Окуучулар үчүн
Python
тилинде программалоо

«Сорос-Кыргызстан» Фонду
Бишкек 2019

УДК 373.167.1
ББК 73 Я 721
И 74

Окуучулар үчүн Python тилинде программалоо: Python программалоо тилин үйрөнүү үчүн окуу куралы / Л. Самыкбаева, А. Беляев, А. Палитаев, И. Ташиев, С.Маматов – «Сорос-Кыргызстан» Фонду, 2019 – 80 б.

И 74

ISBN 978-9967-31-911-0

Бул окуу куралы программалоонун негиздерин үйрөнгүсү келген бардык балдарга арналат. Китеп Python программалоо тили боюнча киришүү курсу болуп саналат жана программалоого үйрөнүү үчүн өздөштүрүш керек болгон негизги түшүнүктөр (берилиштер, өзгөрмөлөр, циклдер, функциялар жана графика) менен тааныштырат. Азыркы күндө Python дүйнөдөгү эң популярдуу программалоо тилдерине кирет. Анын негизинде жөнөкөй тиркемелер жана оюндар гана эмес, автоматташтырылган системалар үчүн татаал программалар да иштелип чыгат. Python тили жаңы үйрөнүп жаткандар үчүн эң жөнөкөй тил, ошондуктан дүйнөнүн көпчүлүк өлкөсүндө аны үйрөнүүнү мектептен башташат. Бул окуу курал Python программалоо тилин мектептин «Информатика» сабагынын программасынын алкагында да, өз алдынча үйрөнүүдө да колдонулушу мүмкүн.

Окурмандардын кеңири чөйрөсү үчүн

Окуучулар үчүн Python тилинде программалоо

Китептин э-версиясы bizdin.kg сайтына жайгаштырылган.

Котормочу: Касымбек Жунусалиев
Текст редактору: Ысмайыл Кадыров
Компьютердик калыпка салган: Абдымалик Токталиев

soros.kg

«Сорос-Кыргызстан» Фонду, 2019

МАЗМУНУ

1. **Python программалоо тили** **4**

2. **Маалыматтардын тиби жана алар менен болгон амалдар** **10**

3. **Шарттуу операторлор** **14**

4. **while жана for циклдери** **17**

5. **Татаал шарттар: and, or, not** **20**

6. **Тизмелер, кортеждер жана сөздүктөр** **22**

7. **Циклдик алгоритмдер** **25**

8. **Камтылган шарттуу амалдар жана циклдер** **30**

9. **Функциялар** **34**

10. **Массивдер** **40**

11. **Саптар жана алар менен болгон амалдар** **45**

12. **Саптарды форматтоо** **51**

13. **Python тилинде графика менен иштөө** **53**

14. **Рекурсия** **58**

15. **Массивдерди иштетүү алгоритмдери** **62**

16. **Тизмелерди сорттоо** **69**

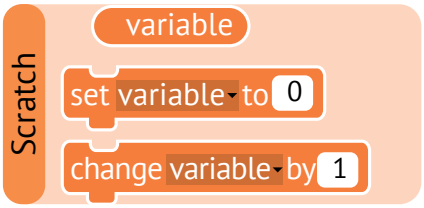
17. **Матрицалар** **74**

18. **Тиркемелер** **79**

1-тема:

Python программалоо тили

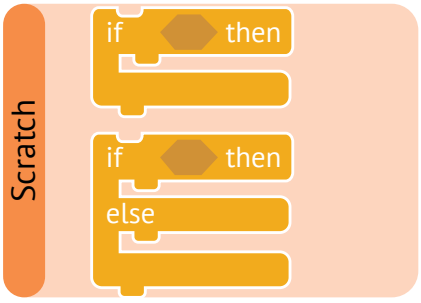
Python программалоо тили – бул жаңы үйрөнүп жаткандар үчүн да жеткиликтүү болгон, ар түркүн багыттагы программаларды түзүү үчүн колдонулган аспап. Эгерде силер Scratchте блоктор менен программа түзүп жүргөн болсоңор, анда Python тилинде программа түзүү силерге оңой эле болот. Келгиле, командалар Python жана Scratch тилдеринде кандай жазыла тургандыгын салыштыралы.




```
variable = 0

variable = variable +1
```
Python

Шарттар ушундай көрүнөт:




```
if True :
 #бир нерсе жасоо керек

if True :
 #бир нерсе жасоо керек
else:
 #бир нерсе жасоо керек
```
Python

Python коду оңой окулат, ал эми интерактивдүү кабыкчасы болсо программаны киргизип, ошол замат жыйынтыгын алганга мүмкүндүк берет.

Бүгүнкү күндө бул программалоо тилинде банктар, телекоммуникациялык компаниялар үчүн программалар жазылат, көптөгөн аналитиктер ушул тилдин жардамындагы маалыматтар менен иштешет. Өтө жөнөкөй синтаксистин натыйжасында бул тилде программалоону баштоо жеңил.

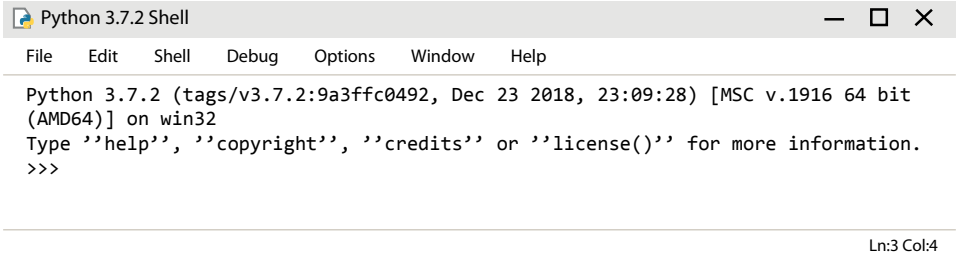
Python тилинде программалоо үчүн өзүңөрдүн компьютериңерге акысыз программалоо чөйрөсүн орнотушуңар керек. Аны ***https://www.python.org/downloads/*** сайтынан жүктөп аласыңар.

Python менен бирге компьютерде программаларды киргизүү үчүн IDLE – иштетүү чөйрөсү да орнотулат.

Python-командаларды жазуу, сактоо жана аткаруу үчүн төмөнкү кадамдарды жасоо керек болот:

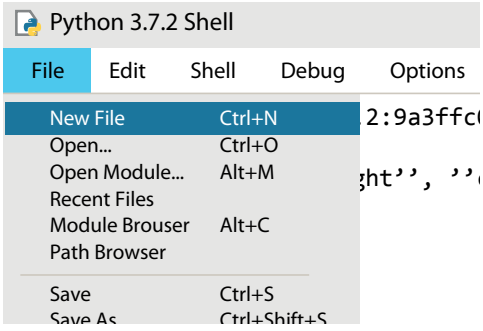
1 -кадам. IDLEни ишке киргизүү

Түзгөн программаңардын жыйынтыгын көрүп тургандай консоль терезеси ачылат.



2 -кадам. Жаңы файлды түзүү

Жаңы программаны жазуу үчүн өзүнчө файл түзүү керек. Ал үчүн **File** менюсунан **New Fileды** тандап алабыз.



3 -кадам. Программаны киргизүү

Ачылган терезеде өзүңүздүн программаңызды киргизиңиз, мисалы:

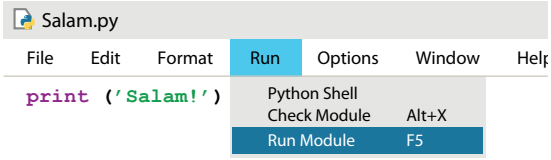
```
print ('Salam!')
```

4 -кадам. Программаны сактоо

File менюсунан **Save As** дегенди тандап, файлыңар үчүн жаңы атты киргизгиле жана **Save** командасын баскыла. Биз алдын ала My scripts папкасын түзүүнү жана ошол жерге өз программаңардын бардыгын сактоону сунуштайбыз.

5 -кадам. Программаны ишке киргизүү

Run менюсунан **Run Module** командасын тандагыла (же тез иштетүү үчүн F5 баскычын баскыла)



Даяр! Консолдо силер төмөнкү маалыматты көрүшүңөр керек:

```
>>>
Salam!
```

Программа биринчиден файлга жазылып (адатта **.py** кеңейтилиши менен) жана ишке киргизүүдө толугу менен аткарылган ыкма *программалык режим* деп аталат. Мындай программалык файл Pythondo скрипт деп аталат (англ. ***Script*** – сценарий).

Pythondo андан тышкары бир да операторду (команданы) камтыбаган бош программалар да болот. Көбүнчө булар комментарийлер (адатта # белгисинен кийин жазылат) – интерпретатор менен иштетилбеген түшүндүрмөлөр.

```
# бул бош программа
```

АНЫКТАМА

Интерпретатор – бул силердин программаңарды окуп, андагы камтылган нускамаларды аткаруучу программа.

Python тилиндеги функциялар

Pythondo көптөгөн функциялар бар, аларды чакырууда андагы камтылган командаларды программалар аткарышат.

`print()`	`input()`	`randint()`
`print` функциясы маалыматты экранга чыгарып берет. Ал өзгөрмөлөрдүн маанилерин, ошондой эле туюнтманын маанисин да чыгара алат.	`input` функциясы тескерисинче, колдонуучу тарабынан программага берилиштерди киргизүүгө мүмкүндүк берет.	`randint` функциясы белгилүү диапазондон кокустук санды кайтарат.

*Кашааларга бул функция иштей турган маанилери көрсөтүлөт.

`print` **функциясы** апострофко же тырмакчаларга алынган символдорду экранга чыгарат. Бул команда аткарылгандан кийин автоматтык түрдө кийинки жаңы сапка өтөт.

Өзгөрмөлөр

Эки санды кошууну аткаруучу программанын алгоритмин түзөбүз:
1) колдонуучудан бүтүн эки санды сурайт;
2) аларды кошот;
3) кошуунун жыйынтыгын чыгарат.

АНЫКТАМА

Өзгөрмө – бул атка, тибине жана маанисине ээ болгон чоңдук. Өзгөрмөнүн мааниси программанын аткарылуу учурунда өзгөрө алат.

Маселенин шартында эске сакташ керек болгон маалыматтар пайда болду дейли. Ушулар үчүн өзгөрмөлөрдү колдонушат. Өзгөрмөлөр (каалагандай эс уячасындай) бир гана маанини сактай алат. Ага жаңы маанини жазууда мурункусу өчүрүлөт жана аны эч кандай калыбына келтирүүгө да болбойт.

Python тилинде өзгөрмөлөр биринчи колдонулганда эле, тагыраак айтканда аларга алгачкы маанилерди ыйгарууда эсте түзүлөт.

Мисалы ыйгаруу операторун аткарууда:

```
a = 4
```

эсте **a** аталыштагы жаңы өзгөрмө түзүлөт («бүтүн сан» тибиндеги объект). Эми ушул ат менен өзгөрмөгө кайрыла берсек болот: анын маанисин окуйбуз жана өзгөртөбүз.

Өзгөрмөлөрдүн атында латын тамгаларын (кичине жана баш тамгалар айырмалуу болот), цифраларды жана астына сызылган сызыкты («_») колдонсо болот.

Колдон келсе өзгөрмөлөргө ал кандай роль аткара тургандыгын ошол замат түшүнө тургандай тиешелүү аттарды бериш керек.

ЭСИҢЕ ТУТ

Өзгөрмөнүн аты цифра менен башталбаш керек, анткени трансляторго кайсы жерден цифра, ал эми кайсы жерден өзгөрмөнүн аты баштала тургандыгын айырмалоо кыйын болуп калат.

Pythondo өзгөрмөнүн тиби автоматтык түрдө аныкталат. Өзгөрмөнүн тиби жөнүндө кеңири маалыматты кийинки темадан карайбыз.

Алгоритмди жазуу үчүн биз үч маселени чечишибиз керек:

1. клавиатурадан эки санды киргизип, аларды өзгөрмөлөргө жазуу (аларды **a** жана **b** деп атайлы);
2. бул эки санды кошуп, жыйынтыгын **c** деген үчүнчү өзгөрмөгө жазуу;
3. **c** өзгөрмөсүнүн маанисин экранга чыгаруу.

Өзгөрмөнүн маанисин киргизүү үчүн `input` оператору колдонулат:

```
a = input()
```

Бул сап аткарылгандан кийин система өзгөрмөнүн маанисин клавиатурадан киргизүүнү күтөт. Enter баскычын басканда, бул маани **a** өзгөрмөсүнө жазылып калат. `input` операторун чакырууда кашаанын ичине түшүндүрмө билдирүүнү жазып койсо болот:

```
a = input ('Бүтүн санды киргиз: ')
```

Эки маанини кошуп, жыйынтыгын **c** өзгөрмөсүнө жазуу өтө оңой:

```
c = a + b
```

«=» символу – бул ыйгаруу оператору, анын жардамы менен өзгөрмөнүн маанилери өзгөртүлөт. Ал төмөнкүдөй аткарылат: «=» символунун оң жагындагы туюнтма аткарылат, андан соң анын жыйынтыгы сол жагында жазылган өзгөрмөгө ыйгарылат. Ошондуктан, мисалы

`i = i + 1`, бул команда **i** өзгөрмөсүнүн маанисин бирге көбөйтөт.

c өзгөрмөсүнүн маанисин экранга `print` оператору менен чыгарабыз:

```
print (c)
```

Бардык программа:

Python тилиндеги программа	Экранга чыккан жыйынтыгы

`a = input ('Бүтүн сан киргизиңиз:')`
`b = input ('Бүтүн сан киргизиңиз:')`
`c = a + b`
`print (c)`
	Бүтүн сан киргизиңиз: 2
	Бүтүн сан киргизиңиз: 3
	23

Биз көрүп тургандай эки сан кошулган жок: программа аларды биринин артына экинчисин жазып, бириктирип эле койду. Анткени мындай жазууда `input` оператору киргизилген маанилерди сан катары эмес, символ катары кабыл алат.

Бул катаны оңдош үчүн маанилерди киргизүүдө алынган символдук сапты бүтүн санга өзгөртүп түзүшүбүз керек. Бул болсо **int** (англ. *integer* – бүтүн) функциясынын жардамында ишке ашат.

```
a = int (input())
b = int (input())
```

Дагы бир варианты бар: эки сан эки сапта эмес бир сапта бош орун менен ажыратылып киргизилет. Бул учурда маанини киргизүү кыйыныраак:

```
a, b = map (int, input().split())
```

map() – берилген функцияны тизменин бардык элементтерине тиешелүү кылат; биздин учурда бул саптагы элементтерди бүтүн санга айланткан **int**() функциясы.

split() – сапты маселелерге жараша бөлүктөргө бөлөт; жыйынтыгында тизмени кайтарат.

Жыйынтыгында **map** () функциясынын ишинен кийин биз эми сандардан турган жаңы тизмени алабыз. Биринчи киргизилген сан (тизменин биринчи элементи) **a** өзгөрмөсүнө, ал эми экинчиси – **b** өзгөрмөсүнө жазылат.

Берилген функцияларды эске алуу менен программаны кайра жазалы:

Python тилиндеги программа	Экранга чыккан жыйынтыгы

`a, b = map (int, input().split())`
`c = a + b`
`print (c)`
	2 3
	5

Эми программа туура иштеп жатат – клавиатурадан киргизилген эки санды кошуп жатат. Бирок мунун эки кемчилиги бар:
1) берилиштерди киргизүүдө колдонуучудан эмне талап кылынып жаткандыгы билинбейт (канча, кандай сандарды киргизүү керек);
2) жыйынтыгы эч нерсени билдирбеген сан түрүндө чыгат.

Ошондуктан программа менен колдонуучунун ортосунда диалогду түзүү үчүн программаны мындайча өзгөртсө болот:

Python тилиндеги программа	Экранга чыккан жыйынтыгы

`print ('Эки бүтүн сан киргизиңиз:')`
`a,b = map(int, input().split())`
`c = a + b`
`print (a,'+',b,'=',c,sep='')`
	Эки бүтүн сан киргизиңиз:
	2 3
	2 + 3 = 5

Колдонуу үчүн маалыматты өзүңүз каалагандай жазсаңыз болот. Жыйынтыгын чыгарууда үч өзгөрмөнүн маанисин жана `print` операторунда бөлүнгөн эки символду «+» жана «=» чыгаруу керек.

```
print (a, '+', b, '=', c)
```

`print` операторундагы ашыкча боштуктарды өчүрүү үчүн бөлгүч (же сепаратор, англ. *separator*) – **sep** колдонулат.

```
print (a, '+', b, '=', c, sep = '')
```

Бул жерде биз бош бөлгүчтү (бош сап) орноттук. Бөлгүч катары каалагандай белгини көрсөтсө болот. Мисалы, эгерде командада sep = `'*'` деп көрсөтсөк

```
print (1, 2, 3, 4, sep = '*')
```

анда экранда төмөнкү жыйынтык чыгат: `1*2*3*4*`

КОМПЬЮТЕРДИК ПРАКТИКУМ:

Убакыт интервалы саат, минута жана секунда менен берилген. Ошол эле интервалды секундада гана көрсөтө турган программаны түзгүлө.

2-тема:

Маалыматтардын тиби жана алар менен болгон амалдар

Өзгөрмөлөрдү колдонуудан мурда алардын типтери менен таанышалы. Мүмкүн болгон аракеттердин көптүгү силердин тандап алган тибиңерге көз каранды болот.

Маалыматтар тиби

Python тилиндеги негизги маалыматтар типтерин тизмелейли:

`int` – бүтүн маанилер;
`float` – чыныгы сан маанилери (бөлчөктүү бөлүгү менен сандар);
`bool` – логикалык маанилер, **True** (чындык «ооба») же **False** (жалган «жок»);
`str` – символ же символдук сап, б.а. символдордун чынжыры (катары).

Pythondогу бүтүн өзгөрмөлөр өтө чоң (же, тескерисинче сөз терс сандар жөнүндө болуп жатса кичинекей) болушу мүмкүн: интерпретатор автоматтык түрдө эсептөөнүн жыйынтыгын сактоого керек болгон өлчөмдө эстин аймагын бөлүп берет. Ошондуктан Pythonдо көп орундуу сандар менен эсептөөлөрдү так жүргүзүү жеңил.

Чыныгы сандарды жазууда бүтүн бөлүгү бөлчөк бөлүгүнөн үтүр менен эмес, чекит менен ажыратылат. Мисалы:

```
x = 123.456
```

Логикалык өзгөрмөлөр `bool` тибине кирет жана `True` (чындык) же `False` (жалган) маанилерин алат.

Python тилиндеги программа	Экранга чыккан жыйынтыгы

`a = 10`
`b = 3`
`c = a/b`
`print ('c =', float (c))`
	c = 3.3333333333333335

`a = 10`
`b = 3`
`c = a/b`
`print ('c =', int (c))`
	c = 3

`a = 10`
`b = 3`
`c = a/b`
`print ('c =', bool (c))`
	c = True

Арифметикалык туюнтмалар жана аракеттер

Pythondo каалагандай арифметикалык амалдарды аткарса болот. Арифметикалык туюнтмалар сапка жазылат. Алар сандарды, өзгөрмөлөрдүн аттарын, арифметикалык амалдардын белгилерин, кашааларды (аракеттердин иретин өзгөртүү үчүн) жана функцияларды чакыруларды камтышы мүмкүн. Мисалы,

```
a = (c + 5 - 1) / 2 * d
```

Амалдардын иретин аныктоодо амалдардын артыкчылыгы (улуулугу) колдонулат. Алар төмөнкү иретте аткарылат:

1. кашаанын ичиндеги амалдар;
2. солдон оңго даражага көтөрүү (**);
3. солдон оңго көбөйтүү (*) жана бөлүү (/);
4. солдон оңго кошуу жана кемитүү.

Тең күчтүү туюнтмалардын жазуулары

```
a = b = 0
```

```
b = 0
a = b
```

```
a += b
a -= b
a *= b
a /= b
```

```
a = a + b
a = a - b
a = a * b
a = a / b
```

Бөлүүнүн жыйынтыгы ар дайым чыныгы сан болоорун эстен чыгарбаш керек. Ал эмес бөлүнүүчү жана бөлүүчү да бүтүн болуп, бири-бирине бөлгөндө бүтүн сан чыкса да ал чыныгы сан. Бөтөн сандарды бөлгөндө бүтүн сан алуу үчүн «//» операторун колдонушат. Ал эми бөлүүнүн калдык бөлүгүн алуу үчүн «%» операторун колдонушат (алар көбөйтүү жана бөлүүдөй эле артыкчылыкка ээ).

