

**O'ZBEKISTON RESPUBLIKASI OLIY VA O'RTA  
MAXSUS TA'LIM VAZIRLIGI**

**MIRZO ULUG'BEK NOMIDAGI O'ZBEKISTON MILLIY  
UNIVERSITETI**

**SAIDOV DONIYOR YUSUPOVICH**

**PYTHON DASTURLASH TILI**

**O'quv – uslubiy qo'llanma**



[https://t.me/N1\\_Kitoblar\\_uz](https://t.me/N1_Kitoblar_uz)

## MUNDARIJA

<b>I. Pythonga kirish .....</b>	<b>4</b>
1.1. Python dasturlash tili.....	4
1.2. Pythonda dastur kodini yozish.....	6
1.3. O'zgaruvchilar va berilganlar turlari .....	8
1.4. Sonlar ustuda amallar.....	11
1.5. Turga keltirish (turga o'girish) funksiyasi .....	13
1.6. Sonlarning turli sanoq sistemalarda tasvirlanishi .....	15
1.7. Shatr ifodalari.....	16
1.8. Mantiqiy amallar .....	17
1.9. Satrlar ustida amallar .....	18
1.10. if - shart amali (operatori).....	21
1.11. Sikl operatorlari.....	23
1.12. Funksiyalar .....	27
1.13. O'zgaruvchilarning ko'rinish sohasi.....	29
1.14. Modullar.....	30
1.15. Istisno holatlar bilan ishlash.....	35
<b>II. Pythonda ro'yxatlar, lug'atlar, kortejlar va to'plamlar .....</b>	<b>41</b>
2.1. Ro'yxatlar.....	41
2.2. Kortejlar .....	50
2.3. Lug'atlar.....	54
2.4. To'plamlar.....	61
<b>III. Fayllar bilan ishlash .....</b>	<b>66</b>
3.1. Fayllarni ochish va yopish .....	66
3.2. Matn fayllari. Matn faylga yozish.....	68
3.3. CSV fayllari bilan ishlash .....	72
3.4. Binar fayllar .....	75
3.5. shelve moduli .....	76
3.6. OS moduli va fayl tizimi ishlashi.....	80
<b>IV. Satrlar .....</b>	<b>83</b>
4.1. Satrlari bilan ishlash.....	83

4.2. Satrlar bilash ishlashning asosiy metodlari.....	85
4.3. Formatlash.....	91
4.4. So'zlarni sanash dasturi .....	94
<b>V. Asosiy ichki modullar.....</b>	<b>97</b>
5.1. <i>random</i> moduli .....	97
5.2. <i>math</i> moduli .....	98
5.3. <i>locale</i> moduli.....	100
5.4. <i>decimal</i> moduli.....	103
<b>VI. Obyektga yo'naltirilgan dasturlash .....</b>	<b>107</b>
6.1. Sinf va obyekt .....	107
6.2. Kostruktorlar .....	108
6.3. Destruktor.....	109
6.4. Sinflarni modullarda aniqlash va ularni bog'lash.....	110
6.5. Inkapsulyatsiya.....	112
6.6. Vorislik.....	116
6.7. Polimorfizm.....	117
6.8. Obyektlarni turlarga tekshirish .....	120
6.9. object sinfi. Obyektni satr ko'rinishida tasvirlanishi .....	120
<b>VII. Sana va vaqt bilan ishlash .....</b>	<b>123</b>
7.1. <i>datetime</i> moduli.....	123
7.2. Sana ustuda bajariladigan asosiy amallar .....	126

# **I.Pythonga kirish**

## **1.1. Python dasturlash tili**

Python - yuqori bosqichli dasturlash tili hisoblanib, tirli xil ilovalarni yaratish uchun mo'ljallangan. Ya'ni Python dasturlash tili yordamida veb-ilovalar, o'yin ilovalari, oddiy (nastol'niy) dasturlar yaratish hamda berilganlar bazasi bilan ishlash mumkin. Ayniqsa Python dasturlash tilining tezlik bilan tarqalishiga uning mashinali o'rgatish va sun'iy intellekt sohalaridagi tadqiqot ishlarida keng qo'llanilishi sabab bo'lgan.

Python dasturlash tiliga 1991 yil Golland dasturchisi Grido Van Rossu asos solgan. Shundan beri ushbu til rivojlanishning ulkan yo'lini bosib o'tdi va 2000 yilda 2.0 versiyasi, 2008 yil esa 3.0 versiyalari chiqarildi. Versiyalar orasidagi muddatning uzoqligiga qaramasdan doima versiya ostilari chiqariladi. Shunday qilib, ushbu material eng oxirgi 3.7 versiyasi asosida tuzilgan.

Python dasturlash tilining asosiy xususiyatlari quyidagilardan iborat:

Skriptli til. Dastur kodi skriptlar ko'rinishida bo'ladi;

Turli dasturlash paradigmlarni, xususan ob'ektga yo'naltirilgan va funksional paradigmlarni o'zida mujassamlagan;

Skriptlar bilan ishlash uchun interpretator kerak bo'lib, u skriptni ishga tushiradi va bajaradi.

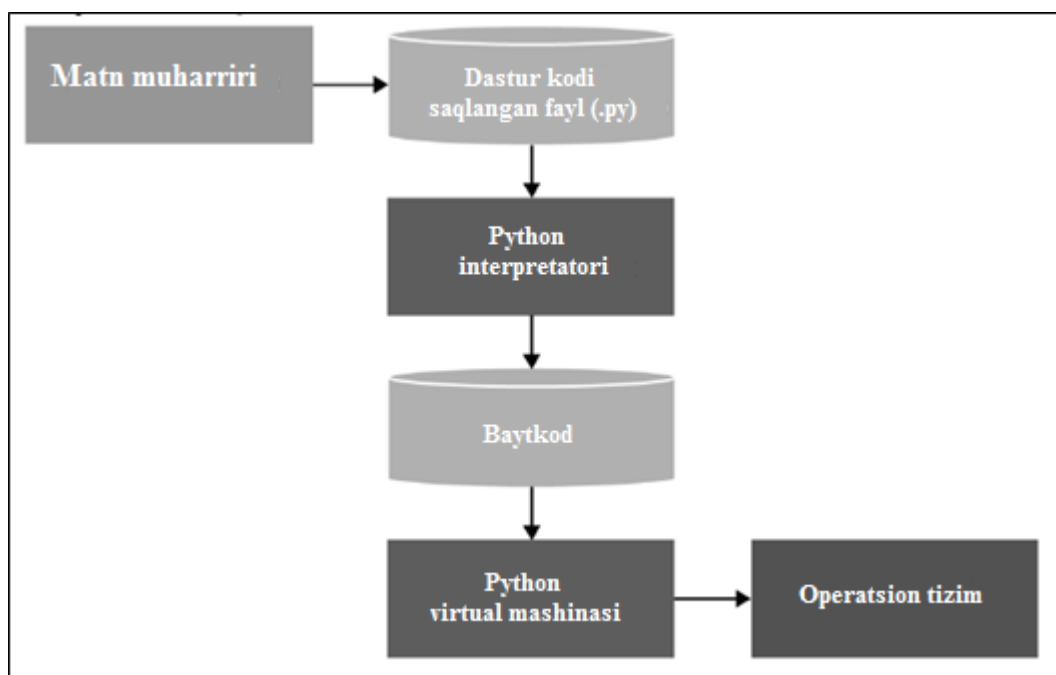
Portativlik va platformaga bog'liqmaslik. Kop'yuterda qanday operasion tizim - Windows, Mac OS, Linux bo'lishidan qat'iy nazar, ushbu operasion tizimda interpretator mavjud bo'lsa, foydalanuvchi tomonidan yozilgan skript kod bajariladi.

Xotiraning avtomatik boshqarilishi;

Turlarga dinamik ajratilishi;

Pythonda dasturning bajarilishi quyidagicha bo'ladi: Dastlab mant muharririda ushbu dasturlash tili asosida ifodalar ketma-ketligidan iborat skript kod yoziladi. Ushbu yozilgan skript kod bajarilish uchun interpretatorga uzatiladi. Interpretator skript kodni oraliq baytkodga tarjima qiladi. Keyin virtual mashina

baytkodni operatsion tizimda bajariladigan instruksiyalar (mashina buyruqlari) to'plamiga o'tkazadi.



**Rasm №1.** Pythonda dasturning bajarilishi jarayoni.

Shu ta'kidlash lozimki, rasman interpretator tomonidan dastlabki kodning baytkodga tarjima qilinishi va virtual mashinaning ushbu baytkodni mashina buyruqlari to'plamiga o'tkazilishi ikkita turli jarayon bo'lsada, ammo amalda ular bitta interpretatorning o'zida birlashtirilgan.

Python juda oddiy dasturlash tili bo'lib, u ixcham shu bilan bir vaqtda sodda va tushinarli sintaksisga ega. Shu sababli Python o'rganish uchun juda oson til sifatida butun dunyoda eng tez tarqalayotgan tillardan biri sifatida e'tirof etiladi. Bundan tashqari ushbu tilda hozirgi kunga kelib, turli sohalarga (veb, o'yin, mul'timediya, ilmiy tadqiqot) mo'ljallangan katta hajmdagi kutbxonalar majmui yaratilgan bo'lib, uning tobora mashhurlashib borishiga sabab bo'lmoqda.

**Pythonni o'rnatish.** Pythonda dastur tuzish uchun interpretator kerak bo'ladi. Uni kompyuteringizdagi o'rnatilgan operatsion tizim turiga mos ravishda <https://www.python.org> rasmiy saytidan kerakli versiyasini tushirib olishingiz mumkin.

## 1.2. Pythonda dastur kodini yozish

Python tilida dastur instruktsiyalar to'plamidan tashkil topgan bo'lib, har bir instruktsiya alohida qatorda joylashgan bo'lishi kerak bo'ladi. Masalan:

```
1 print(3 + 5)
2 print("Python - dasturlash tili!")
```

Python da xat boshi (otstup) juda muhim ahamiyatga ega hisoblanadi. xat boshining noto'g'ri joylashtirilishi dasturda xatolikka olib keladi. Masalan yuqoridagi dastur kodini quyidagicha yozamiz:

```
1 print(3 + 5)
2     print("Python - dasturlash tili!")
```

Ushbu dastur kodi yuqoridagisi bilan bir xil bo'lishiga qaramasdan interpretator xatolik haqida xabar chiqaradi va dastur bajarilmaydi. Shuning uchun ham Pythonda har bir instruktsiya alohida qatorda yozilishi shart. Ushbu hususiyat Pythonning boshqa tillardan, masalan, Java, C# tillaridan farqli jihatlaridan biri hisoblanadi.

Shunga qaramasdan Python tilining ba'zi konstruktsiyalari bir necha qatorlarda yoziladi. Masalan *if* shart konstruktsiyasi shular jumlasidan:

```
1 if 10 < 20:
2     print("Shart bajarildi")
```

Bu holatda 10 soni 20 sonidan kichik va "Shart bajarildi" so'zi chiqariladi. `print("Shart bajarildi")` instruktsiyasi oldida albatta xat boshi bo'lishi shart, chunki u alohida o'zi ishlatilmagan balki *if* shart konstruktsiyasining qismi sifatida qo'llanilgan. Odatda xat boshi 4 ga karrali probellar soni (4, 8,12) bilan yozish kelishilgan, lekin probellar soni 5 va undan ortiq bo'lsa ham dastur ishlaydi.

**Registrga sezuvchanlik.** Python – registrga sezuvchan til hisoblanadi. Shuning uchun *print*, *Print* yoki *PRINT* ifodalari turli ifodalarni anglatadi. Agarda *print* berilganlarni chiqarish ifodasi o'rniga *Print* ishlatilsa xatolik yuz berganligini ifodalovchi “*name 'Print' is not defined*” shaklidagi xabar chiqadi.

**Kommentariyalar (Izohlar).** Pythonda u yoki bu dastur kodlari qismlari nima ish qilishini qayd qilib ketish uchun izohlardan foydalaniladi. Interpretator

dasturni baytkodga tarjima qilayotganda yoki bajarayotganda izohlarni e`tiborsiz qoldiradi. Shuning uchun izohga olingan berilganlar dastur ishlashiga hech qanday ta`sir ko`rsatmaydi.

Python dasturlash tilida izoh qo`yish uchun “#” belgisidan foydalaniladi. Odatda izohlar blokli va satrli izohlarga ajratiladi. Lekin har ikkalasi ham “#” belgisi orqali hosil qilinadi. Farqi satr izohlar dastur kodi yozilgan qatorda koddan keyin yoziladi va u shu satr nima ish bajarishi to`g`risidagi ma`lumotlardan tashkil topadi, ya`ni:

```
1 print("Shart bajarildi") # xabarni konsolga chiqarish
```

Blokli izohlar esa dasturning biror qismi nima ish bajarishi yoki shu qism mazmunini foydalanuvchiga qisqacha ochib berish uchun ishlatilib, dasturni shu qismi kodlaridan oldin alohida satr yoki satrlarda “#” va bitta probel bilan yoziladi, masalan:

```
1 # ushbu funksiya 1 dan n gacha bo`lgan butun sonlarning
2 # yigindisini hisoblaydi
3
4 def Summa (n) :
5     s=0
6     for a in range (1,n+1) :
7         s = s + a
8         print(a, " ",s)
9     return s
```

**Asosiy funksiyalar.** Python o`z ichiga bir necha ichki funksiyalarni qamrab olgan. Ularni ba`zilari dasturlash jarayonida, ayniqsa dasturlash sistaksisini o`rganish paytida juda ko`p qo`llanilganligi sababli ularni alohida qarab chiqamiz.

Ma`lumotni konsol ekraniga chiqarish – *print()* funksiyasi hisoblanadi. Funksiyaga argument sifatida konsolga chiqariluvchi qiymatlar (satr, son, ifoda va x.k.) berilishi mumkin:

```
1 print("Hello python!")
```

Agarda birdaniga bir nechta qiymatlarni chop etish talab qilinsa, u holda ularni *print()* funksiyasiga “,” bilan ajratib kiritiladi:

```
1 print("F.I.SH.:", "Eshmatov", "Toshmat")
```

Natijada ular ekranga probel bilan ajratilgan holatda chop etiladi.

*F.I.O: Eshmatov Toshmat*

Agarda *print()* funksiyasi ma'lumotlarni chop qilish uchun mo'ljallangan bo'lsa, *input()* ekrandan berilganlarni kiritish uchun qo'llaniladi. *input()* funksiyasiga argument sifatida biror bir satr berilishi mumkin. Ushbu satr konsol ekranida aks ettirilib, kiritilishi kerak bo'lgan berilganlar uchun yordamchi taklif vazifasini bajaradi. Masalan:

```
1 name = input("F.I.O.: ")
2 print("Salom", name)
```

Natijaning konsol ekranidagi ko'rinishi quyidagicha bo'ladi:

*F.I.O.: Eshmatov Toshmat*

*Salom Eshmatov Toshmat*

### 1.3. O'zgaruvchilar va berilganlar turlari

Pythonda, boshqa dasturlash tillaridagi kabi o'zgaruvchilar aniq bir turdagi berilganlarni saqlash uchun xizmat qiladi. Pythonda o'zgaruvchilar alfavit belgilari yoki tag chizig'i belgisi bilan boshlanishi va tarkibi son, alfavit belgilari, tag chizig'i belgilaridan iborat bo'lishi, ya'ni bir so'z bilan aytganda identifikator bo'lishi kerak. Bundan tashqari o'zgaruvchi nomi Pythonda ishlatiladigan kalit so'zlar nomi bilan mos tushmasligi shart. Masalan, o'zgaruvchi nomi *and*, *as*, *assert*, *break*, *class*, *continue*, *def*, *del*, *elif*, *else*, *except*, *False*, *finally*, *for*, *from*, *global*, *if*, *import*, *in*, *is*, *lambda*, *None*, *nonlocal*, *not*, *or*, *pass*, *raise*, *return*, *True*, *try*, *while*, *with*, *yield* kabi kalit so'zlar nomi bilan mos tushishi mumkin emas.

Masalan, o'zgaruvchini aniqlash (hosil qilish) quyidagicha amalga oshiriladi:

```
1 a = 14
2 name = "SDY"
```

Yuqorida *a* va *name* o'zgaruvchilari yaratildi va ularga qiymat berildi. Shuni alohida ta'kidlash kerakki, Pythonda o'zgaruvchini dastlab e'lon qilish degan tushuncha mavjud emas (masalan: c++ tilida *int a* kabi o'zgaruvchi e'lon qilinadi), balki o'zgaruvchi kiritiladi va unga qiymat beriladi (masalan: *a=14*). Berilgan



qiymatga ko'ra interpretator o'zgaruvchining turini aniqlaydi. Pythonda o'zgaruvchilarni nomlashning ikki turi: "camel case" va "underscore notation" turlaridan foydalanish tavsiya qilingan.

"camel case" turida o'zgaruvchiga nom berilganda, agar o'zgaruvchi nomi alohida so'zlar birikmasidan tashkil topgan bo'lsa, ikkinchi so'zdan boshlab har bir so'zning birinchi harfi katta harfda (katta registr) bo'lishi talab qilinadi. Masalan:

```
1 firstName = "Saidov"
```

"underscore notation" turida esa so'zlar orasiga tag chizig'i "\_" belgisi qo'yiladi. Masalan:

```
1 first_name = "Saidov"
```

O'zgaruvchilar biror bir turdagi berilganlarni saqlaydi. Pythonda bir necha xildagi berilganlar turlari mavjud bo'lib, ular odatda to'rtta guruhga ajratiladi: sonlar, ketma-ketliklar, lug'atlar va to'plamlar:

*bool (boolean)* – True va False mantiqiy qiymatlar uchun;

*int* – butun sonlar uchun, butun turdagi songa kompyuter xotirasida 4 bayt joy ajratiladi;

*float* – suzuvchan nuqtali sonlar (haqiqiy sonlar) uchun, haqiqiy sonlarni saqlash uchun kompyuter xotirasidan 8 bayt joy ajratiladi;

*complex* – kompleks sonlar uchun;

*str* – satrlar uchun, Python 3.x versiyasidan boshlab satrlar bu- Unicode kodirovkasidagi belgilar ketma-ketligini ifodalaydi;

*bytes* – 0-255 diapazondagi sonlar ketma ketligi uchun

*byte array* – baytlar massivi uchun;

*list* – ro'yxatlar uchun;

*tuple* – kortejlar uchun;

*set* – tartiblanmagan unikal ob`ektlar kolleksiyasi uchun;

*frozen set* – set singari, faqat u o'zgartirilishi mumkin emas (immutable);

*dict* – lug'atlar uchun. Har bir element kalit so'z va qiymat juftligi ko'rinishida ifodalaniladi.

Python –dinamik turlarga ajratuvchi dasturlash tili hisoblanadi. Yuqorida aytib oʻtilganidek, Pythonda oʻzgaruvchi turi unga yuklangan qiymat orqali aniqlanadi. Agarda oʻzgaruvchiga bittalik (‘,’) yoki ikkitalik (“,”) qoʻshtirnoq yordamida satr yuklansa, oʻzgaruvchi *str* turiga ega boʻladi, agarda oʻzgaruvchiga butun son yuklansa – *int*, haqiqiy son yuklansa (masalan: 3.14) yoki eksponentsial koʻrinishdagi qiymat yuklansa (masalan: 11e-1) u *float* turiga ega boʻladi.

Masalan:

```
1 user_id = 234 # int
2 x = 1.2e2 # = 1200.0 float
3 y = 6.7e-3 # = 0.0067 float
4 z = 1.223 # float
5 user_password = "sdy123" # str
6 b = True # bool
```

Pythonda haqiqiy (float) turidagi oʻzgaruvchilar  $[-10^{308}, +10^{308}]$  oraliqdagi sonlar bilan hisoblash ishlarini amalga oshirsa boʻladi, lekin faqat 18 ta raqamlar ketma-ketligi koʻrinadi (konsol ekraniga chiqarilganda). Ixtiyoriy katta yoki kichik sonlarni oʻzgaruvchidagi ifodasi 18 ta belgidan oshib ketsa, u holda eksponentsial orqali yaxlitlab ifodalanadi.

Shuni ham taʼkidlash kerakki, Pythonda oʻzgaruvchiga yangi qiymat berish orqali uning turi oʻzgartirilishi mumkin. Masalan:

```
1 age =17 # int
2 print(age)
3
4 age = "o`n etti" # str
5 print(age)
```

Ushbu dasturda dastlab *age =17* ifodasi orqali *age* oʻzgaruvchisi *int* turiga ega edi. Keyingi *age = "o`n etti"* ifoda bilan uning turi *str* turiga oʻzgartirildi. Bundan keyingi jarayonlarda *age* oʻzgaruvchisi eng ohirgi yuklangan qiymat turiga mos boʻladi.

Oʻzgaruvchilarning turini aniqlashda **type()** – funksiyasidan foydalaniladi.

Masalan:

```
1 age =17
2 print(type(age))
3
4 age = "o`n etti"
5 print(type(age))
```

Konsol ekranidagi natija:

```
<class 'int'>
```

```
<class 'str'>
```

#### 1.4. Sonlar ustuda amallar

**Arifmetik amallar.** Pythonda asosiy arifmetik amallar o'z ma'nosi bo'yicha qo'llaniladi:

**+ - qo'shish amali:**

Ikki sonni yig'indisi

```
1 print(6 + 5) # 11
```

**- - ayirish amali:**

Ikki sonni ayirmasi

```
1 print(6 - 5) # 1
```

**\* - ko'paytirish amali:**

Ikki sonni ko'paytmasi

```
1 print(6 * 5) # 30
```

**/ - bo'lish amali:**

Ikki sonni bo'lish

```
1 print(6 / 5) # 1.2
```

**// - butun qisimli bo'lish amali:**

Ikki sonni bo'linmasi (ushbu amal bo'lish natijasining faqat butun qismini qaytaradi, qoldiq qismi tashlab yuboriladi)

```
1 print(6 // 5) # 1
```

### **% - qoldiqli bo'lish amali:**

Ikki sonni bo'linmasi (ushbu amal bo'lish natijasining faqat qoldiq qismini qaytarib, butun qismi tashlab yuboriladi)

```
1 print(6 % 5) # 1
```

### **\*\* - darajaga ko'tarish (oshirish) amali:**

$a^b$  shaklidagi hisoblashlarda qo'llaniladi

```
1 print(6 ** 2) # 36
```

Ifodada bir nechta arifmetik amallar ketma-ket kelgan bo'lsa, ular prioriteti (ustunligi) bo'yicha bajariladi. Dastlab, yuqori prioritetga ega bo'lgan amallar bajariladi. Amallarning prioriteti kamayish tartibida quyidagi jadvalda ifodalangan:

<b>Amallar</b>	<b>Yo'nalish</b>
**	Chapdan-o'nga
*, /, //, %	Chapdan-o'nga
+, -	Chapdan-o'nga

Misol sifatida quyidagi ifodani qaraymiz:

```
1 son = 12//7 + 2 ** 5 * 3 - 4
2 print(son) # 93
```

Bu erda dastlab eng yuqori prioritetga ega bo'lgan amal – darajaga ko'tarish amali bajariladi ( $2 ** 5 = 32$ ). Keyin ko'paytma ( $32 * 3 = 96$ ), butun qisimli bo'lish ( $12 // 7 = 1$ ), qo'shish ( $1 + 96 = 97$ ) va ayirish ( $97 - 4 = 93$ ) amallari bajariladi. Ifoda bajarilishi natijasida 93 soni konsol ekraniga chiqariladi.

Amallarni qavsga olish orqali ularning bajarilish ketma-ketligini o'zimiz xoxlagan tartibga keltirib olishimiz ham mumkin. Masalan, yuqoridagi ifodani quyidagicha qayta yozamiz:

```
1 son = 12//7 + 2 ** 5 * (3 - 4)
2 print(son) # -31
```

Natijada konsol ekraniga -31 soni chiqariladi.

Shuni alohida ta`kidlash kerakki, arifmetik amallar butun sonlar uchun qanday tartibda bajarilsa, suzuvchan nuqtali haqiqiy sonlar uchun ham xuddi shunday bo`ladi. Agarda ifodada loaqal bitta haqiqiy son ishtirok qilsa natija haqiqiy turda ifodalanadi.

Yuqoridagi barcha arifmetik amallarni o`zlashtirish amali (=) bilan birgalikda (arifmetik amal va undan keyin "=" belgisi ketma-ket yoziladi) ishlatish mumkin. Masalan: +=, -=, \*=, /=, //=, %=, \*\*=. Bunday hollarda ifodaning o`ng tomonidagi barcha amallar hisoblanib, chiqqan natija chap tomondagi o`zgaruvchi natijasi bilan mos arifmetik amal bajariladi va natija yana chap tomondagi o`zgaruvchiga yuklanadi. Masalan:

```
1 son = 2
2 son += 3 # son = son + 3 amaliga teng kuchli, son=5 bo`ladi
3 print(son) # 5
4
5 son -= 1
6 print(son) # 4
7
8 son *= 4
9 print(son) # 16
10
11 son /= 2
12 print(son) # 8
13
14 son **= 2
15 print(son) # 64
```

Yuqoridagi misolda hisoblash natijalari kommentariyalarda ko`rsatilgan.

### 1.5. Turga keltirish (turga o`girish) funksiyasi

Pythonda sonlar ustuda amal bajaruvchi ichki funksiyalar juda ko`p. Xususan, *int()* va *float()* funksiyalari argument sifatida berilgan qiymatlarni mos ravishda butun va haqiqiy sonlarga o`girish uchun ishlatiladi. Masalan:

```
1 a = int(10.0)
2 print(a) # 10
```

```
3
4 b = float("12.3")
5 print(b) # 12.3
6
7 c = str(12)
8 print(c) # "12"
9
10 d = bool(c)
11 print(d) # True
```

Turga keltirish funksiyalari odatda konsol ekranidan kiritilgan qiymatlarni kerakli turga o'girish (chunki konsoldan kiritilgan ixtiyoriy qiymat *str* turiga tegishli bo'lishi oldindan qabul qilingan) va ifodalarda bir turdan ikkinchi turga keltirish zarur bo'lgan hollarda ishlatiladi. Masalan:

```
1 son1 = 3
2 son2 = input()
3 print(son1 + son2)
```

Ushbu dastur bajarilishi jarayonida turlar mos kelmasligi (*TypeError: unsupported operand type(s) for +: 'int' and 'str'*) to'g'risidagi xatolik ro'y bergani haqidagi xabarni chiqaradi. Turga keltirish funksiyasidan foydalanib quyidagicha dasturni qayta yozamiz:

```
1 son1 = 3
2 son2 = "12"
3 res = son1 + int(son2)
4 print(res) #15
```

Ushbu dastur konsol ekraniga 15 degan javobni chiqaradi. Demak turga keltirish amali `int()` joyida to'g'ri qallanilgan.

`float()` – turga keltirish funksiyasi ham xuddi yuqoridagidek ishlatiladi. Faqat suzuvchan nuqtali haqiqiy sonlar ustida amallar bajarilganida natija har doim ham biz kutganday bo'lmaydi. Masalan:

```
1 son1 = 4.01
2 son2 = 5
3 son3 = son1 / son2
```

```
4 print(son3) #0.8019999999999999
```

Ushbu dasturda javob 0.802 chiqishi kerak edi, lekin uni javobi yuqoridagi misolda ko'rinib turganidek 0.8019999999999999 qiymatni ekranga chiqaradi. Bu qiymat hato emas. Haqiqiy sonlarning kompyuter xotirasida saqlanish formati butun sonlarnikidan farqlanadi. Shu sababli suzuvchan nuqtali sonlar qiymati taqriban saqlanadi (absolyut xatolik inobatga olmasa ham bo'ladigan darajada kichik). Shuning uchun haqiqiy sonlarni yahlitlash uchun *round()* funksiyasidan foydalaniladi.

```
1 son1 = 4.01
2 son2 = 5
3 son3 = round (son1 / son2, 4)
4 print(son3) #0.802
```

*round(a,n)* funksiyasi ikkita parametr qabul qilib, dastlabkisi yahlitlanishi kerak bo'lgan qiymat, ikkinchisi verguldan keyin nechta belgi aniqlikda chiqarilishi kerakligini anglatuvchi son.

### 1.6. Sonlarning turli sanoq sistemalarda tasvirlanishi

Odatda, sonli o'zgaruvchilarni aniqlashda ularga qiymat o'nlik sanoq sistemasida beriladi. O'nlik sanoq sistemalardan tashqari Pythonda sonlarni ikkilik, sakkizlik va o'n oltilik sanoq sistemalarida ham ifodalash mumkin. Sonni ikkilik sanoq sistemasida ifodalash uchun qiymat oldiga **0b**, sakkizlikda **0o** va o'n oltilikda **0x** belgilar jufligi qo'yiladi. Masalan:

```
1 son1 = 5
2 son2 = 0b110 # 6
3 son3 = 0o11 # 9
4 son4 = 0x1a # 26
5 print(son1, son2, son3, son4) # 5 6 9 26
6 print(son1 +son3) # 14
```

Turli xil sanoq sistemalarda ifodalangan sonlar ustida bajarilgan amallar natijasi o'nlik sanoq sonlarda ifodalaniladi. Shuning uchun yuqoridagi dasturda `print(son1 +son3)` ifodaning natijasi 14 soni o'nlik sanoq sistemasida ekranga chiqariladi.

