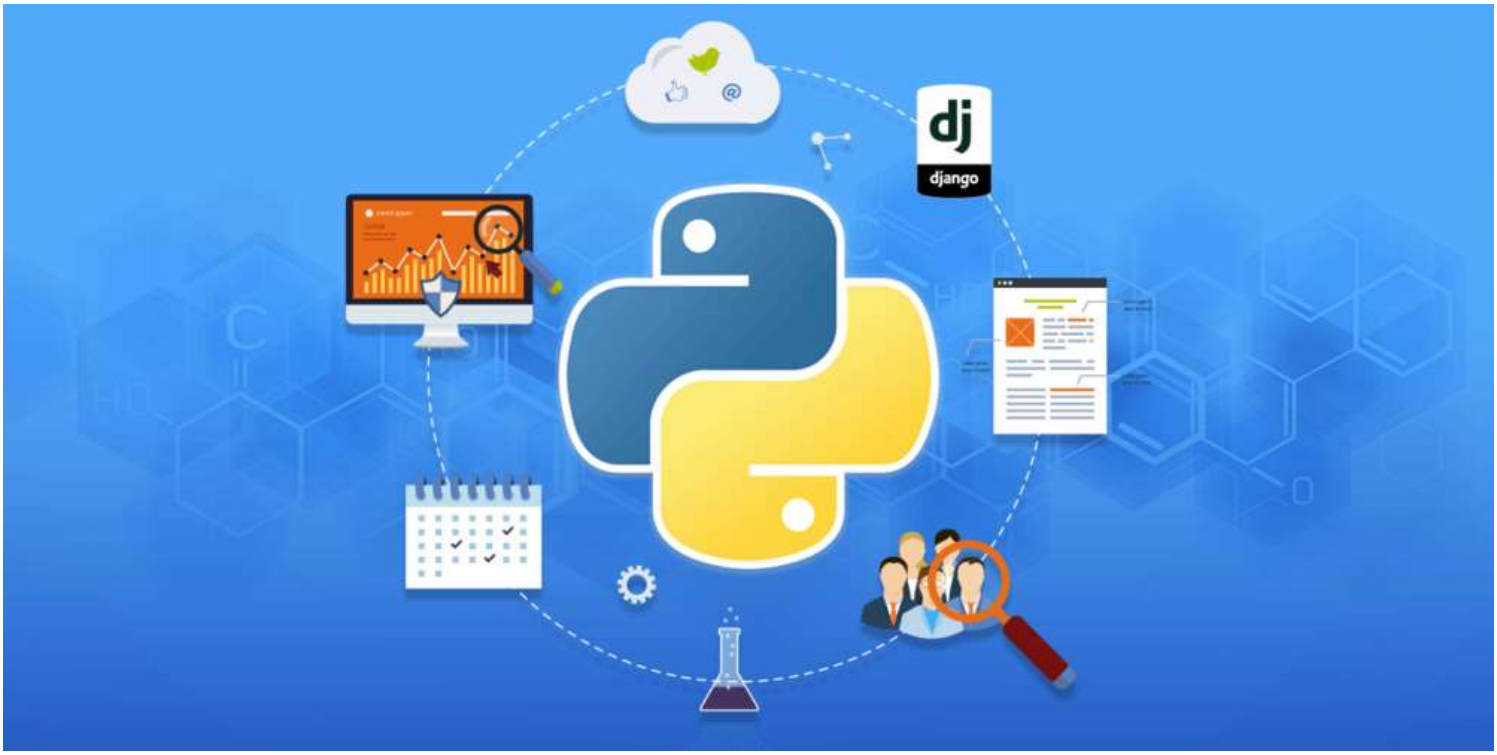
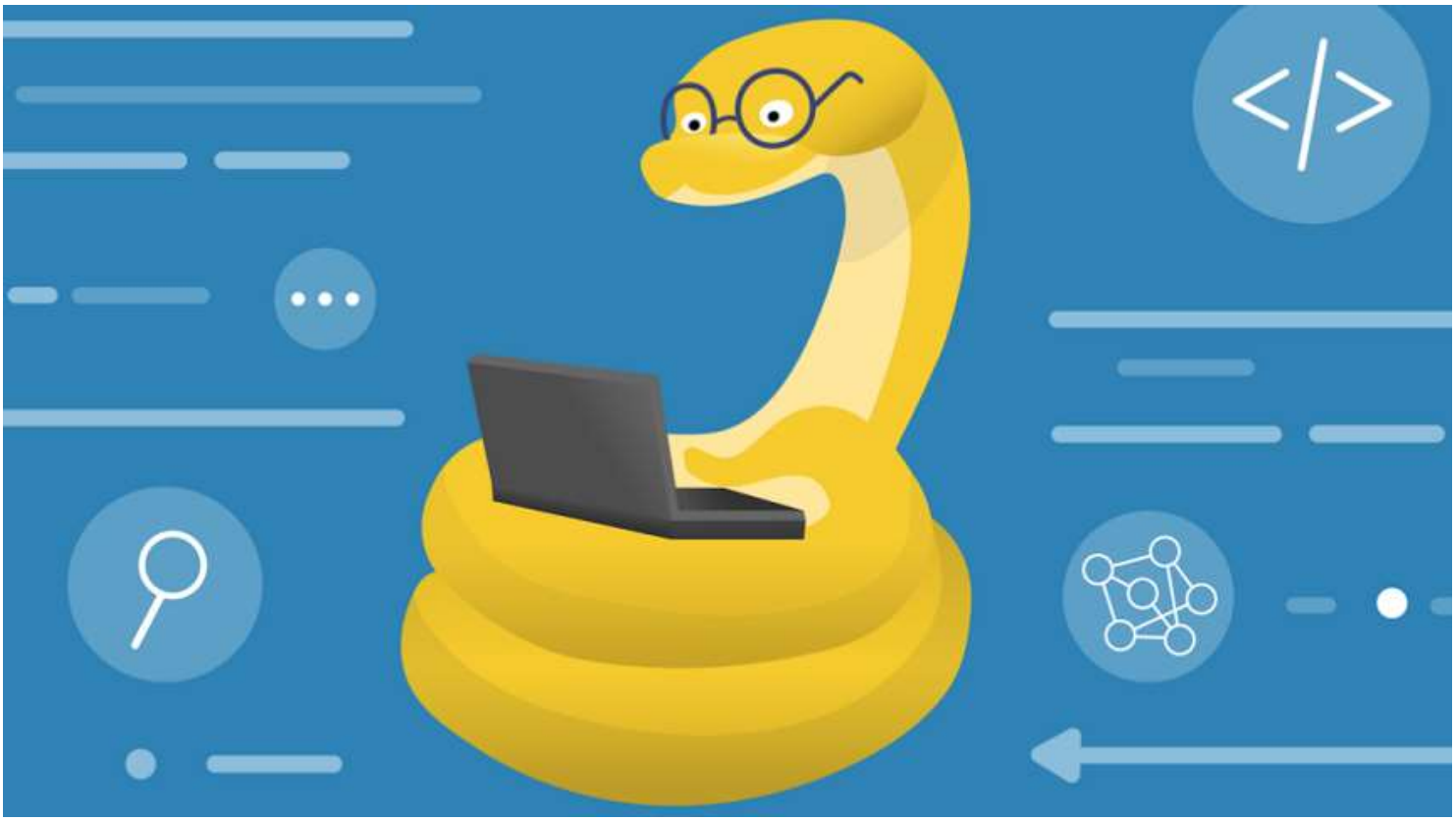


PYTHON ASOSLARI



MUNDARIJA

I - BOB. Python dasturlash tili va sintaksisi

1.1.Python dasturlash tili va uning imkoniyatlari…… 3
1.2. Python dasturlash tili sintaksisi.....…….... 5
1.3. Pythonda o'zgaruvchilar...…….... 8
1.4. Python operatorlari.....… ……..… 12

II - BOB. Pythonda ma'lumot turlari bilan ishlash

2.1. Pythonda ma'lumot turlari……..…… 16
2.2. Pythonda sonlar…..……..…… 17
2.3. Pythonda satrlar……..……..…… 21
2.4. Satrlarni formatlash…...……..…… 28
2.5. Ma'lumot to'plamlari va turlari……..……..…… 30
2.6. List (Ro'yxat) ……..…… 30
2.7. Tuple (Kortej) ……..……..…… 38
2.8. Set (To'plam)……..…… 43
2.9. Dictionary (Lug'at)..……..…….……48

III - BOB. Python statements (bayonnomalar)

3.1. Mantiq elementlari va operatorlari..…… 56
3.2. Pythonda shart operatorlari……..…….... 58
3.3. Pythonda sikllar..…… … …… 61
3.4. While sikli…..……..…… 61
3.5. For sikli……..……..…… 62

IV- BOB. Pythonda funksiya va modullar

4.1. Funksiyalar……..…… 66
4.2. Lambda funksiya……..……..…….... 71
4.3. Pythonda modullardan foydalanish…….... 72
4.4. Pythonda maxsus modullar…… 75
4.5. Sana va vaqt (datetime moduli).…..……..…… 81
4.6. Pythonda matematika…...……..……..…… 83
4.7. Pythonda sanoq sistemasining ishlatilishi..……..…… 87

V- BOB. Pythonda fayllar va istisnolar bilan ishlash

5.1. Pythonda fayllar bilan ishlash……..…… 88
5.2. Pythonda istisnolar bilan ishlash.……..……..... 92
5.3. User input…… … …… 94

VI- BOB. Pythonda obyektga yo'naltirilgan dasturlash (OOP)

6.1. Pythonda OOP tushunchalari……..…… 95
6.2. Pythonda sinf va obyektlar……..…… 98
6.3. Sinflarda konstruktor tushunchasi……..…….. 102
6.3. Sinflarda vorislik tushunchasi…..……..…….. 106

I-BOB. PYTHON DASTURLASH TILI VA SINTAKSISI

PYTHON DASTURLASH TILI VA UNING IMKONIYATLARI

- **Rasmiy sayt –** www.python.org
- **Python** mashhur dasturlash tili. U Guido van Rossum tomonidan yaratilgan va 1991 yilda chiqarilgan.
- **Python** – bu o'rganishga oson va shu bilan birga imkoniyatlari yuqori bo'lgan oz sonlik zamonaviy dasturlash tillari qatoriga kiradi. Python yuqori darajadagi ma'lumotlar strukturasi va oddiy lekin samarador obyektga yo'naltirilgan dasturlash uslublarini taqdim etadi.

Stack Overflow saytining 2019-yildagi dasturchilar o'rtasida dasturlash tillari bo'yicha olib borilgan so'rovnomasida, eng qulay va ko'p foydalaniladigan dasturlash tillari ro'yxatida **Python** JavaScriptdan so'ng ikkinchi o'rinni egallagan. Shu bilan bir qatorda dunyoning **Twitter, Pinterest, HP, Symantec, Instagram va Groupon** kabi yirik korxonalar aynan **Python** dasturlash tilidan foydalanmoqda. **YouTube, DropBox, Google** va **Quora** kabi dunyoning mashhur online platformalarining dasturiy ta'minoti ham aynan python dasturlash tilida yozilganligi ushbu dasturlash tiliga bo'lgan talabning yuqori ekanligini anglatadi. Python nafaqat web sohasida balki sun'iy intellekt va robotexnika sohasida ham yuqori talabga ega til hisoblanadi.

Python DASTURLASH TILIDA YARATILGAN YIRIK LOYIHALAR

Python dasturlash tilining imkoniyatlari

Python – bu o'rganishga oson va shu bilan birga imkoniyatlari yuqori bo'lgan oz sonlik zamonaviy dasturlash tillari qatoriga kiradi. Python yuqori darajadagi ma'lumotlar strukturasi va oddiy lekin samarador obyektga yo'naltirilgan dasturlash uslublarini taqdim etadi.

Python quyidagi sohalarda ishlatiladi:

- Web dasturlash (serverlar bilan)
- Dasturiy ta’minot
- Matematika
- Tizim skriptlari

Nima uchun aynan Pythonni o’rganish kerak ?

- Oddiy, o'rganishga oson, sodda sintaksisga ega, dasturlashni boshlash uchun qulay, erkin va ochiq kodlik dasturiy ta'minot.
- Dasturni yozish davomida quyi darajadagi detallarni, misol uchun xotirani boshqarishni hisobga olish shart emas.
- Ko'plab platformalarda hech qanday o'zgartirishlarsiz ishlay oladi. (Windows, Mac, Linux, Raspberry Pi va boshqalar)
- Interpretatsiya(Интерпретируемый) qilinadigan til.
- Kengayishga (Расширяемый) moyil til. Agar dasturni biror joyini tezroq ishlashini xoxlasak shu qismni C yoki C++ dasturlash tillarida yozib keyin shu qismni python kodi orqali ishga tushirsa(chaqirsa) bo'ladi.
- Juda ham ko'p xilma-xil kutubxonalarga ega.
- xml/html fayllar bilan ishlash;
- http so`rovlari bilan ishlash;
- GUI(grafik interfeys);
- Web dastur tuzish;
- FTP bilan ishlash;
- Rasmli audio video fayllar bilan ishlash;
- Robot texnikada;
- Matematik va ilmiy hisoblashlarni dasturlashda juda ham qo’l keladi.

Pythonni katta proyektlarda ishlatish mumkin. Bularga misol qilib **Google**, *Instagram*, **YouTube** ni misol qilib ko’rsatsak bo’ladi. Bu loyiahalr aynan Python dasturlash tilida yozilgan. Chunki, uni chegarasi yo`q, imkoniyati yuqori. Shuningdek, u sodda va universalligi bilan dasturlash tillari orasida eng yaxshisidir.

PYTHON DASTURLASH TILI SINTAKSISI

Python dasturlash tili sintaksisi o`zi kabi juda sodda:

- Satr oxiri instruksiyaning oxiri hisoblanadi (nuqta vergul shart emas)

  Pythonda sintaksis juda sodda tuzilishga ega. Quyida "**Salom dunyo**" gapini ekranga chiqaruvchi kod ko'rsatilgan:

```
print ("Hello world")
```

- Har bir qator boshidagi bo`sh joy(отступ) muhim ahamiyatga ega. Kiritilgan amallar bo`sh joylarning kattaligiga qarab **blok**larga birlashadi. Bo`sh joy istalgancha bo`lishi mumkin asosiysi bitta kiritilgan blok chegarasida bo`sh joy bir xil bo`lishi kerak. Noto`g`ri qo`yilgan bo`sh joylar xatolik yuz berishiga olib kelishi mumkin. Bitta probel bilan bo`sh joy hosil qilish yaxshi qaror emas uni o`rniga to`rta probel yoki *Tab* belgisini ishlatish kerak.

  Odatda dasturlash tillarida abzats kodni oson o'qilishi uchun ishlatiladi. Ammo Pythonda abzats kodning blokini ajratib ko'rsatadi. Misol keltiramiz:

```
if 5 > 2:
    print("Besh ikkidan katta")
```

  Agar kodimizni mana bunday tarzda yozsak dasturda xatolik yuz beradi:

```
if 5 > 2:
print("Besh ikkidan katta")
```

- Pythonga kiritilgan amallar bir xil shablonda yoziladi. Bunda asosiy amal ikki nuqta bilan tugatiladi va uning orqasidan kiritilgan blok kodi ham joylashadi. Odatda, asosiy amalning ostidagi satr bo`sh joy bilan ajratiladi.

  Bazan bir nechta amalni bitta satrga nuqtali vergul bilan ajratgan holda yozish mumkin.

```
a = 1; b = 2; print(a, b)
```

Buni ko`p ham qo`llamang! Yaxshisi bunday qilmang, o`qishga noqulay.

Kalit so`zlar

False – yolg`on.

True - rost.

None - “bo`sh” obyekt.

while – while tsikli

with / as – konteks menejeri.

break – sikldan chiqish.

class – metod va atributlarda iborat.

continue – sikldan keyingi iteratsiyaga o`tish.

from – moduldan bir nechta funksiyani import qilish.

import – moduldan import.

is –xotirani bitta joyida 2 ta obyektni jo`natsa bo`ladimi.

if - agar.

else – for/else yoki if/elsega qarang.

elif – aks holda, agar.

for – for sikli.

def – funksiyani aniqlash.

del – obyektni yo`qotish.

not –mantiqiy INKOR amali.

or –mantiqiy YOKI amali.

and – mantiqiy VA amali.

lambda –yashirin funksiyani aniqlash.

Pythonda izohlar

Izohlar quyidagi holatlarda ishlatiladi:

- **Koddagi bajarilayotgan ishlarni tushuntirish uchun;**
- **Kodning o’qilishini yanada osonlashtirish uchun;**
- **Kodning ba’zi qismlarini vaqtincha hisobga olmay turish uchun;**

Izohlarni hosil qilish

Izohlar # belgisi bilan hosil qilinadi va python o’sha qismni kod deb qabul qilmaydi:

```
# Bu yerda izoh
print("Salom dunyo")
```

Izohlarni kod yozilgan qator oxiriga yozish ham mumkin:

```
print("Salom dunyo") #Bu yerda izoh
```

Kodning biror qismini izohga kiritsak o’sha qism natija bermaydi. Quyidagi holatda ***Salom dunyo*** jumlasi ekranga chiqmaydi:

```
# print ("Salom dunyo")
print ("Dasturlashni o'rganamiz")
```

Izohlar dastur kodini o'qiyotganlar uchun foydali bo'ladi va dastur nima qilishini oson tushunishga yordam beradi. Unga yechimdagi muhim joylarni, muhim bo'lgan qismlarni yozish mumkin.

Ko’p qatorli izohlar

Python ko’p qatorli izohlar hosil qilish uchun alohida belgiga ega emas . Shuning uchun har bir qator uchun alohida # belgisi ishlatiladi. Ammo 3 talik qo’shtirnoq ichiga yozilgan matnni o’zgaruvchiga biriktirilmasa ko’p qatorli izoh sifatida ishlatish mumkin:

```
"""
Bu izoh
ko'p qatorli
ozihdir
"""

print("Dasturlashni o'rganamiz")
```

PYTHONDA O`ZGARUVCHILAR

Biror ma'lumotni saqlash va uning ustida turli amallarni bajarish uchun bizga o'zgaruvchilar yordam beradi. O'zgaruvchining qiymati, o'z nomi bilan aytib turibdiki, o'zgarishi mumkin. Unda xohlagan qiymatni saqlash mumkin. O'zgaruvchilar kompyuter xotirasidagi joy bo'lib, u yerda siz biror ma'lumotni saqlaysiz. O'zgaruvchining konstantadan farqi, o'zgaruvchiga dastur ishlashi davomida (***run time***) murojaat qilib, uning qiymatini o'zgartira olamiz. Konstantaga esa oldindan ma'lum bir qiymat beriladi va bu qiymatni o'zgartirib bo'lmaydi.

Pythonda o'zgaruvchilar ularni qiymatlarini tenglashtirish bilan hosil qilinadi. O'zgaruvchilarning turini alohida e'lon qilish shart emas. Pythonda barchasi avtomatik tarzda ishlaydi:

```
x = 5
y = "Salom dunyo"
```

O'zgaruvchilarni hosil qilish

O'zgaruvchilar ma'lum bir turdagi qiymatni o'zida saqlovchi konteynerlardir. Boshqa dasturlash tillaridan farqli, Python o'zgaruvchilarni e'lon qilish uchun alohida buyruqqa ega emas. O'zgaruvchilar ularga qiymatni tenglashtirish orqali hosil qilinadi. Quyida biz 2 ta o'zgaruvchini 2 xil trudagi qiymatga biriktiramiz va ekranga chiqaramiz:

```
x = 5
y = "Python"
print(x)
print(y)
```

O'zgaruvchilarni qaysi turda ekanligni e'lon qilish shart emas. Buni Python avtomatik tarzda aniqlaydi. O'zgaruvchilarning turlarini kodning istalgan qismida o'zgartirish ham mumkin.

```
x = "Dastur"    # x-satr
x = 5           # x endi butun son
print(x)
```

Satrli o'zgaruvchilar qo'shtirnoq yoki bittalik tirnoqlar ichiga yozilish bilan e'lon qilinishi mumin:

```
x = "Dastur"
# ikkala o'zgaruvchi ham bir xil
y = 'Dastur'
```

O'zgaruvchi nomlari

O'zgaruvchilar harflar yoki so'zlar bilan ifodalanishi mumkin. Ularni ifodalash uchun ayrim qoidalar mavjud:

- O'zgaruvchi nomi harf yoki tag chiziq bilan boshlanishi kerak;
- O'zgaruvchi nomi raqam bilan boshlanmasligi kerak;
- O'zgaruvchi nomi faqat harflar, raqamlar va tag chiziqdan iborat bo'lishi mumkin;
- O'zgaruvchi nomlari katta-kichik harflarni farqlaydi (ism, iSm, ISM – bular 3 xil o'zgaruvchilar);
- O'zgaruvchi nomi orasida bo'shliq (probel) bo'lmasligi kerak;

```
# To'g'ri nomlangan o'zgaruvchilar:

myvar  = "Python"
my_var = "Python"
_myvar = "Python"
myVar  = "Python"
MYVAR  = "Python"
myvar2 = "Python"

# Noto'g'ri nomlangan o'zgaruvchilar:

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

Bir nechta o'zgaruvchiga qiymat o'zlashtirish

Pythonda bir nechta o'zgaruvchiga qiymatlarni bir qatorning o'zida o'zlashtirish mumkin:

```
x, y, z = "Olma", "Banan", "Nok"
print(x)
print(y)
print(z)
```

Va aksincha, bir qiymatni bir nechta o'zgaruvchiga o'zlashtirish ham mumkin:

```
x = y = z = "Meva"
print(x)
print(y)
print(z)
```

O'zgaruvchilarni ekranga chiqarish

Pythonda o'zgaruvchilarni yoki natijalarni ekranga chiqarish uchun **print** funksiyasidan foydalaniladi. Biror matnga satr o'zgaruvchisini biriktirish uchun "+" belgisi ishlatiladi:

```
x = "maroqlidir"
print ("Dasturlashni o'rganish "+x)
```

Bundan tashqari **“+”** belgisini o’zgaruvchilarni o’zaro biriktirish uchun ham ishlatsa bo’ladi:

```
x = "Dasturlashni"
y = "o'rganamiz"
print (x+y)
```

Sonli o’zgaruvchilar uchun “+” belgisi matematik amal sifatida ta’sir qiladi:

```
x = 5
y = 10
print (x+y)
```

Satr o’zgaruvchini sonli o’zgaruvchiga qo’shmoqchi bo’lsak **Python** xato yuz berganini ma’lum qiladi:

```
x = 5
y = "besh"
print (x+y)
```

Global o’zgaruvchilar

Funksiyadan tashqarida hosil qilingan o’zgaruvchilar global o’zgaruvchilar hisoblanadi. Global o’zgaruvchilar kodning istalgan qismida (funksiya ichida ham, tashqarisida ham) ishlatilishi mumkin. Quyidagi kodda funksiya tashqarisida o’zgaruvchi hosil qilamiz va uni funksiya ichida ishlatamiz:

```
x = "qiziq"

def funksiya():
    print("Dasturlash juda "+ x)

funksiya()
```

Funksiya ichida hosil qilingan o’zgaruvchi lokal o’zgaruvchi deyiladi. Agar lokal va global o’zgaruvchilarni nomlari bir xil bo’lsa , funksiya ichida lokal o’zgaruvchining qiymati funksiya tashqarisida esa global o’zgaruvchining qiymati olinadi:

```
x = "shirin"

def funksiya():
    x = "foydali"
    print("Olma "+ x)

funksiya()

print ("Olma "+ x)
```

Global kalit so'zi

Oddiy holatda funksiya ichida hosil qillingan o'zgaruvchi lokal o'zgaruvchi hisoblanadi. Ammo funksiya ichida ham global o'zgaruvchi hosil qilish mumkin. Buning uchun ***global*** kalit so'zi ishlatiladi.

