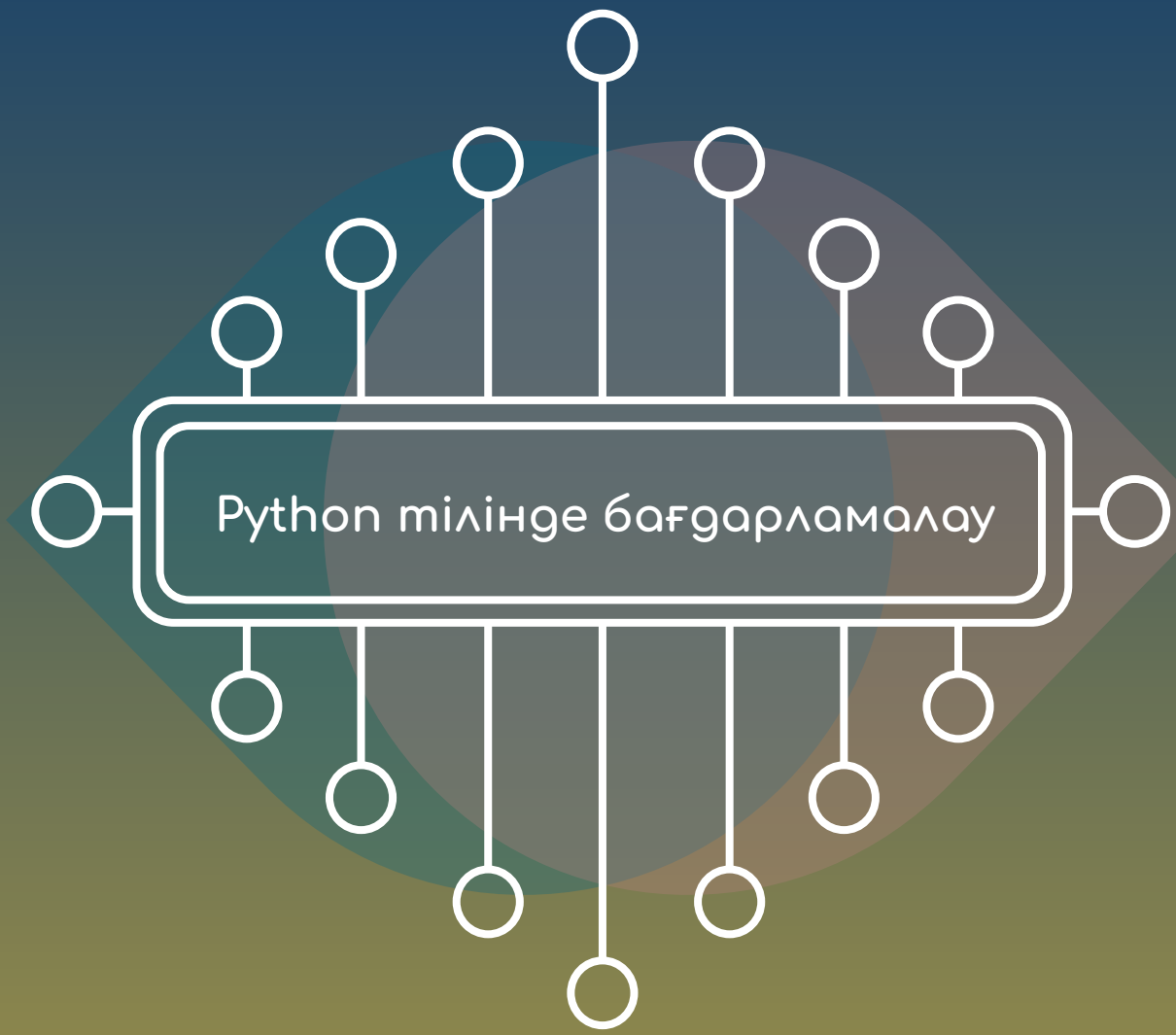


“Жасанды Интеллект инженері болу”
сериясының бірінші кітабы

Мағжан
ҚАЙРАНБАЙ

1

Алғы сөз

Бұл оқулық бағдарламалауға енді ғана аяқ басқан шәкірттерге арналған. Оқулықты оқып, сондай-ақ кішігірім жаттығулар жасаған шәкірт Python тілінде өте қарапайым бағдарламалар жаза алатын дәрежеге көтеріледі. Осы оқулықты толықтай меңгеріп болған соң, алгоритмдік есептерді Python бағдарламалау тілі арқылы көптеп шығарған жөн. Сіз осы оқулықтан осылайша үйренген білімдеріңізді практикалық тұрғыдан шыңдай аласыз.

Оқулық материалдары w3school оқу порталын негізге ала отырып жасалған.

Осы оқулыққа қатысты есептерді www.itboo.kz/pythonproblems.php сайтынан тауып, шығара аласыздар.

Осы кітаптың шығуына және таралуына үлес қосқан **“Academica Courses”** ұйымына алғысым шексіз.

Кітап мазмұны

1. Кіріспе
1.1. Python дегеніміз не?....7
1.2. Python тілі арқылы не істей аламыз?....7
1.3. Неге Python тілін қолдану керекпіз?....7
1.4. Білу керек ақпараттар....7

2. Орнату
2.1. Python тілін Windows-қа орнату....11
2.2. Python тілін Mac-қа орнату....15

3. Синтаксис
3.1. Python бағдарламасын жазу....27
3.2. Python тіліндегі шегініс (Tab)....29

4. Пікір
4.1. Бір жолды пікір....35
4.2. Бірнеше жолды пікір....35

5. Айнымалы
5.1. Айнымалы....39
5.2. Айнымалыны өзге типке ауыстыру (casting)....39
5.3. Мәтіндік айнымалы....39
5.4. Айнымалылардың аталуы....39
5.5. Айнымалыларды мәнге теңеу....41
5.6. Айнымалыларды экранға шығару....41

6. Деректер типі
6.1. Деректер типі....45

7. Сандар
7.1. Сандар түрлері....49
7.2. Сандарды бір түрден екінші түрге алмастыру....49
7.3. Кездейсоқ сандар....49

8. Мәтіндік тип
8.1. Мәтіндік тип....53
8.2. Ішкі мәтін....53
8.3. Мәтінді бір-біріне біріктіру....54

9. Булин (Boolean)
9.1. Булин (Boolean)....57

10. Оператор
10.1. Оператор түрлері....61
10.2. Арифметикалық операторлар....61
10.3. Меншіктеу операторлары....61
10.4. Салыстыру операторлары....62
10.5. Логикалық операторлар....62
10.6. Сәйкестендіру операторлары....63

11. Тізбек
11.1. Тізбектер....67
11.2. Тізбектің ұзындығы....67
11.3. Тізбекке элементті қосу, тізбектен элементті өшіру....67

12. Шартты оператор
12.1. If else....71
12.2. Логикалық оператор....73
12.3. Бірінің ішіне салынған шартты оператор....73
12.4. Қысқартылған if тұжырымы....74

13. For циклі
13.1. For циклі....77
13.2. Break тұжырымы....78
13.3. Continue тұжырымы....78
13.4. Range тұжырымы....79
13.5. Бірінің ішіне салынған for циклі....79
13.6. Pass тұжырымы....80

14. While циклі
14.1. while циклі....85
14.2. break тұжырымы....86
14.3. continue тұжырымы....86
14.4. Бірінің ішіне салынған while циклі....87

15. Функция
15.1. Функция....91
15.2. Функция параметрі....91
15.3. Параметрлер саны....92
15.4. Параметрлерге тізбекті жіберу....92
15.5. Функцияның қайтару мәні....93
15.6. Тізбекке қолданылатын функциялар....93

16. Класс
16.1. Объектіге бағытталған бағдарламалау....99
16.2. Конструктор....100
16.3. Класс функциясы....100
16.4. __str__ функциясы....101

17. Мұрагерлік
18.7. Мұрагерлік....105

18. try except
18.1. try except....109
18.2. Бірнеше except блогы....110
18.3. else блогы....110
18.4. Finally блогы....111

19. Input
19.1. Input....115

20. pip
20.1. pip құралы....119
20.2. pip арқылы пакет орнату....119
20.3. Орнатылған пакеттер тізімі....120

21. Файлмен жұмыс істеу
21.1. Файлмен жұмыс....123
21.2. Файлдан оқу....123
21.3. Файлға жазу....125
21.4. Файл құру....126

1

Кіріспе

1.1. Python дегеніміз не?

Python — қазіргі таңдағы ең танымал бағдарламалау тілдерінің бірі. Python тілін 1991 жылы Гидо ван Россум жазып шығарған. Python тілі келесі мақсаттарда кеңінен қолданылады:

- веб-әзірлеу (сервер жағында)
- бағдарламалық қамтамасыз ету
- математика
- тағы басқа

Төменде Python тіліне қатысты кеңінен тараған сұрақтарға жауап береміз.

1.2. Python тілі арқылы не істей аламыз?

- Python веб-қосымшаларды жасау үшін қолданылады.
- Python дерекқор жүйелеріне қосыла алады. Сонымен бірге файлдарды оқып, оларды өзгерте алады.
- Python үлкен деректерді өңдеу үшін және күрделі математикалық амалдар орындау үшін пайдаланылады.
- Python жылдам прототиптеу үшін немесе өндіріске дайын бағдарламалық жасақтаманы әзірлеу үшін пайдаланылады.

1.3. Python тілін не үшін қолданамыз?

- Python әртүрлі платформаларда жұмыс істейді (Windows, Mac, Linux, Raspberry Pi және т.б.).
- Python тілі ағылшын тіліне ұқсас қарапайым синтаксистік құрылыммен ерекшеленеді.
- Python басқа бағдарламалау тілдеріне қарағанда жазылуы қысқа әрі жеңіл болып келеді.
- Python тілі арқылы прототиптеу процесі өте жылдам жүреді.
- Python процедуралық және объектіге бағытталған немесе функционалды түрде жұмыс істей алады.

1.4. Білуге тиісті ақпараттар

- Python тілінің ең соңғы негізгі нұсқасы — Python 3. Python 3 тілін осы оқулықта қолданатын боламыз. Дегенмен, Python 2 нұсқасы әлі де кеңінен танымал.
- Бұл оқулықта Python мәтіндік редакторда жазылады. Python тілін Thonny, Pycharm, Netbeans немесе Eclipse сияқты өзара біріктірілген даму ортасында да жазуға болады.
- Python тілі басқа бағдарламалау тілдеріне қарағанда, жазуға жеңіл әрі ыңғайлы. Мысалы, экранға “Salem alem” мәтінін экранға шығаратын бағдарламаны Java, C++, Python тілдерінде жазып көрсетейік. Java, C++ бағдарламалау тілдерінде 5-6 жол керек болса, Python тілінде 1 ғана жол керек болады. Мысалы:

Java

```
1. class HelloWorld {
2.     public static void main(String[] args) {
3.         System.out.println("Salem alem");
4.     }
5. }
```

C++

```
1. #include <iostream>
2.
3. int main() {
4.     std::cout << "Hello World!";
5.     return 0;
6. }
```

Python

```
1. print('Salem alem')
```

2

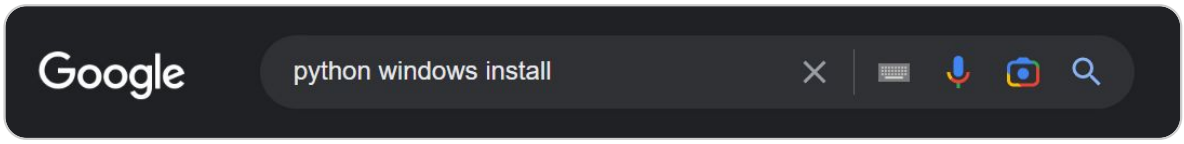
Орнату

Бұл оқулықта Python тілін Windows және Mac операциялық жүйелеріне орнату жолын көрсетеміз.

2.1. Python тілін Windows-қа орнату

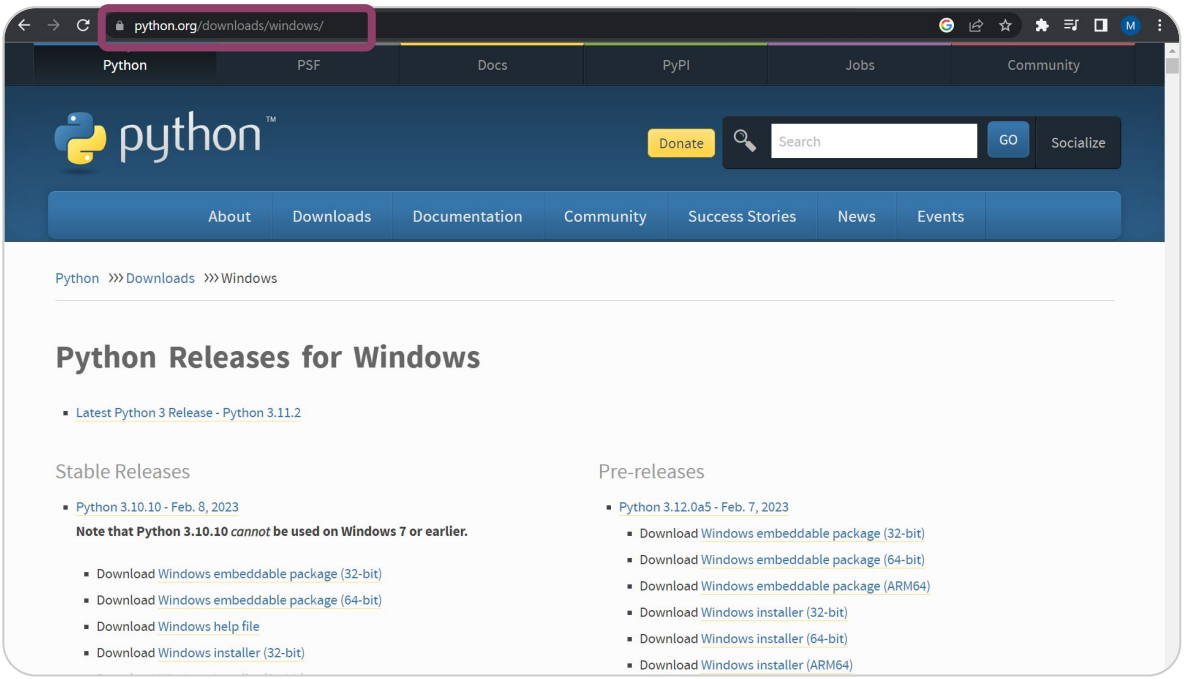
Python тілін Windows операциялық жүйесіне орнату үшін төмендегі қадамдарды жүзеге асыру керек.

1. Алдымен Google-ға “python windows install” мәтінін жазу керек (2.1-сурет).



2.1-сурет: Google-ға “python windows install” мәтінін жазу

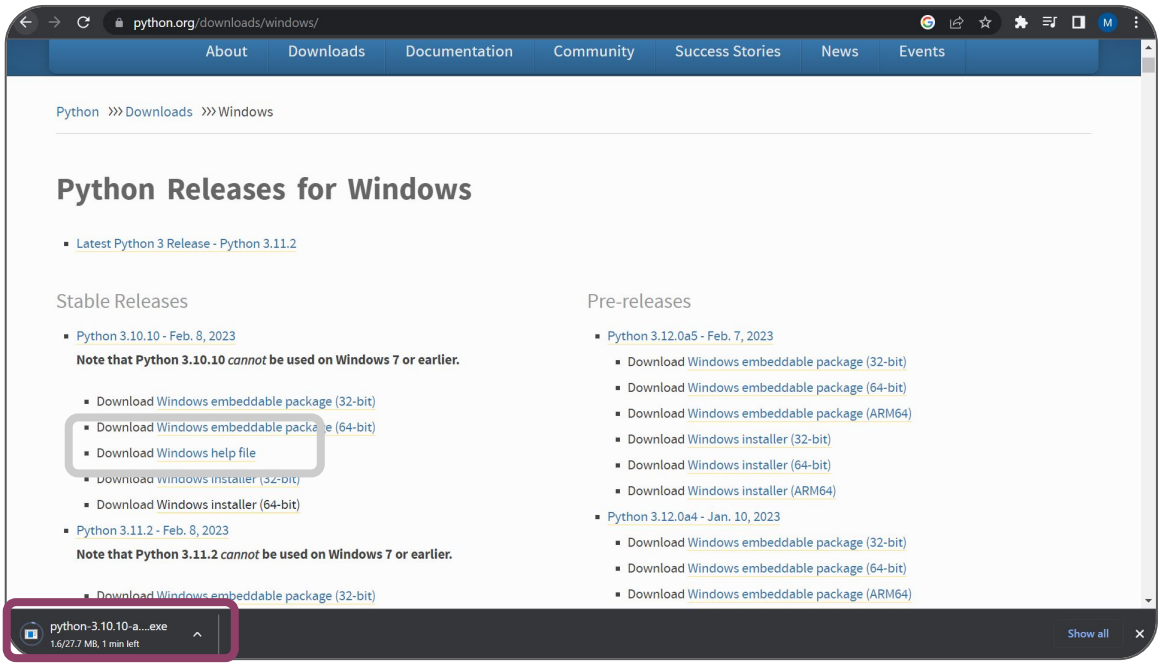
2. Осыдан кейін бірінші сілтемеге немесе мына төмендегі сайтқа көшеміз (python.org/downloads/windows) (2.2-сурет).



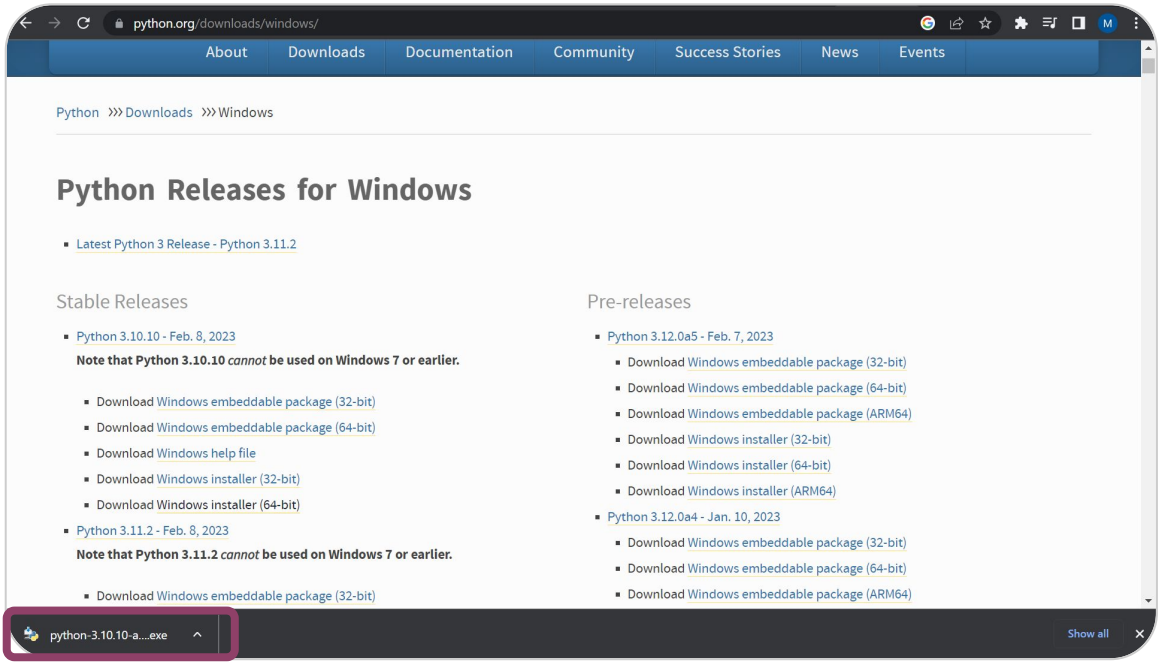
2.2-сурет: Python-ның ресми сайтына кіру

3. Жүктеп алуға арналған файлдардың бірінен “Windows installer (32 bit)” немесе “Windows installer (64 bit)” деген сілтемелердің біріне көшіп, жүктеп аламыз (2.3-сурет).

4. Жүктелген файлды кейін ашып, орнату процесін бастауымыз керек (2.4-сурет).



