

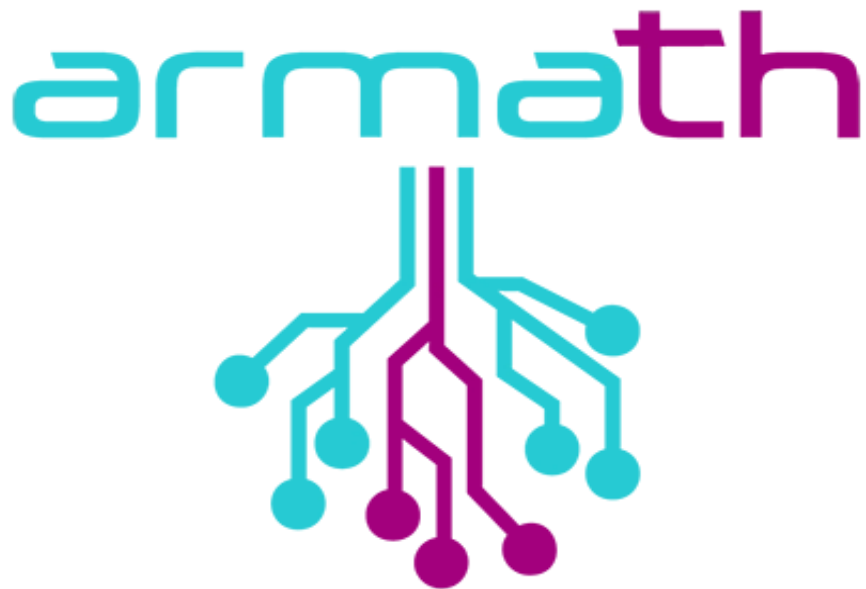
Հեղինակ՝ Մ. Ավետիսյան



# Python Լեզվի հիմունքներ

# Python ծրագրավորման լեզվի հիմունքներ

Հեղինակ՝ Մխիթար Ավետիսյան  
Տարբերակ՝ 1.0  
Ամսաթիվ՝ 21.07.2021 թ.



## Նախաբան

Այս գիրքը նախատեսված է սկսնակների, ինչպես նաև արդեն որոշակի գիտելիքներ ունեցող ծրագրավորողների համար, որոնք ցանականում են սովորել Python լեզուն: Գիրքն իր մեջ ներառում է լեզվի հիմնական հասկացությունները, քերականությունը և այդ լեզվով լուծվող խնդիրների օրինակներ ու առաջադրանքներ: Գրքում հաճախ կահանդիպեք փակագծերում կապույտով գրված լատինատառ բառերի, դրանց մեծ մասը կարևոր տերմիններ են ծրագրավորման մեջ, խորհուրդ է տրվում այդ տերմինները որոնել համացանցում և հավելյալ ուսումնասիրել, որպեսզի կարդալու ընթացքում ավելի խորը պատկերացնեք ծրագրավորման հիմնական կոնցեպտը:

Ինչպես ասել է Ալբերտ Էյնշտեյնը « *Ես երբեք չեմ սովորեցնում իմ աշակերտներին: Ես միայն փորձում եմ ապահովել այն պայմանները, որոնցում նրանք կարող են սովորել:* »

## Բովանդակություն

● Ներածություն .....	5
● Python ինտերպրետատորի տեղադրումը ՕՆ-ում .....	6
● Լեզվի օգտագործումը ինտերակտիվ ռեժիմում .....	7
● Առաջին “Hello World” ծրագիրը Python լեզվով .....	8
● Մեկնաբանություններ (Comments) .....	9
Գլուխ II փոփոխականներ և օպերատորներ .....	11
● Փոփոխականներ (Variables) .....	12
● Օպերատորներ .....	14
● Թվաբանական օպերատորներ .....	14
● Տրամաբանական օպերատորներ .....	15
● Համեմատման օպերատորներ .....	16
● Անդամակցության օպերատորներ .....	16
● Ինքնության օպերատորներ .....	17
Գլուխ III տվյալների տիպերը Python լեզվում .....	18
● Ամբողջ թվեր (Integers) .....	18
● Սահող կետով թվեր (Float) .....	18
● Տողեր (Strings) .....	19
● Տիպերի ձուլում .....	23

● Ցուցակներ -----	23
● Անփոփոխ Ցուցակներ -----	25
● Բառարանային տիպեր (Dictionary) -----	26
Գլուխ IV տվյալների մուտք և ելք -----	28
● Տվյալների մուտք, Input() -----	30
● Տվյալների ելք, Print() -----	30
● Հատուկ սիմվոլների արտածում -----	31
● Առաջադրանքներ -----	32
Գլուխ V պայմանական կառավարման կառուցվածքներ, ցիկլեր, բացառություններ-----	33
● If կառուցվածք -----	34
● Inline If կառուցվածք -----	36
● For ցիկլ -----	36
● While ցիկլ -----	40
● Break հրաման -----	41
● Continue հրաման -----	42
● Բացառությունների մշակում Try, Except -----	43
● Առաջադրանքներ -----	48
Գլուխ VI ֆունկցիաներ և մոդուլներ -----	50
● Ֆունկցիաներ և դրանց սահմանում -----	51

● Մոդուլներ և դրանց ստեղծումը -----	54
● Հիմնական ֆունկցիայի սահմանում (main()) -----	60
● Ներկառուցված մոդուլներ և փաթեթներ -----	63
● Ֆունկցիաների ռեկուրսիա (recursion) -----	65
● Առաջադրանքներ -----	67
Գլուխ VII աշխատանք ֆայլերի հետ -----	69
● Բացել և կարդալ (Text) ֆայլը -----	69
● Գրել (Text) ֆայլում -----	73
● Բացել, կարդալ և գրել երկուսական ֆայլերում -----	73
● Ջնջել և վերանվանել ֆայլը-----	75
● Առաջադրանքներ -----	75
Գլուխ VIII օբյեկտ-կողմնորոշված ծրագրավորում Python լեզվով -----	77
● Դասի սահմանումը (Class) -----	78
● Դասի օբյեկտներ (object) -----	79
● Մեթոդի սահմանում -----	80
● Դասերի ժառանգականություն (Inheritance) -----	81
● Առաջադրանքներ -----	84
Ամփոփում -----	86

## Գլուխ I միջավայրի Նախապատրաստում

Այս գլխում մենք կծանոթանանք Python լեզվին, կիմանանք դրա առանձնահատկությունները, կտեղադրենք Python-ի ինտերպրետատորը մեր օպերացիոն համակարգում (Windows կամ Linux) կևտրենք հարմար աշխատանքային միջավայր, որից հետո կգրենք մեր առաջին ծրագիրը Python լեզվով:

### Ներածություն

Python-ը լայնորեն օգտագործվող բարձր մակարդակի ծրագրավորման լեզու է, որը ստեղծել է Գվիդո վան Ռոսումը 1980-ականների վերջին: Այս լեզուն հիմնականում կենտրոնացված է ծրագրի արագ մշակման և կոդի հեշտ ընթերցանության վրա: Այն ինտերպրետացվող ծրագրավորման լեզու է, որի հրամանները կատարում է արդեն պատրաստի ծրագիրը կամ այլ կերպ ասած ինտերպրետատորը, որն էլ մեզ թույլ է տալիս Python լեզվով գրված ծրագիրն աշխատեցնել առանց այն մեքենայական կոդի ([Machine code](#)) թարգմանելու:

Python լեզվի համար կան մշակված շատ մոդուլներ ([modules](#)) և փաթեթներ ([packages](#)), որոնց շնորհիվ կարող ենք աշխատել ծրագրավորման տարբեր ոլորտներում: Օրինակ՝ գրաֆիկական ([Graphical user interface](#)), ցանցային ([Network](#)

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

[programming](#)), կայքերի մշակում ([Web programming](#)),  
մեքենայական ուսուցում ([Machine Learning](#)), ավտոմատացված  
համակարգերի մշակում ([Automation](#)) և այլն: Python- ը նաև  
օբյեկտ-կողմնորոշված ծրագրավորման ([Object-oriented  
programming](#)) լեզու է, որը կուսումնասիրենք գրքի վերջին  
գլխում:

Python լեզուն ինտերպրետիվ է, այսինքն ինտերպրետատորի  
միջավայրում աշխատանքի ժամանակ կարող ենք կատարել  
([Run](#)) մեկական հրամաններ: Python լեզվում ապահովվում է  
հիշողության ավտոմատ կառավարում, բացառությունների  
մշակման մեխանիզմ և աշխատանք բարձր մակարդակի  
տվյալների կառուցվածքների հետ: ([Data Structures](#))

## Python ինտերպրետատորի տեղադրումը ՕՆ-ում

### Լինուքս ՕՆ-ում

Python ինտերպրետատորի տեղադրման համար Լինուքս  
([Mint](#), [Ubuntu](#), [Debian](#)) օպերացիոն համակարգերում  
անհրաժեշտ է բացել հրամանների տողը ([Terminal](#)) և  
կատարել հետևյալ հրամանները ըստ գրված  
հերթականության:

- 1 `sudo apt-get install software-properties-common`
- 2 `sudo add-apt-repository ppa:deadsnakes/ppa`
- 3 `sudo apt-get update`
- 4 `sudo apt-get install python3.8`



## Windows ՕՆ-ում

Windows-ում տեղադրելու համար այցելենք Python լեզվի պաշտոնական կայքի ներբեռնումների ([Downloads](#)) բաժին և ընտրենք Windows համակարգը ու ներբեռնենք տեղադրող ծրագիրը ([Installer](#)), կամ անցենք հետևյալ հղումով <https://www.python.org/downloads/windows/> , որից հետո ներբեռնենք ձեր օպերացիոն համակարգին համապատասխան տեղադրող ծրագիրը ([Windows installer 32-bit or 64-bit](#)) և տեղադրենք, իսկ ավելի մանրամասն կարող եք տեսնել այստեղ: <https://youtu.be/i-MuSAwgcwCU> :

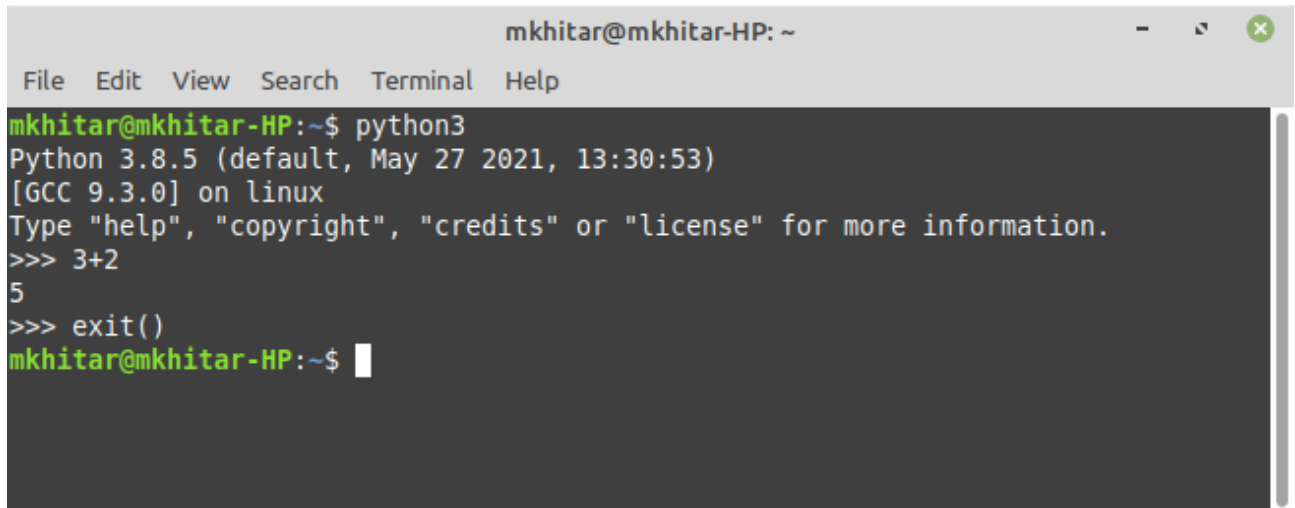
## Լեզվի օգտագործումը ինտերակտիվ ռեժիմում

Արդեն տեղյակ ենք այն մասին, որ Python -ը ինտերակտիվ է, դա մեզ տրամադրում է Python Shell -ը, որն օգտագործվում է մեկ հրամանի կատարման և արդյունքը ցուցադրելու համար, որից հետո նորից սպասում է հրամանի ներմուծման: Փորձենք կատարել մի պարզ հրաման այդ ռեժիմում, սկզբում թողարկենք Python Shell -ը, հրամանային տողում գրելով `python3` հրամանը և կտեսնենք հետևյալ սիմվոլները `>>>` : Դա կնշանակի, որ արդեն ինտերակտիվ աշխատանքի ռեժիմում է և սպասում է հրամանի ներմուծմանը, եթե ներմուծենք `3+2` -ը որպես հրաման ապա կտեսնենք հաջորդ տողում `5` պատասխանը, որից հետո նորից կցուցադրվի `>>>` սիմվոլները, որպեսզի ներմուծենք հաջորդ հրամանը և այդպես շարունակ: Այս կերպ կարող ենք ներմուծել `3>2`, որն կտալի `True` արժեքը, իսկ Python Shell-ը փակելու և

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

հրամանների մուտքն դադարեցնելու համար կարող եք  
օգտագործել `exit()` հրամանը:

Այս գործընթացը կունենա հետևյալ տեսքը:



```
mkhitar@mkhitar-HP: ~  
File Edit View Search Terminal Help  
mkhitar@mkhitar-HP:~$ python3  
Python 3.8.5 (default, May 27 2021, 13:30:53)  
[GCC 9.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 3+2  
5  
>>> exit()  
mkhitar@mkhitar-HP:~$
```

Կարելի է ասել, որ Python Shell -ը հարմար գործիք է  
սկսնակների համար լեզվի հրամանները կատարելու և  
դրանց աշխատանքին ծանոթանալու համար, բայց այս  
գործիքով չենք կարող լիարժեք ծրագիր կազմել, այդ  
նպատակով հաջորդ թեմայում կդիտարկենք թե ինչպես  
կարող ենք կազմել ամբողջական ծրագիր, և կգրենք առաջին  
ծրագիրը Python լեզվով:

## Առաջին *“Hello World”* ծրագիրը Python լեզվով

Նախ անհրաժեշտ է ընտրել մեզ հարմար տեքստային  
խմբագրիչ ([Text editor](#)) այս դասընթացի շրջանակներում  
կօգտագործենք VSCodium -ը, որը [Microsoft Visual Studio Code](#)  
տեքստային խմբագրիչի բաց կոդով ([open source](#))

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

տարբերակն է, այն կարող եք ներբեռնել և տեղադրել  
այստեղից <https://vscode.com/#install> : Այնուհետև  
աշխատանքային սեղանին (**Desktop**) ստեղծեք `python_projects`  
անունով թղթապանակ (**Folder**) և բացեք այն տեքստային  
խմբագրիչի միջավայրում, որից հետո ստեղծեք նոր ֆայլ  
խմբագրիչի միջոցով, ընտրելով միջավայրի վերևում նշված  
File բաժնից New File հրամանը արդյունքում կստեղծվի մի  
անանուն ֆայլ, որտեղ կգրենք մեր ծրագրի կոդը: Որն էլ ունի  
հետևյալ տեսքը:

```
#A program that will print "Hello World"  
print ("Hello World")
```

## Մեկնաբանություններ (**Comments**)

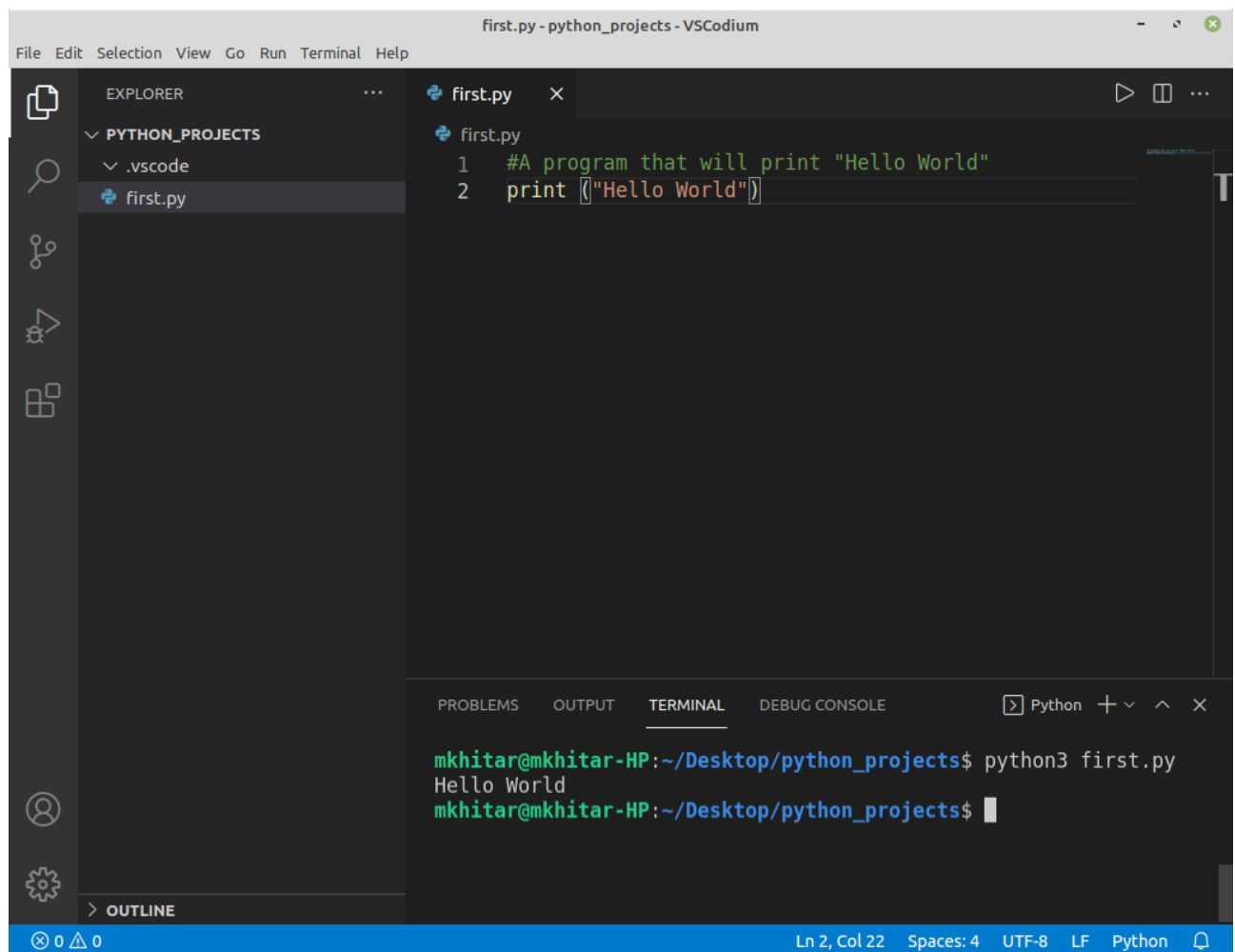
Նկատենք, որ կոդի սկզբում գրված է `#` սիմվոլով սկսվող մի  
տող, ինչպես շատ ծրագրավորման լեզուներում Python- ում  
նույնպես կա մեկնաբանության (**Comment** )  
հասկացողությունը, որը նշվում է `#` սիմվոլով, կամ հետևյալ  
սիմվոլներով `''' '''` օրինակ՝

```
'''  
  
This is a comment  
This is still a comment  
'''
```