```
def funksiya():
    global x
    x = "shirin"
    print("Olma "+x)

funksiya()

print ("Olma "+x)
```

Agar global o'zgaruvchining qiymatini funksiya ichida o'zgartirmoqchi bo'lsangiz ham ***global*** kalit so'zi ishlatiladi:

```
x = "shirin"

def funksiya():
    global x
    x = "foydali"
    print("Olma "+ x)

funksiya()

print ("Olma "+ x)
```

PYTHON OPERATORLARI

Operatorlar o'zgaruvchi va qiymatlar ustida amallar bajarish uchun ishlatiladi. **Python** operatorlari quyidagilar:

- **Arifmetik operatorlar**
- **O'zlashtirish operatorlar**
- **Taqqoslash operatorlari**
- **Mantiq operatorlari**
- **Aniqlash operatorlari**
- **A'zolik operatorlari**
- **Bitli operatorlar**

Arifmetik operatorlar

Arifmetik operatorlar odatiy matematik amallarni bajarish uchun ishlatiladi:

+	**Qo'shish**	**x+y**
-	**Ayirish**	**x-y**
*	**Ko'paytirish**	**x*y**
/	**Bo'lish**	**x/y**
%	**Qoldiqli bo'lish**	**x%y**
//	**Butunli bo'lish**	**x//y**

Ularni amalda sinab ko'rsak yaxshiroq tushunamiz:

```
x = 10
y = 3

print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x % y)
print(x ** y)
print(x // y)
```

```
13
7
30
3.3333333333333335
1
1000
3
```

O'zlashtirish operatorlari

=	x = 5	x=5
+=	x += 3	x = x + 3
- =	x -= 3	x= x - 3
*=	x *= 3	x= x * 3
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x |3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

```
x = 5

x +=3
print(x)

x -=3
print(x)

x *=3
print(x)
```

Taqqoslash operatorlari

Taqqoslash operatorlari qiymatlarni o'zaro taqqoslash uchuyn ishlatiladi:

= =	Teng	x == y
!=	Teng emas	x != y
>	Katta	x > y
<	Kichik	x < y
>=	Katta yoki teng	x >= y
<=	Kichik yoki teng	x <= y

Mantiq operatorlari

Mantiq operatorlar shartlarni birlashtirib ishlatish uchun kerak:

- **and** - Agar ikkala shart ham rost bo'lsa, rost qiymat qaytaradi.
- **or** - Kamida bitta shart rost bo'lsa ham rost qiymat qaytaradi.
- **not** - Shart qiymatini teskarisiga o'zgartiradi, ya'ni rost bo'lsa yolg'on, yolg'on bo'lsa rost bo'ladi.

```
a = 5

print (a>3 and a<10)
print (a>3 or a<4)
print (not(a>3 and a<10))
```

```
True
True
False
```

Aniqlash operatorlari

Aniqlash operatorlari o'zaro 2 ta obyektlarni solishtiradi. Bunda ularning o'zaro qiymatlarini tengligi bo'yicha emas, haqiqatdan ham ular bir xil obyekt ekanligi va bir xil xotira yo'nalishiga ega ekanligi bo'yicha taqqoslanadi. Bu operatorlar 2 ta:

- **is** - Ikkala o'zgaruvchi ham bir xil obyekt bo'lsa rost, aks holda yolg'on qiymat qaytaradi.
- **is not** - Obyektlar bir xil bo'lmasa rost, aks holda yolg'on qiymat qaytaradi.

```
x = ["olma", "banan"]
y = ["olma", "banan"]
z = x


print(x is z)
print(x is y)
print(x == z)

#------------------------------------------------------------------

print(x is not z)
print(x is not y)
print(x != z)
```

```
True
False
True
True
False
True
False
```

A’zolik operatorlari

A’zolik operatorlari biror ketma-ketlik obyektga tegishli ekanligini tekshiradi:

- **in** - Belgilangan qiymat obyektda mavjud bo’lsa, rost qiymat qaytaradi.
- **not in** - Belgilangan qiymat obyektda mavjud bo’lmasa, rost qiymat qaytaradi.

```
x = ["audi", "mustang"]

print("audi" in x)
print("audi" not in x)
```

```
True
False
```

Bitli operatorlar

Bitli operatorlar ikkilik sanoq sistemasi bilan ishlashda kerak bo’ladi:

- **&** **(AND)** - Ikkala bit ham 1 ga teng bo’lsa, 1 ga o’rnatiladi.
- **|** **(OR)** - Kamida bitta bt 1 ga teng bo’lsa, ikkala bitni ham 1 ga o’rnatadi.
- **^** **(XOR)** - Faqat bitta bit 1 ga teng bo’lsa, ikkala bitni ham birga o’rnatadi.
- **~** **(NOT)** - Barcha bitlarni invertlaydi (teskarisiga o’zgartiradi)
- **<<** - O’ngdan chapga nollarni siljitib, chapdagi chetki bo’laklarni tushirib yuboradi.
- **>>** - Chapdan o’ngga bitlarning nusxalari kiritilib siljitib boriladi. O’ngdagi chetki bitlar tushib qoladi.

II-BOB. PYTHONDA MA'LUMOT TURLARI BILAN ISHLASH

PYHONDA MA'LUMOT TURLARI

Dasturlashda ma'lumot turlari muhim tushuncha sanaladi. Har bir ma'lumot turining o'z vazifasi bor. Python quyidagi ma'lumot turlariga ega:

- Matn turi: **str**
- Raqam turi: **int, float, complex**
- Ketma-ketlik turi: **list, tuple, range**
- Ko'rsatish turi: **dict**
- O'rnatish turi: **set, fronzenset**
- Mantiq turi: **bool**
- Binar (ikkilik) turi: **bytes, bytearray, memoryview**

Ma'lumot turini aniqlash

Ma'lumot turini aniqlash uchun **type()** funksiyasi ishlatiladi. Hozirgi misolda x o'zgaruvchisining turini ekranga chiqaramiz:

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

Ma'lumot turlarini o'rnatish

O'zgaruvchiga qiymatni o'zlashtirgan vaqtda uning ma'lumot turini avtomatik tarzda aniqlab uni o'zlashtiradi. Natijada o'zgaruvchi o'sha ma'lumot turini o'zida saqlaydi:

x = "Python" --- **str** (satr turi)
x = 20 --- **int** (butun son turi)
x = ["olma", "banan", "nok"] --- **list** (ro'yxat turi) va hokazo.

Aniq ma'lumot turini o'rnatish

Agr ma'lumot turini aniq ko'rsatmoqchi bo'lsangiz, bu ishni quyidagicha amalga oshirish kerak:

x = **str**("Python") --- **str** (satr turi)
x = **int**(20) --- **int** (butun son turi)
x = **list**(["olma", "banan", "nok"]) --- **list** (ro'yxat turi)

PYTHONDA SONLAR

Pythonda sonli turlar 3 turga bo'linadi:

- **Int**
- **Float**
- **Complex**

Quyidagi misolda 3 xil sonli o'zgaruvchi hosil qilamiz va ularning turlarini ekranga chiqaramiz:

```
x = 1
y = 2.8
z = 1j

print(type(x))
print(type(y))
print(type(z))
```

Consolda yuqoridagi kod bizga quyidagi natijani beradi:

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

Int (butun sonlar)

Int (integer) turidagi sonlar o'z ichiga istalgan oraliqdagi musbat yoki manfiy butun sonlarni oladi:

```
x = 1
y = 345699247453245
z = -2344699247

print(type(x))
print(type(y))
print(type(z))
```

Python interpretatorida yuqorida operator va ifodalar mavzusida ko`rib chiqqan barcha operatorlarni oddiy matemetika kursida ishlatilganidek bajarilishini ko`rdik. Ya'ni ko`paytirish, qo`shish, ayirish, bo'lish, darajaga ko`tarish va hokazo. Endi esa butun sonlar ustida bajarish mumkin bo`lgan qo`shimcha metodlarni ko`ramiz.

int.bit_length() - sonni oldidagi ishora va nollarni hisobga olmasdan uni ikkilik sanoq sistemasida taqdim etish uchun kerakli bo`lgan bitlar soni.

```
>>> bin(n)
'-0b100101'
>>> n.bit_length()
6
>>>
```

int.to_bytes(length, byteorder, *, signed=False) -shu sonni taqdim etuvchi baytlar qatorini qaytaradi.

```
>>> (1024).to_bytes(2, byteorder='big')
b'\x04\x00'
>>> (1024).to_bytes(10, byteorder='big')
b'\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00'
>>> (-1024).to_bytes(10, byteorder='big', signed=True)
b'\xff\xff\xff\xff\xff\xff\xff\xff\xfc\x00'
>>> x=1000
>>> x.to_bytes((x.bit_length() // 8) + 1, byteorder='little')
b'\xe8\x03'
```

classmethod **int.from_bytes(bytes, byteorder, *, signed=False)**-berilgan baytlar qatoriga mos sonni qaytaradi.

```
>>> int.from_bytes(b'\x00\x10', byteorder='big')
16
>>> int.from_bytes(b'\x00\x10', byteorder='little')
4096
>>> int.from_bytes(b'\xfc\x00', byteorder='big', signed=True)
-1024
>>> int.from_bytes(b'\xfc\x00', byteorder='big', signed=False)
64512
>>> int.from_bytes([255, 0, 0], byteorder='big')
16711680
```

Float (haqiqiy sonlar)

Float turidagi sonlar o'z ichiga manfiy yoki musbat o'nli kasr ko'rinishidagi sonlarni oladi:

```
x = 1.10
y = 10.0
z = -38.54

print(type(x))
print(type(y))
print(type(z))
```

Haqiqiy sonlar ham butun sonlar qo`llab quvvatlovchi operatsiyalarni qo`llab quvvatlaydi. Haqiqiy sonlar ustida amal bajarishda foydalanish mumkin bo`lgan qo`shimcha metodlar:

- **float.as_integer_ratio**- shu haqiqiy son bilan juftlik munosabatida bo`lgan butun son.
- **float.is_integer()**- ko`rsatgich butun son bo`lish bo`lmasligini tekshiradi.
- **float.hex()**-float ni hex ga (o`n oltilik sanoq sistemasiga) o`tkazadi.
- classmethod **float.fromhex(s)**- o`n oltilik sanoq sistemasisdan floatga otkazadi. Ya'ni **float.hex()** ni teskarisi.

```
>>> (12.9).is_integer()
False
>>> (13.0).is_integer()
True
>>> (13.0).as_integer_ratio()
(13, 1)
>>> (10.5).hex()
'0x1.5000000000000p+3'
>>> float.fromhex('0x1.5000000000000p+3')
10.5
```

Complex (kompleks sonlar)

Xuddi matamatika sohasidagi kompleks sonlarni Pythonda ham ishlatish mumkin:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

Pythonda komleks sonlar ustida arifmetik amallarni butun va haqiqiy sonlar ustida bajarilgani kabi oddiy bajarish mumkin yani matematika kursida komleks sonlar ustida arifmetik amallar qanday bajarilsa xuddi shunga o`xshab bajariladi.

```
>>> x=complex(1,2)
>>> print(x)
(1+2j)
>>> y=complex(3,4)
>>> print(y)
(3+4j)
>>> z=x+y
>>> print(z)
(4+6j)
>>> z=x*y
>>> print(z)
(-5+10j)
>>> z=x/y
>>> print(z)
(0.44+0.08j)
>>> print(x.imag)# Mavhum qismi chiqaradi
2.0
>>> print(x.real)# Haqiqiy qismni chiqaradi
1.0
```

Sonlarni o'girish

Sonlarni bir turdan boshqasiga osongina o'girish mumkin. Buning uchun **int(), float(), complex()** buyruqlari ishlatiladi:

```
x = 1    #int
y = 2.8  #float
z = 1j   #complex

# int turidan floatga o'tkazish
a = float(x)

# float turidan intga o'tkazish
b = int(x)

# int turidan complexga o'tkazish
c = complex(x)

print(a)
print(b)
print(c)
```

Consolda yuqoridagi kod bizga quyidagi natijani beradi:

```
1.0
1
(1+0j)
```

Tasodifiy son (random moduli)

Tasodifiy sonni hosil qilish ichun Pythonda **random** buyrug'i kiritilgan. Hozir 1 dan 9 gacha bo'lgan sonlar oralig'idan tasodifiy sonni ekranga chiqaruvchi dasturni yaratamiz:

```
import random

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

Bu modul har xil taqsimotlar uchun tasodifiy raqamlarni generatsiya qiladi. Eng ko`p qo`llaniladigan funksiyalari:

- **Random()** -[0.0, 1.0) yarim ochiq diapozondagi tasodifiy sonlarni generatsiya qiladi.
- **Choice(s)** - s ketma- ketlikdan tasodifiy elementni tanlab oladi.
- **Shuffle(s)** - s o`zgaruvchan ketma-ketlik elementlarini joyiga joylashtiradi.
- **Randrange([start], stop, [step])** - renge(start, stop, step) diapozondagi tasodifiy butun raqamni chiqaradi. Choice(range(start, stop, step)) ga analogik holatda.
- **Normalvariate(mu, sigma)** - normal holatda taqsimlangan ketma-ketlikdan raqamni chiqaradi. Bu yerda mu- o`rtacha, sigma-o`rta kvadratli (sigma>0) sonlar.

Boshqa funksiyalar va uning parametrlarini hujjatlashdan aniqlab olish mumkin. Modulda qandaydir holatga tasodifiy raqamlar generatorini joylashtirishga imkon beruvchi **seed(n)** funksiyasi ham mavjud. Masalan: agarda bitta tasodifiy raqamlar ketma-ketligidan ko`p marta foydalanishga ehtiyoj sezilsa.

PYTHONDA SATRLAR

Satrlar – bu belgilar ketma-ketligi. Ko'p hollarda satrlar so'zlar jamlanmasidan tashkil topadi. Pythonda satrlar bilan ishlash juda qulay. Bir qancha satr literallari mavjud. Pythonda satrlar qo'shtirnoq yoki bittalik tirnoqlar bilan ifodalanadi. Ularni **print()** funksiyasi bilan ekranga chiqaramiz.

```
print ("Salom")
print ('Python')
```

Apostrof va qo`shtirnoqdagi satrlar bir narsa. Uni ikki xil variantda keltirilishiga sabab literallarga apostrof va qo`shtirnoq belgilarini maxsus xizmatchi belgilardan foydalanmasdan kiritish mumkinligi deb hisoblanadi.

```
Ism = "San'atbek"

Gap = 'Men "Python dasturlash tili"  nomli kitob yozdim'
```

Satrni o'zgaruvchiga biriktirish

Satrni o'zgaruvchiga biriktirish uchun tenglik belgisi ishlatiladi:

```
a = "Salom"
print(a)
```

Ko'p qatorli satr

Ko'p qatorli satrni hosil qilish uchun uchtalik qo'shtirnoq yoki tirnoqlardan foydalaniladi. Bunda satr qanday holatda yozilgan bo'lsa shundayligicha natija beradi:

```
a = """ Bu
ko'p qatorli
satr """

b = ''' Bu ham
ko'p qatorli
satr '''

print(a)
print(b)
```

Satr konstantalarini birlashtirish uchun ularni yonma-yon joylashtirishning o'zi kifoya. Python avtomat ularni birlashtiradi. Misol uchun: "Ismingiz" "kim?" avtomat "Ismingiz kim?" ga aylanadi.
Eslatma: Bir tirnoq va qo'sh tirnoqdagi satrlar bir-biridan hech ham farq qilmaydi.

Satr – bu massiv

Satrni alohida belgilar ketma-ketligidan tashkil topgan massiv deb hisoblash mumkin. Pythonda belgini ifodalovchi ma’lumot turi yo’qligi uchun bitta belgi deb, bitta elementdan tashkil topgan satrga aytiladi. Satrning istalgan elementiga murojaat qilish mumkin. Buning uchun kvadrat qavslardan foydalanamiz va elementning satrdagi o’rnini kiritamiz.

Eslatma: Elementning satrdagi o’rnini ifodalash uchun sanoq o dan boshlanadi. Ya’ni satrdagi birinchi elementning o’rni 0 ga, ikkinchi elementning o’rni 1 ga teng va hokazo.

Quyidagi misolimizda biz ikkinchi elementni ekranga chiqaramiz:

```
a = "Dasturlash"
print(a[1])
```

Satrlar ustida amallar

Shunday qilib satrlar bilan ishlash haqida gapirdik, endi satrlarning funksiyalari va metodlari haqida gapiramiz. Quyida satrlarning barcha funksiya va metodlari keltirilgan.

- **Konkatenatsiyalash** (qo`shish)

```
>>> s1='spam'
>>> s2='eggs'
>>> s1+s2
'spameggs'
```

- **Satrni takrorlash** (dublikat qilish)

```
>>> print('dunyo'*3)
dunyodunyodunyo
```

- **Satr uzunligi** (len() funksiyasi)

```
>>> gap='bu satrning uzunligi qancha'
>>> len(gap)
27
```

- **Indeks bo`yicha chiqarish**

```
>>> s='spam'
>>> s[0]
's'
>>> s[2]
'a'
>>> s[-2]
'a'
```

- Misoldan ko`rinib turibidiki Python manfiy indeks bo`yicha chiqarishga ruxsat etadi, lekin hisoblash qator oxiridan boshlanadi. Satrdan qism ajratib olishni uning oxiridan boshlash uchun manfiy indeks ishlatiladi.

- **Satrdan qism ya'ni kesma ajratib olish.** Satrdan bir nechta elementdan tashkil topgan qismini ajratib olish mumiin. Buning uchun ajratilayogan qismining boshlang'ich elementining satrdagi o'rni va oxirgi elementining satrdagi o'rni olinadi. Ularni bildirgan sonlar orasiga: belgisi qo'yilgan holda kvadrat qavs ichiga yoziladi.
  Kesmani ajratib olish operatori:[**X:Y**].
  X- kesmaning boshi, **Y** esa –oxiri. Y raqamli belgi kesmaga kirmaydi. Jimlik holatida birinchi indeks 0 ga teng, ikkinchi indeks esa qator uzunligiga teng bo`ladi.

```
>>> s='spameggs'
>>> s[3:5]
'me'
>>> s[2:-2]
'ameg'
>>> s[:6]
'spameg'
>>> s[1:]
'pameggs'
>>> s[:]
'spameggs'
```

Bundan tashqari kesmani ajratib olishda qadamni belgilash mumkin:

```
>>> s[::-1]
'sggemaps'
>>> s[3:5:-1]
''
>>> s[2::2]
'aeg'
```

Satrga oid funksiyalar

Pythonda satr bilan ayrim amallarni bajarish uchun maxsus funksiyalar bor. Ularning soni ancha ko'p, hammasini eslab qolish esa qiyin. Shuning uchun kerakli fuksiyani manbadan tanlab foydalangan ma'qul. Hozir esa shulardan ba'zilarini ko'rib chiqamiz.

Metodlarni chaqirganga Pythondagi satrlar o`zgarmaydigan ketma-ketliklar darajasiga kirishini inobatga olishimiz kerak. Bu degani hamma funksiyalar va metodlar faqat yangi satrni tuzishi mumkin.

```
>>> s='spam'
>>> s[1]='b'
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    s[1]='b'
TypeError: 'str' object does not support item assignment
>>> s=s[0]+'b'+s[2:]
>>> s
'sbam'
```

Shuning uchun hamma metodlar yangi satrni qaytaradilar, va u keyin boshqa nomga ega bo`ladi.

S = ‘str’; S = “str”; S = ‘’‘str’‘’; S = “””str”””- Satrlarni literallari

S = “s\np\ta\nbbb” - ekran bilan ishlash ketma-ketliklari

S = r”C:\temp\new” - Formatlashtirilmagan satrlar

S = b”byte” - Baytlar qatori

S1+S2 - Konkatenatsiya (qo`shish)

S1*3 - Satrni takrorlash

S[i] - Indeks bo`yicha murojaat

S[i:j:step] - Step qadamli i elementdan boshlab j elementgacha bo`lgan kesmani ajratib olish.

len()- Satr uzunligini hisoblaydi. Bunda har bir harf, belgi, xatto bo’shliq ham hisobga olinadi.

```
>>> gap='bu satrning uzunligi qancha'
>>> len(gap)
27
```

S.find(str,[start],[end])- Satrdan satr ostini izlash. Satr ostining birinchi belgisini o`rinini qaytaradi, agar satrda satr osti bo`lmasa -1 ni qaytaradi.

```
>>> s='salom bu Python dasturi'
>>> s.find('Python',1, 5)
-1
>>>
>>> s.find('Python',1, 50)
9
```

S.rfind(str,[start],[end])- Satrdan satr ostini axtarish. Oxirgi kirish raqamini yoki 1 ni qaytaradi

S.index(str,[start],[end])- Satrdan satr ostini axtarish. Birinchi kirish raqamini qaytaradi yoki ValueError istisnosini chaqiradi

S.rindex(str,[start],[end])- Satrdan satr ostini axtarish. Oxirgi kirish raqamini qaytaradi yoki ValueError istisnosini chaqiradi.

S.isdigit() - Satrda raqamlar ishtirok etganligini tekshirish.

S.isalpha() - Satr faqat harflardan iboratligini tekshirish.

S.isalnum() - Satr harf yoki raqamlardan iboratligini tekshiradi.

S.islower() - Satr quyi registrdagi belgilardan iboratligini tekshiradi**.**

S.isspace()-Satrda ko`rinmaydidan belgilar borligini tekshirish (probel, sahifani o`tkazish belgisi(‘\p’), yangi satrga o`tish(‘\n’), koretkani qaytarish(‘\r’), gorizontal tabulyatsiya(‘\t’) va vertikal tabulyatsiya)

S.istitle() - Satrda so`zlar bosh harf bilan boshlanishini tekshirish.

S.starswith(str)- S satr str shablonidan boshlanishini tekshirish.

S.endswith(str)- S satr str shabloni bilan tugashini tekshirish.

**S.join(ro`yxat)-** S ajratuvchiga ega ro`yxatdan qatorni yig`ish.

Ord(belgi)- Belgiga mos ASCII kodni qaytaradi.

Chr(son)- ASCII kodga mos belgini qaytaradi.

S.capitalize()-Satrning birinchi belgisi yuqori registrda qolganlarini quyi registrga o`tkazadi.

S.center(width,[fill])- Chegaralari bo`yicha fill (jimlik holatida probel) belgisi turuvchi markazlashtirilgan satrni qaytaradi.

S.expandtabs(tabsize)- Joriy ustungacha bir yoki bir qancha probellar bilan tabulyatsiyaning hamma belgilari almashtirilgan satr nusxasini qaytaradi. Agarda TabSize ko`rsatilmagan bo`lsa tabulyatsiya hajmi 8 probelga teng bo`ladi.

S.lstrip([chars])- Satr boshidagi probel belgilarini olib tashlash.

S.rstrip([chars]) - Satr oxiridan probel belgilarini olib tashlash.

S.strip([chars]) - Satr boshidan va oxiridan probel belgilarini olib tashlash.

S.partition(shablon) - Birinchi shablon oldida turuvchi qismni keyin shablonni o`zini va shablondan keyin turuvchi qismga ega kortejni qaytaradi. Agarda shablon topilmasa satrga ega bo`lgan kortejni qaytaradi, avval ikki bo`sh satr keyin satrni o`zini.

S.rpartition(sep)- Oxirgi shablon oldida turuvchi qismni keyin shablonni o`zini va shablondan keyin turuvchi qismni qaytaradi. Kortej qator o`zidan va undan keyin ikkita bo`sh qatordan iborat bo`ladi.

S.swapcase()-Quyi registrdagi belgilarni yuqori registrga, yuqorilarni esa quyiga o`tkazadi.

S.title() - Har bitta so`zning birinchi harfini yuqori registrga qolganlarini esa quyi registrga o`tkazadi.

S.zfill(width) - Qator uzunligini Widthdan kam qilmaydi agar kerak bo`lsa birinchi belgilarni nollar bilan to`ldiradi.