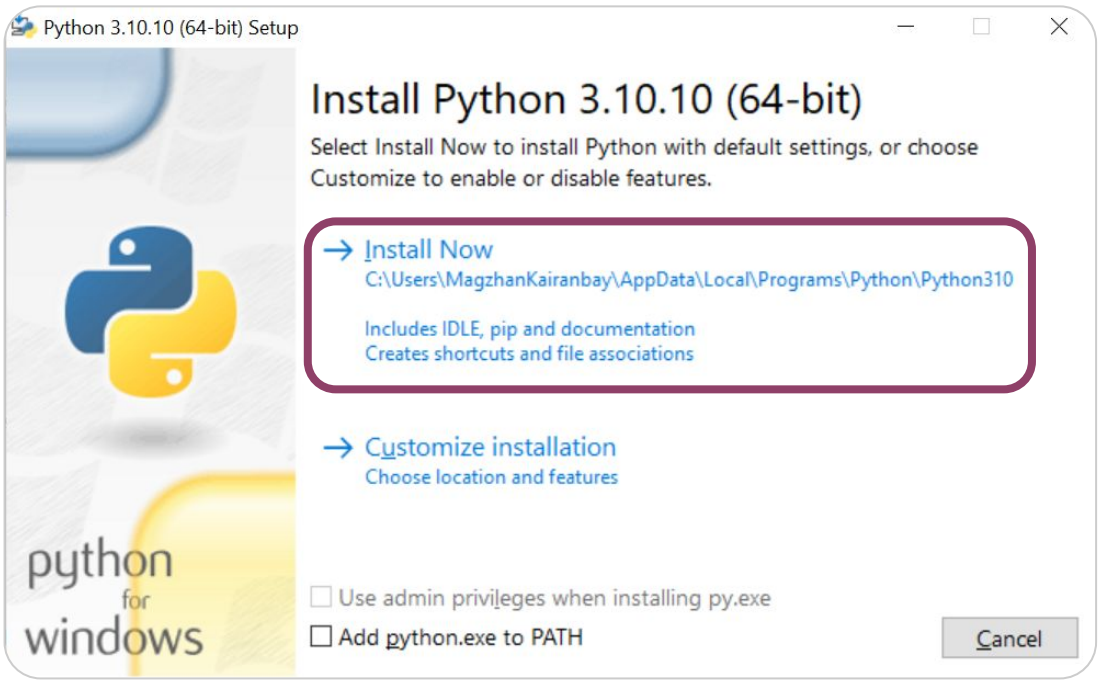
2.3-сурет: “Windows Installer (64 bit)” файлын жүктеу



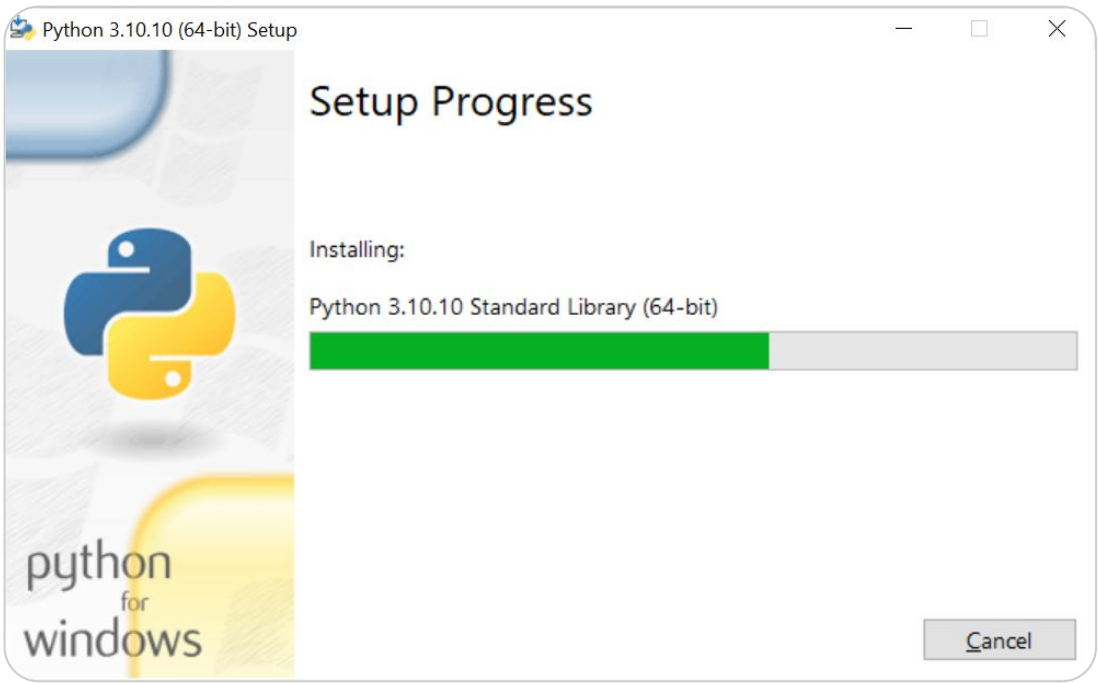
2.4-сурет: “Windows Installer (64 bit)” файлын жүктеп болғаннан кейін, орнату файлын ашу

5. Орнату терезесі ашылғаннан кейін “Install now”-ға басу керек (2.5-сурет).

6. “Install now”-ға басқаннан кейін орнату процесі басталады (2.6-сурет).



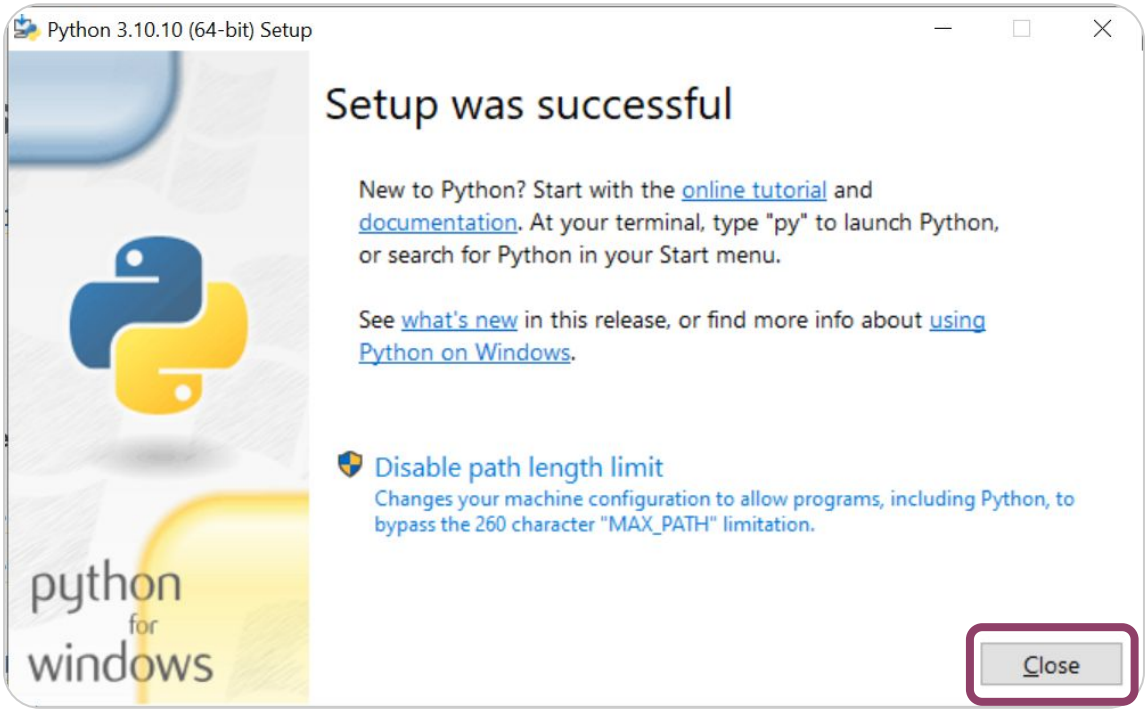
2.5-сурет: “Install now”-ға басу



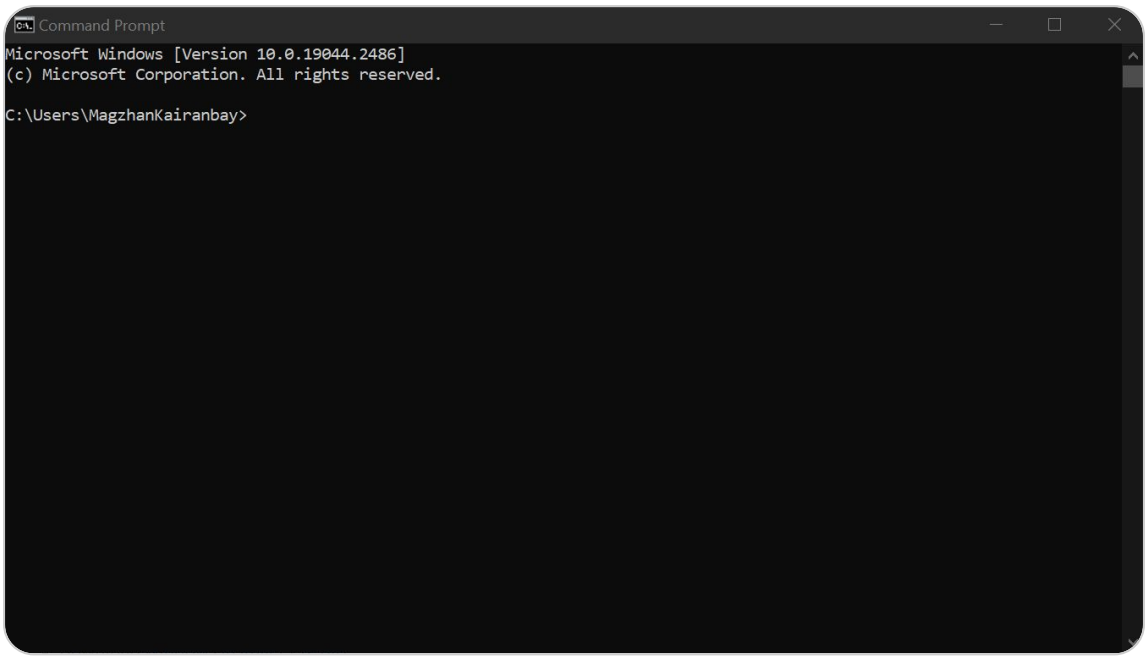
2.6-сурет: Орнату процесінің басталуы

7. Орнату процесі аяқталғаннан кейін, сіз төмендегі терезені көре аласыз (2.7-сурет).

8. Python-ды орнатқаннан кейін, Windows-қа cmd деп жазып, “Command prompt” немесе консольді ашамыз. Консоль төмендегі қара экранмен бейнеленеді (2.8-сурет).



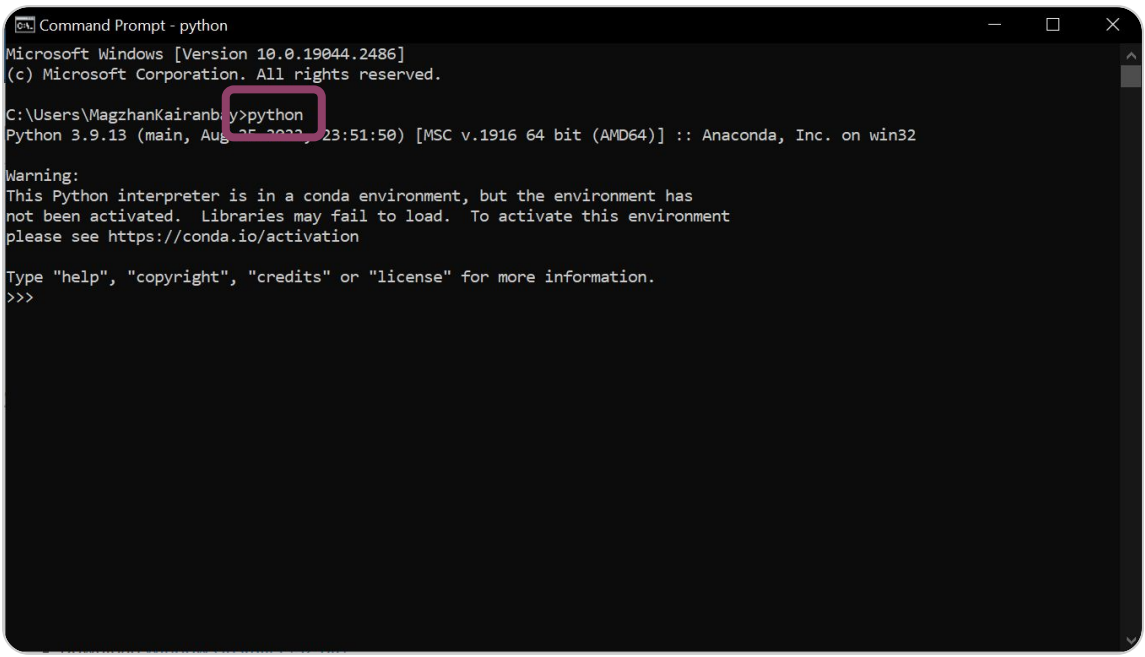
2.7-сурет: Орнатылуы толығымен аяқталғанын көрсететін терезе



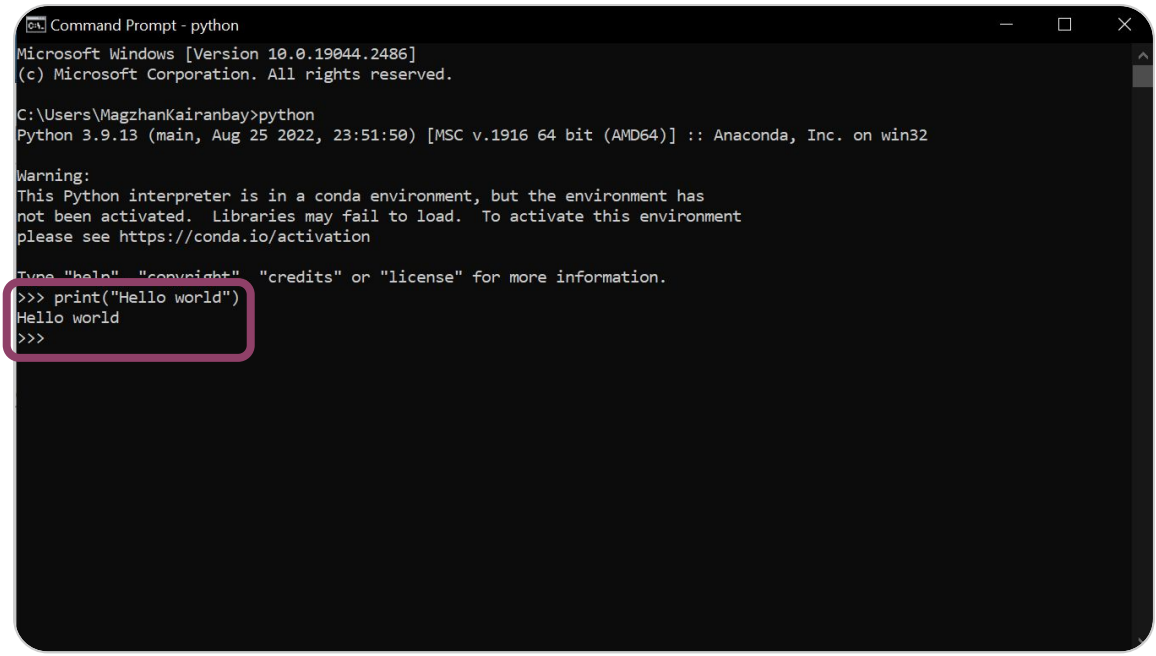
2.8-сурет: Консоль терезесі

9. Консоль терезесі ашылғаннан кейін, python сөзін жазып, Enter батырмасын басуымыз керек. Егер Python дұрыс орнатылған болса, онда төмендегідей нәтижеге қол жеткізесіз (2.9-сурет). Яғни Python терминалы пайда болып, сол жерге Python кодын жаза беруіңізге болады.

10. Python терминалы ашылғаннан кейін, төмендегідей қарапайым кодты жазайық (2.10-сурет).



2.9-сурет: Python кілтті сөзін жазып, Python терминалының ашылуы

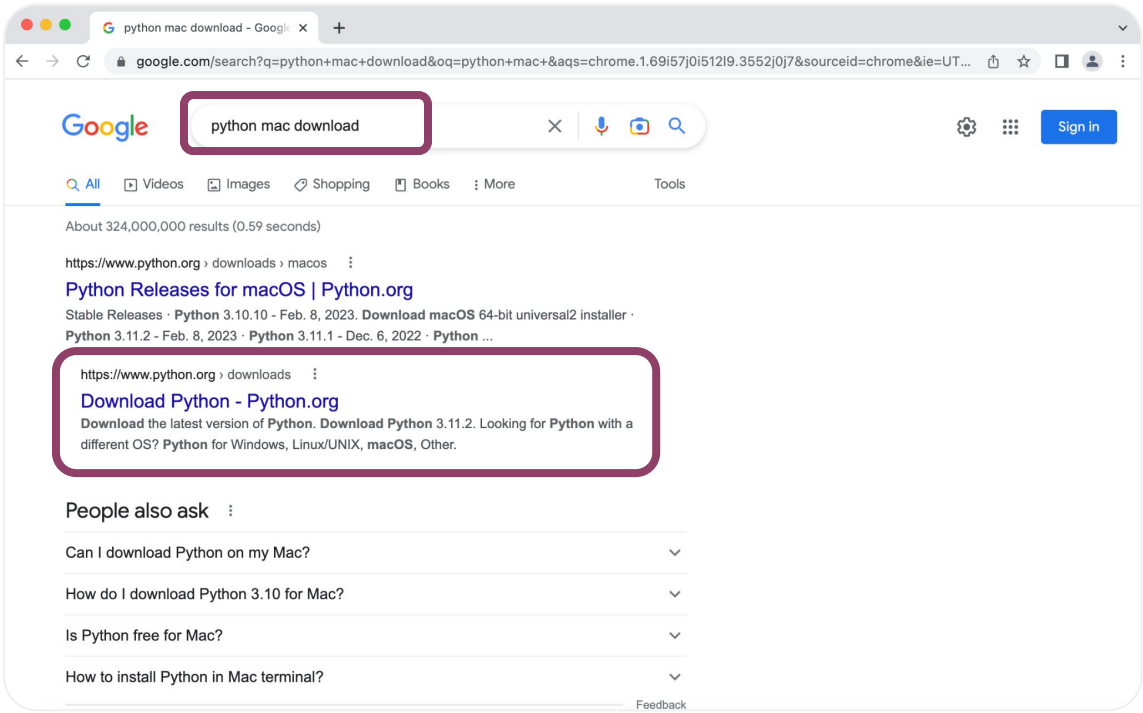


2.10-сурет: Python терминалында қарапайым кодты жазу

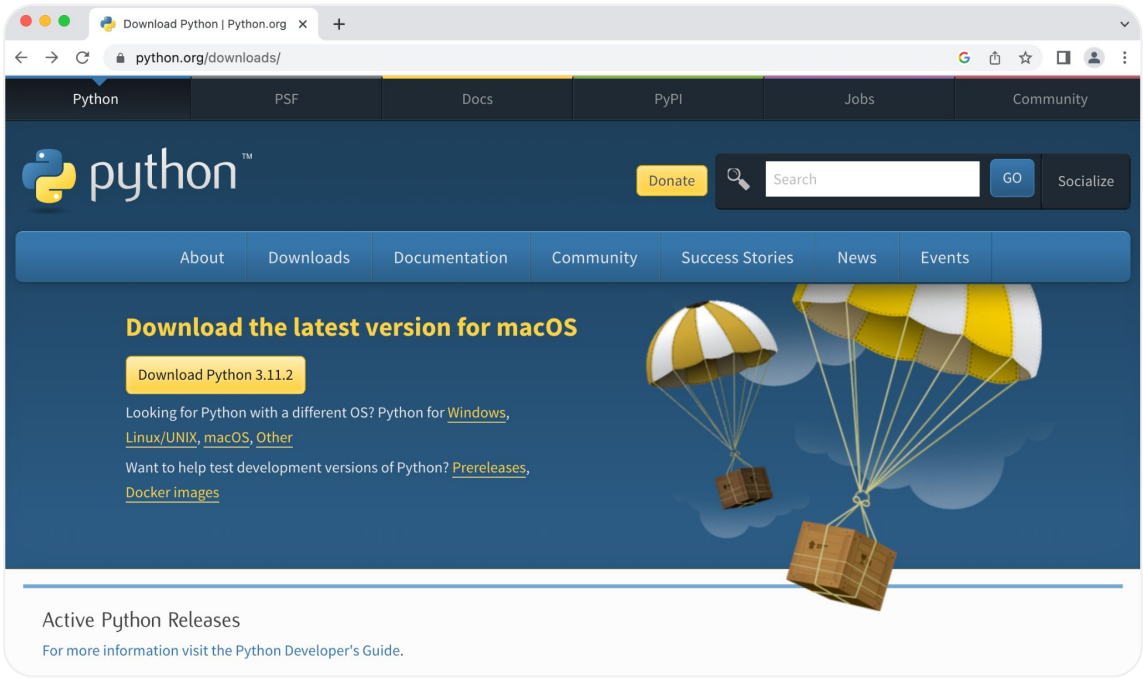
print(“Hello world”) коды экранға “Hello world” текстін шығарады. Егер осы кодты жазып, Enter батырмасын басқаннан кейін экранда “Hello world” мәтінін көрсеңіз, онда сізде Python тілі компьютеріңізге толығымен орнатылып, дұрыс жұмыс істейтінін білдіреді.

2.2. Python тілін Mac-қа орнату

Python тілін Mac операциялық жүйесіне орнату үшін, келесі қадамдарды жүзеге асыру керек.