Մեկնաբանությունները օգնում են ծրագրավորողներին կոդը  
ավելի հասկանալի դարձնելու համար և ոչ միայն իրենց, նաև  
այլ ծրագրավորողների համար, որոնք կաշխատեն այդ

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Ծրագրի վրա, հատկապես այն դեպքում երբ ծրագրի կոդը լինում է բավականին երկար: Այսպիսով խմբագրիչի միջավայրում File բաժնից ընտրենք Save As հրամանը և պահենք ֆայլը .py ընդլայնումով ([Extension](#)), այսքանով առաջին ծրագիրը պատրաստ է, այն ունի հետևյալ տեսքը:



```
first.py - python_projects - VSCodium
File Edit Selection View Go Run Terminal Help

EXPLORER
PYTHON_PROJECTS
  .vscode
  first.py

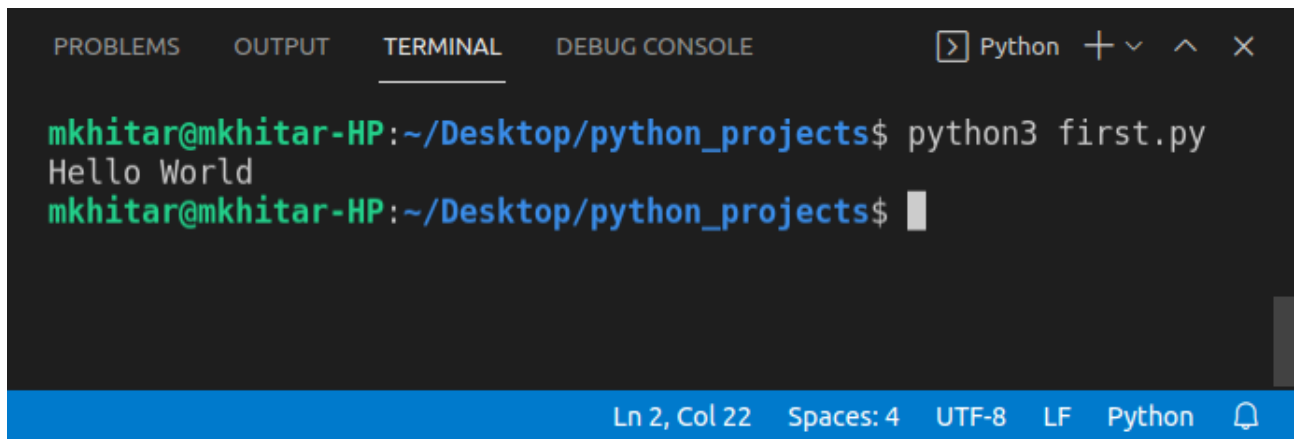
first.py
1  #A program that will print "Hello World"
2  print ("Hello World")

TERMINAL
Python
mkhitar@mkhitar-HP:~/Desktop/python_projects$ python3 first.py
Hello World
mkhitar@mkhitar-HP:~/Desktop/python_projects$
```

Ծրագիրն կատարելու համար խմբագրիչի միջավայրում

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Terminal բաժնից ընտրում ենք New Terminal հրամանը և միջավայրի ներքևում բացված հրամանային տողում կատարում հետևյալ հրահանգը `python3 (name).py`, որտեղ (name)- ի փոխարեն գրում ենք այն ֆայլի անվանումը, որում պահված է կոդը, այս օրինակում դա `first` բառն է, և կունենանք հետևյալ պատկերը:



The screenshot shows a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'TERMINAL', and 'DEBUG CONSOLE'. The 'TERMINAL' tab is active. The prompt is `mkhitar@mkhitar-HP:~/Desktop/python_projects$`. The command `python3 first.py` has been entered, and the output is `Hello World`. The prompt is now `mkhitar@mkhitar-HP:~/Desktop/python_projects$` with a cursor. At the bottom, a status bar shows `Ln 2, Col 22`, `Spaces: 4`, `UTF-8`, `LF`, `Python`, and a bell icon.

## Գլուխ II փոփոխականներ և օպերատորներ

Այս գլխում կծանոթանանք փոփոխականներին և օպերատորներին: Մասնավորապես, կհասկանանք թե ինչ է փոփոխականը և ինչպես հայտարարել ու անվանել: Բացի այդ մենք նաև կիմանանք ընդհանուր գործողությունների մասին, կամ այլ կերպ ասած մաթեմատիկական օպերատորների մասին:

## Փոփոխականներ (Variables)

Փոփոխականներ անվանում են այն տվյալներին, որոնց մենք հայտարարում ենք ծագրի մեջ, սկզբնարժեքավորում, հարկ եղած դեպքում փոփոխում և օգտագործում, կամ այլ կերպ ասած փոփոխականները հիշողության կտորներ են համակարգչում ինֆորմացիա պահելու համար, որոնք ունեն անուններ ընտրված մեր կողմից: Կա փոփոխականների երկու տեսակ՝ գլոբալ ([global](#)) և լոկալ ([local](#)): Գլոբալ փոփոխականների շրջանակը ամբողջ ծրագիրն է, մինչդեռ լոկալ փոփոխականների շրջանակը սահմանափակվում է այն ֆունկցիայով (գործառույթով), որտեղ այն սահմանվում է: Օրինակ՝ եթե Python լեզվում հայտարարենք `Age` անունով փոփոխական և այն սկզբնարժեքավորենք 20 թվով, ապա այդ հրամանը կունենա հետևյալ տեսքը: ԵՎ կլինի գլոբալ քանի որ սահմանված չէ ինչ-որ ֆունկցիայի տիրույթում: Հետագայում դեռ կուսումնասիրենք թե ինչ է ֆունկցիան և ֆունկցիայի տիրույթը:

```
Age = 20
```

Սկզբնարժեքավորում (վերագրում) գործողությունը տեղի է ունենում = սիմվոլի միջոցով: Այս հրամանի կատարման ժամանակ ծրագիրը մեր համակարգչի հիշողությունից հատկացնում է որոշակի տարածք, այդտեղ պահում վերագրված ինֆորմացիան, որը կարող եք փոփոխել օգտագործելով փոփոխականի անունը և վերագրման =

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

սիմվոլը: Ամեն անգամ, երբ հայտարարում եք նոր  
փոփոխական, հարկավոր է նախնական արժեք տալ դրան:

Python- ում փոփոխականների անունները կարող են  
պարունակել միայն տառեր (a - z, A – B), թվեր, կամ ( \_ )

սիմվոլը: Օրինակ՝ `userName`, `username2`, `user_name`  
բացի այդ, կան մի քանի բանալի բառեր (**key words**), որոնք չեն  
կարող օգտագործել որպես փոփոխականի անուն, քանի որ  
դրանք արդեն կան լեզվի քերականության մեջ և ունեն  
հատուկ իմաստներ, դրանցից են `print`, `input`, `if`, `while`, և այլն:

Դիտարկենք մի օրինակ փոփոխականների հայտարարման  
և վերագրման վերաբերյալ, դրա համար տեքստային  
խմբագրիչի միջովոց ստեղծենք նոր ֆայլ և այդ ֆայլում  
գրենք հետևյալ կոդը:

```
x = 5
y = 10
x = y
print ("x = ", x)
print ("y = ", y)
```

Որից հետո պահենք ֆայլը անավանելով այն `test.py` :

Տեքստային խմբագրիչի միջավայրում բացենք հրամանների  
տեղը (**Terminal**) և կատարենք ծրագիրը հետևյալ հրահանգով  
`python3 test.py` :

Կունենանք այսպիսի ելք:

```
x = 10
y = 10
```

Եթե կողի երրորդ տողում  $x = y$  արտահայտությունը փոխարինենք  $y = x$  -ով, ապա կունենանք այսպիսի ելք:

$x = 5$

$y = 5$

## Օպերատորներ

Օպերատորները Python- ում հատուկ սիմվոլներ են, որոնք իրականացնում են թվաբանական կամ տրամաբանական հաշվարկներ: Արժեքը, որի վրա գործում է օպերատորը, կոչվում է օպերանդ:

## Թվաբանական օպերատորներ

Բացի փոփոխականներին արժեքներ վերագրելուց, Python- ում կարող եք օգտագործել մաթեմատիկական օպերատորներ, դրանք են  $+$ ,  $-$ ,  $/$ ,  $*$  : Լեզվում կան նաև  $\%$ ,  $//$ ,  $**$  հավելյալ օպերատորները: Դիտարկենք այս բոլորը օրինակների վրա:

Ենթադրենք  $x = 5$  և  $y = 2$  ապա այդ դեպքում

$x + y = 7$

$x - y = 3$

$x * y = 10$

$x / y = 2.5$



Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

$x // y = 2$  ( $x$  բաժանում է  $y$  վրա և պատասխանը կլորացնում  
դեպի ներքև)

$x \% y = 1$  ( $x$  բաժանում է  $y$  վրա և տալիս է մնացորդը)

$x ** y = 25$  (հաշվում է  $x$ -ի  $y$  աստիճանը) :

Ինչպես շատ ծրագրավորման լեզուներում Python -ում  
նույնպես կա օպերատորով վերագրում գործողությունը,  
օրինակ՝

$x += y$  սա համարժեք է  $x = x + y$  : Որը նշանակում է  $x$ -ին  
վերագրել  $x + y$ -ը: Մնացած օպերատորները ևս կարելի է  
նույն կերպ կիրառել:

## **Տրամաբանական օպերատորներ**

Տրամաբանական օպերատորները Python -ում երեքն են `and`,  
`or` և `not` (և, կամ, ժխտում): Դրանց միջոցով կարող ենք  
ստուգել պայմանական արտահայտությունների ճիշտ կամ  
սխալ լինելը:

`and` -ով կապված արտահայտությունը ճիշտ է, եթե  
միաժամանակ ճիշտ են նրա երկու օպերանդները:  
`or` -ով կապված արտահայտությունը ճիշտ է, եթե նրա երկու  
օպերանդներից գոնե մեկը ճիշտ է:

`not` -ը ունի միայն մեկ օպերանդ, նրանով գրված  
արտահայտությունը ճիշտ է, եթե ճիշտ է այդ օպերանդի  
ժխտումը (հակառակը) :

## Համեմատման օպերատորներ

Համեմատման օպերատորները Python լեզվում `==`, `!=`, `<`, `>`, `<=`, `>=`, որոնք օգտագործվում են երկու օպերանդների միջև համամատություն անցկացնելու համար: Ենթադրենք `a` և `b` համեմատվող օպերանդներն են ապա այդ դեպքում

`a == b` ճիշտ է, եթե `a` -ն հավասար է `b` -ին:

`a != b` ճիշտ է, եթե `a` -ն հավասար չէ `b` -ին:

`a < b` ճիշտ է, եթե `a` -ն փոքր է `b` -ից:

`a > b` ճիշտ է, եթե `a` -ն մեծ է `b` -ից:

`a <= b` ճիշտ է, եթե `a` -ն փոքր կամ հավասար է `b` -ին:

`a >= b` ճիշտ է, եթե `a` -ն մեծ կամ հավասար է `b` -ին:

## Անդամակցության օպերատորներ

Անդամակցության օպերատորները երկուսն են `in` և `not in`, դրանց միջոցով ստուգվում է արդյոք առաջին օպերանդը պարունակում է երկրորդ օպերանդում, որտեղ երկրորդ օպերանդը հաջորդականություն է, օրինակ՝ ցուցակ ([List](#)):

`in` -ը ճիշտ է, եթե աջ օպերանդի հաջորդականության մեջ կա ձախ օպերանդը:

`not in` -ը ճիշտ է, եթե աջ օպերանդի հաջորդականության մեջ չկա ձախ օպերանդը:

## Ինքնության օպերատորներ

Համակարգչի հիշողության մեջ յուրաքանչյուր օբյեկտին Python- ի ինտերպրետատորի կողմից տրվում է եզակի նույնականացման համար (ID): Ինքնության օպերատորները համեմատում են երկու օբյեկտի նույնականացման համարները: Դրանք երկուսն են is և is not :

is -ը ճիշտ է, եթե երկու օպերանդներն ունեն նույն համարը:  
is not -ը ճիշտ է, եթե երկու օպերանդների նույնականացման համարները տարբեր են: Օրինակ՝

```
x = 10
y = 10
print(x is y)
#True
```

Ելքը կլինի True : Բայց եթե փոփոխականները լինեն ցուցակներ այդ դեպքում ելքը կլինի False : Դա այն պատճառով է, որ ինտերպրետատորը հիշողության մեջ առանձնացնում է այդ տիպերը և տալիս տարբեր նույնականացման համարներ:

```
x = [1,2,3]
y = [1,2,3]
print(x is y)
# False
```

## Գլուխ III տվյալների տիպերը Python լեզվում

Այս գլխում մենք կանդրադառնանք Python -ի հիմնական տվյալների տիպերին, մասնավորապես՝ ամբողջ թիվ, սահող կետով թիվ, տող: Նաև կուսումնասիրենք տիպերի ձուլումը, կքննարկենք տվյալների երեք այլ առաջադեմ տիպերը: Ցուցակ, կարգավորված և անփոփոխ ցուցակ և բառարանային տիպեր:

### Ամբողջ թվեր (Integers)

Ամբողջ թվերը այն թվերն են, որոնք չեն պարունակում տասնորդական մաս, օրինակ՝ -4, -3, 0, 7, 9 և այլն:

Օրինակ հայտարարենք մի ամբողջ թվով փոփոխական հետևյալ կերպ: `temperature = 27`

### Սահող կետով թվեր (Float)

Սահող կետով թվերը այն թվերն են, որոնք պարունակում են տասնորդական մաս, օրինակ՝ 1.234, -0.023, 3.14 և այլն:

Օրինակ հայտարարենք մի սահող կետով թվի փոփոխական հետևյալ կերպ: `weight = 60.7`

## Տողեր (Strings)

Python- ում տող ասելով կարող ենք հասկանալ տեքստ ([text](#)), որը հնարավոր է հայատարարել երկու ձև, օրինակ՝

`var_name = ' Initial '` և `var_name = " Initial "` : Այս երկու ձևի միջև էական տարբերություն չկա:

Տողերի համար կարող ենք կիրառել կոնկատենացիա կամ այլ կերպ ասած միավորման գործողությունը, օրինակ՝

```
part_one = " He "  
part_two = " llo "  
con_str = part_one + part_two  
print (con_str)
```

# or we can run

```
con_str = "He" + "llo"  
print (con_str)
```

Տողերի կոնկատենացիան կատարվում է + նշանով:

Python- ում կան մշակված ֆունկցիաներ, որոնք կատարում են որոշակի գործողություններ տողերի հետ, օրինակ՝ `upper()` ֆունկցիան, որը տողի բոլոր տառերը դարձնում է մեծատառ, բայց այսպիսի ֆունկցիաներ կոստումնասիրենք հետագայում:

Տողերում հաճախ կատարվում է ձևաչափման ([Formatting](#)) գործողությունը, այն մեզ տալիս է լավ միջոց տողերի կառավարման համար, և դրանով մենք կարող ենք որոշել թե ինչ տեսք ունենա տողը ցուցադրվելիս:

Python -ում կա երկու հիմնական միջոց տողերի ձևաչափման համար: Առաջինը դա % օպերատորն է տողերում: Այն ունի հետևյալ գրելաձևը:

“ տողը, որը պետք է ձևաչափել ” %( արժեքներ կամ փոփոխականներ, որոնք պետք է տեղադրվեն տողի մեջ, բաժանված ստորակետերով ըստ հերթականության )

Սկզբում “ ” սիմվոլների մեջ գրում ենք այն տողը, որը պետք է ձևաչափել, հետո գրում ենք %( ) սիմվոլները և փակագծերում գրում ենք այն արժեքները կամ փոփոխականները, որոնք պետք է տեղադրվեն տողում: Իրարից տարանջատում, (ստորակետ) սիմվոլով ըստ հերթականության:

Դիտարկենք դրա կիրառությունը օրինակի վրա կատարելով հետևյալ կոդը:

```
mount = 'Ararat'
height = 5165
text = "Mount %s is %d meters high." %(mount, height)
print(text)
```

Արդյունքում կունենանք հետևյալ ելքը:

Mount Ararat is 5165 meters high.