Python тилиндеги программа	Экранга чыккан жыйынтыгы

`d = 85`
`a = d // 10`
`print (a)`
`b = d % 10`
`print (b)`
	8
	5

Терс сандар үчүн

`print (-7 // 2)`	-4
`print (-7 % 2)`	1

Сандар теориясынын көз карашы боюнча, калдык – бул оң сан, ошондуктан -7 = (-4)*2 + 1, башкача айтканда -7ни 2ге бөлгөндөгү тийинди -4кө барабар, ал эми калдык 1ге барабар болот.

Pythondo даражага көтөрүү амалы эки жылдызча менен белгиленет: «**». Мисалы y = $2x^2+z^3$ туюнтмасы мындай жазылат:

```
y = 2*x**2 + z**3
```

1-маселе. Үч бурчтуктун аянтын табуучу программа түзгүлө, эгерде анын негизинин узундугу жана бийиктиги белгилүү болсо.

Геометриядан белгилүү болгондой үч бурчтуктун аянты үч бурчтуктун негизинин (a) жарымын анын бийиктигине (h) көбөйткөнгө барабар:

$$S = \frac{1}{2}ah$$

Бул формуланы мындай жазсак да болот: s=(a*h)/2

Эми программаны түзөлү жана үч бурчтуктун негизинин узундугун **6 см** жана бийиктигин **4 см** деп киргизели.

Python тилиндеги программа

```
a = float(input('Негизинин маанисин киргизиңиз: '))
h = float(input('Бийиктигинин маанисин киргизиңиз: '))
s = (a*h)/2
print ('Жообу: s=',s)
```

Экранга чыккан жыйынтыгы

```
Негизинин маанисин киргизиңиз: 6
Бийиктигинин маанисин киргизиңиз: 4
Жообу: s= 12.0
```

Кокустук сандар

Кокустук бүтүн санды алуу үчүн алгач Pythonго **randint** функциясын жүктөйлү. Ал үчүн консоль терезесинде **import** командасын колдонобуз:

```
>>> from random import randint
>>> randint (1, 10)
7
```

randint() функциясы биринчи сандан экинчи санга чейинки диапазондо кокустук санды тандап алды. Биздин мисалда ал 7 цифрасын тандады.

Стандарттык функциялар

Python тилиндеги көптөгөн стандарттык функциялар кызматы боюнча топторго бөлүнгөн. Ар бир топ **модуль** деп аталган өзүнчө файлга жазылган. Математикалык фукциялар **math** модулунда топтолгон.

Бул модулду кошуу үчүн импорт командасы (модулду жүктөө) колдонулат – `import math`.

Төмөндө сандар менен иштөөчү кээ бир функциялар көрсөтүлгөн:

Команда	Аткарган аракети
`int(x)`	x чыныгы санын калдык бөлүгүн алып салуу менен бүтүн санга айландыруу
`round(x)`	x чыныгы санын жакынкы бүтүн санга чейин тегеректөө
`ceil(x)`	жакынкы чоң санга чейин тегеректөө
`floor(x)`	төмөнкү кичинекей санга чейин тегеректөө
`abs(x)`	сандын модулун эсептөө (абсолюттук чоңдугун)
`sqrt(x)`	x санынын квадраттык тамырын чыгаруу
`sin(x)`	синус x (радиан менен көрсөтүлөт)
`cos(x)`	косинус x (радиан менен көрсөтүлөт)
`tan(x)`	тангенс x (радиан менен көрсөтүлөт)

Функцияларга кайрылуу үчүн чекиттик жазуу колдонулат: алгач модулдун атын, чекиттен кийин функциянын атын көрсөтүү керек:

```
print (math.sqrt(x))
```

Python тилиндеги программа	Экранга чыккан жыйынтыгы
`x = 25` `print (math.sqrt(x))`	5.0

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *Температура Цельсий градуста берилген. Берилген температураны Фаренгейт градуста туюндуруучу программаны түзгүлө. Туюндуруу үчүн формула – t(F) = 9/5 *t(C) + 32.*

2) *Туюнтманын маанисин эсептөөчү программаны жазгыла:*

$$\frac{(a+b)*c+d*\frac{e}{f}}{k+p*\frac{b}{a}+g}*\frac{4}{5}$$

3-тема:

Шарттуу операторлор

Буга чейинки караган мисалдарда операторлор биринин артынан бири удаалаш аткарылган ***сызыктуу*** программаларды жазууга мүмкүндүк берген. Алардын аткарылышы киргизилген маалыматка көз каранды эмес.

Көпчүлүк реалдуу маселелерде кандай маалыматтар келип түшкөнүнө жараша аракеттердин ирети бир аз өзгөрүшү мүмкүн.

Эгерде аракеттин эки вариантынан тандаш керек болсо, анда алгоритмди жазуу үчүн тармактуу конструкция колдонулат. Python тилинде тармактануу шарттуу операторлор аркылуу ишке ашат. Маанисине карата шарттуу операторлор программаны кайсы бир жол боюнча багыттайт. Мисалы, өрт сигналдык системасынын программасы билдиргичтерден алынган маалыматтар температуранын жогорулашынан же түтүн каптоодон кабар берсе, тынчсыздануу сигналын таратышы керек.

`if` **шарттуу оператору** биринчиден шартты текшерет жана андан кийин гана андан аркы нускама боюнча аткаруу же аткарбоо чечимин кабыл алат. `if` оператору кандай иштешин түшүнүү үчүн шартты текшерүү жана тандоого типтүү маселелерден карап көрөлү.

1-маселе. Жашы 21ден ашкандар үчүн гана уруксат бере турган программаны түзөлү.

```
a = int(input('Өзүңүздүн жашыңызды киргизиңиз:  '))
if a >= 21:
      print('Уруксат')
else:
      print('Уруксат эмес')
```

`if` операторундагы шарт кашаага алынбастан жазылат жана кош чекит («:») менен жыйынтыкталат. Шарттын кийинки «бутактары» жаңы саптан оңго жылдыруу менен жазылат.

Эгерде `if` операторунан кийин жазылган шарт туура (чындык) болсо, анда кийинки `else` шарттуу операторуна чейинки бардык командалар аткарылат. Эгерде `if` операторунан кийин жазылган шарт туура эмес (жалган) болсо, анда түз эле `else` операторунан кийинки командалар аткарылат. Биздин мисалда, эгер жашты 13 деп киргизсек, анда `else` шарттуу операторунан кийинки нускама аткарылат: б.а. уруксат берилбейт.

Python тилинде операторлордун сол жактагы чеги боюнча жылышы да чоң мааниге ээ. Көңүл бурсаңар `if` жана `else` сөздөрү бир деңгээлде башталат, ал эми ички блоктогу бардык командалар бул деңгээлге караганда оң жакка бирдей аралыкка жылышкан. Жылдыруу үчүн табуляция символу колдонулат: Tab баскычына бир жолу же төрт жолу бош орун баскычын басуу.

Эгерде бир нече окшош шарттарды киргизүү керек болсо, анда андан кийин нускамалар блогу кеткен кошумча `elif` (`else` – `if`тин кыскартылганы) блогун колдонсо болот.

```
a=int(input('Өзүңүздүн жашыңызды кир-
гизиңиз: '))
if a >= 21:
      print('Уруксат')
elif a >= 18:
      print('Жарым-жартылай уруксат')
else:
      print('Уруксат эмес')
```

ЭСИҢЕ ТУТ

Python тилинде шарттуу операторлуу нускамалардын баш сөзүнүн аягына сөзсүз кош чекит коюлат.

Салыштыруу операторлору

Салыштыруу операторлору эки маанини бири-бири менен салыштырып жыйынтыгында True же False деген маанисин берет.

Математикалык символ	Python оператору	Мааниси	Мисал	Жыйынтык
<	<	Кичине	1 < 2	True
>	>	Чоң	1 > 2	False
⩽	<=	Кичине же барабар	1 <= 2	True
⩾	>=	Чоң же барабар	1 >= 2	False
=	==	Барабар	1 == 2	False
≠	! =	Барабар эмес	1 ! = 2	True

2-маселе. Компания элдин оюн билүү боюнча сурамжылоо жүргүзүп жатат жана аларды 20дан 70 жашка чейинки адамдар кызыктырат. Суралуучунун жашын сураган жана ал ошол белги боюнча «туура келээрин» же «туура келбесин» сураган программа түзгүлө.

`v` өзгөрмөсүнө адамдын жашы жазылсын дейли. Анда программанын керектүү фрагменти төмөнкүдөй болот:

```
if v >= 20 and v <= 70:
    print ('туура келет')
else:
    print ('туура келбейт')
```

Python тилинде кош барабарсыздыктарга уруксат берилет, мисалы:

```
if A < B < C:
```

деген төмөндөгү менен бир мааниде:

```
if A < B and B < C:
```

ЭСИҢЕ ТУТ

= өзгөрмө үчүн маанини ыйгарат (эгер a = b болсо, анда a b болуп калат);

== эки маанини салыштырат (эгер a==b болсо, анда бул салыштырууга суроо-талап, жана программа True же False деген жыйынтыкты чыгарат).

3-маселе. Шарттуу операторлорду жана салыштыруу операторлорун колдонуп, файлды сактоо программасын жазгыла.

```
ans = input('Сиз файлды сактагыңыз келеби? (ооба/жок)')
if ans == 'ооба':
    print('Сактоо үчүн папканы тандаңыз')
if ans == 'жок':
    print('Маалыматтар өчүрүлөт, андан ары колдоно албайсыз')
else:
    print('Ката. Жооптун мындай варианты жок')
```

Чыгуунун жыйынтыгы кандай жооп тандалгандыгына көз каранды болот:

1-вариант:

Сиз файлды сактагыңыз келеби?
(ооба/жок) ооба
Сактоо үчүн папканы тандаңыз

2-вариант:

Сиз файлды сактагыңыз келеби?
(ооба/жок) жок
Маалыматтар өчүрүлөт, андан ары колдоно албайсыз

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *Үч сан берилди. Алардын ичинен канча сан бирдей экенин аныктоочу программа түзгүлө.*
2) *Үч кесиндинин узундуктары берилди. Бул кесиндилер үч бурчтуктун жактары боло ала тургандыгын текшерүүчү программаны түзгүлө.*
3) *Натуралдык сан берилди. Бул сан жуп экенин, 3кө эселүү экенин аныктоочу программаны түзгүлө.*
4) *Сом, доллар жана евро валюталарын алмаштыруу боюнча программа жазгыла.*

4-тема:

while жана for циклдери

Циклдер шарттуу операторлор сыяктуу эле программалоонун маанилүү бөлүгү болуп саналат. Алардын жардамы менен коддун кээ бир бөлүктөрүн кайталатууну уюштурса болот. Python тилинде циклдерди жазуу үчүн эки түрдөгү командалар колдонулат: **`while`** жана **`for`**.

`while` цикли

«While» англис тилинен «ошондой болгон учурда» деп которулат, башкача айтканда цикл (командалардын блогу) берилген шарт аткарылмайынча кайталана берет. Ал үчүн ар бир циклдин кадамынын башында шартты текшерүү аткарылат. Ошондуктан ал баштапкы шарты бар цикл деп аталат.

1-маселе. 1ден 5ке чейинки бардык бүтүн сандарды экранга чыгаралы.

Python тилиндеги программа	Экранга чыккан жыйынтыгы

`d = 0`
`while d<5:`
`  d+=1`
`  print (d)`
	1
	2
	3
	4
	5

Баштапкы шарт мындайча текшерилет: эгерде d өзгөрмөсүнүн мааниси баштапкы учурда 5тен чоң же ага барабар болсо, анда цикл бир жолу да кайталанбайт.

2-маселе. Мындай мисалды карайлы: бүтүн оң сандуу ондук ситемада цифралардын санын аныктоо керек. Баштапкы сан бүтүн типтеги n өзгөрмөсүнө жазылган деп эсептейли.

Маселени чыгаруу үчүн, мааниси циклдин ар бир өтүшүндө өзгөрүп туруучу **эсептегич** өзгөрмөнү колдонобуз. Цифралардын санын эсептөө үчүн ар бир өтүштө эсептегичти чоңойтуу менен бул цифраларды башынан же аягынан бирден бөлүп алып туруш керек. Эсептегичтин баштапкы мааниси нөлгө барабар, анткени алгоритмди аткарууга чейин бир да цифра табыла элек. Акыркы цифраны бөлүп алууда санды бөлчөксүз 10 санына бөлүп коюу жетиштүү. Сандарды бөлүп алуу жана эсептегичти көбөйтүү амалдарын санда канча цифра болсо ошончо жолу аткаруу зарыл.

Качан гана кийинки 10го бөлүүнүн жыйынтыгында бүтүн бөлүгү нөлгө барабар болгондо, бул цикл аяктады дегенди түшүндүрөт.

Python тилинде программа мындай жазылат:

```
count = 0
while n > 0:
      n = n // 10
      count += 1
```

Циклдин айлануусунун саны киргизилген сандын цифрасына барабар болот, башкача айтканда баштапкы берилишке көз каранды. Эгерде циклдин башындагы шарт бузулбаса, анда цикл чексиз иштей берет. Бул учурда «программа циклден чыкпай калды» деп айтышат. Циклден чыкпай калган программаны токтотуу үчүн Ctrl+C баскычын консоль терезесинде басуу керек.

`for` цикли

`for` цикли командаларды керектүү жолу кайталап, программаны кыскартууга мүмкүндүк берет. Жогорку мисалда `for` циклин колдонолу:

Python тилиндеги программа

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

Экранга чыккан жыйынтыгы

```
0
1
2
3
4
```

Бул жерде i өзгөрмөсү (муну циклдин өзгөрмөсү деп аташат) 0дөн 5ке чейинки, 5 өзү кирбейт (б.а. 0дөн 4кө чейин) диапазондо (`in range`) өзгөрөт. Ошентип цикл туптуура 5 жолу кайталанат.

`while` менен жазылган программага окшош жооп алуу үчүн `for` циклин колдонуп программаны өзгөртөлү:

Python тилиндеги программа

```
d = 0
while d < 5 :
      d+=1
      print ( d )
```

```
for i in range(1,6):
    print ( i )
```

Экранга чыккан жыйынтыгы

```
1
2
3
4
5
```

2-маселе. Эки санынын 2^1 нен 2^{10} уна чейин даражаларын чыгарабыз (k = экинин даражалары).

Тең күчтүү туюнтмалардын жазуулары

```
k = 1
while k <= 10 :
    print ( 2**k )
    k += 1
```

```
for k in range(1,11):
    print ( 2**k )
```

Биринчи вариантта **k** өзгөрмөсү үч жолу колдонулат: баштапкы маанисин ыйгарууда, циклдин шартында жана циклдин тулкусунда (1ге чоңойтуу).

Экинчи вариантта **k** өзгөрмөсү баштапкы жана акыркы маанилериндеги эки сандын диапазонунда берилет, мында акыркы маани диапазонго ***кирбейт.***

Циклдик өзгөрмөнүн өзгөрүү **кадамы** берилбесе 1ге барабар болот. Эгерде аны өзгөртүү керек болсо **range** сөзүнөн кийин кашаанын ичинде үчүнчү (кошумча) санды киргизишет – бул керектүү кадам. Мисалы, мындай цикл 2 санынын так даражаларын гана чыгарат:

Python тилиндеги программа	Экранга чыккан жыйынтыгы
`for k in range(1,11,2):` `  print ( 2**k )`	2 8 32 128 512

Циклдин ар бир кадамы менен циклдин өзгөрмөсү өсүүдө эле болбостон, кемүүдө да болушу мүмкүн. Ал үчүн баштапкы мааниси акыркы маанисинен чоң, ал эми кадам – терс болуш керек. Төмөнкү программа 5тен 1ге чейинки натуралдык сандардын квадраттарын кемүү тартибинде чыгарат:

Python тилиндеги программа	Экранга чыккан жыйынтыгы
`for k in range(5,0,-1):` `  print ( k**2 )`	25 16 9 4 1

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *А жана В бүтүн сандарын алган жана Адан Вга чейинки диапазондогу бардык натуралдык сандардын квадраттарын чыгара турган программаны түзгүлө.*
2) *Натуралдык сан берилген. Ал сандын цифраларынын суммаларын чыгарып берген программаны жазгыла.*
3) *Узундугу 20 метр болгон тактай берилген. Бул тактайдан узундуктары 1,5 м жана 2 м болгон бүтүн сандагы канча минималдык кесиндиин даярдоого боло турганын эсептөөчү программаны түзгүлө.*

5-тема:

Татаал шарттар: and, or, not

Программалоодо шартты туура коё билүү өтө маанилүү. Көпчүлүк учурда шарттар татаал болушат, б.а. «ЖАНА», «ЖЕ» жана «ЭМЕС» логикалык операторлору (байламта) менен бириккен бир нече курама шарттардан турушат. Python тилинде алар «and», «or», «not» деген англис сөздөрү менен жазылат.

`and` логикалык оператору (логикалык көбөйтүү)

Туюнтмада `and` байламтасы менен бириккен курама шарттардын баары тең `True` маанисине барабар болсо, анда татаал шарт `True` маанисин кайтарат. Эгер эки туюнтманын бирөө эле жалган болсо, анда шарт жалган:

```
x = 5
if x < 10 and x % 3 == 0:
      print('True')
else:
      print('False')
```

Бул жерде жооп False болот, анткени шарттын экинчи бөлүгүнө ылайык берилген 5 саны 3кө калдыксыз бөлүнбөйт. Эгерде биз туюнтманы x%3==0 `and` x<10 деп жаза турган болсок, ал кайрадан эле False маанисин кайтармак. Бирок мындагы экинчи салыштыруу x<10 интерпретатор тарабынан аткарылмак эмес, аны аткаруунун кажети жок. Анткени биринчи туюнтма (x%3==0) жалган болгондуктан `and` операторунун болушу бардык туюнтманы жалганга чыгарды.

`or` логикалык оператору (логикалык кошуу)

Эгерде жок дегенде бир туюнтма True маанисине ээ болсо True маанисин кайтарат:

```
x = 5
if x < 10 or x % 3 == 0:
      print('True')
else:
      print('False')
```

Бул жерде жооп True болот, анткени шарттын биринчи бөлүгүнө ылайык берилген 5 саны 3кө калдыксыз бөлүнбөсө дагы 10 санынан кичине. Мына ушул үчүн эгерде эки туюнтманын бирөөсү эле True маанисин кайтарса, анда экинчи туюнтма бааланбайт, анткени `or` оператору баары бир True маанисин кайтарат.

`not` логикалык оператору (логикалык тануу)

`Not` унардык операторy чындыкты жалганга кайтарат, ал эми жалганды чындыкка кайтарат. Унардык дегенибиз, анткени ал `and` жана `or` операторлорундай болуп анын оң жагында же сол жагында турган туюнтмаларга эмес андан кийин турган бир эле туюнтмага колдонулат.

1-вариант:

```
x = 8
print (not x < 15)

False
```

2-вариант:

```
x = 8
print (not x > 15)

True
```

Эгерде бир туюнтмада бир эле убакта бир нече же бардык логикалык операторлор колдонулса, анда аткаруу тартиби төмөнкүдөй болот:

1) катыш (<, >, <=, >=, ==, !=)
2) not («ЭМЕС»)
3) and («ЖАНА»)
4) or («ЖЕ»)

Аракеттердин иретин өзгөртүү үчүн тегерек кашаалар колдонулат. Кашаалар пайда болгон учурдагы эсептөөлөрдүн иретинин өзгөрүшүн мисалда карап көрөлү:

1-вариант:

```
a=4
b=6
c=8
result = c==8 or b<a and not a < 7
print (result)
Жыйынтык:
True
Эмне үчүн экендигин түшүндүрөлү:
   c == 8 or b < a and not a < 7
    True      False       True
                          False
                 False
          True
```

2-вариант:

```
a=4
b=6
c=8
result= (c==8 or b<a) and not a < 7
print (result)
Жыйынтык:
False
Эмне үчүн экендигин түшүндүрөлү:
  (c == 8 or b < a) and not a < 7
   True      False       True
             True        False
                 False
```

СУРООЛОР ЖАНА ТАПШЫРМАЛАР:

1) and операторунун жардамы менен бирөө чындыкты, экинчиси– жалганды көрсөткөн эки татаал логикалык туюнтманы түзгүлө.
2) Жогорудагы маселени or операторун колдонуп аткаргыла.

6-тема:

Тизмелер, кортеждер жана сөздүктөр

Көпчүлүк учурда бизге көптөгөн окшош маалыматтарды бир файлда чогултууга туура келет, мисалы, окуучулардын тизмеси же маалымдамадагы телефон номерлер. Python тилинде мындай маалыматтардын топтомун **тизмелерде, кортеждерде** жана **сөздүктөрдө** уюштурса болот.