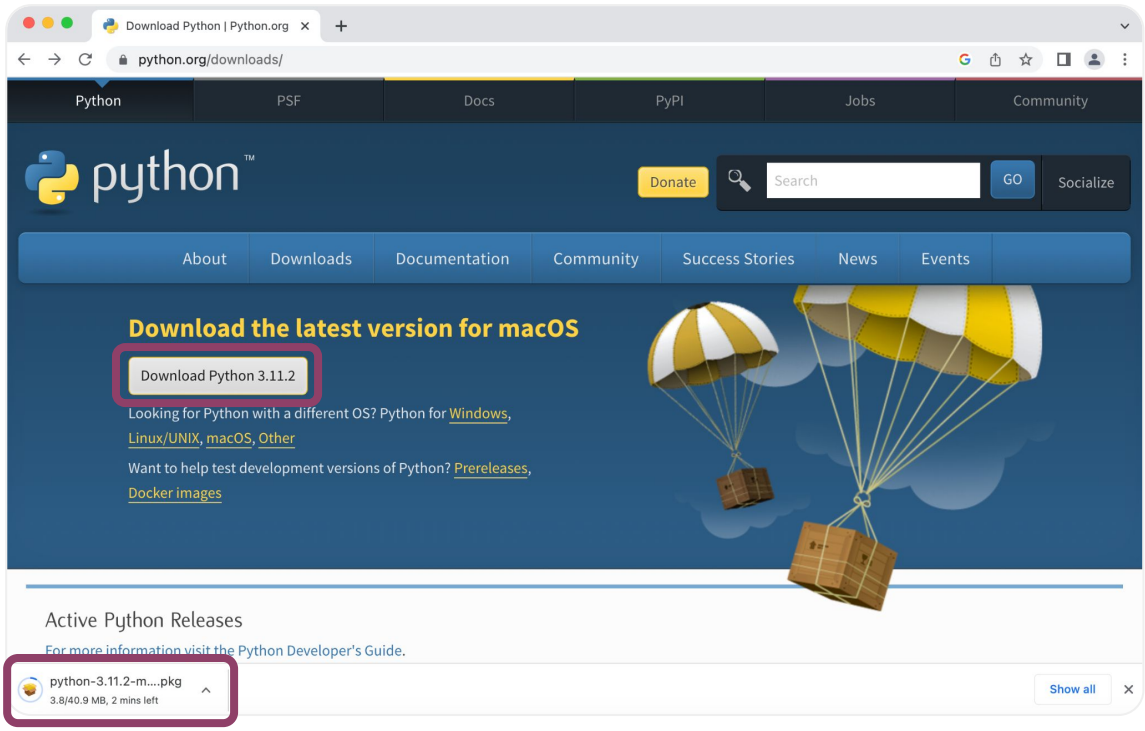
2.11-сурет: “Python mac download” мәтінін Google-ға жазып, бірінші сілтеме арқылы өтіп шығу



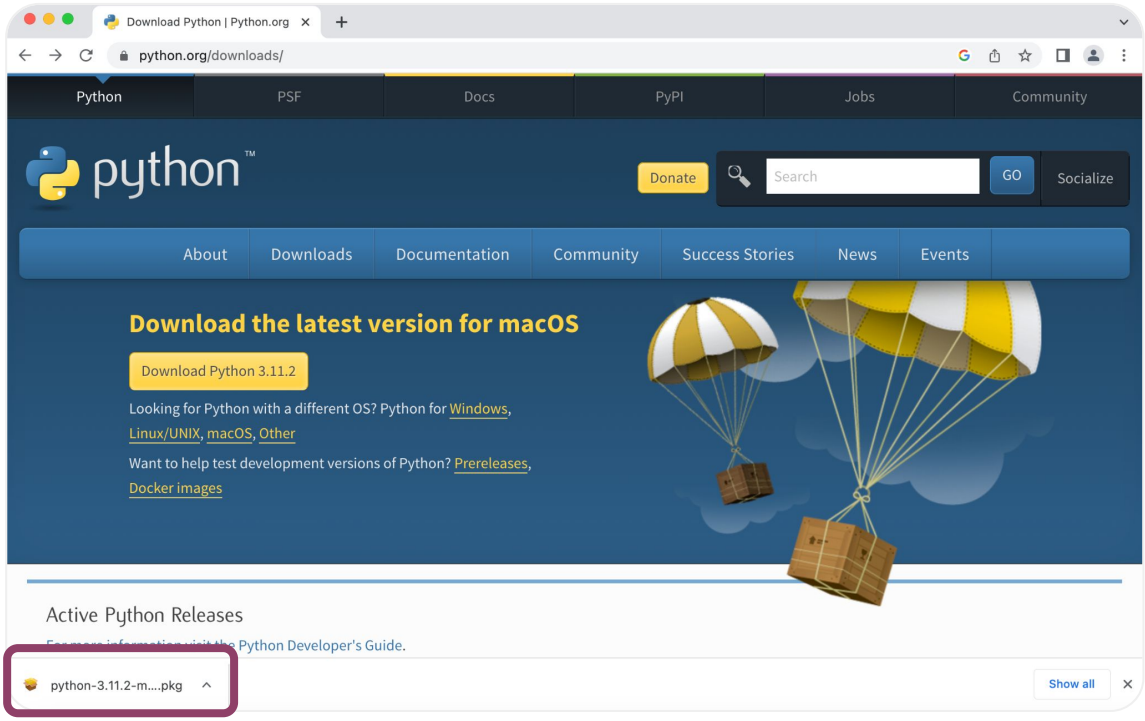
2.12-сурет: Бірінші сілтеме арқылы өтіп шығып, Python-ның ресми сайтына көшу

1. Алдымен Google-ға “python mac download” мәтінін жазамыз. Кейін бірінші сілтеме арқылы өтіп, Python тілінің ресми парақшасына көшеміз (2.11, 2.12-суреттер).

2. “Download Python 3.XX.X” батырмасын басып, Python тілінің соңғы нұсқасын жүктейміз. Python орнату бағдарламасын жүктеп алғаннан кейін, сол бағдарламаны ашып, орнату просцесін бастаймыз.



2.13-сурет: “Download Python 3.XX.X” батырмасын басып, Python тілінің соңғы нұсқасын жүктейміз



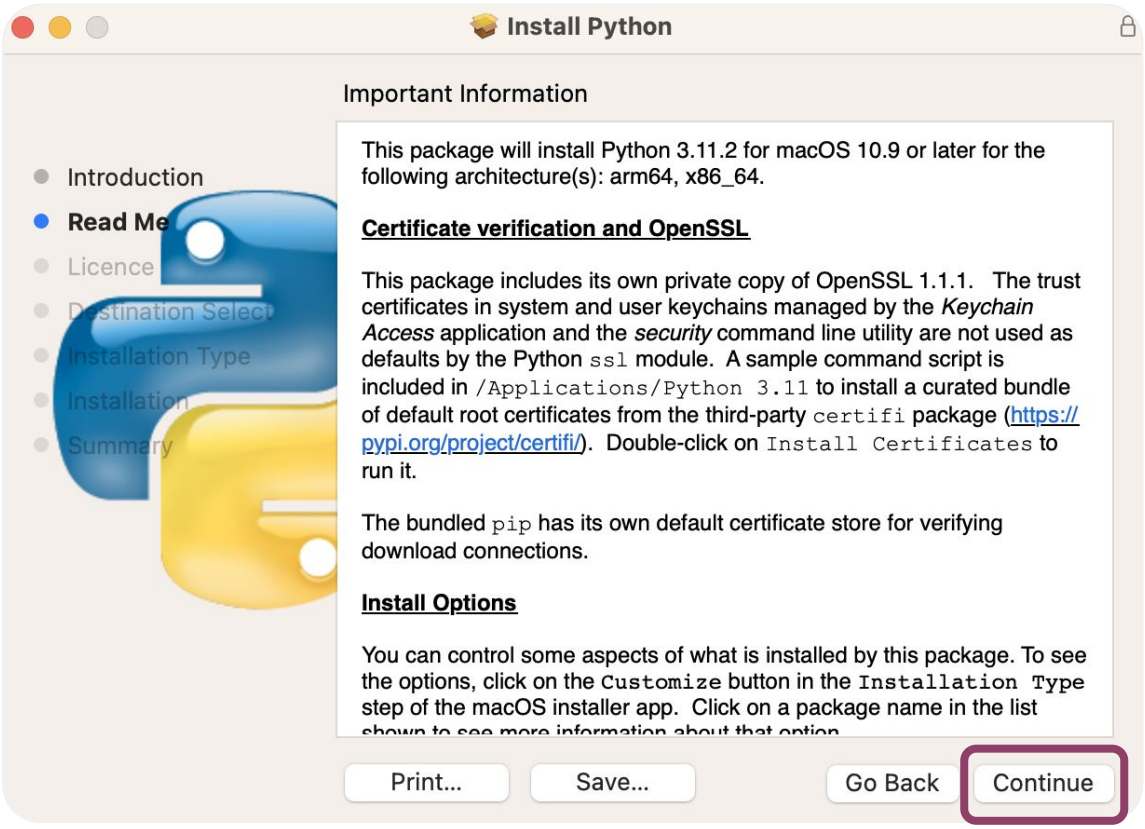
2.14-сурет: Python орнату бағдарламасын жүктеп алғаннан кейін, сол бағдарламаны ашып, орнату просцесін бастаймыз

3. Орнату бағдарламасын ашқаннан кейін “Continue” батырмасын бірнеше рет басу керек (2.15, 2.16, 2.17, 2.18-суреттер).

4. Одан кейін “Agree” және “Install” батырмасын басу керек (2.19, 2.20 суреттер).

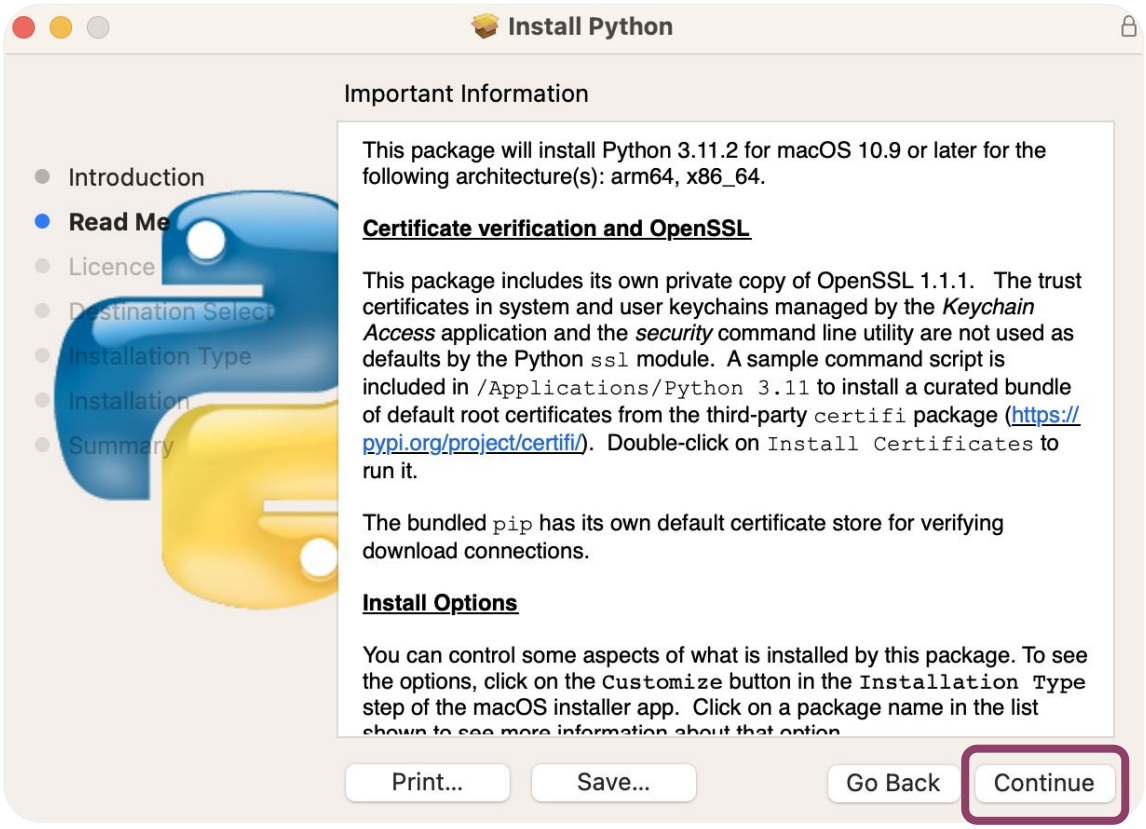
5. Кейін өзіңіздің құпия сөзіңізді жазып, содан соң “Install Software” батырмасын басу керек (2.21 сурет).

2.15-сурет: Орнату бағдарламасын ашқаннан кейін “Continue” батырмасын басу керек

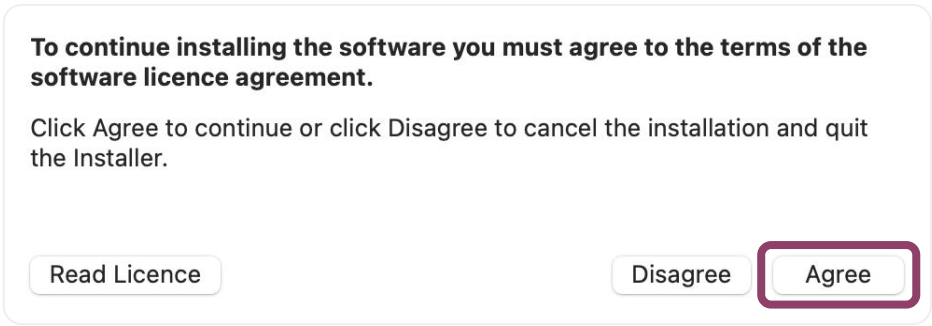


2.16-сурет: Python орнату бағдарламасында “Continue” батырмасын басу

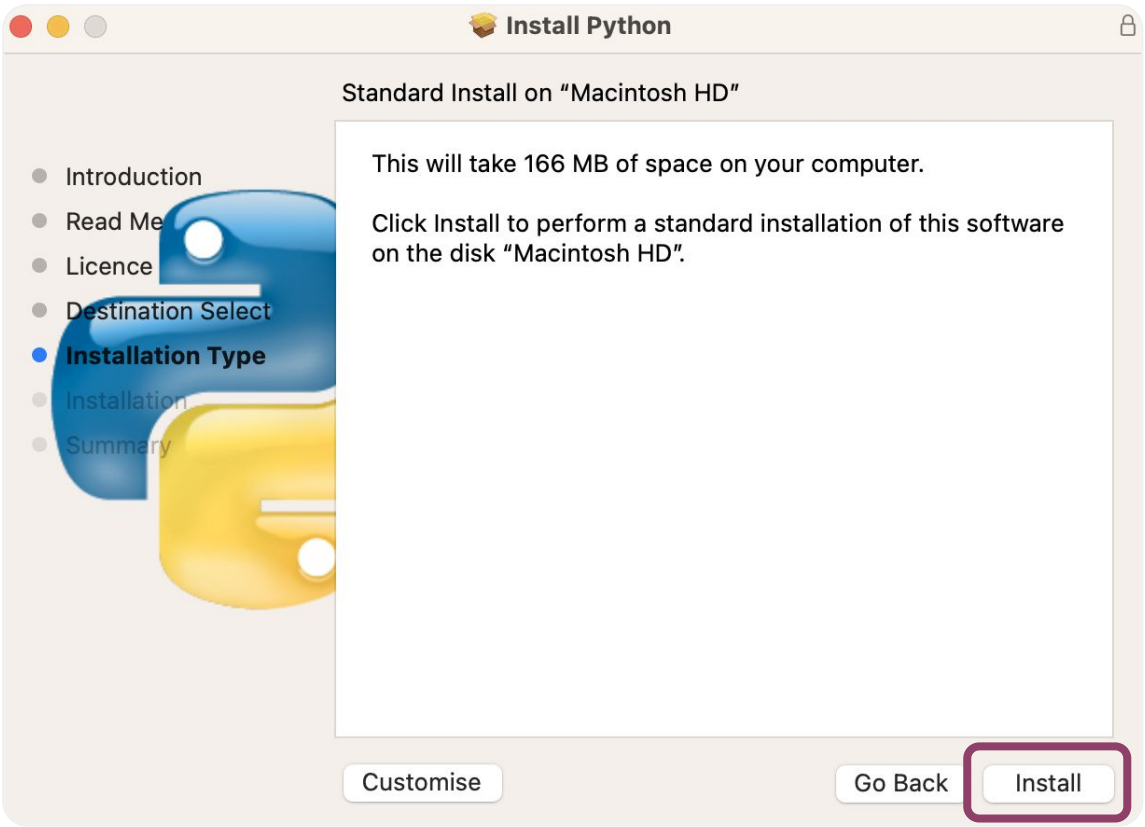
2.17-сурет: Python орнату бағдарламасында “Continue” батырмасын басу



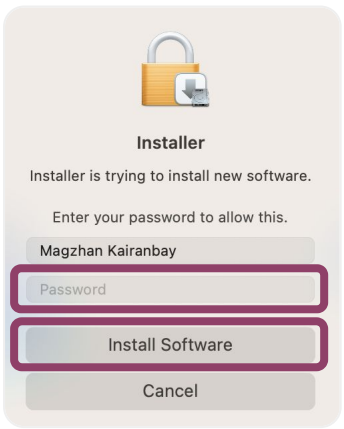
2.18-сурет: Python орнату бағдарламасында “Continue” батырмасын басу



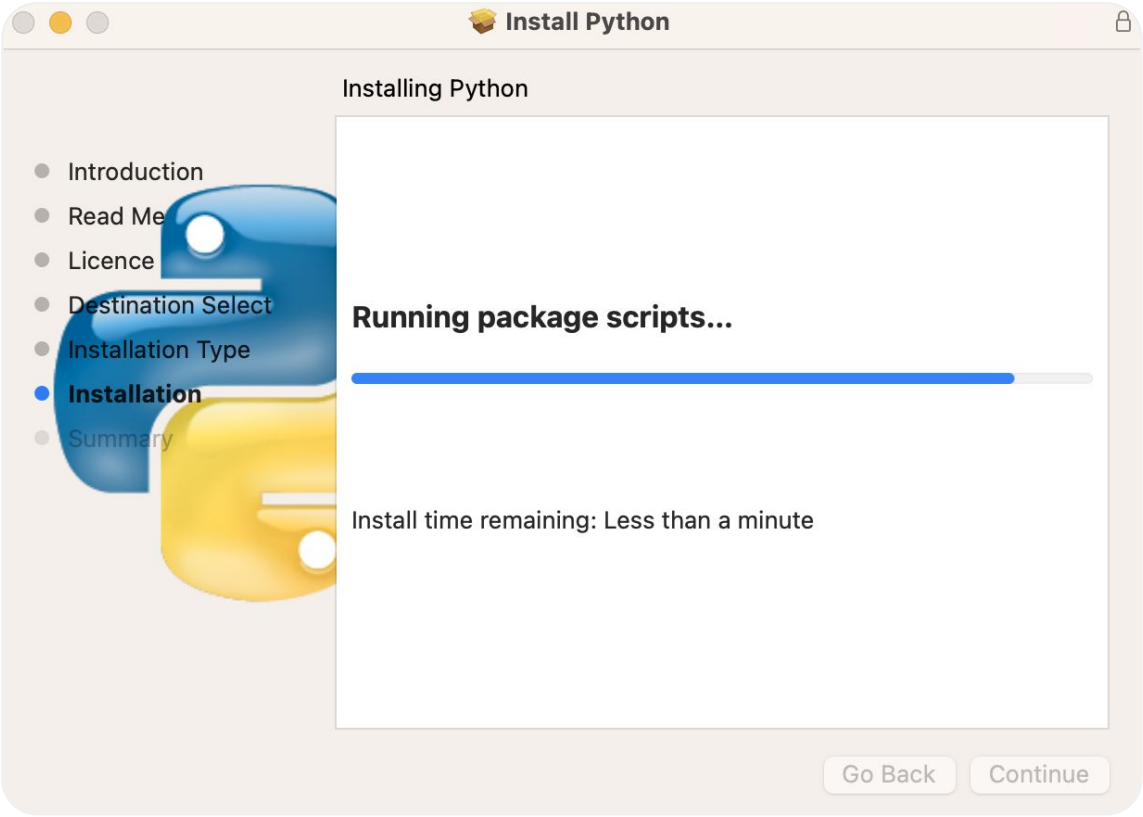
2.19-сурет: Python орнату бағдарламасында “Agree” батырмасын басу



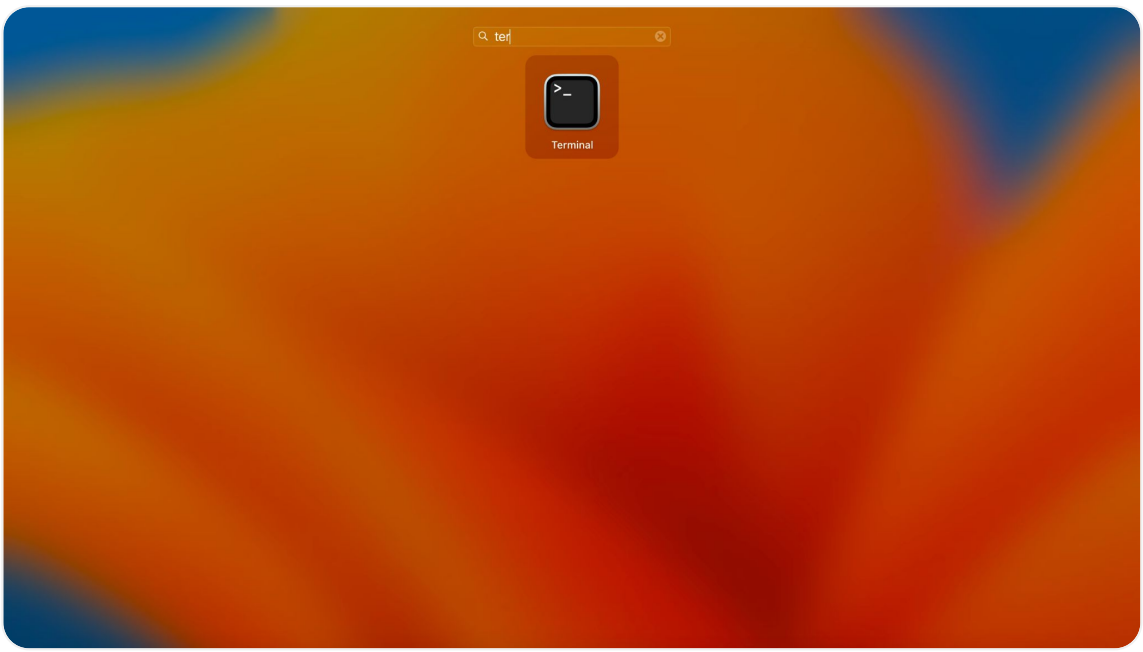
2.20-сурет: Python орнату бағдарламасында “Install” батырмасын басу



2.21-сурет: Құпия сөзді жазып, “Install Software” батырмасын басу



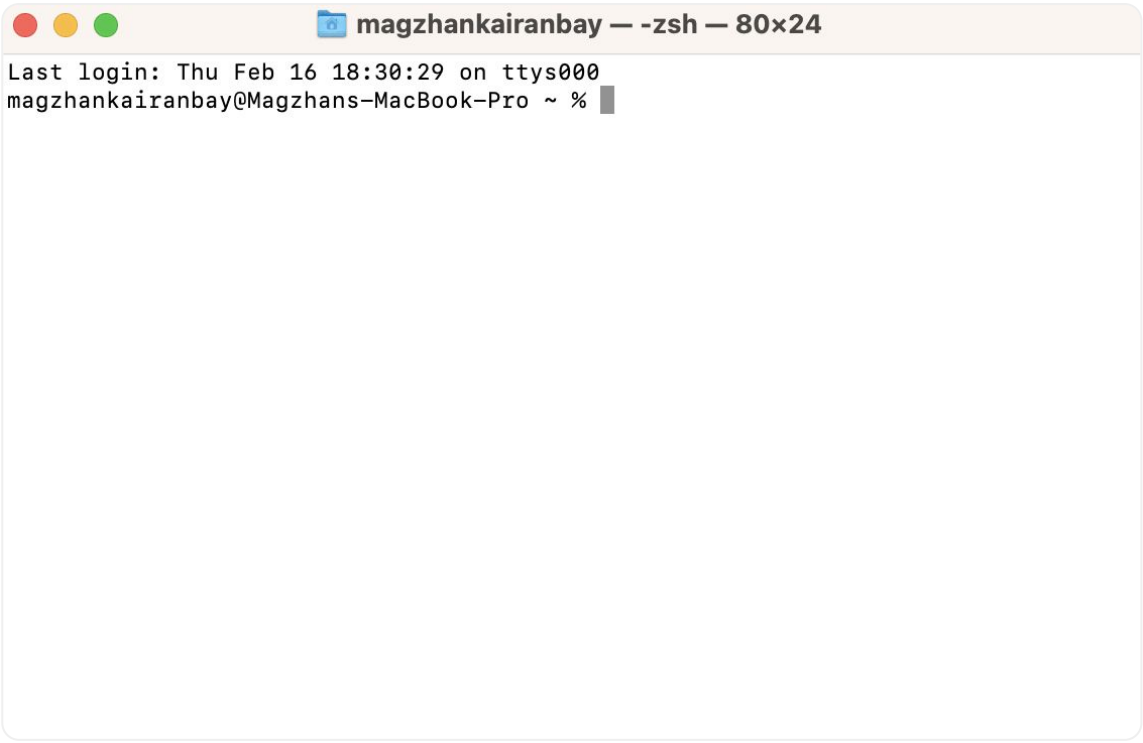
2.22-сурет: Python орнату бағдарламасында орнату процесінің жүруі