Bundan tashqari ixtiyoriy butun sonni ikkilik, sakkizlik va o'n oltilik sanoq sistemalarida ifodalash mumkin. Masalan, quyidagi dasturda 15 sonining turli sanoq sistemalardagi ifodalanishi tasvirlangan:

```
1 son = 15
2 print("{0}".format(son)) # 15
3 print("{0:0b}".format(son)) # 1111
4 print("{0:07b}".format(son)) # 0001111
5 print("{0:0o}".format(son)) # 17
6 print("{0:0x}".format(son)) # f
```

Yuqoridagi dasturning 4-satrida keltirilgan {0:07b} ifodadagi 7 soni yozuvida nechta raqam bo'lishi kerakligini ifodalaydi. Shuning uchun 0001111 natija hosil qilingan. Sonni ifodalovchi qiymat, ko'rsatilgan uzunlikda bo'lmasa, u holda qiymat old qismi 0 raqami bilan to'ldiriladi (yuqorida 1111 qiymat old qismiga 000 raqamlar ketma-ketligi qo'yilgan).

### 1.7. Shart ifodalari

Shart ifodalarini bir qator amallar taqdim qiladi. Ushbu amallarning barchasi ikkita operand qabul qiladi va natija sifatida *boolean* turidagi mantiqiy qiymat qaytaradi. Faqatgina ikkita mantiqiy qiymat mavjud, ular *True* (ifoda rost) *False* (ifoda yolg'on) qiymatlardir.

**Taqqoslash amallari.** Eng sodda shart ifodalariga taqqoslash amallari misol bo'lib, ular ikki qiymatni taqqoslash uchun ishlatiladi. Python quyidagi taqqoslash amallarini qo'llab-quvvatlaydi:

`==` - agar ikki operand teng bo'lsa *True*, aks holda *False* qiymat qaytaradi;

`!=` - agar ikki operand teng bo'lmasa *True*, aks holda *False* qiymat qaytaradi;

`>` (dan katta) – agar birinchi operand ikkinchisidan katta bo'lsa *True*, aks holda *False* qiymat qaytaradi;

`<` (dan kichik) – agar birinchi operand ikkinchisida kichik bo'lsa *True*, aks holda *False* qiymat qaytaradi;

`>=` (dan katta yoki teng) – agar birinchi operand ikkinchisidan katta yoki teng bo'lsa *True*, aks holda *False* qiymat qaytaradi;



`<=` (dan kichik yoki teng) – agar birinchi operand ikkinchisidan kichik yoki teng bo'lsa *True*, aks holda *False* qiymat qaytaradi;

### 1.8. Mantiqiy amallar

Murakkab shartli ifodalarni yozish, odatda mantiqiy amallar yordamida amalga oshiriladi. Pythonda quyidagi mantiqiy operatorlar mavjud:

***and*** (mantiqiy ko'paytirish). Murakkab ifodadagi biror bir qism ifodani qiymati *False* bo'lsa, ifodaning yakuniy qiymati *False*, aks holda *True* qiymat qaytaradi. Masalan:

```
1 yoshi = 21
2 vazni = 72
3 natija = yoshi > 17 and vazni == 72
4 print(natija) # True
```

Yuqoridagi dasturda murakkab mantiqiy amal ikki qismdan `yoshi > 17` va `vazni > 56` qismlardan tashkil topgan bo'lib, ular *and* mantiqiy operatori bilan birlashtirilgan. Agarda ikkala mantiqiy amal *True* qiymat qaytarsa ifodaning qiymati *True* bo'ladi, aks holda *False* qiymat qaytaradi.

Mantiqiy ifodalarda faqatgina taqqoslash amallaridan foydalanish shart emas. Ixtiyoriy mantiqiy amal yoki *boolean* turidagi qiymatlar (*True*, *False*) ham ishlatilishi mumkin. Masalan:

```
1 yoshi = 21
2 vazni = 72
3 t = True
4 natija = yoshi > 17 and vazni > 56 and t
5 print(natija) # True
```

***or*** (mantiqiy qo'shish). Agarda ifodadagi biror bir qism ifoda *True* qiymat qaytarsa, yakuniy natija ham *True*, aks holda *False* bo'ladi.

```
1 yoshi = 21
2 t = False
3 natija = yoshi > 17 or t
4 print(natija) # True
```

*not* ( mantiqiy inkor). Ifodaning qiymatini *True* bo'lsa, natija *False* va aksincha.

```
1 yoshi = 21
2 t = False
3 print(not yoshi > 17) # False
4 print(not t) # True
```

*and* operatorining biror bir operandi *False* qiymatga ega bo'lsa, u holda boshqa operand qiymati tekshirib (hisoblanib) o'tirilmaydi, har doim natija *False* bo'ladi. Bunday xususiyat ish unumdorligini bir oz bo'lsada oshirish imkonini beradi. Xuddi shunaqa xususiyat *or* operatori uchun ham o'rinli. Ya'ni *or* operatorining biror bir operandi qiymati *True* qiymatga ega bo'lsa, boshqa operandlar tekshirilmaydi, natija sifatida har doim *True* qiymati qaytariladi.

Agar bitta ifodada bir nechta mantiqiy operatorlar qatnashgan bo'lsa, u holda ularning ustunligiga (prioritetiga) alohida e'tibor qatarish kerak. Dastlab *not* operatori keyin *and* va eng so'ngra *or* operatori bajariladi. Masalan:

```
1 yoshi = 22
2 xolati = False
3 vazni = 58
4 natija = vazni == 58 or xolati and not yoshi > 21 # True
5 print(natija)
```

Ushbu dasturda keltirilgan ifodadagi mantiqiy amallar quyidagi ketma-ketlikda bajariladi:

1. *not yoshi > 21* mantiqiy ifoda *False* qiymat qaytaradi;
2. *xolati and False (not yoshi > 21)* esa *False* qiymat qaytaradi;
3. *vazni == 58 or True (xolati and not yoshi > 21)* esa *True* qiymat qaytaradi.

Shuni alohida ta'kidlash kerarki, mantiqiy ifodalarda mantiqiy amallarning bajarilish ketma-ketligini qavslar (,) yordamida o'zgartirish mumkin.

### 1.9. Satrlar ustida amallar

Satrlar – qo'shtirnoq ichiga olingan *Unicode* kodidagi belgilar ketma-ketligi orqali ifodalanadi. Pythonda satrlar apostrof ('') va qo'shtirnoqlar ("") orqali

berilishi mumkin. Uchta ketma-ket kelgan apostrof ham satrlarni ifodalashda ishlatiladi.

```
1 ismi = "Yusupov"
2 familiyasi = 'Yusuf'
3 print(ismi, familiyasi) # Yusupov Yusuf
```

Satrlar ustida eng keng tarqalgan amallardan biri bu ularni birlashtirish yoki konkatenatsiya amali hisoblanadi. Satrlarni birlashtirish uchun + amali qo'llaniladi.

Masalan:

```
1 ismi = "Yusupov"
2 familiyasi = 'Yusuf'
3 sharifi = "Qalandarovich"
4 FISH = ismi + " " + familiyasi + " " + sharifi
5 print(FISH) # Yusupov Yusuf Qalandarovich
```

Agar satr va sonlarni birlashtirish talab qilinsa, u holda `str()` funksiyasi yordamida sonni satr turiga keltirish kerak bo'ladi. Masalan:

```
1 ism = "Yusuf"
2 yosh = 33
3 info = "Ismi: " + ism + " yoshi: " + str(yosh)
4 print(info) # Ismi: Yusuf yoshi: 33
```

**Maxsus belgilar:** Pythonda boshqa tillardagi kabi quyidagi maxsus belgilar mavjud:

`\t` – tabulyatsiya belgisi;

`\n` – yangi satrga o'tish belgisi;

`\'` – apostrof belgisi;

`\"` – qo'shtirnoq belgisi.

Quyidagi misolda yuqoridagi barcha maxsus belgilarni qo'llangan holat uchun dastur keltirilgan.

```
1 print("1-chi kurs\n\"O'MU\"\ttalabasi")
```

Konsol ekraniga quyidagicha natija chiqariladi:

*1-chi kurs*

*"O'MU" talabasi*

**Satrlarni taqqoslash:** Satrlarni taqqoslashda satrda ishtirok etayotgan belgilarning registriga alohida e`tibor qaratish lozim. Har qanday raqam ixtiyoriy alfavit belgisidan shartli kichik hamda katta registrli alfavit belgilari kichik registrli avfavit belgilaridan shartli kichik sanaladi. Masalan:

```
1 str1 = "1a"
2 str2 = "ab"
3 str3 = "Aa"
4 print(str1 > str2) # False, chunki str1 ning birinchi
5                     # belgisi raqam
6 print(str2 > str3) # True, chunki str2 ning birinchi
7                     # belgisi kichik registrga ega
```

Yuqoridagi dasturda "1a">"ab" sharti *False* qiymat qaytaradi. Chunki raqam alfavit belgisidan shartli kichik hisoblanadi. Shuni alohida ta`kidlash kerakki, ikki satr solishtirilganda ularning mos tarkibiy elementlari solishritiladi("1a">"ab" holatda, dastlab 1 va "a" tekshiriladi). Agarda solishtirish natijasi teng bo`lsa navbatdagi mos elementlari solishtiriladi. Jarayon birinchi teng bo`lmagan holat topilganda yoki satrlardan birining oxiriga yetib kelinganda tugatiladi. Agar satrlarning dastlabki barcha mos elementlari teng, faqat ularning uzunliklari farqli bo`lsa, u holda uzunligi katta satr shartli katta bo`ladi. Masalan: "abcd"<"abcde" sharti *True*

Bundan tashqari satrlar ustuda amal bajaradigan *upper()* va *lower()* funksiyalari mavjud bo`lib, satr tarkibidagi alfavit belgilarni mos ravishda kichik va katta registrli lariga almashtirish uchun ishlatiladi. Masalan:

```
1 str1 = "Kitob"
2 str2 = "kitob"
3 print(str1 == str2) # False - chunki ularni birinchi
4                     # harflari turli registrda
5 print(str1.lower() == str2.lower()) # True chunki ikkala
67                     # satr ham kichik registrga keltirilgan
```

### 1.10. if - shart amali (operatori)

*if* shart amali shart ifodalarda qo'llanilib, uning natijasiga ko'ra dastur bajarilishi u yoki bu yo'lga yo'naltiriladi. U quyidagi umumiy ko'rinishga ega:

*if* mantiqiy ifoda:

*ifodalar*

[*elif* mantiqiy ifoda:

*ifodalar*]

[*else*:

*ifodalar*]

*if* shart operatorining eng sodda ko'rinishida *if* kalit so'zidan keyin mantiqiy ifoda yoziladi va ikki nuqta (:) qo'yiladi. Keyingi qatordan amallar yoziladi. Har bir amal alohida qatorda yozilishi yoki ularni nuqta vergul (;) bilan ajratgan holda bitta qatordan yozish talab qilinadi. Shuni alohida ta'kidlash kerakki Pythonda boshqa tillardagi kabi *if* shart amalini tana qismini ifodalovchi maxsus belgilar mavjud emas (manasal c++, c# da {,} blok belgilari ishlatiladi). Shu sababli uning tana qismidagi ifodalar *if* kalit so'ziga nisbatan bitta xat boshi (to'rtta probel belgisi) belgisi tashlab yoziladi. Masalan:

```
1 | yoshi = 21
2 | if yoshi >18:
3 |     print("Kirishga ruxsat beriladi")
4 | print("Tamom")
```

Bu erda *if* kalit so'zidan keyin *yoshi >18* mantiqiy ifoda kelgan. Tana qismi bitta ifodadan tashkil topgan, ya'ni *print("Kirishga ruxsat beriladi")* va u *if* ga nisbatan bitta xat boshi tashlab yozilgan. Keyingi qatordagi *print("Tamom")* ifodasi *if* ning tana qismiga tegishli emas, shuning uchun u *if* bilan bir ustunda yozishgan va bu xabar shart bajarilish-bajarilmasligidan qat'iy nazar har doim konsol ekraniga chiqariladi.

Agarda *print("Tamom")* ifodasi oldiga bitta xat boshi qo'ysak, u holda ushbu ifoda ham *if* blokiga tegishli bo'lib qoladi, ya'ni

```
1 | yoshi = 21
```

```

2  if yoshi >18:
3      print("Kirishga ruxsat beriladi")
4      print("Tamom")

```

Ushbu holatda shart bajarilsa, ikkala xabar ham konsol ekraniga chiqariladi, aks holda hech biri chiqarilmaydi.

*if* shart ifodasi *false* qiymat qaytaradigan holatda qandaydir amal bajarilishini aniqlash uchun *else* blokida bajarilishi kerak bo'lgan amallar yoziladi. Masalan:

```

1  yoshi = 21
2  if yoshi >18:
3      print("Kirishga ruxsat beriladi")
4  else:
      print("Kirishga ruxsat berilmaydi")

```

Agar *yoshi >18* shart bajarilsa *if* blokidagi aks holda *else* blokidagi amallar bajariladi.

Bir necha alternativ shartlarni ishlatish uchun qo'shimcha *elif* blokidan foydalaniladi.

```

1  # ax^2+bx+c=0 kvadrat tenglama echimlari soni
2  a= int(input("a="))
3  b= int(input("b="))
4  c= int(input("c="))
5  d = b**2 - 4*a*c
6  if d >0:
7      print("Tenglama 2 ta haqiqiy echimga ega")
8  elif d == 0:
9      print("Tenglama 1 ta haqiqiy echimga ega")
10 else:
11     print("Tenglama haqiqiy echimga ega emas")

```

**Ichma-ich joylashgan *if* shart amali.** *if* shart operatori o'z navbatida boshqa *if* shart operatorlaridan tashkil topgan bo'lishi mumkin. Bunday holatga ichma – ich joylashgan shart ifodasi deyiladi. Masalan:

```

1  protsent = int(input("Protsentni kiriting: "))
2  if protsent > 10:

```

```

3     print("10% dan katta")
4     if protsent > 20:
5         print("20% dan katta")

```

Yuqoridagi misolda ichki *if* ifodasi tashqarisidagiga nisbatan bitta xat boshi tashlab yozilishi shart, aks holda ichma – ich joylashgan shart operatori bo'lmay, alohida shart operatori hosil qilingan bo'ladi.

Quyidagi *if* operatoriga misolda oylik maoshdan shkala bo'yicha tutib qolinadigan jami daromad solig'ini hisoblovchi dastur tuzilgan:

```

1     # Qoidaga ko`ra daromad solig`i eng kam ish haqiga (EKIH)
2     bog`liq
3     maosh = int(input("Oylin summasini kiriting"))
4     EKIH = int(input("Eng kam ish haqini kiriting"))
5     dar_soliq = 0
6     if maosh < 6 * EKIH:
7         dar_soliq = maosh * 0.065
8     elif maosh < 10 * EKIH:
9         dar_soliq = 6 * EKIH * 0.065 + (maosh - 6 * EKIH) * 0.165
10    else:
11        dar_soliq = 6 * EKIH * 0.065 + 4 * EKIH * 0.165 \
12            + (maosh - 10 * EKIH) * 0.225
13    print("Oylikdan ushab qolingani daromad solig`i: ", dar_soliq)

```

### 1.11. Sikl operatorlari

Odatda sikl operatorlari biror - bir jarayonni qandaydir shart asosida takrorlash uchun ishlatiladi. Python da sikl operatorlarining ikki turi, *while* va *for* qaraladi.

#### **while takrorlash operatori**

*while* takrorlash operatori quyidagi umumiy ko'rinishga ega:

*while shart ifodasi:*

*instruksiyalar*

*while* kalit so'zidan keyin shart ifodasi ko'rsatiladi va ushbu shart ifodasi rost qiymat (*True*) bo'lar ekan amallar ketma-ketligi takror va takror bajarilishda

davom ettiriladi. *while* operatorining barcha instruksiyalari undan keyingi qatorda yoziladi va u *while* kalit soʻzidan bitta xat boshi tashlab yoziladi. Masalan:

```
1 sum = 0
2 n = int(input("n="))
3 i = 1
4 while i <= n:
5     sum = sum + i
6     i += 1
7 print("summa(1+...+n) =", sum)
```

Yuqoridagi misolda 1 dan  $n$  gacha boʻlgan sonlar yigʻindisi hisoblash dasturi *while* operatori yordamida amalga oshirilgan. Eʼtibor berilsa *while* operatorining instruksiyalari undan keyingi qatorda bitta xat boshi tashlab yozilgan. Ushbu holatda *while* operatori 2 ta instruksiyalardan tashkil topgan ( $sum = sum + i$  va  $i += 1$ ).

### ***for* takrorlash operatori**

Yana bir takrorlash operatori – *for* operatori hisoblanadi. *for* takrorlash operatori qandaydir sonlar kolleksiyasidagi har bir son uchun chaqiriladi. Sonlar kolleksiyasi *range()* funksiyasi, *list()* funksiyasi yoki [,] qavslarda foydalanuvchi tomonidan shakllantirilgan roʻyxatlar orqali hosil qilinadi. Quyida *for* takrorlash operatorining formal aniqlanishi keltirilgan:

*for* int\_var in funksiya\_range:

instruksiyalar

*for* kalit soʻzidan keyin *int\_var* oʻzgaruvchisi (oʻzgaruvchi nomi ixtiyoriy boʻlishi mumkin) keladi va u faqat butun turdagi qiymatlar qabul qiladi, undan keyin *in* kalit soʻzi (*in* operatori) va *range* funksiyasi chaqirilgan va oxirida “:” belgisi bilan takrorlash operatori asosiy qismi tugaydi. *for* takrorlash operatorining tana qismi bir yoki bir nechta instruksiyalardan tashkil topishi mumkin va ular asosiy qismga nisbatan bitta xat boshi tashlab yoziladi.

Takrorlash operatori bajarilganda *range()* funksiyasi hosil qilgan sonlar kolleksiyasidan sonlar ketma-ket *int\_var* oʻzgaruvchisiga uzatiladi. Sikl boʻyicha



barcha sonlar ketma-ket o'tib bo'lingandan keyin takrorlash operatori o'z ishiti tugatadi.

Quyida 1 dan  $n$  gacha bo'lgan sonlar yig'indisini hisoblash dasturi *for* operatori yordamida amalga oshirilgan:

```
1 sum = 0
2 n = int(input("n="))
3 for i in range(1,n+1):
4     sum = sum + i
5 print("summa(1+...+n) =", sum)
```

Dastlab konsol ekranidan butun son kiritiladi. Siklda  $i$  o'zgaruvchisi aniqlangan bo'lib, u *range()* funksiyasidan qaytarilgan qiymatni o'zida saqlaydi. Bu erda *range()* funksiyasi 2 ta parametr qabul qilgan. Birinchisi sonlar kolleksiyasini boshlang'ich qiymati va ikkinchisi oxirgi qiymati ( oxirgi qiymat kolleksiyaga kirmaydi). Natijada *range()* funksiyasi  $[1, \dots, n-1]$  intervaldagi sonlarni ketma-ket qiymat sifatida qaytaradi va har bir qiymat uchun sikl operatorining tana qismi bajariladi.

***range* funksiyasi.** *range* funksiyasining quyidagi shakllari mavjud:

*range(stop)* – 0 dan *stop* gacha (*stop* kirmaydi) bo'lgan barcha sonlarni qaytaradi;

*range(start, stop)* – *start* (kiradi) dan *stop* (kirmaydi) gacha bo'lgan barcha butun sonlarni qaytaradi;

*range(start, stop, step)* – *start* (kiradi) dan *stop* (kirmaydi) gacha bo'lgan barcha butun sonlar *step* qadam bilan hosil qilinadi va qaytaradi.

Masalan:

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

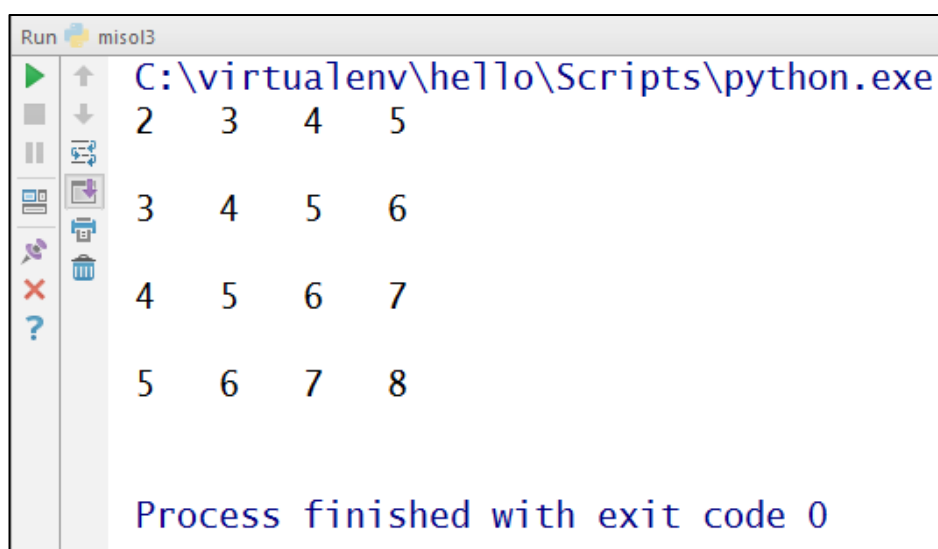
Bu erda *range(5)* funksiyasi  $[0, 1, 2, 3, 4]$  oraliqdagi sonlar kolleksiyasini qaytargan va qaytarilgan sonlarni ro'yxatda jamlash uchun *list()* funksiyasi

qo'llanilgan. `range()` funksiyasining boshqa holatlarda qanday qiymatlar hosil qilishini yuqoridagi dasturdan ko'rish mumkin.

**Ichma-ich joylashgan sikllar.** Biror bir takrorlash operatori tanasida boshqa takrorlash operatorining ishlatilishiga ichma-ich joylashgan sikl deyiladi. Masalan:

```
1 for i in range(1, 5):
2     for j in range(1, 5):
3         print(i + j, end="\t")
4     print("\n")
```

Natija konsolda quyidagi ko'rinishda chiqariladi:



**Rasm №2.** Natijaning konsol ekranidagi ko'rinishi

Yuqoridagi holatda `for i in range(1,5)` bo'yicha tashqi siklning har bir iteratsiyasi uchun, `for j in range(1,5)` bo'yicha ichki sikl bajariladi.

**Sikldan chiqish. `break` va `continue` operatorlari.** Sikllarni boshqarish uchun `break` va `continue` kabi maxsus operatorlardan foydalaniladi. `break` operatori sikldan chiqish uchun ishlatiladi. `continue` operatori siklning navbatdagi iteratsiyasiga o'tish uchun ishlatiladi.

Odatda `break` operatori siklda shart operatorlari bilan birga qo'llaniladi, masalan:

```
1 while True:
2     ch = input("Chiqish uchun 'Y' klavishini bosing")
3     if ch.lower() == 'y':
4         break
```

```

5     s=0
6     n = int(input("n="))
7     for i in range(1,n+1):
8         s += i
9     print("Summa (1,...,n)=", s)

```

Yuqoridagi dasturda foydalanuvchi tomonidan kiritilgan  $n$  uchun 1 dan  $n$  gacha bo'lgan sonlar yig'indisini hisoblash amalga oshirilgan. Agar foydalanuvchi yana boshqa son uchun yig'indini hisoblamoqchi bo'lsa, dasturdan chiqib ketmasdan uni davom ettirishi mumkin. Buning uchun u 'Y' belgisidan boshqa ixtiyoriy belgini ekrandan kiritishi kerak. Sikldan chiqish sharti `if ch.lower() == 'y'` da tekshirilgan.

### 1.12. Funksiyalar

Funksiyalar ma'lum bir vazifani bajaradigan va dasturning boshqa qismlarida qayta ishlatilishi mumkin bo'lgan kod blokini ifodalaydi. Funksiyaning rasmiy ta'rifi quyidagicha:

**def** funksiya\_nomi([parametrlar ro'yxati]):

amallar

funksiyaning aniqlanishi *def* kalit so'zi, funksiya nomi, oddiy ochiluvchi va yopiluvchi qavslar, ikki nuqta hamda funksiya tana qismini ifodalovchi amallar ketma-ketligidan tashkil topadi. Oddiy qavs ichida parametrlar ro'yxati keltirilib, u ixtiyoriy hisoblanadi. Funksiyaning tana qismi uning sarlavha qismiga nisbatan bitta xat boshi tashlab yozilishi shart. Masalan:

```

1     def Display():
2         print("Python tilida funksiya e'loni!")

```

Ushbu funksiyaning nomi Display bo'lib, u parametrga ega emas. Bu funksiya chaqirilganda konsol ekraniga **"Python tilida funksiya e'loni!"** satri chiqariladi.

Funksiyani chaqirish uchun uning nomi va oddiy qavslar ichida mos parametrlariga qiymatlar berish orqali amalga oshiriladi, masalan:

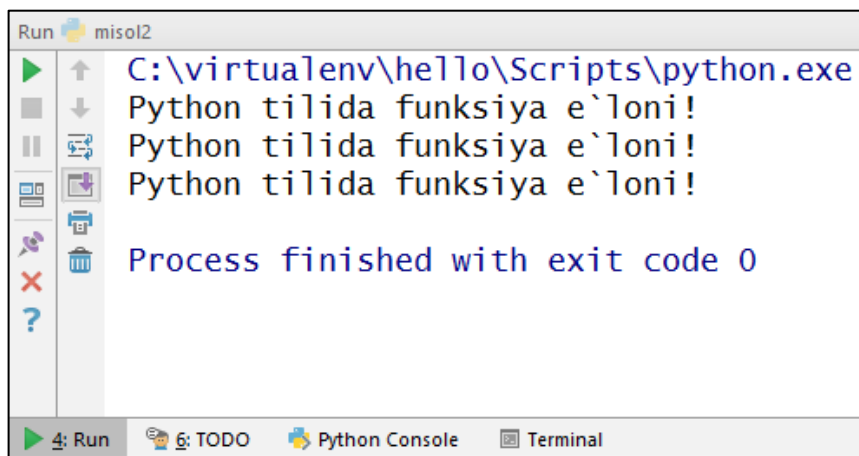
```

1     def Display():
2         print("Python tilida funksiya e'loni!")

```

```
3
4 Display()
5 Display()
6 Display()
```

Bu funksiya uch marta chaqirilmoqda va konsol ekraniga quyidagi ma`lumotlar chiqariladi:



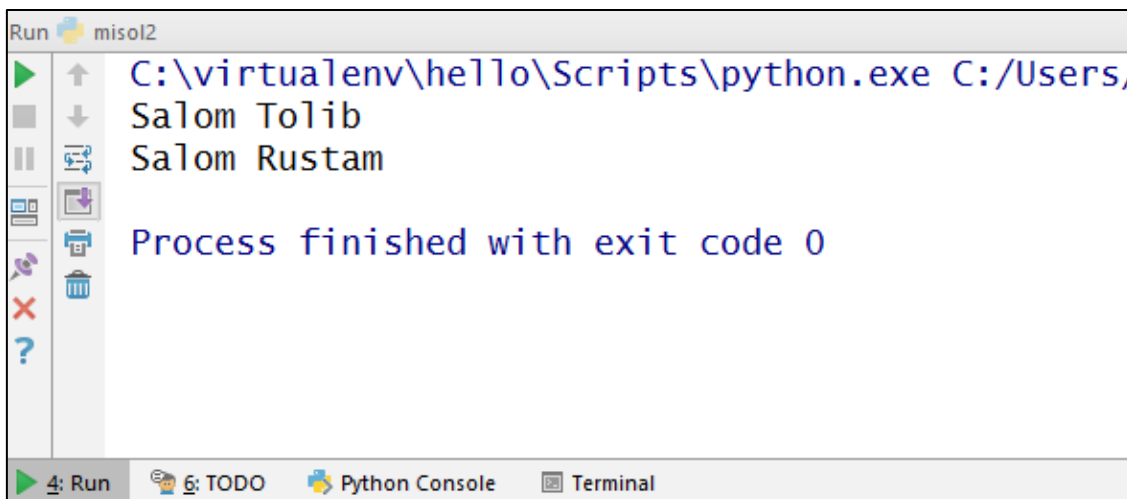
```
Run misol2
C:\virtualenv\hello\Scripts\python.exe
Python tilida funksiya e`loni!
Python tilida funksiya e`loni!
Python tilida funksiya e`loni!
Process finished with exit code 0
```

**Rasm №3.**Natijaning konsolga chiqarilishi.

Quyida parametrli funksiyaning aniqlanishiga misol keltirilgan:

```
1 def Salom(ismi):
2     print("Salom", ismi)
3
4 Salom("Tolib")
5 Salom("Rustam")
```

bu funksiya *ismi* nomli parametrga ega bo`lib, funksiya chaqirilganda parametrga turli qiymatlar berilgan va natijada konsolga quyidagi ma`lumotlar chiqarilgan:



```
Run misol2
C:\virtualenv\hello\Scripts\python.exe C:/Users/
Salom Tolib
Salom Rustam
Process finished with exit code 0
```

**Rasm №4.** Natijaning konsolga chiqarilishi.

### 1.13. O'zgaruvchilarning ko'rinish sohasi

O'zgaruvchilarning ko'rinish sohasi ularning dasturdagi qo'llanilishi mumkin bo'lgan qismi (kontekst)ga aytiladi. Pythonda kontekstlarning ikki turi mavjud: global va lokal.

Global o'zgaruvchilar barcha funksiyalardan tashqarida aniqlangan bo'ladi va ixtiyoriy funksiyaning ichkarisida foydalanilish imkonini beradi. Masalan:

```
1 ismi = "Sardor"
2 def Salom():
3     print("Salom", ismi)
4 def Xayr():
5     print("Xayr", ismi)
6 Salom()
7 Xayr()
```

Bu erda *ismi* o'zgaruvchisi global qilib aniqlangan, shuning uchun ushbu o'zgaruvchi ikkita funksiyaning ichida hech qanday muammosiz ishlatilgan ya'ni ko'rinish sohasi funksiyalarga nisbatan ham global sifatida qo'llanilgan.

