ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ
ՀԱՄԱԼՍԱՐԱՆ

ԳԱՅԱՆԵ ՂՈՒԿԱՍՅԱՆ | ՄԱՐԻՆԵ ԲՈՒՆԻԱԹՅԱՆ
ՄԱՄԻԿՈՆ ԿԱՆՈՅԱՆ

python
ԾՐԱԳՐԱՎՈՐՄԱՆ ՀԻՄՈՒՆՔՆԵՐ

ուսումնական ձեռնարկ

ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

ԳԱՅԱՆԵ ՂՈՒԿԱՍՅԱՆ, ՄԱՐԻՆԵ ԲՈՒՆԻԱԹՅԱՆ,
ՄԱՄԻԿՈՆ ԿԱՆՈՅԱՆ

PYTHON ԾՐԱԳՐԱՎՈՐՄԱՆ ՀԻՄՈՒՆՔՆԵՐ

ՈՒՍՈՒՄՆԱԿԱՆ ՁԵՌՆԱՐԿ

ԵՐԵՎԱՆ
ԵՊՀ ՀՐԱՏԱՐԱԿՉՈՒԹՅՈՒՆ
2026

ՀՏԴ 004.43(07)
ԳՄԴ 32.973.2գ7
Ղ 952

Հրատարակության է երաշխավորել ԵՊՀ գիտական խորհուրդը:

Գրախոսներ ՝
Ռուբեն Գևորգյան, ֆ-մ.գ.թ., տ.գ.դ., պրոֆեսոր
Աննա Հովակիմյան, տեխն. գիտ. թեկն., դոցենտ

Ղուկասյան Գայանե (գլուխ 6-13, Խնդրագիրք), Բունիաթյան Մարինե (գլուխ 1-5), Կանոյան Մամիկոն (գլուխ 14-16)

Ղուկասյան Գ., Բունիաթյան Մ., Կանոյան Մ.
Ղ 952 Python ծրագրավորման հիմունքներ: Ուսումնական ձեռնարկ: Եր., ԵՊՀ հրատ., 2026, 298 էջ:

Ուսումնական ձեռնարկում շարադրված են Python լեզվով ծրագրավորման հիմունքները: Ձեռնարկը կազմված է ԵՊՀ տնտեսագիտության և կառավարման ֆակուլտետում դասավանդվող «Տեղեկատվական տեխնոլոգիաները տնտեսագիտությունում» առարկայի ուսումնական ծրագրին համապատասխան և ներառում է դասավանդման ժամանակ օգտագործված նյութը:

Ուսումնական ձեռնարկում ընդգրկված յուրաքանչյուր բաժնում շարադրված է համապատասխան տեսական նյութը, որին հաջորդում են գործնական օրինակներ, ծրագրային կոդեր, խնդիրներ և մեկնաբանություններ: Ձեռնարկում ներառված են նաև բազմաթիվ առաջադրանքներ ինքնուրույն լուծելու համար:

Ուսումնական ձեռնարկը նախատեսված է ԵՊՀ տնտեսագիտության և կառավարման ֆակուլտետի ուսանողների համար: Այն կարող է օգտակար լինել հարակից մասնագիտությունների ուսանողների և դասախոսների համար, ինչպես նաև այն ընթերցողների համար, որոնք ցանկանում են տիրապետել Python լեզվով ծրագրավորման հիմունքներին:

ՀՏԴ 004.43(07)
ԳՄԴ 32.973.2գ7

ISBN 978-5-8084-2758-7
https://doi.org/10.46991/YSUPH/9785808427587

Գրքում օգտագործված բոլոր ծրագրային կոդերը, գործնական օրինակները, լրացուցիչ նյութերը և թարմացումները հասանելի են նախագծի պաշտոնական GitHub հարթակում:

https://github.com/gayaneghukasyan/python-book

Բովանդակություն

ՆԵՐԱԾՈՒԹՅՈՒՆ

Python ծրագրավորման լեզուն ներկայումս աշխարհում ամենատարածված ծրագրավորման լեզուներից մեկն է: Այն մշակվել է Նիդերլանդների մաթեմատիկայի և համակարգչային գիտության ազգային-հետազոտական ինստիտուտի (Centrum Wiskunde & Informatica (CWI)) ծրագրավորող Գվիդո Վան Ռոսսումի (Guido Van Rossum) կողմից: Լեզվի ստեղծման վրա Ռոսսումը սկսել է աշխատել 1989 թվականից: Python լեզվի անվանումը վերցվել է BBC «Մոնթի Փայթոնի թռչող կրկեսը» (Monty Python's Flying Circus) հեռուստատեսային հումորային շոուից, որի մեծ երկրպագուն էր ծրագրավորողը:[1]

Python լեզվի առաջին տարբերակը (տարբերակ 0.9.0) հրապարակվել է 1991 թվականին: Այն ներառում էր լեզվի մի շարք օգտակար հնարավորություններ, օրինակ՝ տվյալների տարբեր տեսակներ, սխալների մշակման ֆունկցիաներ և այլն:

Հետագայում ներկայացվեցին Python 1.0 (1994 թվական), Python 2.0 (2000 թվական), Python 3.0 (2008 թվական) տարբերակները, որոնք համալրվել էին նոր հնարավորություններով: Python 3.0-ում հնարավորինս (բայց ոչ լրիվ) պահպանվել է համատեղելիությունը Python-ի հին տարբերակների հետ:

Ըստ IEEE Spectrum-ի (Աշխարհի ինժեներական և տեխնոլոգիական համայնքի պրոֆեսիոնալ տուն՝ The professional home for the engineering and technology community worldwide, https://www.ieee.org/) վարկանիշային գնահատման սանդղակի՝ 2023 թվականին ամենապահանջված ծրագրավորման լեզուների տասնյակի առաջատարը Python-ն է, որին հաջորդում են Java, C++, C, JavaScript և այլ լեզուներ:[2] Ըստ տարբեր գնահատումների՝ Python ծրագրավորման լեզուն իր դիրքերն ամրապնդել է երկարաժամկետ հեռանկարում:

Python-ն օբյեկտ-կողմնորոշված ընդհանուր նշանակության բարձր մակարդակի ծրագրավորման լեզու է: Դրա առավելու-

[1] https://aws.amazon.com/ru/what-is/python/

[2] https://spectrum.ieee.org/the-top-programming-languages-2023

թյուններից են ծրագրի արագ մշակումը, կոդի հեշտ ընթեռնելիությունը: Python-ը նաև հեշտ ուսուցանվող լեզու է սկսնակների համար: Python լեզվի մշակողները հետևել են ծրագրավորման որոշ փիլիսոփայության, որն անվանել են «The Zen of Python»:[3] Այդ փիլիսոփայության հեղինակը Թիմ Փեյթերսն է (Tim Peters): «The Zen of Python» փիլիսոփայության դրույթներն են.

- գեղեցիկն ավելի լավ է, քան տգեղը,
- հստակն ավելի լավ է, քան ոչ հստակը,
- պարզն ավելի լավ է, քան բարդը,
- բարդն ավելի լավ է, քան խճճվածը,
- հարթն ավելի լավ է, քան ծալվածը,
- նոսրն ավելի լավ է, քան խիտը,
- ընթերցելիությունը կարևոր է,
- հատուկ դեպքերը բավականաչափ հատուկ չեն կանոնները խախտելու համար,
- այս դեպքում գործնական լինելն ավելի կարևոր է, քան անթերի լինելը,
- սխալները երբեք չպետք է անտեսվեն, եթե դրանք բացահայտորեն չեն անտեսվում,
- հանդիպելով երկիմաստության, թո՛ղ կռահելու գայթակղությունը,
- պետք է լինի մեկ, և ցանկալի է միայն մեկ, ակնհայտ եղանակ դա անելու համար,
- չնայած սկզբում դա կարող է ակնհայտ չլինել, եթե, իհարկե, դուք հոլանդացի չեք,
- հիմա ավելի լավ է, քան երբեք,
- չնայած երբեքը հաճախ ավելի լավ է, քան հենց հիմա,
- եթե իրականացումը դժվար է բացատրել, ապա գաղափարը վատն է,
- եթե իրականացումը հեշտ է բացատրել, ապա գաղափարը, հավանաբար, լավն է,

[3] https://peps.python.org/pep-0020/
«The Zen of Python» փիլիսոփայության դրույթների բնօրինակ տեքստը տե՛ս this մոդուլի ներբեռնմամբ:

- անվանատարածքները հիանալի են։ եկեք ավելին անենք դրանցից։

Python-ի կառուցվածքային հիմնական գծերն են տիպերի դինամիկ որոշումը, հիշողության ավտոմատ կառավարումը, բացառությունների մշակման հզոր մեխանիզմը և այլն։ Այն ինտերպրետացվող ծրագրավորման լեզու է։

Python-ն ունի հզոր գրադարաններ, դրանցից ամենատարածվածներն են՝ Pillow, Math, Numpy, Django, Pandas, Spark, Matplotlib, Tensorflow, Natural Language Toolkit (NLTK), Scikit-Learn, PyTorch, OpenCV և այլն։

Python-ն ունի կիրառման լայն հնարավորություններ։ Այն կիրառվում է բազմաթիվ ոլորտներում, այդ թվում՝ գիտական հաշվարկային ծրագրերի մշակման, խաղերի ստեղծման, արհեստական բանականության (Artificial Intelligence), մեքենայական ուսուցման (Machine Learning), տվյալագիտության (Data Science), պատկերների մշակման (Image Processing) մեջ, Backend վեբ ծրագրավորման մեջ։ Python-ն ունի Framework-եր՝ վեբ կայքեր պատրաստելու համար, որոնցից ամենատարածվածներն են Django-ն և Flask-ը։ Python-ը նաև կարևոր դեր է խաղում DevOps-ում և Infrastructure as Code (IaC)-ում՝ օգտագործելով Ansible և Terraform-ի նման գրադարանները (տե՛ս *գլուխ 16*)։

Ձեռնարկում ուսումնական նյութը ներկայացված է հետևյալ բաժիններով՝ Python լեզվում տվյալների ներդրված տիպերը, օպերատորները, հրամանները, լեզվում ներառված հիմնական կառուցվածքները՝ տողերը, միաչափ և երկչափ ցուցակները, հավաքածուները, բազմությունները, բառարանները, ինչպես նաև այդ կառուցվածքների հետ կապված լեզվում ներդրված մեթոդներն ու ֆունկցիաները։ Ներկայացված է պրոցեդուրային ծրագրավորման սկզբունքը օգտվողի ստեղծած ֆունկցիաների մեխանիզմի հիման վրա։ Նկարագրված են ֆայլերի հետ աշխատանքի եղանակները, լեզվի ստանդարտ համակարգային մոդուլները և օգտվողի կողմից մոդուլների ստեղծման միջոցները, Python-ծրագրերում բացառիկ իրավիճակների հայտնաբերման և մշակման մեխանիզմները։ Ներկայացված են օբյեկտ-կողմնորոշված

ծրագրավորման հիմնական հասկացությունները և սկզբունքները, Python լեզվով դասերի և դրանց հիերարխիայի ստեղծման և կիրառման եղանակները։ Յուրաքանչյուր բաժնում շարադրված է տեսական նյութը, որն ուղեկցվում է օրինակներով և ծրագրային կոդերով։ Ձեռնարկի «Խնդրագիրք» բաժնում ներառված են խնդիրներ ինքնուրույն իրականացնելու համար։

ԳԼՈՒԽ 1. PYTHON ԾՐԱԳՐԱՎՈՐՄԱՆ ՆԵՐԱԾՈՒԹՅՈՒՆ

1.1. Python-ի ներբեռնումը և տեղադրումը: IDLE

Python-ը կարելի է անվճար ներբեռնել և տեղադրել http://python.org պաշտոնական կայքից: Ազատ արտոնագրի շնորհիվ այն օգտագործվում է առանց սահմանափակումների: Աշխատելու համար մեզ անհրաժեշտ կլինի Python-3-ի որևէ տարբերակ (գրքում օգտագործվել է Python-3.13.0-ի տարբերակը):[4]

Python-ը համակարգչում տեղադրելուց (install) հետո կարելի է այն գործարկել: Python ծրագրերը կատարվում են ինտերպրետատորի (interpreter) կողմից: Python ինտերպրետատորը ծրագիր է, որը վերլուծում և կատարում է Python կոդը: Unix և Linux օպերացիոն համակարգերում ինտերպրետատոր կարելի է մուտք գործել՝ մուտքագրելով python հրամանը: Windows և Macintosh համակարգերում ինտերպրետատորը կարող է գործարկվել որպես հավելված Start ընտրացանկից կամ՝ կրկնակի սեղմելով ինտերպրետատորի պատկերի վրա:

Python-ը տեղադրելուց հետո կարելի է գործարկել IDLE միջավայրը (Integrated Development and Learning Environment՝ Զարգացման և ուսուցման ինտեգրված միջավայր):

IDLE միջավայրի բաղադրիչներից է IDLE shell ինտերակտիվ մեկնաբանիչը, որը կոչվում է նաև ինտերակտիվ թարգմանիչ (IDLE Interactive Shell): Այն թույլ է տալիս օգտատերերին Python կոդը մուտքագրել և կատարել տող առ տող ու անմիջապես տեսնել արդյունքները, ինչը այն դարձնում է ուսուցման, թեստավորման և կոդը փորձարկելու հիանալի գործիք: IDLE միջավայրի մյուս բաղադրիչը կոդերի խմբագրիչն է, որը թույլ է տալիս գրել, խմբագրել, պահպանել (որպես .py ֆայլեր) և գործարկել Python-ի ամբողջական ծրագրեր կամ սկրիպտներ: IDLE-ում ինտեգրված է նաև կոդերի վրիպազերծիչը (debugger), որը հեշտացնում է սխալների հայտնաբերումն ու ուղղումը:

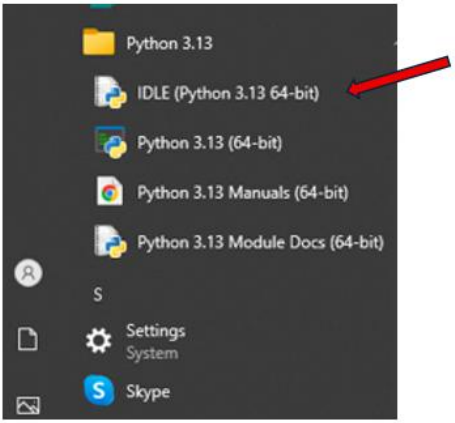
[4] 2025 թվականի հոկտեմբերի 7-ին թողարկվել է Python 3.14.0 տարբերակը:

IDLE-ն միջպլատֆորմային է, այն աշխատում է Windows, macOS և Linux համակարգերում: IDLE-ի առավելություններից են դրա փոքր չափերն ու արագագործությունը: Բացի դրանից, այն առանձին տեղադրում չի պահանջում, քանի որ համակարգչում տեղադրվում է Python-ի հետ միասին:

IDLE-ի ֆունկցիոնալությունը շատ ավելի ցածր է այնպիսի IDE-ներից, ինչպիսիք են PyCharm-ը կամ VS Code-ը և հարմար չէ մեծաքանակ ֆայլերով մեծ նախագծերի համար:

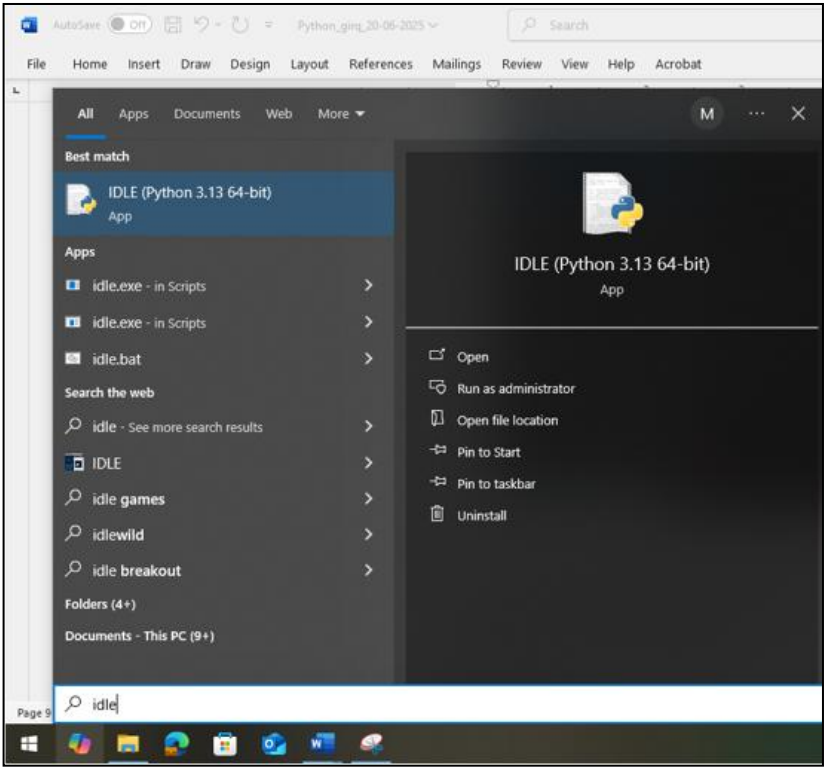
Anaconda Python-ը անվճար, բաց կոդով հարթակ է, որը հնարավորություն է տալիս գրել և գործարկել Python ծրագրավորման լեզվով գրված կոդը: Այն ներառում է Python ինտերպրետատորը, ամենահաճախ օգտագործվող գրադարանների մի շարք և մշակման ու կատարման հարմար միջավայր: Anaconda-ն այն ամենատարածված հարթակն է, որի օգնությամբ կարելի է սովորել և կիրառել Python-ը տվյալագիտության, գիտական հաշվարկների և մեքենայական ուսուցման ոլորտներում: Anaconda հարթակը հասանելի է ինչպես Windows-ի, այնպես էլ macOS-ի և Linux-ի համար:

Ինչպես նշեցինք, Windows համակարգում Python ինտերպրետատորը կարող է գործարկվել որպես հավելված՝ Start ընտրացանկից կամ կրկնակի սեղմելով դրա պատկերի վրա (նկար 1.1):



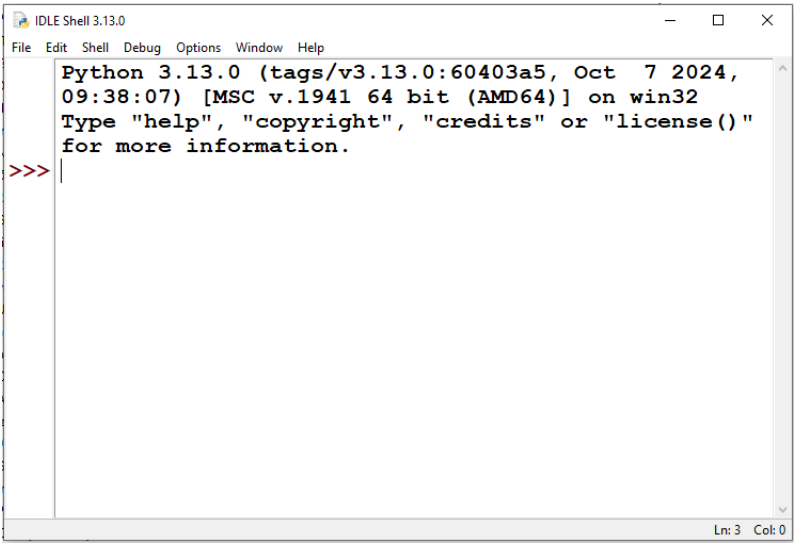
Նկար 1.1. Python ինտերպրետատորի գործարկումը Start ընտրացանկից

Ինտերպրետատորը կարող է գործարկվել նաև Start հրամանացանկի կողքին գտնվող որոնման տողից (Search)՝ մուտքագրելով idle (նկար 1.2)։



Նկար 1.2. Python ինտերպրետատորի գործարկումը Search տողից

Python ինտերպրետատորը գործարկելուց հետո բացվում է հետևյալ IDLE Shell 3.13.0 պատուհանը (նկար 1.3)։



Նկար 1.3. IDLE Shell 3.13.0 պատուհանը

Պատուհանում պարունակվում են տեղեկություններ ինտերպրետատորի մասին: Տվյալ դեպքում հաղորդվում է, որ օգտագործվում է Python 3.13.0 տարբերակը:

Վերջին տողում ինտերպրետատորը օգտվողին տրամադրում է աշխատանքի հրավեր՝ (>>>) նշանի տեսքով:

IDLE-ն աշխատում է երկու ռեժիմով՝ ինտերակտիվ ռեժիմ (Shell Mode) և խմբագրման ռեժիմ (Edit Mode): Դիտարկենք դրանցից յուրաքանչյուրի աշխատանքը մանրամասնորեն:

Ինտերակտիվ ռեժիմ

Ինտերակտիվ ռեժիմում մուտքագրված յուրաքանչյուր հրաման անմիջապես կատարվում է՝ առանց այն ֆայլում պահելու: Python Shell-ը կարելի է օգտագործել որպես հաշվիչ, որը կատարում է տարբեր թվային հաշվարկներ: Հաշվարկներ կատարելիս կարող են օգտագործվել Python-ում ներդրված բազմաթիվ ֆունկցիաներ: Ինտերակտիվ ռեժիմն օգտագործվում է կոդն արագ փորձարկելու և լեզուն սովորելու համար: Այն կարելի է օգտագործել փոփոխականների արժեքները դիտելու, կոդում սխալները հայտնաբերելու, ուղղելու համար, ինչպես նաև՝ կարելի է

փորձարկել գրադարաններն ու մոդուլները՝ մինչև նախագծում դրանց ներդնելը:

Python-ի յուրաքանչյուր հրաման գրվում է հրամանի պատուհանում, հրավերի (>>>) նշանից հետո: Հրամանը կատարվում է, երբ այն գրելուց հետո սեղմվում է Enter ստեղնը: Հրամանի իրականացումից հետո նոր տողում կրկին հայտնվում է աշխատանքի հրավերի նշանը, որը նշանակում է, որ կարելի է գրել նոր հրաման:

Ինտերակտիվ ռեժիմը կոչվում է նաև Python REPL (read-eval-print loop). read՝ ինտերպրետատորը կարդում է հրամանները, eval-ը՝ կատարում է դրանք, print-ը՝ արտածում է արդյունքը, loop-ը՝ կրկնում է հրամանների այս ցիկլը:

Ինտերակտիվ ռեժիմը բացվում է լռելյայն (default), երբ գործարկվում է IDLE-ն: Իսկ եթե գտնվում ենք խմբագրման ռեժիմում, ապա IDLE-ին անցնելու համար հրամանների ընտրացանկից պետք է ընտրել Run -> Python Shell հրամանը:

Python Shell-ը կարող է վերագործարկվել՝ առանց ամբողջ IDLE միջավայրը փակելու: Դրա համար անհրաժեշտ է հիմնական ընտրացանկից ընտրել Shell -> Restart Shell հրամանը կամ միաժամանակ սեղմել Ctrl + F6 ստեղները: Այդ դեպքում բոլոր նախորդ արդյունքները, փոփոխականները կվերանան, և աշխատանքը կարելի է սկսել մաքուր էջից:

Որպես պարզագույն օրինակ՝ IDLE ինտերակտիվ ռեժիմում հաշվենք $(7 + 5.1) \times 6$ և $15 - 2/4$ թվաբանական արտահայտությունների արժեքները:

Նախ հրամանի տողում հավաքենք (7+5.1)*6 արտահայտությունը և սեղմենք Enter ստեղնը մուտքագրումն ավարտելու համար: Հաշվարկի արդյունքն անմիջապես կհայտնվի հաջորդ տողում՝ 72.6, որից հետո կբերվի աշխատանքի նոր հրավեր (>>>), որը կնշանակի, որ Python–ը սպասում է հաջորդ հրամանին:

Այժմ հրամանի տողում հավաքենք 15-2/4 արտահայտությունը: Enter ստեղնը սեղմելուց հետո հաջորդ տողում կարտածվի արդյունքը՝ 14.5, և կրկին կհայտնվի աշխատանքի նոր հրավեր:

Որևէ սխալ հրաման մուտքագրելու դեպքում էկրանին կտրվի հաղորդագրություն սխալի և դրա պատճառների մասին: Օրինակ՝ փորձենք արտածել b փոփոխականի արժեքը, որին որևէ արժեք վերագրված չէ: Այդ դեպքում, կտրվի հաղորդագրություն թույլ տրված սխալի մասին՝ NameError: name 'b' is not defined.

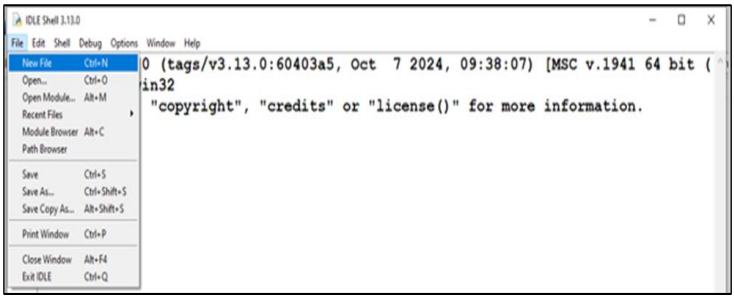
```
>>> b
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    b
NameError: name 'b' is not defined
>>>
```

Խմբագրման ռեժիմ

Ինտերակտիվ ռեժիմում կարելի է գրել ու գործարկել պարզ ծրագրեր: Ավելի ծավալուն ու բարդ ծրագրեր գրելու համար օգտագործվում է խմբագրման ռեժիմը (Edit Mode), որտեղ կարելի է ծրագիրը պահպանել հիշողության կրիչի վրա:

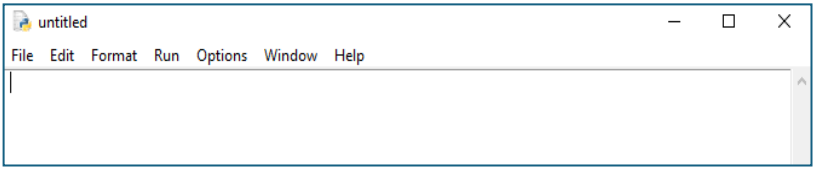
Խմբագրման ռեժիմ մտնելու համար անհրաժեշտ է ստեղծել նոր ֆայլ կամ բացել արդեն գոյություն ունեցող որևէ ֆայլ:

Նոր ֆայլ ստեղծելու համար IDLE Shell պատուհանում պետք է ընտրել File->New File հրամանը կամ ստեղնաշարի վրա միաժամանակ սեղմել <Ctrl+N> ստեղները (նկար 1.4):



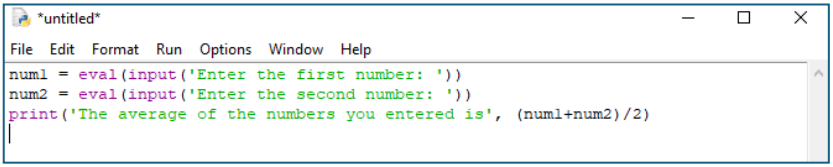
Նկար 1.4. Նոր ֆայլի ստեղծումը

Երկու տարբերակով էլ կբացվի կոդի խմբագրման ռեժիմի untitled վերնագրով պատուհանը (նկար 1.5)։



Նկար 1.5. Խմբագրման ռեժիմի untitled պատուհանը

Գրենք Python լեզվով մեր առաջին ծրագիրը, որը հաշվում է մուտքագրված երկու թվերի միջին թվաբանականը (նկար 1.6)։



Նկար 1.6. Երկու թվերի միջին թվաբանականը հաշվող ծրագիրը

Քննարկենք, թե ինչպես է աշխատում ծրագիրը։ Առաջին և երկրորդ տողերում գրված հրամաններն առաջարկում են ներածել երկու թիվ input() ֆունկցիայի միջոցով։

Python-ի input() ներկառուցված ֆունկցիան օգտագործվում է ստեղնաշարից տվյալներ մուտքագրելու համար։ Ֆունկցիայի շարահյուսությունն է՝

input (տեքստ),

որտեղ տեքստը մուտքագրումից առաջ ծրագրորդին ուղղորդվող հաղորդագրություն է (ոչ պարտադիր)։ input() ֆունկցիան ընդհատում է ծրագրի կատարումը, ցուցադրում է տեքստը (եթե տրամադրված է) և սպասում, որ ծրագրորդն ինչ-որ բան մուտքագրի և սեղմի Enter։ Անկախ նրանից, թե մուտքագրվում է տառեր, թվեր, թե սիմվոլներ, Python-ը միշտ լռելյայն այն պահպանում է որպես տող։ Օրինակ, եթե ծրագրորդը մուտքագրում է 3 + 4, input() ֆունկցիան վերադարձնում է "3 + 4" տողը։ Մուտքագրված տվյալը կարելի է փոխակերպել թվային տիպի int() կամ float() ֆունկցիաների միջոցով։

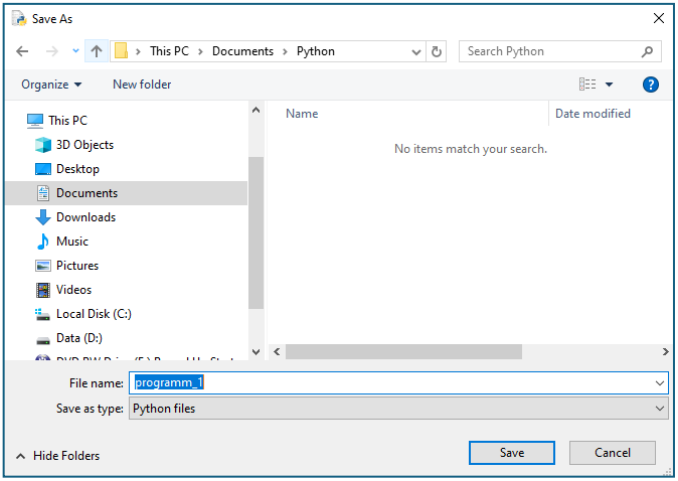
Ներկայացված ծրագրում Python-ի eval(input()) հրամանում կա երկու ներկառուցված ֆունկցիա՝ input() և eval()։ input() ֆունկցիան վերադարձնում է մուտքագրված տվյալը որպես տող, իսկ eval() ֆունկցիան վերադարձված տողը գնահատում է որպես Python-ի արտահայտություն, ապա վերադարձնում է այդ արտահայտության արժեքը։

Ծրագրում input() ֆունկցիայի արգումենտն ապաթարցերի միջև գրված տեքստ է, որը հուշում է ծրագրորդին, թե ո՞ր գումարելին է անհրաժեշտ մուտքագրել։ eval() ֆունկցիան վերլուծում է մուտքագրված տողը, հաշվում գնահատված արտահայտության արժեքը և այն վերագրում num1 (num2) փոփոխականին։

print() ֆունկցիան հաշվարկի արդյունքն արտածում է պատուհանում։ Ֆունկցիայի առաջին արգումենտը տեքստ է, որը գրված է ապաթարցերի մեջ՝ 'The average of the numbers you entered is '։ Երկրորդ արգումենտը արտահայտություն է, որով հաշվվում է num1 և num2 փոփոխականների արժեքների միջին թվաբանականը, ապա արտածվում է այն։

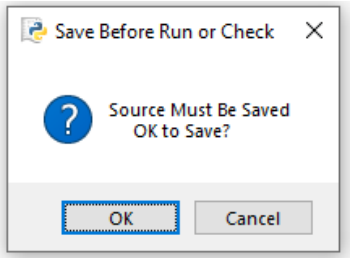
Երբ պատուհանում ծրագրորդը սկսում է հավաքել ծրագրի կոդը, untitled վերնագիրը վերցվում է * * սիմվոլների մեջ, որը հուշում է, որ ծրագիրը դեռևս պահպանված չէ հիշողության մեջ։ Ծրագիրը հիշողության մեջ կարելի է պահել File -> Save As հրամանով կամ՝ միաժամանակ սեղմելով ստեղնաշարի <Ctrl+Sfift+S> ստեղները։ Բոլոր նշված տարբերակներով էլ կհայտնվի Save As պատուհանը, որի File name տողում պետք է

մուտքագրել ֆայլի անունը և ընտրել այն պանակը, որի մեջ պետք է ֆայլը տեղադրվի (նկար 1.7)։ Այն դեպքում, երբ ծրագիրը դեռևս պահպանված չէ հիշողության մեջ, Save հրամանը (<Ctrl+S> ստեղները) համարժեք է Save As հրամանին։



Նկար 1.7. Ֆայլի պահպանումը

Ծրագիրը գործարկելու համար անհրաժեշտ է ընտրացանկից ընտրել Run → Run Module հրամանը կամ սեղմել F5 ստեղնը։ Այս հրամանի ընտրությամբ, եթե ծրագիրը դեռևս պահպանված չէ կամ վերջին արված փոփոխությունը գրանցված չէ, Python-ը նախ առաջարկում է պահպանել ֆայլը։ Այդ դեպքում ծրագիրը պահպանելու և կատարելու համար պետք է սեղմել Ok (նկար 1.8)։

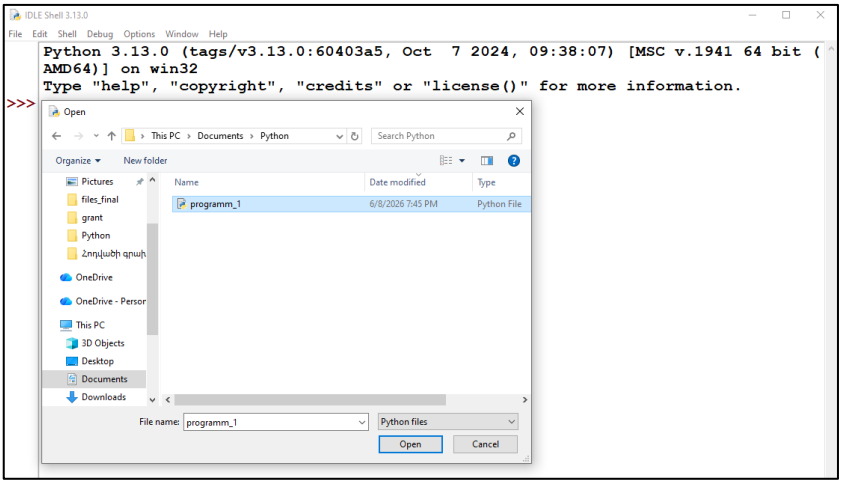


Նկար 1.8. Ֆայլի պահպանումը կատարելուց առաջ

Անվանենք մեր առաջին ծրագիրը programm_1 և այն պահպանենք Python անունով պանակում: Համակարգը ֆայլի անվանը ավելացնում է .py ընդլայնումը: Լռելյայնորեն Python ֆայլերը տեղադրվում են Python-ի համակարգային պանակում:

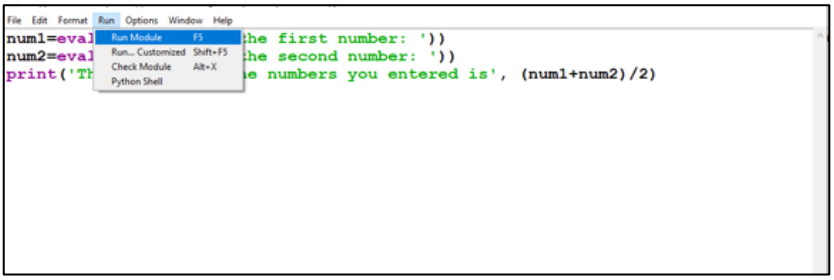
Գոյություն ունեցող Python ֆայլը բացելու համար IDLE Shell պատուհանում պետք է ընտրել File->Open հրամանը կամ ստեղնաշարի վրա միաժամանակ սեղմել <Ctrl+O> ստեղները, այնուհետև բացված Open պատուհանից ընտրել անհրաժեշտ ֆայլը: Ֆայլը կարելի է բացել նաև՝ կրկնակի սեղմելով դրա պիտակի վրա:

Գործարկենք programm_1 ծրագիրը՝ IDLE Shell պատուհանի File-> Open հրամանի միջոցով (նկար 1.9):



Նկար 1.9. Ֆայլի բացումը

Գործողության արդյունքում programm_1 ծրագիրը կբացվի ինտերպրետատորի պատուհանում: Այժմ գործարկենք ծրագիրը Run->Run Module հրամանով (նկար 1.10):



Նկար 1.10. Ծրագրի գործարկումը Run->Run Module հրամանով

Ծրագիրը կառաջարկի մուտքագրել առաջին թիվը՝

Enter the first number:

Մուտքագրենք, օրինակ, 15 թիվը և սեղմենք Enter ստեղնը: Մուտքագրված արժեքը կվերագրվի num1 փոփոխականին:

Ապա ծրագիրը կառաջարկի մուտքագրել երկրորդ թիվը՝

Enter the second number:

Մուտքագրենք 32 թիվը: Enter ստեղնը սեղմելուց հետո num2 փոփոխականը կստանա 32 արժեքը: Հաջորդ հրամանով կհաշվարկվի 15 և 32 թվերի միջին թվաբանականը, որն էլ կարտածվի պատուհանում.

The average of the numbers you entered is 23.5:

Խնդրի լուծումը ներկայացված է ստորև բերված պատուհանում (նկար 1.11):

```
Enter the first number: 15
Enter the second number: 32
The average of the numbers you entered is 23.5
```

Նկար 1.11. Ծրագրի գործարկման արդյունքը

ԳԼՈՒԽ 2. PYTHON ԼԵԶՎԻ ՏԱՐՐԵՐԸ

2.1. Python լեզվի այբուբենը

Այբուբենը տվյալ լեզվում ընդունված այն սիմվոլների բազմությունն է, որոնց միջոցով գրվում են ծրագրերի տեքստերը:

Python լեզվի այբուբենի մեջ մտնում են.

- լատինական այբուբենի փոքրատառերը և մեծատառերը՝
  a b c d e f g h i j k l m n o p q r s t u v w x y z;
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z;
- արաբական թվանշանները՝ 0 1 2 3 4 5 6 7 8 9;
- հատուկ սիմվոլները՝
  ! @ # $ % ^ * ( ) - _ = + | \ [ ] { } : ; “ ‘ < > , . ? / &
- բացակային սիմվոլները՝ բացակ, տաբուլացիա, նոր տողին անցնելու սիմվոլը:

Python լեզվի այբուբենի տարրերի միջոցով կառուցվում են իդենտիֆիկատորները, արտահայտությունները, օպերատորները:

2.2. Իդենտիֆիկատորներ

Իդենտիֆիկատորը ծրագրային օբյեկտի (փոփոխական, ֆունկցիա, դաս, մոդուլ և այլն) անուն է, որը կառուցվում է հետևյալ կանոնների համաձայն.

- իդենտիֆիկատորը կարող է պարունակել լատինական այբուբենի մեծատառեր կամ փոքրատառեր (A-Z, a-z), արաբական թվանշաններ (0 - 9), ընդգծման նշան (_):
- իդենտիֆիկատորը կարող է սկսվել տառով կամ ընդգծման նշանով,
- իդենտիֆիկատորը չի կարող համընկնել Python-ի բանալի բառերի հետ:

Օրինակ, Number_1, lentgth, tiv բառերը իդենտիֆիկատորներ են, իսկ 1rate, a% բառերը՝ ոչ:

Python-ը տառաշարազգայուն (case-sensitive) լեզու է, այսինքն՝ այն տարբերում է մեծատառերն ու փոքրատառերը: Ուս-

տի newprice, NewPrice, NEWprice իդենտիֆիկատորները Python լեզվում տարբեր ծրագրային օբյեկտների անուններ են:

Ծրագրում օբյեկտներին ցանկալի է տալ իմաստալից անուններ, այսինքն՝ սահմանել այնպիսի իդենտիֆիկատորներ, որոնք արտացոլում են օբյեկտների էությունը և հեշտացնում ծրագրի ընթերցումը: Ընդունված է նաև անունը կազմող բառերն իրարից առանձնացնել ընդգծման նշանով: Օրինակ՝ Sum_of_numbers, Product_Price և այլն:

2.3. Բանալի բառեր

Python լեզվում կան բառեր, որոնք ունեն նախապես սահմանված հատուկ իմաստ և օգտագործվում են միայն այդ իմաստով: Դրանք կոչվում են բանալի բառեր (keyword): Այդ բառերը չեն կարող օգտագործվել որպես փոփոխականների իդենտիֆիկատորներ, ֆունկցիաների, դասերի կամ այլ օբյեկտների անունների համար:

Python-ի բոլոր բանալի բառերի ցուցակը կարելի է ստանալ հետևյալ հրամանի միջոցով.

```
>>> help("keywords")
```

կամ՝

```
>>> import keyword
>>> print(keyword.kwlist)
```

Արդյունքում կստանանք Python-ի բանալի բառերի ցուցակը.

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

2.4. Մեկնաբանություններ

Ծրագրի ընթերցումը և ընկալումը դյուրին դարձնելու, ծրագրի որևէ մասի աշխատանքը նկարագրելու համար գրվում են մեկնաբանություններ: Մեկնաբանությունները, ինչպես նաև դատարկ տողերը, ծրագրի մաս չեն կազմում, և ծրագրի կատարման ընթացքում անտեսվում են Python-ի ինտերպրետատորի կողմից: Կարելի է գրել միատող և բազմատող մեկնաբանություններ:

Միատող մեկնաբանությունը սկսվում է վանդականիշով (#), որին հաջորդող բոլոր սիմվոլները մինչև տողի վերջն ընկալվում են որպես մեկնաբանություն: Օրինակ՝

```
# This is a single line comment.
```

Բազմատող մեկնաբանությունը գրվում է մի քանի տողի վրա: Այն վերցվում է եռակի ապաթարցերի (''') կամ եռակի չակերտների մեջ ("""): Օրինակ՝

```
"""
This program calculates the sum, and arithmetic mean of even elements in a vector.
If the vector does not contain even numbers, a message is displayed to that effect.
"""
```

Բազմատող մեկնաբանության յուրաքանչյուր տող կարելի է դիտարկել որպես միատող մեկնաբանություն՝ տողի սկզբում դնելով վանդականիշ (#) սիմվոլը:

Մեկնաբանությունները կարող են օգտագործվել ծրագիրը գործարկելու կամ փոփոխելու ժամանակ կողի որևէ մաս անջատելու համար՝ սահմանելով այն որպես մեկնաբանություն:

ԳԼՈՒԽ 3. PYTHON ԼԵԶՎԻ ՏՎՅԱԼՆԵՐԻ ՏԻՊԵՐԸ: ՓՈՓՈԽԱԿԱՆՆԵՐ

3.1. Տվյալների տիպերը Python-ում

Python-ը օբյեկտ-կողմնորոշված ծրագրավորման լեզու է: Օբյեկտ-կողմնորոշված ծրագրավորման հիմնական հասկացություններից է դասը (class): Դասը տվյալների տիպ է, որի հիման վրա ստեղծվում է օբյեկտը: Օբյեկտն այդ տիպին համապատասխանող տվյալների ներկայացուցիչն է:

Python-ում բոլոր տվյալները օբյեկտներ են, դրանք կա՛մ Python-ում ներկառուցված օբյեկտներ են, կա՛մ էլ սահմանված են օգտագործողի կողմից:

Տվյալները կարող են լինել տարբեր տիպերի: Օրինակ, ապրանքի գինը կարող է պահվել որպես թվային տիպի օբյեկտ, իսկ մարդու ազգանունը՝ որպես տառային սիմվոլների հաջորդականություն՝ տողային տիպի օբյեկտ:

Տվյալների տիպը որոշում է համակարգչի հիշողության մեջ տվյալների ներքին ներկայացումը, այն արժեքների բազմությունը, որոնք կարող են ընդունել այդ տիպի տվյալները, ինչպես նաև այդ արժեքների հետ թույլատրվող գործողությունները:

Python-ը տիպերի դինամիկ որոշման լեզու է: Տիպերի որոշումը կարող է լինել ստատիկ կամ դինամիկ: Առաջին դեպքում տիպը որոշվում է կոմպիլյացիայի ժամանակ, իսկ երկրորդում՝ ծրագրի կոդի կատարման ժամանակ: Որոշ ծրագրավորման լեզուներում (C, C++ և այլն) անհրաժեշտ է նախօրոք հայտարարել փոփոխականի տիպը (ստատիկ որոշում): Ի տարբերություն դրա՝ Python-ը տիպերի դինամիկ որոշման լեզու է, այսինքն՝ փոփոխականի տիպը հայտնի է դառնում դրան վերագրված արժեքի հիման վրա: Այդ դեպքում, նույն փոփոխականը, երբ մի քանի անգամ արժեքավորվում է, կարող է ներկայացնել տարբեր տիպի օբյեկտներ:

Տիպերի դինամիկ որոշման առավելություններից է լեզվում տարբեր տիպի տարրերից կազմված հավաքածուների ստեղծ-

ման հնարավորությունը, ինչպես նաև՝ վերացարկումը ալգորիթմներում: Վերջինս հնարավորություն է ընձեռում գրել, օրինակ, ունիվերսալ ֆունկցիա տողերի և թվերի համար, և այն երկու դեպքում էլ կաշխատի:

Python-ում օբյեկտները բաժանվում են երկու խմբի՝ փոփոխվող (ստեղծվելուց հետո օբյեկտի արժեքը փոփոխելու հնարավորություն՝ առանց դրա ինքնությունը (հիշողության հասցեն) փոխելու) և չփոփոխվող: Օրինակ՝ տողերը (strings) չփոփոխվող օբյեկտներ են, և տողերի հետ գործողությունների արդյունքում ստեղծվում են նոր տողեր:

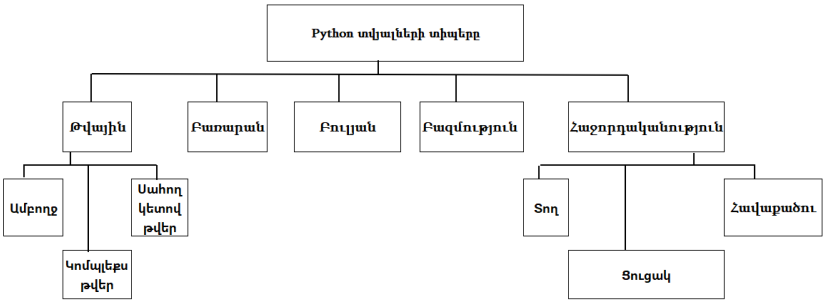
Python 3.13-ում հիմնական ներկառուցված տիպերն են՝

Փոփոխվող տիպեր	Չփոփոխվող տիպեր
list	int
dict	float
set	complex
	bool
	str
	tuple
	bytes
	frozen set

Python-ը տրամադրում է ներկառուցված օբյեկտների հզոր հավաքածու, որոնք օժտված են մեծ արդյունավետությամբ: Իհարկե, կարելի է ստեղծել այդ օբյեկտների սեփական իրականացումները, բայց այդ դեպքում պետք է գործադրել մեծ ջանքեր՝ հասնելու այն արդյունավետությանը, որն ապահովում են ներկառուցված օբյեկտների տեսակները:

Ստորև բերված են Python-ի ստանդարտ կամ ներկառուցված տվյալների տիպերը (նկար 3.1):

- Numeric (թվային) – int, float, complex,
- Sequence Type (հաջորդական) – string, list, tuple,
- Dictionary (բառարան) – (dict),
- Boolean (բուլյան) – (bool),
- Set Type (բազմություն) – set, frozenset:



Նկար 3.1. Python-ի ներկառուցված տվյալների տիպերը

Մինչև տվյալների տիպերի նկարագրությունը՝ ուսումնասիրենք, թե ինչպես են թվերը ներկայացվում համակարգչում։

3.2. Թվերի ներկայացումը համակարգչում

Համակարգչում ցանկացած տիպի տվյալ՝ թվային, տեքստային, գրաֆիկական կամ տեսաձայնային, ներկայացվում է երկուական կոդավորմամբ՝ զրոների ու մեկերի հաջորդականության տեսքով։ Այդ նշանները անգլերենում կոչվում են binary digit։ Այն տարրը, որը կարող է գտնվել երկու տարբեր վիճակներում (0 կամ 1), կոչվում է բիթ։

Ակնհայտ է, որ մեկ բիթի օգնությամբ կարելի է կոդավորել երկու ($2 = 2^1$) սիմվոլ (դրանք են՝ 0 կամ 1), երկու բիթի օգնությամբ՝ չորս ($4 = 2^2$) սիմվոլ (00, 01, 10 և 11), երեք բիթի օգնությամբ՝ ութ ($8 = 2^3$) սիմվոլ (000, 001, 010, 011, 100, 101, 110 և 111)։ Ընդհանրապես, m բիթի օգնությամբ կարելի է կոդավորել $N = 2^m$ սիմվոլ, իսկ N սիմվոլ կոդավորելու համար կպահանջվի $m = \log_2 N$ բիթ։

Այսպիսով, 8 բիթի կամ մեկ բայթի օգնությամբ կարելի է կոդավորել $2^8 = 256$ սիմվոլ։

Թվերի երկուական կոդավորում

Թվերի երկուական կոդավորման հիմքում ընկած է դրանց ներկայացումը երկուական հաշվարկման համակարգում։

Հաշվարկման համակարգը միջոց է թվերը գրառելու, կարդալու և հաշվարկներ կատարելու համար: Հաշվարկման համակարգն ունի հիմք (p) և այբուբեն (A_p), որը պարունակում է p սիմվոլ՝ այդ համակարգում թվերի գրառման համար:

Հաշվարկման համակարգերն անվանում են դիրքային, քանի որ յուրաքանչյուր թվանշանի արժեքը որոշվում է թվի գրառման մեջ այդ թվանշանի՝ տասնորդական կետի նկատմամբ ունեցած դիրքով: Օրինակ՝ 225.75 թվի գրառման ամենաձախ կարգում գրված 2 թվանշանը ցույց է տալիս հարյուրների քանակը, ձախից երկրորդ 2 թվանշանը՝ տասնյակների քանակը, ձախից երրորդ 5 թվանշանը՝ միավորների քանակը, տասնորդական կետից աջ գտնվող 7 թվանշանը՝ մեկ միավորի տասնորդական մասերի քանակը, իսկ վերջին 5 թվանշանը՝ մեկ միավորի հարյուրերորդական մասերի քանակը:

Հաշվարկման համակարգի օրինակ է տասական համակարգը, որի հիմքը՝ p =10, և այբուբենը պարունակում է տասը նիշ՝ A_p= {0, 1, …, 9}: Տասական համակարգում գրված թվի յուրաքանչյուր կարգի մեկ միավորը տասն անգամ մեծ է դրանից աջ գտնվող կարգի մեկ միավորից:

Ցանկացած թիվ տասական համակարգում կարելի է ներկայացնել 10-ի աստիճանների նկատմամբ բազմանդամի տեսքով՝

$(a_n a_{n-1}, \dots, a_1 a_0, a_{-1} a_{-2} \dots)_{10} = a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \dots + a_1 \cdot 10^1 + a_0 \cdot 10^0 + a_{-1} \cdot 10^{-1} + a_{-2} \cdot 10^{-2} \dots$,

որտեղ a_i-ն 10^i աստիճանի գործակիցն է: Այսպես օրինակ՝

$225.75 = 2 \cdot 10^2 + 2 \cdot 10^1 + 5 \cdot 10^0 + 7 \cdot 10^{-1} + 5 \cdot 10^{-2}$:

Նույն 225.75 թիվը երկուական համակարգում գրելու համար այն կներկայացնենք 2-ի աստիճանների նկատմամբ բազմանդամի տեսքով՝

$225.75 = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$:

Այսպիսով, $(225.75)_{10} = (11100001.11)_2$:

Թվերի թարգմանությունը մի համակարգից մյուսը

Ներկայացնենք թվերի թարգմանության ալգորիթմը կամայական p հիմքով հաշվարկման համակարգից՝ 10-ական համակարգի։

2-ական, 8-ական, 10-ական և 16-ական հաշվարկման համակարգերի այբուբենները ներկայացված են ստորև.

A_2 ={0, 1}

A_8 ={0, 1, 2, 3, 4, 5, 6, 7}

A_{10} ={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

A_{16} ={0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Հիշեցնենք, որ p հիմքով հաշվարկման համակարգում.

- թվի գրառման համար օգտագործվում են A_p այբուբենի p սիմվոլ,
- թվի յուրաքանչյուր կարգի մեկ միավորը p անգամ մեծ է իրենից աջ գտնվող կարգի մեկ միավորից,
- թիվը կարելի է ներկայացնել p-ի աստիճանների նկատմամբ բազմանդամի տեսքով.

$$x = (a_n a_{n-1}, \dots, a_1 a_0, a_{-1} a_{-2} \dots)_p = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \dots + a_1 \cdot p^1 + a_0 \cdot p^0 + a_{-1} \cdot p^{-1} + a_{-2} \cdot p^{-2} \dots,$$

որտեղ $a_i \in A_p$:

Որևէ p հիմքով հաշվարկման համակարգում գրված թվի տասական ներկայացումը ստանալու համար բավական է կատարել վերևում ներկայացված գործողությունները։ Օրինակ՝

$$(AB.C4)_{16} = 10 \cdot 16^1 + 11 \cdot 16^0 + 12 \cdot 16^{-1} + 4 \cdot 16^{-2} = 160 + 11 + 0.75 + 0.015625 = (171.765625)_{10}$$

$$(707.6)_8 = 7 \cdot 8^2 + 0 \cdot 8^1 + 7 \cdot 8^0 + 6 \cdot 8^{-1} = 448 + 0 + 7 + 0.75 = (455.75)_{10}$$

$(11001.01)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 16 + 8 + 0 + 0 + 1 + +0 + 0.01 = (25.01)_{10}$:

Այժմ ներկայացնենք տասական համակարգից՝ p հիմքով այլ համակարգի թվերի թարգմանության ալգորիթմը։ Նշենք, որ թվի ամբողջ և կոտորակային մասերի թարգմանության համար օգտագործվում են տարբեր ալգորիթմներ։

Ամբողջ թվերի թարգմանության ալգորիթմը

1. Բաժանել տրված ամբողջ թիվը p-ի, վերցնել բաժանումից ստացված մնացորդը (մնացորդները ստացվում են տասական համակարգում)։

2. Եթե քանորդը հավասար է 0-ի, ապա թիվը թարգմանված է։ Այն ստանալու համար անհրաժեշտ է մնացորդները գրել հակառակ հերթականությամբ, այսինքն՝ առաջին ստացված մնացորդը գրել վերջին տեղում՝ թվի փոքրագույն կարգում, իսկ վերջինը՝ ավագ կարգում։

3. Եթե քանորդը հավասար չէ 0-ի, ապա վերցնել այդ քայլում թվի բաժանումից ստացված քանորդը և վերադառնալ առաջին քայլին։

Կոտորակների թարգմանության ալգորիթմը

1. Բազմապատկել տրված թվի կոտորակային մասը p-ով և վերցնել ստացված թվի ամբողջ մասը։

2. Եթե ստացված թվի կոտորակային մասը հավասար է 0-ի, ապա թիվը թարգմանված է՝ այն գրելու համար անհրաժեշտ է բազմապատկման արդյունքում ստացված ամբողջ մասերը դուրս գրել նույն հերթականությամբ, այսինքն՝ առաջին ստացված ամբողջ մասը գրել առաջին տեղում և այլն։

3. Եթե ստացված թվի կոտորակային մասը հավասար չէ 0-ի, ապա դրանից հանել ամբողջ մասը և վերադառնալ առաջին քայլին։

Նշենք, որ երկու դեպքում էլ հաշվարկները կատարվում են տասական համակարգում։

Օրինակ, ներկայացնենք 225.75 թիվը 2-ական, 8-ական և 16-ական համակարգերում՝ նախապես տրոհելով այն ամբողջ և կոտորակային մասերի՝ (225.75)10 = (225+0,75)10։

Թվի ներկայացումը 2-ական համակարգում	
Ամբողջ մասի ներկայացումը	Կոտորակային մասի ներկա-յացումը
225 ÷ 2 = 112(1) 112 ÷ 2 = 56(0) 56 ÷ 2 = 28(0) 28 ÷ 2 = 14(0) 14 ÷ 2 = 7(0) 7 ÷ 2 = 3(1) 3 ÷ 2 = 1(1) 1 ÷ 2 = 0(1)	0,75 × 2 = 1,5 (1) 0,5 × 2 = 1,0 (1)
Արդյունքը՝ 11100001	Արդյունքը՝ 0.11
Թիվը՝ $(225,75)_{10} = (225+0,75)_{10} = (11100001)_2 + (0,11)_2 = (11100001,11)_2$	

Թվի ներկայացումը 16-ական համակարգում	
Ամբողջ մասի ներկայացումը	Կոտորակային մասի ներ-կայացումը
225 ÷ 16 = 14(1) 14 ÷ 16 = 0(14)	0.75 × 16 = 12.0 (12)
Արդյունքը՝ E1	Արդյունքը՝ 0.C
Թիվը՝ $(225.75)_{10} = (E1)_{16} + (0.C)_{16} = (E1.C)_{16}$	

Թվի ներկայացումը 8-ական համակարգում	
Ամբողջ մասի թարգմանությունը	Կոտորակային մասի թարգմանությունը
225 ÷ 8 = 28(1) 28 ÷ 8 = 3(4) 3÷ 8 = 0(3)	0.75 × 8 = 6.0 (6)
Արդյունքը՝ 341	Արդյունքը՝ 0.6
Թիվը՝ $(225.75)_{10} = (341)_8 + (0.6)_8 = (34.6)_8$	

Նշենք, որ իրական թվի կոտորակային մասը թարգմանելիս հնարավոր է, որ ստացվի անվերջ կոտորակ, ուստի կոտորակնե-րը թարգմանելիս պետք է նշվի ստորակետից հետո անհրաժեշտ նիշերի քանակը։

Թվաբանական գործողությունները երկուական համակարգում

Ստորև բերենք զրոյից ինը թվանշանների ներկայացումը երկուական հաշվարկման համակարգում.

$0 = (0)_2$
$1 = 1 \cdot 2^0 = (1)_2$
$2 = 1 \cdot 2^1 + 0 \cdot 2^0 = (10)_2$
$3 = 1 \cdot 2^1 + 1 \cdot 2^0 = (11)_2$
$4 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = (100)_2$
$5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (101)_2$
$6 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (110)_2$
$7 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (111)_2$
$8 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = (1000)_2$
$9 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (1001)_2$

Երկուական հաշվարկման համակարգում թվաբանական գործողությունները կատարվում են երկուական թվաբանության ֆորմալ կանոններով՝ համաձայն ստորև ներկայացված աղյուսակների.

Գումարում

+	0	1
0	0	1
1	1	10

Բազմապատկում

×	0	1
0	0	0
1	0	1

Կիրառելով գումարման և բազմապատկման աղյուսակները՝ բազմապատկենք $(10)_{10} = (1010)_2$ և $(5)_{10} = (101)_2$ թվերը.

$$\begin{array}{rr} & 1010 \\ \times & 101 \\ \hline & 1010 \\ + & 1010 \\ \hline & 110010 \end{array}$$

Ստուգենք արդյունքը՝ $(10\times5)_{10} = (50)_{10}$,

$$(110010)_2 = 1\cdot2^5 + 1\cdot2^4 + 0\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 = (50)_{10}:$$

Նկատենք, որ գումարման ժամանակ 1 + 1 = 10, սակայն 1 միավորը փոխանցվում է ավագ կարգ: Հանման գործողության ժամանակ՝ 0 - 1= 1, քանի որ ավագ կարգից վերցվում է 1 միավոր: Երկուական համակարգում կատարենք հետևյալ հանման գործողությունը՝ $(26)_{10} - (11)_{10} = (11010)_2 - (1011)_2$.

$$\begin{array}{r} 11010 \\ - \\ \underline{1011\ \ } \\ 1111 \end{array}$$

Ստուգենք արդյունքը՝

$$(26)_{10} = (11010)_2,\ (11)_{10} = (1011)_2,\ (26 - 11)_{10} = (15)_{10}$$
$$(1111)_2 = 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = (15)_{10}$$

Ընդհանրապես, որքան փոքր է հաշվարկման համակարգի p հիմքը, այդքան ավելի երկար է թվի գրառումը: Այսպես, օրինակ, 16-ական, 8-ական և 2-ական համակարգերում 225.75 թվի գրառումներն են.

$$(225.75)_{10} = (E1)_{16} + (0.C)_{16} = (E1.C)_{16}$$
$$(225.75)_{10} = 3\cdot8^2 + 4\cdot8^1 + 1\cdot8^0 + 6\cdot8^{-1} = (341.6)_8$$
$(225.75)_{10} = (11100001.11)_2$:

Թվերի ներկայացումը համակարգչում: Համակարգչում թվերը ներկայացվում են երկուական հաշվարկման համակարգում, քանի որ թվի ներկայացումը զրոների և մեկերի հաջորդականության տեսքով տեխնիկապես ավելի հեշտ է իրականացնել, քան կամայական այլ համակարգում:

Ամբողջ թվերի ներկայացումը: Ամբողջ թվերը համակարգչում ներկայացվում են նշանով կամ առանց նշանի՝ զրոների և մեկերի հաջորդականության տեսքով:

Դիցուք, ամբողջ թիվը պահելու համար նախատեսված է N բիթ պարունակող կարգային ցանց, որի առաջին բիթում պահվում է թվի նշանը, մնացած (N-1) բիթերում՝ թվի գրառումը երկուական համակարգում։

1	2	3	.....		N-1	N
+/-						

Այս կարգային ցանցում պահվող առավելագույն և նվազագույն թվերը համապատասխանաբար հավասար են՝ $A_{max} = 2^{N-1} - 1$ և $A_{min} = -(2^{N-1} - 1)$։

Իրական թվերի ներկայացումը սահող կետով։ Սահող կետով կոչվում են հետևյալ տեսքով ներկայացված թվերը՝ $A = m \cdot q^p$, որտեղ m-ը կոչվում է մանտիս, p-ն՝ կարգ, իսկ q-ն հաշվարկման համակարգի հիմքն է։ Օրինակ.

$$(125)_{10} = 1.25 \cdot 10^2 = 1250 \cdot 10^{-1} = 0.125 \cdot 10^3:$$

Իրական թվերի սահող կետով ներկայացման դեպքում առանձնացվում է թվերի նորմալացված տեսքը, երբ մանտիսը բավարարում է հետևյալ պայմանին՝ $\frac{1}{q} \leq |m| \leq 1$։ Այսպես, եթե $q = 10$, ապա մանտիսը պետք է բավարարի $\frac{1}{10} \leq |m| \leq 1$ պայմանին։

Եթե $q = 2$, ապա մանտիսը պետք է բավարարի $\frac{1}{2} \leq |m| \leq 1$ պայմանին։

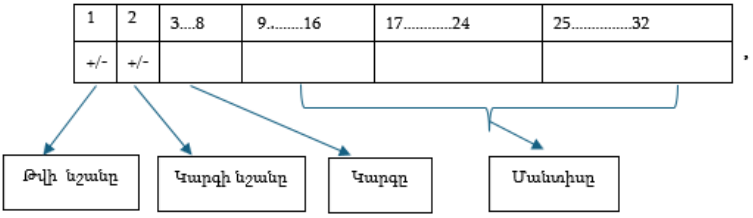
125 թվի՝ վերևում բերված տեսքը, սահող կետով ներկայացման նորմալացված տեսքն է, որտեղ մանտիսը՝ $m = 0.125$, կարգը՝ $p = 3$, իսկ հաշվման համակարգի հիմքը՝ $q = 10$։

Օրինակներ.