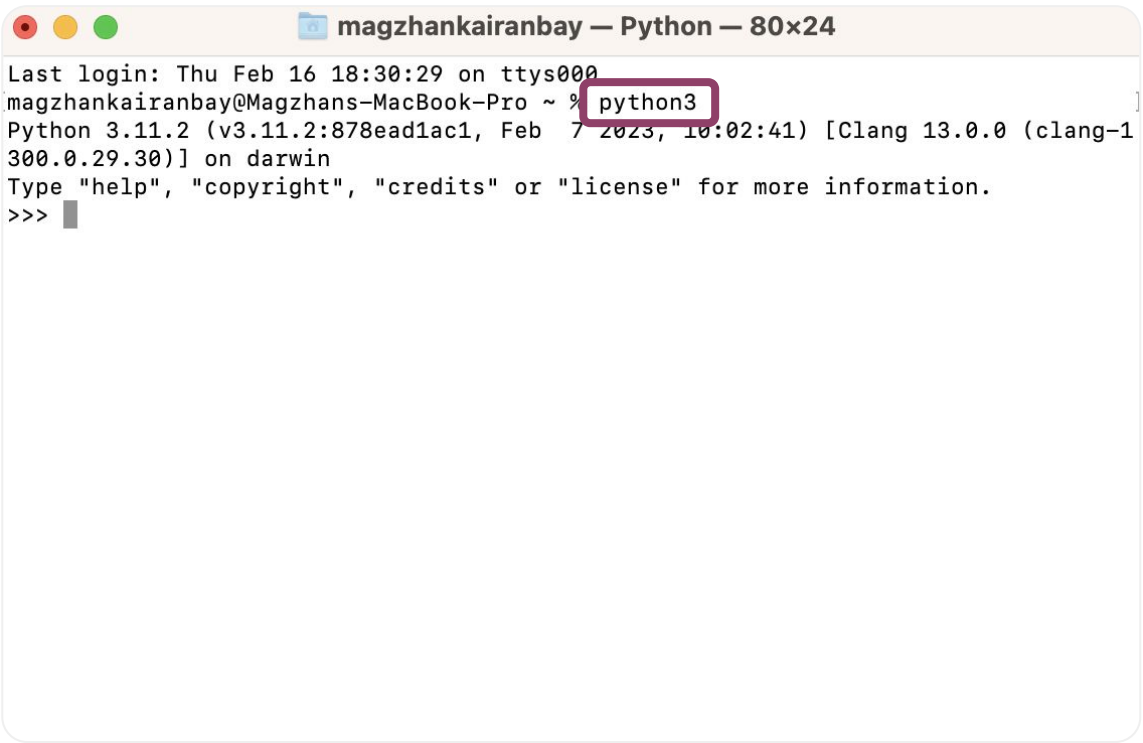
2.23-сурет: Терминал бағдарламасын ашу

6. Кейін “Terminal” бағдарламасын ашу керек (2.23-сурет). Бұл бағдарламаны ашқаннан кейін төмендегідей терезе ашылады (2.24-сурет).

7. “Terminal” терезесі ашылғаннан кейін python3 сөзін жазып, Python тілінде бағдарлама жазуға кіріссек болады (2.25-сурет).

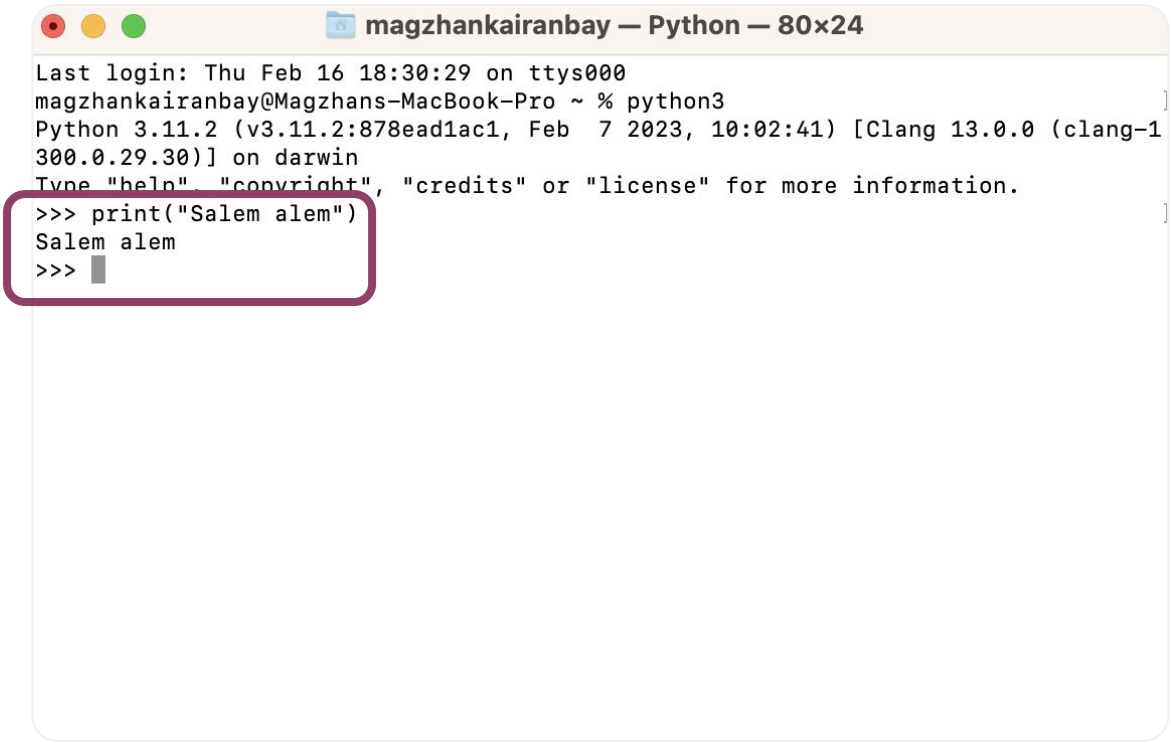


2.24-сурет: Terminal терезесін ашу



2.25-сурет: Терминал терезесінде python3 сөзін жазу арқылы Python кодын жазуға дайындық

8. print(“Salem alem”) кодын жазу арқылы экранға “Salem alem” мәтіні шығатын қарапайым мәтінді жазып шықтық (2.26-сурет).



2.26-сурет: Экранға “Salem alem” мәтіні шығатын қарапайым бағдарлама

Осылайша Mac және Windows операциялық жүйелеріне Python бағдарламалау тілін орнатып, код жазуды бастасақ болады.

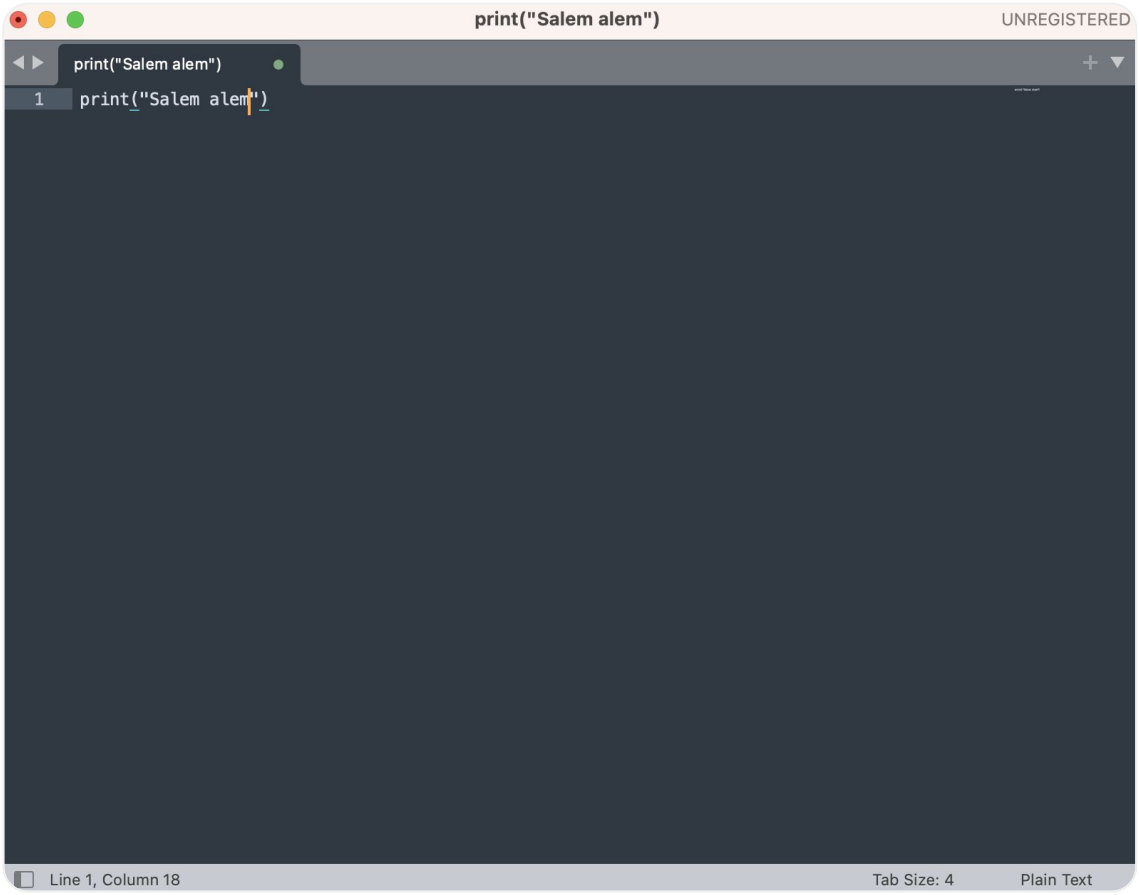
3

Синтаксис

3.1. Python бағдарламасын жазу

Алдыңғы бөлімде экранға қарапайым мәтін шығаратын бағдарлама жазуды үйренген едік. Енді сол кодты арнайы Python файлында жазып, кейін терминал немесе консоль арқылы жүргізуді қарастырамыз. Бұл бөлімде біз Mac операциялық жүйесі арқылы түсіндіреміз. Біздің жасайтын қадамдарымыз Windows операциялық жүйесінде де тура солай болады. Тек бір өзгешелік — Windows операциялық жүйесінде консоль деп атасақ, Mac операциялық жүйесінде терминал деп атаймыз. Бірақ, консоль мен терминалдың атқаратын қызметтері бірдей.

1. Алдымен кез келген бағдарламаны әзірлеу редакторын ашып (Visual Code, Notepad++, Sublime, Atom және тағы басқа), сондағы `print("Salem alem")` мәтінін жазу керек.

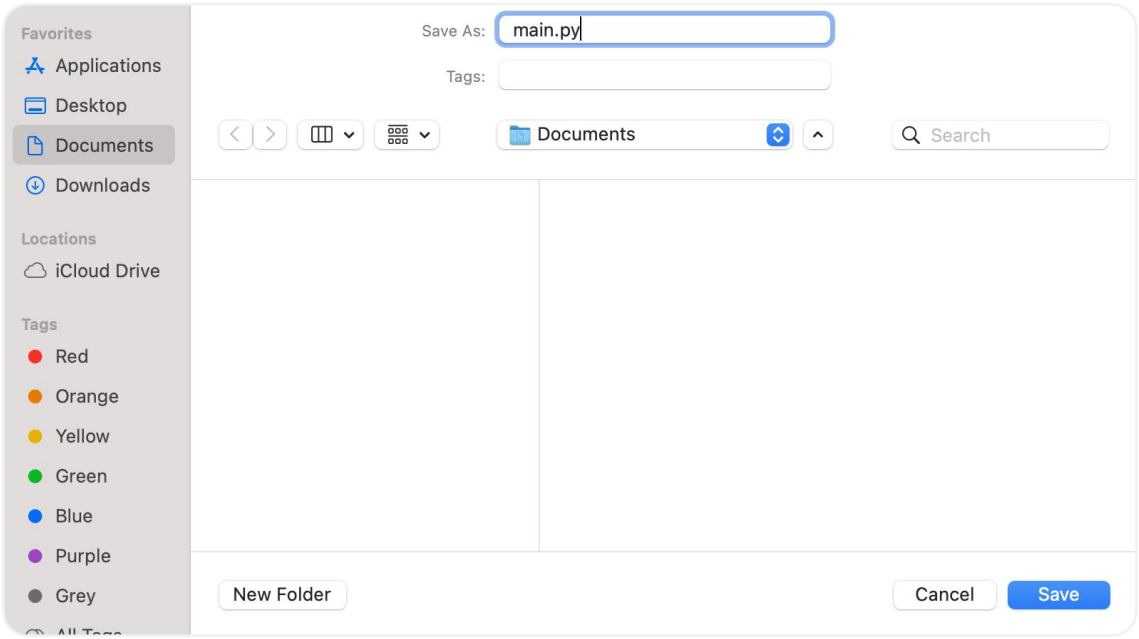


3.1-сурет: Бағдарламаны әзірлеу редакторын ашып (Visual Code, Notepad++, Sublime, Atom және тағы басқа), сонда `print("Salem alem")` мәтінін жазу.

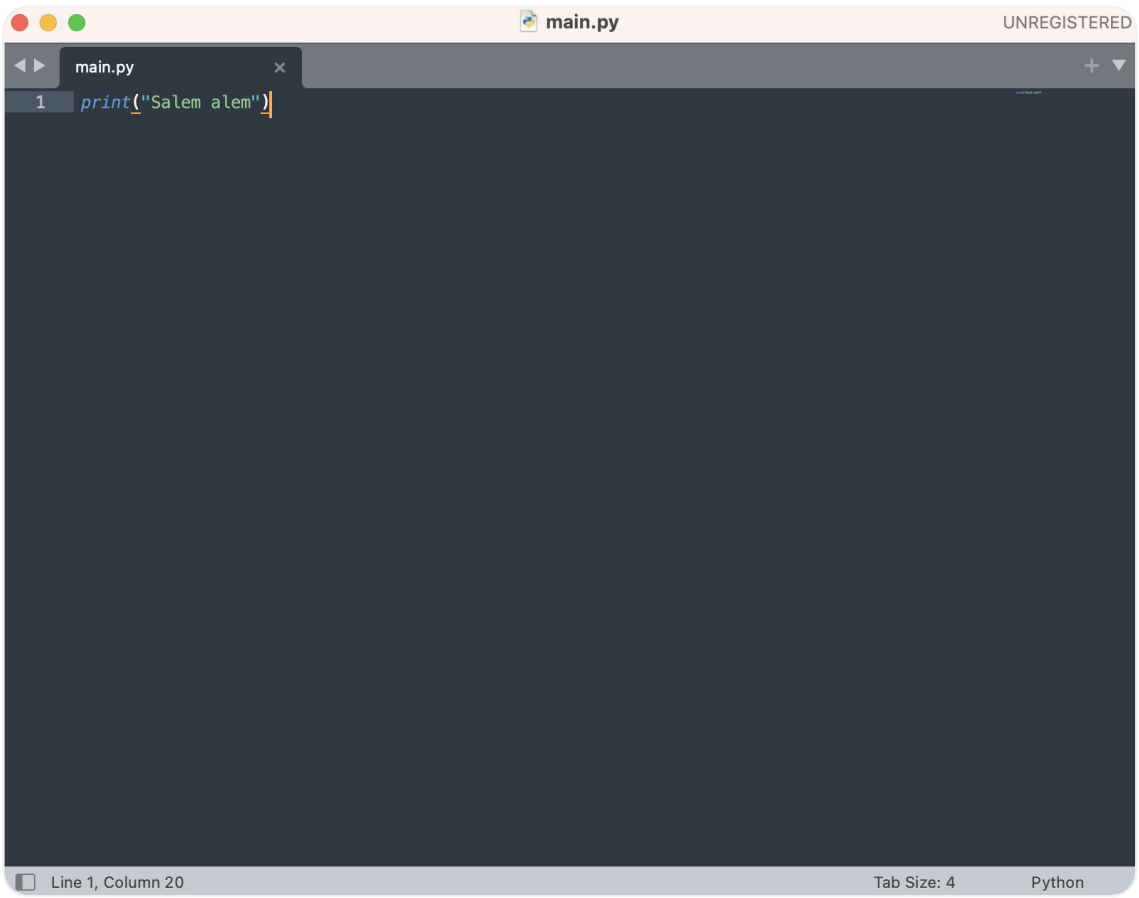
2. Кейін бұл мәтінді файл ретінде сақтауымыз керек. Біздің жағдайда оны main.py файлы ретінде сақтадық. Мұндағы .py — Python файлы екенін білдіреді (3.2-сурет). Файлды сақтау үшін Shift + Command + S батырмаларын бір мезетте бассаңыз болады.

3. Python (.py) файлы ретінде сақтағаннан кейін, файлдағы мәтін түстері өзгереді (3.3-сурет).

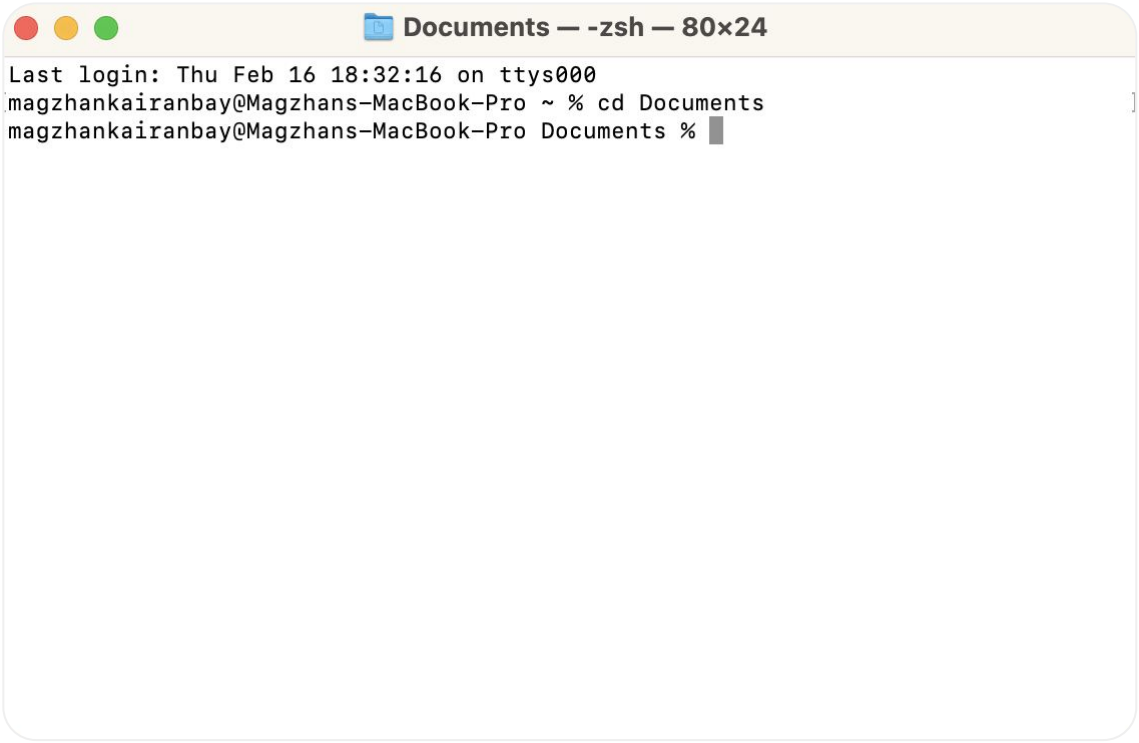
4. Кейін терминал немесе консольді ашып, файл орналасқан жерге дейін бару керек. Ол үшін cd командасын қолданамыз. Яғни cd командасын жазып, одан кейін папка атауын жазу керек (3.4-сурет).