Global o'zgaruvchilardan farqli ravishda lokal o'zgaruvchilar o'zi aniqlangan funksiyaning ichkarisida (tana qismida) ishlatilishi mumkin, ya'ni uning ko'rinish sohasi funksiyaning tana qismini qamrab oladi.

```
1 def Salom():
2     ismi = "Tolib"
3     familiyasi = "Otaboyev"
4     print("Salom", ismi, familiyasi)
5 def Xayr():
6     ismi = "Tolib"
7     print("Xayr", ismi)
8 Salom()
9 Xayr()
```

Bu erda har ikkala funksiyada *ismi* - lokal o'zgaruvchilari aniqlangan va ularning qo'rinish sohalari o'zi joylashgan funksiyaning ichida bo'lib, ularning har biri faqat o'zi joylashgan funksiya ichida amal qiladi.

Agar lokal o'zgaruvchi va global o'zgaruvchi bir xil nomga ega bo'lsa, u holda lokal o'zgaruvchi o'zining ko'rinish sohasida global o'zgaruvchini "yashirib" qo'yadi. Masalan:

```
1 ismi = "Tolib"
2
3 def Salom():
4     print("Salom", ismi)
5 def Xayr():
6     ismi = "G'olib"
7     print("Xayr", ismi)
8 Salom() # Salom Tolib
9 Xayr() # Xayr G'olib
```

Bu erda 1-qatorda *ismi* deb nomlangan global o'zgaruvchi aniqlangan va xuddi shu nom bilan *Xayr()* funksiyasining ichida (6-qatorga qarang) ham lokal o'zgaruvchi aniqlangan. Funksiya ichida aniqlangan lokal o'zgaruvchi, funksiya ichida global o'zgaruvchi "yashirib" qo'yadi. Shuning uchun *Xayr()* funksiyasi chaqirilganda javobga lokal o'zgaruvchining qiymati chiqarilgan.

Agar funksiyalarning ichida global o'zgaruvchining qiymatini o'zgartirish talab qilinsa, u holda *global* kalit so'zidan foydalaniladi.

```
1 def Xayr():
2     global ismi
3     ismi = "G'olib"
4     print("Xayr", ismi)
```

Funksiya ichkarisida global o'zgaruvchining qiymati o'zgartirilishidan oldin *global* kalit so'zi orqali ko'rsatib o'tilishi shart (2-qatorga qarang).

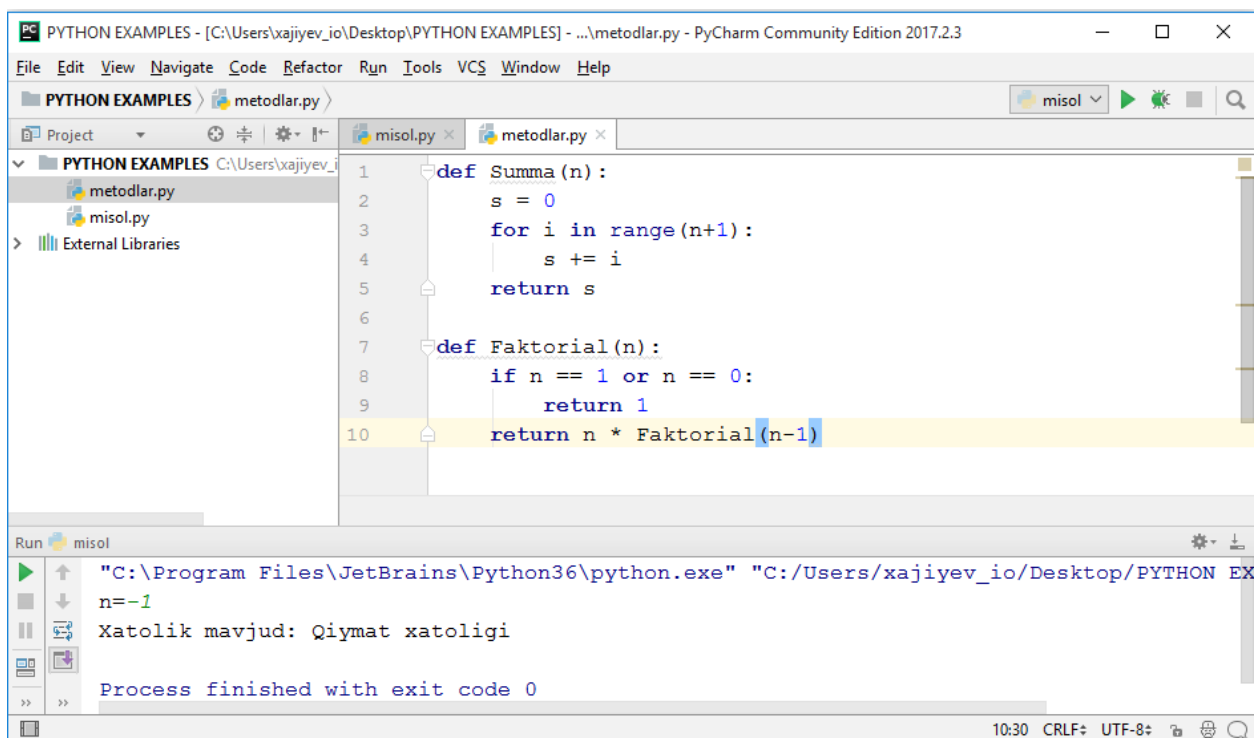
Odatda boshqa dasturlash tillaridagi kabi Python dasturlash tilida ham global o'zgaruvchilardan iloji boricha kamroq foydanish tavsiya qilinadi.

### 1.14. Modullar

Pythonda modullar alohida faylda yozilgan va boshqa dasturlarda qayta qo'llanilishi mumkun bo'lgan kodlar majmuini ifodalaydi. Modullarni hosil qilish

uchun **\*.py** kengaytmali fayl ochiladi va unga bir yoki bir nechta funksiyalar yoziladi. Faylning nomi keyinchalik modulning nomi sifatida qo'llaniladi.

Quyida Pycharm muhitida loyiha ikkita fayldan, **misol.py** nomli asosiy fayl va **metodlar.py** qo'shimcha tashqi modulni ifodalovchi fayldan tashkil topgan holatga misol keltirilgan:



```
1 def Summa(n):
2     s = 0
3     for i in range(n+1):
4         s += i
5     return s
6
7 def Faktorial(n):
8     if n == 1 or n == 0:
9         return 1
10    return n * Faktorial(n-1)
```

```
Run misol
"C:\Program Files\JetBrains\Python36\python.exe" "C:/Users/xajiyev_io/Desktop/PYTHON EX
n=-1
Xatolik mavjud: Qiymat xatoligi
Process finished with exit code 0
```

### Rasm №5. Pythonda modullarni yaratish va ulash.

Yuqoridagi rasmda (5 – rasimga qarang) **metodlar.py** nomli faylda **Summa(n)** va **Faktorial(n)** deb ataluvchi ikkita funksiya aniqlangan, ya'ni:

```
1 def Summa(n):
2     s = 0
3     for i in range(1, n + 1):
4         s += i
5     return s
6
7 def Faktorial(n):
8     if n == 1 or n == 0:
9         return 1
10    return n * Faktorial(n - 1)
```

Ushbu funksiyalarni boshqa bir faylda (5-rasmda *misoll.py* deb nomlangan faylda ko'rsatilgan) yoziladigan kodda modul sifatida ulab qo'llanilishi quyidagi dasturda ko'rsatilgan:

```
1 import metodlar
2
3 try:
4     n = int(input('n='))
5     print('Summa:', metodlar.Summa(n))
6     print('Faktorial', metodlar.Faktorial(n))
7 except:
8     print("Qiyamat kiritish xatoligi ro'y berdi")
```

Moduldan foydalanish uchun dastlab uni dasturga ulash talab qilinadi. Buning uchun *import* kalit so'zi va undan keyin modul fayli nomi ko'rsatilishi kerak:

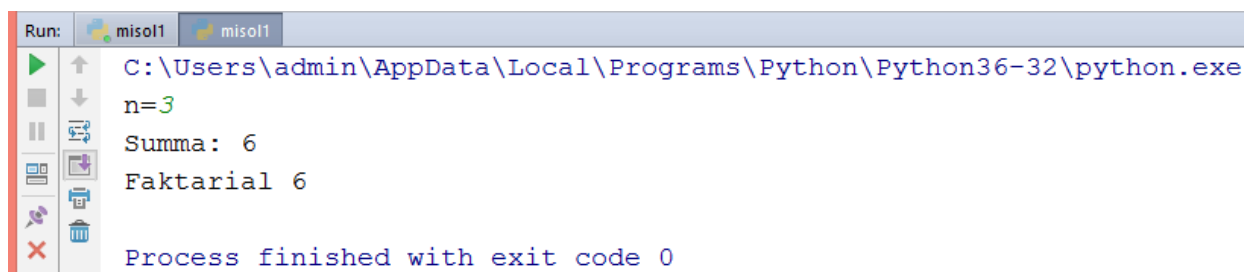
*import metodlar.*

Modul funksionallariga murojaat qilish uchun uning **nomlar fazosi** olishishi kerak. Odatda u modul nomi bilan mos tushadi. Bizning holatda bu *metodlar* deb nomlangan.

Modulning nomlar fazosi olingandan keyin uning ichidagi funksiyalarga *nomlar\_fazosi.funksiya* sxemasi bo'yicha murojaat qilinadi:

*metodlar.Faktorial(n).*

*misoll.py* fayli kompilyatsiya qilinganda unga ulangan modullarda joylashgan funksiyalarga murojaat amalga oshiriladi. Xususan, dastur ishlaganda qonsolda quyidagicha ma'lumotlar chiqariladi:



**Rasm №6.** Natijaning konsolga chiqarilishi.

**Nomlar fazosini sozlash.** Odatda modullar import qilinganda, nomlar fazosi va modul fayli nomlari bir xil bo'ladi va modul funksionallariga shu nomlar fazosi



orqali murojaat qilinadi. Lekin baʼzi holatlarda nomlar fazosi nomining haddan ziyod uzunligi noqulayliklar keltirib chiqarishi mumkin. Bunday holatlarda nomlar fazosiga *as* kalit soʻzidan foydalanib, psevdonom berish va u orqali modul funksionallariga murojaatlarni amalga oshirish mumkin. Masalan:

```
1 import metodlar as mt
2
3 try:
4     n = int(input('n='))
5     print('Summa:', mt.Summa(n))
6     print('Faktorial', mt.Faktorial(n))
7 except:
8     print("Qiyamat kiritish xatoligi ro'y berdi")
```

Bu holatda nomlar fazosi *mt* deb nomlangan.

Modullarni ulashning boshqa usullaridan biri *from* kalit soʻzidan foydalangan holda amalga oshiriladi. Bunda joriy modulning global nomlar fazosiga ulanilayotgan modulning funksionallari chaqiriladi:

```
1 from metodlar import Summa
2 print('Summa:', Summa(n))
```

Bu holatda *metodlar* modulidan *Summa* funksiyasi global nomlar fazosiga import qilingan. Shuning uchun import qilingan funksiyadan foydalanilganda (funksiya joriy faylda chaqirilganida), xuddi bu funksiya shu faylda aniqlangandek, uning oldida nomlar fazosi koʻrsatilmaslgi ham mumkin.

Moduldagi barcha funksiyalarni global nomlar fazosiga birdaniga import qilish uchun maxsus belgi *\** dan foydalaniladi.

```
1 from metodlar import *
2 print('Summa:', Summa(n))
   print('Faktorial', Faktorial(n))
```

Shuni alohida taʼkidlash kerakki, moduldagi barcha funksiyalarni bunday tarzda global nomlar fazosiga import qilish kolliziyaga olib kelishi mumkin. Masalan, agar joriy faylda ham import qilinayotgan moduldagi funksiya nomi bilan ayni mos tushadigan funksiya yoki oʻzgaruvchilar boʻlsa kolliziya holati boʻladi va dastur

ishga tushirilganda xatolikka olib kelishi mumkin. Shuning uchun odatda bu tarzda modulni ulashga maslahat berilmaydi.

**Modul nomi.** 5-rasmdagi keltirilgan dasturda *misol.py* fayli asosiy modulni ifodalab, unga *metodlar.py* moduli ulangan edi. *misol.py* ni ishga tushirganda barcha zarur ishlar bajariladi. Shuningdek, *metodlar.py* ning yakka o'zi ishga tushirilsa hech qanaqa ish bajarilmaydi va konsolga hech nima chiqmaydi. Chunki ushbu modulda faqatgina funksiyalar aniqlangan va boshqa ish bajarilmagan. Lekin *metodlar.py* ga shunday o'zgartirishlar kiritish mumkinki, uning alohida o'zi ham bajarilishi yoki u boshqa modulga ulanishi ham mumkin bo'ladi.

Modul bajarilganda muhit uning nomini aniqlaydi va uning nomini `__name__` (oldin va keyin ikkita tag chizig'i mavjud) global o'zgaruvchisiga yuklaydi. Agar modul bajariluvchi bo'lsa, uning nomi `__main__` (har ikkala tomonida ikkita tag chizig'i bor) ga teng bo'ladi. Agar modul boshqa modulda ishlatilayotgan bo'lsa, u holda bajarilish paytida uning nomi mos ravishda fayl nomi bilan bir xil bo'ladi (faqat `.py` kengaytmasisiz). *metodlar.py* fayliga quyidagicha o'zgartirishlar kiritamiz:

```
1  def Summa(n):
2      s = 0
3      for i in range(1, n + 1):
4          s += i
5      return s
6
7  def Faktorial(n):
8      if n == 1 or n == 0:
9          return 1
10     return n * Faktorial(n - 1)
11
12 def Main():
13     n = 4
14     s = Summa(n)
15     f = Faktorial(n)
16     print('Summa:', s)
```

```
17     print('Faktorial:', f)
18
19 if __name__ == "__main__":
20     Main()
```

Endi bu modul boshqa modullarga bog'liq bo'lmagan holda o'zi ham bajarilishi mumkin. Odatda modullarning bunday aniqlanishi undagi funktsionallarni tekshirib ko'rish uchun qilinadi. Yuqoridagi holatda qo'shimcha *Main* funksiyasi aniqlanib, unda test qilinishi kerak bo'lgan funksiyalar chaqirilgan. Agar shu modulning o'zi bajarilganda (kompilyatsiya qilinganda), u holda *if* sharti bajariladi va natijada *Main* funksiyasi chaqiriladi. Albatta qo'shimcha *Main* aniqlanishi shart emas. Test qilish uchun barcha funksiyalarni *if* operatorining ichida chaqirish ham kifoya bo'lar edi.

Shunday qilib, *metodlar.py* skriptining o'zi alohida ishga tushirilsa, o'z o'zidan *Python* `__name__` global o'zgaruvchisiga `__main__` nomini beradi. Agar u boshqa skriptga qo'shimcha modul sifatida ulangan bo'lsa va boshqa skriptda chaqirilsa, u holda `__name__` global o'zgaruvchisiga *metodlar* nomi beriladi.

### 1.15. Istisno holatlar bilan ishlash

Pythonda kod yozilganda ikki turdagi xatoliklarni uchratish mumkin: Birinchi turi bu **sistaksis xatoligi (syntax error)** bo'lib, bunday turdagi xatoliklar odatda til sintaksisidan noto'g'ri foydalanganda yuzaga keladi. Agarda dasturda sintaksis xatoligi bo'lsa, uni kompilyatsiya qilishning imkoni bo'lmaydi. Kod yozishda Pycharm yoki shunga o'xshash boshqa IDElardan foydalanganda, odatda muhitning o'zi bunday turdagi xatoliklarni aniqlab beradi va ajratib ko'rsatadi; Ikkinchi turdagi xatoliklar bu **bajarilish davridagi xatolik (Runtime error)** bo'lib, odatda bunday turdagi xatoliklar kompilyatsiya bosqichida emas balki dastur bajarilishi jarayonida yuzaga keladigan xatoliklar hisoblanadi. Bunday xatoliklarga istisno holatlar ham deyiladi. Masalan:

```
1 str = "hello"
2 number = int(str)
3 print(number)
```

Bu erda sistaksis xatolik kuzatilmaydi, lekin kompilyatsiya jarayonida *ValueError* deb nomlanuvchi xatolik konsolga chiqariladi. Chunki *str* o'zgaruvchisida satr qiymat mavjud va uni 2-qatorda butun turga o'girilmoqda, bu esa mumkin emas. Bunday turdagi istisno holatlar ayniqsa foydalanuvchi tomonidan qiymatlar konsol ekranidan kiritilganda ro'y berish ehtimolligi kattaroq. Masalan:

```
1 str = input("Satr kiriting:")
2 number = int(str)
3 print(number)
```

Agar foydalanuvchi tomonidan konsoldan sondan farqli bo'lgan qiymatlar kiritilsa xatolik ro'y beradi.

Istisno holatlar ro'y berganda, dastur ishlashdan to'xtatiladi. Bunday holatlarni boshqarish uchun Pythonda maxsus *try except* tuzilmasi mavjud bo'lib, u quyidagicha aniqlanadi:

***try:***

*instruksiyalar*

***except [Istisno\_turi]:***

*instruksiyalar*

Istisno holati yuzaga kelishi mumkin bo'lgan asosiy kodlar *try* kalit so'zidan keyingi qatordan boshlab yoziladi. Agar shu kodda istisno holati ro'y bergan taqdirda, kodlar bajarilishdan to'xtaydi va boshqaruv *except* blokiga uzatiladi. Bu blok *except* kalit so'zidan boshlanib, keyingi qatoridan istisno ro'y berdanda bajariladigan amallar ketma-ketligi yoziladi:

```
1 try:
2     son = int(input("Sonni kiriting: "))
3     print("Kiritilgan son:", son)
4 except:
5     print("O`girish noto`g`ri amalga oshirildi.")
6 print("Dastur tugadi.")
```

Ushbu dasturda *son* o'zgaruvchisiga qiymat sifatida konsoldan satr kiritilsa konsol ekraniga quyidagi ma'lumotlar chiqariladi:

```
Run: misol1 misol1
C:\Users\admin\AppData\Local\Programs\Python\Python36-32\python.exe
Sonni kiriting: a1
O`girish noto`g`ri amalga oshirildi.
Dastur tugadi.

Process finished with exit code 0
```

**Rasm №7.** Konsol ekranida istisno ro'y bergan holatdagi ma'lumotlar.

Agar *son* o'zgaruvchisiga qiymat sifatida konsoldan son kiritilsa konsol ekraniga quyidagi ma'lumotlar chiqariladi:

```
Run: misol1 misol1
C:\Users\admin\AppData\Local\Programs\Python\Python36-32\python.exe
Sonni kiriting: 12
Kiritilgan son: 12
Dastur tugadi.

Process finished with exit code 0
```

**Rasm №8.** Konsol ekranida qiymat to'g'ri kiritilgandagi holat.

Yuqorida, dastur ishga tushirilganda qiymat xato kiritilgandagi (Rasm №7) va to'g'ri kiritilgan paytdagi (Rasm №8) holatlarlarda konsol ekraniga chiqarilgan ma'lumotlarni ko'rish mumkin. Shuni alohida ta'kidlash kerakki, agar *try – except* tuzilmasi qamrovida istisno holati yuzaga kelgan taqdirda dastur to'xtab (qotib) qolmaydi, balki boshqaruv istisno holatini qayta ishlovchi qismga (*except blokiga*) uzatiladi.

Yuqoridagi misolda kodda yuzaga kelishi mumkin bo'lgan barcha istisno holatlarni qayta ishlovchi holat ko'rildi. Ammo qayta ishlanuvchi istisno turini konkretlashtirish ham mumkin. Buning uchun *except* kalit so'zidan keyin istisno turi yoziladi:

```
1 try:
2     son = int(input("Sonni kiriting: "))
3     print("Kiritilgan son:", son)
4 except ValueError:
5     print("O`girish noto`g`ri amalga oshirildi.")
```

```
6 print("Dastur tugadi.")
```

Agar kodda bir necha xil istisno holatlari ro'y berishi mumkin bo'lsa, qo'shimcha *except* ifodalardan foydalangan xolda, ularning har birini alohida qayta ishlashimiz mumkin bo'ladi, masalan:

```
1 try:
2     son1 = int(input("Birinchi sonni kiriting: "))
3     son2 = int(input("Ikkinchi sonni kiriting: "))
4     print("Bo`lish natijasi:", son1/son2)
5 except ValueError:
6     print("O`girish muaffaqiyatsiz amalga oshirildi.")
7 except ZeroDivisionError:
8     print("Nolga bo`lish yuzaga keldi.")
9 except Exception:
10    print("Umumiy istisno holati.")
11 print("Dastur tugadi.")
```

Agar kiritilgan satrni songa o'girishda istisno ro'y bersa *ValueError* bloki, kiritilgan ikkinchi qiymat 0 soniga teng bo'lsa *ZeroDivisionError* bloki orqali qayta ishlanadi. Shu ikki istisno turidan boshqa ixtiyoriy istisno ro'y bersa *Exception* blokida qayta ishlanadi.

**Finally bloki.** Istisno holatlar bilan ishlaganda *finally* blogini ham ishlatish mumkin. Bu blok ixtiyoriy hisoblanib, uning o'ziga xosligi shundan iboratki, ushbu blokda yozilgan kodlar istisno holati ro'y berish bermasligidan qat'iy nazar har doim bajariladi.

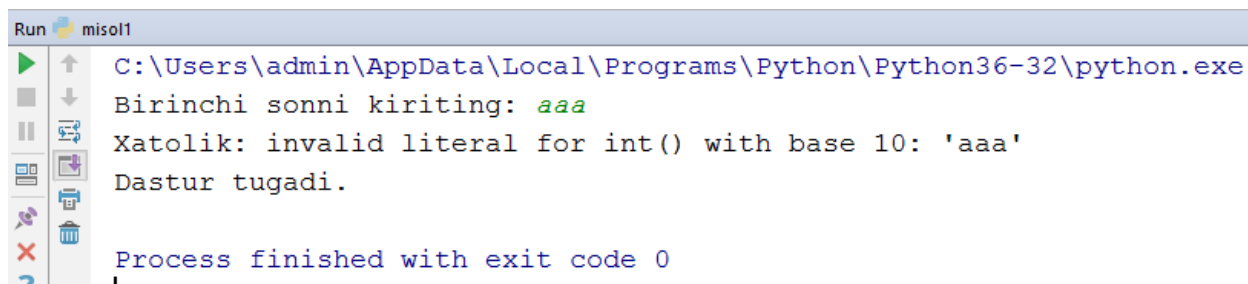
```
1 try:
2     son = int(input("Birinchi sonni kiriting: "))
3 except ValueError:
4     print("O`girish muaffaqiyatsiz amalga oshirildi.")
5 finally:
6     print("Try bloki ishi tugadi")
7 print("Dastur tugadi.")
```

Odatda *finally* blokida dasturda foydalanilgan resurslarni bo'shatish, masalan, faylni yopish kabi qodlar yoziladi.

**Istisno to'g'risida ma'lumot olish.** *as* operatori yordamida *except* blokida foydalanilishi mumkin bo'lgan istisno to'g'risidagi barcha ma'lumotlarni o'zgaruvchiga uzatish mumkin:

```
1 try:
2     son = int(input("Sonni kiriting: "))
3 except ValueError as e:
4     print("Xatolik:", e)
56 print("Dastur tugadi.")
```

Konsoldan noto'g'ri qiymat kiritilgandagi holati quyidagicha:



```
Run misol1
C:\Users\admin\AppData\Local\Programs\Python\Python36-32\python.exe
Birinchi sonni kiriting: aaa
Xatolik: invalid literal for int() with base 10: 'aaa'
Dastur tugadi.
Process finished with exit code 0
```

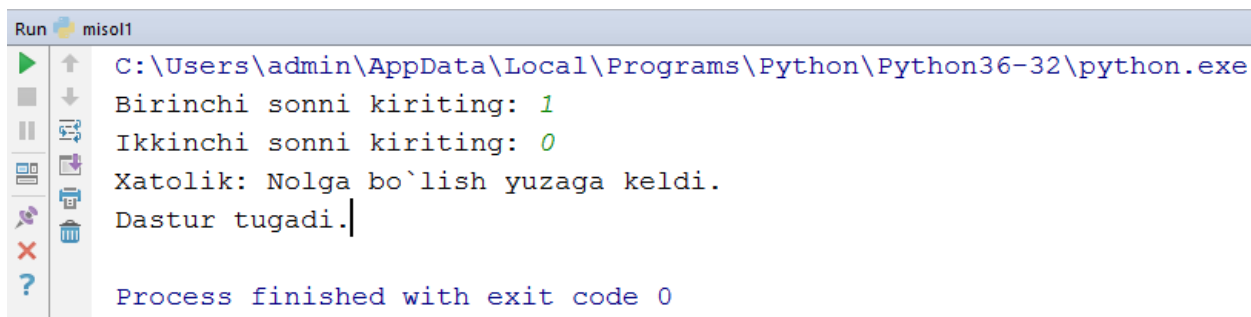
**Rasm №9.** Konsol ekranida qiymat noto'g'ri kiritilgandagi holat.

**Istisno holatini yuzaga keltirish.** Ba'zida istisno holatlarini foydalanuvchining o'zi hosil qilishga to'g'ri keladi. Buning uchun *raise* operatoridan foydalaniladi:

```
1 try:
2     son1 = int(input("Birinchi sonni kiriting: "))
3     son2 = int(input("Ikkinchi sonni kiriting: "))
4     if son2 == 0:
5         raise Exception("Nolga bo'lish yuzaga keldi.")
6     print("Bo'lish natijasi:", son1 / son2)
7 except ValueError:
8     print("O'girish muvaffaqiyatsiz amalga oshirildi.")
9 except Exception as e:
10    print("Xatolik:", e)
11 print("Dastur tugadi.")
```

Bu erda ikkinchi kiritiladigan son nolga teng bo'lsa, istisno hosil qilinadi. Istisno hosil qilinganda konsol ekraniga chiqarilishi kerak bo'lgan ma'lumotlar parametr sifatida beriladi. Yuqoridagi dasturda bu *Exception("Nolga bo'lish yuzaga*

*keldi.*") tarzda yozilgan. Quyida istisno holati ro'y berganda konsol ekraniga chiqariladigan ma'lumotlarga misol sifatida keltirilgan:



```
Run misol1
C:\Users\admin\AppData\Local\Programs\Python\Python36-32\python.exe
Birinci sonni kiriting: 1
Ikkinchi sonni kiriting: 0
Xatolik: Nolga bo'lish yuzaga keldi.
Dastur tugadi.
Process finished with exit code 0
```

**Rasm №10.** Foydalanuvchi tomonidan hosil qilingan istisnoning ro'y bergandagi holati.



## II. Pythonda ro'yxatlar, lug'atlar, kortejlar va to'plamlar

### 2.1. Ro'yxatlar

Ro'yxat (*list*) bu elementlar to'plami yoki ketma-ketligini saqlash uchun mo'ljallangan berilganlar turini ifodalaydi. Ro'yxatlarni hosil qilish uchun kvadrat qavs([]) ichida uning barcha elementlari vergul bilan ajratilgan holda keltiriladi. Ko'pincha, boshqa tillarda shunga o'xshash berilganlar turini massiv deb ataladi. Masalan quyida sonlar ro'yxatini aniqlaymiz:

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

Ro'yxatlarni hosil qilish uchun *list()* konstruktoridan ham foydalaniladi:

```
sonlar1 = [ ]
```

```
sonlar2 = list()
```

Yuqoridagi ikkita ro'yxat o'xshash bo'lib, ular bo'sh ro'yxatni aniqlayapti.

*list()* konstruktori ro'yxat hosil qilish uchun parametr sifatida boshqa ro'yxatni ham qabul qilishi mumkin:

```
1 sonlar1 = [1, 2, 3, 4, 5, 6, 7, 8]
2 sonlar2 = list(sonlar1)
```

Ro'yxat elementiga murojaatni amalga oshirish uchun uning ro'yxatdagi tartib raqamini ifodalovchi indeksi orqali amalga oshiriladi. Indekslar noldan boshlanadi. Ya'ni ikkinchi elementning indeksi 1 ga teng bo'ladi. Ro'yxat elementlariga oxiridan (teskari tartibda) murojaatni amalga oshirish uchun manfiy indekslardan foydalaniladi. Ro'yxatning eng oxirgi elementiga murojaat uchun -1, oxiridan oldingi element uchun -2 indeksleri mos keladi va h.k.. Ro'yxatdagi elementning qiymatini o'zgartirish uchun shu element indeksi orqali unga yangi qiymat yuklash orqali amalga oshiriladi:

```
1 sonlar = [1, 2, 3, 4, 5, 6, 7, 8]
2
3 print(sonlar[0])      #1
4 print(sonlar[1])      #2
5 print(sonlar[-1])     #8
6 print(sonlar[-2])     #7
7
```

```
8 sonlar[0] = 100
9 print(sonlar[0]) #100
```

**Ro'yxatni butun songa ko'paytirish.** Pythonda Ro'yxatni butun songa ko'paytirish amali ham kiritilgan bo'lib, odatda bunday amallar ro'yxatdagi barcha elementlarga ayni bir xil qiymatlarni yuklash uchun qo'llaniladi. Masalan, 6 ta elementining barcha qiymatlari 0 ga teng bo'lgan ro'yxatni aniqlash uchun quyidagicha kod yoziladi:

```
1 son = [0] * 6
2 print(son) # [0, 0, 0, 0, 0, 0]
```

Bundan tashqari, sonlar ketma-ketligining ro'yxatini *range()* funksiyasidan foydalanib hosil qilish ham mumkin bo'lib, u uchta shaklga ega:

- *range(end)* – 0 dan *end* gacha (*end* kirmaydi) bo'lgan sonlar ketma-ketligi hosil qilinadi.
- *range(start, end)* – *start* dan *end* gacha (*end* kirmaydi) bo'lgan sonlar ketma-ketligi hosil qilinadi.
- *range(start, end, step)* – *start* dan *end* gacha (*end* kirmaydi) bo'lgan sonlar ketma-ketligi *step* qadam bilan hosil qilinadi.

```
1 nums = list(range(6))
2 print(nums) #[0, 1, 2, 3, 4, 5]
3
4 nums = list(range(3,6))
5 print(nums) #[3, 4, 5]
6
7 nums = list(range(2,6,2))
8 print(nums) #[1, 3, 5]
```

Quyidagi ikkita ifoda orqali ayni bir xil bo'lgan ro'yxatlar aniqlangan bo'lib, ikkinchi ro'yxatni hosil qilishda *range* funksiyasidan foydalanish orqali kod hajmi bir muncha qisqartirilgan:

```
1 nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2 nums = list(range(10))
```

Ro'yxatlar bir turdagi ob'ektlardan tashkil topishi shart emas. Undagi har bir element turli xil toifaga ega bo'lishi mumkin:

```
1 objs = [1.2, True, 100, 'Satr', object]
2 print(objs) #[1.2, True, 100, 'Satr', <class 'object'>]
```

**Elementlarni birma-bir ko'rib chiqish.** Ro'yxat elementlarini birma-bir ko'rib chiqish uchun *for* yoki *while* takrorlash operatorlaridan foydalanish mumkin.