Тизме (***list***) – бул белгилүү иретте жайгашкан элементтерден турган структура. Ар бир элементке ага кайрылууга мүмкүн болгон номер (же индекс) туура келет. Тизмени түзүү үчүн квадраттык кашаанын ичине үтүр менен ажыратылып, анын бардык элементтери тизмектелет.

Мисалга өзүбүздүн үй-бүлө мүчөлөрүбүздүн тизмесин түзөлү:

```
>>> myfamily = ['father', 'mother', 'sister', 'brother']
```

Бул учурда биздин тизме myFamily деген өзгөрмөдө сакталат. Качан тизме түзүлгөндөн кийин, бул тизме менен иштей турган программаны жазсак болот. Мисалга, циклди колдонуп, үй-бүлөнүн ар бир мүчөсү менен саламдашуу программасын жазалы:

Python тилиндеги программа	Экранга чыккан жыйынтыгы

`myfamily = ['father', 'mother', 'sister', 'brother']`
`for item in myfamily:`
`  print('Hello', item)`
	Hello father
	Hello mother
	Hello sister
	Hello brother

Тизме ар кандай типтеги объекттерди камтышы мүмкүн. Бир эле тизмеге бир убакта сапты, сандарды, башка типтеги объекттерди киргизүүгө болот:

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

Тизмелерди бири-бирине кошсо болот, анда жаңы тизме эки тизмедеги тең элементтерди камтып калат:

```
x = [1, 2, 3, 4]
y = [5, 6, 7, 8]
z = x + y
print (z)
```

Жыйынтык:

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

Тизмелер менен ар түрдүү көп амалдарды жасаса болот:

x in A	**x** элементи **A** тизмесинде бар же жок экенин текшерет. True же False маанисин кайтарат.	`A = [1, 2, 3, 4, 5, 6, 7, 8]`
		`print (2 in A)`
		`Жыйынтык:`
		`True`

min(A)	**A** тизмесинен эң кичине элементти табуу.	`A = [1, 2, 3, 4, 5, 6, 7, 8]`
		`print (min(A))`
		`Жыйынтык:`
		`1`

max(A)	**A** тизмесинен эң чоң элементти табуу.	`A = [1, 2, 3, 4, 5, 6, 7, 8]`
		`print (max(A))`
		`Жыйынтык:`
		`8`

Кортеж (***tuple***) тизме сыяктуу эле элементтердин удаалаштыгынан турат. Бирок андагы элементтерди өзгөртүүгө, кошууга же өчүрүүгө болбойт. Кортеждерди түзүү үчүн үтүр менен бөлүнгөн анын маанилери жайгашкан тегерек кашаалар колдонулат:

```
user = ('Timur', 23, 1/10/1998)
print(user)
```

Кортеждерде объекттердин касиеттерин сактоо ыңгайлуу, мисалы, атын, жашын, туулган датасын. Эгерде кортеж болгону бир элементтен турса, анда кортеждин жалгыз элементинен кийин үтүр белисин коюу керек:

```
user = ('Tom',)
```

Сөздүктөр (***dictionary***) – бул ар бир элементи индекстин ордуна уникалдуу ачкычка ээ болгон маалыматтардын структурасы. Сөздүктүн элементтерин өзгөртө берсе болот. Сөздүктөрдү түзүү үчүн фигуралык кашаалар колдонулат ({}):

```
dictionary = {ачкыч1:мааниси1, ачкыч2:мааниси2, ...}
```

mychool атындагы сөздүктү түзөлү:

```
myschool = {'5 klass':'Anara, Kanat, Pavel', '6 klass':
'Chyngyz, Tina, Emil'}
```

Бул сөздүктө ачкыч катары класстын аттары, ал эми мааниси катары – ошол класстарда окугандардын аттары берилди.

Сөздүккө аны жаңы ачкыч менен белгилеп маанилерди кошсо болот:

```
myschool['7 klass'] = 'Elena, Ainura, Dastan'
print (myschool)
```

Жыйынтык:

```
{'5 klass': 'Anara, Kanat, Pavel', '6 klass': 'Chyngyz,
Tina, Emil', '7 klass':'Elena, Ainura, Dastan'}
```

Элементтин маанисин өзгөртүү үчүн, анын ачкычына жаңы маани берип коюу керек:

```
myschool = {'5 klass':'Anara, Kanat, Pavel', '6
klass':'Chyngyz, Tina, Emil'}
myschool['6 klass'] = 'Matvei, Tina, Salima'
print (myschool)
```

Жыйынтык:

```
{'5 klass':'Anara, Kanat, Pavel', '6 klass':'Matvei, Tina, Salima'}
```

`for` циклин колдонуп, экранга сөздүктүн ачкычтарын эле чыгарса болот:

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

Жыйынтык:
```
5 klass
6 klass
```

Же болбосо сөздүктүн маанисин гана чыгарса болот:

```
for i in myschool:
    print(myschool[i])
```

Жыйынтык:
```
Anara, Kanat, Pavel
Chyngyz, Tina, Emil
```

КОМПЬЮТЕРДИК ПРАКТИКУМ:

«Балыктар» сөздүгүн түзгүлө, ал эми анын элементтерин 3 түргө бөлгүлө: «дарыя», «көл» жана «деңиз» балыктары деп. Экранга биринчи сөздүктүн ачкычтарын, андан кийин элементтерин чыгаргыла.

7-тема:

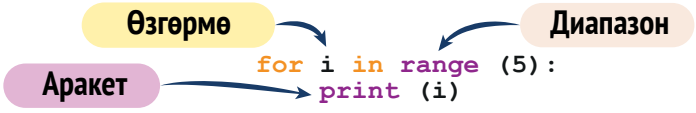
Циклдик алгоритмдер

Каалагандай алгоритмдер 3 конструкциянын жардамы менен жазылаары далилденген: цикл, шарттуу операторлор, сызыктуу алгоритмдер менен.

Силер билгендей, **цикл** – бул окшош аракеттерди көп жолу аткаруу.

Буга чейин биз **while** жана **for** циклдерин үйрөнүп баштаганбыз. **For** цикли **while** циклинен кайсы бир командаларды алдын ала белгилүү санда кайталоо үчүн колдонулгандыгы менен айырмаланат. Ал эми **while** цикли тескерисинче кайсы бир аракеттерди канча жолу кайталай тургандыгы бизге белгилүү болбогон учурларда колдонулат. Ошону менен бирге бизге ошол аткарылганга чейинки циклди кайталаш керек болгон шарт белгилүү.

For циклинин колдонулушун кеңири карап көрөлү. For циклинин Python тилинде жазылышы төмөнкү схема боюнча жүрөт:



Цикл башталганда диапазондун биринчи элементинин мааниси белгиленген аракет аткарыла турган өзгөрмөгө ыйгарылат. Циклдин экинчи өтүшүндө өзгөрмөгө диапазондун экинчи элементинин мааниси ыйгарылат. Жана ушул сыяктуу диапазондогу бардык элементтер менен берилген аракеттер жүргүзүлүп бүтмөйүнчө кайталана берет. Келгиле мисал карайлы: **letter** өзгөрмөсүнө ар бир жолу ***Python*** сабынын жаңы элементи ыйгарылып турсун. Print командасы экранга бул саптын ар бир тамгасын бирден чыгарат:

```
for letter in 'Python':
    print (letter, 'тамгасы', )
>>>
P тамгасы
y тамгасы
t тамгасы
h тамгасы
o тамгасы
n тамгасы
```

Төмөнкү мисалда ар бир кийинки өтүүдө өзгөрмөнүн мааниси берилген диапазондогу санга көбөйүп турат:

```
f = 12
for i in range(1,6):
    f = f + i
print (f)
>>>
27
```

«for i in range(1,6)» цикли беш жолу аткарылат (6 – кирбейт). Циклдин ар бир кадамында f өзгөрмөсү i санына өсүп турат. Баштапкы мааниси f = 12. Циклде маанилери өзгөрүп турат:

1-өтүү: f = 12+1=13
2-өтүү: f = 13 +2=15
3-өтүү: f = 15+3=18
4-өтүү: f = 18+4=22
5-өтүү: f = 22+5=27

Кыскача мындай кылып жазсак болот: f = 12+1+2+3+4+5 = 27

Range функциясынын аргументтери төмөнкүдөй берилет:

- **range (x)** – 0 дөн x ке чейинки маанилерди алат, бирок x – диапазонго кирбейт;
- **range (y, x)** – y тен x ке чейинки бардык маанилерди алат, мында да x диапазонго кирбейт;
- **range (y, x, s)** – y тен x ке чейинки бардык маанилерди s кадамы менен алат.

Мисалы:

```
for i in range(0,15,3):
    print(i)
```

Берилген мисалда for цикли 0дөн 15ке чейинки маанилерди 3 кадам менен алат, жыйынтыгында ал ар бир үчүнчү санды чыгарып берет:

```
>>>
0
3
6
9
12
```

Андан тышкары кадам үчүн терс сандарды да колдонсо болот, анда цикл маанилерди тескери багытта тандап ала баштайт:

```
for i in range(100,0,-20):
    print(i)
>>>
100
80
60
40
20
```

Удаалаштыкты белгилүү санда кайталаган **for** циклинен айырмаланып **while** цикли саны менен эмес, логикалык шарты менен жетектелет. Ошондуктан кодду канча жолу аткараарынын так санын билүүнүн кажети жок.

While циклинин коду логикалык шарт чындык маанисинде (True) болуп турганга чейин кайталана берет.

1-маселе. Ушул циклдин негизинде оюндун программасын түзүп көрөлү. Мында колдонуучу компьютер тарабынан катылган санды табышы керек:

```
import random #кокустук сандар китепканасын жүктөйбүз
number = random.randint(1, 25) #компьютер кокустук санды тандайт
choices = 0  #choices өзгөрмөсүнө аракеттердин санын жазабыз
while choices < 5: #циклди 5 аракетке чейин аткарат
    print('1ден 25ке чейинки санды тап:')  #колдонуучуга
санды киргизүүнү сунуштайт
    guess = input()
    guess = int(guess) #киргизилген сан бүтүн болуш керек
    choices = choices + 1 #ар бир аракетте эсептөө 1ге өсүп турат
    if guess == number:  #эгерде киргизилген сан катылган санга
барабар болсо
        break  #программаны токтот
```

choices өзгөрмөсүнө 0 мааниси ыйгарылган. Ал санды табуу боюнча жасалган ар бир аракет сайын көбөйө берет. Биз программа чексиз циклге түшүп калбашы үчүн аракеттердин санын 5өө менен чектейбиз.

Программа иштеп жатат, бирок ал колдонуучуга эч кандай жыйынтыкты билдирбейт: колдонуучу катылган санды таптыбы же тапкан жокпу, билбейт. Жыйынтыгы мындай көрүнөт:

```
1ден 25ке чейинки санды тап:
5
1ден 25ке чейинки санды тап:
16
```

```
1ден 25ке чейинки санды тап:
7
1ден 25ке чейинки санды тап:
18
1ден 25ке чейинки санды тап:
10
>>>
```

Ал үчүн колдонуучуга анын саны катылган сандан чоң же кичине экенин билдирип тургандай шарттуу операторлорду киргизели. Бул болсо санды тезирээк тапканга жардам берет:

```
import random
number = random.randint(1, 25)
choices = 0
while choices < 5:
    print('1ден 25ке чейинки санды тап:')
    guess = input()
    guess = int(guess)
    choices = choices + 1
    if guess < number: #сан катылган сандан кичине болсо
        print('Менин саным сеникинен чоң')
    if guess > number: #сан катылган сандан чоң болсо
        print('Менин саным сеникинен кичине')
    if guess == number: #сан катылган санга барабар болсо
        break
if guess == number:
    print('Азамат! Сен санды' +str(choices)+ 'аракеттен кийин тап-
тың!')
else:
    print('Тилекке каршы сен санды тапкан жоксуң. Мен' + str(number) +
'санын каткам')
```

Эгерде программаны ишке киргизсек, анда колдонуучу менен баарлашуу варианты төмөнкүдөй болот:

```
1ден 25ке чейинки санды тап:
6
Менин саным сеникинен чоң
```

```
1ден 25ке чейинки санды тап:
17
Менин саным сеникинен кичине
1ден 25ке чейинки санды тап:
14
Менин саным сеникинен чоң
1ден 25ке чейинки санды тап:
15
Азамат! Сен санды 4 аракеттен кийин таптың!
>>>
```

Эми программа колдонуучуга санды табуу үчүн жардамдашат. Мисалы, компьютер 15 санын катса, ал эми колдонуучу 17 санын киргизсе, программа киргизилген сан катылган сандан чоң экендигин көрсөтөт.

СУРООЛОР ЖАНА ТАПШЫРМАЛАР:

1) *Төмөнкү программанын иштөөсүнүн жыйынтыгында алынган y өзгөрмөсүнүн маанисин жазгыла:*

```
y = 5
for i in range(2,6):
      y = y + 4 * i
print (y)
```

2) *Узундугу 20 м тактай берилген. Бул тактайдан узундугу 5 м жана 2 м болгон канча минималдык бүтүн кесиндилерди даярдоого боло тургандыгын чыгаруучу программаны түзгүлө.*

3) *Берилген код боюнча алгоритмдин ар бир кадамындагы өзгөрмөлөрдүн маанилерин таблицага жазгыла:*

k=4 p=1040 m=2

```
while p != m*m:
      k=k+1
      p=p-4
      m=m*2
print (k)
```

k	p	m	m*m

4) *Майнкрафттын каарманы Алекстин минутасына 4 минерал чыгара турган машинасы бар. Ар бир 100 минералга дал ушундай эле минутасына 4 минерал чыгарган машина курууга болот. Бир саат-тан кийин Алекстин канча машинасы боло тургандыгын аныктоочу программаны жазгыла.*

8-тема:

Камтылган шарттуу амалдар жана циклдер

Алгач биз силер менен `if` жана `else` операторлору кандай иштей тургандыгын көрдүк эле. Программада алар аткаруунун эки вариантын көрсөтөт. Бирок программанын алгоритми экиден көп жолду тандоону сунушташы мүмкүн, мисалы үчөөнөн, төртөөнөн же андан көптөн.

Шарттуу операторлордун ичинде ар кандай операторлор, анын ичинде башка шарттуу операторлор да камтылышы мүмкүн. Мисалы бизде мештин ичиндеги билдиргичтен алынган температуранын көрсөтмөсү бар дейли. Эми ал жогорубу, төмөнбү же нормадабы, аныкташ керек. Нормалдуу деп 200дөн 250 градус Цельсийге чейинки температура эсептелет. Өзгөрмөдө температура сакталат. Бир эле шарттуу оператор жетишсиз, анткени мында үч мүмкүн болгон жыйынтык бар. Маселенин чыгарылышын төмөндөгүдөй жазса болот:

```
t = int (input ('Температураны киргизгиле'))
if t > 250:
      print (''Мештеги температура өтө жогору')
else:
      if 200 <= t <= 250:
            print (''Мештеги температура нормада')
      else:
            print ('Мештеги температура нормадан төмөн')
```

Барабардыкты текшерген `if` шарттуу оператору `else (антпесе)` блогунун ичинде жайгашкан, ошондуктан ал камтылган шарттуу оператор деп аталат. Бул мисалдан көрүнүп тургандай, аны колдонуу бир нече варианттан бирөөнү тандап алууга мүмкүндүк берет. Эгерде `else` операторунан кийин эле дагы бир `if` кетсе, `elif` (`else-if` тин кыскартылганы) сөзү менен «каскаддык» тармакты колдонсо болот. Мисалы: берилген x жана y координаталары боюнча координаттык тегиздиктин чейректерин аныктоо керек. **x** жана **y** өзгөрмөлөрүндө клавиатурадан киргизилген бүтүн сандык маанилер сакталат.

```
x = int(input())
y = int(input())
if x > 0 and y > 0:
```

```
        print('Биринчи чейрек')
    elif x > 0 and y < 0:
        print('Төртүнчү чейрек')
    elif y > 0:
        print('Экинчи чейрек')
    else:
        print('Үчүнчү чейрек')
```

Келтирилген программада `if, ..., elif` шарттары кезеги менен текшерилет жана биринчи чыныгы шартка дал келген блок аткарылат. Эгерде бардык текшерилип жаткан шарттар жалган болсо, анда else блогу аткарылат, эгерде ал бар болсо.

Камтылган циклдер

Татаал маселелерде көпчүлүк учурда циклдин ар бир кадамында өзү да циклдик алгоритмди түзгөн түрдүү берилиштерди иштетүүнү аткарууга туура келет. Бул учурда «циклдин ичиндеги цикл» же «камтылган цикл» конструкциясы алынат.

Цикл *камтылган* деп аталат, эгер ал башка циклдин ичинде жайгашса. Биринчи жүрүштө сырткы цикл ички циклди чакырат. Ал өзүнүн акырына чейин аткарылгандан кийин башкаруу кайрадан сырткы циклдин тулкусуна берилет. Экинчи жүрүштө сырткы цикл кайрадан ички циклди чакырат, жана ушинтип бул процесс сырткы цикл аяктамайынча кайталана берет.

1-маселе. Экранга көбөйтүүнүн таблицасын чыгарабыз. Ал үчүн сырткы циклде 1ден 9га чейинки сандарды терип чыгуу керек. Ал сандардын ар бирине ички циклде 1ден 9га чейинки сандарды терип чыгуу керек.

```
1 2 3 4  5  6  7  8  9
2 4 6 8 10 12 14 16 18
```

Бир көк цифрага 9га чейинки цифралардын бир катары туура келет. 1 жана 2 көк цифралар сырткы циклде, ал эми кара цифралар ички циклде жайгашкан.

Мындагы ички циклде биринчи катардагы сырткы жана ички циклдердин өзгөрмө-эсептегичтерин көбөйтүш керек.

Ушундай жол менен сырткы циклдин бир аткаруусуна ички циклдин 9 аткаруусу туура келет жана көбөйтүүнүн таблицасынын бир сабы түзүлөт. Ар бир саптан кийин жаңысына өтүү керек: бул ички цикл аткарылып бүткөндөн кийин, сырткы циклде жүргүзүлөт.

Андан тышкары таблицаны тургузуу үчүн форматталган киргизүүнү колдонуш керек, б.а. мамычалардын туурасын (\t) берүү керек, антпесе сандар жылышып калат, анткени ар бир саптагы цифралардын саны ар башка.

Биздин код мындай болуп көрүнөт:

```
for i in range(1,10):#1ден 10го чейинки биринчи көбөйтүүчү
    for j in range(1,10):#1ден 10го чейинки экинчи көбөйтүүчү
        print(i*j, end='\t')
    print()
```

Жыйынтык:

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

2-маселе. 2ден 100гө чейинки интервалдагы бардык жөнөкөй сандарды табуучу программаны түзөлү.

Жөнөкөй сан – бул 1 санына жана өзүнө гана калдыксыз бөлүнгөн сандар. Мисалы 5 – жөнөкөй сан, анткени ал 1ге жана 5ке гана калдыксыз бөлүнөт, ал эми 6 – курама сан, анткени 6га жана 1ден тышкары ал 2ге жана 3кө да калдыксыз бөлүнөт.

- Бир жөнөкөй сан болуп эсептелбейт, ошондуктан биздин мисалда диапазон 2 цифрасынан башталат.
- Шарт боюнча, эгерде n саны 2ден n ге чейинки диапазондо бөлүүчүгө ээ эмес болсо, анда ал жөнөкөй. Ал эми бул интервалда жок дегенде бир бөлүүчү чыгып калса, анда сан курама.
- n санынын кандайдыр бир `k` санына бөлүнүүчүлүгүн текшерели, эгер калдык нөлгө барабар болсо, анда `n` саны `k` санына бөлүнөт.
- Эгер жок дегенде бир бөлүүчүсү табылса, анда сан курама жана бул маселеде андан аркы бөлүүчүлөрдү издөөнүн кажети жок. Ал үчүн n % k==0 болсо, анда `break` **операторунун жардамында циклден тез арада чыгуу** аткарылат.
- flag өзгөрмөсү сандын жок дегенде бир өзгөрмөсү бар же жок экенин көрсөтөт.

Ошентип программабызды мындай кылып жазсак болот (мында `n`, `k` — бүтүн сандык өзгөрмөлөр):

```
for n in range(2,101):
    flag = False
    for k in range(2,n):
        if n % k == 0:
            flag = True
            break
    if not flag:
        print (n)
```

ЭСИҢЕ ТУТ

Цикл n жолу кайталанышы үчүн диапазондун акыркы саны n+1 болуш керек.

3-маселе. Экранда эки түрдөгү символдордун жардамы менен «тик бурчтук» тарталы. Тик бурчтуктун четтери «0» символу менен, ал эми анын ичи «1» символу менен тартылсын.