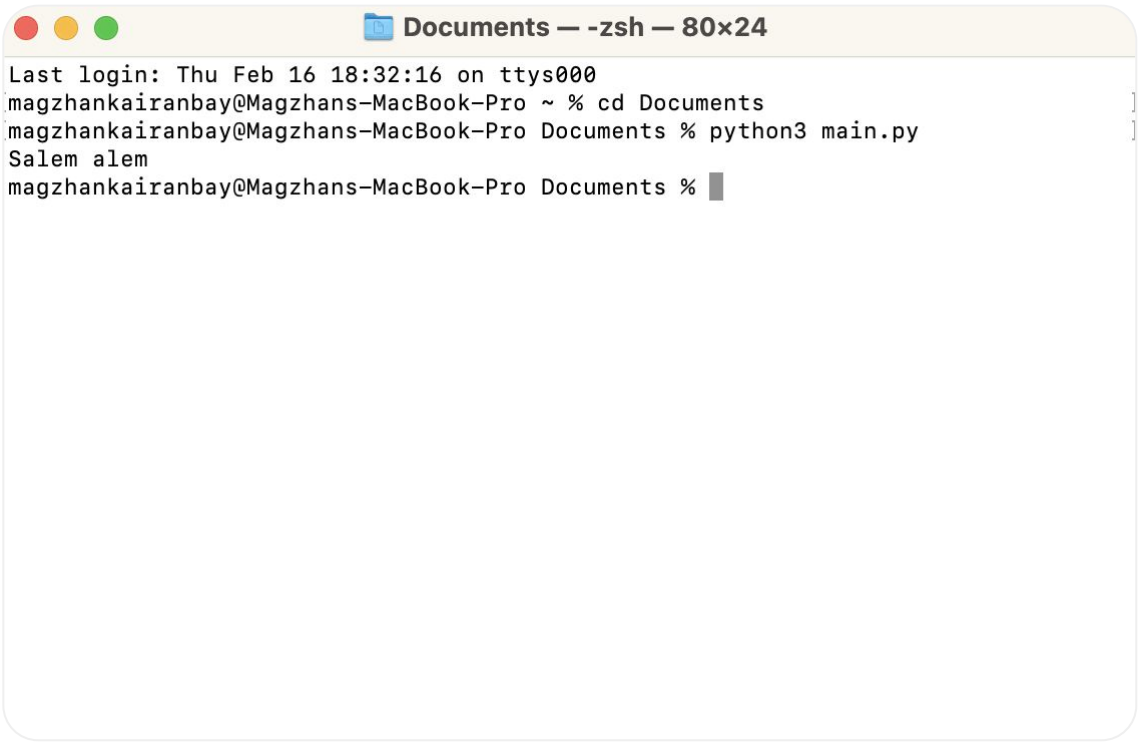
3.2-сурет: Бағдарламаны Python (.py) файлы ретінде сақтау



3.3-сурет: Бағдарламаны Python (.py) файлы ретінде сақтағаннан кейін мәтін түстері өзгереді

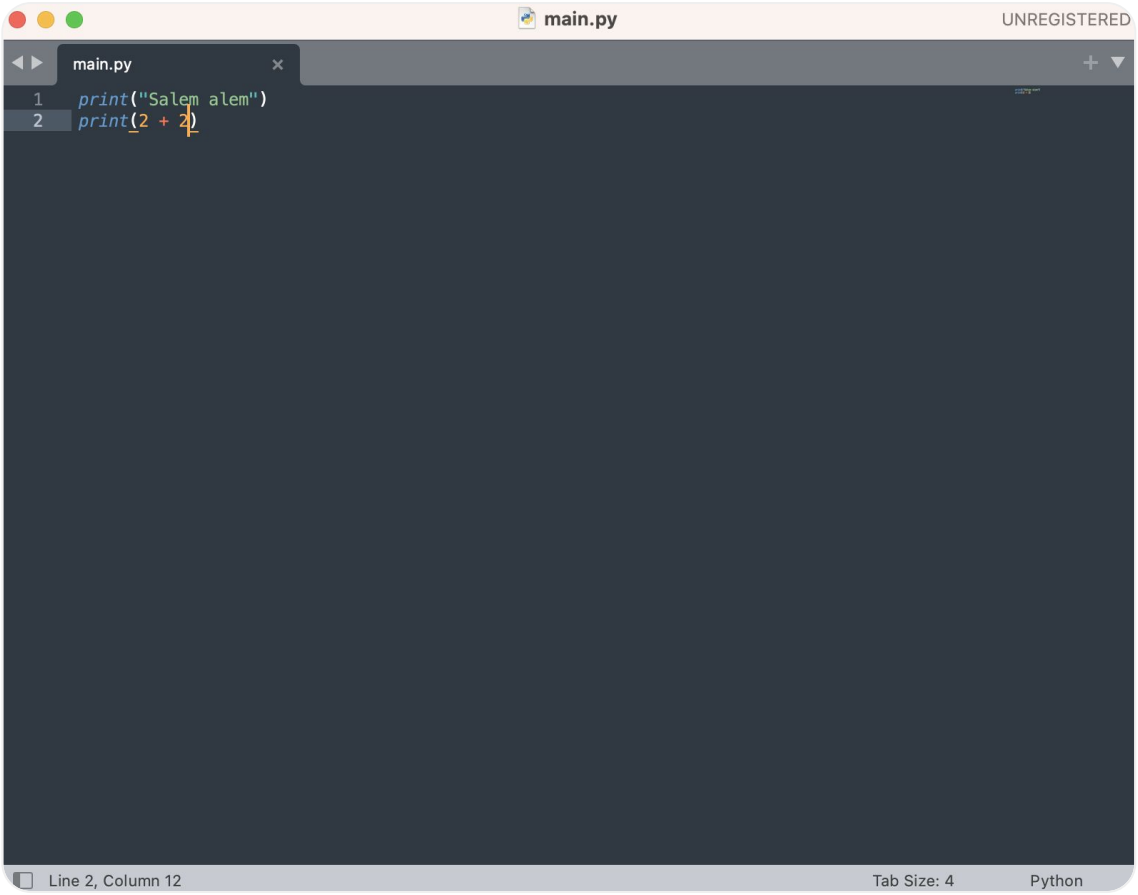


3.4-сурет: Терминал немесе консольді ашып, файлымыз орналасқан жерге дейін cd командасы арқылы жету

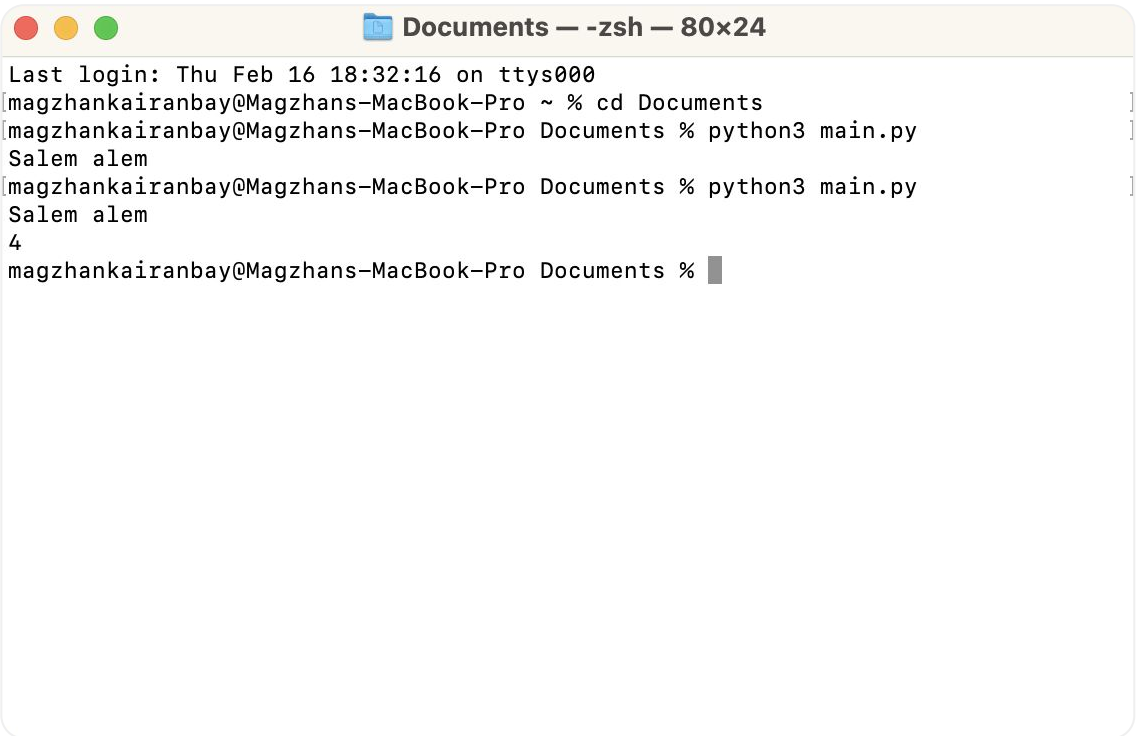


3.5-сурет: Python бағдарламасын python3 немесе python командасы арқылы жүргізу

5. Python бағдарламасына жеткеннен кейін бағдарламамызды python3 немесе python командасы арқылы жүргізу керек. Ол үшін python3 немесе python командасын жазып, содан кейін біздің файл атауын жазған керек, яғни python3 main.py немесе python main.py (3.5-сурет). Бұл командадан кейін экранға **“`Salem alem`”** мәтіні шығады.



3.6-сурет: Python бағдарламасына өзгеріс енгізу



3.7-сурет: Өзгерген бағдарламаның нәтижесін экранға шығару

6. Енді бұл бағдарламаға келесі түрлендіру тәсілдерін енгізейік. Яғни `print("Salem alem")` мәтінінен кейін `print(2 + 2)` кодын қосамыз. Бұл кодты косқаннан кейін, бағдарламаны қайта жүгіртейік (3.6, 3.7-суреттер). Ол үшін `python3 main.py` командасын қайта жазамыз. Бұл командадан кейін экраннан `Salem alem` және `4` мәтіндерін көреміз.

3.2. Python тіліндегі шегініс (Tab)

Шегініс (Tab немесе табуляция) — код жолының басындағы бос орындарды атайды. Басқа бағдарламалау тілдерінде кодтағы шегініс әдемі код жазу үшін қолданылса, ал Python бағдарламалау тілінде шегініс өте маңызды және шегіністің дұрыс қолданылуы бағдарламаның дұрыс жұмыс істеуіне тікелей әсер етеді. Python тілінде код блогтарын жазу үшін шегіністер пайдаланады. Мысалы, төмендегі код бөліктерінде шегіністердің дұрыс және бұрыс қолданылуы көрсетілген.

Шегіністің дұрыс қолданылуы

```
if 5 > 2:
    print("Salem alem")
```

Шегіністің дұрыс мес қолданылу мысалдары

```
if 5 > 2:
print("Five is greater than two!")

if 5 > 2:
 print("Five is greater than two!")

if 5 > 2:
       print("Five is greater than two!")
```

Бір код блогында бірдей шегіністерді пайдалану керек, әйтпесе Python коды қате көрсететін болады. Төмендегі мысалда қате код блогы көрсетілген, себебі, бір код блогында түрлі шегініс саны қолданылған.

```
if 5 > 2:
    print("Salem alem")
        print("Salem alem")
```

4

Пікір

Бағдарлама жазу барысында кейбір жерлерінде бағдарламаның түсінікті болуы үшін пікір жазуға тура келеді. Сонымен бірге бағдарламаның кейбір бөліктерін толығымен баяндау кезінде де пікірлер жазуға тура келеді. Бағдарламаны топпен бірге жазу барысында, сіздің жазған бағдарламаңызды кейін басқа біреулер жалғастырып жазуы да мүмкін. Сол кезде дұрыс құжатталған және түсініксіз жерлері пікір арқылы түсіндірілген код жоғарғы деңгейдегі код болып табылады. Сол себепті код жазу барысында кодыңызға пікір қосып жазу жақсы үрдіс болып есептеледі. Бұл бөлімде Python тіліндегі пікірлермен танысатын боламыз.

4.1. Бір жолды пікір

Python тілінде пікірлерді бір немесе бірнеше жолға жазуға болады. Бір жолдан тұратын пікірді жазу үшін **#** символы қолданылады. Яғни **#** символынан кейін жазылған мәтін код емес, бірақ ол қарапайым мәтін болады. Бұл мәтінге өзіңіз қалаған сөздеріңізбен кодқа түсіндірме жазуыңызға болады. Python тілі бұл пікірді код ретінде қарастырмайды.

Мысал 4.1:

```
a = 2 # a саны 2-ге тең болса
b = 3 # b саны 3-ке тең болса
```

4.2. Бірнеше жолды пікір

Бірнеше жолдан тұратын пірікді жазу үшін бір жолды пікірді бірнеше жолға бөліп жазуға болады. Мысалы:

Мысал 4.2:

```
# a саны 2-ге тең болса
# b саны 3-ке тең болса
a = 2
b = 3
```

Бір жолды піріклер бірнеше жолға бөліп жазылған. Сонымен бірге бірнеше жолды пікірлерді төмендегідей етіп жазса да болады. Яғни пікірді бастарда **"""** символдарын және аяқтарда **"""** символдарын жазамыз. Пікірді бұлай жазу жолын түсіну үшін жоғардағы мысалды былайша түрлендіреміз.

Мысал 4.3:

```
"""
a саны 2-ге тең болса
b саны 3-ке тең болса
"""
a = 2
b = 3
```

5

Айнымалы

5.1. Айнымалы

Айнымалылар деректерді жадта сақтап, олармен жұмыс істеуге мүмкіндік береді. Айнымалыларды жариялағанда, алдымен оның атын жазып, кейін оны қандай да бір мәнге теңестіреміз.

Мысал 5.1:

```
a = 2
b = “Salem”
print(a)
print(b)
```

Экранға шығатын мән:

```
2
Salem
```

Python тілінде айнымалыларды жариялағанда, олардың типін көрсетпейміз. Жоғарыдағы мысал үшін `a` айнымалысы сандық типте болса, `b` айнымалысы мәтіндік типте болады. Бірақ әр айнымалы үшін олардың типін көрсетпей, тек олардың атын және сәйкесінше теңестірілген мәнін көрсетеміз.

5.2. Айнымалыны басқа типке өзгерту (casting)

Айнымалыларды бір типтен екінші бір типке өзгерте аламыз. Бағдарламалауда бұл әдіс — casting деп аталады. Casting-тің жұмыс істеу принципін көру-білу үшін төмендегі мысалды қарастырайық. Мәселен, 3 санын үш түрлі типке өзгерту керек делік. Бірінші тип — мәтіндік тип, екінші тип — бүтін сандық, ал үшінші тип — нақты сандық тип болсын. Яғни бірінші жағдайда біз мәтіндік дерекпен, екінші жағдайда бүтін сандық дерекпен, ал үшінші жағдайда нақты сандық дерекпен жұмыс істейміз.

Мысал 5.2:

```
x = str(3)    # x — '3'-ке тең, яғни x — мәтін
y = int(3)    # y — 3-ке тең, яғни y — бүтін сан
z = float(3)  # z — 3.0-ке тең, яғни z — нақты сан
```

5.3. Мәтіндік айнымалы

Мәтіндік айнымалыларды жариялағанда мынадай тырнақшалар қолданылады: **“”** немесе **‘’**. Яғни айнымалының мәні осы тырнақшалардың ішінде орналасуы керек.

Мысал 5.3:

```
name = “Magzhan”
name = ‘Magzhan’
```

5.4. Айнымалылардың аталуы

Айнымалыларға қандай да бір атау бергенде, оларға қысқаша атаулар (`x, y, z` т.б.) немесе сол айнымалыны сипаттайтын толық атау беруге болады, мысалы: `first_name` (адам аты),

`last_name` (жөні) және т.б. Айнымалыға атау бергенде төмендегі заңдылықтарды ұстану керек:

- Айнымалының атауы әріппен немесе астын сызу таңбасымен (_) басталуы керек.
- Айнымалының атауы саннан басталуы керек.
- Айнымалының атауы тек әріптік-сандық таңбаларды және астын сызу (`A-z`, `0-9` және _) символдарын қамтуы керек.
- Айнымалының атауларында қарапайым әріп пен бас әріптердің өзгешелігін ескерген жөн, мысалы: `age`, `Age` және `AGE` үш түрлі айнымалы болып табылады.

Төмендегі мысалда айнымалыға дұрыс атау беру жағдайлары көрсетілген.

Мысал 5.4:

```
myvar = "Magzhan"
my_var = "Magzhan"
_my_var = "Magzhan"
myVar = "Magzhan"
MYVAR = "Magzhan"
myvar2 = "Magzhan"
```

Ал төмендегі мысалда, керісінше, айнымалыларға бұрыс атау беру жағдайлары көрсетілген.

Мысал 5.5:

```
2myvar = "Magzhan"
my-var = "Magzhan"
my var = "Magzhan"
```

Бірнеше сөзден тұратын айнымалыларды оқу қиынға түсуі мүмкін, мысалы: `firstname, lastname, cityoforigin` және т.б. Бұл жағдайда төмендегідей стильдер қолданылады.

4.1-сурет: Бірнеше сөзден тұратын айнымалыларды атау жолдары

5.5. Айнымалыларды белгілі бір мәнге теңеу

Бірнеше айнымалыны бірнеше мәнге теңеуге болады. Мысалы, бізде `x, y, z` айнымалылары беріліп, оларды `"Orange", "Banana", "Cherry"` мәндеріне теңеу керек болсын делік. Ондай жағдайда бұл айнымалыларды төмендегідей жолмен жариялаймыз:

```
x = "Orange"
y = "Banana"
z = "Cherry"
```

Немесе үш жолға үш айнымалыны жариялап, сонан соң мәнге теңестіргеннен гөрі олардың бәрін бір жолға былайша жарияласақ болады:

```
x, y, z = "Orange", "Banana", "Cherry"
```

Бұлайша жариялау жолы алдыңғы жариялау жолымен салыстырғанда қысқа болады.

Енді, бірнеше айнымалы бір мәнге тең болсын дейік. Мысалы:

```
x = “Orange”
y = “Orange”
z = “Orange”
```

Мұндай жағдайда үш айнымалыны үш жолда емес, бір жолға келесідей былайша аламыз:

```
x = y = z = "Orange"
```

5.6. Айнымалыларды экранға шығару

Айнымалыларды экранға шығарып көрсету үшін `print` функциясы қолданылады. Бұл функцияны жазғаннан кейін жақшаның ішіне керекті айнымалыны жазу керек.

Мысал 5.6:

```
name = “Magzhan”
print(name)
```

Экранға шығатын мән:

```
Magzhan
```

Экранға бірнеше айнымалыны шығару керек болса, онда оларды үтір символы арқылы бөліп жазуға болады.

Мысал 5.7:

```
firstName = “Magzhan”
lastName = “Magzhan”
print(firstName, lastName)
```

Экранға шығатын мән:

```
Magzhan
Kairanbay
```

6

Деректер типі

6.1. Деректер типі

Python тілінде деректердің типі төмендегідей болып келеді.

Деректер типінің атауы	Деректер типінің түрлері
Мәтіндік тип	`str`
Сандық тип	`int, float, complex`
Тізбектік тип	`list, tuple, range`
Сәйкестендіру типі	`dict`
Сэттік тип	`set, frozenset`
Булиндік тип	`bool`
Бинарлы тип	`bytes, bytearray, memoryview`
None типі	`NoneType`

6.1-кесте: Деректер типінің атауы мен түрлері

Бұл типтердің көпшілігінің атауы мен түрлері сіздерге беймәлім болғанымен, олардың көпшілігін осы оқулықтың кейінгі бөлімдерінде өтетін боламыз.

Деректің типін анықтау үшін type функциясы қолданылады. Ол үшін **type** кілтті сөзін және осы сөзден кейін жақшаның ішіне айнымалыны жазу керек. Мысалы:

Мысал 6.1:

```
x = 5
print(type(x))
```

Экранға шығатын мән:

```
<class 'int'>
```

Бұдан түсінетініміз, **x** айнымалысының типі **int**, яғни бүтін сан (integer).

Жиі кездесетін деректерді және оларды жариялау жолдары төмендегі кестеде көрсетілген (6.2-кесте).

Деректердің мысалы	Деректердің типі
x = “Hello world”	str
x = 20	int
x = 20.5	float
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	Tup le
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = True	bool
x = None	NoneType

6.2-кесте: Жиі кездесетін деректер және оларды жариялау жолдары

7

Сандар

7.1. Сандардың түрлері

Python тілінде сандардың келесідей үш түрі бар:

- Бүтін сандар
- Нақты сандар
- Комплекс сандар

Бүтін сандар — оң немесе теріс ондық үтірсіз сандарды атайды. Мысалы: -1, 0, 23, 35656222554887711, 23423432.
Нақты сандар — оң немесе теріс ондық үтірі бар сандарды атайды. Мысалы: -1.4, 0, 23.3434, 35656222554887711.3437, 23423432.3431.
Комплекс сандар — күрделі математикалық сандар. Комплекс сандарды бұл оқулықта қарастырмаймыз.

7.2. Сандардың бір түрден екінші түрге өзгерту

Сандарды бір түрден екінші түрге өзгерткенде сандық типтің атауын жазып, кейін жақшаның ішіне сол санды жазу керек. Бүтін санды нақты санға өзгерту мысалы мынадай:

Мысал 7.1:

```
x = 5
new_x = float(x)
print(new_x)
```

Экранға шығатын мән:

```
5.0
```

Ал нақты санды бүтін санға өзгерту мысалы мынадай:

Мысал 7.2:

```
x = 5.5
new_x = int(x)
print(new_x)
```

Экранға шығатын мән:

```
5
```

7.3. Кездейсоқ сандар

Python тілінде кездейсоқ сандарды генерациялау үшін random кітапханасы қолданылады.

`random` кітапханасының ішінде `randrange` функциясын қолдану арқылы біз қандай да бір аралықтағы кездейсоқ сандарды генерациялай аламыз. Бұл тәсілмен генерациялау жолы мынадай:

Мысал 7.2:

```
import random

print(random.randrange(1, 10))
```

Бұл мысалда біз `randrange` функциясымен 1 мен 10 арасындағы санды генерациялай аламыз.

8

Мәтіндік тип

8.1. Мәтіндік тип

Python тілінде мәтіндер бір тырнақшамен немесе қос тырнақшамен қоршалалы. Мысалы «salem» сөзін былайша жазуға болады: **“salem”** немесе **‘salem’**. Мәтіндер әдетте символдар тізбегі ретінде қарастырылады. Яғни **“salem”** сөзі бес символдан тұрады. Бірінші символ — **s**, екінші символ — **a**, үшінші символ — **l**, төртінші символ — **e** және бесінші символ — **m**. Әр символға қол жеткізу үшін мәтіндік айнымалының атын, кейін тік жақшаның ішіне символдың реттік нөмірін жазу керек. Реттік нөмір индекс деп аталады. Бағдарламалауда индекс нөлден басталады. Сол себепті бірінші символ индексі — 0, екінші символ индексі — 1, үшінші символ индексі — 2, төртінші символ индексі — 3 және бесінші символ индексі — 4. Мысалы, **strVal = “salem”** болса, онда үшінші символға қол жеткізу үшін **strVal[2]** деп жазамыз.