*for* orqali birma-bir ko'rib chiqish:

```
1 poytaxtlar = ['London', 'Parij', 'Moskva', 'Tashkent']
2 for poytaxt in poytaxtlar:
3     print(poytaxt)
```

*while* orqali birma-bir ko'rib chiqish:

```
1 poytaxtlar = ['London', 'Parij', 'Moskva', 'Tashkent']
2 i = 0
3 while i < len(poytaxtlar):
4     print(poytaxtlar[i])
5     i += 1
```

Bu erda *while* sikli orqali birma-bir ko'rib chiqishda *len* funksiyasidan foydalanilgan bo'lib, uning yordamida ro'yxat uzunligi olinadi. Bundan tashqari sanagich vazifasini bajaruvchi *i* o'zgaruvchisi bilan ro'yxatdagi barcha elementlarga murojaat amalga oshirilgan.

**Ro'yxatlarni taqqoslash.** Ikkita ro'yxat bir xil elementlardan tashkil topgan bo'lsa, bunday ro'yxatlar teng hisoblanadi.

```
1 nums1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2 nums2 = list(range(10))
3 if nums1 == nums2:
4     print("Bu ro'yxatlar aynan teng.")
5 else:
6     print("Bu ro'yxatlar teng emas.")
```

Bu holatda ikkita ro'yxat teng hisoblanadi.

## Ro'yxatlar bilan ishlashda qo'llaniladigan funksiyalar va metodlar.

Ro'yxatlar bilan ishlashda qo'llaniladigan bir nechta metodlar mavjud bo'lib, ularning eng asosiylari quyida keltirilgan:

- ***append(item)***: ro'yxat oxiriga *item* elementini qo'shish;
- ***insert(index, item)***: Ro'yxatga *index* indeksi bo'yicha *item* elementini qo'shish;
- ***remove(item)***: ro'yxatdan *item* elementini o'chirish. Ushbu metod ro'yxatdagi birinchi uchragan *item* elementini o'chiradi. Agar bunday element ro'yxatda mavjud bo'lmasa *ValueError* istisno holati ro'y beradi;
- ***clear()***: ro'yxatni tozalash, ya'ni ro'yxatdagi barcha elementlarni o'chirish;
- ***index(item)***: ro'yxatdagi *item* elementining joylashgan indeksini qiymat sifatida qaytaradi. Agar bunday element ro'yxatda mavjud bo'lmasa, u holda *ValueError* istisno holati ro'y beradi;
- ***pop([index])***: ro'yxatdan *index* indeksi bo'yicha elementni o'chiradi va qiymat sifatida qaytaradi. Agar indeks ko'rsatilmasa ro'yxatdan oxirgi elementni o'chiradi va qiymat sifatida qaytaradi. Bundan tashqari agar ro'yxat bo'sh bo'lsa, u holda *ValueError* istisno holati ro'y beradi;
- ***count(item)***: ro'yxatdagi *item* elementlar sonini qiymat sifatida qaytaradi;
- ***sort([key])***: ro'yxat elementlarini tartiblaydi. Kelishuv bo'yicha, ro'yxat elementlarini o'sish bo'yicha tartiblaydi. *key* parametri orqali tartiblash funksiyasini (mezonini) berish mumkin;
- ***reverse()***: ro'yxat elementlarini teskari tartibda joylashtirish uchun qo'llaniladi.

Bundan tashqari, Python ro'yxatlar bilan ishlashda qo'llaniladigan bir nechta standart funksiyalarni ham o'z ichiga qamrab olgan:

- ***len(list)***: ro'yxat uzunligini (elementlari sonini) qiymat sifatida qaytaradi;
- ***sorted(list,[key])***: tartiblangan ro'yxatni qiymat sifatida qaytaradi;
- ***min(list)***: ro'yxatdagi eng kichik elementni qaytaradi;
- ***max(list)***: ro'yxatdagi eng katta elementni qaytaradi.

**Ro'yxatga elementlarni qo'shish va o'chirish.** Ro'yxatga elementlarni qo'shish uchun *append*, *insert* metodlari, elementlarni o'chirishda esa *remove*, *clear*, *pop* metodlaridan foydalaniladi, masalan:

```
1 users = ["Tom", "Bob"]
2
3 # ro'yxat oxiriga element qo'shish
4 users.append("Alice") # ["Tom", "Bob", "Alice"]
5 # ro'yxatdagi ikkinchi o'ringa element qo'shish
6 users.insert(1, "Bill") # ["Tom", "Bill", "Bob", "Alice"]
7
8 # element indeksini olish
9 i = users.index("Tom")
10 # indeks bo'yicha elementni ro'yxatdan o'chirish
11 removed_item = users.pop(i) # ["Bill", "Bob", "Alice"]
12
13 # ro'yxatdagi oxirgi qiymatni olish
14 last_user = users[-1]
15 # ro'yxatdan oxirgi elementni o'chirish
16 users.remove(last_user) # ["Bill", "Bob"]
17
18 print(users)
19
20 # ro'yxatning barcha elementlarini o'chirish
21 users.clear()
```

**Elementni ro'yxatda mavjudligini tekshirish.** Odatda, *index*, *remove* kabi metodlardan foydalanilganda, ularning parametrlarida ko'rsatilgan element ro'yxatda mavjud bo'lmasa, istisno holati yuzaga keladi. Bunday holatlarning oldini olish uchun, oldin ushbu elementning ro'yxatda mavjudligini tekshirish kerak bo'ladi. Buning uchun *in* kalit so'zidan foydalaniladi:

```
1 companies = ["Microsoft", "Google", "Oracle", "Apple"]
2 item = "Oracle" # o'chirish kerak bo'lgan element
3 if item in companies:
4     companies.remove(item)
```

```
5
6 print(companies) # ['Microsoft', 'Google', 'Apple']
```

Yuqoridagi holatda, agar ro'yxatda *item* elementi mavjud bo'lsa, *item in companies* ifodasi *True* qiymat qaytaradi. Shuning uchun *if* shart ifodasi agar shu element mavjud bo'lsagina navbatdagi amallarni bajaradi, natijada istisno holatining oldi olinadi.

**count() metodi.** Ushbu metod orqali ro'yxatda biror element nechi marta qatnashganligi topiladi. Masalan:

```
1 nomlar = ["Anvar", "Sobir", "Sobir", "Qosim"]
2 nom_soni = nomlar.count("Sobir")
3 print(nom_soni) #2
```

**Tartiblash.** Ro'yxatdagi elementlarni o'sib borish bo'yicha tartiblash uchun *sort()* metodi qo'llaniladi:

```
1 nomlar = ["Anvar", "Sobir", "Sobir", "Qosim"]
2 nomlar.sort()
3 print(nomlar) # ['Anvar', 'Qosim', 'Sobir', 'Sobir']
```

Agar teskari tartibda tartiblash kerak bo'lsa, *sort()* metodidan keyin *reverse()* metodidan foydalanish kifoya qiladi:

```
1 nomlar = ["Anvar", "Sobir", "Sobir", "Qosim"]
2 nomlar.sort()
3 nomlar.reverse()
4 #nomlar.sort(reverse=True) deb ham ishlatish mumkin
5 print(nomlar) # ['Sobir', 'Sobir', 'Qosim', 'Anvar']
```

Shuni alohida ta'kidlash lozimki, tartiblashda ob'ektlar taqqoslanadi. Ob'ekt sifatida son kelsa muammo yo'q, o'sish yoki kamayib borish bo'yicha tartiblanadi. Lekin ob'ekt sifatida satr kelsa, u holda ular mos belgilari bo'yicha taqqoslanadi. Taqqoslashda belgilarning ASCII kodlari taqqoslanadi. Shuning uchun har qanday katta registrga ega lotin yozuvidagi belgi kichik registrdagi lotin yozuvidagi belgidan kichik bo'ladi, masalan: "Abc" < "abc":

```
1 nomlar = ["anvar", "Sobir", "sobir", "Qosim", "tolib"]
2 nomlar.sort()
```

```
3 print(nomlar) # ['Qosim', 'Sobir', 'anvar', 'sobir', 'tolib']
```

Tartiblashda `sort()` metodi o'rniga standart `sorted()` funksiyasidan ham foydalanish mumkin bo'lib, uning quyidagi ikkita shakli mavjud:

- `sorted(list)` – ro'yxat elementlarini tartiblash uchun;
- `sorted(list, key)` – ro'yxat elementlarini `key` mezon (funksiya) asosida tartiblash uchun ishlatiladi.

```
1 nomlar = ["anvar", "Sobir", "sobir", "Qosim", "tolib"]
2 sorted_nomlar = sorted(nomlar, key = str.lower)
3 print(sorted_nomlar) # ['anvar', 'Qosim', 'Sobir', 'sobir',
4                       # 'tolib']
```

`sorted()` funksiyasidan foydalanilganda qiymat sifatida ushbu funksiya tartiblangan elementlardan tashkil topgan yangi ro'yxatni qaytaradi, tartiblanayotgan ro'yxat esa o'zgarishsiz qoladi.

**Minimal va maksimal qiymatlar.** Pythonda *max*, *min* deb nomlanuvchi mos ravishda ro'yxatdan eng maksimal va eng minimal qiymatlarni topish uchun mo'ljallangan standart funksiyalari mavjud.

```
1 sonlar = [12, 45, 23, -35, 2]
2 print(min(sonlar)) # -35
3 print(max(sonlar)) # 45
```

**Ro'yxatlarni ko'chirish.** Ro'yxat – o'zgaruvchan (mutable) turga mansub bo'lib, agar ikkita o'zgaruvchi ayni bir ro'yxatga murojaat qilayotgan bo'lsa, u holda birining o'zgarishi ikkinchisiga ham ta'sir qiladi:

```
1 vil1 = ["Toshkent", "Xorazm", ]
2 vil2 = vil1
3 vil2.append('Buxoro')
4 print(vil1) # ['Toshkent', 'Xorazm', 'Buxoro']
5 print(vil2) # ['Toshkent', 'Xorazm', 'Buxoro']
```

Bu misolda har ikkala `vil1` va `vil2` o'zgaruvchilar yordamida ayni bir ro'yxatga murojaat bo'lgan. Shuning uchun `vil2` o'zgaruvchisi orqali ro'yxatga yangi element qo'shilganda mos ravishda `vil1` ham o'zgargan. Bu holat yuzaki ko'chirish (*shallow copy*) deyiladi. Albatta ro'yxatlarni ko'chirganda ikkita alohida ro'yxat

hosil qiladigan tarzda ham ko'chirish (*deep copy*) mumkin. Buning uchun ichki *copy* modulda *deepcopy()* metodidan foydalaniladi:

```
1 import copy
2 vil1 = ["Toshkent", "Xorazm", ]
3 vil2 = copy.deepcopy(vil1)
4 vil2.append('Buxoro')
5 # ikkita alohida ro'yxat hosil bo'ldi
6 print(vil1) # ['Toshkent', 'Xorazm']
7 print(vil2) # ['Toshkent', 'Xorazm', 'Buxoro']
```

**Ro'yxat qismini ko'chirish.** Agar butun bir ro'yxatni emas, balki uning bir qismini ko'chirish zarur bo'lsa, u holda maxsus sintaksisdan foydalaniladi. Ushbu sintaksis quyidagi shakllarda qo'llaniladi:

*list(:end)* - *end* parametri orqali ro'yxatning qaysi elementigacha ko'chirish kerakligini bildiruvchi indeks nomeri beriladi;

*list(start:end)* – *start*, *end* parametrlar orqali ro'yxatning *start* dan *end* gacha bo'lgan elementlarini ko'chirish kerakligini bildiruvchi indeks nomerlari beriladi;

*list(start:end:step)* – *start*, *end*, *step* parametrlar orqali ro'yxatning *start* dan *end* gacha bo'lgan elementlarini *step* qadam bilan ko'chirish kerakligini bildiruvchi qiymatlar beriladi. *step* parametrning kelishuv bo'yicha qiymati 1 ga tengdir.

```
1 vil = ["Toshkent", "Xorazm", 'Buxoro', 'Navoi', 'Jizzax']
2 vil1 = vil[:2]
3 print(vil1) # ['Toshkent', 'Xorazm']
4 vil2 = vil[2:4]
5 print(vil2) # ['Buxoro', 'Navoi']
6 vil3 = vil[1:5:2]
7 print(vil3) # ['Xorazm', 'Navoi']
```

**Ro'yxatlarni birlashtirish.** Ro'yxatlarni birlashtirish uchun (+) amali qo'llaniladi:

```
1 vil1 = ["Toshkent", "Xorazm", 'Buxoro']
2 vil2 = ['Navoi', 'Jizzax']
3 vil = vil1 + vil2
4 print(vil)
```



**Ichma – ich joylashgan ro'yxatlar.** Ro'yxat elementlari son, satr kabi oddiy turdagi qiymatlariga bo'lib qolmay, balki ro'yxatni ham ifodalashi mumkin. Odatda bunday ro'yxatlardan jadvallar bilan ishlashda ko'p foydalaniladi. Ushbu holatda tashqi ro'yxatning har bir elementi jadvaldagi bitta qatorni ifodalovchi ro'yxatdan iborat bo'ladi:

```
1 ishchilar = [  
2     ["Tolib", 33],  
3     ["Akmal", 30],  
4     ["Botir", 27] ]  
5  
6 print(ishchilar[0]) # ["Tolib", 33]  
7 print(ishchilar[0][0]) # Tolib  
8 print(ishchilar[0][1]) # 33
```

Bu erda ichki ro'yxatni elementiga murojaat qilish uchun `[][]` indekslar juftligidan foydalanilgan. Xususan, `ishchilar[0][1]` – birinchi ichki ro'yxatning ikkinchi elementiga murojaat bo'lgan.

Ro'yxatga elementlarni qo'shish, o'chirish, o'zgartirish kabi jarayonlar oddiy ro'yxatlardagi kabi amalga oshiriladi:

```
1 ishchilar = [  
2     ["Tolib", 33],  
3     ["Akmal", 30],  
4     ["Botir", 27] ]  
5  
6 print(ishchilar[0]) # ["Tolib", 33]  
7 print(ishchilar[0][0]) # Tolib  
8 print(ishchilar[0][1]) # 33  
9 # ro`yxat yaratish  
10 ishchi = list()  
11 ishchi.append("Rustam")  
12 ishchi.append(21)  
13 # Tashqi ro`yxatga yaratilgan ro`yxatni qo`shish  
14 ishchilar.append(ishchi)  
15 print(ishchilar[-1]) # ["Rustam", 21]
```

```

16 # Tashqi ro'yxatning oxirgi elementiga element qo'shish
17 ishchilar[-1].append("+998909737066")
18 print(ishchilar[-1]) # ['Rustam', 21, '+998909737066']
19 # tashqi ro'yxatning oxirgi elementining oxirgi elementini
20 # o'chirish
21 ishchilar[-1].pop()
22 print(ishchilar[-1]) # ["Rustam", 21]
23 # tashqi ro'yxatning oxirgi elementini o'chirish
24 ishchilar.pop(-1)
25 # birinchi elementni o'zgartirish
26 ishchilar[0] = ["Sobir", 18]
27 print(ishchilar) # [['Sam', 18], ['Akmal', 30], ['Botir', 27]]

```

Murakkab ro'yxatlar elementlariga murojaat ichma – ich joylashgan takrorlash operatorlari orqali amalga oshirilishi mumkin:

```

1 ishchilar = [
2     ["Tolib", 33],
3     ["Akmal", 30],
4     ["Botir", 27] ]
5
6 for ishchi in ishchilar:
7     for i in ishchi:
8         print(i, end="|")
9 # Konsolga quyidagi ma'lumotlar chiqariladi
10 # Tolib|33|Akmal|30|Botir|27|

```

## 2.2. Kortejlar

Kortej (tuple) elementlar ketma-ketligini ifodalovchi, ko'p jihatlari bo'yicha ro'yxatga o'xshaydigan, lekin undan farqli ravishda o'zgarmaydigan (immutable) tur hisoblanadi. Shuning uchun kortejga yangi element qo'shish, undan elementni o'chirish yoki o'zgartirish kiritishga ruxsat etilmaydi.

Kortejni hosil qilish uchun oddiy qavs “(, )” dan foydalanib, unda elementlar vergul bilan ajratilgan tarzda joylashtiriladi:

```

1 user = ("Akmal", 21)
2 print(user) # ('Akmal', 21)

```

Bundan tashqari kortejni aniqlash uchun elementlar ketma – ketligi vergul bilan ajratilgan holda oddiy qavslarsiz ham amalga oshirsa bo’ladi:

```
1 user = "Akmal", 21
2 print(user) # ('Akmal', 21)
```

Shuni alohida ta’kidlash kerakki, kortej faqat bitta elementdan tashkil topsa ham vergul ishlatiladi:

```
1 user = "Akmal",
2 print(user) # ('Akmal',)
```

Ro’yxat asosida kortej hosil qilish uchun maxsus *tuple* funksiyasidan foydalaniladi va uning argumentiga qiymat sifatida ro’yxat beriladi:

```
1 user_list = ["Yusuf", 'Tolib', 'Rustam']
2 user_tuple = tuple(user_list)
3 print(user_tuple) # ('Yusuf', 'Tolib', 'Rustam')
```

Kortej elementlariga murojaat xuddi ro’yxatlardagi kabi indeksleri orqali amalga oshiriladi. Indeksler kortej boshiga nisbatan 0 dan, kortej oxiriga nisbatan -1 dan boshlanadi:

```
1 users = ("Yusuf", 'Qodir', 'Erkin', 'Oybek')
2 print(users[0]) # Yusuf
3 print(users[2]) # Erkin
4 print(users[-1]) # Oybek
5 print(users[1:3]) # ('Qodir', 'Erkin')
```

Kortej o’zgaraydigan tur bo’lganligi sababli uning elementini o’zgartirib bo’lmaydi. Masalan: `users[0] = "Rahim"` kabi kod yozilsa, Python interpretatori xatolik to’g’risida xabar chiqaradi.

Kortej elementlarini, ularning soniga mos o’zgaruvchilarga birdaniga yuklash ham mumkin:

```
1 users = ("Yusuf", 'Qodir', 'Erkin',)
2 a, b, c = users
3 print(a) # Yusuf
4 print(b) # Qodir
5 print(c) # Erkin
```

Kortejlarning bunday xususiyati ayniqsa funksiyalar bilan ishlashda juda foydali hisoblanadi. Masalan, funksiya natija sifatida birdan ortiq qiymat qaytarsa, aslida kortej turidagi bitta qiymat qaytarayotgan bo'ladi. Funksiyadan qaytgan qiymatlarni bir nechta o'zgaruvchilarga yuklash orqali, alohida bir - biriga bog'liq bo'lmagan qiymatlarga ega bo'linadi:

```
1 def get_user():
2     name = "Yusuf"
3     age = 33
4     is_married = True
5     return name, age, is_married
6
7 user = get_user()
8 print(user[0]) # Yusuf
9 print(user[1]) # 33
10 print(user[2]) # True
11
12 # yoki alohida o'zgaruvchilarga yuklanadi
13 name, age, ismarried = get_user()
14 print(name) # Yusuf
15 print(age) # 33
16 print(ismarried) # True
```

**len()** funksiyasi orqali kortejning uzunligi (elementlari soni) topiladi:

```
1 user = ("Erkin", 30, True)
2 print(len(user)) # 3
```

Kortej elementlariga *for* va *while* takrorlash operatorlari orqali murojaat quyidagicha amalga oshiriladi:

*for* orqali

```
1 user = ("Erkin", 30, True)
2
3 for u in user:
4     print(u)
```

*while* orqali

```
1 user = ("Erkin", 30, True)
```

```

2 i=0
3 while i < len(user):
4     print(user[i])
5     i +=1

```

Kortejda biror elementning mavjudligini xuddi ro'yxatlardagi kabi *in* operatoridan foydalanib amalga oshiriladi:

```

1 user = ("Erkin", 30, True)
2
3 if 'Erkin' in user:
4     print("Foydalanuvchining ismi Erkin")

```

**Murakkab kortejlar.** Kortej o'z ichiga boshqa kortejlarni element sifatida qamrab olsa, bunday kortejlar murakkab kortejlar hisoblanadi:

```

1 davlatlar = (
2     ("O`zbekiston", 33.8,
3     ("Toshkent", 2.65),
4     ("Samarqand", 1.1),
5     ("Urganch", 0.223))
6     ),
7     ("Qozog`iston", 17.5,
8     ("Nur Sultan", 1.1),
9     ("Olmaota", 2.3))
10    ))
11 for davlat in davlatlar:
12     nomi, aholi_soni, shaharlar = davlat
13     print(nomi, '-', aholi_soni)
14     print("Katta shaharlari:")
15     for shahar in shaharlar:
16         print(shahar[0], '-', shahar[1])

```

Bu erda *davlatlar* korteji ikki elementdan tashkil topgan bo'lib, ular ham kortejlardir va davlat nomi, aholisi, katta shaharlari to'g'risidagi ma'lumotlarni ifodalaydi. O'z navbatida shaharlar ham yana kortejlardan tashkil topgan bo'lib, ularning har bir elementi shahar nomi va aholisini ifodalaydigan kortejdan tashkil

topgan. Xullas, jami ichma-ich joylashgan 4 ta kortejdan iborat berilganlarning murakkab strukturasi misol keltirilgan.

### 2.3. Lug'atlar

Python dasturlash tilida ro'yxatlar va kortejlar bilan bir qatorda lug'atlar (dictionary) deb nomlanuvchi berilganlarning ichki tuzilmasi mavjud. Lug'atlar ham xuddi ro'yxatlar kabi elementlar to'plamini saqlaydi. Lug'atdagi har bir element unikal kalitga ega bo'ladi va unga biror bir qiymat bog'lanadi.

Lug'at quyidagicha sistaksis bo'yicha aniqlanadi:

```
dictionary = { kalit1:qiymat1, kalit2:qiymat2, ....}
```

Quyida lug'atlarga misol keltirilgan:

```
1 users = {1: "Tom", 2: "Bob", 3: "Bill"}
2
3 elements = {"Au": "Oltin", "Fe": "Temir", "H": "Vodorod",
4 "O": "Kislorod"}
```

Bu erda *users* ro'yxatida kalit sifatida son, qiymat sifatida satr qo'llanilgan. *element* ro'yxatida esa qiymat sifatida ham kalit sifatida ham satr ishlatilgan.

Lekin kalitlar va qiymatlar bir turga mansub bo'lishi shart emas. Ular har xil turdagi qiymatlar bo'lishi mumkin:

```
1 objects = {1: "Tom", "2": True, 3: 100.6}
```

Bundan tashqari bo'sh lug'atlarni ham yaratish mumkin:

```
1 object1 = {}
2 # yoki
3 object2 = dict()
```

**Ro'yxatlar yordamida lug'at xosil qilish.** Lug'atlar tuzilmaviy jihatidan ro'yxatlarga o'xshamasada, lekin ba'zi bir maxsus ro'yxatlar asosida *dict()* funkuyasi orqali ro'yxatlar hosil qilish mumkin. Buning uchun ro'yxat o'z navbatida ro'yxatlar to'plamidan tashkil topgan bo'lishi kerak. Ichki ro'yxatlar ikkita elementlardan tashkil topishi shart bo'lib, mos ravishda birinchi element kalitga, ikkinchi element qiymatga akslantiriladi:

```
1 users_list = [{"909837022", "Tolib"},
```

```

2     ["909939343", "Bobur"],
3     ["903943493", "Alibek"] ]
4 users_dict = dict(users_list)
5 print(users_dict) # {'909837022': 'Tolib', '909939343':
67                          #'Bobur', '903943493': 'Alibek'}

```

Xuddi shu tarzda kortejlarni ham lug'atlarga aylantirish mumkin. Buning uchun ikki o'lchamli kortejning ichki kortejlari o'z navbatida ikkitadan elementdan tashkil topgan bo'lishi shart:

```

1 users_tuple = ( ("909837022", "Tolib"),
2                 ("909939343", "Bobur"),
3                 ("903943493", "Alibek") )
4 users_dict = dict(users_tuple)
5 print(users_dict) # {'909837022': 'Tolib', '909939343':
6                     # 'Bobur', '903943493': 'Alibek'}

```

**Lug'at elementini o'zgartirish.** Lug'at elementiga murojaat qilish uning kaliti yordamida amalga oshiriladi:

*dictionary[kalit]*

Masalan lug'at elementiga murojaat qilish va uni o'zgartirish quyidagicha amalga oshiriladi:

```

1 users = {
2     "Bir": "Tolib",
3     "Ikki": "Bobur",
4     "Uch": "Alisher" }
5 # Lug'atning "Bir" kalitli elementiga murojaat uchun
6 print(users["Bir"]) # Tolib
7 # Lug'atdagi "Uch" kalitli element qiymatini o'zgartiramiz
8 users["Uch"] = "Baxtiyor"
9 print(users["Uch"]) # Baxtiyor

```

Lug'at elementiga kaliti orqali qiymat berganda shunday kalit lug'atda mavjud bo'lmasa, u holda lug'atga yangi element qo'shiladi. Masalan, yuqoridagi misolda `users["To'rt"] = "Ibrohim"` tarzida yangi element qo'shishimiz mumkin, Chunki lug'atda "To'rt" kalitli element mavjud emas.

Lekin, lug'atda mavjud bo'lmagan kalit orqali uning elementiga murojaat qilinganda, Python interpretatori *KeyError* turidagi istisno xatoligi yuzaga kelganligi haqida xabar chiqaradi. Masalan, yuqoridagi misol uchun `user = users["Besh"]` kabi ishlatsak xatolik ro'y beradi. Bunday istisno xalotlarning oldini olish uchun Pythonda ***Kalit in Lug'at*** ifodasidan foydalaniladi. Ushbu ifoda agarda shunday kalitli element lug'atda mavjud bo'lsa *True* qiymat, aks holda *False* qiymat qaytaradi, masalan:

```
1 bahoDict = {"A": 5, "B": 4, "C": 3}
2 key = "D"
3 if key in bahoDict:
4     baho = bahoDict[key]
5     print(baho)
6 else:
7     print("Element topilmadi") # Javob: Element topilmadi
```

Shu bilan birga, lug'atning biror elementini olish uchun *get* metodidan ham foydalanish mumkin bo'lib u ikki xil shaklda qo'llaniladi:

- *get(key)* – lug'atning *key* kalitli elementni qaytaradi. Agar lug'atda *key* kalitli element mavjud bo'lmasa *None* qiymati qaytariladi.
- *get(key, default)* - lug'atning *key* kalitli elementni qaytaradi. Agar lug'atda *key* kalitli element mavjud bo'lmasa *default* qiymati qaytariladi.

Masalan:

```
1 bahoDict = {"A": 5, "B": 4, "C": 3}
2 key = "A"
3 baho = bahoDict.get(key)
4 print(baho) # 5
5 # yoki
6 key = "D"
7 baho = bahoDict.get(key, "Noma'lum qiymat")
8 print(baho) # Noma'lum qiymat
```

**Lug'atdan elementni o'chirish.** Lug'atdan kalit orqali elementni o'chirish uchun *del* operatoridan foydalaniladi:

```
1 bahoDict = {"A": 5, "B": 4, "C": 3, "D": 2}
```



```

2 print(bahoDict) # {'A': 5, 'B': 4, 'C': 3, 'D': 2}
3
4 del bahoDict["C"]
5 print(bahoDict) # {'A': 5, 'B': 4, 'D': 2}

```

Shuni alohida ta’kidlash lozimki, agar lug’atda bunday kalit mavjud bo’lmasa *KeyError* istisno xatoligi yuzaga keladi. Ushbu xatolikni oldini olish uchun dastlab bunday kalit lug’atda bor yoki yo’qligini tekshirish tavsiya qilinadi:

```

1 bahoDict = {"A": 5, "B": 4, "C": 3, "D": 2}
2 baho = "A"
3 if baho in bahoDict:
4     son = bahoDict[baho]
5     del bahoDict[baho]
6     print(son, "o'chirildi")
7 else:
8     print("Element topilmadi")
9 # Javob: 5 o'chirildi

```

O’chirishning boshqa bir usuli – *pop()* metodi orqali amalga oshiriladi. U ikki xil shaklda qo’llaniladi:

*pop(key)* – *key* kaliti bo’yicha elementni o’chiradi va qiymat sifatida o’chirilgan elementni qaytaradi. Agar berilgan kalit bo’yicha element topilmasa, *KeyError* istisno holati yuzaga keladi;

*pop(key, default)* – *key* kaliti bo’yicha elementni o’chiradi va qiymat sifatida o’chirilgan elementni qaytaradi. Agar berilgan kalit bo’yicha element topilmasa, *default* qiymati qaytariladi.

```

1 bahoDict = {"A": 5, "B": 4, "C": 3, "D": 2}
2 key = "A"
3 baho = bahoDict.pop(key)
4 print(baho) # 5
5 # ikkinchi marta yana shu kalit bo'yicha o'chirishga urinamiz
6 baho2 = bahoDict.pop(key, "Bunday baho mavjud emas!")
7 print(baho2) # Bunday baho mavjud emas!