S.strip() funksiyasi satrning boshi va oxirida bo’shliqlar bo’lsa, olib tasdiqlaydi:

```
a = " Dastur "
print(a.strip())
```

S.lower() funksiyasi satrdagi barcha so’zlarni kichik harf bilan yozilishini ta’minlaydi:

```
a = "Markaziy Osiyo"
print(a.lower())
```

S.upper() funksiyasi satrdagi barcha so’zlarni katta harflar bilan yozadi:

```
a = "Markaziy Osiyo"
print(a.upper())
```

S.replace() funksiyasi satrdagi bir so'z yoki harfni boshqasi bilan almashtiradi:

```
a = "Bor"
b = "Markaziy Osiyo"

print (a.replace("r", "y"))
print (b.replace("Markaziy", "O'rta"))
```

```
Boy
O'rta Osiyo
```

S.split() funksiyasi satrni ko'rsatilgan belgi bo'yicha qismlarga bo'lib chiqadi. Masalan, vergul ko'rsatilsa satrdagi har bir vergulgacha bo'lgan so'z yoki harflar bitta alohida qism deb olinadi. Yoki probel (bo'shliq) ko'rsatilsa har bir probelgacha bo'lgan qismi ajratiladi.

```
a = "Python bilan ishlash qiziqarli"
print(a.split(" "))
```

```
['Python', 'bilan', 'ishlash', 'qiziqarli']
```

Satrlarni tekshirish

Aniq bir jumla yoki harf(belgi) satrda bor yoki yo'qligini **in** yoki **not** kalit so'zlari bilan tekshirish mumkin. Bunday holatda qidirilaytogan jumla bor bo'lsa **True** (rost) , aks holda **False** (yolg'on) qiymati qaytariladi. Quyidagi kodimizda "ol" jumlasi borligini tekshirib ko'ramiz:

```
txt = "Olmaxon yerdagi olmani olib ketdi"
x = "ol" in txt
print(x)
```

Endi satrda "ol" jumlasi yo'qligini tekshiramiz. Bu yerda "ol" jumlasi satrda borligi uchun **False** (yolg'on) qiymati qaytariladi:

```
txt = "Olmaxon yerdagi olmani olib ketdi"
x = "ol" not in txt
print(x)
```

Satrlar formati

Biz satr va sonli o'zgaruvchilarni birgalikda to'g'ridan to'g'ri ishlata olmaymiz. Satr ichida sonli o'zgaruvchini qo'llash uchun **format()** funksiyasidan foydalaniladi. Bu funksiya sonli qiymatni olib satrli o'zgaruvchiga aylantiradi va **{}** belgisi qo'yilgan joy o'rnida o'sha qiymatni joylashtiradi.

```
yosh = 21
matn = "Mening yoshim {} da"

print(matn.format(yosh))
```

format() funksiyasi bilan istalgancha sonli qiymatlarni bir satrga joylash mumkin. Uning o'zi qiymatlarni tartib bo'yicha tegishli joylarga qo'yib chiqadi:

```
raqam = 2
kilo = 3
pul = 5000

savdo = "{}-do'kondan {} kg olmani {} so'mga sotib oldim"

print(savdo.format(raqam, kilo, pul))
```

```
2-do'kondan 3 kg olmani 5000 so'mga sotib oldim
```

Qiymatlarni joylashtirish tartibini o'zingiz aniq belgilab bermoqchi bo'lsangiz indeks sonlaridan foydalanishingiz kerak bo'ladi. Eslatib o'tamiz dasturlashda sanoq 0 dan boshlanadi:

```
raqam = 2
kilo = 3
pul = 5000

savdo = "{1} kg olmani {0}-do'kondan {2} so'mga sotib oldim"

print(savdo.format(raqam, kilo, pul))
```

```
3 kg olmani 2-do'kondan 5000 so'mga sotib oldim
```

Satrlarni formatlashni yana bir necha xil usullari bor. Ular bilan keyingi dasrda tanishamiz.

Maxsus belgilar

Ekran bilan ishlash ketma-ketliklari- klaviatura yodamida kiritish murakkab bo`lgan belgilarni yozishga imkon beradi. Ayrim belgilar borki, ularni satr ichida to'g'ridan to'g'ri qo'llab bo'lmaydi. Ularni satr ichida qo'llanganda doimo \ (backslash) belgisi ham bo'lishi shart.
Masalan, qo'shtirnoqni satr ichida shunchaki qo'llasak, dasturda xatolik yuz beradi:

```
txt = "Akam bilan "Paxtakor" metrosida ko'rishamiz"
print(txt)
```

Dasturda xatolik bo'lmasligi uchun qo'shtirnoqni \" ko'rinishida belgilaymiz.

```
txt = "Akam bilan \"Paxtakor\" metrosida ko'rishamiz"
print(txt)
```

Satr ichida qo'llanadigan boshqa maxsus belgilardan namuna:

- \' bittalik qo'shtirnoq
- \\ backslash belgisi
- **\n** yangi qatorga o'tish
- **\t** tabulyatsiya (so'zni bir harf kengligida surish)

Xizmatchi belgilar	Vazifasi
\n	Keyingi qatorga o`tish
\a	Qo`ng`iroq
\f	Keyingi betga o`tish
\r	Koretkani qaytarish
\t	Gorizontal tabulatsiya
\v	Vertical tabulatsiya
\N{id}	Unicode ma’lumotlar bazasining ID identifikatori
\uhhhh	Unicode ning 16 lik ko`rinishidagi 16 bitli belgisi
\Uhhhh. . .	Unicode ning 32 lik ko`rinishidagi 32 bitli belgisi
\xhh	Belgining 16 lik kodi
\ooo	Belgining 8 lik kodi
\0	Null belgisi (satr oxiri belgisi emas)

SATRLARNI FORMATLASH

Satrni formatlash **format()** funksiyasi bilan amalga oshiriladi. Bu narsa bizga satr ichiga qiymatlarini joylashtirsh uchun kerak bo’lgan joyga maxsus qavslar qo’yiladi va **format()** funksiyasi bilan kerakli qiymat joylashtiriladi.

```
narx = 30
satr = "Mahsulot narxi {} so'm"
print(satr.format(narx))
```

```
Mahsulot narxi 30 so'm
```

Ko’proq qiymatlarda formatlash

Satr ichiga ko’proq qiymatlarni ham joylashtirsa bo’ladi. Maxsus qavslar ham shuncha bo’lishi kerak:

```
sana = 5
oy = "avgust"
yil = 2020
bugun = "Bugun {} - {}, {} - yil"

print(bugun.format(sana, oy, yil))
```

```
Bugun 5 - avgust, 2020 - yil
```

Indeks raqamlari orqali formatlash

Qiymatlar to'g'ri joylashishiga amin bo'lishi uchun ularning tartibini indeks bilan ko'rsatsa bo'ladi:

```
sana = 5
oy = "avgust"
yil = 2020
bugun = "Bugun {0} - {1}, {2} - yil"

print(bugun.format(sana, oy, yil))
```

```
Bugun 5 - avgust, 2020 - yil
```

Yoki bir qiymatni takror ishlatish uchun ham indeks soni raqami bilan unga murojaat qilamiz:

```
soat = 3
fan = "Dasturlash"
dars = "Bugun {0} soat darsimiz bor. {0} - darsimiz {1}"

print(dars.format(soat, fan))
```

```
Bugun 3 soat darsimiz bor. 3 - darsimiz Dasturlash
```

Nomli indekslar

Indekslarni raqamlab ishlatishdan tashqari, ularni nomlab ishlatsak ham bo'ladi va bu quyidagicha amalga oshadi:

```
satr = "Uning ismi {ism}, yoshi {yosh} da"

print(satr.format(ism = "Abbosbek", yosh = 20))
```

```
Uning ismi Abbosbek, yoshi 20 da
```

MA'LUMOT TO'PLAMLARI VA TURLARI

Pythonda ma'lumot to'plamlarining turlari 4 xil. Ulardan odatda bir nechta yoki undan ham ko'p qiymatlarni saqlashda foydalanamiz. Bizga kerakr bo'lganda o'sha to'plamlarga murojaat qilib tegishli qiymatlarni olamiz.

Har bir ma'lumot to'plamining o'z xususiyatlari bor va shunga ko'ra ularni kerakli joyda tanlab ishlatamiz. Darsimiz davomida barcha turlarning xususiyatlarini ko'rib chiqamiz:

- **List** – tartiblangan va o'zgaruvchan ro'yxat. Elementlarini dublikatlash mumkin.
- **Tuple** – tartiblangan va o'zgarmas ro'yxat. Elementlarini dublikatlash mumkin.
- **Set** – Tartiblanmagan va indekslanmagan to'plam. Elementlari dublikatlanmaydi.
- **Dictionary** – tartiblanmagan,o'zgaruvchan va indekslangan to'plam. Elementlari dublikatlanmaydi.

Yuqoridagi xususiyatlardan kelib chiqqan holda tegishli joylarda qo'llaniladi. Ularni birma-bir ko'rib keyingi mavzularda ko'rib chiqamiz.

LIST (RO'YXAT)

List- Pythonda erkin turdagi obyektlarning o`zgaruvchan qatorlashgan kolleksiyasi hisoblanadi (*massivga o`xshash, lekin tiplar har xil bo`lishi mumkin*). Ro`yxatlardan foydalanish uchun ularni tuzish kerak. List – aytib o'tganimizdek tartiblangan va o'zgaruvchan ro'yxat. Ro`yxatni har xil yondashuvlar yordamida yaratish mumkin. Uni kvadrat qavslar bilan hosil qilamiz:

```
mashina = ["Audi", "Mustang", "Ferrari"]
print(mashina)
```

list() konstruktori

List ro'yxatini **list()** konstruktori yordamida hosil qilish mumkin. Bunday holatda kvadrat qavslar ishlatilmaydi:

```
meva = list(("olma", "banan", "apelsin", "nok", "uzum"))
print(meva)
```

Elementlarga murojaat

List elementlariga murojaat qilish uchun, murojaat qilinayotgan elementning indeksi ko'rsatiladi. Sanoq har doimgidek 0 dan boshlanadi. Quyidagi kodimiz isga tushsa, ekranga ikkinchi element chiqadi:

```
mashina = ["Audi", "Mustang", "Ferrari"]
print(mashina[1])
```

```
Mustang
```

Manfiy indeks

Manfiy indeks sanoq oxiridan boshlanishini bildiradi. Masalan, -1 eng oxirgi, -2 oxiridan ikkinchi element va hokazo.

Quyidagi dasturimiz ishga tushsa, oxirgi element ekranga chiqadi:

```
mashina = ["Audi", "Mustang", "Ferrari"]
print(mashina[-1])
```

```
Ferrari
```

Indeks oralig'i

Ro'yxatning ma'lum bir qismidagi bir nechta elementni tanlab olish uchun o'sha indekslar oralig'ini kiritamiz. Bunda uning boshlanish va oxirgi nuqtalari kiritiladi. Element tanlashda oxirgi nuqta hisobga kirmaydi. Ya'ni boshlang'ich nuqtadan boshlanib oxirgi nuqtadan bitta oldingi elementgacha olinadi. Hozir biz ro'yxatdan ikkinchi, uchinchi va to'rtinchi elementlarni tanlab olamiz:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]
print(meva[1:4])
```

```
['banan', 'apelsin', 'nok']
```

Agar indekslar oralig'ida boshlang'ich nuqtani olib tashlasak, tanlash ro'yxat boshidan boshlanadi. Agar oxirgi nuqatani olib tashlasak, tanlash ro'yxat oxirigacha davom etadi. Quyidagi kodimizda avval ro'yxat boshidan uchinchi elementgacha , so'ngra, ikkinchi elementdan ro'yxat oxirigacha bo'lgan elementlarni ekranga chiqaramiz:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]
print(meva[:4])
print(meva[1:])
```

```
['olma', 'banan', 'apelsin', 'nok']
['banan', 'apelsin', 'nok', 'uzum']
```

Element qiymatini o'zgartirish

List ro'yxatidagi istalgan element qiymatini o'zgartirish mumkin. Buning uchun uning indeksi orqali murojaat qilib, yangi qiymatni biriktiramiz. Hozir ro'yxatdagi birinchi elementni o'zgartiramiz:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]
meva[0]="anor"
print(meva)
```

```
['anor', 'banan', 'apelsin', 'nok', 'uzum']
```

Ro'yxat bo'ylab sikl

Ro'yxatdagi elementlarni for siklidan foydalanib ham tanlab olish mumkin. **For** sikli haqida batafsil alohida mavzuda bilib olasiz. Hozir esa bu sikl bilan elementlarni qanday ekranga chiqarishni ko'rib oling:

```
mashina = ["Audi", "Mustang", "Ferrari"]

for x in mashina:
    print(x)
```

```
Audi
Mustang
Ferrari
```

Elementning mavjudligini tekshirish

Biror elementning ro'yxatda mavjudligini tekshirish uchun in operatoridan foydalaniladi. Hozir ro'yxatda nok borligini tekshiramiz:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]

if "nok" in meva:
    print("Ha, nok bor")
else:
    print("Nok yo'q")
```

Ro`yxatning funksiya va metodlari

Ro`yxatni yaratgandan so`ng uning ustida turli amallarni bajarish kerak bo`ladi, albatta, buning uchun esa Pythonni o`ziga kiritilgan bir qancha funksiya va metodlar bor.

Metod	Vazifasi
List.append(x)	Ro`yxat oxiridan element qo`shish
List.extend(L)	Oxiriga hamma elementlarni qo`shib list ro`yxatini kengaytiradi.
List.insert(i,x)	i-elementga x qiymatini kiritadi
List.remove(x)	Ro`yxatdan x qiymatga ega elementni o`chiradi
List.pop([i])	Ro`yxatning i-elementini o`chiradi va qaytaradi. Agarda indeks ko`rsatilmagan bo`lsa oxirgi element o`chiriladi
List.index(x,[start],[end])	X qiymatga teng start dan end gacha birinchi elementni qaytaradi
List.count(x)	X qiymatga teng elementlar sonini qaytaradi
List.sort([key=funksiya])	Funksiya asosida ro`yxatni saralaydi
List.reverse()	Ro`yxatni ochadi
List.copy()	Ro`txatning nusxalaydi
List.clear()	Ro`yxatni tozalaydi

Keling **list** ya’ni ro`yxatda metodlarni qo`llanilishini misollar yordamida ko`rib chiqamiz.

Ro’yxat uzunligi

Ro’yxatda nechta element borligini aniqlash uchun **len()** funksiyasi ishlatiladi.

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]

print(len(meva))
```

Element qo’shish

append() funksiyasi bilan ro’yxat oxiridan yangi element qo’shish mumkin:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]

meva.append("anor")
print(meva)
```

```
['olma', 'banan', 'apelsin', 'nok', 'uzum', 'anor']
```

Agar elementni ro'yxat oxiriga emas, balki uning ma'lum bir o'rniga qo'shmoqchi bo'lsak **insert()** funksiyasini ishlatamiz. Buning uchun qo'shmoqchi bo'lgan o'rnimizning indeksi ham kiritiladi. Masalan hozir ro'yxatning boshiga yangi elemetni qo'shamiz:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]

meva.insert(0, "anor")
print(meva)
```

```
['anor', 'olma', 'banan', 'apelsin', 'nok', 'uzum']
```

Elementni o'chirish

Ro'yxatdan elementni o'chirishning bir nechta usullari bor.

remove() funksiyasi belgilangan elementni ro'yxatdan o'chiradi. Bunda uning indeksi emas balki o'zi ko'rsatiladi:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]

meva.remove("banan")
print(meva)
```

```
['olma', 'apelsin', 'nok', 'uzum']
```

pop() funksiyasi ko'rsatilan indeks bo'yicha elementni ro'yxatdan o'chiradi. Agar indeks ko'rsatilmasa avtomatik tarzda ro'yxat oxiridagi elementni o'chiradi:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]

meva.pop()
print(meva)
```

```
['olma', 'banan', 'apelsin', 'nok']
```

del kalit so'zi bilan ko'rsatilgan indeks bo'yicha element ro'yxatdan o'chiriladi. Agar shunchaki ro'yxat nomi ko'rsatilsa, butun ro'yxat o'chiriladi. Hozir misolimizda, avvalo, bir elementni o'chiramiz, so'ngra ro'yxatning o'zini o'chiramiz:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]

del meva[1]
print(meva)

del meva
print(meva)
```

clear() funksiyasi ro'yxat elementlarini tozalaydi, ya'ni ro'yxat bo'm-bo'sh bo'lib qoladi:

```
meva  = ["olma", "banan", "apelsin", "nok", "uzum"]

meva.clear()
print(meva)
```

```
[]
```

Ro'yxatdan nusxa olish

Bir ro'yxatdan ikkinchi ro'yxatni **list2 = list1** tarzida hosil qilib bo'lmaydi. Chunki bunda **list2 list1** ga yo'llanma(silka) bo'lib qoladi. Shu sababli **list1** da bo'lgan o'zgarishlar **list2** ga ham ta'sir qiladi. Shuning uchun bir ro'yxat ikkinchisiga nusxalanadi. Shunda 2 ta bir xil alohida ro'yxatlar hosil bo'ladi.

Ro'yxatdan nusxa olish uchun **copy()** funksiyasi ishlatiladi.

```
meva1  = ["olma", "banan", "apelsin", "nok", "uzum"]

meva2 = meva1.copy()
print(meva2)
```

```
['olma', 'banan', 'apelsin', 'nok', 'uzum']
```

Ro'yxatdan nusxa olishning boshqa usuli **list()** funksiyasi:

```
meva1  = ["olma", "banan", "apelsin", "nok", "uzum"]

meva2 = list(meva1)
print(meva2)
```

Ro'yxatlarni qo'shish

Pythonda ikki yoki undan ko'p ro'yxatlarni o'zaro qo'shishning turli usullari bor. Eng oson yo'li "+" operatoridan foydalanish.

Shuni eslatish lozimki, ro'yxat nafaqat satr va harflar, baki sonli o'zgaruvchilardan ham iborat bo'la oladi:

```
a = [1, 2, 3, 4, 5]
b = [5, 6, 7]

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

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

Bir ro'yxatga boshqasini qo'shishning yana bir yo'li – ikkinchi ro'yxatning elementlarini bittalab qo'shib chiqish:

```
mashina1 = ["Audi", "Mustang", "Ferrari"]
mashina2 = ["BMW", "MErcedes", "Porsche"]

for x in mashina2:
    mashina1.append(mashina2)

print(mashina1)
```

extend() funksiyasi ham bir ro'yxatdagi elementlarni ikkinchisiga qo'shib chiqadi. Qo'shilayotgan elementlar avtomatik tarzda ro'yxat oxiridan boshlab qo'shiladi.

```
a = [1, 2, 3, 4, 5]
b = [5, 6, 7]

a.extend(b)
print(a)
```

count() va index()

- **count()** funksiyasi belgilangan qiymatga teng elementlar sonini aniqlaydi.
- **index()** funksiyasi belgilangan elementning indeksini aniqlaydi. Agar bunday elementlar bir nechta bo'lsa, faqat birinchisining indeksini aniqlaydi.

Hozir ro'yxatda nechta 5 soni borligi va uning indeksini aniqlaymiz:

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

x = a.count(5)
print(x)

x = a.index(5)
print(x)
```

sort() va reverse()

- **sort()** funksiyasi ro'yxatni tartiblaydi. Agar ro'yxat sonlardan tashkil topgan bo'lsa, o'sish tartibida, satr yoki farflardan tashkil topgan bo'lsa, alifbo bo'yicha tartiblaydi.
- **reverse()** funksiyasi ro'yxatning joriy holatdagi tartibini teskarisiga o'zgartiradi.

Hozir ikki xil ro'yxatni avval tartiblaymiz, so'ngra ularni teskarisiga o'zgartiramiz:

```
meva = ["olma", "banan", "apelsin", "nok", "uzum"]
a = [1, 2, 3, 4, 5]

meva.sort()
a.sort()
print(meva)
print(a)

meva.reverse()
a.reverse()
print(meva)
print(a)
```

```
['apelsin', 'banan', 'nok', 'olma', 'uzum']
[1, 2, 3, 4, 5]
['uzum', 'olma', 'nok', 'banan', 'apelsin']
[5, 4, 3, 2, 1]
```

TUPLE (KORTEJ)

Kortejlar bir nechta ob'yektlarni birgalikda saqlashga xizmat qiladi. Ularni ro'yxatlarga o'xshatish mumkin. Lekin ular ro'yxatlar kabi boy funksionallikka ega emas. Ularning asosiy jihati qatorlarga o'xshab o'zgarmasliklaridir. **Kortej**- elementlar orasini vergul bilan ajratish orqali hosil qilinadi.

Kortejga ma'no jihatdan o`zgarmas ro`yxat deb ta'rif berdik. Shu o`rinda savol tug`iladi. Ro`yxat bo`lsa kortej nimaga kerak:

- **Turli holatlardan himoyalanish**. Bu degani kortej o`zgartirishlardan himoyalangan bo`ladi, rejali (bu yomon) va tasodifiy (bu yaxshi) o`zgarishlardan xalos bo`ladi.
- **Kichik hajm.** So`zlar bilan ifodalamasdan.

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> b = [1, 2, 3, 4, 5, 6]
>>> a.__sizeof__()
72
>>>
>>> b.__sizeof__()
88
```

- **Kortejdan lug`at kaliti sifatida foydalanish mumkin**:

```
>>> d = {(1, 1, 1) : 1}
>>> d
{(1, 1, 1): 1}
>>> d = {[1, 1, 1] : 1}
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    d = {[1, 1, 1] : 1}
TypeError: unhashable type: 'list'
```

Kortej afzalliklari haqida bilib oldik. Endi kortej bilan qanday ishlashni ko`ramiz. Bu xuddi ro`yxatlar bilan ishlashga o`xshaydi.

Tuple ro'yxati tartiblangan, o'zgarmas ro'yxat. Uning elementlarini o'zgartirib bo'lmaydi. Bu ro'yxatni oddiy qavslar bilan yoki **tuple()** konstruktori bilan hosil qilinadi:

```
a = ("kitob", "daftar", "ruchka")

b = tuple(("qog'oz", "qalam", "qaychi"))

print(a)
print(b)
```

Bir elementli to'plam

Bitta elementli tuple kortejini hosil qilish uchun alement qiymatidan so'ng vergul qo'yish kerak. Aks holda bunday kortej hosil bo'lmaydi:

```
a = ("kitob",)
print(type(a))

b = ("kitob")
print(type(b))
```

Elementlarga murojaat

Tuple elementiga murojaat qilish uning indeksini ko'rsatish bilan amalga oshiriladi:

```
a = ("kitob", "daftar", "ruchka")
print(a[0])
```

Manfiy indeks

Manfiy indeks sanoqning oxiridan boshlanishini anglatadi. Masalan, -1 eng oxorgi, -2 oxiridan ikkinchi element va hokazo.

```
a = ("kitob", "daftar", "ruchka")
print(a[-1])
```

```
ruchka
```

Indeks oralig'i

Ro'yxatning ma'lum qismidagi bir nechta elementni tanlab olish uchun o'sha indekslar oralig'ini kiritamiz. Bunda uning boshlanish va oxirgi nuqtalari kiritiladi. Element tanlashda oxirgi nuqta hisobga kirmaydi. Ya'ni boshlang'ich nuqtadan boshlanib oxirgi nuqtadan bitta oldingi elementgacha olinadi.

Hozir ekranga ikkinchi, uchinchi va to'rtinchi elementlarni tanlab ekranga chiqaramiz:

```
a = ("kitob", "daftar", "ruchka", "qog'oz", "qalam")
print(a[1:4])
```

```
('daftar', 'ruchka', "qog'oz")
```

Element qiymatlarini o'zgartirish

Tuple to'plamidagi elementni to'g'ridan-to'g'ri o'zgaartirib bo'lmaydi. Yuqorida aytganimizdek u o'zgarmas. Biroq bu muammoning ham yechimi bor. **Tuple** ro'yxatini avval **list** ro'yxatiga aylantirib, so'ngra istalgan elementni o'zgartiriladi va yana **tuple** ro'yxatiga aylantiriladi:

```
a = ("kitob", "daftar", "ruchka")
b = list(a)
b[2] = "qalam"
a = tuple(b)

print(a)
```

```
('kitob', 'daftar', 'qalam')
```

Ro'yxat bo'ylab sikl

Tuple to'plamida ham for siklidan foydalanib elementlarni tanlab olish mumkin. Hozir shu usulda elementlarni ekranga chiqaramiz:

```
a = ("kitob", "daftar", "ruchka")

for x in a:
    print(x)
```

Elementning mavjudligini tekshirish

Biror elementning to'plamda mavjudligini in kalit so'zi orqali tekshiramiz. Masalan, ro'yxatimizda kitob borligini tekshiramiz:

```
a = ("kitob", "daftar", "ruchka")

if "kitob" in a:
    print("kitob bor")
else:
    print("Kitob yo'q")
```

Kortejning funksiya va metodlari

- **count(x)** - kortejdagi x elementi sonini qaytaradi.
- **index(x)** - kortejdagi x elementining indeksini qaytaradi.
- **any()** - agar kortej elementi mavjud bo`lsa True qiymat qaytaradi, aks holda (kortej bo`sh bo`lsa) False qiymat qaytaradi.

- **max()** - kortejning maksimal elementini qaytaradi.
- **min()** - kortejning minimal elementini qaytaradi.
- **len()** - kortejning uzunligini qaytaradi.
- **sorted()** - kortej elementlaridan iborat yangi tartiblangan ro`yxatni qaytaradi.
- **sum()** - kortej elementlari yig`indisini qaytaradi.

Keling **tuple** ya'ni kortejda metodlarni qo`llanilishini misollar yordamida ko`rib chiqamiz.