Мейли тик бурчтуктун узундугу 10 символго жана туурасы 7 символго барабар болсун. Сырткы цикл саптарды терип жатып, биринчи жана акыркы цифраларга 0дү коюш керек. Эгерде сап биринчи же акыркысы болсо (эсептөө 0дөн башталгандыктан 0- жана 6-саптар), 0 башынан аягына чейин тизилип чыгат. Калган бардык башка учурларда 1 цифрасын коёбуз. Программаны жазалы:

```
for i in range(7):              #сапты 7 жолу чыгарабыз
    if i==0 or i==6:            #эгерде сап 1-чи же акыркы болсо
        for j in range(20):     #бардык 20 жолу
            print('0',end='')       #0ду чыгарабыз
    else:                           #антпесе
        print('0',end='')           #0ду чыгарабыз
        for j in range(1,19):       #1 жана 19-дан башкасына
            print('1',end='')       #1 цифрасы менен чыгарабыз
        print('0',end='')
    print()
```

СУРООЛОР ЖАНА ТАПШЫРМАЛАР:

1) *Айдын номерин алып, ошого тиешелүү жыл мезгилин чыгаруучу же ката жөнүндө маалымат берүүчү программаны жазгыла.*
2) *5 кг, 10 кг жана 15 кг алма бата турган ящиктер берилди. 100 кг алманы бөлүштүрүү үчүн канча ар кандай өлчөмдөгү ящик керек боло тургандыгын аныктоочу программаны түзүү керек.*

9-тема:

Функциялар

Мурда силер интерпретатордун өзүндө **орнотулган** функцияларды колдонуп келгенсиңер, мисалы:

`print()` – *экранга тегерек кашаанын ичиндегилердин бардыгын чыгарат;*
`str()` – *берилиштерди саптык типке өзгөртүп түзөт;*
`int()` – *берилиштерди бүтүн санга өзгөртүп түзөт;*
`float()` – *бүтүн сандарды бөлчөк типке өзгөртүп түзөт;*
`round()` – *санды модулу боюнча чоң жагына тегеректейт.*

Булардан башка биз тигил же бул маселелерди аткартуу үчүн өзүбүздүн функцияларды түзүп алсак болот. Бул үчүн Python тилинде эгер кайсы бир алгоритм (же фрагменти) кайталанып жатса, аны функция катары формага келтирсе боло турган мүмкүнчүлүгү каралган.

Ал үчүн жаңы функцияга ат берип жана анын алгоритмин баяндоо керек. Мындан кийин программада функциянын атын жазганда эле өзүнүн кирген жана чыккан берилиштери менен тиешелүү алгоритм ишке кирет. Функцияны аткаргандан кийин программанын иши функцияны чакырган командадан кийин кайра улана берет.

Мисалга, программанын бир нече жеринде экранга «Программада ката» деген билдирүүнү чыгарыш керек болуп жатат. Аны мындайча жасасак болот:

```
print ('Программада ката')
```

Бул чыгаруу операторун керек болгон жердин бардыгында коё берсек, анда бул эсти толтуруп жибериши мүмкүн. Эгерде билдирүүнүн текстин өзгөртүш керек болуп калса, анда бул чыгаруу операторлорун бүткүл программанын ичинен издеш керек болот. Мына так ушундай учурлар үчүн кошумча алгоритм – функциялар колдонулат. Аларга программанын каалаган жеринен кайрыла берсе болот. `error` функциясын жазалы:

```
def error():
    print ('Программада ката')
n = int (input())
if n < 0:
    error()
```

Биз `error` деген жаңы функцияны киргиздик.

Фукциянын аты `def` (англ. *define* — аныктоо) ачкыч сөзү менен башталат, андан кийин гана функциянын уникалдуу аталышы берилет (мисалы: `def sum`). Аталышынан кийин функциянын параметрлери киргизилген кашаалар жана кош чекит коюлат. Функциянын тулкусу жылдыруу менен жазылат. Функцияны программанын башка жеринде иштетиш үчүн, анын аталышы менен (кашааларын кошуп) чакырыш керек. Мисалы: **error()**.

Эгерде кандайдыр бир аракеттер программанын ар кайсы жерлеринде бир нече жолу кайталанса, анда функцияны колдонуу кодду бир топ кыскартууга мүмкүндүк берет. Кээде өтө чоң программаларды жөнөкөйлөтүп жана ыңгайлуу кылуу үчүн бир нече функцияларга бөлүп алышат. Анын татаал алгоритмдеринин өзүнчө этаптарын функция түрүндө көрсөтөт. Мындай ыкма бардык программаны түшүнүктүү кылат.

Функция жана алардын аргументтери

Функцияларга аргументтерди–аткарылуучу аракеттерди өзгөртүү үчүн кошумча берилиштерди берсе болот.

Мисалы таблица же бөлүүчү сызыкты тартуу үчүн экранга бир символду көп жолу чыгарыш керек дейли. n өзгөрмөсү үчүн бул маселени чечүүчү программаны мындай жазсак болот:

```
n = 125 #ушунча жолу
s = '_' #символ
while n > 0:
      print (s, end = '')
      n -= 1;
```

Аяктоочу символ (адатта “жаңы сап” символу) – **end** аталыштагы аргументи менен `print` функциясынын чакырылышына көңүл буралы.

ЭСИҢЕ ТУТ

Функциянын аты кичинекей латын тамгаларынан туруш керек, ал эми сөздөр бири-биринен төмөнкү сызык символу менен ажыратылышы керек. Бул кодду окуу үчүн ыңгайлуу кылат (snake case).

Кайталануучу символдун чыгаруу циклинин кошумча алгоритмин функция түрүндө жазсак болот. Бул функцияга *аргументин* бериш керек – символ жана аны канча жолу кайталаш керектигин көрсөткөн сан. Анда:

```
def print_char(s, n): #аргументи менен функциянын аты
      k = n
      while k > 0:
            print (s, end = '')
            k -= 1
print_char ('-', 10)  #аргументтер
```

Негизги программа `print_char` функциясын чакыруунун болгону бир командасын гана камтыйт. Кашаанын ичинде тире («-») символун 10 жолу чыгарыш керектигин көрсөтүүчү функциянын аргументи көрсөтүлгөн.

Глобалдык жана локалдык өзгөрмөлөр

Көпчүлүк учурларда функцияларды берилиштерди иштетүү үчүн колдонушат. Бул берилиштер глобалдык же локалдык болушу мүмкүн. **Локалдык өзгөрмөлөр** функцияга анын атынан кийин тегерек кашаанын ичинде көрсөтүлгөн аргументтер аркылуу берилет. Локалдык өзгөрмөлөр ошол функциянын гана «көрүнүү зонасында» жайгашат жана программанын калган бөлүгүнө жеткиликтүү эмес. Ал эми **глобалдык өзгөрмөлөр** программанын бардыгында жеткиликтүү. Аларга аты боюнча кайрылса болот жана аны менен байланышкан маанилерди алса болот.

1-маселе. Программанын негизинде өзгөрмөлөрдүн типтерин карайлы:

```
def rectangle():
      a = float(input('Туурасы: '))
      b = float(input('Бийиктиги: '))
      s = a*b
      print('Аянты: ', s)
def triangle():
      a = float(input('Негизи: '))
      h = float(input('Бийиктиги: '))
      s = 0.5*a*h
      print('Аянты: ', s)
      figure = input('1-тик бурчтук, 2-тик бурчтук:')
if figure == '1':
      rectangle()
elif figure == '2':
      triangle()
```

Бул маселеде 5 өзгөрмө бар, анын ичинде `figure` гана глобалдуу `rectangle()` функциясындагы `a` жана `b` жана `triangle()` функциясындагы `a` жана `h` өзгөрмөлөрү – локалдык. Ошону менен бирге ар кайсы функциядагы локалдык өзгөрмөлөр – ар башка өзгөрмөлөр.

Функциядан маанилерди кайтаруу

Ар бир функция белгилүү бир жыйынтыкты берет. А түгүл силер жыйынтык катары маанисин кайтарууну көрсөтпөсөңөр да, ал баары бир None (эч нерсе) деген жыйынтыкты берет.

Функциянын мааниси эмнеге барабар экендигин көрсөтүү үчүн артынан маани-жыйынтыгы жазылган `return` (*англ. кайтаруу*) операторун колдонушат.

Жыйынтыгы сан, символ, символдук сап же каалагандай башка объект болушу мүмкүн.

2-маселе. Сандын цифраларын кошууну эсептөөчү функцияны түзөлү (мисалы, 147 саны үчүн сандарды кошуу керек: 1+4+7=12). Цифраларды кошууну акыркысынан баштайлы, биздин мисалда бул 7 цифрасы.

1. Сандын акыркы цифрасын алуу үчүн, санды 10го бөлгөндөгү калдыгын алуу керек (147 % 10 = 7).
2. Алынган калдыкты баштапкы мааниси нөлгө барабар болгон «суммага» (sum = 0) кошобуз. Эми сумма 7ге барабар.
3. Андан соң биз бүтүн сандык бөлүү операторун колдонуп, сандын акыркы цифрасын «бөлүп салабыз» (147 // 10 = 14).
4. 14 > 0 болгондуктан биз циклдин башына кайрылабыз. Цикл n маниси нөлгө барабар болгонго чейин уланат.
5. Мындан кийин кайрадан 4 цифрасын бөлүп салабыз (14 % 10 = 4) жана аны суммага кошобуз (4 +7 = 11).
6. Аны бардык сандан бөлүп алабыз (14 // 10 = 1).
7. Акыркы бир орундуу санды суммага кошобуз (11 + 1 = 12).

Жыйынтыгы: 12

Ушундай жол менен биздин программа төмөнкүдөй жазылат:

```
n = int(input('Санды киргизиңиз: '))
def digits_sum (n):
    total = 0
    while n > 0:
        total += n%10
        n = n // 10
    return total
#негизги программа
print (digits_sum(n))
```

3-маселе. Берилген сандын цифраларынын суммасы 3кө бөлүнөөрүн же бөлүнбөсүн аныктоочу программаны түзөлү. Ошондой эле эгер алынган сумманын саны бир цифрадан көп болсо, анда сумма бир цифрадан турганча аракетти кайталатуу керек.

Мисалы, 123456789 санынын цифраларынын суммасы 45ке барабар. Сан эки орундуу, демек дагы цифралардын суммасын табабыз: 4 + 5 = 9. 3кө бөлүнгөн 9 санын алдык (9%3==0). Демек баштапкы 123456789 саны 3кө бөлүнөт. Бул алгоритмден төмөнкү программага келебиз:

```
def sum(n):
    sum = 0
    while n>0:
        sum += n % 10
        n = n // 10
    return sum
#негизги программа
k = int(input('Санды киргизиңиз: '))
while k > 9: #цифралардын суммасы бир цифра болгонго чейин
    k = sum(k) #функцияны чакырабыз
if k%3==0:
    print ('Сан 3кө бөлүнөт')
else:
    print ('Сан 3кө бөлүнбөйт')
```

Функцияларды бир гана негизги программадан эле эмес башка функциялардан да чакырса болот. Мисалы, үч ар түрдүү сандардын ортосундагысын (б.а. эки сандын ортосунда жайгашкан санды) табуучу функция мындай аныкталышы мүмкүн:

```
def middle ( a, b, c ):
    mi = min ( a, b, c )
    ma = max ( a, b, c )
    return a + b + c - mi - ma
```

Программа **min** жана **max** даяр функцияларын колдонду. Муну чыгаруунун идеясы, эгер үч сандан максималдык жана минималдык санды кемитип салса, так ошол үчүнчү сан калат дегенде.

Функция бир нече маанини кайтарышы да мүмкүн. Мисалы эки санды бөлүүдөн алынган тийиндини да, калдыкты да чыгаруучу программа төмөнкүдөй жазылат:

```
def divmod ( x, y ):
    d = x // y
    m = x % y
    return d, m
```

Функциянын жыйынтыгын эки ар түрдүү өзгөрмөлөргө жазса болот:

```
a, b = divmod ( 7, 3 )
print ( a, b ) #2 1
```

Эгерде бир эле өзгөрмөнү көрсөтсөк, биз кортежди алабыз – тегерек кашаага алынган элементтердин катары:

```
q = divmod ( 7, 3 )
print ( q ) #(2, 1)
```

Кээде программада бир эле жолу колдонулуп жана бир нече аргументи менен анча татаал эмес аракеттерди аткарган функцияны түзүү үчүн **lambda-функцияларды** колдонушат. lambda-функция бул анонимдик функция, б.а. **def** сыяктуу өзүнүн атына ээ эмес. Функциянын жазылышы **lambda** сөзүнөн башталат жана бош орундан кийин функциянын аргументтери көрсөтүлөт. Андан соң кош чекиттен кийин жыйынтыгы функцияда кайтарылган амалдар көрсөтүлөт. Мындай функцияны эки санды көбөйтүү мисалында карайлы:

```
multiple = lambda x, y: x * y #2 аргументи менен lambda-функция
print (multiple (2, 5)) #жыйынтык 10
```

Жогорку мисалда биз lambda-функцияны **multiple** өзгөрмөсүнө ыйгардык. Бирок мындай функцияны бир сап менен эле өзгөрмөнү колдонбостон да жазсак болот:

```
print ((lambda x, y: x * y)(2, 6))
```

Программалоодогу негизги көндүмдүрдүн бири – бул бир эле кодду бир нече жолу жазбоо болуп саналат. Качан код кайталана баштаса, бул кайталанган коддун бөлүгүн функцияга айлантуу керектигин түшүнүш керек.

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *3 берилген санды өсүү тартибинде экранга чыгаруучу функцияны жазгыла.*
2) *Эки натуралдык сандын эң чоң жалпы бөлүүчүсүн табуучу фукцияны жазгыла.*
3) *Сандардын төмөнкүдөй белгилерин чыгаруучу функцияны жазгыла: кайтарылган жыйынтыгы менен аргументи бүтүн сан болуш керек, 0 – эгерде аргумент 0 болсо, -1 эгер сан терс болсо, 1 – эгер сан оң болсо.*

10-тема:

Массивдер

Азыркы компьютерлердин эң негизги кызматы – бул көп көлөмдөгү маалыматты иштетүү. Ошону менен бирге маалымат сакталган миңдеген (же миллиондогон) уячанын ар бирине кайрылып туруш керек. Мындай учурларда ат бир уячага эмес, ар бир уяча өзүнүн номерине ээ болгон уячалардын тобуна берилет. Эстин мындай аймагы массив деп аталат.

Массив – бул жалпы атка ээ болгон эсте жакын жайгашышкан (кошуна уячаларда) өзгөрмөлөрдүн тобу. Массивдеги ар бир уяча уникалдуу номерге (индекске) ээ.

Python тилинде бир өлчөмдүү массивдер элементтердин тизмеси түрүндө болот. Ошондуктан массивдер менен иштөө үчүн тизмелерди колдонушат (берилиштер тиби list). Python тилинде тизме – бул ар бири өзүнүн номерине (индекс) ээ болгон элементтердин жыйындысы. Номерлөө ар дайым нөлдөн башталат, катары боюнча экинчи элемент 1 номерине ээ ж.б.

Тизмени түрдүү жолдор менен түзсө болот. Анын эң жөнөкөй ыкмасы — элементтердин тизмесин үтүр менен ажыратып, чарчы кашаанын ичинде жазуу:

```
a = [1, 3, 4, 23, 5]
```

Тизме – бул динамикалык структура, анын өлчөмүн программанын иштөө убагында өзгөртсө (элементтерди өчүрсө же кошсо) болот.

Тизмелерди «+» белгисинин жардамында «кошсо» болот. Ошентип жогорку мисалды мындай жазса да болот:

```
a = [1, 3] + [4, 23] + [5]
```

Бирдей тизмелерди кошуу көбөйтүү менен алмаштырылат «*». Нөлдөр менен толтурулган 10 элементтен турган тизме мындай түзүлөт:

```
a = [0]*10
```

Мындан татаал учурларда тизмелердин генераторлору колдонулат. Анда жаңы түзүлгөн тизменин элементтери циклди колдонуу менен толтурулат:

```
a = [i for i in range(10)]
```

Силер билгендей `for i in range(10)` цикли 0дөн 9га чейинки i нин бардык маанилерин коюп чыгат. `for` сөзүнүн алдындагы туюнтма (биздин учурда – `i`) – бул ар бир `i` үчүн тизменин улам кийинки элементине жазыла турган өзгөрмө. Берилген мисалда тизме `i` өзгөрмөсү удаалаш алып жаткан маанилер менен толтурулат:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ушуну эле, `range` функциясынын жардамы менен алынган берилиштер менен тизме түзүү үчүн `list` функциясын колдонуп алсак болот:

```
a = list (range(10))
```

Тизмени бул сандардын квадраттары менен толтуруу үчүн мындай генераторду колдонсо болот:

```
a = [i*i for i in range(10)] #[0,1,4,9,16,25,36,49,64,81]
```

Генератордун жазуусунун аягында тандап алуу шартын кошуп койсо болот. Бул учурда тизмеге ушул шартты канааттандырган циклде тандалып алынган элементтер гана камтылат.

Мисалы, төмөнкү генератор 0дөн 9га чейинки диапазондогу бардык жуп сандардын тизмесин түзөт:

```
a = [i for i in range(10) if i%2 ==0] #[0,2,4,6,8]
```

Көпчүлүк учурда тесттик жана окуу программаларында массивди кокустук сандар менен толтурушат. Муну генератордун жардамында кылса болот:

```
from random import randint
a = [randint(20,100) for x in range(10)]
```

Мында 10 элементтен турган массив түзүлөт жана [20, 100] диапазонундагы кокустук сандар менен толтурулат. Ал үчүн **`random`** модулунан импорттолгон **`randint`** функциясы колдонулат.

Тизменин узундугу (андагы элементтердин саны) `len` функциясы менен аныкталат. Мисалга:

```
a = [1, 3, 4, 23, 5]
n = len(a) #5
```

Мындан ары бардык мисалдарда биз **n** элементтен (бүтүн сандардан) турган **a** тизмеси массив түрүндө түзүлдү деп эсептейбиз. **i** өзгөрмөсү тизменин элементинин индексин түшүндүрөт.

Циклди пайдалануу менен 3 элементтен турган тизмени түзүү үчүн төмөнкү программаны жазабыз:

Python тилиндеги программа

```
n = 3
a = [0]*3
for i in range(n):
  print('a[', i, ']=', sep='', end="")
  a[i] = int(input())
```

Экранга чыккан жыйынтыгы

```
a[0] = 1
a[1] = 2
a[2] = 3
```

n элементинен турган массив түзүп, анын маанилерин киргизүүнү тизме генераторунун жардамында жүргүзсө да болот.

```
a = [int(input()) for i in range(n)]
```

Бул жерде колдонуучу **int** функциясынын жардамында бүтүн санга өзгөртүлүп түзүлгөн n сандагы элементтерди киргизиш керек жана бул сан массивге кошулат.

Киргизүүнүн дагы бир варианты бар, мында бардык массив бир эле сапта киргизилген. Бул учурда **input** функциясынан алынган сапты **split** ыкмасынын жардамы менен бөлүктөргө ажыратуу керек:

```
data = input()
s = data.split()
print (s)
```

Тизмени чыгаруу үчүн **print** функциясын колдонобуз. Тизменин ар бир элементин чыгаруу үчүн анын индексин чарчы кашаада жана бирден элементти сапта көрсөтүү менен төмөнкү программаны пайдаланабыз:

Python тилиндеги программа

```
a = [1, 2, 3, 4, 5];
for i in range(0, len(a)):
  print('a[', i, ']=', a[i], sep='')
```

Экранга чыккан жыйынтыгы

```
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
```

Элементтерди иргөө

Тизменин элементтерин иргөө, биз тизменин бардык элементтерин циклде карап жана керек болсо анын ар бири менен кандайдыр бир аракеттерди жасайбыз дегенди түшүндүрөт. Циклдин өзгөрмөсү 0дөн n—1ге чейин өзгөрөт, мында n – тизменин элементтеринин саны:

```
for i in range(0,n):
      a[i] + = 1
```

Бул мисалда **a** тизмесинин бардык элементтери 1ге көбөйөт.

Эгерде тизмени өзгөртүүнүн кереги жок болсо, анда анын элементтерин иргөө үчүн мындай циклди колдонуу ыңгайлуу:

```
a = [1,2,3,4,5]
for x in a:
      print(x)
```

Мында `print(x)` тин ордуна «**x**» өзгөрмөсүнө жазылган элементи менен иштеген каалагандай башка операторлорду колдонсо болот. Көңүл бурсаңар, циклдин тулкусундагы «**x**» өзгөрмөсүнүн өзгөрүшү катаны берет.