Երկուական համակարգում $A = -10110.11$ և $B = 0.00011$ թվերի սահող կետով ներկայացման նորմալացված տեսքերն են՝

$A = -0.1011011 \cdot 2^5 = -0.1011011 \cdot 2^{101}$,
$B = -0.11 \cdot 2^{-3} = -0.11 \cdot 2^{-11}$:

Ենթադրենք, թիվը պահվում է չորս բայթում, որից առաջին բայթում պահվում են թվի նշանը, կարգի նշանն ու կարգը, իսկ մնացած՝ երկրորդ, երրորդ և չորրորդ բայթերում՝ մանտիսի նորմալացված ձևը: Այդ դեպքում կարգային ցանցը կունենա հետևյալ տեսքը.



Բացասական թվերի ներկայացումը համակարգչում: Թվաբանական գործողությունները համակարգչում կատարվում են հատուկ սարքավորման՝ գումարիչի միջոցով: Թվաբանական հանման գործողությունը փոխարինվում է հանրահաշվական գումարման գործողությամբ՝ $A + B = A + (-B)$:

Բացասական թվերի մեքենայական ներկայացման համար օգտագործվում են ուղիղ, հակադարձ և լրացուցիչ կոդերը ($[A]_{\text{ուղ.}}$, $[A]_{\text{հակ.}}$ և $[A]_{\text{լր.}}$), իսկ գումարիչը թվաբանական գործողությունները կատարում է թվերի լրացուցիչ կոդերի հետ: Դրական թվերի հակադարձ և լրացուցիչ կոդերը համընկնում են ուղիղ կոդի հետ: Այսպես, եթե $A = +1101$, ապա ութ բիթ պարունակող կարգացանցում դրանք կներկայացվեն հետևյալ տեսքով (նշանի բիթում 0 է).

$$[A]_{\text{ուղ.}} = 00001101,\ [A]_{\text{հակ.}} = 00001101,\ [A]_{\text{լր.}} = 00001101:$$

Բացասական թվերի համար ուղիղ կոդում նշանի բիթը գրվում է մեկ: Այսպես, եթե $A = -a_1a_2 \dots a_n$, ապա դրա մեքենայական ներկայացումն ուղիղ կոդում կլինի՝ $[A]_{\text{ուղ.}} = 1a_1a_2 \dots a_n$:

Երկուական հաշվարկման համակարգում բացասական թվերի հակադարձ և լրացուցիչ կոդերի ստացման համար օգտագործվում է հետևյալ ալգորիթմը:

Հակադարձ կոդը ստանալու համար երկուական ներկայացման մեջ մեկերը փոխարինում են զրոներով, զրոները՝ մեկերով՝ անփոփոխ թողնելով նշանի բիթը։

Լրացուցիչ կոդը ստանալու համար հակադարձ կոդի ամենափոքր կարգին գումարվում է մեկ։

Նույն գործողությունները կատարելով հակառակ ուղղությամբ՝ կարելի է թվի լրացուցիչ կոդից ստանալ դրա ուղիղ կոդը։

Կառուցենք $B = -1101$թվի համար ուղիղ, հակադարձ և լրացուցիչ կոդերը։

$$[B]_{\text{ուղ.}} = 10001101,\ [B]_{\text{հակ.}} = 11110010, [B]_{\text{լր.}} = 11110011:$$

Հեշտ է ստուգել, որ թվերի լրացուցիչ կոդերի գումարը հավասար է դրանց գումարի լրացուցիչ կոդին։ Այսինքն, եթե $A + B = C$, ապա $[A]_{\text{լր.}} + [B]_{\text{լր.}} = [A + B]_{\text{լր.}}$։

Նշենք, որ լրացուցիչ կոդերի գումարման ժամանակ գումարվում են բոլոր բիթերը՝ ներառյալ նշանի բիթը, և եթե կարգացանցից դուրս է գալիս մեկը, ապա այն անտեսվում է։ Ստուգենք այս պնդումը օրինակի վրա։

$A = (101)_2 = (5)_{10}$ $B = (111)_2 = (7)_{10}$	$[A]_{\text{լր.}} = 00000101$ $[B]_{\text{լր.}} = 00000111$ $[A]_{\text{լր.}} + [B]_{\text{լր.}} = (00001100)_2 = (+12)_{10}$
$A = (-101)_2 = (-5)_{10}$ $B = (111)_2 = (-7)_{10}$	$[A]_{\text{լր.}} = 11111011$ $[B]_{\text{լր.}} = 11111001$ $[A]_{\text{լր.}} + [B]_{\text{լր.}} = (10001100)_2 = (-12)_{10}$
$A = (-101)_2 = (-5)_{10}$ $B = (111)_2 = (7)_{10}$	$[A]_{\text{լր.}} = 11111011$ $[B]_{\text{լր.}} = 00000111$ $[A]_{\text{լր.}} + [B]_{\text{լր.}} = (00000010)_2 = (+2)_{10}$
$A = (101)_2 = (5)_{10}$ $B = (111)_2 = (-7)_{10}$	$[A]_{\text{լր.}} = 00000101$ $[B]_{\text{լր.}} = 11111001$ $[A]_{\text{լր.}} + [B]_{\text{լր.}} = (10000010)_2 = (-2)_{10}$

3.3. Փոփոխականներ: Վերագրման օպերատոր

Հիշողության մեջ պահվող տվյալներին կարելի է դիմել փոփոխականների միջոցով: Փոփոխականներն այն մեծություններն են, որոնք ծրագրի կատարման ընթացքում կարող են ստանալ տարբեր արժեքներ:

Փոփոխականները Python-ում հղումներ են հիշողության մեջ գտնվող օբյեկտներին: Երբ որ փոփոխականին վերագրվում է որևէ արժեք, Python-ի ինտերպրետատորը ստեղծում է համապատասխան տիպի օբյեկտ՝ այդ արժեքով, որին էլ հղվում է տվյալ փոփոխականը: Երբ որ տվյալ փոփոխականին վերագրվում է մեկ այլ արժեք, ինտերպրետատորը հիշողության մեջ ստեղծում է նոր օբյեկտ, և փոփոխականը հղվում է այդ օբյեկտին:

Փոփոխականի հայտարարումը և դրան արժեք վերագրելը կատարվում է վերագրման օպերատորի միջոցով: Նախ գրվում է փոփոխականի անունը, այնուհետև դրվում է '=' նշանը, ապա՝ այն արժեքը, որը վերագրվելու է այդ փոփոխականին:

Python-ի ցանկացած օբյեկտ ունի իր եզակի նույնականացուցիչը՝ (identifier)՝ id-ն, որը կցվում է օբյեկտին, երբ այն ստեղծվում է և մնում է օբյեկտի ողջ կյանքի ընթացքում: id-ն հիշողության այն հասցեն է, որտեղ տեղակայված է օբյեկտը: Յուրաքանչյուր անգամ, երբ ծրագիրը գործարկվում է, ստեղծվում է նոր նույնականացուցիչ: id() ֆունկցիան Python-ում վերադարձնում է իրեն որպես արգումենտ փոխանցված օբյեկտի նույնականացուցիչը:

Օրինակ՝ x = 5 հրամանով հայտարարվում է x փոփոխականը և այն ստանում է 5 արժեքը:

Փոփոխականի արժեքավորման ժամանակ կատարվում է հետևյալը.

- համակարգչի հիշողության մեջ ստեղծվում է ամբողջ տիպի (int) օբյեկտ 5 արժեքով, և այդ օբյեկտին կցվում է id,
- օգտագործելով '=' վերագրման օպերատորը՝ x փոփոխականը հղվում է 5 օբյեկտին:

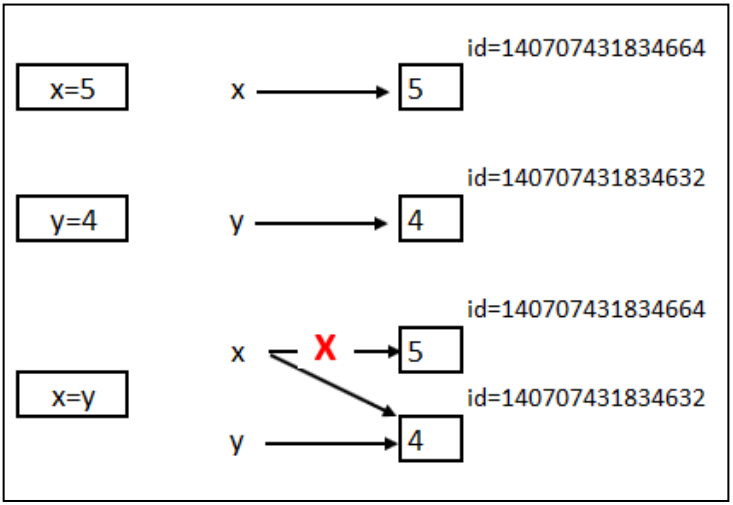
Քննարկենք հետևյալ օրինակը.

```
>>>x=5
>>>y=4
```

```
>>>id(x)
140707431834664
>>>id(y)
140707431834632
>>>x=y
>>>id(x)
140707431834632
```

Սկզբում համակարգչի հիշողության մեջ ստեղծվում է ամբողջ տիպի օբյեկտ 5 արժեքով, և x փոփոխականը հղվում է այդ օբյեկտին, որի id(x)=140707431834664: Այնուհետև, ստեղծվում է ամբողջ տիպի նոր օբյեկտ 4 արժեքով, և y փոփոխականը հղվում է այդ օբյեկտին, որի id(y)=140707431834632:

x=y հրամանով x փոփոխականին վերագրվում է y փոփոխականի արժեքը՝ 4: Քանի որ 4 օբյեկտը կա հիշողության մեջ, ուստի այլևս նոր օբյեկտ չի ստեղծվում, և x փոփոխականը հղվում է նույն 4 օբյեկտին, այդ պատճառով էլ id(x)=140707431834632:



Թվային տիպի օբյեկտին հղումը կարելի է ջնջել՝ օգտագործելով del օպերատորը: Օրինակ.

del var1
del var1, var2
del var1, var2, var3 և այլն:

Հնարավոր է ջնջել մեկ կամ մի քանի օբյեկտների հղումներ, ընդ որում, օբյեկտը իրականում չի ջնջվում, այլ ջնջվում է օբյեկտի հղումը: Օբյեկտը ջնջելու պատասխանատուն Python-ի աղբահավաքն է (garbage collector), որն օգտագործվում է ավտոմատ կերպով վերականգնելու այն հիշողությունը, որն այլևս հասանելի չէ կամ չի օգտագործվում ծրագրի կողմից:

Այսպիսով, փոփոխականները Python-ում հղումներ են հիշողության մեջ գտնվող օբյեկտներին: Ի տարբերություն այլ ծրագրավորման լեզուների՝ Python-ում անհրաժեշտություն չկա նախապես հայտարարել փոփոխականները, որպեսզի ինտերպրետատորը հիշողության մեջ տեղ հատկացնի դրանց: Հայտարարումը տեղի է ունենում այն ժամանակ, երբ արժեք է վերագրվում փոփոխականին: Երբ տվյալ փոփոխականին վերագրվում է մեկ այլ արժեք, ինտերպրետատորը հիշողության մեջ ստեղծում է նոր օբյեկտ, և փոփոխականը հղվում է այդ օբյեկտին:

3.4. Թվային տիպերը

Թվային տիպերը նախատեսված են թվային արժեքների համար: Python-ը աջակցում է տարբեր թվային տիպեր.

- int (ամբողջ թվեր);
- float (սահող կետով թվեր);
- complex (կոմպլեքս թվեր):

Այս տիպերը սահմանվում են համապատասխանաբար որպես Python int, Python float և Python complex դասեր Python-ում: Python 3-ում չկա սահմանափակում, թե որքան երկար կարող է լինել ամբողջ թիվը:[5]

[5] Python 2-ում կար նաև long տիպը՝ երկար ամբողջ թվեր: Python 3-ում բոլոր ամբողջ թվերը ներկայացված են որպես երկար թվեր, և long տիպը չի օգտագործվում:

int, float, complex տիպերը չփոփոխվող տիպեր են։

int տիպը։ int տիպի տվյալները ներկայացնում են ամբողջ թվեր՝ + կամ – նշանով կամ առանց նշանի կամ զրո: Եթե թվի դիմաց նշանը բացակայում է, ապա թիվը համարվում է դրական: Թվերը կարելի է գրել 10-ական, 8-ական և 16-ական համակարգերում:

Օրինակներ՝ 0, 13, -28, +384 թվերն ամբողջ են:

Հետևյալ օրինակներում գրված է 10 թիվը հաշվարկման տարբեր համակարգերում՝

10, 012, 0xA:

Դիտարկենք ծրագրի հետևյալ հատվածը.

```
num1 = 15
num2 = -20
num3 = 0
print('num1=',num1)
print('num2=',num2)
print('num3=',num3)
print('type of num1:',type(num1))
print('type of num2:',type(num2))
print('type of num3:',type(num3))
```

Ծրագրի կատարման արդյունքն է.

```
num1= 15
num2= -20
num3= 0
type of num1: <class 'int'>
type of num2: <class 'int'>
type of num3: <class 'int'>
```

Ծրագրում օգտագործված type(x) ներկառուցված ֆունկցիան վերադարձնում է x փոփոխականի տիպը։

float տիպը։ float տիպը նախատեսված է իրական թվերի համար:

Օրինակներ՝ 5.26, -7.385, .4, -.45, 0 թվերը float տիպի են:

Սահող կետով գրված իրական թվերը (էքսպոնենտային տեսքի իրական թվերը) ևս float տիպի են: Օրինակ՝ +5.6437e8, -3.1415e12, .6e-1 թվերը սահող կետով իրական float տիպի թվեր են: Դիտարկենք հետևյալ ծրագիրը.

```
num1 = 5.26
num2 = -7.385
num3 = -.45
num4=+5.6437e8
num5=-3.1415e12
num6=.6e-1

print('num1=',num1)
print('num2=',num2)
print('num3=',num3)
print('num4=',num4)
print('num5=',num5)
print('num6=',num6)

print('type of num1:',type(num1))
print('type of num2:',type(num2))
print('type of num3:',type(num3))
print('type of num4:',type(num4))
print('type of num5:',type(num5))
print('type of num6:',type(num6))
```

Ծրագրի կատարման արդյունքն է.

```
num1= 5.26
num2= -7.385
num3= -0.45
num4= 564370000.0
num5= -3141500000000.0
num6= 0.06
type of num1: <class 'float'>
type of num2: <class 'float'>
type of num3: <class 'float'>
```

```
type of num4: <class 'float'>
type of num5: <class 'float'>
type of num6: <class 'float'>
```

int և float տիպի տվյալների հետ կարող են կատարվել հետևյալ գործողությունները՝ գումարում, հանում, բազմապատկում, բաժանում, ամբողջ բաժանում (քանորդից փոքր կամ հավասար ամենամեծ ամբողջ թիվը), մնացորդով բաժանում, աստիճան բարձրացնելու գործողություն:

complex տիպը: complex տիպը նախատեսված է կոմպլեքս թվերի համար: Կոմպլեքս թիվն ունի a + bj տեսքը, որտեղ a-ն և b-ն իրական թվեր են, իսկ j-ն՝ կեղծ միավորը՝ $j^2 = -1$: Օրինակ՝ 3 + 3j, -2.1+4j, j թվերը կոմպլեքս թվեր են:

complex տիպի տվյալների հետ կարող են կատարվել գումարման, հանման, բազմապատկման և բաժանման գործողությունները: Օրինակ՝

```
# Python complex
num1 = 6+3j # ստեղծում է 6 + 3j կոմպլեքս թիվը
num2 = -5 - 4j # ստեղծում է -5 - 4j կոմպլեքս թիվը
num3 = complex(8, -2)
sum_num = num1+num3
sub_num = num1-num3
prod_num = num1*num3
div_num = num1/num3

print('type of num1:',type(num1))
print('type of num2:',type(num2))
print('type of num3:',type(num3))
print('The sum of complex numbers=', sum_num)
print('The difference of complex numbers=', sub_num)
print('The product of complex numbers=', prod_num)
print('The quotient of the division of two complex numbers=', div_num)
```

Ծրագրի կատարման արդյունքն է.

type of num1: <class 'complex'>
type of num2: <class 'complex'>
type of num3: <class 'complex'>
The sum of complex numbers= (14+1j)
The difference of complex numbers= (-2+5j)
The product of complex numbers= (54+12j)
The quotient of the division of two complex numbers= (0.6176470588235294+0.5294117647058824j)

3.5. Տրամաբանական տիպ (bool)

Տրամաբանական տիպի (bool) տվյալն ընդունում է երկու արժեք՝ True (ճիշտ) կամ False (սխալ)։ Բուլյան արժեքները վերադարձվում են համեմատությունների և տրամաբանական գործողությունների արդյունքում։ Օրինակ՝

```
a = 7
b = 11
# Համեմատության օպերատորներ
c=(a == b)
print('a==b', c)
print('type of c:',type(c))
print('a<b', a < b)
```

Արդյունքը՝

```
a==b False
type of c: <class 'bool'>
a<b True
```

3.6. Տիպերի ձևափոխության ֆունկցիաներ

Տվյալները կարելի է մի տիպից ձևափոխել մեկ այլ տիպի։

float տիպը ձևափոխվում է int տիպի՝ int() ֆունկցիայի միջոցով։ Օրինակ՝

```
num1_fl=32.65
print('num1_fl=', num1_fl)
print('type of num1_fl:', type(num1_fl))
num1_int=int(num1_fl) # float տիպի փոփոխականը ձևափոխվում է int տիպի
print('num1_int=', num1_int)
print('type of num1_int:', type(num1_int))
num2_fl=-3.3
print('num2_fl=', num2_fl)
print('type of num2_fl:', type(num2_fl))
num2_int=int(num2_fl) # float տիպի փոփոխականը ձևափոխվում է int տիպի
print('num2_int=', num2_int)
print('type of num2_int:', type(num2_int))
```

Պատասխան՝

```
num1_fl= 32.65
type of num1_fl: <class 'float'>
num1_int= 32
type of num1_int: <class 'int'>
num2_fl= -3.3
type of num2_fl: <class 'float'>
num2_int= -3
type of num2_int: <class 'int'>
```

int տիպը ձևափոխվում է float տիպի՝ float() ֆունկցիայի միջոցով: Օրինակ՝

```
num_int=-32
print('num_int=', num_int)
print('type of num_int:', type(num_int))
num_f=float(num_int) # int տիպի փոփոխականը ձևափոխվում է float տիպի
print('num_f=',num_f)
print('type of num_f:', type(num_f))
```

Արդյունքը՝

```
num_int= -32
type of num_int: <class 'int'>
num_f= -32.0
type of num_f: <class 'float'>
```

bool() ֆունկցիայի միջոցով կարելի է int և float տիպի փոփոխականները ձևափոխել bool տիպի: Դիտարկենք հետևյալ ծրագիրը՝

```
num_int=-7
num_int_bool=bool(num_int) # int տիպի փոփոխականը ձևափոխվում է bool տիպի
print('num_int_bool=',num_int_bool)
print('type of num_int_bool:', type(num_int_bool))
num_f=0.0
num_f_bool=bool(num_f) # float տիպի փոփոխականը ձևափոխվում է bool տիպի
print('num_f_bool=', num_f_bool)
print('type of num_f_bool:', type(num_f_bool))
```

Պատասխան՝

```
num_int_bool= True
type of num_int_bool: <class 'bool'>
num_f_bool= False
type of num_f_bool: <class 'bool'>
```

ԳԼՈՒԽ 4. PYTHON ՕՊԵՐԱՏՈՐՆԵՐ: ԱՐՏԱՀԱՅՏՈՒԹՅՈՒՆՆԵՐ

Օպերատորներն օգտագործվում են փոփոխականների հետ գործողություններ կատարելու համար: Օպերատորներն ըստ օպերանդների (գործողության մասնակիցների) քանակի լինում են ունար և բինար: Եթե գործողությունը կատարվում է երկու օպերանդի հետ, ապա օպերատորը կոչվում է բինար, իսկ մեկ օպերանդի դեպքում՝ ունար: Python օպերատորներն են.

- Թվաբանական օպերատորներ (Arithmetic operators),
- Համեմատության օպերատորներ (Comparison operators),
- Վերագրման օպերատորներ (Assignment operators),
- Տրամաբանական օպերատորներ (Logical operators),
- Բիթային օպերատորներ (Bitwise operators),
- Անդամակցության օպերատորներ (Membership operators),
- Նույնականացման օպերատորներ (Identity operators):

4.1. Թվաբանական օպերատորներ: Թվաբանական արտահայտություններ

Թվաբանական օպերատորներն օգտագործվում են թվային տվյալների հետ թվաբանական գործողություններ կատարելու համար: Դրանք են՝ «+», «-», «*», «**», «/», «//», «%»: Python-ի թվաբանական օպերատորները նկարագրված են ստորև ներկայացված աղյուսակում (աղյուսակ 4.1):

Օպերատոր	Գործողությունը	Նկարագրություն
+	Գումարում	Գումարում է երկու օպերանդների արժեքները:
-	Հանում	Առաջին օպերանդի արժեքից հանում է երկրորդի արժեքը:
*	Բազմապատկում	Բազմապատկում է երկու օպերանդների արժեքները:

/	Բաժանում	Առաջին օպերանդի արժեքը բաժանում է երկրորդի արժեքի վրա:
**	Աստիճան բարձրացնելու գործողություն	Առաջին օպերանդի արժեքը բարձրացնում է երկրորդի արժեքով աստիճան:
//	Ամբողջ բաժանում	Առաջին օպերանդի արժեքը բաժանում է երկրորդի արժեքի վրա և վերադարձնում է քանորդից փոքր կամ հավասար ամենամեծ ամբողջ թիվը:
%	Մոդուլով բաժանում	Առաջին օպերանդի արժեքը բաժանում է երկրորդի արժեքի վրա և վերադարձնում է ստացված մնացորդը

Աղյուսակ 4.1. Python թվաբանական օպերատորները

Python լեզվում կա բաժանման երեք գործողություն՝ «/», «//», «%»:

«/» բաժանումը թվաբանական գործողություն է, որը վերադարձնում է երկու թվերի բաժանումից ստացված քանորդը:

«//»-ը կոչվում է ամբողջ բաժանում, որը վերադարձնում է երկու թվերի բաժանումից ստացված քանորդից փոքր կամ հավասար ամենամեծ ամբողջ թիվը:

«%»-ը կոչվում է մոդուլով բաժանում, որի արդյունքը երկու թվերի բաժանումից ստացված մնացորդն է: Python-ը մոդուլով բաժանումը կատարում է հետևյալ բանաձևով.

$$a \% b = a - (a // b) * b:$$

Դիտարկենք հետևյալ ծրագիրը.

```
num1=15
num2=4
addition = num1+num2
print("Addition:", addition)
```

```
subtraction = num1-num2
print("Subtraction:", subtraction)

multiplication = num1*num2
print("Multiplication:", multiplication)

division = num1/num2
print("Division:", division)

integer_division = num1//num2
print("Integer Division:", integer_division)

modulus = num1%num2
print("Modulus:", modulus)

exponentiation = num1** num2
print("Exponentiation:", exponentiation)
```

Ծրագրի կատարման արդյունքն է.

```
Addition: 19
Subtraction: 11
Multiplication: 60
Division: 3.75
Integer Division: 3
Modulus: 3
Exponentiation: 50625
```

Այն դեպքում, երբ բաժանարարը հավասար է զրոյի, ինտերպրետատորը կտա հաղորդագրություն սխալի մասին՝ “ZeroDivisionError” և կդադարեցնի կոդի կատարումը։

Մի փոքր մանրամասնորեն քննարկենք բաժանման գործողություններն այն դեպքում, երբ բաժանելին կամ բաժանարարը կամ՝ երկուսը միասին, ամբողջ բացասական թվեր են։ Նախ դիտարկենք այն դեպքը, երբ բաժանելին բացասական թիվ է։

```
# Բաժանելին բացասական է
num1=-15
```

```
num2=4
print('num1/num2=', num1/num2)
print('num1//num2=', num1//num2)
print('num1%num2=', num1%num2)
```

Ծրագրի կատարման արդյունքն է.

```
num1/num2= -3.75
num1//num2= -4
num1%num2= 1
```

Այստեղ num1/num2=-3.75, որի ամբողջ մասը -4 թիվն է, ուստի num1//num2=-4: Բաժանումից ստացված մնացորդը կհաշվարկվի հետևյալ կանոնով՝ -15-(-4*4)=1, այնպես որ num1%num2=1:

Այժմ դիտարկենք այն դեպքը, երբ բաժանարարը բացասական թիվ է:

```
# Բաժանարարը բացասական թիվ է
num1=15
num2=-4
print('num1/num2=', num1/num2)
print('num1//num2=', num1//num2)
print('num1%num2=', num1%num2)
```

Ծրագրի կատարման արդյունքն է.

```
num1/num2= -3.75
num1//num2= -4
num1%num2= -1
```

Այսպիսով, երբ num1=15 և num2=-4, կրկին num1/num2=-3.75 և num1//num2= -4: Իսկ մնացորդը հաշվարկվել է հետևյալ ձևով՝ 15-(-4*(-4))=-1, այնպես որ num1%num2=-1:

Այժմ, դիցուք, և՛ բաժանելին, և՛ բաժանարարը բացասական ամբողջ թվեր են:

```
# Բաժանելին և բաժանարարը բացասական են
num1=-15
```

```
num2=-4
print('num1/num2=', num1/num2)
print('num1//num2=', num1//num2)
print('num1%num2=', num1%num2)
```

Ծրագրի կատարման արդյունքն է.

```
num1/num2= 3.75
num1//num2= 3
num1%num2= -3
```

Այսպիսով, երբ num1=-15 և num2=-4, քանորդը՝ num1/num2=3.75: Քանի որ քանորդի ամբողջ մասը 3 թիվն է, ուստի num1//num2=3: Իսկ մնացորդը հաշվարկվել է հետևյալ ձևով՝ -15-(-4*3)=-3, այնպես որ num1%num2=-3:

Python լեզվում // և % գործողությունները կարելի է կատարել նաև սահող կետով թվերի հետ: Այն դեպքում, երբ բաժանելին կամ բաժանարարը կամ երկուսը միասին սահող կետով թվեր են, արդյունքը վերադարձվում է սահող կետով թվով: Օրինակներ՝

```
print(10.5 // 3.0) # 3.0
print(10.5 % 3.0) # 1.5
print(10.0 // 2.5) # 4.0
print(10.0 % 2.5) # 0.0
print(5.0 // -2.0) # -3.0
print(5.0 % -2.0) # -1.0
print(-10.5 // 3.0) # -4.0
print(-10.5 % 3.0) # 1.5
print(-10.5 // -3.0) # 3.0
print(-10.5 % -3.0) # -1.5
```

Միմյանց հետ թվաբանական օպերատորներով ու փակագծերով միացված հաստատունների, փոփոխականների և ֆունկցիաների համախումբը կոչվում է թվաբանական արտահայտություն: Թվաբանական արտահայտության մեջ մտնող փոփոխականների ցանկացած թույլատրելի արժեքների համար որոշվում է որոշակի թվային արժեք: Ներկայացնենք թվաբանական արտահայտությունների օրինակներ.

256
x * y / z + x / (y * z)
(a * * 3 + b * *3) / (a * c)
(-b+sqrt(b**2-4*a*c))/(2*a)
x/(1+pow(x,2)/((2*x)**3))
cos(2*a-b)+sin(2*a-b)

Բերված արտահայտություններում օգտագործվել են pow(), sqrt(), sin() և cos() մաթեմատիկական ֆունկցիաները, որոնք գտնվում են Python-ի math մոդուլում: Մաթեմատիկական ֆունկցիաներին դիմելու համար անհրաժեշտ է import հրամանով math մոդուլը նախապես ներբեռնել հիշողության մեջ (մոդուլների մասին՝ տե՛ս *գլուխ 13*):

Թվարկենք math մոդուլում գտնվող որոշ մաթեմատիկական ֆունկցիաներ.

- pow(num, power) - num թվի power աստիճանը,
- sqrt(num) - num թվի քառակուսի արմատը,
- ceil(num) - թիվը կլորացնում է մինչև ամենամոտ մեծ ամբողջ թիվը,
- floor(num) - թիվը կլորացնում է մինչև ամենամոտ փոքր ամբողջ թիվը,
- factorial(num) - num թվի ֆակտորիալը,
- degrees(x) - փոխակերպում է x անկյունը ռադիաններից աստիճանների,
- radians(x) - փոխակերպում է x անկյունը աստիճաններից ռադիանների,
- cos(rad) - ռադիաններով տրված rad անկյան կոսինուսը,
- sin(rad) - ռադիաններով տրված rad անկյան սինուսը,
- tan(rad) - ռադիաններով տրված rad անկյան տանգենսը,
- acos(x) - x թվի արկկոսինուսը (ռադիաններով),
- asin(x) - x թվի սինուսը (ռադիաններով),
- atan(x) - x թվի արկտանգենսը (ռադիաններով),
- log(n) - n թվի լոգարիթմը բնական հիմքով,
- log(n, base) - n թվի լոգարիթմը base հիմքով,
- log10(n) - n թվի լոգարիթմը տասը հիմքով,

- exp(x) - e թվի x աստիճանը,
- pi: $\pi = 3.141592$ ...,
- e: $e = 2.718281\ldots$,
- inf: $+\infty$,
- nan: թիվ չէ (“Not a number” (NaN))

Նշենք, որ abs() ֆունկցիան, որը վերադարձնում է x թվի բացարձակ արժեքը, ներկառուցված է և հասանելի է առանց math մոդուլը ներբեռնելու: Ներկառուցված ֆունկցիաներից է նաև pow() ֆունկցիան:[6]

Հետևյալ օրինակում կիրառված են math մոդուլի մի քանի մաթեմատիկական ֆունկցիաներ: Օգտագործման համար ֆունկցիայի անունից առաջ գրվում է մոդուլի անունը, որից հետո դրվում է կետ՝ math.:

```
import math
sq = math.sqrt(121)
print('Square root of 121:', sq)
fact = math.factorial(6)
print('Factorial of 6:',fact)

ceil = math.ceil(-2.7)
print('Ceiling of -2.7:',ceil)

floor = math.floor(-2.7)
print('Floor of -2.7:',floor)
e = math.e
print('Value of e:',e)

ln = math.log(math.e**2)
print('Natural logarithm of e**2:', ln)

sin= math.sin(math.radians(30))
print('Sine of 30 degrees:',sin)
```

[6] Python-ի ներկառուցված pow() և math.pow() ֆունկցիաները տարբերվում են օգտագործման դեպքերով:

Պատասխան՝

```
Square root of 121: 11.0
Factorial of 6: 720
Ceiling of -2.7: -2
Floor of -2.7: -3
Value of e: 2.718281828459045
Natural logarithm of e**2: 2.0
Sine of 30 degrees: 0.49999999999999994
```

Թվաբանական արտահայտության մեջ մտնող գործողությունները կատարվում են ըստ իրենց առաջնահերթության կարգի, ընդ որում, հավասարազոր գործողությունները կատարվում են ձախից աջ։ Փակագծերն ապահովում են գործողության կատարման առաջնահերթությունը։

Առաջնահերթությունը	Գործողությունը
1	Ֆունկցիաներ ու փակագծեր
2	**
3	- (նշանափոխություն)
4	* և /
5	//
6	%
7	+ և -

Աղյուսակ 4.2. Թվաբանական գործողությունների կատարման կարգը

4.2. Համեմատության օպերատորներ

Համեմատության օպերատորներն (աղյուսակ 4.3) օգտագործվում են երկու արտահայտությունների արժեքներն իրար հետ համեմատելու համար։ Համեմատությունն իր մեջ մտնող փոփոխականների թույլատրելի արժեքների համար ընդունում է True կամ False արժեքները։

Օպերատոր	Գործողությունը	Նկարագրություն
==	հավասար է	Վերադարձնում է True, եթե ձախ կողմում գրված օպերանդի արժեքը հավասար է աջ կողմում գրված օպերանդի արժեքին, False՝ հակառակ դեպքում:
!=	հավասար չէ	Վերադարձնում է True, եթե ձախ կողմում գրված օպերանդի արժեքը հավասար չէ աջ կողմում գրված օպերանդի արժեքին, False՝ հակառակ դեպքում:
>	մեծ է	Վերադարձնում է True, եթե ձախ կողմում գրված օպերանդի արժեքը մեծ է աջ կողմում գրված օպերանդի արժեքից, False՝ հակառակ դեպքում:
<	փոքր է	Վերադարձնում է True, եթե ձախ կողմում գրված օպերանդի արժեքը փոքր է աջ կողմում գրված օպերանդի արժեքից, False՝ հակառակ դեպքում:
>=	մեծ է կամ հավասար	Վերադարձնում է True, եթե ձախ կողմում գրված օպերանդի արժեքը մեծ է կամ հավասար աջ կողմում գրված օպերանդի արժեքից, False՝ հակառակ դեպքում:
<=	փոքր է կամ հավասար	Վերադարձնում է True, եթե ձախ կողմում գրված օպերանդի արժեքը փոքր է կամ հավասար աջ կողմում գրված օպերանդի արժեքից, False՝ հակառակ դեպքում:

Աղյուսակ 4.3. Համեմատության օպերատորները

Օրինակ՝

```
a = 12
b = 19
print('a == b', a == b)
print('a != b', a != b)
print('a < b', a < b)
print('a <= b', a <= b)
print('a > b', a > b)
print('a >= b', a >= b)
```

Ծրագրի կատարման արդյունքն է.

```
a == b False
a != b True
a < b True
a <= b True
a > b False
a >= b False
```

4.3. Տրամաբանական օպերատորներ: Տրամաբանական արտահայտություններ

Python լեզվի տրամաբանական օպերատորներն են՝ and, or, not (աղյուսակ 4.4): Տրամաբանական օպերատորները, բացի not օպերատորից, բինար են:

Օպերատոր	Գործողությունը	Նկարագրություն
and	տրամաբանական արտադրյալ (և)	Վերադարձնում է True այն և միայն այն դեպքում, երբ երկու օպերանդների արժեքները True են:
or	տրամաբանական գումար (կամ)	Վերադարձնում է False այն և միայն այն դեպքում, երբ երկու օպերանդների արժեքները False են:

not	տրամաբանական ժխտում	Ժխտում է օպերանդի արժեքը՝ True-ն դարձնում է False, իսկ False-ը՝ True:

Աղյուսակ 4.4. Տրամաբանական օպերատորները

Միմյանց հետ տրամաբանական օպերատորներով կապված տրամաբանական հաստատունների, փոփոխականների, ֆունկցիաների ու համեմատությունների համախումբը կոչվում է տրամաբանական արտահայտություն: Տրամաբանական արտահայտությունն ընդունում է True կամ False արժեքը:

Բերենք տրամաբանական արտահայտությունների մի քանի օրինակներ.

```
(x>=a) and (x<=b)
(a or b and c) and not ((p or b) or (q and b))
not(x and not x or x or not x)
not (((-p) == (-p)) or f) and d
```

Տրամաբանական արտահայտությունների մեջ գործողությունների կատարման հերթականությունն է՝

Առաջնահերթությունը	Գործողությունը
1	փակագծեր
2	>, <, >=, <=, !=, ==
3	not
4	and
5	or

Աղյուսակ 4.5. Տրամաբանական գործողությունների կատարման կարգը

Օրինակ՝

```
a=2; b=5; x=3
print('a < 10 and b > 5', a < 10 and b > 5)
print('a < 3 or b> 20', a < 3 or b> 20)
print('not(a == b)', not(a == b))
print('(x>=a) and (x<=b)', (x>=a) and (x<=b))
```

```
print('(x<a) or (x>b)', (x<a) or (x>b))
print('not(a<3)', not(a<3))
```

Ծրագրի կատարման արդյունքն է.

```
a < 10 and b > 5 False
a < 3 or b> 20 True
not(a == b) True
(x>=a) and (x<=b) True
(x<a) or (x>b) False
not(a<3) False
```

Python-ում կարելի է օգտագործել նաև շղթայական համեմատության օպերատորները: Օրինակ, (x>=a) and (x<=b) գրության փոխարեն կարելի է օգտագործել հետևյալ համառոտ գրությունը՝ a<=x<=b:

Օրինակ՝

```
x = 5
print('1 < x < 10', 1 < x < 10)
print('10 < x < 20', 10 < x < 20 )
print('x < 10 < x*10 < 100', x < 10 < x*10 < 100)
print('10 > x <= 9', 10 > x <= 9)
print('5 == x > 4', 5 == x > 4)
```

Ծրագրի կատարման արդյունքն է.

```
1 < x < 10 True
10 < x < 20 False
x < 10 < x*10 < 100 True
0 > x <= 9 True
5 == x > 4 True
```

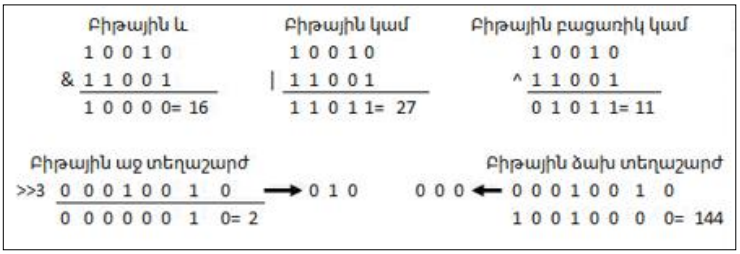
4.4. Բիթային օպերատորներ

Python լեզվի բիթային օպերատորներն (աղյուսակ 4.5) աշխատում են երկուական կոդերի հետ: Արդյունքը վերադարձվում է տասական համակարգում:

Օպերատոր	Գործողությունը	Նկարագրություն
&	բիթային և	Վերադարձնում է 1 այն և միայն այն դեպքում, երբ երկու բիթերն էլ մեկ են, 0՝ հակառակ դեպքում:
|	բիթային կամ	Վերադարձնում է 1, երբ երկու բիթերից գոնե մեկը հավասար է մեկ, 0՝ մնացած դեպքերում:
^	բիթային բացառող կամ	Վերադարձնում է 0, երբ երկու բիթերն էլ միաժամանակ մեկ են կամ զրո, 1՝ երբ իրարից տարբեր են:
~	բիթային ժխտում	Ունար օպերատոր է, մեկը դարձնում է զրո, իսկ զրոն՝ մեկ:
>>	աջ տեղաշարժ	Ձախ օպերանդի բիթերը տեղաշարժվում են դեպի աջ՝ աջ օպերանդի արժեքի չափով:
<<	ձախ տեղաշարժ	Ձախ օպերանդի բիթերը տեղաշարժվում են դեպի ձախ՝ աջ օպերանդի արժեքի չափով: Աջից ազատված բիթերը լրացվում են զրոներով:

Աղյուսակ 4.5. Python լեզվի բիթային օպերատորները

Օրինակ: Դիցուք, x=18, y=25: Այդ թվերը երկուական համակարգում կունենան հետևյալ տեսքը՝ $x = 18 = (10010)_2$, $y = 25 = (11001)_2$: Կատարենք հետևյալ բիթային գործողությունները.



Արդյունքները ստուգենք ստորև ներկայացված ծրագրով.

```
x=18; y=25; z=3, q=-19
print('x&y=', x&y)
print('x|y=', x|y)
print('x^y=', x^y)
print('~x=', ~x)
print('~q=', ~q)
print('x<<z=', x<<z)
print('x>>z=', x>>z)
```

Ծրագրի կատարման արդյունքն է.

```
x&y= 16
x|y= 27
x^y= 11
~x= -19
~q= 18
x<<z= 144
x>>z= 2
```

Python-ում բիթային ժխտման '~' օպերատորն աշխատում է՝ դրական թվերի համար կիրառելով ~x=-(x+1) բանաձևը, իսկ բացասական թվերի համար՝ ~x = -x - 1 բանաձևը: Ուստի՝ ~18= -(18+1)=-19, իսկ ~-19= -(-19)-1=18:

4.5. Վերագրման օպերատորներ

Վերագրման օպերատորներն օգտագործվում են փոփոխականներին արժեքներ վերագրելու համար: Պարզագույն վերագրման օպերատորն ունի հետևյալ շարահյուսությունը.

varname=expression,

որտեղ varname-ը փոփոխականի անունն է, իսկ expression-ը՝ արտահայտությունը:

Վերագրման օպերատորի կատարման ժամանակ նախ հաշվվում է աջ մասում գրված expression արտահայտության արժեքը՝ դրա մեջ մտնող փոփոխականների ընթացիկ արժեքների համար, ապա արդյունքը վերագրվում է varname փոփոխականին:

Օրինակներ՝

```
x=5
x=x+1  #x=6
y=x**2  #y=36
x=y  #x=36
```

Բացի պարզ վերագրման օպերատորից՝ կան նաև բարդ վերագրման օպերատորներ, որոնք հնարավորություն են տալիս աջ ու ձախ օպերանդների հետ գործողություն կատարել և արդյունքը վերագրել ձախ օպերանդին: Այդ օպերատորներն են՝ +=, -=, *=, /=, //=, %=, **=, &=, |=, ^=, >>=, <<=:

Օրինակ՝ x+=y (գումարում վերագրումով) օպերատորը համարժեք է x=x+y օպերատորին: Այդ օպերատորը ձախ օպերանդի արժեքին գումարում է աջ օպերանդի արժեքը և ստացածը վերագրում ձախ օպերանդին:

Ստորև ներկայացված աղյուսակ 4.6-ում նկարագրված են Python-ի վերագրման օպերատորները:

Օպերատոր	Նկարագրություն	Գործողություն
=	Ձախ օպերանդին վերագրում է աջ օպերանդի արժեքը:	x=y
+=	Ձախ օպերանդի արժեքին գումարում է աջ օպերանդի արժեքը և ստացածը վերագրում ձախ օպերանդին:	x+=y համարժեք է՝ x=x+y
-=	Ձախ օպերանդի արժեքից հանում է աջ օպերանդի արժեքը և ստացածը վերագրում ձախ օպերանդին:	x-=y համարժեք է՝ x=x-y
*=	Ձախ օպերանդի արժեքը բազմապատկում է աջ օպերանդի արժեքով և ստացածը վերագրում ձախ օպերանդին:	x*=y համարժեք է՝ x=x*y
**=	Ձախ օպերանդի արժեքը բարձրացնում է աստիճան՝ աջ օպերանդի արժեքով և ստացածը վերագրում ձախ օպերանդին:	x**=y համարժեք է՝ x=x**y
/=	Ձախ օպերանդի արժեքը բաժանում է աջ օպերանդի արժեքի վրա և քանորդը վերագրում ձախ օպերանդին:	x/=y համարժեք է՝ x=x/y
//=	Ձախ օպերանդի արժեքը բաժանում է աջ օպերանդի արժեքի վրա և քանորդի ամբողջ մասը վերագրում ձախ օպերանդին:	x//=y համարժեք է՝ x=x//y
%=	Ձախ օպերանդի արժեքը բաժանում է աջ օպերանդի արժեքի վրա և ստացված մնացորդը վերագրում ձախ օպերանդին:	x%=y համարժեք է՝ x=x%y
&=	Կատարում է «բիթային և» գործողությունը և արդյունքը վերագրում ձախ օպերանդին:	x&=y համարժեք է՝ x=x&y
|=	Կատարում է «բիթային կամ» գործողությունը և արդյունքը վերագրում ձախ օպերանդին:	x|=y համարժեք է՝ x=x|y

^=	Կատարում է բիթային «բացառիկ կամ» գործողությունը և արդյունքը վերագրում ձախ օպերանդին:	x^=y համարժեք է՝ x=x^y
>>=	Կատարում է ձախ օպերանդի «բիթային աջ տեղաշարժ» գործողությունը աջ օպերանդի արժեքով և արդյունքը վերագրում ձախ օպերանդին:	x>>=y համարժեք է՝ x=x>>y
<<=	Կատարում է ձախ օպերանդի «բիթային ձախ տեղաշարժ» գործողությունը աջ օպերանդի արժեքով և արդյունքը վերագրում ձախ օպերանդին:	x<<=y համարժեք է՝ x=x<<y

Աղյուսակ 4.6. Վերագրման օպերատորները

Օրինակներ.

```
x = 24; y=7
a=15; b=3
x%=y            # x=x%y
print('x=', x)  # x= 3
a >>= b         # a=a>>b
print('a=', a)   # a=1
y&=b            # y=y&b
print('y=',y)   # y=3
```

Python-ը հնարավորություն է տալիս միաժամանակ միևնույն արժեքը վերագրել մի քանի փոփոխականների: Օրինակ.

```
m=n=p = 10
```

Այս հրամանով նախ ստեղծվում է ամբողջ տիպի օբյեկտ 10 արժեքով, ապա բոլոր երեք փոփոխականները հղվում են այդ օբյեկտին:

Python-ը հնարավորություն է տալիս նաև մի քանի փոփոխականների միաժամանակ վերագրել տարբեր արժեքներ: Օրինակ.

```
x, y, z = 10, 20, "a"
```

Այս հրամանով x և y փոփոխականներին վերագրվել են ամբողջ տիպի օբյեկտներ, համապատասխանաբար 10 և 20 արժեքներով, իսկ z փոփոխականին վերագրվել է տողային տիպի օբյեկտ՝ "a" արժեքով:

Հաճախ անհրաժեշտ է լինում երկու՝ x և y փոփոխականների արժեքները տեղափոխել իրար հետ՝ x-ի արժեքը վերագրել y-ին, իսկ y-ի արժեքը՝ x-ին: Դա կարելի է անել, օրինակ, հետևյալ եղանակով.

```
hold=x
x=y
y=hold
```

Python-ում նախատեսված է նաև հետևյալ գեղեցիկ գրելաձևը, որը միաժամանակ և՛ ըմբռնելի է, և՛ համառոտ.

```
x,y=y,x
```

4.6. Անդամակցության օպերատորներ

Python լեզվի անդամակցության օպերատորները ստուգում են օբյեկտի անդամակցությունը որևէ տողի, ցուցակի, բառարանի, հավաքածուի կամ բազմության: Python-ն առաջարկում է երկու օպերատոր օբյեկտի անդամակցությունը ստուգելու համար: Դրանք են «in» և «not in» օպերատորները:

Ստորև ներկայացված աղյուսակում նկարագրված են Python-ի անդամակցության օպերատորները:

Օպերատոր	Նկարագրություն
x in y	True, եթե ձախ օպերանդը առկա է աջ օպերանդի մեջ, False՝ հակառակ դեպքում:
x not in y	True, եթե ձախ օպերանդը առկա չէ աջ օպերանդի մեջ, False՝ հակառակ դեպքում::

Աղյուսակ 4.7. Անդամակցության օպերատորները

«in» օպերատորն օգտագործվում է ստուգելու համար, թե արդյոք նշված տարրը գոյություն ունի՞ տողի, ցուցակի, բառարանի, հավաքածուի կամ բազմության մեջ, թե՞ ոչ: Եթե այո, ապա գործողության արդյունքը True է, False՝ հակառակ դեպքում:

Օրինակ.

```
lst = [10, 20, 30, 40, 50] # ցուցակ
str = "Python Programming" # տող
dct = {'TV':20000,'Monitor':15000,'Mouse':1200,'laptop':17000} # բառարան

# ցուցակում 30 թվի առկայության ստուգում
print(30 in lst)  #True

# str տողում q սիմվոլի առկայության ստուգում
print('q' in str)   #False


# dct բառարանում 'Mouse' բանալու առկայության ստուգում
print('Mouse' in dct)  #True
```

«not in» օպերատորը վերադարձնում է True, եթե նշված տարրը չի գտնվում որևէ տողի, ցուցակի, բառարանի, հավաքածուի կամ բազմության մեջ, և False՝ հակառակ դեպքում:

Օրինակներ.

```
list1 = [10, 20, 30, 40, 50] # ցուցակ
str = "Python Programming" # տող
dct = {'TV':20000,'Monitor':15000,'Mouse':1200,'laptop':17000} # բառարան
# ' not in' operator

print(30 not in list1)  #False
print('q' not in str)   #True
print('Mouse' not in dct)  #False
```

4.7. Նույնականացման օպերատորներ

Նույնականացման օպերատորներն օգտագործվում են երկու օբյեկտները համեմատելու համար, արդյո՞ք դրանք նույնն են, թե՞ ոչ: Նույնականացման օպերատորներն են՝ «is» և «is not»:

Ստորև ներկայացված աղյուսակում նկարագրված են Python-ի նույնականացման օպերատորները:

Օպերատոր	Նկարագրություն
x is y	Վերադարձնում է True, եթե երկու օպերանդներն էլ հղվում են նույն օբյեկտին, False՝ հակառակ դեպքում:
x is not y	Վերադարձնում է False, եթե երկու օպերանդներն էլ հղվում են նույն օբյեկտին, True՝ հակառակ դեպքում:

Աղյուսակ 4.8. Նույնականացման օպերատորները

«is» օպերատորը վերադարձնում է True, եթե երկու օպերանդներն էլ հղվում են նույն օբյեկտին, False՝ հակառակ դեպքում: Օրինակ՝

```
num1 = 5; num2 = 5

lst1 = [1, 2, 3]
lst2 = [1, 2, 3]
lst3 = lst1

str1 = "Python Programming"
str2 = "Python programming"

print(num1 is num2) #True
print(lst1 is lst2) #False
print(lst1 is lst3) #True
print(str1 is str2) #False

print('id(num1):', id(num1),'\n','id(num2):', id(num2))
print('id(lst1):', id(lst1),'\n','id(lst2):','id(lst2)','\n','id(lst3):',id(lst3))
print('id(str1):', id(str1),'\n','id(str2):', id(str2))
```

Ծրագրի գործարկման արդյունքն է՝

```
True
False
True
False
id(num1): 140707522274344
id(num2): 140707522274344
id(lst1): 3083169693312
id(lst2): 3083131534336
id(lst3): 3083169693312
id(str1): 3083175476080
id(str2): 3083175476016
```

Ծրագրի կատարման արդյունքներից երևում է, որ num1 և num2 փոփոխականները հղվում են միևնույն օբյեկտին, քանի որ դրանց id նույնականացուցիչները նույնն են։ lst1 և lst2 փոփոխականները հղվում են տարբեր օբյեկտների, lst1 և lst3 օբյեկտները՝ նույն օբյեկտին, իսկ str1 և str2 փոփոխականները՝ տարբեր օբյեկտների։

«is not» օպերատորը վերադարձնում է False, եթե երկու օպերանդները հղվում են միևնույն օբյեկտին, True՝ հակառակ դեպքում։ Ստորև բերված ծրագրում համեմատվել են առաջին խնդրում բերված օբյեկտները՝ «is not» օպերատորի միջոցով։

```
num1 = 5; num2 = 5
lst1 = [1, 2, 3]
lst2 = [1, 2, 3]
lst3 = lst1

str1 = "Python Programming"
str2 = "Python programming"

print(num1 is not num2) #False
print(lst1 is not lst2) #True
print(lst1 is not lst3) #False
print(str1 is not str2) #True
```

4.8. Օպերատորների կատարման առաջնահերթությունը

Օպերատորների, փակագծերի, հաստատունների, փոփոխականների ու ֆունկցիաների միջոցով կառուցվում են արտահայտություններ: Արտահայտությունն ունի որոշակի արժեք իր մեջ մտնող փոփոխականների ցանկացած թույլատրելի արժեքների համար: Արտահայտության արժեքը հաշվվում է ըստ օպերատորների կատարման կարգի առաջնահերթության: Փակագծերն ապահովում են գործողության կատարման առաջնահերթությունը:

Ստորև ներկայացված աղյուսակում ավելի բարձր առաջնահերթություն ունեցող գործողությունը գտնվում է ավելի ցածր առաջնահերթությամբ գործողությունից վերև, իսկ հավասարազոր գործողությունները գտնվում են նույն տողում: Արտահայտության մեջ հավասարազոր գործողությունները կատարվում են ձախից աջ: Այս կանոնից բացառություն է կազմում աստիճան բարձրացնելու գործողությունը: Երկու աստիճան բարձրացնելու գործողություններից սկզբում կկատարվի աջը, իսկ հետո՝ ձախը:

Օպերատորը	Նկարագրությունը
()	փակագծեր
**	աստիճան բարձրացնելու գործողություն
+x, -x, ~x	ունար գումարում, հանում (նշանափոխություն), բիթային ժխտում
*, /, //, %	բազմապատկում, բաժանում, ամբողջ բաժանում, մոդուլով բաժանում
+, -	գումարում, հանում
<<, >>	բիթային ձախ և աջ տեղաշարժ
&	բիթային “և”
^	բիթային “բացառիկ կամ”

|	բիթային “կամ”
in, not in, is, is not, <, <=, >, >=, !=, ==	անդամակցության, նույնականացման և համեմատության օպերատորներ
not	տրամաբանական ժխտում
and	տրամաբանական “և”
or	տրամաբանական “կամ”

Աղյուսակ 4.9. Օպերատորների կատարման առաջնահերթությունը Python-ում

Օրինակ՝ հաշվել (a**2%2 == 0 and b**3%2 == 1) or (a%2 > b%2) արտահայտության արժեքը, երբ a=13 և b=2:

Լուծում:

(a**2%2 == 0 and b**3%2 == 1) արտահայտության արժեքը հավասար է False, երբ a=13 և b=2, քանի որ 13**2%2 ==0 արտահայտության արժեքը հավասար է False, իսկ 2**3%2 ==1 արտահայտության արժեքը՝ նույնպես False:

(a%2 > b%2) արտահայտության արժեքը հավասար է True, երբ a=13 և b=2, քանի որ 13%2=1, 2%2=0:

False or True=True, հետևաբար տրված արտահայտության արժեքը հավասար է True, երբ a=13 և b=2:

Աստիճան բարձրացնելու գործողությունը աջ կողմնորոշված է, այսինքն՝ x ** y ** z արտահայտությունը կատարվում է աջից ձախ որպես x ** (y ** z), այլ ոչ թե որպես (x ** y) ** z:

Օրինակ՝ դիտարկենք ստորև բերված կոդը՝

```
print(2 ** 3 ** 4) # հաշվում է 2-ի 81 աստիճանը
print((2 ** 3) ** 4) # հաշվում է 8-ի 4 աստիճանը
```

Արտածվում է.

```
2417851639229258349412352
4096
```

Ծրագրավորման հիմնական հասկացություններից են ալգորիթմն ու ծրագիրը: Ալգորիթմը գործողությունների կարգավորված հաջորդականություն է, որը վերջավոր թվով քայլերից հետո հանգեցնում է դրված խնդրի լուծմանը, իսկ ծրագիրը այդ ալգորիթմի գրառումն է ծրագրավորման որևէ լեզվով:

Ցանկացած ալգորիթմ կարելի է ներկայացնել հետևյալ երեք կառուցվածքների միջոցով՝ գծային, ճյուղավորված և ցիկլային: Այն ալգորիթմները, որոնք իրականացվում են այդ կառուցվածքների միջոցով, համապատասխանաբար կոչվում են գծային, ճյուղավորված և ցիկլային ալգորիթմներ: Ծրագրավորման լեզուներն ունեն հրամաններ նշված ալգորիթմներն իրականացնելու համար:

Python լեզվով ծրագիրը ծրագրային միավորների հաջորդականություն է: Յուրաքանչյուր ծրագրային միավոր կազմված է Python հրամաններից, որոնցից յուրաքանչյուրն իրականացնում է որոշակի գործողություն:

Յուրաքանչյուր հրաման գրվում է առանձին տողի վրա: Որպես հրամանի ավարտ հանդես է գալիս տողի ավարտը: Եթե հրամանը ներդրված չէ, այն պետք է սկսվի տողի սկզբից, հակառակ դեպքում կտրվի հաղորդագրություն կատարված սխալի մասին՝ unexpected indent (նկար 5.1):

Նկար 5.1. Հաղորդագրություն սխալի մասին, երբ հրամանը գրված է չթույլատրված խորությամբ

Բերված օրինակում import հրամանի առջևում կա ավելորդ բացակ, որը հանգեցրել է առաջացած սխալի մասին հաղորդագրությանը։

Հրամանի մի մասը կարող է տեղափոխվել հաջորդ տող՝ տեղադրելով «\» նշանը տողի վերջում, օրինակ՝

```
x = 15 + 20  \
   + 30
print(x)
```

Նույնը կարելի է անել նաև՝ վերցնելով արտահայտությունը փակագծերի մեջ։

```
x = (15 + 20
   + 30)
print(x)
```

Եթե անհրաժեշտ է մի քանի հրամաններ տեղադրել մեկ տողում, ապա դրանք պետք է իրարից անջատել կետ-ստորակետով։ Օրինակ՝

```
x = 10; y = 20; z = x  +  y  # երեք հրաման մեկ տողի վրա
print(z)
```

Հարկ է նշել Python-ի մի հետաքրքիր առանձնահատկություն։ Այն չի պարունակում փակագծեր, որոնցով նշվում են ֆունկցիաների ու դասերի բլոկները (օրինակ՝ Pascal ծրագրավորման լեզվում begin. . . end, կամ՝ C-ում ձևավոր փակագծեր {. . . }): Python-ում բլոկի հրամանները գրվում են խորությամբ՝ որոշակի թվով սահմանված բացակներով կամ տաբուլյացիայով։ Օրինակ.

```
i = 1
while i < 11:
   print(i)
   i += 1
print("End of program")
```

Բերված օրինակում while բլոկի ներսում բոլոր հրամաններն ունեն նույն քանակությամբ բացակներ իրենցից առաջ, գրված են նույն խորությամբ: Դրանով էլ հենց Python-ի ինտերպրետատորը ճանաչում է բլոկի հրամանները: Հրամանները, որոնք ունեն նույն քանակությամբ բացակներ իրենցից առաջ, կազմում են բլոկի մարմինը (block body):

Վերևում բերված օրինակում while բլոկի մարմինը կազմված է երկու հրամանից, դրանք են.

```
print(i)
i += 1
```

Ծրագրի վերջին տողում գրված print("End of program") հրամանը չի պատկանում բլոկի մարմնին, քանի որ այն չի գտնվում նույն խորության վրա: Նշենք, որ Python-ում բլոկի մարմնի հրամաններն առանձնացնելու համար լռելյայն ընդունված է չորս բացակ:

5.1. if պայմանական հրամանը: pass հրամանը

Python լեզվում if հրամանը կիրառվում է գործողությունների ճյուղավորումներ կազմակերպելու համար՝ կախված որևէ պայմանի ճիշտ կամ սխալ լինելուց:

if պայմանական հրամանի շարահյուսությունն է.

```
if պայման:
հրամանների խումբ_1
else:
հրամանների խումբ_2
```

else ճյուղի բացակայության դեպքում if հրամանն ընդունում է հետևյալ տեսքը.

```
if պայման:
հրամանների խումբ
```

Առաջին կառուցվածքում if հրամանն աշխատում է հետևյալ ձևով. նախ ստուգվում է պայմանը, որը տրամաբանական արտահայտություն է: Եթե այդ արտահայտության արժեքը հավասար է True (պայմանը ճիշտ է), ապա կատարվում է հրամանների խումբ_1-ը, իսկ else-ից հետո գրված հրամանները չեն կատարվում: Հակառակ դեպքում՝ կատարվում է հրամանների խումբ_2-ը, իսկ հրամանների խումբ_1-ը բաց է թողնվում: Այնուհետև երկու դեպքում էլ ղեկավարումը տրվում է if-ին հաջորդող հրամանին:

if հրամանի երկրորդ կառուցվածքում, երբ else ճյուղը բացակայում է, պայմանի True արժեքի դեպքում կատարվում է նշված հրամանների խումբը, իսկ False արժեքի դեպքում՝ այն բաց է թողնվում: Այնուհետև ղեկավարումը տրվում է if-ին հաջորդող հրամանին:

Դիտարկենք հետևյալ խնդիրները:

Խնդիր: Գրել ծրագիր, որը գեներացնում է [1,10] միջակայքից որևէ պատահական ամբողջ թիվ, իսկ ծրագրորդը փորձում է կռահել այդ թիվը: Ծրագիրը պետք է տա հաղորդագրություն այն մասին, թե ծրագրորդն արդյոք կռահե՞լ է թիվը, թե՞ ոչ, և արտածի գեներացված պատահական թիվը:

```
from random import randint
number=randint(1,10) # գեներացվում է պատահական ամբողջ թիվ
num=eval(input('Enter your number: ')) # մուտքագրած թիվը
if num==number:
   print('You won. The random number=', number)
else:
   print('You didn\'t guess. The random number =', number)
```

Ծրագրի կատարման արդյունքը:

```
Enter your number: 6
You didn't guess. The random number = 5
```

Ծրագրում օգտագործվել է randint() ֆունկցիան, որը գեներացնում է [1,10] միջակայքից որևէ պատահական ամբողջ թիվ:

Այս ֆունկցիան գտնվում է random մոդուլում, ուստի ֆունկցիային դիմելու համար նախապես import հրամանով մոդուլը ներմուծվում է հիշողության մեջ (մոդուլների մասին՝ տե՛ս *գլուխ 13*):

Ինչպես տեսնում ենք, ծրագրի կատարման ժամանակ պատահականորեն գեներացված էր 5 թիվը, սակայն ծրագրողն այն ճիշտ չէր կռահել, ուստի տրվել էր «You didn't guess. The random number = 5" հաղորդագրությունը:

Խնդիր: Խանութի վաճառողը ստանում է հավելավճար իր վաճառքի ծավալի 5%-ի չափով, եթե այն գերազանցում է 90000 պ.դ.մ: Գրել ծրագիր, որը հաշվում է վաճառողի հավելավճարի մեծությունը (h)՝ ըստ նրա վաճառքի ցուցանիշի (v):

```
v=eval(input('enter sales: '))
if v<=90000:
   r=0
else:
   r=5
h=0.01*v*r
print('h=', h)
```

if... else հրամանը կարելի է գրել նաև մեկ տողում: Այդ դեպքում դրա շարահյուսությունն է՝

հրաման1 if պայման else հրաման2

if... else հրամանի այս միատող կառուցվածքում պայմանի True արժեքի դեպքում կատարվում է հրաման1-ը, իսկ False արժեքի դեպքում՝ հրաման 2-ը: Օրինակ՝ գրենք ծրագիր, որը ստուգում է, արդյոք մուտքագրված թիվը վերջանո՞ւմ է 7-ով, թե՞ ոչ:

```
num=eval(input('Enter a number'))
r=num%10
print('The number ends with 7') if r==7 else print('The number doesn't end with 7')
```

Հաճախ ծրագրում անհրաժեշտ է լինում ստուգել ոչ թե մեկ, այլ մի քանի պայման: Նման դեպքերում օգտագործվում է պայմանական հրամանի if... elif... else կառուցվածքը, այն է՝

```
if պայման_1:
հրամանների խումբ_1
elif պայման_2:
հրամանների խումբ_2
……
elif պայման_n:
հրամանների խումբ_n
else:
հրամանների խումբ
```

if... elif... else հրամանը նախ ստուգում է պայման_1-ը: Եթե այդ պայմանը տեղի ունի, ապա կատարվում է հրամանների խումբ_1-ը, որից հետո ղեկավարումը տրվում է if պայմանական հրամանին հաջորդող հրամանին: Եթե պայման_1-ը տեղի չունի, ապա այդ դեպքում ստուգվում է հաջորդ՝ պայման_2-ը: Եթե պայման_2-ը տեղի ունի, ապա կատարվում է հրամանների խումբ_2-ը, որից հետո ղեկավարումը տրվում է if պայմանական հրամանին հաջորդող հրամանին: Եթե պայման_2-ը տեղի չունի, այդ դեպքում ստուգվում է հաջորդ՝ պայման_3-ը և այսպես շարունակ: Եթե նշված պայմաններից և ոչ մեկը տեղի չունի, ապա կատարվում է else-ից հետո գրված հրամանների խումբը և անցում է կատարվում ծրագրի հաջորդ հրամանին:

Խնդիր: Գրել ծրագիր, որը հաշվարկում է անշարժ գույքի վաճառքով զբաղվող գործակալին տրվող պարգևավճարի մեծությունը (h)՝ կախված նրա վաճառքի ցուցանիշից (v) հետևյալ կանոնով՝ պարգևավճարը կազմում է վաճառքի ցուցանիշի 5%-ը՝ եթե այն մեծ է 2 մլն դրամից և չի գերազանցում 4 մլն դրամը, 10%-ը՝ եթե մեծ է 4 մլն դրամից և չի գերազանցում 6 մլն դրամը, և 15%-ը՝ եթե 6 մլն դրամից ավելի է:

```
v=eval(input('enter sales: '))
if v<=2000000:
    r=0
elif v<=4000000:
    r=5
```

```
elif v<=6000000:
    r=10
else:
    r=15
h=0.01*v*r
print('h=', h)
```

if պայմանական հրամանում կարող են լինել նաև ներդրված if հրամաններ, օրինակ՝

```
if պայման_1:
    հրամանների խումբ_1
else:
    if պայման_2:
      հրամանների խումբ_2
else:
      հրամանների խումբ_3
```

Խնդիր: Ներդրված if հրամանը կիրառելով՝ գրենք ծրագիր, որն արտածում է քառակուսի հավասարման արմատների քանակը:

```
a=eval(input('Enter non zero a=')) #հավասարման ավագ անդամի գործակիցը
b=eval(input('Enter b=')) #հավասարման միջին անդամի գործակիցը
c=eval(input('Enter c=')) #հավասարման ազատ անդամը
D=b**2-4*a*c
if D>0:
    print('The equation has two roots')
else:
    if D==0:
        print('The equation has one root')
    else:
        print('The equation has no roots')
```

Բերված ծրագրում ներածվում են քառակուսի հավասարման ավագ և միջին անդամների գործակիցները, ազատ անդամը, ապա հաշվվում է դիսկրիմինանտի արժեքը: Եթե դիսկրիմինանտը դրական է, ուրեմն քառակուսի հավասարումն ունի երկու իրարից տարբեր արմատ, հավասար է զրոյի՝ մեկ արմատ (երկու իրար հավասար արմատ), բացասական է՝ իրական արմատներ չունի:

Օրինակ՝ մուտքագրենք $x^2 + x + 1 = 0$ քառակուսի հավասարման գործակիցները՝ $a = 1, b = 1, c = 1$: Քանի որ դիսկրիմինանտը բացասական է, ուրեմն կաշխատի else ճյուղում ներդրված else ճյուղը (նկար 5.2).