```
k=('2','12','9','78','15','12',)
j=tuple()
print(k)
print('kortejdagi x element soni',k.count('12'))
print('kortejdagi x element indeksi',k.index('78'))
print('kortejni tekshirish',any(k))
print('kortejni tekshirish',any(j))
print('max element:',max(k))
print('min element:',min(k))
print('kortej uzunligi:',len(k))
```

```
('2', '12', '9', '78', '15', '12')
kortejdagi x element soni 2
kortejdagi x element indeksi 3
kortejni tekshirish True
kortejni tekshirish False
max element: 9
min element: 12
kortej uzunligi: 6
['12', '12', '15', '2', '78', '9']
```

Tuplening uzunligi

Tuple to'plamining uzunligi, yani nechta elementdan tashkil topganligini **len()** funksiyasi bilan aniqlash mumkin:

```
a = ("kitob", "daftar", "ruchka")

print(len(a))
```

Element qo'shish

Tuple korteji o'zgarmas bo'lgani uchun unga element qo'shib bo'lmaydi. U boshida nechta element hosil qilgan bo'lsa, shuncha element bilan qoladi. Ammo istisno tariqasida, yuqorida elementning qiymatini o'zgartirganimiz kabi shu usulda yangi element qo'shsa bo'ladi.

Tuple larni qo'shish

Ikki yoki undan ortiq tuple larni o'zaro qo'shish uchun "+" operatori kifoya:

```
a = ("kitob", "daftar", "ruchka")
b = ("qalam", "qog'oz")

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

count() va index()

- **count()** funksiyasi belgilangan qiymatga teng elementlar sonini aniqlaydi.
- **index()** funksiyasi belgilangan elementning indeksini aniqlaydi. Agar bunday elementlar bir nechta bo'lsa, faqat birinchisining indeksini aniqlaydi.

Hozir kortejda nechta 3 soni borligi va uning indeksini aniqlaymiz:

```
toq_son = (1, 3, 5, 3, 3, 7)
x = toq_son.count(3)
print(x)

y = toq_son.index(3)
print(y)
```

```
3
1
```

SET (TO'PLAM)

Pythonda to'plamlar bilan ishlash uchun maxsus **set** deb nomlanuvchi ro'yxat turi mavjud. Pythondagi to`plam- tasodifiy tartibda va takrorlanmaydigan elementlardan tashkil topgan "konteyner" deyiladi. To'plam elementlari tartiblanmagan va indekslanmagan tarzda bo'ladi. To'plamni hosil qilish uchun maxsus qavslardan foydalaniladi. Yoki **set()** konstruktori ishlatiladi:

```
toq_son = {1, 3, 5, 7, 9}
print(toq_son)
juft_son = set((2, 4, 6))
print(juft_son)
```

Set to'plaminig funksiya va metodlari

- **len(s)** - to`plamdagi elementlar soni(to`plam hajmi).
- **x in s** - 'x' 's' to`plamga tegishli bo`ladimi yo`qmi shuni tekshiradi
- **set.isdisjoint(other)** -agarda set va other umumiy elementlarga ega bo`lmasalar rost qiymat qaytaradi.
- **set==other** - set ning hamma elementlari otherga tegishli bo`ladilar otherni hamma elementlari setga tegishli bo`ladilar.
- **set.issubset(other)** yoki **set<=other**-set ning hamma elementlari other ga tegishli bo`ladilar.
- **set.issuperset(other)** yoki **set>=other** -analogik holat.
- **set.union(other, …)** yoki **|other|…**-bir qancha to`plamlar birlashmasi.
- **set.intersection(other, …)** yoki **&other&…** - kesib olish.
- **set.difference(other, …)** yoki **-other-…** - other ga tegishli bo`lmagan set ning hamma elementlar to`plami.
- **set.symmetric_difference(other); set^other**- birinchi to`plamda uchraydigan, lekin ularning ikkala to`plamning kesishmasida uchramaydigan elementlar.
- **set.copy**-to`plam nusxasi

To`plamni to`g`ridan-to`g`ri o`zgartiradigan operatsiyalar

- **Set.update(other, …); set|=other| …** - to`plam birlashmasi
- **Set.intersection_update(other, …); set&=other&…** - to`plam kesishmasi
- **Set.difference_update(other, ...); set -= other | ...** -to`plam ayirmasi
- **Set.symmetric_difference_update(other); set ^= other** - birinchi to`plamda uchraydigan, lekin ularning ikkala to`plamning kesishmasida uchramaydigan elementlar tashkil topgan to`plam.
- **Set.add(elem)**- to`plamga element qo`shadi.
- **Set.remove(elem)**- to`plamdagi elementni o`chiradi. Agarda ko`rsatilgan element to`plamda mavjud bo`lmasa KeyError ni qaytaradi.
- **Set.discard(elem)**- gar to`plamda ko`rsatilgan element bo`lsa uni o`chiradi.
- **Set.pop()**- to`plamdagi birinchi elementni o`chiradi, lekin top`lam elementlari tartib bilan joylashmagani uchun birinchi element qaysiligini aniq ko`rsatib bo`lmaydi.
- **Set.clear()**- to`plamni tozaydi.

Elementlarga murojaat

To'plamlar tartiblanmagan ro'yxat bo'lganligi uchun ularning elementlariga indeks orqali murojaat qilib bo'lmaydi. To'plam elementlariga murojaat qilish uchun for siklidan yoki aniq bir element borligini tekshirish uchun in kalit so'zidan foydalanamiz:

```
toq_son = {1, 3, 5, 7, 9}

for x in toq_son:
    print(x)

prin("---------\n")
print(3 in toq_son)
```

Element qo'shish

To'plam hosil qilingandan so'ng uning elementlarini o'zgartirib bo'lmaydi, ammo yangi element qo'shish mumkin. Agar to'plamga bitta element qo'shish kerak bo'lsa, **add()** fuksiyasi, agar bir nechta element qo'shish kerak bo'lsa, **update()** funksiyasi ishlatiladi.

```
toq_son = {1, 3, 5, 7, 9}

toq_son.add(9)
print(toq_son)

toq_son.update([11, 13, 15])
print(toq_son)
```

To'plam uzunligi

To'plamning uzunligi, ya'ni nechta elementdan tashkil topganligini **len()** kalit so'zi bilan aniqlanadi:

```
meva = {"nok", "banan", "shaftoli"}
print(len(meva))
```

Elementni o'chirish

Elementni to'plamdan o'chirish uchun **remove()** va **discard()** funksiyalari ishlatiladi. Bu funksiyalarning farqi shundaki, **remove()** funksiyasi bilan o'chirmoqchi bo'lgan elementimiz to'plamda mavjud bo'lmasa, kod ishga tushganda xatolik ro'y beradi. **discard()** funksiyasi bilan esa bu holat kuzatilmaydi.

Hozir ikkala usul bilan ham elementlarni o'chirib ko'ramiz:

```
toq_son = {1, 3, 5, 7, 9}

toq_son.remove(1)
print(toq_son)

toq_son.discard(5)
print(toq_son)
```

Elementni to'plamdan **pop()** funksiyasi bilan ham o'chirish mumkin. Ammo **pop()** funksiyasi xususiyatiga ko'ra ro'yxat oxiridagi elementni o'chiradi. To'plam esa tartiblanmagan ro'yxat. Shuning uchun bu funksiya aynan qaysi elementni o'chirishini oldindan bilolmaymiz. Biroq o'chirilgan elementni aniqlash mumkin:

```
meva = {"nok", "banan", "shaftoli"}

x = meva.pop()
print(meva)
```

clear()

clear() funksiyasi to'plamni bo'shatadi, ya'ni barcha elementlarini o'chiradi:

```
meva = {"nok", "banan", "shaftoli"}

meva.clear()
print(meva)
```

del kalit so'zi to'plamni butunlay o'chiradi:

```
meva = {"nok", "banan", "shaftoli"}

del meva
print(meva)
```

To'plamni qo'shish

To'plamlarni o'zaro bir-biriga qo'shish uchun maxsus funksiyalar mavjud:

union() funksiyasi ikkala to'plam elementlarini boshqa bir yangi to'plamga o'zlashtiradi. Agar to'plamlarda bir xil elementlar uchrab qolsa, ularning faqat bittasi olinadi.

```
harf1 = {"a", "b", "c", "d"}
harf2 = {"c", "e", "e", "f"}

harf3 = harf1.union(harf2)
print(harf3)
```

update() funksiyasi bir to'plam elementlarini boshqa biriga qo'shadi. Bunda ham bir xil elementlar uchrab qolsa, ularning faqat bittasi olinadi.

Nusxa olish

Biror to'plamning aynan o'zidek yana bitta to'plam hosil qilish uchun nusxa olish kerak. Buning uchun **copy()** funksiyasidan foydalanamiz:

```
harf1 = {"a", "b", "c", "d"}

harf = harf1.copy()
print(harf)
```

Muhim funksiyalar

Hozir biz ko'rib chiqmoqchi bo'lgan funksiyalar to'plamlar bilan ishlash uchun zarur funksiyalardir. Ular to'plamlarning o'ziga xos xususiyatlariga tayangan holda ishlab chiqilgan.

difference(), difference_update()

- **difference()** funksiyasi **x** to'plamda bor, lekin **y** to'plamda yo'q bo'lgan elementlardan tashkil topgan to'plam hosil qiladi.
- **difference_update()** funksiyasi agar ikkala to'plamda bir xil elementlar mavjud bo'lsa, o'sha elementni o'chiradi.

```
x = {"a", "b", "c", "d"}
y = {"g", "c", "e", "d"}

z = x.difference(y)
print(z)

x.difference_update(y)
print(x)
```

interselection(), interselection_update()

- **interselection()** funksiyasi qaysi elementlar ikkala to'plamda ham mavjud bo'lsa, o'sha elementlardan tashkil topgan yangi to'plam hosil qiladi.
- **interselection_update()** funksiyasi **x** to'plamdagi element **y** to'plamda ham mavjud bo'lsa, o'sha elementni qoldiradi. Qolganlarini esa o'chirib yuboradi.

```
x = {"a", "b", "c", "d"}
y = {"g", "c", "e", "d"}

z = x.intersection(y)
print(z)

x.intersection_update(y)
print(x)
```

isdisjoint()

isdisjoint() funksiyasi agar **x** to'plamdagi birorta ham element **y** to'plamda mavjud bo'lmasa, rost qiymat qaytaradi.

Quyidagi kodimizda rost qiymat qaytariladi. Chunki **x** to'plamdagi elementlarning hech biri **y** to'plamda majud emas:

```
x = {"a", "b", "c"}
y = {"l", "m", "n", "o"}

z = x.isdisjoint(y)
print(z)
```

issubset(), issuperset()

- **issubset()** funksiyasi agar **x** to'plamdagi barcha elementlar **y** to'plamda ham mavjud bo'lsa, rost qiymat qaytaradi.
- **issuperset()** funksiyasi esa teskarisi, ya'ni agar **y** to'plamdagi barcha elementlar **x** to'plamda ham mavjud bo'lsa, rost qiymat qaytaradi.

Quyidagi misolimizda **x** to'plamdagi barcha elementlar **y** to'plamda mavjud, ammo **y** to'plamdagi barcha elementlar ham **x** to'plamda mavjud emas. Shuning uchun avval rost, keyin esa yolg'on qiymat ekranga chiqadi:

```
x = {"a", "b", "c"}
y = {"l", "m", "n", "o", "k", "q", "t", "b"}

z = x.issubset(y)
print(z)

z = x.issuperset(y)
print(x)
```

symmetric_difference(), symmetric_difference_update()

- **symmetric_difference()** funksiyasi ikkala to'plamda ham mavjud bo'lgan bir xil elementlardan tashqari barcha elementlarni olib yangi to'plam hosil qiladi.
- **symmetric_difference_update()** funksiyasi **x** to'plamga **y** to'plamdan o'zida mavjud bo'lmagan barcha elementlarni olib qo'shadi.

```
x = {"a", "b", "c"}
y = {"l", "c", "a", "o", "k", "t", "b"}

z = x.symmetric_difference(y)
print(z)

z = x.symmetric_difference_update(y)
print(x)
```

DICTIONARY (LUG'AT)

Pythondagi lug`atlar kalit bo`yicha kirishga ruxsat etuvchi erkin obyektlarning tartiblangan jamlanmasi. Ularni yana assotsiativli massivlar yoki hesh jadvallar deb nomlaydilar. Soddaroq qilib aytadigan bo`lsak lug'at xuddi manzillar kitobiga o'xshaydi, ya'ni biror insonning ismini bilgan holda uning manzili yoki u bilan bo'g'lanish ma'lumotlarini olish mumkin.

Dictionary – tartiblanmagan, o'zgaruvchan va indeksil to'plam. Bu to'plamda kalit-qiymat (***key-value***) tushunchasi mavjud, ya'ni maxsus kalit va ularga mos keluvchi qiymatlar juftlgidan tashkil topgan. Chap tarafda kalitlar, o'ng tomonda esa ularga mos keluvchi qiymatlar joylashgan bo'ladi. Buni hoir dictionary to'plamini hosil qilib bilib olamiz. Bu quyidagicha amalga oshiriladi:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

print(avto)
```

dict() konstruktori

dict() konstruktori bilan ham yangi to'plam hosil qilish mumkin. Bu quyidagicha amalga oshiriladi:

```
avto = dict(brend="chevrolet", model="Malibu", yil=2016)
print(avto)
```

Elementlarga murojaat

Dictionary elementlariga murojaat qilish uchun ularning kalitlarini kvadrat qavs ichida ko'rsatish yoki **get()** funksiyasidan foydalanish mumkin. Hozir ikkala usuldan ham foydalanamiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

x = avto["model"]
y = avto.get("yil")

print(x)
print(y)
```

Qiymatlarni o'zgartirish

Istalgan qiymatni o'zgartirish uchun unga kalit orqali murojaat qilamiz, so'ngra qiymatini o'zgartiramiz. Masalan quyidagi avtomobil haqidagi ma'lumotda yilni o'zgartiramiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

avto["yil"] = 2018

print(avto)
```

Sikldan foydalanish

Dictinary to'plamida **for** siklidan foydalangan holda uning elementlariga murojaat qilish mumkin. Bunday holatda qiymatlarga emas, balki kalitlarga murojaat bo'ladi. Hozir to'plamdagi kalitlarni ekranga chiqaramiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

for x in avto:
    print(x)
```

Agar qiymatlarning o'ziga murojaat qilmoqchi bo'lsak, **values** funksiyasidan foydalanamiz yoki yuqoridagidan biroz boshqacharoq tarzda amalga oshiramiz. Quyidagi kodimizda har ikkala usuldan ham foydalangan holda qiymatlarni ekranga chiqaramiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

#1-usul
for x in avto:
    print(avto[x])

#2-sul
for x in avto.values():
    print(x)
```

Agar kalit va qiymatlarning ikkalasiga ham bir vaqtda murojaat qilmoqchi bo'lsak, **items()** funksiyasidan foydalanamiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

for x,y in avto.items():
    print(x,y)
```

Kalit so'z mavjudligini aniqlash

Biror kalit to'plamda bor yoki yo'qligini aniqlash uchun in kalit so'zi ishlatiladi:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

if "yil" in avto:
    print("Ha, mavjud")
else:
    print("Yo'q mavjud emas")
```

Lug`atning funksiya va metodlari

- **Dict.clear()**- lug`atni tozalaydi.
- **Dict.copy()**-lug`at nusxasini qaytaradi.
- Classmethod **dict.fromkeys(seq[, value])**- Seq dan kalitni va Value qiymatlariga ega bo`lgan lug`atni yaratadi.
- **Dict.get(key[, default])**-kalit qiymatini qaytaradi, lekin u bo`lmasa xatolik beradi, default (jimlikda None) qaytaradi.
- **Dict.items()**-juftliklarni qaytaradi(kalit, qiymat)
- **Dict.keys()**- lug`atdagi kalitlarni qaytaradi
- **Dict.pop(key[default])**-kalitni yo`qotib qiymatni qaytaradi. Agarda kalit bo`lmasa defaultni qaytaradi.
- **Dict.popitem()**- juftlikni o`chirib qaytaradi (kalit, qiymat). Agarda lug`at bo`sh bo`lsa KeyError istisnoni chaqiradi. Esingizda tursin lug`atlar tartibli emas.

- **Dict.setdefault(key [, default])**-kalit qiymatni qaytaradi, lekin u bo`lmasa xatolik bermaydi, default qiymatga ega kalitni yaratadi (jimlikda None).
- **Dict.update([other])**- other dan juftliklarni (kalit, qiymat) kiritib lug`atni to`ldiradi. Mavjud bo`lgan kalitlar qaytadan yoziladilar. None (eski lug`at) qaytaradi.
- **Dict.values()**-lug`atdagi qiymatni qaytaradi.

Keling **tuple** ya'ni kortejda metodlarni qo`llanilishini misollar yordamida ko`rib chiqamiz.

```
d=dict(ismi='Gulnoza', yoshi='8', maktabi='1')
print()
print('lug`atning qiymati:',dict.values(d))
print()
print('lugatdagi juftliklar yani kalit va uning qiymatlari:',dict.items(d))
print()
print('lugatning kalitlari:',dict.keys(d))
print()
print('lugatning nusxasi:',dict.copy(d))
```

Natija:

```
lug`atning qiymati: dict_values(['Gulnoza', '8', '1'])
lugatdagi juftliklar yani kalit va uning qiymatlari:dict_items([('ismi','Gulnoza'),
('yoshi', '8'), ('maktabi', '1')])
lugatning kalitlari: dict_keys(['ismi', 'yoshi', 'maktabi'])
lugatning nusxasi: {'ismi': 'Gulnoza', 'yoshi': '8', 'maktabi': '1'}
```

Dictionary uzunligi

Dictionary to'plamida nechta element, yani **kalit-qiymat** juftligi borligini aniqlash uchun **len()** funksiyasidan foydalanamiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

print(len(avto))
```

Element qo'shish

Yangi elementni, ya'ni **kalit-qiymat** juftligini qo'shish quyidagicha amalga oshiriladi. Masalan, biz mashinamizning rangi haqida ma'lumot beruvchi element qo'shamiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

avto["rang"] = "qora"
print(avto)
```

Elementlarni o'chirish

Dictionary to'plamidan elementni o'chirishning turli xil yo'llari mavjud. Barchasini birma-bir ko'rib chiqamiz:

Birinchi usul – **pop()** funksiyasi yoki **del** kalit so'zi. Ikkalasi ham ko'rsatilgan kalit bo'ycicha elementni o'chiradi. Hozir ularni qanday ishlatishni ko'ramiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

avto.pop("model")
print(avto)

del avto["yil"]
print(avto)
```

Keyingi usul – **popitem()** funksiyasi to'plamga oxirgi bo'lib kiritilgan elementni o'chiradi (*Python 3.7 dan oldingi versiyalarda bu funksiya ixtiyoriy biror elementni o'chiradi*).

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

avto.popitem()
print(avto)
```

Oxirgi usul – esa **clear()** funksiyasi. Bu funksiya to'plamni bo'shatadi, ya'ni barcha elementlarini o'chiradi. Natijada to'plam bo'm-bo'sh holatga keladi.

del kalit so'zi bilan to'plamning o'zini butkul o'chirish ham mumkin. Bilamizki, to'plam nomi bilan undagi biror kalitni ko'rsatsak, del o'sha kalit bo'yicha elementni o'chiradi. Ammo endi faqat to'plam nomini kiritsak, bu kalit so'zi butun to'plamni o'chiradi.

Quyidagi kodimizda dastlab, to'plamni bo'shatamiz, so'ngra uni butkul o'chiramiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

avto.clear()
print(avto)

del avto
print(avto)
```

Nusxa olish

Agar biror dictionary to'plamidan nusxa olib ayna uning o'zidek to'plam hosil qilmoqchi bo'lsak, buni maxsus yo'l bilan qilish kerak bo'ladi. Bunday holatda bizga **copy()** yoki **dict()** maxsus funksiyalari yordamga keladi. Har ikkala funsiyadan ham foydalanish mumkin. Hozir buni misolda ko'rib chiqamiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

avto2 = avto.copy()
print(avto2)

avto3 = dict(avto)
print(avto3)
```

Joylashtirilgan to'plamlar

Bitta dictionary to'plamini o'z ichiga bir nechta ana shunday to'plam saqlashi mumkin. Buning uchun ularni quyidagicha hosil qilish kerak:

```
avto = {
    "avto1": {
        "model": "Nexia",
        "yil": 2016
}
    "avto2": {
        "model": "Spark",
        "yil": 2018
}
    "avto3": {
        "model": "Captiva",
        "yil": 2019
}
}
print(avto)
```

Agar allaqachon mavjud to'plamlarni bitta to'plamga yig'moqchi bo'lsangiz, quyidagicha amalga oshiriladi:

```
avto1 = {
        "model": "Nexia",
        "yil": 2016     }

avto2 = {
        "model": "Spark",
        "yil": 2018     }

avto3 = {
        "model": "Captiva",
        "yil": 2019     }

avto = {
    "avto1": avto1,
    "avto2": avto2,
    "avto3": avto3 }

print(avto)
```

setdefault()

setdefault() fuksiyasi ko'rsatilgan kalit bo'yicha element qiymatini qaytaradi. Agar bunday kalit to'plamda mavjud bo'lmasa, shu kalit va biz ko'rsatgan qiymatni yangi element sifatida to'plamga qo'shadi.

Hozir tekshirib ko'ramiz, agar model kaliti to'plamda mavjud bo'lsa, bizga uning qiymati ko'rsatilsin. Aks holda shunday kalitga **Captiva** qiymatini biriktirib, to'plamga qo'shilsin.

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

x = avto.setdefault("model", "Captiva")

print(avto)
```

update()

update() funksiyasi to'plamga yangi element (*kalit-qiymat juftligi*) qo'shadi. Bunda har bir vaqtning o'zida istalgancha element qo'shish mumkin.

Hozir biz to'plamga yangi element qo'shamiz:

```
avto = {
    "brend": "Chevrolet",
    "model": "Malibu",
    "yil": 2016
}

avto.update({"rang": "qora"})

print(avto)
```

III-BOB. PYTHON STATEMENTS (BAYONNOMALAR)

MANTIQ ELEMENTLARI VA OPERATORLARI

Mantiq elementlari 2 xil qiymatdan birini qabul qiladi. True (rost) yoki False (yolg'on). Dasturlashda mantiq elementlarini bilish shart. Pythonda istalgan shartni tekshirib True yoki False qiymatlariga ega bo'lishi mumkin. Masalan, 2 ta qiymatni o'zaro taqqoslasak, Pythonda bizga mantiq elementlari bilan javob qaytaradi. Quyidagi dasturni ishga tushirsak, ekranga faqat True va False qiymatlari chiqadi:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

If operatori bilan shart tekshirilganda ham Python bizga mantiq elementlari bilan javob qaytaradi. Mantiq elementlarining asosiy vazifasi bizga biror shartning bajarilishi rost yoki yolg'on ekanligini ifodalab berishdir. Va shunga qarab Pythonga biror yangi amalni bajarish yoki bajarmaslikni buyruq beramiz.

