўринишдаги қийматлар орасидан маълумотларни олиш учун формат. Қийматларни олиш учун форматда ишлатиладиган буйруқлар қуйидагилар ҳисобланади:

- %d – ойнинг куни рақам кўринишда;
- %m – ойнинг рақами;
- %y – йилнинг охирги икки рақами;
- %Y – йил рақам кўринишда;
- %H – соат кўриниши 24 соатлик;
- %M – дақиқани олиш;
- %S – сониянинг кўриниши.

```
from datetime import datetime
deadline = datetime.strptime("22/05/2017", "%d/%m/%Y")
print(deadline)
deadline = datetime.strptime("22/05/2017 12:30", "%d/%m/%Y %H:%M")
print(deadline)
deadline = datetime.strptime("05-22-2017 12:30", "%m-%d-%Y %H:%M")
print(deadline)
```

8.2. Саналар устида амаллар

Вақт ва санани форматлаш

date ва **time** объектларини форматлаш учун иккала классларда ҳам **strftime(format)** методи мавжуд. Бу метод вақт ёки санани ўзгартириш керак бўлган форматни кўрсатувчи битта параметр қабул қилади.

Форматни кўрсатиш учун қуйидаги форматлаш буйруқларидан бирини қўллаш мумкин:

- %a – қисқаритилган ҳафта куни, масалан, **Wednesday** сўзидан – **Wed** (ҳафталар автоматик инглизча номланади);
- %A – ҳафта куни, Масалан: **Wednesday**;
- %b – қисқартирилган ойнинг номи, масалан, **Oct** (**October** ойнинг қисқатмаси);
- %B – ойнинг тўлиқ номи, масалан, **October**;
- %d – 0 билан тўлдирилган ойнинг куни, масалан, 01;
- %m – 0 билан тўлдирилган ойнинг тартиб рақами, масалан, 05;
- %y – йилнинг охирги 2 хонасини кўрсатиш;
- %Y – йилни тўлиқ кўрсатиш;

- %H – соатни 24 соатлик форматда кўрсатиш, масалан, 13;
- %I – соатни 12 соатли форматда кўрсатиш, масалан, 01;
- %M – дақиқа;
- %S – сония формати;
- %f – микросония формати;
- %P – AM/PM кўрсатгичлари;
- %C – жорий ҳудуд учун формат қилинган сана ва вақт;
- %x – жорий ҳудуд учун форматланган сана;
- %X – жорий маҳаллий ҳудуд учун вақт формати.

Турли форматлардан фойдаланиш:

```
from datetime import datetime
now = datetime.now()
print(now.strftime("%Y-%m-%d"))          # 2017-05-03
print(now.strftime("%d/%m/%Y"))          # 03/05/2017
print(now.strftime("%d/%m/%y"))          # 03/05/17
print(now.strftime("%d %B %Y (%A)"))     # 03 May 2017 (Wednesday)
print(now.strftime("%d/%m/%y %I:%M")) # 03/05/17 01:36
```

Ой номлари ва ҳафта кунларини кўрсатиш учун автоматик равишда инглиз номлари ишлатилади. Агар маҳаллий ҳудуд форматидан фойдаланиш керак бўлса, уни **locale** модули орқали олдиндан кўрсатиш ёки ўрнатиш лозим бўлади:

```
from datetime import datetime
import locale
locale.setlocale(locale.LC_ALL, " uz-UZ")
now = datetime.now()
print(now.strftime("%d %B %Y (%A)"))
```

Вақт ва санани қўшиш ва олиб ташлаш

Кўпинча саналар билан ишлашда маълум вақт оралиғиини киритиш ёки маълум бир вақтни олиб ташлаш керак бўлади. Айнан шундай операциялар учун **datetime** модулида **timedelta** класси ишлатилади. Бу класс маълум бир вақт оралиғини белгилаш учун хизмат қилади. Маълум бир вақт оралиғини белгилаш учун **timedelta** классининг конструкторидан фойдаланиш мумкин:

```
timedelta([days] [,seconds] [,microseconds] [,milliseconds] [,minutes] [,hours]
[,weeks])
```

Конструктор параметрлари кетма-кет равишда кун, сония, микросония, миллисония, дақиқа, соат ва ҳафталардан ташкил топган. Бир нечта даврларни белгилаш қуйидагича:

```
from datetime import timedelta
three_hours = timedelta(hours=3)
```

```
print(three_hours)
three_hours_thirty_minutes = timedelta(hours=3, minutes=30)
two_days = timedelta(2)
two_days_three_hours_thirty_minutes = timedelta(days=2, hours=3, minutes=30)
```

Timedelta дан фойдаланиб саналарни қўшиш ёки олиб ташлаш мумкин. Қуйида бунга мисол келтирилган:

```
from datetime import timedelta, datetime
now = datetime.now()
print(now)
two_days = timedelta(2)
in_two_days = now + two_days
print(in_two_days)
```

Бундан ташқари, ихтиёрий соат ёки дақиқадан сўнгги вақтларни бериш мумкин. Масалан, 10 соат 15 дақиқа олдин соат неча бўлганини қуйидаги дастурда келтириб ўтилган. Бунда кўрсатилган вақт жорий вақтдан айрилган:

```
from datetime import timedelta, datetime
now = datetime.now()
till_ten_hours_fifteen_minutes = now - timedelta(hours=10, minutes=15)
print(till_ten_hours_fifteen_minutes)
```

Timedelta хусусиятлари

timedelta класси ёрдамида бирон-бир вақт оралиғини олиш учун бир нечта хусусиятлар мавжуд:

- **days** – кунлар сонини қайтаради;
- **seconds** – сонлар сонини қайтаради;
- **microseconds** – микросониялар сонини қайтаради.

Бундан ташқари, **total_seconds()** методи ёрдамида ҳақиқий сонларда сонияларни олиш мумкин. Мисол учун, икки сана орасидаги вақт оралиғи қуйидагича аниқланади:

```
from datetime import timedelta, datetime
now = datetime.now()
twenty_two_may = datetime(2019, 5, 22)
period = twenty_two_may - now
print("{} kun {} sekund {} mikrosekund".format(period.days, period.seconds,
period.microseconds))
print("Jami: {} sekund".format(period.total_seconds()))
```

Саналарни таққослаш

Сатрлар ва сонлар билан бир қаторда, саналарни ҳам стандарт таққослаш оператори ёрдамида таққослаш мумкин:

```
from datetime import datetime
```

```
now = datetime.now()
deadline = datetime(2019, 5, 22)
if now > deadline:
        print("Taqdimot topshirish muddati o`tdi")
elif now.day == deadline.day and now.month == deadline.month and now.year ==
deadline.year:
        print("Bugun taqdimot topshirish kuni ")
else:
        period = deadline - now
        print("Kun {} qoldi ".format(period.days))
```

Назорат саволлари:

1. Саналар билан ишлаш учун қайси классдан фойдаланилади?
2. Вақтлар билан ишлашда қайси класс ёрдам беради?
3. Саналарни таққослаш мумкинми?
4. Саналар устида қандай амаллар бажариш мумкин?

IX. Graphic User Interface (GUI) билан ишлаш

9.1. Tkinter. Илова ойнасини яратиш

Аксарият дастурлар бугунги кунда консолга қараганда интуитив ва фойдаланувларга қулай бўлган график интерфейсдан фойдаланишади. Python дастурлаш тилида фойдаланувчининг график интерфейсидан фойдаланиб, турли кўринишдаги дастурлар лойиҳалаш мумкин. Python дастурлаш тилида график интерфейс билан ишлаш учун махсус асбоблар тўплами тулкит ишлатилади. Бу тулкит **tkinter (Tulkit Interface)** деб номланади.