```

Agar lug’atdagi barcha elementlarni o’chirish talab qilinsa, *clear()* metodidan foydalanish mumkin:

```

1 bahoDict = {"A": 5, "B": 4, "C": 3, "D": 2}
2 # Lug'atning barcha elementlarini ekranga chiqaramiz
3 print(bahoDict) # {'A': 5, 'B': 4, 'C': 3, 'D': 2}
4 bahoDict.clear()
5 # clear metodini qo'llagandan so'ng yana
6 # lug'atning barcha elementlarini ekranga chiqaramiz
7 print(bahoDict) # {}

```

**Lug'atlarni ko'chirish va birlashtirish.** Lug'atlarni ko'chirish uchun *copy()* metodidan foydalanilib, qiymat sifatida ushbu lug'atning elementlaridan tashkil topgan boshqa lug'at hosil qilinadi, masalan:

```

1 l = {"ismi": "Sardor", "yoshi": 34}
2 l2 = l.copy()
3 print(l) # {'ismi': 'Sardor', 'yoshi': 34}
4 print(l2) # {'ismi': 'Sardor', 'yoshi': 34}

```

Lug'atlarni birlashtirish uchun *update()* metodidan foydalaniladi:

```

1 l1 = {"ismi": "Sardor", "yoshi": 34}
2 l2 = {"kursi": 1, "yo`nalishi": "IAT"}
3 l1.update(l2)
4 print(l1) # {'ismi': 'Sardor', 'yoshi': 34, 'kursi': 1,
5           # 'yo`nalishi': 'IAT'}
6 print(l2) # {'ismi': 'Sardor', 'yoshi': 34}

```

Yuqoridagi holatda *l2* tarkibi o'zgarishsiz qoladi va *l1* lug'at tarkibiga boshqa lug'at elementlari qo'shiladi.

**Lug'at elementlariga murojaat.** Lug'at elementlariga murojaat uning kaliti orqali amalga oshiriladi. Ayniqsa *for* operatori orqali lug'at elementlarini uning kaliti orqali olish juda qulay hisoblanadi:

```

1 talabalar = {
2     "+99890123": "Tolmas",
3     "+99890124": "Bobur",
4     "+99890125": "Alisher" }
5 for tal in talabalar:
6     print(tal, " - ", talabalar[tal])

```

Javobga quyidagi natija chiqariladi:

```
+99890123 - Tolmas
+99890124 - Bobur
+99890125 - Alisher
```

bu erda *for* operatoridagi *t* o'zgaruvchiga ketma – ket lug'at kaliti qiymatlari yuklanadi (chapdan o'nga qarab) va shu kalit orqali lug'at elementiga murojaat amalga oshiriladi.

Lug'at elementlariga murojaat qilishning yana bir usuli *items()* metodini qo'llash orqali amalga oshiriladi. Yuqoridagi dastur kodi *items()* metodi orqali quyidagicha yoziladi va ayni bir xil natijaga erishiladi:

```
1 talabalar = {
2     "+99890123": "Tolmas",
3     "+99890124": "Bobur",
4     "+99890125": "Alisher" }
5 for nomer, ism in talabalar.items():
6     print(nomer, " - ", ism)
```

*items()* metodi qiymat sifatida kortejlar to'plamini qaytaradi. Har bir kortej elementi kalit (*nomer*) va qiymatlar (*ism*) juftligidan tashkil topadi.

Lug'atdan faqat kalitlarini olish uchun *keys()* va faqat qiymatlarini olish uchun *values()* metodlaridan foydalaniladi, masalan:

```
1 talabalar = {
2     "+99890123": "Tolmas",
3     "+99890124": "Bobur",
4     "+99890125": "Alisher" }
5 # lug'atning kalitlariga murojaat
6 print("Kalitlar:")
7 for kalit in talabalar.keys():
8     print(kalit, end='; ')
9 # lug'atning qiymatlariga murojaat
10 print("\n Qiymatlar:")
11 for qiymat in talabalar.values():
12     print(qiymat, end='; ')
```

Ushbu dastur ishga tushirilganda quyidagi javob ekranga chiqariladi:

*Kalitlar:*

+99890123; +99890124; +99890125;

*Qiymatlar:*

*Tolmas; Bobur; Alisher;*

**Kompleks (murakkab) lug'atlar.** Lug'atlar faqatgina *int, str, float, bool* kabi oddiy turlarga oid berilganlardangina emas, balki *list, tuple, set, dict* kabi murakkab tuzilmaviy berilganlardan ham tashkil topishi mumkin:

```
1 loginData = {
2     "Zafar":
3         {
4             "email": "zafar@nuu.uz",
5             "tel": "+99890933",
6             "manzil": "Univer ko`chasi 4"
7         },
8     "Rustam":
9         {
10            "email": "rustam@nuu.uz",
11            "tel": "+998902222",
12            "manzil": "Dekanat ko`chasi 105"
13        } }
14 print(loginData)
```

Yuqorida keltirilgan misolda *loginData* lug'ati (tashqi lug'at) o'z navbatida boshqa lug'atlar (ichki lug'atlar) dan tashkil topgan. Bunday hollarda ichki lug'atni elementlariga quyidagi tarzda murojaat qilinadi:

```
1 lData = loginData["Zafar"]["tel"]
2 print(lData)
```

lug'atda mavjud bo'lmagan kalit orqali uning elementiga murojaat amalga oshirilganda Python interpretatori *KeyError* turidagi istisno xatoligini yuzaga keltiradi:

```
1 lData = loginData["Zafar"]["telegram"] #KeyError
```

bu erda “telegram” kalit so’zi mavjud emas. Shuning uchun istisno xatoligi ro’y berdi. Bunday xatoliklarning oldini olish uchun dastlab kalitning lug’atda bor yoki yo’qligini tekshirish tavsiya qilinadi:

```
1 key = "telegram"
2 if key in loginData["Zafar"]:
3     print(loginData["Tom"]["telegram"])
4 else:
5     print("telegram kaliti topilmadi")
```

Umuman olganda, murakkab lug’atlar (ichma – ich joylashgan) ustuda amallar oddiy lug’atlardagi kabi amalga oshiriladi.

## 2.4. To’plamlar

To’plamlar elementlar majmuini ifodalashning yana bir ko’rinishi hisoblanadi. To’plamlarni aniqlash uchun figurali qavs (‘{’, ‘}’) dan foydalanilib, elementlar unda ketma-ket sanaladi:

```
1 talabalar = {"Bobur", "Zafar", "Alisher"}
2 print(talabalar) # {'Bobur', 'Zafar', 'Alisher'}
```

To’plamni tashkil qiluvchi elementlar qiymatlari unikal bo’lishi kerak, agar elementlar qiymatlari ayni bir xil bo’lsa, ya’ni bir xil element takrorlansa, u holda barcha takrorlanuvchi qiymatlar bitta deb hisoblanadi:

```
1 son = {"1", "1", "2", "2", "2"}
2 print(son) # {'2', '1'}
```

Bu erda to’plam elementlari ikkita “1” va uchta “2” qiymatlar orqali hosil qilingan. Lekin ekranga to’plam elementlari chop qilinganda to’plam faqatgina ikki elementdan tashkil topganligini ko’rish mumkin.

To’plamni yaratish uchun `set()` funksiyasidan ham foydalanish mumkin. Ushbu funksiyadan foydalanib to’plam yaratilganda parametriga qiymat sifatida ro’yxat yoki kortej ham berilishi mumkin:

```
1 tubSonlar = [2, 3, 5, 7, 11]
2 tubSonlarTuplami = set(tubSonlar)
3 print(tubSonlarTuplami) # {2, 3, 5, 7, 11}
```

Ayniqsa `set()` funksiyasi bo'sh to'plam hosil qilish uchun juda qulay hisobladi:

```
1 son = set()
2 print(son) # set()
```

To'plam uzunligi (to'plam elementlari soni) ni toppish uchun `len()` funksiyasidan foydalaniladi:

```
1 son = {3, 4, 5, 6}
2 print(len(son)) # 4
```

**To'plamga element qo'shish.** To'plamga element qo'shish uchun `add()` metodidan foydalaniladi:

```
1 son = set()
2 son.add(2)
3 son.add(4)
4 print(son) # {2, 4}
```

**To'plamdan elementni o'chirish.** To'plamdan elementni o'chirish uchun `remove()` metodi qo'llanilib, uning argumentiga o'chirilishi kerak bo'lgan element beriladi. Agar o'chirilishi kerak bo'lgan element to'plamda mavjud bo'lmasa, u holda `KeyError` istisno xatoligi ro'y beradi. Shuning uchun to'plamdan elementni o'chirishdan oldin `in` operatori orqali shu elementning lug'atda mavjud yoki yo'qligini tekshirish tavsiya qilinadi:

```
1 ismlar = {"Anvar", "Abbos", "Abror"}
2 ism = "Abror"
3 if ism in ismlar:
4     ismlar.remove(ism)
5 print(ismlar) # {'Anvar', 'Abbos'}
```

To'plamdan elementni o'chirishning boshqa usuli ham mavjud bo'lib, `discard()` metodi orqali amalga oshiriladi. Ushbu usulda element to'plamdan o'chirilganda, agar o'chirilayotgan element to'plamda mavjud bo'lmasa ham istisno xatoligi ro'y bermaydi:

```
1 ismlar = {"Anvar", "Abbos", "Abror"}
2 ism = "Abbos"
```

```
3 ismlar.discard(ism)
4 print(ismlar) # {'Anvar', 'Abror'}
```

To'plamning barcha elementlarini birdaniga o'chirish uchun ya'ni to'plamni tozalash uchun *clear()* metodidan foydalaniladi:

```
1 ismlar = {"Anvar", "Abbos", "Abror"}
2 ismlar.clear()
3 print(ismlar) # set()
```

To'plam elementlariga *for* operatori orqali murojaatni (perebor) amalga oshirish mumkin:

```
1 ismlar = {"Anvar", "Abbos", "Abror"}
2 for ism in ismlar:
3     print(ism)
```

bu erda to'plamni har bir elementi *ism* o'zgaruvchisiga ketma-ket yuklanadi va keyingi hisoblashlarda ishlatilishi mumkin.

### **To'plamlar ustuda amallar.**

To'plamlar ustuda turli xil amallarni bajarish mumkin bo'lib, ular metod va funksiyalar orqali amalga oshiriladi. Quyida ulardan eng ko'p qo'llaniladiganlarini qarab chiqamiz:

*copy()* metodi biror bir to'plamdan nusxa olish uchun ishlatiladi, masalan:

```
1 ismlar = {"Anvar", "Abbos", "Abror"}
2 ismlar2 = ismlar.copy()
3 print(ismlar) # {'Abbos', 'Anvar', 'Abror'}
4 print(ismlar2) # {'Abbos', 'Anvar', 'Abror'}
```

*union()* metodi ikkita to'plamni birlashtiradi va qiymat sifatida yangi to'plamni qaytaradi:

```
1 famil = {"Axmedov", "Niyazov"}
2 ism = {"Sardor", "Tohir"}
3 FIO = famil.union(ism)
4 print(FIO) # {'Axmedov', 'Tohir', 'Sardor', 'Niyazov'}
```

*intersection()* metodi ikkita to'plamni kesishmasini olish uchun ishlatib, qiymat sifatida yangi to'plam qaytaradi. Ya'ni ikkita to'plam uchun umumiy bo'lgan elementlarni olish uchun *intersection()* metodi qo'llaniladi:

```

1 famil = {"Axmad", "Sardor", "Ikrom"}
2 ism = {"Sardor", "Tohir", "Ikrom"}
3 ism2 = famil.intersection(ism)
4 print(ism2) # {'Sardor', 'Ikrom'}

```

*intersection()* metodi o'rniga unga ekvivalent bo'lgan & ( mantiqiy ko'paytirish) amalini ham qo'llash mumkin:

```

1 famil = {"Axmad", "Sardor", "Ikrom"}
2 ism = {"Sardor", "Tohir", "Ikrom"}
3 ism2 = famil & ism
4 print(ism2) # {'Sardor', 'Ikrom'}

```

*difference()* metodi to'plamlar ayirmasini topish uchun qo'llanilib, qiymat sifatida yangi to'plam qaytaradi. Ya'ni birinchi to'plamda mavjud va ikkinchi to'plamda yo'q bo'lgan elementlarni topishda ishlatish mumkin. *difference()* metodiga ekvivalent amal bu '-' amalidir:

```

1 famil = {"Axmad", "Sardor", "Ikrom"}
2 ism = {"Sardor", "Tohir", "Ikrom"}
3 ism2 = famil.difference(ism)
4 ism3 = famil - ism
5 print(ism2) # {'Axmad'}
6 print(ism3) # {'Axmad'}

```

*issubset()* metodi qaralayotgan to'plam boshqa to'plam (argumentida berilgan) ning qism to'plami yoki yo'qligini tekshirish uchun ishlatiladi:

```

1 famil = {"Axmad", "Sardor", "Ikrom"}
2 ism = {"Sardor", "Ikrom"}
3 print(ism.issubset(famil)) # True
4 print(famil.issubset(ism)) # False

```

*issuperset()* metodi qaralayotgan to'plam boshqa to'plamni (argumentida berilgan) o'z ichiga olishi yoki olmasligini tekshirish uchun ishlatiladi:

```

1 famil = {"Axmad", "Sardor", "Ikrom"}
2 ism = {"Sardor", "Ikrom"}
3 print(ism.issuperset(famil)) # False
4 print(famil.issuperset(ism)) # True

```



**frozenset.** *frozenset* - o'zgartirib bo'lmaydigan to'plamlarni yaratish uchun ishlatiladi. Ushbu turdagi to'plamga yangi element qo'shish, o'chirish yoki element qiymatini o'zgartirishga ruxsat berilmaydi. *frozenset* turidagi to'plam odatda ro'yxat, kortej yoki oddiy to'plam (*set*) orqali hosil qilinadi:

```
1 famil = {"Axmad", "Sardor", "Ikrom"}
2 fam = frozenset(famil)
3 print(fam) # frozenset({'Sardor', 'Ikrom', 'Axmad'})
```

*frozenset* turidagi to'plamlar ustuda quyidagi amallarni bajarish mumkin:

*len(s)* – *s* to'plam uzunligi (elementlari soni)ni qaytaradi;

*x in s* – *True* qiymat qaytaradi, agar *x* element *s* to'plamning tarkibida mavjud bo'lsa;

*x not in s* – *True* qiymat qaytaradi, agar *x* element *s* to'plamning tarkibida mavjud bo'lmasa;

*s.issubset(t)* – *True* qiymat qaytaradi, agar *t* to'plam *s* to'plamni o'z ichiga olsa;

*s.issuperset(t)* – *True* qiymat qaytaradi, agar *s* to'plam *t* to'plamni o'z ichiga olsa;

*s.union(t)* – *s* va *t* to'plamlarning birlashmasidan tashkil topgan yangi to'plamni qaytaradi;

*s.intersection(t)* – *s* va *t* to'plamlarning kesishmasidan tashkil topgan yangi to'plamni qaytaradi;

*s.difference(t)* – *s* to'plamdan *t* to'plamni ayirishdan hosil bo'lgan yangi to'plamni qaytaradi;

*s.copy()* – *s* to'plamning nusxasini qaytaradi.

### III. Fayllar bilan ishlash

Pythonda turli xil fayl turlari bilan ishlash imkoniyati mavjud bo'lib, shartli ravishda ularni ikki turga bo'lish mumkin: matn va binar fayllar. Matn fayllari, masalan, kengaytmasi cvs, txt, html, umuman, matn shaklida ma'lumot saqlaydigan barcha fayllarlarni o'z ichiga oladi. Binar fayllar tasvirlar, audio va video fayllar va boshqalardan iborat. Fayl turiga qarab u bilan ishlash biroz farq qilishi mumkin.

Fayllar bilan ishlaganda, quyidagi tartibdagi operatsiyalar ketma-ketligini amalga oshirish talab etiladi:

1. *open()* metodi yordamida faylni ochiladi;
2. *read()* metodi yordamida faylni o'qish yoki *write()* metodi yordamida faylga yozish amalga oshiriladi;
3. *close()* metodi faylni yopadi.

#### 3.1. Fayllarni ochish va yopish

Fayllar bilan ishlash uchun avval faylni *open()* metodi yordamida ochish zarur. *open()* metodidan quyidagi ko'rinishda foydalaniladi:

```
1 open(file, mode)
```

Funksiyaning birinchi parametri faylning yo'lini ifodalaydi. Fayl yo'li absolyut bo'lishi mumkin, ya'ni disk harfidan boshlanadi, masalan, C://qandaydirpapka/somefile.txt. Yoki nisbiy bo'lishi mumkin, masalan, qandaydirpapka/ somefile.txt - bu holda, fayl Python ishlaydigan skript joylashgan katalogda hosil qilinadi. Ikkinchi argument *mode* - bu faylni ochish rejimi bo'lib, fayl bilan qanday ish bajarilishiga qarab, 4 turdagi fayllar bilan ishlash rejimidan birini qo'llash mumkin:

- *r (Read)* - Fayl o'qish uchun ochadi. Fayl topilmasa, *FileNotFoundError* xatolik qaytaradi;
- *w (Write)*. Fayl yozish uchun ochadi. Agar fayl yo'q bo'lsa, u hosil bo'ladi. Bunday fayl allaqachon mavjud bo'lsa, u yangidan yaratiladi va shunga mos ravishda eski ma'lumotlar o'chiriladi.

- *a (Append)*. Faylni qayta yozish uchun fayl ochiladi. Agar fayl yo'q bo'lsa, u hosil bo'ladi. Bunday fayl allaqachon mavjud bo'lsa, ma'lumotlar oxiridan yozish davom ettiriladi.
- *b (Binary)*. Binar fayllar bilan ishlash uchun foydalaniladi. *w* va *r* kabi rejimlar kombinatsiyasi bilan birgalikda ishlatiladi.

Fayl bilan ishlashni tugatgandan so'ng uni *close()* metodi bilan yopish kerak bo'ladi. Ushbu metod fayl bilan bog'liq barcha resurslarni bo'shatadi.

Misol uchun, "salom.txt" matnli faylni yozish uchun ochamiz:

```
1 meningfaylim = open("salom.txt", "w")
2 meningfaylim.close()
```

Faylni ochishda yoki u bilan ishlashda turli xil istisno holatlarga duch kelish mumkin, masalan, unga ruxsat yo'q bo'lishi mumkin. Bunday holatlarda, dastur ishlash jarayonida xatolik yuz beradi va dastur bajarilishi *close()* metodi chaqirilishiga yetib bormaydi va shunga muvofiq fayl yopilmaydi. Bu kabi holatlarni oldini olish uchun istisnolardan foydalaniladi:

```
1 try:
2     somefile = open("salom.txt", "w")
3     try:
4         somefile.write("Salom olam")
5     except Exception as e:
6         print(e)
7     finally:
8         somefile.close()
9 except Exception as ex:
10    print(ex)
```

Bu erda, fayl bilan bajariladigan barcha amallar ketma-ketligi *try* blokida yoziladi. Agar biror bir istisno to'satdan kelib chiqsa, u holda *finally* blokida fayl blokirovka qilinadi.

Fayllar bilan ishlashning yanada qulayroq *with* konstruktsiyasi mavjud:

```
1 with open(file, mode) as file_obj:
2     #buyruqlar
```

Bu konstruktsiya ochiq fayl uchun *file\_obj* o'zgaruvchi aniqlanadi va buyruqlar ketma-ketligi bajariladi. Ular bajarilgandan so'ng, fayl avtomatik ravishda yopiladi. Blokda amallar ketma-ketligini bajarishda istisnolar yuzaga kelsa ham, fayl avtomatik ravishda yopiladi.

*with* konstruktsiyasi yordamida, yuqoridagi misolni quyidagicha qayta yozish mumkin:

```
1 with open("salom.txt", "w") as somefile:
2     somefile.write("Salom Python")
```

### 3.2. Matn fayllari. Matn faylga yozish

Matn faylini yozish uchun ochishda *w* (qayta yozish) yoki *a* (yozuv qo'shish) rejimini qo'llaniladi. So'ngra, faylga yozish uchun *write(str)* metodidan foydalanilib, *str* parametriga yozilishi kerak bo'lgan satr uzatiladi. Shuni eslatib o'tish joizki, bu parametr satr bo'lishi shart, shuning uchun raqamlar yoki boshqa turdagi ma'lumotlarni yozish zarur bo'lsa, dastlab ularni satr turiga keltirish talab qilinadi.

"salom.txt" fayliga ba'zi ma'lumotlarni yozamiz:

```
1 with open("salom.txt", "w") as fayl:
2     fayl.write("salom olam")
```

Joriy Python skriptining joylashgan papkasini ochsak, u yerda salom.txt faylini ko'rish mumkin. Ushbu fayl har qanday matn muharriridan ochilishi mumkin va agar kerak bo'lsa o'zgartirilishi ham mumkin.

Keling, ushbu faylga yana bitta qator qo'shamiz:

```
1 with open("salom.txt", "a") as fayl:
2     fayl.write("\nHayr, olam")
```

Agar satrni yangi qatorga yozish zarur bo'lsa, u holda, yuqoridagi kabi, yozilayotgan satr oldidan "\n"(yangi satrga o'tish belgisi) qo'yish yetarli bo'ladi. Yukunida salom.txt faylida quyidagi matn hosil bo'ladi:

*salom olam*

## Hayr, olam

Faylga yozishning yana bir usuli - ma'lumotlarni konsolga chiqarish uchun ishlatiladigan standart `print()` metodan foydalanish orqali amalga oshiriladi:

```
1 with open("salom.txt", "a") as salom_fayl:
2     print("Salom, olam", file=salom_fayl)
```

`print` metodi yordamida ma'lumotlarni faylga chiqarish uchun faylning nomi `file` parametri orqali beriladi va birinchi parametr faylga yoziladigan ma'lumotni ifodalaydi.

## Fayldan o'qish

Fayldan o'qish uchun `r` (*Read*) rejimida ochiladi va uning mazmunini turli usullar bilan o'qish mumkin:

- `readline()` - fayldan bir qator o'qiydi;
- `read()` - faylning butun tarkibini bir qatorga o'qiydi;
- `readlines()` - faylning barcha satrlarini ro'yxatga oladi.

Masalan, biz yuqorida yozilgan fayllarni satrlar bo'yicha ko'rib chiqamiz:

```
1 with open("salom.txt", "r") as fayl:
2     for satr in fayl:
3         print(satr, end="")
```

Biz, albatta, har bir qatorni o'qish uchun `readline()` metodini ishlatmasak ham, har bir yangi satrni olish uchun ushbu metod avtomatik ravishda chaqiriladi. Shuning uchun ham, `readline()` metodini siklda chaqirishdan ma'no yo'q va satrlar yangi satr `"\n"` belgisi bilan ajratilganligi uchun, yangi satrga chop qilish zaruriyati qolmaydi va `end=""` qiymati `print` metodining ikkinchi parametri sifatida uzatiladi. Endi satrlarni alohida o'qish uchun `readline()` metodini to'g'iridan-to'g'ri chaqiramiz:

```
1 with open("salom.txt", "r") as fayl:
2     str1 = fayl.readline()
3     print(str1, end="")
```

```
4     str2 = fayl.readline()
5     print(str2)
```

Konsol ektaniga quyudagi natijalar chiqariladi:

```
salom olam
```

```
hayr, olam
```

*readline()* metodini alohida qatordagi satrlarni o'qish uchun *while* siklida ham foydalanish mumkin:

```
1     with open("salom.txt", "r") as fayl:
2         satr = fayl.readline()
3         while satr:
4             print(satr, end="")
5             satr = fayl.readline()
```

Fayl kichik bo'lsa, *read()* metodidan foydalanib, uni birdan o'qishingiz mumkin:

```
1     with open("salom.txt", "r") as fayl:
2         mazmun = fayl.read()
3         print(mazmun)
```

Hamda, *readlines()* metodi yordamida fayldagi barcha satrlar ro'yxatga o'qib olinadi, ya'ni elementlari fayldagi satrlardan tashkil topgan ro'yxat hosil qilinadi:

```
1     with open("salom.txt", "r") as faly:
2         mazmun = fayl.readlines()
3         str1 = mazmun [0]
4         str2 = mazmun [1]
5         print(str1, end="")
6         print(str2)
```

Ba'zida fayldagi ma'lumotlar ASCII dagi belgilardan farqlanishi mumkin. Ushbu holatda fayldan berilganlarni o'qish to'g'ri bo'lishi uchun kodlash parametrini ishlatib kodlashni aniq belgilab olishimiz mumkin:

```
1     faylnomi = "salom.txt"
2     with open(faylnomi, encoding="utf8") as file:
3         matn = file.read()
```

Quyidagi dastur orqali foydalanuvchi tomonidan kiritilgan satrlar massivi dastlab faylga yozish amalga oshirilgan, so'ngra ularni fayldan konsolga qayta o`qib, chop qilish amalga oshirilgan:

```
1 # fayl nomi
2 FILENAME = "habarlar.txt"
3 # bo'sh ro'yxat aniqlaymiz
4 xabarlar = list()
5
6 for i in range(4):
7     xabar = input("Satrni kiriting " + str(i + 1) + ": ")
8     xabarlar.append(xabar + "\n")
9
10 # ro'yxatni faylga yozish
11 with open(FILENAME, "a") as fayl:
12     for xabar in xabarlar:
13         fayl.write(xabar)
14
15 # xabarlarni fayldan o'qiyamiz
16 print("Xabarlarni o'qish")
17 with open(FILENAME, "r") as fayl:
18     for xabar in fayl:
19         print(xabar, end="")
```

Dastur ishlashining namunasi:

```
1 Satrni kiriting 1: salom
2 Satrni kiriting 2: tinchlik so'zi
3 Satrni kiriting 3: buyuk ish
4 Satrni kiriting 4: Python
5 Xabarlarni o'qish
6 Salom
7 tinchlik so'zi
8 buyuk ish
9 Python
```

### 3.3. CSV fayllari bilan ishlash

Ma'lumotni qulay shaklda saqlashning keng tarqalgan fayl formatlaridan biri csv formatidir. CSV faylidagi har bir satr vergul bilan ajratilgan alohida ustunlardan iborat bo'lgan yozuv yoki satrni aks ettiradi. Aslida, bu format "Vergul bilan ajratilgan qiymatlar (Comma Separated Values)" deb nomlanadi. CSV formati matnli fayl formati bo'lsa-da, Python u bilan ishlashni soddalashtirish uchun maxsus ajralmas CSV modulini taqdim etadi.

Quyidagi misolda modulning ishini ko'rib chiqamiz:

```
1 import csv
2
3 FILENAME = "users.csv"
4
5 users = [
6     ["Ali", 25],
7     ["Sobir", 32],
8     ["Dilnoza", 14]
9 ]
10
11 with open(FILENAME, "w", newline="") as fayl:
12     writer = csv.writer(fayl)
13     writer.writerows(users)
14
15 with open(FILENAME, "a", newline="") as fayl:
16     user = ["Shaxnoza", 18]
17     writer = csv.writer(fayl)
18     writer.writerow(user)
```

Faylga ikki o'lchovli ro'yxat yoziladi – har bir satr bitta foydalanuvchini ifodalaydigan jadval. Har bir foydalanuvchi esa ikkita maydon - ism va yoshni o'z ichiga oladi. Ya'ni, uchta satr va ikki ustunli jadvalni ifodalaydi.

Yozish uchun fayl ochilganda, uchinchi parametr sifatida *newline=""* qiymati ko'satildi - bo'sh satr operatsion tizimidan qat'i nazar, fayllardan to'g'ri satrlarni o'qishga imkon beradi.



Yozish uchun `csv.writer(file)` funksiyasi tomonidan qaytariladigan `writer` obyektini olishimiz kerak. Ushbu funktsiyaga ochiq fayl topshiriladi. Hamda, mos ravishda yozish `writer.writerows(users)` metodi yordamida amalga oshiriladi. Bu usul qatorlar to'plamini parametr sifatida oladi. Bizning holatimizda bu ikki o'lchovli ro'yxat hisoblanadi.