8.2. Ішкі мәтін

Ішкі мәтіні мәтіннің қандайда бір бөлігі болып есептеледі. Мысалы, **“salem alem”** мәтіні үшін мына келесі бөліктер ішкі мәтін бола алады: **“sa”, “alem”, “salem ”, “m al”, “m”**. Python тілінде ішкі мәтіндерді шығарып алу жолдары келесідей.

8.2.1. Аралық ішкі мәтін

Егер біз қандай да бір аралықтағы мәтін бөлігін қайтарғымыз келсе онда сол аралықтың индекстерін жазу керек. Бұл аралық индекстерін қос нүкте арқылы бөлу керек. Мысалы, бізде **strVal = "Hello, World!"** мәтіні болсын. Бұл мәтіннің **“llo”** бөлігін қайтарғымыз келеді делік. Бұл жағдайда **strVal[2:5]** деп жазу керек. Бұл мысалда бізге керекті аралықтар 2-ші мен 5-ші индекстердің аралығында орналасқан.

8.2.2. Мәтіннің басынан басталған ішкі мәтін

Егер біз мәтіннің басынан бастап қандай да бір символға дейінгі ішкі мәтінді шығарып алғымыз келсе, онда тік жақшаның ішінде алдымен қос нүктені, кейін ішкі мәтін аяқталатын индекс нөмірін жазу керек. Мысалы, біздегі **strVal = "Hello, World!"** мәтінінің бастапқы бес символдан тұратын ішкі мәтінін қайтару керек болса, онда **strVal[:5]** деп жазу керек. Ал **strVal[:5]** өз кезегінде **“Hello”** деген мәтінге тең.

8.2.3. Мәтіннің соңына дейін жалғасқан ішкі мәтін

Егер бізге қандай да бір символдан бастап мәтіннің соңына дейін жалғасатын ішкі мәтінді қайтару керек болса, онда тік жақшаның ішінде алдымен керекті символ индексін, одан кейін қос нүктені жазамыз. Мысалы, жоғардағы **strVal = "Hello, World!"** мәтінінің екінші символынан бастап мәтін соңына дейін жалғасқан ішкі мәтінді қайтару керек болса, онда **strVal[2:]** деп жазу керек. Ал **strVal[2:]** өз кезегінде **“llo, World!”** деген мәтінге тең.

8.2.4. Негативті индекс

Мәтін символдарын шығарып алу үшін негативті индекстерді жазуға болады. Негативті индекс реттік нөмірді мәтін соңынан есептейді. Мысалы, **strVal = "Hello, World!"** мәтіні үшін **strVal[-5] – ‘o’** әріпін қайтарса, ал **strVal[-2] – ‘d’** әріпін қайтарады. Ал **strVal[-5:-2] – ‘o’** мен **‘d’** аралығындағы ішкі мәтінді қайтарады, яғни **“orl”**. Мұнда соңғы символ **‘d’** қосылмайды.

8.3. Мәтіндерді бір-біріне біріктіру

Мәтіндерді біріктіру **“+”** операторы арқылы жүзеге асады. Мысалы, бізде мынадай екі мәтін болсын дейік: `firstStr = “salem”`, `secondStr = “alem”`. Енді осы екі мәтінді біріктірейік. Ол үшін `+` операторын қолданамыз, яғни `thirdStr = firstStr + “ ” + secondStr`. Бұл біріктіруден кейін `thirdStr` айнымалысы **“salem alem”** мәтініне тең болады. Мұнда ескеретін бір жайт: біз тек ғана мәтіндерді бір-бірімен біріктіре аламыз. Ал мәтін мен сандық деректер біріге алмайды. Мысалы: `“alem” + 6` қате тұжырым болып табылады. Себебі, `6` сандық тип. Бұл санды мәтіндік форматқа ауыстыру үшін `str` функциясын жазу керек, яғни `str(6)`. Сандық типті мәтіндік типке айналдырып, сонан соң оларды біріктіргенде, ешқандай да қате болмайды. Яғни `“salem” + str(6)` коды ешқандай қатені көрсетпейді.

9

Булин (Boolean)

9.1. Булин (Boolean)

Бағдарламалауда жиі кездесетін типтердің бірі — булиндік немесе логикалық тип. Булиндік тип екі мәнді қабылдай алады. Оның біріншісі — жалған (`0` немесе `False`), ал екіншісі — шындық (`1` немесе `True`). Булиндік тип бағдарламалауда жиі қолданылады. Мысалы, мына тұжырымдарды қарастырайық.

```
1.  print(10 > 9)
2.  print(10 == 9)
3.  print(10 < 9)
```

- Бірінші жағдайда 10 саны 9-дан үлкен болғандықтан, бұл тұжырым — `True`, яғни шындық мәнін қайтарады.
- Екінші жағдайда 10 саны 9-ға тең болмағандықтан, 10 == 9 тұжырымы — `False`, яғни жалған мәнін қайтарады.
- Үшінші жағдайда 10 саны 9-дан үлкен болғандықтан, 10 < 9 тұжырымы — `False`, яғни жалған мәнін қайтарады.

Булиндік тип туралы мынаны білулеріңіз керек:

Булиндік тип екі мәнге ие бола алады. Оның біріншісі — жалған (`0` немесе `False`), ал екіншісі — шындық (`1` немесе `True`).

10

Оператор

10.1. Операторлардың түрлері

Оператор деп — айнымалыға қолданылатын амалды атаймыз. Python тілінде мынадай операторлар бар:

- Арифметикалық операторлар
- Меншіктеу операторлары
- Салыстыру операторлары
- Логикалық операторлар
- Сәйкестендіру операторлары
- Мүшелік операторлар
- Биттік операторлар

Бұл операторлардың әр қайсысын төменде қарастыратын боламыз.

10.2. Арифметикалық операторлар

Арифметикалық операторлар сандарға қолданылатын математикалық операцияларды атайды. Арифметикалық операциялар мынадай:

Оператор	Атауы	Мысал
+	Қосу	x + y
-	Азайту	x / y
*	Көбейту	x * y
/	Бөлу	x / y
%	Қалдық табу	x % y
**	Дәрежелеу	x ** y
//	Бөліп, жуықтау	x // y

10.1-кесте: Арифметикалық операторлар

10.3. Меншіктеу операторлары

Меншіктеу операторлары қандай да бір мәнді қандай да бір айнымалыға меншіктеу процесін атайды. Меншіктеу операцияларының кейбіреуі төмендеге кестеде толық мағынасымен көрсетілген (10.2-кесте).

Оператор	Мысал	Мағынасы
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

10.2-кесте: Меншіктеу операторлары

10.4. Салыстыру операторлары

Салыстыру операторлары қандай да бір екі мәнді салыстыру үшін қолданылады. Салыстыру операторлары мына кестеде көрсетілген (10.3-кесте).

Оператор	Атауы	Мысал
==	Тең	x == y
!=	Тең емес	x != y
>	Үлкен	x > y
<	Кіші	x < y
>=	Үлкен немесе тең	x >= y
<=	Кіші немесе тең	x <= y

10.3-кесте: Салыстыру операторлары

10.5. Логикалық операторлар

Логикалық операторлар шартты түрде берілген тұжырымдарды бір-бірімен біріктіру үшін қолданылады. Логикалық операторлао төмендегі кестеде түсінік беріледі.

Оператор	Анықтамасы	Мысал
`and`	Екі мәннің екеуі де `True` болған жағдайда `True` қайтарады, әйтпесе `False` қайтарады	`x < 5 and  x < 10`
`or`	Екі мәннің біреуі `True` болған жағдайда `True` қайтарады, әйтпесе `False` қайтарады	`x < 5 or x < 4`
`not`	Егер мән `True` болса, `False` қайтарады, ал егер мән `False` болса, онда `True` қайтарады	`not(x < 5 and x < 10)`

10.4-кесте: Логикалық операторлар

10.6. Сәйкестендіру операторлары

Сәйкестендіру операторлары қандай да бір екі айнымалының бірдей екенін анықтайды. Мұндағы бір өзгешелік — айнымалылардың бірдей екені тек бірдей мәнмен шектелмейді, бірақ, сонымен бірге, айнымалылардың бір жадыда сақталуымен анықталады. Сәйкестендіру операторлары төмендегі кестеде көрсетілген.

Оператор	Анықтамасы	Мысал
`is`	Екі объект те бірдей болса `True`, әйтпесе `False` қайтарады	`x is y`
`is not`	Екі обьект екі түрлі болса `False`, әйтпесе `True` қайтарады	`x is not y`

10.5-кесте: Сәйкестендіру операторлары

11

Тізбек

11.1. Тізбектер

Тізбектер бірнеше мәнді бір айнымалыда сақтап тұрады. Мысалы, бізде мынадай айнымалылар бар делік.

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

Бұл айнымалылардың саны — үшеу. Ал егер бізде мың мән болса, онда біз мың айнымалыны жариялауымыз керек пе? Жоқ. Мұндай жағдайда тізбектерді қолдануымызға болады. Тізбектер бірнеше мәнді бір айнымалыда сақтап тұруға мүмкіндік береді. Жоғарыдағы мысал үшін үш түрлі айнымалыны жарияламай, тек бір ғана тізбекті былайша жариялауымызға болады: `cars = ["Ford", "Volvo", "BMW"]`. Тізбектер тік жақшалардың көмегімен жарияланады. Тік жақшаның ішіне тізбек мүшелерін үтір арқылы жазамыз. Тізбектің әр мүшесіне қол жеткізу үшін сол тізбектің атауын кейін тік жақшаның ішінде сол элементтің индексін жазамыз. Индекс нөлден басталады. Мысалы, жоғардағы мысал үшін бірінші элементке қол жеткізу үшін `cars[0]`, екінші элементке қол жеткізу үшін `cars[1]` және үшінші элементке қол жеткізу үшін `cars[2]` деп жазамыз. Яғни жалпы жағдай үшін `n`-ші элементке қол жеткізу үшін `(n - 1)`-ші индексті көрсету қажет.

11.2. Тізбектің ұзындығы

Тізбектің ұзындығын табу үшін `len()` функциясы қолданылады. Жоғардағы мысал үшін `cars` тізбегінің ұзындығын қайтару үшін `len(cars)` деп жазу керек. Мұндағы `len(cars)` үш мәнін қайтарады, себебі тізбекте үш элемент бар.

11.3. Тізбекке элементті қосу және тізбектен элементті өшіру

Тізбекке элементті қосу үшін `append()` функциясы қолданылады. Яғни тізбекке қандай да бір элементті қосу үшін алдымен сол тізбектің атауын, одан кейін нүктені, нүктеден кейін `append` кілтті сөзі, сосын ашылған және жабылған жақша ішіне бізге керекті элементті жазу керек. Жоғардағы мысал үшін **“Honda”** дейтін жаңа элементті қосайық.

Мысал 11.1:

```
cars.append("Honda")
print(cars)
```

Экранға шығатын мән:

```
['Ford', 'Volvo', 'BMW', 'Honda']
```

Тізбектен элементті өшіру үшін `pop()` немесе `remove()` функциялары қолданылады. `pop()` функциясында элементтің индексін беру керек болса, ал `remove()` функциясында элементтің мәнін беру керек болады. Жоғарыдағы мысал үшін элементті `pop()` және `remove()` функциялары арқылы екінші **“Volvo”** элементін өшірейік.

Мысал 11.2:

```
cars.append("Honda")
print(cars)

cars.pop(1)
```

Экранға шығатын мән:

```
['Ford', 'BMW', 'Honda']
```

Мысал 11.3:

```
cars.append("Honda")
print(cars)

cars.remove(“Volvo”)
```

Экранға шығатын мән:

```
['Ford', 'BMW', 'Honda']
```

`pop()` функциясында екінші элементтің индексін берсек (1 саны), ал `remove()` функциясында өшіретін элементтің мәнін, яғни **“Volvo”** деп жаздық.

12

Шартты оператор

Қандай да бір шартты тексеру үшін шартты оператор қолданылады. Шартты операторлардың жиі кездесетіні **if else** операторы. Төменде **if else** операторын жан-жақты түсіндірейік.

12.1. If else

12.1.1. If

if else операторындағы **if** "егер" деген мағынаны білдіреді. Яғни белгілі бір шартты тексеру үшін **if** ("егер") кілтті сөзін қолданамыз. **if** сөзінен кейін логикалық тип **True** немесе **False** мәндеріне тексеріледі. Логикалық тип тексерілгеннен кейін (**True** мәніне тең немесе тең емес екенін анықтағаннан кейін) қос нүкте символы (:) қойылады. Егер логикалық тип **True** мәнін қайтарса, онда қос нүктеден кейінгі амалдар жиынтығы орындалады. Толық жазылу жолы мынадай:

```
if логикалық тип: # мұндағы логикалық тип True немес False мәнін қайтарады
        қандай да бір операция жиынтығы # логикалық тип True-ге тең болған жағдайда ғана орындалады
```

Мысал 12.1:

```
hour = 12
if hour < 12:
        print("Good morning")
```

Экранға шығатын мән:

```
Good morning
```

if-тен кейінгі тұжырым келесі жолдан бастап және tab батырмасын басу арқылы жазылуы керек.

12.1.2. If else

if операторындағы логикалық тип **False** мәніне тең болса, онда қос нүктеден кейінгі амалдар орындалмайды. Егер логикалық тип **False** мәнін қайтарғанда қандай да бір амалдар орындалу үшін **else** тұжырымы қолданылады. **else** ағылшынша "әйтпесе" - деген мағынаны білдіреді. **else** тұжырымынан кейін қос нүкте қоюлуы қажет. **if else** тұжырымының жазылуы мынадай:

```
if логикалық тип: # мұндағы логикалық тип True немес False мәнін қайтарады.
        қандай да бір операция жиынтығы # логикалық тип True-ге тең болған жағдайда ғана орындалады
else:
        қандай да бір операция жиынтығы # логикалық тип False-қа тең болған жағдайда ғана орындалады
```

Мысал 12.2:

```
hour = 15
if hour < 12:
	print(“Good morning”)
else:
	print(“Good afternoon”)
```

Экранға шығатын мән:

```
Good afternoon
```

12.1.3. Elif

Бірақ, егер сағат 20-ға тең болса (**hour = 20**), онда “Good afternoon” (“Қайырлы күн”) мәтінін шығару дұрыс болмайды. Мұндай жағдайда “Good evening” (“Қайырлы кеш”) деген мәтінді шығарып алу керек. Ондай бағдарлама жазып шығару үшін **elif** тұжырымы қолданылады. **elif** тұжырымы **else if** тұжырымының қысқашасын білдіреді. **elif** тұжырымынан кейін логикалық операторды жазып, одан кейін қос нүкте қою қажет. **elif** тұжырымын қолданып жоғарыдағы бағдарламаны жазып шығайық.

Мысал 12.3:

```
hour = 20
if hour < 12:
	print(“Good morning”)
elif hour >= 12 and hour < 18:
	print(“Good afternoon”)
else:
	print(“Good evening”)
```

Экранға шығатын мән:

```
Good evening
```

Жоғарыдағы мысалда, егер сағат 12-ден кіші болса (**if hour < 12**), онда экранға “Good morning” (“Қайырлы таң”) деген мәтін шығарамыз. Ал егер сағат (**hour**) 12-мен 18-дің арасында болса (**elif hour >= 12 and hour < 18**), онда экранға “Good afternoon” (“Қайырлы күн”) деген мәтінді шығарамыз. Егер осы екі жағдай да орындалмаса (**else**), онда экранға “Good evening” (“Қайырлы кеш”) деген мәтінді шығарамыз. Бір бағдарламада бірнеше **elif** тұжырымын жазуға да болады, мысалы:

```
if … :
	…
elif:
	…
elif:
	…
elif:
	…
elif:
	…
```

12.2. Логикалық оператор

12.2.1. Қарапайым логикалық оператор

Қарапайым логикалық операторлар қандай да бір екі мәнді тексеру арқылы жүзеге асады. Бұл қарапайым логикалық операторлар төмендегідей болуы мүмкін:

- Теңдік: `a == b` (`a`-ның мәні `b`-ның мәніне тең болса `True`, әйтпесе `False` қайтарады)
- Теңсіздік: `a != b` (`a`-ның мәні `b`-ның мәніне тең болмаса `True`, әйтпесе `False` қайтарады)
- Кіші: `a < b` (`a`-ның мәні `b`-ның мәнінен кіші болса `True`, әйтпесе `False` қайтарады)
- Кіші немесе тең: `a <= b` (`a`-ның мәні `b`-ның мәнінен кіші немесе тең болса `True`, әйтпесе `False` қайтарады)
- Үлкен: `a > b` (`a`-ның мәні `b`-ның мәнінен үлкен болса `True`, әйтпесе `False` қайтарады)
- Үлкен немесе тең: `a >= b` (`a`-ның мәні `b`-ның мәнінен үлкен немесе тең болса `True`, әйтпесе `False` қайтарады)

Осы логикалық операторларды `if` немесе `elif` тұжырымынан кейін жаза аламыз. Бірақ, қарапайым логикалық операторлардан басқа, күрделі логикалық операторларды да жазуға болады.

12.2.2. Күрделі логикалық оператор

Күрделі логикалық операторлар бірнеше қарапайым логикалық операторлардың жиынтығынан тұрады. Ол қарапайым операторлар `and`, `or` және `not` кілтті сөздері арқылы бірігуі мүмкін. Бұл кілтті сөздер төмендегі логикалық тұжырымдарды білдіреді:

- `a and b` — егер `a` және `b` тұжырымының екеуі де `True`-ға тең болса, онда (`a and b`) да `True`-ға тең, әйтпесе (`a and b`) тұжырымы `False`-қа тең болады. `and` — ағылшынша “және” дегенді білдіреді.
- `a or b` — егер `a` және `b` тұжырымының біреуі `True`-ға тең болса, онда (`a or b`) да `True`-ға тең, әйтпесе (`a or b`) тұжырымы `False`-қа тең болады. `or` — ағылшынша “немесе” дегенді білдіреді.
- `not a` — егер `a`-ның мәні `True`-ға тең болса, онда (`not a`) тұжырымы `False`-қа тең, әйтпесе, егер `a`-ның мәні `False`-қа тең болса, онда (`not a`) тұжырымы `True`-ға тең. `not`- ағылшынша “жоқ, болдырмау” деген мағынада қолданылады.

Мысал 12.4:

```
if A == 1 AND B != 2 AND C >= 3:
      print(“A мәні 1-ге тең, B мәні 2-ге тең емес, C мәні 3-тен үлкен немесе тең”)
```

12.3. Бірінің ішіне салынған шартты оператор

`if` тұжырымы бірінің ішіне кіріп жазыла алады. Бірінің ішіне салынған `if` тұжырымын түсіндіру үшін келесі мысалға көз жүгіртейік. Бізге `a` саны берілсін. Егер бұл сан 10-нан үлкен болса, экранға “10-нан үлкен” деген мәтінді шығару керек. Ал егер `a` саны 20-дан да үлкен болса, онда экранға қосымша “20-дан да үлкен” деген мәтін шығару керек. Ал егер `a` саны 10-нан кіші болса, онда экранға “10-нан кіші” деген мәтін шығару керек. Енді осы мысалды Python тілі арқылы жазып шығайық.

Мысал 12.5:

```
a = 25
if a > 10:
       print(“10-нан үлкен”)
       if > 20:
              print(“20-дан да үлкен”)
else:
       print(“10-нан кіші”)
```

Экранға шығатын мән:

```
10-нан үлкен
20-дан да үлкен
```

Осылайша **if** тұжырымын бірінің ішіне бірнеше рет салып жазуға болады.

Мысал 12.6:

```
if логикалық амал:
       if логикалық амал:
              if логикалық амал:
                     if логикалық амал:
                            ...
```

12.4. Қысқартылған if тұжырымы

if тұжырымын қысқартып былайша жазуға да болады:

```
if логикалық тұжырым: орындалатын амал
```

Яғни орындалатын амал келесі жолда емес, бірақ сол жолда қос нүктеден кейін жазылады. Сонымен қатар **if else** тұжырымын қысқартып былайша да жазуға болады:

```
орындалатын амал : if логикалық тұжырым else орындалатын басқа амал
```

13

For циклі

Адамның компьютерден айырмашылығы — адам қандай да бір іс-әрекетті бұлжытпай және тоқтамастан орындай алмайды. Ал, компьютер сол іс-әрекетті тоқтамастан, ұзақ уақыт бойы орындай алатын қабілетке ие. Осындай іс-әрекетті тоқтамастан жасау үшін цикл түсінігі қолданылады. Python бағдарламалау тілінде циклдің екі түрі бар. Бірі — `for` циклі, екіншісі — `while` циклі. Алдымен `for` циклін түсіндіруден бастайық.

13.1. For циклі

Бізге, мысалы мынадай тізбек берілсін. Бұл тізбекте барлық элементтерді экранға шығару керек делік. Қарапайым шығару жолы төмендегідей болады:

Мысал 13.1:

```
fruits = [“apple”, “banana”, “cherry”, “blue berry”]
print(fruits[0])
print(fruits[1])
print(fruits[2])
print(fruits[3])
```

Экранға шығатын мән:

```
apple
banana
cherry
blue berry
```

Бұл бағдарламаны жазып шығу үшін 5 жол кетті. Ал тізбекте 10000 элемент болса, онда 10000 жолдан тұратын код жазу керек пе? Жоқ. Ондай жағдайда `for` циклін қолдансақ болады. `for` циклі арқылы код былайша жазылады.

Мысал 13.1 (жалғасы):

```
fruits = [“apple”, “banana”, “cherry”, “blue berry”]
for x in fruits:
       print(x)
```

Экранға шығатын мән:

```
apple
banana
cherry
blue berry
```

`for` тұжырымын қолданғанда алдымен `for` кілтті сөзін жазып, содан кейін кез келген айнымалыны жариялаймыз. Бұдан соң `in` кілтті сөзін жазып, одан кейін біздегі бар тізбекті жазамыз. `for` тұжырымынан соң қайталанатын операциялар жиынтығын жазу керек. Жалпы жазылу жолы төмендегідей:

```
for қандай да бір айнымалы in тізбек:
       операция жиынтығы
```

Python тіліндегі басқа тұжырымдар сияқты **for** циклінен кейін қос нүкте қою керек.