Masalan hozir dasturimizda bir shartni tekshiramiz. Agar u to'g'ri bo'lsa, ekranga to'g'ri deb chiqsin. Aks holda, noto'g'ri deb xabar bersin.

```
a = 100
b = 30

if a>b:
    print("To'g'ri")
else:
    print("Notog'ri")
```

Qiymatlarni tekshirish

bool() funksiyasi qiymatlarni tekshirib, True yoki False qiymat qaytaradi. Odatda hamma qiymat True natijani beradi. Faqat son qiymatlari 0 bo'lmasligi, satr va boshqa o'zgaruvchilar bo'sh qiymatga ega bo'lmasligi kerak. Quyidagilar faqat True qiymat qaytaradi.

```
x = "Salom"
y = 15
z = ["olma", "anor", "banan"]

print(bool(x))
print(bool(y))
print(bool(z))
```

Funksiyada mantiq elementlari

Funksiyalarni mantiq elementlari bilan javob qaytaradigan qilib hosil qilish ham mumkin:

```
def myfunction():
    return True

print(myFunktion())
```

Funksiyaning mantiq elementlari asosida boshqa amallar bajarish ham mumkin. Hozir funksiya rost qiymat qaytarsa, ekranga rost deb, aks holda yolg'on deb xabar beruvchi dastur tuzamiz:

```
def myfunc():
    return False

if myfunc():
    print("rost")
else:
    print("yolg'on")
```

Python mantiq elementlari bilan javob qaytaruvchi ko'plab ichki funksiyalarga ega. Masalan, qiymatning biror ma'lumot turiga tegishli ekanligi yoki yo'qligini tekshiruvchi **isinstance()** funksiyasi. Quyidagi kodimizda x o'zgaruvchisi int turiga kirishini tekshiramiz:

```
x = 300

print(isinstance(x, int))
```

Mantiq operatorlari

Mantiq operatorlar shartlarni birlashtirib ishlatish uchun kerak:

- **and** - Agar ikkala shart ham rost bo'lsa, rost qiymat qaytaradi.
- **or** - Kamida bitta shart rost bo'lsa ham rost qiymat qaytaradi.
- **not** - Shart qiymatini teskarisiga o'zgartiradi, ya'ni rost bo'lsa yolg'on, yolg'on bo'lsa rost bo'ladi.

```
a = 5

print (a>3 and a<10)
print (a>3 or a<4)
print (not(a>3 and a<10))
```

```
True
True
False
```

Agar bir vaqtning o'zida bir emas, balki bir nechta shartlarni tekshirmoqchi bo'lsak, mantiq operatorlari (**and, or**) juda qo'l keladi. Bunda 2 xil shartdan kamida bittasi bajarilishi, yoki ikkalasi ham bajarilishini tekshirib ko'rsak bo'ladi. Masalan, hozir uchta sonni olib o'zaro taqqoslaymiz. Bunda bir son qolgan ikkalasidan ham kattaligini yoki kamida bittasidan kattaligini tekshiramiz:

```
a = 10
b = 15
c = 20

if a>b and b>c:
    print("Ikkalasidan ham katta")
elif b>a or b>c:
    print("Kamida bittasidan katta")
```

PYTHONDA SHART OPERATORLARI

Pythonda shart operatorlari shartni tekshirish uchun ishlatiladi. Pythonda shart operatorini bir necha xil ko`rinishi mavjud:

- **if (mantiqiy ifoda):-** shart operatorining bu ko`rinishi mantiqiy ifoda rost bo`lgan holda qandaydir kod bajarilishi uchun ishlatiladi.
- **if (mantiqiy ifoda):**…**else-**shart operatorining bu ko`rinishida mantiqiy ifoda rost bo`lsa, birinchi ifodalar bloki bajariladi(bu blok "**if-blok**" deb nomlanadi), **aks holda** keyingi ifodalar bloki bajariladi(bu blok "**else-blok**" deb nomlanadi).
- **if (mantiqiy ifoda):**…**elif(mantiqiy ifoda):**…**else-** shart operatorining bu ko`rinishida oldingi shart yolg`on bo`lganda keyingi shart tekshiriladi. Bu ifoda o`zida ikkita bir-biriga bog`liq bo`lgan **if else-if else** ifodani bir ifodada **if elif else** saqlaydi. Bu dasturni o`qishni osonlashtiradi.

Demak endi bu holatlarning barchasini misollar yordamida ko'rib chiqamiz.

IF

if kalit so'zi biror shartning bajarilishi yoki bajarilmasligini tekshiradi. Masalan, bir qiymat ikinchisidan kattaligi yoki ular o'zaro teng emasligi va hokazo kabi shartlarni tekshirish mumkin. Hozir oddiy misol qilib a sonni b sonidan katta ekanlgini tekshirib ko'ramiz. Agar shart bajarilsa, "**HA**" degan yozuv ekranga chiqsin:

```
a = 50
b = 30

if a>b:
    print("HA")
```

Shart tekshirilgach, bajariladigan amalni keyingi qatorda yozishda, xuddi abzatsdan yozgan kabi yozish kerak aks holda dasturda xatolik yuz beradi. Tushinish uchun avval yuqoridagi kodga qarang, keyin quyidagi kodga e'tibor bering. Bu kodimiz ishga tushganda xatolik yuz beradi. Chunki so'nggi qator abzatsdan boshlanishi kerak edi.

```
a = 50
b = 30

if a>b:
print("HA")
```

else

else kalit so'zi "**aks holda**" jumlasi kabidir. Shartimiz bajarilmaganda nima amal bajarish kerakligini ko'rsatish uchun qo'llaniladi. Masalan, a soni b sonidan katta bo'lsa, "**HA**" yozuvini ekranga chiqaramiz, agar bus hart bajarilmasa, "**YO'Q**" yozuvi ekranga chiqarilsin:

```
a = 50
b = 90

if a>b:
    print("HA")
else:
    print("YO'Q")
```

elif

agar bir emas, malki ko'proq shartlarni tekshirishga to'g'ri kelsa, **elif** kalit so'zini ishlatamiz. Bunda **if** kalit so'zi bilan shart tekshiriladi, qolganlari esa **elif** kalit so'z bilan tekshiriladi.

```
a = 50
b = 30

if a>b:
    print("a soni b sondan katta")
elif a==b:
    print("ular o'zaro teng")
elif a<b:
    print("a soni b sondan kichik")
else:
    print("Hech qaysi shart bajarilmadi !!!")
```

Pass

if kalit so'zi bilan shart tekshirilgandan keyin bajariladigan amalni albatta yozishimiz kerak. Aks holda dasturda xatolik yuz beradi. Ammo hali nima amal bajarish kerakligini o'ylab ko'rmagan bo'lsak, u yerga **pass** so'zini qo'yish kifoya. Bu so'z tufayli dastur ishga tushganda aynan o'sha qismni hisobga olmasdan o'tib ketadi. Natijada dasturning qolgan qismlariga bu ta'sir qilmaydi.

```
a = 33
b = 99

if b>a:
    pass

print("pass so'zi bo'lmaganda ushbu yozuv ekranga chiqmas edi.")
```

PYTHONDA SIKLLAR

Python dasturlash tilida ikki xil sikl ishlatiladi. Bular **while** va **for** sikllari. Ularning qulayligi shundaki, ular belgilangan nuqtaga yetmaguncha ko'rsatilgan amalni qayta-qayta bajaraveradi. Shu sababli biz bir amalni qayta-qayta yozib o'tirmaymiz.

while va **for** qo'llanish usuli va joyiga ko'ra farqlanadi. Bu dasrda **while** bilan tanishamiz.

WHILE SIKLI

while sikliga odatda bir shart berish kerak bo'ladi va o'sha shart bajarilmaguncha u biz ko'rsatgan amalni qayta-qayta bajaraveradi. **while** sikli quyidagi umumiy ko`rinishga ega:

```
while (shart):
    sikl_tanasi
```

While sikl operatorining ishlash tartibi

- Agar (shart) rost (**true**) qiymatga ega bo`lsa, **sikl_tanasi** bajariladi. Qachonki shart yolg`on (**false**) qiymatga teng bo`lsa sikl tugatiladi.
- Agar (shart) true qiymatga ega bo`lmasa sikl tanasi biror marta ham bajarilmaydi.

Masalan 1 da 10 gacha bo'lgan sonlarni ekranga chiqarishimiz kerak bo'lsa, buni quyidagicha amalga oshiramiz:

Avval, boshlang'ich nuqtani belgilaymiz, ya'ni o'zgaruvchi 1 ga teng bo'ladi. So'ngra shunday shart beramizki toki o'sha shart o'zgaruvchi 11 dan kichik ekan, uni har safar ekranga chiqarib shu songa 1 ni qo'shib ketaversin. Natijada o'zgaruvchimiz toki 10 ga yetguncha ushbu amalni bajaraveradi. 11 ga yetganda esa shart bajarilmay qoladi va sikl to'xtaydi.

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

break

break kalit so'zi siklni to'xtatadi. Asosiy sikl davom etayotgan bo'lsa ham, biz belgilagan nuqtada siklni to'xtatadi. Masalan yuqoridagi misolni olamiz. Uni shunday o'zgartiramizki, o'zgaruvchimizning qiymati 5 ga yetganda sikl to'xtaydi va qolgan sonlarni ekranga chiqarmaydi:

```
i = 1
while i < 11:
    print(i)
    if i == 5:
        break
    i+=1
```

continue

continue kalit so'zi bilan siklning ba'zi nuqtalaridan sakrab o'tish mumkin. Masalanm biz 6 dan tashqari 1 dan 10 gacha bo'lgan sonlarni ekranga chiqaramiz. Bunda 6 soni hisobga olinmay undan o'tib ketiladi:

```
i = 1
while i < 11:
    i+=1
    if i == 6:
        continue
    print(i)
```

else

else kalit so'zi sikl to'xtagandan so'ng ham yan bir amal bajarish imkoni beradi. Masalan, sikl to'xtgandan so'ng to'xtaganligi haqida ma'lumot ekranga chiqsin:

```
i = 1
while i < 11:
    print(i)
    i+=1
else:
    print("sikl to'xtadi")
```

FOR SIKLI

Python dasturlash tilida **for** operatori C va Paskal dasturlash tillarida qo`llanishidan farq qiladi. Python da **for** operatori biroz murakkabroq, lekin **while** sikliga qaraganda ancha tezroq bajariladi. **For…in** operatori obyektlar ketma-ketligida iteratsiyani amalga oshiradi, ya'ni bu sikl har qanday iteratsiya qilinadigan obyekt bo`ylab o`tadi(satr yoki ro`yxat bo`ylab) va har bir o`tish vaqtida sikl tanasini bajaradi.

Python dasturlash tilida **for** sikli asosan to'plam va ro'yxatlar bilan ishlatiladi. **For** sikli bilan to'plam yoki ro'yxatning har bir elementiga murojaat qilish mumkin. Masalan, quyidagi ro'yxatning har bir elementini ekranga chiqaramiz:

```
meva = ["olma", "anor", "banan"]
for in meva:
    print(a)
```

Satr bo'ylab sikl

Satr bo'ylab sikl amalga oshirilsa satrdagi har bitta harfga murojaat bo'ladi. Chunki satr harflar to'plamidan tashkil topgan. Hozir quyidagi so'zning barcha harflarini ekranga chiqaramiz:

```
for a in "dastur":
    print(a)
```

break

break kalit so'zi bilan siklni to'xtatamiz, hattoki sikl to'xtamagan bo'lsa ham. Masalan, "**dastur**" so'zining harflarini birma-bir ekranga chiqarish siklini ishga tushuramiz va "**s**" harfiga yetganda siklni to'xtatamiz:

```
for x in "dastur":
    print(x)
    if x == "s":
        break
```

Endi e'tiborimizda bir narsaga qaratsak. Yuqoridagi kodda print buyrug'i break buyrug'idan oldinroq qo'ygan edik. Shu sababli avval **"s"** harfi ekranga chiqib, so'ng sikl to'xtadi. Endi print buyrug'ini pastroqqa qo'yamiz. Bunda **"s"** harfi ekranga chiqmay qoladi, chunki sikl undan avvalroq to'xtaydi.

```
for x in "dastur":
    if x == "s":
        break
    print(x)
```

continue

continue kalit so'zi siklning ayrim joylaridan sakrab o'tadi. Aniqroq qilib aytganda sikl davomida ayrim nuqtalarga kelganda ko'rsatilgan amalni bajarmay ketadi.

Masalan, "**python**" so'zidagi harflarni ekranga chiqaramiz va shunda "**h**" harfini tashlab ketamiz:

```
for x in "python":
    if x == 'h':
        continue
    print(x)
```

range() va xrange()

range() funksiyasi biror amalni belgilangan marta bajarish yoki biror oraliqdagi sonlarga murojaat qilsh uchun qo'llaniladi. Bunda **range()** ichiga kerakli son qo'yiladi va sanoq avtomatik tarzda o dan boshlanib ko'rsatilgan songacha davom etadi. Ammo uning o'zi hisobga kirmaydi.

Tushunish uchun misol ko'ramiz. 0 dan 5 gacha (5 soni hisobga kirmaydi) bo'lgan sonlarni ekranga chiqaramiz:

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

```
0
1
2
3
4
```

Yuqorida biz **range()** funksiyasida sanoq avtomatik 0 dan boshlanishini aytib o'tdik. Biz uni o'zimiz istagan sondan boshlashimiz ham mumkin.

Masalan 1 dan 5 gacha bo'lgan sonlarni ekranga chiqaramiz. Bunda sanoq 1 dan boshlanishi uchun 1 sonini ham kiritamiz. Demak, biz 1 dan 6 gacha bo'lgan oraliqni kiritamiz:

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

```
1
2
3
4
5
```

range() funksiyasida sonlar avtomatik bittaga ortib boradi. Ammo bu holatni ham o'zgartirish mumkin. Bunda oraliqni ko'rsatgandan so'ng sanoq nechtaga ortishini ham kiritamiz. Shunda funksiya ichidagi dastlabki ikkita son oraliqni, uchinchi son esa sanoq nechtaga ortiqshini ko'rsatadi.

Masalan, 1 dan 10 gacha bo'lgan faqat juft sonlarni ekranga chiqarmoqchimiz. Bunda oraliqni 2 dan 11 gacha deb belgilaymiz. Shunda sanoq 2 dan boshlanadi va 10 gacha davom etadi. Har safar sanoq ikkitaga ortishi uchun uchinchi bo'lib 2 soni kiritamiz:

```
for x in range(2, 11, 2):
    print(x)
```

```
2
4
6
8
10
```

Katta diapazondagi raqamlardan foydalanib ro`yxatni yaratish **range()** funksiyasi o`zini oqlamaydi yoki ba'zi hollarda xotira yetishmaydi.

```
>>> l=range(10000000)
Traceback (innermost last):
  File "<stdin>", line 1, in ?
MemoryError
```

Shunday hollarda Python da **xrange()** funksiyasidan foydalaniladi.

else

else kalit so'zi sikl tugagach ham yan bir amal bajarish imkonini beradi. Odatda bundan sikl tugagani haqida ma'lumot berishda foydalaniladi.

Masalan, "**python**" so'zini besh marta ekranga chiqarmoqchimiz va sikl tugagach shu haqida xabar beramiz. Bunda endi e'tibor bering, **range()** funksiyasi bilan sanoq asosida sonlarni ekranga chiqarmayapmiz, balki shuncha marta bir xil amalni bajaryapmiz:

```
for x in range(5):
    print("python")
else:
    print("\nSikl tugadi!")
```

Sikl ichida sikl

Sikl ichida sikl qo'llanganda ichki sikl tashqi siklning har bitta bosqichida bir martadan bajariladi. Hozir har bitta rangni har bir mashina bilan birgalikda qo'llab ko'ramiz:

```
rang = ["qora", "oq", "qizil"]
mashina = ["Spark", "Nexia", "Lacetti"]

for x in rang:
    for y in mashina:
        print(x,y)
```

pass

for sikli ham xuddi **while** sikli singari bo'sh bo'lishi mumkin emas. Ya'ni sikl davomida albatta nima amal bajarilishini kiritishimiz lozim. Ammo bu amal hali aniq bo'lmasa kodimizda xatolik yuz bermasligi uchun **pass** kalit so'zidan foydalanamiz va dastur ishga tushganda o'sha qism hisobga olinmay ketiladi. Masalan, hozir sikl davomida bajarilishi kerak bo'lgan amalni kiritmay pass kalit so'zini kiritamiz. Bunda xatolik yuz bermaydi, chunki pass kalit so'zi qo'yilgan. Ammo hech qanday amal ham bajarilmaydi, chunki biror amal bajarish haqida buyruq berilmagan.

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

IV-BOB. PYTHONDA FUNKSIYA VA MODULLAR

FUNKSIYALAR

Funksiya - bu bitta, bog'liq bo'lgan harakatni amalga oshirish uchun ishlatiladigan uyushgan, qayta ishlatilishi mumkin bo'lgan kodlar bloki. Funktsiyalar sizning arizangiz uchun yaxshiroq modullik va kodni yuqori darajada qayta ishlatilishini ta'minlaydi.

Siz allaqachon bilganingizdek, Python sizga **print ()** va shu kabi ko'plab ichki funktsiyalarni beradi. Pythonda mavjud bo'lgan tiplarni o'zgartiruvchi va ba'zi bir qo'shimcha funksiyalar bilan quyida tanishishingiz mumkin:

- **bool(x)-** rostlikka tekshirishni standart usulidan foydalanuvchi bool tipiga o`zgartirish. Agar x yolg`on bo`lsa yoki tushirib qoldirilgan bo`lsa, False qiymatini qaytaradi, aksincha esa True qaytaradi.
- **bytearray([manba, [kodlash[xatolar]]])-** bytearray ga o`zgartirish. Bytearray- 0<=x<256 diapazondagi butun sonlarni o`zgarmas ketma-ketligi. Konstruktor argumentlari bytearray() ga mos ko`rinishga ega bo`ladi.
- **complex([real],[image])-** kompleks songa o`zgartirish.
- **dict(object)-** lug`atga o`zg artirish.
- **float([x])**-haqiqiy songa o`zgartirish. Agar argument ko`rsatilmagan bo`lsa, 0.0 qaytaradi.
- **int([object],[asosiy sanoq sistemasi])-** butun sonni berilgan sanoq sistemasidan o`nlik sanoq sistemasiga o`tkazish.
- **list([object])**-ro`yxat tuzadi.
- **memoryview(object)-** memoryview obyektini tuzadi.
- **object()**-hamma obyektlarga asos bo`lgan bosh obyektni qaytaradi.
- **range([start=0], stop,[step=1])-** step qadamli start dan stop gacha bo`lgan arifmetik progressiya.
- **set(object)**-to`plamni yaratadi.
- **slice([start=0], stop, [step=1])**-step qadamga ega startdan stopgachaga bo`lgan kesma obekti.
- **tuple(obj)-** kortejga o`zgartiradi.

- **abs(x)-** absolyut raqamni (sonni modulini) qaytaradi.
- **all(ketmaketlik)-** agarda hamma elementlar haqiqiy bo`lsa (yoki ketmaketlik bo`sh bo`lsa) True ni qaytaradi.
- **any(ketmaketlik)-**agarda elementlardan hech bo`lmaganda bittasi haqiqiy bo`lsa True ni qaytaradi. Bo`sh ketmaketlik uchun False qaytaradi.
- **ascii(object)-** repr ga o`xshab obyekt ko`rinishiga mos qatorni ekranga xuddi shunday qaytaradi.
- **bin(x)-** butun sonni ikkilik sanoq sistemasiga o`tkazadi
- **chr(x)-** x ning Unicode ga mos belgini qaytaradi.
- **classmethod(x)-** sinf metodi ko`rsatgan funksiyani taqdim etadi.
- **compile(source, filename, mode, flags=0, don't_inherit=False)-** ketmaketlik eval yoki exec funksiyalari bilan bajariladigan dastur kodiga komplyatsiya qilinishi. Qator karetkani qaytaruvchi belgilar yoki nolga teng baytlarga ega bo`lmasligi kerak.
- **delattr(object, name)-** "name" nomidan atributni qaytaradi.
- **dir([object])-** obyekt nomlarining ro`yxati, agar obyekt ko`rsatilmagan bo`lsa, local maydondagi nomlar ro`yxati.
- **divmod(a,b) –** a ni b ga bo`lganda hosil bo`lgan bo`linmaning butun va qoldiq qismi.
- **enumerate(iterable, start=0)-** nomer va unga mos ketmaketlik a'zosidan tarkib topgan kortejni har bir o`tishda taqdim etuvchi iteratorni qaytaradi.
- **eval(expression, globals=None, locals=None)-** dastur kodi qatorini bajaradi.
- **filter(function, iterable)-** function yordamida rost qiymatni elementlarga qaytaruvchi iteratorni qaytaradi.
- **format(value [,format_spec])-** formatlash (qatorni formatlash).
- **getattr(object, name,[default])-** obyekt atributini yoki default.globals()-global nomlar lugatini chiqaradi.
- **hasattr(object, name)-** "name" nomidagi atribut obyektga ega ekanligini tekshiradi.
- **hash(x)-** ko`rsatilgan obyektning heshini qaytaradi.
- **help([object])-** dasturni yordam qismiga kiritilgan ma'lumotnoma tizimini chaqirish.
- **hex(x)-** butun sonni o`n oltilik sanoq sistemasiga o`tkazish.
- **id(object)-**obyekt manzilini qaytaradi .
- **input([promt])-** foydalanuvchi tomonidan kiritilgan qatorni qaytaradi. Promt-foydalanuvchiga yordam.
- **isinstance(object, ClassInfo)-**agarda obyekt classinfo yoki uning sinfosti ekzemplari bo`lsa rost qiymat qaytaradi. Agarda ekzemplar berilgan tipdagi obyekt bo`lmasa, funksiya yolg`on qiymat qaytaradi.

- **issubclass(sinf, ClassInfo)**-agarda sinf ClasssInfo sinfostisi bo`lsa rost qiymat qaytaradi. Sinf o`z-o`ziga sinfosti bo`ladi.
- **iter(x)**- iterator obyektini qaytaradi.
- **len(x)**-ko`rsatilgan obektni elementlar sonini qaytaradi.
- **locals()**-lokal nomlar lug`ati.
- **map(function, iterator)**-ketmaketlikning har bir elementiga function funksiyasini qo`llash orqali yaratiladigan iterator.
- **max(iter,[args…]*[, key])**-ketma-ketlikning maksimal elementi.
- **min(iter,[args…]*[, key])**-ketmaketlikning minimal elementi.
- **next(x)**-iteratorning keyingi elementini qaytaradi.
- **oct(x)**- butun sonni sakkizlik sanoq sistemasiga o`tkazadi.
- **open(file, mode='r', buffering=None, encoding=None, errors=None, newline=None, closefd=True)**- faylni ochadi va kerakli oqimni qaytaradi.
- **ord(x)**- belgi kodi.
- **pow(x, y[,r])**-(x**y)%r.
- **reversed(object)**-yoyilgan obyektning iteratori.
- **print([object,…],*,sep=" ", end='/n', file=sys.stdout)**- ma'lumotlarni ekranga chop etish.
- **round(X,[N])**- verguldan keyin N- belgilargacha to`g`rilash.
- **setattr(obekt, nom, qiymat)**- obyekt atributini belgilash.
- **sorted(iterable[, key][, reverse])**- tartiblangan ro`yxat.
- **staticmethod(function)**- funksiya uchun statistik metod.
- **sum(iter, start=0)**-ketmaketlik elementlarini yig`indisi.
- **type(object)**- obyekt tipini qaytaradi.
- **type(name, bases, dict)**- name sinfidagi yangi ekzemplarni qaytaradi.
- **vars([object])**- obyekt atributlarining ro`yxati. Jimlik holatida- local nomlar lug`ati.

Biz hozir yuqorida Python dasturida kiritilgan funksiyalar bilan tanishdik. Ammo lekin siz o'zingizning funksiyalaringizni ham yaratishingiz mumkin. Ushbu funksiyalar foydalanuvchi tomonidan belgilangan funksiyalar deb ataladi.

Funksiya koddagi bir blok hisoblanadi. U faqat chaqirilgandagina ishlaydi. Ya'ni qandaydir funksiya tuzilgan, ammo uni hali ishlashiga buyruq bermasak kodimiz ishga tushganda bu funksiya bajarilmaydi.

Funksiyaga ma'lumotlar uzatishimiz mumkin va bu ma'lumotlar funksiya uchun parametrlar hisoblanadi. Funksiya bizga ma'lumotlarni natija sifatida qaytarishi mumkin.

Funksiyalarni hosil qilish

Funksiyalar **def** kalit so'zi bilan hosil qilinadi. **def** so`zidan so`ng **funksiya nomi** va qavs ichida **formal parametrlar ro`yxati** ko`rsatiladi. Funksiya tanasini hosil qiluvchi instruksiyalar keyingi qatordan boshlab bo`sh joy(отступь) bilan yoziladi. Quyidagi kodimiz ishga tushsa, bizga hech qanday natija bermaydi. Chunki biz faqat funksiya hosil qilgan bo'lamiz:

```
def my_func():
    print("Funksiya ishga tushdi")
```

Funksiyani chaqirish

Avval aytganimizdek funksiya faqat chaqirilgandagina ishlaydi. Uni chaqirish uchun funksiyaning nomi qavslar bilan yozamiz. Yuqoridagi kodimiz natija berishi uchun o'sha funksiyani chaqiramiz va funksiya ishga tushadi:

```
def my_func():
    print("Funksiya ishga tushdi")

my_func()
```

```
Funksiya ishga tushdi
```

Argumentlar

Funksiyada ma'lumotlar argumentlar orqali uzatiladi. Argumentlar funksiya hosil qilayotganda funksiya nomidan so'ng qavslar ichiga kiritiladi. Argumentlar bir emas bir nechta bo'lishi mumkin. Bunday holatda ularni vergul bilan ajratib yoziladi.

Quyidagi misolimizda bizda ism degan argument bor. Funksiya hosil qilinganda argumentni qayerda qo'llash kerakligini ko'rsatamiz. Funksiyani chaqirayotganda esa o'sha argument o'rnida qanday qiymat bo'lishi kerakligini ko'rsatamiz:

```
def my_func(ism):
    print(ism + " Hamidov")

my_func("Mahmud")
my_func("Shahzod")
my_func("Odil")
```

```
Mahmud Hamidov
Shahzod Hamidov
Odil Hamidov
```

Funksiya tuzilayotganda nechta argument bilan tuzilsa, chaqirilayotganda ham shuncha argument bilan chaqirilishi kerak. Aks holda xatolik yuz beradi.