Agar bitta yozuv qo'shish zarur bo'lsa, ya'ni, bir o'lchamli ro'yxat, masalan, `["Shaxnoza", 18]`, bu holda `writer.writerow(user)` metodidan foydalaniladi. Natijada, skriptni ishga tushirgandan so'ng, quyidagi tarkibga ega bo'lgan `users.csv` fayli shu papkada paydo bo'ladi:

1	Ali,25
2	Sobir,32
3	Dilnoza,14
4	Shaxnoza,18

Fayldan o'qish uchun, aksincha, `reader` obyektini yaratishimiz kerak:

```
1 import csv
2
3 FILENAME = "users.csv"
4
5 with open(FILENAME, "r", newline="") as fayl:
6     reader = csv.reader(fayl)
7     for row in reader:
8         print(row[0], " - ", row[1])
```

`reader` obyektini olayotganda, biz uning barcha satrlarini ko'chirib olishimiz mumkin:

1	Ali - 25
2	Sobir - 32
3	Dilnoza - 14
4	Shaxnoza - 18

### Lug'atlar bilan ishlash

Yuqoridagi misolda har bir yozuv yoki satr alohida ro'yxat bo'lib, masalan, `["Shaxnoza", 18]`. Bundan tashqari, CSV modullari lug'atlar bilan ishlash uchun

maxsus qo'shimcha xususiyatlarga ega. Xususan, `csv.DictWriter()` funksiyasi faylga yozish imkonini beruvchi `writer` obyektini qaytaradi va `csv.DictReader()` funksiyasi esa fayldan o'qish uchun `reader` obyektini qaytaradi. Masalan:

```
1  import csv
2
3  FILENAME = "users.csv"
4
5  users = [
6      {"ism": "Ali", "yosh": 25},
7      {"ism": "Sobir", "yosh": 32},
8      {"ism": "Dilnoza", "yosh": 14}
9  ]
10
11 with open(FILENAME, "w", newline="") as fayl:
12     ustunlar = ["ism", "yosh"]
13     writer = csv.DictWriter(fayl, fieldnames=ustunlar)
14     writer.writeheader()
15
16     # bir qancha qatorlarni yozish
17     writer.writerows(users)
18
19     user = {"ism": "Shaxnoza", "yosh": 18}
20     # bitta qatorni yozish
21     writer.writerow(user)
22
23 with open(FILENAME, "r", newline="") as fayl:
24     reader = csv.DictReader(fayl)
25     for row in reader:
26         print(row["ism"], "-", row["yosh"])
```

Qatorlarni `writerow()` va `writerows()` metodlari yordamida ham yozish mumkin. Ammo endi har bir satr alohida lug'at, hamda, ustun sarlavhalari `writeheader()` metodidan foydalanib yoziladi va ikkinchi parametr sifatida `csv.DictWriter` metodiga ustunlar to'plami uzatiladi.

Ustunlardan foydalangan holda satrdan berilganlarni o'qishda satr ichidagi alohida qiymatlarga `row["ism"]` ko'rinishida murojaat qilishimiz mumkin.

### 3.4. Binar fayllar

Binar fayllarda, matnli fayllardan farqli o'laroq, ma'lumotlar baytlar majmui sifatida saqlanadi. Pythonda ular bilan ishlash uchun *pickle* – ichki modul kerak bo'ladi. Ushbu modul ikkita metodni taqdim etadi:

- `dump(obj, file)` - obyektни ikkilik faylga yozadi;
- `load(file)` - binar fayldan obyektga ma'lumotlarni o'qiydi.

O'qish yoki yozish uchun binar faylni ochishda yozishni ("w") yoki o'qish ("r") rejimidan tashqari "b" rejimidan foydalanish kerakligini ham hisobga olish kerak.

Masalan, ikkita obyektни saqlash kerak bo'lsin:

```
1 import pickle
2
3 FILENAME = "user.dat"
4 ism = "ali"
5 yosh = 25
6
7 with open(FILENAME, "wb") as file:
8     pickle.dump(ism, file)
9     pickle.dump(yosh, file)
10
11 with open(FILENAME, "rb") as file:
12     ism = pickle.load(file)
13     yosh = pickle.load(file)
14     print("Ism: ", ism, "\tYosh: ", yosh)
15
```

`dump()` funksiyasi bilan ikki obyekt ketma-ket yoziladi. Shu sababli, fayldan ketma-ketlikda `load()` funksiyasi yordamida o'qiganimizda mos ravishda yozilgan obyektlarni olamiz. Dastur ishlaganda konsol ekraniga quyidagilar chiqariladi:

```
1 Ism: Ali Yosh: 25
```

Shu tarzda, faylga obyektlar to'plamini saqlashimiz va undan o'qib olishimiz mumkin:

```
1 import pickle
2
3 FILENAME = "users.dat"
4
5 users = [
6     ["Ali", 25, True],
7     ["Sobir", 32, False],
8     ["Dilnoza", 14, False]
9 ]
10
11 with open(FILENAME, "wb") as file:
12     pickle.dump(users, file)
13
14 with open(FILENAME, "rb") as file:
15     users_from_file = pickle.load(file)
16     for user in users_from_file:
17         print("Ism: ", user[0], "\tYosh: ", user[1],
18               "\tUylangan(turmushga chiqan): ", user[2])
```

Faylga *dump()* funksiyasidan foydalanib qanday obyekt yozilsa, fayldan *load()* funksiyasi orqali xuddi shu obyekt o'qiladi.

Natijaning konsolga chiqarilishi:

```
1 Ism: Ali Yosh: 25 Uylangan(turmushga chiqan): True
2 Ism: Sobir Yosh: 32 Uylangan(turmushga chiqan): False
3 Ism: Dilnoza Yosh: 14 Uylangan(turmushga chiqan): False
```

### 3.5. shelve moduli

Binar fayllar bilan ishlash uchun Pythonda yana bitta modul - shelve modulidan foydalanish mumkin. Obyektlarni maxsus kalitga ega faylga saqlaydi. Keyinchalik, bu kalit yordamida avvaldan saqlangan obyektни fayldan oqib olinadi.

*shelve* modul orqali ma'lumotlarni ishlash jarayoni lug'atlarga o'xshash, shuning uchun obyektlarni saqlash va o'qib olish uchun kalitlardan foydalaniladi.

Faylni ochish uchun *shelve* moduli *open()* funksiyasidan foydalanadi:

```
1 open(fayl_manzili[, flag="c"[, protocol=None[,  
writeback=False]])
```

*flag* parametri quyidagi qiymatlarni qabul qilishi mumkin:

- *c* - faylni o'qish va yozish uchun ochadi (*flag=c*, *flag* parametrining kelishuv bo'yicha qiymati aynan *c* ga teng). Agar fayl mavjud bo'lmasa, u yaratiladi;
- *r* - fayl faqat o'qish uchun ochiladi;
- *w* - fayl faqat yozish uchun ochiladi;
- *n* - fayl yozish uchun ochilgan. Agar fayl mavjud bo'lmasa, fayl yaratiladi. Agar mavjud bo'lsa, u holda fayl qayta yozish uchun ochiladi.

Faylga ulanishni yopish uchun *close()* metodidan foydalaniladi:

```
1 import shelve  
2 d = shelve.open(faylnomi)  
3 d.close()
```

Yoki faylni *with* operatoridan foydalanib ochishingiz mumkin. Quyidagi dasturda faylga bir nechta obyektни saqlash va keyin undan o'qib olish ko'rsatilgan:

```
1 import shelve  
2  
3 FILENAME = "davlatlar"  
4 with shelve.open(FILENAME) as states:  
5     states["Toshkent"] = "O\`zbekiston"  
6     states["Moskva"] = "Rossiya"  
7     states["Pekin"] = "Xitoy"  
8     states["Seul"] = "Korea"  
9  
10 with shelve.open(FILENAME) as states:  
11     print(states["Toshkent"])  
12     print(states["Pekin"])
```

Berilganlarni faylga yozish – bu ma`lum bir kalitga qiymatni yuklash tarzida amalga oshiriladi:

```
1 states["Toshkent"] = "O\'leksiton"
```

Fayldan ma`lumotlarni o`qib olish kalit bo'yicha qiymatni olishga ekvivalent tarzda amalga oshiriladi:

```
1 print(states["Toshkent"])
```

Kalit sifatida satrlar ishlatiladi.

Ma'lumotni o'qiyotganda so'ralgan kalit mavjud bo'lmasa, u holda istisno holati yuzaga keladi. Shu kabi vaziyatlarda, odatda, kalitni mavjud yoki yo'qligini tekshirish tavsiya etiladi va buning uchun *in* operatoridan foydalaniladi, masalan:

```
1 with shelve.open(FILENAME) as states:
2     key = "Ostana"
3     if key in states:
4         print(states[key])
```

Fayldan ma`lumotlarni o`qib olish uchun *get()* metodidan ham foydalanish mumkin. Ushbu metodning birinchi parametri – kalit bo'lib, u orqali qiymat olinadi, ikkinchi parametr esa, birinchi parametrda keltirilgan kalit topilmasa qaytariladigan kelishuv bo'yicha qiymatni ifodalaydi.

```
1 with shelve.open(FILENAME) as states:
2     state = states.get("Ostana", "Aniqlanmagan")
3     print(state)
```

*for* operatoridan foydalanib, fayldagi barcha qiymatlarni ketma-ket ko'rib chiqish mumkin:

```
1 with shelve.open(FILENAME) as states:
2     for key in states:
3         print(key, " - ", states[key])
```

*keys()* metodi fayldan natija sifatida barcha kalitlarni, *values()* metodi esa barcha qiymatlarni qaytaradi:

```
1 with shelve.open(FILENAME) as states:
2     for city in states.keys():
```

```

3         print(city, end=" ") # Toshkent Moskva Pekin Seul
4     print()
5     for country in states.values():
6         print(country, end=" ") # O'zbekiston Rossiya
7     Xitoy Korea

```

Yana bir metod - *items()* metodi har bir elementi kalit va qiymat juftligini tashkil qiluvchi kortejlardan iborat kortejlar to'plamini qaytaradi:

```

1     with shelve.open(FILENAME) as states:
2         for state in states.items():
3             print(state)

```

Natija:

```

1     ("Toshkent", "O'zbekiston")
2     ("Moskva", "Rossiya")
3     ("Pekin", "Xitoy")
4     ("Seul", "Korea")

```

### Ma'lumotlarni yangilash

Ma'lumotlarni o'zgartirish uchun kalitga yangi qiymat o'zlashtirish va yangi berilganlarni qo'shish uchun yangi kalit bo'yicha qiymat berish kifoya:

```

1     import shelve
2
3     FILENAME = "states2"
4     with shelve.open(FILENAME) as states:
5         states["Tashkent"] = "O'zbekiston"
6         states["Moskva"] = "Rossiya"
7         states["Pekin"] = "Xitoy"
8         states["Seul"] = "Korea"
9
10    with shelve.open(FILENAME) as states:
11        states["Tashkent"] = "O'zbekiston Respublikasi"
12        states["Ostona"] = "Kozog'iston"
13        for key in states:
14            print(key, " - ", states[key])

```

## Ma'lumotlarni o'chirish

Faydan elementni o'chirish va qiymat sifatida o'chirilgan elementni qaytarish uchun `pop()` metodidan foydalanilib, unga birinchi parametr sifatida o'chirilishi kerak bo'lgan element kaliti hamda ikkinchi parametriga faylda birinchi parametrda ko'rsatilgan kalit mavjud bo'lmasa natija sifatida qaytariladigan kelishuv bo'yicha qiymat beriladi:

```
1 with shelve.open(FILENAME) as states:
2     state = states.pop("Tashkent", "Topilmadi")
3     print(state)
```

`del` operatorini ham o'chirish uchun ishlatilishi mumkin:

```
1 with shelve.open(FILENAME) as states:
2     del states["Pekin"] # Pekin kalit bilan obyektini
    o'chirish
```

`clear()` metodi yordamida hamma elementlarni o'chirish mumkin:

```
1 with shelve.open(FILENAME) as states:
2     states.clear()
```

## 3.6. OS moduli va fayl tizimi ishlashi

Kataloglar va fayllar bilan ishlash uchun bir qancha imkoniyatlar ichki `os` moduli tomonidan ta'minlanadi. Ushbu modul ko'p funktsiyani o'z ichiga olgan bo'lsada, faqat asosiylarini ko'rib chiqamiz:

- **mkdir()** – yangi papka yaratadi;
- **rmdir()** – papkani o'chiradi;
- **rename()** – fayl nomini o'zgartiradi;
- **remove()** – faylni o'chiradi.

### Papkalarni yaratish va o'chirish

Papka yaratish uchun `mkdir()` funksiyasidan foydalaniladi, unga yaratilayotgan papkaga yo'li beriladi:



```
1 import os
2
3 #Skript joylashgan adresga nisbiy yo'l
4 os.mkdir("hello")
5 # absolyut adres
6 os.mkdir("c://somedir")
7 os.mkdir("c://somedir/hello")
```

Papkani o'chirish uchun `rmdir()` funksiyasidan foydalanilib, o'chirilishi kerak bo'lgan papkanining yo'li beriladi:

```
1 import os
2
3 #Skript joylashgan adresga nisbiy yo'l
4 os.rmdir("hello")
5 # absolyut adres
6 os.rmdir("c://somedir/hello")
```

### Faylni qayta nomlash

Fayllarni qayta nomlash uchun `rename(source, target)` funksiyasidan ishlatiladi, birinchi parametrda manba fayl yo'li, ikkinchisida esa yangi fayl nomi beriladi. Fayl joylashgan yo'lini berishda ham mutlaq va ham nisbiy yo'llar berilishi mumkin. Misol uchun, "somefile.txt" fayli "C://SomeDir/" papkasida joylashganligini tasavvur qiling. Uni "hello.txt" fayliga o'zgartirish:

```
1 import os
2
3 os.rename("C://SomeDir/somefile.txt",
4 "C://SomeDir/hello.txt")
```

### Faylni o'chirish

Biror faylni o'chirish uchun `remove()` funksiyasida foydalaniladi. Ushbu funksiyaga parametr sifatida o'chirilishi lozim bo'lgan fayl yo'li beriladi, masalan:

```
1 import os
2
3 os.remove("C://SomeDir/hello.txt")
```

### Fayl mavjudligi

Agar mavjud bo'lmagan faylni ochishga harakat qilinsa, u holda Python *FileNotFoundError* istisno holati ro'y berganligi to'g'risida xatolik xabarini qaytaradi. Bunday istisno holatini uchun *try...except* konstruksiyasidan foydalanishimiz mumkin. Biroq, faylni ochishdan avval *os.path.exists(path)* metodi yordamida fayl mavjudligini tekshirish lozim. Tekshirilishi kerak bo'lgan yo'l bu metodga uzatiladi:

```
1 filename = input ("Faylning yo'lini kiriting:")
2 if os.path.exists(filename):
3     print("ko'rsatilgan fayl mavjud")
4 else:
5     print("Fayl mavjud emas")
```

## IV. Satrlar

### 4.1. Satrlari bilan ishlash

Satr Unicode kodirovkasidagi belgilar ketma-ketligini ifodalashda qo'llaniladi. Satrlarning alohida belgilariga kvadrat qavs ichida indekslarini ko'rsatish orqali murojaat qilishimiz mumkin:

```
1 string = "hello world"
2 c0 = string[0] # h
3 print(c0)
4 c6 = string[6] # w
5 print(c6)
6
7 c11 = string[11] # xatolik IndexError: satr indeksi
8 chegaradan chiqib ketdi
9 print(c11)
```

Indekslash noldan boshlanadi, shuning uchun satrning birinchi belgisi 0 indeksiga ega bo'ladi. Agar satrda mavjud bo'lmagan indeksga murojaat qilinsa, *IndexError* istisno xatoligi yuzaga keladi. Masalan, yuqoridagi holatda, satr 11 ta belgidan iborat bo'lib, uning belgilari 0 dan 10 gacha indekslarga ega bo'ladi

Satrn oxirdan boshlab uning belgilariga murojaat qilish uchun manfiy indekslashdan foydalanish mumkin. Ya'ni, satrdagi eng oxirgi belgini -1 indeks orqali, oxirdan bitta oldingi belgisini -2 indeks orqali va hakozi tartibda olish mumkin.

```
1 string = "hello world"
2 c1 = string[-1] # d
3 print(c1)
4 c5 = string[-5] # w
5 print(c5)
```

Satrlar o'zgarmaydigan (immutable) tur hisoblanadi, shuning uchun ham satrni o'zgartirishga harakat qilinsa xatolik yuzaga keladi, masalan:

```
1 ss = "hello world"
2 ss[1] = "R"
```

Bu erda *ss* satrning 2-chi belgisiga yangi qiymat berish uchun  $ss[l] = "R"$  tarzda amal yozilgan. Natijada xatolik ro'y beradi. Chunki satr tarkibidagi belgilarni uning indeksi orqali o'zgartirishga yo'l qo'yilmaydi.

Faqatgina satrga boshqa qiymat yozish orqali satrni to'liq o'zgartira olamiz.

### Satr ostilarini olish

Agar zarur bo'lsa, satrdan nafaqat bitta belgini, balki satr ostisini ham olishimiz mumkin. Buni amalga oshirish uchun quyidagi sintaksisdan foydalanamiz:

- `string[: end]` – 0-indeksdan boshlab *end*gacha belgilar ketma-ketligini oladi;
- `string[start: end]` – *start* indeksdan *end* indeksigacha bo'lgan belgilarni oladi;
- `string[start: end: step]` – *step* qadam bilan *start* indeksdan boshlab *end* indeksigacha bo'lgan belgilar ketma-ketligini oladi.

Satr ostisini olish uchun barcha variantlardan foydalanamiz:

```
1 string = "hello world"
2
3 # 0 dan 5 gacha belgilar
4 sub_string1 = string[:5]
5 print(sub_string1) # hello
6
7 # 2 dan 5 gacha belgilar
8 sub_string2 = string[2:5]
9 print(sub_string2) # llo
10
11 # 2 dan 9 gacha bitta belgi tashlab
12 sub_string3 = string[2:9:2]
13 print(sub_string3) # lowr
```

### ord va len funksiyalar

Agar satr Unicode belgilaridan tashkil topgan bo'lsa, u holda *ord()* funksiyasi yordamida belgining Unicode kodlashdagi sonli qiymatini olish mumkin:

```
1 print(ord("A")) # 65
```

Satr uzunligi (satrdagi belgilar soni)ni olish uchun *len()* funksiyasidan foydalanish mumkin:

```
1 string = "hello world"
2 length = len(string)
3 print(length) # 11
```

### Satrdan izlash

Satrdan *term* satr ostisini qidirish uchun *term in string* ifodasidan foydalanamiz. Agar satr ostisi topilsa ifoda *True* qiymat qaytaradi, aks holda *False* qiymat qaytaradi:

```
1 string = "hello world"
2 exist = "hello" in string
3 print(exist) # True
4
5 exist = "sword" in string
6 print(exist) # False
```

### Satrlarni ajratish

*for* sikl operatori yordamida satrning barcha elementlarini ajratib olish mumkin:

```
1 string = "hello world"
2 for char in string:
3     print(char)
```

## 4.2. Satrlar bilash ishlashning asosiy metodlari

Eng ko'p qo'llanilishi mumkin bo'lgan asosiy metodlarini ko'rib chiqamiz:

- *isalpha(str)* – satr faqat alifbo belgilaridan iborat bo'lsa *True*, aks holda *False* qaytaradi;
- *islower(str)* – satr faqat kichik harflardan iborat bo'lsa *True*, aks holda *False* qaytaradi;

- *isupper(str)* – satr faqat katta harflardan iborat bo'lsa *True*, aks holda *False* qaytaradi;
- *isdigit(str)* – satrdagi barcha belgilar raqamlardan iborat bo'lsa, u holda *True*, aks holda *False* qaytaradi;
- *isnumeric(str)* – satr sonni ifodalasa, *True* qaytaradi;
- *startswith(str)* – satr *str* satr ostisi bilan boshlangan bo'lsa, *True* qaytaradi;
- *endwith(str)* – satr *str* satr ostisi bilan tugagan bo'lsa, *True* qaytaradi;
- *lower()* – satrdagi barcha alfavit belgilarini quyi registrga o'tkazadi;
- *upper()* – satrdagi barcha avfavit belgilarini yuqori registrga o'tkazadi;
- *title()* – satrdagi barcha so'zlarning boshlang'ich harflari katta harfga aylantiradi;
- *capitalize()* – satrda faqat birinchi so'zning birinchi harfini yuqori registrga o'tkazadi;
- *rstrip()* – satrning boshidagi bo'sh joylarni olib tashlaydi;
- *rstrip()* – satrning oxiridagi bo'sh joylarni olib tashlaydi;
- *strip()* – satr boshidagi va oxiradi bo'sh joylarni olib tashlaydi;
- *ljust(width)* – satrning uzunligi *width* parametrlaridan kichik bo'lsa, *width* qiymatini to'ldirish uchun satrning o'ng tomoniga bo'shliqlar qo'shiladi va satr o'zi chap tomonga tekislanadi;
- *rjust(width)* – satr uzunligi *width* parametrlaridan kichik bo'lsa, *width* qiymatini to'ldirish uchun satrning chap qismiga bo'shliqlar qo'shiladi va satr o'zi o'ng tomonga tekislanadi;
- *center(width)* – satrning uzunligi *width* parametridan kichik bo'lsa, *width* qiymatini to'ldirish uchun satrning chap va o'ng tomonlariga bo'sh joylar qo'shiladi va satr markazida joylashadi;
- *find(str[, start[, end]])* – satrdagi birinchi satr ostisi indeksini qaytaradi. Satr ostisi topilmasa, -1 sonini qaytariladi;
- *replace(old, new [, num])* – satr ostilarini almashtiradi;

- `split([delimiter [, num]])` – ajratuvchi belgilar asosida satrni qismlarga ajratadi;
- `join(strs)` – Ma`lum bir ajratuvchi bo'yicha satrlarni bir satrga birlashtiradi.

Quyidagi dasturda `isnumeric()` metodidan foydalangan xolda klaviatura yordamida kiritiladigan qiymatning son yoki yo'qligi tekshirilgan:

```
1 string = input("Son kiriting: ")
2 if string.isnumeric():
3     number = int(string)
4     print(number)
```

Satrning ma`lum bir satr ostisi bilan boshlanishi yoki tugashiga tekshirish:

```
1 file_name = "hello.py"
2
3 starts_with_hello = file_name.startswith("hello") # True
4 ends_with_exe = file_name.endswith("exe") # False
```

Satrning boshida va oxirida bo'sh joylarni olib tashlash:

```
1 string = "  hello world!  "
2 string = string.strip()
3 print(string)           # hello world!
```

Satrlarga bo'sh joylar qo'shish va tekislash:

```
1 print("iPhone 7:", "52000".rjust(10))
2 print("Huawei P10:", "36000".rjust(10))
```

Konsol chiqishi:

```
1 iPhone 7:         52000
2 Huawei P10:      36000
```

## Satrdan qidirish

Pythonda satrdan satr ostisini qidirish uchun `find()` metodni foydalaniladi, bu metod satr ostisining satrdagi birinchi belgisining indeksni qaytaradi va uchta shaklga ega:

- `find(str)` – satr ostisini satrdan izlash boshidan oxirigacha o'tkaziladi;
- `find(str, start)` – izlash `start` indeksdan boshlanadi;

- *find(str, start, end)* – izlash *start* indeksdan *end* indeksgacha bo‘ladi.

Satr ostisi topilmasa -1 qiymatini qaytaradi.

```

1 welcome = "Hello world! Goodbye world!"
2 index = welcome.find("wor")
3 print(index) # 6
4
5 # поиск с 10-го индекса
6 index = welcome.find("wor", 10)
7 print(index) # 21
8
9 # поиск с 10 по 15 индекс
10 index = welcome.find("wor", 10, 15)
11 print(index) # -1

```

### Satrlarni almashtirish

Satrlardagi biror satr ostisini boshqa satr ostisiga almashtirish uchun *replace()* metodidan foydalaniladi:

- *replace(old, new)* – *old* satr ostisini *new* satr ostisiga almashtiradi;
- *replace(old, new, num)* – *num* parametri dastlabki nechta satr ostisini yagisi bilan almashtirish lozimligini ifodalaydi. Qolganlari o‘zgarishsiz qoladi.

```

1 phone = "+1-234-567-89-10"
2
3 # defislarni bo'sh joylarga almashtirish
4 edited_phone = phone.replace("-", " ")
5 print(edited_phone) # +1 234 567 89 10
6
7 # defislarni o'chirish
8 edited_phone = phone.replace("-", "")
9 print(edited_phone) # +12345678910
10
11 # bitta defisni o'chirish
12 edited_phone = phone.replace("-", "", 1)
13 print(edited_phone) # +1234-567-89-10

```



## Satr ostilariga ajratish

*split()* metodi satrlarni ajratuvchiga bo'liq ravishda satr ostilariga ajratish uchun qo'llaniladi. Ajratuvchilar sifatida ixtiyoriy satr yoki belgilarni berish mumkin. Ushbu metod quyidagi ko'rinishga ega:

- *split()* – ajratuvchi sifatida bo'sh joy (probel) qo'llaniladi;
- *split(delimiter)* – ajratuvchi sifatida *delimiter* ishlatiladi;
- *split(delimiter, num)* – ajratuvchi sifatida *delimiter* ishlatiladi, *num* esa nechta bo'lakka bo'lish kerakligini ko'rsatadi. Agar satrni *delimiter* orqali bo'laklarga ajratganda, bo'laklar soni *num* da keltirilgan sondan katta bo'lsa, u xolda dastlabki *num* ta bo'lak ajratiladi, qolgan qismi ajratilmaydi.

```
1 text = "Ali, Vali, G`ani - Sobir, Said qani?"
2
3 txt = text.split()
4 print("Parametrsiz ishlatish, bu erda ajratuvchi probel
5 bo'ladi:")
6 print(txt)
7 print("3-chi bo'lak:")
8 print(txt[2])
9
10 print("Ajratuvchi sifatida vergul ishlatilgan:")
11 txt = text.split(",")
12 print(txt)
13 print("2-chi bo'lak:")
14 print(txt[1])
15
16 print("Ajaratuvchi probel va bo'laklar soni 3 ta:")
17 txt = text.split(" ", 3)
18 print(txt)
19 print("4-chi bo'lak:")
20 print(txt[3])
```

Yuqoridagi dastur ishga tushganda quyidagicha javob konsolga chiqariladi:

*Parametrsiz ishlatish, bu erda ajratuvchi probel bo'ladi:*

*['Ali,', 'Vali,', 'G`ani', '-', 'Sobir,', 'Said', 'qani?']*

*3-chi bo'lak:*

*G`ani*

*Ajratuvchi sifatida vergul ishlatilgan:*

*['Ali', 'Vali', 'G`ani - Sobir', 'Said qani?']*

*2-chi bo'lak:*

*Vali*

*Ajaratuvchi probel va bo'laklar soni 3 ta:*

*['Ali,', 'Vali,', 'G`ani', '- Sobir, Said qani?']*

*4-chi bo'lak:*

*- Sobir, Said qani?*

### **Satrlarni birlashtirish**

Satrlarni oddiy “Qo’shish” amali bilan ham birlashtirish mumkin ekanligini ko’rib chiqdik. *join()* metodi yordamida ham satrlarni birlashtirish mumkin, bu metod satrlardan iborat ro’yxatni bitta satrga aylantiradi. Buning uchun berilgan satrni ajratuvchi sifatida qarab satrdan *join()* metodiga murojaat qilinadi:

```
1 words = ["Let", "me", "speak", "from", "my", "heart", "in",
2 "English"]
3
4 # ajratuvchi - bo'sh joy
5 sentence = " ".join(words)
6 print(sentence) # Let me speak from my heart in English
7
8 # ajratuvchi - vertical chiziq
9 sentence = " | ".join(words)
10 print(sentence) # Let | me | speak | from | my | heart |
11 #in | English
```

*join()* metodiga ro'yxat o'rniga oddiy satrni ham berish mumkin, bu holda ajratuvchi satrdagi belgilar o'rniga tushadi:

```
1 word = "hello"
2 joined_word = "|".join(word)
3 print(joined_word)      # h|e|l|l|o
```

### 4.3. Formatlash

Satr turida aniqlangan *format()* metodi satrlarni formatlash imkonini beradi. Formatlashda satrda aniqlangan to'ldiruvchilar o'rniga ularning qiymatlarini qo'yish mumkin. Satrda to'ldiruvchilar maxsus “{}” figurali qavslar ichida aniqlanadi.

#### Parametrlarni nomlash

Formatlanayotgan satrda partametrlarni aniqlash mumkin va ularga *format()* metodi ichida qiymat beriladi:

```
1 text = "Hello, {first_name}.".format(first_name="Tom")
2 print(text)  # Hello, Tom.
3
4 info = "Name: {name}\t Age: {age}".format(name="Bob",
5 age=23)
6 print(info)  # Name: Bob      Age: 23
```

Bundan tashqari, *format()* metodida argumentlar satrdagi parametrlar bilan bir xil nom bilan aniqlanishi shart. Shunday qilib, agar parametr birinchi holatda bo'lgani kabi *first\_name* deb ataladigan bo'lsa, unda qiymati tayinlangan argument ham *first\_name* deb nomlanadi.

#### O'rinlar bo'yicha parametrlar

Parametrlarni nomlashdan tashqari noldan boshlab raqamlash ham mumkin, bu holda *format()* metodiga parametrlarning faqat qiymatlari uzatiladi va parametrlar kelish tartibi bo'yicha satrga joylashtiriladi:

```
1 info = "Name: {0}\t Age: {1}".format("Bob", 23)
2 print(info)      # Name: Bob      Age: 23
```

Bunday holda parametni satrda bir necha bor foydalanish mumkin:

```
1 text = "Hello, {0} {0} {0}.".format("Tom")
```

### O'rin almashtirish

O'rin almashtirish va maxsus to'ldiruvchilar satrga formatli qiymalarni berishning yana bir usuli hisoblanadi. Formatlash uchun quyidagi maxsus to'ldiruvchilardan foydalanishimiz mumkin:

- s – satr qo'yish uchun;
- d – butun son qo'yish uchun;
- f – haqiqiy son qo'yish uchun. Bu tur uchun kasr qismidagi xonalar sonini nutqa orqali berish mumkin.
- % – 100 ga ko'paytiradi va foiz belgisini qo'shadi;
- e – sonni eksponentsial ko'rinishda chiqaradi.