Եթե ցանկանում եք տեղադրել փոփոխականը բացատով, ապա կարող եք տեղադրման տեղում % -ի և նրա տիպը որոշող սիմվոլի միջև դնել ըստ անհրաժեշտության քանակի բացատ: Օրինակ՝

կողի երրորդ տողում ավելացնենք բացատները, կունենանք հետևյալ տեսքը:

```
text = "Mount % s is % d meters high." %(mount, height)
```

Որից հետո փոփոխականը կտեղադրվի արդեն տրված քանակի բացատով:

Այս աղյուսակում նշված է սիմվոլներին համապատասխան ակնկալվող տիպեր: Եթե տեղադրվող տվյալի տիպը չհամապատասխանի ակնկալվող տիպին, ապա ծրագիրը չի կատարվի և կհաղորդի սխալի(*Error*) մասին:

Սիմվոլ	Ակնկալվող տիպ
%s	Տող ( <i>string</i> )
%d	Ամբողջ թիվ ( <i>integer</i> )
%f	Սահող կետով թիվ ( <i>float</i> )

Նաև հնարավոր է որոշել թե որքան չափ զբաղեցնի փոփոխականը նշված տեղում: Օրինակ՝

```
pi = 3.141592653589  
text = "The pi number is approximately equal %3.2f " %(pi)  
print(text)
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Որտեղ 3 -ը նշանակում է, որ ամբողջ մասը կարող է լինել մինչև երեք սիմվոլ իս կետից հետո 2-ը այն, որ տասնորդական մասը կարող է լինել մինչև երկու սիմվոլ: Արդյունքում կունենանք հետևյալ ելքը:

```
The pi number is approximately equal 3.14
```

Երկրորդ միջոցը, որով կարող եք տողերը ձևաչափել դա `format()` մեթոդի կիրառումն է: Այն ունի հետևյալ գրելաձևը:

“ տողը, որը պետք է ձևաչափել ” .`format()` արժեքներ կամ փոփոխականներ, որոնք պետք է տեղադրվեն տողի մեջ, բաժանված ստորակետերով ըստ հերթականության )

Երբ օգտագործում ենք `format()` մեթոդը այդ ժամանակ ձևաչափվող տողում `%(տիպի սիմվոլ)` - ի փոխարեն գրում ենք ձևավոր փակագծեր, որի մեջ առաջին տեղում գրվում է արժեքի դիրքը `format()` մեթոդի պարամետրերի դաշտում հետո : սիմվոլը ու ակնկալվող տիպը: Օրինակ՝ `{0 : 3.2f }`

Առաջին օրինակի կողք `format()` մեթոդով գրելու դեպքում կունենա հետևյալ տեսքը:

```
mount = 'Ararat'
height = 5165
text = "Mount {0 : s} is {1 : d} meters high." .format(mount,
```



Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

```
height)  
print(text)
```

Կարևոր է իմանալ, որ գրեթե բոլոր ծրագրավորման  
լեզուներում դիրքի հաշվարկը սկսվում է 0 թվանշանից:

Այդպես է նաև

Python -ում, ուստի այս կողում առաջին փոփոխականի դիրքը  
նշված է 0, որը համապատասխանում է `format()` մեթոդի 0-  
ական դիրքում գտնվող պարամետրին:

## Տիպերի ձևեր

Երբեմն ծրագրում անհրաժեշտ է լինում մի տիպի տվյալը  
ձևափոխել մեկ այլ տիպի, օրինակի համար ամբողջ թիվը  
դարձնել տող: Դրա համար Python լեզուն ունի երեք  
նախատեսված ֆունկցիաներ, որոնք կարող են դա  
ապահովել: Դրանք են `int()`, `float()`, `str()`,  
համապատասխանաբար ամբողջ թիվ, սահող կետով թիվ և  
տեղ: Յուրաքանչյուրը այս երեք ֆունկցիաներից որպես  
պարամետր ստանում է մնացած երկու տիպերից որևէ մեկը  
և ձևափոխում այն իրեն համապատասխանող տիպին:  
Օրինակ՝ `float('2')` կամ `float(2)` -ը կտա `2.0` արդյունքը:

## Ցուցակներ

Python -ում ցուցակը կարող ենք հասկանալ որպես տվյալների  
հավաքածու, օրինակ՝ եթե մեզ անհրաժեշտ է պահել հինգ  
օգտագործողի ([User](#)) անուն, ապա հինգ առանձին

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

փոփոխականի Փոխարեն մենք կարող ենք դրանք պահել  
որպես ցուցակ:

Ցուցակ հայտարարում ենք հետևյալ գրելաձևով:

Ցուցակի անուն = [ սկզբնական արժեքներ, որոնք միմիանցից  
բաժանված են ստորակետով ]

Օրինակ՝ `user_age = [21, 22, 23, 24, 25]` :

Ցուցակի տարրերը կարող են լինել նաև տարբեր տիպերի:  
`my_list = [21, "Ararat ", 3.14]` :

Մենք կարող ենք նաև հայտարարել ցուցակ առանց  
նախապես արժեքներ տալու: Պարզապես գրելով ցուցակի  
անուն = [] կունենանք դատարկ ցուցակ, որտեղ արժեքներ  
կարող ենք լցնել օգտագործելով `append()` ֆունկցիան:  
Ցուցակի տարրերին կարող ենք դիմել (փոփոխել) իրենց  
ինդեքսներով ([index](#)), որոնք համարակալվում են 0 -ից:  
`user_age[0] = 21`, իսկ բացասական ինդեքսներ տալու դեպքում  
կդիտարկի ցուցակի տարրերը վերջից, օրինակ:  
`user_age[-1] = 25`, `user_age[-2] = 24` և այդպես շարունակ:

Դուք կարող եք ցուցակը կամ դրա մի մասը վերագրել այլ  
փոփոխականի: Օրինակ՝

```
user_age_two = user_age[2:4]
# user_age_two == [23, 24]
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

ապա `user_age_two = [23, 24]` : Երբ վերագրում ենք ինչ-որ կտոր մի ցուցակից մեկ այլ փոփոխականի ինդեքսով, ապա Python լեզվում սկզբի ինդեքսի տարրը չի ներառվում, իսկ վերջի ինդեքսի տարրը ներառվում է:

Տարրի ավելացումը ցուցակում `append()` ֆունկցիայի միջոցով:

```
user_age_two.append(26)
# user_age_two == [23, 24, 26]
```

`user_age_two` կլինի `[23, 24, 26]` :

Տարրի հեռացումը ցուցակից `del` հրամանի միջոցով:

```
del user_age_two[1]

# user_age_two == [23, 26]
```

`user_age_two` կլինի `[23, 26]` :

Տարրի հեռացումը ցուցակից `remove()` ֆունկցիայի միջոցով:

```
user_age_two.remove(26)
# user_age_two == [23]
```

`user_age_two` կլինի `[23]` :

## **Անփոփոխ Ցուցակներ**

Այս տիպերը ցուցակներ են, որոնց արժեքները փոփոխել հնարավոր չէ: Դրանց Python -ում կարճ անվանում են Tuple :

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Կարող եք օգտագործել այն ժամանակ, երբ ձեր ծրագիրը պետք է պահի տարվա ամիսների անվանումները:

Tuple հայտարարում ենք հետևյալ գրելաձևով:

Tuple -ի անունը = (սկզբնական արժեքներ, որոնք միմիանցից բաժանված են ստորակետով )

Օրինակ՝

```
months_of_year = ("Jan", "Feb", "Mar", "Apr", "May",  
"Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
```

Tuple -ի տարրերին դիմում ենք նույնպես ինդեքսներով:

## **Բառարանային տիպեր (Dictionary)**

Բառարանը հարակից տվյալների զույգերի հավաքածու է: Օրինակ, եթե ուզում ենք հինգ օգտագործողի անունը և տարիքը պահել մենք կարող ենք դրանք պահել բառարանային տիպով:

Բառարանային տիպի փոփոխական հայտարարելու գրելաձևը հետևյալն է:

Բառարանի անունը = { բառարանի բանալի : տվյալ }

Որտեղ բառարանի բանալի -ն պետք է լինի եզակի (միակը

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

այդ բառարանի մեջ) օրինակ՝

```
Arm_mount = {"Ararat": 5165, "Aragats": 4090, "Ararat": 5165}
```

բառարան չենք կարող հայտարարել, քանի որ "Ararat" -ը որպես բանալի է և կրկնվում է: Բառարանում զույգերը միմիանցից բաժանվում են ստորակետով:

Օրինակ՝ այսպիսի բառարան կարող ենք հայտարարել:

```
Arm_mount = {"Ararat": 5165, "Aragats": 4090, "Suphan": 4058 }
```

Բառարան կարելի էք հայտարարել նաև օգտագործելով dict() մեթոդը:

```
Arm_mount = dict( Ararat = 5165, Aragats = 4090, Suphan = 4058, Sis = "Not Available")
```

Ուշադրություն դարձրեք, երբ օգտագործում ենք dict() մեթոդը բառարանի բանալիները չեն գրվում " " սիմվոլների մեջ:

Բառարանի տարրերին դիմալու համար օգտագործում ենք բառարանի բանալիները, որոնք զույգերի առաջին արժեքներն են:

Եթե պետք է իմանալ John -ի տարիքը ապա գրում ենք  
`Arm_mount["Ararat"]`

և կստանանք 5165 արժեքը: Նույն կերպ կարող ենք փոփոխել բառարանում այդ բանալին ունեցող զույգի տվյալը (արժեքը):

Փոխենք Sis -ի տարիքը 3925, դա կգրվի հետևյալ կերպ:  
`Arm_mount["Sis"] = 3925`

Որից հետո բառարանը կունենա այս տեսքը

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

```
# Arm_mount == {"Ararat": 5165, "Aragats": 4090, "Suphan": 4058, "Sis":  
3925 }
```

Բառարանը նույնպես կարող ենք հայտարարել դատարկ, այսինքն սկզբնական ոչ մի արժեք նրան չտանք: Հետո բառարանում տարրեր կարող ենք ավելացնել:

Ավելացնենք “Arnos”: 3550 գույգը Arm\_mount բառարանում, այսպես Arm\_mount [“Arnos”] = 3550

Հիմա բառարանը պարունակում է հետևյալ տարրերը:

```
# Arm_mount == {"Ararat": 5165, "Aragats": 4090, "Suphan": 4058, "Sis":  
3925, "Arnos": 3550 }
```

Բառարանից տարր կարող ենք ջնջել del հրամանով հետևյալ կերպ: del Arm\_mount [“Arnos”]

Այսպիսով վերջնական պարունակում է հետևյալ տարրերը:

```
# Arm_mount == {"Ararat": 5165, "Aragats": 4090, "Suphan": 4058, "Sis":  
3925 }
```

## **Գլուխ IV տվյալների մուտք և ելք**

Մուտքը և ելքը տերմինաբանությունն է, որը վերաբերում է համակարգչային ծրագրի և դրա օգտագործողի միջև հաղորդակցությանը: Մուտքագրողը, օգտվողն է, որը ինչ-որ բան (տվյալ) է տալիս ծրագրին, իսկ ելքը՝ օգտագործողին ինչ-որ բան տվող ծրագիրը:

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Այժմ, երբ մենք ծանոթ ենք փոփոխական հասկացողությանը և ունենք գրված մեր առաջին “Hello World” ծրագիրը Python -ով, ապա այս գլխում մենք կվերադառնանք մեր գրած ծրագրին և կփորձենք այն դարձնել ինտերակտիվ, մուտք և ելք հսկացողությունների միջոցով:

Այս ամենը կկատարենք Python -ում ներկառուցված երկու շատ կարևոր ֆունկցիաների մեջոցով, դրանք են `input()` և `print()` :

Դիտարկենք օրինակի վրա:

```
my_name = input("Please enter your name: ")
my_age = input("Please enter your age: ")
print ("Hello World, my name is", my_name, "and I am",
my_age, "years old.")
```

Այս ծրագիրը կատարելուց մենք կստանանք հաղորդագրություն `Please enter your name:` և սպասի մուտքի: Ենթադրենք մուտքագրել ենք `Ararat` : Դրանից հետո կստանանք նոր հաղորդագրություն, որտեղ կասվի `Please enter your age:` ենթադրենք մուտքագրել եք `22` : Ծրագրի աշխատանքի ավարտին մենք կտեսնենք հետևյալ տքստը `Hello World, my name is Ararat and I am 22 years old.` :

## Տվյալների մուտք, **Input()**

Վերևում գրված ծրագրի մեջ մենք օգտագործեցինք `input()` ֆունկցիան ստանալու համար անուն և տարիք:

```
my_name = input("Please enter your name: ")
```

Այն աշխատում է հետևյալ կերպ իր պարամետրում գրված տողը դուրս է բերում էկրանին, որտեղ էլ նախատեսված է գրել թե ինչ է սպասվում օգտագործողի կողմից մուտք անելու համար: Այս ֆունկցիայի աշխատանքի արդյունքում արժեքները որը պահել ենք `my_name` և `my_age` փոփոխականներում տողային տիպի են:

## Տվյալների ելք, **Print()**

`Print ()` ֆունկցիան օգտագործվում է օգտվողներին տեղեկատվություն ցուցադրելու համար: Այն ընդունում է զրոյական կամ ավելի արտահայտություններ որպես պարամետրեր՝ բաժանված ստորակետերով: Վերևում գրված ծրագրի մեջ այս ֆունկցիան օգտագործել ենք 5 պարամետրով:

```
print ("Hello World, my name is", my_name, "and I am", my_age, "years old.")
```

Նկատենք, որ փոփոխականների անունները չենք դնում “ ” սիմվոլների մեջ: Մեկ այլ ձև, որով կարող ենք տպել այս տեքստը փոփոխականներով կլինեն տողերի ձևաչափման



Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

%( )

օպերատորի միջոցով և այն կունենար հետևյալ տեսքը:

```
print ("Hello World, my name is %s and I am %s years old." %  
(my_name, my_age))
```

Կամ կարող ենք օգտագործել `format()` մեթոդը ըստ  
ցանկության:

## **Հատուկ սիմվոլների արտածում**

Երբեմն անհրաժեշտ է լինում տպել հատուկ սիմվոլներ,  
օրինակ՝

“ \n ” - նոր տող անցնելու սիմվոլը: Այս դեպքում անհրաժեշտ է  
օգտագործել \ սիմվոլը:

Օրինակի համար, եթե գրենք այս հրամանը `print ("Hello \n  
World")`

կունենանք այսպիսի ելք

```
Hello  
World
```

իսկ եթե մեզ պետք է արտածել նաև հատուկ սիմվոլը ապա  
կավելացնենք \ -ը ևս մեկ անգամ այն հատուկ սիմվոլի  
սկզբում, որը պետք է արտածել:

```
print ("Hello \n World")  
# Hello \n World
```

Python -ում հատուկ սիմվոլը դա \ -ն է և այն արտածելու

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

համար մեզ պետք է օգտագործել հենց այդ սիմվոլը:

```
print (" \ " )  
# \
```

## Առաջադրանքներ

Գիտելիքները ամրապնդելու համար առաջարկվում է լուծել հետևյալ խնդիրները:

- 1) Գրեք ծրագիր, որն օգտագործողից ընդունում է շրջանագծի շառավիղը և հաշվարկում մակերեսը ու ցույց է տալիս ելքում:
- 2) Գրեք ծրագիր, որն ընդունում է օգտագործողի անունն ու ազգանունը և դրանք ելքում տալով հակառակ հերթականությամբ:
- 3) Գրեք ծրագիր, որն օգտագործողից ընդունում է երկու թիվ, որից հետո այդ փոփոխականների արժեքները տեղերով փոխում: (օգտագործել երրորդ փոփոխական)
- 4) Գրեք ծրագիր, որն հետևյալ ցուցակից կտալի առաջին և վերջին տարրը: `color_list = ["Red","Green","White","Black"]`
- 5) Գրեք ծրագիր, որն ընդունում է ամբողջ թիվ ( $n$ ) և հաշվարկում է  $n + nn + nnn$  արժեքը:
- 6) Գրեք ծրագիր՝ 6 շառավղով գնդի ծավալը ստանալու համար: Գնդի ծավալի հշվման բանաձևը հետևյալն է

$$\frac{4}{3}\pi r^3$$

- 7) Գրեք ծրագիր, որն օգտագործողից ընդունում է եռանկյան երեք կողմերի չափերը (a,b,c) և հաշվում պարագիծ ու մակերես հետո տպում ելքում:
- 8) Գրեք ծրագիր լուծելու համար  $(x + y) * (x + y)$  արտահայտությունը: Օրինակ տեղադրելով  $x = 4, y = 3$  ելքում պետք է տպի 49 արժեքը:
- 9) Գրեք ծրագիր՝  $(x1, y1)$  և  $(x2, y2)$  կետերի միջև հեռավորությունը հաշվարկելու համար: Կետերը կարող է մուտքագրել օգտագործողը:
- 10) Գրեք ծրագիր, որը բարձրությունը դյույմներով կդարձնի սանտիմետր: Բարձրությունը կներմուծի օգտագործողը, կարող եք օգտվել նրանից, որ 1 դյույմ = 0,0254 մետր:
- 11) Գրեք ծրագիր՝ տրված ցուցակից առաջին տարրը հեռացնելու համար:
- 12) Գրեք ծրագիր՝ տրված երեք թվերի միջին թվաբանականը գտնելու համար:

## **Գլուխ V պայմանական կառավարման կառուցվածքներ, ցիկլեր, բացառություններ**

Այս գլխում կձանոթանանք այնպիսի կառուցվածքների և հրամանների որոնք մեր ծրագիրը կդարձնեն ավելի խելացի: Կձանոթանանք ընտրության կատարելու կառուցվածքների, ցիկլերի ու դրանց ընդհատման հրամաններին, ինչպես նաև բացառություններ մշակող գործիքների: Նախքան սկսելը հիշենք այս գրքի սկզբում ծանոթացել ենք [Համեմատման օպերատորներին](#), որոնք էլ հաճախ կրառվում են ընտրության կատարող կառուցվածքներում:

## If կառուցվածքը

If-ը ամենատարածված ու հաճախ օգտագործվող կառավարման կառուցվածքներից է: Այն թույլ է տալիս ծրագրին գնահատել որոշակի պայմանի բավարարման առկայությունը, և կատարել համապատասխան գործողություն՝ ելնելով արդյունքից: If -ի կառուցվածքը հետևյալն է:

if պայման 1 :

    կատարել ինչ-որ Ա գործողություն

elif պայման 2 :

    կատարել ինչ-որ Բ գործողություն

elif պայման 3 :

    կատարել ինչ-որ Գ գործողություն

elif պայման 4 :

    կատարել ինչ-որ Դ գործողություն

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

else:

կատարել ինչ-որ այլ գործողություն

Շատ այլ ծրագրավորման լեզուներում ինչպիսիք են C, C++ և այլն If -ը և դրա նման կառուցվածքները ունենում են կատարելու համար կադի չափը սահմանող բլոկեր նշված { ... } (ձևավոր) փակագծերում, բայց Python -ում այդպես չէ: Որպեսզի ավելի լավ պատկերացնանք այս կառուցվածքի աշխատանքը դիտարկենք այն օրինակի վրա:

```
user_input = input('Enter 1 or 2: ')
```

```
if user_input == "1" :
```