Tkinter алоҳида тузилган модул кўринишида ўзида барча керакли график компонентларни сақловчи – тугмалар, матн майдонлари ва бошқаларни ўз ичига олади. График дастурларни қуришда асосий базали қисми ойна яратишдир. Кейин эса бошқа график интерфейс компонентлари таркибий қисмлари ойнага жойлаштирилади. Шунинг учун аввал оддий ойна яратиш керак. Оддий ойна яратиш учун қуйидаги код ёзилади:

```
from tkinter import *
root = Tk()
root.title("Python grafik dasturi")
root.geometry("400x300")
root.mainloop()
```

График ойнани яратиш учун **tkinter** модулида мавжуд бўлган конструктор **Tk()** дан фойдаланилади. Яратилган ойна **root** номли ўзгарувчисига узатилади ва бу ўзгарувчи орқали ойнанинг хусусиятларини ўзгартириш мумкин. Хусусан, **title()** методи ёрдамида яратилган ойнанинг сарлавҳасини ўзгартириш имконини беради.

9.1–расм: Python дастурида GUI ойнаси

Geometry() методи ёрдамида ойна ўлчамларини ўзгартириш. Ойна ўлчамини ўрнатиш учун "ЭНИ x БЎЙИ" форматида **geometry()** методига сатр сифатида узатилади. Агар дастур ойнасини яратишда

geometry() методи чақирилмаса, автоматик тарзда ойна ўлчамлари ўрнатилади. Ойнани чиқариш ва фойдаланувчи билан мулоқотни таъминлаш учун ойнани ишга туширувчи **mainloop()** методини чақириш керак. Натижада дастур коди ишга туширилганда 9.1–расмда келтирилган ойна ҳосил бўлади.

Ойнанинг дастлабки ҳолати

Python дастурлаш тилида дастлабки ҳолатда экраннинг чап ва юқори томонида жойлашади. Лекин **geometry()** методи керакли қийматлар бериб, унинг ўрнини ўзгартириш мумкин.

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

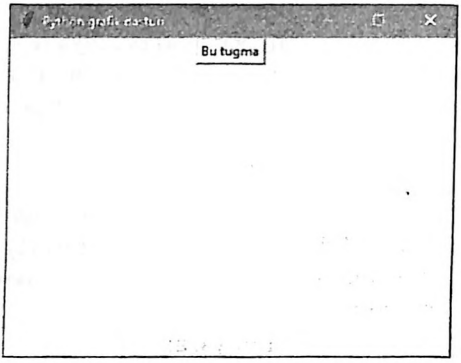
Юқорида келтирилган дастурда, эътибор берилса, **geometry()** методининг формати "ЭНИ x БЎЙИ + X + Y" кўринишда ёзилган. Ҳосил бўлаётган ойна экраннинг юқори чап бурчагидан 300 пиксел ўнгга, 250 пиксел пастга сурилади.

9.2. Илова ойнасига тугмалар жойлаштириш

Tkinter компоненталар тўплами виджетларни ўз ичига олади, булардан бири тугма **(Button)** ҳисобланади. Ойнага тугма жойлаштириш қуйидаги дастурда кўрсатилган:

```
from tkinter import *
root = Tk()
root.title("Python da GUI ")
root.geometry("300x250")
btn = Button(text="Hello")
btn.pack()
root.mainloop()
```

Тугма яратиш учун **Button()** конструктори ишлатилади. Ушбу конструкторда **text** параметрини ишлатиб тугма матнини ўрнатиш мумкин. Тугмани ойнада жойлаштиришни **pack()** методи чақирилади. Натижада ойнанинг юқори қисмида тугма жойлашади:

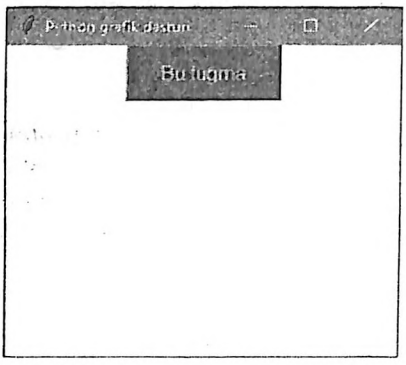


9.2–расм: Ойнага тугма жойлаштириш

Ҳар бир виджет, шу жумладан, тугма визуализацияга таъсир қиладиган ва конструктор орқали созлаш мумкин бўлган бир қатор атрибутларга эга:

```
from tkinter import *
root = Tk()
root.title("Python grafik dasturi")
root.geometry("300x250")
btn = Button(text="Bu tugma", background="#555", foreground="#ccc",
padx="20", pady="8", font="16")
btn.pack()
root.mainloop()
```

Тугма ҳосил қилиш учун ишлатилган конструкторнинг **padx, pady, font** параметрлари сонли қийматлар қабул қилади, **background** ва **foreground** параметрлари эса ўн олтилик саноқ системасида берилган ранглар қийматини олади.



9.3–расм: Тугма кўринишини ўзгартириш

Умуман олганда, **Button** компонентасининг конструктори қуйидаги параметрларни олиши мумкин:

```
Button(master, options)
```

master параметри тугма жойлаштирилиши керак бўлган контейнерни англатади. Юқоридаги ҳолатда бу график ойнанинг ўзи бўлиши:

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

Агар дастурда битта ойна яратилса, тугма ва бошқа элементлари автоматик тарзда шу ойнага жойлашади. Агар дастурда биттадан ортиқ ойналар яратилса, яратилаётган элементларни қайси ойнага тегишлилигини кўрсатиш керак бўлади. Бунинг учун яратилаётган элементнинг конструкторидаги биринчи параметрга ойнани ифодаловчи ўзгарувчини ёзишга тўғри келади.

Иккинчи параметр **options** параметрлар мажмуасидан ташкил топган бўлиши мумкин. Бу параметрларни қуйида кўрсатиб ўтилган:

- **activebackground** – тугма босилган ҳолдаги тугманинг ранги;
- **activeforeground** тугма босилгандаги матннинг ранги;
- **bd** – чегара қалинлиги (автоматик ҳолатда 2);
- **bg/background** –тугманинг фон ранги;
- **fg/foreground** – тугманинг матн ранги;
- **font** – матн шрифти, Масалан: font="Arial 14" – Arial шрифти 14px катталикда ёки font = ("Verdana", 13, "bold") – қалин 13 px Verdana шрифти;
- **height** – тугма баландлиги;
- **highlightcolor** – сичқонча тугманинг устига боргандаги ранги;
- **image** – тугмадаги расм;
- **justify** – ёзувни текислашни белгилаш. **LEFT** чап томони бўйича, **CENTER** – марказлаштириш, **RIGHT** – ўнг томон бўйича текислаш;
- **padx** – тугмани ўнг ва чап чегараси билан матни орасидаги масофа;

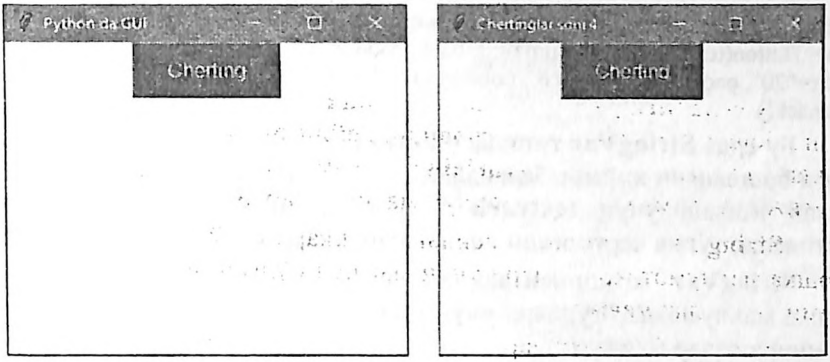
- **pady** – тугмани юқори ва пастки чегараси билан матни орасидаги масофа;
- **relief** – чегара турини танлаш, чегара тури қуйидагиларни қабул қилиш мумкин: **SUNKEN, RAISED, GROOVE, RIDGE;**
- **state** – тугманинг ҳолатини белгилайди, тугма ҳолатлари қуйидагилар бўлиши мумкин: **DISABLED, ACTIVE, NORMAL** (автоматик);
- **text** – тугма матни;
- **textvariable** – **StringVar** элементига боғланишни ўрнатади;
- **underline** – тугма матнидаги чизиладиган белги тартибини билдиради. Автоматик ҳолатда қиймати –1 га тенг;
- **width** – тугма кенглигини белгилайди;
- **wraplength** – ушбу хусусиятнинг қиймати мусбат бўлса, тугманинг матни шу узунликга мослаб чиқилади.