Նկար 5.2. Քառակուսի հավասարման արմատների քանակը հաշվող ծրագրի աշխատանքը

Python լեզվի if հրամանում, ցիկլերում, ֆունկցիաների կամ դասերի սահմանումներում չեն կարող լինել դատարկ բլոկներ: Եթե անհրաժեշտ է, որ լինեն կոդի բլոկներ, բայց դրանք ոչինչ չանեն (լինեն դատարկ), պետք է օգտագործել pass հրամանը: pass հրամանը թույլ է տալիս խուսափել սխալներից, երբ դատարկ կոդը թույլատրված չէ, իսկ հետո՝ այն փոխարինել անհրաժեշտ կոդով: Օրինակ՝

```
a = 10
b = 30
if b > a:
    pass
```

Փոխարինելով pass հրամանը print(b, 'is greather than', a) հրամանով, կստանանք՝

```
a = 10
b = 30
if b > a:
    print(b, 'is greather than', a)
```

Հետևյալ դատարկ for ցիկլում առանց pass հրամանի կառաջանար սխալ (for հրամանի մասին տե՛ս *պարագրաֆ 5.2*)։

```
for x in [0, 1, 2]:
    pass
```

Նկար 5.3. Առանց pass հրամանի դատարկ for ցիկլում ծագած սխալի մասին տրված հաղորդագրությունը

Ստորև բերված օրինակում pass հրամանն օգտագործվել է myfunction() ֆունկցիայի սահմանման մեջ։ Ֆունկցիայի մարմինը դատարկ է, որը թույլատրված չէ (ֆունկցիաների մասին տե՛ս *գլուխ 11*)։

```
def myfunction():
    pass
```

pass հրամանը թույլ է տալիս խուսափել սխալից, հետագայում այն կարելի է փոխարինել համապատասխան կողով, օրինակ՝ print('Hello!') հրամանով.

```
def myfunction():
    print('Hello!')
```

5.2. for հրամանը: range() ֆունկցիան

for հրամանը կրկնում է մեկ կամ մի քանի հրամանների խումբը՝ նախապես հայտնի քանակությամբ: for հրամանի շարահյուսությունն է.

```
for ցիկլի փոփոխական in range(start,stop,step):
    ցիկլի մարմին
```

for հրամանում range() ֆունկցիայի պարամետրերն են. start՝ ցիկլի փոփոխականի սկզբնական արժեքը, stop՝ վերջնական արժեքը (որը չի ներառվում), step՝ ցիկլի փոփոխականի արժեքի փոփոխման քայլը:

for հրամանը կատարվում է հետևյալ կերպ. ցիկլի փոփոխականին վերագրվում է start սկզբնական արժեքը, ապա այն համեմատվում է stop վերջնական արժեքի հետ: Քանի դեռ ցիկլի փոփոխականի արժեքը փոքր է վերջնական արժեքից դրական քայլի դեպքում կամ մեծ է վերջնական արժեքից բացասական քայլի դեպքում, կատարվում է ցիկլի մարմինը (ցիկլի մեջ մտնող հրամանների խումբը), հակառակ դեպքում՝ for հրամանն ավարտում է իր աշխատանքը: Ցիկլի յուրաքանչյուր հերթական քայլը կատարելուց հետո ցիկլի փոփոխականի արժեքը փոխվում է step քայլով: Այն մասնավոր դեպքում, երբ ցիկլի փոփոխականն ամբողջ տիպի է, և step քայլը բացակայում է, լռությամբ ընդունվում է, որ քայլը հավասար է 1:

Եթե range() ֆունկցիան ունի միայն մեկ արգումենտ, նշանակում է, որ դա stop արժեքն է: Ֆունկցիայի շարահյուսությունն է՝

```
for ցիկլի փոփոխական in range(stop):
    ցիկլի մարմին
```

Այս դեպքում ցիկլի փոփոխականի start սկզբնական արժեքը լռությամբ հավասար է 0, step քայլը՝ 1, ուստի ցիկլի փոփոխականն ընդունում է 0, 1, 2, ..., (stop-1) արժեքները, և ցիկլը կատարվում է ճիշտ stop անգամ:

Եթե range() ֆունկցիան ունի երկու արգումենտ, նշանակում է, դրանք start և stop արժեքներն են: Այս դեպքում կրկին ցիկլի

փոփոխականի step քայլը լռությամբ ընդունվում է 1: Շարահյուսությունն է՝

for i in range(start, stop):
ցիկլի մարմին

Բերենք մի քանի օրինակներ:

Հրաման	Ցիկլի i փոփոխականի ընդունած արժեքները
for i in range(10):	i=0,1,2,3,4,5,6,7,8,9
for i in range(1,10):	i=1,2,3,4,5,6,7,8,9
for i in range(3,7):	i=3,4,5,6
for i in range(2,15,3):	i=2,5,8,11,14
for i in range(9,2,-1):	i=9,8,7,6,5,4,3

Դիտարկենք հետևյալ օրինակը.

```
for i in range(5):
   print ('Python')
```

Այս օրինակում ցիկլի i փոփոխականն ընդունում է 0, 1, 2, 3, 4 արժեքները, և ցիկլը կատարվում է հինգ անգամ: Արդյունքում հինգ անգամ արտածվում է Python բառը:

```
Python
Python
Python
Python
Python
```

Դիտարկենք մի քանի խնդիրներ:

Խնդիր: Գրել ծրագիր, որը երեք անգամ հաջորդաբար մուտքագրում է որևէ թիվ և արտածում դրա քառակուսին:

```
for i in range(3):
   num=eval(input('Enter a number: '))
   print('The square of the number', num, 'is: ', num**2)
print('The end')
```

Գործարկենք ծրագիրը, կունենանք.

```
Enter a number: 51
The square of the number 51 is:  2601
Enter a number: 24
The square of the number 24 is:  576
Enter a number: -12
The square of the number -12 is:  144
The end
```

Խնդիր: Հետևյալ ծրագիրը նախ տպում է A տառը, ապա՝ B տառը, հետո հինգ անգամ տպում է C, D տառերը և վերջում՝ E տառը:

```
print('A')
print('B')
for i in range(5):
   print('CD')
print('E')
```

Ծրագրի կատարման արդյունքն է՝

```
A
B
CD
CD
CD
CD
CD
E
```

Համոզվելու համար, որ ծրագրում i ցիկլի փոփոխականն ընդունում է 0, 1, 2, 3, 4 արժեքները, արտածենք i-ի արժեքները.

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

Կստանանք.

```
0
1
2
3
4
```

Այժմ ծրագիրը գրենք հետևյալ ձևով.

```
print('A','B','CD'*5,'E', sep='; ')
```

Ծրագրի այս հատվածում print() ֆունկցիայի sep օպցիոնալ պարամետրի արժեքը սահմանել է այն սիմվոլը, որով անջատվելու են արտածվող մեծությունները: Լռելյայն այդ արժեքը բացակն է: Ծրագրում որպես անջատիչ սահմանվել է ';' սիմվոլը, ուստի արդյունքում արտածվել է՝ A;B;CDCDCDCDCD;E տեքստը:

Դիտարկենք ևս մեկ օրինակ:

```
for i in range(5,0,-1):
    print(i,end=' ')
print('The end')
```

Այս ծրագրում print() ֆունկցիայի end=' ' պարամետրի արժեքը նշանակում է, որ տպելիս անցում չի կատարվում նոր տողի, և i փոփոխականի արժեքներն ու 'The end' տեքստը արտածվում են նույն տողում: Ծրագրի կատարման արդյունքում կստանանք՝ 5 4 3 2 1 The end

5.3. while հրամանը: break և continue հրամանները

while հրամանը նույնպես պատկանում է ցիկլի հրամանների թվին: while հրամանի շարահյուսությունն է.

```
while պայման:
ցիկլի մարմին
```

while հրամանը կատարում է հրամանների խումբը այնքան ժամանակ, քանի դեռ գրված պայմանը ճիշտ է:

Բացատրենք while հրամանի աշխատանքը հետևյալ օրինակի միջոցով: Դիցուք, պետք է տպել [1, 10] միջակայքի ամբողջ թվերը: Իհարկե, դա կարելի է կատարել՝ օգտագործելով for հրամանը.

```
for i in range(1,11):
   print(i,end=' ')
```

Գրենք նույն խնդիրը while հրամանի միջոցով:

```
i=1
while i<11:
   print(i, end=' ')
   i+=1
```

Ծրագրի կատարման արդյունքն է՝

1 2 3 4 5 6 7 8 9 10

Ծրագրում while հրամանն աշխատում է հետևյալ ձևով: Նախ i ցիկլի փոփոխականը ստանում է 1 արժեքը, ապա այդ արժեքը համեմատվում է 11-ի հետ: Քանի որ 1<11, կատարվում է ցիկլի մարմինը, այն է՝ տպվում է i-ի արժեքը՝ 1, ապա i փոփոխականի արժեքը ավելացվում է 1-ով: Այժմ i փոփոխականի արժեքը՝ i=2: Նոր արժեքը համեմատվում է 11-ի հետ, և, քանի որ 2<11, տպվում է 2 թիվը, ու i փոփոխականի արժեքը ավելացվում է 1-ով: Եվ այսպես շարունակ՝ i փոփոխականի արժեքը տպվում է այնքան ժամանակ, քանի դեռ այն փոքր է 11-ից: while հրամանն ավարտում է իր աշխատանքը, երբ որ i-ն հավասարվում է տասնմեկի:

Խնդիր: Գտնել տրված m և n երկու բնական թվերի ամենամեծ ընդհանուր բաժանարարը՝ (m, n):

Երկու բնական թվերի ամենամեծ ընդհանուր բաժանարարը գտնենք Էվկլիդեսի ալգորիթմով: Ըստ ալգորիթմի՝ համեմատվում են m և n փոփոխականների արժեքները: Եթե m>n, ապա m փոփոխականին վերագրվում է (m-n) արժեքը, իսկ m<n դեպքում՝ n փոփոխականին վերագրվում է (n-m) արժեքը, որից հետո համեմատվում են փոփոխականների նոր արժեքները: Գործընթացն

ավարտվում է այն ժամանակ, երբ m=n: m (կամ n) փոփոխականի ստացված արժեքն էլ հանդիսանում է m և n երկու բնական թվերի ամենամեծ ընդհանուր բաժանարարը՝ (m, n):

```
m=eval(input('Enter the first number: '))
n=eval(input('Enter the second number: '))
while m!=n:
   if m>n:
      m=m-n
   else:
      n=n-m
print('(m,n)=',m)
```

Գործարկենք ծրագիրը: Մուտքագրենք 96 և 108 թվերը ու հաշվենք դրանց ամենամեծ ընդհանուր բաժանարարը:

```
Enter the first number: 96
Enter the second number: 108
(m,n)= 12
```

Խնդիր: Ստեղնաշարից հաջորդաբար մուտքագրել իրական թվեր և հաշվել ոչ բացասական թվերի գումարը և քանակը: Ծրագրի աշխատանքն ավարտել այն ժամանակ, երբ ստեղնաշարից կմուտքագրվի որևէ բացասական թիվ:

```
s=0
q=0
number=eval(input('Enter a number: '))
while number>=0:
   s+=number
   q+=1
   number=eval(input('Enter a number: '))
print('s=', s)
print('You entered', q, 'non negative numbers')
```

Ստորև բերված է ծրագրի կատարման արդյունքը, երբ մուտքագրվել են 5, 6, 8, 0, -3 թվերը:

```
Enter a number: 5
Enter a number: 6
Enter a number: 8
Enter a number: 0
Enter a number: -3
s= 19
You entered 4 non negattive numbers
```

Խնդիր: Տրված x կետում հաշվել e^x ֆունկցիայի արժեքը նախօրոք տրված $\varepsilon > 0$ ճշտությամբ.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots$$

Հաշվարկները կկատարենք հետևյալ ռեկուրենտ առնչությունների միջոցով.

$$e^x = \sum_{k=0}^{\infty} u_k,\ \ u_k = \frac{x}{k} u_{k-1},\ \ S_k = S_{k-1} + u_k,\ \ (k = 1, 2, \ldots)$$

որտեղ $u_0 = 1, S_0 = 1$:

Հաշվարկները կդադարեցնենք այն ժամանակ, երբ հերթական գումարելին բացարձակ արժեքով կդառնա ավելի փոքր, քան նախօրոք տրված ε-ը՝ $|u_k| < \varepsilon$:

```
eps=eval(input('Enter the occurrence: eps='))
x=eval(input('Enter x='))
sum=0
sk=1
k=0
while abs(sk)>=eps:
   sum=sum+sk
   k+=1
   sk*=x/k
print('e**', x, '=', sum)
```

Հաշվենք e^x ֆունկցիայի արժեքները x-ի հետևյալ արժեքների համար՝ x=0, 1, 2, -5:

```
Enter the occurrence: eps=0.00001
Enter x=0
e** 0 = 1
Enter the occurrence: eps=0.00001
Enter x=1
e** 1 = 2.71827876984127

Enter the occurrence: eps=0.00001
Enter x=2
e** 2 = 7.389046015712681

Enter the occurrence: eps=0.00001
Enter x=-5
e** -5= 0.006745540097709548
```

Կարող են հանդիպել դեպքեր, երբ while հրամանը երբեք չի դադարեցնում իր աշխատանքը, այլ անվերջ կրկնվում է, օրինակ.

```
i=0
while i<20:
   print(i)
```

Բերված օրինակում i փոփոխականի արժեքը երբեք չի փոխվում, այն միշտ հավասար է զրոյի, որը փոքր է 20-ից, ուստի այս ցիկլն անվերջ կատարվում է:

Մյուս կողմից, հնարավոր է, որ while հրամանը երբեք չկատարվի: Օրինակ.

```
x=20
while x<1:
   print (x)
```

Բերված ծրագրում x փոփոխականի արժեքը հավասար է 20, հետևաբար x<1 տրամաբանական արտահայտության արժեքը միշտ False է, այսինքն՝ while հրամանը երբեք չի կատարվի:

while ցիկլի հրամանի աշխատանքը կարելի է հարկադրաբար ընդհատել break հրամանի միջոցով:

Հետևյալ ծրագրում while հրամանը պետք է արտածի 7-ից փոքր բնական թվերը՝ սկսած 1-ից։ Սակայն, երբ ցիկլի փոփոխականի արժեքը հավասարվում է 4-ի, այդ թիվը տպելուց հետո break հրամանով ցիկլի աշխատանքը դադարեցվում է։

```
i = 1
while i < 7:
  print(i, end=' ')
  if i == 4:
    break
  i += 1
```

Ծրագրի կատարման արդյունքն է՝

```
1 2 3 4
```

Continue հրամանը, ի տարբերություն break հրամանի, ոչ թե ընդհատում է ցիկլի աշխատանքը, այլ չի կատարում այդ հրամանից մինչև ցիկլի մարմնի վերջը գտնվող հրամանները, և ցիկլի աշխատանքը շարունակվում է հաջորդ իտերացիայից։ Օրինակ՝ ստորև բերված ծրագրում ցիկլի մարմինը չի կատարվում i = 4 արժեքի դեպքում, և աշխատանքը շարունակվում է հաջորդ՝ i=5 քայլից։

```
i = 0
while i < 7:
  i += 1
  if i == 4:
    continue
  print(i, end=' ')
```

Ծրագրի կատարման արդյունքն է՝

```
1 2 3 5 6 7
```

while հրամանն ունի նաև while ... else տարբերակը։

while պայման:

հրամանների խումբ_1
else:
հրամանների խումբ_2

while հրամանն այս կառուցվածքով աշխատում է հետևյալ ձևով: Քանի դեռ գրված պայմանը ճիշտ է, կատարվում է հրամանների առաջին խումբը, հակառակ դեպքում՝ հրամանների երկրորդ խումբը: Սակայն, եթե հրամանների առաջին խմբում հանդիպում է break հրաման, այդ դեպքում ցիկլի աշխատանքը դադարեցվում է, և else հրամանախումբը ընդհանրապես չի կատարվում:

Հետևյալ ծրագիրը ստուգում է, թե արդյոք բնական թիվը (սկսած մեկից) փո՞քր է յոթից: Եթե այո, թիվն արտածվում է, հակառակ դեպքում կատարվում է else ճյուղը, և ցիկլն ավարտում է իր աշխատանքը:

```
i = 1
while i < 7:
  print(i)
  i += 1
else:
  print("i is greater than or equal to 7")
```

Նշենք, որ for ցիկլերում ևս, ինչպես while ցիկլերի դեպքում, ցիկլի աշխատանքը դադարեցնելու կամ ցիկլի մեկ իտերացիան շրջանցելու համար կարելի է օգտագործել break և continue հրամանները համապատասխանաբար: for … else կառուցվածքն աշխատում է այնպես, ինչպես while … else հրամանը:

Հետևյալ ծրագիրը տպում է այն ամբողջ թվերը, որոնք պատկանում են [0,6] միջակայքին: Հակառակ դեպքում՝ աշխատում է else ճյուղը, տրվում է հաղորդագրություն, և ցիկլն ավարտում է իր աշխատանքը:

```
for x in range(7):
  print(x)
else:
  print("x is greater than or equal to 7")
```

5.4. Խնդիրներ

Խնդիր 1: Մուտքագրել տասը թիվ և հաշվել 100-ից մեծ թվերի գումարը, քանակն ու միջին թվաբանականը: Եթե այդ պայմանին բավարարող թիվ չի մուտքագրվել, տալ այդ մասին հաղորդագրություն:

```
s=0; count=0
for i in range(10):
    num=eval(input('Enter a number: '))
    if num>100:
        s+=num
        count+=1
print('s=', s, 'count=', count)
if count==0:
    print("There aren't numbers greather than 100")
else:
    print('the average of numbers is', s/count)
```

Խնդիր 2: Հաշվել առաջին 100 բնական թվերից այն թվերի քանակը, որոնց քառակուսիները վերջանում են 5-ով:

```
count=0
for i in range(1,101):
    if i**2%10==5:
        count+=1
print(count)
```

Խնդիր 3: Հաշվել առաջին 100 բնական թվերի գումարը և միջին թվաբանականը:

```
s=0
for i in range(1,101):
    s+=i
print('The sum is: ', s)
print('The average is: ', s/100)
```

Խնդիր 4: Գրել ծրագիր, որը ստուգում է, արդյոք մուտքագրված թիվը պա՞րզ է, թե՞ բաղադրյալ:

Խնդրի լուծման համար մուտքագրված n թիվը հաջորդաբար կբաժանենք 2, 3, … , n//2 (թվի կեսի ամբողջ մասը) թվերի: Եթե որևէ քայլում բաժանումից ստացված մնացորդը հավասար լինի զրոյի, ուրեմն թիվը բաղադրյալ է, ուստի կտրվի հաղորդագրություն այդ մասին, և ծրագիրը կավարտի իր աշխատանքը: Հակառակ դեպքում (ոչ մի քայլում բաժանումից ստացված մնացորդը հավասար չէ զրոյի) կտրվի հաղորդագրություն, որ մուտքագրված թիվը պարզ է, և ծրագիրը կավարտի իր աշխատանքը:

```
number=eval(input('Enter a number: '))
num1=number//2+1
flag=0 #պարզ թիվ
for i in range(2,num1):
    if number%i==0:
        flag=1
        break
if flag==1:
    print('The number', number, 'is not prime')
else:
    print('The number', number, 'is prime')
```

Այժմ գրենք նույն խնդրի ծրագիրը while հրամանի միջոցով:

```
T=True #թիվը պարզ է
number=eval(input('Enter a number: '))
i=2
while i<=number//2:
    if number%i==0:
        T=False
        break
    else:
        i+=1
if T==False:
    print('The number',number,'is not a prime number')
```

```
else:
   print('The number',number,'is a prime number')
```

Խնդիր 5: Գեներացնել [10, 20] միջակայքին պատկանող որևէ պատահական ամբողջ թիվ և Python բառը գրել այդքան անգամ:

```
from random import randint
n=randint(10,20)
print('n=',n)
for i in range(n):
   print('Python')
```

Խնդիր 6: Գեներացնել [1, 100] միջակայքից հիսուն պատահական թիվ և հաշվել այն թվերի քանակը, որոնք բազմապատիկ են 7-ին:

```
from random import randint
count=0
for i in range(50):
   number=randint(1,100)
   if number%7==0:
      count+= 1
print('Number of multipies of 7 =',count)
```

Խնդիր 7: Ստեղնաշարից թույլ է տրվում մուտքագրել թվեր մինչև հարյուր անգամ: Պետք է հաշվել ներածված ոչ բացասական թվերի գումարն ու քանակը: Ցիկլի աշխատանքը անհրաժեշտ է դադարեցնել այն ժամանակ, երբ ստեղնաշարից կներածվի որևէ բացասական թիվ:

```
s=0; q=0; i=1
while i<100:
   number=eval(input('Enter a number: '))
   if number>=0:
      s+=number
      q+=1
      i+=1
```

```
    else:
        break
print ('s=',s)
print('q=',q)
```

Python ծրագրավորման մեջ թույլ տրված սխալներից է հրամանի՝ խորքից ոչ ճիշտ գրելը: Օրինակ՝ խնդիր 6-ի ծրագիրը գրենք հետևյալ ձևով.

```
from random import randint
count=0
for i in range(50):
    number=randint(1,100)
    if number%7==0:
        count +=1
    print('Number of multipies of 7 =',count)
```

Այս գրության մեջ print հրամանը վերաբերում է for հրամանին, ուստի count փոփոխականի արժեքը կարտածվի յուրաքանչյուր անգամ, երբ գեներացվում է նոր պատահական թիվ, այսինքն՝ i ցիկլի փոփոխականի յուրաքանչյուր արժեքի դեպքում՝ ճիշտ 50 անգամ:

Այժմ դիտարկենք նույն խնդրի ծրագրի հրամանների մեկ այլ գրություն.

```
from random import randint
count=0
for i in range(50):
    number=randint(1,100)
    if number%7==0:
        count+=1
 print('Number of multipies of 7 =', count)
```

Այս ծրագրում print հրամանը if հրամանի մասն է, հետևաբար count փոփոխականի արժեքը կարտածվի յուրաքանչյուր անգամ, երբ գեներացված պատահական թիվը կլինի 7-ի բազմապատիկ:

ԳԼՈՒԽ 6. ՏՈՂԵՐ

6.1. Տողերի սահմանումը և հիմնական գործողությունները

Տողը (Strings) սիմվոլների հաջորդականություն է: Այն վերցվում է ապաթարցերի կամ չակերտների մեջ: Տողերը լինում են միատող կամ բազմատող: Միատող տեքստը վերցվում է ապաթարցերի կամ չակերտների մեջ, իսկ երկար տողը՝ եռակի չակերտների կամ ապաթարցերի մեջ: Երկար տողը կարելի է տրոհել մասերի և տեղադրել առանձին տողերի վրա: Այն տողը, որը ոչ մի սիմվոլ չի պարունակում, կոչվում է դատարկ տող և գրվում է հետևյալ ձևով՝ '': Օրինակներ՝

```
s1='This is a line written on one line.'
s2="This is also a line written on one line."
s3="""Strings in python are surrounded by either single quotation marks, or double quotation marks. If we need a string to span multiple lines then we can use triple quotes"""
```

Python 3-ում յուրաքանչյուր տող լռելյայն Unicode սիմվոլների հաջորդականություն է:[7] Unicode հաջորդականությունում սիմվոլները ներկայացվում են uXXXX ձևաչափով, որտեղ XXXX-ը 16-ական համակարգի թիվ է:

Օրինակ՝ լատինական այբուբենի տառերին համապատասխանող կոդերն են՝

```
A - u0041, a - u0097
B - u0042, b - u0098
C - u0043, a - u0099
……..……
```

Հայերեն այբուբենի տառերին համապատասխանող կոդերն են՝

```
Ա - u0531, ա - u0561
Բ - u0532, բ - u0562
```

[7] https://home.unicode.org/

Գ - u0533, գ-u0563

.....

Հետևյալ կոդը արտածում է լատինական այբուբենի առաջին երեք մեծատառերը.

```
print('\u0041')
print('\u0042')
print('\u0043')
```

ord(symbol) ֆունկցիան վերադարձնում է symbol սիմվոլին համապատասխանող թիվը, իսկ chr(number) ֆունկցիան հակառակը՝ number ամբողջ թվին համապատասխանող սիմվոլը: Հետևյալ կոդով արտածվում են հայերեն այբուբենի առաջին տասը մեծատառերը, փոքրատառերը ու դրանց համապատասխան Unicode թվերը:

```
for number in range(1329,1339): #թվերը գրված են 10-ական համակարգում
    print(chr(number), '-', number, ', ', end='')
for number in range(1377,1387):
    print(chr(number), '-', number, ', ', end='')
```

Ծրագրի աշխատանքի արդյունքն է՝

Ա - 1329 , Բ - 1330 , Գ - 1331 , Դ - 1332 , Ե - 1333 , Զ - 1334 , Է - 1335 , Ը - 1336 , Թ - 1337 , Ժ - 1338 , ա - 1377 , բ - 1378 , գ - 1379 , դ - 1380 , ե - 1381 , զ - 1382 , է - 1383 , ը - 1384 , թ - 1385 , ժ - 1386 ,

Տողը կարելի է ներածել input() ֆունկցիայի միջոցով: Օրինակ՝

```
s1=input('Enter a string')
```

Տողի երկարությունը կարելի է հաշվել len() ներկառուցված ֆունկցիայի միջոցով: Օրինակ՝

```
len('Python')=6
```

Տողային տվյալների համար նախատեսված են կցում՝ (+) (concatenation) և տողի կրկնում՝ (*) գործողությունները:

Կցում գործողությամբ տողերը միավորվում են մեկ տողի մեջ: Երկու տող կցելիս երկրորդ տողը կցվում է առաջին տողի վերջին: Կցման արդյունքում ստացվում է տող, որի երկարությունը հավասար է կցվող տողերի երկարությունների գումարին: Օրինակ՝

```
s1='Programming'
s2='in'
s3='Python'
s=s1+' '+s2+' '+s3
print(s) #պատասխան՝ Programming in Python
```

Տողի կրկնում գործողությամբ տողը կրկնվում է նշված թվով անգամ: Օրինակ՝

```
print('AB'*5) # 'ABABABABAB'
```

Խնդիր: Հաջորդաբար մուտքագրել տասը սիմվոլ և դրանք կցել իրար:

```
s=''
for i in range(10):
    str=input('Enter a symbol: ')
    s+=str
print(s)
```

in օպերատորը: in օպերատորը ստուգում է որևէ ենթատողի առկայությունը մեկ այլ տողի մեջ: Օրինակ՝

```
str='University'
if 'i' in str:
    print('Your string contains the letter i')
```

Արդյունքը՝ Your string contains the letter i:

not in օպերատորը: not in օպերատորը ստուգում է որևէ ենթատողի՝ մեկ այլ տողում ընդգրկված չլինելը: Օրինակ՝

```
str='University'
if 'g' not in str:
   print('Your string does not contain the letter g')
```

Արդյունքը՝ Your string does not contain the letter g:

6.2. Տողի տարրերի ինդեքսավորումը

Տողի տարրերն ինդեքսավորվում են ամբողջ թվերով: Տողի որևէ տարրի դիմելու համար գրվում է տողի անունը, ապա, քառակուսի փակագծերի մեջ՝ այդ տարրի ինդեքսը:

Ինդեքսավորումը կարող է կատարվել առաջին տարրից սկսած՝ ձախից աջ ուղղությամբ: Այդ դեպքում որպես ինդեքսներ օգտագործվում են ոչբացասական ամբողջ թվերը՝ սկսած զրոյից: Ինդեքսավորումը կարող է կատարվել նաև աջից դեպի ձախ: Այդ դեպքում որպես ինդեքսներ օգտագործվում են բացասական ամբողջ թվերը:

Օրինակ՝ str='Python String'

սիմվոլ	P	y	t	h	o	n		S	t	r	i	n	g
ինդեքսները ձախից աջ	0	1	2	3	4	5	6	7	8	9	10	11	12
ինդեքսները աջից ձախ	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Օրինակներ.

```
str= 'Hello'
str[0]= 'H'  # տողի առաջին տարրը
str[1]= 'e'   # տողի երկրորդ տարրը
str[-1]= 'o'  # տողի վերջին տարրը
str[-2]= 'l'  # տողի նախավերջին տարրը
```

Եթե ծրագրում փորձենք դիմել գոյություն չունեցող ինդեքսով տարրի, ապա կառաջանա սխալ և կտրվի սխալի մասին հետևյալ հաղորդագրությունը՝ “index Error: string index is out of range”:

Օրինակ՝ վերևում բերված ծրագրում, եթե դիմենք str[8] տարրին, կստանանք այդպիսի հաղորդագրություն, քանի որ ութ ինդեքսով տարր այդ տողում գոյություն չունի:

Python–ը հնարավորություն է տալիս դիմելու տողի տարրերի խմբի՝ օգտագործելով հասցեավորման [:] օպերատորը, որի տեսքն է՝

տողի անուն [m : n : p],

որտեղ m-ը, p-ն և n-ը ամբողջ թվեր են: Այս օպերատորով ընտրվում են տողի m-րդ ինդեքսից մինչև n-ից փոքր ինդեքսով այն տարրերը, որոնք իրարից բաժանված են p քայլով: Եթե p քայլը բացակայում է, ապա լռելյայն ընդունվում է, որ այն հավասար է մեկի: Ընդ որում՝

- եթե ինդեքսում գրված է [:], ապա ընտրվում են այդ տողի բոլոր տարրերը,
- եթե ինդեքսում գրված է [m::p], ապա ընտրվում են այդ տողի m-րդ ինդեքսից սկսած բոլոր այն տարրերը, որոնք իրարից բաժանված են p քայլով,
- եթե ինդեքսում գրված է [m:], ապա ընտրվում են այդ տողի m-րդ ինդեքսից սկսած մինչև վերջ բոլոր տարրերը,
- բացասական քայլը շրջում է տողը (գրում է հակառակ հերթականությամբ):

Օրինակներ.

```
str='ABCDEFGHIJKLMN'
str[3:6]= 'DEF'  # 3-ից մինչև 5 ինդեքսն ընկած բոլոր տարրերը
str[:6]= 'ABCDEF'      #առաջին 6 տարրերը

str[6:]= 'GHIJKLMN'  #6-րդ ինդեքսից մինչև վերջ բոլոր տարրերը
str[-3:]= 'LMN'              #վերջին 3 տարրը
```

str[:]='ABCDEFGHIJKLMN' #ամբողջ տողը

str[1:10:2]= 'BDFHJ' #1-ից մինչև 9 ինդեքսն ընկած բոլոր տարրերը՝ 2 քայլով

str[: :-1]= 'NMLKJIHGFEDCBA' #ամբողջ տողը՝ գրված հակառակ կարգով

str[4:-2]= 'EFGHIJKL' #4-ից մինչև -3 ինդեքսն ընկած բոլոր տարրերը՝ 1 քայլով:

Ինչպես արդեն նշել ենք, տողը չփոփոխվող օբյեկտ է, այսինքն՝ տողերի հետ գործողությունների արդյունքում ստեղծվում են նոր տողեր:

Դիցուք, ցանկանում ենք տրված տողի որևէ ինդեքսով տարր փոխարինել այլ սիմվոլով: Դիտարկենք, թե ինչպես կարելի է կատարել դա:

Օրինակ՝ տրված str='abcdefghi' տողում անհրաժեշտ է հինգ ինդեքսով տարրը փոխարինել 'y' սիմվոլով: Այս փոխարինումը չենք կարող կատարել s[5]= 'y' հրամանով այն պատճառով, որ տողը չփոփոխվող օբյեկտ է: Դրա համար պետք է կառուցել նոր տող.

```
str1=str[:5]+ 'y'+str[6:]
```

Այս հրամանով առաջին հինգ տարրին (ինդեքսները՝ 0,1,2,3,4) կկցվի 'y' սիմվոլը (ինդեքսը՝ 5), ապա կկցվեն 6-րդ ինդեքսից սկսած մնացած բոլոր տարրերը:

Տրված str տողը կարելի է արտածել print() ֆունկցիայի միջոցով հետևյալ հրամանով.

```
for i in range(len(str)):
    print(str(i))
```

կամ՝

```
for c in str:
    print(c)
```

6.3. Տողերի ներկառուցված մեթոդներն ու ֆունկցիաները

Տողերի բոլոր մեթոդների ցուցակը կարելի է արտածել dir(str) հրամանով: Ցուցակում մեթոդների անունները սկսվում են '_' սիմվոլով: Որևէ մեթոդի վերաբերյալ տեղեկատվություն կարելի է ստանալ՝ դիմելով help օգնությանը: Օրինակ՝ count() մեթոդի վերաբերյալ տեղեկատվություն ստանալու համար պետք է հավաքել help(str.count):

Հետևյալ աղյուսակում ներկայացված են տողերի հիմնական ներկառուցված մեթոդներն ու ֆունկցիաները Python-ում:

Մեթոդը	Գործողությունը
capitalize()	Տողի առաջին սիմվոլը դարձնում է մեծատառ:
lower()	Տողի բոլոր մեծատառերը դարձնում է փոքրատառ:
upper()	Տողի բոլոր փոքրատառերը դարձնում է մեծատառ:
swapcase()	Տողի բոլոր մեծատառերը դարձնում է փոքրատառ, բոլոր փոքրատառերը՝ մեծատառ:
replace(x,y)	Տողում հանդիպող բոլոր x ենթատողերը փոխարինում է y տողով:
count(str, start=0, end=len(string))	Հաշվում է, թե քանի անգամ է str ենթատողը հանդիպում string տողում՝ սկսած start ինդեքսից մինչև (end-1) ինդեքսը: Եթե start և end ինդեքսները բաց են թողնված, ապա լռելյայն դիտարկվում է տողը սկզբից մինչև վերջ:
find(str, start=0, end=len(string))	Եթե str ենթատողը հանդիպում է string տողում՝ սկսած start ինդեքսից մինչև (end-1) ինդեքսը, ապա վերադարձնում է ենթատողի սկզբի ինդեքսը, հակառակ դեպքում՝ վերադարձնում է 1: Եթե start և end ինդեքսները բաց են թողնված, ապա լռելյայն դիտարկվում է տողը սկզբից մինչև վերջ:
replace(old_text, new_text [, max])	Տողում բոլոր old_text ենթատողերը փոխարինում է new_text տողով՝ max հատ (ոչ պարտադիր պարամետր է):

index(str, start=0, end=len(string))	Եթե str ենթատողը հանդիպում է string տողում՝ դիտարկված start ինդեքսից մինչև (end-1) ինդեքսը, ապա վերադարձնում է ենթատողի առաջին հանդիպման ինդեքսը, հակառակ դեպքում՝ գեներացնում է բացառություն: Եթե start և end ինդեքսները բաց են թողնված, ապա լռելյայն դիտարկվում է տողը սկզբից մինչև վերջ:
startswith(str, start=0,end=len(string))	Վերադարձնում է True, եթե string տողի start ինդեքսից մինչև (end-1) ինդեքսը վերցրած ենթատողը սկսվում է str-ով և False՝ հակառակ դեպքում: Եթե start և end ինդեքսները բաց են թողնված, ապա լռելյայն դիտարկվում է տողը սկզբից մինչև վերջ:
endswith(str, start=0, end=len(string))	Վերադարձնում է True, եթե string տողի start ինդեքսից մինչև (end-1) ինդեքսը վերցված ենթատողը ավարտվում է str-ով, False՝ հակառակ դեպքում: Եթե start և end ինդեքսները բաց են թողնված, ապա լռելյայն դիտարկվում է ամբողջ տողը՝ սկզբից մինչև վերջ:
expandtabs(tabsize=8)	Տողում հանդիպող tab-երը փոխարինում է tabsize-ի քանակով բացակներով (լռելյայն՝ 8 հատ):
isalpha()	Վերադարձնում է True, եթե տողի բոլոր սիմվոլները տառեր են, False՝ հակառակ դեպքում:
isdigit()	Վերադարձնում է True, եթե տողի բոլոր սիմվոլները թվանշաններ են (0-9) կամ Unicode-ի թվանշաններ, False՝ հակառակ դեպքում:
isalnum()	Վերադարձնում է True, եթե տողի բոլոր սիմվոլները տառեր կամ թվեր են, False՝ հակառակ դեպքում:
islower()	Վերադարձնում է True, եթե տողի բոլոր սիմվոլները փոքրատառ են, False՝հակառակ դեպքում:

isupper()	Վերադարձնում է True, եթե տողի բոլոր սիմվոլները մեծատառ են, False՝ հակառակ դեպքում:
isnumeric()	Վերադարձնում է True, եթե տողի բոլոր սիմվոլները թվանշաններ (0-9) են, վերին կամ ստորին ցուցիչներ, կոտորակներ և Unicode-ի այլ թվային նիշեր, False՝ հակառակ դեպքում:
isspace()	Վերադարձնում է True, եթե տողը պարունակում է միայն բացակներ, False՝ հակառակ դեպքում:
len(string)	Վերադարձնում է տողի երկարությունը (սիմվոլների քանակը):
max(str)	Վերադարձնում է ամենամեծ կոդն ունեցող սիմվոլը:
min(str)	Վերադարձնում է ամենափոքր կոդն ունեցող սիմվոլը:
sep.join(seq)	Տողային տարրերից կազմված seq հաջորդականության բոլոր տարրերը միավորում է մեկ տողի մեջ՝ օգտագործելով sep սիմվոլը որպես անջատիչ (լռելյայն որպես անջատիչ օգտագործվում է բացակը):
string.split(separator, maxsplit)	string տողը ձևափոխվում է ցուցակի՝ օգտագործելով separator սիմվոլը որպես անջատիչ տողի բառերն իրարից առանձնացնելու համար (լռելյայն որպես անջատիչ օգտագործվում է բացակը):

Աղյուսակ 6.1. Տողերի ներկառուցված մեթոդներն ու ֆունկցիաները

lower(), upper(), replace() մեթոդները չեն փոխում սկզբնական տողը: Այդ դեպքում տողը փոխելու համար պետք է գրել հետևյալ հրամանը.

```
s=s.lower()
```

Դիտարկենք տողերի ներկառուցված մեթոդների վերաբերյալ մի քանի օրինակներ: Դիցուք տրված է s տողը:

print(s.count(' ')) #հաշվում է s տողում բացակների քանակը:

s=s.upper() #s տողի բոլոր սիմվոլները դարձնում է մեծատառ:

s=s.replace('red','yellow') #s տողում հանդիպող բոլոր 'red' ենթատողերը փոխարինում է 'yellow' տողով:

print(s.index('a') # արտածում է տողում a տառի առաջին ինդեքսը:

Խնդիր: Գրել ծրագիր, որը ներածում է որևէ տող և ստուգում է, թե արդյոք այդ տողը սկսվո՞ւմ է տառով, թե՞ ոչ և տալիս է համապատասխան հաղորդագրություն:

```
str=input('Enter a text: ')
if str[0].isalpha():
   print('The first character is a letter')
else:
   print('The first character isn\'t a letter')
```

Խնդիր: Գրել ծրագիր, որը ստուգում է, թե մուտքագրված տողում բոլոր տարրերը տառե՞ր են, թե՞ ոչ և տալիս է համապատասխան հաղորդագրություն:

```
str=input('Enter a text: ')
if not str.isalpha():
   print('Your text contains not only letters, but also other symbols')
else:
   print('Your string contains only letters')
```

Խնդիր: Գրել ծրագիր, որն արտածում է մուտքագրված տողում 'a' սիմվոլի բոլոր ինդեքսները:

```
str=input('Enter a text')
for i in range(len(str)):
   if str[i]=='a':
      print(i)
```

Եթե, օրինակ, մուտքագրենք 'Yerevan State University' տողը, ապա ծրագրի կատարման արդյունքում կստանանք.

5
10

Խնդիր: Գրել ծրագիր, որը եռապատկում է մուտքագրված տողի բոլոր սիմվոլները:

```
str=input('Enter a text: ')
str1=''
for i in range(len(str)):
    str1+= str[i]*3
print(str1)
```

կամ՝

```
str=input('Enter a text: ')
str1=''
for c in str:
    str1+= c*3
print(str1)
```

Եթե մուտքագրենք 'Python' տողը, արդյունքում կունենանք՝

PPPyyyttthhhooonnn

Խնդիր: Գրել ծրագիր, որը նախ տպում է մուտքագրված տողի առաջին սիմվոլը, հետո՝ առաջին ու երկրորդ սիմվոլները, ապա առաջին, երկրորդ ու երրորդ սիմվոլները և այսպես շարունակ:

```
str=input('Enter a text: ')
str1=''
for i in range(len(str)):
    str1+= str[i]
    print(str1, end=' ')
```

կամ՝

```
str=input('Enter a text: ')
str1=''
```

```
for i in range(len(str)):
    print(str[:i+1], end=' ')
```

Եթե մուտքագրենք, օրինակ՝ 'Python' տողը, ապա արդյունքում կստանանք՝

```
P Py Pyt Pyth Pytho Python
```

Տողում չերևացող սիմվոլները (Escape Character) գրելու համար օգտագործվում է '\' սիմվոլը: Դրանցից են՝

\n	Newline
\t	Tab
\b	Backspace
\e	Escape

Որպեսզի հատուկ կոդերը երևան տողում, անհրաժեշտ է երկու անգամ օգտագործել '\' սիմվոլը, օրինակ՝ '\\t': Օրինակ՝ print('name\\tsurname') հրամանով կարտածվի՝

```
name\tsurname:
```

Դատարկ տող տպելու համար կարելի է օգտագործել print() հրամանը կամ նոր տողի անցնելու սիմվոլը՝ \n:

```
print()
print('name \n surname')
```

Արդյունքը՝

```
name
surname:
```

Տողում '\' սիմվոլը, ապաթարցը և չակերտը գրելու համար օգտագործվում է '\' սիմվոլը (համապատասխանաբար՝ \\, \' և \''): Եթե տողը վերցված է չակերտների մեջ, ապա տողում ապաթարցը գրվում է առանց '\' սիմվոլի: Կամ, եթե տողը վերցված է ապա-

թարցերի մեջ, ապա չակերտն է գրվում առանց '\' սիմվոլի։ Օրինակ՝ հետևյալ երկու տողերը նույնն են.

```
s='He\'s a teacher'
s="He's a teacher"
```

Խնդիր: Տրված s տողում երկու տաբուլյացիաները փոխարինել բացակով։

```
s="Yerevan\tstate\tuniversity"
print(s)
s=s.expandtabs(2)
print(s)
```

Ծրագիրը գործարկելիս նախ տպվում է տրված տողը, ապա՝ երկու տաբուլյացիաներից յուրաքանչյուրը բացակով փոխարինելուց հետո ստացված տողը.

```
Yerevan	state	university
Yerevan state university
```

6.4. f-string ֆորմատավորում

Տողերի ֆորմատավորումը օգնում է կառավարել տեղեկատվության ելքը հարմար և ընթեռնելի ձևով: Python-ը, իր ճկունության և հզոր ներկառուցված գործառույթների շնորհիվ, մշակողներին առաջարկում է մի քանի տարբերակ այս նպատակին հասնելու համար, որոնցից է f-string ֆորմատավորումը: Այն ներկայացվել է 2016 թվականին թողարկված Python 3.6 տարբերակում:

f-string տողերը ապահովում են Python արտահայտությունները տողերի մեջ ուղղակիորեն ներդնելու պարզ և ընթեռնելի միջոց, ինչը այն դարձնում է ֆորմատավորման նախընտրելի մեթոդ:

f-string տողերը սկսվում են 'f' կամ 'F' սիմվոլներից որևէ մեկով և պարունակում են արտահայտություններ՝ {} ձևավոր փակագծերի ներսում: Այս արտահայտությունները գնահատվում են կատարման ժամանակ և այնուհետև ֆորմատավորվում __format__ արձանագրության միջոցով: Օրինակ՝

```
print(f"The sum is {10 + 5}") # Արդյունքը: The sum is 15
```

Հետևյալ օրինակում f-string տողի ձևավոր փակագծերը պարունակում են student և age փոփոխականները: print հրամանի կատարման ժամանակ արտածվում են այդ փոփոխականների համապատասխան արժեքները:

```
student = "Karina"
score = 20
print(f"Hello, {student}! Your grade in mathematics is {score}.")
# Արդյունքը: Hello, Karina! Your grade in mathematics is 20.
```

Հետևյալ օրինակում Python-ը հաշվում է f-string տողի փակագծերում ներառված արտահայտությունների արժեքները՝ փոփոխականների տրված արժեքների համար և արտածում դրանք:

```
a = 520
r = 10
print(f" The {r} percentage of number {a} is equal to {a*r/100}.")
# Արդյունքը՝ The 10 percentage of number 520 is equal to 52.0.
```

Python 3.8-ից սկսած՝ f-string տողերը դարձել են ավելի օգտակար վրիպազերծման (debugging) համար՝ '=' սիմվոլի ավելացման շնորհիվ: Այս գործառույթը մեկ տողում ցուցադրում է ինչպես փոփոխականների անունները, այնպես էլ դրանց արժեքները: Օրինակ՝

```
a = 10
b = 20
print(f"{a=}, {b=}") # Արդյունքը՝ a=10, b=20
```

f-string տողերը կարող են աշխատել տարբեր տիպերի տվյալների հետ: Հետևյալ օրինակում մշակվում են ամբողջ և սահող կետով թվային տվյալներ:

```
int_number = 15
float_number = 18.6
print(f"Integer: {int_number}")
```

```
print(f"Float: {float_number}")
```

Արդյունքը՝

```
Integer: 15
Float: 18.6
```

f-string-ը աջակցում է նաև ամսաթվի և ժամանակի ֆորմատավորումը: Ֆորմատավորման կոդերն են.

- %B - ամսվա լրիվ անունը,
- %d – ամսվա օրը (երկնիշ),
- %Y – տարի (չորս նիշով):

Օրինակ՝

```
from datetime import datetime
today = datetime.today()
print(f"{today:%B %d, %Y}") #Արդյունքը՝ January 21, 2025
```

f-string տողերը աշխատում են ցուցակների հետ՝ թույլ տալով մուտք գործել առանձին տարրերի կամ ամբողջ ցուցակի մեջ (ցուցակների մասին տե՛ս *գլուխ 7*): Օրինակ՝

```
products = ["Refrigerator", "Iron", "Air Conditioner"]
print(f"First product: {products[0]}")
print(f"All products: {', '.join(products)}")
```

Արդյունքը՝

```
First product: Refrigerator
All products: Refrigerator, Iron, Air Conditioner
```

Բառարանների հետ աշխատելիս նույնպես կարելի է հեշտությամբ մուտք գործել և ֆորմատավորել բառարանի արժեքները (բառարանների մասին տե՛ս *գլուխ 10*), ինչպես ցույց է տրված ստորև ծրագրում.

```
employee_info = {
    "name": "Arman Grigoryan",
    "age": 28,
```

```
    "position": "manager"
}
print(f"employee: {employee_info['name']} is {employee_info['position']}")
```

Արդյունքը՝

```
employee: Arman Grigoryan is manager
```

f-string-ը թույլ է տալիս կատարել թվաբանական գործողություններ, օգտագործել պայմանական օպերատորներ, Python ներկառուցված ֆունկցիաներ: Օրինակ՝

```
number1 = 132
number2 = 8
result = number1>number2
print(f"{'number1 > number2' if result else 'number1 <= number2'}")
```

Արդյունքը՝ number1 > number2

Հետևյալ օրինակում f-string-ում օգտագործվում են ցուցակների որոշ ներկառուցված ֆունկցիաներ:

```
lst = [7, 2, 3, -8, 5]
print(f"List length: {len(lst)}")
print(f"Maximum value: {max(lst)}")
print(f"Sum: {sum (lst)}")
```

Արդյունքը՝

```
List length: 5
Maximum value: 7
Sum: 9
```

f-string տողերում կարելի է ֆորմատավորել ինչպես թվային, այնպես էլ տողային տվյալները: Տասնորդական թվերը ֆորմատավորելու համար f-տողերն օգտագործում են երկու կետ, որին հաջորդում է կետ և տասնորդական նիշերի քանակը: Օրինակ՝

```
e = 2.71828182
num1 = 29.99999
percentage= 0.9575

print(f"e to 1 decimal places: {e:.1f}")
print(f"e to 2 decimal places: {e:.2f}")
print(f"e to 3 decimal places: {e:.3f}")
print(f"num1 rounded to 2 decimals: {num1:.2f}")
print(f"Percentage with 1 decimal: {percentage:.1%}")
```

Արդյունքը՝

```
e to 1 decimal places: 2.7
e to 2 decimal places: 2.72
e to 3 decimal places: 2.718
num1 rounded to 2 decimals: 30.00
Percentage with 1 decimal: 95.8%
```

Տեքստերի ֆորմատավորման ժամանակ կարելի է տալ տեքստի երկարությունը, կարելի է տեքստը հավասարեցնել ձախ, աջ կողմում կամ կենտրոնում՝ համապատասխանաբար օգտագործելով '<', '>' և '^' օպերատորները։

- <N: Ձախ կողմում հավասարեցնում է տողը և աջ կողմում այն լրացնում է բացակներով մինչև N երկարություն։
- >N: Աջ կողմում հավասարեցնում է տողը և ձախ կողմում այն լրացնում է բացակներով մինչև N երկարություն։
- ^N: Հավասարեցնում է տողը կենտրոնում, ինչպես նաև ձախ ու աջ կողմերում լրացնում է բացակներով մինչև N երկարություն։
- .N: Նշում է տողի ճշգրիտ երկարությունը։ Եթե այն պարունակում է N-ից ավելի նիշ, այն կտրվում է։

Հետևյալ ծրագրում ավելացվում են զրոներ սկզբնական տողի ձախ մասում՝ աջ կողմում հավասարեցման համար, աջ կողմում՝ ձախ կողմում հավասարեցման համար, կամ ձախ և աջ կողմերում՝ կենտրոնում հավասարեցման համար՝ մինչև այն հասնի 9 նիշ երկարության։

```
print(f"{123:0>9}") # Արդյունքը: 000000123
print(f"{123:0<9}") # Արդյունքը: 123000000
print(f"{123:0^9}") # Արդյունքը: 000123000
```

Օրինակ՝ Հետևյալ ծրագրում նկարվում է հավասարասրուն եռանկյուն։

```
print(f"{'*':^7}")
print(f"{'***':^7}")
print(f"{'*****':^7}")
print(f"{'*******':^7}")
```

Արդյունքը՝

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

Աղյուսակային տվյալների համար, օգտագործելով տեքստի երկարությունը հատուկ նիշերով լրացնելու միջոցով տողը ըստ ձախ, աջ կողմերի կամ ըստ կենտրոնի հավասարեցնելու հնարավորությունը, կարելի է ստեղծել պրոֆեսիոնալ տեսք ունեցող տվյալների աղյուսակներ: Օրինակ՝

```
products = ["TV", "Monitor", "Refrigerator"]
prices = [520, 290, 670]
print("Product Prices")
print("-" * 21)
for product, price in zip(products, prices):
    print(f"{product:<15} | {price:>5}")
```

Արդյունքը՝

```
Product Prices
---------------------
TV              |   520
Monitor         |   290
Refrigerator    |   670
```

Չևավոր փակագիծ արտածելու համար F-string տողում պետք է օգտագործել կրկնակի ձևավոր փակագիծ՝ {{}}: Օրինակ՝

```
print(f"{{Hello, World!}}")
print(f"{{{{Hello, World!}}}}")
```

Արդյունքը՝

```
{Hello, World!}
{{Hello, World!}}
```

Այսպիսով, f-string ֆորմատավորումը իր ճկունությամբ ապահովում է տեղեկատվության ներկայացման հասկանալի եղանակ, ինչը հատկապես կարևորվում է տվյալները օգտատերերին ցուցադրելիս կամ ֆայլերում դրանք գրելիս: f-string տողերի մասին ավելի մանրամասն տեղեկատվություն կարելի է ստանալ հետևյալ հղումով՝ https://www.datacamp.com/tutorial/python-f-string:

ԳԼՈՒԽ 7. ՑՈՒՑԱԿՆԵՐ

7.1. Ցուցակների սահմանումը և հիմնական գործողությունները

Ցուցակը (List) պատկանում է Python-ի ներկառուցված հաջորդականությունների թվին: Ցուցակը տարրերի համախմբություն է, որում յուրաքանչյուր տարրի ամրակցված է ինդեքս (ինդեքսներ): Ցուցակի անդամները գրվում են քառակուսի փակագծերի մեջ և իրարից անջատվում են ստորակետով: Օրինակ՝ L=[1, 2, 3, 4, 5]:

Ցուցակը կարող է լինել դատարկ: Դատարկ է այն ցուցակը, որը ոչ մի տարր չի պարունակում: Դատարկ ցուցակի գրությունն է՝ []:

Ցուցակը կարող է գրվել մի քանի տողի վրա: Օրինակ՝

```
L=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15]:
```

Ցուցակը կարող է պարունակել տարբեր տիպի տվյալներ, այդ թվում՝ նաև ցուցակ: Օրինակ, հետևյալ L ցուցակը պարունակում է և՛ թվային ու տողային տվյալներ, և՛ ցուցակ:

```
L=[1, 5, 3.14, 'ABC', 'G34', [21, 65, 0]]:
```

Ցուցակի տարրերը կարելի է ներածել input() ֆունկցիայի միջոցով: Տարրերը մուտքագրվում են քառակուսի փակագծերի մեջ՝ իրարից անջատելով ստորակետով:

```
L=eval(input('Enter a list: '))
print(L)
print('The first element is', L[0]) #L[0]-ցուցակի առաջին տարրը
```

Ծրագրի կատարման արդյունքը ներկայացված է ստորև.

```
Enter a list: [11,22,3,5,8,31]
[11, 22, 3, 5, 8, 31]
The first element is 11
```

Ինչպես տողի, այնպես էլ ցուցակի տարրերը ինդեքսավորվում են՝ սկսած զրոյից: Ցուցակի որևէ տարրի դիմելու համար գրվում է ցուցակի անունը, ապա քառակուսի փակագծերի մեջ՝ այդ տարրի ինդեքսը: Հետևաբար, վերևի օրինակում L[0]-ն ցուցակի առաջին անդամն է:

Ցուցակների համար սահմանված են in և not in, կցում (+), կրկնում (*) գործողությունները:

Կցում գործողությամբ ցուցակները միավորվում են մեկ ցուցակի մեջ: Երկու ցուցակների միավորման ժամանակ երկրորդ ցուցակը կցվում է առաջին ցուցակի վերջին: Կցման արդյունքում ստացվում է ցուցակ, որի տարրերի քանակը հավասար է կցվող ցուցակների տարրերի քանակների գումարին:

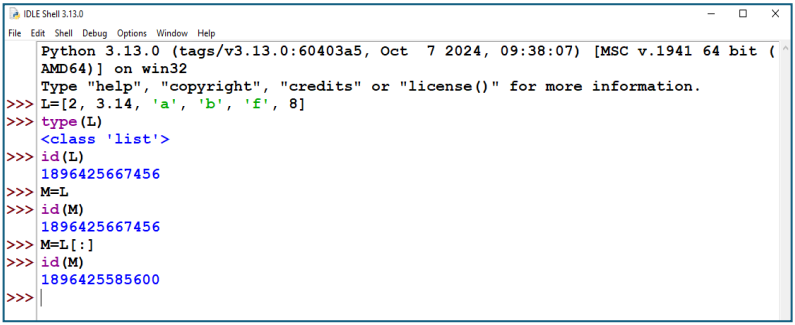
Կրկնում գործողությամբ ցուցակը կրկնվում է նշված քանակով: Ստորև բերված են մի քանի օրինակներ կցում և կրկնում գործողությունների վերաբերյալ.

```
[5,6]+[77,8,9,10]= [5, 6, 77, 8, 9, 10]
[2,3.14,'a', 'b']+['f',8]= [2, 3.14, 'a', 'b', 'f', 8]
['ab']*3=['ab', 'ab', 'ab']
[0]*5=[0, 0, 0, 0, 0]
```

in գործողությունը ստուգում է նշված տարրի պատկանելիությունը ցուցակին, իսկ not in գործողությունը՝ ոչ պատկանելիությունը: Հետևյալ ծրագրում ստուգվում է, թե արդյոք L ցուցակը պարունակո՞ւմ է 'b' սիմվոլը, թե՞ ոչ, և այդ մասին տրվում է համապատասխան հաղորդագրություն:

```
L=[2, 3.14, 'a', 'b', 'f', 8]
if 'b' in L:
   print("The list L contains'b'")
else:
   print("The list L does not contain'b'")
```

Եթե որևէ L ցուցակ վերագրենք մեկ այլ ցուցակի, օրինակ՝ M=L, ապա դրանով L ցուցակը չի կրկնօրինակվի, այլ M և L փոփոխականները կհղվեն նույն օբյեկտի վրա: L ցուցակը կարելի է կրկնօրինակել M=L[:] հրամանով (նկար 7.1):



Նկար 7.1. Ցուցակի կրկնօրինակումը

Python-ը հնարավորություն է տալիս տողերի նմանությամբ ցուցակի տարրերի խմբի հետ կատարել գործողություններ՝ օգտվելով հասցեավորման [:] օպերատորից (տե՛ս *պարագրաֆ 6.2*)։

Հետևյալ աղյուսակում L=['AB','CD','EF','GH','IJ','KL','MN'] ցուցակի համար կատարված են հասցեավորման գործողություններ։

Հասցեավորման գործողությունը	Արդյունքը
print(L[3:6])	['GH', 'IJ', 'KL']
print(L[:6])	['AB', 'CD', 'EF', 'GH', 'IJ', 'KL']
print(L[6:])	['MN']
print(L[-3:])	['IJ', 'KL', 'MN']
print(L[:])	['AB', 'CD', 'EF', 'GH', 'IJ', 'KL', 'MN']
print(L[1:6:2])	['CD', 'GH', 'KL']
print(L[::-1])	['MN', 'KL', 'IJ', 'GH', 'EF', 'CD', 'AB']
print(L[2:-2])	['EF', 'GH', 'IJ']

7.2. Ցուցակների ներկառուցված ֆունկցիաներն ու մեթոդները

Աղյուսակ 7.1-ում բերված են ցուցակների հիմնական ներկառուցված ֆունկցիաները Python-ում:

Ֆունկցիա	Գործողությունը
len(list)	Վերադարձնում է list ցուցակի երկարությունը:
sum(list)	Վերադարձնում է list ցուցակի տարրերի գումարը:
min(list)	Վերադարձնում է list ցուցակի փոքրագույն տարրը:
max(list)	Վերադարձնում է list ցուցակի առավելագույն տարրը:
sorted(list)	Տրված list ցուցակից ստեղծում է նոր, աճման կարգով դասավորված ցուցակ:
del list[index]	Տրված list ցուցակից հեռացնում է index կարգահամարով տարրը:

Աղյուսակ 7.1. Ցուցակների ներկառուցված ֆունկցիաները Python-ում

Դիտարկենք հետևյալ օրինակը ցուցակների ներկառուցված ֆունկցիաների կիրառմամբ:

```
L=[2,5,7,1,4,8,1,9,4]
print('sum=', sum(L))
print('len=', len(L))
print('max=', max(L))
print('min=', min(L))
print('The sorted list:', sorted(L))
print('The original list:', L)
```

Արդյունքը՝

```
sum= 41  #ցուցակի տարրերի գումարը
len= 9     #ցուցակի երկարությունը
max= 9    #ցուցակի առավելագույն տարրը
min= 1    #ցուցակի փոքրագույն տարրը
```

The sorted list: [1, 1, 2, 4, 4, 5, 7, 8, 9] # նոր, աճման կարգով դասավորված ցուցակը

The original list: [2, 5, 7, 1, 4, 8, 1, 9, 4] #սկզբնական L ցուցակը

Ստացված արդյունքից երևում է, որ sorted(L) ֆունկցիայով ստեղծվել է նոր ցուցակ, իսկ սկզբնական ցուցակը մնացել է անփոփոխ:

Ցուցակը հնարավոր է կարգավորել նաև ըստ որևէ բանալու, օրինակ՝ ըստ տարրերի երկարության: Ցուցակը տարրերի երկարության աճման կարգով դասավորելու համար sorted(list, key, reverse) ֆունկցիայի key օպցիոնալ պարամետրին (բանալի) տրվում է len արժեքը՝ key=len: Նույն ցուցակը տարրերի երկարության նվազման կարգով դասավորելու համար նաև պետք է reverse օպցիոնալ պարամետրին տալ True արժեքը՝ reverse=True:

```
L = ['abc', 'ababh', 'dajg', 'b1']
print(sorted(L, key=len))
print(sorted(L, key=len, reverse=True))
```

Պատասխան՝

['b1', 'abc', 'dajg', 'ababh'] #L ցուցակը դասավորվել է ըստ տարրերի երկարության աճման կարգի:

['ababh', 'dajg', 'abc', 'b1'] #L ցուցակը դասավորվել է ըստ տարրերի երկարության նվազման կարգի:

Ցուցակների մեթոդները: Ցուցակների բոլոր մեթոդները կարելի է արտածել dir(list) հրահանգով: Արտածված ցուցակում մեթոդների անունները սկսվում են '__' սիմվոլով: Որևէ մեթոդի վերաբերյալ տեղեկատվություն կարելի է ստանալ՝ դիմելով help օգնությանը: Օրինակ՝ append() մեթոդի վերաբերյալ տեղեկատվություն ստանալու համար պետք է հավաքել help(list.append):

```
>>>dir(list)
['__add__', '__class__', '__class_getitem__', '__contains__',
'__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
```

'__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

Նկարագրենք ցուցակների մեթոդներից մի քանիսը:

Մեթոդը	Գործողությունը
append(x)	Ցուցակի վերջում ավելացվում է x տարրը:
extend(x)	x ցուցակի արժեքները կցում է առկա ցուցակին:
sort()	Կարգավորում է ցուցակը:
count(x)	Հաշվում է ցուցակում x տարրի կրկնությունների քանակը:
index(x)	Վերադարձնում է x տարրի առաջին հանդիպման ինդեքսը:
reverse()	Շրջում է ցուցակը (դասավորում է հակառակ հերթականությամբ):
remove(x)	Ցուցակից հեռացնում է առաջին հանդիպած x տարրը:
pop(p)	Հեռացնում է p ինդեքսով տարրը ցուցակից և վերադարձնում է դրա արժեքը:
insert(p,x)	Ցուցակում որպես p ինդեքսով տարր ներածում է x-ը:

Ցուցակի և տողի մեթոդների միջև կա տարբերություն: Տողի մեթոդները չեն փոխում սկզբնական տողը, իսկ ցուցակի մեթոդները փոխում են:

Այժմ բերենք մի քանի խնդիրներ՝ ցուցակի մեթոդների կիրառությունների վերաբերյալ:

Խնդիր: Դիցուք տրված է L=[5, 6, 7, 8, 9, 10] ցուցակը:

ա) Ցուցակի երկրորդ տարրը դարձնել 60:

L[1]=60

Արդյունքը՝

L=[5, 60, 7, 8, 9, 10]

բ) Ստացված նոր ցուցակում ներածել 1 ինդեքսով տարր՝ 100 արժեքով:

L.insert(1,100)

Արդյունքը՝

L=[5, 100, 60, 7, 8, 9, 10]

գ) Հեռացնել բ) կետում ստացված ցուցակի երրորդ անդամը.

```
del L[2]
```

Արդյունքը՝

L=[5, 100, 7, 8, 9, 10]

դ) Հեռացնել գ) կետում ստացված ցուցակի առաջին երեք տարրը.

```
del L[:3]
```

Արդյունքը՝

L=[8, 9, 10]

ե) Տրված L=[5, 6, 7, 8, 9, 10] ցուցակին կցել [11,12,13] ցուցակը։

```
L=[5, 6, 7, 8, 9, 10]
L.extend([11,12,13])
print(L) #Արդյունքը՝ [5, 6, 7, 8, 9, 10, 11, 12, 13]
```

Խնդիր։ Գրել ծրագիր, որը գեներացնում է ցուցակ, որի տարրերը [1,1000] միջակայքին պատկանող հարյուր պատահական ամբողջ թվեր են, որից հետո արտածում է այդ ցուցակը:

```
from random import randint
L=[]
for i in range(100):
   L=L+[randint(1,100)]
print(L)
```

Կամ այլ եղանակով.

```
from random import randint
L=[]
for i in range(100):
   L.append(randint(1,1000))
print(L)
```

Խնդիր։ Գրել ծրագիր, որը գեներացնում է ցուցակ, որի տարրերը [1,50] միջակայքից տասը պատահական ամբողջ թվեր են, ապա յուրաքանչյուր տարր փոխարինում է իր քառակուսիով և արտածում ստացված ցուցակը:

```
from random import randint
L=[]
for i in range(10):
   L.append(randint(1,50))
   L[i]=L[i]**2
print(L)
```

Խնդիր: Հաշվել տրված ցուցակի 10-ից մեծ տարրերի քանակը:

```
L=eval(input('Enter a list: '))
count=0
for i in range(len(L)):
   if L[i]>10:
      count+=1
print(count)
```

Ծրագիրը կարող ենք գրել նաև հետևյալ եղանակով.

```
L=eval(input('Enter a list: '))
count=0
for c in L:
   if c>10:
      count+=1
print(count)
```

Խնդիր: Գեներացնել L ցուցակ, որը պարունակում է [1, 100] հատվածից հիսուն պատահական ամբողջ թիվ: Կազմել նոր ցուցակ, որի i-րդ անդամը ցույց է տալիս i թվի կրկնությունների քանակը L ցուցակում (առաջին թիվը՝ մեկի կրկնությունների, երկրորդ անդամը՝ երկուսի, …, 50-րդ անդամը՝ 50-ի):

```
from random import randint
L=[]
for i in range(50):
   L.append(randint(1,101))
print(L)
```

```
L1=[]
for i in range(len(L)):
   L1.append(L.count(i+1))
print(L1)
```

Ծրագրի աշխատանքի ժամանակ գեներացվել է հետևյալ ցուցակը՝

[47, 1, 14, 31, 11, 70, 78, 73, 63, 13, 11, 35, 33, 62, 61, 26, 24, 90, 50, 25, 96, 30, 13, 38, 47, 43, 64, 74, 26, 69, 95, 1, 49, 78, 41, 94, 8, 97, 58, 6, 28, 78, 13, 42, 29, 98, 75, 21, 84, 96]

Ստացվել է հետևյալ արդյունքը՝

[2, 0, 0, 0, 0, 1, 0, 1, 0, 0, 2, 0, 3, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 2, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 2, 0, 1, 1]

Խնդիր: Արտածել տրված ցուցակի առաջին երկու ամենափոքր և վերջին երկու ամենամեծ տարրերը:

```
L=eval(input('Enter a list: '))
L.sort()
print('The smallest two elements are: ', L[:2])
print('The largest two elements are: ', L[-2:])
```

Խնդիր: Գտնել և արտածել $x_1, x_2, \ldots x_n$ իրական թվերի հաջորդականության փոքրագույն տարրն ու դրա կարգահամարը:

Առաջարկվում է հետևյալ ալգորիթմը: Իրական թվերի հաջորդականությունը տրվում է X ցուցակի միջոցով: Նախ min փոփոխականին վերագրվում է ցուցակի առաջին տարրը, և դրա կարգահամարը հիշվում է index փոփոխականի մեջ: Այնուհետև ցուցակի մնացած բոլոր տարրերը հաջորդաբար համեմատվում են min փոփոխականի ընթացիկ արժեքի հետ: Եթե համեմատվող հերթական տարրը փոքր է min-ից, ապա min-ին վերագրվում է այդ տարրը, իսկ index-ին՝ դրա կարգահամարը:

```
X=eval(input('Enter a list: '))
print(X)
min=X[0]; index=0
```

```
for i in range(1,len(X)):
    if X[i]<min:
        min=X[i]; index=i
print('min=', min, 'index=', index)
```

Խնդիր: Դասավորել $x_1, x_2, \ldots x_n$ իրական թվերի հաջորդականությունը աճման կարգով:

Իրական թվերի հաջորդականությունը աճման կարգով դասավորելու համար առաջարկվում է հետևյալ ալգորիթմը: Նախ փնտրվում է հաջորդականության փոքրագույն տարրը: Գտնված փոքրագույն տարրն ու առաջին տարրը փոխանակվում են իրենց տեղերով: Այնուհետև, երկրորդ քայլում՝ սկսած երկրորդ տարրից, փնտրվում է հաջորդականության փոքրագույն տարրը: Գտնված փոքրագույն տարրն ու երկրորդ տարրը փոխանակվում են իրենց տեղերով և այսպես շարունակ: Գործողությունը կատարվում է $(n-1)$ անգամ, որից հետո n-րդ տեղում մնում է հաջորդականության ամենամեծ տարրը:

```
X=eval(input('Enter a list: '))
print(X)
for k in range(len(X)):
    min=X[k]; index=k
    for i in range(k+1, len(X)):
        if X[i]<min:
            min=X[i]; index=i
    X[index],X[k]=X[k],X[index]
print(X)
```

Python-ում կան ցուցակների հետ աշխատող ֆունկցիաներ, որոնք գտնվում են random մոդուլում:Ներկայացնենք դրանցից մի քանիսը:

choice(L)	L ցուցակից պատահականորեն ընտրում է որևէ տարր:
sample(L,n)	L ցուցակից պատահականորեն ընտրում է n հատ տարր:
shuffle(L)	Խառնում է L ցուցակի տարրերի դասավորվածությունը:

Խնդիր: Գույների տրված ցուցակից պատահականորեն ընտրել որևէ գույն:

```
from random import choice
colors=['red','green','blue', 'grey','yellow']
color=choice(colors)
print(color)
```

Խնդիր: Գույների տրված ցուցակից պատահականորեն ընտրել երկու գույն:

```
from random import sample
colors=['red','green','blue', 'grey','yellow']
color2=sample(colors,2)
print(color2)
```

Խնդիր: Տրված է մրցույթի ինը մասնակիցների ցուցակ: Մրցույթից առաջ ցուցակը խառնվում է, և մասնակիցները ելույթ են ունենում ըստ այդ նոր ցուցակի: Գրել ծրագիր, որը կկազմի մասնակիցների՝ պատահական հերթականությամբ ելույթ ունենալու ցուցակը:

```
from random import shuffle
players=['A','B','C','D', 'E','F','G','H','I']
print('The original list of players:', players)
shuffle(players)
print('The new list of players:', players) #տպում է նոր ցուցակը
```

Արդյունքը՝

```
The original list of players: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
The new list of players: ['I', 'D', 'F', 'A', 'C', 'B', 'H', 'E', 'G']
```

Խնդիր: Մրցույթի տասը մասնակիցների ցուցակը պատահականորեն տրոհել երկու թիմի:

```
from random import shuffle
players=['A','B','C','D', 'E','F','G','H','I','J']
```

```
shuffle(players) # խառնվում է մասնակիցների ցուցակը
print(players)
team1=[]
team2=[]
for i in range(0, len(players), 2):
   team1.append(players[i])
   team2.append(players[i+1])
print('team1:',team1)
print('team2:',team2)
```

Արդյունքը՝

```
['J', 'G', 'F', 'D', 'C', 'H', 'B', 'A', 'E', 'I']
team1: ['J', 'F', 'C', 'B', 'E']
team2: ['G', 'D', 'H', 'A', 'I']
```

Ցուցակների և տողերի հետ աշխատելիս կարևոր են նաև տողերի split() և join() մեթոդները (տե՛ս *պարագրաֆ 6.3*):

split() մեթոդը տողը ձևափոխում է ցուցակի: Մեթոդն ունի արգումենտ, որը ցույց է տալիս, թե ո՞ր սիմվոլն է օգտագործվելու տողի բառերն իրարից առանձնացնելու համար: Որպես անջատիչի լռելյայն արժեք օգտագործվում է բացակը:

Օրինակ՝

```
str='This is a string'
L=str.split()
print(L)
```

Արդյունքում տրված str տողը կձևափոխվի հետևյալ L ցուցակին.

```
['This', 'is', 'a', 'string']
```

Հետևյալ ծրագրում տողը ցուցակի ձևափոխելիս տողի սիմվոլներն իրարից առանձնացնելու համար որպես անջատիչ օգտագործվում է '-' սիմվոլը:

```
str='091-345-678'
```

```
L=str.split('-')
print(L)
```

Արդյունքը՝ ['091', '345', '677']

punctuation փոփոխականը: Տողում գտնվող բոլոր կետադրական նշանները, այդ թվում՝ նաև բացակային նշանները, համարվում են տողի մաս: string մոդուլում գտնվող punctuation փոփոխականի արժեքը հատուկ սիմվոլներ պարունակող հետևյալ տողն է՝ '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

punctuation փոփոխականի արժեքը կարելի է արտածել՝ ներմուծելով այն string մոդուլից՝

```
from string import punctuation
print(punctuation)
```

Խնդիր: Գրել ծրագիր, որը մուտքագրված տողից հեռացնում է այն սիմվոլները, որոնք պարունակվում են punctuation փոփոխականում:

```
from string import punctuation
str='This .is a ho$use. Th(is is a ni&ce ^hou*se'
for p in punctuation:
   str=str.replace(p,'')
print(str))
```

Արդյունքում կստանանք հետևյալ տողը.

This is a house This is a nice house

Խնդիր: Ստեղնաշարից մուտքագրել որևէ տող և բառ: Հաշվել տողում բառի կրկնությունների քանակը:

```
from string import punctuation
str=input('Enter a string: ')
word=input('Enter a word: ')
for p in punctuation:
   str=str.replace(p,' ') #հեռացվում են հատուկ սիմվոլները
```

```
str=str.lower()
L=str.split()
print(word, 'appears', L.count(word), 'times')
```

Դիցուք, մուտքագրել ենք հետևյալ տողը.

'This?is.a$house.This is%a&nice ^house'

Մուտքագրված տողում փնտրենք 'house' բառը և արտածենք դրա կրկնությունների քանակը: Ծրագրի կատարման արդյունքն է.

```
Enter a string: This?is.a$house.This is%a&nice ^house
Enter a word: house
house appears 2 times
```

join() մեթոդը տողային տարրերից կազմված ցուցակի տարրերը միավորում է մեկ տողի մեջ: Մեթոդն ունի արգումենտ, որը ցույց է տալիս, թե ո՞ր սիմվոլն է օգտագործվելու որպես անջատիչ՝ ցուցակի տարրերը տողում միավորելու համար: Հետևյալ օրինակում որպես անջատիչ օգտագործվել են տարբեր սիմվոլներ:

```
L=['The', 'join', 'method', 'takes', 'a', 'list',
'of', 'strings', 'and', 'joins', 'them', 'together', 'into', 'a', 'single',
'string.']
print(' '.join(L))
print(''.join(L))
print(','.join(L))
print('***'.join(L))
```

Պատասխան՝

The join method takes a list of strings and joins them together into a single string.

Thejoinmethodtakesalistofstringsandjoinsthemtogetherintoasingle string.

The,join,method,takes,a,list,of,strings,and,joins,them,together,int o,a,single,string.

The***join***method***takes***a***list***of***strings***and***joins***them***together***into***a***single***string.

Ներկայացված ծրագրում L ցուցակի տարրերը join() մեթոդով միավորվել են մեկ տողի մեջ: Առաջին հրամանով ստացված տողում տարրերն իրարից անջատվել են բացակով, երկրորդում՝ կցվել են իրար, երրորդում՝ անջատվել են ստորակետով, իսկ չորրորդ հրամանով՝ անջատվել են երեք աստղանիշով:

Խնդիր: Հետևյալ օրինակում ներմուծվում է որևէ տող, ապա list() ֆունկցիայով այդ տողը ձևափոխվում է սիմվոլների ցուցակի, որից հետո shuffle() ֆունկցիայով ցուցակի տարրերը պատահականորեն խառնվում են, ապա՝ join մեթոդով միավորվում մեկ տողի մեջ:

```
from random import shuffle
str=input('Enter a string: ')
L=list(str)
shuffle(L)
new_str=''.join(L)
print(new_str)
```

Ծրագիրն աշխատեցնենք մի քանի անգամ՝ յուրաքանչյուր դեպքում մուտքագրելով 'house' տողը: Առաջին անգամ աշխատեցնելիս պատուհանում արտածվել է sehou տողը, երկրորդ անգամ՝ hoseu, երրորդ անգամ՝ useoh: Դա տեղի է ունենում այն պատճառով, որ shuffle() ֆունկցիան ամեն անգամ պատահականորեն է խառնում ցուցակի տարրերը:

```
>>>
========================= RESTART: D:\Lists_orinak_5.py =========================
Enter a word: house
sehou
>>>
========================= RESTART: D:\Lists_orinak_5.py =========================
Enter a word: house
hoseu
>>>
========================= RESTART: D:\Lists_orinak_5.py =========================
Enter a word: house
useoh
>>>
```

Այժմ դիտարկենք հետևյալ ծրագիրը, որում բերվում են ցուցակի տարրերի հետ աշխատելու այլ գրելաձևեր՝ օգտագործելով range() ֆունկցիան:

```
str='Exercise'
L=[5, 1.8, 8, 2.5,6, 13]
M=['January', 'February', 'March', 'April', 'May', 'June']
print([0 for i in range(10)])
print([i**3 for i in range(1,5)])
print([i**2 for i in L])
print([s*2 for s in str])
print([m[0] for m in M])
print([i for i in L if i<8])
print([m[0] for m in M if len(m)<6])
```

Ծրագրի կատարման արդյունքն է.

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 8, 27, 64]
[25, 3.24, 64, 6.25, 36, 169]
['EE', 'xx', 'ee', 'rr', 'cc', 'ii', 'ss', 'ee']
['J', 'F', 'M', 'A', 'M', 'J']
[5, 1.8, 2.3, 6]
['M', 'A', 'M', 'J']
```

Այժմ, օգտագործելով ցուցակների հետ աշխատելու համառոտ գրելաձևերը, լուծենք մի քանի խնդիր:

Խնդիր: Գրել ծրագիր, որը գեներացնում և արտածում է [1,100] միջակայքին պատկանող պատահական հիսուն թվից կազմված ցուցակ:

```
from random import randint
L=[randint(1,101) for i in range(50)]
print(L)
```

Խնդիր: Տրված L ցուցակի յուրաքանչյուր տարր փոխարինել իր քառակուսիով:

```
L=[i**2 for i in L]
```

Խնդիր: Հաշվել տրված L ցուցակի 100-ից փոքր տարրերի քանակը:

```
print(len([i for i in L if i>100]))
```

Խնդիր: Գրել ծրագիր, որը գեներացնում և արտածում է [1,100] միջակայքին պատկանող պատահական հարյուր թվից կազմված ցուցակ: Այնուհետև կազմել նոր ցուցակ, որի առաջին անդամը հավասար է տրված ցուցակում մեկերի քանակին, երկրորդ անդամը՝ երկուսների քանակին և այլն:

```
from random import randint
L=[randint(1,101) for i in range(100)]
print(L)
L1=[L.count(i) for i in range(1,101)]
print(L1)
```

7.3. Երկչափ ցուցակներ (Two-dimensional lists): Ներդրված ցիկլեր

Python լեզվում երկչափ ցուցակի (Two-dimensional list) տարրերը ևս ցուցակներ են: Այն սովորաբար օգտագործվում է տվյալների աղյուսակով՝ տողերով և սյուներով, պահելու համար: Երկչափ ցուցակի յուրաքանչյուր տարրի ամրակցված է երկու ինդեքս:

Դիտարկենք հետևյալ 3 × 3 չափի L ցուցակը: Այն կազմված է երեք ցուցակից, որոնցից յուրաքանչյուրն ունի երեք տարր:

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

Երկչափ L ցուցակի որևէ տարրի կարելի է դիմել երկու ինդեքսների միջոցով՝ L[i][j]: Օրինակ՝ L[0][0]=1, L[2][1]=8:

Երկչափ ցուցակի ստեղծումը: Python լեզվում երկչափ ցուցակ կարելի է ստեղծել ներդրված ցիկլերի միջոցով: Օրինակ՝

```
L=[[i,j] for i in range(3) for j in range(2)]
print(L)
```

Բերված ծրագրում for հրամանով i փոփոխականը նախ ստանում է 0 արժեքը, որի համար j փոփոխականը ընդունում է 0 և 1 արժեքները: Ապա i փոփոխականը ստանում է 1 արժեքը, որի համար j-ն կրկին ընդունում է 0 և 1 արժեքները: Եվ վերջապես, i փոփոխականը ստանում է 2 արժեքը, որի համար j-ն ընդունում է 0 և 1 արժեքները: Արդյունքում ստեղծվում է հետևյալ ցուցակը՝

[[0, 0], [0, 1], [1, 0], [1, 1], [2, 0], [2, 1]]:

Նույն ցուցակը կարելի է ստեղծել նաև հետևյալ ծրագրով.

```
L=[]
for i in range(3):
    for j in range(2):
        L.append([i,j])
print(L)
```

Հետևյալ ծրագրով ստեղծվում է $n \times m$ չափի ցուցակ, որը պարունակում է միայն զրոներ: Այն բաղկացած է m հատ զրո պարունակող n ցուցակից:

```
n = int(input("Enter number of rows: "))
m = int(input("Enter number of columns: "))
A = []
A = [[0]*m]*n
print(A)
```

Ծրագրում նախ ստեղծվում է m հատ զրոներից կազմված ցուցակ՝ կրկնում '*' գործողության միջոցով, ապա ստացված ցուցակը կրկնվում է n անգամ: Գործարկենք ծրագիրը և ստեղծենք 4×5 չափի զրոյական ցուցակ:

```
Enter number of rows: 4
Enter number of columns: 5
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

Նույն ցուցակը կարելի է ստեղծել նաև հետևյալ ծրագրով՝

```
n = int(input("Enter number of rows: "))
m = int(input("Enter number of columns: "))
print([[0 for i in range(m)] for j in range(n)])
```

Ստեղծված ցուցակը կարելի է փոփոխել՝ դրա տարրերին վերագրելով այլ արժեքներ հետևյալ ծրագրով՝

```
n = int(input("Enter number of rows: "))
m = int(input("Enter number of columns: "))
A = []
A = [[0]*m]*n
for i in range(n):
    for j in range(m):
        A[i][j]=int(input('Enter a number: '))
print('List A:')
for i in range(n):
    for j in range(m):
        print(A[i][j], end='')
    print()
```

Գործարկենք ծրագիրը, ստեղծենք 3×3 չափի զրոներից կազմված ցուցակ, ապա փոփոխենք այդ ցուցակի տարրերի արժեքները, կստանանք՝

```
Enter number of rows: 3
Enter number of columns: 3
Enter a number: 1
Enter a number: 9
Enter a number: 8
Enter a number: 3
Enter a number: 2
Enter a number: 7
Enter a number: 4
Enter a number: 8
Enter a number: 6
```

List A:
1 9 8
3 2 7
4 8 6

Երկչափ ցուցակի ներածումն ու արտածումը: Երկչափ ցուցակը կարելի է մուտքագրել՝ հաջորդաբար մուտքագրելով դրա կազմի մեջ մտնող ցուցակները՝ յուրաքանչյուր ցուցակի մուտքագրումն ավարտելուց հետո սեղմելով Enter: Ցուցակների տարրերը մուտքագրելիս իրարից բաժանվում են բացակներով: Օրինակ՝

```
n = int(input("Enter number of rows: ")) #Մատրիցի տողերի քանակը
A = [[int(item) for item in input().split()] for row in range(n)]
print(A)
```

Ներածենք, օրինակ, հետևյալ երկչափ ցուցակը՝ A=[[1,2,3], [4,5,6], [7,8,9]]: Գործարկելով ծրագիրը՝ կստանանք.

```
Enter number of rows: 3
1 2 3
4 5 6
7 8 9
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

A երկչափ ցուցակի տարրերը կարելի է ներածել նաև հետևյալ ծրագրով: Ծրագրում օգտագործված է f-string ֆորմատավորում (*պարագրաֆ 6.4*) :

```
A=[]
for i in range(n):
    row = [] #ընթացիկ տողը (այն նախապես դատարկ է)
    for j in range(m):
        # մուտքագրվում է ընթացիկ տողի հերթական տարրը
        Item = int(input(f"Enter element at position ({i},{j}): "))
```

row.append(item) # մուտքագրված տարրը ավելացվում է տողին

A.append(row) # ընթացիկ row տողը ավելացվում է A մատրիցին

```
for i in range(n):
    for j in range(m):
        print(A[i][j], end=" ")
    print()
```

Երկչափ ցուցակը կարելի է արտածել տարբեր ձևերով։ Օրինակ, արտածենք A=[[1,2,3], [4,5,6], [7,8,9]] ցուցակը հետևյալ հրամանով՝

```
for i in range(3):
    for j in range(3):
        print(A[i][j], end=' ')
```

Կստանանք հետևյալ արդյունքը՝

```
1 2 3 4 5 6 7 8 9
```

Ցուցակը կարելի է արտածել նաև pprint մոդուլի pprint() ֆունկցիայով (pretty-print)։

```
from pprint import pprint
L=[[1,2,3],
[4,5,6],
[7,8,9]]
pprint(L)
```

Արդյունքը՝

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

Դիտարկենք մի քանի խնդիր երկչափ ցուցակների վերաբերյալ։

Խնդիր: Գեներացնել [1,50] միջակայքին պատկանող պատահական ամբողջ թվերից կազմված $n \times m$ չափի մատրից և արտածել այն:

$n \times m$ չափի մատրիցը կարելի է դիտարկել որպես ցուցակ, որը կազմված է n ներդրված ցուցակից, որոնցից յուրաքանչյուրը պարունակում է m տարր:

Ծրագրում նախ ստեղծվել է $n \times m$ չափի զրոյական մատրից, ապա մատրիցի տարրերը փոփոխվել են՝ դրանց վերագրվել են [1,50] միջակայքին պատկանող պատահական ամբողջ թվեր:

```
from random import randint
n=3; m=4
A=[]
for i in range(n):
   A.append([0]*m)
for i in range(n):
   for j in range(m):
      A[i][j]=randint(1,50)
for i in range(n):
   for j in range(m):
      print(A[i][j], end=' ')
   print()
```

Խնդիր: Գեներացնել [1,100] միջակայքին պատկանող պատահական ամբողջ թվերից կազմված $n \times m$ չափի մատրից, հաշվել այդ մատրիցի տարրերի գումարը և միջին թվաբանականը:

```
from pprint import pprint
from random import randint
n=3; m=4
A=[]
for i in range(n):
   A.append([0]*m)

S=0
for i in range(n):
   for j in range(m):
```

```
        A[i][j]=randint(1,100)
        S+=A[i][j]

pprint(A)
print('Sum=',S, 'Average=',S/(n*m))
```

Խնդիր: Գեներացնել [1,50] միջակայքին պատկանող պատահական ամբողջ թվերից կազմված $n \times m$ չափի մատրից: Հաշվել այդ մատրիցի զույգ թվերի գումարը և միջին թվաբանականը: Եթե այդպիսի թվեր չկան, տալ այդ մասին համապատասխան հաղորդագրություն:

```
from pprint import pprint
from random import randint
n = int(input("Enter number of rows: "))
m = int(input("Enter number of columns: "))

A=[]
for i in range(n):
   A.append([0]*m)
for i in range(n):
   for j in range(m):
      A[i][j]=randint(1,50)
pprint(A)

s=count=0
for i in range(n):
   for j in range(m):
      if A[i][j]%2==0:
         s+=A[i][j]
         count+=1

if count!=0:
   print ('count=', count, 'sum=', s, 'average=', s/count)
else:
   print("The matrix doesn't contain even numbers")
```

Ծրագրի կատարման արդյունքում գեներացվել է հետևյալ երկչափ ցուցակը, որի համար հաշվարկվել է զույգ թվերի քանակը, գումարն ու միջին թվաբանականը.

```
[[41, 22, 36, 7], [50, 44, 12, 3], [9, 7, 3, 23]]
count= 5 sum= 164 average= 32.8
```

Նշենք, որ ծրագրում զույգ թվերի քանակը կարելի է հաշվել նաև հետևյալ եղանակով՝

```
count=0
count=sum(1 for i in range(n) for j in range(m) if A[i][j]%2==0)
print(count)
```

Խնդիր: Ներածել $n \times m$ չափի իրական թվերից կազմված մատրից: Հաշվել և արտածել այդ մատրիցի յուրաքանչյուր տողի տարրերի միջին թվաբանականը:

```
from pprint import pprint
n = int(input("Enter number of rows: "))
m = int(input("Enter number of columns: "))

A=[]
for i in range(n):
    row = []
    for j in range(m):
        item = float(input(f"Enter element at position ({i},{j}): "))
        row.append(item)
    A.append(row)

pprint(A)
L=[] #L վեկտորի տարրերը A մատրիցի տողերի միջին թվաբանականներն են
for i in range(n):
    s=0
    for j in range(m):
        s+=A[i][j]
```

```
    L.append(s/m)
print(L)
```

Խնդիր: Ներածել $n \times m$ չափի իրական թվերից կազմված մատրից: Հաշվել և արտածել այդ մատրիցի յուրաքանչյուր սյան մեծագույն տարրը:

```
# L ցուցակի տարրերը A մատրիցի սյուների մեծագույն տարրերն են
from pprint import pprint
n=eval(input('enter the number of rows'))
m=eval(input('enter the number of columns'))

A=[]
for i in range(n):
    row = []
    for j in range(m):
        item  = float(input(f"Enter element at position ({i},{j}): "))
        row.append(item)
    A.append(row)

L=[]
for j in range(m):
    max=A[0][j]
    for i in range(n):
        if A[i][j]>max:
            max=A[i][j]
    L.append(max)
print(L)
```

Խնդիր: Տրված են $n \times m$ չափի $A = \{a_{ij}\}$ $(i = 1,2, \dots, n; j = 1,2, \dots, m)$ և $m \times k$ չափի $B = \{b_{ij}\}$ $(i = 1,2, \dots, m; j = 1,2, \dots, k)$ մատրիցները: Հաշվել $C = A \times B$ արտադրյալը:

C մատրիցը $n \times k$ չափի է, որի c_{ij} տարրը հաշվարկվում է հետևյալ բանաձևով.

$$c_{ij} = \sum_{p=1}^{m} a_{ip} b_{pj} \ \ (i = 1,2, \dots, n; j = 1,2, \dots, k)$$

```
from pprint import pprint
n=eval(input('Enter the number of rows in the first matrix:'))
m=eval(input('Enter the number of columns in the first matrix:'))
k=eval(input('Enter the number of columns in the second matrix:'))

A=[]; B=[]; C=[]
for i in range(n):
    row = []
    for j in range(m):
        item = float(input(f"Enter element matrix A at position ({i},{j}): "))
        row.append(item)
    A.append(row)
for i in range(m):
    row = []
    for j in range(k):
        item = float(input(f"Enter element matrix B at position ({i},{j}): "))
        row.append(item)
    B.append(row)
 for p in range(m):

    C[i][j]+=A[i][p]*B[p][j]
for i in range(n):
    for j in range(k):
        print(C[i][j], end=' ')
    print()
```

Խնդիր: Դիցուք տրված է L ցուցակը, որի տարրերը երկու տարր պարունակող ցուցակներ են: Կազմել նոր ցուցակ, որում

երկչափ ցուցակի յուրաքանչյուր տարր գրված է հակառակ հերթականությամբ:

Ենթադրենք, տրված է հետևյալ ցուցակը՝ L=[[5,6],[7,8],[9,10]]

```
L=[[5,6],[7,8],[9,10]]
L1=[[y,x] for x,y in L]
print(L1)
```

Ծրագրի կատարման արդյունքն է՝ [[6, 5], [8, 7], [10, 9]]

ԳԼՈՒԽ 8. ՀԱՎԱՔԱԾՈՒՆԵՐ

8.1. Տվյալների հավաքածուների սահմանումը և հիմնական գործողությունները

Հավաքածուն (Tuple) տարրերի չփոփոխվող կարգավորված համախումբ է: Ի տարբերություն ցուցակի, այն չի կարող փոփոխվել ստեղծվելուց հետո, այսինքն՝ հնարավոր չէ հավաքածուին ավելացնել տարրեր, հեռացնել կամ փոխել որևէ տարր: Սակայն, եթե հավաքածուն պարունակում է փոփոխվող տիպի տարր, օրինակ՝ ցուցակ, ապա վերջինիս տարրերը կարելի է փոփոխել:

Tuple-ը կարող է պարունակել տարբեր տիպի տարրեր: Tuple-ի յուրաքանչյուր տարրի ամրագրված է ինդեքս:

Tuple հավաքածուի տարրերը գրվում են կլոր փակագծերի մեջ և իրարից անջատվում են ստորակետով: Օրինակ՝

```
t = (5, 6, 7)
print(t)    # Տպում է (5, 6, 7)
print(type(t))   # Տպում է <class 'tuple'>
```

Ընդհանրապես, Python-ում մի քանի տարրերի համախումբը, որոնք իրարից բաժանված են ստորակետով, դիտարկվում է որպես հավաքածու: Այսպիսով, հավաքածու կարելի է ստեղծել նաև առանց փակագծերի՝ տարրերն իրարից անջատելով ստորակետով: Օրինակ՝

```
t = 4, 5, 6
print(t)
print(type(t))
```

Արդյունքը՝

```
(4, 5, 6)
<class 'tuple'>
```

Նկատենք, որ երկու կամ ավելի փոփոխականների արժեքներն իրար հետ տեղափոխելու համար մենք օգտագործել ենք հավաքածուներ.

```
a,b = b,a
```

Դատարկ հավաքածուն գրվում է հետևյալ ձևով՝ t=(), իսկ մեկ տարր պարունակող հավաքածուն՝ t=(5,), այստեղ փակագծում միակ տարրից հետո դրվում է ստորակետ: Ստորակետ չդնելու դեպքում t=(5)-ը դիտվում է որպես ամբողջաթիվ օբյեկտ: Օրինակ՝

```
my_tuple = (5)
print(type(my_tuple)) #<class 'int'>
my_tuple = (5,)
print(type(my_tuple)) <class 'tuple'>
```

Ինչպես ցուցակի, այնպես էլ հավաքածուի տարրերը ինդեքսավորվում են՝ սկսած զրոյից: Որևէ տարրի դիմելու համար գրվում է հավաքածուի անունը, որից հետո քառակուսի փակագծերի մեջ՝ այդ տարրի ինդեքսը:

Հավաքածուների համար սահմանված են կցման (+), կրկնության (*) գործողությունները, in, not in օպերատորները:

Երկու հավաքածուների կցման ժամանակ երկրորդ հավաքածուն կցվում է առաջին հավաքածուի վերջին: Կցման արդյունքում ստացվում է հավաքածու, որի տարրերի քանակը հավասար է կցվող հավաքածուների տարրերի քանակների գումարին:

Կրկնում գործողությամբ հավաքածուն կրկնվում է նշված թվով անգամ:

in օպերատորը ստուգում է, թե արդյոք հավաքածուն պարունակո՞ւմ է նշված տարրը, իսկ not in օպերատորը՝ արդյոք հավաքածուն չի՞ պարունակում նշված տարրը:

Տողերի ու ցուցակների նմանությամբ հավաքածուի տարրերի խմբի հետ նույնպես կարելի է կատարել գործողություններ՝ կիրառելով հասցեավորման [:] օպերատորը: Օրինակ՝

```
t1 = (1, 2, 'a', 4, 5)
t2=(3,7,'c','fd')
print(t1[4])     # t1 հավաքածուի հինգերորդ տարրը
print(t2[1])     # t2 հավաքածուի երկրորդ տարրը
```

```
print(t1[:])      # t1 հավաքածուն ամբողջությամբ
print(t1[2:])     # t1 հավաքածուի տարրերը՝ 2 ինդեքսից մինչև վերջ
print(t2[:-2])    # t2-ի տարրերը՝ սկզբից մինչև նախավերջինը
print(t1+t2)      #կցում է t1 և t2 հավաքածուները, ապա՝ տպում
print(t1*3)       #t1 հավաքածուն կրկնում է 3 անգամ և տպում
```

Արդյունքը՝

```
5
7
(1, 2, 'a', 4, 5)
('a', 4, 5)
(3, 7, 'c')
(1, 2, 'a', 4, 5, 3, 7, 'c', 'fd')
(1, 2, 'a', 4, 5, 1, 2, 'a', 4, 5, 1, 2, 'a', 4, 5)
```

Հավաքածուի մի տարրից մյուսին կարելի է անցնել for հրամանի միջոցով։ Օրինակ՝

```
t = (1, 2, 'a', 4, 'bc', 5)
for x in t:
    print(x, end=" ")
```

Արդյունքը՝ 1 2 a 4 bc 5

Հավաքածուի մեջ կարող են ներդրվել հավաքածուներ։ Օրինակ՝

```
t1 = (1, 2, 'a', 4, 5)
t2=(3,7,'c','fd')
t3 = (t1, t2)
print(t3)
```

Արդյունքը՝ ((1, 2, 'a', 4, 5), (3, 7, 'c', 'fd'))

Python-ում հնարավոր չէ թարմացնել հավաքածուի տարրերը՝ ավելացնել կամ հեռացնել որևէ տարր, փոփոխել տարրի արժեքը։ Օրինակ՝ փորձենք ստորև սահմանված հավաքածուի երկրորդ տարրի արժեքը փոխել՝ դրան վերագրելով 100։

```
t = (1, 2, 3, 4, 5)
t[1] = 100
print(t)
```

Արդյունքում կարտածվի հաղորդագրություն, որ Python-ը չի աջակցում հավաքածուի տարրերի թարմացմանը.

```
Traceback (most recent call last):
  File "C:/Examples/tuple_2.py", line 2, in <module>
    t[1] = 100 # updating an element
TypeError: 'tuple' object does not support item assignment
```

Python-ում հնարավոր չէ ստեղծված հավաքածուից տարր հեռացնել, սակայն կարելի է ջնջել ամբողջ հավաքածուն del հրամանով: Օրինակ՝

```
t = (1, 2, 3, 4, 5)
del t
print(t) # հավաքածուի տպում այն հեռացնելուց հետո
```

Արդյունքում տրվում է հաղորդագրություն սխալի մասին՝ NameError, քանի որ հավաքածուն ջնջելուց հետո այն այլևս գոյություն չունի:

```
Traceback (most recent call last):
  File "C:/Examples/tuple_2.py", line 3, in <module>
    print(t)
NameError: name 't' is not defined
```

8.2. Հավաքածուների ներկառուցված ֆունկցիաներն ու մեթոդները

Հավաքածուների համար Python-ում նախատեսված են հետևյալ ներկառուցված ֆունկցիաները.

Ֆունկցիա	Գործողությունը
len(tuple)	Վերադարձնում է հավաքածուի երկարությունը:
min(tuple)	Վերադարձնում է հավաքածուի նվազագույն տարրը:
max(tuple)	Վերադարձնում է հավաքածուի առավելագույն տարրը:
tuple(seq)	seq հաջորդականությունը փոխակերպում է հավաքածուի

Աղյուսակ 8.1. Հավաքածուների ներկառուցված ֆունկցիաները

Հետևյալ կոդը որոնում և արտածում է թվային տվյալներ պարունակող հավաքածուի մեծագույն տարրը:

```
tuple = ( 11, 3, 41, 1, 2, 7 )
r = max(tuple)
print('Maximum of tuple is', r)
```

Այժմ ենթադրենք, որ հավաքածուն պարունակում է տողային տիպի տարրեր և գտնենք այդպիսի հավաքածուի մեծագույն և փոքրագույն տարրերը:

```
tuple1, tuple2 = ('maths', 'che', 'phy', 'bio'), ('mat', 'che', 'phy', 'qio')
print ("Maximum of tuple1 is: ", max(tuple1))
print ("Minimum of tuple1 is: ", min(tuple1))
print ("Maximum of tuple2 is: ", max(tuple2))
print ("Minimum of tuple2 is: ", min(tuple1))
```

Արդյունքը՝

```
Maximum of tuple1 is:  phy
```

```
Minimum of tuple1 is:  bio
Maximum of tuple2 is:  qio
Minimum of tuple2 is:  bio
```

Հավաքածուները, ինչպես ցուցակները, ունեն count() և index() մեթոդները, որոնք ներկառուցված մեթոդներ են: Բայց, քանի որ հավաքածուն կարգավորված և չփոփոխվող օբյեկտ է, այն չունի sort() և reverse() մեթոդները, որոնք փոփոխում են օբյեկտը:

count(value) մեթոդը հաշվում և վերադարձնում է value տարրի կրկնությունների քանակը հավաքածուի մեջ: Մեթոդի շարահյուսությունն է՝

```
tuple.count(value)
```

Հետևյալ ծրագրում հաշվվում է տրված թվային և տողային տարրերի կրկնությունների քանակը երկու տարբեր հավաքածուների մեջ: Այն դեպքում, երբ որոնվող տարրը բացակայում է հավաքածուի մեջ, վերադարձվում է 0:

```
tuple1 = (1, 3, 5, 8, 7, 5, 4, 6, 8, 5)
tuple2 = ('Python', 'Fortran', 'Python','Assembler', 'Python', 'Assembler')
r1=tuple1.count(5)
r2=tuple2.count('Assembler')
r3=tuple2.count('Basic')
print('Count of 5 in tuple1 is:', r1)
print('Count of Assembler in tuple2 is:', r2)
print('Count of Basic in tuple2 is:', r3)
```

Պատասխան՝

```
Count of 5 in tuple1 is: 3
Count of Assembler in tuple2 is: 2
Count of Basic in tuple2 is: 0
```

Ստորև ներկայացված ծրագիրը count() մեթոդով հաշվում է որոնվող տարրի կրկնությունների քանակը այն դեպքում, երբ այդ տարրը ցուցակ կամ հավաքածու է։

```
tuple = (0, 1, 'Python', [1,2], (1, 2), 1, ('A', 'B'),
        [1, 2], 'Fortran', (0), ('A', 'B'), [1, 2], (1, 2))
r4 = tuple.count([1, 2])
print('Count of [1, 2] in tuple is:', r4)
r5 = tuple.count((1, 2))
print('Count of (1, 2) in tuple is:', r5)
r6 = tuple.count(('A', 'B'))
print("Count of ('A', 'B') in tuple is:", r6)
r7 = tuple.count(('M', 'N'))
print("Count of ('M', 'N') in tuple is:", r7)
```

Արդյունքը.

```
Count of [1, 2] in tuple is: 3
Count of (1, 2) in tuple is: 2
Count of ('A', 'B') in tuple is: 2
Count of ('M', 'N') in tuple is: 0
```

index() մեթոդը վերադարձնում է որոնվող տարրի առաջին հանդիպման ինդեքսը հավաքածուի մեջ։ Մեթոդի շարահյուսությունն է՝

```
tuple.index(element, start, end)
```

Այս մեթոդով tuple հավաքածուի մեջ որոնվում է element տարրը՝ սկսած start ինդեքսով տարրից մինչև end ինդեքսով տարրը։ Մեթոդը վերադարձնում է որոնվող element տարրի առաջին հանդիպման ինդեքսը հավաքածուի մեջ։ Եթե start և end պարամետրերը տրված չեն, ապա որոնումը կատարվում է ամբողջ հավաքածուի մեջ՝ սկզբից մինչև վերջ։ Որոնվող տարրի բացակայության դեպքում ծագում է սխալ և տրվում է հաղորդագրություն այդ մասին։ Օրինակ՝

```
tuple = ( 20,3,3,5,7,8,9,6,3,5,4,3,5,5)
r1 = tuple.index(5)
print('Index of 5 is', r1)
r2 = tuple.index(5,4,12)
print('Index of 5 from index 4 to 12 is', r2)
print(tuple.index(30))  #30 թիվը առկա չէ հավաքածուի մեջ
```

Արդյունքը՝

```
Index of 5 is 3
Index of 5 from index 4 to 12 is 9
Traceback (most recent call last):
  File "C:/Examples/tuple_3.py", line 6, in <module>
    print(tuple.index(30))
ValueError: tuple.index(x): x not in tuple
```

tuple() ֆունկցիայի միջոցով կարելի է հաջորդականությունը փոխակերպել հավաքածուի: Օրինակ.

```
t1 = tuple([1,2,3])
t2 = tuple('abcde')
```

Պատասխան՝

```
t1= (1, 2, 3) # ցուցակը փոխակերպում է հավաքածուի
t2= ('a', 'b', 'c', 'd', 'e')   # տողը փոխակերպում է հավաքածուի
```

ԳԼՈՒԽ 9. ԲԱԶՄՈՒԹՅՈՒՆՆԵՐ

9.1. Բազմությունների սահմանումը և հիմնական գործողությունները

Բազմությունը (Set) Python լեզվում չկրկնվող, չփոփոխվող տարրերի չկարգավորված համախումբ է: Բազմությունը փոփոխվող օբյեկտ է, այսինքն՝ բազմության ստեղծումից հետո կարելի է դրան տարրեր ավելացնել կամ հեռացնել: Սակայն բազմության տարրերը պետք է լինեն չփոփոխվող (օրինակ՝ թվեր, տողեր, հավաքածուներ):

Բազմության տարրերը վերցվում են ձևավոր փակագծերի մեջ և իրարից անջատվում են ստորակետով: Բազմության տարրերին հնարավոր չէ դիմել ինդեքսների միջոցով:

Ստեղծենք հետևյալ բազմությունը և տպենք այն: Նշենք, որ բազմության տարրերի արտածման համար հատուկ հերթականություն սահմանված չէ, և յուրաքանչյուր անգամ այն կարող է լինել տարբեր:

```
v = {'A', 'C', 4, '5', 'B'}
print('The set v is:', v)
```

Տպվում է՝

```
The set v is: {'B', 4, 'A', '5', 'C'}
```

Եթե բազմությունում կան կրկնվող տարրեր, ապա արտածելիս կրկնությունները հեռացվում են, և յուրաքանչյուր տարր հանդես է գալիս ընդամենը մեկ անգամ: Օրինակ՝

```
v = {'A', 4, 'C', 4, '5', 'B',4}
print( v)
```

Արտածվում է՝

```
{4, 'C', 'B', '5', 'A'}, որտեղ 4 թվի կրկնությունները հեռացված են:
```

Նշենք, որ True և 1 (ինչպես նաև՝ False և 0) արժեքները բազմությունում համարվում են նույն տարրը և դիտարկվում են որպես կրկնություն։

Python-ի բազմությունների հետ կարելի է կատարել հետևյալ գործողությունները.

- կրկնությունների հեռացում,
- բազմությանը անդամակցության ստուգում in և not in օպերատորներով,
- մաթեմատիկական գործողություններ՝ բազմությունների միավորում, հատում, տարբերություն և սիմետրիկ տարբերություն։

Բազմություն կարելի է ստեղծել մի քանի եղանակով։ Նախ, ինչպես արդեն նշել ենք վերևում, բազմություն կարելի է ստեղծել՝ ձևավոր փակագծերի մեջ վերցնելով բազմության տարրերը և դրանք իրարից անջատելով ստորակետով։ Օրինակ՝

```
#Բազմության ստեղծում՝ տարրերը ձևավոր փակագծերի մեջ ներառելով
set5 = {10, 10, 'program', 'program', 3.14, (10, 20, 30), None}
print('set5:', set5)
```

Ստեղծված բազմությունն է.

```
set5: {None, 3.14, 'program', 10, (10, 20, 30)}
```

Բազմություն կարելի է ստեղծել նաև ներկառուցված set() ֆունկցիայի միջոցով՝ կիրառելով այն ցուցակի, հավաքածուի կամ տողի նկատմամբ։

```
# Բազմության ստեղծում set() ֆունկցիայի միջոցով
set1 = set([1, 1, 1, 2, 2, 3,4,4])      # ցուցակից
set2 = set(('d', 'd', 'b', 'b', 'c'))    # հավաքածուից
set3 = set('programming')                # տողից
print('Set1:', set1)
print('Set2:', set2)
print('Set3:', set3)
```

Պատասխան՝

```
Set1: {1, 2, 3, 4}
Set2: {'d', 'b', 'c'}
Set3: {'i', 'a', 'o', 'g', 'm', 'n', 'r', 'p'}
```

Ինչպես նշեցինք, ցուցակի, հավաքածուի կամ տողի հիման վրա set() ֆունկցիայի միջոցով բազմություն ստեղծելիս տարրերի կրկնությունները հեռացվում են, և յուրաքանչյուր տարր հանդես է գալիս ընդամենը մեկ անգամ: Այս հատկությունը կարելի է օգտագործել ցուցակները կրկնություններից մաքրելու համար: Օրինակ, հեռացնենք [3, 6, 3, 5] ցուցակի կրկնությունները: Դրա համար նախ set() ֆունկցիայի միջոցով [3, 6, 3, 5] ցուցակի հիման վրա կստեղծենք {3, 6, 5} բազմությունը, որից հեռացվում են կրկնությունները, ապա list() ֆունկցիայի միջոցով ստացված բազմությունը կփոխակերպենք ցուցակի.

```
L=list(set([3, 6, 3, 5]))
```

Արդյունքում ստացվում է կրկնություններից մաքրված [3, 5, 6] ցուցակը:

Բազմություն կարելի է ստեղծել նաև range() ֆունկցիայի միջոցով՝ որևէ միջակայքի հիման վրա: Օրինակ՝

```
set4=set(range(10))
```

Հրամանի կատարման արդյունքում ստեղծվում է հետևյալ բազմությունը.

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Կարևոր է հիշել, որ բազմության տարրերը չեն կարող լինել փոփոխվող տիպեր, օրինակ՝ ցուցակ, բառարան: Հետևյալ օրինակում փորձ է արվել ստեղծել բազմություն, որի տարրերի թվում է [10, 20, 30] ցուցակը:

```
Set6 = {10, 'program', 3.14, [10, 20, 30], {10, 20, 30}}
print('Set6:', set6)
```

Ծրագրի կատարման արդյունքում ծագել է սխալ և տրվել է հաղորդագրություն սխալի մասին.

```
Traceback (most recent call last):
  File  "C:/Examples/sets_4.py", line 2, in <module>
    Set6 = {10, 'program', 3.14, [10, 20, 30], {10, 20, 30}}
TypeError: unhashable type: 'list'
```

9.2. Բազմությունների ներկառուցված ֆունկցիաներն ու մեթոդները

Բազմությանը մեկ չփոփոխվող տարր կարելի է ավելացնել add() մեթոդով, իսկ մի քանի տարր՝ update() մեթոդով: Այս մեթոդների համար որպես արգումենտներ կարող են հանդես գալ չփոփոխվող տարրեր: Ընդ որում, եթե կան տարրերի կրկնություններ, ապա դրանք հեռացվում են: Բացի դրանից, եթե փորձենք ավելացնել փոփոխվող տարր, կտրվի հաղորդագրություն սխալի մասին: Օրինակ՝

```
myset = set()
myset.add('a')        #մեկ չփոփոխվող տարրի ավելացում
print(myset)
myset.update({'b','c'})    #չփոփոխվող տարրերից կազմված բազմության ավելացում
print(myset)
myset.update(['d', 'd', 'd']) #չփոփոխվող տարրերից կազմված ցուցակի ավելացում
print(myset)
myset.update(['e'], ['f'])    #չփոփոխվող տարրերից մի քանի ցուցակի ավելացում
print(myset)
myset.update('fgh')    #տողային տարրի ավելացում
print(myset)
myset.update([[1, 2], [3, 4]]) #փոփոխվող տարրերի ցուցակ ավելացնելու փորձ
```

```
print(myset)
```

Պատասխան՝

```
{'a'}
{'b', 'c', 'a'}
{'b', 'c', 'a', 'd'}
{'f', 'd', 'b', 'c', 'a', 'e'}
{'f', 'h', 'd', 'g', 'b', 'c', 'a', 'e'}
Traceback (most recent call last):
  File "C:/Examples/sets_6.py", line 12, in <module>
    myset.update([[1, 2], [3, 4]])  # #փոփոխվող տարրերի ցուցակ ավելացնելու փորձ
TypeError: unhashable type: 'list'
```

Ծրագրի վերջին հատվածում փորձ է արվել բազմությանն ավելացնել փոփոխվող տարրերից ցուցակ, և ծրագրի գործարկման արդյունքում տրվել է հաղորդագրություն սխալի մասին՝ TypeError: unhashable type: 'list':

Դիտարկենք որոշ խնդիրներ բազմությունների մեթոդների վերաբերյալ: Հետևյալ խնդրում բազմությունների միավորման ու հատման համար կիրառվել են union() և intersection() մեթոդները համապատասխանաբար:

Խնդիր: Արտածել s1= {0, 1, 2, 3, 4} և s2={0, 1} բազմությունների միավորումն ու հատումը:

```
s1 = set(range(5))
s2 = set(range(2))
print('s1:',s1)
print('s2:',s2)
print('s1 ∩ s2:',s1.intersection(s2)) #բազմությունների հատումը
print('s1 ∪ s2:',s1.union(s2)) #բազմությունների միավորումը
```

Ծրագրի կատարման արդյունքն է՝

```
s1: {0, 1, 2, 3, 4}
s2: {0, 1}
```

s1 ∩ s2: {0, 1}
s1 ∪ s2: {0, 1, 2, 3, 4}

Բազմությունից տարրեր կարելի է հեռացնել հետևյալ չորս մեթոդներից որևէ մեկով։

Մեթոդը	Նկարագրությունը
discard(x)	Հեռացնում է x տարրը։ Եթե այդպիսի տարր չկա բազմության մեջ, ոչինչ չի անում։
remove(x)	Հեռացնում է x տարրը։ Եթե այդպիսի տարր չկա բազմության մեջ, գեներացնում է հաղորդագրություն՝ KeyError։
pop()	Հեռացնում է որևէ պատահական տարր և վերադարձնում այն կամ գեներացնում է հաղորդագրություն՝ KeyError, եթե բազմությունը դատարկ է։
clear()	Հեռացնում է բոլոր տարրերը։

```
myset = {1, 2, 3, 4}
print(myset)
myset.discard(1)  # հեռացվում է բազմության մեջ առկա 1 թիվը
print(myset)
myset.discard(5)  # հեռացվում է տարր, որն առկա չէ
print(myset)
myset.remove(4) # հեռացվում է բազմության մեջ առկա 4 թիվը
print(myset)
myset.remove(5) # տարրի հեռացում, որն առկա չէ
print(myset)
```

Արդյունքը՝

```
{1, 2, 3, 4}
{2, 3, 4}
{2, 3, 4}
{2, 3}
Traceback (most recent call last):
  File "C:/Examples/sets_7.py", line 9, in <module>
    myset.remove(5)   # the item was absent in the set
```

KeyError: 5

Դիտարկենք նաև հետևյալ ծրագիրը, որտեղ myset բազմությունից pop() մեթոդով պատահականորեն հեռացվում է որևէ տարր, որից հետո clear() մեթոդով հեռացվում են մնացած բոլոր տարրերը: Այնուհետև, կրկին փորձ է արվում հեռացնել որևէ պատահական տարր pop() մեթոդով, բայց, քանի որ բազմությունն արդեն դատարկ է, ծագում է սխալ, և այդ մասին տրվում է հաղորդագրություն՝ KeyError: 'pop from an empty set':

```
myset = {1, 2, 3, 4}
print(myset)
myset.pop() # հեռացվում է մեկ պատահական տարր
print(myset)
myset.clear() # հեռացվում են մնացած բոլոր տարրերը
print(myset)
myset.pop() # կրկին փորձ է արվում հեռացնել մեկ պատահական տարր
print(myset)
```

Ծրագրի կատարման արդյունքը՝

```
{1, 2, 3, 4}
{2, 3, 4}
set()
Traceback (most recent call last):
  File "C:/Examples/sets_8.py", line 7, in <module>
    myset.pop()
KeyError: 'pop from an empty set'
```

Python-ի ներկառուցված տվյալների տիպերից է frozenset-ը: Այն, ինչպես set բազմությունը չկրկնվող, չփոփոխվող տարրերի չկարգավորված հավաքածու է: Սակայն, ի տարբերություն բազմության, frozenset-ում չի կարելի տարր ավելացնել կամ հեռացնել:

frozenset կարելի է ստեղծել ցուցակից, հավաքածուից կամ բազմությունից: Օրինակ՝

```
list = [1, 3, 6, 3, 9]
frozenset1 = frozenset(list)
print("Frozenset from list:", frozenset1)
tuple = ("Fortran", "Basic", "Python")
frozenset2 = frozenset(tuple)
print("Frozenset from tuple:", frozenset2)
frozenset3 = frozenset()
print("Empty frozenset:", frozenset3)
```

Արդյունքը՝

```
frozenset from list: frozenset({1, 3, 6, 9})
frozenset from tuple: frozenset({'Basic', 'Fortran', 'Python'})
Empty frozenset: frozenset()
```

Եթե frozenset-ին փորձենք ավելացնել որևէ տարր, ապա կտրվի հաղորդագրություն սխալի մասին: Օրինակ՝

```
languages = frozenset({"Fortran", "Basic", "Python"})
print('Initial frozenset', languages)
languages.add("Assembler")
print(languages)
```

Արդյունքը՝

```
Initial frozenset frozenset({'Fortran', 'Python', 'Basic'})
Traceback (most recent call last):
  File "C:/Examples/sets_9.py", line 3, in <module>
    languages.add("Assembler")
AttributeError: 'frozenset' object has no attribute 'add'
```

ԳԼՈՒԽ 10. ԲԱՌԱՐԱՆՆԵՐ

10.1. Բառարանի սահմանումը և ստեղծումը

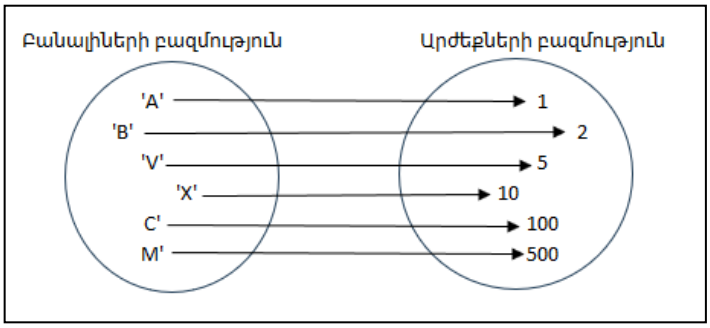
Բառարանը (Dictionaries) այնպիսի տարրերի համախմբություն է, որոնցից յուրաքանչյուրը բանալի-արժեք զույգ է: Բանալին բառարանում պետք է լինի եզակի, չկրկնվող, քանի որ դրանով նույնականացվում է տարրը, իսկ արժեքները կարող են կրկնվել: Բանալին պետք է լինի չփոփոխվող տիպի տվյալ (օրինակ՝ տող, թիվ), իսկ արժեքը՝ կամայական տիպի:

Բառարանը սահմանելու համար դրա տարրերը վերցվում են ձևավոր փակագծերի մեջ՝ {} և իրարից անջատվում են ստորակետով, իսկ բանալին իր արժեքից անջատվում է վերջակետով՝ (:):

Բառարանը կարող է լինել դատարկ: Դատարկ բառարանը հետևյալն է՝ {}: Բառարանը փոփոխվող օբյեկտ է: Բառարանի համար սահմանված չէ կարգավորման գաղափարը, այն չկարգավորվող օբյեկտ է:

Սահմանենք հետևյալ d բառարանը:

```
d={'A':1, 'B':2, 'V':5, 'X':10, 'C':100, 'M':1000}
```



Նկար 10.1. Բառարանի բանալիների ու արժեքների կապը

d բառարանի առաջին տարրի բանալին 'A'-ն է, իսկ 1-ը՝ դրա արժեքը, երկրորդ տարրում 'B'-ն բանալին է, իսկ 2-ը՝ դրա արժեքը և այլն: Բանալին գործում է, ինչպես ինդեքսը: Դրանով կարելի է դիմել համապատասխան արժեքներին: Բառարանի որևէ տար-

րի դիմելու համար գրվում է բառարանի անունը, ապա քառակուսի փակագծերում՝ այդ տարրի բանալու արժեքը: Օրինակ՝ d['A']-ով դիմում ենք բառարանի առաջին տարրին:

Դիտարկենք հետևյալ days անունով բառարանը, որը պարունակում է տարվա ամիսների անունները և համապատասխան օրերի քանակը: Բառարանի յուրաքանչյուր տարրի բանալին ամսվա անունն է, իսկ արժեքը՝ տվյալ ամսվա օրերի քանակը:

```
days={'January':31,'February':28,'March':31,'April':30,'May':31,'June':30,'July':31,'August':31,'September':30,'October':31,'November':30,'December':31}
```

Որևէ տարրի արժեքը կարելի է ստանալ՝ դիմելով համապատասխան բանալուն: Օրինակ՝

```
days['January']=31
days['April']=30
```

Բառարանը կարելի է թարմացնել՝ ավելացնել նոր տարրեր, փոփոխել գոյություն ունեցող տարրը կամ հեռացնել որևէ տարր:

Բառարանում որևէ տարրի արժեքը փոխելու համար բավական է բանալիով դիմել տվյալ տարրին և փոխել արժեքը: Օրինակ՝ դիտարկենք վերևում ներկայացված d բառարանը: 'A' բանալիով տարրի արժեքը 100 դարձնելու համար պետք է գրել.

```
d['A']=100
```

Այժմ բառարանում ավելացնենք նոր տարր 'E' բանալիով և 1 արժեքով: Բառարանում այդ նոր տարրը կարելի է ավելացնել հետևյալ հրամանով.

```
d['E']=1
```

Համոզվենք, որ բառարանում ավելացել է 'E' բանալիով նոր տարր: Արտածենք d բառարանը.