13.2. Break тұжырымы

Циклдің қандай да бір қадамынан шығу үшін **break** тұжырымы қолданылады. **break** тұжырымын түсіну үшін мына мысалға назар аударайық:

Мысал 13.2:

```
fruits = [“apple”, “banana”, “cherry”, “blue berry”]
for x in fruits:
	if x == “banana”:
		break
	print(x)
```

Экранға шығатын мән:

```
apple
```

Алдымен **for** циклі арқылы тізбектің әр элементі бойынша жүгіріп шығамыз. Әр қадамда сол элемент **“banana”**-ға тең немесе тең емес екеінін **if** тұжырымы арқылы тексереміз. Егер тең болған жағдайда **break** кілтті сөзі арқылы **for** циклінен шығамыз. Циклдан шығу дегеніміз — ары қарай цикл қадамдарын орындамауды білдіреді. Сол себепті **apple**-дан кейін **banana, cherry, blue berry** атты элементтер экранға шықпады.

13.3. Continue тұжырымы

Циклдің қандай да бір қадамын орындамау үшін **continue** тұжырымы қолданылады. **continue** тұжырымын түсіндіру үшін мына мысалды қарастырайық:

Мысал 13.2:

```
fruits = [“apple”, “banana”, “cherry”, “blue berry”]
for x in fruits:
	if x == “banana”:
		continue
	print(x)
```

Экранға шығатын мән:

```
apple
cherry
blue berry
```

Мұнда әр қадамымызда жүгіріліп жатқан элементті **banana** мәнімен салыстырамыз. Егер ол элемент **banana**-ға тең болса, онда ары қарай орындалуға тиісті амалдарды орындамай, **for** циклінің келесі қадамына көшеміз. Сол себепті жоғарыдағы мысалда **banana** мәнін экранға шығару амалын секіріп, келесі қадамдарға көшеміз де, сол элементтерді экранға шығардық.

13.4. Range тұжырымы

for циклінде қадамдарды өзгерту үшін **range** тұжырымы қолданылады. **range** тұжырымы мынадай параметрлерді қабылдай алуы мүмкін. Біріншісі — қадамның бастапқы мәні, екіншісі — соңғы мәні және үшіншісі — қадамның мәні. Бұл мәндерді мына мысалмен түсіндірейік.

Мысал 13.4:

```
for x in range(2, 10, 2):
	print(x)
```

Экранға шығатын мән:

```
2
4
6
8
10
```

Бұл бағдарламада экранға 2-ден 10-ға дейінгі тек жұп сандар шығады. Себебі, бастапқы параметрді 2-ге, екінші параметрді 10-ға және үшінші параметрді 2-ге теңедік. Бастапқы параметр — бастау мәнін, екінші параметр — аяқталу мәнін және үшінші параметр қадам мәнін білдіреді. Егер қадам мәнін жазбасақ, онда ол 1-ге теңеледі. Ал егер бастапқы мәнді жазбасақ, онда ол 0-ге теңеледі. Мысалы:

Мысал 13.5:

```
for x in range(5):
	print(x)
```

Экранға шығатын мән:

```
0
1
2
3
4
```

Яғни экранға 0-ден 5-ке дейінгі сандар шықты.

13.5. Бірінің ішіне салынған for циклі

if тұжырымы сияқты **for** тұжырымы да бірінің ішіне салына алады. Мұндай жағдайды түсіндіру үшін мына мысалға назар аударайық

Мысал 13.6:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
	for y in fruits:
		print(x, y)
```

Экранға шығатын мәндер:

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Бұл жағдайда алдымен `adj` тізбегі бойынша жүгіреміз. Кейін `adj` тізбегінің әр элементі үшін `fruits` тізбегі арқылы жүгіреміз. Бұл бірінің ішіне салынған циклдер қадамдары мына төмендегі кестеде түсіндірілген.

1-ші цикл		2-ші цикл		Экранға шығатын мән
қадам	adj тізбегі	қадам	fruits тізбегі	
1	red	1	apple	red apple
		2	banana	red banana
		3	cherry	red cherry
2	big	1	apple	big apple
		2	banana	big banana
		3	cherry	big cherry
3	tasty	1	apple	tasty apple
		2	banana	tasty banana
		3	cherry	tasty cherry

13.1-кесте: Бірінің ішіне салынған цикл қадамдары

13.6. Pass тұжырымы

Кей жағдайда `for` циклінің ішінде ешқандай операция жасау керек болмайды. Мұндай жағдайда `pass` тұжырымы қолданылады. Мұндайда `for` циклінен кейін `pass` кілтті сөзін жазу жеткілікті болады.

```
for қандай да бір айнымалы in тізбек:
       pass
```

14

While циклі

14.1. While циклі

Циклдің екінші түрі — **while** циклі. Бұл циклдің жазылу жолы мынадай:

```
while қандай да бір логикалық амал дұрыс болмағанша:
        қандай да бір амалдарды орындау
```

while циклін түсіндіру үшін төмендегі мысалға назар аударайық.

Мысал 14.1:

```
i = 0
while i < 3:
        print(i)
        i += 1
```

Экранға шығатын мән:

```
0
1
2
```

Алдымен **i = 0** айнымалысын жариялаймыз. Содан кейін **while** циклінде сол айнымалы 3-тен кіші болғанша **print(i)** және **i += 1** амалдарын орындаймыз. Яғни **while** кілтті сөзінен кейін қандай да бір логикалық амал шындық (**True**) мәнге тең болғанша **while** цикліне тиісті амалдарды орындаймыз. Логикалық амалдан кейін қос нүкте символы қойылады. Бұл мысалды толығымен түсіну үшін мына кестеге назар аударайық (Кесте 14.1).

i айнымалысының мәні (итерация)	Логикалық амал	while цикліне тиесілі амалдар	Экранға шығатын мәтін
i = 0	0 < 3 (True)	print(i) i += 1	0
i = 1	1 < 3 (True)	print(i) i += 1	1
i = 2	2 < 3 (True)	print(i) i += 1	2
i = 3	3 < 3 (False)	Циклдан шығамыз, себебі логикалық амалдың мәні **False**-қа тең	

14.1-кесте: while циклінің қадамдары

14.2. break тұжырымы

for цикліндегідей **while** цикліне де **break** тұжырымын қолдана аламыз. Бұл тұжырымның басты мақсаты — циклден шығу. **break** тұжырымының **while** цикліндегі қолданысын төмендегі мысал арқылы көрсетейік.

Мысал 14.2:

```
i = 0
while i < 3:
	print(i)
	i += 1
	if i == 1:
		break
```

Экранға шығатын мән:

```
0
```

Алдымен **i = 0** айнымалысын жариялаймыз. Кейін **while** циклінде сол айнымалы 3-тен кіші болғанша **print(i)** және **i += 1** амалдарын орындаймыз. Яғни **while** кілтті сөзінен кейін қандай да бір логикалық амал шындық (**True**) мәнге тең болғанша **while** цикліне тиісті амалдарды орындаймыз. Бірақ, әр қадамымызда **i** мәнін 1-мен салыстырып отырамыз. Егер **i = 1** болған жағдайда **break** тұжырымын шақыру арқылы отырып, циклден шығып, әрі қарай цикл қадамдарын орындауды тоқтатамыз

14.3. continue тұжырымы

for цикліндегідей **while** цикліне де **continue** тұжырымын қолдана аламыз. Бұл тұжырымның басты мақсаты — циклдегі қандай да бір шарт үшін цикл қадамын ары қарай орындамай, цикл қадамынан аттап кету. **continue** тұжырымының **while** цикліндегі қолданысын төмендегі мысал арқылы көрсетейік.

Мысал 14.3:

```
i = 0
while i < 3:
	i += 1
	if i == 1:
		continue
	print(i)
```

Экранға шығатын мән:

```
2
3
```

Алдымен **i = 0** айнымалысын жариялаймыз. Содан кейін **while** цикліне сол айнымалы 3-тен кіші болғанша **print(i)** және **i += 1** амалдарын орындаймыз. Яғни **while** кілтті сөзінен кейін қандай да бір логикалық амал шындық (**True**) мәнге тең болғанша, **while** цикліне тиісті амалдарды орындаймыз.

Бірақ, әр қадамымызда `i` мәнін 1-мен салыстырып отырамыз. Егер `i = 1` болған жағдайда `continue` тұжырымын шақыру арқылы, циклдің осы қадамындағы амалдарды орындамай, циклдің келесі қадамына көшеміз. Бұл мысалды толығымен түсіну үшін мына кестеге назар аударайық (Кесте 14.2).

і мәні (итерация)	Логикалық амал	while цикліне тиесілі амалдар	if шартты операторы	Экранға шығатын мәтін
`i = 0`	`0 < 3 (True)`	`i += 1`	`i == 1 (True) continue`	
`i = 1`	`1 < 3 (True)`	`i += 1`	`i == 2 (False)`	`print(i) 2`
`i = 2`	`2 < 3 (True)`	`i += 1`	`i == 3 (False)`	`print(i) 3`
`i = 3`	`3 < 3 (False)`	Циклдан шығамыз, себебі логикалық амалдың мәні `False`-қа тең		

14.2-кесте: continue қосылған while циклінің қадамдары

14.4. Бірінің ішіне салынған while циклі

`if` және `for` тұжырымдары сияқты `while` тұжырымы да бірінің ішіне салына алады. Сонымен бірге, бұл үш амалды өзара араластырып бірінің ішіне сала аламыз.

15

Функция

15.1. Функция

Көп жағдайда бағдарламаны жазу барысында қандай да бір код блогы қайталана беруі мүмкін. Сол кезде код блогын қайталап жазу — жаман үрдіс. Сол себепті функция қолданылады. Функция — шақырылған кезде ғана жұмыс істейтін код блогы. Яғни сол қайталанатын код бөлігін жеке функция ретінде жазып, керек кезде шақыртып орындаса болады. Сондай-ақ параметрлер ретінде функцияға қандай да бір деректерді жіберуге де болады. Функция нәтиже ретінде сол параметрлермен қандай да бір жұмыс атқарып, белгілі бір нәтижеге жете алады. Функцияның қарапайым мысалы мынадай.

Мысал 15.1:

```
def my_function():
      print("Hello from a function")
```

Функцияны жариялау үшін **def** кілтті сөзі қолданылады. **def** кілтті сөзінен кейін функцияның атын жазу керек. Функцияның атын өзіңіз қалағаныңыздай жаза аласыз. Бірақ функцияның аты ол өзі орындайтын амалдарды сипаттайтындай атау болуы қажет. Сонымен қатар айнымалыларға атау берген сияқты, функцияларға ат беру кезінде бас және кіші латын әріптері, сандар және астыңғы сызу (_) символын қолдана алады. Функцияның аты кіші латын әріпімен басталу керек. Функцияның атынан кейін ашылған және жабылған жақшаны жазамыз. Бұл жақшалардың ішіне функцияға керекті параметрлерді жазса да болады. Сонымен бірге ешқандай параметрсіз функцияны жариялаасақ та болады. Жабылған жақшадан кейін қос нүкте койып, келесі жолдан бастап, аралықты қолдана отырып, функцияның денесін жазамыз. Функцияның денесінде функция атқару керек болатын амалдарды орындаймыз. Жоғардағы мысал үшін **print("Hello from a function")** мәтіні функцияның денесі және атқаратын қызметі болып табылады. Функцияны шақыру үшін функцияның атын, ашылған және жабылған жақшаларды жазу керек.

Мысал 15.2:

```
def my_function():
      print("Hello from a function")

my_function()
```

Экранға шығатын мән:

```
Hello from a function
```

15.2. Функцияның параметрі

Көп жағдайда бағдарламаға функцияға қажетті параметрлерді немесе қандай да бір деректерді беру керек болады. Мұндай жағдайда деректерді функцияларға параметр ретінде беруге болады. Параметрлер функцияның атынан кейін жақшаның ішінде көрсетіледі. Неше түрлі параметрді өз қалауыңыз бойынша функцияға бере аласыз. Бұл параметрлерді үтір арқылы бөлу керек. Кейбір әдебиеттерде параметрлерді аргументтер деп те атайды. Төмендегі мысалда функцияға **fname** дейтін бір ғана параметр жалғанған. Функция шақырылғанда жақша ішінде бізге керекті параметрді жазу керек.

Мысал 15.3:

```
def my_function(fname):
	print(fname + " is kazakh writer")

my_function("Magzhan")
my_function("Akhmet")
my_function("Ilyas")
```

Экранға шығатын мән:

```
Magzhan is kazakh writer
Akhmet is kazakh writer
Ilyas is kazakh writer
```

Функцияны жариялағанда жақша ішіне параметрдің атын жазамыз. Содан кейін функция денесінде сол параметрмен белгілі бір жұмысты орындаймыз. Функцияны шақырған сайын әр түрлі параметрлер беріп, экраннан әр түрлі мәтіндерді көреміз. Бұл мәтіндердің айырмашылығы — берілген параметрдің өзгешелігінде.

15.3. Параметрлер саны

Функцияны жариялағанда және шақырғанда параметрлер саны бірдей болу керек. Мысалы, төмендегі бағдарламада қате бар. Себебі, параметрлер саны бірдей емес, әр-түрлі.

Мысал 15.4:

```
def my_function(fname, lname):
	print(fname + " " + lname)

my_function("Emil")
```

Ал мына функция дұрыс орындалады. Себебі, функция жариялаған кездегі және шақырған кездегі параметрлер саны бірдей.

Мысал 15.5:

```
def my_function(fname, lname):
	print(fname + " " + lname)

my_function("Magzhan", "Kairanbay")
```

Экранға шығатын мән:

```
Magzhan Kairanbay
```

15.4. Параметрге тізбекті жіберу

Функцияның параметріне тізбекті де бере аламыз. Төмендегі мысалда тізбек функция параметрі ретінде беріліп, содан кейін `for` циклі арқылы сол тізбектегі барлық элементтерді экранға шығарамыз.

Мысал 15.6:

```
def my_function(food):
      for x in food:
            print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Экранға шығатын мән:

```
apple
banana
cherry
```

15.5. Функцияның қайтару мәні

Егер функция қандай да бір мәнді қайтару керек болса, онда **return** кілтті сөзі қолданылады. **return** кілтті сөзінен кейін қайтарылатын мән жазылу керек. Төмендегі мысалды қарастырайық.

Мысал 15.7:

```
def my_function(x):
      return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Экранға шығатын мән:

```
15
25
45
```

Бұл жағдайда қайтарылатын мән параметрді беске көбейткен сан болып табылады.

15.6. Тізбекке қолданылатын функциялар

Тізбектің өзіне қатысты функциялары болады. Тізбекке қатысты функциялардың кейбіреулері мына кестеде көрсетілген (15.1-кесте).

Функцияның аттауы	Сипаттамасы
`append()`	Берілген элементті тізімнің соңына қосады
`clear()`	Тізімнен барлық элементтерді жояды

`copy()`	Тізімнің көшірмесін қайтарады
`count()`	Параметр ретінде берілген элементтің санын қайтарады
`index()`	Параметр ретінде берілген мәннің тізбектегі бірінші индексін қайтарады
`insert()`	Көрсетілген орынға (индекс ретінде) элемент қосады
`pop()`	Белгіленген орындағы элементті жояды
`remove()`	Параметр ретінде берілген мәні бар бірінші элементті жояды
`reverse()`	Тізім ретін өзгертеді (тізім соңын басына, ал басын соңына аударады)
`sort()`	Тізімді сорттайды

15.1-кесте: Тізбекке қатысты функциялар

Бұл функцияларды қолдану үшін тізбектің атын, кейін функцияның атауын б ал параметрлер керек жағдайда параметрлерді жазамыз. Төмендегі мысалдарға назар аударайық.

Мысал 15.8:

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
print(fruits)
```

Экранға шығатын мән:

```
['apple', 'banana', 'cherry', 'orange']
```

Мысал 15.9:

```
fruits = ['apple', 'banana', 'cherry']
fruits.reverse()
print(fruits)
```

Экранға шығатын мән:

```
['cherry', 'banana', 'apple']
```

Мысал 15.10:

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.index("cherry")
print(x)
```

Экранға шығатын мән:

```
2
```

Мысал 15.11:

```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort()
print(cars)
```

Экранға шығатын мән:

```
['BMW', 'Ford', 'Volvo']
```

16

Класс

16.1. Объектіге бағытталған бағдарламалау

Python бағдарламалау тілі объектіге бағытталған болып табылады. Объектіге бағытталған дегеніміз — бағдарлама құру барысында объектілерді қолдану арқылы бағдарлама құру. Әр бір объектінің өзіне тиісті параметрлері және функциялары болады. Мәселен, қандай да бір адамды объект ретінде қарастырсақ, онда сол адамның параметрлері (кей әдебиеттерде аттрибуттары деп те қолданылады) мынадай болуы мүмкін: сол адамның есімі, тегі, бойының ұзындығы және т.б. Яғни, объектінің параметрлері сол объектіні сипаттайтын заттар болып табылады. Сонымен бірге, сол адамның қызмет атқаратын функциялары бар. Олар, мысалы: көру, жүру, ұйықтау және т.б. Яғни бұл функциялар — объектінің (біздің жағдайда қандай да бір адамның) атқаратын қызметтері. Бұл жағдайда біз адамның қандай да бір түрін объект ретінде қарастырдық. Яғни бізде он адам болса, онда бізде он объект бар деген сөз. Бұл он адамның бәріне тән аттрибуттары (аты, жөні, бойының ұзындығы және т.б.) мен функцияларын (көру, жүру, ұйықтау және т.б.) жоғарыда көрсеттік. Бұл жағдайда адам — класс, ал қандай да бір адам — сол классқа тиесілі объект болып табылады. Мұнда класс — қандай да бір ұқсас объектілерді сипаттайтын аттрибуттар мен функциялар жиынтығы. Ал объект — сол класстың өкілі. Әр объектінің әр аттрибуты бойынша өзіне тән мәні бар. Мысалы, адам аты және тегінен тұратын класс болып табылса (адам классының аттрибуттары — адамның аты және тегі), онда сол классқа тиесілі бірінші объектінің параметрлері төмендегідей болуы мүмкін: бірінші объектінің (адамның) аты — Мағжан, ал тегі — Жұмабаев, ал екінші объектінің (адамның) аты — Ахмет, ал тегі — Байтұрсынов. Яғни қандай да бір классқа (біздің жағдайда адам классына) тиесілі объект сол классқа тиесілі аттрибуттар бойынша (біздің жағдайда адамның аты және тегі) өзіне тән мәні бар. Классты және объектіні Python тілінде жариялау жолын көрсетейік.

Мысал 16.1:

```
class Person:
       name = “Magzhan”

p1 = Person()
print(p1.name)
```

Экранға шығатын мән:

```
Magzhan
```

Python тілінде классты жариялау үшін `class` кілтті сөзін қолданамыз. `class` кілтті сөзінен кейін класс атауын жазамыз. Класс атауы бас әріптен басталып, сол классты сипаттайтын сөз болуы қажет. Класс атауын жазғаннан кейін қос нүкте қою керек. Содан кейін келесі жолдарда аралықты қолданып класс аттрибуттарын жазамыз. Жоғарыдағы мысал үшін `Person` классында тек бір ғана аттрибут бар, ол — `name`, яғни адам аты. Енді сол класстың объектісін жариялайық. Ол үшін объектінің атын жазып (біздің жағдайда `p1`), содан кейін ол айнымалыны `Person` классына теңестіреміз. Класс атауынан кейін ашылған және жабылған жақша болуы керек. Бұл жақшалар конструкторды білдіреді. Конструкторды келесі тақырыпта толығымен түсіндіретін боламыз. Объектінің параметріне қол жеткізу үшін сол объектінің атын (біздің жағдайда `p1`) жазып, содан кейін нүктеден соң класс аттрибуттарын жазамыз (біздің жағдайда `name`). Ал `p1.name = “Magzhan”` болғандықтан, экранға `Magzhan` сөзі шығады. Айнымалының мәнін өзгерткен сияқты, объектінің де аттрибуттарының мәнін де өзгерте аламыз. Жоғардағы мысал үшін `p1.name = “Akhmet”`, содан кейін `print(p1.name)` деп жазсақ, экранға `Akhmet` деген сөз шығады.

16.2. Конструктор

Конструктор — ағылшын тілінен “құру” деген мағынаны білдіреді. Конструктор объектіге бағытталған бағдарламалауда қандай да бір классқа тиесілі объектті құру үшін қолданылатын функция. Конструктордің ішінде класс аттрибуттарының мәні жарияланады. Конструктор функция сияқты параметрлер қабылдай алады. Егер ешқандай параметр берілмесе онда мұндай конструкторда аттрибуттар қандай да бір тұрақты мәнге меншіктеледі (мысалы 0-ге немесе бос мәтінге — “”). Ал егер конструкторға параметрлер берілсе, онда конструктор сол параметрлерді класс аттрибуттарына теңеп, солайша класс объектісін құрады. Конструктордың Python тіліндегі қолданысын келесі мысалды қарастырайық.

Мысал 16.2:

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

p1 = Person("Magzhan", 31)

print(p1.name)
print(p1.age)
```

Экранға шығатын мән:

```
Magzhan
31
```

Python тілінде конструктор **__init__** функциясы арқылы жарияланады. Конструктордың бірінші параметрі сол класс аттрибуттарына қол жеткізу үшін қолданылады. Сол себепті, көбінесе бірінші параметрді **self** (“осы, өзі” деп аударылады) деп атайды. **self** параметі арқылы класс аттрибуттарына қол жеткізіп (**self.name, self.age** деп жазу арқылы), класс аттрибуттарын конструкторға берілген парамертлерге меншіктейміз. Объектіні жариялаған кезде, класстың сыртында конструкторды шақырып, сол конструкторға керекті параметрлерді береміз. Осылайша классқа тиесілі объектті жариялап шығамыз. Жоғардағы мысал үшін, экранға шығатын мәндер **Magzhan** және **31**.

