O'ZBEKISTON RESPUBLIKASI

OLIY VA O'RTA MAXSUS TA'LIM VAZIRLIGI

Namangan muhandislik – qurilish instituti

PYTHON DASTURLASH TILI

Namangan 2021 yil

UO‘K:

KBK:

I –

Jakbarov O., Goyipov U., Akbarov B.

Python dasturlash tili: O`quv qo`llanma – N.: “Namangan” nashriyoti, 2021 – 200 b.

Taqrizchilar: Namangan muhandislik-texnologiya instituti, Texnologik jarayonlarni avtomatlashtirish va boshqarish va informasion texnologiyalar kafedrasi dosenti **K.D.Ismanova**

Namangan davlat universiteti “Amaliy matematika” kafedrasining dosenti, pedagogika fanlari nomzodi, dotsent **N.Ataxanov**

Namangan muhandislik-qurilish instituti “Oliy matematika” kafedrasi professori, fizika-matematika fanlari doktori **V.Xojiboev**

Mazkur o`quv qo`llanma oliy ta`lim tizimining 5330200-Informatika va axborot texnologiyalari (sanoat ishlab chiqarishida) va 5330200 – Axborot tizimlari va texnologiyalari (tarmoqlar va sohalar bo’yicha) ta`lim yo`nalishlari talabalari uchun mo`ljallangan. Qo’llanmada Python dasturlash tilining imkoniyatlari va unda ilovalar yaratish bo`yicha ma’lumotlar talabalar amaliy ko’nikmalarga ega bo’lishlari uchun bob va mavzular kesimida amaliy misollar yordamida tushintirib berilgan. Shuningdek, o`quv qo`llanmada Python tili bo`yicha asosiy tushunchalar amaliy masalalar bilan bog`lab mustaqil o`rganishga yo`naltiriladi. Ushbu o`quv qo`llanma talabalarni mantiqiy fikr yuritish qobiliyatini oshirish, ilmiy adabiyotlar bilan mustaqil ishlashga o`rgatish va yangi kompyuter texnologiyalari yordamida zamonaviy ilovalar ishlab chiqishni o`rgatishga mo`ljallangan. Shuningdek, o`quv qo`llanmadan mazkur tilni o`rganmoqchi bo`lgan barcha bilim oluvchilar keng foydalanishlari tavsiya etiladi.

ISBN 978-9943-328-58-2

Jakbarov O., Goyipov U., Akbarov B.

MUNDARIJA

SO’Z BOSHI

Bugungi kunda Internetning ommaviyligi haqida gapirish o’rinsiz. Internet hayotimizning bir bo’lagiga aylandi, biz uning xizmatlaridan har kuni foydalanishga odatlandik. Hozirda ixtiyoriy inson webtexnologiyalarning inson hayotining ta’lim, kommersiya, siyosat, ko’ngil ochar, … bo’laklariga kirib borganligini tasavvur eta oladi va uning guvohi va foydalanuvchisiga aylanmoqda.

Xususan, Axborot texnologiyalari va kommunikasiyalari sohasida boshqaruv tizimini yanada takomillashtirish, elektron davlat xizmatlari va telekommunikasiya xizmatlari spektrini kengaytirish, telekommunikasiya infratuzilmasini rivojlantirish maqsadida, O’zbekiston Respublikasi Prezidenti SH.Mirziyoev tomonidan 2018 yil 19 fevral PF-5349-sonli “Axborot texnologiyalari va kommunikasiyalari sohasini yanada takomillashtirish chora-tadbirlari to’g’risida”gi Farmonida ham bugungi kundagi AKT sohasida qator kamchiliklar va ularni bartaraf etish bo’yicha alohida topshiriqlar berib o’tiladi.

Mazkur qo’llanma zamonaviy dasturlar yaratish, qiziqarli va o’ziga tortuvchi dasturlarni yaratish, vaqti kelsa ma’lumotlarmi yangilash kabi vazifalarni o’rgatishga mo’ljallangan.