```
>>>d
{'A': 1, 'B': 2, 'V': 5, 'X': 10, 'C': 100, 'M': 1000, 'E': 1}
```

Այժմ d բառարանում 1 արժեքը կրկնվում է՝ d['A']=1 և d['E']=1:

Նկատենք, որ բերված եղանակով չէինք կարող նոր տարր ավելացնել ցուցակում:

Եթե փորձենք դիմել բառարանի տարրի՝ գոյություն չունեցող բանալու միջոցով, ապա կստանանք սխալի մասին հաղորդագրություն: Օրինակ, եթե դիմենք d բառարանի 'N' բանալիով տարրի, ապա կստանանք հաղորդագրություն սխալի մասին, քանի որ նման տարր գոյություն չունի.

```
>>> d={'A':1, 'B':2, 'V':5, 'X':10, 'C':100, 'M':1000}
>>> d['N']
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    d['N']
KeyError: 'N'
```

del ֆունկցիայով կարելի է բառարանից հեռացնել որևէ տարր կամ ջնջել բառարանն ամբողջությամբ: clear() մեթոդով հեռացվում են բառարանի բոլոր տարրերը:

```
d={'A':1, 'B':2, 'V':5, 'X':10, 'C':100, 'M':1000}
del d['A']  # d բառարանից հեռացվում է 'A' բանալիով տարրը
print(d)
d.clear()       # d բառարանից հեռացվում են բոլոր տարրերը
print(d)
del d           #ջնջում է d բառարանն ամբողջությամբ
print(d)
```

Ծրագրի կատարման արդյունքն է.

```
{'B': 2, 'V': 5, 'X': 10, 'C': 100, 'M': 1000}
{}
Traceback (most recent call last):
  File "C:/Examples/dict_1.py", line 7, in <module>
    print(d)
NameError: name 'd' is not defined
```

Ծրագրում del ֆունկցիայով բառարանից նախ հեռացվել է 'A' բանալիով տարրը, ապա clear() մեթոդով՝ բառարանի մնացած բոլոր տարրերը: Հեռացնելուց հետո print(d) հրամանով արտած-

վել է դատարկ բառարան՝ {}: Այնուհետև del ֆունկցիայով բառարանը ջնջվել է ամբողջությամբ, որից հետո print(d) հրամանի կատարումը տվել է հաղորդագրություն առաջացած սխալի մասին՝ NameError: name 'd' is not defined, քանի որ d անունով բառարան այլևս սահմանված չէ:

10.2. Բառարանների ներկառուցված ֆունկցիաներն ու մեթոդները

Python-ում բառարանի համար սահմանված են հետևյալ ներկառուցված ֆունկցիաները.

Ֆունկցիա	Նկարագրությունը
dict1== dict2	Համեմատում է dict1և dict2 բառարանները: Եթե դրանք համընկնում են, վերադարձնում է True, հակառակ դեպքում՝ False:
len(dict)	Վերադարձնում է dict բառարանի երկարությունը:
str(dict)	dict բառարանը փոխակերպում է տողի:
del dict[key]	dict բառարանից հեռացվում է key բանալիով տարրը:
del dict	Ջնջում է dict բառարանը:

Աղյուսակ 10.1. Բառարանների ներկառուցված ֆունկցիաները

Python-ում բառարանների համար նախատեսված մեթոդներն են.

Մեթոդ	Նկարագրությունը
dict.clear()	Ջնջում է dict բառարանի բոլոր տարրերը:
dict.copy()	Ստեղծում է dict բառարանի կրկնօրինակը:
dict.fromkeys(seq, value)	Ստեղծում է նոր բառարան՝ dict, տրված seq հաջորդականության բանալիներով և value արժեքով: value արժեքի բացակայության դեպքում բանալիներին վերագրվում է None արժեքը:
dict.get(key,	Վերադարձնում է key բանալիով տարրի

default=None)	արժեքը: Եթե այդ բանալին բացակայում է բառարանում, վերադարձնում է default արժեքը:
dict.keys()	Վերադարձնում է բառարանի բանալինե-րը:
dict.values()	Վերադարձնում է բառարանի արժեքները:
dict.items()	Վերադարձնում է բառարանի բանալի-ար-ժեք զույգերի ցուցակը:
dict.setdefault(key, default=None)	Վերադարձնում է key բանալիով տարրի արժեքը: Եթե այդ բանալին բացակայում է բառարանում, ներածում է key բանալիով և default արժեքով տարրը: Իսկ եթե default արժեքը տրված չէ, այդ դեպքում key բա-նալուն վերագրվում է None արժեքը:
dict1.update(dict2)	dict2 բառարանի բանալի-արժեք զույգերը ավելացնում է dict1 բառարանին:
dict.pop(key)	Հեռացնում է key բանալիով տարրը և վե-րադարձնում է դրան համապատասխա-նող արժեքը: Եթե բառարանում այդպիսի բանալի չկա, այդ մասին տրվում է հաղոր-դագրություն:
dict.popitem()	Հեռացնում և վերադարձնում է բառարանի վերջին տարրը:

Աղյուսակ 10.2. Բառարանների ներկառուցված մեթոդները

Դիտարկենք մի քանի օրինակներ բառարանի մեթոդների վերաբերյալ:

clear() մեթոդը Python-ում օգտագործվում է բառարանից բո-լոր տարրերը (բանալի-արժեք զույգերը) հեռացնելու համար: Այս մեթոդը կանչելուց հետո բառարանը կդառնա դատարկ, և դրա երկարությունը կլինի 0: Օրինակ՝

```
d = {1: "one", 2: "two"}
d.clear() #հեռացնում է բառարանի բոլոր տարրերը
print(d) #արտածում է՝ {}
```

print(len(d)) # բառարանի երկարությունը տարրերը ջնջելուց հետո՝ 0

copy() մեթոդը վերադարձնում է բառարանի պատճենը: Օրինակ.

```
d1 = {1: 'one', 2: 'two', 10: [1, 2, 3], 20: 'c'}
print("Original Dictionary:", d1)
d2 = d1.copy() #ստեղծում է բառարանի կրկնօրինակը
print("New copy:", d2)
```

Ծրագրի կատարման արդյունքն է.

```
Original Dictionary: {1: 'one', 2: 'two', 10: [1, 2, 3], 20: 'c'}
New copy: {1: 'one', 2: 'two', 10: [1, 2, 3], 20: 'c'}
```

fromkeys(seq, val) մեթոդը ստեղծում է նոր բառարան: seq հաջորդականության մեջ տրվում են բառարանի բանալիները, իսկ val-ը այն արժեքն է, որը պետք է վերագրվի այդ բոլոր բանալիներին: val արժեքի բացակայության դեպքում բանալիներին վերագրվում է None արժեքը:

Հետևյալ օրինակում ստեղծվում են չորս նոր բառարաններ, որոնց բանալիները տրված են seq հաջորդականության մեջ: Առաջին բառարանի բոլոր բանալիների արժեքները None են, երկրորդինը՝ ևս None, բայց՝ լռելյայն: Երրորդ բառարանի բոլոր բանալիներին վերագրված է 1 արժեքը, իսկ չորրորդ բառարանի բանալիներին՝ [1,2,3] ցուցակը:

```
seq = ('a', 'b', 'c')
print('The new dictionary with None values:', dict.fromkeys(seq, None))
print('The new dictionary with default None values:', dict.fromkeys(seq))
lst1=1
print('The new dictionary with values as 1:', dict.fromkeys(seq, lst1))
lst2=[1,2,3]
```

print('The new dictionary with list values:', dict.fromkeys(seq, lst2))

Արդյունքը.

The new dictionary with None values: {'a': None, 'b': None, 'c': None}
The new dictionary with default None values: {'a': None, 'b': None, 'c': None}
The new dictionary with values as 1: {'a': 1, 'b': 1, 'c': 1}
The new dictionary with list values: {'a': [1, 2, 3], 'b': [1, 2, 3], 'c': [1, 2, 3]}

dict.get(key, default_value) մեթոդը վերադարձնում է key բանալիով տարրի արժեքը, եթե այդ բանալին առկա է բառարանում: Եթե բառարանում key բանալի չկա, և երկրորդ արգումենտը բացակայում է, ապա մեթոդը լռելյայն վերադարձնում է None: Իսկ եթե key և default_value երկու արգումենտներն էլ առկա են, բայց բառարանում key բանալի չկա, ապա մեթոդը վերադարձնում է default_value արժեքը:

```
dict = {'Name': 'Mary', 'Age': '21','Gender': 'Female'}
print(dict.get('Name'))
print(dict.get('Gender'))
print(dict.get('Country'))
print(dict.get('Country', 'Not found'))
```

Ծրագրի կատարման արդյունքն է.

Mary
Female
None
Not found

setdefault(key, default_value) մեթոդը վերադարձնում է key բանալիով տարրի արժեքը, եթե այդ բանալին առկա է բառարանում: Հակառակ դեպքում՝ բառարանում ներմուծվում է key բա-

նալիով նոր տարր, որին վերագրվում է default_value արժեքը, եթե այդ արժեքը տրված է: Իսկ եթե default_value արժեքը տրված չէ, այդ դեպքում key բանալուն վերագրվում է None արժեքը: Օրինակ՝

```
dict = {'a': 1, 'b': 2, 'c':3}
print('The original dictionary:', dict)
print('setdefault() returned:', dict.setdefault('b', 20)) # բանալին առկա է
print('After using setdefault():', dict)
val = dict.setdefault('d', 'four') # բանալին գոյություն չունի
print("Return value of setdefault():", val)
print("Dictionary after using setdefault():", dict)
val1 = dict.setdefault('e')
print("Return value of setdefault():", val1)
print("Dictionary after using setdefault():", dict)
```

Ծրագրի գործարկման արդյունքում կունենանք.

```
The original dictionary: {'a': 1, 'b': 2, 'c': 3}
setdefault() returned: 2
After using setdefault(): {'a': 1, 'b': 2, 'c': 3}
Return value of setdefault(): four
Dictionary after using setdefault(): {'a': 1, 'b': 2, 'c': 3, 'd': 'four'}
Return value of setdefault(): None
Dictionary after using setdefault(): {'a': 1, 'b': 2, 'c': 3, 'd': 'four', 'e': None}
```

dict.pop(key) մեթոդը հեռացնում է key բանալիով տարրը բառարանից և վերադարձնում է դրան համապատասխանող արժեքը: Եթե բառարանում այդպիսի բանալիով տարր առկա չէ, այդ մասին տրվում է հաղորդագրություն: Օրինակ՝

```
scores = {'Ann': 7, 'Mary': 1, 'Bob': 2}
print('The dictionary before deletion : ' , scores)
val = scores.pop('Mary')
```

```
print('The removed value:', val)
print('Dictionary after deletion is : ' , scores)
val1=scores.pop('Aram')
```

Ծրագրի կատարման արդյունքն է.

```
The dictionary before deletion :  {'Ann': 7, 'Mary': 1, 'Bob': 2}
The removed value: 1
Dictionary after deletion is :  {'Ann': 7, 'Bob': 2}
```

```
Traceback (most recent call last):
File "C:/Examples/dict_2.py", line 6, in <module>
   val1=scores.pop('Aram') # the key is not existing.
KeyError: 'Aram'
```

dict.popitem() մեթոդը dict հեռացնում է բառարանի վերջին տարրը (բանալի-արժեք զույգը) և վերադարձնում այն։ Օրինակ՝

```
d = {'a': '1', 'b': '2', 'c': '3'}
removed_item = d.popitem()
print(removed_item)
print(d)
```

Արդյունքը՝

```
('c', '3')
{'a': '1', 'b': '2'}
```

dict1.update(dict2) մեթոդը dict2 բառարանի տարրերն ավելացնում է dict1 բառարանին։ Օրինակ՝

```
scores1 = {'Ann': 20, 'Henry': 15}
scores2 = {'Bob': 16, 'Kate': 20}
scores1.update(scores2)
print(scores1)
```

Արդյունքը՝

```
{'Ann': 20, 'Henry': 15, 'Bob': 16, 'Kate': 20}
```

Խնդիր: Կազմել revenue անունով բառարան, որը պարունակում է խանութի մեկ օրվա վաճառքի ցուցանիշները: Բառարանում յուրաքանչյուր տարրի բանալին ապրանքի անունն է, իսկ արժեքը՝ այդ ապրանքի վաճառքից ստացված հասույթը: Հաշվել խանութի բոլոր ապրանքների վաճառքից ստացված մեկ օրվա հասույթը՝ TR:

```
revenue={'TV':20000,'Monitor':15000,'Mouse':1200,'refrigerator':3
0000,'iron':560,'laptop':17000}
TR=sum([revenue[product] for product in revenue])
print(TR)
```

Ծրագրի կատարման արդյունքն է՝ 83760:
Ծրագիրը կարող ենք գրել նաև հետևյալ եղանակով.

```
revenue={'TV':20000,'Monitor':15000,'Mouse':1200,'refrigerator':3
0000,'iron':560,'laptop':17000}
Total_Revenue=0
for product in revenue:
   Total_Revenue +=revenue[product]
print(TR)
```

in և not in օպերատորները: Այս օպերատորներով կարելի է ստուգել որևէ բանալիով տարրի առկայությունը բառարանում: Ինչպես արդեն նշել ենք, բառարանում որևէ գոյություն չունեցող բանալուն հղումը հանգեցնում է սխալի: Օրինակ՝ դիցուք տրված է հետևյալ բառարանը.

```
d={'A':1, 'B':2, 'V':5, 'X':10, 'C':100, 'M':1000}
```

'K' բանալիով տարրի դիմելը՝ d['K'], հանգեցնում է սխալի, և տրվում է հաղորդագրություն գոյություն չունեցող բանալիով տարրին հղում տալու մասին՝ KeyError: 'K': Ուստի, սխալը կանխելու նպատակով, որևէ տարրի հղում կատարելու համար, նախապես պետք է ստուգել համապատասխան բանալու առկայությունը բառարանում:

Օրինակ, եթե ցանկանում ենք իմանալ խանութում որևէ ապրանքի մեկ օրվա վաճառքից հասույթը, նախապես պետք է ստուգենք այդ ապրանքատեսակի (բանալու) առկայությունը բառարանում:

```
revenue={'TV':20000,'Monitor':15000,'Mouse':1200,'refrigerator':30000,'iron':560,'laptop':17000}
prod=input('Enter product_name: ')
if prod in revenue:
   print('The revenue from this product is: ', revenue[prod])
else:
   print('Not in dictionary')
```

Ստուգենք բջջային հեռախոսի (mobile_phone) առկայությունը խանութում.

```
Enter product_name: mobile_phone
```

Հարցման պատասխանն է՝ Not in dictionary

Իսկ այժմ ստուգենք հեռուստացույցի առկայությունը և, բառարանում դրա առկայության դեպքում, արտածենք այդ ապրանքի վաճառքից հասույթը.

```
Enter product_name: TV
The revenue from this product is:  20000
```

Ինչպես տեսնում ենք, 'TV' բանալի կա բառարանում, և հեռուստացույցի վաճառքից հասույթը կազմել է 20000 պ.դ.մ.:

Բառարանի բանալիների և արժեքների տպումը: Բառարանի բանալիները և դրանց արժեքները կարելի է արտածել: Ցույց տանք revenue բառարանի օրինակով:

```
for prod in revenue:
   print(prod)        #տպում է բանալիները
```

Արդյունքը՝

```
TV
Monitor
```

Mouse
refrigerator
iron
laptop

Արտածենք revenue բառարանի բանալիներին համապատասխանող արժեքները.

```
for prod in revenue:
   print(revenue[prod])  #տպում է բանալիների արժեքները
```

Արդյունքը՝

20000
15000
1200
30000
560
17000

Բանալիների ու արժեքների ցուցակ (List of keys and values): Բառարաններն ունեն մեթոդներ, որոնցով կարելի է պատրաստել բանալիների և համապատասխան արժեքների ցուցակներ:

Դիցուք ունենք հետևյալ բառարանը՝ d={'A':1, 'B':2, 'V':5, 'X':10, 'C':100, 'M':1000}

Մեթոդը	Նկարագրությունը	Արդյունքը
list(d)	Վերադարձնում է d բառարանի բանալիներից կազմված ցուցակ:	['A', 'B', 'V', 'X', 'C', 'M']
list(d.values())	Վերադարձնում է d բառարանի արժեքներից կազմված ցուցակ:	[1, 2, 5, 10, 100, 1000]
list(d.items())	Վերադարձնում է d բառարանի (բանալի, արժեք) զույգերից կազմված ցուցակ:	[('A', 1), ('B', 2), ('V', 5), ('X', 10), ('C', 100), ('M', 1000)]

Խնդիր: Արտածել տրված d բառարանի այն բանալիների ցուցակը, որոնց արժեքը հավասար է 10:

```
d={'A':1, 'B':20, 'V':15, 'X':10, 'C':100, 'M':10}
L=[x[0] for x in d.items() if x[1]==10]
print(L)
```

Արդյունքը՝ ['X', 'M']:

Բառարան կարելի է ստեղծել նաև մեկ այլ եղանակով՝ dict ֆունկցիայի միջոցով: Օրինակ՝

```
d=dict([('A', 1), ('B', 2), ('V', 5), ('X', 10), ('C', 100), ('M', 1000)])
```

Այս հրամանով ստեղծվում է հետևյալ բառարանը՝

```
d={'A':1, 'B':2, 'V':5, 'X':10, 'C':100, 'M':1000}
```

Խնդիր: Տրված է բառերի ցուցակ: Ստեղծել և արտածել բառարան, որում բանալիները տրված բառերն են, իսկ արժեքները՝ այդ բառերի երկարությունները: Արտածել նաև (բանալի, արժեք) զույգերի ցուցակը:

```
L=['Python','object','programming','dictionary','lists']
dict={w:len(w) for w in L}
print(dict)
L=list(dict.items())
print(L)
```

Ծրագրի կատարման արդյունքն է՝

```
{'Python': 6, 'object': 6, 'programming': 11, 'dictionary': 10, 'lists': 5}
[('Python', 6), ('object', 6), ('programming', 11), ('dictionary', 10), ('lists', 5)]
```

ԳԼՈՒԽ 11. ՖՈՒՆԿՑԻԱՆԵՐ

11.1. Ֆունկցիայի սահմանումը: Արգումենտներ: Վերադարձվող արժեքներ

Ծրագրի ծավալի մեծացմանը զուգընթաց դժվարանում է դրա ընթերցումը, ըմբռնումը, սխալների հայտնաբերումն ու կարգաբերումը: Ֆունկցիաները կիրառվում են ալգորիթմի առանձին հատվածները որպես ինքնուրույն ծրագրային միավորներ ձևակերպելու համար: Ֆունկցիաների կիրառումը հնարավորություն է տալիս ծրագիրը դիտելու որպես փոխկապակցված ինքնուրույն ծրագրային միավորների ամբողջություն, որը հեշտացնում է ծրագրի կարդալը, սխալների ուղղումը, թույլ է տալիս խնայել հիշողությունը, քանի որ յուրաքանչյուր ֆունկցիա սահմանվում է մեկ անգամ, սակայն կարող է բազմիցս կանչվել և կատարվել: Ֆունկցիայի սահմանումն ունի հետևյալ տեսքը՝

```
def ֆունկցիայի անուն (պարամետրեր):
    ֆունկցիայի մարմին
return արտահայտություն
```

Ֆունկցիայի սահմանումը սկսվում է def հրամանով, որին հաջորդում է ֆունկցիայի անունը: def հրամանը ստեղծում է նոր ֆունկցիա-օբյեկտ և կապում է այն ֆունկցիայի անվան հետ: Ինչպես ցանկացած այլ վերագրման գործողության դեպքում, ֆունկցիայի անունը դառնում է ֆունկցիա-օբյեկտի հղում:

Ֆունկցիայի անվանը հաջորդող կլոր փակագծերի մեջ գրվում են ֆունկցիայի մուտքային պարամետրերը, հետո դրվում է վերջակետ: Պարամետրերի միջոցով ֆունկցիային փոխանցվում են տվյալներ: Այնուհետև խորությամբ գրվում է ֆունկցիայի մարմինը՝ (function body): return հրամանով ֆունկցիան վերադարձնում է արտահայտության արժեքը: Նշենք, որ ֆունկցիան հայտարարելիս մուտքային պարամետրերը և return հրամանը պարտադիր չեն:

Սահմանենք հետևյալ ֆունկցիան՝

```
def function_1():
  print('Python')
```

Ձևակերպված ֆունկցիայի անունն է function_1: Այն տպում է 'Python' բառը: Ծրագրի ցանկացած մասից կարելի է դիմել function_1() ֆունկցիային, և յուրաքանչյուր կանչով այն կտպի 'Python' բառը: Օրինակ՝

```
function_1()
print('Functions are defined with the def statement.')
function_1()
```

Ծրագրի այս հատվածում function_1() ֆունկցիային դիմել ենք երկու անգամ, և յուրաքանչյուր անգամ տպվել է 'Python' բառը: Բացի դրանից՝ տպվել է նաև 'Functions are defined with the def statement' տեքստը: Ծրագրի կատարման արդյունքն է՝

```
Python
Functions are defined with the def statement.
Python
```

Ֆունկցիայի կանչը: Սահմանված ֆունկցիան աշխատեցնելու համար պետք է այն կանչել՝ մուտքային պարամետրերին փոխանցելով համապատասխան արժեքներ: Ֆունկցիայի կանչի ժամանակ փոխանցվող արժեքները կոչվում են ֆունկցիայի փաստացի պարամետրեր կամ արգումենտներ:

Ֆունկցիայի կանչն ունի հետևյալ տեսքը՝

Ֆունկցիայի անուն(արգումենտների ցուցակ)

Ֆունկցիային կարելի է դիմել հետևյալ տիպի արգումենտների փոխանցումով՝

- դիրքային արգումենտներ (positional arguments),
- լռելյայն արգումենտներ (default arguments),
- բանալի արգումենտներ (keyword arguments),
- փոփոխական քանակով արգումենտներ (variable-length arguments):

Դիրքային արգումենտներ: Դիրքային արգումենտների քանակն ու հերթականությունը ֆունկցիայի կանչի ժամանակ պետք է ճշգրտորեն համընկնեն մուտքային պարամետրերի քանակի ու հերթականության հետ:

Վերևում ներկայացված ծրագրում function_1() ֆունկցիան 'Python' բառը տպում էր ընդամենը մեկ անգամ: Այժմ ֆունկցիայի ձևակերպման մեջ ավելացնենք n պարամետրը, որը ցույց է տալիս, թե քանի անգամ է պետք տպել 'Python' բառը:

```
def function_2(n):
   print('Python'*n)
```

Դիմենք ֆունկցիային ծրագրից.

```
function_2(3)
function_2(4)
times=6
function_2(times)
```

Ծրագրի առաջին տողում n պարամետրին փոխանցվել է 3 արժեքը, և 'Python' բառը տպվել է երեք անգամ, երկրորդ տողում՝ չորս անգամ: Երրորդ տողում times փոփոխականին վերագրվել է 6 արժեքը, և այն փոխանցվել է n պարամետրին ու 'Python' բառը տպվել է վեց անգամ: Արդյունքը՝

```
PythonPythonPython
PythonPythonPythonPython
PythonPythonPythonPythonPythonPython
```

function_1() և function_2() ֆունկցիաները տպում էին 'Python' բառը: Այժմ սահմանենք երրորդ՝ function_3() ֆունկցիան, որն արգումենտում ստանում է ցանկացած str տողն ու n թիվը, և տպում է str տեքստը n անգամ:

```
def function_3(str,n):
   print(str*n)
```

Դիտարկենք ծրագրի հետևյալ հատվածը.

```
function_3('Hello World!',3)
```

```
word='program'
times=4
function_3(word,times)
```

function_3() ֆունկցիայի առաջին կանչի ժամանակ 'Hello World!' տեքստը կտպվի երեք անգամ, իսկ երկրորդ կանչով՝ program բառը չորս անգամ:

Եթե դիրքային արգումենտների քանակը լինի մուտքային պարամետրերի քանակից քիչ կամ շատ, ապա կծագի սխալ և կտրվի հաղորդագրություն սխալի մասին: Օրինակ՝ հետևյալ my_func() ֆունկցիան ունի երկու մուտքային պարամետր:

```
def my_func(name, surname):
  print(name + " " + surname)
```

Դիմենք my_func() ֆունկցիային՝ փոխանցելով ընդամենը մեկ արգումենտ՝

```
my_func("Mary")
```

Արդյունքում կարտածվի հաղորդագրություն ծագած սխալի մասին՝

```
Traceback (most recent call last):
  File "C:/Examples/func_1.py", line 3, in <module>
    my_func("Mary")
TypeError: my_func() missing 1 required positional argument: 'surname'
```

Վերադարձվող արժեքներ: Ֆունկցիան կարող է վերադարձնել արժեք return հրամանով:

Խնդիր: Գրել ֆունկցիա, որը հաշվում է a թվի r տոկոսը:

```
def function_rate(a,r):
    return(a*r/100)
```

Կանչենք ֆունկցիան և հաշվենք 90 թվի 5 տոկոսը.

```
p=function_rate(90,5)
print(p)
```

Արդյունքում ֆունկցիան կվերադարձնի 4.5 արժեքը:

Ֆունկցիան կարող է վերադարձնել մեկից ավելի արժեքներ: Այդ դեպքում արժեքները վերադարձվում են ցուցակով, դրանք վերցվում են քառակուսի փակագծերի մեջ՝ [] և իրարից անջատվում են ստորակետով:

Խնդիր: Գրել ֆունկցիա, որը լուծում է $ax^2 + bx = c = 0$ $(a \neq 0)$ քառակուսի հավասարումը:

```
from math import sqrt
def sq_equation(a,b,c):
    d=b**2-4*a*c
    if d<0:
        return('The equation has no solution')
    elif d==0:
        x1=-b/(2*a)
        return(x1)
    else:
        x1=(-b-sqrt(d))/(2*a)
        x2=(-b+sqrt(d))/(2*a)
        return[x1,x2]
```

Դիմենք ֆունկցիային.

```
print(sq_equation(1,-5,6))   #վերադարձնում է երկու արմատ՝ [2.0, 3.0]
print(sq_equation(1,1,1))  #The equation has no solution
print(sq_equation(1,-6,9)) #վերադարձնում է մեկ արմատ՝ 3.0
```

Լռելյայն արգումենտներ: Python լեզուն հնարավորություն է տալիս ֆունկցիայի պարամետրերի համար սահմանել լռելյայն արժեքներ և ֆունկցիայի կանչի ժամանակ դրանց չփոխանցել արժեքներ: Այդ դեպքում օգտագործվում է պարամետրի լռելյայն հայտարարված արժեքը: Ֆունկցիայի սահմանման ժամանակ լռելյայն պարամետրերը գրվում են պարամետրերի ցուցակի վերջում՝ բոլոր ոչ լռելյայն պարամետրերից հետո:

Դիտարկենք հետևյալ ֆունկցիան.

```
def mprint(str, n=1):
   print(str*n)
```

Այս ֆունկցիայում n պարամետրի լռելյայն արժեքը հավասար է մեկի: Դիմենք ֆունկցիային.

```
mprint('abc', 3)
mprint('abc')
```

Ֆունկցիային առաջին անգամ դիմելիս n պարամետրը ստանում է 3 արժեքը, և 'abc' տողը տպվում է երեք անգամ: Երկրորդ կանչի ժամանակ, քանի որ n պարամետրին արժեք չի փոխանցվում, այն ստանում է լռելյայն հայտարարված 1 արժեքը, և 'abc' տողը տպվում է ընդամենը մեկ անգամ: Արդյունքում՝

```
abcabcabc
abc
```

Բանալի արգումենտներ: Ֆունկցիայի սահմանման մեջ պարամետրերն ունեն տրված հերթականություն, որը հաշվի է առնվում ֆունկցիայի կանչի ժամանակ: Բանալի արգումենտները ֆունկցիայի կանչի ժամանակ նույնականացվում են պարամետրերի անվամբ, որը թույլ է տալիս արժեքները փոխանցել ցանկացած հերթականությամբ:

Օրինակ՝ սահմանենք հետևյալ report() ֆունկցիան՝

```
def report(team, manager, project, participants):
   print('team:', team, '\n', 'manager name:', manager,' \n', 'project:', project, '\n', 'participants:', participants)
```

Այս ֆունկցիայի պարամետրերն են. team՝ թիմի անունը, manager՝ թիմի կառավարիչը, project՝ նախագծի վերնագիրը, participants՝ մասնակիցների քանակը: Ֆունկցիային կարելի է դիմել բանալի արգումենտների կամայական հերթականությամբ՝

դրանց վերագրելով արժեքներ վերագրման օպերատորի միջոցով։ Դիմենք ֆունկցիային հետևյալ հրամանով՝

report(participants=10, project = 'House Design', manager = 'Mary', team='Creo')

Աշխատելուց հետո ֆունկցիան կվերադարձնի՝

team: Creo
manager name: Mary
project: House Design
participants: 10

Փոփոխական քանակով արգումենտներ։ Ինչպես արդեն նշել ենք, ֆունկցիայի կանչի ժամանակ արգումենտների քանակը պետք է համընկնի մուտքային պարամետրերի քանակի հետ։ Սակայն Python-ը հնարավորություն է տալիս ֆունկցիային փոխանցել փոփոխական քանակով դիրքային արգումենտներ։ Դրա համար ֆունկցիայի սահմանման մեջ օգտագործվում է tuple տիպի փոփոխական, որից առաջ դրվում է «*» նշանը։ Հավաքածուն իր մեջ պահելու է ոչ բանալի բառով արգումենտներ։ Այն դատարկ է, եթե լրացուցիչ արգումենտներ չեն փոխանցվել ֆունկցիայի կանչի մեջ։

Հետևյալ օրինակում սահմանվել է Sum_Param անունով ֆունկցիան, որի արգումենտների քանակը փոփոխվող է։ Ֆունկցիան հաշվում և վերադարձնում է իր արգումենտների արժեքների գումարը։

```
def Sum_Param(*vartuple):
    print('len(vartuple)=', len(vartuple))
    sum=0
    for i in range(len(vartuple)):
        print('vartuple[',i,']=',vartuple[i])
        sum+=vartuple[i]
    print('Sum of elements =',sum)
    return
Sum_Param(10, 20, 30)
```

Ծրագրի կատարման արդյունքն է՝

```
len(vartuple)= 3
vartuple[ 0 ]= 10
vartuple[ 1 ]= 20
vartuple[ 2 ]= 30
Sum of elements = 60
```

11.2. Փոփոխականների տեսանելիության տիրույթը: Լոկալ ու գլոբալ փոփոխականներ

Ծրագրում բոլոր փոփոխականներն ունեն գործողության իրենց տիրույթը: Փոփոխականի գործողության տիրույթը որոշում է ծրագրի այն մասը, որտեղ կարող է օգտագործվել տվյալ փոփոխականը: Ըստ գործողության տիրույթի՝ առանձնացվում են լոկալ և գլոբալ փոփոխականներ: Փոփոխականները, որոնք սահմանված են ֆունկցիայի մարմնի մեջ, ունեն լոկալ տեսանելիության տիրույթ և կոչվում են լոկալ փոփոխականներ, իսկ ֆունկցիայից դուրս սահմանված փոփոխականներն ունեն գլոբալ տեսանելիության տիրույթ և կոչվում են գլոբալ փոփոխականներ:

Ֆունկցիայի ներսում սահմանված փոփոխականը լոկալ փոփոխական է, այն գործում է միայն տվյալ ֆունկցիայի սահմաններում և գոյություն չունի այդ ֆունկցիայից դուրս:

Դիտարկենք հետևյալ ծրագիրը, որտեղ սահմանված են երկու ֆունկցիաներ՝ func1() և func2(), որոնցից յուրաքանչյուրի ներսում սահմանված է i լոկալ փոփոխականը:

```
def func1():
    for i in range(10):
        print(i, end=' ')
def func2():
    i=100
    func1()
    print()
    print(i)
func2()
```

Ծրագրի վերջին տողում կանչվել է func2() ֆունկցիան։ func2() ֆունկցիայում i փոփոխականին վերագրվել է 100 արժեքը, որից հետո կանչվել է func1() ֆունկցիան, որտեղ ևս կա i անունով փոփոխական։ Բայց, քանի որ երկու ֆունկցիաներում սահմանված յուրաքանչյուր i փոփոխական լոկալ է, ուստի երկրորդ ֆունկցիայի i=100 արժեքը գործում է միայն այդ ֆունկցիայի ներսում, իսկ առաջին ֆունկցիայում i փոփոխականը ստանում է 0-ից մինչև 9 արժեքները, որոնք էլ արտածվում են։ Վերջում տպվում է i փոփոխականի արժեքը։ Եվ, քանի որ գտնվում ենք երկրորդ ֆունկցիայի ներսում, i փոփոխականի արտածվող արժեքը հավասար է 100։ Արդյունքում ստացվում է՝

```
0 1 2 3 4 5 6 7 8 9
100
```

Գլոբալ փոփոխականները հասանելի են տարբեր ֆունկցիաների համար։ Դրանք նկարագրվում են global հրամանով։ Դիտարկենք հետևյալ ծրագիրը.

```
def f1():
    global x
    x= 0

def f2():
    print(x)

x=30
f1()
f2()
```

Ծրագրում x փոփոխականը հայտարարված է որպես գլոբալ փոփոխական։ x=30 վերագրման օպերատորով x փոփոխականը ստացել է 30 արժեքը, որից հետո դիմել ենք f1() ֆունկցիային։ Այստեղ x փոփոխականի արժեքը փոխվում է, փոփոխականին վերագրվում է 0 արժեքը։ Քանի որ այն գլոբալ փոփոխական է, ուրեմն այդ արժեքը փոխվում է բոլոր ֆունկցիաներում՝ և՛ f1()-ում, և՛ f2()-ում։ Այդ պատճառով էլ f2() ֆունկցիային դիմելուց հետո x փոփոխականի արժեքը կտպվի 0։

11.3. Ռեկուրսիվ ֆունկցիաներ

Ռեկուրսիվ ֆունկցիան այնպիսի ֆունկցիա է, որը խնդիրը լուծելու համար կանչում է ինքն իրեն՝ այն բաժանելով ավելի փոքր, պարզ ենթախնդիրների:

Ռեկուրսիայի ստանդարտ օրինակներից է ֆակտորիալ ֆունկցիան: n թվի ֆակտորիալը 1-ից մինչև n բոլոր բնական թվերի արտադրյալն է՝ $n! = 1 \times 2 \times ... \times n$, $0! = 1$: Գրենք n բնական թվի ֆակտորիալը հաշվող fact() ֆունկցիան՝ ցիկլի for հրամանի միջոցով:

```
def fact(n):
    nf=1
    for i in range(2,n+1):
        nf*=i
    return nf
```

Ռեկուրսիան ենթադրում է ֆունկցիայի սահմանում ինքն իր միջոցով: Օրինակ՝ ֆակտորիալ ֆունկցիան կարող է սահմանվել ինքն իր միջոցով՝ $n! = n(n-1)!$

Խնդիր: Գրել ռեկուրսիվ ֆունկցիա, որը հաշվում է տրված n թվի ֆակտորիալը:

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)

print(fact(3))
```

Երեք թվի ֆակտորիալը հաշվելու համար fact(3)-ը դիմում է fact(2)-ին, fact(2)-ը՝ fact(1)-ին, fact(1)-ը՝ fact(0)-ին: Այնուհետև fact(0)-ն վերադարձնում է 1 արժեքը, և այդ արժեքը փոխանցում fact(1)-ին, fact(1)-ը վերադարձնում է 1 արժեքը, և այդ արժեքը փոխանցում fact(2)-ին, fact(2)-ը վերադարձնում է 2 արժեքը, և այդ

արժեքը փոխանցում fact(3)-ին, և, վերջապես, fact(3)-ը վերադարձնում է 6 արժեքը։

Խնդիր։ Վիճակախաղի տոմսի համարը կազմված է երեք թվից՝ 0-ից մինչև 99-ը ներառյալ։ Հաշվել վիճակախաղի հաղթող համարը գուշակելու հավանականությունը։

Վիճակախաղի հաղթող համարը գուշակելու հավանականությունը կհաշվենք հետևյալ բանաձևով՝

$p = \frac{1}{A_n^k}$, որտեղ A_n^k-ն n տարր պարունակող բազմության k-կարգավորությունների քանակն է և հաշվվում է հետևյալ բանաձևով՝

$$A_n^k = \frac{n!}{(n-k)!}$$

Խնդրում n=100, k=3: Ծրագրում ֆակտորիալը հաշվելու համար կդիմենք նախորդ խնդրի fact() ֆունկցիային։

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)

n=100; k=3
print('The probability of guessing a winning lottery ticket: ', 1/(fact(n)/fact(n-k)))
```

Վիճակախաղի շահող համարը գուշակելու հավանականությունը հավասար է՝

The probability of guessing a winning lottery ticket: 1.0307153164296022e-06

Խնդիր։ Գրել ռեկուրսիվ ֆունկցիա, որը գտնում է թվի պարզ բաժանարարները։

Ստորև սահմանված prime_div() ֆունկցիան ունի երկու պարամետր՝ number բնական թիվը և L ցուցակը, որը նախապես դատարկ է։ Այն պարունակելու է պարզ բաժանարարների ցուցակը։ Ֆունկցիան սահմանվում է ինքն իր միջոցով։ Յուրաքանչյուր ան-

գամ, երբ ֆունկցիան գտնում է որևէ i պարզ բաժանարար, այն կցում է L ցուցակին, որն արդեն պարունակում է նախորդ քայլերում գտնված բոլոր պարզ բաժանարարները: Այնուհետև ֆունկցիան number թիվը բաժանում է գտնված i պարզ բաժանարարի վրա և դիմում է ինքն իրեն՝ prime_div (number//i): Այն դեպքում, երբ ֆունկցիան չի գտնում որևէ պարզ բաժանարար, number թիվը կցվում է ցուցակին, քանի որ այդ դեպքում number-ը պարզ թիվ է:

```
def prime_div(number, L=[]):
  for i in range(2,number//2+1):
     if number%i==0:
        return L+[i]+ prime_div (number//i)
  return L+[number]
print(prime_div (45))
```

Ստացված պարզ բաժանարարների ցուցակն է՝ [3, 3, 5]

11.4. Անանուն ֆունկցիաներ: lambda ֆունկցիաներ

lambda ֆունկցիաները անանուն ֆունկցիաներ են: lambda ֆունկցիան կարող է ունենալ ցանկացած քանակությամբ արգումենտներ, բայց՝ միայն մեկ արտահայտություն: Մեկ արտահայտությամբ սահմանափակման պատճառով lambda ֆունկցիաները չունեն return հրաման: Ֆունկցիան հաշվում է արտահայտության արժեքը և վերադարձնում է այն: Շարահյուսությունը՝

lambda արգումենտներ: արտահայտություն

Հետևյալ lambda ֆունկցիան ստուգում է՝ թիվը դրակա՞ն է, բացասակա՞ն, թե՞ զրո:

```
n = lambda x: "Positive" if x > 0 else "Negative" if x < 0 else "Zero"

print(n(5))
print(n(-3))
print(n(0))
```

Արդյունքը՝

```
Positive
Negative
Zero
```

Իսկ հետևյալ lambda ֆունկցիան ստուգում է՝ թիվը զո՞յգ է, թե՞ կենտ։

```
check = lambda x: "Even" if x % 2 == 0 else "Odd"
print(check(4))
print(check(7))
```

Արդյունքը՝

```
Even
Odd
```

Հետևյալ lambda ֆունկցիան ունի երկու արգումենտ։ Այն բազմապատկում է a արգումենտը b արգումենտով և վերադարձնում արդյունքը։

```
x = lambda a, b : a * b
print(x(5, 6))
```

Արդյունքը՝ 30։

lambda ֆունկցիան կարելի է օգտագործել որպես անանուն ֆունկցիա մեկ այլ ֆունկցիայի ներսում։ Հետևյալ օրինակում սահմանվում է myfunc() ֆունկցիան, որն ընդունում է մեկ արգումենտ, և այդ արգումենտը բազմապատկվում է որևէ թվով։

```
def myfunc(n):
  return lambda a : a * n
m2 = myfunc(2)
m3 = myfunc(3)

print(m2(11))
print(m3(11))
```

Արդյունքը՝

```
22
33
```

Խնդիր: Ստանալ տրված [-2, 1, 0, -1, 2] ցուցակի դրական տարրերից կազմված ցուցակը: Ծրագրում lambda ֆունկցիան օգտագործվել է որպես անանուն ֆունկցիա filter() ֆունկցիայի ներսում:

```
numbers = [-2, 1, 0, -1, 2]
positive_numbers = filter(lambda n: n > 0, numbers)
print(list(positive_numbers))
```

Արդյունքը՝

```
[1, 2]
```

11.5. filter(), zip(), map() ֆունկցիաները

Python լեզվի filter(), zip(), map() ներկառուցված ֆունկցիաները նախատեսված են iterable տիպի օբյեկտների հետ աշխատելու համար: iterable տիպի օբյեկտը կարող է վերադարձնել իր տարրերը մեկ առ մեկ, ինչը թույլ է տալիս ցիկլի հրամանների միջոցով անցնել այդպիսի օբյեկտի տարրերի վրայով: Python լեզվում iterable տիպի ներկառուցված օբյեկտներ են հաջորդականությունները (տողերը, ցուցակները, հավաքածուները), բազմությունները, բառարանները և ֆայլային օբյեկտները:

filter() ֆունկցիան: Python լեզվում ներկառուցված filter() ֆունկցիան օգտագործվում է iterable տիպի օբյեկտից տրված պայմանին բավարարող տարրերը ստանալու համար: Շարահյուսությունը հետևյալն է՝

filter(function, iterable տիպի օբյեկտ):

function() ֆունկցիան կիրառվում է օբյեկտի տարրերի վրա և վերադարձնում է True, եթե տարրը բավարարում է տրված պայմանին, և False՝ հակառակ դեպքում: filter() ֆունկցիան պահում է

օբյեկտի միայն այն տարրերը, որոնց համար function() ֆունկցիայի արժեքը հավասար է True:

Խնդիր: Տրված թվային ցուցակից առանձնացնել այն թվերը, որոնց վերջին թվանշանը 4 է:

```
def ends_with4(n):
    return n % 10 == 4

lst= [112, 208, 35, 4, 51, 60]
b = filter(ends_with4, lst)
print(list(b)) #b ֆիլտր-օբյեկտը փոխակերպվում է ցուցակի՝
[114, 4, 64]
```

Ներկայացված ծրագրում ends_with4() ֆունկցիան ստուգում է, թե արդյո՞ք թվի վերջին թվանշանը չորս է: Ֆունկցիան վերադարձնում է True, եթե թիվը բավարարում է այդ պայմանին, և False՝ հակառակ դեպքում: filter() ֆունկցիան կիրառում է ends_with4() ֆունկցիան տրված lst ցուցակի յուրաքանչյուր տարրի վրա: Արդյունքում b ցուցակում ներառվում են միայն այն թվերը, որոնց վերջին թվանշանը 4 է:

Խնդիր: Տողային տարրեր պարունակող ցուցակից արտածել այն տարրերը, որոնց երկարությունը փոքր չէ վեցից:

```
lst = ["Python", "Assembler", "Fortran", "Pascal", "Java"]

def len_gr6(s):
  if len(s) >=6:
    return True
  else:
    return False

lst_fltr = filter(len_gr6, lst)
for x in lst_fltr:
  print(x)
```

Ծրագրի կատարման արդյունքն է՝

Python
Assembler
Fortran
Pascal

Խնդիր։ Արտածել [1,100] միջակայքի պարզ թվերի ցուցակը։

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, n//2+1):
        if n % i == 0:
            return False
            break
    return True

print(list(filter(is_prime,range(1,101))))
```

zip() ֆունկցիան։ Python լեզվում ներկառուցված zip() ֆունկցիան օգտագործվում է iterable տիպի տարբեր օբյեկտների նույն ինդեքսով կամ նույն դիրքն ունեցող տարրերը առանձին հավաքածուների (tuples) մեջ խմբավորելու համար։ zip() ֆունկցիայի շարահյուսությունն է՝

```
zip(*iterables),
```

որտեղ *iterables-ը մեկ կամ մի քանի iterable տիպի օբյեկտներ են։

zip() ֆունկցիան վերադարձնում է tuple-ների ցուցակ, որոնցից յուրաքանչյուրում խմբավորված են մուտքային օբյեկտների նույն ինդեքսով համապատասխան տարրերը։

Եթե պարամետրեր չեն փոխանցվում, zip() ֆունկցիան վերադարձնում է դատարկ ցուցակ։ Եթե փոխանցվում է միայն մեկ պարամետր, ապա արդյունքը մեկական տարր պարունակող tuple-ների ցուցակ է։ Եթե փոխանցվում են մի քանի պարամետրեր, ապա յուրաքանչյուր tuple պարունակում է մեկական տարր յուրաքանչյուր մուտքային հաջորդականությունից (նույն ինդեքսով)։ Օրինակ՝

```
lst1 = [1, 2, 3]
lst2 = ['a', 'b', 'c']

z0 = zip() # պարամետր չի փոխանցվում
print(list(z0)) # արդյունքը՝ []

z1 = zip(lst1) # փոխանցվում է մեկ պարամետր
print(list(z1)) # արդյունքը՝ [(1,), (2,), (3,)]

z2 = zip(lst1, lst2) # փոխանցվում է երկու պարամետր
print(list(z2)) # արդյունքը՝ [(1, 'a'), (2, 'b'), (3, 'c')]
```

z0 ցուցակը դատարկ է, քանի որ zip() ֆունկցիային պարամետրեր չեն փոխանցվել, z1-ը մեկական տարր պարունակող tuple-ների ցուցակ է, իսկ z2-ը՝ երկուական տարր պարունակող tuple-ների ցուցակ, քանի որ zip() ֆունկցիային փոխանցվել են երկու պարամետր՝ zip(lst1, lst2)։

Եթե zip() ֆունկցիային փոխանցված օբյեկտներն ունեն տարբեր երկարություններ, ապա արդյունքում ստացված ցուցակի երկարությունը հավասար է մուտքային հաջորդականությունների երկարություններից ամենափոքր արժեքին։ Օրինակ՝

```
products = ['TV', 'Monitor', 'Phone']
prices = [500, 300]
product_prices = zip(products, prices)
print(list(product_prices))
```

Արդյունքը՝

```
[('TV', 500), ('Monitor', 300)]
```

Ներկայացված ծրագրում products ցուցակն ունի երեք տարր, իսկ prices ցուցակը՝ երկու (բացակայում է Phone ապրանքի գինը), հետևաբար zip() ֆունկցիան վերադարձնում է միայն երկու հավաքածուից կազմված ցուցակ։

zip() ֆունկցիայով կարելի է կատարել նաև հակառակ գործողությունը։ Եթե տրված է հավաքածուների ցուցակ, ապա կարելի

է այն տրոհել առանձին iterable օբյեկտների՝ օգտագործելով * օպերատորը: Օրինակ՝

```
lst = [('Ann', 22), ('Mary', 18), ('Mane', 31)]
Names, Ages = zip(*lst)
print("Names:", Names)
print("Ages:", Ages)
```

Արդյունքը՝

```
Names: ('Ann', 'Mary', 'Mane')
Ages: (22, 18, 31)
```

Ծրագրում zip() ֆունկցիան խմբավորել է հավաքածուների առաջին տարրերը Names օբյեկտում, իսկ երկրորդ տարրերը՝ Ages օբյեկտում:

zip() ֆունկցիան կարող է օգտագործվել նաև բառարանի բանալի-արժեք զույգերից ցուցակ պատրաստելու համար: Օրինակ՝

```
d = {'Names': 'Ann', 'Age': 22, 'Score': '60'}
keys = d.keys()
print(keys)
values = d.values()
print(values)
res = zip(keys, values)
print(list(res))
```

Ծրագրի կատարման արդյունք՝

```
dict_keys(['Names', 'Age', 'Score'])
dict_values(['Ann', 22, '60'])
[('Names', 'Ann'), ('Age', 22), ('Score', '60')]
```

map() ֆունկցիան: Python լեզվում ներկառուցված map() ֆունկցիան նախատեսված է ցուցակների, հավաքածուների, տողերի, բազմությունների, բառարանների և այլ iterable տիպի օբյեկտների հետ աշխատելու համար: map() ֆունկցիայի շարահյուսությունը հետևյալն է.

map(function, iterable1, iterable2, ...),

որտեղ function-ը հղում է function() ֆունկցիային:

map() ֆունկցիան կանչում է function() ֆունկցիան, որը կիրառվում է iterable1, iterable2... օբյեկտների նույն դիրքն ունեցող տարրերի համար: Օրինակ՝ երկու օբյեկտի դեպքում ֆունկցիան վերցնում է դրանց առաջին տարրերը, ապա՝ երկրորդ տարրերը և այլն: map() ֆունկցիան վերադարձնում է map-օբյեկտ:

function() ֆունկցիայի արգումենտների քանակը պետք է համընկնի map() ֆունկցիայի iterable օբյեկտների քանակի հետ: Հակառակ դեպքում, կծագի TypeError տիպի սխալ, որի մասին կտրվի համապատասխան հաղորդագրություն:

Հետևյալ ծրագրում map() ֆունկցիան կիրառվել է երկու ցուցակների համապատասխան տարրերը գումարելու համար:

```
num1 = [100, 200, 300]
num2 = [10, 20, 30]

def add_nums(a, b):
   return a + b

summa = list(map(add_nums, num1, num2))
print(summa)

summa = list(map(add_nums, num1))
print(summa)
```

Պատասխան՝

```
[110, 220, 330]
Traceback (most recent call last):
  File "C:/Examples/map1.py", line 10, in <module>
    summa = list(map(add_nums, num1))
TypeError: add_nums() missing 1 required positional argument: 'b'
```

Ծրագրում summa=list(map(add_nums, num1, num2)) հրամանով add_nums() ֆունկցիան գումարել է num1 և num2 երկու ցուցակների համապատասխան տարրերը, map() ֆունկցիան ար-

դյունքը վերադարձրել է որպես map օբյեկտ, որից հետո list() ֆունկցիան ստացված map օբյեկտը փոխակերպել է ցուցակի։

summa=list(map(add_nums, num1)) հրամանում add_nums() ֆունկցիայի արգումենտների թվում բացակայել է մեկ դիրքային արգումենտ, որի պատճառով էլ ծագել է TypeError սխալը, և տրվել է այդ մասին հետևյալ հաղորդագրությունը՝ TypeError: add_nums() missing 1 required positional argument: 'b'

Խնդիր։ Տրված է տողային տարրեր պարունակող ցուցակ։ Կառուցել այդ տարրերի երկարություններից կազմված ցուցակ։

```
words = ["mathematics", "physics", "chemistry", "biology", "astronomy"]
lengths_words = map(len, words)
print(list(lengths_words))
```

Պատասխան՝ [11, 7, 9, 7, 9]

Այս ծրագրում map() ֆունկցիայի առաջին արգումենտը ներկառուցված len() ֆունկցիային հղումն է։ Ֆունկցիան հաջորդաբար հաշվում է բառերի words ցուցակի (երկրորդ արգումենտը) յուրաքանչյուր տարրի երկարությունը։ map() ֆունկցիան ստեղծում է lengths_words օբյեկտը, որը պարունակում է տողային տարրերի երկարությունները։ Այդ օբյեկտը list() ֆունկցիայով փոխակերպվել է ցուցակի։

Նշենք, որ map-օբյեկտը կարող է օգտագործվել միայն մեկ անգամ, առաջին օգտագործումից հետո օբյեկտը դատարկվում է։ Դիտարկենք հետևյալ ծրագիրը՝

```
nums = ['1', '2', '3']
result = map(int, nums) #ստեղծվում է map-օբյեկտ
print('Առաջին մուտք։', list(result))
print('Երկրորդ մուտք։', list(result))
```

Պատասխան՝

Առաջին մուտք։ [1, 2, 3]
Երկրորդ մուտք։ []

Ներկայացված ծրագրում map() ֆունկցիայի առաջին արգումենտը ներկառուցված int() ֆունկցիային հղում է: int() ֆունկցիան nums ցուցակի տարրերը ձևափոխել է ամբողջ թվերի: Ստեղծվել է result օբյեկտը, ապա այն list() ֆունկցիայով փոխակերպվել է ցուցակի, որն էլ արտածվել է: Առաջին օգտագործումից հետո result օբյեկտը դատարկվել է, այդ պատճառով էլ արտածվել է դատարկ ցուցակ:

map-օբյեկտը կրկին օգտագործելու համար անհրաժեշտ է այն հիշել ցուցակում կամ տվյալների այլ համախմբերում: Օրինակ՝

```
nums = ['1', '2', '3']
result = map(int, nums)
nums_list=list(result) #map տիպի օբյեկտը հիշվում է nums_list ցուցակում
print('Առաջին մուտք', nums_list)
print('Երկրորդ մուտք', nums_list)
```

Պատասխան՝

```
Առաջին մուտք [1, 2, 3]
Երկրորդ մուտք [1, 2, 3]
```

map() ֆունկցիան աշխատում է տարբեր տեսակի ֆունկցիաների հետ՝ ներկառուցված, lambda, օգտատիրոջ կողմից սահմանված, ստանդարտ գրադարանային ֆունկցիաներ:

map() ֆունկցիայի կիրառումը Python լեզվի ներկառուցված ֆունկցիաների հետ:

Հետևյալ ծրագրում կիրառվել են round(), len(), str() և abs() ֆունկցիաները՝ թվերը կլորացնելու, տողերի երկարությունները որոշելու, թվերը տողերի փոխակերպելու և թվերի բացարձակ արժեքը ստանալու համար:

```
# թվերի կլորացում
floats = [5.24, 9.78, 7.56]
rounded = map(round, floats)
print(list(rounded))
```

```
# տողերի երկարության որոշում
words = ['Introduction', 'to', 'Python','Programming']
lengths = map(len, words)
print(list(lengths))

# թվերի փոխակերպում տողերի
numbers = [15, 26.1, 37, 48]
strings = map(str, numbers)
print(list(strings))

# թվերի բացարձակ արժեքների հաշվում
numbers = [-1, 2, 3, -4, 0]
absolute = map(abs, numbers)
print(list(absolute))
```

Պատասխան՝

```
[5, 10, 8]
[12, 2, 6, 11]
['15', '26.1', '37', '48']
[1, 2, 3, 4, 0]
```

Հետևյալ օրինակում տրված թվային ցուցակի տարրերը բարձրացվել են խորանարդ՝ map() ֆունկցիայի հետ կիրառելով lambda ֆունկցիա։

```
nums = [1, 2, 3, 4, 5]
cubs = map(lambda x: x ** 3, nums)
print(list(cubs)) [1, 8, 27, 64, 125]
```

map() ֆունկցիան կարող է աշխատել նաև օգտագործողի սահմանած ֆունկցիայի հետ։ Ենթադրենք տրված է ցուցակ, որի տարրերը բառարաններ են։ Դրանցից յուրաքանչյուրը պարունակում է որևէ ապրանքի անունը (name), գինը (price) և մեկ օրվա ընթացքում վաճառված քանակությունը (quantity)։ Հաշվել յուրաքանչյուր ապրանքի վաճառքից ստացված հասույթը։

Ստորև ներկայացված ծրագրում նախ սահմանվել է հասույթը հաշվող profit_product() ֆունկցիան, ապա map() ֆունկցիայով ստացվել է ցուցակ, որը պարունակում է մեկ օրվա ընթացքում բոլոր ապրանքների վաճառքից ստացված համապատասխան հասույթները։

```
def profit_product(product):
    return product['price'] * product['quantity']

# Ապրանքների ցուցակը
product = [
    {"name": "Notebook", "price": 236000, "quantity": 3},
    {"name": "TV", "price": 345000, "quantity": 7},
    {"name": "Monitor", "price": 98000, "quantity": 4}
]

total_profit = map(profit_product, product)
print(list(total_profit)) # պատասխան՝ [708000, 2415000, 392000]
```

map() ֆունկցիան կարող է աշխատել նաև ստանդարտ գրադարանի ֆունկցիաների հետ։ Ներմուծենք math մոդուլը տարբեր մաթեմատիկական հաշվարկներ կատարելու համար։

```
import math
nums = [1, 4, 9, 16]
sqrt_nums = map(math.sqrt, nums) #քառակուսի արմատի հաշվարկ
print(list(sqrt_nums))

num_floats = [3.14, 2.717, 1.49]
# թվի կլորացում դեպի իրենից մեծ ամենամոտ ամբողջ թիվը
ceil_numbers = map(math.ceil, num_floats)
print(list(ceil_numbers))

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

```
squared = map(math.pow, numbers, [3]*len(numbers)) #հաշվում է թվի խորանարդը

print(list(squared))
```

Պատասխան՝

```
[1.0, 2.0, 3.0, 4.0]
[4, 3, 2]
[1.0, 8.0, 27.0, 64.0]
```

ԳԼՈՒԽ 12. ԱՇԽԱՏԱՆՔ ՖԱՅԼԵՐԻ ՀԵՏ

12.1. Ֆայլը բացելն ու փակելը

Python-ն ունի ֆայլերի հետ աշխատելու հնարավորություններ՝ կարդալ ֆայլից, գրել ֆայլի մեջ: Այս հնարավորությունը խիստ կարևորվում է մեծածավալ տվյալների հետ աշխատելիս, նաև այն դեպքերում, երբ միևնույն տվյալները պետք է օգտագործել տարբեր ծրագրերում: Նման դեպքերում ավելի հարմար է տվյալները պահել որևէ ֆայլի մեջ և ծրագրի աշխատանքի ժամանակ բացել ֆայլն ու դրանք կարդալ ֆայլից: Նույն ձևով նպատակահարմար է նաև ծրագրի աշխատանքի արդյունքները պահել ֆայլի մեջ՝ հետագայում օգտագործելու համար: Մենք կանգ կառնենք .txt տեքստային ֆայլերի հետ աշխատանքի վրա, չնայած Python-ի հնարավորությունները դրանով չեն սահմանափակվում: Python-ը կարող է աշխատել նաև այլ ֆորմատներով ֆայլերի հետ՝ CSV, HTML, JSON (CSV - Comma-separated values, HTML - HyperText Markup Language, JSON - JavaScript Object Notation): Ֆայլերի հետ աշխատելու համար Python-ում նախատեսված են մի շարք ֆունկցիաներ և մեթոդներ:

Ֆայլերի հետ աշխատելու հնարավորություններն ուսումնասիրելու համար նախ ստեղծենք textfile_1.txt անունով ֆայլը, որի բովանդակությունը ներկայացված է ստորև.

```
Suppose we have a text file called textfile_1.txt.
We will learn how to work with data stored in text files.
We want to read its contents into Python.
```

Մինչև ֆայլից կարդալը կամ ֆայլում գրելը անհրաժեշտ է այն նախապես բացել՝ Python-ի ներկառուցված open() ֆունկցիայի միջոցով, որի շարահյուսությունը հետևյալն է՝

```
file_object = open(file_name, access_mode)
```

open() ֆունկցիան ստեղծում է file_object ֆայլային օբյեկտը, որով դիմելու է ֆայլին և վերադարձնում է ֆայլի պարունակությունը որպես Python օբյեկտ: Այստեղ՝

- file_name-ը բացվող ֆայլի անունն է / ճանապարհը,
- access_mode-ը այն ռեժիմն է, որով պետք է բացել ֆայլը:

Ֆայլը բացելու համար նախատեսված են տարբեր ռեժիմներ, որոնք նկարագրված են ստորև ներկայացված աղյուսակ 12.1-ում:

Ռեժիմը	**Նկարագրությունը**
r	Ֆայլը բացում է միայն կարդալու համար: Սա լռելյայն արժեք է:
r+	Բացում է ֆայլը կարդալու և գրելու համար: Եթե ֆայլը գոյություն չունի, առաջանում է մուտքի/ելքի սխալ:
rb	Բացում է ֆայլը երկուական ձևաչափով կարդալու համար:
rb+	Բացում է ֆայլը երկուական ձևաչափով կարդալու և գրելու համար:
w	Բացում է ֆայլը միայն գրելու համար: Եթե ֆայլն արդեն գոյություն ունի, այն փոխարինվում է նորով, հակառակ դեպքում ստեղծվում է նոր ֆայլ գրելու համար:
w+	Բացում է ֆայլը գրելու և կարդալու համար: Եթե ֆայլն արդեն գոյություն ունի, այն փոխարինվում է նորով: Եթե ֆայլը գոյություն չունի, ստեղծվում է նոր ֆայլ կարդալու և գրելու համար:
wb	Բացում է ֆայլը միայն գրելու համար երկուական ձևաչափով: Եթե ֆայլն արդեն գոյություն ունի, այն փոխարինվում է նորով, հակառակ դեպքում ստեղծվում է նոր ֆայլ գրելու համար:
wb+	Բացում է ֆայլը գրելու և կարդալու համար երկուական ձևաչափով: Եթե ֆայլն արդեն գոյություն ունի, այն փոխարինում է նորով: Եթե ֆայլը գոյություն չունի, ստեղծվում է նոր ֆայլ կարդալու և գրելու համար:

a	Բացում է ֆայլը վերջից կցելու համար: Եթե ֆայլը գոյություն չունի, ստեղծվում է նոր ֆայլ գրելու համար:
a+	Բացում է ֆայլը կարդալու և վերջից կցելու համար: Եթե ֆայլը գոյություն չունի, ստեղծվում է նոր ֆայլ կարդալու և գրելու համար:
ab	Բացում է ֆայլը վերջից կցելու համար երկուական ձևաչափով: Եթե ֆայլը գոյություն չունի, ստեղծվում է նոր ֆայլ գրելու համար:
ab+	Բացում է ֆայլը կարդալու և վերջից կցելու համար երկուական ձևաչափով: Եթե ֆայլը գոյություն չունի, ստեղծվում է նոր ֆայլ կարդալու և գրելու համար:

Աղյուսակ 12.1. Ֆայլը բացելու ռեժիմները

Դիցուք, textfile_1.txt ֆայլը գտնվում է "C:/Examples/text_files" պանակում: Այդ դեպքում, օգտագործելով open() ֆունկցիան, միայն կարդալու համար ֆայլը կբացենք հետևյալ հրամանով.

```
f=open("C:/Examples/text_files/textfile_1", "r")
```

Եթե "textfile_1.txt" ֆայլը գտնվում է ընթացիկ պանակում, այդ դեպքում ուղին նշելու անհրաժեշտություն չկա.

```
f = open("textfile_1.txt ", "r")
```

Ֆայլն օգտագործելուց հետո պետք է այն փակել close() մեթոդով: close() մեթոդը կատարում է բոլոր փոփոխությունները ֆայլում և փակում այն: Մեթոդի շարահյուսությունն է.

```
file.close()
```

Օրինակ՝հետևյալ ծրագիրը բացում է "textfile_1.txt" ֆայլը կարդալու համար, կարդում է, ապա փակում ֆայլը.