Maxsus to'ldiruvchining umumiy ko'rinishi quyidagicha:

```
1 {:to'ldiruvchi}
```

To'ldiruvchilarga bog'liq ravishda qo'shimcha parametrlar qo'shish mumkin.

Masalan, *float* turidagi son uchun quyidagicha:

```
1 {: [belgilar_soni] [vergul] [.kasr_qismidagi_belgilar_soni]  
to'ldiruvchi}
```

*format()* metodi chaqirilganda, unga argument sifatida to'ldiruvchi o'rniga yoziladigan qiymatlar beriladi:

```
1 welcome = "Hello {:s}"  
2 name = "Tom"  
3 formatted_welcome = welcome.format(name)  
4 print(formatted_welcome) # Hello Tom
```

*format()* natijasi sifatida formatlangan yangi satr qaytadi.

### Butun sonlarni formatlash:

```
1 source = "{:d} belgilar"  
2 number = 5  
3 target = source.format(number)  
4 print(target) # 5 belgilar
```

Agar formatlanayotgan son 999 dan katta bo'lsa, sonning raqamlarni guruhlarga ajratish uchun verguldan foydalanamiz:

```
1 source = "{:,d} belgilar "  
2 print(source.format(5000)) # 5,000 belgilar
```

Haqiqiy son, yani, *float* turidagi sonlarning kasr qismidagi xonalar sonini aniq qilib belgilash uchun to'ldiruvchi oldidan, nuqtadan so'ng ularning sonini qo'yishimiz mumkin:

```
1 number = 23.8589578  
2 print("{:.2f}".format(number)) # 23.86  
3 print("{:.3f}".format(number)) # 23.859  
4 print("{:.4f}".format(number)) # 23.8590  
5 print("{:,.2f}".format(10001.23554)) # 10,001.24
```

Yana bir parametr harflardagi formatlangan qiymatning minimal kengligini belgilash imkonini beradi:

```
1 print("{:10.2f}".format(23.8589578)) # 23.86  
2 print("{:8d}".format(25)) # 25
```

Foizini ko'rsatish uchun "%" kodini ishlatish yaxshiroq:

```
1 number = .12345  
2 print("{:%}".format(number)) # 12.345000%  
3 print("{:.0%}".format(number)) # 12%  
4 print("{:.1%}".format(number)) # 12.3%
```

Ekspontensial belgilarda raqamni ko'rsatish uchun "e" to'ldirgichi ishlatiladi:

```
1 number = 12345.6789  
2 print("{:e}".format(number)) # 1.234568e+04  
3 print("{:.0e}".format(number)) # 1e+04  
4 print("{:.1e}".format(number)) # 1.2e+04
```

### *format()* metodisiz formatlash

Umuman olganda, *format()* metodidan foydalanmasdan ham satrlarni formatlash mumkin:

*satr*%(*paratmetr1*, *paratmetr 2*, *paratmetrN*)

bu erda formatlarnishi kerak bo'lgan satrda barcha to'ldiruvchilar (% belgili to'ldiruvchi bundan mustasno) figurali qavslarga olinmasdan yoziladi va satrdan keyin % belgisi qo'yiladi, undan so'ng qavs ichida mos argumentlar ketma-ketligi yoziladi. Foiz belgisi to'ldiruvchilar old qismida ko'rsatiladi:

```
1 info = "Ism: %s \t Yosh: %d" % ("Tom", 35)
2 print(info) # Ism: Tom      Yosh: 35
```

Formatlanayotgan satr va argumentlar ro'yxati orasidagi % belgisi – operator vazifasini bajaradi va natija sifatida formatlangan yangi sarni qaytaradi.

Bundan tashqari, raqamlarni formatlash usullari ham qo'llaniladi:

```
1 number = 23.8589578
2 print("%0.2f - %e" % (number, number)) # 23.86 -
3 #2.385896e+01
```

#### 4.4. So'zlarni sanash dasturi

Quyidagi dastur yordamida biror bir matn faylidagi so'zlar sonini hisoblash misol sifatida keltirilgan:

```
1  #!/ Dastur faylda so'zlarni sanaydi
2  import os
3
4
5  def get_words(filename):
6      with open(filename, encoding="utf8") as file:
7          text = file.read()
8          text = text.replace("\n", " ")
9          text = text.replace(",", "").replace(".",
10         "").replace("?", "").replace("!", "")
11         text = text.lower()
12         words = text.split()
13         words.sort()
14         return words
15
16
```

```

17 def get_words_dict(words):
18     words_dict = dict()
19
20     for word in words:
21         if word in words_dict:
22             words_dict[word] = words_dict[word] + 1
23         else:
24             words_dict[word] = 1
25     return words_dict
26
27
28 def main():
29     filename = input("Fayl yo'lini kiriting: ")
30     if not os.path.exists(filename):
31         print("Belgilangan fayl mavjud emas ")
32     else:
33         words = get_words(filename)
34         words_dict = get_words_dict(words)
35         print("So'zlar soni: %d" % len(words))
36         print("Noyob so'zlar soni: %d" % len(words_dict))
37         print("Barcha so'zlar ishlatilgan:")
38         for word in words_dict:
39             print(word.ljust(20), words_dict[word])
40
41 if __name__ == "__main__":
42     main()

```

Bu yerda, *get\_words()* funksiyasida, matni dastlabki bo'laklash (segmentirovka) amalga oshirilgan. Bunday holda barcha tinish belgilari o'chiriladi va bo'sh joylar bilan almashtiriladi. So'ngra matn so'zlarga bo'linadi. Kelishuv bo'yicha ajratuvchi sifatida bo'sh joydan foydalanilgan. So'ngra, *get\_words\_dict()* funksiyasi yordamida so'zlardan lug'at yaratilgan. Bu yerda kalit sifatida noyob (unikal) so'zlar olinadi, qiymat sifatida esa so'zning matndagi takrorlanishlari soni olinadi. *main()* funksiyada faylga yo'l ochiladi va yuqorida ko'rsatilgan funktsiyalarga qilingan murojaatlar bajariladi va barcha statistika ma'lumotlari olinadi.

## Dastur ishga tushirilganda konsolda chop qilingan natijalar:

1	Fayl yo'lini kiriting: C:\SomeFile.txt
2	So'zlar soni: 33
3	Noyob so'zlar soni: 30
4	Barcha so'zlar ishlatilgan:
5	agar 1
6	aniqlanadi 1
7	argumentga 1
8	argumentlar 1
9	ataladigan 1
10	bilan 2
11	bir 1
12	birinchi 1
13	bo'lgani 1
14	bo'lsa 1
15	bundan 1
16	deb 2
17	first_name 2
18	format() 1
19	ham 1
20	holatda 1
21	kabi 1
22	metodida 1
23	nom 1
24	nomlanadi 1
25	parametr 1
26	parametrlar 1
27	qilib 1
28	qiymati 1
29	satrdagi 1
30	shunday 1
31	tashqari 1
32	tayinlangan 1
33	unda 1
34	xil 1



## V. Asosiy ichki modullar

Python dasturlarda biz foydalanishimiz mumkin bo'lgan bir qator ichki modullarni taqdim etadi. Ulardan asosiylarini ko'rib chiqaylik.

### 5.1. *random* moduli

*random* moduli tasodifiy sonlarni generatsiya qilishni boshqaradi. Uning asosiy funksiyalari:

- `random()` – 0 dan 1 gacha tasodifiy sonni hosil qiladi;
- `randint()` – ma'lum bir oraliqdagi tasodifiy sonni hosil qiladi;
- `randrange()` – ma'lum sonlar to'plamidan tasodifiy sonni qaytaradi;
- `shuffle()` – ro'yxatni aralashtirib yuboradi;
- `choice()` – ro'yxatning tasodifiy elementini qaytaradi.

`random()` funksiyasi 0.0 dan 1.0 gacha bo'lgan tasodifiy suzuvchi nuqtali sonni qaytaradi. Agar bizga katta diapazondagi son zarur bo'lsa, masalan 0.0 dan 100.0 gacha, u holda tasodifiy funksiyaning natijasini 100 ga ko'paytiramiz.

```
1 import random
2
3 number = random.random() #qiymati 0.0 dan 1.0 gacha
4 print(number)
5 number = random.random() * 100 #qiymati 0.0 dan 100.0 gacha
6 print(number)
```

`randint(min, max)` funksiyasi *min* va *max* qiymatlari orasidagi tasodifiy sonni qaytaradi.

```
1 import random
2
3 number = random.randint(20, 35) #qiymati 20 dan 35 gacha
4 print(number)
```

`randrange()` funksiyasi ma'lum bir sonlar to'plamidan tasodifiy butun sonni qaytaradi. U quyidagi uchta shaklga ega:

- `randrange(stop)` – 0 dan *stop* gacha oraliqdagi tasodifiy butun sonni qaytaradi;

- *randrange(start, stop)* – to'plam *start* dan *stop* gacha oraliqdagi tasodifiy sonni qaytaradi;
- *randrange(start, stop, step)* – to'plam *start* dan *stop* gacha oraliqdan tasodifiy son qaytaradi, bunda oraliqdagi oldingi olingan tasodifiy sondan *step* qadamga farq qiladi.

```

1 import random
2
3 number = random.randrange(10) #qiymati 0 dan 10 gacha
4 print(number)
5 number = random.randrange(2, 10) #qiymati 2 dan 10 gacha
6 print(number)
7 number = random.randrange(2, 10, 2) # qiymat oralig'i 2,
8 #4, 6, 8, 10
9 print(number)

```

### Ro'yxatlar bilan ishlash

Ro'yxatlar bilan ishlash uchun *random* modulida ikkita funksiya aniqlangan: *shuffle()* funksiyasi ro'yxatni tasodifiy ravishda aralashtiradi; *choice()* funksiyasi ro'yxatdan tasodifiy bitta elementni qaytaradi:

```

1 numbers = [1, 2, 3, 4, 5, 6, 7, 8]
2 random.shuffle(numbers)
3 print(numbers) # 1
4 random_number = random.choice(numbers)
5 print(random_number)

```

### 5.2. *math* moduli

Ichki *math* moduli Pythonda matematik, trigonometrik va logarifimik amallarni bajaruvchi funksiyalarni o'zida jamlaydi. Quyida ulardan bir nechta sanab o'tilgan:

- *pow(num, power)* – *num* sonini *power* darajaga ko'taradi;
- *sqrt(num)* – *num* sonini kvadratik ildizga oladi;

- $\text{ceil}(num)$  – soni eng yaqin katta butun songacha yaxlitlaydi;
- $\text{floor}(num)$  – soni eng yaqin kichik butun songacha yaxlitlaydi;
- $\text{factorial}(num)$  – sonning faktorialini hisoblaydi;
- $\text{degrees}(rad)$  – radiandan gradusga o'tkazadi;
- $\text{radians}(grad)$  – gradusdan radianga o'tkazadi;
- $\text{cos}(rad)$  – radianda burchak kosinusini hisoblaydi;
- $\text{sin}(rad)$  – radianda burchak sinusini hisoblaydi;
- $\text{tan}(rad)$  – radianda burchak tangensini hisoblaydi;
- $\text{acos}(rad)$  – radianda burchak arkkosinusini hisoblaydi;
- $\text{asin}(rad)$  – radianda burchak arksinusini hisoblaydi;
- $\text{atan}(rad)$  – radianda burchak arktangensini hisoblaydi;
- $\text{log}(n, base)$  –  $base$  asosga ko'ra  $n$  ning logorifimini hisoblaydi;
- $\text{log10}(n)$  –  $n$  sonning o'nli logorifimini hisoblaydi.

Bir qancha funksiyalarni qo'lash bo'yicha misol:

```

1  import math
2
3  # 2 ni 3 darajasi
4  n1 = math.pow(2, 3)
5  print(n1)  # 8
6
7  # yuqoridagi amalni quyidagicha bajarsa ham bo'ladi
8  n2 = 2 ** 3
9  print(n2)
10
11 # kvadrat ildizga olish
12 print(math.sqrt(9))  # 3
13
14 # yaqin katta butun son
15 print(math.ceil(4.56))  # 5
16
17 # yaqin kichik son
18 print(math.floor(4.56))  # 4

```

```

19
20 # radiandan gradusga o'tish
21 print(math.degrees(3.14159)) # 180
22
23 # gradusdan radianga o'tish
24 print(math.radians(180)) # 3.1415.....
25 # kosinus
26 print(math.cos(math.radians(60))) # 0.5
27 # sinus
28 print(math.sin(math.radians(90))) # 1.0
29 # tangens
30 print(math.tan(math.radians(0))) # 0.0
31 #logarifm
32 print(math.log(8, 2)) # 3.0
33 print(math.log10(100)) # 2.0

```

*math* modulida bir qator o'zgarmas sonlar ham aniqlangan, *PI* va *E*:

```

1 import math
2
3 radius = 30
4 # 30 radiusli doira yuzasi
5 area = math.pi * math.pow(radius, 2)
6 print(area)
7
8 # 10 sonning natural logarifmi
9 number = math.log(10, math.e)
10 print(number)

```

### 5.3. locale moduli

Pythonda sonlarni formatlashda kelishuv bo'yicha "Angliya-Sakson" tizimidan foydalanadi. Bu tizim sonlarning razryadlarini (uchta alohida bo'laklarga ajratilgandagi bo'laklarni) bir – biridan vergul bilan, butun va kasr qismni esa nuqta bilan ajratadi. Masalan, Yevropa qit'asida boshqa tizimdan foydalanishadi.

Bu tizimda esa razryadlar nuqta bilan, butun va kasr qism esa vergul bilan ajratiladi:

```
1 # Angliya-Saxon tizimi
2 1,234.567
3 # Yevropa tizimi
4 1.234,567
```

Sonlarni formatlashda belgilangan tartibni aniqlash muammosini yechish uchun Pythonda *locale* moduli aniqlangan. Mahalliy tartibni o'rnatish uchun *locale* modulida *setlocale()* funksiyasi aniqlangan. U ikkita parametr qabul qiladi:

```
1 import locale;
2 locale.setlocale(category, locale)
```

Birinchi parametr funksiyada sonmi, valyutami yoki valyuta-sonmi qo'llanishini ko'rsatadi. Birinchi parametr sifatida quydagilardan birini berishimiz mumkin:

- LC\_ALL – hamma kategoriyalar bo'yicha mahalliyashtirishni ta'minlaydi;
- LC\_NUMERIC – sonlarni mahalliyashtirish;
- LC\_MONETARY – valyutani mahalliyashtirish;
- LC\_TIME – sana va vaqtni mahalliyashtirish;
- LC\_CTYPE – belgilarni yuqori yoki quyi registrga o'tkazishni mahalliyashtirish;
- LC\_COLLATE – satrlarni solishtirishni mahalliyashtirish.

*setlocale()* funksiyaning ikkinchi parametri foydalanish zarur bo'lgan mahalliy tartibni ko'rsatadi. Windows OS da ikkita belgidan iborat ISO bo'yicha kodni ishlatish mumkin, masalan, "us" – AQSH, "de" – Germaniya, "ru" – Rossiya va "uz" – O'zbekiston uchun. Lekin, MacOS da esa davlat kodi va til kodini ko'rsatish zarur, masalan, "us-US" – AQSH, "de-DE" – Germaniya, "ru-RU" – Rossiya va "uz-UZ" – O'zbekiston uchun. Kelishuv bo'yicha "en-US" ishlatiladi.

Bevosita, sonlarni va valyutalarni formatlash uchun *locale* moduli ikkita funksiya taqdim etadi:

- `currency(num)` – valyutani formatlaydi;
- `format(str, num)` – `num` sonnini `str` satridagi to'ldiruvchi o'rniga qo'yadi;

Quyidagi to'ldiruvchilardan foydalaniladi:

- `d` – butun sonlar uchun;
- `f` – suzuvchi nuqtali sonlar uchun;
- `e` – eksponentsial sonlarni yozish uchun.

Har bir to'ldiruvchi oldidan `%` foiz belgisini qo'yiladi, masalan:

1	<code>"%d"</code>
---	-------------------

Sonning kasr qismini chiqarishda to'ldiruvchi oldidan, nuqtadan so'ng nechta kasr qismda raqam aks etishini ko'rsatish mumkin:

1	<code>%.2f # kasr qismida ikkita raqam</code>
---	---

Sonlarni va valyutani mahalliyashtirishning o'zbek tili uchun qo'llanilishi:

1	<code>import locale</code>
2	
3	<code>locale.setlocale(locale.LC_ALL, "uz") # Windows uchun</code>
4	<code># locale.setlocale(locale.LC_ALL, "uz_UZ") # MacOS uchun</code>
5	
6	<code>number = 12345.6789</code>
7	<code>formatted = locale.format("%f", number)</code>
8	<code>print(formatted) # 12345,678900</code>
9	
10	<code>formatted = locale.format("%.2f", number)</code>
11	<code>print(formatted) # 12345,68</code>
12	
13	<code>formatted = locale.format("%d", number)</code>
14	<code>print(formatted) # 12345</code>
15	
16	<code>formatted = locale.format("%e", number)</code>
17	<code>print(formatted) # 1,234568e+04</code>
18	

```

19 money = 234.678
20 formatted = locale.currency(money)
21 print(formatted) # 234,68 so'm

```

Agarda aniq kodning o'rniga ikkinchi parametrda bo'sh satr uzatilsa, u holda Python joriy ishchi mashinadagi tartibni oladi. *getlocale()* funksiyasi yordamida joriy tartibni olish mumkin:

```

1 import locale
2
3 locale.setlocale(locale.LC_ALL, "")
4
5 number = 12345.6789
6 formatted = locale.format("%.02f", number)
7 print(formatted) # 12345,68
8 print(locale.getlocale()) # ('Russian_Russia', '1251')

```

#### 5.4. decimal moduli

Suzuvchi nuqtali sonlar bilan ishlashda hisoblash natijasining to'g'iri emasligiga duch kelamiz:

```

1 number = 0.1 + 0.1 + 0.1
2 print(number) # 0.30000000000000004

```

*round()* funksiyasini sonni yaxlitlash qo'llash yordamida bu muammoni yechishi mumkin. Bundan tashqari *decimal* ichki modulini ishlatish orqali ham bu muammoni yechish mumkin.

Bu moduldagi *Decimal* sinfi asosiy sonlar bilan ishlash komponentasi hisoblanadi. Bu sinfni qo'llash uchun *Decimal* konstruktori yordamida uning obyektini yaratish zarur. Konstruktorga argument sifatida sonning satrdagi ifodasi uzatiladi:

```

1 from decimal import Decimal
2
3 number = Decimal("0.1")

```

Bundan so'ng *Decimal* obyektini arifmetik amallarda ishlatish mumkin:

```
1 from decimal import Decimal
2
3 number = Decimal("0.1")
4 number = number + number + number
5 print(number) # 0.3
```

*Decimal* bilan amallarda butun sonlarni ham ishlatish mumkin:

```
1 number = Decimal("0.1")
2 number = number + 2
```

Ammo, kasrli amalarda *float* va *Decimal* aralashtirmaslik zarur:

```
1 number = Decimal("0.1")
2 number = number + 0.1 #bu yerda xatolik yuz beradi
```

Sonning kasr qismdagi raqamlar nechta bo'lishini quyidagicha aniqlash mumkin:

```
1 number = Decimal("0.10")
2 number = 3 * number
3 print(number) # 0.30
```

"0.10" satr sonning kasr qismida ikkita belgi bo'lishini ko'rsatadi, agarda oxirgi belgilar nol bo'lsa ham. Mos ravishda "0.100" satr sonning kasr qismida 3 belgi bo'lishini anglatadi.

### Sonlarni yaxlitlash

*Decimal* obyektlari sonlarni yaxlitlaydigan *quantize()* metodiga ega. Bu metodning birinchi argumenti sifatida sonning yaxlitlash formatini ko'rsatadigan *Decimal* obyekti uzatiladi:

```
1 from decimal import Decimal
2
3 number = Decimal("0.444")
4 number = number.quantize(Decimal("1.00"))
5 print(number) # 0.44
6
```



```

7 number = Decimal("0.555678")
8 print(number.quantize(Decimal("1.00"))) # 0.56
9
10 number = Decimal("0.999")
11 print(number.quantize(Decimal("1.00"))) # 1.00

```

Foydalanilayotgan "1.00" satr sonning kasr qismidagi belgilar soni ikkita belgigacha yaxlitlanishini ko'rsatadi.

Kelishuv bo'yicha *ROUND\_HALF\_EVEN* o'zgarmasi soni yuqori tomonga yaxlitlashni anglatadi, agarda son juft bo'lmasa va keyingisi 4 dan katta bo'lsa, masalan:

```

1 from decimal import Decimal, ROUND_HALF_EVEN
2
3 number = Decimal("10.025")
4 print(number.quantize(Decimal("1.00"), ROUND_HALF_EVEN)) #
5 10.02
6
7 number = Decimal("10.035")
8 print(number.quantize(Decimal("1.00"), ROUND_HALF_EVEN)) #
9 10.04

```

Bu erda yaxlitlash tartibi *quantize()* funksiyasiga ikkinchi parametr sifatida uzatilgan. "1.00" satri sonning kasr qismini yaxlitlash ikkita belgigacha bo'lishini anglatadi. Lekin, birinchi holda, "10.025" kasr qismidagi ikkinchi son-2 juft, shuning uchun, keyingi son 5 bo'lishiga qaramasdan, ikki uchga yaxlitlanmayapti. Ikkinchi holda, "10.025" ikkinchi son-3 juft emas, shuning uchun ham 4 ga yaxlitlanayapti.

Bunday yaxlitlashdagi tartib har doim ham foydali bo'lmasligi mumkin, shu sababli ham, quyidagi o'zgamaslardan foydalanish orqali tartibni qayta aniqlash mumkin:

- *ROUND\_HALF\_UP* – soni yuqori tomonga yaxlitlaydi, agarda undan keyingi son 5 yoki 5 dan katta bo'lsa;

- *ROUND\_HALF\_DOWN* – soni yuqori tomonga yaxlitlaydi, agarda undan keyingi son 5 dan katta bo'lsa:

```

1 number = Decimal("10.026")
2 print(number.quantize(Decimal("1.00"), ROUND_HALF_DOWN))
3 #10.03
4 number = Decimal("10.025")
5 print(number.quantize(Decimal("1.00"), ROUND_HALF_DOWN))
6 # 10.02

```

- *ROUND\_05UP* – faqat 0 ni birga yaxlitlaydi, agarda undan keyin 5 kelsa:

```

1 number = Decimal("10.005")
2 print(number.quantize(Decimal("1.00"), ROUND_05UP)) # 10.01
3
4 number = Decimal("10.025")
5 print(number.quantize(Decimal("1.00"), ROUND_05UP)) # 10.02

```

- *ROUND\_CEILING* – keyingi son qanday bo'lishidan qa'tiy nazar soni yuqori tomonga yaxlitlaydi:

```

1 number = Decimal("10.021")
2 print(number.quantize(Decimal("1.00"), ROUND_CEILING))
3 # 10.03
4 number = Decimal("10.025")
5 print(number.quantize(Decimal("1.00"), ROUND_CEILING))
6 # 10.03

```

- *ROUND\_FLOOR* – keyingi son qanday bo'lishidan qa'tiy nazar soni yaxlitlamaydi:

```

1 number = Decimal("10.021")
2 print(number.quantize(Decimal("1.00"), ROUND_FLOOR)) # 10.02
3 number = Decimal("10.025")
4
5 print(number.quantize(Decimal("1.00"), ROUND_FLOOR)) # 10.02

```

## VI. Obyektga yo'naltirilgan dasturlash

### 6.1. Sinf va obyekt

Pythonda obyektga yo'naltirilgan dasturlash tamoyilini ham ishlatish mumkin bo'lib, bu o'z navbatida dastur komponentlarini sinflar ko'rinishida ifodalash imkonini beradi.

Sinf obyektning shabloni yoki formal tavsifi hisoblanadi. Obyekt esa ushbu sinfning haqiqiy shaklangan nusxasi (экземпляри) hisoblanadi. Quyidagicha tatbiq qilish mumkin: hammadam inson to'g'risida qandaydir tassavvur mavjud – ikkita qo'li, ikkita oyog'i, boshi, oshqozoni, asab tizimi va boshqalari mavjud. Demak, shablon bor – bu shablond sinf deb atash mumkin. Haqiqatdan ham, mavjud bo'lgan odamni bu sinfning obyektini deyish mumkin.

Kod nuqtai nazaridan sinf – funksiyalar va o'zgaruvchilar to'plamini ma'lum bir vazifalarni bajarish uchun o'zida birlashtiradi. Sinfning funksiyalari odatda metodlar deb ataladi. Ular sinfning xususiyatlarini ifodalaydi. Sinfning o'zgaruvchilari esa atributlar deb nomlanadi – ular sinfning holatini saqlaydi.

Sinf *class* kalit so'zi bilan aniqlanadi:

```
1 class sinf_nomi:
2     sinf_metodilari
```

Sinf obyektini yaratish uchun quyidagi sintaksis ishlatiladi:

```
1 obyekt_nomi = sinf_nomi([parametrlar])
```

Masalan, insonni ta'riflaydigan *Person* sinfini aniqlaymiz:

```
1 class Person:
2     name = "Ali"
3
4     def display_info(self):
5         print("Salom, mening ismim", self.name)
6
7
8 person1 = Person()
```

```

9 person1.display_info() # Salom, mening ismim Ali
10
11 person2 = Person()
12 person2.name = "Salim"
13 person2.display_info() # Salom, mening ismim Salim

```

*Person* sinfida insonning ismini saqlovchi *name* atributi va inson haqida ma'lumot chiqaradigan *display\_info* metodlari aniqlangan.

Ixtiyoriy sinfning metodi aniqlayotganda hamma metodlarning birinchi parametri sifatida obyektning o'ziga ko'rsatgich bo'lgan *self* (bir qator dasturlash tillarida parametrning analogi sifatida *this* kalit so'zi ishlatiladi) parametri bo'lishini hisobga olish kerak. Ushbu ko'rsatgich yordamida sinfning atriburlariga va metodlariga murojaat qilishimiz mumkin. Xususiyl holda, *self.name* ifodasi orqali foydalanuvchi ismini olish mumkin.

*Person* sinfini aniqlagandan so'ng, *person1* va *person2* uning ikkita obyektini yaratamiz. Obyekt nomlaridan foydalanish orqali uning metodlariga va atributlariga murojaat qilishimiz mumkin. Ushbu holda, qora oynaga sartni chiqaruvchi *display\_info()* metodini har bir obyektidan chaqiramiz va ikkinchi obyektning *name* atributini o'zgartiramiz. *display\_info* metodiga murojaatda *self* parametiriga hech qanday qiymat berish shart emas.

## 6.2. Kostruktorelar

Sinf obyektlarini yaratish uchun konstruktorelardan foydalaniladi. Yuqorida aniqlangan *Person* sinfida esa kelishuv bo'yicha konstruktorendan foydalandik.

```

1 person1 = Person()
2 person2 = Person()

```

Shu bilan birga, sinf ichida `__init__` deb nomlanuvchi maxsus metod yordamida konstruktorelarni oshkor ravishda aniqlash ham mumkin. Misol uchun, *Person* sinfiga konstruktore qo'shamiz.

```

1 class Person:
2     #Konstruktore
3     def __init__(self, name):
4         self.name = name # ismni name atributiga beramiz

```

```

5
6     def display_info(self):
7         print("Salom, mening ismim", self.name)
8
9     person1 = Person("Ali")
10    person1.display_info() # Salom, mening ismim Ali
11
12    person2 = Person("Salim")
13    person2.display_info() # Salom, mening ismim Salim

```

Konstruktorning birinchi parametr sifatida joriy obyektga *self* ko'rsatgichi bo'ladi. Ko'pincha, konstruktorga atributlar o'rnatiladi. Ushbu holda ham, *self.name* atributi uchun o'rnatiladigan foydalanuvchining ismi ikkinchi parametr sifatida konstruktorga uzatilyapti. *Person* sinfining *name* atributini sinfnining oldingi ko'rinishidagi kabi oshkor aniqlash shart emas. *self.name = name* atributiga qiymat berishning o'zida *Person* sinfida oshkormas *name* atriburini aniqlaydi.

```

1    person1 = Person("Ali")
2    person2 = Person("Salim")

```

Natijada quyidagiga ega bo'lamiz:

```

1    Salom, mening ismim Ali
2    Salom, mening ismim Salim

```

### 6.3. Destruktor

Obyekt bilan ishlash tugagandan so'ng *del* operatorini ishlatish orqali uni xotiradan o'chirib tashlash mumkin:

```

1    person1 = Person("ALi")
2    del person1 # Xotiradan o'chirish
3    # person1.display_info() person1 xotirdan o'chgani
4    #sababli bu metod ishlaymaydi.

```

Shuni eslatish joizki, bu ishni qilish shart emas, ya'ni, skrip ishlashi tugashi bilan ham obyektlar xotiradan o'chiriladi.