Тугма ҳодисаларини бошқариш

Тугмалар билан ишлашда тугмаларга турли кўринишдаги ҳодисалар ўрнатиш ва уларни бошқариш амалларини бажариш мумкин. Тугмага ҳодисалар ўрнатиш учун тугманинг конструкторидан фойдаланиш мумкин. Масалан, тугма чертилганда, айрим буйруқлар кетма-кетлиги бажарилиш дастури қуйидагича:

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

Бу ерда ишни қайта ишловчи сифатида **click_button** функцияси ўрнатилган. Бу фукнция тугманинг чертилишлар сонини сақлаш ва уни ойнага чиқарувчи глобал clicks ўзгарувчини ўзгартиради:



9.4–расм: Тугмани ойнага жойлаштириб унга ҳодиса ўрнатиш

9.3. Элементлар хусусиятини ўзгартириш

Дастур коди ёрдамида формага жойлаштирилган тугмаларнинг баъзи бир хусусиятларини ўзгартиришга тўғри келади. Лекин яратилган тугманинг хусусиятларини ўзгартириш учун ҳеч қандай методлар мавжуд эмас. Шунда савол туғилади. Қандай қилиб тугманинг хусусиятларини ўзгартириш мумкин?

Тугманинг матнини ўзгартириш учун **StringVar** номли оралиқ компонентадан фойдаланиш мумкин. Ушбу компонента сатрга боғланишни яратишга имкон беради. Бу компонентада иккита метод мавжуд:

- **get()** – **StringVar** компонентасининг қийматини қайтаради;
- **set(str)** – **StringVar** га қиймат ўрнатувчи метод.

StringVar да яратилган объектни визуал кўринишдаги компонента билан боғлаш учун конструкторда бу элементнинг қийматини **textvariable** параметрини ўрнатиш керак. Ойнада ҳосил бўлган тугмани чертишда унинг матнини ўзгартириш дастури қуйида берилган.

```
from tkinter import *
clicks = 0
def click_button():
        global clicks
        clicks += 1
        buttonText.set("Chertishlar soni {}".format(clicks))
root = Tk()
root.title("Python da GUI")
root.geometry("300x250")
buttonText = StringVar()
```

```
buttonText.set("Chertishlar soni {}".format(clicks))
btn = Button(textvariable=buttonText, background="#555", foreground="#ccc",
padx="20", pady="8", font="16", command=click_button)
btn.pack()
```

Бу ерда **StringVar** типидаги **buttonText** ўзгарувчиси яратилди. Унга бошланғич қиймат белгиланди, кейин **btn** ўзгарувчисига матн билан ишлаш учун **textvariable** параметри орқали боғланади. Натижада, тугма чертилгани учун матни ўзгартирилди.

StringVar компонентасидан ташқари, қўшимча равишда бошқа маълумотлар турлари учун мавжуд бўлган бир қатор ўхшаш компоненталар мавжуд:

- **IntVar** – бутун сонлар билан ишловчи компонента;
- **BooleanVar** – Мантиқий қийматлар билан ишловчи компонента;
- **DoubleVar** – ҳақиқий қийматлар билан ишловчи компонента.

IntVar типида яратилган ўзгарувчига боғланиб, босишлар сонини аниқлаш мумкин:

```
from tkinter import *
def click_button():
    clicks.set(clicks.get() + 1)
root = Tk()
root.title("Python da GUI ")
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()
```

Config методи

Юқорида фақат матнни ўзгартирувчи компоненталар билан танишдик. Дастурлашда фақат матнни эмас, балки бошқа хусусиятларни ёки бошқа компоненталарнинг параметрларини ўзгартириш мумкин бўлган усул ҳам мавжуд. Ушбу усулда **config()** методини элементга чақиришдан иборат бўлиб, унда керакли параметрлар ўрнатилади. Қуйида config методидан фойдаланиш кўрсатиб ўтилган:

```
from tkinter import *
clicks = 0
def click_button():
```

```
    global clicks
    clicks += 1
    btn.config(text="Chertishlar soni {}".format(clicks))
root = Tk()
root.title("Python da GUI ")
root.geometry("300x250")
btn = Button(text="Chertishlar soni 0", background="#555",
foreground="#ccc",padx="20", pady="8", font="16", command=click_button)
btn.pack()
root.mainloop()
```

9.4. Элементларни жойлаштириш

Pack() методи

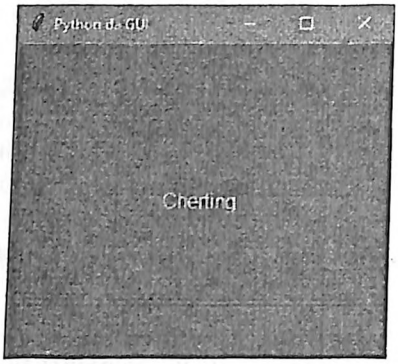
Ойнада элементларни жойлаштириш учун турли усуллар қўлланилади ва энг осон йўли элементдан **pack()** методини чақиришдир. Ушбу усул қуйидаги параметрларни олади:

- **expand** – агар бу параметрнинг қиймати рост бўлса, компонентани контейнернинг барча майдонни тўлдиришга рухсат бериш;
- **fill** – компонентани тўлдириш қийматини кўрсатиш. Бу параметр қуйидаги қийматларни ҳосил қилади: **NONE** (бошланғич қиймат, жойлаштириш мавжуд эмас), **X** (компонента фақат горизонтал йўналишда кенгаяди), **Y** (компонента фақат вертикал кўринишда кенгаяди) ва **BOTH** (копонента горизонтал ва вертикал кўринишда кенгаяди).
- **side** – компонентани жойлаштириш йўналиши. Бу параметр қуйидаги қийматларни қабул қилади: **TOP** (компонента контейнернинг юқори қисмига жойлаштирилади, одатий ҳолат), **BOTTEMUR** (компонента контейнернинг қуйи қисмига жойлатирилади), **LEFT** (компонента контейнернинг чап қисмига жойлатирилади), **RIGHT** (компонента контейнернинг ўнг қисмига жойлатирилади).