```
f =open("textfile_1.txt", "r")
print(f.read())
f.close()
```

12.2. Ֆայլից կարդալն ու ֆայլում գրելը

Ֆայլից կարդալը: Python-ում .txt ֆայլը կարդալու համար կա երեք եղանակ: Դրանք են.

- read()
- readline()
- readlines()

Տեքստային ֆայլը կարդալու առաջին եղանակը ֆայլի պարունակությունն ամբողջությամբ տողի մեջ ներբեռնելն է read() մեթոդով, որի շարահյուսությունն է՝

```
file_object.read([n])
```

Այստեղ n-ը բացված ֆայլից կարդացվող բայթերի քանակն է, որը ոչ պարտադիր պարամետր է: Ֆունկցիան ֆայլը կարդում է սկզբից և վերադարձնում է կարդացած պարունակությունը մեկ տողով: Եթե n-ը նշված չէ, ֆունկցիան կարդում է ամբողջ ֆայլը: Օրինակ՝

```
s=open('textfile_1.txt').read()
print(s)
```

Այստեղ նախ open() մեթոդով բացվում է textfile_1.txt ֆայլը, ապա read() մեթոդով այն ամբողջությամբ ներբեռնվում է, և արդյունքը վերադարձվում է հետևյալ տողով.

```
'Suppose we have a text file called textfile_1.txt.
We will learn how to work with data stored in text files.
We want to read its contents into Python.'
```

Հետևյալ հրամանով ևս բացվում է textfile_1.txt ֆայլը, սակայն այս անգամ read() մեթոդով ընթերցվում է տասը բայթ՝ n=10.

```
s=open('textfile_1.txt').read(10)
print(s)
```

Արդյունքում վերադարձվում է հետևյալ տողը.

'Suppose we'

Երկրորդ՝ readline() մեթոդը, ֆայլը կարդում է տող առ տող: Այն կարդում է տողը մինչև հաջորդ նոր տողի նիշը: Մեթոդի շարահյուսությունը հետևյալն է.

file_object. readline()

Հետևյալ ծրագրում textfile_1.txt տեքստային ֆայլը նախ բացվում է կարդալու համար, այնուհետև readline() մեթոդով ընթերցվում է առաջին տողը, որից հետո ֆայլը կարդալու ընթացիկ ցուցիչը կանգ է առնում երկրորդ տողի նիշի վրա:

```
s=open('C:/Examples/text_files/textfile_1,'r')
print(s.readline())
```

Արդյունքը՝

'Suppose we have a text file called textfile_1.txt.'

Եթե շարունակենք նույն ֆայլից կարդալ readline() մեթոդով, ապա այն արդեն կվերադարձնի հաջորդ՝ երկրորդ, տողը, և ֆայլը կարդալու ընթացիկ ցուցիչը կանգ կառնի երրորդ տողի նիշի վրա:

```
s=open('C:/Examples/text_files/textfile_1,'r')
print('First call:')
print(s.readline())
print('Second call:')
print(s.readline())
```

Արդյունքը՝

```
First call:
Suppose we have a text file called textfile_1.txt.
Second call:
We will learn how to work with data stored in text files.
```

readlines() մեթոդը վերադարձնում է ֆայլի տողերից կազմված ցուցակ, որտեղ ցուցակի յուրաքանչյուր տարր ներկայացնում է ֆայլի մեկ տող: Հետևյալ օրինակում textfile_1.txt տեքստային ֆայլը բացվում է կարդալու համար, այնուհետև readlines() մեթոդը վերադարձնում է այդ ֆայլի տողերից կազմված ցուցակ:

```
s=open('textfile_1.txt','r')
print(s.readlines())
```

Արդյունքը՝

['We will learn how to work with data stored in text files.\n', 'This is a text file called textfile_1.txt.\n', 'We want to read its contents into Python.\n']

textfile_1.txt տեքստային ֆայլի բովանդակությունը տող առ տող կարելի է ներբեռնել ցուցակի մեջ նաև հետևյալ եղանակով՝

```
lines=[line.strip() for line in open('textfile_1.txt')]
print(lines)
```

Այս ծրագրով նախ textfile_1.txt ֆայլը լռությամբ բացվում է կարդալու համար, ապա strip() մեթոդով հեռացվում են բոլոր տողերի սկզբում և վերջում գտնվող բացակային սիմվոլները և պատրաստվում է lines ցուցակը, որի տարրերը ֆայլի առանձին տողերն են.

['We will learn how to work with data stored in text files.', 'This is a text file called textfile_1.txt.', 'We want to read its contents into Python.']

Նշենք, որ strip() մեթոդը հեռացնում է բացակային սիմվոլները բոլոր տողերի սկզբից և վերջից: Եթե անհրաժեշտ է պահպանել տողերի սկզբի բացակային սիմվոլները, ապա պետք է օգտագործել rstrip() մեթոդը:

Եթե ֆայլը կարդանք նախ read() մեթոդով, ապա՝ readline() մեթոդով, ապա երկրորդ մեթոդը կվերադարձնի դատարկ տող:

Այս դեպքում պետք է ստեղծել նոր ֆայլային օբյեկտ կամ օգտագործել seek() մեթոդը։

Հետևյալ ծրագրում read() մեթոդով ֆայլը կարդալուց հետո readline() մեթոդը կվերադարձնի դատարկ տող։ Օրինակ՝

```
s=open('textfile_1.txt','r')
print('First call - method read():')
print(s.read())
print('Second call - method readline():')
print(s.readline())
```

Տեքստային ֆայլի մեջ գրելը։ Տեքստային ֆայլի մեջ գրելու համար նախատեսված է write() մեթոդը, որի շարահյուսությունը հետևյալն է.

```
fileObject.write(string).
```

որտեղ string պարամետրը բացված ֆայլում գրվող տողն է։ Այս մեթոդը վերցնում է մեկ տողային պարամետր, և այն գրում է ֆայլում։ Ֆայլում նոր տողից սկսելու համար պետք է գրել '\n' սիմվոլը։

Օրինակ՝ հետևյալ ծրագիրը բացում է writefile.txt անունով տեքստային ֆայլը և գրում է դրանում write() մեթոդով։

```
file1=open('writefile.txt','w')
print('There are several ways to write to files.', file=file1)
print('Here is one of those ways.', file=file1)
print('We will write to a file called writefile.txt.', file=file1)
file1.close()
```

Ներկայացված ծրագրի առաջին տողում գրված հրամանով բացվում է writefile.txt ֆայլը։ 'w' արգումենտը նշանակում է, որ ֆայլը բացվում է գրելու համար։ Ստեղծվում է file1 անունով ֆայլային օբյեկտը, որով դիմելու ենք ֆայլին։ Ֆայլում գրելու համար օգտագործվում է file օպցիոնալ արգումենտը, որով տրվում է այն ֆայլը, որում պետք է գրել, այնուհետև տող առ տող գրվում է այդ ֆայլի մեջ։ Գրելն ավարտելուց հետո ֆայլը փակվում է close() հրամանով։

Նշենք, որ եթե writefile.txt ֆայլը նախապես գոյություն ունի, ապա դրա բովանդակությունը կփոխվի։

Դիտարկենք մի քանի խնդիր։

Խնդիր։ Գրել ծրագիր, որը կարդում է radius.txt ֆայլում բերված շրջանագծերի շառավիղների ցուցակը, հաշվում է համապատասխան շրջանների մակերեսները և արդյունքները գրում է square.txt անունով ֆայլի մեջ։

```
import math
file1=open('square.txt','w')
rad=[line.strip() for line in open('radius.txt')]
for r in rad:
   print(math.pi*eval(r)**2, file=file1)
file1.close()
```

Դիցուք, radius.txt ֆայլում գրված են հետևյալ շրջանագծերի շառավիղները.

```
2.5
5
4.5
2.1
3
```

Այդ դեպքում ծրագրի աշխատանքից հետո square.txt ֆայլում կգրվեն տրված շառավիղներով շրջանների համապատասխան մակերեսները.

```
19.625
78.5
63.585
13.8474
28.26
```

Խնդիր։ Գրել ծրագիր, որը ստուգում է աշակերտների գիտելիքները որևէ թեմայի վերաբերյալ։ Հարցերը պահված են questions.txt տեքստային ֆայլում, իսկ պատասխանները՝ answers.txt ֆայլում։ Դա բավականաչափ հարմար է, քանի որ

թույլ է տալիս յուրաքանչյուր ֆայլ խմբագրել առանձին, օրինակ՝ ավելացնել հարցեր, խմբագրել, հեռացնել դրանք, ճշգրտել պատասխանները: Ծրագիրը ստուգման գործընթացի ավարտին ամփոփում և արձանագրում է սովորողների ճիշտ պատասխանների ընդհանուր քանակը:

```
questions = [line.strip() for line in open('questions.txt')]
answers = [line.strip() for line in open('answers.txt')]
num_right = 0

for i in range(len(questions)):
    ans = input(questions[i])
    if ans.lower()==answers[i].lower():
        print('Correct')
        num_right +=1
    else:
        print('Wrong. The answer is', answers[i])
    print('You have', num_right, 'out of', i+1, 'right.')
```

Խնդիր: Տրված է wordlist.txt ֆայլը, որի յուրաքանչյուր տողում գրված է որևէ բառ անգլերենով, ընդ որում, բոլոր բառերն իրարից տարբեր են: Բառերը կարելի է ներբեռնել wordlist ցուցակի մեջ հետևյալ հրամանի միջոցով.

```
wordlist = [line.strip() for line in open('wordlist.txt')]
```

Պահանջվում է.

ա) Տպել բոլոր այն բառերը, որոնց երկարությունը հավասար է հինգի:

```
for word in wordlist:
if len(word)==5:
print(word)
```

Կամ՝

```
print([word for word in wordlist if len(word)==5])
```

բ) Տպել բոլոր այն բառերը, որոնք սկսվում են ‘ch'-ով կամ 'sh'-ով։

```
for word in wordlist:
if word[:2]=='ch' or word[:2]=='sh':
print(word)
```

գ) Հաշվել բոլոր այն բառերի քանակը, որոնք սկսվում են ձայնավորով։

```
count = 0
for word in wordlist:
if word[0] in 'aeiou':
count+=1
print(count)
```

12.3. Ֆայլերի վերանվանում, տեղափոխում ու ջնջում

Python-ի os մոդուլն ունի ֆայլերը կառավարելու այնպիսի հնարավորություններ, ինչպիսիք են, օրինակ, ֆայլերի վերանվանումը, տեղափոխումը, ջնջումը։ os մոդուլն օգտագործելու համար անհրաժեշտ է նախ ներբեռնել այն, ապա կիրառել համապատասխան մեթոդները։

Ֆայլերը վերանվանվում են rename() մեթոդով։ Այն ունի երկու արգումենտ՝ ֆայլի ընթացիկ անունը (current_file_name) և նոր անունը (new_file_name)։ Մեթոդի շարահյուսությունը հետևյալն է՝

```
os.rename(current_file_name, new_file_name)
```

Հետևյալ ծրագիրը գոյություն ունեցող test1.txt ֆայլը վերանվանում է test2.txt։

```
import os
os.rename( "test1.txt", "test2.txt" )
```

Ֆայլը ջնջվում է os.remove() մեթոդով, որի շարահյուսությունն է՝

```
os.remove("file_name"),
```

որտեղ file_name-ը ջնջվող ֆայլի անունն է: Ջնջենք, օրինակ, test1.txt ֆայլը.

```
import os
os.remove("test1.txt ")
```

Եթե ֆայլը գտնվում է այլ պանակում, ապա անհրաժեշտ է նշել դրա ճանապարհը: Հետևյալ օրինակում ջնջվում է "test_1.txt" ֆայլը, որը գտնվում է 'C:/Examples/text_files' պանակում.

```
import os
file1='C:/Examples/text_files/test_1.txt'
os.remove(file1)
```

Սխալից խուսափելու համար, նախքան ֆայլը ջնջելը, os.path.exists մեթոդով պետք է ստուգել, թե արդյոք այդ ֆայլը գոյություն ունի՞: Մեթոդը վերադարձնում է True, եթե ֆայլը գոյություն ունի, False՝ հակառակ դեպքում: Օրինակ՝

```
import os
file1='C:/Examples/text_files/test_1.txt'
if os.path.exists(file1):
   os.remove(file1)
else:
  print("The file does not exist")
```

Ամբողջ պանակը կարելի է հեռացնել os.rmdir() մեթոդով, ընդ որում՝ կարելի է հեռացնել միայն դատարկ պանակները:

Օրինակ՝ ջնջենք myfolder դատարկ պանակը, որը գտնվում է 'C:/Examples/text_files' պանակում.

```
import os
os.rmdir('C:/Examples/text_files/myfolder')
```

os մոդուլի replace() մեթոդով կարելի է ֆայլը և պանակը մի վայրից տեղափոխել մեկ այլ վայր, ընդ որում, եթե վերջինս պա-

րունակում է նույն անունով ֆայլ կամ պանակ, ապա այն կվերագրվի: replace() մեթոդի շարահյուսությունն է.

os.replace(current_path, destination_path)

os.replace() մեթոդն ունի երկու արգումենտ՝

- current_path - ֆայլի ընթացիկ ուղին,
- destination_path – ֆայլի նոր ուղին:

Հետևյալ ծրագրում Book_orders.txt ֆայլը Old_orders պանակից տեղափոխվում է New_orders պանակ:

```
import os
current_path = 'Old_orders/Book_orders.txt'
destination_path = 'New_orders/Book_orders.txt'
os.replace(current_path,destination_path)
```

Ֆայլերը մի վայրից մեկ այլ վայր կարելի է տեղափոխել նաև shutil մոդուլի move() մեթոդով, որի շարահյուսությունը հետևյալն է.

shutil.move(current_path, destination_path)

ԳԼՈՒԽ 13. ՄՈԴՈՒԼՆԵՐ

13.1. Մոդուլներ: Մոդուլի ներմուծումը

Մոդուլը (Module) Python ֆայլ է (.py ընդլայնմամբ), որը պարունակում է ֆունկցիաներ, դասեր, փոփոխականներ և կարող է ներմուծվել Python այլ կոդի մեջ:

Մոդուլները Python կոդի կառավարելի կառուցվածքի ստեղծման հստակ և տրամաբանական միջոցն են: Կապակցված ֆունկցիաների, դասերի և փոփոխականների խմբավորումը մեկ մոդուլի շրջանակներում նախագծի կառուցվածքը դարձնում է հեշտ կողմնորոշելի և արդյունավետ: Նախագծի վրա աշխատելիս մոդուլները թույլ են տալիս տարբեր ծրագրորդների միաժամանակ աշխատել ծրագրի տարբեր մասերի վրա: Յուրաքանչյուր մշակող կարող է կենտրոնանալ հանձնարարված մոդուլների վրա՝ նվազեցնելով պատահական կոնֆլիկտների ռիսկը:

Յուրաքանչյուր մոդուլ ունի իր անհատական տիրույթը, ուստի մի մոդուլում սահմանված փոփոխականները, ֆունկցիաները և դասերը պատահաբար չեն հակասի մեկ այլ մոդուլի նույն անվանում ունեցող տարրերին՝ կանխելով անվանումների բախումները և դրանով կոդն ավելի հուսալի դարձնելով:

Մոդուլը (module) Python կոդ պարունակող .py ֆայլ է: Երբ մոդուլը ներմուծվում է, դրա սահմանումներն ու հրահանգները տեղափոխվում են ընթացիկ կոդի տիրույթ:

Փաթեթը (package) կապակցված մոդուլների հավաքածու է, որը կազմակերպված է պանակների հիերարխիայում: Փաթեթը պարունակում է __init__.py ֆայլ: Փաթեթը թույլ է տալիս ավելի արդյունավետորեն կառավարել միասին աշխատող բազմաթիվ մոդուլներ:

Գրադարանը (library) կարող է պարունակել մոդուլներ և փաթեթներ: Գրադարանի օրինակ է Python-ի ստանդարտ գրադարանը (Python Standard Library), որը Python-ի տեղադրման մաս է կազմում և առաջարկում է ֆունկցիոնալության լայն շրջանակ: Այս գրադարանում ներկառուցված են բազմաթիվ մոդուլներ, որոնց մի մասին մենք արդեն ծանոթ ենք և կիրառել ենք:

Մոդուլում պարունակվող ֆունկցիաներին դիմելու համար նախ պետք է մոդուլը ներմուծել հիշողության մեջ import հրամանի միջոցով: import հրամանի շարահյուսությունը հետևյալն է.

import module_1, module_2,... module_N

Օրինակ՝ import math հրամանով math ստանդարտ մաթեմատիկական մոդուլը (որը պարունակում է մաթեմատիկական ֆունկցիաներ) ներմուծվում է հիշողության մեջ, որից հետո կարելի է դիմել մոդուլի ֆունկցիաներին: Որևէ ֆունկցիայի դիմելու համար պետք է գրել մոդուլի անունը, դնել կետ, գրել ֆունկցիայի անունը համապատասխան արգումենտով: Օրինակ՝

```
import math
print(math.sqrt(121)) #121 թվի քառակուսի արմատը
```

Կարելի է միաժամանակ ներմուծել մի քանի մոդուլներ: Հետևյալ ծրագրում ներմուծվում են math և random մոդուլները, և կիրառվում են դրանց պատկանող տարբեր ֆունկցիաներ։

```
import math, random
print(math.sqrt(16)) # sqrt() ֆունկցիան
print(math.pi) # pi ֆունկցիան
print(math.sin(1.5)) # sin() ֆունկցիան
print(math.cos(0.5)) # cos() ֆունկցիան
print(math.tan(0.23)) # tan() ֆունկցիան
print(math.factorial(6)) # factorial() ֆունկցիան

print(random.randint(0, 10)) # պատահական ամբողջ թիվ [0,10] միջակայքից
print(random.random()) # պատահական իրական թիվ [0,1] միջակայքից
List = [1, 8, False, 100, "python", 36, "set"]
print(random.choice(List)) # ցուցակից պատահական տարրի ընտրություն
```

Հնարավոր է ներմուծել մոդուլի բոլոր տարրերը՝ օգտագործելով from ... import * հրամանը, որի շարահյուսությունն է.

from մոդուլի անուն import *

Հետևյալ օրինակում ներմուծվում են math մոդուլի բոլոր ֆունկցիաները, ապա, հաշվարկների համար կիրառվում են sqrt() և factorial() ֆունկցիաները։

```
from math import *
print(sqrt(16))
print(factorial(6))
```

Python-ի from... import հրամանը հնարավորություն է տալիս ներմուծելու մոդուլի որոշ ֆունկցիաներ։ from...import հրամանի շարահյուսությունն է.

from մոդուլի անունն է՝ import name_1, name_2, ... name_N

Օրինակ՝

```
from math import sqrt
```

Այս հրամանով Python-ը հիշողության մեջ բեռնում է միայն sqrt() ֆունկցիան, և այժմ այդ ֆունկցիան կարող է կանչվել՝ առանց նշելու մոդուլի անունը.

```
from math import sqrt
print(sqrt(16))
```

Խնդիր։ Հետևյալ ծրագրով random ներկառուցված մոդուլից ներմուծվում են choice() և shuffle() ֆունկցիաները։ Խաղացողների տրված ցուցակը նախ խառնվում է shuffle() ֆունկցիայով, ապա choice() ֆունկցիայով այդ ցուցակից պատահականորեն ընտրվում է մեկ խաղացող։

```
from random import choice, shuffle
players=['A','B','C','D', 'E','F','G','H','I']
shuffle(players)
print(players)
player=choice(players)
print('The selected player is:', player)
```

Արդյունքը՝

['H', 'A', 'B', 'D', 'F', 'I', 'C', 'G', 'E']
The selected player is: I

Ընդհանրապես, երբ Python ինտերպրետատորը հանդիպում է import հրամանին, այն ներմուծում է մոդուլը միայն այն դեպքում, երբ այդ մոդուլը առկա է որոնման ուղում: Որոնման ուղին այն պանակների ցուցակն է, որում ինտերպրետատորը որոնում է կատարում մոդուլը ներմուծելու համար:

Մոդուլը ներմուծելու համար Python ինտերպրետատորը փնտրում է մոդուլը ուղիների հետևյալ հաջորդականությամբ.

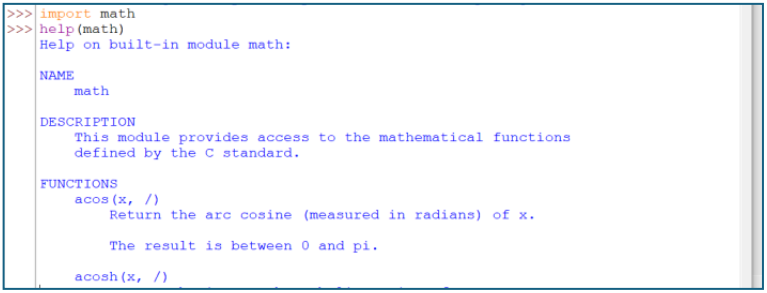
- ընթացիկ պանակում,
- PYTHONPATH միջավայրային փոփոխականում սահմանված պանակներում,
- լռությամբ սահմանված պանակում:

Python-ի dir() ֆունկցիան վերադարձնում է մոդուլում պարունակվող բոլոր ֆունկցիաների և փոփոխականների անունների ցուցակը: Օրինակ՝ հետևյալ հրամանով կարելի է ստանալ math մոդուլում պարունակվող բոլոր ֆունկցիաների ցուցակը.

```
>>>dir(math)
```

Որևէ մոդուլի մասին տեղեկություն կարելի է ստանալ help()-ի միջոցով (նկար 13.1), օրինակ՝

```
>>>import math
>>>help (math)
```



Նկար 13.1. Մոդուլի մասին տեղեկատվության ստացումը help()-ի միջոցով

Մոդուլում պարունակվող որևէ ֆունկցիայի մասին տեղեկություն կարելի է ստանալ՝ դիմելով help()-ի օգնությանը՝ փակագծերում գրելով մոդուլի ու ֆունկցիայի անունները։ Օրինակ՝

```
>>>help(math.sqrt)
```

Հարցման արդյունքում կստանանք տեղեկատվություն sqrt() ֆունկցիայի մասին.

```
Help on built-in function sqrt in module math:
sqrt(x, /)
   Return the square root of x.
```

math մոդուլը ներբեռնելիս ստեղծվում է math անունով փոփոխական, որի տիպը կարելի իմանալ type() ֆունկցիայով.

```
>>> import math
>>> type(math)
<class 'module'>
```

type() ֆունկցիան վերադարձնում է math փոփոխականի տիպը՝ module։ Այս փոփոխականը հղվում է մոդուլային օբյեկտին, որը պարունակում է հղումներ ֆունկցիաներին (ֆունկցիա օբյեկտներին).

13.2. Ինչպե՞ս ստեղծել ու ներմուծել սեփական մոդուլները

Ինչպես արդեն ասվել է, Python մոդուլները Python կոդ պարունակող .py ֆայլեր են։ Շատ մոդուլներ հասանելի են Python Standard Library-ի միջոցով, դրանք տեղադրվում են Python-ի տեղադրման հետ միասին։ Բացի այդ, ծրագրորդն ինքը կարող է ստեղծել իր սեփական մոդուլները։

Ստեղծենք մեր սեփական մոդուլը։ Այդ նպատակով նախ ստեղծենք own_mod.py անունով ֆայլը, որը պարունակում է հետևյալ ֆունկցիայի կոդը.

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

Ստեղծենք երկրորդ ֆայլը՝ main_program.py անունով, որտեղից կներբեռնենք own_mod մոդուլի f(x) ֆունկցիան։

main_program.py ֆայլը կարելի է տեղադրել own_mod.py ֆայլի հետ ինչպես նույն, այնպես էլ մեկ այլ պանակում։ Մոդուլը ներբեռնելիս Python-ի ինտերպրետատորը մոդուլը փնտրում է նախ ընթացիկ պանակում, հետո՝ PYTHONPATH միջավայրային փոփոխականում սահմանված պանակներում, ապա՝ լռությամբ սահմանված պանակում։ Եթե մոդուլը և main_program.py ֆայլը գտնվում են տարբեր պանակներում, ապա պետք է տալ մոդուլի տեղակայման ուղին։

sys մոդուլի path փոփոխականը վերադարձնում է այն ուղիների ցուցակը, որտեղ Python-ը լռելյայն պահպանում է իր սեփական մոդուլները։ Դա կարելի է ստանալ հետևյալ հրամանով.

```
>>>import sys
>>>sys.path
```

Այժմ ներբեռնենք own_mod մոդուլը և դիմենք դրանում գտնվող f() ֆունկցիային.

```
>>> import own_mod
>>> own_mod.f(2)
```

Այժմ ստեղծենք երկրորդ մոդուլը՝ math_lib.py անունով, որը պարունակում է հետևյալ ֆունկցիաները՝ fib1(), fib2(), gcd().

```
def fib1(n):   # Ֆիբոնաչիի թվերը մինչև n թիվը
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):  # Ֆիբոնաչիի թվերը մինչև n թիվը
    result = []
    a, b = 0, 1
    while a < n:
```

```
        result.append(a)
        a, b = b, a+b
    return result

def gcd(m,n): #երկու թվերի ամենամեծ ընդհանուր բաժանարարը
    while n!=0:
        temp=n
        n=m%n
        m=temp
    return m)
```

Նույն պանակում ստեղծենք երկրորդ ֆայլը՝ main_program_2.py անունով, որտեղից էլ կկանչենք math_lib.py մոդուլի fib2() և gcd() ֆունկցիաները։

```
import math_lib
print(math_lib.fib2(500))
print(math_lib.gcd(36,120))
```

Արդյունքը՝

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
12
```

Մոդուլի անունը հասանելի է որպես __name__ գլոբալ փոփոխականի արժեք: Հետևյալ օրինակում ներբեռնվում է math_lib մոդուլը, և մոդուլի անունը հասանելի է դառնում __name__ գլոբալ փոփոխականի արժեքով։

```
>>>import math_lib
>>>math_lib.__name__
```

Արդյունքը՝

```
'math_lib'
```

Մոդուլի ֆունկցիայի անունը կարելի է վերագրել լոկալ փոփոխականի և օգտագործել այդ անունը ֆունկցիայի կանչի ժամանակ: Օրինակ՝

```
>>>gcd=math_lib.gcd
>>>gcd(15,25)
5
>>>fbn=math_lib.fib1
>>>fbn(1500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Մոդուլի տարբեր ֆունկցիաների կարելի է դիմել նաև from... import հրամանով, օրինակ՝

```
>>>from math_lib import fib2, gcd
>>>fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>>gcd(24,160)
8
```

Եթե import հրամանի մեջ մոդուլի անվանը հաջորդում է as բանալի բառը, ապա մոդուլի անունը վերագրվում է as բառին անմիջապես հաջորդող փոփոխականին: Օրինակ՝

```
>>>import math_lib as math_func
>>>math_func.gcd(30,65)
5
```

Նմանապես, եթե from…import հայտարարության մեջ ֆունկցիայի անվանը հաջորդում է as բառը, ապա ֆունկցիայի անունը վերագրվում է as բառին անմիջապես հաջորդող փոփոխականին: Օրինակ՝

```
>>>from math_lib import fib1 as fibonacci
>>>fibonacci(300)
0 1 1 2 3 5 8 13 21 34 55 89 144 233
```

13.3. Համակարգային մոդուլներ

Համակարգային մոդուլների վերաբերյալ ամբողջական և մանրակրկիտ տեղեկություն կարելի է ստանալ «Python documentation» www.python.org փաստաթղթում:

Այս բաժնում մենք անդրադարձել ենք os համակարգային մոդուլին, որն ապահովում է ֆայլերի ու պանակների հետ աշխատանքի տարբեր մեթոդներ: Նկարագրենք այդ մեթոդներից մի քանիսը:

os.getcwd() մեթոդը վերադարձնում է ընթացիկ աշխատանքային պանակը:

```
import os
print("The current directory is:", os.getcwd())
```

Հրամանի կատարման արդյունքից պարզ է, որ ընթացիկ պանակն է՝

```
The current directory is:
C:\Users\AppData\Local\Programs\Python\Python313
```

os.mkdir() մեթոդով ընթացիկ աշխատանքային պանակում ստեղծվում է նոր, դատարկ պանակ: Օրինակ՝ ստեղծենք «folder1» անունով պանակը:

```
os.mkdir("folder1")
```

Եթե կրկին գործարկենք os.mkdir("folder1") հրամանը, ապա, քանի որ "folder1" անունով պանակ արդեն գոյություն ունի, կծագի սխալ, և կտրվի այդ մասին հետևյալ հաղորդագրությունը՝ FileExistsError: Այդ իսկ պատճառով մինչև մեթոդի կիրառումը անհրաժեշտ է նախապես **os.path.isdir()** մեթոդով ստուգել՝ այդպիսի անունով պանակ գոյություն ունի՞, թե՞ ոչ:

```
if not os.path.isdir("folder1"):
    os.mkdir("folder1")
```

os.path.isdir() ֆունկցիան կվերադարձնի True, եթե մեթոդին փոխանցված անունով պանակ արդեն գոյություն ունի:

os.chdir() մեթոդը փոխում է ընթացիկ պանակը: Օրինակ, ընթացիկ պանակ դարձնենք հենց նոր ստեղծված folder1 անունով պանակը:

```
os.chdir("folder1")
print("The current directory is: \n", os.getcwd())
```

os.chdir("..") հրամանով կարելի է վերադառնալ նախորդ պանակին:

os.makedirs() մեթոդով կարելի է ստեղծել իրար մեջ ներդրված պանակներ հետևյալ ուղղագրությամբ.

```
os.makedirs("folder1/folder2/folder3")
```

Այս հրամանով ստեղծվում են իրար մեջ հաջորդաբար ներդրված երեք պանակներ:

Python-ի os մոդուլն ունի ֆայլեր ու պանակները կառավարելու այնպիսի մեթոդներ, ինչպիսին են ֆայլերի վերանվանումը՝ **os.rename(),** տեղափոխումը՝ **os.replace(),** ջնջումը՝ **os.remove(),** պանակի ջնջումը՝ **os.rmdir(),** ֆայլի գոյությունը՝ **os.path.exists(),** որոնք նկարագրվել են *12.3 պարագրաֆում*:

os.listdir() ֆունկցիան վերադարձնում է ցուցակ, որը պարունակում է նշված պանակում պարունակվող պանակների ու ֆայլերի անունները: Եթե ֆունկցիայի արգումենտների ցուցակը դատարկ է, ապա այն վերադարձնում է ընթացիկ աշխատանքային պանակի պարունակությունը: Օրինակ՝

```
import os
print('The current working directory is:\n',os.getcwd())
print("All folders and files in the directory:\n", os.listdir())
```

Ծրագրի կատարման արդյունքը՝

```
The current working directory is:
 C:\Examples\text_files
```

All folders and files in the directory:
['55.py', '69.py', 'New_orders', 'Old_orders', 'own_mod.py', 'radius.txt', 'square10.txt', 'textfile_1.txt', 'writefile.txt']

Օպերացիոն համակարգում ֆայլի մասին տեղեկատվություն ստանալու համար օգտագործվում է **os.stat()** ֆունկցիան: Ներքոնշյալ հրամանով բերվում է տեղեկատվություն ընթացիկ աշխատանքային պանակում գտնվող radius.txt տեքստային ֆայլի մասին:

```
import os
print(os.stat("radius.txt"))
```

Արդյունքը՝

os.stat_result(st_mode=33206, st_ino=281474977069505, st_dev=10956712540155926370, st_nlink=1, st_uid=0, st_gid=0, st_size=21, st_atime=1737268173, st_mtime=1737221324, st_ctime=1737221290)

Արդյունքում տեղեկատվությունը բերվում է հավաքածուի տեսքով, որտեղ ֆայլի հատկություններն են՝

st_size – ֆայլի չափը բայթերով,
st_atime – վերջին մուտքի ժամանակը վայրկյաններով,
st_mtime – վերջին փոփոխության ժամանակը,
st_ctime – Windows-ում սա ֆայլի ստեղծման ժամանակն է, իսկ Linux-ում՝ մետատվյալների վերջին փոփոխության ժամանակը:

os.stat() ֆունկցիայով կարելի է ստանալ որևէ կոնկրետ ատրիբուտի, օրինակ, ֆայլի չափի վերաբերյալ տեղեկատվություն հետևյալ ձևով.

```
print("File size:", os.stat("radius.txt").st_size)
```

Արդյունքը՝ File size: 21

ԳԼՈՒԽ 14. ԲԱՑԱՌՈՒԹՅՈՒՆՆԵՐ

14.1. Բացառություններ: Python-ի ներկառուցված բացառությունները

Ծրագրի կատարման ընթացքում կարող են հանդիպել իրավիճակներ, որոնք խաթարում են աշխատանքի նորմալ ընթացքը: Երբ Python-ի ծրագրում առաջանում է այդպիսի իրավիճակ, այն անպայման պետք է մշակվի, հակառակ դեպքում ծրագիրը կընդհատի իր աշխատանքը: Այդպիսի իրավիճակները կոչվում են բացառություններ (բացառիկ իրավիճակներ):

Բացառությունները տարբերվում են սխալներից: Սխալները սովորաբար կոդի տրամաբանության կամ կարգավորման խնդիրներ են, ինչպես օրինակ՝ շարահյուսական սխալները: Դրանք պետք է շտկվեն ծրագրավորողի կողմից: Իսկ բացառություններն առաջանում են այնպիսի իրավիճակների պատճառով, ինչպիսիք են, օրինակ, անվավեր մուտքագրումը, ֆայլերի բացակայությունը, ցանցային խնդիրները և այլն: Բացառությունները կարող են մշակվել ծրագրի կողմից:

Բացառությունը Python օբյեկտ է, որը ներկայացնում է սխալ: Դիտարկենք հետևյալ օրինակը.

```
x = int(input("Enter: "))
print(5/x)
```

Եթե գործարկենք այս ծրագիրը, ապա կտեսնենք, որ x փոփոխականին 0 արժեք կամ տառ վերագրելը հանգեցնում են սխալների, դրանք են՝ զրոյի վրա բաժանելու կամ տառը թվի փոխակերպելու սխալները:

Օրինակ՝ x փոփոխականին 0 արժեք վերագրելու դեպքում կծագի ZeroDivisionError սխալը, և կտրվի հետևյալ հաղորդագրությունը՝

```
Enter: 0
Traceback (most recent call last):
  File "C:/Examples/exceptions_1.py", line 2, in <module>
    print(5/x)
ZeroDivisionError: division by zero
```

Կամ x փոփոխականին տառ վերագրելու դեպքում կծագի ValueError սխալը, և կտրվի հետևյալ հաղորդագրությունը՝

```
Enter: a
Traceback (most recent call last):
  File "C:/Examples/exceptions_1.py", line 1, in <module>
    x = int(input())
ValueError: invalid literal for int() with base 10: 'a'
```

Հետևյալ ծրագրում ծագում է KeyError սխալը, քանի որ փորձ է արվում մուտք գործել fruit բառարան՝ գոյություն չունեցող բանալիով։

```
fruit = {"name": "apple", "color": "green"}
print(fruit["price"])
```

Տրվում է հետևյալ հաղորդագրությունը.

```
Traceback (most recent call last):
  File "C:/Examples/exeptions_2.py", line 2, in <module>
    print(fruit["price"])
KeyError: 'price'
```

Python լեզուն ունի բազմաթիվ ներկառուցված բացառություններ, որոնցից յուրաքանչյուրը ներկայացնում է որոշակի տիպի սխալ։ Ստորև, համառոտ նկարագրությամբ, ներկայացված են աղյուսակում թվարկված Python-ի հիմնական ներկառուցված բացառությունները (աղյուսակ 14.1).

Բացառությունը	Նկարագրությունը
BaseException	Բազիսային բացառություն՝ բոլոր ներկառուցված բացառությունների համար։
Exception	Բազիսային բացառություն՝ բոլոր ոչ ելքային բացառությունների համար։
ArithmeticError	Բազիսային բացառություն՝ թվաբանական գործողությունների հետ կապված բոլոր սխալների համար։
ZeroDivisionError	Բաժանում զրոյի։
OverflowError	Առաջանում է, երբ թվաբանական գործողության արդյունքը գերազանցում է տվյալների տիպի ներկայացման առավելագույն սահմանաչափը։
AttributeError	Առաջանում է, երբ ատրիբուտի հղումը կամ վերագրումը ձախողվում է։
IndexError	Առաջանում է, երբ փորձում է արվում դիմել գոյություն չունեցող ինդեքսով տարրի։
KeyError	Առաջանում է, երբ բառարանում բանալին չի գտնվում։
MemoryError	Առաջանում է, երբ գործողությունը սպառում է հիշողությունը։
NameError	Առաջանում է, երբ լոկալ կամ գլոբալ փոփոխականը չի գտնվում։
OSError	Առաջանում է, երբ համակարգին վերաբերող գործողությունը (օրինակ՝ ֆայլի մուտք/ելք) ձախողվում է։
TypeError	Առաջանում է, երբ գործողությունը կամ ֆունկցիան կիրառվում է անհամապատասխան տիպի օբյեկտի վրա։
ValueError	Առաջանում է, երբ ֆունկցիան ստանում է ճիշտ տիպի, բայց անհամապատասխան արժեք ունեցող արգումենտ։
ImportError	Առաջանում է, երբ ներմուծման հրամանը խնդիրներ ունի։

ModuleNotFoundError	Առաջանում է, երբ մոդուլը չի գտնվել:
IOError	Առաջանում է, երբ մուտք/ելքի գործողությունը (օրինակ՝ ֆայլում կարդալը կամ գրելը) ձախողվում է:
FileNotFoundError	Առաջանում է, երբ ֆայլը չի գտնվել:
KeyboardInterrupt	Առաջանում է, երբ օգտատերը փորձում է ընդհատել ծրագրի կատարումը՝ սեղմելով Ctrl+C ստեղները:
SystemExit	Առաջանում է, երբ ծրագիրը փակելու համար կանչվում է sys.exit() ֆունկցիան:
RuntimeError	Առաջանում է, երբ ծրագրում տեղի է ունենում ընդհանուր տիպի սխալ:
RecursionError	Առաջանում է, երբ գերազանցվում է ռեկուրսիայի առավելագույն խորությունը:
SyntaxError	Առաջանում է, երբ կոդի շարահյուսության մեջ առկա է սխալ:
IndentationError	Առաջանում է, երբ կոդում կա սխալ խորությամբ գրություն:
TabError	Առաջանում է Tab-ի կամ բացակների անհամապատասխան օգտագործումից:
UnicodeError	Առաջանում է, երբ տեղի է ունենում Unicode-ի հետ կապված կոդավորման կամ ապակոդավորման սխալ:

Աղյուսակ 14.1. Python-ի հիմնական ներկառուցված բացառությունները

Python լեզուն ունի ծրագրում հանդիպած բացառությունները մշակելու հզոր մեխանիզմ:

14.2. Բացառությունների մշակումը Python-ում

Ինչպես նշեցինք նախորդ պարագրաֆում, երբ Python-ը ծրագրի կատարման ընթացքում հանդիպում է այնպիսի իրավիճակի, որը չի կարողանում հաղթահարել, առաջանում է բացա-

ռություն։ Տանք բացառիկ իրավիճակների մշակման մեխանիզմի ընդհանուր նկարագիրը Python-ում։

"try:", "except:", "finally:" մեխանիզմը։ Ծրագրի այն բոլոր հրամանները, որոնց կատարման ընթացքում պետք է հայտնաբերել սխալները և ապահովել բացառիկ իրավիճակների մշակում, տեղավորվում են "try:" բլոկի մեջ։ “try:” բլոկից հետո անհրաժեշտ է ներառել մեկ կամ մեկից ավելի թվով "except:" հայտարարություններ, որոնցից յուրաքանչյուրը բացառիկ իրավիճակի մշակման կոդ է։ Երբ որ ծրագրի կատարման ընթացքում ծագում է բացառիկ իրավիճակ, կառավարումը փոխանցվում է համապատասխան "except:" բլոկին, որն էլ կատարում է տվյալ իրավիճակի մշակումը։ Եթե բլոկում սխալներ չեն հայտնաբերվել, ապա ղեկավարումը տրվում է “else:” բլոկին, որը գտնվում է “try:” բլոկից դուրս։ Նշենք, որ “else:” բլոկը կարող է բացակայել։

"try:" բլոկի հետ կարելի է օգտագործել նաև "finally:" բլոկը։ "finally:" բլոկը ծրագրի այն հատվածն է, որտեղ կարելի է տեղադրել ցանկացած կոդ, որը պետք է կատարվի անկախ այն բանից՝ "try:" բլոկում բացառություն առաջացե՞լ է, թե՞ ոչ։ Այս բլոկը ևս կարող է բացակայել։

"try:", "except:", "finally:" բլոկների շարահյուսությունը հետևյալն է.

```
try:
   բոլոր հրամանները
except բացառություն_1:
   բացառություն_1-ի մշակում
except բացառություն_2:
   բացառություն_2-ի մշակում
.....................................................
else:
   հրամաններ, որոնք կատարվում են, երբ բացառիկ իրավիճակ չի ստեղծվում
finally:
   հրամաններ, որոնք կատարվում են անկախ այն բանից՝ բացառություն տեղի՞ է ունեցել, թե՞ ոչ։
```

Դիտարկենք հետևյալ ծրագիրը.

```
try:
   x = int(input("Enter number: "))
   print(5/x)
except:
   print("Error dividing by zero")
```

Գործարկենք ծրագիրը և x փոփոխականին վերագրենք 0 արժեքը։

```
Enter number: 0
Error dividing by zero
```

Այս ծրագրում "try:" բլոկը պարունակում է հրամաններ, որոնցում կարող է բացառիկ իրավիճակ առաջանալ։ Քանի որ x փոփոխականին վերագրվել է 0 արժեքը, ծագում է բացառություն, և կառավարումը փոխանցվում է "except:" բլոկին, ու արտածվում է «Error dividing by zero» տեքստը։

Կրկին գործարկենք ծրագիրը և այս անգամ x փոփոխականին վերագրենք որևէ տառ։ Կտեսնենք, որ սխալ է առաջանում, այս անգամ՝ տառը թվի փոխարկելու սխալ, սակայն կրկին կառավարումը տրվում է նույն "except:" բլոկին, և արտածվում է նույն՝ «Error dividing by zero» տեքստը։

```
Enter number: a
Error dividing by zero
```

Սա տեղի է ունենում այն պատճառով, որ գրված է միակ "except:" բլոկը, որը կառավարում է բոլոր տեսակի սխալները, քանի որ նշված չի բացառության տեսակը։

Python-ում զրոյի վրա բաժանումը հանգեցնում է ZeroDivisionError տեսակի սխալի, իսկ տիպի սխալ փոխակերպումը՝ ValueError։ Ուստի "try:" բլոկի մեջ կներառենք այնպիսի "except:" հայտարարություններ, որոնք մշակում են այդ բացառությունները։ Այդ դեպքում, բացառիկ իրավիճակի տեսակից կախված, մշակման ղեկավարումը կտրվի համապատասխան "except:" հայտարարությանը։

```
try:
   x = int(input("Enter number: "))
   print(5/x)
except ZeroDivisionError:
   print("Error dividing by zero")
except ValueError:
   print("Error converting to a number")
```

Գործարկենք ծրագիրը և x փոփոխականին վերագրենք 0 արժեքը, կստանանք.

```
Enter number: 0
Error dividing by zero
```

Կրկին գործարկենք ծրագիրը և x փոփոխականին վերագրենք 'a' արժեքը, կստանանք.

```
Enter number: a
Error converting to a number
```

Միևնույն "except:" բլոկը կարելի է օգտագործել մի քանի բացառություններ մշակելու համար: Շարահյուսությունը հետևյալն է.

```
try:
   բոլոր հրամանները
except(բացառություն_1, բացառություն_2, ..., բացառություն_N):
Այս հրամանների խումբը կատարվում է, եթե ծագում է որևէ բացառություն տրված բացառությունների ցանկից:
.....
else:
Այս հրամանների խումբը կատարվում է, եթե բացառիկ իրավիճակ չի ստեղծվում:
```

Հետևյալ օրինակում առաջին "except:" հայտարարությունը մշակում է ValueError և TypeError բացառությունները, իսկ երկ-

րորդ "except:" հայտարարությունը՝ IndexError բացառությունը: Վերջինս առաջանում է, երբ փորձ է արվում դիմել ցուցակի գոյություն չունեցող ինդեքսով տարրի:

```
a = ["1", "two", 3]
try:
   sum=int(a[0]) + int(a[1])
   print(sum)
except (ValueError, TypeError) as e:
   print("Error", e)
except IndexError:
   print("Index out of range.")
```

Ծրագրի գործարկման արդյունքում առաջանում է TypeError տիպի սխալ՝ կապված ցուցակի "two" տարրը ամբողջ տիպի փոխակերպելու հետ, ուստի այդ բացառությունը մշակվում է և տրվում է հետևյալ հաղորդագրությունը.

```
Error invalid literal for int() with base 10: 'two'
```

Բացառության մասին տեղեկատվությունը կարելի է վերագրել ինչ-որ փոփոխականի՝ օգտագործելով as հայտարարությունը: Այս փոփոխականը ստանում է բացառության արժեքը, որը պարունակում է բացառության պատճառը: Օրինակ՝

```
try:
    x = int(input("Enter a number: "))
    print(5/x)
except ZeroDivisionError as z:
    print(z)
except ValueError as v:
    print(v)
else:
    print("No exceptional situations occurred!")
finally:
    print("The end!")
```

Գործարկենք ծրագիրը x փոփոխականի տարբեր արժեքների համար։

```
========RESTART: C:/Examples/exceptions_6.py ====
Enter a number: 0
division by zero
The end!
========RESTART:C:/Examples/exceptions_6.py ====
Enter a number: a
invalid literal for int() with base 10: 'a'
The end!

========RESTART: C:/Examples/exceptions_6.py ====
Enter a number: 10
0.5
No exceptional situations occurred!
The end!
```

Որոշ իրավիճակներում, երբ անցանկալի պայման է առաջանում, ծրագրորդը ինքը կարող է կանգնեցնել ծրագիրը՝ առաջացնելով բացառություն։ Դա կարելի է անել raise բանալի բառի միջոցով։ Շարահյուսությունը հետևյալն է.

raise {name_of_ the_ exception_class}

Կամ բազիսային տեսքը՝

raise Exception("user text"):

Ստորև բերված կոդում ստուգվում է, արդյոք թիվը զո՞յգ է, թե՞ կենտ։ Եթե թիվը կենտ է, ապա ստեղծվում է բացառություն raise բանալի բառի միջոցով։

```
n = 11
if n % 2 != 0:
   raise Exception("The number shouldn't be an odd integer")
```

Քանի որ ծրագրում n փոփոխականին վերագրվել է 11 արժեքը, որը կենտ է, ապա առաջանում է սխալ։

```
Traceback (most recent call last):
  File "C:/Examples/exceptions_7.py ", line 3, in <module>
    raise Exception("The number shouldn't be an odd integer")
Exception: The number shouldn't be an odd integer
```

Բացառությունների մշակումը կարելի է օգտագործել ֆունկցիաներ գրելիս, ֆայլերի հետ աշխատելիս և այլն։ Հետևյալ list_find անունով ֆունկցիան lst ցուցակում փնտրում է target տարրը և վերադարձնում դրա առաջին հանդիպման ինդեքսը։ Եթե այդպիսի տարր չկա, ծագում է սխալ, և գեներացվում է բացառություն՝ ValueError, որը մշակվում է համապատասխան except բլոկի մեջ։

```
def list_find(lst, target):
    try:
        index = lst.index(target)
    except ValueError:
    #ValueError: value is not in list
        index = -1
    return index

print(list_find([3,5,6,7], 5)) #ցուցակում փնտրվում է 5 թիվը
print(list_find([3,5,6,7], -6)) #ցուցակում փնտրվում է -6 թիվը
```

Գործարկենք ծրագիրը՝ Արդյունքը՝ 1 -1

14.3. Օգտագործողի սահմանած բացառություններ

Python-ը հնարավորություն է տալիս օգտագործողին ստեղծելու սեփական բացառությունները՝ ժառանգած ստանդարտ ներկառուցված բացառություններից։ Սա կողը դարձնում է ավելի անվտանգ։ Սեփական բացառությունը ստեղծելու համար պարզապես պետք է սահմանել նոր դաս, որը ժառանգում է Exception

բազային դասից կամ որևէ այլ ներկառուցված բացառությունից (դասերի մասին տե՛ս *գլուխ 15*):

Հետևյալ օրինակում ValidationError-ը բացառություն է, որը ժառանգում է Exception բազային բացառության վարքագիծը:

```
class ValidationError(Exception):
   pass
```

Օգտագործողի կողմից ստեղծված բացառությունը առաջանում է try: բլոկում և մշակվում է except: բլոկում, ինչպես ցույց է տրված հետևյալ կոդում՝

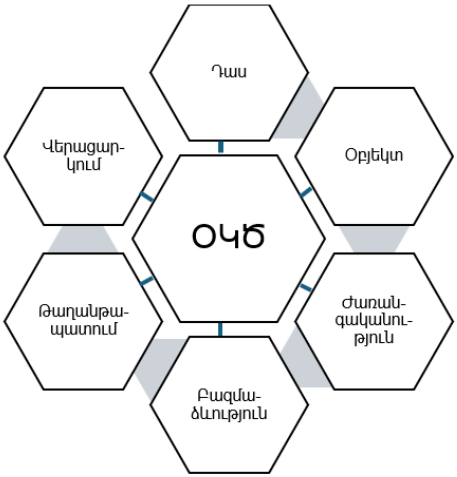
```
class ValidationError(Exception):
   pass

def person_age(age):
   if age < 0:
      raise ValidationError('Age cannot be negative')
   elif age > 130:
      raise ValidationError('Age cannot be more than 130')
   return True

try:
   person_age(150)
except ValidationError as e:
   print(e)
```

Այս օրինակում person_age ֆունկցիան օգտագործում է մեր ստեղծած ValidationError բացառությունը՝ մուտքագրված տարիքը ստուգելու համար: Ֆունկցիան ոչ ճիշտ տվյալներով կանչելը հանգեցնում է ValidationError բացառության, որը որսվում և մշակվում է: Գործարկելով ծրագիրը՝ կունենանք.

```
Age cannot be more than 130
```

ԳԼՈՒԽ 15. ՕԲՅԵԿՏ-ԿՈՂՄՆՈՐՈՇՎԱԾ ԾՐԱԳՐԱՎՈՐՄԱՆ ՀԻՄՈՒՆՔՆԵՐԸ

Օբյեկտ-կողմնորոշված ծրագրավորումը՝ ՕԿԾ (Object-Oriented Programming - OOP), Python լեզվի հիմնարար հասկացություն է, որը թույլ է տալիս կառուցել մոդուլային և մասշտաբային ծրագրեր: ՕԿԾ հիմնական հասկացություններն են՝ դաս, օբյեկտ, բազմաձևություն, թաղանթապատում, ժառանգականություն, տվյալների վերացարկում (նկար 15.1):



Նկար 15.1. ՕԿԾ հիմնական հասկացությունները

15.1. Դասեր ու օբյեկտներ

Օբյեկտ-կողմնորոշված ծրագրավորման հիմնական հասկացություններից է *դասը*: Այն ծրագրորդի կողմից սահմանված տիպ է, որի հիման վրա ստեղծվում է օբյեկտը: Օբյեկտը այդ սահմանված տիպին համապատասխանող տվյալների կոնկրետ կառուցվածքն է, տվյալ նկարագրությամբ որոշվող տվյալների ներկայացուցիչը:

Սահմանվող դասը նոր տիպ է, ուստի անհրաժեշտ է նկարագրել այդ տիպի բոլոր հատկություններն ու այն գործողու-

թյունները, որ կարելի է կատարել այդ տիպի տվյալների հետ: Դասի նկարագրության ընդհանուր տեսքը հետևյալն է՝

```
class դասի անուն:
դասի մարմին
```

Python-ում դասը սահմանվում է class բանալի բառի միջոցով, որին հետևում է դասի անունը: Դասերի անունները սովորաբար սկսվում են մեծատառով: Դասի մարմինը պարունակում է նկարագրվող դասի անդամները, որոնք կարող են լինել ատրիբուտներ (հատկություններ, փոփոխականներ) կամ մեթոդներ (ֆունկցիաներ): Դատարկ դասի համար օգտագործվում է pass հրամանը սխալից խուսափելու համար:

Ստեղծենք x ատրիբուտով MyClass դասը, ապա՝ MyClass տիպի p1 փոփոխականը և արտածենք x ատրիբուտի արժեքը:

```
class MyClass:
  x = 5

p1 = MyClass() #ստեղծվում է MyClass տիպի p1 օբյեկտը
print(p1.x) #տպում է x ատրիբուտի արժեքը
```

__init__() մեթոդը: Python լեզվում դասն ունի ներկառուցված __init__ () մեթոդը, որը կոչվում է կոնստրուկտոր (constructor): Այն ավտոմատ կերպով կանչվում է, երբ ստեղծվում է սահմանված դասի նոր օբյեկտ: Կոնստրուկտորն օգտագործվում է օբյեկտ ստեղծելիս դասի ատրիբուտներին արժեքներ վերագրելու (սկզբնարժեքավորելու) համար: Դասի նոր օբյեկտ ստեղծելու համար գրվում է դասի անունը, ապա փակագծերի մեջ այն արժեքները, որոնք պետք է վերագրվեն ատրիբուտներին:

Դիտարկենք հետևյալ օրինակը.

```
class Students:
  def __init__(self, name, surname, address, age):
    self.name = name
    self.surname = surname
    self.address=address
    self.age = age
```

```
# Students դասի օբյեկտների ստեղծում
student1 = Students('Anahit', 'Avagyan', 'Yerevan', 18)
student2 = Students('Suren', 'Gevorgyan', 'Yerevan', 23)
print('name:',student1.name,'surname:',student1.surname,'address
:',student1.address, 'age:',student1.age)
print('name:',student2.name,'surname:',student2.surname,'address
:',student2.address, 'age:',student2.age)
```

Ծրագրի կատարման արդյունքը՝

```
name: Anahit surname: Avagyan address: Yerevan age: 18
name: Suren surname: Gevorgyan address: Yerevan age: 23
```

Ներկայացված ծրագրում սահմանվել է Students դասը, որն ունի name, surname, address, age ատրիբուտները, որոնք այդ դասի ներսում սահմանված փոփոխականներ են: Նկարագրված դասն օգտագործելով՝ կարելի է ստեղծել բազմաթիվ օբյեկտներ՝ տարբեր հատկություններով ու վարքագծով: student1 և student2 օբյեկտները ստեղծելիս __init__() մեթոդն օգտագործվել է այդ ատրիբուտներին արժեքներ վերագրելու համար:

self պարամետրը: Դասի մեթոդների առաջին արգումենտը self հատուկ փոփոխականն է (կարող է լինել այլ անուն), որն օգտագործվում է ընթացիկ օբյեկտին հղվելու համար: Մեթոդի կանչի ժամանակ Python-ը ավտոմատ կերպով օբյեկտի անունը փոխանցում է self փոփոխականին ամեն անգամ, երբ դասը հղում է կատարում իր փոփոխականներից կամ մեթոդներից մեկին, դրանցից առաջ պետք է գրված լինի self: Օրինակ՝

```
self.name = name
```

Օբյեկտի հատկությունների փոփոխումը: Օբյեկտի հատկությունները կարելի է փոփոխել, օրինակ, հետևյալ կերպ՝

```
student1.age=70
```

Օբյեկտի հատկությունների ջնջումը: Օբյեկտի հատկությունները կարելի է ջնջել del հրամանով: Ջնջենք, օրինակ, student1

օբյեկտի name հատկությունը: Ջնջելուց հետո, եթե տպենք name փոփոխականի արժեքը, կտրվի հաղորդագրություն, որ օբյեկտն այդպիսի հատկություն չունի՝ AttributeError: 'Students' object has no attribute 'name':

```
del student1.name
print(student1.name)
```

Արդյունքը՝

```
Traceback (most recent call last):
  File "C:/Examples/class_1.py", line 11, in <module>
    print(student1.name)
AttributeError: 'Students' object has no attribute 'name'
```

Օբյեկտի ջնջումը: Օբյեկտները կարելի է ջնջել՝ օգտագործելով del հրամանը: Օրինակ՝

```
del student1
```

Այժմ, եթե փորձենք արտածել student1 օբյեկտի հատկությունները, կծագի սխալ, և կտրվի հաղորդագրություն սխալի մասին, այն է՝ այդպիսի օբյեկտ գոյություն չունի:

```
Traceback (most recent call last):
  File "C:/Examples/class_2.py ", line 14, in <module>
    print(student1.name, student1.surname, student1.address, student1.age)
NameError: name 'student1' is not defined. Did you mean: 'Students'?
```

pass հրամանը: Դասի սահմանումը չի կարող դատարկ լինել, բայց, եթե ինչ-ինչ պատճառներով դասի մարմինը դատարկ է, պետք է տեղադրել pass հրամանը՝ սխալից խուսափելու համար: Օրինակ՝

```
class Person:
    pass
```

Հետևյալ օրինակում ստեղծվել է Example դասը a և b ատրիբուտներով և add() մեթոդով, որը գումարում է a և b փոփոխականների արժեքները: Ապա ստեղծվել է Example դասի e օբյեկտը, որի ստեղծման ժամանակ դասի ատրիբուտները սկզբնարժեքավորվել են 8 և 6 արժեքներով:

```
class Example:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def add(self):
        return self.a + self.b

e = Example(8, 6)
print(e.add())
```

Ծրագրի կատարման արդյունքը՝ 14

15.2. Ժառանգում

Ժառանգումը (Inheritance) մեխանիզմ է, որը թույլ է տալիս դասին ձեռք բերելու մեկ այլ դասի հատկություններն ու մեթոդները: Այն խթանում է կոդի վերօգտագործումը՝ խուսափելով կոդի կրկնությունից ու նվազեցնելով ծրագրում սխալներ թույլ տալու հավանականությունը:

Դասը կարող է ժառանգել մեկ այլ դասի հատկությունները՝ ավելացնելով իրեն յուրահատուկ բնութագրիչներ: Այսինքն՝ ժառանգումը թույլ է տալիս ընդարձակել դասի վարքագիծը՝ ժառանգելով բազային դասի ֆունկցիոնալությունը: Ժառանգող դասը կոչվում է ենթադաս (Child class, երեխա դաս, դուստր դաս), իսկ այն դասը, որից ժառանգվում են հատկություններն ու մեթոդները՝ ծնող դաս (Parent class):

Ժառանգման տեսակներն են՝

- Ժառանգում մեկ ծնող դասից:
- Բազմակի ժառանգում, երբ ենթադասը ժառանգում է մեկից ավելի ծնող դասերից:

- Բազմամակարդակ ժառանգում, երբ ենթադասը ժառանգում է մեկ ծնող դասից, որն էլ իր հերթին ժառանգում է մեկ այլ ծնող դասից:
- Հիերարխիկ ժառանգում, երբ մի քանի ենթադասեր ժառանգում են մեկ ծնող դասից: Այս կառուցվածքը ձևավորում է ծառանման հիերարխիա, որը թույլ է տալիս դուստր դասերին ժառանգել ծնող դասի ատրիբուտներն ու մեթոդները՝ միաժամանակ պահպանելով իրենց սեփական հատկություններն ու վարքագիծը:
- Հիբրիդային ժառանգում, որը ժառանգման երկու կամ ավելի տեսակների համադրություն է:

Ենթադասի՝ ծնող դասից ժառանգման շարահյուսությունն է.

```
class ParentClass:
   Ծնող դասի մարմինը
class ChildClass(ParentClass):
   Ենթադասի մարմինը
```

Ստեղծենք Person անունով դաս՝ name և age հատկություններով և printname() մեթոդով՝

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

Այժմ ստեղծենք Student ենթադասը, որը ժառանգում է Person դասի բոլոր հատկություններն ու մեթոդները: Դա կարելի է անել հետևյալ ձևով.

```
class Student(Person):
  pass
```

Ստեղծված Student ենթադասն ունի նույն հատկություններն ու մեթոդները, ինչ Person դասը: Սակայն ենթադասին չեն ավելացվել որևէ այլ հատկություններ կամ մեթոդներ, քանի որ

Student ենթադասի մարմինը դատարկ է: Ստեղծենք Student ենթադասի նոր օբյեկտ և կիրառենք printname() մեթոդը՝

```
std = Student("Mike", 25)
std.printname()
```

Կոդն ամբողջությամբ հետևյալն է՝

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

class Student(Person):
  pass

std = Student("Mike", 25)
std.printname()
```

Ծրագրի կատարման արդյունքը՝ Mike 25:

Ենթադասը կարող է ժառանգել ծնող դասի հատկություններն ու մեթոդները կամ մեթոդները նաև __init__() ֆունկցիայի միջոցով: Վերհիշենք, որ __init__() ֆունկցիան կանչվում է ավտոմատ կերպով ամեն անգամ, երբ ստեղծվում է դասի նոր օբյեկտ: Որպեսզի դուստր դասը ժառանգի ծնող դասի հատկություններն ու մեթոդները, ենթադասի __init__() ֆունկցիայից պետք է կանչել ծնող դասի __init__() ֆունկցիան հետևյալ ձևով.

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

Գործարկենք կոդը.

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

```
    self.name = name
    self.age = age
  def printname(self):
    print(self.name, self.age)

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

std = Student("Mike", 25)
std.printname()
```

Ծրագրի կատարման արդյունքն է՝ Mike 25:

Python-ն ունի նաև super() ֆունկցիան, որը թույլ է տալիս դիմել ծնող դասի մեթոդներին ու ատրիբուտներին դուստր դասի ներսից: Օրինակ՝ ստեղծենք Student ենթադասը, որը կժառանգի Person դասի բոլոր մեթոդներն ու հատկությունները՝ super() ֆունկցիայով.

