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 unio'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+5jy
= 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 haqidagapiramiz. 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

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

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

Satrlar formati

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

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
```

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

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

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

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

Nomli indekslar

Indekslarni raqamlab ishlatishdan tashqari, ularni nomlab ishlatsak ham bo’ladi va buquyidagicha 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

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

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

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 oxirginuqta 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(mashina)

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)

print("--------\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

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

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

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:

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 egabo`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
}
```

Element qo’shish

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

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

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

Elementlarni o’chirish

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

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

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

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

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:

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 elementlarinio’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")
```

update()

update() funksiyasi to’plamga yangi element (*kalit-qiymat juftligi*) qo’shadi. Bunda har birvaqtning 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)
```