Көптөгөн маселелерде берилген шартты канааттандырган тизмедеги бардык элементтерди табуу жана аларды иштетүү талап кылынат. Мындай маселелердин эң жөнөкөйү – керектүү элементтерди эсептөө болуп саналат. Бул маселени чечүү үчүн баштапкы мааниси нөл болгон өзгөрмө-эсептегичти киргизүү керек. Андан соң циклде тизменин бардык элементтерин карайбыз. Эгерде каралып жаткан элементте берилген шарт аткарылса, анда эсептегичти бирге көбөйтөбүз.

a массивинде класстагы балдардын бою жөнүндө маалымат жазылган дейли. Бойлору 120 см ден чоң бирок 150 см ден кичине болгондордун санын аныктайлы. Төмөнкү программада count өзгөрмө-эсептегичи колдонулду:

```
count = 0
for x in a:
      if (120 < x < 150):
      count + = 1
```

Эми маселе кичине татаалдансын: балдардын орточо боюн табуу керек. Ал үчүн кошумча өзүнчө бир өзгөрмөдө бардык керектүү маанилерди кошуу, ал эми циклдин аягында бул сумманы бардык маанилердин санына бөлүү керек. Сумма жыйналган sum өзгөрмөсүнүн баштапкы мааниси да нөлгө барабар болуш керек:

```
count = 0
sum = 0
for x in a:
      if 120 < x < 150:
            count + = 1
            sum + = x
print (sum / count)
```

Тизменин элементтерин суммалоо – бул көп таралган амал, ошондуктан Python тилинде элементтерди суммалоо үчүн атайын sum деген орнотулган функция бар:

```
a = [1,2,3,4,5]
print (sum(a))
```

Анын жардамында мурунку маселени бир аз тыканыраак чечсек болот: биринчиден кошумча массивге бардык керектүү элементтерди бөлүп алуу, андан кийин алардын суммасын жалпы санына (тизменин узундугуна) бөлүп коюу керек.

Жаңы тизмени түзүү үчүн шарттуу операторду колдонобуз:

```
b = [x for x in a if 120 < x < 150]
print (sum(a)/len(a))
```

Тандоо шарты **a** тизмесинен **b** тизмесине шартты канааттандырган гана элементтерди берди. Ал эми класстагы балдардын бойлорунун орточосун чыгаруу үчүн жаңы тизменин элементтеринин суммасын алардын санына бөлүп коюу жетиштүү.

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *Тизмеде окуучулардын баалары сакталган. Клавиатурадан киргизген бааларга барабар болгон тизменин элементтеринин номерлерин чыгаруучу программаны түзгүлө.*
2) *Дене тарбия сабагында окуучулардын бойлорун тизмеге жазышты. Бул тизмеден эң бийик жана эң жапыз бойлуу окуучуну табуучу программа түзгүлө.*

11-тема:

Саптар жана алар менен болгон амалдар

Адегенде компьютерлер эсептөөчү машиналар катары колдонулган, азыр болсо алардын негизги кызматы тексттик маалыматтарды иштетүү болуп бара жатат. Python тилинде текст менен иштөөчү негизги тип – бул саптар (**str** тиби) англ. ***string***.

Сап – бул бир же кош тырмакчанын ичине алынган символдордун удаалаштыгы: `'Бул сап'` = `"Бул сап"`

Тизмелерден (массивдерден) айырмаланып, саптар берилиштердин структурасына кирбейт. Ошону менен бирге саптарды иреттелген элементтердин удаалаштыгы катары карап, алар менен тизменин элменттериндей эле иштесе болот.

```
>>> s = 'Бул сап'
>>> s[0] #көрсөтүлгөн  индекси менен элементтерди кайтарат
'Б'
>>> s[5:] #5-индекстен акыркысына чейинки элементтер
'ап'
```

Бирок Python тилинде саптар өзгөртүлбөйт. Башкача айтканда берилген саптын кайсы бир бөлүнгөн элементин башкасына алмаштырууга болбойт. Мындай учурда программа ката деп чыгарат. А бирок берилген саптын символдорунан керектүү өзгөртүүлөрдү киргизип **жаңы сапты** түзсө болот.

Сапты клавиатурадан киргизип, андагы бардык «а» тамгаларын «б» тамгаларына алмаштырып, экранга чыгаруунун толук программасын көрөлү:

```
s = input('Сапты киргизиңиз: ')
s1 = ''
for c in s:  #s сабындагы бардык символдорду иргеп чыгат
    if c == 'а': #өзгөрмөнүн мааниси «а» га дал келсе
        c = 'б' #анда аны «б» тамгасына алмаштырабыз
    s1 = s1 + c
print (s1)
```

Бирок бул ыкма өтө жай иштейт. Символдорду алмаштырыш керек болгон практикалык маселеде эң жакшысы даяр **replace** методун колдонуу керек.

Мындан тышкары көпчүлүк учурда тексттин сабын иштетүү керек болот: бул саптын бөлүгүн (сапча) алуу, эки сапты бир сапка бириктирүү, же саптын бөлүгүн өчүрүү. Мисалы, саптарды **бириктирүү** (чиркештирүү) үчүн «+» оператору колдонулат. Бул амал конкатенация деп аталат:

```
s1 = 'Кутман'
s2 = 'күн'
s = s1 + ' ' + s2 + '!'
print (s)
>>>
Кутман күн!
```

ЭСИҢЕ ТУТ

Саптын узундугу саптын касиети len дин (length) жардамы менен аныкталат. Ал үчүн n өзгөрмөсүнө бош орундары менен кошуп саптагы белгилердин санын (бүтүн сан) берүүчү s сабы жазылат:

n = len(s)

Саптарды иштетүү үчүн үзүмдөрдү колдонуу

Python тилинде көп учурда саптын аныкталган бөлүгүн иштетүү үчүн белгилеп алган үзүмдөр (англ. slicing) колдонулат. Үзүмдөр мындайча жазылат: [X:Y]. X – үзүмдүн башталышы, Y – үзүмдүн аягы. Y индексиндеги символ үзүмгө кирбейт. Алгач көрсөтүлбөсө, биринчи индекс 0гө, ал эми экинчиси — саптын узундугуна барабар.

Мисалы, s[3:8], s сабындагы 3-символдон 7-символго чейин (б.а. 8-ге чейин бирок 8 өзү кирбейт) үзүмдү түшүндүрөт.

Мааниси	s = 'китепканачы' сапчасы үчүн мисалдар

Саптын бөлүгүн (сапча) белгилеп алуу үчүн	`s1 = s[2:8]`
	`#s1 сабына 3-элементтен баштап 8-элементи кошулган маани жазылат`
	`>>> тепкан`

Саптын бөлүгүн өчүрүү үчүн	`s1 = s[:3] + s[9:]`
	`#башынан баштап 3-элементке чейин, жана 9-элементтен аягына чейин кесип алабыз жана аларды s1 сабында сактайбыз`
	`>>> китачы`

Саптын ичине жаңы фрагмент коюу	`s1 = s[:3] + 'ABC' + s[3:]`
	`#3-элементтен кийин ABCны кошуп кетебиз`
	`>>> китABCепканачы`

Сапты реверстөө (аны тескерисинче буруу)	`s1 = s[::-1] #сөзду тескери жазып чыгарат` `>>> ычанакпетик`

Берилген кадам аркылуу элементтерди тандоо	`s1 = s[::2]`
	`#ар бир экинчи символду кайтарат`
	`>>> ктпааы`

Саптар методу

Python тилинде саптар менен иштөө үчүн көптөгөн камтылган методдор бар. Алардын ичинен кызыктууларын карап көрөлү.

1 **upper** жана **lower** методдору сапты тиешелүү түрдө жогорку жана төмөнкү регистрлерге өткөрөт. **title** методу болсо биринчи эле тамгаларды жогорку регистрге, калганын төмөнкү регистрге өткөрөт:

```
s = 'aAbB cC'
s1 = s.upper() #'AABB CC'
s2 = s.lower() #'aabb cc'
s2 = s.title() #'Aabb Cc'
```

2 **split** методу сапты бош орундар боюнча бөлүүгө мүмкүндүк берет. Жыйынтыгында сөздөрдөн тизме алынат. Эгерде колдонуучу программада ар бири өзүнчө тизмедегидей иштетилсин деп, бир сапта катар сөздөрдү же сандарды киргизүү керек болсо, анда split методун колдонсо болот:

```
s = 'Дүйшөмбү Шейшемби Шаршемби Бейшемби Жума'
s1 = s.split () #['Дүйшөмбү','Шейшемби','Шаршемби','Бейшем-
би','Жума']
```

3 **join** методу тескерисинче тизмеден сапты курайт. Ал үчүн алдына сап-бөлгүч коюлат, ал эми кашаанын ичинде тизме берилет:

```
s = ['Дүйшөмбү', 'Шейшемби', 'Шаршемби', 'Бейшемби','Жума']
s1 = '-'.join (s) #Дүйшөмбү-Шейшемби-Шаршемби-Бейшемби-Жума
```

4 **find** методу саптын бөлүгү (сапча) менен иштейт. Ал саптагы сапчаны издейт жана табылган сапчанын биринчи элементинин индексин кайтарат. Эгерде сапча табылбаса анда 1ди кайтарат.

```
s = 'Дүйшөмбү Шейшемби Шаршемби Бейшемби Жума'
s1 = s.find ('Шаршемби') #жообу: 18, сапчанын 1-элементи -
«Ш» тамгасынын индекси
```

Мындан тышкары бул метод менен берилген үзүмдөгү саптын элементинин индексин табууга болот. Үзүмдү көрсөтүү үчүн, анын башталышын жана аягын үтүр менен ажыратылган цифралар менен берүү керек. Эгерде экинчи цифра көрсөтүлбөсө, анда издөө саптын аягына чейин жүргүзүлөт:

```
s2 = s.find ('ү', 4) #8, 4-индекстен аягына чейинки бөлүгүн-
дөгү биринчи «ү» нүн индекси.
```

```
s1 = s.find ('ү', 0, 9) #2, башынан 9-индекске чейинки бөлүгүндөгү биринчи «ү» нүн индекси
```

5 **replace** методу бир сапчаны башкасына алмаштырат. Бул учурда баштапкы сап өзгөрбөйт, болгону жаңы сапка модификацияланат (өзгөртүп түзүлөт). Ал болсо жаңы s1 өзгөрмөсүнө ыйгарылат:

```
s = 'Дүйшөмбү Шейшемби Шаршемби Бейшемби Жума'
s1 = s.replace ('б', 'В') #ДүйшөмВү–ШейшемВи–ШаршемВи–Бейшемви–Жума
```

Кээде бизге сапты бөлүш керек болот. Бөлүнгөн бөлүгү жаңы саптан башталгыдай кылып, бул учурда биз \n белгисин колдонобуз.

```
print ('Дүйшөмбү \n Шейшемби \n Шаршемби \n Бейшемби \n Жума') #бардык сөздөр мамыча түрүндө чыгат.
```

Эгерде жаңы сапты жылдыруу менен чыгаруу керек болсо, анда \t белгисин колдонобуз.

```
print ('Дүйшөмбү \n\t Шейшемби \n\t Шаршемби \n\t Бейшемби \n\t Жума') #бардык сөздөр мамыча түрүндө чыгат, ар бир кийинки саптын алдына орун таштапалат.
```

Жогорудагы үйрөнгөн командаларды колдонуп, сапты иштетүүнүн мисалын карап көрөлү.

1-маселе. Клавиатурадан атын, фамилиясын жана атасынын атын камтыган сап киргизилет, мисалы:

Айтматов Чыңгыз Төрөкулович

Ар бир эки сөз бири-биринен бош орундар менен ажыратылган, саптын башында бош орун жок. Бул сапты иштетүүнүн натыйжасында фамилия жана инициалдарын эле камтыган жаңы сап пайда болуш керек:

Айтматов Ч. Т.

Чыгаруу:

1 Сапты клавиатурадан киргизебиз:

```
s = input('Фамилия, атыңыз жана атаңыздын атын киргизиңиз: ')
```

2 Киргизилген сапты бош орун менен ажыратылган өзүнчө сөздөргө бөлүп чыгабыз. Ал үчүн **split** методун колдонобуз. Бул массивде үч элемент болот: **fio[0]** – фамилиясы, **fio[1]** – аты, **fio[2]** – атасынын аты:

```
fio = s.split()
```

3 Инициалдары менен фамилияны чогултабыз:

```
fioshort = fio[0]+' '+fio[1][0]+'.'+ fio[2][0]+ '.'
```

Толук программа мындай көрүнөт:

```
s = input('Фамилия, атыңыз жана атаңыздын атын киргизиңиз: ')
fio = s.split ()
fioshort = fio[0] + ' ' + fio[1][0] + '.' + fio[2][0] + '.'
print (fioshort)
```

Саптарды салыштыруу жана сорттоо

Биз алфавит боюнча сорттоону сөздү тезирээк табуу үчүн колдонобуз. Эгерде сөздүктөрдө сөздөр алфавит менен жайгашпаганда, анда биз бир сөздү издөө үчүн эле бир топ убакыт коротмокпуз. Python тилинде саптарды сорттоо кандай принцип менен түзүлгөн?

Көрсө, сандар сыяктуу эле тамгалар да өзүнүн салмагына ээ экен. Алфавитте биринчи турган тамга жеңилирээк, б.а. «а» «б» га караганда жеңил, ошондуктан сорттоодо ал биринчи чыгат. Мындан саптарды да сандар сыяктуу эле салыштырууга болот деген жыйынтык келип чыгат.

Сөздөрдү салыштырууда, алгач биринчи тамгалары салыштырылат, эгерде алар айырмаланса, анда салыштыруунун жыйынтыгы аныкталат. Андан кийинки тамгалар салыштырылбай калат. Ал эми эгерде биринчи тамгалары барабар болсо, анда кийинки 2 элементи салыштырылат жана ушул сыяктуу аягына чейин кетет. Мисалы «паровоз» сөзү «пароход» сөзүнө караганда кичине: алар 5-тамгада айырмаланышат жана «в» < «х».

Эгерде силерде текшерүү үчүн символдор бүтүп калса, анда кыска сап узун сапка караганда кичине, ошондуктан «пар» < «парк». Баш тамгалар кичине тамгадан жеңил, анткени сөздүн башында турушат.

Бирок компьютер «алфавиттик тартипти» кайдан «билет»? Көрсө, саптарды салыштырууда символдордун ASCII жана Unicode коддору колдонулат экен. Демек:

«ПАР» < «Пар» < «пар»

«ПАР» жана «Пар» деген сөздөрдү салыштыралы. Биринчи символ эки сөздө тең бирдей, ал эми экинчиси айырмаланат – биринчи сөздө баш тамга, ал эми экинчисинде ошол эле тамга, бирок кичине. Символдор таблицасында баш тамгалар кичине тамгадан биринчи турушат, ошондуктан кичине коддорго ээ. Ошондуктан «А» < «а», «П» < «п» жана «ПАР» < «Пар».

Ал эми башка символдор (цифралар, латын тамгалары) менен кандай болот? Коддук таблицада цифралар ирети менен жайгашкан жана латын тамгаларынан мурун турушат; латын тамгалары – орус тамгаларынан мурун, орус жана латын тамгаларынын баш тамгалары тиешелүү тилдердин кичине тамгаларынан мурун турат. Ошондуктан

«5STEAM» < «STEAM» < «Steam» < «steam» < «ПАР» < «Пар» < «пар»

Мисалы, сөздүн ичиндеги тамгаларды сорттоо үчүн, программаны мындай жазуу керек:

```
s = 'Дүйшөмбү'
s1 = ''.join (sorted (s))
print (s1)
>>> Дбймшүүө
```

1-маселе. Клавиатурадан бир нече сөздөрдү (мисалы фамилияларды) киргизүү жана аларды экранга алфавиттик тартипте чыгаруу керек.

Бул маселени чыгаруу үчүн, үтүр менен ажыратылып киргизилген фамилияларды тизмеге кайра жазып алуу ыңгайлуу, андан соң sorted методу менен сорттоп коюу керек:

```
s = input('Фамилияңызды киргизиңиз: ') #Абакиров Муканова
Бебинов Семенова Запруда
s1 = s.split() #сапчалар менен тизме түзөт ['Абакиров', 'Му-
канова', 'Бебинов', 'Семенова', 'Запруда']
s2 = ' '.join (sorted (s1))
print (s2)
>>> Абакиров Бебинов Запруда Муканова Семенова
```

СУРООЛОР ЖАНА ТАПШЫРМАЛАР:

1) Клавиатурадан бир нече сөз киргизүүнү жана ошол сөздөрдүн ичинен эң кыска сөздүн узундугун аныктоочу программа түзгүлө.
2) isdigit() саптык методу сап жалаң гана цифралардан тураарын текшерип берет. Киргизүүдө программа эки бүтүн санды сураган жана алардын суммасын чыгарып берүүчү программаны түзгүлө. Туура эмес киргизүү учурунда программа ката деп токтоп калбастан, кайрадан эле сандарды сурап тургандай кылыш керек.

12-тема:

Саптарды форматтоо

Python тилинде саптарды форматтоо шаблондун кайсы бир ордуна башка текстти коюу дегенди түшүндүрөт. Ордуна коюу ошол замат жүргүзүлөт. Мисалы, форматтоону колдонуу менен даяр шаблон аркылуу мындай чакыруу билеттерин жасоого болот:

Урматтуу Алина!
Сизди чакырабыз: Ачык эшиктер күнүнө.
Окуянын датасы: 1-май
Урматтоо менен Тимур.

Бир жолу ордуна коюу мындай жазылат:

```
print ('Урматтуу {}!'.format ('Алина'))
```

Башкача айтканда, шаблондун текстинен кийин бош фигуралуу кашаалар коюлат, андан кийин **.format ()** командanын кашаасынын ичине коюу керек болгон маанилери көрсөтүлөт. Программа ишке киргенде, ордуна коюлуучу текст фигуралуу кашаалардын ордуна коюлат.

Бир нече коюулар болсо, фигуралуу кашаага сөздөрдүн индекстери, ал эми сөздөрдүн өзү **.format**тан кийин кортежде коюлат. Ал шаблон мындай жазылат:

```
print ('Урматтуу {0}! \n Сизди чакырабыз: {1}.\n Окуянын
датасы: {2} \n Урматтоо менен {3}.'.format ('Алина', 'Ачык
эшиктер күнүнө', '1-май', 'Тимур'))
```

Текстти **.format**ты колдонбостон башка ыкма менен да форматтаса болот. Бул ыкма бир аз туура эмес жана эскирип калган деп эсептелет. Бирок, эгер силер коддон % белгисин кезиктирсеңер, анда бул жерде форматтоо колдонулду деп эсептесеңер болот. Биздин чакыруу шаблонубузду % операторунун жардамында жазып көрөлү:

```
print ('Урматтуу %s! \n Сизди чакырабыз:%s.\n Окуянын дата-
сы: %d %s \n Урматтоо менен %s.' % ('Алина', 'Ачык эшиктер
күнүнө', '1-май', 'Тимур'))
>>>
Урматтуу Алина!
Сизди чакырабыз: Ачык эшиктер күнүнө.
Окуянын датасы: 1-май
Урматтоо менен Тимур.
```

Силер байкадыңарбы, кээ бир жерде биз %d деп, кээ бир жерде %s деп жаздык? Алар ордуна коюуга эмнени колдонуп жатканыбызды аныктайт:

%s – сапты коёт;
%d – бүтүн санды коёт;
%f – бөлчөк санды коёт.

Санды сапка жана сапты санга өзгөртүп түзүү

Практикалык маселелерде көбүнчө символдордун катары түрүндө жазылган сандарды сандык мааниге жана тескерисинче айландырууга туура келет. Бул үчүн Python тилинде атайын стандарттык функциялар бар:

int – сапты бүтүн санга айландырат

```
s = '123'
N = int ( s ) #N = 123
```

float – сапты чыныгы санга (бөлчөк) айландырат

```
s = '123.456'
X = float ( s ) #X = 123.456
```

str – бүтүн жана бөлчөк сандарды сапка айлантат

```
N = 123
s = str ( N ) #s = '123'
X = 123.456
s = str ( X ) #s = '123.456'
```

Эгерде сапты санга айландыра албай калса (мисалы, сапта тамгалар камтылса), анда ката пайда болот да программа аяктайт.

Бөлчөк сандар үчүн (float тиби), бөлчөк бөлүгүнөн канча белгини чыгарууну көрсөтүп койсок болот, мисалы:

```
X = 34.8589578
print('{:.2f}'.format(X))   #34.86
print('{:.3f}'.format(X))   #34.859
```

Эгерде чоң сандарда үтүрдү биз разряддарды бөлүү үчүн колдонгубуз келсе, анда мындай жазабыз:

```
print('{:,.2f}'.format(10001.23554))   #10,001.24
```

КОМПЬЮТЕРДИК ПРАКТИКУМ:

Колдонуучудан логин жана паролду сураган шаблонду жазгыла. Эгерде логин же пароль туура эмес болуп калса, «мындай логин жана пароль табылган жок» деген билдирүүнү чыгарсын. Ал эми логин жана пароль туура болсо, анда колдонуучунун атын көрсөткөн саламдашууну чыгаргыла.