Масалан, **expand** ва **fill** параметрларидан фойдаланиб тугмани бутун контейнар бўйлаб жойлаштириш қуйидагича амалга оширилади:

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
root.geometry("300x250")
btn1 = Button(text="Cherting", background="#555", foreground="#ccc",
padx="15", pady="6", font="15")
```

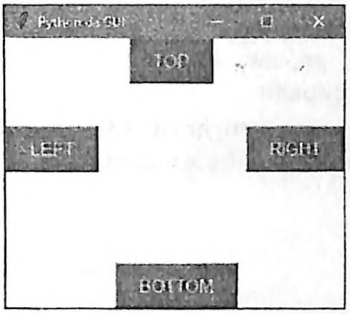
```
btn1.pack(expand = True, fill=BOTH)
root.mainloop()
```



9.5–расм: Компонентани контейнер бўйлаб жойлаштириш

side параметридан фойдаланиш қуйидагича амалга оширилади:

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



9.6–расм: **side** параметри ёрдамида копоненталарни жойлаш

side ва **fill** параметрлари комбинациясидан фойдаланиб, компоненталарни вертикал бўйлаб кенгайтириш мумкин:

```
btn1 = Button(text="CLICK ME", background="#555", foreground="#ccc",
padx="15", pady="6", font="15")
btn1.pack(side=LEFT, fill=Y)
```

Place() методи

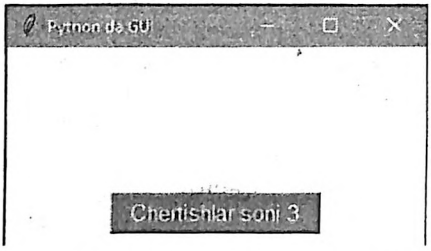
place() методи жойлашув параметрини аниқроқ созлаш имкониятини беради. У қуйидаги параметрларни олади:

- **height** ва **width** – компонентанинг баландлиги ва кенглигини мос равишда ўрнатиш (пикселларда);
- **relheight** ва **relwidth** – компонентанинг баландлиги ва кенглигини белгилайди, бу параметрнинг қиймати 0.0 ва 1.0 оралиғидаги ҳақиқий сон бўлиб, бу юқори контейнер баландигининг баландиги ва кенглигини нисбатини кўрсатади;
- **x** ва **y** – контейнернинг юқори чап бурчагига нисбатан мос равишда компоненталарни горизонтал ва вертикал ўрнатиш (пикселларда);
- **relx** ва **rely** – контейнернинг юқори чап бурчагига нисбатан мос равишда компоненталарни горизонтал ва вертикал ўрнатиш, қийматлар 0.0 ва 1.0 ораликда ҳақиқий сондан иборат;
- **bordermode** – компоненталарда чегара форматини белгилайди. **INSIDE** (стандарт) ва **OUTSIDE** қийматларини қабул қилиши мумкин;
- **anchor** – компоненталарни контейнерга маҳкамлашни билдиради. **Noth** (юқори қисм), **South** (қуйи қисм), **East** (ўнг томон), **West** (чап томон) ва **Center** (марказ бўйича).

Қисқартирилган кўринишда ҳам ишлатилиши мумкин: **n, e, s, w, ne, nw, se, sw, c**. Масалан: **nw** қиймати юқори чап бурчакни билдиради.

Ойнанинг марказида кенглиги 130 пиксел ва баландиги 30 пиксел бўлган тугма қуйидагича жойлаштирилади:

```
from tkinter import *
clicks = 0
def click_button():
    global clicks
    clicks += 1
    btn.config(text="Chertishlar soni{}".format(clicks))
root = Tk()
root.title("Python da GUI")
root.geometry("300x250")
btn = Button(text="Chertishlar soni 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()
```

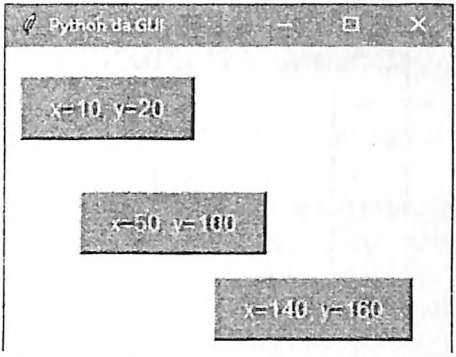


9.7–расм: Тугмани марказга жойлаштириш

Place() методидан фойдаланилганда компоненталарни контейнерда кўринадиган қилиш учун **pack()** методидан фойдаланиш шарт эмас.

```
from tkinter import *
root = Tk()
root.title("Python da GUI ")
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()
```



9.8–расм: Тугмаларни place() методи ёрдамида жойлаштириш

grid() методи

grid() методи компонентаралари жойлашуви учун бошқа методлар каби аниқ бир ҳаракат билан эмас, балки бошқача йўналишда компонентаралари жойлаштиради. Компонентаралари маълум бир шартли сетка ячейкасига жойлаштириб беради.

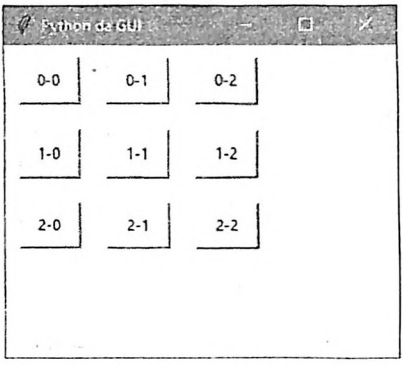
grid() методи қуйидаги параметрлардан иборат:

- **column** – контейтердаги устунлар сони;
- **row** – контейнердаги сатрлар сони;
- **columnspan** – компонентани қанча устун эгаллашини кўрсатади;
- **rowspan** – компонентани қанча сатр эгаллашини кўрсатади;
- **ipadx** ва **ipady** – мос равишда горизонтал ва вертикал йўналишда компонентанинг чегараси билан матни орасидаги масофа;
- **padx** и **pady** – мос равишда горизонтал ва вертикал йўналишда компонента ва ячейканинг чегарасигача бўлган масофа;

Масалан, **grid()** методи ёрдамида 9 та тугмадан иборат контейтер қуйидагича тузилади:

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
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()
```



9.9–расм: grid() методи ёрдамида компоненталарни жойлаштириш

9.5. Label матн элементи

Python дастурлаш тилида матнли меткалар **Label** компонентаси ёрдамида ифодаланади. Ушбу компонента статик матн ҳосил қилишга имкон беради. **Label** компонентасини яратиш учун иккита параметр қабул қилувчи конструктордан фойдаланилади:

Label(master, options)

master параметри юқори контейнерга ҳавола, **options** параметрлари эса қуйидаги келтирилган ўзгарувчиларни параметр сифатида қабул қилади:

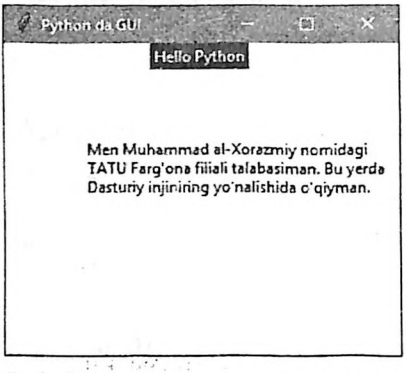
- **anchor** – матнни жойлаштириш;
- **bg/background** – фон ранги;
- **bitmap** – компонентада чизиладиган тасвирга ҳавола;
- **bd**: чегара қалинлиги;
- **fg/foreground** – ёзув ранги;
- **font** – ёзувнинг шрифти;
- **height** – компонентанинг баландиги;
- **cursor** – компонентага сичқончани олиб борилгандаги кўриниши;
- **image** – компонентага чиқадиган тасвирга ҳавола;

- **justify** – ёзувни текислашни белгилаш. **LEFT** – чап томон бўйича текислаш, **CENTER** – марказ бўйича, **RIGHT** – ўнг томон бўйича текислаш;
- **relief** –чегара типини танлайди, агар қиймат берилмаса, автоматик равишда **FLAT** қийматини қабул қилади;
- **padx** – компонентанинг ўнг ва чап чегараси билан матни орасидаги масофа;
- **pady** – компонентанинг юқори ва пастки чегараси билан матни орасидаги масофа;
- **text** – компонентанинг сарлавҳасини ўрнатиш;
- **textvariable** – компонентани **StringVar** компонентаси билан боғлашни ўрнатиш;
- **underline** – компонента матнидаги ост чизиқларни белгилаш. Одатий ҳолатда қиймати –1, яъни ҳеч қандай чизиқ чизилмаган;
- **width** – компонентанинг кенглиги;
- **wraplength** – компонентанинг сарлавҳасини сатрларга жойлаштиришни белгилаш;