Shu bilan birga, axborot texnologiyalari va kommunikasiyalarini boshqarish va joriy etish sohasidagi muammo va kamchiliklarni bartaraf etishga xissa qo’shish maqsadida ushbu sohaning mutaxassislarini yetishtirishda manba sifatida ushbu o’quv qo’llanma ishlab chiqildi.

1-BOB. PYTHON DASTURLASH TILI BILAN TANISHISH

O`quv modullari

Python, CNRI, CWI, proyekt, OYD.

1.1. Python dasturlash tili yaratilishi tarixi

Python dasturlash tilini yaratilishi 1980 yil oxiri 1990 yil boshlaridan boshlangan. O`sha paytlarda uncha taniqli bo`lmagan Gollandiyaning CWI instituti xodimi Gvido van Rossum ABC tilini yaratilish proektida ishtirok etgan edi. ABC tili Basic tili o`rniga talabalarga asosiy dasturlash konsepsiyalarini o`rgatish uchun mo`ljallangan til edi. Bir kun Gvido bu ishlardan charchadi va 2 hafta davomida o`zining Macintoshida boshqa oddiy tilning interpretatorini yozdi, bunda u albatta ABC tilining ba'zi bir g`oyalarini o`zlashtirdi. Shuningdek, Python 1980-1990 yillarda keng foydalanilgan Algol-68, C, C++, Modul3 ABC, SmallTalk tillarining ko`plab xususiyatlarini o`ziga olgandi. Gvido van Rossum bu tilni internet orqali tarqata boshladi. Bu paytda o`zining "Dasturlash tillarining qiyosiy taqrizi" veb sahifasi bilan internetda to 1996 yilgacha Stiv Mayevskiy ismli kishi taniqli edi. U ham Macintoshni yoqtirardi va bu narsa uni Gvido bilan yaqinlashtirdi. O`sha paytlarda Gvido BBC ning "Monti Paytonning havo sirki" komediyasining muxlisi edi va o`zi yaratgan tilni Monti Payton nomiga Python deb atadi (ilon nomiga emas).

Til tezda ommalashdi. Bu dasturlash tiliga qiziqqan va tushunadigan foydalanuvchilar soni ko`paydi. Boshida bu juda oddiy til edi. Shunchaki kichik interpretator bir nechta funksiyalarga ega edi. 1991 yil birinchi OYD (Obyektga Yo`naltirilgan Dasturlash) vositalari paydo bo`ldi.

Bir qancha vaqt o`tib Gvido Gollandiyadan Amerikaga ko`chib o`tdi. Uni CNRI korparatsiyasiga ishlashga taklif etishdi. U o`sha yerda ishladi va korparatsiya shug`ullanayotgan proektlarni Python tilida yozdi va bo`sh ish vaqtlarida tilni interpretatorini rivojlantirib bordi. Bu 1990 yil Python 1.5.2 versiyasi paydo bo`lguncha davom etdi. Gvidoning asosiy vaqti korparatsiyani proektlarini yaratishga ketardi bu esa unga yoqmasdi. Chunki uning Python dasturlash tilini rivojlantirishga

vaqti qolmayotgandi. Shunda u o`ziga tilni rivojlantirishga imkoniyat yaratib bera oladigan homiy izladi va uni o`sha paytlarda endi tashkil etilgan BeOpen firmasi qo`llab quvvatladi. U CNRI dan ketdi, lekin shartnomaga binoan u Python 1.6 versiyasini chiqarib berishga majbur edi. BeOpen da esa u Python 2.0 versiyani chiqardi. 2.0 versiyasi bu oldinga qo`yilgan katta qadamlardan edi. Bu versiyada eng asosiysi til va interpretatorni rivojlanish jarayoni ochiq ravishda bo`ldi.

Shunday qilib 1.0 versiyasi 1994 yil chiqarilgan bo`lsa, 2.0 versiyasi 2000-yil, 3.0 versiyasi esa 2008-yil ishlab chiqarildi. Hozirgi vaqtda uchinchi versiyasi keng qo`llaniladi.