13-тема:

Python тилинде графика менен иштөө

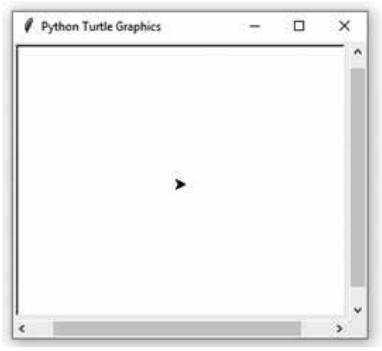
Turtle модулунун жардамында сүрөт тартуу

Python программалоо чөйрөсүндө жөнөкөй сүрөт тартуу үчүн Turtle (ташбака) модулу колдонулат. Ал төмөнкү команда менен кошулат:

```
from turtle import* #ташбака менен иштөөчү командаларды жүктөйт
reset () #позицияны нөлгө айлантат жана калемди күйгүзөт
```

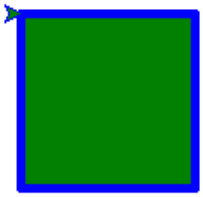
Программаны ишке киргизүү менен силер борборунда калем (ташбака) жайгашкан графика үчүн терезени көрө аласыңар. Мисалы, **forward** командасы ташбаканы алдыга жылдырат. Кашаанын ичинде пиксель менен кадамдардын саны көрсөтүлөт. **right** жана **left** бурулуу командалары үчүн кашаанын ичинде бурууга керек болгон градустардын саны көрсөтүлөт. Эми квадратты тартып көрөлү:

```
from turtle import*
reset ()
forward (100) #алдыга карай 100 пик-
селге кыймылда
right (90)  #90˚ка оңго бурул
forward (100)
right (90)
forward (100)
right (90)
forward (100)
```



Кодду кыскартуу үчүн **for** циклин жана фигурага дагы кошумча параметрлерди берели:

```
from turtle import*
width (5)      #сызыктын жоондугу - 5 пиксель
color('blue','green') #1-түс – сызыктын, 2-түс – боёктун түсү
begin_fill() #аймакты толтуруу үчүн ташбакага байкоо жүргүзүү
for i in range (4):
    forward (100)
    right (90)
end_fill() #begin_fill()ден баштап, аймакты түс менен толтуруу
>>>
```



Up() командасын колдонуп, калемди көтөрүп, башка орундан баштап сүрөттү (же текстти) тартып баштасаңар болот. Башка сүрөттү тартуудан мурун кайра калемди **down()** командасы менен түшүрүү керек.

Айлананы тартуу үчүн **circle(r,n)** командасы колдонулат, мында r – айлананын радиусу, n – биз тарткан айлананын бөлүгү, градус менен. Эгер n = 180 градус болсо, анда калем жарым айлана сызат, n = 360 градус болсо, толук айлана сызат.

Чекитти тартуу үчүн **dot(r, color)** командасы колдонулат, мында r – чекиттин радиусу (пиксель менен), color – чекит тартыла турган түс.

1-маселе. Айлана чиели, анын узундугу боюнча берилген сандагы чекиттер бирдей бөлүштүрүлүп жайгашсын:

```
from turtle import*
def circ(d, r, rBig): #параметрлери менен circ функциясы: чекиттердин саны, чекиттин радиусу, айлананын радиусу
    for i in range(d):
        circle(rBig, 360 / d)#чекиттердин санын айлана боюнча бөлүштүрөбүз
    dot(r, 'red')
up()
goto(150, 0) #калемди 150 пикселге оңго жылдырабыз
setheading(90) #калемди 90° ка бурабыз
down()      #калемди сүрөт тартууну баштоо үчүн түшүрөбүз
circ(15, 10, 150) #параметрлерге маанилерин беребиз
screen.mainloop() #программанын аткарылышын токтотобуз
```

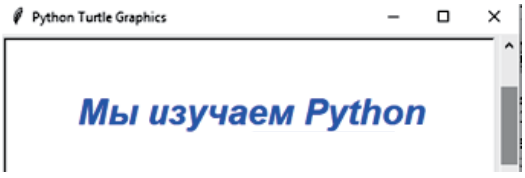
Фигураларды чыгаруудан башка графикалык терезеде текстти да тартса болот. Ал үчүн параметрлери менен **write()** командасы колдонулат:

```
write(text, move, align, font = (fontname, fontsize, fontstyle))
```

- **text** параметрине тырмакчага алынып текст өзү жазылат;
- **align** параметри «left», «right», «center» маанилерин алат жана тексттин абалын ташбакага салыштырмалуу өзгөртөт, маанилери тырмакчада жазылат;
- **font** параметри fontname, fontsize, fontstyle маанилерин алат:
  - **fontname**ге тырмакчада шрифттин аты жазылат;
  - **fontsize** шрифттин өлчөмү үчүн жооп берет;
  - **fontstyle** тексттин стилине жооп берет (**normal** – кадимки, **bold** – кара, **italic** – курсивдүү, **bold italic** – кара, курсивдүү текст).

2-маселе. Формattалган текстти чыгаруучу программа түзөлү:

```
from turtle import*
color ('blue')
write('Биз Python тилин үйрөнөбүз', 'center', font=('Roboto', 24, 'bold italic'))
>>>
```



clear() жана **reset()** командалары сүрөттөрдөн экранды тазалайт жана калемди борборго жайгаштырат.

Графикалык объекттерди түзүү үчүн Tkinter менен иштөө

Tk графикалык китепканасы менен иштөө үчүн Tkinter модулу жогорку деңгээлдеги графика жана анимация менен иштөөгө мүмкүндүк берет. Turtle модулу сыяктуу эле tkinter модулун да кошуп алабыз:

```
from tkinter import*.
```

3-маселе. Алгач кадимки эле баскычты (кнопка) түзүп көрөлү. Ал үчүн Button виджетин колдонобуз, кашаанын ичинде баскычтын параметрлерин беребиз. text маанисинин жардамы менен баскычтын атын жазабыз. Төмөнкү программаны иштетели:

```
from tkinter import*
tk = Tk() #tk ат менен Tk объектисин түзөбүз
btn = Button(tk, text='Пуск')
#«Пуск» тексти менен баскыч түзөбүз
btn.pack() #баскычты терезенин ичинде жайгаштырабыз
>>>
```

Бирок бул баскычты баскандан эч нерсе чыкпайт. Баскычты басуу кандайдыр бир аракет менен коштолсун үчүн программага жыйынтыгы **command** командасы аркылуу чыга тургандай функция киргизели:

БУЛ КЫЗЫКТУУ!

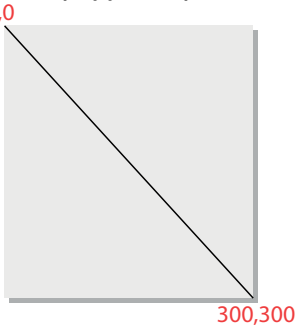
Виджеттер – бул кээ бирлери бардык программалоо тилдеринде стандарттык болгон, программанын графикалык интерфейсин түзүү үчүн атайын базалык блоктор. Мисалы бул кнопкалардын, желекчелердин же жылдырып көрүү тилкелеринин виджеттери. Алар аттары менен гана айырмаланышы мүмкүн: мисалы, классикалык желекчелер (check box), Tkinter де check button деп аталышат.

```
from tkinter import*
def btn_act(): #консолдо жыйынтык чыгара турган функция
    print('Оюн башталды!')
tk = Tk()
btn = Button (tk, text='Пуск', command=btn_act) #баскычка
басканда функциядагы билдирүү чыгарылат
btn.pack()
```

Эми баскычка баскан сайын консолдо ушул маалымат чыгып турат:

```
>>> «Оюн башталды!»
```

Башка виджет – Canvas (англ. холст) берилген аянтта сүрөт тартууга мүмкүндүк берет. Сүрөт тартуу холсттун өлчөмдөрүн: холсттун туурасын (width) жана бийиктигин (height) берүү менен башталат.



Холсттогу сүрөттүн баштапкы чекитин белгилөө үчүн X, У координаталары колдонулат. Координаттар горизонталь боюнча (X) сол четинен, ал эми вертикаль боюнча (У) жогорку четинен канча пикселге жылышкандыгын аныктайт.

Сызыкты түзүү үчүн **create_line()** методу колдонулат, кашаанын ичинде 4 сан көрсөтүлөт. Биринчи экөө сызыктын башталыш координаталары, кийинки экөө – акыркы координаталары болот. Төмөнкү программаны жазалы:

```
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=300, height=300)
canvas.pack()
canvas.create_line(0, 0, 300, 300)
```

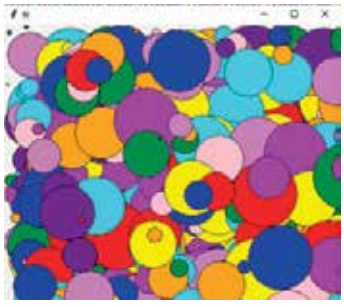
Айлана сызуу үчүн **create_oval()** методу колдонулат:

```
canvas.create_oval(10, 10, 80, 80, outline= 'red', fill=
'green', width=2)
```

Берилген жазууда биринчи 4 параметри фигураны чектөө координаталарын аныктайт. Б.а., бул ичинде айлана сызыла турган квадраттын жогорку сол бурчтун жана төмөнкү оң жак бурчтарынын x жана y координаталары.

4-маселе. Холстто диаметрлери жана түстөрү кокустук сан менен алынган тегеректерди сызуучу программаны түзөлү.

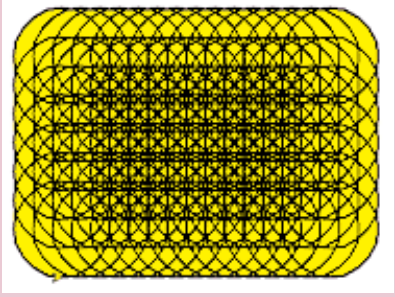
```
from random import* #random модулунан randint жана choice функцияларын жүктөйбүз
from tkinter import*
size = 500 #size өзгөрмөсүн киргизебиз
tk = Tk()
diapason = 0 #тегеректердин санын чектөө үчүн diapason өзгөрмөсүн киргизебиз
my_canvas = Canvas (tk, width=size, height=size) #size өзгөрмөсүнүн маанисин колдонуп холст түзөбүз
my_canvas.pack()#холстту терезенин ичине жайгаштырабыз
while diapason <1000: #цикл ушул шартка чейин кайталанат
    color = choice(['green', 'red', 'blue', 'orange', 'yellow', 'pink', 'purple', 'violet', 'magenta', 'cyan'])
#тегеректердин түстөрүн кокустан тандоо үчүн тизме түзөбүз
    x1 = randint(0,size) #x, y коорд-ды кокустан тандоо
    y1 = randint(0,size)
    d=randint(0,size/5) #тегеректердин диаметрлерин каалагандай тандоо, бирок size/5 тен чоң эмес
    my_canvas.create_oval(x1,y1,x1+d,y1+d,fill=color)#тегеректерди түзөбүз жана кокустан тандалган түс менен ичин боёйбуз
    tk.update()
    diapason+=1 #циклдин кадамы, эсептегич
```



КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *Turtle модулунун жардамында айланалардан килем жасагыла. Мында айланалар бир түс менен, ал эми килемдин фону башка түс менен боелсун.*

2) *Tkinter модулунун жардамында холстко жоондугу жана түсү кокустан тандалган сызыктарды чыгаруучу программаны түзгүлө.*



14-тема:

Рекурсия

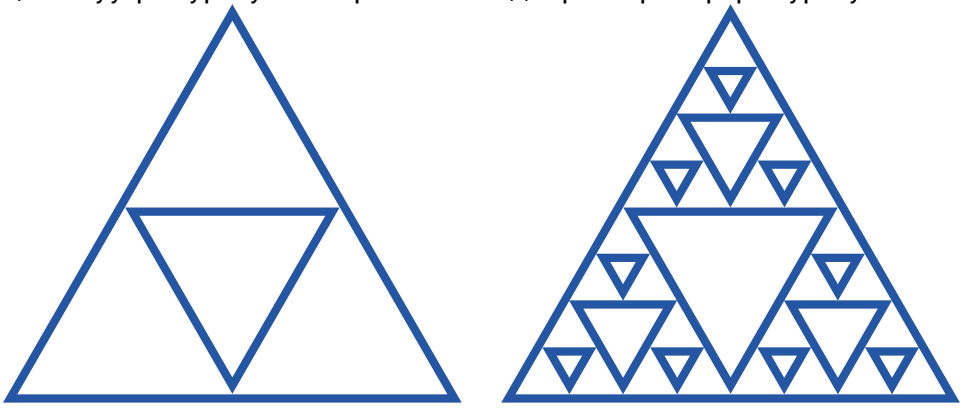
Көпчүлүк учурда бир функция башка функцияны чакырышы мүмкүн. Бирок ошол эле функция өзүн өзү да чакырышы мүмкүн. Программа түздөн түз өзүн чакырган (жөнөкөй рекурсия) же кыйыр (башка функциялар аркылуу), мисалы, А функциясы В функциясын чакырат, ал эми В функциясы А ны чакырган кырдаалдар **рекурсия** деп аталат.

Байкагандай, кыйыр кайрылууда чынжырдагы бардык функциялар – рекурсивдүү.

Рекурсивдүү объекттердин популярдуу мисалдары болуп – **фракталдар** эсептелет. Мындай ат менен математикадагы өзүнө **окшоштукка ээ** фигураларды аташат. Бул демек, алар ар бири бүтүн фигурага окшош болгон кичинекей өлчөмдөгү фигуралар менен куралган дегенди түшүндүрөт.

1-сүрөт. Серпинскийдин үч бурчтугу (1915-жылы польшалык математик В. Серпинский тарабынан сунушталган эң биринчи фракталдардан болуп саналат).

Тең жактуу үч бурчтук кичирээк өлчөмдөгү 4 барабар үч бурчтукка бөлүнөт



(сол жактагы сүрөт), андан кийин борбордогудан башка ар бир алынган үч бурчтук андан ары да майда 4 барабар үч бурчтукка бөлүнүшөт ж.б.

1-маселе. n! саны үчүн факториалды эсептөө мисалында рекурсия кантип иштээрин карайлы. Мисалы, 3 санынын факториалын эсептөө үчүн, 3*2*1ге көбөйтүү керек, б.а. n*(n-1)*((n-1)-1) амалын аткаруу керек.

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n - 1)
print(factorial(3))
>>>
6   #жыйынтык
```

ЭСИҢЕ ТУТ

n санынын факториалы деп, 1ден n ге чейинки бардык натуралдык сандардын көбөйтүндүсүн айтабыз:

n!=1*2*3*4*...*n

Мисалы, 5 санын факториалы 120га барабар 120 (5!=1*2*3*4*5).

Рекурсивдүү функциялар программалоодо көптөгөн маселелерди чечишет, бирок тилекке каршы, аларды жазууда көп учурда каталар кетет. Эң көп таралган ката – бул чексиз рекурсия, бул учурда функцияны чакыруу чынжыры эч качан бүтпөйт жана компьютердин эси толуп калмайынча улана берет.

Көптөгөн чексиз рекурсиянын негизги эки себеби:

1. Рекурсиядан чыгуунун туура эмес жазуусу. Мисалы, эгер биз факториалды эсептөө программасында `if n==0` деген текшерүүнү унутуп койсок, анда factorial(0), factorial(-1) ди чакырат, ал болсо factorial(-2)-ни ж.у.с.
2. Функциянын параметрлеринин туура эмес жазылышы. Мисалы, эгер factorial(n) функциясы кайрадан эле factorial(n) функциясын чакырса да чексиз чынжыр алынат.

Ошондуктан, рекурсивдүү функцияны иштеп чыгууда эң биринчи кезекте ***рекурсиянын базалык шарты*** деп аталган рекурсияны аяктоо шартын туура түзүү керек.

Рекурсиянын түз жана тескери жүрүшү

Рекурсиянын кийинки деңгээлине кирүүгө чейин функция тарабынан аткарылган аракеттер рекурсиянын түз жүрүшүндө аткарылгандар деп аталат. Ал эми тереңирээк деңгээлден учурдагы деңгээлге кайткандагы аткарылган аракеттер – рекурсиянын тескери жүрүшүндөгү аткарылгандар деп аталат.

2-маселе. Натуралдык санды экилик системага которо турган функцияны түзүп көрөлү. Санды экилик системага которуунун стандарттык алгоритмин мындай жазса болот:

```
def printBin (n):
    while n!= 0:
        print (n % 2, end = '')
        n = n // 2
printBin(14) #функцияны чакыруу үчүн мисалы 14 санын киргизели
>>>
0111 #жыйынтык
```

Негизги проблема бул, экилик сандар тескери, б.а. биринчи кезекте акыркы разряд чыгарылгандыгында.

Бул маселени чыгаруунун ар кандай ыкмалары бар. Алардын бардыгы бөлүүдөн кийин калдыктарын эстеп калып (мисалы, символдук сапка), андан соң бардык жыйынтык алынгандан кийин гана аны экранга чыгарууга негизделген.

Рекурсияны колдонуп көрөлү. Идея мындай: n санынын экилик жазуусун чыгаруу үчүн, алгач n//2 санынын экилик жазуусун чыгаруу керек, андан кийин анын калдыгын бөлүп чыгуу керек (n%2). Эгерде алынган сан (параметр) нөлгө барабар болсо, анда функциядан чыгуу керек. Мисал үчүн 14 санын алалы:

14 // 2 = 7, 7 % 2 = 1
7 // 2 = 3, 3 % 2 = 1
3 // 2 = 1, 3 % 2 = 1
1 // 2 = 0, чыгуу

Мындай алгоритм эң жөнөкөй программаланат. **print_bin** аталыштагы функцияны жазалы:

```
def print_bin (n):
    if n == 0: return 0
    print_bin (n // 2)
    print (n % 2, end = '') #калдыктарды бир сапка жазуу
printBin (14) #функцияны чакыруу үчүн мисал катары 14 санын
киргизебиз
>>>
1110 #жыйынтык
```

Ушуну эле циклдин жардамында да жасаса болот. Мындан маанилүү корутунду чыгат: ***рекурсия циклди алмаштырат***. Ошону менен бирге программа көпчүлүк учурда түшүнүктүүрөөк болуп калат.

3-маселе. a санын b даражасына көтөрүү (a**b) үчүн рекурсивдүү функцияны жазалы.

Санды даражага көтөрүүнүн алгоритми төмөнкүдөй аткарылат:

(a *...(a * (a * (a *(a * 1)))))

Математика курсунан биз a**0 = 1 экенин билебиз.

Маселени чыгаруу:

```
def func_step(a, b):
    if b == 0:
        return 1
    else:
        return a * func_step(a, b-1)
print(func_step(2, 4))  #мисалы, 2нин 4-даражасы
>>>
16  #жыйынтык
```

ЭСИҢЕ ТУТ

Чексиз рекурсиянын жыйынтыгын чыгарууну токтотуу үчүн **Ctrl+C** баскычтарынын комбинациясын консолдо басуу керек.

БУЛ КЫЗЫКТУУ!

Рекурсиянын адамдын жашоосундагы мисалдары:

- Сиз банкка процентке акча салдыңыз жана эсептелген проценттер сиздин эсебиңизде банкта калат жана аларга да проценттер эсептелет. Сиз качан банктан чогулган акчаларды алып кеткенче, процесс улана берет.
- Бири-бирине караган күзгүлөрдөгү объекттин чагылышы да чексиз рекурсиянын мисалы болот.

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *n натуралдык саны берилген. 1ден n ге чейинки бардык натуралдык сандарды чыгаргыла.*

2) *n берилген саны үчүн 1ден n ге чейинки бардык сандардын суммасын чыгаруучу рекурсивдүү функцияны жазгыла.*

15-тема:

Массивдерди иштетүү алгоритмдери

6-темада силер тизме, кортеж жана сөздүк түрүндөгү массивдерди үйрөнө баштадыңар, алар менен ар кандай амалдарды жасадыңар жана массивдерге маалыматты киргизип жана чыгарганды билдиңер.

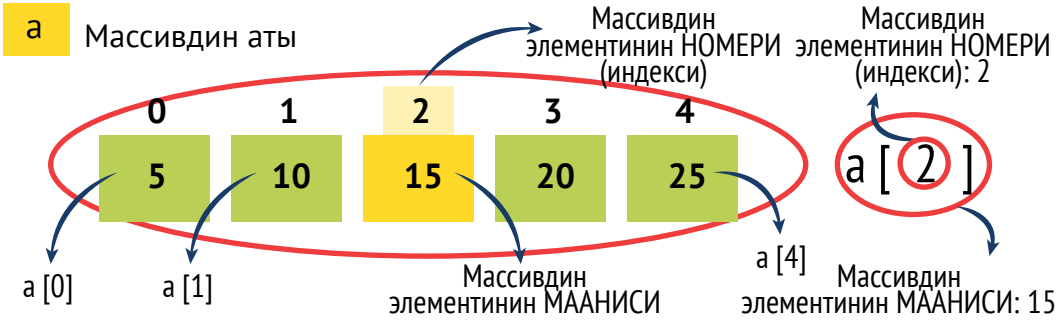
Бул темада биз массивдеги берилиштерди иштетүүнүн негизги стандарттык алгоритмдерин тереңирээк карайбыз:

- издөө
- модификациялоо
- сорттоо

ОШОНДОЙ ЭЛЕ КАРА
10-тема
Массивдер

ЭСИҢЕ ТУТ

Массив – бул жалпы атка ээ болгон эсте жакын жайгашкан (кошуна уячаларда) өзгөрмөлөрдүн тобу. Массивдеги ар бир уяча уникалдуу номерге (индекске) ээ.



Массивден издөө

Издөө алгоритмин маселенин мисалында карап көрөлү. **a** – массивдин аты, **n** – массивдеги элементтердин саны (бүтүн сан), ал эми **i** – өзгөрмөсү тизменин элементтеринин индексин билдирет.

1-маселе. Массивден **x** өзгөрмөсүнүн маанисине барабар болгон элементти табуу керек же ал элемент жок деген билдирүүнү чыгаруу керек. Чыгаруу алгоритми – бул массивдин биринчиден аягына чейинки бардык элементтерин карап чыгуу. Качан гана **x** ке барабар элемент табылганда, циклден чыгып, жыйынтыкты көрсөтүш керек.

Ал үчүн **nx** өзгөрмөсү менен циклди колдонобуз, массивдин бардык элементтерин иргейбиз жана керек болгон маани табылса эле циклди аяктайбыз.