Илова ойнасида энг оддий матнни жойлаштириш:

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
root.geometry("300x250")
label1 = Label(text="Hello Python", fg="#eee", bg="#333")
label1.pack()
poetry = "Men Muhammad al-Xorazmiy nomidagi\nTATU Farg'ona filiali
talabasiman. Bu yerda\nDasturiy injiniring yo'nalishida o'qiyman."
label2 = Label(text=poetry, justify=LEFT)
label2.place(relx=.2, rely=.3)
root.mainloop()
```

Матнни кейинги сатрга ўтказиш учун “\n” эскейп кетма-кетлигидан фойдаланиш мумкин.



9.10–расм: Label компонентасига матнларни жойлаш

9.6. Entry киритиш майдони

Entry компонентаси матн киритиш учун майдон ҳосил қилади. **Entry** конструктори қуйидаги параметрларни олади:

Entry(master, options)

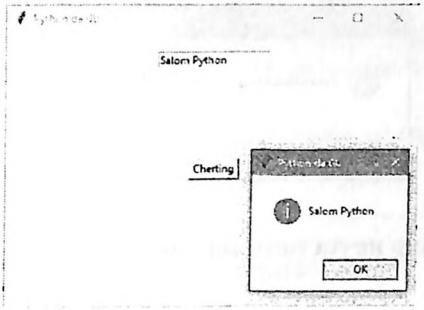
Бу ерда **master** асосий ойнага ҳавола, **options** эса қуйидаги параметрлар тўплами:

- **bg** – фон ранги;
- **bd** – чегара қалингили;
- **cursor** – компонентага сичқончани олиб борилгандаги қўриниши;
- **fg** – ёзув ранги;
- **font** – ёзув шрифти;
- **justify** – ёзувни текислашни белгилаш. **LEFT** – чап томон бўйича текислаш, **CENTER** – марказ бўйича, **RIGHT** – ўнг томон бўйича текислаш;
- **relief** –чегара типини танлайди, агар қиймат берилмаса автоматик равишда **FLAT** қийматини қабул қилади;
- **selectbackground** – ёзувнинг ажратилган қисмидаги фон ранги;
- **selectforeground** – ёзувнинг ажратилган қисмидаги ёзув ранги;
- **show** – чиқариладиган белгилар учун маска тайинланади;
- **state** – элементнинг ҳолати, **NORMAL** (автоматик) ва **DISABLED** қийматларни қабул қилиш мумкин;
- **textvariable** – компонентани **StringVar** компонентаси билан боғлашни ўрнатиш;

– **width** – компонентанинг кенглиги.

Entry компонентаси киритилган маълумотларни тугма чертилгандаги ҳодиса ёрдамида бошқариш:

```
from tkinter import *
from tkinter import messagebox
def show_message():
    messagebox.showinfo("Python da GUI", message.get())
root = Tk()
root.title("Python da GUI")
root.geometry("300x250")
message = StringVar()
message_entry = Entry(textvariable=message)
message_entry.place(relx=.5, rely=.1, anchor="c")
message_button = Button(text="Cherting", command=show_message)
message_button.place(relx=.5, rely=.5, anchor="c")
root.mainloop()
```



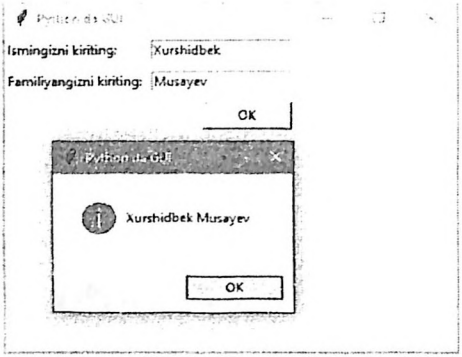
9.11–расм: Хабар ойнасига маълумот чиқариш

Бу ерда хабарни чиқариш учун матн майдонига киритилган матнни кўрсатиш учун **showinfo()** функциясини ўз ичига олган қўшимча **messagebox** модули ишлатилди. Киритилган матн олиш учун **StringVar** компонентидан фойдаланилди.

Киритиш майдони контейнерларда бир эмас бир нечта бўлса ҳар бир киритиш майдони учун алоҳида – алоҳида StringVar компонентидан фойдаланилади:

```
from tkinter import *
from tkinter import messagebox
def display_full_name():
    messagebox.showinfo("Python da GUI", name.get() + " " + surname.get())
root = Tk()
root.title("Python da GUI")
name = StringVar()
```

```
surname = StringVar()
name_label = Label(text="Ismingizni kiriting:")
surname_label = Label(text="Familiyangizni kiriting:")
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()
```



9.12–расм: Бир нечта матн майдонлардан фойдаланиш

Entry методи

Entry методи бир нечта методларга эга. Уларнинг асосийлари қуйида берилган:

- **insert(index, str)** – матн майдонига маълум бир индекс бўйича сатрни жойлаш;
- **get()** – матн майдонига киритилган сатрни олиш;
- **delete(first, last=None)** – **first** индексдан бошлаб белгилар мажмуасини ўчириш. Агар **last** параметри кўрсатилган бўлса, унда ўчириш **last** индексгача амалга оширилади. Иккинчи параметр сифатида **END** қийматидан фойдаланиш мумкин. Бунда ўчириш сатрни охиригача амалга оширилади.

Юқорида келтирилган методларга мисол:

```
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("Python da GUI")
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)
name_entry.insert(0, "Temur")
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()
```

Дастурни ишга тушириш билан икки матн майдонига ёзувлар автоматик равишда ёзилади:

```
name_entry.insert(0, "Temur")
surname_entry.insert(0, "Soyer")
```

delete() методини чақириш билан иккала майдонни тозалаш амалга оширилади:

```
def clear():
        name_entry.delete(0, END)
        surname_entry.delete(0, END)
```

Иккинчи тугма **get()** методидан фойдаланиб киритилган сатрларини хабар ойнаси орқали экранга чиқаради:

```
def display():
        messagebox.showinfo("GUI Python", name_entry.get() + " " +
surname_entry.get())
```

9.7. Checkbutton, Radiobutton, Listbox компонентaлари

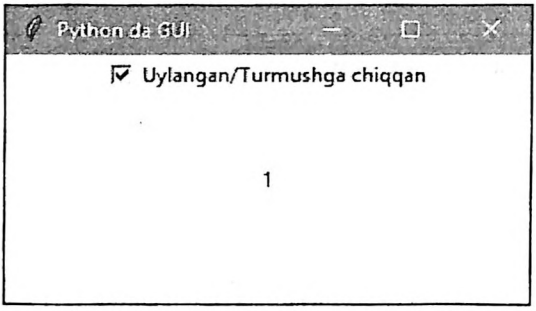
Checkbutton компонентаси

Checkbutton компонентаси икки турғун ҳолатнинг бири ўзида сақлаш хусусиятига эга бўлган элементдир. Бу ҳолатлар белгиланган ёки белгиланмаган ҳолатлардир. Масалан,

фойдаланувчининг оилали ёки оиласизлигини текшириш учун ушбу компонентадан фойдаланиш мумкин.

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
root.geometry("300x150")
ismarried = IntVar()
ismarried_checkbutton = Checkbutton(text="Uylangan / Turmushga chiqqan",
variable=ismarried)
ismarried_checkbutton.pack()
ismarried_label = Label(textvariable=ismarried)
ismarried_label.place(relx=.5, rely=.5, anchor="c")
root.mainloop()
```

Checkbutton компонентасининг ўзига ҳос хусусияти **IntVar** компонентасига **variable** параметри билан уланиш қобилиятидир. Агар компонента белгиланган бўлса, **IntVar** компонентасининг қиймати 1 га, белгиланмаган бўлса – 0 га тенг бўлади. **IntVar** компонентаси ёрдамида керакли бўлган қийматни ўрнатиш ёки уни олиш мумкин.