Masalan, quyidagi misolimizda ikkita – ism va familiya argumentli funksiya tuzamiz va uni shu ikkita argument orqali chaqiramiz:

```
def my_func(ism,  familiya):
    print(ism + " " + familiya)

my_func("Abbosbek", "Ibragimov")
```

```
Abbosbek Ibragimov
```

*args

Bir argument orqali bir nechta qiymatlarda foydalanmoqchi bo'lsak, funksiya tuzilayotgan vaqtda argument nomi oldidan * belgisi qo'yiladi. Bu usul bilan ko'proq qiymatlar to'plamiga ega bo'lamiz va bir argumentni bir nechta qiymatlar bilan ishlatishimiz mumkin.

```
def mevalar(*meva):
    print(meva[0] + "," + meva[2])

mevalar("anjir", "gilos", "uzum")
```

```
anjir,uzum
```

Qiymat qaytarish

Funksiyalar vazifasiga ko'ra ikki turga bo'linadi. Bular qiymat qaytaradigan va qiymat qaytarmaydigan funksiyalar. Biz yuqorida hosil qiligan funksiyalarimiz bu qiymat qaytarmaydigan funksiyalar hisoblanadi. Endi esa qiymat qaytaruvchi funksiyalar hosil qilish bilan tanishamiz. Qiymat qaytaruvchi funksiyalar hosil qilish uchun **return** so'zidan foydalanamiz.

Masalan, istalgan sonning kvadratini chiqaruvchi funksiya tuzsak:

```
def kvadrat(x):
    return x*x

print(kvadrat(5))
```

```
25
```

Bunda yuqoridagi dasturga e'tibor bersangiz funksiya **return** kalit so'zi orqali x argumentning ikkinchi darajasini ya'ni kvadratini qaytarmoqda va bu shuning uchun ham biz yaratgan fuksiya qiymat qaytaruvchi funksiyaga misol bo'la oladi.

LAMBDA FUNKSIYA

Lambda funksiyasi kichik anonim funksiya hisoblanadi. Unda istalgancha argument qatnashishi mumkin va barchasi bir ifodada yoziladi. Hozir kiritilgan sonni 10 ga oshiradigan **lambda** funksiya hosil qilamiz:

```
x = lambda a: a + 10
print(x(2))
```

```
12
```

Endi ikki va uch argumetli lambda funksiyalarini tuzami. Avvalgisi ikki sonning o'zaro ko'paytmasini, keyingisi esa barcha sonlar yig'indisini topadi.

```
x = lambda a, b : a*b
print(x(5,6))

y = lambda a, b, c : a+b+c
print(y(7,9,5))
```

```
30
21
```

Nega lambda funksiya ishlatamiz ?

Lambda fuksiyalarni funksiya ichida boshqa bir anonim funksiya sifatida ishlatish qulay. Masalan, bir argumentli funksiya bor va uning argumenti noma'lum bir songa ko'payadi. Shu funksiyani lambda funksiya yordamida istalgan sonni ikkilantiradigan va uchlantiradiga funksiyaga aylantiramiz.

```
def myfunc(n):
    return lambda a: a*n

ikkilantir = myfunc(2)
uchlantir = myfunc(3)

print(ikkilantir(5))
print(uchlantir(5))
```

```
10
15
```

PYTHONDA MODULLARDAN FOYDALANISH

Modul – bu biz yozgan kodimizning fayl ko'rinishi. Bitta katta dasturimiz ko'pgina modullardan tashkil topishi mumkin. Pythonda modul hosil qilish uchun yozga kodimizning **.py** fayl kengaytmasi bilan saqlashimiz kerak bo'ladi.

Masalan, quyidagi salomlash funksiyasi yozilgan kodimizni **salom.py** nomi bilan saqlaymiz va shu nomli modul hosil bo'ladi:

```
def salomlashish (ism):
    print("Salom"+ ism)
```

Moduldan foydalanish

Endi biz katta dastur tuzishni boshlar ekanmiz, biz barcha kodni bir modulga yozib ishlatishimiz noqulay bo'ladi. Shuning uchun umumiy dasturni modullarga bo'lib ishlatganimiz ma'qul. Bu ancha qulay bo'ladi. Chunki biror moduldagi funksiya yoki ma'lumot kerak bo'lsa uni qayta-qayta yozib o'tirmasdan o'sha modulning o'zidan olib ishlatishimiz mumkin. Masalan, yuqorida funksiya tuzib uni **salom.py** moduliga saqlab qo'ydik. Endi biz yangi modul ochib unda **salom.py** modulidagi funksiyani ishlatamiz. Buning uchun **import** kalit so'zi bilan **salom.py** modulini chiqaramiz. So'ngra undagi **salomlash()** funksiyasini olib ishlatamiz.

E'tibor bering, bu yerda shunchaki funksiya nomini yozib ishlatayapmiz. Funksiyaning o'zi esa biz chaqirgan modulda tuzilgan:

```
import salom

salom.salomlash("Abbosbek")
```

```
Salom Abbosbek
```

Modulda o'zgaruvchilar

Modullar nafaqat funksiya, balki o'zgaruvchilarni ham o'z ichiga olishi mumkin. Shu sababli bir moduldagi ma'lumotdan boshqa modullarda ham foydalanish mumkin. Masalan, **avto** nomli dictionary o'zgaruvchisini **mashina.py** moduliga saqlaymiz:

```
avto = {
    "brend": "Audi",
    "model": "R8",
    "rang": "kumush",
    "yil" : 2018
}
```

Endi yangi modul ochamiz va unda mashinaning modelini ekranga chiqarishi buyuramiz:

```
import mashina

a = mashina.avto["model"]
print(a)
```

```
R8
```

Modulni nomlash

Biz murojaat qilgan modulning nomi uzunroq bo'lsa, uni kodimizda keyinchalik bu uchun nom bilan ishlatishimiz biroz noqulay bo'ladi. Ammo bizda uni kod ichida o'zimiz uchun qulay nom bilan ishlatish imkoniyati bor. Buning uchun modulni chaqirayotgan vaqtda uni pass kalit so'zi bilan o'zimizga qulay qilib nomlab olamiz. Natijada, kod ichida uni shu qulay nom bilan ishlatishimiz mumkin bo'ladi.

Masalan, avvalroq biz saqlagan **mashina.py** moduliga murojaat qilamiz va uni o'zimiz uchun m sifatida belgilaymiz. So'ngra uni shu nom bilan ishlatamiz:

```
import mashina  as m

a = m.avto["model"]
print(a)
```

```
R8
```

dir() funksiyasi

dir() maxsus funksiyasi istalgan modulga tegishli barcha funksiya yoki o'zgaruvchilar ro'yxatini chiqarib beradi. Xoh u maxsus modul bo'lsin, xoh o'zimiz tuzgan, barchasi uchun amal qiladi.

Masalan, Pythonda matamatik hisob-kitoblar uchun **math** moduli mavjud. Undagi barcha funksiyalar ro'yxati kerak bo'lsa, uni quyidagicha ekranga chiqaramiz:

```
import math

x = dir(math)
print(x)
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copy
sign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm
1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd'
, 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'ldex
p', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi
', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', '
tanh', 'tau', 'trunc']
```

Keraklisini olish

Tasavvur qiling biz tuzgan biror modulda ko'p funksiya yoki o'zgaruvchilar bor. Bizga ulardan faqat bittasi kerak. Qolganlari shart emas. Bunday vaziyatda butun boshli modulni olmasdan, shunchaki undagi kerakli funksiyani olishimiz mumkin.

Tushunish uchun bir modul tuzamiz, uning nomi **suhbat.py** bo'lsin. Unda ikkita funksiya tuzamiz. Bular **salom()** va **xayr()** funksiyalari bo'ladi:

```
def salom(ism)
    print("Salom "+ism)

def xayr(ism):
    print("Xayr "+ism)
```

Endi yangi modul ochamizda, **suhbat.py** modulidagi faqat **xayr()** funksiyasini olib ishlatamiz:

```
from suhbat import xayr

xayr ("Abbosbek")
```

```
Xayr Abbosbek
```

E'tirbor bering: ilgari biror modulni chaqirganimizdan keyin undagi funksiyalar oldidan shu modul nomini qo'yib, so'ng nuqta (.) va kerakli funksiyani yozar edik. Agar o'sha moduldan ayna bir funksiyaning o'zini chaqirsak, shunchaki funksiya nomi yozilib ishlatiladi.

Masalan: **suhbat.xary("**Madaminbek**")** emas, shunchaki **xayr("Madaminbek")** tarzida yoziladi.

PYTHONDA MAXSUS MODULAR

Pythonda ayrim narsalarga mo'ljallangan tayyor, maxsus modulla bor. Ularning har birining o'z vazifasi bor va biz o'zimizga kerak o'rinda ularga murojaat qilib ishlatamiz. Bunday modular ro'yxati, ularning vazifalari va ularni qo'llash haqida yana qo'shimcha adabiyotlardan olishingiz mumkin.

Standart kutubxonalar

Python tili standart kutubxonasining modullarini shartli ravishda mavzular bo`yicha quyidagi guruhlarga ajratish mumkin:

- Bajarish davri servislari. Modular: sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, intertools, locale, gettext.
- Siklni qayta ishlashni qo`llab-quvvatlovchi. Modullar: pdb, hotshot, profile, unittest, pydoc. Paketlar: docutils, distutils.
- OS (fayllar, protseslar) bilan ishlash. Modullar: os, os.path, getopt, glob, popen2, shutil, select, signal, stat, tempfile.
- Matnlarni qayta ishlovchi. Modullar: string, re, StringIO, codecs, difflib, mmap, sgmllib, htmllib, htmlentitydefs. Paket: xml.
- Ko`p oqimli hisoblashlar. Modullar: threading, thread, Queque.
- Ma'lumotlarni saqlash. Arxivlash. Modullar: pickle, shelve, anydbm, gdbm, gzip, zlib, zipfile, bz2, csv, tarfile.
- Platformaga tobe modullar. UNIX uchun: commands, pwd, grp, fcntl, resource, termios, readline, rlcompleter. Windows uchun: msvcrt, _winreg, winsound.
- Tarmoqni qo`llab-quvvatlash. Internet protokollari. Modullar: cgi, Cookie, urllib, urlparse, htttplib, smtplib, poplib, telnetlib, socket, asyncore. Serverlarga misollar: SocketServer, BaseHTTPServer, xmlrpclib, asynchat.
- Internetni qo`llab-quvvatlash. Ma'lumotlar formatlari. Modullar: quopri, uu, base64, binhex, binascii, rfc822, mimetools, MimeWriter, multifile, mailbox. Paket: email.
- Python uchun. Modullar: parser, symbol, token, keyword, incpect, tokenize, pyclbr, py_compile, compileall, dis, compiler.
- Grafik interfeys. Modul: Tkinter.

Ko`pincha modullar o`zida bir yoki bir nechta sinflarni saqlaydilar. Bu sinflar yordamida kerakli tipdagi obyekt yaratiladi, lekin gap moduldagi nomlar haqida emas, aksincha shu obtekt atributi haqida boradi. Bir nechta modullar faqat erkin obyetlar ustida ishlash uchun umumiy bo`lgan funksiyalardan iborat bo`ladilar.

Platform moduli

Hozir biz **platform** modulini ishlatamiz. Bu modul bilan biz qaysi operatsion sistemada ishlayotganimizni bilish mumkin. Masalan, deylik biz **Windows** operatsion tizimidagi kompyuterni ishlatmoqdamiz. Demak kodni ishga tushirsak, ekranga **Windows** yozuvi chiqib keladi.

```
import platform

x = platform.system()
print(x)
```

```
Windows
```

Pickle moduli

Pythonning **pickle** moduli yordamida har qanday obyektni faylga saqlash va keyinchalik fayldan o'qib olish mumkin. Bunday imkoniyat ob'yektlarni uzoq muddat saqlashda qo'l keladi.

```
import pickle
# obyektni saqlash fayli

shoplistfile = 'shoplist.data'

# xaridlar ro'yxati

shoplist = ['olma', 'mango', 'sabzi']

# faylga yozish

f = open(shoplistfile, 'wb')

pickle.dump(shoplist, f) # obyektni faylga yozamiz

f.close()

del shoplist # shoplist o'zgaruvchisini o'chirib tashlaymiz

# fayldan o'qish

f = open(shoplistfile, 'rb')

storedlist = pickle.load(f) # ob'yektni fayldan yuklab olish

print(storedlist)
```

Natija:

```
['olma', 'mango', 'sabzi']
```

Bu misolda obyektni faylga yozish uchun birinchi galda faylni binar yozish **(“wb”)** rejimida ochilyapti, so’ng pickle modulining dump funksiyasi chaqirilyapti. Bu jarayon “konservatsiya” (“pickling”) deyiladi. Shundan so’ng obyektni fayldan o’qib olish uchun pickle modulining **load** funksiyasidan foydalanilyapti.

Sys moduli

Sys moduli Python interpretatorida dasturni bajaruvchi muhitdir. Quyida bu modulni eng ko`p qo`llaniladigan obyektlari keltilgan:

- **Exit([c])-** dasturdan chiqish. Tugatishning raqamli kodini yuborish mumkin: agarda dasturni tugatish muvafaqqiyatli amalga oshsa 0 ni yuboradi, aksincha bo`lsa ya’ni xatolik yuz bersa boshqa raqamlarni yuboradi.
- **Argv-** buyruqlar qatori argumentlari ro`yxati. Oddiy holatda sys.argv[0] buyruqlar qatoriga ishga tushirilgan dastur nomini va boshqa parametrlar yuboriladi.
- **Platform-** interpretator ishlaydigan platforma.
- **Stdin, stdout, stderr-** standart kiritish, chiqarish, xalolarni chiqarish. Ochiq faylli obyektlar.
- **Version-** interpretator versiyasi.
- **Serecursionlimit(limit)-** rekursiv chaqirishlarni maksimal kiritish darajasini o`rnatadi.
- **Exc_info()-**kiritish-chiqarish istisnosi haqida ma’lumot.

Copy moduli

Bu modul obyektlarni nusxalashga mo`ljallangan funksiyalarga ega. Boshida Pyhtonda sal sarosimaga solish uchun “paradoks” ni ko`rib chiqish tavsiya etiladi.

```
list1 = [0, 0, 0]
list = [list1] * 3
print(list)
list[0][1] = 1
print (list)
```

Va biz kutmagan natija paydo bo`ladi:

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[[0, 1, 0], [0, 1, 0], [0, 1, 0]]
```

Gap shundaki bu yerda lst ro`yxati shu ro`yxatnining izohiga ega. Agarda rostdan ham ro`yxatni ko`paytirmoqchi bo`lsak, copy modulidagi **copy()** funksiyasini qo`llash kerak.

```
from copy import copy
lst1 = [0, 0, 0]
lst = [copy(lst1) for i in range(3)]
print (lst)
lst[0][1] = 1
print (lst)
```

Endi kutilgan natija paydo bo`ladi:

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[[0, 1, 0], [0, 0, 0], [0, 0, 0]]
```

Copy modulida yuqori aniqalikda nusxalash uchun **deepcopy()** funksiyasi bor bu funksiya yordamida obektlar butun imkoniyati bilan rekursiv nusxalanadi.

Random moduli

Bu modul har xil taqsimotlar uchun tasodifiy raqamlarni generatsiya qiladi. Eng ko`p qo`llaniladigan funksiyalari:

- **Random()-**[0.0, 1.0) yarim ochiq diapozondagi tasodifiy sonlarni generatsiya qiladi.
- **Choice(s)-** s ketma- ketlikdan tasodifiy elementni tanlab oladi.
- **Shuffle(s)-** s o`zgaruvchan ketma-ketlik elementlarini joyiga joylashtiradi.
- **Randrange([start], stop, [step])-** renge(start, stop, step) diapozondagi tasodifiy butun raqamni chiqaradi. Choice(range(start, stop, step)) ga analogik holatda.
- **Normalvariate(mu, sigma)-** normal holatda taqsimlangan ketma-ketlikdan raqamni chiqaradi. Bu yerda mu- o`rtacha, sigma-o`rta kvadratli (sigma>0) sonlar.

Boshqa funksiyalar va uning parametrlarini hujjatlashdan aniqlab olish mumkin. Modulda qandaydir holatga tasodifiy raqamlar generatorini joylashtirishga imkon beruvchi **seed(n)** funksiyasi ham mavjud. Masalan: agarda bitta tasodifiy raqamlar ketma-ketligidan ko`p marta foydalanishga ehtiyoj sezilsa.

Os moduli

Os moduli-har xil operatsion sistemalarning o`ziga xos xususiyatlari bilan ishlovchi kategoriyadagi asosiy modul hisoblanadi. Bu modul funksiyalari ko`plab operatsion sistemalarda ishlaydilar. Kataloglarni bo`luvchi os moduli va u bilan bog`liq bo`lgan ifodalar konstanta ko`rinishida berilgan.

Konstanta	Vazifasi
Os.curdir	Joriy katalog
Os.pardir	Bosh katalog
Os.sep	Yo`lning elementlarini taqsimlovchi
Os.altsep	Boshqa yo`lning elementlarini taqsimlovchi
Os.pathsep	Yo`llar ro`yxatidagi yo`llarni taqsimlovchi
Os.defpath	Yashirin yo`llar ro`yxati
Os.linesep	Satrni yakunlovchi belgi

Kataloglarni bo`luvchi os moduli ifodalari konstanta ko`rinishida

Pythondagi dastur operatsion tizimda alohida jarayon ko`rinishida ishlaydi. Os modulining funksiyalari protsesda, muhitda bajariladigan turli xildagi ahamiyatga ega bo`lgan kirishlarga ruxsat etadilar. Os modulining eng muhim ruxsat etuvchi obyektlaridan biri deb environ o`rab oluvchi muhiti o`zgaruvchilarning lug`ati hisoblanadi. Masalan o`rab oluvchi muhit o`zgaruvchilar yordamida web server CGI-ssenariyga bir qancha parametrlarni o`tkazadi.

Quyidagi misolda PATH o`rab oluvchi muhiti o`zgaruvchini olish mumkin:

```
import os
PATH=os.environ['PATH']
```

Funksiyalarning katta qismi fayllar va kataloglar bilan ishlashga mo`ljallangan. Quyida **UNIX** va **Windows** OT lar uchun ruxsat etilgan funksiyalar taqdim etilgan:

- **Access(path, flags)**- path nomli fayl yoki catalog ruxsat etish(доступь) ni tekshiradi. Buyurma qilishga rucsatning tartibi flags raqami bilan belgilanadi. U esa yaratilgan kombinatsiyalar os.F_OK (fayl mavjud), os.R_OK (fayldan o`qish mumkin), os.W_OK (faylga yozish mumkin) va os.X_OK (fayllarni bajarishni, katalogni ko`rib chiqish mumkin) bayroqlari bilan belgilash mumkin.
- **Chdir(path)**- path ni joriy ishchi katalog qiladi.
- **Getcwd()**- joriy ishchi catalog.
- **Chmod(path, mode)**- mode ga path bo`lgan ruxsat etish rejimini belgilaydi. Ruxsat etish tartibi bayroqlarni kombinatsiya qilib belgilashi mumkin. Bu ishda chmod() harakatda bo`lgan tartibni to`ldirmaydi, uni yangidan belgilamaydi, uni yangidan belgilaydi.
- **Listdir(dir)**- dir katalogidagi fayllar ro`yxatini qaytaradi. Ro`yxatga maxsus belgilar “.” va “..” kirmaydi.
- **Mkdir(path [, mode])**- path katalogini tuzadi. Jimlik holatida mode tartibi 0777 ga teng bo`ladi, bu degani S_IRWXU|S_IRWXG|S_IRWXO agarda stat moduli konstantalari bilan foydalansak.

```
import os
os.mkdir('C:\Users\Guljakhon\Desktop\Новая папка\katalog\dir2')
#ko`rsatilgan manzilda dir2 nomli yangi katalog yaratadi.
```

```
import os
os.mkdir('./dir2')
#joriy manzilda dir2 nomli yangi catalog yaratadi.
```

- **Makedirs(path [,mode])-** hamma kataloglarni yaratuvchi, agarda ular mavjud bo`lmasalar mkdir() analogi oxirgi katalog mavjud bo`lgandan so`ng mustasnoni ishga tushiradi.
- **Remove(path), unlink(path)-** path katalogini yo`qotadi. Kataloglarni yo`qotish uchun rmdir() va removedirs() dan foydalanadi.
- **Rmdir(path)-** path nomli bo`sh katalogni yo`qotadi.
- **Removedirs(path)-** birinchi bo`sh bo`lgan kataloggacha pathni yo`q qiladi. Agarda yo`lda eng oxirgi kiritilgan katalog osti bo`sh bo`lmasa OSError mustasnosini ishga tushiradi.
- **Rename(src, dst)-** src fayli yoki katallogini dst deb qayta nomlaydi.
- **Renames(src, dst)-** rename() analogi dst yo`li uchun kerakli kataloglarni yaratadi va src yo`lining bo`sh kataloglarini yo`qotadi.
- **Stat(path)-** path haqidagi malumotni o`nta elementlik kortej shaklida qaytaradi. Kortej elementlariga kirish uchun stat moduli konstantalaridan foydalanish mumkin. Masalan stat.ST_MTIME (faylning oxirgi modifikatsiyasi vaqti).
- **Utime(path, times)-** oxirgi modifikatsiya (mtime) va faylga kirishga ruxsat(atime) larini belgilaydi. Boshqa holatlarda times ikki elementli kortej (atime, mtime) sifatida ko`rib chiqiladi. Qaysidir faylni atime va mtime ni olish uchun stat() va stat modulining konstantalarini barobar ishga tushirib olish mumkin.
- Os moduli protsesslar bilan ishlash uchun quyidagi funksiyalarni taqdim etadi (ular ham UNIX hamda windowsda ishlaydilar).
- **System(cmd)-** alohida oynada cmd buyruqlar satrini bajaradi. U C tilining system kutubxonasi chqirig`iga analogik bo`ladi. Qaytarilgan qiymat foydalanadigan platformadan tobe bo`ladi.
- **Times()-** beshta elementdan iborat bo`lgan kortejni qaytaradi. U ish jarayoni vaqtini lahzalarda ko`rsatadi, qo`shimcha protsesslar vaqtini, qo`shimcha protsesslarning axborot tizimlari vaqtini, va o`tgan zamonda qotib qolgan vaqtni ko`rsatadi (masalan tizim ishga tushgan paytdan).
- **Getloadavg()-** coo, uchta qiymatlik kortejni qaytaradi.

SANA VA VAQT (DATETIME MODULI)

Sana yoki vaqt bilan ishlash uchun **datetime** modulini ishga solamiz. Bu modul bilan yildan tortib millisekundlargacha ma'lumot olish mumkin.

Masalan, quyidagi kodni ishga tushirsak, joriy vaqtni ko'rsatadi. Bunda to'liq holda, ya'ni yil, oy, kun, soat, minut, sekund, millisekundlar ko'rinadi:

```
import datetime as dt

x = dt.datetime.now()
print(x)
```

```
2020-11-14 14:00:56.561937
```

O'lchov turlari

datetime modulida juda ko'p funksiyalar bor. Ular orqali vaqtni nafaqat to'liq holda, balki faqatgina bizga kerakli holda ham aniqlashimiz mumkin. Masalan, bizga faqat joriy yil yoki bugungi hafta kuni kerak. Bularning alohida funksiyalari bor. Quyidagi kodimizda dastlab faqat joriy yilni so'ngra bugungi hafta kunini ekranga chiqaramiz:

```
import datetime as dt

x = dt.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

```
2020
Saturday
```

Vaqtni belgilash

Biz o'zimiz sana yoki vaqtni belgilashimiz mumkin. Buning uchun kerakli sonlarni ko'rsatsak kifoya. Masalan, hozir sanani 28-noyabr 2020-yil deb belgilaymiz:

```
import datetime as dt

x = dt.datetime(2020, 11, 28)

print(x)
```

```
2020-11-28 00:00:00
```

strtime() funksiyasi

strtime() funksiyasi vaqtga oid turli ma'lumotlarni turli formatlarda bizga qaytaradi. Ushbu fuksiyani maxsus format kodlari bilan qo'llash kerak. Masalan, joriy oyning nomini ko'rsatish kodi **%B**. Demak hozirgi oy nomini ekranga chiqarish quyidagicha bo'ladi.