```
a = [1, 5, 4, 31, 10] #5 элементи бар массив берилген
x = int(input('Введите x: ')) #Изделген x маанисин киргизүү
nx = 0 #nx өзгөрмөсү табылган элементтин номерин сактайт
for item in a:          #a массивинин элементтери үчүн
    if item == x:       #эгерде элемент x ке барабар болсо
        nx = item       #анда ал nx өзгөрмөсүнө жазылат
        break           #циклди аяктайт
if nx > = 1:            #nx – өзгөрмөсү өзгөрдү
    print ('Табылды!')#изделген маани табылды деп чыгарат
else:
    print ('Табылган жок')
```

Циклден чыгуу үчүн **break** оператору колдонулат, табылган элементтин номери «**nx**» өзгөрмөсүндө сакталат. Эгерде анын мааниси 0 боюнча калса (циклдин аткарылышында өзгөрбөсө), анда массивде **x** ке барабар элемент жок.

Бул маселени дагы бир жол менен чыгарса болот. Ал үчүн массивде **x** ке барабар биринчи табылган элементтин индексин кайтарган **index** функциясын колдонуу керек:

```
a = [16, 29, -5, -11, 23, 14, -7, 23, 18] #массивдин мисалы
print (a.index (23))
>>>
4
```

Эгерде массивде изделген элемент жок болсо, программа катаны кайтарат жана ишти токтотот.

2-маселе. Массивде **m** өзгөрмөсүнө жазылып кала турган максималдуу элементти табабыз. Ал үчүн массивдеги бардык элементти биринин артынан экинчисин карап чыгуу керек. Эгерде массивдин берилген элементи мурунку максималдуу элементтен чоң болсо (**m** өзгөрмөсүндө жайгашкан), анда **m** өзгөрмөсүндө максималдуу элементтин жаңы маанисин эстеп калабыз.

Массивдин элементтери бизге белгисиз болгондугуна байланыштуу **m** ге нөл же терс санды жазышыбыз керек. Эгерде ал **m = a[0]** болсо, анда иргөө цикли экинчи элементтен башталат, башкача айтканда **a[1]**ден:

```
a = range (1, 20)
m = a[0]
for i in range (1, 20):
    if a[i] > m:
        m = a[i]
print (m)
```

Мына, дагы бир варианты:

```
a = range (1, 20)
m = a[0]
for x in a:
    if x > m:
        m = x
print (m)
```

Мунун айырмасы, ал индекс-өзгөрмөнү колдонбойт, бирок **a[0]** элементин эки жолу карайт (2-жолу – бардык элементтерди иргөө циклинде).

Максималдык жана минималдык элементтерди издөө көп учурда колдонулгандыктан, Python тилинде тиешелүү **max(), min()** камтылган функциялары каралган.

```
a = range (1,20)
m = max (a)
print (m)
```

Массивди модификациялоо

Массивдер менен иштөө үчүн жана андагы берилиштерди өзгөртүү үчүн, Python тилинде көптөгөн ар кандай функциялар бар, мисалы:

ФУНКЦИЯ	МААНИСИ	МИСАЛ

print (a)	а тизмесин экранга чыгарат	`a = [16, 'b', 34, 'c']#бардык мисалдар үчүн базалык тизме`
		`print (a)`
		`>>> [16, 'b', 34, 'c']`

append ()	Тизменин артына бир элемент кошот	`a.append (18)`
		`print (a)`
		`>>> [16, 'b', 34, 'c', 18]`

clear ()	Тизменин бардык элементин өчүрөт	`a.clear ()`
		`print (a)`
		`>>> []`

count ()	Берилген мааниси менен элементтердин санын кайтарат	`print (a.count (16))#тизмеде мааниси 16га барабар канча элемент бар экендигин эсептейт` `>>> 1`

ФУНКЦИЯ	МААНИСИ	МИСАЛ

extend ()	Базалык тизменин аягына башка тизмени кошот	`B = [18, 'h'] #экинчи тизме`
		`a.extend (b)`
		`print (a)`
		`>>> [16, 'b', 34, 'c', 18, 'h']`

index ()	Биринчи табылган окшош элементтин индексинин номерин кайтарат	`print (a.index (34))` `>>> 2`

insert ()	Элементтерди индекси боюнча коюп чыгат	`a.insert (1, 22) #индексти эсептөө 0дөн башталгандыктан, базалык тизмеде 1 индексинде 'b' турат, демек анын ордуна 22 цифрасы коюлуп, калгандары оңго жылат.`
		`print (a)`
		`>>> [16, 22, 'b', 34, 'c']`

pop ()	Берилген индекстеги элементти өчүрөт	`a.pop (0) #0 индексиндеги маанини өчүрүү`
		`print (a)`
		`>>> ['b', 34, 'c']`
		
		`a.pop ()`
		`print (a)`
		`>>> [16,'b', 34] #эгерде мааниле-ри берилбесе, анда акыркы элемент өчүрүлөт`

remove ()	Берилген мааниси менен 1-элементти өчүрөт, эгерде элемент табылбаса ValueError билдирүүсү чыгат	`a.remove (34)`
		`print (a)`
		`>>> [16, 'b', 'c']`

reverse ()	Тизменин элементтерин тескери иретте жайгаштырат	`a.reverse ()`
		`print (a)`
		`>>> ['c', 34, 'b', 16]`

sort ()	Тизмени сорттойт (бир типтеги элементтүү тизме үчүн гана)	`a = [16, 8, 34, 3] #тизмеде жалаң бүтүн (int) сандар`
		`a.sort ()`
		`print (a)`
		`>>> [3, 8, 16, 34] #өсүү тартибинде сорттойт`
		
		`a = ['m', 'b', 'o', 'c'] #тизмеде жалаң символдор (str)`
		`a.sort ()`
		`print (a)`
		`#алфавит боюнча сорттойт`
		`>>> ['b', 'c', 'm', 'o']`

Булардын кээ бирөөлөрүн тереңирээк карайлы: массивдин реверси, массивдин элементтерин жылдыруу жана керектүү элементтерди тандоо.

Массивдин реверси

Массивдин реверси *(reverse)* – бул анын элементтерин тескери тартипте жайгаштыруу: биринчи элемент акыркысы болуп, ал эми акыркы элемент биринчи жайгашып калат.

Python тилинде массивдер 0дөн башталат. Ошондуктан эгерде элементтердин жалпы саны N болсо, анда i үчүн каршысындагы элемент төмөнкү формула менен аныкталат: **N - i - 1**. Мисалы (төмөнкү таблицаны карагыла), 2 индексиндеги элементти алмаштыра турган элементтин индексин табуу үчүн, төмөнкү формула менен чыгарабыз:

N i
9 – 2 – 1 = 6

Демек, 2 индексиндеги элемент 6 индексиндеги элемент менен алмашат.

Массивдин элементтери	16	29	-5	-11	23	14	-7	23	18
Индекстер	0	1	2	3	4	5	6	7 же (n -2)	8 же (n -1)

Элементтин экинчи түгөйүн табуу үчүн, биз массивдин биринчи жарымындагы эле индекстерди карайбыз. Бул болсо циклди массивдин ортосунан токтотуу керек дегенди түшүндүрөт:

```
a = [16, 29, -5, -11, 23, 14, -7, 23, 18]
n = len(a)  #массивдеги элементтердин санын эсептейбиз
for i in range(n//2):#массивдин 1-жарымынын индекстери
    a[i], a[n-i-1] = a[n-i-1], a[i] #элементтердин ордун алмаштырабыз
print(a)
>>>
[18, 23, -7, 14, 23, -11, -5, 29, 16]
```

Массивди реверстөө амалын стандарттык **reverse()** методунун жардамында да аткарса болот:

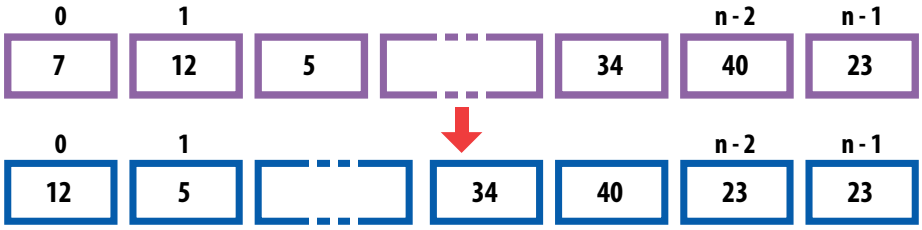
```
a.reverse ()
```

ЭСИҢЕ ТУТ

Python тилинде индекстердин маанилери терс болушу да мүмкүн. Бул учурда номерлөө аягынан баштап жүрөт. Мисалы, массивдин эң акыркы элементинин индекси -1, андан беркисиники -2 ж.б.

Массивдин элементтерин жылдыруу

Массивдин элементтерин коюуда же өчүрүүдө алардын бардыгын же бөлүгүн тиги же бул жакка жылдыруу керек болот. Массивди көп учурда 1-элементи сол жагында жайгашкандай, таблица түрүндө көрсөтүшөт. Ошондуктан солго жылдыруу – бул **a[1] a[0]**дүн ордуна, **a[2] a[1]**дин ордуна ж.у.с. бардык элементтерди бир уячага жылдыруу болуп саналат.



Акыркы элемент өзүнүн ордунда эле калат, б.а. кайталанат, анткени ал жаңы маанини эч кайдан ала албайт, массивдин элементтери түгөндү.

Алгоритмин жазалы:

```
a = [16, 29, -5, -11, 23, 14, -7, 23, 18]
n = len(a)
for i in range (n-1):
    a[i] = a[i+1]
print(a)
>>>
[29, -5, -11, 23, 14, -7, 23, 18, 18]
```

Көңүл бурсаңар мында массивдин чегинен чыгып кетпеши, б.а. жок элемент a=[n] ге кайрылбашы үчүн цикл a=[n-1] де аяктап жатат.

Биз көрүп тургандай, биринчи элемент жок болуп кетти, ал эми акыркы элемент эки жолу кайталанды. Негизи биринчи элементтин мурунку маанисин акыркынын ордуна жазып койсо деле болот. Мындай жылдыруу циклдик деп аталат. Ал үчүн алдын ала (цикл башталганга чейин) биринчи элементти **c** кошумча өзгөрмөсүнө сактап коюу керек, ал эми цикл аяктагандан кийин аны массивдин акыркы уячасына жазып салуу керек:

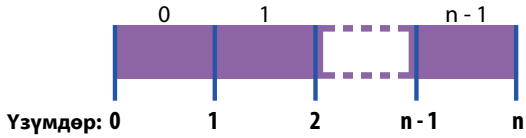
```
a = [16, 29, -5, -11, 23, 14, -7, 23, 18]
n = len(a)
c = a[0]
for i in range(n-1):
    a[i] = a[i+1]
a[n-1] = c
print(a)
>>>
[29, -5, -11, 23, 14, -7, 23, 18, 16]
```

Pythondун камтылган мүмкүнчүлүктөрүн колдонуп жылышууну оңой аткарса болот:

```
a = a[1:n] + [a[0]]
```

Бул жерде a[0] биринчи элементи a[1:n] кесилип алынган массивдин артына коюлат. Эми ал нөлдөн эмес бирден башталып калат.

Оң жактагы сүрөттөн биз a[0:2] үзүмү 0дөн 2ге чейинки бардык элементтерди камтыйт, б.а.: a[0] жана a[1]ди.



Керектүү элементтерди тандоо

Кайсы бир шартты канааттандырган **a** массивинин бардык элементтерин **b** массивине чогултуу керек. Ал үчүн баштапкы массивдин элементтерин иргеп, улам кийинки элемент бизге туура келсе, анда экинчи эсептегичти колдонуп, аны жаңы массивге кошобуз. Мисалы, 2-тизмеге так сандарды чогултабыз:

```
a = [16, 29, -5, -11, 23, 14, -7, 23, 18]
b = []
for x in a:
    if x % 2 == 0:
        b.append(x)
print(b)
>>>
[16, 14, 18]
```

Чыгаруунун экинчи варианты – шарты менен генераторду колдонуу:

```
a = [16, 29, -5, -11, 23, 14, -7, 23, 18]
b = [x for x in a if x % 2 == 0]
print(b)
```

Бул жерде **b** тизмесине 2ге гана бөлүнгөн элементтер тандалды.

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *Массивди (0, 20) интервалындагы кокустук сандар менен толтургула. X санын киргизгиле жана Хке барабар бардык маанилерди тапкыла.*

2) *N элементтен турган сан маанисиндеги бир өлчөмдүү массив берилген. Массивдин элементтерин оң жакка айланма жылдыруу-ну аткар, б.а. a(1) --> a(2); a(2) --> a(3); ... a(n) --> a(1).*

16-тема:

Тизмелерди сорттоо

Көпчүлүк учурда керектүү маалыматты издөөнү жеңилдетүү үчүн биз сорттоону колдонобуз. Мисалы, сөздүктө алфавит боюнча сөздөрдү сорттоо издөөнү жеңилдетет.

Программалоодо **сорттоо** – бул массивдин элементтерин берилген иретте жайгаштыруу болуп саналат.

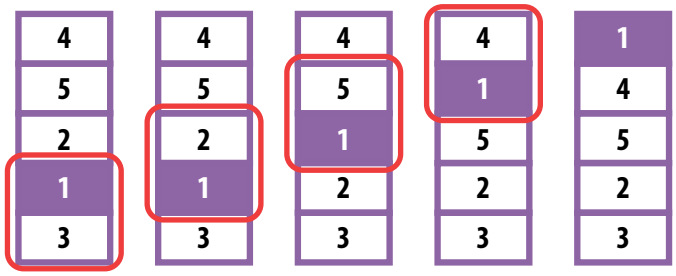
Сорттоо ирети ар кандай болушу мүмкүн: сандар үчүн адатта мааниле-ринин өсүү (кемүү) тартибинде сорттоо каралат. Мисалы, [3 1 0 5 2 7] бир өлчөмдүү массивин өсүү тартибинде сорттоодо [0 1 2 3 5 7] массиви алынат. Символдук берилиштер адатта алфавиттик тартипте сорттолушат.

Негизи сорттоонун көптөгөн ыкмалары ойлоп табылган. Жалпысынан аларды эки топко бөлүп караса болот: 1) жөнөкөй, бирок өтө жай иштөөчү (чоң массивдер үчүн), 2) татаал, бирок тез иштөөчү.

Сорттоонун эң эле көрсөтмөлүү методдорунун бири болгон – «көбүкчө методун» карайлы.

Көбүкчө методу (алмаштыруу менен сорттоо)

Бул методдун аты белгилүү физикалык кубулуштан алынган: суудагы аба көбүкчөсү өйдө көтөрүлөт. Массивди бул ыкма менен сорттоо үчүн, массивден солдон оңду көздөй коңшу элементтерди экиден салыштырып өтүү керек. Качан сол жактагы элемент оң жактагысынан чоң болуп калса, анда алардын ордун алмаштыруу керек. Ошентип 1-өтүштөн кийин эң чоң элемент тизменин акырына келип калат. Эми тизмени кайрадан иргеп чыгуу керек. 2-өтүштө эң чоң элемент тизменин аягында тургандыктан аны салыштырбайбыз. Б.а. салыштыруулардын саны 1ге азаят. Демек, циклдеги өтүштөрүнүн саны тизменин элементтеринин санынан 1ге аз болот.



Өзүнүн ордуна биринчи (минималдык) элементти орноткон биринчи өтүштү псевдокоддо мындай жазса болот:

```
i  үчүн n-2ден 0гө чейин, кадам -1
    if (a[j] > a[j+1])
        ордун алмаштыр  a[j] жана a[j+1]
```

Мында **i** өзгөрмөсү минималдык элемент жазылган уячанын индексин сактайт. Алгач бул биринчи уяча болот. **j** өзгөрмөсү учурда каралып жаткан уячаны билдирет.

Бир өтүштө мындай цикл бир элементти гана орунга коёт. Экинчи элементти «тартыш» үчүн, циклдин башындагы нин акыркы мааниси гана айырмалуу болгон ушул сыяктуу эле дагы бир циклди жазуу керек. Эң чоң элемент өзүнүн ордунда тургандыктан ага тийишпешибиз керек:

```
for i in range(n-1): #өтүүлөрдүн саны
    for j in range(n-i-1): #акыркы элемент каралбайт
        if a[j] > a[j+1]:
            a[j], a[j+1] = a[j+1], a[j]
```

Ушундай циклдерден n-1 санда жасоо керек, б.а. тизмедеги элементтердин санынан 1ге кем. Эмне үчүн n эмес? Анткени эгерде n-1 элемент өздөрүнүн ордуна коюлса, анда калганы автоматтык түрдө өзүнүн ордуна турат – башка орун жок.

Көбүкчө методу боюнча толук программаны жазалы:

```
from random import randint
n = 10
a = [randint (1,99) for n in range (n)]
print (a)
for i in range (n-1): #тизмеден өтүүлөрдүн саны
    for j in range (n-i-1): #салыштыруулардын саны іге азаят
        if a [j] > a [j+1]: #j жана j+1 элементтерин салыштырат
            a[j], a [j+1] = a[j+1], a[j] #керек болсо эле-
менттердин ордун алмаштырат
print(a)
>>>
[5, 84, 90, 37, 30, 32, 29, 62, 17, 99]
[5, 17, 29, 30, 32, 37, 62, 84, 90, 99]
```

Тандоо методу

Дагы бир популярдуу сорттоо методу – бул тандоо методу. Алгач бардык массив каралып чыгат, минималдык элемент табылганда ал массивдин эң башына коюлат. Экинчи жүрүштө калган бардык, б.а. экинчиден баштап акыркы элементке чейин каралат, кайрадан эң кичине элемент табылат жана ал 2-орунга жылдырылат. Андан ары 3-элементтен баштап дагы эң кичине элементи табылат жана 3-орунга жылдырылат, ж.б.у.с. (n-1)-элементке чейин.

Тизмени тандоо методу боюнча сорттоонун алгоритмин жазалы. Мында i өзгөрмөсү минималдык элемент жазылган уячанын индексин сактайт, ал эми j өзгөрмөсү каралып жаткан элементти сактайт.