9.13–расм: Checkbutton компонентасидан фойдаланиш

Checkbutton компонентасининг конструктори бир нечта параметрларни ўз ичига олади, улар ёрдамида компонетанинг кўринишини созлаш мумкин:

```
Checkbutton(master, options)
```

master параметри асосий ойнага ҳавола ҳисобланади, **options** параметри эса ўз ичига қуйидаги параметрларни олиши мумкин:

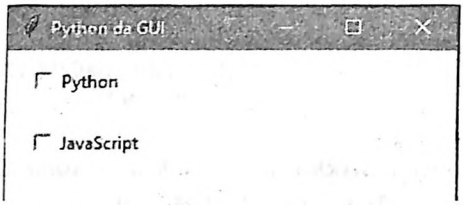
- **activebackground** – компонента босилган ҳолдаги фон ранги;
- **activeforeground** – компонентани босилган ҳолдаги ёзув ранги;
- **bg** – компонентанинг фон ранги;

- **bd** – компонентанинг атрофдаги чегараси;
- **command** – компонента чертилган ҳолатдаги чақириладиган функцияга ҳавола;
- **cursor** – компонентага сичқончани олиб борилгандаги кўриниши;
- **disabledforeground** – **DISABLED** ҳолатдаги ёзувнинг ранги;
- **font** – ёзувнинг шрифти;
- **fg** – ёзув ранги;
- **height** – компонентанинг баландлиги;
- **image** – компонентада чиқариладиган график тасвир;
- **justify** – матнни текислаш, **CENTER**, **LEFT**, **RIGHT** қийматларни олади;
- **offvalue** – компонентани белгиланмаган ҳолатдаги қийматини белгилаш, автоматик равишда 0 га тенг;
- **onvalue** – компонентани белгиланган ҳолатдаги қийматини белгилаш, автоматик равишда 1 га тенг;
- **padx** – ўнг ва чап томонидан компонента чегараси билан ёзувгача бўлган масофа;
- **pady** – юқори ва паст томонидан компонента чегараси билан ёзувгача бўлган масофа;
- **relief** – компонентанинг кўриниши, стандарт ҳолатда **FLAT** қийматига тенг;
- **selectcolor** – компонентадаги катакнинг ранги;
- **selectimage** – компонента белгиланган ҳолатдаги катакнинг тасвири;
- **state** – компонентанинг ҳолати, қийматлари **NORMAL**, **DISABLED** ва **ACTIVE** га тенг бўлаши мумкин;
- **text** – компонентанинг матни;
- **variable** – компонента ҳолатини ўзида сақловчи ўзгарувчига ҳавола;
- **width** – компонентанинг кенглиги;
- **wraplength** – компонента матнидаги белгиларни бошқа сатрга ўтказишни амалга ошириш.

Юқорида келтирилган параметрлардан фойдаланилган ҳолда, қуйида дастур берилган:

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
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()
```



9.14–расм: Бир нечта компонеталардан фойдаланиш

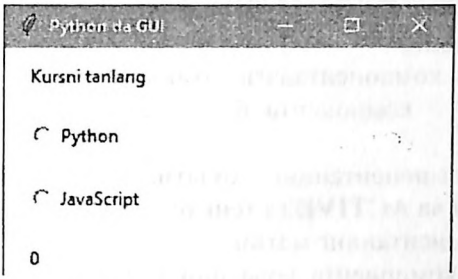
Radiobutton компонентаси

Radiobutton компонентаси икки ҳолатда бўлиши мумкин бўлган калитни ифодалайди. Ҳудди **Checkbutton** компонентасига ўхшайди, бироқ **Checkbutton** компонентасидан фарқи шундаки, **Radiobutton** гуруҳида фақат биттаси бир вақтнинг ўзида фаоллашади. **Radiobutton** компонентасидан фойдаланиш қуйидагича амалга оширилади:

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
root.geometry("300x250")
header = Label(text="Kursni tanlang", 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()
```

Бу ерда иккита **Radiobutton** компонетаси берилган. Бу икки компонента бир хил **IntVar** компонентасига ёки ўзгарувчисига боғланган. Шу билан бирга, улар **value** параметри бўйича турли

қийматларга эга. Шунинг учун битта компонента чертилганда иккинчи компонента автоматик тарзда пассив ҳолатга ўтади.



9.15–расм: Radiobutton компонентасининг кўриниши

Radiobutton компонентасини созлаш учун компонентанинг конструкторидан фойдаланиш мумкин. Бу конструктор қуйидаги кўринишга эга:

Radiobutton (master, options)

Биринчи параметр **master** параметри асосий ойнага ҳавола бўлиб, қайси контейнерга тегишлилигини билдиради. Иккинчи параметр **options** параметри ҳисобланиб, яратилаётган компонентага бошланғич қийматлар беради. Бу параметрлар қуйидагилардан иборат бўлиши мумкин:

- **activebackground** – компонента босилган ҳолдаги фон ранги;
- **activeforeground** – компонентани босилган ҳолдаги ёзув ранги;
- **bg** – компонентанинг фон ранги;
- **bd** – компонентанинг атрофидаги чегараси;
- **command** – компонента чертилган ҳолатидаги чақириладиган функцияга ҳавола;
- **cursor** – компонентага сичқончани олиб борилгандаги кўриниши;
- **disabledforeground** – **DISABLED** ҳолатдаги ёзувнинг ранги;
- **font** – ёзувнинг шрифти;
- **fg** – ёзув ранги;
- **height** – компонентанинг баландлиги;
- **image** – компонентада чиқариладиган график тасвир;
- **justify** – матнни текислаш, **CENTER**, **LEFT**, **RIGHT** қийматларни олади;
- **padx** – ўнг ва чап томонидан компонента чегараси билан ёзувгача бўлган масофа;

- **pady** – юқори ва паст томонидан компонента чегараси билан ёзувгача бўлган масофа;
- **relief** – компонентанинг кўриниши, стандарт ҳолатда **FLAT** қийматига тенг;
- **selectcolor** – компонентадаги катакнинг ранги;
- **selectimage** – компонента белгиланган ҳолатдаги катакнинг тасвири;
- **state** – компонентанинг ҳолати, қийматлари **NORMAL**, **DISABLED** ва **ACTIVE** га тенг бўлиши мумкин;
- **text** – компонентанинг матни;
- **variable** – компонента ҳолатини ўзида сақловчи ўзгарувчига ҳавола;
- **width** – компонентанинг кенглиги;
- **wraplength** – компонента матнидаги белгиларни бошқа сатрга ўтказишни амалга ошириш;