Bundan tashqari, `__del__` ichki metodini aniqlash orqali sinfdagi destruktorni qayta aniqlash mumkin. Bu metod `del` operatori ishlaganda yoki obyektlar xotiradan avtomatik o'chganda ishlaydi. Masalan:

```
1 class Person:
2     # Konstruktor
3     def __init__(self, name):
4         self.name = name # Ismni o'rnatamiz
5
6     def __del__(self):
7         print(self.name, "Xotiradan o'chdi")
8
9     def display_info(self):
10        print("Salom, mening ismim ", self.name)
11
12
13 person1 = Person("Ali")
14 person1.display_info() # Salom, mening ismim Ali
15 del person1 # Xotiradan o'chirish
16 person2 = Person("Salim")
17 person2.display_info() # Salom, mening ismim Salim
```

Natija:

```
1 Salom, mening ismim Ali
2 Ali Xotiradan o'chdi
3 Salom, mening ismim Salim
4 Salim Xotiradan o'chdi
```

#### 6.4. Sinflarni modullarda aniqlash va ularni bog'lash

Qoidaga ko'ra, sinflar alohida modullarda joylashadi va asosiy skriptda ular import qilinadi. Aytaylik bitta loyihada ikkita fayli mavjud: `main.py` (dasturning asosiy skripti) va `classes.py` (sinflar aniqlangan skript).

`classes.py` faylida ikkita sinf aniqlaymiz:

```

1  class Person:
2      # konstruktor
3      def __init__(self, name):
4          self.name = name # nomni o'rnatamiz
5
6      def display_info(self):
7          print("Salom, ismim", self.name)
8
9
10 class Auto:
11     def __init__(self, name):
12         self.name = name
13
14     def move(self, speed):
15         print(self.name, "tezlik bilan harakatlanayapti ",
16 speed, "km/s")

```

*Person* sinfiga qo'shimcha ravishda avtomobilni harakterlovchi *move* va *name* atributi bor *Auto* sinfi aniqlangan. Bu sinflarga bog'lanamiz va *main.py* asosiy dastur skriptida foydalanamiz:

```

1  from classes import Person, Auto
2
3  ali = Person("Ali")
4  ali.display_info()
5
6  bmw = Auto("Malibu")
7  bmw.move(65)

```

Sinflarga bog'lanish ham moduldan funksiyani importi kabi amalga oshiriladi. Yoki to'liq modulga ham bog'lanish mumkin:

```

1  Import classes

```

Natijada quyidagini olamiz:

```

1  Salom, ismim Ali
2  Malibu tezlik bilan harakatlanayapti 65 km/s

```

## 6.5. Inkapsulyatsiya

Kelishuv bo'yicha sinflardagi atributlar umumiy ruxsatga ega bo'ladi, ya'ni, dasturning ixtiyoriy joyidan obyektning atributlariga ruxsat olish va ularni o'zgartirish mumkin. Masalan:

```
1 class Person:
2     def __init__(self, name):
3         self.name = name # ismini o'rnatish
4         self.age = 1 # yoshni o'rnatish
5
6     def display_info(self):
7         print("Ism:", self.name, "\tYosh:", self.age)
8
9
10 ali = Person("Ali")
11 ali.name = "O'rgamchak odam" # name atributini
12 o'zgartirish
13 ali.age = -129 # age atributini o'zgartirish
14 ali.display_info() # Ism: O'rgamchik odam      Yosh: -129
```

Lekin ushbu holda, misol uchun, yosh yoki odamning ismini noto'g'ri o'zgartirish mumkin, masalan, yuqoridagi kabi yoshiga manfiy son yozish. Ushbu holda obyekt atributiga murojaatni nazorat qilish haqida savol paydo bo'ladi.

Bunday muammo inkapsulyatsiya tushunchasi bilan chambarchas bog'liq. *Inkapsulyatsiya* obyektga yo'naltirilgan dasturlashning fundamental tushunchalaridan biri hisoblanadi. Kodning murojaat joyidan obyekt atributiga to'g'ridan-to'g'ri murojaat qilishni chegaralaydi.

Python dasturlash tilida sinf atributlarini inkapsulyatsiya yordamida ko'rinmas yoki yopiq va chegaralangan murojaatni maxsus metodlar orqali o'rnatish mumkin. Odatda ular, xususiyat deb ham ataladi.

Yuqoridagi aniqlangan sinfga quyidagicha xususiyatlar qo'shamiz:

```
1 class Person:
2     def __init__(self, name):
3         self.__name = name # ismini o'rnatish
```



```

4         self.__age = 1 # yoshni o'rnatish
5
6     def set_age(self, age):
7         if age in range(1, 100):
8             self.__age = age
9         else:
10            print("Mumkin bo'lmagan yosh")
11
12    def get_age(self):
13        return self.__age
14
15    def get_name(self):
16        return self.__name
17
18    def display_info(self):
19        print("Ism:", self.__name, "\tYosh:", self.__age)
20
21
22 ali = Person("Ali")
23
24 ali.__age = 43 # age atributi o'zgamaydi
25 ali.display_info() # Ism: Ali Yosh: 1
26 ali.set_age(-3486) # Mumkin bo'lmagan yosh
27 ali.set_age(25)
28 ali.display_info() # Ism: Ali Yosh: 25

```

Yopiq atribut yaratish uchun atribut nomi oldidan ikkita tag chiziq qo'yish lozim: `self.__name`. Bunday atributlarga faqat ushbu sinfning ichidagina murojaat qilish mumkin, lekin, sinfdan tashqarida murojaat qilib bo'lmaydi. Masalan, atributga qiymat yozib bo'lmaydi:

```

1 ali.__age = 43

```

Uning qiymatini olishga urinsak xatolik yuz beradi:

```
1 print(ali.__age)
```

Lekin `__age` atributinga sinf tashqarisida qiymat berish kerak bo'lishi mumkin. Buning uchun xususiyat yaratiladi va u orqali atribut qiymatiga murojaat amalga oshiriladi. Masalan quyida `get_age()` metodi orqali `__age` atributi qiymatini sinf tashqarisida olish mumkin:

```
1 def get_age(self):  
2     return self.__age
```

Ushbu metod getter yoki aksessor deb ham nomlanadi.

Qiymatni o'zgartirish uchun esa ya'na boshqa xususiyat aniqlanadi:

```
1 def set_age(self, value):  
2     if value in range(1, 100):  
4         self.__age = value  
5     else:  
6         print("Mumkin bo'lmagan yosh")
```

Bu yerda esa biz kelayotgan qiymatni shartga tekshirish mumkin bo'ladi. Bu metod setter yoki myuteytor deb ham nomlanadi.

Har bir yopiq atribut uchun bir juft xususiyat aniqlash shart emas. Yuqoridagi misolda faqat konstruktor yordamida qiymatlarni berish mumkin. Qiymatni olish uchun `get_name` metodidan foydalaniladi.

### Xususiyatlar annotatsiyasi

Yuqorida qanday qilib xususiyatlarni yaratishni ko'rib chiqdik. Pythonda ya'nada qulayroq usul mavjud. `@` belgisi bilan boshlanuvchi annotatsiyadan foydalanish orqali bu usul amalga oshiriladi.

Getter-xususiyat yaratish uchun `@property` annotatsiyasini qo'yish lozim. Setter-xususiyat qo'yish uchun esa `@setter_xususiyatnomi.setter` annotatsiyasini o'rnatish lozim.

*Person* sinfini annotatsiyadan foydalanib qayta yozamiz:

```
1 class Person:  
2     def __init__(self, name):  
4         self.__name = name # ismni o'rnatamiz  
5         self.__age = 1 # yoshni o'rnatamiz
```

```

6
7     @property
8     def age(self):
9         return self.__age
10
11    @age.setter
12    def age(self, age):
13        if age in range(1, 100):
14            self.__age = age
15        else:
16            print("Mumkin bo'lmagan yosh")
17
18    @property
19    def name(self):
20        return self.__name
21
22    def display_info(self):
23        print("Ism:", self.__name, "\tYosh:", self.__age)
24
25    ali = Person("Ali")
26
27    ali.display_info() # Ism: Ali Yosh: 1
28    ali.age = -3486 # Mumkin bo'lmagan yosh
29    print(ali.age) # 1
30    ali.age = 36
31    ali.display_info() # Ism: Ali Yosh: 36
32

```

Birinchi navbatda, setter-xususiyati getter-xususiyatidan so'ng aniqlanishiga e'tibor qaratish zarur. Ikkinchidan, setter va getter *age* bir xil nomlangan. Shuning uchun ham, getter *age* nomlangan, shu sababdan ham setterga `@age.setter` annotatsiyasi o'rnatilgan.

Natijada, getter va setter *ali.age* ifodasi bilan murojaat qilish mumkin bo'ladi.

## 6.6. Vorislik

Vorislik avvaldan mavjud sinf asosida yangi sinf imkonini beradi. Vorislik inkapsulyatsiya bilan bir qatorda obyektga yo'naltirilgan dasturlashning asosini tashkil qiladi. Vorislikning asosiy tushunchalari bu: `super_sinf` va `sinf_osti` lar hisoblanib, `sinf_osti` sinfi `super_sinf`dan hamma ochiq atributlar va metodlarni voris qilib oladi. `Super_sinf` ko'pincha asos yoki ota sinf deb ham nomlanadi, `sinf_osti` esa hosilaviy sinf, voris sinf yoki bola sinf ham deb nomlanadi.

Sinflar vorisliklari uchun sintaksis quyidagi ko'rinishga ega bo'ladi:

```
1 class sinfostisi (supersinf):
2     sinfostisi_azolari
```

O'tgan darslarda insonni tasvirlaydigan *Person* sinfini yaratgandik. Korxonada ishlovchi ishchini tasvirlaydigan sinfni yaratish zarur deb qaraylik. Buning uchun yangi *Employee* sinfni noldan yaratishimiz mumkin. Biroq, *Employee* sinfining atributlari va metodlari ham *Person* sinfidagilar kabi bo'ladi, chunki ishchi ham inson. Shuning uchun ham *Employee* sinfini qaytadan yaratishda ma'no qolmaydi. Ushbu holda vorislikni ishlatish qulay hisoblanadi.

*Person* sinfidan *Employee* sinfiga vorislik olamiz:

```
1 class Person:
2     def __init__(self, name, age):
3         self.__name = name # ismni o'rnatamiz
4         self.__age = age # yoshni o'rnatamiz
5
6     @property
7     def age(self):
8         return self.__age
9
10    @age.setter
11    def age(self, age):
12        if age in range(1, 100):
13            self.__age = age
14        else:
15            print("Mumkin bo'lmagan yosh")
```

```

16
17     @property
18     def name(self):
19         return self.__name
20
21     def display_info(self):
22         print("Ism:", self.__name, "\tYosh:", self.__age)
23
24
25 class Employee(Person):
26     def details(self, company):
27         print(self.name, company, " kompaniyasida ishlaydi
28 ")
29
30 ali = Employee("Ali", 23)
31 ali.details("Google")
32 ali.age = 33
33 ali.display_info()
34

```

*Employee* sinfi to'lig'icha *Person* sinfnig funksionlarini qabul qiladi va qo'shimcha ravishda *details()* metodi qo'shildi.

Shuni ta'kidlash kerakki, *Employee* sinfi uchun *Person* sinfidagi `__name` yoki `__age` turidagi yopiq atributlardan tashqari barcha metodlar va atributlar *self* kalit so'z orqali murojaat mavjud. *Employee* obyektini yaratishda biz aslida *Person* sinfining konstruktoridan foydalanamiz. Bundan tashqari, ushbu obyektida *Person* sinfining barcha metodlarini chaqirishimiz mumkin.

## 6.7. Polimorfizm

Polimorfizm obyektga yo'naltirilgan dasturlashning yana bir tayanch tushunchalaridan biri bo'lib, bazaviy sinfdan voris qilib olingan metodlar funksional vazifalarini o'zgartirish imkoniyatini beradi.

Masalan, quyidagi sinflar ierarxiyasini olaylik:

```

1 class Person:

```

```

2     def __init__(self, name, age):
3         self.__name = name # ismni o'rnatamiz
4         self.__age = age # yoshni o'rnatamiz
5
6     @property
7     def name(self):
8         return self.__name
9
10    @property
11    def age(self):
12        return self.__age
13
14    @age.setter
15    def age(self, age):
16        if age in range(1, 100):
17            self.__age = age
18        else:
19            print("Mumkin bo'lmagan yosh")
20
21    def display_info(self):
22        print("Ism:", self.__name, "\tYosh:", self.__age)
23
24    class Employee(Person):
25        def __init__(self, name, age, company):
26            Person.__init__(self, name, age)
27            self.company = company
28
29        def display_info(self):
30            Person.display_info(self)
31            print("Kompaniya:", self.company)
32
33    class Student(Person):
34        def __init__(self, name, age, university):
35            Person.__init__(self, name, age)
36            self.university = university

```

```

37
38     def display_info(self):
39         print("Talaba ", self.name, " universitetda
40 o'qiydi", self.university)
41
42 people = [Person("Ali", 23), Student("Salim", 19,
43 "Milliy"), Employee("Karim", 35, "Google")]
44
45 for person in people:
46     person.display_info()
47     print()

```

Ishchini ifodalovchi *Employee* voris sinfi ichida, o'zining konstruktori aniqlangan. Obyektni yaratishda ishchi faoliyat yuritayotgan kompaniya nomini ham kiritish kerak. Buning uchun konstruktor to'rtta parametrli qilib yaratish lozim: *self* standart parametrining o'zi, *ism* va *yosh* parametrlari va *kompaniya* parametrlari.

*Employee* sinfi konstruktoring o'zida *Person* asos sinf (bazaviy sinf, tayanch sinf) konstruktori chaqiriladi . Asos sinf metodlariga murojaat quyidagi ko'rinishdagi sintaksisga ega:

```

1     supersinf.sinfnomi(self [, parametrlar])

```

Shuning uchun bazaviy sinf konstruktoriga *yosh* va *ism* argumentlari uzatilayapti. *Employee* sinfiga esa *Person* sinfidan tashqari *self.company* atributi qo'shilyapti. Bundan tashqari, *Employee* sinfida *Person* sinfidagi *display\_info()* metodi qayta aniqlanayapti. Bu qayta aniqlangan metod ishchining yoshi va ismidan tashqari ishlaydigan kompaniyasi nomini ham chop qiladi. Yosh va ismni chiqarish kodini qayta yozmaslik uchun qayta aniqlangan metodda *Person* sinfi *display\_info()* metodi chaqirilmoqda.

Shu tarzda talabani ifodalovchi *Student* sinfi ham aniqlanmoqda. Unda ham konstruktor va *display\_info()* metodi qayta aniqlanmoqda, faqat, bu metodda bazaviy sinf *display\_info()* metodi chaqirilmagan.

Dasturning asosiy qismida *Person* sinfining 3 ta obyektlaridan iborat bo'lgan ro'yxat shakllantirilgan bo'lib, ulardan ikkitasi *Student* va *Employee* sinfi obyektlari

hisoblanadi. Takrorlashda har bir obyektning `display_info()` metodiga murojaat qilingan. Python dastur bajarilish davomida sinflar shajarasini hisobga olib, har bir obyekt uchun `display_info()` metodining kerakli versiyasini chaqiradi. Natijada quyidagini olamiz:

```
1  Ism: Ali      Yosh: 23
2
3  Talaba Salim universitetda o'qiydi Milliy
4
5  Ism: Karim   Yosh: 35
6  Kompaniya: Google
```

### 6.8. Obyektlarni turlarga tekshirish

Obyektlar bilan ishlash davomida obyektning turlariga bog'liq ravishda u yoki bu amalni bajarish zarurati paydo bo'ladi. `isinstance()` ichki metodi yordamida obyektlarni turlarga tekshirish mumkin. Bu funksiya ikkita parametr qabul qiladi:

```
1  isinstance(object, type)
```

Birinchi parametr obyektни ko'rsatadi, ikkinchi parametr esa obyekt solishtirilayotgan turni ko'rsatadi. Agarda obyekt ushbu `type` turiga tegishli bo'lsa, u holda funksiya `True` qiymat, aks holda `False` qiymat qaytaradi. Masalan, yuqorida aniqlangan sinflar shajarasini qaraymiz:

```
1  for person in people:
2      if isinstance(person, Student):
3          print(person.university)
4      elif isinstance(person, Employee):
5          print(person.company)
6      else:
7          print(person.name)
8      print()
```

### 6.9. object sinfi. Obyektни satr ko'rinishida tasvirlanishi

Pythonning uchinchi versiyasidan boshlab hamma sinflar umumiy bazaviy `object` sinfining metodlarini voris qilib olashi ta'minlandi.



*object* sinfining eng ko'p ishlatiladigan metodlaridan bir `__str__()` bo'lib, obyektning satr ko'rinishida chiqarish uchun chaqiladi. Sinflarni aniqlashda ushbu metodni qayta aniqlash muhim amaliy ahamiyatga egadir.

Misol uchun, *Person* sinfini olamiz va uning satr ko'rinishini chiqaramiz:

```
1 class Person:
2     def __init__(self, name, age):
3         self.__name = name # ismni o'rnatamiz
4         self.__age = age # yoshni o'rnatamiz
5
6     @property
7     def name(self):
8         return self.__name
9
10    @property
11    def age(self):
12        return self.__age
13
14    @age.setter
15    def age(self, age):
16        if age in range(1, 100):
17            self.__age = age
18        else:
19            print("Mumkin bo'lmagan yosh")
20
21    def display_info(self):
22        print("Ism:", self.__name, "\tYosh:", self.__age)
23
24 ali = Person("Ali", 23)
25 print(ali)
```

Dastur ishlashi natijasida quyidagilarni olamiz:

```
1 <__main__.Person object at 0x000000C1913BDD30>
```

Bu natija obyekt haqida qoniqarli ma'lumot emas. Shuning uchun `__str__()` metodini qayta aniqlaymiz:

```

1 class Person:
2     def __init__(self, name, age):
3         self.__name = name # ismni o'rnatamiz
4         self.__age = age # yoshni o'rnatamiz
5
6     @property
7     def name(self):
8         return self.__name
9
10    @property
11    def age(self):
12        return self.__age
13
14    @age.setter
15    def age(self, age):
16        if age in range(1, 100):
17            self.__age = age
18        else:
19            print("Mumkin bo'lmagan yosh")
20
21
22    def display_info(self):
23        print("Ismi:", self.__name, "\tYoshi:", self.__age)
24
25    def __str__(self):
26        return "Ismi: {} \t Yoshi: {}".format(self.__name,
27 self.__age)
28
29 ali = Person("Ali", 23)
30 print(ali)

```

Ushbu xolatda `__str__()` metodi inson haqidagi bazaviy ma'lumotlarni o'zida aks ettiruvchi satr qaytaradi. Endi natija quyidagicha ko'rinish oladi:

```
1 Ism: Ali      Yosh: 23
```

## VII. Sana va vaqt bilan ishlash

### 7.1. *datetime* moduli

Sana va vaqt bilan ishlaydigan quyidagi asosiy funksiyalar *datetime* modulida jamlangan:

- `date`;
- `time`;
- `datetime`.

#### *date* sinfi

Sana bilan ishlash uchun *datetime* modulida aniqlangan *date* sinfidan foydalaniladi. *date* sinfi obyektini yaratish uchun uchta parameter: yil, oy va kun parametrlarini qabul qiladigan *date* sinfi konstruktoridan foydalaniladi:

```
1 date(year, month, day)
```

Masalan, qandaydir sana yaratamiz:

```
1 import datetime
2
3 yesterday = datetime.date(2017, 5, 2)
4 print(yesterday) # 2017-05-02
```

*today()* metodidan foydalanish orqali joriy sanani olish mumkin:

```
1 from datetime import date
2
3 today = date.today()
4 print(today) # 2017-05-03
5 print("{}.{}.{}".format(today.day, today.month,
6 today.year)) # 2.5.2017
```

*day*, *month* va *year* xususiyatlari orqali mos ravishda kun, oy va sanani olish mumkin.

#### *time* sinfi

Vaqt bilan ishlash uchun *time* sinfidan quydagicha foydalaniladi:

```
1 time([hour] [, min] [, sec] [, microsec])
```

Konstruktor soat, minut, sekund va mikro sekundlarni ketma-ket ravishda qabul qiladi va bu parametrlar shart bo'lmagan parametrlardir. Agarda birorta parameter konstruktorda berilmasa, u holda kelishuv bo'yicha nol qiymat olinadi.

```
1 from datetime import time
2
3 current_time = time()
4 print(current_time) # 00:00:00
5
6 current_time = time(16, 25)
7 print(current_time) # 16:25:00
8
9 current_time = time(16, 25, 45)
10 print(current_time) # 16:25:45
```

### ***datetime* sinfi**

*datetime* sinfi bir vaqtning o'zida sana va vaqt bilan ishlash imkoniyatini yaratadi. *datetime* sinfi obyektini yaratish uchun quyidagi konstruktor ishlatiladi:

```
1 datetime(year, month, day [, hour] [, min] [, sec] [,
microsec])
```

Birinchi uchta parametr yil, oy va kunlar zaruriy parametrlar hisoblanadi, qolgan uchtasi esa shart bo'lmagan, hamda, kelishuv bo'yicha ular nol qiymat oladi.

```
1 from datetime import datetime
2
3 deadline = datetime(2017, 5, 10)
4 print(deadline) # 2017-05-10 00:00:00
5
6 deadline = datetime(2017, 5, 10, 4, 30)
7 print(deadline) # 2017-05-10 04:30:00
```

Joriy vaqtni va sanani olish uchun *now()* metodidan foydalaniladi.

```

1  from datetime import datetime
2
3  now = datetime.now()
4  print(now)  # 2017-05-03 11:18:56.239443
5
6  print("{}.{}.{}  {}:{}".format(now.day, now.month,
7  now.year, now.hour, now.minute))  # 3.5.2017  11:21
8
9  print(now.date())
10 print(now.time())

```

*day*, *month*, *year*, *hour*, *minute*, *second* parametrlari orqali sana va vaqtning alohida qiymatlarini olish mumkin. *date()* va *time()* metodlari orqali esa mos ravishda alohida sana va vaqtni olish mumkin.

### Satrdan sanaga o'tkazish

*datetime* sinfida *strptime()* metodi mavjud bo'lib, u satr ko'rinishidagi berilgani vaqtga o'tkazadi. Bu metod ikkita parametr qabul qiladi:

```

1 .strptime(str, format)

```

Birinchi *str* parametri sana va vaqtning satr ko'rinishi, hamda ikkinchi *format* parametri satrdagi sana va vaqt orasi qanday ajratilganligi formati.

Formatni aniqlash uchun quyidagi kodlarni ishlatamiz:

- %d – oy kuni son ko'rinishida;
- %m – oyning tartib raqami;
- %y – yil ikkita raqamdan iborat;
- %Y – yil to'rta raqamdan iborat;
- %H – soat 24 soatlik formatda;
- %M – minut;
- %S – sekund.

Har xil formatlarga misol:

```

1  from datetime import datetime
2

```

```

3 deadline = datetime.strptime("22/05/2017", "%d/%m/%Y")
4 print(deadline) # 2017-05-22 00:00:00
5
6 deadline = datetime.strptime("22/05/2017 12:30", "%d/%m/%Y
7 %H:%M")
8 print(deadline) # 2017-05-22 12:30:00
9
10 deadline = datetime.strptime("05-22-2017 12:30", "%m-%d-%Y
11 %H:%M")
12 print(deadline) # 2017-05-22 12:30:00

```

## 7.2. Sana ustuda bajariladigan asosiy amallar

### Sana va vaqtni formatlash

Ushbu sinflar doirasida sana va vaqt obyektlarini formatlash uchun *strptime(format)* metodi mavjud. Bu metod formatlashni ko'rsatuvchi bitta parametr qabul qiladi.

Formatlashni amalga oshirishimiz uchun quyida aniqlangan formatlash kodlaridan birini ishlatish mumkin:

- %a – hafta kuni uchun abbreviatoriya. Masalan, Wed – Wednesday so'zidan (kelishuv bo'yicha ingliz tilidagi so'zlar ishlatiladi);
- %A – hafta kun to'liq, masalan, Wednesday;
- %b – oy kuni uchun abbreviatoriya. Masalan, Oct (October so'zining qisqartmasi);
- %B – oy nomi to'liq, masalan, October;
- %d – oy kuni, nol qo'shilgan, masalan, 01;
- %m – oy raqami, nol qo'shilgan, masalan, 05;
- %y – yil ikkita raqamdan iborat;
- %Y – yil to'rta raqamdan iborat;
- %H – soat 24 soatlik formatda, masalan, 13
- %I – soat 12 soatlik formatda, masalan, 01
- %M – Minut;

- %S – sekund.
- %f – mikrosekund;
- %p - AM/PM ko'rsatgich;
- %c – sana va vaqt, joriy mahalliy bo'yicha formatlangan;
- %x – sana, joriy mahalliy bo'yicha formatlangan;
- %X - vaqt, joriy mahalliy bo'yicha formatlangan.

Har xil formatlar:

```

1  from datetime import datetime
2  now = datetime.now()
3  print(now.strftime("%Y-%m-%d"))           # 2017-05-03
4  print(now.strftime("%d/%m/%Y"))         # 03/05/2017
5  print(now.strftime("%d/%m/%y"))         # 03/05/17
6  print(now.strftime("%d %B %Y (%A)"))    # 03 May 2017
7  (Wednesday)
8  print(now.strftime("%d/%m/%y %I:%M"))   # 03/05/17
9  01:36

```

Oy va kunlarning nomini kiritishda kelishuv bo'yicha ingliz tilidagi nomlar ishlatiladi. Agarda joriy mahalliy formatlarni o'rnatish zarur bo'lsa, u holda *locale* modulidan foydalaniladi:

```

1  from datetime import datetime
2  import locale
3
4  locale.setlocale(locale.LC_ALL, "Uz")
5
6  now = datetime.now()
7  print(now.strftime("%d %B %Y (%A)"))    # 08 avgust 2019
8  (payshanba)

```

### Sana va vaqtlarni qo'shish va ayirish

Sana va vaqt bilan ishlashda ma'lum bir vaqt oraliqdagi sanani qo'shish yoki ayirish zarurati tug'uladi. *datetime* modulida bu ishlarni amalga oshirish uchun

maxsus *timedelta* sinfi aniqlangan. Ushbu sinf ma'lum bir vaqt oralig'dani aniqlaydi.

Vaqt oralig'ini aniqlash uchun *timedelta* sinfi konstruktori quyidagicha ishlatiladi:

```
1 timedelta([days] [, seconds] [, microseconds] [, milliseconds] [,
minutes] [, hours] [, weeks])
```

Konstrutorga mos ketma-ketlikda kunlar, sekundlar, mikrosekundlar, milisekundlar, minutlar, soatlar va haftalarni berish mumkin.

Bir qancha oraliqlarni aniqlaymiz:

```
1 from datetime import timedelta
2
3 three_hours = timedelta(hours=3)
4 print(three_hours) # 3:00:00
5 three_hours_thirty_minutes = timedelta(hours=3, minutes=30)
6 # 3:30:00
7
8 two_days = timedelta(2) # 2 days, 0:00:00
9
10 two_days_three_hours_thirty_minutes = timedelta(days=2,
11 hours=3, minutes=30) # 2 days, 3:30:00
```

*timedelta* obyektini ishlatish orqali qo'shish va ayirish amallarini bajarishimiz mumkin. Masalan, ikki kundan keyingi sanani olamiz:

```
1 from datetime import timedelta, datetime
2
3 now = datetime.now()
4 print(now) # 2019-08-08 19:23:41.774384
5 two_days = timedelta(2)
6 in_two_days = now + two_days
7 print(in_two_days) # 2019-08-10 19:23:41.774384
```

Yoki 10 soatu 15 minut oldin qanday vaqt bo'lganini aniqlaymiz, buning uchun joriy vaqtdan 10 soatu 15 minutni ayirish kerak:



```

1  from datetime import timedelta, datetime
2
3  now = datetime.now()
4  till_ten_hours_fifteen_minutes = now - timedelta(hours=10,
5  minutes=15)
6  print(till_ten_hours_fifteen_minutes)

```

### ***timedelta* xususiyatlari**

*timedelta* sinfi bir qancha xususiyatlarga ega bo'lib, ular orqali vaqt oraliqlarini olish mumkin:

- `days` – kunlar miqdorini qaytaradi;
- `seconds` – sekundlar miqdorini qaytaradi;
- `microseconds` – mikrosekundlar miqdorini qaytaradi.

Bundan tashqari umumiy sekundlar miqdorini qaytaruvchi *total\_seconds()* metodi ham mavjud, hamda bunga kunlar, sekundlar va mikrosekundlar kiradi.

Masalan, ikki sana oralig'idagi vaqtni aniqlaymiz:

```

1  from datetime import timedelta, datetime
2
3  now = datetime.now()
4  twenty_two_may = datetime(2019, 12, 22)
5  period = twenty_two_may - now
6  print("{} kun {} sekund {}
7  mikrosekund".format(period.days, period.seconds,
8  period.microseconds))
9  # 135 kun 16024 sekund 750740 mikrosekund
10
11 print("Hammasi: {} sekund".format(period.total_seconds()))
12 # Hammasi: 11680024.75074 sekund

```

### **Sanalarni solishtirish**

Satrlar va sonlar kabi sanani ham standart taqqoslash amallari yordamida solishtirish mumkin:

```

1  from datetime import datetime

```

```
2
3 now = datetime.now()
4 deadline = datetime(2019, 5, 22)
5 if now > deadline:
6     print("Dasturni topshirish muddati o'tdi")
7 elif now.day == deadline.day and now.month ==
8 deadline.month and now.year == deadline.year:
9     print("Bugun dasturni topshirish muddati")
10 else:
11     period = deadline - now
12     print("Qoldi {} kun".format(period.days))
```