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

```
    print ("How are you?")
```

```
elif user_input == "2" :
```

```
    print ("Python Rocks!")
```

```
    print ("I love Python")
```

```
else:
```

```
    print ("You did not enter a valid number")
```

Սկզբում ծրագիրը օգտվողին կպահանջի ներմուծել 1 կամ 2, որից հետո կանցնի ստուգման , եթե (if) օգտվողը ներմուծել է 1, ապա կտալի Hello World How are you? տեքստը այլապես , եթե (elif) ներմուծել է 2, ապա կտալի Python Rocks! I love Python և հակառակ դեպքում (else) You did not enter a valid

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

number :

## Inline If կառուցվածքը

Այս կառուցվածքը նման է if -ին մի տարբերությամբ սա ավելի կարճ է և նախատեսված է երկու տարբեր գործողությունների համար, որի կառուցվածքը հետևյալն է:  
կատարել և գործողություն if պայման else այլ գործողություն

Օրինակ այն կարելի է կիրառել այսպես:

```
num1 = 12 if my_int == 10 else 13
```

այս օրինակում num1 փոփոխականին կվերագրվի 12 , եթե my\_int հավասար է 10 -ի հակառակ դեպքում կվերագրվի 13 :

Մեկ այլ օրինակ դիտարկենք:

```
print ("This is task A" if my_int == 10 else "This istask B")
```

եթե my\_int հավասար լինի 10 ապա կտպի This is task A տեքստը  
հակառակ դեպքում This istask B :

## For ցիկլը

For ցիկլը կատարում է կոդի բլոկը այնքան անգամ, մինչև հայտարարության մեջ նշված պայմանն այլևս տեղի չի ունենում:

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Այս ցիկլը կարող ենք օգտագործել իտերացվող ([iterable](#))  
տվյալների վրա:

Python -ում իտերացվող տվյալներ են համարվում տողերը,  
ցուցակները, tuple -երը : For ցիկլի տեսքը հետևյալն է:

```
for ցիկլի փոփոխականի անուն in իտերացվող տիպ :  
    կատարել ինչ-որ գործողություն  
# print ( ցիկլի փոփոխականի անուն )
```

Օրինակ՝

```
pets = ['cats', 'dogs', 'rabbits', 'hamsters']  
  
for my_pets in pets:  
    print (my_pets)
```

Այս օրինակի սկզբում մենք հայտարարել ենք `pets` անունով  
ցուցակ, որից հետո ցիկլի իտերացիայով անցնում ենք  
ցուցակի յուրաքանչյուր տարրով: Այսպիսով երբ ծրագիրը  
սկսի կատարել ցիկլը, ապա տեղի կունենա հետևյալ  
գործընթացը:

Առաջին իտերացիայի ժամանակ `my_pets` -ին կվերագրվի  
ցուցակի առաջին տարրը ('cats') և կտպի այն: Երկրորդ  
իտերացիայի ժամանակ կվերագրվի երկրորդ տարրը ու  
կտպի այն և այսպես շարունակ մինչև ցուցակի տարրերը  
ավարտվեն: Արդյունքում կունենանք հետևյալը:

```
cats  
dogs  
rabbits  
hamsters
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Կարող ենք նաև ցույց տալ տարրերի ինդեքսները  
օգտագործելով `enumerate()` ֆունկցիան:

```
for index, my_pets in enumerate(pets):  
    print (index, my_pets)
```

```
# 0 cats  
# 1 dogs  
# 2 rabbits  
# 3 hamster
```

Կամ ցիկլով անցնել տողի սիմվոլների վրայով:

```
message = "Hello"
```

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

```
# H  
# e  
# l  
# l  
# o
```

Հիմա տեսնենք, թե ինչպես կարելի է `For` ցիկլի միջոցով անցնել թվերի հաջորդականության վրայով, դրա համար կօգտագործենք `range()` ֆունկցիան: Սա ներկառուցված ֆունկցիա է Python -ում այն գեներացնում է թվերի հաջորդականության ցուցակ, որի գրելաձևը հետևյալն է: `range` (սկիզբ, վերջ, քայլ), որտեղ սկիզբ և վերջ պարամետրերում նշվում է այն միջակայքը, որտեղից պետք է



գեներացվեն թվերը, իսկ քայլ պարամետրում նշվում է թե ինչ չափով պետք է հաջորդ թիվը տարբերվի նախորդից: Եթե սկիզբ պարամետրը նշված չէ, ապա այն լռելյայն ընդունում է 0 արժեք, բացի այդ այս պարամետրը միշտ խորհուրդ է տրվում տալ 0, քանի որ տվյալների տիպերում ինդեքսները սկսվում են համարակալել զրոից: Եթե քայլ պարամետրը չենք նշում այն լռելյայն ընդունում է 1 արժեքը: Կա մի տարօրինակ նրբություն կապված այս ֆունկցիայի հետ, այն է երբ չենք նշում սկիզբ և քայլ պարամետրերը, ապա ֆունկցիայում նշվում է լինում միայն վերջ պարամետրը ու այն գեներացնում է 0 -ից մինչև նշված արժեքը ցուցակ, ընդ որում, այդ արժեքը չի նարառվում ցուցակի մեջ:

## Օրինակներ՝

```
range(5)  
# [0, 1, 2, 3, 4]
```

```
range(4,8)  
# [4,5,6,7]
```

```
range(4,10,2)  
# [4,6,8]
```

Թե ինչպես օգտագործենք `range()` ֆունկցիան For ցիկլի հետ, դրա համար դիտարկենք այս օրինակը:

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

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

```
# 0
```

```
# 1
```

```
# 2
```

```
# 3
```

```
# 4
```

## While ցիկլը

While ցիկլը կատարում է կոդի բլոկը այնքան անգամ, մինչև հայտարարության մեջ նշված պայմանը տեղի ունի (ճիշտ է) : Ցիկլի կառուցվածքը ունի հետևյալ տեսքը:

```
while պայմանը ճիշտ է :
```

```
    կատարել ինչ-որ գործողություն
```

Շատ հաճախ երբ օգտագործում ենք While ցիկլը մեզ պետք է նախապես հայտարարել փոփոխական ցիկլը կառավարելու համար և պայմանի դաշտում փոփոխականի արժեքը, որի արժեքով էլ պայմանավորված է ցիկլի աշխատանքը: Դիտարկենք օրինակի վրա:

```
counter = 5
```

```
while counter > 0 :
```

```
    print ("Counter = ", counter)
```

```
    counter = counter -1
```

```
# Counter =5
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

```
# Counter =4  
# Counter =3  
# Counter =2  
# Counter =1
```

Այս ցիկլի հետ աշխատելիս, պետք է առանձնակի ուշադրություն դարձնել դրա կառավարմանը, որպեսզի չստեղծեք անցանկալի անվերջ ցիկլ: Այս կոդում ցիկլի կառավարումը ապահովում է `counter` փոփոխականը, որին սկզբից վերագրված է 5 արժեքը, իսկ ցիկլի յուրաքանչյուր իտերացիայի ժամանակ դրա արժեքը փոքրանում է 1 -ով, և ինչ-որ պահից սկսված երբ փոփոխականի արժեքը կդառնա 0, ապա ցիկլը կդադարի գործել:

## Break հրամանը

Հաճախ, երբ օգտագործում ենք ցիկլերը, անհրաժեշտություն է առաջանում դրանք ընդհատելու ինչ-որ պայմանից ելնելով: Դրա համար նախատեսված է Python -ում `break` հրամանը

Օրինակ՝

```
for l in range(5) :  
    print(l)  
    if l == 2 :  
        break
```

Կունենանք հետևյալ ելքը:

```
# 0
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

# 1

# 2

Քանի որ ցիկլի յուրաքանչյուր իտերացիայի ժամանակ ստուգվում է, եթե ցիկլի փոփոխականը հավասար է 2, ապա ընդհատել: Այդ պատճառով էլ, երբ փոփոխականը դառնում է 2, ցիկլը ընդհատվում է, և ցուցակի մնացած տարրերը չեն տպվում: Նկատենք նաև, որ այս օրինակում օգտագործել ենք For ցիկլի մեջ If -ը, կարող է լինել նաև If -ի մեջ While կամ հակառակը, այս տիպի կառուցվածքները անվանում են տեղադրված (ներդրված) կառավարման կառուցվածքներ:

## Continue հրամանը

Մեկ այլ օգտակար հրաման ցիկլերի համար, դա continue հրամանն է: Այն օգտագործվում է, երբ ցիկլի որևէ իտերացիայի ժամանակ ինչ-որ պայմանից ելնելով անհրաժեշտ է այդ իտերացիան (գործողությունը) բաց թողնել և անցնել ցիկլի հաջորդ իտերացիային:

Դիտարկենք հետևյալ կոդը:

```
for l in range(5) :  
    if l == 3 : continue  
    print('l=',l)
```

Այս կոդը կատարելուց ծրագիրը կաշխատի հետևյալ կերպ, ցիկլով կանցնի 0 -ից 5 միջակայքի ամբողջ թվերի վրայով (բացառությամբ 5 -ի) և յուրաքանչյուր իտերացիայի

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Ժամանակ կստուգի, եթե ցիկլի փոփոխականը հավասար է 3-ի, ապա բաց կթողնի կատարվելիք գործողությունը (`print('I=',I)`) և կանցնի հաջորդ իտերացիային:

## Բացառությունների մշակում **Try, Except**

Այս դեկլարման կառուցվածքը նախատեսված է կառավարելու ծրագրի ընթացքը, երբ տեղի է ունենում սխալ, կամ բացառություն: Այլ կերպ ասած բացառությունը այն դեպքն է, երբ ծրագրի կատարման ընթացքում տեղի է ունենում մի իրադարձություն, որը խանգարում է ծրագրի բնականոն աշխատանքին: Կառուցվածքի գրելաձևը հետևյալն է:

```
try :  
    անել ինչ-որ բան  
except :  
    անել ինչ-որ այլ բան երբ սխալ է տիղի ունեցել
```

Ավելի պարզ կլինի սա դիտարկել օրինակի վրա:

```
try :  
    result = 8/0  
    print(result)  
except :  
    print("Error: can't the number divide to zero")
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Երբ կատարեք ծրագիրը ելքում կտեսնենք հետևյալ տեքստը  
Error: can't the number divide to zero, քանի որ սկզբում ծրագիրը  
փորձում է կատարել `result = 8/0` հրամանը, բաց չի ստացվում  
թիվը բաժանել զրոյի, ուստի ծրագիրը անտեսում է `try` : բլոկի  
կոդը և կատարում `except` : -ում գրվածը:

Եթե ցանկանում եք ավելի կոնկրետ սխալի  
հաղորդագրություններ ցույց տալ ձեր օգտվողներին  
կախված սխալից, ապա `except` : բանալի բառից հետո կարող  
եք նշել սխալի տեսակը: Հիմնական տեսակները, որոնք  
հաճախ են օգտագործվում հետևյալներն են:

`ZeroDivisionError` : Այս տեսակի բացառությունը կատարվում է,  
երբ բաժանման կամ մոդուլի գործողության երկրորդ  
օպերանդը զրո է:

`ValueError` : Կատարվում է, երբ գործողությունը կամ  
գործառույթը ստանում է ճիշտ տիպի, բայց ոչ հավանական  
արժեք ունեցող արգումենտը: Օրինակ՝ օգտատերը  
մուտքագրեց մի տող, որը հնարավոր չէ ձևափոխել ամբողջ  
թիվի:

`IOError` : Կատարվում է, երբ տեղի է ունենում մուտքի և ելքի  
գործողության ձախողում: Օրինակ՝ փորձվում է  
ներկառուցված `open()` ֆունկցիայով բացել ինչ-որ ֆայլ, բայց  
ֆայլը չի գտնվել

`ImportError` : Կատարվում է, երբ ներմուծման հրամանը չի

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

գտնում մոդուլի սահմանումը (մոդուլների մասին  
կտեղեկանանք հաջորդ գլխում) :

**IndexError** : Կատարվում է, երբ հաջորդականության (տող,  
ցուցակ) ինդեքսը (ցուցիչը) տիրույթից դուրս է:

**KeyError** : Կատարվում է, երբ բառարանային տիպի բանալին  
չի գտնվել:

**NameError** : Կատարվում է, երբ լոկալ կամ գլոբալ  
փոփոխականի կամ տվյալի անուն չի գտնվել:

**TypeError** : Կատարվում է, երբ գործողություն է կատարվում  
այն օբյեկտների հետ, որոնց համար այդ գործողությունը  
սահմանված չէ: Օրինակ՝ + օպերատորը կիրառելուց տողի և  
ամբողջ թվի մեջև:

Այս տեսակներից օգտվելով կարելի է մշակել բացառություն  
այն դեպքի համար երբ օգտագործողը մուտքագրի ոչ թե  
պահանջված արժեքը այլ ուրիշ արժեք: Դա կարելի է  
կատարել հետևյալ կերպ:

```
try :  
    age = int(input("Enter your age. \n"))  
  
except ValueError :  
    Print("Error: you must enter a number.")
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Երբ կատարենք այս ծրագիրը և մուտքում թվի փոխարեն տանք տառ կամ այլ սիմվոլ՝ կստանանք հետևյալ սխալի տեքստը

Error: you must enter a number.

Python լեզվում նաև թույլատրվում է վերասահմանել բացառության տեսակի անունը, օգտագործելով `as` բանալի բառը:

Այսպիսով վերևում գրված կոդը կարելի է ձևափոխել այսպես:

try :

```
    age = int(input("Enter your age. \n"))
except ValueError as v:
    print(v)
```

Հնարավոր է նաև դիտարկել այնպիսի բացառություններ, որոնք նախատեսված չեն լեզվում, բայց ինչ-որ պատճառներից ելնելով տեղի են ունեցել: Դրանց տեսակը նշվում է պարզապես `Exception` (բացառություն), դա կարող ենք դիտարկել որպես վերջին բացառություն, նաև ընդունված է անունը սահմանել `e`, բայց պարտադիր չէ: Եթե այդ բացառությունը ավելացնենք վերևի կոդում, ապա այն կունենա հետևյալ տեսքը:

try :

```
    age = int(input("Enter your age. \n"))

except ValueError as v:
    print(v)
```



Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

```
except Exception as e :  
    print("Unknown error: ", e)
```

Կարող եք օգտագործել `else` բանալի բառը՝ սահմանելու համար կողի բլոկ, որը պետք է կատարվի, եթե սխալներ չեն առաջացել:

```
try :  
    age = int(input("Enter your age. \n"))  
except ValueError as v:  
    print(v)  
except Exception as e :  
    print("Unknown error: ", e)  
else :  
    print("Nothing went wrong. Your age is entered ", age)
```

Կամ կարող եք նշել `finally` : բլոկը, որն կկատարվի անկախ նրանից, `try` : բլոկի կողմից սխալ առաջացրել է, թե ոչ:

```
try :  
    age = int(input("Enter your age. \n"))  
except ValueError as v:  
    print(v)  
except Exception as e :  
    print("Unknown error: ", e)
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

```
else :  
    print("Nothing went wrong. Your age is entered ", age)  
  
finally :  
    print("Checking is finished, if something gone wrong then you  
        already had seen error text, otherwise you seen  
        your age.")
```

Python լեզուն հնարավորություն է տալիս անհրաժեշտության դեպքում, կամ ելնելով ինչ-որ պայմաններից ծրագրավորողի մակարդակում գցել ([throw](#)) բացառություններ, օգտագործելով `raise` բանալի բառը: Դիտարկենք օրինակի վրա:

```
x = -1  
  
if x < 0:  
    raise Exception("Error: A numbers below zero not  
    accepted.")
```

## Առաջադրանքներ

Առաջարկվում է լուծել հետևյալ խնդիրները:

- 1) Գրեք ծրագիր, որը օգտագործողից կպահանջի ներմուծել թիվ և կստուգի թե թիվը, որ միջակայքից է (0,100), [100,1000), թե՞ [1000, 2000] :

- 2) Գրեք ծրագիր, որը կհաշվի օգտագործողի կողմից ներմուծված երկու թվերի տարբերության մոդուլը:
- 3) Գրեք ծրագիր, որը կհաշվի առաջին  $n$  դրական ամբողջ թվերի գումարը:
- 4) Գրեք ծրագիր՝ ամբողջ թվի թվանշանների գումարը հաշվելու համար: Թիվը ներմուծում է օգտագործողը:
- 5) Գրեք ծրագիր, որը օգտագործողից կպահանջի ներմուծել տաս տարբեր թվեր և կգտնի նրանցից մեծագույնը ու փոքրագույնը, կտպի այն ելքում:
- 6) Գրեք ծրագիր, որը կստուգի թիվը դրական է, բացասական թե՞ զրո: Թիվը ներմուծում է օգտագործողը:
- 7) Գրեք ծրագիր, որը կհաշվի տրված տողում սինվոլների քանակը և կտպի ելքում:
- 8) Գրեք ծրագիր, որը կհաշվի թվի ֆակտորյալը: Թիվը կներմուծի օգտագործողը իսկ ծրագիրը կտպի պատասխանը ելքում:
- 9) Գրեք ծրագիր, որը օգտագործողից կպահանջ ներմուծել երկու թիվ  $x$  և  $y$ , կհաշվի  $x$ -ի  $y$  աստիճանը և կտպի ելքում:
- 10) Գրեք ծրագիր, որը կտպի մինչև 100 թիվը եղած Ֆիբոնաչիի հաջորդականություն թվերը: 0, 1, 1, 2, 3, ...,
- 11) Գրեք ծրագիր՝ գտնելու համար, տրված ամբողջ թվի բաժանարարների քանակը զույգ է թե՞ կենտ:
- 12) Գրեք ծրագիր օգտագործողի կողմից ներմուծված երկու թվերի ընդհանուր բաժանարարները գտնելու և

Ելքում դրանք տպելու համար:

- 13) Գրեք ծրագիր, որը տրված ցուցակի և տրված `n` թվի համար կասի `true` (ճիշտ է), եթե այդ ցուցակում կա այնպիսի երկու թվեր, որոնց գումարը հավասար է `n` թվին:
- 14) Գրեք ծրագիր, որը կսահմանի բառարանային տիպերեք տարրից բաղկացած: Հետո պետք է գրել կոդ, որն կփորձի տպել այնպիսի տարրի տվյալ, որի բանալին չկա սահմանված բառարանում: ԵՎ դրա համար մշակել այնպիսի բացառություն, որը կտաի “բանալին առկա չէ բառարանում” և կպահանջի օգտվողից մուտքագրել այդ բանալուն համապատասխան տվյալը:  
Բառարանում կստեղծի ստացած բանալիով ու տվյալով տարր: Վերջնական (*finally*) բլոկում գրել այնպիսի կոդ, որը ելքում կտաի բառարանի բոլոր տարրերը:

## Գլուխ VI Ֆունկցիաներ և մոդուլներ

Նախքան այս գլխին հասնելը գրքում հաճախ ենք օգտագործում ֆունկցիաներ, օրինակի համար տիպերի ծովամն ժամանակ, կամ տողերի հետ աշխատելիս: Հիմա ավելի խորը կուսումնասիրենք և կհասկանանք թե ինչ է ֆունկցիան ինչի համար է նախատեսված և ինչպես դրանց ճիշտ օգտագործենք մեր ծրագրում: Կիմանանք նաև թե ինչ

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Են իրենցից ներկայացնում մոդուլները Python լեզվում, ու ինչպես են դրանց օգտագործում:

## Ֆունկցիաներ և դրանց սահմանումը

Ֆունկցիան (**function**) նախապես գրված կոդի բլոկ է, որի իրականացվող կոդը գրվում է մեկ անգամ, բայց այդ ֆունկցիան ծրագրում կարող է օգտագործվել բազմաթիվ անգամ, որն էլ ապահովում է կոդի հեշտ ընկալումը և կրկնությունների բացառում:

Ֆունկցիան օգտագործելու համար անհրաժեշտ է կանչել այն, յուրաքանչյուր ֆունկցիայի տրվում է անուն և կանչել ասելով հասկանում ենք գրել այդ ֆունկցիայի անունը կոդի այն տեղում, որտեղ կա տվյալ ֆունկցիայի աշխատանքի անհրժեշտությունը:

Օրինակի համար մենք հաճախ օգտագործել ենք `print()` ֆունկցիան, երբ անհարաժեշտ է եղել ինչ-որ տեքստ տպել ելքում:

Մենք կարող ենք ֆունկցիա սահմանել հետևյալ կերպ:

```
def ֆունկցիայի_անուն(պարամետրեր) :  
    այն_կոդը_որը_պետք_է_իրականացնի_ֆունկցիան  
    return արժեք
```

Ֆունկցիա սահմանելուց օգտագործում ենք երկու բանալի բառեր առաջինը `def` (սահմանել), որից հետո գրում ենք ֆունկցիայի անունը ընտրված մեր կողմից, փակագծերում տալիս այն պարամետրերը, որը պետք է ֆունկցիան ստանա իր գործողությունը կատարելու համար, մեկից ավել լինելու դեպքում միմիանցից անջատում ենք ստորակետով, եթե ֆունկցիան պարամետրեր չունի ապա փակագծերը թողնում ենք դատարկ: Երկրորդը `return` բանալի բառն է, որով ֆունկցիան վերադարձնում է իր կատարած գործողությունների արդյունքում ստացված արժեքը: `return` բառը կարող է մի ֆունկցիայում լինել մեկից ավելի անգամ կախված թե, որ պայմանից ելնելով ինչ պետք է վերադարձնի ֆունկցիան, երբ կատարվում է `return` հրամանը դրանով ծրագիրը դուրս է գալիս ֆունկցիայից: Եթե ֆունկցիան ոչինչ վերադարձնելու կարիք չունի, ապա կարող ենք բաց թողնել `return` բանալի բառը կամ գրել `return none`, կամ պարզապես `return :`

Գրենք մի պարզագույն ֆունկցիայի օրինակ, որն ունի հետևյալ տեսքը:

```
def sum(a, b) :  
    c = a + b  
  
    return c
```

```
print(sum(4, 3))
```

Այս ծրագրի սկզբում սահմանում ենք մի ֆունկցիա, որը ստանում է երկու պարամետր, հայտարարում փոփոխական այդ պարամետրերի գումարը պահում է ստեղծված

փոփոխականում, որից հետո վերադարձնում այդ փոփոխականի արժեքը և դուրս գալիս ֆունկցիայից: Բայց այս գործընթացը տեղի չի ունենա, եթե մենք չկանչենք այն կատարաող (`sum()`) ֆունկցիային: Այս ֆունկցիայի կանչը տեղի է ունենում հենց `print()` ֆունկցիայի պարամետրերի դաշտում: `sum()` -ը ստանում է 4 և 3 թվերը գումարում դրան և վերադարձնում 7 արժեքը, իսկ `print()` ֆունկցիան տպում է այն ելքում:

Մեկ այլ կարևոր հասկացություն կապված ֆունկցիաների հետ, այն է փոփոխականի հասանելիության տիրույթ: Երբ սահմանում ենք ֆունկցիա և այդ ֆունկցիայի ներսում հայտարարում փոփոխական ապա այդ փոփոխականը կոչվում է լոկալ, քանի որ այն հասանելի է միայն ֆունկցիայի ներսում, իսկ ֆունկցիայից դուրս նրան դիմել չեն կարող: Գրքի սկզբում արդեն ասվել է, որ փոփոխականները լինում են երկու տեսակի լոկալ և գլոբալ, այստեղից պարզ է դառնում, որ գլոբալ են կոչվում այն փոփոխականները, որոնք հասանելի են ծրագրի թե ֆունկցիաների ներսում և թե դրանցից դուրս: Վերևի գրված օրինակում `c` -ն լոկալ է, քանի որ հայտարարված է ֆունկցիայի ներսում և դրանից դուրս նա հասանելի չէ:

Եթե ֆունկցիայից դուրս փորձենք դիմել նրա լոկալ փոփոխականներին, ապա կստանանք `NameError`: տիպի սխալ, որում կասվի, որ այդ փոփոխականը սահմանված չէ: Այս սխալի տեսակի մասին արդեն տեղեկացել ենք բացառությունների մշակման ժամանակ:

## Մոդուլներ և դրանց ստեղծումը

Python -ում Մոդուլը (**module**) ֆայլ է, որը բաղկացած կոդից: Այն կարող է սահմանել ֆունկցիաներ, դասեր (**class**) և փոփոխականներ, ինչպես նաև կարող է ներառել կատարվող կոդ: Python- ի ցանկացած ֆայլ կարող է համարվել որպես մոդուլ: Կարճ ասած Python -ի կոդ պարունակող ֆայլը, որը ունի `.py` ընդլայնում, կոչվում է մոդուլ:

Python -ը հայտնի է իր բազմաթիվ ներկառուցված ֆունկցիաներով, մեզ հայտնի են օրինակ՝ `format()`, `input()`, `print()`, `int()`, և այլն: Այս ֆունկցիաները պահվում են ֆայլերում, որոնք էլ անվանում ենք մոդուլներ: Կան այնպիսի ներկառուցված ֆունկցիաներ, որոնք օգտագործելու համար նախ պետք է դրանք կցել կամ ավելացնել մեր ծրագրում: Այդ գործողության համար Python -ում նախատեսված է `import` հրամանը, երբ ինտերպրետորը հանդիպում է այս հրամանին, ապա գտնում է համապատասխան անունով մոդուլը (ֆայլը) և ավելացնում այն մեր ծրագրում:

`import` հրամանի օգտագործման համար կա ընդունված երկու հիմնական ձև : Առաջինը, գրում ենք `import` մոդուլի անուն, որից հետո կցված մոդուլի անդամներին կամ ֆունկցիաներին կարող ենք դիմել օգտագործելով (`.`) օպերատորը մոդուլի անունի հետ միասին: Օրինակ կցենք `math` (մաթեմատիկական) մոդուլը մեր ծրագրին և օգտագործենք քառակուսի արմատ հաշվող ֆունկցիան: Դա



Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

կարվի հետևյալ կերպ:

```
import math
print(math.sqrt(25))
# 5.0 վերադարձնում է սահող կետով թիվ:
```

Եթե ձեր համակարգչում `math` մոդուլը բացակայում է ապա կարող եք այն ավելացնել, հրամանի տողում ([terminal](#)) կատարելով հետևյալ հրահանգը: `pip install python-math`, այս հրահանգով դուք կարող եք ավելացնել այլ մոդուլներ, որոնք առկա են PyPI ([Python Package Index](#)) ծրագրային ապահովման պահոցում կիրառելով  
`pip install առկա մոդուլի անուն` այս հրահանգի կատարման համար անհարաժեշտ է, որ ձեր համակարգիչը հասանելիություն ունենա համացանցն ([internet](#)) :

Երկրորդ ձևը, որով կարելի է կցել մոդուլներ մեր ծրագրին հետևյալն է:

```
from մոդուլի անուն import ֆունկցիայի կամ փոփոխականի անուն
```

Այս ձևը հարմար է օգտագործել, երբ մեզ տվյալ մոդուլից անհրաժեշտ է կոնկրետ ֆունկցիա կամ փոփոխական և ոչ ամբողջ մոդուլը: Այդ դեպքում մենք կարող ենք կցել `math` մոդուլից միայն մեզ անհրաժեշտ ֆունկցիան կամ փոփոխականը մեկից ավել լինելու դեպքում

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

կառանձնացնենք ստորակետով ու կարող ենք դրանց  
օգտագործել արդեն առանց (.) օպերատորի:

Օրինակ՝

```
from math import cos, sin  
print("cos(0)=",cos(0), "\nsin(0)=", sin(0))
```

Python լեզուն նաև հնարավորություն է տալիս մոդուլները  
վերանվանել, դրա անհրաժեշտությունը կարող է առաջանալ,  
երբ մոդուլի անունը պետք է վերաիմաստավորել, կամ ավելի  
կարճ գրել հետագայում օգտագործելու համար: Այդ  
գործողությունը կատարվում է *as* բանալի բառի օգնությամբ:

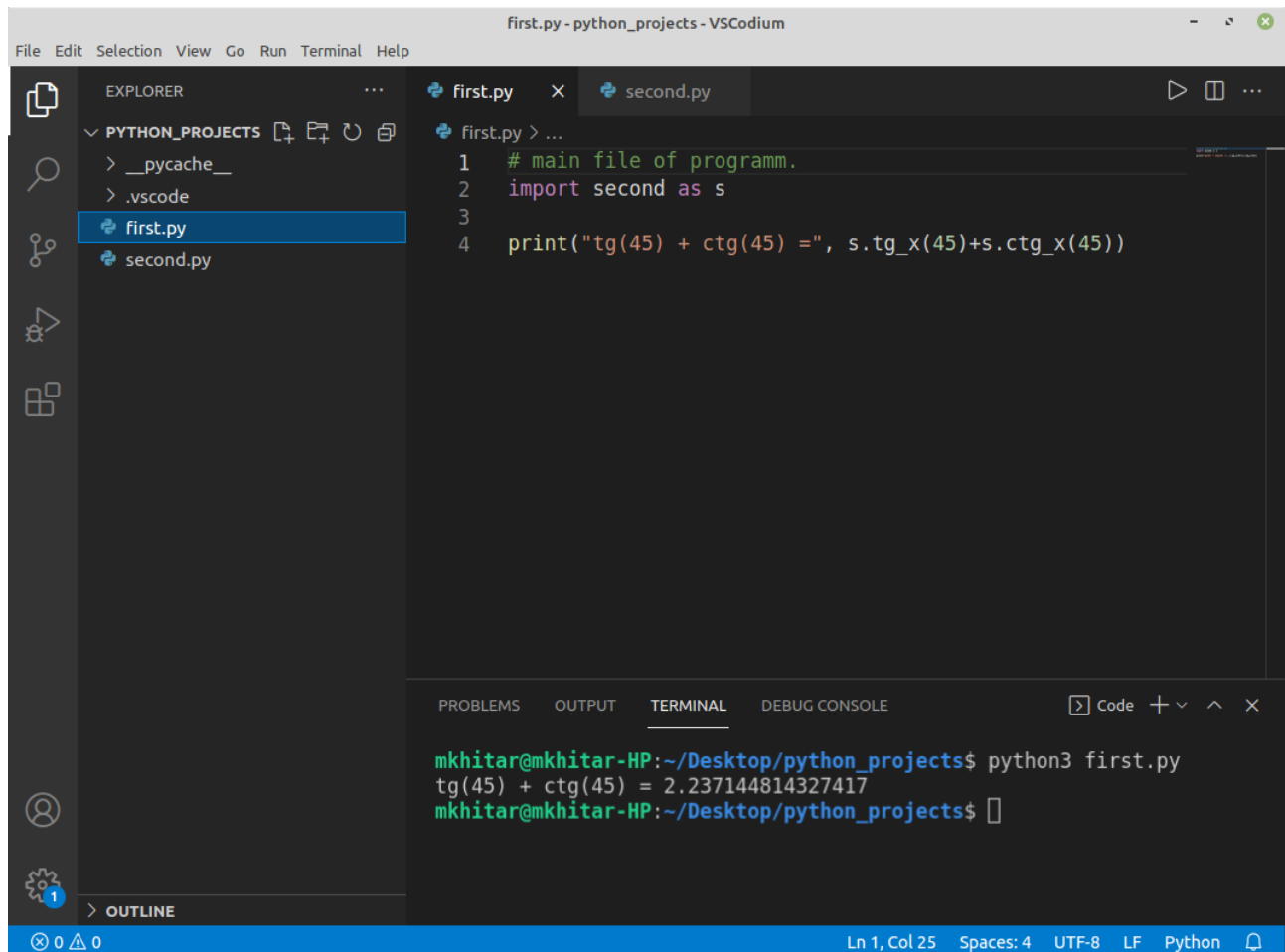
```
Import math as m  
print(m.sqrt(25))
```

Ուսումնասիրեցինք թե ինչ են մոդուլները և ինչպես դրանք  
օգտագործենք մեր ծրագրում: Կսովորենք թե ինչպես  
ստեղծել մեր սեփական մոդուլը, դիտարկենք օրինակի վրա:

Սկզբում պետք է տեքստային խմբագրիչի միջոցով ստեղծել  
նոր *.py* ընդլայնումով ֆայլ, այն նույն թղթապանակում, որտեղ  
ստեղծել էինք առաջին հիմնական ծրագրի ֆայլը: Հետո նոր  
ստեղծված ֆայլում ավելացնենք կոդը, որը պետք է կցել  
հիմնական ծրագրի ֆայլին և պահպանել այն: Պահելուց  
հետո անհրաժեշտ է արդեն վերևում նշված որևէ  
եղանակներից մեկով կցել երկրորդ ֆայլը առաջին  
հիմնական ծրագրի ֆայլին: Հետևյալ նկարներում ամփոփ

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Ներկայացված է ամբողջ գործընթացը և ծրագրի  
աշխատանքի արդյունքը:



The screenshot shows the Visual Studio Code (VS Code) interface. The Explorer panel on the left displays the file structure of a project named 'PYTHON\_PROJECTS'. It contains a subdirectory with files like '\_\_pycache\_\_' and '.vscode', and two Python files: 'first.py' and 'second.py'. The 'first.py' file is selected and its content is visible in the main editor. The code in 'first.py' is as follows:

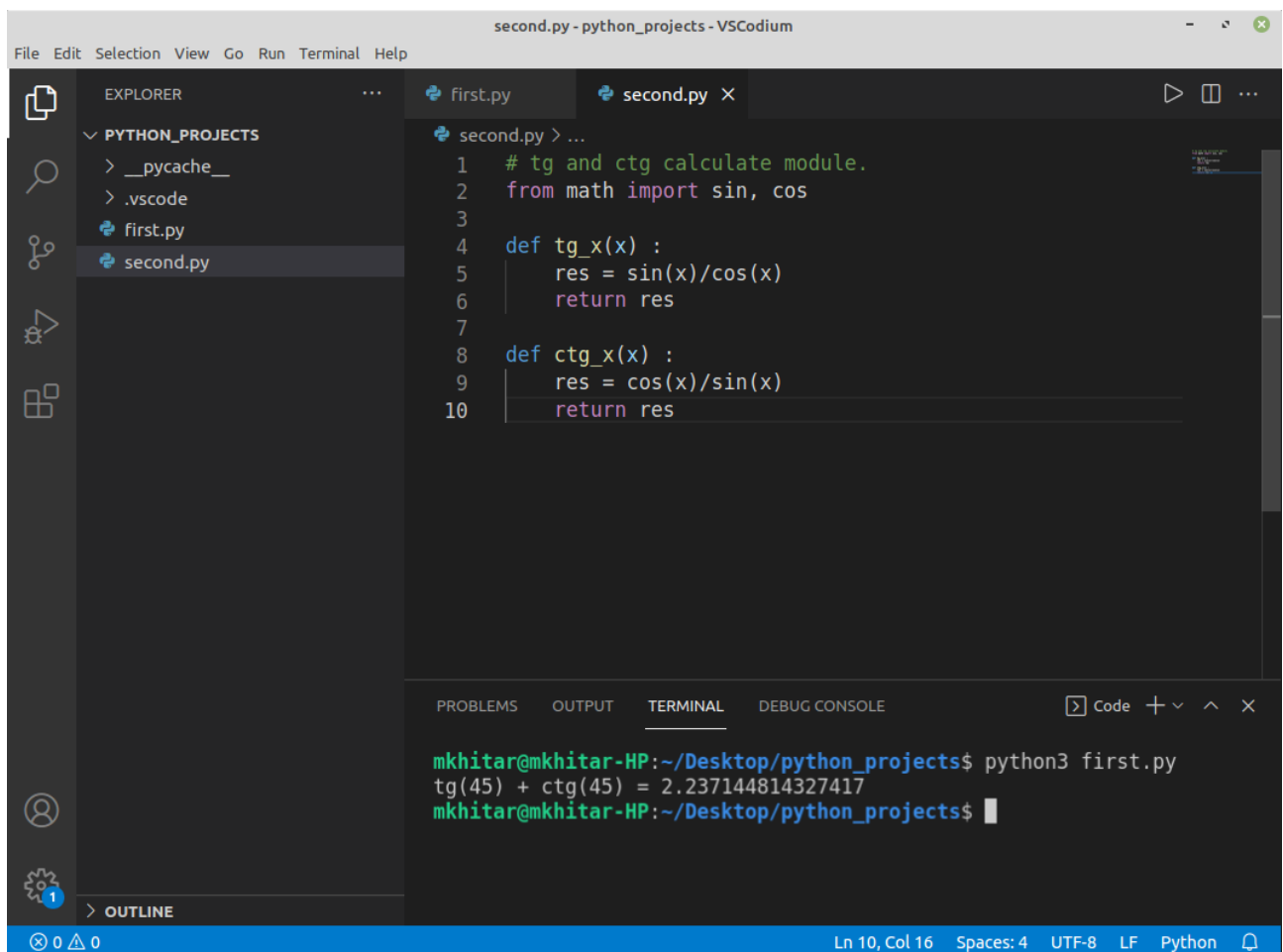
```
1 # main file of programm.
2 import second as s
3
4 print("tg(45) + ctg(45) =", s.tg_x(45)+s.ctg_x(45))
```

The bottom panel shows the TERMINAL output, where the command 'python3 first.py' has been executed, resulting in the output: 'tg(45) + ctg(45) = 2.237144814327417'.