Radiobutton компонентасидан фойдаланилган ҳолда мураккаброқ кўринишда бўлган дастур қуйида берилган:

```
from tkinter import *
languages = [("Python", 1), ("JavaScript", 2), ("C#", 3), ("Java", 4)]
def select():
    l = language.get()
    if l == 1:
        sel.config(text="Tanlangan kurs Python")
    elif l == 2:
        sel.config(text="Tanlangan kurs JavaScript")
    elif l == 3:
        sel.config(text="Tanlangan kurs C#")
    elif l == 4:
        sel.config(text="Tanlangan kurs Java")
root = Tk()
root.title("Python da GUI")
root.geometry("300x280")
header = Label(text="Kursni tanlang", 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()

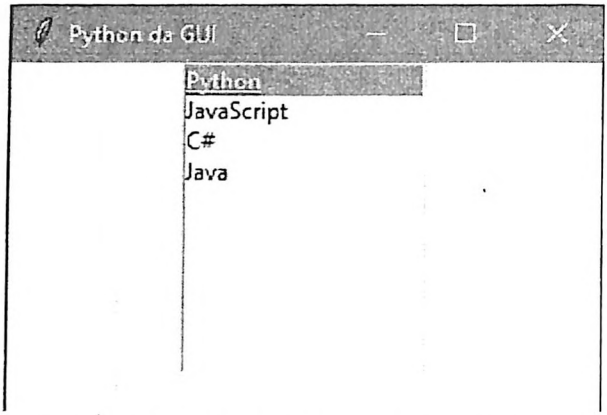
9.16–расм: Бир нечта Radiobutton лардан фойдаланиш

Listbox компонентаси

Tkinterда **Listbox** компонентаси объектларнинг рўйхатини ташкил этиш учун ишлатилади. Оддий рўйхат ташкил этиш қуйидагича амалга оширилади:

```
from tkinter import *
languages = ["Python", "JavaScript", "C#", "Java"]
root = Tk()
root.title("Python da GUI")
root.geometry("300x280")
languages_listbox = Listbox()
for language in languages:
    languages_listbox.insert(END, language)
languages_listbox.pack()
root.mainloop()
```

Listbox компонентасини тўлдириш учун **languages** рўйхатидан фойдаланилган, бу рўйхатнинг элементларига **for** цикли ёрдамида мурожаат амалга оширилган. Ҳар бир элемент **insert()** методи ёрдамида **Listbox** компонентасига қўшилади. **insert()** методининг биринчи параметри сифатида жойлаштирилаётган элементнинг қўшиш индекси жойлаштирилади. Элементи рўйхатнинг охирги қисмига жойлаштириш учун **END** қиймати ишлатилиши мумкин.



9.17–расм: Listbox компонентасининг кўриниши

Listbox компонентаси созлаш учун унинг конструкторидан фойдаланиш керак. **Listbox** компонентаси конструктори қуйидаги параметрлардан бир ёки бир нечтасини қабул қилиши мумкин:

- **bg** – фон ранги;
- **bd** – компонентанинг атрофидаги чизиқ ранги;
- **cursor** – курсорни **Listbox** компонентасининг устига боргандаги кўриниши;
- **font** – шрифт созламалари;
- **fg** – матн ранги;
- **height** – компонентанинг баландлиги. Агар кўрсатилмаса, компонента автоматик равишда 10 та сатр чиқариш учун мосланади;
- **highlightcolor** – компонента актив ҳолдаги ранги;
- **highlightthickness** – компонентанинг актив ҳолдаги чизиқлар қалинлиги;
- **relief** – компонентанинг турини танлайди. Агар кўрсатилмаса, автоматик ҳолатда **SUNKEN** қийматини олади. Олиши мумкин бўлган қийматлар **FLAT, RAISED, GROOVE, RIDGE;**
- **selectbackground** – белгиланган сатрнинг элементи учун фон ранги;
- **selectmode** – кўп белгиланадиган сатрларга рухсат беришни ўрнатиш. Қуйидаги қийматларни қабул қилиши мумкин: **BROWSE, SINGLE, MULTIPLE, EXTENDED**. Масалан: кўп

сатрларни белгилаш учун **MULTIPLE** ёки **EXTENDED** қийматларидан фойдаланиш мумкин;

- **width** – компонентанинг кенглигини белгиларда ифодалаш. Автоматик тарзда 20 та белги қабул қилинган;
- **xscrollcommand** – горизонтал айлантиришда бажариладиган буйруқ;
- **yscrollcommand** – вертикал айлантиришда бажариладиган буйруқ.

Listbox компонентаси билан ишлашда айлантиришларни яратиш бир мунча қийинчилик келтириб чиқаради. Буни яратиш қуйидаги дастурда берилган:

```
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("Python da GUI")
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()
```

Югургич (scroll) яратиш учун **Scrollbar** компонентасидан фойдаланилади. **Listbox** компонентасини вертикал бўйлаб ўтказиш учун, унга **yscrollcommand=scrollbar.set** параметри ва қиймати берилади.

Listbox компонентасининг асосий методлари

Listbox компонентасининг ҳолати ва ундаги маълумотлар билан ишлаш учун бир қатор методлар мавжуд. Булардан бир нечтаси қуйида берилган:

- **curselection()** – танланган аъзоларнинг индекслар тўпламини қайтаради;
- **delete(first, last = none)** – индекслари **[first, last]** оралиғидаги элементларни ўчиради. Иккинчи параметрда қиймат мавжуд бўлмаса, унда фақат биринчи параметрда кўрсатилган аъзони ўчиради;

– **get(first, last = None)** – **[first, last]** оралиғидаги кўрсатилган аъзоларни ўз ичига олган туркум (кортеж) қайтаради. Иккинчи параметр инобатга олинмаган бўлса, фақатгина **first** индексидаги аъзонинг матнини қайтаради;

– **insert(index, element)** – **element** қийматини маълум бир индекс бўйича жойлаштиради;

– **size()** – аъзолар сонини қайтаради.

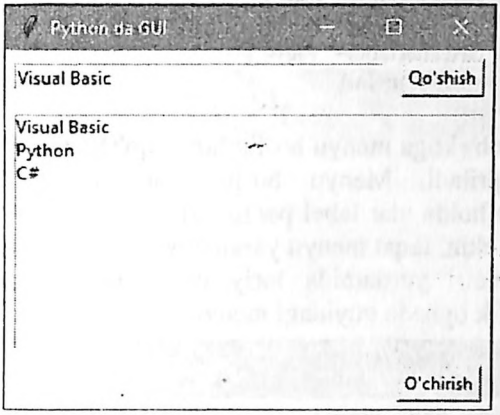
Ушбу усулларни кўриб чиқиш учун қуйида дастур кўрсатилган:

```
from tkinter import *
# Belgilangan elementni o`chirish
def delete():
    selection = languages_listbox.curselection()
    # indeks bo`yicha o`chiriladigan indeksni olish
    # selected_language = languages_listbox.get(selection[0])
    languages_listbox.delete(selection[0])
# yangi element qo`shamiz
def add():
    new_language = language_entry.get()
    languages_listbox.insert(0, new_language)
root = Tk()
root.title("Python da GUI")
# Yozuv maydoni va royxatga qo`shish uchun tugma
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)
# royxat tuzamiz
languages_listbox = Listbox()
languages_listbox.grid(row=1, column=0, columnspan=2, sticky=W+E, padx=5,
pady=5)
# royxatga boshlang`ich elementlarini kiritamiz
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()
```

Рўйхатнинг аъзоларини ўзгартириш учун бу ерда иккита тугма келтирилган. Биринчи тугма матн майдонига киритилган маълумотни **Listbox** компонентасининг аъзоси сифатида жойлаштиради. Бунинг учун дастурда **add()** номли функция

келтирилган, бу функция **insert()** методини ишлатиб рўйхатнинг бошига кўрсатилган қийматни қўшади.

Иккинчи тугма танланган аъзони ўчириш учун хизмат қилади. Буни амалга ошириш учун биринчи навбатда танланган индексларни **curselection()** методи ёрдамида топиб олиш лозим. Бу мисолда фақат битта элемент танганганлиги учун **selection[0]** нинг қийматини олиш етарли. Кўрсатилган индексли аъзони ўчириш учун **delete()** методидан фойдаланилади.



9.18–расм: Listbox компонентасидан фойдаланиш

9.8. Менюлар яратиш

Python дастурлаш тилида асосий менюлар яратиш учун **Menu** компонентасидан фойдаланилади. Менюлар ўз ичига кўп элементларни олиши мумкин ва бу элементларнинг ўзи ҳам менюлардан ташкил топиши мумкин. **Menu** компонентасидан фойдаланишда менюга қўшилаётган элементга қараб методлар турлича бўлиши мумкин. **Menu** компонентасининг қуйидагича методлари мавжуд:

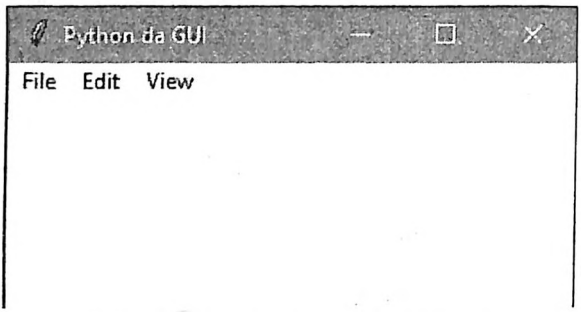
- **add_command(options)** – **options** параметри орқали меню элементи қўшиш;
- **add_cascade(options)** – меню элементи қўшади ва уни ўз навбатида меню остига рухсат беради;
- **add_separator()** – менюларни ажратувчи чизиқ қўшади;
- **add_radiobutton(options)** – менюга бир танловли меню элементини қўшади;

– **add_checkbutton(variantlar)** – менюга кўп танловли меню элементини қўшади.

Қуйида оддий меню яратиш кўрсатилган:

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
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()
```

Menyu ob'ektiga menyu bo`limlarini qo'shish uchun add_cascade() metodi chaqiriladi. Menyu bo`lim parametrlari ushbu metodga qo`shiladi, bu holda ular label parametri belgilangan matn yorlig'i bilan ifodalanadi. Lekin, faqat menyu yaratish yetarli emas. Config() metodida menyu parametri yordamida joriy oyna uchun o'rnatilishi kerak. Natijada, grafik oynada quyidagi menyu mavjud:



9.19–расм: Менюнинг кўриниши

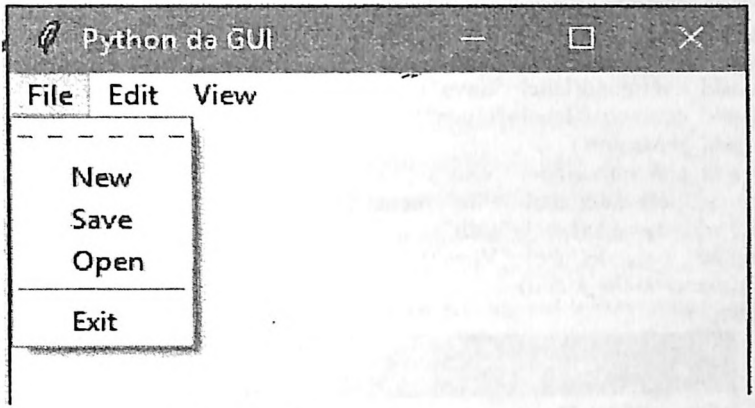
Юқорида кеклтирилган менюларга остки менюлар қўшиш қуйидагича амалга оширилади:

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
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()
```

file_menu ўзгарувчидан фойдаланиб, меню яратилади, бу яратилган менюни бошқа яратилаётган менюнинг параметри сифатида кўрсатилиб, меню ости ҳосил қилинади:

```
main_menu.add_cascade(label="File", menu=file_menu)
```



9.20–расм: Ост меню ҳосил қилиш

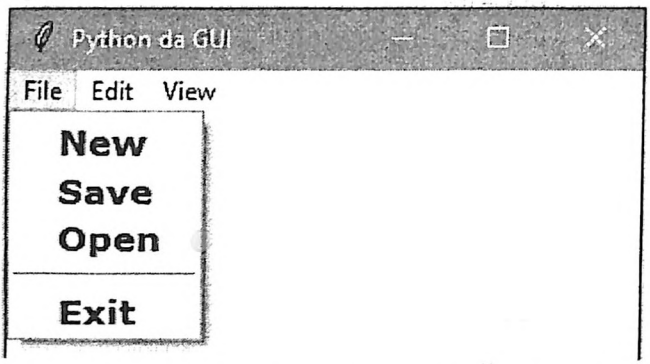
Яратилаётган менюларни созлаш учун қуйидаги параметрлардан менюнинг конструкторида фойдаланиш мумкин:

- **activebackground** – фаол меню элементининг ранги;
- **activeborderwidth** – фаол меню элементининг чегара қалинлиги;
- **activeforeground** – фаол меню элементининг матн ранги;
- **bg** – фон ранги;
- **bd** – чегара қалинлиги;
- **cursor** – меню элементининг устида сичқонча курсорининг кўриниши;
- **disabledforeground** – меню элементининг пассив ҳолатдаги ранги;
- **shrift** – матни шрифти;

– **fg** – матн ранги;

– **tearoff** – менюни ойнадан ажратиш мумкин. Меню остиларини яратишда, скриншот қилишда менюнинг юқори қисмида узуқ-узуқ чизиқларни мумкин ва менюни ўчириш ҳам мумкин. Агар **tearoff = 0** қиймати қабул қилса, меню остини узиб бўлмайди.

```
from tkinter import *
root = Tk()
root.title("Python da GUI")
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()
```



9.21–расм: Меню остининг хусусиятларини ўзгартириш

Меню билан ўзаро алоқа

Меню элементларининг ўзига хос хусусиятларидан бири – бу фойдаланувчи томонидан берилган буйруқларга мос равишда жавоб қайтариш ҳисобланади. Буни амалга ошириш учун ҳар бир менюдаги элемент учун босилганда бажариладиган функцияларга ҳавола кўрсатадиган **command** параметрини ўрнатиш мумкин.

```
from tkinter import *
from tkinter import messagebox
def edit_click():
        messagebox.showinfo("Python da GUI", "Edit menyusi chertildi")
root = Tk()
root.title("Python da GUI")
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()
```

Назорат саволлари:

1. Tkinter нима?
2. Элементлар хусусиятини тушунтириб беринг?
3. Label компонентасини тушунтириб беринг?
4. Маълумотларни киритиш майдонини тушунтиринг?
5. Бир ва кўп танловли компоненталарни тушунтириб беринг?
6. Менюлар яратиш қандай амалга оширилади?
7. Менюлар билан ишловчи методларни тушунтириб беринг?

Мундарижа:

Х.Ш.МУСАЕВ,
А.М.ҚАЮМОВ

PYTHON ДАСТУРЛАШ ТИЛИ

(Ўқув қўлланма)

Тошкент – «Aloqachi» – 2020

Муҳаррир: Қ. Маткурбонов
Тех. муҳаррир: А. Тоғаев
Мусаввир: Б. Эсанов
Мусаҳҳиҳа: Ф. Тагаева
Компьютерда саҳифаловчи: Ш. Тўхтамуродов

Нашр. лиц. ii № 176, 11.06. 2010.
Босишга рухсат этилди 22.07.2019.
Бичими 60x84 $^{1}/_{16}$. «Times Uz» гарнитураси.
Шартли босма табоғи 9,5. Нашр босма табоғи 9,0.
Адади 100. Буюртма № 97.

«Nihol print» ОК да чоп этилди.
Тошкент шаҳри, Мухтор Ашрафий кўчаси, 99./101.