```
for i in range(n-1):
    n_min = i
for j in range(i+1,n):
    if a[j] < a[n_min]:
      n_min = j
if i != n_min:
    a[i], a[n_min] = a[n_min], a[i]
```

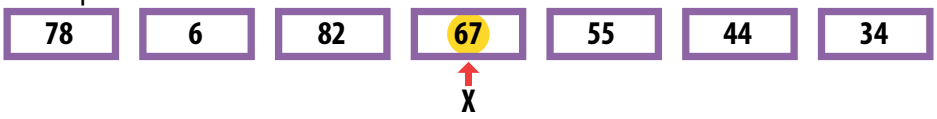
Бул жерде табылган минималдык элемент өзүнүн ордунда турбаса гана орун алмашуу жүрөт, б.а. **i! = nmin** болгондо. Минималдык элементтин номерин издөө циклде аткарылгандыктан, бул сорттоо алгоритми да камтылган циклди түзөт.

«Тез сорттоо» же quicksort

Көбүкчөлүү сорттоо менен тандоо методу чоң берилиштеги массивдер үчүн (10000дей элементтер) жай иштейт. Ошондуктан чоң массивдерди сорттоо үчүн атайын «тез сорттоонун» рекурсивдүү алгоритми колдонулат (англ. quicksort).

Мында сорттоонун идеясы мындай: алгач массив болжол менен ортосунан тең экиге бөлүнөт, андан соң сол жак бөлүгүндөгү элементтерден «ортодо турган» элементтен чоң же ага барабар элементтер тандалып алынат жана оң бөлүгүнө орун алмаштырылат. Андан ары оң жагы жана сол жагы өзүнчө массивдер катары каралат да кайрадан экиге бөлүнүшөт.

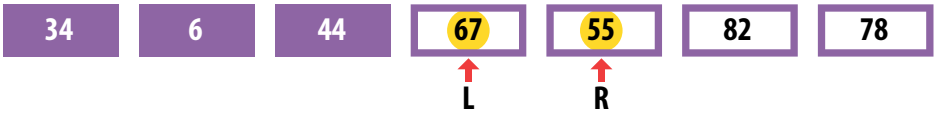
Бул методдун маңызын түшүнүш үчүн мисал карайлы. Бизге төмөндөгүдөй массив берилсин:



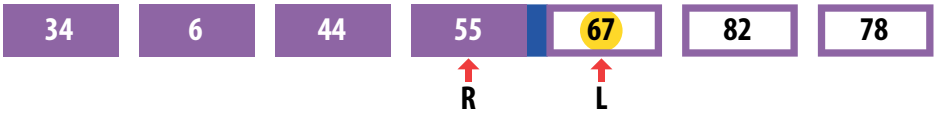
X деп массивдин ортоңку элементин карайлы, б.а. 67. Массивдин сол жагындагы **X**тен чоң же ага барабар элементти табабыз. Бул элемент 78 саны. Бул элементтин индексин **L** менен белгилейли. Эми **X**тен оң жакта турган эң кичине элементти табалы. Бул 34 саны. Анын индексин **R** тамгасы менен белгилейбиз.

Эми ушул эки элементтин ордун алмаштырабыз. **L** өзгөрмөсүн оңго, ал эми **R** өзгөрмөсүн солго жылдыруу менен биз ордун алмаштыра турган кийинки түгөйдү табабыз. Булар 82 жана 44 сандары.

Орундары алмаша турган кийинки түгөйлөр – 67 жана 55 сандары.



Бул орун алмашуудан кийин, андан аркы издөө **L** өзгөрмөсү **R** өзгөрмөсүнөн чоң болуп калгандыгына алып келет, б.а. массив экиге бөлүнүп калды. Жыйынтыгында **A[L]** дин сол жагында жайгашкан массивдин бардык элементтери **X** тен кичине же барабар, ал эми **A[R]**дин оң жагында жайгашкан бардык элементтер **X** тен чоң же барабар.



Ошентип, баштапкы массивди сорттоо, массивдин эки бөлүгүн сорттоого, б.а. ошол эле типтеги эки маселени (бирок кичинекей өлчөмдөгү) чечүүгө алып келди. Эми ушул алгоритмди массивдин алынган эки бөлүгүнө колдонуу керек: биринчи бөлүгү – 1ден **R**ге чейинки элементтер, экинчи бөлүгү - **L**ден акыркы элементке чейин.

Биз айтып кеткендей сорттоонун ылдамдыгы чоң массивдер менен иштөөдө маанилүү. Төмөндөгү таблицада кокустук маанилер менен толтурулган ар кандай өлчөмдөгү массивдердин биз үйрөнгөн үч алгоритмди колдонуп сорттоо убактысы (секунда менен) көрсөтүлгөн.

№	Көбүкчө методу	Тандоо методу	Тез сорттоо
1000	0,09 с	0,05 с	0,002 с
5000	2,4 с	1,2 с	0,014 с
15000	22 с	11 с	0,046 с

Таблицадан көрүнүп тургандай, мисалы 15 миң элементи менен массивди тез сорттоо, көбүкчө методуна караганда 500 эсе тез иштейт экен.

Мурунку темадан Python тилинде массивди сорттоо үчүн **sorted** деген атайын камтылган функция бар экенин билебиз. Ал көптөгөн белгилүү берилиштер менен оңой иштеген ***timsort*** гибриддик алгоритмин колдонот.

Андан тышкары аны жазууда төмөнкүлөрдү эске тутуш керек:

```
a.sort ()  #a тизмесинин өзү сорттолот
b = sorted (a) #b массивине өсүү тартибинде сорттолгон a массиви жайгаштырылат.
```

Адатта сорттоо өсүү тартибинде же тагыраак айтканда, ар бир кийинки элемент мурункусунан чоң же барабар болушу керек. Массивди кемүү тартибинде сорттоо үчүн (ар бир кийинки элемент мурункусунан кичине же ага барабар) сорттоо функциясына reverse = True деп көрсөтүш керек:

```
b = sorted (a, reverse = True) #же
a.sort (reverse = True)
```

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) *N элементтен турган бир өлчөмдүү сандык маанидеги массив берилген. Баштапкы массивдин элементтеринен эки жаңы массив тургузгула. Биринчисине 3кө бөлүнгөн гана сандар, ал эми экинчисине 5ке бөлүнгөн гана сандар кирсин.*

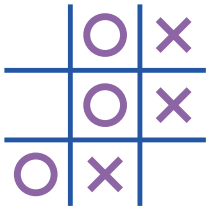
2) *Биринчи суроодогу программаны уланткыла жана 3кө бөлүнгөн сандарды өсүү тартибинде, ал эми 5ке бөлүнгөн сандарды кемүү тартибинде жайгаштыра тургандай кылып программаны толуктагыла.*

17-тема:

Матрицалар

Матрица – бул таблицалык структурага ээ болгон эки өлчөмдүү массив. Эки өлчөмдүү массивдер деле бир өлчөмдүү массивдердей баяндалат. Айырмасы эки өлчөмдүү массивдин элементинде эки координата (эки индекс) – элемент жайгашкан саптын жана мамычанын номерлери бар.

Мисалы, «X жана O» оюну үчүн программаны түзүүдө бош клеткаларга «-1» кодун, нөл бар клеткага 0 кодун, ал эми Хтер бар клеткага 1 кодун ыйгарса болот:



	0	1	2
0	-1	0	1
1	-1	0	1
2	0	1	-1

Python тилинде таблицалар менен иштөө үчүн тизмелерди колдонушат. Эки өлчөмдүү таблица – бул ар бир элементи тизме болуп сакталган тизме («тизмелердин тизмеси»). Мисалы, сүрөттө көрсөтүлгөн таблицаны мындай жазсак болот:

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

Бул тизменин бир сапка да жазса болот:

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

Бирок адам матрицаны таблица катары кабыл алгандыктан, аны экранда да таблица түрүндө чыгарган жакшы. Ал үчүн матрицаны мындай жазсак болот:

```
a = [[-1, 0, 1], [-1, 0, 1], [0, 1, -1]]
for i in range ( len(a) ):
    for j in range ( len(a[i]) ):
        print ( '{:4d}'.
format(a[i][j]), end = '' )
    print ()
>>>
```

`{:4d}` чыгаруу форматы – мамычалардын арасындагы талааны кеңейтет: биздин мисалда ар бир элементтин алдында 4 бош орун коюлат.

```
-1  0  1
-1  0  1
 0  1 -1
```

Мында **i** – камтылган тизменин индекси (саптын санын аныктайт), ал эми **j** – камтылган тизменин ичиндеги элементтин индекси (мамычалардын санын аныктайт); **len(a)** – бул чоң тизмедеги камтылган тизмелердин саны (бул жерде алар 3), **len(a[i])** – мамычалардын саны менен дал келген камтылган тизмедеги элементтердин саны.

a [i] [j] – бул j индекси менен i камтылган тизмесиндеги элемент:

`a[0][0]==-1, a[0][1]==0, a[0][2]==1, a[1][0]==-1,` ж. у. с.

Бул мисалды мындай жазса да болот:

```
for row in a:      #a сабында
    for elem in row:  #саптагы элемент үчүн
        print(elem, end=' ') #элементтерди чыгар
    print()
```

1-маселе. Матрицаны кокустук сандар менен толтурабыз. Саптын жана мамычанын санын клавиатурадан киргизебиз.

Матрицанын ар бир элементине каалагандай маанини ыйгарууга болот. Индекстер экөө болгондуктан матрицаны толтуруу үчүн камтылган циклди колдонобуз. Мындан ары **n** сабынан жана **m** мамычасынан турган **a** матрицасы бар деп, ал эми **i** жана **j** – саптын жана мамычанын индексин билдирген бүтүн сандуу өзгөрмөлөр деп эсептейли. Бул мисалда матрица кокустук сандар менен толтурулат жана экранга чыгарылат:

```
import random
n = int(input ('Саптын санын киргизгиле: '))
m = int(input ('Мамычанын санын киргизгиле: '))
a = []
for i in range (n):
    a.append([])
    for j in range (m):
        a[i].append (random.randint (10,40))#ар бир элемент-
ке 10дон 40ка чейинки кокустук сан ыйгарылат
for i in a:  #ар бир камтылган тизме жаңы саптан чыгарылат
    print (i) #бирок чарчы кашаада
>>>
[33, 16, 31, 33]  #n=2, m=4 болгондогу кокустук сандар
[39, 35, 11, 15]
```

Ушул эле маселени жыйынтыгында кашаалар жок болгондой кылып кыскача жазалы:

```
import random
n = int(input ('Саптын санын киргизгиле: '))
m = int(input ('Мамычанын санын киргизгиле: '))
a = [[random.randint(10, 40) for i in range(m)] for j in
range(n)]
print(' '.join([str(elem) for elem in row])) #бардык эле-
менттер биригишет, ортосунда бош орун аркылуу тизмектелет
```

Эки өлчөмдүү массивди иштетүү

Матрицанын бардык элементтерин иргеп чыгуу үчүн дал ушундай эле камтылган эки циклди колдонуу керек. Биринчи цикл саптын номерлерин иргейт, экинчи цикл болсо саптын ичиндеги элементтерди иргейт. Мисалы, бардык элементтердин суммасы (s) мындай эсептелет:

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
            s = 0
            for i in range(len(a)):
                for j in range(len(a[i])):
                    s + = a[i][j]
            print(s) #жыйынтыгы 45
```

Бул жазуулар үчүн даяр функция **sum** ду колдонсо да болот:

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
            s = 0
            for row in a:
                s + = sum(row)
            print (s)
```

Матрицанын кээ бир элементтерин иштетүүнү маселенин мисалында карап көрөлү.

2-маселе. n саптан жана n мамычадан турган квадраттык массив берилсин дейли. Диагоналды бирлер менен, анын сол жагындагы аймакты 2лер менен, ал эми оң жагындагы аймакты нөлдөр менен толтуруу керек.

```
1  0  0  0
2  1  0  0
2  2  1  0
2  2  2  1
```

Негизги диагональ – бул a[0,0], a[1,1], ..., aA[n-1,n-1] элементтери, б.а. саптын номери мамычанын номерине барабар. Негизги диагоналды бирлер менен толтуруш үчүн бир цикл керек:

```
for i in range(n):    #A[i][i] менен иштейбиз
```

```
a[i][i] = 1          #аны бирлер менен толтурабыз
```

Диагоналдан оң жактагы элементтерди нөлдөр менен толтурушубуз керек, ал үчүн i номериндеги ар бир саптагы a[i][j] элементтерине j=i+1, ..., n-1 үчүн 0 маанисин ыйгарышыбыз керек. Бул жерден бизге камтылган цикл керек болот:

```
for i in range(n):
    for j in range(i + 1, n):
        a[i][j] = 0
```

Ушундай жол менен эле j=0, ..., i-1 үчүн a[i][j] элементине 2 маанисин ыйгарабыз:

```
for i in range(n):
    for j in range(0, i):
        a[i][j] = 2
```

Эгерде сырткы циклдерди бирөөгө чогултсак, анда мындай болгон бир чыгарылышын алабыз:

```
n = 4
a = [[0] * n for i in range(n)]
for i in range(n):
    for j in range(0, i):
        a[i][j] = 2
    a[i][i] = 1
    for j in range(i + 1, n):
        a[i][j] = 0
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

КОМПЬЮТЕРДИК ПРАКТИКУМ:

1) ***n*** *саптан жана* ***m*** *мамычадан турган тик бурчтуу* ***a*** *матрицасын кокустук сандар менен толтургула. Массивдин элементтеринин орточо арифметикалык маанисин тапкыла.*
2) *Матрицанын ар бир сабы үчүн табылган орточо маанилеринин эң чоңун аныктагыла.*

№1 тиркеме

Арифметикалык операторлор

Оператор	Мааниси	Мисалдар
+	**Кошуу** – операторлон сол жана оң жактагы маанилерин суммалайт	*10 + 5 жыйынтыгында 15 болот* *27 + -3 жыйынтыгында 24 болот* *9,4 + 7 жыйынтыгында 16,4 болот*
-	**Кемитүү** – сол операндан оң операндды кемитет	*15 - 5 жыйынтыгында 10 болот* *20 - -3 жыйынтыгында 23 болот* *13,4 - 7 жыйынтыгында 6,4 болот*
*	**Көбөйтүү** – операндарды көбөйтөт	*5 * 5 жыйынтыгында 25 болот* *7 * 3,2 жыйынтыгында 22,4 болот* *-3 * 12 жыйынтыгында -36 болот*
/	**Бөлүү** – сол жактагы операндды оң жактагысына бөлөт	*15 / 5 жыйынтыгында 3 болот* *5 / 2 жыйынтыгында 2,5 болот*
%	**Модулу боюнча бөлүү** – сол жактагы операндды оң жактагысына бөлөт жана калдыгын чыгарып берет	*6 % 2 жыйынтыгында 0 болот* *7 % 2 жыйынтыгында 1 болот* *13,2 % 5 жыйынтыгында 3,19999999 болот*
**	**Даражага көтөрүү** – сол операндды оң жактагы даражага көтөрөт	*5 ** 2 жыйынтыгында 25 болот* *2 ** 3 жыйынтыгында 8 болот* *-3 ** 2 жыйынтыгында -9 болот*
//	**Бүтүн бөлүктүү бөлүү** – бөлүүнүн жыйынтыгында бүтүн бөлүгү гана чыгарылат. Үтүрдөн кийинки бөлүгү алып салынат	*12 // 5 жыйынтыгында 2 болот* *4 // 3 жыйынтыгында 1 болот* *25 // 6 жыйынтыгында 4 болот*

Салыштыруу операторлору

==	Эки операнд тең барабар экенин салыштырат, эгерде барабар болсо, анда шарт чындык болот.	*5 == 5 жыйынтыгында True болот* *True == False жыйынтыгында False болот* *“hello”== “hello” жыйынтыгында True болот*
!=	Эки операнд тең барабар экенин салыштырат, эгерде барабар эмес болсо, анда шарт чындык болот.	*12 != 5 жыйынтыгында True болот* *False != False жыйынтыгында False болот* *“hi”!= “Hi” жыйынтыгында True болот*
<>	Эки операнд тең барабар экенин салыштырат, эгерде барабар эмес болсо анда шарт чындык болот.	*12 <> 5 жыйынтыгында True болот* *!= операторуна окшош*
>	Сол операнд оң жактагыга караганда чоң экенин салыштырат, эгер чоң болсо, анда шарт чындык болот.	*5 > 2 жыйынтыгында True болот* *True > False жыйынтыгында True болот* *“A”>“B” жыйынтыгында False болот*
<	Сол операнд оң жактагыга караганда кичине экенин салыштырат, эгер кичине болсо, анда шарт чындык болот.	*3 < 5 жыйынтыгында True болот* *True < False жыйынтыгында False болот* *“A”<“B” жыйынтыгында True болот*
>=	Сол операнд оң жактагыга караганда чоң же барабар экенин салыштырат, эгер чоң же барабар болсо, анда шарт чындык болот.	*1 >= 1 жыйынтыгында True болот* *23 >=3,2 жыйынтыгында True болот* *“C”>=“D” жыйынтыгында False болот*
<=	Сол операнд оң жактагыга караганда кичине же барабар экенин салыштырат, эгер кичине же барабар болсо, анда шарт чындык болот.	*4 <= 5 жыйынтыгында True болот* *0 <= 0,0 жыйынтыгында True болот* *-0?001 <= -36 жыйынтыгында False болот*

Ыйгаруу операторлору

=	Сол жактагы операндга оң жактагы операнддын маанисин ыйгарат.	*b = 23 b өзгөрмөсүнө 23 маанисин ыйгарат*
+=	Оң жактагы операнддын маанисине сол жактагынын маанисин кошуп, аны сол жактагы операндга ыйгарат.	*b = 5* *a = 2* *b += a барабар: b = b + a, b = 7*
-=	Сол жактагы операнддын маанисинен оң жактагынын маанисин кемитип, аны сол жактагы операндга ыйгарат.	*b = 5* *a = 2* *b -= a барабар: b = b - a, b = 3*
*=	Оң жактагы операнддын маанисине сол жактагынын маанисин көбөйтүп, аны сол жактагы операндга ыйгарат.	*b = 5* *a = 2* *b *= a барабар: b = b * a, b = 10*
/=	Сол жактагы операнддын маанисин оң жактагынын маанисине бөлүп, аны сол жактагы операндга ыйгарат.	*b = 10* *a = 2* *b /= a барабар: b = b / a, b = 5*
%=	Операнддарды модулу боюнча бөлөт жана жыйынтыгын сол жактагыга ыйгарат.	*b = 5* *a = 2* *b %= a барабар: b = b % a, b = 1*
**=	Сол жактагы операндды оң жактагынын маанисиндей даражага көтөрөт жана жыйынтыгын сол жактагыга ыйгарат.	*b = 3* *a = 2* *b **= a барабар: b = b ** a, b = 9*
//=	Сол жактагы операндды оң жактагыга бүтүн бөлүктүү бөлөт жана жыйынтыгын сол жактагыга ыйгарат.	*b = 11* *a = 2* *b //= a барабар: b = b // a, b = 5*

Логикалык операторлор

Оператор	Мааниси	Мисалдар
and	Логикалык оператор “ЖАНА”, эгерде эки операнд тең чындык болсо шарт чындык болот.	*True and True барабар True* *True and False барабар False* *False and True барабар False* *False and False барабар False*
or	Логикалык оператор “ЖЕ”, эгерде жок дегенде бир операнд чын болсо, анда бардык сүйлөм чындык болот.	*True or True барабар True* *True or False барабар True* *False or True барабар True* *False or False барабар False*
not	Логикалык оператор “ЭМЕС”, операнддын логикалык маанисин карама-каршысына өзгөртөт.	*not True барабар False* *not False барабар True*

Мүчөлүк операторлор

Мүчөлүк операторлор сап, тизме, кортеж же сөздүктөргө окшогон курама маалыматтардын тибинде элементтердин бар же жок экенин текшерет.

Оператор	Мааниси	Мисалдар
in	Эгерде элемент удаалаштыкта бар болсо анда чындыкты, ал эми жок болсо анда жалганды кайтарып берет.	*“cad” in “cadilac” кайтарат True 1 in[2,3,1,6] кайтарат True “hi” in {“hi”:2, “bye”:1} кайтарат True 2 in {“hi”:2, “bye”:1} кайтарат False (сөздүктөрдө маанисиндеги эмес, ачкычтардагы элементтин бардыгы текшерилет)*
not in	Эгерде элемент удаалаштыкта жок болсо анда чындыкты кайтарып берет.	*in операторунун жыйынтыгына карама-каршы жыйынтыктар.*

Теңдештиктер операторлору

Окшоштуктар операторлору компьютердин эсиндеги эки объекттин жайгашышын салыштырат.

Оператор	Мааниси	Мисалдар
is	Эгерде эки операнд тең бир объектти көрсөтсө, анда чындыкты кайтарат.	*x is y чындыкты кайтарып берет, эгерде id(x) барабар id(y) болсо*
is not	Эгерде эки операнд тең бир объектти көрсөтсө, анда жалганды кайтарат.	*x is y чындыкты кайтарып берет, эгерде id(x) барабар эмес id(y) болсо.*