Սկզբի նկարում բացված է մեր հիմնական ծրագրի ֆայլը  
(first.py), որին էլ կցվել է նույն թղթապանակում գտնվող  
երկրորդ (second.py) ֆայլը վերանվանելով այն կարճ որպես s  
այնուհետև այդ ֆունկցիաներին տվել ենք որոշ արժեքներ և  
դրանց գումարը տպել ելքում:

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Հաջորդ նկարում բացված է երկրորդ ֆայլի պարունակությունը, որտեղ նույնպես կցված է Python -ի արդեն իսկ եղած մոդուլներից մեկը: `math` -ից վերցված է երկու ֆունկցիա, որոնք էլ օգտագործել ենք մեր կոդից սահմանված ֆունկցիաների մեջ: Հիմնական ծրագրի ֆայլը կատարելուց ստանում ենք նկերների մեջ երևացող ելքում ստացված արդյունքը:



The screenshot shows the VS Code interface with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project named 'PYTHON\_PROJECTS' with files like '\_\_pycache\_\_', '.vscode', 'first.py', and 'second.py'. The code editor displays the content of 'second.py', which defines two functions: `tg_x(x)` and `ctg_x(x)`. The terminal shows the command `python3 first.py` being executed, resulting in the output `tg(45) + ctg(45) = 2.237144814327417`.

```
second.py - python_projects - VSCodeium
File Edit Selection View Go Run Terminal Help

EXPLORER
PYTHON_PROJECTS
  > __pycache__
  > .vscode
  first.py
  second.py

second.py > ...
1 # tg and ctg calculate module.
2 from math import sin, cos
3
4 def tg_x(x) :
5     res = sin(x)/cos(x)
6     return res
7
8 def ctg_x(x) :
9     res = cos(x)/sin(x)
10    return res

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
mkhitar@mkhitar-HP:~/Desktop/python_projects$ python3 first.py
tg(45) + ctg(45) = 2.237144814327417
mkhitar@mkhitar-HP:~/Desktop/python_projects$
```

Քանի որ Python -ը այնքան արագ չէ, որքան C, C++, C#, Java ծրագրավորման լեզուները, և բացի դրանից, երբ մենք կցում ենք նոր մոդուլներ մեր ծրագրին, ապա ծրագրում կոդի ծավալը ավելի է մեծանում հետևաբար աշխատանքը սկսում է դանդաղել: Այդ պատճառով լեզվի համար մշակել են այնպիսի մեխանիզմ, որը հնարավորինս կարագացնի ծրագրի աշխատանքը: Այդ մեխանիզմի աշխատանքը կայանում է հետևյալում, **Byte Compiled** (բայթերի թարգմանված) ֆայլերի օգտագործումը մոդուլների փոխարեն: Սկզբում Python -ի ինտերպրետատորը թարգմանում է մեր գրած կոդը միջանկյալ կոդի, որը կոչվում է բայթկոդ (**bytecode**) դրանից հետո նոր թարգմանում այն մեր համակարգչին համապատասխան մեքենայական կոդի: Բայթկոդի թարգմանված ֆայլերը ունենում են **.pyc** ընդլայնում: Երբ մենք կցում ենք նոր մոդուլ մեր ծրագրին և կատարում ծրագիրը ապա առաջին իսկ աշխատանքից հետո գեներացվում է մեր մոդուլի բայթկոդով ֆայլը: Դա մեզ օգնում է նրանում, որ երբ այդ մոդուլը օգտագործենք այլ ծրագրում, կամ նույն ծրագրում երկրորդ անգամ, ապա այն կկատարվի արդեն ավելի արագ: Քանի որ մոդուլի կցման համար անհրաժեշտ վերամշակման մի մասն արդեն արված է բայթկոդում և ծրագիրը օգտագործում է այն: Այս բայթկոդի թարգմանված ֆայլերը պլատֆորմից անկախ են (**platform-independent**): Python -ը ունի հնարավորություն նախապես մոդուլը թարգմանելու բայթկոդի դրա համար օգտագործում են `py_compile` մոդուլը:

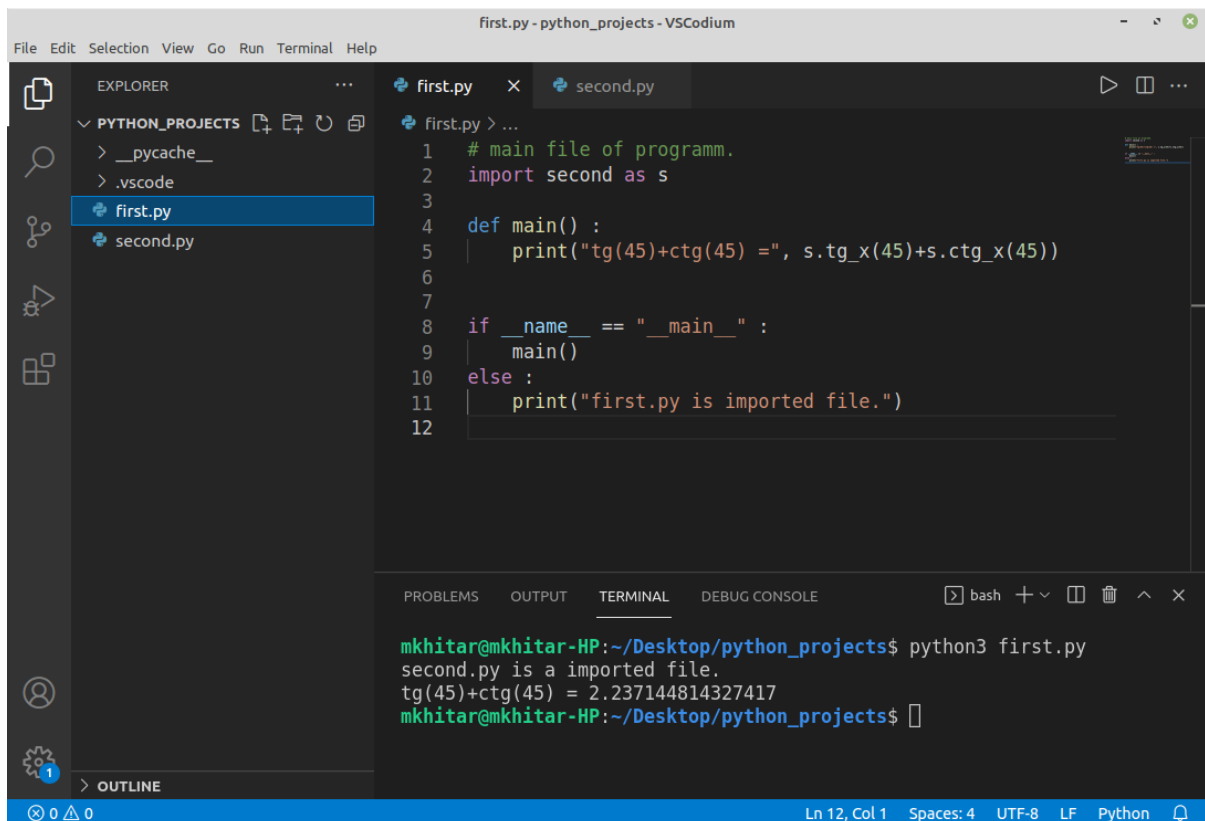
## **Հիմնական ֆունկցիայի սահմանում** **(main())**

Այսպիսով երբ մենք ծանոթ ենք ֆունկցիաներին և մոդուլներին, ապա անհրաժեշտ է հասկանալ ևս մեկ շատ կարևոր գաղափար, այն է ծրագրում հիմնական (առաջնային) ֆունկցիայի սահմանումը: Ինչպես շատ ծրագրավորման լեզուներում այնպես էլ Python -ում տրված է հնարավորություն սահմանել հիմնական ֆունկցիա և կանչել այն դեպքում, երբ մեր ծրագրի ֆայլը կատարվում է ինտերպրետատորում որպես հիմնական ծրագրի ֆայլ: Մեզ մնում է միայն պարզել, եթե տվյալ ծրագրի ֆայլը, որը ներկա պահին կատարվում է ինտերպրետատորում հիմնականն է, ապա առաջնային անհրաժեշտ է այդ ծրագրում կատարել մեր կողմից ստեղծված հիմնական ֆունկցիան: Այդ ֆունկցիան, որպես կանոն, ընդունված է անվանել `main()` : Պարզելու համար տվյալ ծրագրի ֆայլը հիմնականն է թե կցված է ու երկրորդական Python -ում կա ներկառուցված փոփոխական `__name__` անունով: Որտեղ ինտերպրետատորի կողմից լրացվում այն ֆայլի անունը, որից նա կատարում է հրամաններ, և եթե այդ ֆայլը տրվել է նրան որպես հիմնական, ապա այդ փոփոխականի արժեքում ֆայլի անվան փոխարեն գրվում է “`__main__`” արժեքը: Մնում է մայն `if` պայմանի կառուցվածքով ստուգել և կանչել մեր կողմից սահմանված հիմնական ֆունկցիան հետևյալ կերպ:

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

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

Ավելացնենք հիմնական ֆունկցիան նկարներում  
պատկերված ծրագրի մեջ, ապա դրանք կունենան հետևյալ  
տեսքը:



The screenshot shows the VS Code interface with a file explorer on the left containing 'first.py' and 'second.py'. The main editor displays the content of 'first.py':

```
1 # main file of programm.  
2 import second as s  
3  
4 def main() :  
5     print("tg(45)+ctg(45) =", s.tg_x(45)+s.ctg_x(45))  
6  
7  
8 if __name__ == "__main__" :  
9     main()  
10 else :  
11     print("first.py is imported file.")  
12
```

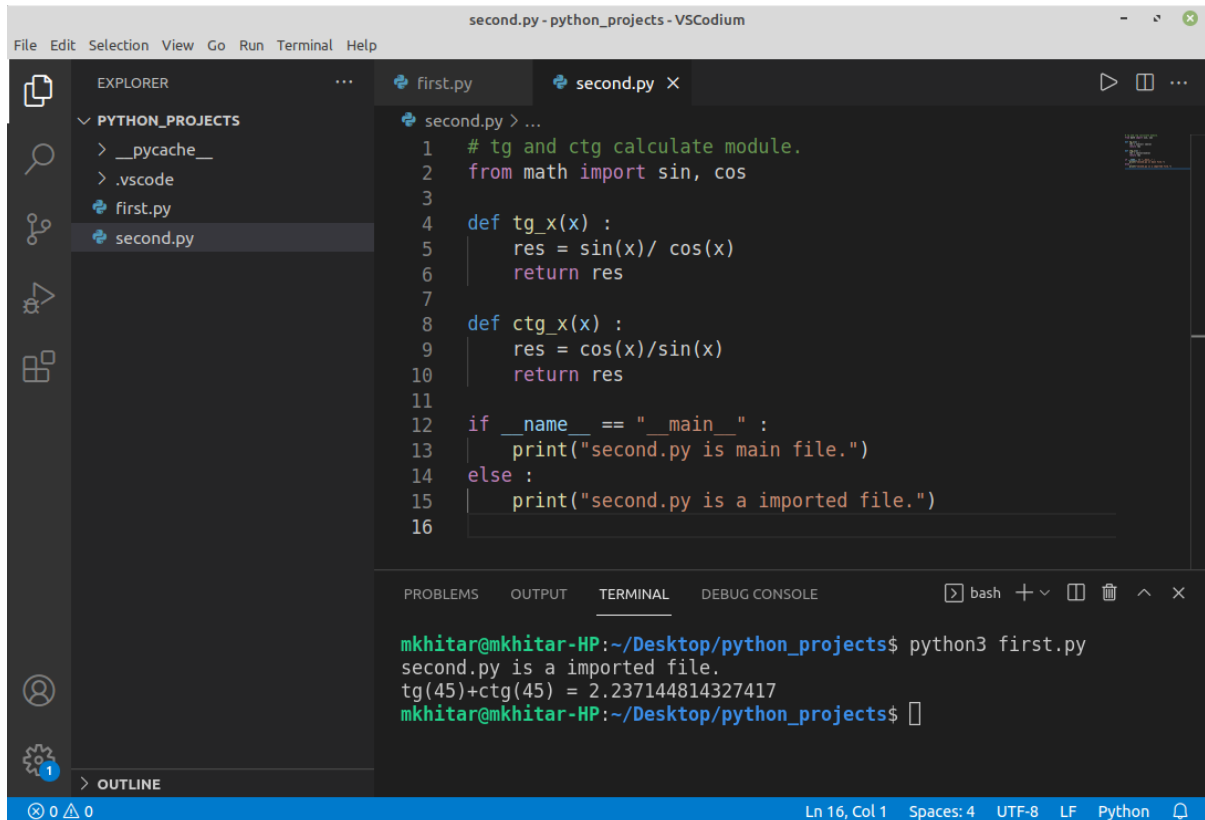
The terminal at the bottom shows the execution of the script:

```
mkhitar@mkhitar-HP:~/Desktop/python_projects$ python3 first.py  
second.py is a imported file.  
tg(45)+ctg(45) = 2.237144814327417  
mkhitar@mkhitar-HP:~/Desktop/python_projects$
```

Այսպիսով հրամաննի տողում մենք ինտերպրետատորին  
փոխանցում ենք first.py ֆայլը, որն էլ նա համարում է  
հիմնական, և այդ ֆայլում գրված if պայմանը դառնում է

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Ճիշտ, հետևաբար կանչվում է մեր կոդմիջ սահմանված `main()` անունով ֆունկցիան որպես ծրագրի հիմնական կատարող ֆունկցիա, և ելքում տպվում է հաշվարկի արդյունքը:



The screenshot shows the VS Code interface with a file explorer on the left showing a project named 'PYTHON\_PROJECTS' containing 'first.py' and 'second.py'. The main editor displays the content of 'second.py', which defines two functions, `tg_x(x)` and `ctg_x(x)`, and a main block that prints a message if it's the main file or another message if it's imported. Below the editor, the terminal shows the command `python3 first.py` being executed, resulting in the output: `second.py is a imported file.` and `tg(45)+ctg(45) = 2.237144814327417`.

```
second.py - python_projects - VSCodium
File Edit Selection View Go Run Terminal Help

EXPLORER
PYTHON_PROJECTS
  > __pycache__
  > .vscode
  first.py
  second.py

second.py > ...
1 # tg and ctg calculate module.
2 from math import sin, cos
3
4 def tg_x(x) :
5     res = sin(x)/ cos(x)
6     return res
7
8 def ctg_x(x) :
9     res = cos(x)/sin(x)
10    return res
11
12 if __name__ == "__main__" :
13     print("second.py is main file.")
14 else :
15     print("second.py is a imported file.")
16

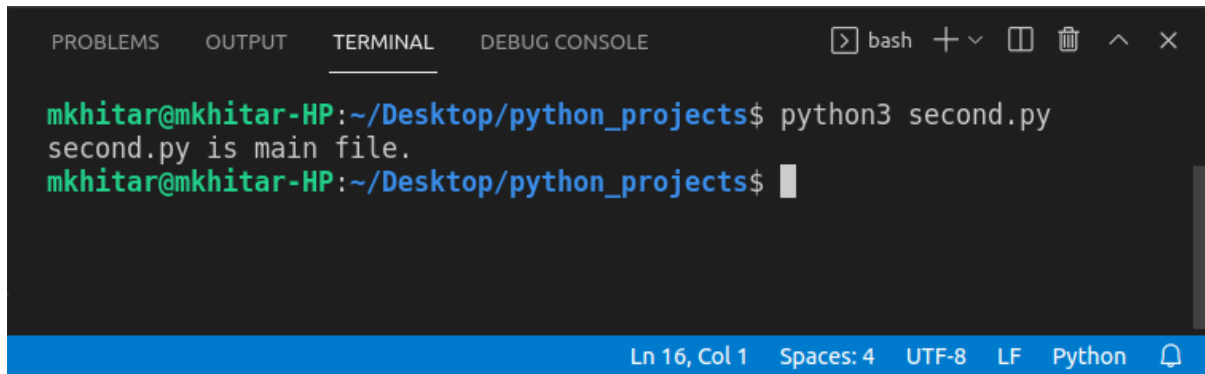
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
bash
mkhitar@mkhitar-HP:~/Desktop/python_projects$ python3 first.py
second.py is a imported file.
tg(45)+ctg(45) = 2.237144814327417
mkhitar@mkhitar-HP:~/Desktop/python_projects$

Ln 16, Col 1 Spaces: 4 UTF-8 LF Python
```

`second.py` ֆայլը արդեն նախապես որոշվել էր որպես երկրորդական ֆայլ, այն նախատեսված էր որպես մոդուլ հիմնական ծրագրի համար: Այդ պատճառով էլ այս ֆայլում կատարվել է `else` բլոկը և ելքում տպվել, որ այն կցված ֆայլ է: Բայց, եթե մենք ինտերպրետատորին կատարելու համար տրամադրենք `second.py` ֆայլը, ապա նա կհասկանա դա որպես հիմնական և կկատարվի `if` բլոկի կոդը: Պատկերը կլինի հետևյալը:



Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0



The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, TERMINAL, and DEBUG CONSOLE. The terminal is running a bash shell. The user has executed the command `python3 second.py`, and the output is `second.py is main file.`. The terminal prompt is `mkhitar@mkhitar-HP:~/Desktop/python_projects$`. The status bar at the bottom indicates the cursor is at line 16, column 1, with 4 spaces, UTF-8 encoding, LF line endings, and the file is a Python script.