```
import datetime as dt

x = dt.datetime.now()

print(x.strftime("%B"))
```

```
November
```

Quyida maxsus format kodlariga misollar keltiramiz. Ularni xuddi yuqoridagi kod singari sinab ko'rsangiz tushunish yanada osonroq bo'ladi:

- **%a** – hafta kuni (qisqa)
- **%A** – hafta kuni (to'liq)
- **%w** – hafta kuni (raqam shaklida)
- **%d** – oyning sanasi
- **%b** – oy nomi (qisqa)
- **%B** – oy nomi (to'liq)
- **%m** – oy (raqam ko'rinishida)
- **%y** – yil (qisqa)
- **%Y** – yil (to'liq)
- **%H** – soat (00-23)
- **%I** – soat (00-12)
- **%p** – kun vaqti (AM/PM)
- **%M** – minut (00-59)
- **%S** – sekund (00-59)
- **%j** – yildagi kun raqami (001-366)
- **%U** – yildagi hafta raqami, Yakshanba birinchi kun sifatida (00-53)
- **%W** – yildagi hafta raqami, Dushanba birinchi kun sifatida (00-53)
- **%c** – mahalliy sana va vaqt
- **%x** – mahalliy sana
- **%X** – mahalliy vaqt

PYTHONDA MATEMATIKA

Amallar bajarilish ketma-ketligi

2 + 3 * 4 ifodada qaysi amal birinchi bajariladi: qo'shishmi yoki ko'paytirish?

Matematika fanida ko'paytirish birinchi bajarilishi ko'rsatilgan. Demak, ko'paytirish operatori qo'shish operatoriga qaraganda katta prioritetga(muhimlik darajasiga) ega. Quyidagi jadvalda Python operatorlari prioriteti ko'rsatilgan. Bunda yuqoridan pastga qarab Python operatorlari prioriteti oshib boradi. Bu shuni anglatadiki, ixtiyoriy ifodada Python oldin eng quyidagi operatorlarni hisoblaydi va keyin esa yuqoridagilarini. Amaliyotda esa amallarni qavslar bilan aniq ajratish tavsiya etiladi. Bu dastur kodini oson o'qishga yordam beradi.

Operator	Izoh
Lambda	lambda ifoda
Or	Mantiqiy 'yoki'
And	Mantiqiy 'va'
Not x	Mantiqiy 'emas'
in, not in	Tegishlilikni tekshirish
is, is not	Bir xillikni tekshirish
<, <=, >, >=, !=, ==	Taqqoslash
|	'yoki' bit operatori
^	'shartlik yoki' bit operatori
&	'va' bit operatori
<<, >>	Surilishlar
+, –	Qo'shish va ayirish
*, /, //, %	Ko'paytirish, bo'lish, qoldiqsiz bo'lish va qoldiqlik bo'lish
+x, -x	Musbat va manfiy
~x	'emas' bit operatori
**	Darajaga ko'tarish
x.attribute	Atributga link
x[index]	Indeks bo'yicha murojat

x[index1:index2]	Kesib olish
f(argumentlar …)	Funksiyani chaqirish
(ifoda, …)	Kortej (Связка или кортеж)
[ifoda, …]	Ro'yxat (Список)
{kalit:qiymat, …}	Lug'at (Словарь)

Bu jadvalda bir xil prioritetga ega bo'lgan operatorlar bir qatorda joylashgan. Misol uchun '+' va '-'.

Hisoblash tartibini o'zgartirish

Ifodalarni o'qishni osonlashtirish uchun qavslarni ishlatish mumkin. Misol uchun, **2 + (3 * 4)** ni tushunish oson operatorlar prioriteni bilish lozim bo'lgan **2 + 3 * 4** ifodadan ko'ra. Qavslarni o'ylab ishlatish kerak. Ortiqcha qavslarni ishlatishdan saqlaning. Misol uchun: **(2 + (3 * 4)).**
Qavslarni ishlatishni ya'na bir afzalligi hisoblash tartibini o'zgartirish imkonini beradi. Misol uchun, qo'shish amalini ko'paytirish amalidan birinchi bajarish kerak bo'lsa, quyidagicha yozish mumkin: **(2 + 3) * 4.**

Pythonda matematik hisob-kitoblar uchun o’zining maxsus funksiyalariga ega. Bu funksiyalar tayyor holda bo’lib, kerakli natijalarni tezda chiqarib beradi.

min() funksiyasi berilgan sonlar ichida eng kichigini, **max()** funksiyasi esa eng kattasini aniqlaydi.

```
x = min(3, 8, 11)
y = max(3, 8, 11)

print(x)
print(y)
```

```
3
11
```

abs() funksiyasi sonning absolyut qiymatini aniqlaydi.

pow(x,y) funksiyasi **x** ni **y** darajaga ko’taradi.

```
x = abs(-5)
y = pow(5, 3)

print(x)
print(y)
```

```
5
25
```

Math va cmath moduli

Yuqoridagi ko'rganlarimiz Pythondagi ichki funksiyalar edi. Ularni to'g'rida-to'g'ri ishlatish mumkin. Ammo boshqa bir guruh matematik funksiyalar borki ular **math** moduliga mansub. Shuning uchun ularni ishlatishdan avval math moduliga murojaat qilamiz. Masalan, biror sonning kvadrat ildizini hisoblamoqchimiz. Buning uchun maxsus **sqrt()** funksiyasi mavjud. Uning ishlatilishi uchun **math** moduliga murojaat qilamiz:

```
import math

x = math.sqrt(64)
print(x)
```

```
8.0
```

Math va cmath modullarida haqiqiy va compleksli argumentlar uchun matematik funksiyalar to`plangan. Bu C tilida foydalaniladigan funksiyalar. Quyida math modulining funksiyalari keltirilgan. Qayerda z harfi bilan argumentga belgilash kiritilgan bo`lsa, u cmath modulidagi analogik funksiya ham shunday belgilanishini bildiradi.

- **acos(z)-**arkkosinus z.
- **asin(z)-** arksinus z.
- **atan(z)-** arktangens z.
- **atan2(y, x)-** atan(y/x).
- **ceil(x)-** x ga teng yoki katta eng kichik butun son.
- **cos(z)-** kosinus z.
- **cosh(x)-** giperbolik x kosinusi.
- **e-** e konstantasi.
- **exp(z)-** eksponenta (bu degani e**z)
- **fabs(x)-**x absolute raqami.
- **floor(x)-** xga teng yoki kichik eng katta butun son
- **fmod(x,y)-** x ni y ga bo`lgandagi qoldiq qism.
- **frexp(x)-** mantisa va tartibni (m, i) juftligi kabi qaytaradi, m- o`zgaruvchan nuqtali son, i esa- x=m*2**i ga teng butun son bo`ladi. Agarda 0-(0,0) qaytarsa boshqa paytda 0.5<=abs(m)<1.0 bo`ladi.
- **factorial(x)-** x ning faktoriali. N!=1*2*3*…*n
- **hypot(x,y)-** sqrt(x*x+y*y)
- **ldexp(m,i)-** m*(2**i).
- **log(z)-** natural logarifm z.
- **log10(z)-** o`nlik logarifm z.

- **log2(z)**-logarifm ikki asosga ko`ra z.
- **modf(x)-** (y,q) juftlikda x ning butun va kasr qismini qaytaradi.
- **pi-**pi konstantasi.
- **pow(x,y)-** x**y.
- **sin(z)-** z ning sinusi.
- **sinh(z)-** z ning giperbolik sinusi.
- **sqrt(z)-** z ning kvadrat ildizi.
- **tan(z)-** z ning tangensi.
- **tanh(z)-** z ning giperbolik tangensi.
- **trunc(x)-** x haqiqiy sonning butun qismini qaytaradi.
- **degrees(x)-**x ni radiandan gradusga o`tkazish.
- **radians(x)-** x ni gradusdan radianga o`tkazish.

math.ceil() funksiyasi eng yaqin yuqori butun songacha yaxlitlaydi.

math.floor() funksiyasi esa eng yaqin pastki butun songacha yaxlitlaydi.

Quyidagi misolimizda birinchi funksiyaning natijasi 2, keyingisi esa 1 bo'ladi:

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x)
print(y)
```

```
2
1
```

Matematikadagi **PI** sonining qiymati pythondagi konstantalar ro'yxatida bor. Undan bemalol foydalanish mumkin:

```
import math

x = math.pi
print(x)
```

```
3.141592653589793
```

Bu darsimizda Pythonda matematika bo'limi bilan tanishdik. **math** va **cmath** modulining funksiyalari juda ko'p, ularning bacrhasiga doir misollarni ko'rib chiqa olmaymiz. Yana ham ko'proq funksiyalar haqida bilish uchun qo'shimcha adabiyotlarga murojaat qilishni maslahat beramiz.

PYTHONDA SANOQ SISTEMASINING ISHLATILISHI

Maktab kursidagi informatika faninidan bizga ma'lumki, sonlar nafaqat o`nlik sanoq sistemasida balki boshqa sanoq sistemalarida ham bo`lishi mumkin. Masalan: kompyuter ikkilik sanoq sistemasidan foydalanadi ya'ni 19-soni ikkilik sanoq sistemasida (kompyuterda) 10011 ko`rinishida ifodalanadi. Bundan tashqari sonlarni bir sanoq sistemasidan ikkinchi sanoq sistemasiga o`tkazish kerak. Python bu uchun bir qancha funksiyalarni taqdim etadi:

- **int([object],[sanoq sistemasi asosi])-** butun sonni berilgan sanoq sistemasidan o`nlik sanoq sistemasiga o`tkazadi.
- **bin(x)-** butun sonni ikkilik sanoq sistemasiga o`tkazadi
- **hex(x)-** butun sonni o`n oltilik sanoq sistemasiga o`tkazadi
- **oct(x)-** butun sonni sakkizlik sanoq sistemasiga o`tkazadi.

```
>>> bin(19)
'0b10011'
>>> oct(19)
'0o23'
>>> hex(19)
'0x13'
>>> int('10011',2)
19
>>> int('0b10011',2)
19
>>> a=int('19')# satrni songa o`tkazadi
>>> b=int(19.5)# haqiqiy sonni butun qismini qaytaradi
>>> print(a,b)
19 19
```

V-BOB. PYTHONDA FAYLLAR VA ISTISNOLAR BILAN ISHLASH

PYTHONDA FAYLLAR BILAN ISHLASH

Fayllar bilan ishlash pythonda muhim qismlardan biri. Ayniqsa, web dasturlar bilan ishlashda. Pythonda fayllarni hosil qilish, o'qish, yangilash va o'chirish imkoniyati mavjud.

Fayllarni ochish **open()** funksiyasi bilan amalga oshadi. Bunda ushbu funksiya 2 ta parameter qabul qiladi: **Fayl nomi** va **rejimi**. Rejim deganda faylni qay maqsadda ochish nazarda tutiladi. Bu rejimlar quyidagilar:

- **"r"** – **Read** – faylni o'qish uchun ochish. Agar fayl mavjud bo'lmasa, xatolik yuz beradi.
- **"a"** – **Append** – faylga qo'shimcha qo'shish uchun ochish. Agar fayl mavjud bo'lmasa yangi fayl ochadi.
- **"w"** – **Write** – faylga yozish uchun ochish. Agar fayl mavjud bo'lmasa, yangi fayl ochadi.
- **"x"** – **Create** – yangi fayl hosil qilish. Agar bunday fayl mavjud bo'lsa xatolik yuz beradi.

Bundan tashqari qo'shimcha 2 ta rejim bor. Ular yuqoridagilar bilan qo'llaniladi:

- **"t"** – **Text** – matn turi ya'ni fayl matndan iborat bo'ladi.
- **"b"-- Binary** – binar rejim (ikkilik). Masalan rasmlarni binary rejimda joylashtirish.

Fayl ochish

Fayl ochish quyidagicha bo'ladi. Agar bizda qandaydir **.txt** fayl bo'lsa, uni o'qish rejimida ochish bunday qilamiz:

```
f = open("fayl_nomi.txt")
```

E'tibor berdingizmi, bu yerda hech qanday rejim ko'rsatilmadi. Chunki **"r"** va **"t"** rejimlari avtomatik hisoblanadi . Ya'ni hech qanday rejim ko'rsatilmasa, ular ishga tushadi. Faylimiz o'qilish uchun ochiladi va u matn ko'rinishida bo'ladi. Bularni aniq ko'rsatgan holda ham ochish mumkin. Bu quyidagicha bo'ladi:

```
f = open("fayl_nomi.txt", "rt")
```

Fayllarni o'qish

Avvalgi darsda nima maqsadda ochishimizga qarab, turli rejimlarini borligini ko'rib chiqdik. Hozir biz faylni ochib o'qish uchun ochib ko'ramiz.

Avval **.txt** kengaytmali biror faylga 4 – 5 qatorli matn kiritamiz, uni python faylimiz joylashgan papkaga bitr nom bilan saqlaymiz. Uni **open()** funksiyasi bilan ochamiz va **read()** funksiyasi bilan o'qiymiz:

```
f = open("fayl_nomi.txt", "r")

print(f.read())
```

Agar fayl boshqa bir papkada joylashgan bo'lsa o'sha faylga yo'llanmani ko'rsatish kerak:

```
f = open("D:\fayllarim\fayl_nomi.txt", "r")

print(f.read())
```

Faqat ma'lum qismni o'qish

read() funksiyasi fayldagi butun matnni o'qiydi. Ammo bizga uning faqatgina ma'lum bir qismi kerak bo'lsa, uni belgilab ko'rsatishimiz kerak. Quyidagi misolimizdagi kod matnning dastlabki 10 ta harf yoki belgisini ekranga chiqaradi:

```
f = open("fayl_nomi.txt", "r")

print(f.read(10))
```

Qatorlarni o'qish

Matnni qatorma qator o'qish ham mumkin. **readline()** funksiyasi aynan shuning uchun mo'ljallangan. Uni bir marta ishlatsak birinchi qator o'qiladi. Yana ishlatsak ikkinchisi va hokazo o'qiladi. Quyidagi kodimiz ishga tushsa, birinchi va ikkinchi qatorlarni o'qiydi.

```
f = open("fayl_nomi.txt", "r")

print(f.readline())
print(f.readline())
```

Faylni yopish

Fayl bilan ishlab bo'lgach albatta uni yopish kerak. Buni **close()** funksiyasi bilan amalga oshiramiz. Yuqoridagi kodimizda faylni ochib dastlabki ikkita qatorni o'qigan edik. Endi o'sha faylni yopamiz.

```
f = open("fayl_nomi.txt", "r")

print(f.readline())
print(f.readline())

f.close()
```

Faylga yozish

Avvalgi darslarda fayllarni nima maqsadda ochishimizga qarab turli rejimlar borligini ko'rib chiqdik. Hozir biz faylni ochib o'qish uchun faylni ochib ko'ramiz.

Avval **.txt** kengaytmali biror faylga 4-5 qatorli papka kiritamiz. Uni python faylimiz joylashgan papkaga biror nom bilan saqlaymiz. Uni **open()** funksiyasi bilan ochamiz va **read()** funksiyasi bilan o'qiymiz:

```
f = open("fayl_nomi.txt", "a")
f.write("Matnga qo'shimcha qo'shdik.")
f.close()

# Endi faylni o'qiymiz
f = open("fayl_nomi.txt", "r")
print(f.read())
f.close()
```

Agar faylni **"w"** rejimida ochib unga ma'lumot kiritsak, o'sha fayldagi avvalgi ma'lumotlar o'chib ketadi. Uning o'rniga biz kiritgan ma'lumot qoladi:

```
f = open("fayl_nomi.txt", "w")
f.write("Matnga qo'shimcha qo'shdik.")
f.close()

# Endi faylni o'qiymiz
f = open("fayl_nomi.txt", "r")
print(f.read())
f.close()
```

Yangi fayl ochish

Yangi fayl ochish uchun ham **open()** funksiyasini ishlatamiz. Uni **"x"** rejimida ochish kerak. Agar bunday fayl allaqachon mavjud bo'lsa dasturda xatolik yuz beradi.

“a” va **“w”** rejimlari aslida yozish uchun ishlatilsada biz ochmoqchi bo’lgan fayl mavjud bo’lmasa ular avtomatik tarzda shu nomli yangi fayl ochadi.

my_file nomli yangi fayl hosil qilish quyidagicha bo’ladi:

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

Faylni o’chirish

Faylni o’chirish uchun os moduliga murojaat qilamiz va undagi **os.remove()** funksiyasidan foydalanamiz. Masalan, biror faylimiz bor. Uni nomini bilamiz. Uni o’chirish quyidagicha bo’ladi:

```
import os

os.remove("fayl_nomi.txt")
```

Fayl mavjudligni tekshirish

Fayl mavjudligini tekshirib, agar u mavjud bo’lsa o’chirish quyidagicha bo’ladi:

```
import os

if os.path.exists("fayl_nomi.txt"):
    os.remove("fayl_nomi.txt")
else:
    print("Bunday fayl mavjud emas")
```

Papkani o’chirish

Agar biror bir papkaning o’zini o’chirmoqchi bo’lsak **os.mdir()** funksiyasini ishlatamiz. Ammo biz faqat bo’sh papkalarni o’chirishimiz mumkin. Masalan, bizda **dasturlar** degan papka bor va u bo’m bo’sh. Uni o’chirish uchun mana bunday qilish kerak:

```
import os

os.rmdir("dasturlar")
```

PYTHONDA ISTISNOLAR BILAN ISHLASH

Agar kodimizda xatolik yuz bersa yoki istisno holatlar bo'lib qolsa Python bizga xatolik haqida xabar beradi. Bunday istisno holatlar bilan ishlash uchun **try** va **except** blokidagi amal bajariladi.

Masalan, biz **x** degan o'zgaruvchini ekranga chiqarmoqchi bo'lamiz. Lekin unday o'zgaruvchining o'zi yo'q bo'lsa xatolik yuz beradi va dastur ishlamaydi. Shuning uchun biz shunday qilamizki, agar **x** o'zgaruvchi mavjud bo'lmasa, bu haqida xabar berilsin:

```
try:
    print(x)
except:
    print("x o'zgaruvchi mavjud emas")
```

```
x o'zgaruvchi mavjud emas
```

except kalit so'zini ishlatganimizda Python istalgan turdagi istisno holat uchun amal qiladi. Ammo biz qaysi turdagi istisno holatini tekshirishni o'zimiz belgilashimiz mumkin. Bunday holatlarning turlari ko'p, biz esa asosiylarini sanab o'tamiz:

- **NameError** – murojaat qilinayotgan obyekt topilmasa, ishga tushadi.
- **ValueError** – o'zgaruvchining qiymati unga mos bo'lmagan turda bo'lsa ishga tushadi. Masalan, biror harfli qiymatni son deb qabul qilmoqchi bo'lsak, shunday bo'ladi.
- **TypeError** - o'zaro nomutanosib qiymatlar bilan amallar bajarilsa ishga tushadi. Masalan harfga son qo'shmoqchi bo'lganimizda.
- **ZeroDivisionError** – Istalgan sonni nolga bo'lish holati bo'lganda ishga tushadi.

Hozir biror sonni nolga bo'lishni tekshiramiz.

Buni shunchaki except bilan ham yoki **ZeroDivisionError** bilan ta'kidlab ham tekshirish mumkin. Natija bir xil bo'ladi:

```
try:
    5/0
except:
    print("nolga bo'lish mumkin emas")

try:
    7/0
except ZeroDivisionError:
    print("nolga bo'lish mumkin meas")
```

```
nolga bo'lish mumkin emas
nolga bo'lish mumkin meas
```

else

else kalit so'zi hech qanday xatolik yuz bermaganda bajariladigan amalni ko'rsatish uchun ishlatiladi:

```
try:
    print("Salom")
except:
    print("Dasturda xatolik bor")
else:
    print("Hech qanday xatolik yo'q")
```

```
Salom
Hech qanday xatolik yo'q
```

finally

finally bloki ichida ko'rsatilgan amal xatolik bo'lishi yoki bo'lmasligidan qat'iy nazar bajariladi.

```
try:
    print(x)
except:
    print("x mavjud emas")
else:
    print("Hech qanday xatolik yo'q")
finally:
        print("Tekshiruv tugadi")
```

```
x mavjud emas
Tekshiruv tugadi
```

Istisno holatini hosil qilish

Dasturchi sifatida o'zimiz ham istisno holatini tuzishimiz mumkin. Buning uchun **raise** kalit so'zini ishlatamiz. Masalan, biror son agar noldan kichik bo'lsa dasturimiz xatolik haqida xabar berishi kerak bo'lsa:

```
x = -1

if x < 0:
    raise Exception("Manfiy son aniqlandi")
```

raise kalit so'zi bilan qanday turdagi istisno holati bo'lishini ham o'zimiz belgilashimiz mumkin. Hozir **TypeError** istisnoli holatini tuzamiz. Bunda agar kiritilgan qiymati butun sonli o'zgaruvchi bo'lmasa xatolik haqida xabar berilsin:

```
x = "abc"

if type(x) is not int:
    raise TypeError("Qiymat butun son bo'lishi kerak")
```

USER INPUT

Biz dasturimizda foydalanuvchidan biror ma'lumot kiritishni so'rashimiz mumkin. User input ana shunday xizmat bo'lib, u **input()** funksiyasi yordamida ishlaydi. **Python 2.7** versiyasida bu funksiya **raw_input()** bo'lgan.

```
ism = input("Ismingizni kiriting: ")

print("Sizning ismingiz: "+ism)
```

```
Ismingizni kiriting: Abbosbek
Sizning ismingiz: Abbosbek
```

Foydalanuvchi nafaqat so'z balki son kiritishi ham mumkin va shu son ustida amal bajarishga ega bo'lamiz. Bunda endi kiritilayotgan ma'lumotni son deb qabul qilishimiz buyurishimiz kerak.

Hozir kiritilgan sonning kvadratini chiqaruvchi dastur tuzamiz.

```
x = int(input("Son kiriting: "))

kv_x = x*x

print(kv_x)
```

```
Son kiriting: 7
49
```

VI-BOB. PYTHONDA OBYEKTGA YO'NALTIRILGAN DASTURLASH (OOP)

PYTHONDA OOP TUSHUNCHALARI

Boshqa umumiy maqsadli tillar singari, python ham boshidan beri ob'ektga yo'naltirilgan til hisoblanadi. **Python** - ob'ektga yo'naltirilgan dasturlash tili. Bu bizga ob'ektga yo'naltirilgan yondashuv yordamida dasturlarni ishlab chiqishga imkon beradi. Python-da biz osongina sinflar va obyektlarni yaratishimiz va ulardan foydalanishimiz mumkin.

Ob'ektga yo'naltirilgan dasturlash tizimining asosiy printsiplari quyida keltirilgan:

- **Object** (Ob'ekt)
- **Class** (Sinf)
- **Method** (metod, usul)
- **Inheritance** (Meros olish)
- **Polymorphism** (Polimorfizm)
- **Data Abstraction** (Ma'lumotlarni olish)
- **Encapsulation** (Inkapsulyatsiya)

Object (Ob'ekt)

Ob'ekt - bu holat va xulq-atvor, xususiyatlarga ega bo'lgan shaxs. Bu sichqoncha, klaviatura, stul, stol, ruchka va boshqa turdagi har qanday haqiqiy ob'ekt bo'lishi mumkin.

Python-dagi hamma narsa ob'ekti bo'lib, deyarli hamma narsada atributlar va metodlar mavjud. Barcha funksiyalar funksiya manba kodida belgilangan **__doc__** qatorini qatorini qaytaradigan o'rnatilgan **doc** atributiga ega.

Class (Sinf)

Sinf ob'ektlar to'plami sifatida aniqlanishi mumkin. Bu ba'zi bir o'ziga xos atributlar va usullarga ega bo'lgan mantiqiy shaxs. Masalan: agar sizda ishchilar sinfingiz bo'lsa, unda u atribut va usulni, ya'ni elektron pochta identifikatori, ism, yosh, ish haqi va boshqalarni o'z ichiga olishi kerak.

Sintaksis

```
class ClassName:
    <Bayonot-1>
    .
    .
    <Bayonot-N>
```

Method (metod, usul)

Metod - bu ob'ekt bilan bog'liq bo'lgan funksiya. Python-da metod faqat sinf misollari uchun xos emas. Har qanday ob'ekt turi metodlariga ega bo'lishi mumkin.

Inheritance (Meros olish)

Merosxo'rlik - bu haqiqiy dunyo meros tushunchasini simulyatsiya qiladigan ob'ektga yo'naltirilgan dasturlashning eng muhim jihati. Bola ob'ekti ota-onaning barcha xususiyatlarini va xatti-harakatlarini egallashini belgilaydi.