16.3. Класс функциясы

Жоғарыда класс аттрибуттары және класс функциялары бар деп айтып кетіп едік. Осыған дейін класс аттрибуттарын қарастырып келдік. Енді осы жолы класс функциясын қарастыратын боламыз. Класс функциясы класс аттрибуттары сияқты классқа тиесі болып табылады. Класс функциясын шақыру үшін объект атауын, кейін функция атауын жазу керек. Қарапайым функцияларды шақырған сияқты функция атауынан, кейін ашылған және жабылған жақша жазып, жақша ішіне керекті параметрлерді берсек болады. Класс функциясының Python тіліндегі қолданысы келесідей:

Мысал 16.3:

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

	def introduce(self):
		print("Hello my name is " + self.name)

p1 = Person("Magzhan", 31)
p1.introduce()
```

Экранға шығатын мән:

```
Hello my name is Magzhan
```

Person классының ішінде **introduce** функциясын жариялап, сол функцияға **self** параметрін береміз. **self** параметрін қолдану — класс аттрибуттарына қол жеткізу. Функция денесінде экранға **“Hello my name is ”** және **name** атты класс аттрибутын шығарамыз. Класс функциясын тексеру үшін **p1** атты класс объектісін **“Magzhan”** және **31** параметрлерін бере отырып құрамыз. Кейін **p1** объектісінің **introduce** функциясын шақырамыз. Ол үшін объект атауы (біздің жағдайда **p1**), кейін нүкте арқылы функция атауын жазамыз. Функцияны шақырғаннан кейін экранға **“Hello my name is Magzhan”** мәтіні шығады.

16.4. __str__ функциясы

Көптеген жағдайларда объектілерді экранға шығару керек болады. Жоғарыдағы мысал үшін, егер біз **print(p1)** деп жазатын болсақ, онда экранға **<__main__.Person object at 0x15039e602100>** сияқты мәтін шығады. Ал, бізге сол объектінің аттрибуттары шығу керек. Сол себепті **__str__** функциясы қолданылады. **__str__** функциясы арқылы экранға объект аттрибуттарының мәнін өзімізге керекті форматта шығаруға болады. **__str__** функциясының қолданысы төмендегідей.

Мысал 16.4:

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

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

Экранға шығатын мән:

```
Magzhan(31)
```

`print(p1)` командасын шақырғанда `__str__` функциясы шақырылып, класс аттрибуттарын экранға мәтіндік форматта шығарамыз.

17

Мұрагерлік

17.1. Мұрагерлік

Объектіге бағытталған бағдарламалауда мұрагерлік өте жиі кездеседі. Мұрагерлік дегеніміз — басқа класстан барлық функуиялар мен аттрибуттарды мұраға алатын құбылыс. Мұндағы ата-ана класы — мұралыққа берген негізгі класс, ал бала класс — мұралыққа алған класс. Ата-ана классы кез-келген класс бола алады. Сол себепті ата-ана классын жариялағанда қарапайым класс жариялағандай боламыз. Мысалы, `Person` классын жариялайық.

Мысал 17.1:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

x = Person("Magzhan", "Kairanbay")
x.printname()
```

Экранға шығатын мән:

```
Magzhan Kairanbay
```

Енді осы класстан мұрагерлік болатын `Student` классын жариялайық. Ол класстың ішінде ештеңе болмасын. Ол үшін `pass` кілтті сөзін жазу керек.

Мысал 17.2:

```
class Student(Person):
       pass

x = Student("Magzhan", "Kairanbay")
x.printname()
```

Экранға шығатын мән:

```
Magzhan Kairanbay
```

Бұл жағдайда `Student` классы `Person` классынан мұрагерлікті талап етеді. Мұрагерлік дегеніміз `Person` классының аттрибуттары мен функциялары `Student` классында да бар дегенді білдіреді. Яғни `Student` классының объектісін жарияласақ, бұл объектіде `firstname, lastname` аттрибуттары мен `printname` функциясы болады. Сол себепті, `Student` объектісін жариялағанда `firstname` және `lastname` сияқты аттрибуттарды конструкторға береміз. Содан кейін `Student` классының объектісінде `printname` функциясын шақыра аламыз. Бұл функцияны шақырғанда экранға біз берген адам аты-жөні шығу керек. Бұл жағдайда біз ата-анасынан аттрибуттар мен функцияларды мұраға алатын бала классын құрдық. Бұдан бөлек бала классқа тиесілі қосымша аттрибуттар мен функцияларды және сол классқа тиесілі конструкторды қосуға болады. Бұған көз жеткізу үшін төмендегідей мысалды қарастырайық.

Мысал 17.1:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
        print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)

x = Student("Magzhan", "Kairanbay", 2013)
x.welcome()
```

Экранға шығатын мән:

```
Welcome Magzhan Kairanbay to the class of 2019
```

Мұнадғы ата-ана классы — алдыда құрған **Person** классы. Student классының өзіне тән **__init__** конструкторы бар. Бұл конструктор ата-анасының конструкторын шақырады. Ата-анасының конструкторын шақыру үшін **super().__init__(fname, lname)** коды қолданылады. Мұндағы, **super()** ата-анасына сілтеме көрсетсе, **__init__** ата-анасының конструкторын көрсетеді. Бұл конструктор параметр ретінде **fname** және **lname** параметрін қабылдайды. **Student** классына тиесілі **graduationyear** параметрін **Student** классының конструкторында меншіктейміз. Яғни **self.graduationyear = year**. Бұған қоса **Student** классына тиесілі **welcome** функциясын жазамыз. Яғни **Student** классында екі функция бар десек болады. Бірі — ата-анасынан мұраға алған — **printname** функциясы, ал екіншісі — өзінің жеке функциясы — **welcome**. **Student** классын құрғаннан кейін сол класстың объектісін жарияласақ болады. Біздің жағдайда **Student** классы үш аттрибутты қабылдайды (**Student("Magzhan", "Kairanbay", 2013)**). Оның екеуі ата-анасынан мұраға алған — **fname** және **lname** аттрибуттары, ал үшіншісі — өзінің жеке **graduationyear** аттрибуты. **Student** классының **welcome** функциясын шақырғанда, экранға “**Welcome Magzhan Kairanbay to the class of 2019**” мәтіні шығады.

18

Try except

18.1. Try except

`try except` тұжырымы қателермен жұмыс істеу үшін қолданылады. `try except` тұжырымы келесідей төрт бөліктен тұрады:

- `try` — код блогында қателердің бар жоғын тексеруге мүмкіндік береді
- `except` — қатені өңдеу үшін қолданылатын блог
- `else` — қате болмаған кездегі амалдар жиынтығы орындалатын блог
- `finally` — жоғарыдағы жағадайлардың бәрін орындап болғаннан кейінгі (қате болса да, болмаса да) орындалатын блог

Код жазу барысында қандай да бір қателер пайда болуы мүмкін. Бұл қателер пайда болғанда бағдарлама өзінің жұмыс істек қызметін тоқтатады. Бірақ, бағдарламада қате пайда болған жағдайда да ары қарай жұмыс істеу үшін `try except` блогын қолдану керек болады. Орындауға керекті амалдарды `try` блогының ішіне салып, содан кейін сол блогта қате пайда болған жағдайда, `except` блогында сол қатемен жұмыс істейтін код блогы жазылады. Мысалы, біз экранға `x` айнымалысын жарияламастан шығарып көрейік. `x` айнымалысын жарияламағандықтан, бағдарлама қате шығарып, өз қызметін тоқтатуы керек. Ал егер сол кодты `try` блогының ішіне жазсақ, мынадай бағадарлама шығады.

Мысал 18.1:

```
try:
  print(x)
except:
  print("Айнымалы жарияланбады")
```

Экранға шығатын мән:

```
Айнымалы жарияланбады
```

Пайда болған қатені `try` блогы анықтап, кейін сол қатені `except` бөлігіне береді. `except` бөлігінде экранға “Айнымалы жарияланбады” дейтін мәтінді шығарамыз. Яғни, `try except` блогын қолдана отырып, бағдарлама қатені анықтайды, сол қатемен ары қарай жұмыс істеп, осылайша өз жұмыс істеу қызметін тоқтатпайды. Ал егер `try except` блогын қолданбай, жоғардағы бағдарламаны жазсақ, бағдарлама өз қызметін тоқтатып, төмендегідей қате шығады.

Мысал 18.2:

```
print(x)
```

Экранға шығатын мән:

```
Traceback (most recent call last):
  File "demo_try_except_error.py", line 3, in <module>
    print(x)
NameError: name 'x' is not defined
```

Бұл қатеден көріп отырғанымыздай, экранға “x айнымалысы жарияланбады” деген мәтін шықты.

18.2. Бірнеше except блогы

Қателердің түрлері көп бола береді. Қателердің түрлеріне байланысты қандай да бір қатенің түрін анықтап, сол қатемен жұмыс жасауға болады. Ол үшін әр қатегің түрі үшін бірнеше **except** блогын жазуға болады. Әр **except** блогынан кейін қатенің атын жазу керек. Бірнеше қатені өңдеуді тереңірек түсіну үшін, мына мысалға назар аударайық.

Мысал 18.3:

```
try:
	print(1 / 0)
	print(x)
except NameError:
	print("x айнымалысы жарияланбаған")
except ZeroDivisionError:
	print("Санды 0-ге бөле алмаймыз")
```

Экранға шығатын мән:

```
Санды 0-ге бөле алмаймыз
```

Бұл кодты жүгіртсек, экранға **"Санды 0-ге бөле алмаймыз"** мәтіні шығады. Себебі, алғашқыда бір санын нөлге бөлу барысында қате шығады. Ал бұл қатенің түрі **ZeroDivisionError** деп аталғандықтан, **ZeroDivisionError ecxept** блогындағы кодтардың жиынтығын орындаймыз. Ал енді бірді нөлге бөлмей, екіге бөліп, кодты қайта жүгіртейік.

Мысал 18.4:

```
try:
	print(1 / 2)
	print(x)
except NameError:
	print("x айнымалысы жарияланбаған")
except ZeroDivisionError:
	print("Санды 0-ге бөле алмаймыз")
```

Экранға шығатын мән:

```
0.5
x айнымалысы жарияланбаған
```

Бұл жағдайда экранға **“x айнымалысы жарияланбаған”** мәтіні шығады. Себебі, **print(1 / 2)** код бөлігінде ешқандай қате жоқ, ал **print(x)** бөлігі **NameError** қатесін тудыратындықтан, **NameError except** бөлігіндегі кодтың блогын орындап, экранға **“x айнымалысы жарияланбаған”** мәтінін шығарамыз.

18.3. Else блогы

Егер **try** блогында қате анықталмаса, онда **else** блогындағы кодты жүгіртуге болады. **else** блогын тереңірек түсіну үшін, төмендегі мысалғы назар аударайық.

Мысал 18.5:

```
try:
  print("Сәлем әлем")
except:
  print("Қате пайда болды")
else:
  print("Ешқандай қате жоқ")
```

Экранға шығатын мән:

```
Сәлем әлем
Ешқандай қате жоқ
```

Бұл кодта ешқандай қате болмағандықтан, алдымен **“Сәлем әлем”**, кейін экранға **else** блогындағы **“Ешқандай қате жоқ”** мәтіні шығады.

18.4. Finally блогы

try except блогында қате бар болса да, немесе жоқ болса да, **finally** блогын қосу арқылы сол блоктағы кодтар жиынтығын орындай аламыз. **finally** блогының қолданысын дұрыс түсіну үшін төмендегі мысалды қарастырайық.

Мысал 19.6:

```
try:
  print(x)
except:
  print("Қате пайда болды")
finally:
  print("try except блогы аяқталды")
```

Экранға шығатын мән:

```
Қате пайда болды
try except блогы аяқталды
```

Бұл бағдарламада алдымен қате пайда болып, ол қате **except** бөлігіндегі **"Қате пайда болды"** мәтіні арқылы өңделеді. Соңында **finally** блогындағы **"try except блогы аяқталды"** мәтіні экранға шығады. Ал егер **try** блогындағы **print(x)** мәтінін **print(“Сәлем әлем”)** кодына алмастырсақ, экранға алдымен **“Сәлем әлем”**, содан кейін **“try except блогы аяқталды”** деген мәтін шығады. Себебі, қате пайда болса да, болмаса да **finally** блогындағы код орындалады.

19

Input

19.1. Input

Осыған дейін біз деректерді кодта жариялап отырдық. Бірақ, көп жағдайда деректер сырттан бағдарламаға енеді. Сол деректерді қолдана отырып, біздің бағдарлама ары қарай жұмыс істей береді. Осы бөлімде деретерді сырттан қалай оқу керек екенін үйрететін боламыз. Python тілінің 3-нұсқасында сырттан деректерді оқу үшін **input()** функциясы қолданылады. **input()** функциясының қолданылуы мынадай:

Мысал 19.1:

```
username = raw_input("Enter username:")
print("Username is: " + username)
```

Консоль немесе терминалдан оқитын деректер:

```
Magzhan
```

Экранға шығатын мән:

```
Username is: Magzhan
```

Яғни, алдымен экранға **“Enter username: ”** мәтіні шығады. Содан кейін, біз өз тарапымыздан өз атымызды енгізуіміз керек. Содан соң, экранға **“Username is: _____”** мәтіні шығады, мұндағы _____ сіз енгізген мәтін. Консольдан деректерді оқыған кезде, оларды әрқашан мәтін ретінде оқимыз. Бірақ кейде сандарды да оқу керек болады. Сол кезде оқыған мәтінді сандық форматқа ауыстыру керек. Сандық форматтағы деректерді оқып, оны сандық форматқа ауыстыру мысалы мынадай.

Мысал 19.2:

```
a = raw_input(“Enter number a: ”)
b = raw_input(“Enter number b: ”)
a = int(a)
b = int(b)
c = a + b
print(c)
```

Консоль немесе терминалдан оқитын деректер:

```
3
4
```

Экранға шығатын мән:

```
7
```

20

pip

20.1. pip құралы

Түрлі бағдарламаларды жазу барысында олар неше түрлі құралдарды қажет етеді. Мысалы, жасанды интеллект бойынша жоба жасағанда соған қатысты құралдар керек болады. Ал сайт жасау барысында ол сайт жасауға арналған құралдар қажет етеді. Бұл құралдарды “кітапханалар” деп те атайды. Мысалы, сіз кітапханаға барсаңыз, өзіңізге керекті кітап іздейсіз. Сол сияқты жоба жасау барысында кітапханадан өзіңізге керекті құралдармен жұмыс істейсіз. Кітапхананы кейде “пакет” деп те атайды. Ал пакеттер сізге керекті модульдардан тұрады. Python бағдарламалау тілінде пакеттерді орнату үшін **pip** кілтті сөзі қолданылады. **pip** кілтті сөзі сіздің компьютеріңізде жұмыс жасайтынын тексеру үшін төмендегі команданы терминалға немесе консольға жазу керек: **pip --version**. Бұл командадан кейін **pip** құралының қайда орналасқаны және нұсқасы экранға шығады.

Мысал 20.1:

```
pip --version
```

Экранға шығатын мән:

```
pip 22.2.2 from C:\Users\MagzhanKairanbay\anaconda3\lib\site-packages\pip (python 3.9)
```

Егер **pip** орнатылмаса, онда мына қадамдарды жасау керек:

```
1.  curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
2.  python get-pip.py
```

Бұл қадамдардан кейін сізде **pip** құралы орнатылады.

20.2. pip арқылы пакетті орнату

pip арқылы пакетті орнату үшін **pip install** және пакеттің атын жазу керек. Мысалы, **camelcase** пакетін орнату үшін **pip install camelcase** командасын жазу керек. Керекті пакетті орнатқаннан кейін, сол пакетті бағдарламаңызға **import** кілтті сөзі арқылы қоса аласыз. Біздің мысалымызда — **import camelcase**. Керекті пакетті бағдарламаға қосқаннан кейін, сол пакеттің функцияларын қолдана аламыз. Жоғардағы **camelcase** мысалын қарасытырайық.

Мысал 20.2:

```
import camelcase

c = camelcase.CamelCase()

txt = "hello world"

print(c.hump(txt))
```

Экранға шығатын мән:

```
Hello World
```

`camelcase` пакетінің басты мақсаты — әр сөздің бірінші әріпін бас әріпке ауыстыру. Біздегі **“hello world”** мәтінінің әр сөзін `camelcase` пакетінің `hump` функциясы арқылы ауыстырдық. Ол үшін алдымен `camelcase` пакетінің конструкторын қолдана отырып `c` объектісін жарияладық. Сол объектінің `hump` функциясына біздегі мәтінді параметр ретінде бердік.

Егер бізге қандай да бір пакетті өшіру керек болса, онда `pip uninstall` және пакеттің атын жазу керек. Мысалы, `camelcase` пакеті үшін `pip uninstall camelcase` командасы `camelcase` пакетін компьютерден өшіреді.

20.3. Орнатылған пакеттер тізімі

Компьютеріңізге орнатылған пакеттер тізімін көру үшін `pip list` командасын жазу керек.

Мысал 20.1:

```
pip list
```

Экранға шығатын мән:

```
Package         Version
-----------------------
camelcase       0.2
mysql-connector 2.1.6
pip             18.1
pymongo         3.6.1
setuptools      39.0.1
```

Бұл команданы жазғаннан кейін пакеттің аттары және олардың нұсқасы шығады.

21

Файлмен жұмыс істеу

21.1. Файлмен жұмыс

Бағдарлама жазу барысында файлдармен жұмыс істеуге тура келеді. Яғни қандай да бір деректерді файлдан оқып, басқа деректерді файлға жазу керек болады. Python тіліде файлдармен жұмыс істеу үшін `open()` функциясы қолданылады. Бұл функция екі параметр қабылдайды, біріншісі — файл аты, ал екіншісі — файлмен жұмыс әдісі. Файлмен жұмыс әдісінің келесідей төрт жолы бар.

- `r` — Оқу — әдепкі мән. Файлдан деректерді оқу үшін файлды ашады. Егер файл жоқ болса, онда қате қайтарады.
- `a` — Append — файлға деректерді қосу үшін ашады. Егер сіз көрсеткен файл жоқ болса, онда сіз көрсеткендей файл құрады.
- `w` — Write — файлға деректерді жазу үшін файлды ашады. Егер сіз көрсеткен файл жоқ болса, онда сіз көрсеткендей файл құрады.
- `x` — Файлды құру. Егер сіз қалаған файл бар болса, онда қате қайтарады.

Бұларға қосымша файлды текстік форматта немесе бинарлы форматта қарасыту керектігін көрсететін келесідей жолдары бар.

- `t` — Мәтін — Әдепкі мән. Мәтіндік режим
- `b` — Бинарлы — екілік режим (мысалы, суреттер)

Файлдан деректерді оқу үшін келесі код жазу керек: `f = open("demofile.txt")`. Бұл код бөлігі келесі код бөлігімен бірдей: `f = open("demofile.txt", "rt")`. Себебі, `r` — файлдан деректерді оқу үшін қолданылса, ал `t` — мәтіндік режимде оқуды білдіреді.

21.2. Файлдан оқу

Бізде `demofile.txt` атты файл бар делік. Бұл файлда келесідей деректер болсын:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Бұл деректерді оқу үшін `open()` функциясын қолданамыз. Бұл функция файл объектісін қайтарады. Бұл объектінің `read()` функциясы арқылы файл ішіндегі деректерді оқимыз.

Мысал 21.1:

```
f = open("demofile.txt", "r")
print(f.read())
```

Экранға шығатын мән:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Егер бізге керекті файл басқа жерде орналасса, онда сол файлдың орналасқан жерін көрсету керек.

Мысал 21.2:

```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

Егер біз файлдың бастапқы бес символын ғана оқығымыз келетін болса, онда **read** функциясына 5 санын параметр ретінде беру керек.

Мысал 21.3:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Экранға шығатын мән:

```
Hello
```

Егер біз файлдың бірінші жолын ғана оқығымыз келсе, онда **readline()** функциясын қолдану керек.

Мысал 21.4:

```
f = open("demofile.txt", "r")
print(f.readline())
```

Экранға шығатын мән:

```
Hello! Welcome to demofile.txt
```

Ал егер осы файлдың екі жолын оқығымыз келсе, онда **readline()** функциясын екі рет жазу керек.

Мысал 21.5:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

Экранға шығатын мән:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
```

for циклін қолдана отырып файлдың әр жолы арқылы жүгіре аламыз.

Мысал 21.6:

```
f = open("demofile.txt", "r")
for x in f:
      print(x)
```

Экранға шығатын мән:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Файлдармен жұмыс істегенде файлдарды жабу жақсы үрдіс. Әйтпесе файлдар құртылып кетуі мүмкін. Файлдарды жабу үшін `close()` функциясы қолданылады.

Мысал 21.7:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Экранға шығатын мән:

```
Hello! Welcome to demofile.txt
```

21.3. Файлға жазу

Python тілінде файлға жазудың екі түрі бар. Біріншісі — деректерді файл соңына біріктіріп жазу (бұл жағдайда `a` параметрі қолданылады), ал екіншісі — файлдың ішіндегі деректерді өшіріп, үстіне жазу (ал бұл жағдайда `w` параметрі қолданылады). Келесі мысалдардың біріншісінде (мысал 21.8) біз файлдың соңына деректерді жазамыз, ал екіншісінде файлдағы деректерді өшіріп, кейін ішіне жаңа деректерді жазамыз (мысал 21.9).

Мысал 21.8:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

f = open("demofile2.txt", "r")
print(f.read())
```

Экранға шығатын мән:

```
Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

Мысал 21.8:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

f = open("demofile3.txt", "r")
print(f.read())
```

Экранға шығатын мән:

```
Woops! I have deleted the content!
```

21.4. Файл құру

Файлды құру үшін `open()` функциясына керекті параметр символдарын беру қажет. Келесі параметр символдары жаңа файл құру үшін қолданылады.

- `a` — Append — файлға деректерді қосу үшін ашады. Егер сіз көрсеткен файл жоқ болса, онда сіз көрсеткендей файл құрады.
- `w` — Write — файлға деректерді жазу үшін файлды ашады. Егер сіз көрсеткен файл жоқ болса, онда сіз көрсеткендей файл құрады.
- `x` — Файлды құру. Егер сіз қалаған файл бар болса, онда қате қайтарады.

Мысалы, келесі код бөлігі арқылы біз жаңа файл құрамыз. Бұл файлдың ішінде ешқандай дерек жоқ.

Мысал 21.9:

```
f = open("myfile.txt", "x")
```

Кітап дизайнері, авторы
Мағжан Қайранбай

Редакторы
Жұмабай Қайранбай