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

Ստեղծված Student ենթադասն ունի Person դասի բոլոր մեթոդներն ու հատկությունները: super() ֆունկցիայով կարելի է նաև ենթադասին ավելացնել հատկություններ և մեթոդներ: Օրինակ՝ Student ենթադասում ավելացնենք surname հատկությունը:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def printname(self):
    print(self.name, self.age)
class Student(Person):
  def __init__(self, name, surname, age):
    super().__init__(name, age)
    self.lastname = surname
```

```
std = Student("Aram", "Avagyan", 25)
print(std.lastname)
```

Արդյունքը՝ Avagyan:

Իսկ այժմ Student ենթադասին ավելացնենք նոր՝ Congrats մեթոդը:

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

  def printname(self):
    print(self.name, self.age)

class Student(Person):
  def __init__(self, name, surname, age):
    super().__init__(name, age)
    self.lastname = surname

  def congrats(self):
    print("Congratulations on your graduation!", self.name,
self.lastname)

std = Student("Aram", "Avagyan", 25)
std.congrats()
```

Արդյունքում արտածվում է հետևյալ տեքստը՝

Congratulations on your graduation! Aram Avagyan

Այժմ գրենք կոդ, որն ընդգրկում է ժառանգման տարբեր տեսակներ:

```
# 1. Ժառանգում մեկ ծնող դասից.

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

```
class Employee(Person): # Employee ենթադասը ժառանգում է Person դասից
    def __init__(self, name, salary):
        super().__init__(name)
        self.salary = salary

# 2. Բազմակի ժառանգում՝ ենթադասը ժառանգում է մեկից ավելի ծնող դասերից.

class Job:
    def __init__(self, salary):
        self.salary = salary

class EmployeePersonJob(Employee, Job): # Ժառանգում է Person և Job դասերից
    def __init__(self, name, salary):
        Employee.__init__(self, name, salary)
        Job.__init__(self, salary)

# 3. Բազմամակարդակ ժառանգում

class Manager(EmployeePersonJob): # Ժառանգում է EmployeePersonJob-ից
    def __init__(self, name, salary, department):
        EmployeePersonJob.__init__(self, name, salary)
        self.department = department

# 4. Հիերարխիկ ժառանգում


class AssistantManager(EmployeePersonJob): # Ժառանգում է EmployeePersonJob-ից
    def __init__(self, name, salary, team_size):
        EmployeePersonJob.__init__(self, name, salary)
        self.team_size = team_size

# 5. Հիբրիդային ժառանգում

# Ժառանգում է Manager և AssistantManager-ից
class SeniorManager(Manager, AssistantManager):
```

```
    def __init__(self, name, salary, department, team_size):
        Manager.__init__(self, name, salary, department)
        AssistantManager.__init__(self, name, salary, team_size)

# Օբյեկտների ստեղծում

# 1. ժառանգում մեկ ծնող դասից
emp = Employee("John", 40000)
print(emp.name, emp.salary)

# 2. բազմակի ժառանգում
emp2 = EmployeePersonJob("Alice", 50000)
print(emp2.name, emp2.salary)

# 3. բազմամակարդակ ժառանգում
mgr = Manager("Bob", 60000, "HR")
print(mgr.name, mgr.salary, mgr.department)

# 4. հիերարխիկ ժառանգում
asst_mgr = AssistantManager("Charlie", 45000, 10)
print(asst_mgr.name, asst_mgr.salary, asst_mgr.team_size)

# 5. հիբրիդային ժառանգում
sen_mgr = SeniorManager("David", 70000, "Finance", 20)
print(sen_mgr.name, sen_mgr.salary, sen_mgr.department,
sen_mgr.team_size)
```

15.3. Բազմաձևություն

Օբյեկտ-կողմնորոշված ծրագրավորման հիմնական սկզբունքներից է բազմաձևությունը (պոլիմորֆիզմը)։ Բազմաձևությունը տարբեր տեսակի օբյեկտների համար ընդհանուր ինտերֆեյս օգտագործելու ունակությունն է։ Այն ենթադրում է միևնույն գաղափարի բազմակի իրականացումների հնարավորություն:

Բազմաձևությունը թույլ է տալիս ֆունկցիային, մեթոդին կամ օպերատորին տարբեր կերպ գործել՝ կախված այն օբյեկտի տիպից, որի համար կիրառվում է:

Մեկնաբանենք բազմաձևությունը հետևյալ մատչելի օրինակի միջոցով: Հեռակառավարման վահանակը կարող է կառավարել տարբեր սարքավորումներ, ինչպիսիք են հեռուստացույցը, օդորակիչը կամ երաժշտական համակարգը: Երբ սեղմում եք միացման կոճակը, յուրաքանչյուր սարք տարբեր կերպ է արձագանքում՝ հեռուստացույցը միանում է, օդորակիչը սկսում է սառեցնել, երաժշտական համակարգը նվագարկում է երաժշտություն: Բազմաձևությունը այստեղ նշանակում է նույն ինտերֆեյսը (միացման կոճակ), բայց տարբեր վարքագիծ՝ կախված սարքից (օբյեկտից):

Մենք արդեն հանդիպել ենք Python-ում օպերատորների, ֆունկցիաների, մեթոդների բազմաձևությունների:

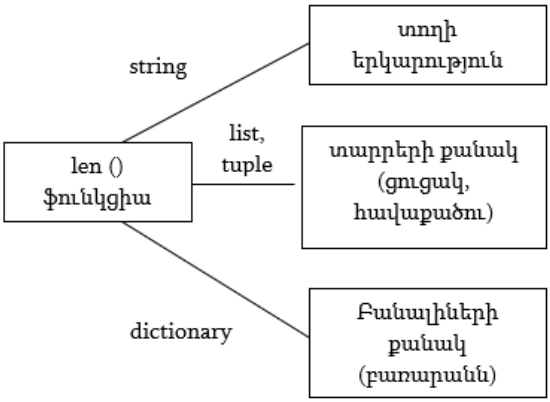
Օրինակ, «+» օպերատորը կատարում է տարբեր գործողություններ տվյալների տարբեր տեսակների հետ: Թվային տվյալների դեպքում «+» օպերատորն օգտագործվում է օպերանդները գումարելու համար, իսկ տողային տվյալների կամ ցուցակների դեպքում՝ կցման համար: Սա Python-ում բազմաձևության ամենապարզ օրինակներից մեկն է:

```
num1 = 1
num2 = 2
print(num1 + num2) #թվերի գումարում
str1 = "Python"
str2 = "Programming"
print(str1+" "+str2) #տողերի կցում
print([1, 2] + [3, 4]) #ցուցակների կցում
```

Արդյունքը՝

```
3
Python Programming
[1, 2, 3, 4]
```

Python-ում կան նաև ներկառուցված ֆունկցիաներ, որոնք կարող են ընդունել տարբեր տիպի արգումենտներ: Այդպիսի ֆունկցիաներից մեկը len() ֆունկցիան է, որը կարող է աշխատել այնպիսի տվյալների հետ, ինչպիսիք են տողը, ցուցակը, հավաքածուն, բազմությունը, բառարանը և վերադարձնել տվյալների յուրաքանչյուր տիպին բնորոշ արդյունք: Տողերի դեպքում len() ֆունկցիան վերադարձնում է տողի սիմվոլների քանակը, ցուցակների և հավաքածուների համար՝ տարրերի քանակը, բառարանների դեպքում՝ բանալիների քանակը (նկար 15.2):



Նկար 15.2. len() ֆունկցիայի աշխատանքը տարբեր տիպի տվյալների հետ

```
print(len("University"))
print(len(["Fortran", "Assembler", "C++"]))
print(len({"Name": "Mary", "Age": 21}))
mytuple = ("pen", "book", "notebook")
print(len(mytuple))
```

Արդյունքը՝

```
10
3
2
```

3

Բազմաձևությունը հաճախ հանդիպում է դասի մեթոդներում, որտեղ տարբեր դասեր կարող են ունենալ նույն անունով մեթոդ: Օրինակ՝ ենթադրենք ունենք չորս դաս՝ Person, Student, Teacher, Doctor, և դրանք բոլորն էլ ունեն activity() անունով մեթոդ: Ուսանողի հիմնական գործունեությունը սովորելն է, ուսուցչինը՝ սովորեցնելը, բժշկինը՝ բուժելը:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def activity(self):
    print("Type of activity!")

class Student:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def activity(self):
    print("Learn!")

class Teacher:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def activity(self):
    print("Teach!")

class Doctor:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def activity(self):
    print("Treat!")
```

```
person1 = Person("Avag", 25) #ստեղծվում է Person դասի օբյեկտ
student1 = Student("Marina", 18) #ստեղծվում է Student դասի օբյեկտ
teacher1 = Teacher("Gagik", 47) #ստեղծվում է Teacher դասի օբյեկտ
doctor1=Doctor("Aram",42) #ստեղծվում է Doctor դասի օբյեկտ

for prs in (person1, student1, teacher1, doctor1):
  prs.activity()
```

Արդյունքը՝

```
Type of activity!
Learn!
Teach!
Treat!
```

Բազմաձևության շնորհիվ activity() նույն մեթոդը կիրառվել է բոլոր երեք դասերի համար։

Բազմաձևության ամենատարածված տեսակներից են գերբեռնումը (overloading) և վերասահմանումը (overriding)։ Վերասահմանումը ենթադրում է ենթադասի՝ ծնող դասից ժառանգած մեթոդի վարքագծի փոփոխություն, մինչդեռ ծանրաբեռնումը թույլ է տալիս ստեղծել մեթոդի մի քանի տարբերակ նույն անունով, բայց տարբեր պարամետրերով։ Այս տեխնիկաները հաճախ օգտագործվում են կոդը տարբեր սցենարներին հարմարեցնելու և տարբեր տեսակի օբյեկտների հետ միատեսակ կերպով աշխատելու հնարավորություն ապահովելու համար։

Վերասահմանման միջոցով ենթադասն ապահովում է ծնող դասում արդեն սահմանված մեթոդի իր սեփական իրականացումը (նույն անունով և պարամետրերով)։ Այս մեխանիզմը թույլ է տալիս ենթադասին փոփոխելու կամ ընդլայնելու ժառանգված մեթոդի վարքագիծը։ Վերասահմանումը օգտագործվում է միայն դասերի հիերարխիայում և հնարավոր է միայն այն մեթոդների համար, որոնք ենթադասը ժառանգում է ծնող դասից։ Բացի դրա-

նից՝ ենթադասի մեթոդը պետք է ունենա նույն անունն ու պարամետրերը, ինչ ծնող դասի մեթոդը։

Դիտարկենք հետևյալ ծրագիրը։

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def activity(self):
        print("Type of activity!")

class Student(Person):
    pass #Student ենթադասը ժառանգում է ծնող դասի մեթոդներն ու ատրիբուտները

class Teacher(Person):
    def activity(self): # ծնող դասի activity() մեթոդի վերասահմանում
        print("Teach!")

class Doctor(Person):
    def activity(self): # ծնող դասի activity() մեթոդի վերասահմանում
        print("Treat!")

person1 = Person("Avag", 25) #Ստեղծվում է person1 օբյեկտը
student1 = Student("Marina", 18) #Ստեղծվում է student1 օբյեկտը
teacher1 = Teacher("Gagik", 47) #Ստեղծվում է teacher1 օբյեկտը
doctor1=Doctor("Aram",42) #Ստեղծվում է doctor1 օբյեկտը

for prs in (person1, student1, teacher1, doctor1):
    print(prs.name, prs.age)
    prs.activity()
```

Արդյունքը՝

```
Avag 25
Type of activity!
Marina 18
Type of activity!
Gagik 47
Teach!
Aram 42
Treat!
```

Ներկայացված ծրագրում Student, Teacher և Doctor ենթադասերը ժառանգում են Person ծնող դասի հատկություններն ու մեթոդները: Student ենթադասը Person դասից ժառանգում է name, age հատկություններն ու activity() մեթոդը: Teacher և Doctor ենթադասերը նույնպես ժառանգում են name, age հատկությունները և activity() մեթոդը, բայց երկուսն էլ վերասահմանում են այդ մեթոդը:

Ստորև բերված ծրագրում իրականացված են Shape, Rectangle, Circle դասերը, որտեղ Shape-ը հիմնական դաս է Rectangle, Circle դասերի համար, իսկ դրանից ժառանգված դասերի օբյեկտները երկրաչափական պատկերներ են: Rectangle դասի օբյեկտները ուղղանկյուններ են, իսկ Circle դասի օբյեկտները՝ շրջաններ:

```
import math
class Shape:
  def area(self):
    pass
  def perimeter(self):
    pass

class Rectangle(Shape):
  def __init__(self, length, width):
    self.length = length
    self.width = width
  def area(self): #area() մեթոդի վերասահմանում
    return self.length * self.width
```

```
  def perimeter(self): #perimeter() մեթոդի վերասահմանում
    return 2 * (self.length + self.width)
class Circle(Shape):
  def __init__(self, radius):
    self.radius = radius
  def area(self): #area() մեթոդի վերասահմանում
    return math.pi * self.radius * self.radius
  def perimeter(self): #perimeter() մեթոդի վերասահմանում
    return 2 * math.pi * self.radius

# Օբյեկտների ստեղծում
radius=float(input("Input the radius of the circle: "))
a=float(input("Input the length of the rectangle: "))
b=float(input("Input the width of the rectangle: "))

rectangle = Rectangle(a, b) #ստեղծում է ուղղանկյուն a և b կող-
մերով
circle = Circle(radius) #ստեղծում է radius շառավղով շրջան

shapes = [rectangle, circle]
for shape in shapes:
  print("Area:", shape.area())
  print("Perimeter:", shape.perimeter())
```

Արդյունքը՝

```
Input the radius of the circle: 3
Input the length of the rectangle: 5
Input the width of the rectangle: 4
Area: 20.0
Perimeter: 18.0
Area: 28.274333882308138
Perimeter: 18.84955592153876
```

Ծրագրում Shape դասում սահմանված են պատկերի մակերեսն ու պարագիծը հաշվող ֆունկցիաներ, որոնց մարմինը դատարկ է: Rectangle և Circle ենթադասերը ժառանգում են Shape

ծնող դասից area() և perimeter() մեթոդները և վերասահմանում դրանք։

Երբեմն անհրաժեշտ է ոչ թե ամբողջությամբ փոխել ծնողական մեթոդի վարքագիծը, այլ լրացնել այն։ Նման դեպքերում կարելի է օգտագործել super() ֆունկցիան՝ ենթադասից ծնող դասի մեթոդը կանչելու համար։

Գերբեռնումը (overloading) մեխանիզմ է, որը թույլ է տալիս ստեղծել մի քանի մեթոդներ նույն անունով, բայց տարբեր պարամետրերով։ Python-ում մեթոդների գերբեռնումը իրականացվում է լռելյայն պարամետրերի կամ *args և **kwargs պարամետրերի միջոցով։

Հետևյալ ծրագրում մեթոդների գերբեռնումը իրականացվում է լռելյայն պարամետրերի միջոցով։ print_data() մեթոդը կարող է կանչվել մեկ կամ երկու պարամետրով։

```
class Printer:
 def print_data(self, data, times=1):
    for _ in range(times):
       print(data)

printer = Printer()
printer.print_data("Hello") # արտածում է "Hello" բառը մեկ անգամ
printer.print_data("Hello", 3) # արտածում է "Hello" բառը երեք անգամ
```

Արդյունքը՝

```
Hello
Hello
Hello
Hello
```

Հետևյալ ծրագրում add() մեթոդն ընդունում է ցանկացած քանակությամբ արգումենտներ։

```
class Calculator:
```

```
    def add(self, *args):
        return sum(args)
calc = Calculator()
print(calc.add(2, 3)) # արտածում է 5
print(calc.add(1, 2, 3, 4, 5)) # արտածում է 15
```

15.4. Թաղանթապատում

Python ծրագրավորման մեջ ծրագրի ամբողջականության և անվտանգության ապահովման համար խիստ կարևորվում է դասերում տվյալների պաշտպանությունը:

Թաղանթապատումը (Encapsulation) տվյալների պաշտպանության հիմնարար սկզբունք է, որը միավորում է կոդն ու տվյալները և դրանց պաշտպանում չարտոնված մուտքերից ու պատահական փոփոխություններից: Կոդի և տվյալների միավորումով է հնարավոր դառնում օբյեկտի ստեղծումը: Տեխնիկապես, թաղանթապատումը օբյեկտ-կողմնորոշված ծրագրավորման սկզբունք է, որտեղ տվյալները (փոփոխականները) ու մեթոդները (ֆունկցիաները) միավորված են մեկ դասում: Թաղանթապատումը վերահսկում է տվյալների թարմացումները, բարելավում է մոդուլայնությունը՝ թաքցնելով ներքին իրականացման մանրամասները:

Այսպիսով, թաղանթապատումը տվյալների պաշտպանության մեխանիզմ է, որը ներառում է.

- դասի ներքին իրականացման թաքցնում,
- դասի անդամներին հասանելիության վերահսկում:

Դասում փոփոխականները կամ մեթոդները կարող են լինել բաց (Public), պաշտպանված (Protected), փակ (Private):

Բաց (Public) փոփոխականներն ու մեթոդները հասանելի են ինչպես դասի ներսում, այնպես էլ դասից դուրս կամ այլ մոդուլներից: Python-ում բոլոր փոփոխականները լռելյայն բաց են: Դա նշանակում է, որ ծրագրի ցանկացած մասից կարելի է դիմել օբյեկտի ատրիբուտին և փոխել այն:

Public տիպի փոփոխականները սահմանվում են առանց որևէ նախածանցի (օրինակ՝ self.name): Հետևյալ օրինակը ցույց է

տալիս, թե ինչպես կարելի է դասից դուրս, ծրագրի մեկ այլ մասից դիմել Public տիպի ատրիբուտին (name, age) և փոխել այն։

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

    def print_person(self):
        print("Name:", self.name)
        print("Age: ", self.age)

prs = Person("Aram", 39)
prs.name = "Gagik" # փոխվում է name ատրիբուտի արժեքը
prs.age = 23 # փոխվում է age ատրիբուտի արժեքը
prs.print_person()
```

Արդյունքը՝

```
Name: Gagik
Age:  23
```

Ներկայացված ծրագրում name և age փոփոխականները Public տիպի են և հասանելի են թե՛ Person դասի ներսում, թե՛ դասից դուրս։ print_person() մեթոդը ևս Public տիպի է և հասանելի է կոդի ցանկացած մասից։ Այս մեթոդը տպում է name և age ատրիբուտների արժեքները։

Ծրագրում ստեղծվում է Person դասի կոնկրետ ներկայացուցիչ՝ prs օբյեկտը, որից հետո փոխվում են դրա ատրիբուտների արժեքները։ Սա հնարավոր է դառնում այն պատճառով, որ name և age ատրիբուտները Public տիպի են։ Այնուհետև print_person() մեթոդով տպվում են prs օբյեկտի ատրիբուտների փոփոխված արժեքները։

Փոփոխականները և մեթոդները կարող են լինել պաշտպանված (Protected)։ Այդպիսի մեթոդներն ու փոփոխականները հասանելի են միայն դասում և դրա ենթադասերում։ Python-ում

պաշտպանված անդամները սահմանվում են նախածանցով՝ մեկ ընդգծման նշանով (օրինակ՝ self._ age):

Հետևյալ օրինակում ցույց է տրվում, որ Protected տիպի անդամները նախատեսված են դասում և ենթադասերում օգտագործելու համար:

```
class Person:
    def __init__(self):
        self._age = 23 # Protected տիպի ատրիբուտ

class Employee(Person):
    def display_age(self):
        print(self._age) # հասանելի է ենթադասում

emp = Employee() # օբյեկտի ստեղծում
emp.display_age()
```

Արդյունքը՝ 23:

Բերված կոդում _age-ը պաշտպանված ատրիբուտ է: Քանի որ Employee ենթադասը ժառանգում է Person դասից, ուստի _age ատրիբուտը հասանելի է այս ենթադասում: display_age() մեթոդը տպում է _age պաշտպանված փոփոխականի արժեքը և հասանելի է դասում ու ենթադասում:

Փակ անդամները (Private տիպ) փոփոխականներ կամ մեթոդներ են, որոնք օգտագործվում են դասի ներսում և ուղղակիորեն հասանելի չեն դրսից: Python-ում փակ անդամները սահմանվում են նախածանցով՝ կրկնակի ընդգծման նշանով (օրինակ՝ self.__name):

Դիտարկենք հետևյալ ծրագիրը, որում salary ատրիբուտը հայտարարված է Private տիպի:

```
class PrivateSalary:
    def __init__(self):
        self.__salary = 50000 # Private տիպի ատրիբուտ
    def salary(self):
        return self.__salary
```

```
prs = PrivateSalary()
print(prs.__salary) # ծագում է AttributeError տիպի սխալ
```

Արդյունքը՝

```
Traceback (most recent call last):
  File "C:/Examples/class_4.py", line 7, in <module>
    print(prs.__salary)  # ծագում է AttributeError տիպի սխալ
AttributeError: 'PrivateSalary' object has no attribute '__salary'. Did you mean: 'salary'?
```

Ծրագրում ստեղծվել է PrivateSalary անունով դասը, որում salary ատրիբուտը փակ տիպի է, ուրեմն այն կարող է հասանելի լինել միայն դասի ներսում: Ապա ստեղծվում է PrivateSalary դասի ներկայացուցիչ՝ prs օբյեկտը: salary() մեթոդը Public տիպի մեթոդ է, որը վերադարձնում է __salary ատրիբուտի արժեքը: Ծրագրում print(prs.__salary) հրամանն առաջացնում է AttributeError սխալ, քանի որ __salary փոփոխականը Private տիպի է, ուստի և հասանելի չէ դասի դրսից:

getter և setter մեթոդները:

Python-ում getter և setter մեթոդներն օգտագործվում են Private տիպի ատրիբուտներին հասանելիություն ապահովելու և դրանք փոփոխելու համար:

Հետևյալ օրինակը ցույց է տալիս, թե ինչպես կիրառել getter և setter մեթոդները՝ Private տիպի ատրիբուտին (__salary) անվտանգորեն դիմելու և այն թարմացնելու համար:

```
class Employee:
    def __init__(self):
        self.__salary = 100000 # Private փոփոխական

    def get_salary(self): # getter մեթոդը
        return self.__salary

    def set_salary(self, amount): # setter մեթոդը
        if amount > 0:
```

```
            self.__salary = amount
        else:
            print("Invalid salary amount!")

emp = Employee()
print(emp.get_salary())

emp.set_salary(60000)
print(emp.get_salary())
```

Արդյունքը՝

```
100000
60000
```

Ծրագրում ստեղծվել է Employee դասի emp օբյեկտը: __salary-ն Private տիպի ատրիբուտ է, ուստի այն ուղղակիորեն հասանելի չէ դասի դրսից: get_salary() և set_salary() մեթոդներն ապահովում են __salary փակ փոփոխականին հասանելիություն և դրա արժեքի փոփոխում դասի դրսից՝ միաժամանակ տվյալները պաշտպանված պահելով:

15.5. Տվյալների աբստրակցիան Python-ում

Python-ում տվյալների աբստրակցիան (վերացարկումը) թույլ է տալիս թաքցնել ծրագրի իրականացման բարդ ներքին մանրամասները և ցույց տալ միայն էական կողմերը, ինչը կոդը դարձնում է ավելի պարզ, նվազեցնում է պատահական փոփոխությունների հավանականությունը, հեշտացնում է կոդի բաղադրիչների թարմացումը կամ փոփոխումը:

Դասերի հիերարխիա ստեղծելիս անհրաժեշտ է լինում, որ մի շարք դասեր աջակցեն նույն ինտերֆեյսին՝ նույն ատրիբուտների և մեթոդների հավաքածուին: Այս խնդիրը կարող է մասամբ լուծվել ժառանգման միջոցով, սակայն ծնող դասի մեթոդի իրականացումը միշտ չէ, որ հարմար է ենթադասերի համար: Այս խնդիրը լուծվում է աբստրակտ դասի միջոցով, որը սահմանում է

ընդհանուր ինտերֆեյս ենթադասերի հավաքածուի համար՝ տրամադրելով բոլոր ենթադասերին ընդհանուր ատրիբուտներ ու մեթոդներ: Այն նաև հնարավորություն է տալիս ենթադասերին իրականացնել աբստրակտ մեթոդները:

Python լեզվում աբստրակցիան իրականացվում է աբստրակտ բազային դասերի (abstract base classes-ABC's) և abc մոդուլի միջոցով: ABC աբստրակտ բազային դասը իր ենթադասերի համար սահմանում է ընդհանուր ինտերֆեյս, որը ծառայում է որպես նախագիծ այլ դասերի համար և թույլ է տալիս ենթադասերում իրականացնել մեթոդները՝ թաքցնելով ներքին բարդ տրամաբանությունը:

@abstractmethod դեկորատորը (@abstractmethod decorator) օգտագործվում է աբստրակտ բազային դասի (ABC) ներսում աբստրակտ մեթոդ հայտարարելու համար, որը պետք է իրականացվի ցանկացած կոնկրետ ենթադասի կողմից:

Շարահյուսությունը՝

```
from abc import ABC, abstractmethod
class MyClass(ABC):
    @abstractmethod
    def my_method(self):
        pass
```

որտեղ MyClass-ը աբստրակտ դաս է, my_method()-ը աբստրակտ մեթոդ է, որը պետք է իրականացվի ենթադասերում: Այս սկզբունքը թույլ է տալիս ենթադասերին սահմանել իրենց սեփական վարքագիծը: Օրինակ՝

```
from abc import ABC, abstractmethod
class Person(ABC):
    @abstractmethod
    def activity(self): # Abstract մեթոդ
        pass

class Student(Person):
    def activity(self):
```

```
        return "I am a student!"

std = Student()
print(std.activity())
```

Արդյունքը՝

```
I am a student!
```

Այս ծրագրում Person-ը աբստրակտ դաս է՝ աբստրակտ activity() մեթոդով, որը չունի իրականացում (դասի մարմինը դատարկ է): Student ենթադասն իրականացնում է այս մեթոդը և վերադարձնում է "I am a student!" տեքստը:

Python-ում աբստրակցիան իրականացվում է հետևյալ հիմնական բաղադրիչների միջոցով՝ աբստրակտ մեթոդներ (abstract methods), կոնկրետ մեթոդներ (concrete methods), աբստրակտ հատկություններ (abstract properties)**:** Այս տարրերը միասին աշխատում են ենթադասերի համար հստակ կառուցվածք սահմանելու նպատակով՝ միաժամանակ թաքցնելով իրականացման մանրամասները:

Աբստրակտ մեթոդները հայտարարագրեր են, որոնք չունեն աբստրակտ դասի ներսում սահմանված մարմին: Դրանց հիման վրա ենթադասերը ստեղծում են իրենց սեփական իրականացումները: Օրինակ՝

```
from abc import ABC, abstractmethod
class Person(ABC):
    @abstractmethod
    def activity(self): # աբստրակտ մեթոդ
        pass
```

Այստեղ activity()-ն Person դասի աբստրակտ մեթոդ է, որը չունի աբստրակտ դասի ներսում սահմանված մարմին: Նշենք, որ եթե ենթադասը չի իրականացնում աբստրակտ դասի բոլոր աբստրակտ մեթոդները, ապա Python-ն առաջացնում է TypeError՝ այդ ենթադասի օբյեկտ ստեղծելու ժամանակ:

Կոնկրետ մեթոդները լիովին իրականացված մեթոդներ են աբստրակտ դասի շրջանակում: Ենթադասը կարող է ժառանգել և օգտագործել դրանք անմիջապես՝ խթանելով կոդի վերօգտագործումը առանց ընդհանուր ֆունկցիոնալությունը վերաասահմանելու անհրաժեշտության: Օրինակ՝

```
from abc import ABC, abstractmethod
class Person(ABC):
    @abstractmethod
    def activity(self): # աբստրակտ մեթոդ
        pass

    def greeting(self): # կոնկրետ մեթոդ
        return "Hello!"
```

greeting() մեթոդը Person դասի կոնկրետ մեթոդ է, որն իրականացված է և կարիք չունի, որ այն վերաասահմանվի ենթադասի կողմից:

Աբստրակտ հատկությունները գործում են ինչպես աբստրակտ մեթոդները: Աբստրակտ հատկությունները հայտարարվում են @property դեկորատորով և նշվում են որպես աբստրակտ՝ օգտագործելով @abstractmethod-ը: Ենթադասը պետք է իրականացնի այս հատկությունները: Օրինակ՝

```
from abc import ABC, abstractmethod
class Person(ABC):
    @property
    @abstractmethod
    def Faculty(self): # աբստրակտ հատկություն
        pass

class Student(Person):
    @property
    def Faculty(self):
        return "Economics"

std = Student()
```

```
print(std.Faculty)
```

Արդյունքը՝

```
Economics
```

Այստեղ Faculty-ն Person դասի աբստրակտ հատկություն է, քանի որ հայտարարված է որպես @property և @abstractmethod: Student դասն իրականացնում է Faculty հատկությունը:

Աբստրակտ դասի համար չեն կարող ուղղակիորեն օբյեկտներ ստեղծվել, քանի որ աբստրակտ դասը պարունակում է աբստրակտ մեթոդներ կամ հատկություններ, որոնք չունեն իրականացումներ: Աբստրակտ դասի համար օբյեկտ ստեղծելու փորձը հանգեցնում է TypeError-ի: Օրինակ՝

```
from abc import ABC, abstractmethod
class Person(ABC):
    @abstractmethod
    def activity(self):
        pass
prs = Person()
```

Արդյունքը՝

```
Traceback (most recent call last):
  File "C:/Examples/class_5.py", line 17, in <module>
    class Person(ABC):
NameError: name 'ABC' is not defined
```

Այստեղ Person դասն աբստրակտ է, որը պարունակում է activity() աբստրակտ մեթոդը: Person դասի prs օբյեկտի ստեղծման ժամանակ ծագել է «name 'ABC' is not defined», սխալը, քանի որ այդ աբստրակտ դասի activity() մեթոդն իրականացված չէ, ուստի օբյեկտ ստեղծել հնարավոր չէ:

ԳԼՈՒԽ 16. PYTHON-Ի ԱՐԴԻ ԿԻՐԱՌՈՒԹՅՈՒՆՆԵՐ

Python-ն ունի կիրառման լայն հնարավորություններ՝ շնորհիվ իր հզոր մասնագիտացված գրադարանների, որոնցից ամենատարածվածներն են՝ Pillow, Math, Numpy, Django, Pandas, Spark, Matplotlib, Tensorflow, Natural Language Toolkit (NLTK), Scikit-Learn, PyTorch, OpenCV և այլն:

Python-ը կիրառվում է բազմաթիվ ոլորտներում, այդ թվում՝ գիտական հաշվարկային ծրագրերի մշակման, խաղերի ստեղծման, ռոբոտատեխնիկայի, կիբերանվտանգության, արհեստական բանականության (Artificial Intelligence), մեքենայական ուսուցման (Machine Learning), տվյալագիտության (Data Science), պատկերների մշակման (Image Processing), Backend վեբ ծրագրավորման մեջ: Python-ն ունի Framework-եր՝ վեբ կայքեր պատրաստելու համար, որոնցից ամենատարածվածներն են Django-ն և Flask-ը:

Անդրադառնանք Python-ի արդի կիրառություններին տվյալագիտության, տվյալների վերլուծության, մեքենայական ուսուցման, տվյալների ճարտարագիտության, վեբ ծրագրավորման, ծրագրային ապահովման մշակման ոլորտներում:

Python-ը տվյալների վերլուծության համար: Տվյալները ներկայումս հանդիսանում են թանկարժեք ակտիվ, և ընկերությունների մեծ մասը հետաքրքրված է համապատասխան տվյալների հավաքագրմամբ ու վերլուծությամբ: Այստեղ Python-ը գերազանցում է ցանկացած մրցակցություն:

Python-ը տվյալների վերլուծության նպատակով, իր համապարփակ ստանդարտ գրադարանից բացի, տրամադրում է նաև լրացուցիչ մոդուլների հավաքածու, որը ստեղծված է հատուկ վերլուծության նպատակներով:

Տվյալների վերլուծության ամենահայտնի Python գրադարաններն են pandas-ը և NumPy-ն: Այս գործիքները թույլ են տալիս անել գրեթե ամեն ինչ տվյալների հետ՝ ինչպիսին են դրանց մաքրումն ու մշակումը, վիճակագրության ուսումնասիրությունը կամ տվյալներում թաքնված միտումների վիզուալիզացիան:

Այս երկու գրադարաններից բացի՝ կարելի է օգտագործել նաև այլ գրադարաններ տվյալների հետ կապված տարբեր առաջադրանքների համար, ինչպիսին են տվյալների վիզուալիզացիան, web scraping կամ վարկածների թեստավորումը:

Python-ը տվյալների վիզուալիզացիայի համար: Տվյալների վիզուալիզացիան տվյալների վերլուծության ինքնուրույն մաս է, որն օգնում է ներկայացնել տվյալները: Python-ն առաջարկում է տվյալների վիզուալիզացիայի գործիքների լայն տեսականի: Դրանցից ամենատարածվածներն են matplotlib-ը և seaborn-ը, որոնք թույլ են տալիս ստեղծել տարբեր տեսակի գրաֆիկներ ու դիագրամներ հաշվետվությունների և հետազոտական վերլուծության համար՝ գծապատկերներ, սյունակային դիագրամներ, հիստոգրամներ, անիմացիոն գրաֆիկներ, քլաստերային քարտեզներ և այլն:

Python-ում կան նաև այլ գրաֆիկական գրադարաններ՝ թե՛ բազմաֆունկցիոնալ (օրինակ՝ Plotly, Bokeh կամ Altair), թե՛ ավելի մասնագիտացված (missingno՝ բացակայող արժեքների վիզուալիզացիայի համար, Toyplot՝ էլեկտրոնային հրատարակչության ինտերակտիվ գծապատկերներ ստեղծելու համար, GeoPandas՝ քարտեզներ կառուցելու համար և այլն):

Python-ը մեքենայական ուսուցման համար: Մեքենայական ուսուցումը (ML) ընկած է տվյալագիտության խնդիրների մեծ մասի հիմքում: Այն ներկայացնում է արհեստական բանականության (AI) ոլորտ, որն ալգորիթմների օգտագործմամբ տալիս է մեքենաներին պատմական տվյալների հիման վրա օրինաչափություններ ու միտումներ սովորելու ունակություն՝ անհայտ տվյալների վերաբերյալ կանխատեսումներ կատարելու համար:

Մեքենայական ուսուցումը լայնորեն կիրառվում է բիզնեսում, բժշկության մեջ, ֆինանսական ոլորտում, ռոբոտատեխնիկայում, խաղերի ծրագրավորման ոլորտում, պատկերների մշակման և տեքստերի վերլուծության խնդիրներում: Այժմ Python ծրագրավորման լեզուն դարձել է մեքենայական ուսուցման և արհեստական բանականության նախագծերի ծրագրավորման հիմնական լեզու:

Օգտագործելով ML տեխնոլոգիաները՝ կարելի է ստեղծել մոդելներ, որոնք կարող են, օրինակ, ճշգրտորեն կանխատեսել ընկերության հաճախորդների արտահոսքի մակարդակը, գնահատել անձի՝ որոշակի հիվանդություն ունենալու ռիսկը, կանխատեսել խարդախ բանկային գործարքները և այլն: Ամենատարածված Python ML գրադարաններն են՝ Scikitlearn-ը, Keras-ը, TensorFlow-ը և PyTorch-ը:

Տվյալագիտության տեխնոլոգիան անհրաժեշտ պահանջ է նաև բանկերի համար՝ մրցակցությանը դիմակայելու, ավելի շատ հաճախորդներ ներգրավելու, առկա հաճախորդների հավատարմությունը մեծացնելու, տվյալների վրա հիմնված ավելի արդյունավետ որոշումներ կայացնելու, բիզնեսը հզորացնելու, գործառնական արդյունավետությունը բարձրացնելու, առկա ծառայությունները/արտադրանքը բարելավելու և նորերը ներդնելու, անվտանգությունը ամրապնդելու համար:

Տվյալագիտության տեխնոլոգիան թույլ է տալիս բանկային ոլորտին հաջողությամբ կատարել բազմաթիվ առաջադրանքներ, այդ թվում՝ ներդրումային ռիսկերի վերլուծություն, հաճախորդների սեգմենտացիա, հաճախորդների արտահոսքի մակարդակի կանխատեսում, անհատականացված մարքեթինգ, հաճախորդի տրամադրվածության վերլուծություն, վիրտուալ օգնականներ և չաթբոտեր, խարդախության ռիսկերի բացահայտում:

Մեքենայական ուսուցման հիմնական ալգորիթմների տեսակներից են.

- Վերահսկվող (supervised) մեքենայական ուսուցումը, որի մեթոդներից են՝ k-ամենամոտ հարևանի մեթոդը, լոգիստիկ ռեգրեսիան, օժանդակ վեկտորային մեքենաները, որոշումների ծառը, պատահական անտառը, ժամանակային շարքերի վերլուծությունը, նեյրոնային ցանցերը և այլն:
- Չվերահսկվող (unsupervised) մեքենայական ուսուցումը, որի մեթոդներից են՝ քլաստերային վերլուծությունը, կապի վերլուծությունը, ինքնակազմակերպվող քարտեզները, գլխավոր բաղադրիչների վերլուծությունը, անոմալիաների ճանաչումը և այլն:

Python-ը արհեստական բանականության և խորը ուսուցման մեջ: Արհեստական բանականությունը կարևոր տեխնոլոգիական և գիտական ոլորտ է, որն արագորեն վերափոխում է մեր աշխարհը:

Արհեստական բանականության հիմնական ուղղություններից են արհեստական նեյրոնային ցանցերը (ANN, Artificial Neural Networks), որոնք մարդու ուղեղի կենսաբանական նեյրոնային ցանցի պարզեցված մոդելներ են: Դրանք բաղկացած են բազմաթիվ արհեստական նեյրոններից, որոնք փոխկապակցված են և օգտագործվում են մեքենայական ուսուցման և խորը ուսուցման առաջադրանքներ կատարելու համար:

Python-ը կենտրոնական դեր է խաղում արհեստական բանականության և խորը ուսուցման մեջ՝ խթանելով բնական լեզվի մշակման (NLP) և համակարգչային տեսողության տեխնոլոգիաների զարգացումը: Այս տիրույթները զգալի աճ են ապրում, և գրադարանների ու framework-երի հարուստ էկոհամակարգը Python-ը դարձնում է արհեստական բանականության լուծումներ մշակելու համար նախընտրելի լեզու:

SpaCy, NLTK և Hugging Face Transformers-ի նման գրադարանները հնարավորություն են տալիս կատարել այնպիսի առաջադրանքներ, ինչպիսիք են տրամադրության վերլուծությունը, լեզվի թարգմանությունը, տեքստի ամփոփումը և չաթբոտի մշակումը: Python-ի բազմակողմանիությունը թույլ է տալիս հետազոտողներին և մշակողներին ներդնել ժամանակակից մոդելներ և կատարելագործել նախապես պատրաստված մոդելները հատուկ կիրառությունների համար:

Python-ը համակարգչային տեսողության նորարարության առաջատարն է՝ OpenCV, PyTorch և TensorFlow գրադարաններով: Այս գործիքներն օգտագործվում են դեմքի ճանաչման, օբյեկտների հայտնաբերման, պատկերների դասակարգման և տեսանյութերի վերլուծության համար կիրառություններ մշակելու նպատակով: Python-ի օգտագործման հեշտությունը և մոդելների առկայությունը այն հասանելի են դարձնում ինչպես սկսնակների, այնպես էլ մասնագետների համար:

Թվարկենք տարբեր ոլորտներում արհեստական բանականության հաջողված նախագծերի մի քանի օրինակներ։

1. Բնական լեզվի մշակում՝ GPT (Generative Pretrained Transformer) OpenAI-ի կողմից ստեղծված արհեստական բանականության հզորագույն մոդելների շարք է։ Սկսած պատմական GPT-3-ից մինչև նորագույն GPT-5.5 մոդելները՝ տիրապետում են բարդ տրամաբանության, ստեղծում են բնական տեքստ, պատասխանում են բարդ հարցերի, գրում հոդվածներ և կոդավորում բազմաթիվ լեզուներով։ GPT-5.5-ը մասնագիտացված է բարդ, բազմաքայլ խնդիրների լուծման, «ագենտային» (agentic) ինքնուրույն աշխատանքի, առաջադեմ ծրագրավորման և գիտական տեքստերի վերլուծության մեջ։

2. Համակարգչային տեսողություն՝ Proximus AI (անօդաչու թռչող սարքեր գյուղատնտեսության համար)։ Այս նախագիծն օգտագործում է համակարգչային տեսողությամբ անօդաչու սարքեր՝ օգնելու ֆերմերներին բարելավել իրենց դաշտերի կառավարումը, հայտնաբերել բույսերի հիվանդությունները և օպտիմալացնել բերքատվությունը։

3. Առողջապահություն՝ IBM Watson։ IBM Watson-ը օգտագործվում է ուռուցքաբանության մեջ՝ բժշկական տվյալները վերլուծելու և քաղցկեղի բուժման առաջարկներ տրամադրելու համար։ Այն օգնում է բժիշկներին ընտրել հիվանդների համար լավագույն բուժումները։

4. Ինքնավար տրանսպորտային միջոցներ՝ Waymo։ Այն մշակում է ինքնավար մեքենաների տեխնոլոգիա, որը հաջողությամբ փորձարկվել է ճանապարհներին և օգտագործվում է առանց վարորդի տաքսիների փորձնական նախագծերում։

5. Ֆինանսներ՝ Alibaba-ի Ant Financial (Alipay), որն օգտագործում է արհեստական բանականությունը գործարքները վերլուծելու, խարդախ գործարքները հայտնաբերելու և ֆինանսական ծառայություններ մատուցելու համար։

6. Առաջարկների համակարգեր՝ Netflix-ի առաջարկությունների համակարգ։ Netflix-ն օգտագործում է արհեստական բանականությունը ֆիլմեր և հեռուստաշոուներ առաջարկելու համար՝ հիմնվելով օգտատիրոջ դիտումների պատմության և նմանատիպ հետաքրքրությունների վրա։

7. Խաղեր՝ DeepMind-ի AlphaGo: Այն դարձավ առաջին ծրագիրը, որը հաղթեց աշխարհի չեմպիոնին Go-ում՝ որը հին խաղ է և համարվում է արհեստական բանականության համար ամենադժվարներից մեկը:

8. Ռոբոտաշինություն՝ Boston Dynamics-ի Spot: Այս ռոբոտն օգտագործվում է տարբեր ոլորտներում՝ ներառյալ սարքավորումների տեղադրումը և սպասարկումը, վտանգների ստուգումը և նույնիսկ զվարճանքի արդյունաբերությունը:

9. Կրթություն՝ Duolingo: Duolingo-ն օգտագործում է արհեստական բանականության տեխնոլոգիաներ անհատականացված ուսուցման և լեզվի գնահատման համար:

10. Գիտական հետազոտություններ՝ DeepMind-ի AlphaFold: AlphaFold-ը օգտագործում է խորը ուսուցում՝ սպիտակուցային կառուցվածքները կանխատեսելու համար, ինչը հիմնարար նշանակություն ունի կենսաբանական և բժշկական հետազոտություններում:

Սրանք տարբեր ոլորտներում հաջողված նախագծերի ընդամենը մի քանի օրինակներ են, և դրանց ցանկն անընդհատ ընդլայնվում է:

Python-ը ծրագրային ապահովման մշակման համար: Բացի տվյալագիտության ոլորտներում բազմակողմանի կիրառությունից՝ Python-ը օգտագործվում է ծրագրային ապահովման մշակման մեջ՝ ներառյալ ավտոմատացված անընդհատ կոմպիլյացիան, վերահսկողությունը, նախատիպերի ստեղծումը, սխալների հայտնաբերումը, փորձարկումը և ծրագրային ապահովման սպասարկումը:

Python լեզվի ճկունությունն ու հզորությունը թույլ են տալիս ստեղծել բարդ թվային հաշվարկներ ներառող ծրագրեր:

Python-ի միջոցով կարելի է ստեղծել աուդիո կամ վիդեո ծրագրեր՝ հիմնվելով արհեստական բանականության կամ մեքենայական ուսուցման տեխնիկայի, API-ների (Application Programming Interface - Կիրառական ծրագրավորման ինտերֆեյս), GUI-ների (Graphical User Interface - գրաֆիկական օգտա-

գործողի ինտերֆեյս) կամ ցանկացած այլ ծրագրային ապահովման վրա։

Python-ը վեբ մշակման համար։ Python-ը օգտագործվում է կայքի back end ծրագրավորման համար՝ կիրառելով որոշ տարածված framework-եր (օրինակ՝ Django կամ Flask), որոնք ունեն մասնագիտացված ներկառուցված մոդուլներ, որոնք հնարավորություն են տալիս տվյալներ փոխանակել սերվերների հետ, մշակել տեղեկատվություն, մուտք գործել տվյալների բազաներ, URL երթուղի (Uniform Resource Locator), բովանդակության կառավարում և կայքի անվտանգության պահպանում։ Վերջին տարիներին աճում է FastAPI-ի օգտագործումը API-ներ մշակելու համար դրա արդյունավետության և օգտագործման հեշտության շնորհիվ։

Python-ի առավելություններն օգտագործվում են ինչպես ստարտափների, այնպես էլ տեխնոլոգիական այնպիսի հսկաների կողմից, ինչպիսիք են Google-ը, Youtube-ը, Facebook-ը, Instagram-ը, Wikipedia-ն, Netflix-ը, Quora-ն, Reddit-ը, Dropbox-ը, IBM-ը, Amazon-ը։

Python-ը նախագծերի ավտոմատացման և սկրիպտավորման համար։ Python-ը հիանալի գործիք է կրկնվող առաջադրանքները ավտոմատացնող ծրագրեր գրելու համար (սկրիպտավորում), մասնավորապես, ֆայլերի և պանակների հետ աշխատելու համար, օրինակ՝ ստեղծել, վերանվանել, տրոհել, միավորել կամ ջնջել ֆայլեր, թարմացնել ֆայլերի բովանդակությունը։ Ավտոմատացումը կարելի է օգտագործել ինտերնետից տեղեկատվություն որոնելու և ներբեռնելու, առցանց ձևաթղթեր լրացնելու և ներկայացնելու, կանոնավոր ծանուցումներ կամ էլեկտրոնային փոստեր ուղարկելու համար։

Python-ը նաև կարևոր դեր է խաղում DevOps-ում և Infrastructure as Code (IaC)-ում՝ օգտագործելով Ansible և Terraform-ի նման գրադարանները։

ԽՆԴՐԱԳԻՐՔ

Գլուխ 4. Python օպերատորները: Արտահայտություններ

1. Արտածել ստորև բերված ուղղանկյունը:

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

2. Արտածել ստորև բերված պատկերը:

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

3. Արտածել ստորև բերված եռանկյունը:

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

4. Արտածել ստորև բերված եռանկյունը:

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

5. Արտածել ստորև բերված հավասարասրուն եռանկյունը:

```
  *
 ***
```

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

6. Արտածել ստորև բերված պատկերը։

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

7. Արտածել ստորև բերված շեղանկյունը։

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

8. Հաշվել և արտածել հետևյալ արտահայտության արժեքը.

$$\frac{624.5 - (32.7 + 24.3) * 9}{15.8 + 9.3 * 5 + 7}$$

9. Գրել ծրագիր, որը հաշվում և տպում է մուտքագրված իրական թվի քառակուսին։

10. Ներածել որևէ x թիվ։ Ապա, օգտագործելով sep օպցիոնալ պարամետրը՝ տպել x, 2x, 3x, 4x և 5x արժեքները՝ դրանք միմյանցից անջատելով երեք գծիկով։

11. Ներածել երեք թիվ, հաշվել ու արտածել այդ թվերի գումարն ու միջին թվաբանականը։

12. Հաշվել և արտածել տրված a կողմով քառակուսու պարագիծն ու մակերեսը։

13. Մուտքագրել որևէ ժամանակահատված վայրկյաններով և արտածել այն՝ արտահայտված րոպեներով ու վայրկյաններով։

14. Հաշվել և արտածել n տարով r տոկոսով բանկում ներդրված a մայր գումարի վերջնական արժեքը (n, r, a, փոփոխականների արժեքները մուտքագրել):

15. Գրել ծրագիր, որը փոխում և տպում է x, y և z երեք փոփոխականների արժեքներն այնպես, որ x-ը ստանա y-ի արժեքը, y-ը՝ z-ի, իսկ z-ը՝ x-ի:

16. Գրել ծրագիր, որն օգտվողին առաջարկում է մուտքագրել $y = x^n$ տեսքի որևէ ֆունկցիա և տպում է դրա ածանցյալը՝ $y' = n \cdot x^{n-1}$:

17. Հաշվել հետևյալ թվաբանական արտահայտությունների արժեքը:

ա) x * y / z + x / (y * z), եթե x = 10, y = 4, z = 5

բ) (a * * 3 + b * *3) / (a * c), եթե a = 2, b = 3, c = 4

գ) (-b+math.sqrt(b**2-4*a*c))/(2*a), եթե a = 2, b = 3, c = 1

դ) x/(1+math.pow(x,2)/((2*x)**3)), եթե x=5

ե) math.cos(2*a-b)+math.sin(2*a-b), եթե a=5, b=10

զ) math.pi+math.e

է) 0.49*math.exp(2*a-b)+math.log10(100*b), եթե a=5, b=10

ը) math.sqrt(t+k)**2+1//math.floor(-k+1.4), եթե k=15, t=1

թ) (d - x) ** 5 // (math.sqrt(c + d - 9) % 2 * 2 + 1.2), եթե c = 24, d = 10, x = 8

ժ) math.fabs (-p - q)//17 * math.ceil(-p //y + 0.5), եթե p=21, q=20, y=7

ի) (math.sqrt(t + p) ** 2 + 1) // math.ceil(-p + 9.4), եթե p = 6, t = 24

լ) math.sqrt(p - q) ** 2 + 65 //math.floor(-p - 1.4), եթե p=26, q = 1

խ) math.sqrt ((m + n) ** 2) // ((m + n) % 2 + x), եթե m = 35, n = 30, x = 17

ծ) math.tan((math.ceil(- b) + math.floor (- b)) - (math.ceil(f) + math.floor(f) -2)), եթե f = - 1.7, b = 2.6

կ) (math.ceil(- r) + math.floor(- r)) // (math.ceil(- a) + math.floor(- a)), եթե a = 1.4, r = 2,6

հ) (d + x) ** 2 /(math.sqrt(c + d) * 2 % 2 + 1.8), եթե c = 149, d = 20, x = 1

ձ) (d - x) ** 2 / math.floor((math.sqrt(c + d - 100) * 2 % 2 + 1.2)), եթե c = 154, d = 10, x = 17

ղ) (math.sqrt(t + k) ** 2 + 1) // math.fabs(math.ceil(-k + 1.4) + math.floor(-k + 1.4)), եթե k=63, t = 1

ճ) math.sqrt(math.fabs(((math.ceil(- b) // (- b)) * (math.ceil(f) + math.floor(f) -1)))), եթե f = - 7.7, b = 12,2

մ) math.sqrt(t + k) ** 2 + 1 / math.ceil(-k + 1.4) * math.sin(k % 2-t), եթե k=15, t = 1

18. Հաշվել հետևյալ տրամաբանական արտահայտությունների արժեքը.

ա) (a or b and c) and not ((p or b) or (q and b)), եթե q = False, p = True

բ) not (((-p) < (-p)) and f) or d, եթե d = False

գ) (b and a) or not (a and b) or b, եթե b = False

դ) not(-a>-a or f)or f, եթե f = True

ե) not (f and not f) or q, եթե q = False

զ) x and not x

է) x or not x

ը) not(x and not x or x or not x)

թ) (p and q and r) and not (f and not f) or q, եթե q = False, p=q=r=False

ժ) (a or b and c or d) and not ((True or x) or (False and y))

ի) not (((-p) == (-p)) or f) and d

լ) (math.ceil(- b) <= (- b)) or (b + f)

խ) not(-p!=-p and f) or d

ծ) not (((-p) == (-p)) or f) and d

կ) not(x >= c or x <=c) and not p

հ) not (b or m and n) and not ((q == p) or (q != p))

ձ) (x >= c or x <= c) or not y

ղ) not (c or not c) and (b or not c and f)

ճ) not (a or b and c) or(not(p !=p) or (p == p))

մ) (a or b and c) and not ((p == p) or (q != b))

Գլուխ 5. Python-ի հրամանները: if պայմանական հրամանը: pass հրամանը

Գրել ծրագիր, որը հաշվում և արտածում է տրված ֆունկցիայի արժեքը արգումենտների տրված արժեքների համար:

19. $y = \begin{cases} x^2 + tg^3(x), & \text{եթե } -1 \le x \le 1 \\ \cos(x), & \text{հակառակ դեպքում} \end{cases}$

20. $y = \begin{cases} sin^2(4x) + arctg^5(x), & \text{եթե } -5 \le x \le 5 \\ \ln(|x|), & \text{հակառակ դեպքում} \end{cases}$

21. $y = \begin{cases} 3e^{2x} + sin(x^2 - 1), & \text{եթե } -10 \le x \le 10 \\ \ln(|x|) - 4e^{\cos(x)}, & \text{հակառակ դեպքում} \end{cases}$

22. $y = \begin{cases} (x+1)^n, & \text{եթե } x > 0 \\ x^2, & \text{եթե } x < -10 \\ |x|, & \text{հակառակ դեպքում} \end{cases}$

23. $y = \begin{cases} 2x^4 + |tg(x)|, & \text{եթե } -2 \le x \le 1 \\ x^2 - \frac{tg^3(x)}{\sqrt{x}}, & \text{եթե } 1 < x \le 10 \\ \sin(x), & \text{հակառակ դեպքում} \end{cases}$

24. $y = \begin{cases} 2x^3 + tg(x), & \text{եթե } -2 \le x \le 1 \\ x^2 - \sqrt{x}, & \text{եթե } 1 < x \le 10 \\ \sin(2x), & \text{հակառակ դեպքում} \end{cases}$

25. $y = \begin{cases} (1+x^2)^4, & \text{եթե } -2 \le x \le 3 \\ cos^2 x + (4 + x^3), & \text{եթե } x > 3 \\ x, & \text{հակառակ դեպքում} \end{cases}$

26. $y = \begin{cases} \sqrt[3]{x + |x|}, & \text{եթե } 1 \le x \le 7 \\ e^{\cos(2x)}, & \text{եթե } x > 7 \\ \ln(4 + x^2), & \text{հակառակ դեպքում} \end{cases}$

27. $y = \begin{cases} a^2 + b^2, & \text{եթե } a + b < 3 \\ a^2 - b^2, & \text{եթե } a + b > 10 \\ -a^3, & \text{հակառակ դեպքում} \end{cases}$

28. $y = \begin{cases} 3sin^2|a + b|, & \text{եթե } a^2 + b^2 < 10 \\ 2a^2 + 3b, & \text{եթե } a^2 + b^2 > 10 \\ 10, & \text{հակառակ դեպքում} \end{cases}$

29. $F(x) = \begin{cases} x^2 + 1, x \leq 0 \\ x + n!, x > 0 \end{cases}$

30. Ներածել որևէ եռանիշ թիվ: Արտածել.

ա) այդ թվի թվանշանները, դրանց գումարը և միջին թվաբանականը:

բ) այդ թվի թվանշաններից ամենամեծն ու ամենափոքրը:

գ) արտածել True, եթե այդ թվի միավորների կարգում գրված թվանշանը հավասար է տասնավորների և հարյուրավորների թվանշանների գումարին, և False՝ հակառակ դեպքում:

դ) արտածել True, եթե այդ թվի մեջ կան իրար հավասար թվանշաններ և False՝ հակառակ դեպքում:

ե) արտածել նոր եռանիշ թիվ՝ այդ թվի թվանշանները դասավորված աճման (նվազման) կարգով:

31. Ապրանքի մատակարարը պատվիրատուին տրամադրում է գների զեղչային համակարգ մեծաքանակ գնումների դեպքում: Ապրանքի միավորի գինը 32 $ է: Այն դեպքում, երբ պատվիրված ապրանքի քանակը 10-ից 99 է, տրամադրվում է 5% զեղչ, 100-ից 299 խմբաքանակի դեպքում՝ 10%, իսկ 300-ից ավելի քանակի դեպքում՝ 15%: Գրել ծրագիր, որը հաշվում է պատվիրատուին տրամադրված զեղչի մեծությունը և գնումների գումարային արժեքը՝ կախված ապրանքի քանակից:

32. Գրել ծրագիր, որը հաշվարկում է անշարժ գույքի վաճառքով զբաղվող գործակալին տրվող պարգևավճարի մեծությունը՝ կախված նրա վաճառքի ցուցանիշից հետևյալ կանոնով՝ պարգևավճարը կազմում է վաճառքի ցուցանիշի 5%-ը՝ եթե այն մեծ է

2 մլն դրամից և չի գերազանցում 4 մլն դրամը, 10%-ը՝ եթե մեծ է 4 մլն դրամից և չի գերազանցում 6 մլն դրամը, և 15%-ը՝ եթե 6 մլն դրամից ավելի է։

Python-ի for և while հրամանները: range() ֆունկցիան, break և continue հրամանները

33. Արտածել Python բառը 100 անգամ:

34. Արտածել 1-ից 20 ամբողջ թվերը և դրանց քառակուսիները:

35. Օգտագործելով for հրամանը՝ տպել 8, 11, 14, 17, 20, . . , 83, 86, 89 թվերը։

36. Օգտագործելով for հրամանը՝ տպել 100, 98, 96, ..., 4, 2 թվերը:

37. Օգտագործելով for հրամանը՝ արտածել ստորև բերված տառերի հաջորդականությունը.