## Ներկառուցված մոդուլներ և փաթեթներ

Python լեզուն ունի մի շարք ներկառուցված մոդուլներ, որոնք կհեշտացնեն մեր առջև դրված խնդրի կատարումը, կամ կտան որոշ գործիքակազմ խնդիրը լուծելու համար: Դիտարկենք որոշ ներկառուցված մոդուլներ, որոնք հաճախ են օգտագործվում:

`sys` մոդուլը տրամադրում է տարբեր ֆունկցիաներ և փոփոխականներ, որոնք օգտագործվում են Python-ի գործարկման ժամանակաշրջանում ([runtime environment](#)): Այն թույլ է տալիս աշխատել ինտերպրետատորի հետ:

Դիտարկենք մեր ինտերպրետատորի տարբերակը ելքում տպող ծրագիր:

```
import sys

print(sys.version)
```

`time` մոդուլը տրամադրում է ժամանակը կոդում ինտեգրելու բազմաթիվ եղանակներ, օրինակ՝ կոդի կատարման ընթացքում սպասելը կամ ձեր կոդի արդյունավետությունը ստուգել:

`random` գեներացնում է պատահական փոփոխականներ: Այն կարող է օգտագործվել պատահականորեն որոշ գործողություններ կատարելու համար, օրինակ՝ պատահական թիվ ստանալու, ցուցակից պատահական տարրեր ընտրելու, և նմանատիպ այլ գործողությունների համար:

`math` մոդուլը սահմանված է ամենահայտնի մաթեմատիկական ֆունկցիաները, ներառյալ եռանկյունաչափական ֆունկցիաներ, լոգարիթմական ֆունկցիաներ և այլն: Բացի այդ, այն նաև սահմանում է երկու մաթեմատիկական հաստատուններ,  $\pi$  և  $e$  թվերը:

`turtle` մոդուլը հնարավորություն է տալի նկարել, և գծագրել:

Կա մեկ այլ հասկացողություն այն է որոշը մոդուլների հավաքածուն կարող է իրենից ներկայացնել փաթեթ ([package](#)), որն էլ կարող է հնարավորություն տալ կոնկրետ ծրագրավորման ոլորտի խնդիրներ լուծելու: Այդպիսի փաթեթներից են [PyQt](#), [NumPy](#), [Requests](#), [Pytest](#) և այլն:

## Ֆունկցիաների ռեկուրսիա (recursion)

Մի բան, որը պարունակում է իրեն, կամ իր նմանին որակվում է որպես ռեկուրսիա: Դասական օրինակ է, երբ հայելին տեղադրված է մեկ այլ հայելու դիմաց: Եկեք հասկանանք թե այն ինչպես է օգտագործվում ծրագրավորման մեջ: Ռեկուրսիան ծրագրավորման մեջ հիմնական գաղափարներից մեկն է: Շատ ծրագրավորման լեզուներ, ինչպես նաև Python -ը՝ հնարաորություն են տալիս օգտագործել ռեկուրսիան (կրկնարկում)՝ թույլատրելով ֆունկցիային կանչել ինքն իրեն ծրագրի տեքստի մեջ:

Այժմ, երբ մենք ունենք որոշակի գաղափարներ ռեկուրսիայի մասին, եկեք տանք ռեկուրսիվ ֆունկցիայի սահմանումը Python -ում: Ֆունկցիան որը կանչում է ինքն իրեն և կատարում, արդյունքում վերադարձնելով արժեք, կամ կանչում է իրեն նորից, ելնելով ինչ-որ պայմաններից կոչվում է ռեկուրսիվ:

Պարզության համար դիտարկենք ֆունկցիայի օրինակ, որը կհաշվի  $n$  թվի ֆակտորյալը ռեկուրսիայի միջոցով:  
Մաթեմատիկորեն այն կհաշվենք հետևյալ կերպ՝  
$$n! = n * (n-1) * (n-2) * (n-3) * (n-4) \dots * 2 * 1$$
  
Իսկ ֆունկցիայի իրականացումը կունենա այս տեսքը:

```
def factorial(n) :  
    if n == 1 :  
        return 1  
    else :  
        return n * factorial(n-1)
```

Հիմա երբ մենք կանչենք այս ֆունկցիան և պարամետրում տանք 5 արժեքը, ապա կտաի 120 որպես վերջնական

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

արդյունք, հետևյալ կերպ:

```
print(factorial(5))  
# 120
```

Ֆունկցիան իր աշխատանքի ընդացքում ստուգում է, եթե պարամետրում ստացել է 1, ապա վերադարձնում է հենց 1, հակառակ դեպքում վերադարձնում է պարամետրի արժեքը բազմապատկած ֆունկցիայի կանչին նոր պարամետրով: Որտեղ այդ նոր պարամետրը նախորդից փոքր է մեկով: Այսպիսով երբ պարամետրը փեքրանա այնքան որ հավասարվի 1 -ի ֆունկցիայի կանչը կավարտվի:

Python -ում կան նաև կարճ ֆունկցիաներ, կամ այլ կերպ ասած `lambda` ֆունկցիաներ, որոնք նման են սովորական ֆունկցիաներին, բացառությամբ, որ դրանք սահմանելիս անուն չունեն, և պարունակում են ընդամենը մեկ տող հրաման: Գրելաձևը հետևյալն է:

`lambda` պարամետրեր : ինչ-որ գործողություն

Օրինակ սահմանենք կարճ ֆունկցիա, որը կգումարի 5 արժեքը տրված պարամետրին և կվերադարձնի արդյունքը:

```
x = lambda a : a + 5  
print(x(5))  
# 10
```

## Առաջադրանքներ

Առաջարկվում է լուծել հետևյալ խնդիրները:

- 1) Գրեք ֆունկցիա, որը ցուցակից կգտնի մեծագույն տարրը:
- 2) Գրեք ֆունկցիա, որը կվերադարձնի ցուցակի բոլոր թվերի գումարը:
- 3) Գրեք ֆունկցիա, որը կվերադարձնի ցուցակի բոլոր թվերի արտադրյալը:
- 4) Գրեք ֆունկցիա, որը պարամետրում ստանում է տողը և վերադարձնում այն հակառակ շրջված:
- 5) Գրեք ֆունկցիա՝ տրված ցուցակից զույգ թվերը տպելու համար:
- 6) Գրեք ֆունկցիա, որը տպում է Պասկալի եռանկյունը:
- 7) Գրեք ծրագիր՝ տրված ցուցակից պատահական տարր ընտրելու համար: Օգտագործեք `random.choice()` ֆունկցիան, որը վերադարձնում է պատահականորեն ընտրված տարրը նշված հաջորդականությունից:
- 8) Գրեք ծրագիր, որը կգեներացնի ֆիքսված երկարությամբ պատահական տող տրված այբուբենի ցուցակից: Օգտագործեք `random.choice()` ֆունկցիան:
- 9) Գրեք ծրագիր, որը կգեներացնի պատահական թիվ  $[0, 10]$  հատվածում: Օգտագործեք `random.random()` ֆունկցիան, որը վերադարձնում է  $[0, 1]$  հատվածից պատահական թիվ:
- 10) Գրեք ծրագիր՝ տրված ցուցակի տարրերի դիրքը

պատահականորեն փոխելու համար: Օգտագործեք `random.shuffle()` ֆունկցիան, որը պատահականորեն կարգավորում է հաջորդականությունը:

- 11) Գրեք ծրագիր՝ պատահական հինգ տարր պարունակող ամբողջ թվերի ցուցակ ստեղծելու և այդ ցուցակից պատահականորեն երեք տարրանոց ենթացուցակ կառուցելու համար: Օգտագործեք `random.sample()` ֆունկցիան:
- 12) Գրեք ֆունկցիա, որը տրված ռադիանը կվերափոխի աստիճանի:  $1 \text{ ռադ.} = 57.2958^\circ$
- 13) Գրեք ծրագիր, որը տված երկու թվերը կբազմապատկի առանց ( `*` ) օպերատորը օգտագործելու:
- 14) Գրեք ծրագիր, երկուական թիվը տասնորդականի վերածելու համար:
- 15) Ստեղծել Python -ի մոդուլ, որում սահմանված կլինի մաթեմատիկական գործողությունների ֆունկցիաներ `+`, `-`, `/`, `*` և օգտվելով այդ մոդուլից հիմնական ֆայլում գրեք պարզագույն հաշվիչի ծրագիր: Անհրաժեշտ է նաև սահմանել հիմնական ֆունկցիա, որում կղեկավարվի ծրագրի ընթացքը:
- 16) Գրեք ծրագիր, որը օգտագործողից կստանա թվերի հաջորդականության ցուցակ և ձեր կողմից սահմանված ֆունկցիայի միջոցով կստուգի տվյալ հաջորդականությունը հանդիսանում է արդյոք թվաբանական կամ երկրաչափական պրոգրեսիա: Եվ մեկ այլ ֆունկցիա, որն կախված հաջորդականության տեսակից կհաշվի հաջորդ տարրը:

- 17) Գրեք ռեկուրսիվ ֆունկցիա, որը տրված տողի սիմվոլները կտալի հակառակ հերթականությամբ:
- 18) Սահմանեք կարճ ֆունկցիա, որը կհաշվի պարամետրում ստացած երեք թվերի արտադրյալը:

## **Գլուխ VII աշխատանք ֆայլերի հետ**

Այս գլխում մենք կիմանանք, թե ինչպես կարելի է օգտագործողից մուտքագրված մեծ քանակությամբ ինֆորմացիան պահել: Նման դեպքերում ավելի հարմար միջոց է ինֆորմացիան որպես արտաքին ֆայլ պահելը և մեր ծրագրին անհրաժեշտ ինֆորմացիան ֆայլից կարդալը: Համակարգչային ֆայլը օգտագործվում է թվային ձևաչափով տվյալների պահպանման համար, ինչպիսիք են պարզ տեքստը, պատկերի տվյալները կամ ցանկացած այլ բովանդակություն: Սկսենք ուսունասիրել աշխատանքը տեքստային ֆայլի հետ:

### **Բացել և կարդալ (Text) ֆայլը**

Նախ և առաջ մեզ պետք է տեքստային ֆայլ, ուստի ստեղծենք այն խմբագրիչի միջոցով նշելով ընդլայնումը .txt,

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Ֆայլի մեջ գրենք ինչ-որ տեքստ, և պահենք նույն թղթապանակում, որտեղ գտնվում է մեր ծրագրի կոդը պարունակող ֆայլը:

Ենթադրենք ստեղծված ֆայլը ունի `simple.txt` անունը և նրանում գրված է հետևյալ տեքստը երկու տողով:

1. This is a simple text file
2. for storing my Python data.

Դիտարկենք հետևյալ ծրագրի օրինակը:

```
file_simple = open("simple.txt", ' r ')\nfirst_line = file_simple.readline()\nsecond_line = file_simple.readline()\nprint (first_line, second_line)\nfile_simple.close()
```

Ֆայլը բացելու համար օգտագործում ենք `open()` ֆունկցիան, որը ընդունում է երկու պարամետր:

Առաջինը դա ֆայլի հասցեն է մեր օրինակում այն հենց ֆայլի անունն է, քանի որ տեքստային ֆայլը և ծրագրի կոդը պարունակող ֆայլը գտնվում են նույն հասցեում կամ այլ կերպ ասած նույն թղթապանակի մեջ, եթե տեքստային ֆայլը գտնվեր այլ թղթապանակում մենք ստիպված կլինեիք բացի անունից նաև գրել նրա գտնվելու վայրի ամբողջական հասցեն:

Երկրորդ պարամետրը ընդունում է ֆայլի բացման ռեժիմը: Հիմնական օգտագործվող ռեժիմներն հետևյալներն են:

‘ r ’ միայն ֆայլից կարդալու համար:

‘ w ’ միայն ֆայլում գրելու համար: Եթե ֆայլը գոյություն չունի



ապա կստեղծի, իսկ եթե գոյություն ունի ապա ֆայլի ողջ պարունակությունը կմաքրի նորը գրելու համար:

‘ a ’ ֆայլում վերջից գրելու (ավելացնելու) համար: Եթե ֆայլը գոյություն չունի ապա կստեղծի նորը, իսկ եթե ունի, ապա գրելուց կավելացնի ֆայլի վերջում: Պահպանելով ֆայլում մինչև բացելը եղած պարունակությունը:

‘ r+ ’ կարդալու և գրելու համար:

Վերևի օրինակում ֆայլը բացելուց հետո օգտագործված է `readline()` ֆունկցիան ֆայլից մեկ տող կարդալու համար: Այս ֆունկցիան տողը կարդալուց հետո անցնում է հաջորդ տողին և հիշում տեղը, դա էլ տալիս է մեզ այն հնարավորությունը, որ յուրաքանչյուր անգամ այս ֆունկցիան կանչելիս այն կարդում է հաջորդ նոր տողը:

Այսպիսով ծրագիրը կատարելուց ելքում կտալի տեքստային ֆայլի պարունակությունը: `close()` ֆունկցիայով փակում ենք բացված ֆայլը: Ֆայլը փակելը պարտադիր է որպեսզի մեր կատարած գործողությունները այդ ֆայլի հետ պահպանվի և ազտվի ֆայլի կոդից զբաղեցված ռեսուրսը:

Python -ում կա մեկ այլ եղանակ ֆայլի հետ աշխատանքի համար դա `with` և `as` բանալի բառերի օգտագործումն է, որը ինքնաբերաբար հոգ է տանում ֆայլի փակման մասին, երբ այն դուրս կգա բլոկից, նույնիսկ սխալի դեպքում: Գրելաձևն հետևյալն է:

```
with open("simple.txt", ' r ') as file_simple :
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

### # Ֆայլի հետ գործողություններ

Խորհուրդ է տրվում օգտագործել այս աշխատանքի ձևը, քանի որ այն պատասխանատու է ֆայլի փակման համար և ունի հարմար գրելաձև:

Դիտարկենք for ցիկլի միջոցով ինչպես կարելի է տպել ֆայլի պարունակությունը տող առ տող հետևյալ օրինակում:

```
file_simple = open ("simple.txt", ' r ')\nfor line in file_simple :\n    print (line)\nfile_simple.close()
```

Այս կոդում չի օգտագործվում `readline()` ֆունկցիան քանի որ նրա գործը կատարում է for ցիկլը `print ()` ֆունկցիայի հետ միասին:

Ֆայլերից կարդալու համար հաճախ օգտագործվում է մեկ այլ մեթոդ, որն էլ կարդում է ֆայլից նախատեսված բուֆերի ([buffer](#)) չափով, թույլ չտալով, որ ծրագիրը օգտագործի ավելի շատ համակարգչային ռեսուրս ([computer resources](#)) : Դա անելու համար կօգտագործենք `read()` ֆունկցիան հետևյալ կերպ:

```
file_simple = open ("simple.txt", ' r ')\nbuffer = file_simple.read(21)\nprint(buffer)\nfile_simple.close()\n\n# This is a simple text
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

`read()` ֆունկցիայի պարամետրում գրվում այն սիմվոլների քանակը (բուժերի չափը), որը պետք է կարդա ֆայլից: Լռելայն `-1` է, ինչը նշանակում է կարդալ ամբողջ ֆայլը:

## Գրել (Text) ֆայլում

Մինչև հիմա դիտարկում էիք օրինակներ, թե ինչպես կարելի է ֆայլից կարդալ, բայց հիմա կտեսնենք, թե ինչպես կարելի է գրել ֆայլում:

Սկսենք նրանից, որ ֆայլը պետք է բացել, օգտագործենք `'a'` (ավելացման) ռեժիմը, որպեսզի բացելուց հետո ֆայլի պարունակությունը չջնջվի: Դիտարկենք հետևյալ կոդը:

```
file_simple = open('simple.txt', 'a')
file_simple.write('\nNew line.')
file_simple.write('\nAnother new line!')
file_simple.close()
```

Տեքստային ֆայլում գրելու համար օգտագործում ենք `write()` ֆունկցիան:

## Բացել, կարդալ և գրել երկուական ֆայլերում

Երկուական ֆայլ ասելով հասկանում ենք ոչ տեքստային ցանկացած ֆայլ, օրինակ կարող է լինել պատկեր կամ վիդեո ֆայլ: Երկուական ֆայլերի հետ աշխատելու համար

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

օգտագործվում են ‘rb’ և ‘rw’ ռեժիմները  
համապատասխանաբար ֆայլից կարդալու և գրելու համար:

Դիտարկենք պատկերը պատճենող ծրագրի օրինակ: Դրա  
համար պետք է մեր ծրագրի թղթապանակում ունենանք .jpg  
ընդլայնումով ցանկացած պատկերի ֆայլ, ենթադրենք դա  
my\_photo.jpg ֆայլն է: Որից հետո գրենք և կատարենք  
հետևյալ ծրագիրը:

```
input_file = open ("my_photo.jpg", 'rb')
output_file = open("copy.jpg", 'wb')
buffer = input_file.read(10)

while len(buffer):
    output_file.write(buffer)
    buffer = input_file.read(10)

input_file.close()
output_file.close()
```

Այս ծրագիրը նախապես գոյություն ունեցող երկուական  
պատկերի ֆայլից while ցիկլի միջոցով կարդում է 10 բուլֆերի  
չափով և գրում է իր կեղմից ստեղծած մեկ այլ ֆայլում:  
Արդյունքում ստացվում է, որ երկրորդ ֆայլը դառնում է  
առաջինի պատճենը, որի անունը օրինակում copy.jpg է:  
Ծրագրի ավարտից հետո, երբ բացենք այդ ֆայլը կտեսնենք  
նույն պատկերը ինչ առաջին my\_photo.jpg ֆայլում է: Կոդում  
նաև օգտագործել ենք len() ներկառուցված ֆունկցիան, որը  
վերադարձնում է Python -ում ցանկացած իտերացվող տիպի  
երկարությունը այլ կերպ ասած տարրերի քանակը:

## Ջնջել և վերանվանել ֆայլը

Երկու այլ օգտակար ֆունկցիաներ, որոնք պետք կգան ֆայլերի հետ աշխատելիս: Դրանք են `remove()` և `rename()`, այս ֆունկցիաները հասանելի են `os` ներկառուցված մոդուլում և նախքան օգտագործելը անհրաժեշտ է կցել այդ մոդուլը:

Առաջին ֆունկցիան ջնջում է ֆայլը: Գրելաձևը հետևյալն է:  
`remove(' ֆայլի անուն ')` :

Երկրորդը վերանվանում է ֆայլը: Գրելաձևը հետևյալն է:  
`rename(' հին ֆայլի անուն ', ' նոր ֆայլի անուն ')` :

## Առաջադրանքներ

Առաջարկվում է լուծել հետևյալ խնդիրները:

- 1) Գրեք ծրագիր օգտագործողի կողմից ներմուծված անունով տեքստային ֆայլը ամբողջությամբ կարդալու համար:
- 2) Գրեք ծրագիր օգտագործողի կողմից ներմուծված անունով տեքստային ֆայլից առաջին `n` տողը կարդալու համար:
- 3) Գրեք ծրագիր, որը կստեղծի օգտագործողի կողմից ներմուծված անունով տեքստային ֆայլ, կգրի այդ ֆայլում ինչ-որ տեքստ և կպահի:
- 4) Գրեք ծրագիր, որը կկարդա ֆայլ տող առ տող և կպահի այն ցուցակում:

- 5) Գրեք ծրագիր, որը կգտնի ֆայլում ամենաերկար տողը և կտալի այն ելքում:
- 6) Գրեք ծրագիր, որը կհաշվի ֆայլում տողերի քանակը և կտալի ելքում:
- 7) Գրեք ծրագիր, որը կհաշվի ֆայլում կրկնվող բառերի քանակը և կտալի ելքում:
- 8) Գրեք ծրագիր, որը տրված ցուցակի պարունակությունը կգրի ֆայլի մեջ:
- 9) Գրեք ծրագիր, որը կպատճենի ֆայլի պարունակությունը մեկ այլ ֆայլում:
- 10) Գրեք ծրագիր, որը մի քանի տեքստ ֆայլից կարդում է առաջին տողերը և ավելացնում այլ ֆայլի մեջ:
- 11) Գրեք ծրագիր, որը օգտագործողի կողմից ներմուծված անունով ֆայլը ջնջում է:
- 12) Գրեք ծրագիր, որը օգտագործողի կողմից ներմուծված անունով ֆայլը վերանվանում է ներմուծված այլ անունով:
- 13) Գրեք ծրագիր, որը առաջին ֆայլի յուրաքանչյուր տող միավորում է երկրորդ ֆայլի համապատասխան տողի հետ և գրում մեկ այլ ֆայլի մեջ:
- 14) Գրեք ֆունկցիա, որը պարամետրում կստանա ֆայլի փոփոխականը և կստուգի ֆայլը բաց է թե ոչ:
- 15) Գրեք ծրագիր, որը ֆայլի մեջ կգրի հետևյալ պատկերը

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

```
* * * * *  
*       *  
*       *  
* * * * *
```

## **Գլուխ VIII օբյեկտ-կողմնորոշված ծրագրավորում Python լեզվով**

Սա գրքի վերջին գլուխն է, որում էլ կծանոթանանք օբյեկտ-կողմնորոշված ծրագրավորման հետ, այն լայնորեն օգտագործվում է ժամանակից ծրագրավորման մեջ:

Օբյեկտ-կողմնորոշված ծրագրավորումը, ծրագրավորման մոտեցում է, որի գաղափարական հիմք են հանդիսանում դասը ([class](#)) և օբյեկտը ([object](#)) : Օբյեկտը կարող է սահմանվել որպես տվյալների դաշտ, որն ունի եզակի հատկություններ և վարք, իսկ դասը այն է ինչը սահմանում է օբյեկտի տվյալները, հատկությունները և գործողությունները դրանց հետ:

Օրինակի համար դասը կարող է նկարագրել մարդու անձը, իր տվյալների դաշտում սահմանելով անուն, ազգանուն, տարիք, և ուրիշ այլ փոփոխականներ:

## Դասի սահմանումը (Class)

Python -ում դաս սահմանելու համար օգտագործվում է `class` բանալի բառը, որին հաջորդում է դասի անունը և բլոկը, որի մեջ պարտադիր `def` հրամանով սահմանվում է դասը իրականացնող կամ նրա օբյեկտը սկզբնարժեքավորող մեթոդը, `__init__(self)`, Այն նման է կոնստրուկտոր ([constructor](#)) հասկացողությանը C++ լեզվում: Մոթոդը նույնպես ֆունկցիա է, պարզապես մեթոդ անվանում են այն ֆունկցիաներին, որոնք հանդիսանում են ինչ-որ դասի անդամ: Դասի սահմանաման կառուցվածքը հետևյալն է:

```
class դասի անուն :
```

```
    def __init__(self) :  
        pass
```

Այս կառուցվածքում `pass` հրամանը դասի անդամ չէ, այն ուղղակի հրաման է, որը թարգմանիչին հուշում է, որ այդ բլոկում կոդը թերի է և կատարելու անհրաժեշտություն չկա, որպեսզի սխալներ չառաջանան կատարելիս: Այստեղ էլ օգտագործել են այդ նպատակով, քանի որ մեթոդը լիարժեք իրականացված չէ: Դասը իրականացնող մեթոդը պարտադիր միշտ պետք է լինի դասում այն ընդունում է մեկ և ավել պարամետրեր: Առաջինը `self` -ն է, որը հղվում է բուն օբյեկտին այն նման է C++ լեզվում [this](#) -ին: Մնացած պարամետրերը, որոնք տրվում են դասը սահմանողի կամ նկարագրողի կողմից, միմյանցից բաժանվում են ստորակետով: Ավելի լավ պատկերացնելու համար դիտարկենք դասի սահմանումը օրինակի վրա:



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

## Դասի օբյեկտներ (object)

Հիմա երբ մենք արդեն սահմանել ենք Person դասը, ապա կարող ենք ստեղծել այդ դասի օբյեկտ: Դասի օբյեկտ ստեղծելու գրելաձևը հետևյալն է:

օբյեկտի անուն = դասի անուն(պարամետրեր)

Person դասի համար այն կլինի այսպես:

p1 = Person("Ararat", "Simonyan", 35 )

p2 = Person("Anna", "Nazaryan", 27)

Դասը կարելի է նաև հասկանալ որպես նոր տվյալի տիպ, սահմանված մեր կողմից:

Օբյեկտների ատրիբուտներին (attribut) կարելի է դիմել ( . ) օպերատորի միջոցով:

```
print(p1.first_name)
```

Սա կտալի Ararat Անունը:

## Մեթոդի սահմանում

Մեթոդները ֆունկցիաներ են, որոնք սահմանված են դասի ներսում և կարող են կանչվել միայն այդ դասի օբյեկտի համար օգտագործելով ( . ) օպերատորը: Օրինակի համար `__init__(self)` ֆունկցիան մեթոդ է, որը պարտադիր է դասը սահմանելուց ստեղծել: Այս մեթոդից բացի մենք կարող ենք ավելացնել այլ մեթոդներ դասում, նույն կերպ ինչպես սահմանված է `__init__(self)` մեթոդը: Այսպիսով ավելացնենք վերևում սահմանված `Person` դասում `info` մեթոդ, որը ելքում կտալի տվյալ անձ օբյեկտի մասին ինֆորմացիա:

```
class Person :  
    def __init__(self, first_name, last_name, age) :  
        self.first_name = first_name  
        self.last_name = last_name  
        self.age = age  
    def info(self) :  
        print("Name is ",self.first_name, ", surname is ",  
              self.last_name, ", age is ", str(self.age))
```

Այնուհետև, երբ մենք կանչենք `info` մեթոդը `p1` օբյեկտի համար, `p1.info()` ելքում կտեսնենք հետևյալ տեքստը:  
Name is Ararat, surname is Simonyan, age is 35

Python -ում ծրագրի կոդը գրելուց հատկապես ուշադրություն դարձրեք ներդրված կառուցվածքը պահելուն, օրինակ՝ ֆունկցիայի կամ մեթոդի բլոկում գրվող կոդը պետք է գրել ավելի խորքից քան նրա սահմանման հրամանն է գրված

(def), և նույն տրամաբանությամբ էլ դասում մեթոդը պետք է գրել ավելի խորքից քան դասի սահմանման հրամանն է գրված (class) : Նույն գրելաձևը օգտագործեք ցիկլերի և այլ կառավարման կառուցվածքների համար որպեսի խուսափեք գրելաձևի սխալներից (*syntax error*) :

## Դասերի ժառանգականություն (Inheritance)

Ժառանգականությունը այն գործընթացն է, որով մի դասը վերցնում է մյուսի ատրիբուտները (հատկանիշները) և մեթոդները, այլ կերպ ասած ժառանգում է: Նորաստեղծ ժառանգող դասը կոչվում է զավակ դաս (*child class*), իսկ որից նա ժառանգել էր կոչվում է ծնող դաս (*parent class*) : Չավակ դասերը բացի ժառանգած հատկանիշներից և մեթոդներից կարող են ընդլայնել ծնող դասի հատկանիշներն ու մեթոդներն: Այսինքն զավակ դասերը ժառանգում են ծնողի բոլոր հատկանիշները և մեթոդները, բայց կարող են նաև սահմանել հատկանիշներ և մեթոդներ, որոնք հատուկ են իրենց համար:

Որպես օրինակ կարելի է դիտարկել հարթության մեջ կետը, որն ունենում է երկու կոորդինատ  $x$  և  $y$ , որոնք էլ որոշում են կետի դիրքը հարթության մեջ: Ենթադրենք ունենք սահմանված *Point* անունով դաս, որի օբյեկտներն կետեր են ինչ-որ հարթությունից, ապա կարող ենք սահմանել մեկ այլ *Circle* անունով դասը, որը կժառանգի *Point* դասի հատկանիշներն և բացի այդ, ընդլայնելով իր հատկանիշները

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

կավելացնի  $r$  շառավղի հասկացողությունը, և այն իրենից  
կներկայացնի շրջանագծի դաս: Քանի որ շրջանագիծը դա  
հարթության մեջ ինչ-որ կենտրոնով փակ կոր է, որի կետերը  
հավասարաչափ են հեռացված այդ կենտրոնից, և այդ  
կետերի հեռավորությունը կենտրոնից նշանակվում է  $r$ -ով  
իսկ կենտրոնը կետ է, որն ունի կորդինատներ  $x$  և  $y$  :

Python -ում զավակ դաս ստեղծելու համար անհրաժեշտ է  
սահմանել նոր դաս և այդ դասի անվանը կից փակագծերում  
գրել ծնող դասի անունը, թե որից պետք է ժառանգի:

Այսինքն նոր զավակ դասի սահմանման ձևը կլինի այսպես:

```
class զավակ դասի անուն(ծնող դասի անուն) :  
    def __init__(self) :  
        pass
```

Հիմա իրականացնենք վերևում նշված կետի և շտջանգծի  
դասերը, և կազմակերպենք նրանց մեջը ժառանգման  
գործընթացը: Նախ սահմանենք `Point` դասը:

```
class Point :  
    def __init__(self, x, y) :  
        self.x = x  
        self.y = y
```

Որից հետո նոր սահմանենք զավակ դասը, որը պետք է  
ժառանգի `Point` -ից:

```
class Circle :
```

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

```
def __init__(self, x, y, r) :  
    Point.__init__(self, x, y)  
    self.r = r
```

Եթե զավակ դասում մենք սահմանենք միայն իր իրականացման ֆունկցիան, ապա ժառանգման գործընթացը կդադարեցվի, քանի որ նոր ստեղծված զավակ դասը չի օգտագործում այն փոփոխականները, որոնք ժառանգել է ծնող դասից: Այդ պատճառով էլ զավակ դասի իրականացման ֆունկցիայում կանչում ենք ծնող դասինը և փոխանցում նրան անհրաժեշտ պարամետրերը, որով կսկզբնարժեքավորի իր կողմից փոխանցված փոփոխականները (հատկանիշները) : Կարելի էր Circle շրջանագծի դասում իրականացնող ֆունկցիան գրել այնպես, որ նա սկզբնարժեքավորեր և օգտագործեր նաև ծնող դասից փոխանցված փոփոխականները: Պարզ ասած շրջանագծի դասը կարելի էր սահմանել նաև այս կերպ :

```
class Circle :  
    def __init__(self, x, y, r) :  
        self.x = x  
        self.y = y  
        self.r = r
```

Հիշեք, որ դասը իրականացնող (սկզբնարժեքավորող) ֆունկցիաները դրանք մեթոդներ են տվյալ դադի համա և կանչվում են ամեն անգամ երբ մենք ստեղծում ենք այդ դասի օբյեկտ:

Python -ում կա ներկառուցված super() ֆունկցիան, որը տույլ է տալիս կանչել կամ դիմել ծնող դասի բոլոր մեթոդներին և փոփոխականներին առանց գրելու ծնող դասի անունը և self պարամետրը:

Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Այսինքն սկզբանական **Circle** դասի սահմանման մեջ իր ծնող դասը իրականացնող ֆունկցիայի կանչը կարելի է փոխարինել `super()` ֆունկցիայով հետևյալ կերպ:

```
super().__init__(x,y)
```

## Առաջադրանքներ

Առաջարկվում է լուծել հետևյալ խնդիրները:

- 1) Գրեք ծրագիր, որում նկարագրված կլինի **Square** անունով դաս: Այդ դասի օբյեկտները կլինեն քառակուսիներ, որոնցում կպահվի իրենց կողմի երկարությունը և մակերեսը: Սահմանել մեթոդ այդ դասում, որը կտալի տվյալ քառակուսու մասին ինֆորմացիա ելքում:
- 2) Գրեք ծրագիր, որում նկարագրված կլինի **Vehicle** անունով դաս: Այդ դասի օբյեկտները կլինեն ավտոմեքենաներ, որոնցում կպահվի նրանց մակնիշը, արտադրման տարեթիվը, տիպը (մարդատար, բեռնատար), քաշը, շարժիչի ծավալը, գույնը: Սահմանել մեթոդ այդ դասում, որը կհաշվի և կտալի թե ինչքան ժամանակ է շահագործվում տվյալ ավտոմեքենան սկսած արտադրման տարեթվից: Այս մեթոդը իրականացնելու

համար կցեք `datetime` մոդուլը և օգտագոծեք դրանում սահմանված `now()` և `year()` ֆունկցիաները:

- 3) Գրեք ծրագիր, որում նկարագրված կլինի `Computer` անունով դաս: Այդ դասի օբյեկտները կլինեն համակարգիչներ, որոնցում կպահվի նրանց արտադրող ընկերության անունը, մոդելը, էկրանի չափը (դույմ), օպերատիվ հիշողության չափը, պրոցեսորի արագությունը նրանում եղած միջուկների քանակը, տեսաքարտի հզորությունը, կոշտ սկավառակի չափը: Դասում սահմանել մեթոդ որը կհզորացնի տվյալ համակարգչի պարամետրերը երկու անգամ:
- 4) Գրեք ծրագիր, որը կսահմանի `Rectangle` անունով դաս և այն կժառանգի `Square` դասից: Կավելացնի այդ դասում ուղղանկյան համար անհրաժեշտ երկրորդ կողմը: Կսահմանի երկու մեթոդ, որոնք կհաշվեն ուղղանկյան մակերեսը և պարագիծը ու կտպեն ելքում:
- 5) Գրեք ծրագիր, որը կսահմանի `Bus` անունով դաս և այն կժառանգի `Vehicle` դասից: Կավելացնի այդ դասում նստատեղերի քանակ պահող փոփոխական: Կսահմանի մեթոդ, որը տվյալ ավտոմեքենայի նստատեղերի քանակը կդարձնի 20, գույնը դեղին իսկ ժարժիչի ծավալը կմեացնի երեք անգամ:
- 6) Գրեք ծրագիր, որը կսահմանի `Laptop` անունով դաս և այն կժառանգի `Computer` դասից: Կավելացնի այդ դասում, գին, սենսորային էկրան (`true`, `false`): Կսահմանի մեթոդ, որը կհաշվի տվյալ համակարգչի գինը, եթե այն զեղչվի ինչ որ տոկոսային արժեքով, որն էլ կստանա

կատանա պարամետրում:

- 7) Գրեք ծրագիր, որը 6) -ում սահմանված դասից կսարքի օբյեկտ, որի պարամետրերը ներմուծված են օգտագործողի կողմից:
- 8) Գրեք ծրագիր, որը կստեղծի 4) -ում սահմանված դասի երկու տարբեր օբյեկտ, այնուհետև կսահմանի մի ֆունկցիա, որը պարամետրում ստանալով այդ օբյեկտները կտալի դրանցից ամենամեծը:

## Ամփոփում

Շնորհավորում եմ այսքանով գրքի նյութը ավարտվեց: Ընթերցելով այս գիրքը դուք ծանոթացաք Payton ծրագրավորման լեզվի հիմունքներին: Օգտագործելով ձեռք բերած գիտելիքները արդեն կարող եք կազմել լիարժեք ծրագրեր և ծրագրային մոդուլներ: Գիտելիքները ավելի ընդլայնելու համար խորհուրդ եմ տալիս ինքնուրույն ուսումնասիրել Payton -ի հայտնի փաթեթները ([packages](#)), որպեսզի կիրառեք ձեր գիտելիքները կոնկրետ ծրագրավորման ոլորտում:



Python ծրագրավորման լեզվի հիմունքներ  
Տարբերակ՝ 1.0

Շնորհակալություն գիրքն ընթերցելու համար, հուսով եմ այն օգտակար էր ձեզ համար:

Եթե գրքում հանդիպել եք ինչ-որ թերության կամ սխալ տեղեկության, ինչպես նաև դիմում, բողոք կամ այլ առաջարկի դեպքում կորոդ եք կապնվել և տեղեկացնել ինձ գրելով իմ անձնական էլ. հասցեին:  
[mkhitar.a.y@gmail.com](mailto:mkhitar.a.y@gmail.com)

Օգտակար հղումներ:

Խնդիրներ և լուծումներ - [shorturl.at/cioCl](http://shorturl.at/cioCl)

Python 3.9.6 documentation - <https://docs.python.org/3/>