Merosdan foydalanib, biz boshqa sinfning barcha xususiyatlari va xatti-harakatlaridan foydalanadigan sinfni yaratishimiz mumkin. Yangi sinf hosil bo'lgan sinf yoki bola klassi, xossalari olingan sinf esa asosiy sinf yoki ota-ona sinfi sifatida tanilgan.

Bu kodning qayta ishlatilishini ta'minlaydi.

Polymorphism (Polimorfizm)

Polimorfizm tarkibida ikkita "poli" va "morflar" so'zlari mavjud. Poli ko'p, morflar esa shakllar degan ma'noni anglatadi. Polimorfizm bilan biz bitta vazifani har xil usulda bajarish mumkinligini tushunamiz. Masalan, sizda sinf hayvonlari bor, va barcha hayvonlar gapirishadi. Ammo ular boshqacha gapirishadi. Bu erda "gapirish" harakati ma'noda polimorf va hayvonga bog'liq. Shunday qilib, mavhum "hayvon" tushunchasi aslida "gapirmaydi", lekin aniq hayvonlar (it va mushuklar kabi) "gapirish" harakatini aniq amalga oshiradilar.

Encapsulation (Inkapsulyatsiya)

Inkapsulyatsiya – obyektga yo’naltirilgan dasturlashning muhim jihati hisoblanadi. U metodlar va o’zgaruvchilarga kirishni cheklash uchun ishlatiladi. Inkapsulyatsiya kod va ma’lumotlar tasodifan o’zgartirilishidan bir bir ichida birlashtiriladi.

Data abstraction (Ma'lumotlarni abstraktsiya qilish)

Ma'lumotlarni ajralish va inkapsulyatsiya qilish ikkalasi ham ko'pincha sinonim sifatida ishlatiladi. Ikkalasi ham deyarli sinonimdir, chunki ma'lumotlar abstraktsiyasiga inkapsulyatsiya orqali erishiladi.

Abstraktsiya ichki tafsilotlarni yashirish va faqat funksionallikni ko'rsatish uchun ishlatiladi. Biron bir narsani mavhumlashtirish, bu narsa funktsiyalar yoki butun dastur bajaradigan ishlarning mohiyatini o'z ichiga olishi uchun narsalarga nom berishni anglatadi.

Ob'ektga yo'naltirilgan va protseduraga yo'naltirilgan dasturlash tillari

Index	Object-based Programming	Procedural Programming
1	Ob'ektga yo'naltirilgan dasturlash - bu muammolarni yechishga qaratilgan yondashuv va hisoblash ob'ektlar yordamida amalga oshiriladigan joyda qo'llaniladi.	Protsedurali dasturlash hisoblashlarni bosqichma-bosqich bajarish bo'yicha ko'rsatmalar ro'yxatidan foydalanadi
2	Bu rivojlanish va texnik xizmat ko'rsatishni osonlashtiradi.	Protsessual dasturlashda loyiha uzoq davom etganda kodlarni saqlash oson emas.
3	Shunday qilib, haqiqiy muammolarni osongina hal qilish mumkin.	Bu haqiqiy dunyoni taqlid qilmaydi. U funktsiyalar deb nomlangan kichik qismlarga bo'linib, bosqichma-bosqich ko'rsatmalar bilan ishlaydi
4	Ma'lumotlarni yashirishni ta'minlaydi. Shunday qilib, protsessual tillardan ko'ra xavfsizroq. Siz shaxsiy ma'lumotlarga biron bir joydan kira olmaysiz.	Protsedurali til ma'lumotlarni bog'lashning to'g'ri usulini taqdim etmaydi, shuning uchun u xavfsiz emas.
5	Ob'ektga yo'naltirilgan dasturlash tillarining misoli C ++, Java, .Net, Python, C # va boshqalar.	Protsessual tillarning namunalari: C, Fortran, Paskal, VB va boshqalar.

PYTHONDA SINF VA OBYEKTLAR

Biz allaqachon muhokama qilganimizdek, sinf virtual ob'ekt bo'lib, uni ob'ektning rejasi sifatida ko'rish mumkin. Sinf paydo bo'lganida obyekt paydo bo'ldi. Keling, buni bir misol orqali tushunaylik.

Aytaylik, sinf binoning prototipidir. Bino polga, eshiklarga, derazalarga va hokazolarga oid barcha ma'lumotlarni o'z ichiga oladi, biz ushbu detallarga asoslanib, xohlagancha bino yasay olamiz. Demak, binoni sinf sifatida ko'rish mumkin va biz shu sinfning shuncha ob'ektini yaratishimiz mumkin.

Boshqa tomondan, ob'ekt sinfning misoli. Ob'ektni yaratish jarayonini **instantatsiya** deb atash mumkin.

O'quv qo'llanmasining ushbu qismida biz python-da sinflar va ob'ektlarni yaratishni muhokama qilamiz. Atributga sinf ob'ekti yordamida qanday erishish mumkinligi haqida ham gaplashamiz.

Python – obyektga yo’naltirilgan dasturlash tili. Pythonda deyarli barcha narsa **obyekt** hisoblandi. Ularning o’z xususiyatlari va funksiyalari bor.

Sinflar esa *obyekt konstruktorlari* hisoblanadi. Ular bilan obyektlar tuziladi.

Sinf hosil qilish

Sinf hosil qilish uchun **class** kalit so’zi ishlatiladi. Hozir biz **Son** degan sinf hosil qilamiz. Shu sinf nomini **print** so’zi bilan ekranga chiqarish buyrug’ini bersak, shu sinf mavjudligi haqida ma’lumot chiqadi:

```
class Son:
    x = 5

print(Son)
```

```
<class '__main__.Son'>
```

Obyekt hosil qilish

Sinflar *obyekt konstruktorlari* ekanligini aytgan edik. Hozir yuqorida hosil qilgan sinfimiz orqali yangi **obyekt** hosil qilamiz. Uning nomi **s1** bo’ladi.

```
class Son:
    x = 5

s1 = Son()
print(s1.x)
```

```
5
```

__init__() funksiyasi

Yuqoridagi misollarimizdagi **sinf** va **obyektlar** bilan shunchaki sodda ko'rinishda tanishib chiqdik. Ammo ular haqiqiy dasturlar tuzishga yaroqsiz. Sinflarning mohiyatini tushunish uchun **__init__()** ichki funksiyasini bilishimiz lozim.

Har bir sinf tuzilgan paytda **__init__()** funksiyasi mavjud bo'ladi. **__init__()** funksiyasi obyektlar tuzilayotgan paytda ularning xususiyatlariga qiymatlarni yoki bajarilishi kerak bo'lgan operatsiyalarni biriktiradi.

Hozir **Ishchi** degan sinf hosil qilamiz va unda **ism** va **yosh** ko'rsatkichlariga qiymatlar o'zlashtirish uchun **__init__()** funksiyasidan foydalanamiz.

Keyin **__init__()** funksiyasi har safar yangi obyekt tuzilganda avtomatik tarzda ishlaydi.

Eslatib o'tamiz, **__init__()** funksiyasini yozayotganda har ikkala tarafdan ham ikkitadan (__) tag chiziq yoziladi.

```
class Ishchi:
    def __init__(self, ism, yosh):
        self.ism = ism
        self.yosh = yosh

p1 = Ishchi ("Abbosbek", 20)

print(p1.ism)
print(p1.yosh)
```

```
Abbosbek
20
```

Obyekt funksiyalari

Obyektlar ham funksiyaga ega bo'lishi mumkin. Bu funksiyalar sinf ichida tuziladi va obyektlar tomonida ishlatiladi. Masalan, obyekt o'zini tanishtirish funksiyasini tuzamiz:

```
class Ishchi:
    def __init__(self, ism, yosh):
        self.ism = ism
        self.yosh = yosh

    def tanish(self):
        print("Mening ismim "+ self.ism)

p1 = Ishchi ("Abbosbek", 20)
p1.tanish()
```

```
Mening ismim Abbosbek
```

self parametri

self parametri sinfga tegishli o'zgaruvchilarga murojaat qila olish uchun ishlatiladi. U o'ziga xos yo'llovchi vositadir. U aynan **self** deb nomlanishi shart emas, boshqa nomlarni ishlatish ham mumkin. Faqat u sinfdagi istalgan funksiyaning ilk parametri sifatida yozilishi shart.

Hozir yuqoridagi misolimizdagi **self** parametrlarini **abc** deb o'zgartiramiz va natija o'zgarmaydi.

```
class Ishchi:
    def __init__(abc, ism, yosh):
        abc.ism = ism
        abc.yosh = yosh

    def tanish(abc):
        print("Mening ismim "+ abc.ism)

p1 = Ishchi ("Abbosbek", 20)
p1.tanish()
```

```
Mening ismim Abbosbek
```

Obyekt xususiyatini o'zgartirish

Biror obyektning xususiyatlarini osongina o'zgartirishimiz mumkin. Masalan, dastlab tuzgan obyektimiz 22 yosh bo'lsa, so'ng uni 25 yoshga o'zgartiramiz:

```
class Ishchi:
    def __init__(abc, ism, yosh):
        abc.ism = ism
        abc.yosh = yosh

p1 = Ishchi ("Abbosbek", 20)

p1.yosh = 25

print(p1.yosh)
```

```
25
```

Obyekt xususiyatini o'chirish

Obyekt xususiyatlarini o'chirish ham mumkin. Hozir obyektimizdagi **yosh** xususiyatini o'chiramiz. So'ng uni ekranga chiqarish buyrug'ini beramiz. Dastur ishga tushgach xatolik haqida xabar beriladi.

```
class Ishchi:
    def __init__(abc, ism, yosh):
        abc.ism = ism
        abc.yosh = yosh

p1 = Ishchi ("Abbosbek", 20)

del p1.yosh

print(p1.yosh)
```

```
AttributeError: 'Ishchi' object has no attribute 'yosh'
```

Obyektni o'chirish

Obyektni o'chirish uchun **del** kalit so'zini obyekt nomi bilan qo'llaymiz. Natijada obyekt butkul o'chib ketadi.

Quyidagi kodimizda ham xatolik haqida xabar beriladi. Sababi, biz o'chib ketgan obyektni ekranga chiqarmoqchi bo'lyabmiz:

```
class Ishchi:
    def __init__(abc, ism, yosh):
        abc.ism = ism
        abc.yosh = yosh

p1 = Ishchi ("Abbosbek", 20)

del p1

print(p1)
```

```
NameError: name 'p1' is not defined
```

SINFLARDA KONSTRUKTOR TUSHUNCHASI

Konstruktor - bu sinfning instansiya a'zolarini initsializatsiya qilish uchun ishlatiladigan maxsus metod (funktsiya) turi.

Konstruktorlar ikki xil bo'lishi mumkin:

- Parametrlangan konstruktor
- Parametrlanmagan konstruktor

Ushbu sinf ob'ektini yaratganimizda konstruktor ta'rifi bajariladi. Shuningdek, konstruktorlar ob'ekt uchun biron bir ishga tushirish vazifasini bajarish uchun yetarli resurslar mavjudligini tasdiqlaydilar.

Python-da konstruktor yaratish

Pythonda __init__ metodi sinf konstruktorini simulyatsiya qiladi. Ushbu usul sinfni qo'zg'atganda chaqiriladi. Biz __init__ ta'rifiga qarab, sinf ob'ektini yaratishda istalgan sonli argumentlarni berishimiz mumkin. Bu asosan sinf atributlarini ishga tushirish uchun ishlatiladi. Har bir sinf konstruktorga ega bo'lishi kerak, hatto u oddiygina konstruktorga tayansa ham.

Employee sinfining atributlarini ishga tushirish uchun quyidagi misolni ko'rib chiqing.

Example:

```
class Employee:
    def __init__(self,name,id):
        self.id = id; self.name = name;

    def display (self):
        print("ID: %d \nName: %s"%(self.id,self.name))

emp1 = Employee("John",101)
emp2 = Employee("David",102)

#accessing display() method to print employee 1 information
emp1.display();
#accessing display() method to print employee 2 information
emp2.display();
```

```
ID: 101
Name: John
ID: 102
Name: David
```

Misol: Sinf ob'ektlari sonini hisoblash

```
class Student:
    count = 0
    def __init__(self):
        Student.count = Student.count + 1

s1=Student()
s2=Student()
s3=Student()
print("The number of students:",Student.count)
```

```
The number of students: 3
```

Pythonning parametrlanmagan konstruktorga misoli

```
class Student:
    # Constructor - parametrlanmagan
    def __init__(self):
        print("This is non parametrized constructor")

    def show(self,name):
        print("Salom",name)

student = Student()
student.show("Abbosbek")
```

```
Salom Abbosbek
```

Pythonning parametrlangan konstruktorga misol

```
class Student:
    # Constructor - parameterized
    def __init__(self, name):
        print("This is parametrized constructor")
        self.name = name

    def show(self):
        print("Hello",self.name)

student = Student("John")
student.show()
```

```
Hello John
```

Python ichki sinf vazifalari

Sinfda aniqlangan ichki funktsiyalar quyidagi jadvalda tavsiflangan.

SN	Funksiya	Vazifasi
1	**getattr** (obj, name, default)	Ob'ektning atributiga kirish uchun ishlatiladi.
2	**setattr** (obj, name, value)	U ob'ektning o'ziga xos atributiga ma'lum bir qiymatni belgilash uchun ishlatiladi.
3	**delattr** (obj, name)	U ma'lum bir atributni o'chirish uchun ishlatiladi.
4	**hasattr** (obj, name)	Ob'ektda o'ziga xos atribut bo'lsa, u haqiqiy qiymatni qaytaradi.

Misol:

```
class Student:
    def __init__(self,name,id,age):
        self.name = name;
        self.id = id;
        self.age = age

#creates the object of the class Student
s = Student("John",101,22)

#prints the attribute name of the object s
print(getattr(s,'name'))

# reset the value of attribute age to 23
setattr(s,"age",23)

# prints the modified value of age
print(getattr(s,'age'))

# prints true if the student contains the attribute with name id

print(hasattr(s,'id'))

# deletes the attribute age
delattr(s,'age')

# this will give an error since the attribute age has been deleted
print(s.age)
```

```
John
23
True
AttributeError: 'Student' object has no attribute 'age'
```

O'rnatilgan sinf atributlari

Boshqa atributlar bilan bir qatorda, python klassida sinf haqida ma'lumot beradigan ba'zi bir o'rnatilgan sinf atributlari mavjud.

O'rnatilgan sinf atributlari quyidagi jadvalda keltirilgan:

- **__dict__** - Bu sinf nomlari maydoni haqidagi ma'lumotlarni o'z ichiga olgan lug'atni taqdim etadi.
- **__doc__** - U sinf hujjatiga ega bo'lgan qatorni o'z ichiga oladi
- **__name__** - U sinf nomiga kirish uchun ishlatiladi.
- **__module__** - Ushbu sinf aniqlangan modulga kirish uchun foydalaniladi.
- **__bases__** - Unda barcha asosiy sinflarni o'z ichiga olgan korniş mavjud.

Misol:

```
class Student:
    def __init__(self,name,id,age):
        self.name = name;
        self.id = id;
        self.age = age

    def display_details(self):
        print("Name:%s, ID:%d, age:%d"%(self.name,self.id))

s = Student("John",101,22)
print(s.__doc__)
print(s.__dict__)
print(s.__module__)
```

```
{'name': 'John', 'id': 101, 'age': 22}
__main__
```

SINFLARDA VORISLIK TUSHUNCHASI

Vorislik - bu atama sinflarga xosdir. **Vorislik** deb bir sinfdagi barcha funksiya va xususiyatlarni boshqa bir sinf o'ziga o'zlashtirishiga aytiladi.

Funksiyalari meros qilib olinadigan sinf **ona sinf** deyiladi.

Meros qilib olingan funksiyalarni o'ziga o'zlashtiradigan sinf **voris sinf** deyiladi.

Ona sinf hosil qilish

Istalgan sinf **ona sinf** bo'lishi mumkin. Shu sababli ona sinfni hosil qilish xuddi oddiy sinfni hosil qilish kabidir.

Hozir **Odam** degan sinf hosil qilamiz. Unda **ism** va **familiya** parametrlari va tanish degan funksiyasi bo'ladi. So'ngra shu sinf orqali **x** obyekt hosil qilamiz:

```
class Odam:
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

    def tanish(self):
        print(self.ism, self.familiya)

x = Odam ("Abbosbek", "Ibragimov")
x.tanish()
```

```
Abbosbek Ibragimov
```

Voris sinf hosil qilish

Voris sinf hosil qilish uchun yangi sinf tuzilayotganda ona sinfni paramet sifatida kiritamiz. Shunda voris sinf ona sinfdan barcha xususiyatlarni o'zlashtiradi.

Hozir **Talaba** degan sinf hosil qilamiz. **Odam** sinfi uning onam sinfi bo'ladi. Qavslar ichida ona sinfni kirittamiz va uning barcha xususiyatlarini voris sinf o'zlashtiradi. Qo'shimcha parametr qo'shish shart emas, ammo sinf hosil qilayotganda ichi bo'sh bo'lishi ham mumkin emas. Agar hechnarsa yozishni istamasak xatolik yuz bermasligi uchun **pass** kalit so'zini qo'shib qo'yamiz:

```
class Odam:
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

    def tanish(self):
        print(self.ism, self.familiya)
```

```
# Endi voris sinf ya'ni bola sinfni hosil qilamiz

class  Talaba (Odam):
    pass

x = Talaba ("Asadbek", "Suvonov")
x.tanish()
```

```
Asadbek Suvonov
```

__init__() funksiyasini qo'shish

Avvalgi misolimizda **voris sinf** hosil qilganimizda **pass** kalit so'zi bilan cheklanib qo'ya qoldik. Shu sababli voris sinf barcha funksiyalarni avtomatik tarzda o'zlashtirgan edi. Endi voris sinfga **__init__()** funksiyasi bilan parametrlarini joylashtiramiz. Bunda voris sinf ona sinfdagi **__init__()** funksiyasidan emas o'zidagidan foydalanadi.

```
class Odam:
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

    def tanish(self):
        print(self.ism, self.familiya)

# Endi voris sinf ya'ni bola sinfni hosil qilamiz

class  Talaba (Odam):
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

x = Talaba ("Asadbek", "Suvonov")
x.tanish()
```

```
Asadbek Suvonov
```

Ammo ona sinfdagi **__init__()** funksiyasidan foydalanmoqchi bo'lsak, voris sinfdagi **__init__()** funksiyasi ichiga ona sinfning shu funksiyasini yozamiz:

```
class Odam:
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

    def tanish(self):
        print(self.ism, self.familiya)
```

```
# Endi voris sinf ya'ni bola sinfni hosil qilamiz

class  Talaba (Odam):
    def __init__(self, ism, familiya):
        Odam.__init__(self, ism, familiya)

x = Talaba ("Asadbek", "Suvonov")
x.tanish()
```

```
Asadbek Suvonov
```

super() funksiyasi

Sinflar bilan ishlash uchun maxsus **super()** funksiyasi ham mumkin. Bu funksiya ona sinfdagi barcha funksiya va parametrlarni voris sinfga o'zlashtiradi:

```
class Odam:
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

    def tanish(self):
        print(self.ism, self.familiya)

# Endi voris sinf ya'ni bola sinfni hosil qilamiz

class  Talaba (Odam):
    def __init__(self, ism, familiya):
        super().__init__(ism, familiya)

x = Talaba ("Asadbek", "Suvonov")
x.tanish()
```

```
Asadbek Suvonov
```

Parametr qo'shish

Voris sinf hosil qilingach unga yana qo'shimcha parameter qo'shmoqchi bo'lsak quyidagicha amalga oshirish mumkin. Hozir **yil** parametrini qo'shamiz:

```
class Odam:
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

    def tanish(self):
        print(self.ism, self.familiya)
```

```
# Endi voris sinf ya'ni bola sinfni hosil qilamiz

class  Talaba (Odam):
    def __init__(self, ism, familiya):
        super().__init__(ism, familiya)
        self.yil = 2002

x = Talaba ("Asadbek", "Suvonov")
print(x.yil)
```

```
2002
```

Yuqoridagi misolimizda yangi parametrni qo'shgan zahotimiz unga qiymat berdik. Endi **__init__()** funksiyasining o'ziga yil parametrini qo'shib unga o'zlashtiramiz. Shundan so'ng uning qiymatini yangi obyekt hosil qilayotganda o'zimiz kirtishimiz kerak bo'ladi.

```
class Odam:
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

    def tanish(self):
        print(self.ism, self.familiya)

# Endi voris sinf ya'ni bola sinfni hosil qilamiz

class  Talaba (Odam):
    def __init__(self, ism, familiya, yil):
        super().__init__(ism, familiya)
        self.yil = 2002

x = Talaba ("Asadbek", "Suvonov", 2002)
print(x.yil)
```

```
2002
```

Funksiya qo'shish

Voris sinfga qo'shimcha funksiyalar ham qo'shish mumkin. Natijada u ona sinfdan o'zlashtirgan funksiyalari va biz qo'shgan qo'shimcha funksiyalarga ega bo'ladi.

Hozir voris sinfga **tugilgan()** funksiyasini qo'shamiz. Bu funksiya talabaning tug'ilgan yili haqida ma'lumot beradi:

```
class Odam:
    def __init__(self, ism, familiya):
        self.ism = ism
        self.familiya = familiya

    def tanish(self):
        print(self.ism, self.familiya)

# Endi voris sinf ya'ni bola sinfni hosil qilamiz

class  Talaba (Odam):
    def __init__(self, ism, familiya, yil):
        super().__init__(ism, familiya)
        self.yil = 2002

    def tugilgan(self):
        print("Men" , self.yil , " - yilda tug'ilganman")

x = Talaba ("Asadbek", "Suvonov", 2002)
x.tugilgan()
```

```
Men 2002  - yilda tug'ilganman
```

XOTIMA

Mazkur qo'llanmada Python dasturlash tilining yaratilish tarixi va imkoniyatlari haqida yozilgan. Python dasturida ishlaydigan foydalanuvchilar uchun uning sintaksisi, asosiy operatorlari, fayllar, funksiyalar bilan ishlash sanoq sistemalari va satrlar bilan ishlash haqida muhim zaruriy ma'lumotlar keltirilgan. Qo`llanmani o`qigan har bir qiziquvchi Python dasturlash tilining sintaksisi o`zi kabi sodda va oson ekanligini, o`zgaruvchilarning tipini e'lon qilinmasligini, shuningdek sonlar bilan ishlaganda nafaqat butun va haqiqiy sonlar ustida balki kompleks sonlar ustida ham amallar bajarishni ko`rsatilgan misollar yordamida o`rgana oladi va uni amaliyotda bajara oladi. Satrlar bilan ishlash va ular ustida amallar bajarish haqida ham yetarlicha ma'lumotlar keltirilgan.

Ushbu qo'llanmada yana ro`yxat, kortej, lug`at va to`plam tushunchalari va ularni qanday yaratish mumkinligi ular ustida amallar bajarish haqida ma'lumotlar keltirilib misollar yordamida tushuntirilgan. Shuningdek, modul tushunchasi, Python dasturining juda boy kutubxonaga ega ekanligi haqidagi ma'lumotlar berilgan bo`lib, ko`plab modullar shu jumladan platform, pickle, sys, copy, datetime, math, cmath, random, os modullari, ularning funksiyalari va qo`llanilishi haqida yozilib, misollar keltirish yordamida amaliyotda qo`llab tushuntirilgan. Bo'lajak dasturchilar o'z ustilarida mustaqil ishlashlari va Python dasturida ishlash bo`yicha bilim saviyalarini oshirish uchun standart modullardan foydalanishlari taklif etilgan. Bundan tashqari shuningdek, Python dasturlash tilining obyektga yo'naltirilgan dasturlash (OOP) bo'limi haqida batafsil misollar orqali keng tushunchalar berilgan. Qo'llanmadan foydalangan bo'lajak dasturchilar OOP ning class, obyekt, vorislik, polimorfizm kabi asosiy tushunchalar haqida batafsil ma'lumotga ega bo'lishadilar. Bu qo'llanmada qisqa qilib aytganda Python dasturlash tilining afzallik tomonlari tushuntirib berildi va shular asosida o'zbek tilida qo'llanma yaratildi.

Ushbu qo'llanma internet tarmog'idagi ma'lumotlar asosida yaratildi.

Qo'llanmada kamchiliklar va xatoliklar bo'lgan bo'lsa uzur so'raymiz.

Bizni to'g'ri tushunasiz degan umiddamiz.

Biz ham xuddi sizdek oddiy insonmiz !!!

Bir ishni bajarishni mo'ljalladingizmi, unga bugunoq kirishing.

Vaqtni boy bermang !!!

(BILL GEYTS)