AAAAAAAAAABBBBBBBCDCDCDCDAAAAAAAAAAEFFFFFFG

38. Ֆիբոնաչիի թվերը կազմում են հաջորդականություն, որտեղ առաջին երկու թվերը հավասար են մեկի, իսկ դրանից հետո յուրաքանչյուր թիվ հավասար է երկու նախորդ թվերի գումարին: Տրված n բնական թվի համար արտածել Ֆիբոնաչիի հաջորդականության n-րդ անդամը:

39. Մուտքագրել ուղղանկյան a լայնությունը և b երկարությունը, ապա կառուցել ու տպել այդ ուղղանկյունը: Ուղղանկյունը պետք է ունենա խնդիր 1-ում նշված տեսքը:

40. Կառուցել խնդիր 3-ում բերված պատկերը՝ օգտագործելով for հրամանը: Եռանկյան h բարձրությունը մուտքագրել ստեղնաշարից:

41. Կառուցել խնդիր 4-ում բերված պատկերը՝ օգտագործելով for հրամանը: Եռանկյան h բարձրությունը մուտքագրել ստեղնաշարից:

42. Կառուցել խնդիր 5-ում պատկերված հավասարասրուն եռանկյունը՝ օգտագործելով for հրամանը: Եռանկյան h բարձրությունը մուտքագրել ստեղնաշարից:

43. Գեներացնել և տպել [3, 6] միջակայքից 50 պատահական ամբողջ թիվ:

44. Գրել ծրագիր, որը գեներացնում է [1, 50] միջակայքին պատկանող որևէ պատահական թիվ՝ x, ապա [2, 5] միջակայքին պատկանող որևէ պատահական թիվ՝ y և հաշվում ու արտածում է x^y:

45. Գեներացնել [1, 10] միջակայքին պատկանող որևէ պատահական ամբողջ թիվ և տպել Hello բառը այդքան անգամ:

46. Գրել ծրագիր, որը գեներացնում է [1, 10] միջակայքին պատկանող որևէ պատահական ամբողջ թիվ և տպում է ձեր անունը այդքան անգամ:

47. Գեներացնել և տպել [1, 10] միջակայքին պատկանող որևէ պատահական իրական թիվ՝ ստորակետից հետո երկու նիշ ճշտությամբ:

48. Գրել ծրագիր, որը գեներացնում է 50 պատահական ամբողջ թիվ. առաջինը՝ [1, 2] միջակայքից, երկրորդը՝ [1, 3], երրորդը՝ [1, 4] և այլն, հիսուներորդը՝ [1, 51] միջակայքից:

49. Մուտքագրել որևէ բնական թիվ և արտածել դրա ա) վերջին և նախավերջին թվանշանները, բ) վերջին երկու թվանշաններից կազմված թիվը:

50. Հաշվել մուտքագրված n բնական թվի ֆակտորիալը՝ $n! = 1 \times 2 \times \ldots \times n$:

51. Գրել ծրագիր, որը գեներացնում է [1, 10] միջակայքին պատկանող որևէ պատահական ամբողջ թիվ: Օգտվողը փորձում է կռահել, թե ի՞նչ թիվ է գեներացված: Ծրագիրը պետք է տա հաղորդագրություն այն մասին, թե արդյոք օգտվողը ճի՞շտ է կռահել թիվը, թե՞ ոչ:

52. Արտածել մուտքագրված բնական թվի բոլոր բաժանարարները:

53. Հաշվել ու արտածել մուտքագրված բնական թվի բաժանարարների գումարը:

54. Հաշվել մուտքագրված տասը թվերից 50-ից մեծ թվերի քանակը:

55. Հաշվել առաջին 100 բնական թվերից այն թվերի քանակը, որոնց քառակուսիները վերջանում են 4-ով:

56. Գրել ծրագիր, որը գեներացնում է [1, 100] միջակայքից 1000 պատահական ամբողջ թիվ և հաշվում դրանցից 12-ին բազմապատիկ թվերի քանակը:

57. Հաշվել [1, 100] միջակայքի այն թվերի քանակը, որոնց քառակուսիներն ավարտվում են 4-ով, 9-ով:

58. Հաշվել հետևյալ գումարը՝ S=1 – 2 + 3 – 4 + ··· + 1999 – 2000:

59. Թիվը կոչվում է կատարյալ, եթե այն հավասար է իր բոլոր բաժանարարների գումարին՝ բացի իրենից: Գրել ծրագիր, որը ստուգում է, մուտքագրված թիվը կատարյա՞լ է, թե՞ ոչ:

60. Մուտքագրել տասը թիվ:

ա) տպել ամենամեծ և ամենափոքր թվերը,

բ) տպել այդ թվերի գումարը և միջին թվաբանականը,

գ) տպել երկրորդ ամենամեծ թիվը,

դ) եթե թվերի մեջ կա 100-ից մեծ թիվ, ապա այդ մասին տպել հաղորդագրություն:

61. Խաղացողին առաջարկվում է կռահել [1, 10] միջակայքից գեներացված ամբողջ պատահական թիվը: Խաղացողը խաղում է հինգ անգամ: Յուրաքանչյուր ճիշտ պատասխանի համար նա ստանում է 10 միավոր, իսկ սխալ պատասխանի դեպքում՝ հանվում է 1 միավոր: Գրել ծրագիր, որը խաղի ավարտից հետո հայտնում է, թե խաղացողը քանի միավոր է հավաքել:

62. Գրել ծրագիր, որը մուտքագրված բնական n թվի և իրական x թվի համար հաշվում և արտածում է $y = x^n$:

Գրել ծրագիր, որը տրված $x \in R$ և $n \in N$ արժեքների համար հաշվում և արտածում է S գումարը:

63. $S = \sum_{i=1}^{n} \frac{x^i}{i!}, x \in R, n \in N$

64. $S = \sum_{i=1}^{n} \frac{(n+i)!}{x^i}, x \in R, n \in N$

65. $S = \sum_{i=1}^{n} (-1)^{i+1} \frac{x^{2i}}{(2i+2)!}, x \in R, n \in N$

66. $S = \sum_{i=1}^{n} (-1)^{i} \frac{x^{2i+1}}{(2i+1)!}, x \in R, n \in N$

67. $S = \sum_{k=1}^{n} \frac{x^{n-k}}{(2k+1)(2k+2)}, x \in R$

68. $S=\prod_{i=0}^{n}\frac{x^i}{(i+1)!}(i+2),\ n\in N, x\in R$

69. $S=\sum_{k=1}^{100}\frac{k^k+1}{k!}$

70. $S=\sum_{k=1}^{n}\frac{k}{k+1}$, $n\in N$

71. $S=\sum_{k=1}^{n}k^k\cdot k!$, $n\in N$

72. $S=\prod_{k=1}^{n}\frac{k^k\cdot k!}{k+1}$, $n\in N$

73. $S=\sum_{k=1}^{n}\frac{1}{k(k+1)(k+2)\ldots(k+n)}$, $n\in N$

74. $S=\sum_{k=1}^{n}\frac{x^k k!}{(x+1)(2x+1)\ldots(kx+1)}$, $n\in N$, $x\in R$

75. $P=(2k)!!=2\cdot 4\cdot 6\cdot\ldots\cdot 2k, k\in N$

76. $P=(2k+1)!!=1\cdot 3\cdot 5\cdot\ldots\cdot(2k+1), k\in N$

Գլուխ 6. Տողեր

77. Հաջորդաբար մուտքագրել այբուբենի տառերը և դրանք կցել իրար:

78. Մուտքագրել որևէ տող և այդ տողի 5-րդ ինդեքսով տարրը փոխարինել 'y' տառով:

79. Ստուգել, մուտքագրված տողը սկսվո՞ւմ է տառով, թե՞ ոչ և տալ այդ մասին համապատասխան հաղորդագրություն:

80. Ստուգել, արդյո՞ք տրված տողում բոլոր սիմվոլները տառեր են և տալ այդ մասին համապատասխան հաղորդագրություն:

81. Արտածել մուտքագրված տողում 'a' տառի հանդիպման բոլոր ինդեքսները: Եթե 'a' տառը չի հանդիպում տողում, ապա տալ այդ մասին հաղորդագրություն:

82. Հաշվել և տպել մուտքագրված տողում առկա 'a' սիմվոլների քանակը։

83. Հաշվել և տպել մուտքագրված տողում 'a' սիմվոլին հաջորդող բոլոր 'o' սիմվոլների քանակը։ Եթե դրանք չեն հանդիպում, ապա տալ այդ մասին հաղորդագրություն։

84. Տրված է տող, որը պարունակում է երկու 'z' սիմվոլ։ Հաշվել և տպել դրանց միջև գտնվող սիմվոլների քանակը։

85. Տրված է տող, որը պարունակում է առնվազն մեկ 'a' սիմվոլ։ Ստանալ նոր տող, որի տարրերը ստացվում են տրված տողից արտաքսելով բոլոր 'a' սիմվոլները։

86. Գրել ծրագիր, որը եռապատկում է մուտքագրված տողի բոլոր սիմվոլները։

87. Գրել ծրագիր, որը տպում է մուտքագրված տողի նախ առաջին սիմվոլը, հետո՝ առաջին և երկրորդ սիմվոլները, ապա առաջին, երկրորդ ու երրորդ սիմվոլները և այսպես շարունակ։

88. Տրված է տող, որը պարունակում է առնվազն մեկ 'x' սիմվոլ։ Ստանալ նոր տող, որի տարրերը ստացվում են տրված տողի մեջ յուրաքանչյուր 'x' սիմվոլ փոխարինելով երկու հատ 'y' սիմվոլով։

89. Մուտքագրել որևէ տող։ Արտածել նոր տող, որի տարրերը ստացվում են տրված տողի մեջ տեղերով փոխելով առաջին և վերջին տարրերը, երկրորդ ու նախավերջին տարրերը և այլն։

90. Տրված է տող, որը պարունակում է առնվազն մեկ 'x' սիմվոլ։ Ստանալ նոր տող, որը ստացվում է տրված տողից հեռացնելով առաջին 'x' սիմվոլին հաջորդող բոլոր սիմվոլները։

91. Տրված է տող, որը պարունակում է որևէ ազգանուն։ Արտածել True, եթե տողը վերջանում է 'yan'-ով և False՝ հակառակ դեպքում։

92. Տրված է տող, որը պարունակում է առնվազն մեկ 'x' սիմվոլ։ Հաշվել տողում գտնվող 'x' սիմվոլների ինդեքսների միջին թվաբանականը։

93. Գրել ծրագիր, որը մուտքագրված տողը գրում է փոքրատառերով, ապա հեռացնում է բոլոր ստորակետերն ու վերջակետերը։

94. Գրել ծրագիր, որը մուտքագրված տողի համար՝

ա) տպում է տողի տարրերի թիվը,

բ) տողը կրկնում է տասը անգամ,
գ) տպում է տողի առաջին տարրը,
դ) տպում է տողի առաջին երեք տարրերը,
ե) տպում է տողի վերջին երեք տարրերը,
զ) տպում է տողը հակառակ հերթականությամբ,
է) տպում է տողի յոթերորդ տարրը: Եթե տողում այդպիսի տարր չկա, տալ այդ մասին հաղորդագրություն,
ը) տպում է տողը՝ առանց իր առաջին և վերջին տարրերի,
թ) յուրաքանչյուր 'a' տառ փոխարինել 'e' տառով,
ժ) յուրաքանչյուր ստորակետ փոխարինել բացակով:

95. Տողում բառերի քանակը հաշվելու պարզ միջոց է տողի բացակների քանակը հաշվելը (եթե բառերն իրարից անջատված են մեկ բացակով): Գրել ծրագիր, որը հաշվում և արտածում է մուտքագրված տողում բառերի քանակը, եթե հայտնի է, որ բառերն իրարից անջատված են մեկ բացակով:

96. Գրել ծրագիր, որը ստուգում է, թե արդյոք մուտքագրված բանաձևն ունի՞ նույն թվով բացող և փակող փակագծեր:

97. Հաշվել ու տպել մուտքագրված տողում ձայնավորների քանակը:

98. Տրված է որևէ տող: Արտածել True, եթե այդ տողը աջից և ձախից կարդացվում է նույն ձևով և False՝ հակառակ դեպքում (այդպիսի բառերը կոչվում են պոլինդրոմներ):

99. Գրել ծրագիր, որն առաջարկում է օգտվողին մուտքագրել 'a' տառը պարունակող որևէ բառ: Այնուհետև ծրագիրը տպում է հետևյալ երկու տողերը. առաջին տողում պետք է լինի տողի մինչև առաջին 'a' տառը ներառող մասը, իսկ երկրորդ տողում՝ տողի մնացած մասը:

100. Գրել ծրագիր, որն առաջարկում է օգտվողին մուտքագրել որևէ բառ, այնուհետև մեծատառով գրել այդ բառի յուրաքանչյուր կենտ կարգահամարով տառը:

101. Գրել ծրագիր, որն առաջարկում է օգտվողին մուտքագրել երկու տող: Եթե տողերն ունեն նույն երկարությունը, ապա ծրագիրը պետք է միացնի դրանք միմյանց և տպի, հակառակ դեպքում՝ տպի համապատասխան հաղորդագրություն և ավարտի իր աշխատանքը:

102. Գրել ծրագիր, որն առաջարկում է օգտվողին մուտքագրել իր անունը, ազգանունը, հայրանունը փոքրատառերով, այնուհետև տպել դրանք՝ յուրաքանչյուր բառի առաջին տառը փոխելով մեծատառի (ենթադրվում է, որ անունը, ազգանունը, հայրանունը իրարից բաժանված են մեկական բացակով):

103. Չօգտագործելով in օպերատորը՝ գրել ծրագիր, որն առաջարկում է օգտվողին մուտքագրել որևէ տող և որևէ տառ, այնուհետև ստուգել, թե այդ տառը հանդիպո՞ւմ է տողում, թե՞ ոչ:

104. Չօգտագործելով count մեթոդը, գրել ծրագիր, որն առաջարկում է օգտվողին մուտքագրել որևէ տող և որևէ տառ, այնուհետև հաշվել, թե այդ տառը քանի՞ անգամ է հանդիպում տողում:

105. Չօգտագործելով index մեթոդը, գրել ծրագիր, որն առաջարկում է օգտվողին մուտքագրել որևէ տող և որևէ տառ, այնուհետև տպել մուտքագրված տողի մեջ այդ տառի առաջին հանդիպման ինդեքսը: Եթե տվյալ տառը չի հանդիպում տողի մեջ, ապա այդ մասին պետք է տրվի հաղորդագրություն:

Գլուխ 7. Ցուցակներ: Ցուցակների ներկառուցված ֆունկցիաներն ու մեթոդները

106. Մուտքագրել ամբողջ թվերի որևէ ցուցակ և կատարել հետևյալ գործողությունները.

ա) Տպել ցուցակի տարրերի քանակը:

բ) Տպել ցուցակի առաջին և վերջին տարրերը:

գ) Տպել ցուցակը հակառակ հերթականությամբ:

դ) Տպել True, եթե ցուցակը պարունակում է հինգ թիվը և False՝ հակառակ դեպքում:

ե) Տպել ցուցակում հինգերի քանակը:

զ) Ցուցակից հեռացնել առաջին և վերջին անդամները, դասավորել մնացած անդամները աճման կարգով և տպել ստացված նոր ցուցակը:

է) Ավելացնել ցուցակի սկզբում նոր տարր և տպել ստացված ցուցակը:

ը) Ավելացնել ցուցակի վերջում նոր տարր և տպել ստացված ցուցակը:

թ) Ավելացնել ցուցակի երկրորդ տեղում նոր տարր և տպել ցուցակը։

ժ) Ավելացնել ցուցակի նախավերջին տեղում նոր տարր և տպել ցուցակը։

ի) Հաշվել ցուցակի հինգից փոքր թվերի գումարը, քանակը և միջին թվաբանականը։

լ) Հաշվել ցուցակում հինգից մեծ թվերի արտադրյալը։

107. Գրել ծրագիր, որը ստեղծում է [1,100] միջակայքին պատկանող 20 պատահական թվերից կազմված ցուցակ, ապա կատարել հետևյալ գործողությունները.

ա) Տպել ցուցակը։

բ) Տպել ցուցակի տարրերի միջին թվաբանականը։

գ) Տպել ցուցակի առաջին (վերջին) ամենամեծ տարրը և դրա կարգահամարը։

դ) Տպել ցուցակի առաջին (վերջին) ամենափոքր տարրը և դրա կարգահամարը։

ե) Տպել ցուցակի ամենամեծ ու ամենափոքր տարրերը։

զ) Տպել ցուցակի երկրորդ ամենամեծ ու երկրորդ ամենափոքր տարրերը։

է) Տպել ցուցակի վերջին ամենամեծ և առաջին ամենափոքր տարրերի միջև եղած տարրերը։

ը) Հաշվել ցուցակի զույգ թվերի քանակը և միջին թվաբանականը։ Եթե ցուցակում զույգ թիվ չկա, տալ այդ մասին հաղորդագրություն։

108. Տրված է [8,9,10] ցուցակը։ Կատարել հետևյալ գործողությունները և տպել յուրաքանչյուր քայլում ստացված նոր ցուցակը.

ա) Ներածել 17 թիվը՝ երկու ինդեքսի ներքո։

բ) Ստացված ցուցակի վերջում ավելացնել 4, 5 և 6 թվերը։

գ) Ստացված ցուցակից հեռացնել առաջին տարրը։

դ) Տեսակավորել ստացված ցուցակը աճման (նվազման) կարգով։

ե) Կրկնապատկել ստացված ցուցակը։

զ) Երեք ինդեքսով տարրի արժեքը դարձնել 25։

109. Գեներացնել ցուցակ, որի տարրերը [1,15] հատվածին պատկանող պատահական ամբողջ թվեր են: Ցուցակի տասից մեծ տարրերի արժեքները դարձնել 100:

110. Գեներացնել ցուցակ, որի տարրերը տողեր են: Ստեղծել նոր ցուցակ, որը բաղկացած է այդ տողերից՝ հեռացնելով դրանց առաջին սիմվոլները:

111. Տրված են նույն երկարությունն ունեցող L և M թվային ցուցակները: Կազմել նոր ցուցակ՝ N, որի i-րդ տարրը L և M ցուցակների համապատասխան տարրերից մեծն է:

112. Տրված են նույն երկարությունն ունեցող L և M թվային ցուցակները: Կազմել նոր ցուցակ՝ N, որի i-րդ տարրը L և M ցուցակների համապատասխան տարրերի գումարն է:

113. Տրված է L թվային ցուցակը:

ա) Հաշվել ու տպել զույգ (կենտ) տարրերի գումարը և միջին թվաբանականը:

բ) Հաշվել ու տպել զույգ (կենտ) կարգահամարներով տարրերի գումարը:

գ) Հաշվել ու տպել զույգ (կենտ) կարգահամարներով տարրերի քառակուսիների արտադրյալը:

դ) Հաշվել ու տպել տրված k թվին բազմապատիկ ինդեքսներով տարրերի քանակը:

ե) Հաշվել ու տպել դրական, բացասական և զրոյին հավասար տարրերի քանակը:

զ) Հաշվել դրական (բացասական) տարրերի գումարը, քանակը, միջին թվաբանականը:

է) Հաշվել ու տպել այն տարրերի քանակը, որոնք հավասար են տրված a թվին: Եթե այդ պայմանին բավարարող թվեր չկան, տալ այդ մասին հաղորդագրություն:

ը) Հաշվել ու տպել այն զույգ (կենտ) տարրերի քանակը, որոնք փոքր են կամ հավասար տրված a թվից: Եթե այդ պայմանին բավարարող թվեր չկան, այդ մասին տալ հաղորդագրություն:

թ) Հաշվել ու տպել այն տարրերի միջին թվաբանականը, որոնք պատկանում են տրված [a, b] հատվածին: Եթե այդ պայմանին բավարարող թվեր չկան, տալ հաղորդագրություն:

ժ) Հաշվել ու տպել այն տարրերի արտադրյալը, որոնք բազմապատիկ են հինգին և չեն պատկանում տրված [c, d] հատվածին: Եթե այդ պայմանին բավարարող թվեր չկան, այդ մասին տալ հաղորդագրություն:

114. Տրված է L ցուցակը, որի տարրերը իրական թվեր են:

ա) Գտնել ցուցակի մեծագույն ու փոքրագույն տարրերը և դրանց կարգահամարները:

բ) Գտնել ցուցակի մեծագույն ու փոքրագույն տարրերը և դրանց կրկնությունների քանակը:

գ) Գտնել ցուցակի մեծագույն (փոքրագույն) տարրը: Եթե կան մեծագույն (փոքրագույն) տարրի կրկնություններ, ապա հեռացնել դրանք և տպել ստացված ցուցակը:

դ) Գրել ծրագիր, որը հաշվում և արտածում է, թե քանի դրական տարր կա իր առավելագույն և նվազագույն տարրերի միջև: Եթե այդպիսի տարր չկա, տալ այդ մասին հաղորդագրություն:

ե) Դիցուք L ցուցակի մեջ չկան զրոներ: Կառուցել նոր ցուցակ՝ M, սկզբում տեղադրելով տրված ցուցակի դրական, ապա՝ բացասական տարրերը:

115. Տրված է L ցուցակը, որի տարրերը բնական թվեր են:

ա) Գտնել հինգին բազմապատիկ թվերի քանակը: Եթե ցուցակում հինգին բազմապատիկ թվեր չկան, տալ այդ մասին հաղորդագրություն:

բ) Գտնել այն թվերի քանակը, որոնք հինգի վրա բաժանելիս տալիս են մեկ մնացորդ: Եթե ցուցակում այդ պայմանին բավարարող թվեր չկան, տալ այդ մասին հաղորդագրություն:

116. Տրված է L ցուցակը, որի տարրերը բնական թվեր են: Կառուցել նոր ցուցակ՝ M, որը պարունակում է այդ ցուցակի պարզ (բաղադրյալ) թվերը:

117. Տրված է L ցուցակը, որի տարրերը բնական թվեր են: Կառուցել նոր ցուցակ՝ M, որը կազմված է L ցուցակի այն տարրերից, որոնք իրենց կարգահամարի վրա բաժանվում են առանց մնացորդի: Ցուցակի առաջին տարրն (զրո կարգահամարով) ընդգրկել առանց ստուգման:

118. Գեներացնել [1,50] միջակայքին պատկանող 100 պատահական թվերից բաղկացած ցուցակ:

ա) Գտնել և արտածել այդ ցուցակում այն տարրը, որն ամենամոտն է ցուցակի բոլոր տարրերի միջին թվաբանականին:

բ) Արտածել այդ ցուցակի պարզ թվերի գումարը:

գ) Որոշել և արտածել զույգ տարրերի և 5-ով ավարտվող տարրերի քանակը:

119. Տրված է ցուցակ, որը պարունակում է գործազուրկների թվաքանակներն ըստ Երևան քաղաքի և ՀՀ մարզերի: Արտածել ՀՀ գործազուրկների ընդհանուր թվաքանակն ու միջինը:

120. Տրված է ցուցակ, որը պարունակում է զբաղվածների թվաքանակներն ըստ Երևան քաղաքի և ՀՀ մարզերի: Արտածել ՀՀ զբաղվածների ընդհանուր թվաքանակն ու միջինը:

Երկչափ ցուցակներ: Ներդրված ցիկլեր

121. Տրված է իրական տարրերով մատրից՝ $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$: Հաշվել.

ա) $S = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}$

բ) $S = \sum_{i=1}^{n} \prod_{j=1}^{m} a_{ij}$

գ) $S = \prod_{i=1}^{n} \sum_{j=1}^{m} a_{ij}$

122. Տրված է իրական տարրերով մատրից՝ $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$:

ա) Հաշվել և տպել մատրիցի դրական տարրերի գումարը: Եթե մատրիցում դրական տարր չկա, արտածել համապատասխան հաղորդագրություն:

բ) Հաշվել ու տպել մատրիցի բացասական տարրերի արտադրյալը: Եթե մատրիցում բացասական տարր չկա, արտածել համապատասխան հաղորդագրություն:

գ) Հաշվել ու տպել մատրիցի այն տարրերի արտադրյալը և քանակը, որոնք պատկանում են տրված $[a, b]$ միջակայքին: Եթե մատրիցում այդպիսի տարր չկա, արտածել համապատասխան հաղորդագրություն:

դ) Հաշվել ու տպել մատրիցի դրական, բացասական և զրոյին հավասար տարրերի քանակները:

ե) Գտնել ու տպել մատրիցի փոքրագույն (մեծագույն) տարրը և դրա կարգահամարները:

զ) Գտնել ու տպել մատրիցի տարրերի բացարձակ արժեքների փոքրագույն և մեծագույն արժեքների միջին թվաբանականը:

123. Տրված է $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$ մատրիցը, որի տարրերը բնական թվեր են:

ա) Հաշվել ու տպել մատրիցի զույգ տարրերի միջին թվաբանականը: Եթե մատրիցում այդպիսի տարր չկա, արտածել համապատասխան հաղորդագրություն:

բ) Հաշվել և տպել մատրիցի կենտ տարրերի միջին երկրաչափականը: Եթե մատրիցում այդպիսի տարր չկա, արտածել համապատասխան հաղորդագրություն:

գ) Հաշվել ու տպել մատրիցի այն տարրերի արտադրյալը և քանակը, որոնք առանց մնացորդի բաժանվում են տրված N բնական թվին: Եթե մատրիցում այդպիսի տարր չկա, արտածել համապատասխան հաղորդագրություն:

դ) Հաշվել ու տպել մատրիցի այն տարրերի արտադրյալը և քանակը, որոնք երեքի վրա բաժանվելիս տալիս են մեկ մնացորդ: Եթե մատրիցում այդպիսի տարր չկա, արտածել համապատասխան հաղորդագրություն:

ե) Հաշվել ու տպել մատրիցի այն տարրերի արտադրյալը և քանակը, որոնք պատկանում են տրված $[a, b]$ միջակայքին և առանց մնացորդի բաժանվում են տրված N բնական թվին: Եթե մատրիցում այդպիսի տարր չկա, ապա արտածել համապատասխան հաղորդագրություն:

զ) Հաշվել ու տպել մատրիցի զույգ և կենտ տարրերի միջին թվաբանականները:

է) Հաշվել ու տպել մատրիցի պարզ և բաղադրյալ տարրերի միջին թվաբանականները:

124. Տրված է $A = (a_{ij}),\ i = 1, \dots, m, j = 1, \dots, m$ քառակուսային մատրիցը, որի տարրերը բնական թվեր են։

ա) Հաշվել ու արտածել մատրիցի գլխավոր անկյունագծի տարրերի քառակուսիների գումարը։

բ) Հաշվել ու տպել մատրիցի գլխավոր անկյունագծի կենտ տարրերի քառակուսիների արտադրյալը։

125. Կառուցել ու արտածել $A = (a_{ij}),\ i = 1, \dots, m, j = 1, \dots, m$ քառակուսային մատրիցը, որտեղ a_{ij}-երը տրված $[p, q]$ միջակայքին պատկանող պատահական թվեր են։

126. Կառուցել ու արտածել $A = (a_{ij}),\ i = 1, \dots, m, j = 1, \dots, m$ քառակուսային մատրիցը, որի գլխավոր անկյունագծի տարրերը հավասար են զրոյի, իսկ մնացած տարրերը որոշվում են հետևյալ բանաձևով՝ $a_{ij} = (i + 1)^2/(2j)$։

127. Կառուցել և արտածել $A = (a_{ij}),\ i = 1, \dots, m, j = 1, \dots, m$ քառակուսային մատրիցը, որտեղ $a_{ij} = i^2 + j^2$։

128. Գեներացնել [1,50] միջակայքին պատկանող պատահական ամբողջ թվերից կազմված մատրից $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$։

ա) Արտածել A մատրիցը։

բ) Արտածել A մատրիցի շրջված մատրիցը։

129. Գեներացնել [p,q] միջակայքին պատկանող պատահական բնական թվերից կազմված քառակուսային մատրից $A = (a_{ij}), i, j = 1, \dots, n$։

ա) Արտածել A մատրիցի տարրերը՝ փոխարինելով գլխավոր անկյունագծից վերև գտնվող տարրերը զրոներով։

բ) Արտածել A մատրիցի տարրերը՝ փոխարինելով գլխավոր անկյունագծից ներքև գտնվող տարրերը մեկերով։

գ) Արտածել A մատրիցի տարրերը՝ փոխարինելով գլխավոր անկյունագծից վերև գտնվող տարրերը (-1)-ով, գլխավոր անկյունագծից ներքև գտնվող տարրերը մեկերով, իսկ գլխավոր անկյունագծի տարրերը՝ զրոներով։

130. Գեներացնել [p,q] միջակայքին պատկանող պատահական ամբողջ թվերից կազմված քառակուսային մատրից՝ $A = (a_{ij}), i, j = 1, \dots, n$։

ա) Հաշվել ու արտածել A մատրիցի գլխավոր անկյունագծի վրա և դրանից վերև գտնվող տարրերի գումարը։

բ) Հաշվել ու արտածել A մատրիցի գլխավոր անկյունագծի վրա և դրանից ներքև գտնվող զույգ տարրերի միջին թվաբանականը։

գ) Հաշվել ու արտածել A մատրիցի գլխավոր անկյունագծից վերև գտնվող այն տարրերի միջին թվաբանականը, որոնք երեքի վրա բաժանվելիս տալիս են մեկ մնացորդ, և գլխավոր անկյունագծից ներքև գտնվող այն տարրերի միջին թվաբանականը, որոնք երեքի վրա բաժանվելիս տալիս են երկու մնացորդ։

131. Տրված է իրական տարրերով մատրից՝ $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$։

ա) Հաշվել ու արտածել $\max_i \min_j a_{ij}$։

բ) Հաշվել ու արտածել $\min_j \max_i a_{ij}$։

132. Տրված է իրական տարրերով մատրիցը $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$, ինչպես նաև $k \in [1, m]$ և $p \in [1, n]$ բնական թվերը։

ա) Դասավորել A մատրիցի k-րդ տողն աճման (նվազման) կարգով և արտածել ստացված մատրիցը։

բ) Դասավորել A մատրիցի p-րդ սյունն աճման (նվազման) կարգով և արտածել ստացված մատրիցը։

գ) Տեղերով փոխել A մատրիցի k-րդ տողն առաջին տողի հետ ($k \neq 1$) և արտածել ստացված մատրիցը։

դ) Տեղերով փոխել A մատրիցի p-րդ սյունն առաջին սյան հետ ($p \neq 1$) և արտածել ստացված մատրիցը։

133. Տրված են իրական տարրերով $A = (a_{ij})$ և $B = (b_{ij})$ ($i = 1, \dots, m, j = 1, \dots, n$) մատրիցները։

ա) Կառուցել ու արտածել $C = A + B$ և $D = A - B$ մատրիցները։

բ) Կառուցել ու արտածել $C = A * B$ մատրիցը։

գ) Կառուցել ու արտածել M և N մատրիցները, որտեղ $m_{ij} = a_{ij} * b_{ij}$, $n_{ij} = a_{ij}/b_{ij}$։ Ենթադրվում է, որ $b_{ij} \neq 0$։

134. Տրված է իրական տարրերով մատրից՝ $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$։

ա) Հաշվել ու արտածել A մատրիցի բոլոր տողերի մեծագույն տարրերի արտադրյալը։

բ) Հաշվել ու արտածել A մատրիցի բոլոր սյուների փոքրագույն տարրերի միջին թվաբանականը։

135. Տրված է իրական տարրերով մատրից $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$։

ա) Կառուցել ու արտածել $(b_1, b_2, \dots b_m)$ վեկտորը, որտեղ b_i-ն i -րդ տողի տարրերի գումարն է։

բ) Կառուցել ու արտածել $(b_1, b_2, \dots b_m)$ վեկտորը, որտեղ b_i-ն i-րդ տողի առավելագույն տարրն է։

գ) Կառուցել ու արտածել $(b_1, b_2, \dots b_m$) վեկտորը, որտեղ b_i-ն i -րդ տողի՝ տրված N բնական թվի վրա առանց մնացորդի բաժանվող տարրերի միջին թվաբանականն է։

136. Տրված է իրական տարրերով մատրից՝ $A = (a_{ij}), i = 1, \dots, m, j = 1, \dots, n$։

ա) Կառուցել և արտածել $(c_1, c_2, \dots c_n)$ վեկտորը, որտեղ c_j-ն j-րդ սյան տարրերի արտադրյալն է։

բ) Կառուցել ու արտածել $(c_1, c_2, \dots c_n)$ վեկտորը, որտեղ c_j-ն j-րդ սյան նվազագույն տարրն է։

գ) Կառուցել ու արտածել $(b_1, b_2, \dots b_m)$ վեկտորը, որտեղ b_i-ն i-րդ տողի՝ տրված $[p, q]$ միջակայքին պատկանող զույգ տարրերի միջին թվաբանականն է։

137. Տրված է ցուցակ, որը պարունակում է մեկ տարում բանկի հարյուր հաճախորդի կողմից ներդրված ավանդների մեծությունները ըստ ամիսների։ Որոշել յուրաքանչյուր ամսում ներդրված բոլոր ավանդների գումարը և միջին ավանդի մեծությունը։

138. Տրված է ցուցակ, որը պարունակում է հարյուր ձեռնարկության կողմից սպառված էլեկտրաէներգիայի մեծություններն ըստ ամիսների։ Հաշվել յուրաքանչյուր ամսում բոլոր ձեռնարկությունների կողմից սպառված էլեկտրաէներգիայի ընդհանուր մեծությունն ու միջինը։

139. Տրված է ցուցակ, որը պարունակում է ձեռնարկության հարյուր աշխատակցի աշխատավարձերն ըստ ամիսների։ Հաշվել յուրաքանչյուր աշխատակցի միջին աշխատավարձը մեկ տա-

րում, ինչպես նաև յուրաքանչյուր ամսում ձեռնարկության ընդհանուր աշխատավարձը։

140. Տրված է ցուցակ, որը պարունակում է ՀՀ արտահանման (ներմուծման) ծավալներն ըստ տասը երկրների մեկ տարվա բոլոր ամիսների համար։ Հաշվել և արտածել յուրաքանչյուր ամսվա համար արտահանման (ներմուծման) ընդհանուր և միջին ծավալը։

141. Տրված են հարյուր դիմորդի՝ մաթեմատիկայի, հայոց լեզվի և կենսաբանության քննություններից ստացած գնահատականները։ Հաշվել ու արտածել յուրաքանչյուր դիմորդի գումարային գնահատականը և միջին գնահատականը։

142. Տրված են տասը ձեռնարկության մեկ տարվա եկամուտներն ըստ ամիսների։ Հաշվել յուրաքանչյուր ձեռնարկության ընդհանուր եկամուտը, ամսեկան միջին եկամուտը։

Գլուխ 8. Հավաքածուներ

143. Գրել ծրագիր, որը ամբողջ թվերից բաղկացած հավաքածուի համար.

ա) Տպում է հավաքածուի տարրերի քանակը։

բ) Տպում է հավաքածուի առաջին ու վերջին տարրերը։

գ) Տպում է True, եթե հավաքածուն պարունակում է հինգ թիվը և False՝ հակառակ դեպքում։

դ) Հաշվել հավաքածուի մեջ պարունակվող հինգից փոքր թվերի գումարը, քանակն ու միջին թվաբանականը։

ե) Ջնջել ամբողջ հավաքածուն։

144. Գրել ծրագիր, որը մուտքագրված հավաքածուի համար.

ա) Տպում է հավաքածուի առաջին երեք տարրը։

բ) Տպում է հավաքածուի վերջին երեք տարրը։

գ) Տպում է հավաքածուի ութերորդ տարրը։ Եթե տրված հավաքածուի մեջ չկա ութերորդ տարր, արտածել այդ մասին հաղորդագրություն։

դ) Տպում է հավաքածուն՝ առանց իր առաջին և վերջին տարրերի։

145. Ներածել իրական թվերից կազմված որևէ հավաքածու, ապա կատարել հետևյալ գործողությունները.

ա) Տպել հավաքածուն։

բ) Տպել հավաքածուի տարրերի միջին թվաբանականը։

գ) Տպել հավաքածուի ամենամեծ և ամենափոքր տարրերը։

դ) Տպել հավաքածուի առաջին (վերջին) ամենամեծ տարրը և դրա կարգահամարը։

ե) Տպել հավաքածուի առաջին (վերջին) ամենափոքր տարրը և դրա կարգահամարը։

զ) Տպել հավաքածուի երկրորդ ամենամեծ և երկրորդ ամենափոքր տարրերը։

է) Հաշվել հավաքածուի զույգ թվերի քանակը և միջին թվաբանականը։ Եթե հավաքածուն զույգ թիվ չի պարունակում, տալ այդ մասին հաղորդագրություն։

146. Տրված են (8,9,10) և ('a','b','c','d','e','f','g','h',) երկու հավաքածուները։

ա) Կրկնապատկել առաջին հավաքածուն և տպել այն։

բ) Կցել առաջին ու երկրորդ հավաքածուները և տպել ստացված հավաքածուն։

147. Տրված է հետևյալ հավաքածուն, որի տարրերի մեջ կան ցուցակներ ու հավաքածուներ՝

tuple = (5, 6, 'Python', [5,6], (5, 6), 1, ('M', 'N'), [5, 6],
'Fortran', (0), ('M', 'N'), [5, 6], (5, 6))

ա) Հաշվել [5,6] տարրի կրկնությունների քանակը։

բ) Հաշվել (5, 6) տարրի կրկնությունների քանակը։

գ) Հաշվել ('M', 'N') տարրի կրկնությունների քանակը։

148. Տրված են նույն երկարությունն ունեցող t1 և t2 թվային հավաքածուները։ Կազմել նոր հավաքածու՝ t3, որի i-րդ տարրը t1 և t2 հավաքածուների համապատասխան տարրերից մեծագույնն է։

149. Տրված են նույն երկարությունն ունեցող t1 և t2 թվային հավաքածուները։ Կազմել նոր հավաքածու՝ t3, որի i-րդ տարրը t1 և t2 հավաքածուների համապատասխան տարրերի գումարն է։

150. Տրված է t թվային հավաքածուն։

ա) Հաշվել ու տպել զույգ (կենտ) տարրերի գումարը և միջին թվաբանականը։

բ) Հաշվել ու տպել զույգ (կենտ) կարգահամարներով տարրերի գումարը։

գ) Հաշվել ու տպել զույգ (կենտ) կարգահամարներով տարրերի քառակուսիների արտադրյալը։

դ) Հաշվել ու տպել տրված k թվին բազմապատիկ ինդեքսով տարրերի քանակը։

ե) Հաշվել ու տպել դրական, բացասական և զրոյին հավասար տարրերի քանակը։

զ) Հաշվել դրական (բացասական) տարրերի գումարը, քանակը, միջին թվաբանականը։

է) Հաշվել ու տպել այն տարրերի քանակը, որոնք հավասար են տրված a թվին։ Եթե այդ պայմանին բավարարող թվեր չկան, տալ այդ մասին հաղորդագրություն։

ը) Հաշվել և տպել այն զույգ (կենտ) տարրերի քանակը, որոնք փոքր են կամ հավասար տրված a թվից։ Եթե այդ պայմանին բավարարող թվեր չկան, այդ մասին տալ հաղորդագրություն։

թ) Հաշվել ու տպել այն տարրերի միջին թվաբանականը, որոնք պատկանում են տրված [a, b] հատվածին։ Եթե այդ պայմանին բավարարող թվեր չկան, այդ մասին տալ հաղորդագրություն։

ժ) Հաշվել ու տպել այն տարրերի արտադրյալը, որոնք բազմապատիկ են հինգին և չեն պատկանում տրված [c, d] հատվածին։ Եթե այդ պայմանին բավարարող թվեր չկան, այդ մասին տալ հաղորդագրություն։

151. Տրված է t հավաքածուն, որի տարրերը բնական թվեր են։

ա) Գտնել հինգին բազմապատիկ թվերի քանակը։ Եթե հավաքածուի մեջ հինգին բազմապատիկ թվեր չկան, տալ այդ մասին հաղորդագրություն։

բ) Գտնել այն թվերի քանակը, որոնք հինգի բաժանելիս տալիս են մեկ մնացորդ։ Եթե հավաքածուի մեջ այդ պայմանին բավարարող թվեր չկան, տալ այդ մասին հաղորդագրություն։

152. Տրված է t1 հավաքածուն, որի տարրերը բնական թվեր են։ Կառուցել նոր հավաքածու՝ t2, որը պարունակում է այդ հավաքածուի պարզ (բաղադրյալ) թվերը։

153. Տրված է t1 հավաքածուն, որի տարրերը բնական թվեր են: Կառուցել նոր հավաքածու՝ t2, որը կազմված է t1 հավաքածուի այն տարրերից, որոնք իրենց կարգահամարին բաժանվում են առանց մնացորդի: Հավաքածուի առաջին տարրը (զրո կարգահամարով) ընդգրկել առանց ստուգման:

154. Տրված է t հավաքածուն, որի տարրերը բնական թվեր են:

ա) Արտածել այդ հավաքածուի պարզ թվերի գումարը:

բ) Որոշել ու արտածել զույգ և 5-ով ավարտվող տարրերի քանակը:

Գլուխ 9. Բազմություններ

155. Գրել ծրագիր, որն արտածում է երկու մուտքագրված բազմությունների միավորումն ու հատումը:

156. Տրված L=['i', 'a', 'o', 'g', 'm', 'n', 'r', 'p', 'a', 'o', 'g', 'o', 'm', 'i'] ցուցակից հեռացնել կրկնությունները՝ կիրառելով set() և list() ֆունկցիաները:

157. Մուտքագրել որևէ բազմություն և դրան ավելացնել՝

ա) մեկ չփոփոխվող տարր,

բ) մի քանի չփոփոխվող տարր,

գ) չփոփոխվող տարրերից կազմված բազմություն,

դ) չփոփոխվող տարրերից կազմված ցուցակ,

ե) չփոփոխվող տարրերից կազմված մի քանի ցուցակ,

զ) հեռացնել նշված առկա տարրը,

է) հեռացնել որևէ պատահական տարր,

ը) հեռացնել բոլոր տարրերը:

Գլուխ 10. Բառարաններ

158. Գրել ծրագիր, որն առաջարկում է օգտվողին մուտքագրել ապրանքների անվանումներն ու գները և պահպանել դրանք products անունով բառարանում: Բառարանի յուրաքանչյուր տարրի բանալին ապրանքի անվանումն է, իսկ արժեքը՝ գինը: Մուտքագրել որևէ ապրանքի անվանում և տպել դրա գինը կամ տալ հաղորդագրություն, եթե ապրանքը բառարանում չկա:

159. Օգտագործելով նախորդ խնդրում ստեղծված բառարանը՝ արտածել մուտքագրված գինն ունեցող բոլոր ապրանքների անունները:

160. Ստեղծել բառարան, որի բանալիներն ամիսների անուններն են, իսկ արժեքները՝ համապատասխան ամսվա օրերի քանակը:

ա) մուտքագրել որևէ ամսվա անուն և տպել, թե քանի օր ունի այդ ամիսը,

բ) տպել բառարանի բոլոր բանալիները՝ դասավորված այբբենական կարգով,

գ) տպել բոլոր այն ամիսների անունները, որոնք ունեն 31 օր,

դ) տպել (բանալի-արժեք) զույգերը՝ դասավորված ըստ ամիսների օրերի քանակի աճման կարգի:

ե) տպել (բանալի-արժեք) զույգերը՝ դասավորված ըստ ամիսների անունների այբբենական կարգի:

161. Ստեղծել բառարան, որը պարունակում է տասը օգտանուն ու գաղտնաբառ: Ծրագիրը պետք է առաջարկի օգտվողին մուտքագրել իր օգտանունն ու գաղտնաբառը: Եթե օգտանունը բառարանում չկա, ծրագիրը պետք է տա հաղորդագրություն, որ անձը համակարգի վավեր օգտատեր չէ: Եթե օգտվողի անունը կա բառարանում, բայց օգտվողը ճիշտ չի մուտքագրում գաղտնաբառը, ծրագիրը պետք է տա հաղորդագրություն, որ գաղտնաբառն անվավեր է: Եթե գաղտնաբառը ճիշտ է, ապա ծրագիրը պետք է օգտվողին ասի, որ նա այժմ մուտք է գործել համակարգ:

162. Մուտքագրել ֆուտբոլային ակումբների թիմերի անունները, նրանց հաղթանականների ու պարտությունների քանակները: Պահպանել այս տեղեկատվությունը բառարանում, որտեղ բանալին թիմի անունն է, իսկ բանալու արժեքը՝ [wins, losses] ([հաղթանակներ, պարտություններ]) ցուցակը:

ա) Օգտագործելով ստեղծված բառարանը՝ մուտքագրել որևէ թիմի անուն և տպել այդ թիմի հաղթանակների քանակը:

բ) Օգտագործելով բառարանը՝ ստեղծել ցուցակ, որի տարրերը թիմերի հաղթանակների քանակներն են:

Գլուխ 11. Ֆունկցիաներ

163. Սահմանել ֆունկցիա, որը հաշվում և վերադարձնում է երկու բնական թվերի ամենամեծ ընդհանուր բաժանարարը։

164. Սահմանել ֆունկցիա, որը վերադարձնում է True, եթե թիվը պարզ է, և False, եթե այն բաղադրյալ է։

165. Սահմանել ֆունկցիա, որը վերադարձնում է թվի բաժանարարների գումարն ու քանակը։

166. Սահմանել ֆունկցիա, որը վերադարձնում է True, եթե թիվը կատարյալ է և False՝ հակառակ դեպքում։

167. Սահմանել ֆունկցիա, որը վերադարձնում է թվի՝ հակառակ կարգով գրված թիվը։

168. Գրել ֆունկցիա, որը լուծում է $ax^2 + bx + c = 0$ ($a, b, c \in R, a \neq 0$) քառակուսային հավասարումը։

169. Գրել ֆունկցիա, որը լուծում է երկու անհայտով երկու գծային հավասարումների համակարգը։

170. Գրել ծրագիր, որը հաշվարկում է կազմակերպության աշխատակցի աշխատավարձը՝ կախված տվյալ կազմակերպությունում նրա աշխատանքային ստաժից հետևյալ կանոնով՝ աշխատակցի ֆիքսված աշխատավարձին գումարվում է հավելավճար, որը կազմում է ֆիքսված աշխատավարձի 5%-ը, եթե ստաժը չի գերազանցում 7 տարին, 10%-ը՝ եթե ստաժը 8-12 տարի է և 15%-ը՝ եթե ստաժը 12 տարուց ավելի է։

171. Սահմանել ֆունկցիա, որը հաշվարկում և վերադարձնում է հանրախանութում հաճախորդի կողմից կատարած գնումների արժեքը, եթե նրան տրամադրվում է զեղչ՝ կախված իր կուտակային քարտի վրա ունեցած միավորներից։ Զեղչը կազմում է գնումների արժեքի 5%-ը, եթե կուտակային քարտի միավորները 25-50 են, 8%-ը՝ եթե միավորները 51-100 են, 10%-ը՝ եթե 101-150 են և 15%-ը՝ 150-ից ավելի դեպքում։

172. Սահմանել ֆունկցիա, որին փոխանցվում է երկու բնական թիվ։ Ֆունկցիան վերադարձնում է առաջին թիվը երկրորդին բաժանելիս ստացված ամբողջ մասն ու մնացորդը։

173. Սահմանել ֆունկցիա, որին փոխանցվում է տող և որևէ բնական թիվ: Ֆունկցիան տողը կրկնում է այդքան անգամ և վերադարձնում է ստացված տողը:

174. Սահմանել ֆունկցիա, որը հաշվում ու վերադարձնում է թվի ֆակտորիալը:

Գլուխ 12. Աշխատանք ֆայլերի հետ

175. Գրել ծրագիր, որը ctemp.txt ֆայլից կարդում է օրական միջին ջերմաստիճանների ցուցակը՝ ըստ Ցելսիուսի, դրանք արտահայտում է Ֆարենհայթով (ft=ct*9/5+32 բանաձևով) և արդյունքները գրում ftemp.txt անունով ֆայլի մեջ:

176. Տրված է grades.txt անունով ֆայլը, որի յուրաքանչյուր տող պարունակում է ուսանողի օգտանուն՝ գրված մեկ բառով, և երեք քննությունից ստացած գնահատականները, որոնք իրարից բաժանված են բացակներով, ինչպես բերված է ստորև.

GAdamyan 14 15 18

Tsargsyan 5 10 16

Գրել ծրագիր, որը.

ա) Հաշվում և արտածում է, թե քանի՞ ուսանող է բարեհաջող հանձնել բոլոր երեք քննությունները, եթե յուրաքանչյուր քննության անցողիկ միավորը տասն է:

բ) Յուրաքանչյուր ուսանողի համար հաշվում և արտածում է գումարային միավորը, միջին միավորը:

գ) Արտածում է առավելագույն գումարային միավոր հավաքած ուսանողների ցուցակը (առավելագույն գումարային միավորը՝ 60):

դ) Արտածում է այն ուսանողների թիվը, որոնց գումարային միավորը բարձր է ընդհանուր գումարային միջինից:

177. Տրված է logfile.txt անունով տեքստային ֆայլը, որը պարունակում է տեղեկատվություն օգտատերերի՝ համակարգից օգտվելու վերաբերյալ: Ֆայլի յուրաքանչյուր տող պարունակում է համակարգից օգտվողի օգտանունը և համակարգ մուտք գործելու ու ելքի ժամանակները, որոնք բաժանված են ստորակետներով հետևյալ ձևով՝ օգտատիրոջ անուն, մուտքի ժամանակ և ելքի ժամանակ: Ֆայլի բնորոշ տողը հետևյալն է.

Armen Sahakyan, 14:22, 14:37

Ժամերը տրված են 24-ժամյա ձևաչափով։ Բոլոր մուտքերն ու ելքերը տեղի են ունենում մեկ աշխատանքային օրվա ընթացքում։

Գրել ծրագիր, որը տպում է բոլոր այն օգտատերերի ցուցակը, որոնք համակարգում են եղել առնվազն մեկ ժամ։

178. Տրված է gdp.txt տեքստային ֆայլը, որը պարունակում է աշխարհի 195 երկրների ՀՆԱ-ի արժեքները 2023 թվականի համար։ Գրել ծրագիր, որը դասավորում է երկրների ցուցակը ըստ ՀՆԱ նվազման կարգի և արտածում է առաջին տասնյակը։

179. Տրված է wordlist.txt ֆայլը, որի յուրաքանչյուր տողում գրված է որևէ բառ (տառերի կամայական հաջորդականություն) անգլերենով, ընդ որում, բոլոր բառերն իրարից տարբեր են։ Պահանջվում է գտնել և արտածել.

ա) բոլոր այն բառերը, որոնք ավարտվում են 'th'-ով,

բ) բոլոր այն բառերը, որոնց երկրորդ, երրորդ և չորրորդ տառերը 'tho' են,

գ) քանի՞ բառ կա ֆայլում, որոնք պարունակում են 'r', 's', 't', 'l', 'n', 'e' տառերից առնվազն մեկը,

դ) բոլոր այն բառերը, որոնք չեն պարունակում ձայնավորներ,

ե) բոլոր այն բառերը, որոնք պարունակում են բոլոր ձայնավորները,

զ) ամենաերկար բառը,

է) բոլոր պոլինդրոմները (այն բառերը, որոնք աջից և ձախից նույն ձևով են կարդացվում),

ը) բոլոր այն բառերը, որոնք պարունակում են 'q' տառը, բայց որը չի հաջորդում 'u' տառին,

թ) բոլոր բառերը, որոնք պարունակում են 'zu',

ժ) բոլոր բառերը, որոնք պարունակում են 'ab' մեկից ավելի անգամ,

ի) բոլոր բառերը, որոնք պարունակում են և՛ 'z', և՛ 'w',

լ) բոլոր բառերը, որոնց առաջին տառը 'a' է, երրորդ տառը՝ 'e', իսկ հինգերորդ տառը՝ 'i',

խ) բոլոր երկտառ բառերը,

ծ) բոլոր չորս տառ ունեցող բառերը, որոնք սկսվում և ավարտվում են նույն տառով,

կ) բոլոր բառերը, որոնց առաջին երկու և վերջին երկու տառերը նույնն են,

հ) 'abcd'*'dcba' ձևի բոլոր բառերը, որտեղ *-ը կամայական հաջորդականություն է,

ձ) այն բառը, որն ունի ամենաշատ 'i'-երը։

Գլուխ 13. Մոդուլներ

180. Ստեղծել figures.py մոդուլը, որը պարունակում է տարբեր ֆունկցիաներ՝ պատկերների պարագծերը և մակերեսները հաշվելու համար։

Շրջանի համար ստեղծել circle_perimeter() ֆունկցիան, որը հաշվում է շրջանագծի երկարությունը և circle_area() ֆունկցիան, որը հաշվում է շրջանի մակերեսը։ radius արգումենտի համար սահմանել լռելյայն արժեք՝ 1։

Եռանկյան համար ստեղծել triangle_perimeter() ֆունկցիան, որը հաշվում է եռանկյան պարագիծը և triangle_area() ֆունկցիան, որը հաշվում է եռանկյան մակերեսը։ Եռանկյան կողմերի երկարությունների համար սահմանել լռելյայն արժեքներ՝ 1, 2, 3 համապատասխանաբար։

Քառակուսու համար ստեղծել square_perimeter() ֆունկցիան, որը հաշվում է քառակուսու պարագիծը և square_area() ֆունկցիան, որը հաշվում է քառակուսու մակերեսը։ Քառակուսու կողմի (a) համար սահմանել լռելյայն արժեք՝ 1։

Նույն պանակում ստեղծել երկրորդ ֆայլը՝ main_program1.py անունով, որտեղից պետք է ներբեռնել figures.py մոդուլի ֆունկցիաները։

181. Ստեղծել strings_mod.py մոդուլը, որը պարունակում է տարբեր ֆունկցիաներ՝ տողային տիպի տվյալների հետ աշխատելու համար։ Դրանք են.

str_isalpha() ֆունկցիան ստուգում է, թե արդյոք տողի բոլոր տարրերը տառե՞ր են, թե՞ ոչ և վերադարձնում է True կամ False համապատասխանաբար։

str_reverse() ֆունկցիան վերադարձնում է տողի շրջված տողը։

str_concat() ֆունկցիան կցում է արգումենտում ստացված տողերը և վերադարձնում կցված տողը։

str_max() ֆունկցիան գտնում և վերադարձնում է տողի ամենամեծ տարրը և դրա ինդեքսը։

Նույն պանակում ստեղծել երկրորդ ֆայլը՝ main_program2.py անունով, որտեղից պետք է ներբեռնել այդ մոդուլի ֆունկցիաները։

182. Ստեղծել numbers_mod.py մոդուլը, որը պարունակում է տարբեր ֆունկցիաներ՝ բնական թվերի հետ աշխատելու համար։ Դրանք են՝

num_gcd() ֆունկցիան վերադարձնում է թվի բաժանարարների քանակը։

num_gcd_sum() ֆունկցիան վերադարձնում է թվի բաժանարարների գումարը։

num_prime() ֆունկցիան վերադարձնում է True, եթե թիվը պարզ է, և False՝ հակառակ դեպքում։

num_perfect() ֆունկցիան վերադարձնում է True, եթե թիվը կատարյալ է և False՝ հակառակ դեպքում։

num_reverse() ֆունկցիան վերադարձնում է թվի՝ հակառակ կարգով գրված թիվը։

Նույն պանակում ստեղծել երկրորդ ֆայլը՝ main_program3.py անունով, որտեղից պետք է ներբեռնել այդ մոդուլի ֆունկցիաները։

183. Ստեղծել seq_mod.py մոդուլը, որը պարունակում է տարբեր ֆունկցիաներ՝ հաջորդականությունների հետ աշխատելու համար։ Դրանք են.

seq_reverse() ֆունկցիան վերադարձնում է հաջորդականությունը՝ գրված հակառակ հերթականությամբ։

seq_isalpha() ֆունկցիան ստուգում է, արդյոք հաջորդականության բոլոր տարրերը տառե՞ր են, թե՞ ոչ և վերադարձնում է True կամ False համապատասխանաբար։

seq_min() ֆունկցիան գտնում և վերադարձնում է հաջորդականության ամենափոքր տարրը և դրա ինդեքսը։

seq_len() ֆունկցիան վերադարձնում է հաջորդականության երկարությունը։

Նույն պանակում ստեղծել երկրորդ ֆայլը main_program4.py անունով, որտեղից պետք է ներբեռնել այդ մոդուլի ֆունկցիաները։

Գլուխ 14. Բացառություններ

184. Գրել ծրագիր, որը հավաքածուների հետ աշխատելիս մշակում է հետևյալ հնարավոր բացառությունները՝ հավաքածուի տարրերի թարմացում, որևէ տարրի հեռացում, գոյություն չունեցող տարրի դիմում։

185. Գրել ծրագիր, որը բառարանների հետ աշխատելիս մշակում է հետևյալ հնարավոր բացառությունները՝ հղում գոյություն չունեցող բառարանի, հղում գոյություն չունեցող բանալիով տարրի։

186. Գրել ծրագիր, որը տեքստային ֆայլերի հետ աշխատելիս մշակում է հետևյալ հնարավոր բացառությունները՝

- փորձ է արվում գրելու կամ կարդալու համար բացել գոյություն չունեցող ֆայլ,

- փորձ է արվում գրել այնպիսի ֆայլի մեջ, որում գրելու թույլտվություն չկա։

- փորձ է արվում ջնջել գոյություն չունեցող ֆայլ։

187. Գրել ծրագիր, որը մշակում է հետևյալ բացառությունը՝ էլեկտրոնային հասցեն վավեր չէ (չի պարունակում '@' սիմվոլը)։ Կատարել հետևյալ քայլերը՝

- Սահմանել նոր բացառության դաս՝ InvalidEmailError, որը ժառանգում է Exception բազային դասից։ Դասի ատրիբուտներն են՝ էլեկտրոնային հասցե՝ email, հաղորդագրություն՝ msg="Invalid email format"։

- Դասի __str__() մեթոդը վերադարձնում է msg հաղորդագրությունը։

- Երբ տրված էլեկտրոնային հասցեն չի պարունակում '@' սիմվոլը, ապա բացառությունը ծագում է,

- «try-except» բլոկը որսում է սխալը և տպում է msg հաղորդագրությունը։

188. Գրել ծրագիր, որը մշակում է հետևյալ բացառությունը՝ ապահովելու համար, որ մարդու տարիքը լինի (0–120) վավեր միջակայքում: Կատարել հետևյալ քայլերը.

- Սահմանել նոր բացառության դաս՝ InvalidAgeError, որը ժառանգում է Exception բազային դասից: Դասի ատրիբուտներն են՝ տարիքը (age), հաղորդագրություն՝ msg="Age must be between 0 and 120":

- Դասի __str__() մեթոդը վերադարձնում է msg հաղորդագրությունը:

- set_age() ֆունկցիան օգտագործում է ստեղծված InvalidAgeError բացառությունը՝ մուտքագրված տարիքը ստուգելու համար: Այդ բացառությունը ծագում է, եթե տարիքը գտնվում է վավեր միջակայքից (0–120) դուրս՝ age < 0 կամ age > 120:

- «try-except» բլոկը որսում է սխալը և տպում է "Age must be between 0 and 120" հաղորդագրությունը:

Գլուխ 15. Օբյեկտ-կողմնորոշված ծրագրավորման հիմունքները

189. Սահմանել Vector դասը, որի օբյեկտները երկչափ տարածության վեկտորներ են: Վեկտորի սկզբնակետը համընկնում է ուղղանկյուն դեկարտյան կոորդինատների համակարգի սկզբնակետի հետ, իսկ վերջնակետը՝ այդ համակարգում տրված կետի հետ:

Դասի ատրիբուտներն են՝

– հարթության կետի կոորդինատները,

Դասի մեթոդները և ֆունկցիաներն են՝

– կետի հեռավորությունը կոորդինատների սկզբնակետից հաշվող ֆունկցիան,

- հորիզոնական առանցքի հետ վեկտորի կազմած անկյան կոսինուսը հաշվող ֆունկցիան:

ա) Ստեղծել դասի vector1, vector2 օբյեկտները: Օբյեկտները ստեղծելիս օգտագործել __init__() մեթոդը՝ ատրիբուտներին արժեքներ վերագրելու համար: Արտածել օբյեկտի ատրիբուտների արժեքները:

բ) Ջնջել vector2 օբյեկտը:

190. Սահմանել Person դասը, որի օբյեկտները մարդիկ են։

Դասի ատրիբուտներն են՝

– անուն (name), ազգանուն (surname), մասնագիտությունը (profession)։

Դասի մեթոդն է՝

– printname() մեթոդը, որն արտածում է անունը, ազգանունը և մասնագիտությունը։

ա) Ստեղծել Person դասի prs օբյեկտը։

բ) Սահմանել Doctor ենթադասը, որը Person դասից ժառանգում է նույն հատկություններն ու մեթոդները։ Doctor ենթադասի օբյեկտները բժիշկներ են։

գ) Ստեղծել Doctor ենթադասի Doc օբյեկտը։

դ)Արտածել օբյեկտի Doc ատրիբուտների արժեքները։

191. Սահմանել Person դասը, որի օբյեկտները մարդիկ են։

Դասի ատրիբուտներն են՝

– անուն (name), ազգանուն (surname), տարիք (age)։

Դասի մեթոդներն են՝

– printname() մեթոդը, որն արտածում է անունը, ազգանունը և տարիքը։

ա) Ստեղծել Person դասի prs օբյեկտը։

բ)Սահմանել Student ենթադասը, որի օբյեկտներն ուսանողներ են։ Student ենթադասը Person դասից ժառանգում է նույն հատկություններն ու մեթոդները։

գ) Ավելացնել Student ենթադասին հասցե (address) հատկությունը։

դ) Ավելացնել Student ենթադասին test_result() մեթոդը, որն արտածում է "You have successfully passed the exam!" տեքստը։

ե) Ստեղծել Student ենթադասի std օբյեկտը։ Արտածել օբյեկտի անունը, ազգանունը և քննությունը հաջողությամբ հանձնելու մասին տեքստը։

192. Սահմանել Shape, Rectangle, Circle դասերը, որտեղ Shape-ը բազային դաս է Rectangle, Circle դասերի համար, որից ժառանգված դասերի օբյեկտները երկրաչափական պատկերներ են։ Rectangle դասի օբյեկտները ուղղանկյուններ են, իսկ Circle դասի օբյեկտները՝ շրջաններ։

Shape դասում սահմանել պատկերի մակերեսը և պարագիծը հաշվող ֆունկցիաներ՝ area() և perimeter(), որոնց մարմինը դատարկ է: Rectangle և Circle ենթադասերը պետք է ժառանգեն Shape ծնող դասից area() և perimeter() մեթոդները և վերասահմանեն դրանք:

193. Սահմանել Person դասը: Դասը պետք է ունենա երկու private տիպի ատրիբուտ՝

- name (String)՝ "Anahit" լռելյայն արժեքով:
- age (int)՝ 27 լռելյայն արժեքով:

Դասը պետք է ապահովի Public մեթոդներ՝ այս private ատրիբուտներին հասանելիություն ունենալու և դրանք փոփոխելու համար՝ օգտագործելով getter և setter հետևյալ մեթոդները՝

getter մեթոդներ՝ get_name() և get_age()

setter մեթոդներ՝ set_name(name) և set_age(age):

194. Սահմանել Vehicle (տրանսպորտային միջոց) աբստրակտ դասը: Դասն ունի՝

- @property դեկորատորով հայտարարված Color() աբստրակտ հատկությունը,
- @abstractmethod դեկորատորով հայտարարված աբստրակտ move() մեթոդը:

Աբստրակտ հատկությունը և մեթոդը իրականացված չեն դասի ներսում:

Car և Boat դասերը Vehicle աբստրակտ դասի ենթադասեր են: Car ենթադասի օբյեկտները մեքենաներ են, իսկ Boat ենթադասինը՝ նավակներ: Car և Boat ենթադասերը պետք է իրականացնեն Color() հատկությունը, որն արտածում է տրանսպորտային միջոցի գույնը, և move() մեթոդը, որն արտածում է, թե որտեղ է շարժվում տվյալ տրանսպորտային միջոցը: Օրինակ, մեքենայի համար՝ "The car moves on the road", իսկ նավակի համար՝ "The boat sails on the water": Ապա ստեղծել car և boat օբյեկտները և արտածել համապատասխան փոխադրամիջոցի գույնն ու տեղաշարժվելու վայրը:

РЕЗЮМЕ

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА PYTHON

Гаяне Гукасян, Марине Буниатян, Мамикон Каноян

Учебное пособие описывает основы программирования на языке Python. Пособие составлено в соответствии с учебной программой предмета «Информационные технологии в экономике», преподаваемого на экономическом факультете ЕГУ, и включает материалы, используемые в процессе обучения. Учебное пособие предназначено для студентов экономического факультета ЕГУ. Оно может быть полезно студентам и преподавателям смежных специальностей, а также читателям, желающим освоить программирование на языке Python.

Учебное пособие представляет учебный материал в следующих разделах: форматы представления чисел в компьютерах, встроенные в язык Python типы данных, операторы, команды, основные структуры, входящие в язык: строки, одномерные и двумерные списки, кортежи, множества, словари, а также методы и функции, встроенные в язык и относящиеся к этим структурам. Принцип процедурного программирования представлен на основе механизма пользовательских функций. Описаны методы работы с файлами, стандартные системные модули языка и средства создания модулей пользователем, механизмы обнаружения и обработки исключительных ситуаций в программах на Python. Представлены основные понятия и принципы объектно-ориентированного программирования, методы создания и использования классов и их иерархии в языке Python. Каждый раздел содержит теоретический материал, сопровождаемый примерами и программными кодами. По основным темам сформулированы и решены многочисленные задачи. Пособие также включает многочисленные задания для самостоятельного решения.

SUMMARY

THE FUNDAMENTALS OF PYTHON PROGRAMMING

Gayane Ghukasyan, Marine Buniatyan, Mamikon Kanoyan

This tutorial describes the fundamentals of Python programming. It is designed in accordance with the curriculum of the "Information Technologies in Economics" course taught at the Faculty of Economics at Yerevan State University and includes materials used in the course. This tutorial is intended for students of the Faculty of Economics at Yerevan State University. It may be useful for students and faculty of related fields, as well as for readers wishing to master Python programming.

This tutorial presents the educational material in the following sections: number representation formats in computers; data types built into the Python language; operators; commands; the main structures included in the language (strings, one-dimensional and two-dimensional lists, collections, sets, and dictionaries); as well as methods and functions built into the language and related to these structures. The principle of procedural programming is presented using the mechanism of user-defined functions. This book describes file handling methods, standard system language modules, user-created module tools, and mechanisms for detecting and handling exceptions in Python programs. It also presents the fundamental concepts and principles of object-oriented programming, as well as methods for creating and using classes and their hierarchies in Python. Each section contains theoretical material, supported by examples and program code. Numerous problems are formulated and solved on the main topics. The book also includes numerous assignments for independent work.

ՕԳՏԱԳՈՐԾՎԱԾ ԳՐԱԿԱՆՈՒԹՅԱՆ ՑԱՆԿ

1. Python documentation. www.python.org.
2. Brian Heinold. A Practical Introduction to Python Programming. Department of Mathematics and Computer Science. Mount St. Mary's University. 2012. 263 p.
3. Lundh, Frederick. The Python Imaging Library Handbook. http://effbot.org/imagingbook/.
4. Lutz, Marc. Learning Python, 5th ed. O'Reilly Media, 2013.
5. Марченко, А. Л. Python: Большая книга примеров. Москва: Издательство Московского университета, 2023. 361 с.
6. Маккини У. Python и анализ данных / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 540 с.: ил.
7. Սարգսյան Ս. Գ., Հովակիմյան Ա. Ս., Հակոբյան Ս. Ռ. Մեքենայական ուսուցում Python լեզվի կիրառմամբ: Ուսումնական ձեռնարկ: -Եր., ԵՊՀ հրատ., 2025, 162 էջ:
8. Python Tutorial. https://www.geeksforgeeks.org/python/python-programming-language-tutorial/

ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

ԳԱՅԱՆԵ ՂՈՒԿԱՍՅԱՆ, ՄԱՐԻՆԵ ԲՈՒՆԻԱԹՅԱՆ,
ՄԱՄԻԿՈՆ ԿԱՆՈՅԱՆ

PYTHON ԾՐԱԳՐԱՎՈՐՄԱՆ ՀԻՄՈՒՆՔՆԵՐ

ՈՒՍՈՒՄՆԱԿԱՆ ՁԵՌՆԱՐԿ

Համակարգչային ձևավորումը՝ Կ. Չալաբյանի
Կազմի ձևավորումը՝ Ա. Պատվականյանի
Հրատ. խմբագրումը՝ Լ. Հովհաննիսյանի

Հրատարակվել է՝ 09.06.2026։
Չափսը՝ 60x84 1/16։ Տպ. մամուլը՝ 18.625։
Տպաքանակը՝ 100։

ԵՊՀ հրատարակչություն
ք. Երևան, 0025, Ալեք Մանուկյան 1
www.publishing.ysu.am

publishing.ysu.am