1.2. Python dasturlash tili imkoniyatlari

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

Pythonning o`ziga xosligi

- ✓ Oddiy, o`rganishga oson, sodda sintaksisga ega, dasturlashni boshlash uchun qulay, erkin va ochiq kodlik dasturiy ta'minot.
- ✓ Dasturni yozish davomida quyi darajadagi detallarni, misol uchun xotirani boshqarishni hisobga olish shart emas.
- ✓ Ko'plab platformalarda hech qanday o'zgartirishlarsiz ishlay oladi.
- ✓ Interpretatsiya qilinadigan til.
- ✓ Kengayishga moyil til. Agar dasturni biror joyini tezroq ishlashini xoxlasak shu qismni C yoki C++ dasturlash tillarida yozib keyin shu qismni python kodi orqali ishga tushirsa(chaqirsa) bo'ladi.
- ✓ Juda ham ko'p xilma-xil kutubxonalarga ega.
- ✓ xml/html fayllar bilan ishlash
- ✓ http so`rovlari bilan ishlash
- ✓ GUI (grafik interfeys)
- ✓ Web ssenariy tuzish
- ✓ FTP bilan ishlash

- ✓ Rasmli audio video fayllar bilan ishlash
- ✓ Robot texnikada
- ✓ Matematik va ilmiy hisoblashlarni programmalash

Pythonni katta proyektlarda ishlatish mumkin. Chunki, uni chegarasi yo`q, imkoniyati yuqori. Shuningdek, u sodda va universalligi bilan programmalash tillari orasida eng yaxshisidir.

Qisqacha qilib aytganda Python quyidagi sohalarda faol ishlatiladi:

1. Tizimli dasturlash.
2. Grafik interfeysli dasturlarni ishlab chiqish.
3. Dinamik web-saytlarni yaratish.
4. Komponentlarning integratsiyasi.
5. Ma'lumotlar bazalari bilan ishlash uchun dasturlarni ishlab chiqish.
6. Prototiplarni tezkor yaratish.
7. Ilmiy hisoblash uchun dasturlarni ishlab chiqish.
8. O'yin dasturlar yaratish

Python dasturlarini ishga tushirish uchun nima qilishimiz kerak? Ushbu savolga javob berishdan oldin, dasturlarning kompyuterda qanday ishlashini ko'rib chiqamiz. Dasturlarning bajarilishi operatsion tizim tomonidan amalga oshiriladi (Windows, Linux va boshqalar). Operatsion tizimning vazifalari dastur uchun resurslarni taqsimlash, kiritish/chiqarish qurilmalariga kirishni taqiqlash yoki ruxsat berishdan iborat.

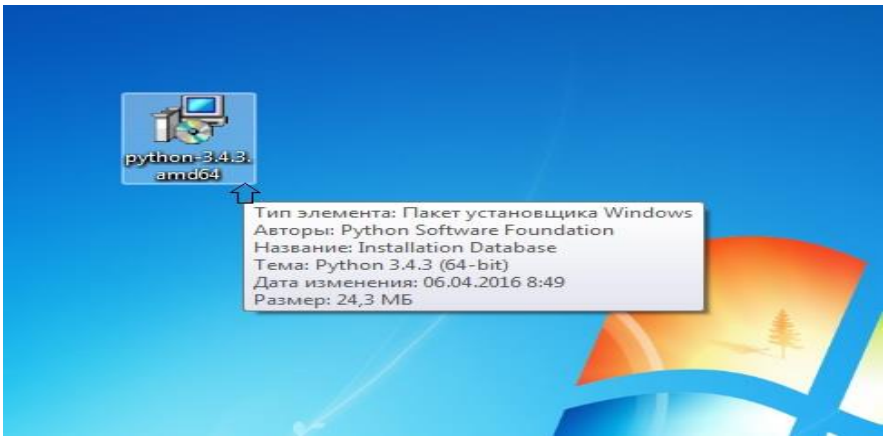
Python dasturlarini ishga tushirish uchun sizga Python tarjimoni (virtual mashina) kerak bo'ladi. Ushbu dastur Python dasturchisidan operatsion tizimning barcha xususiyatlarini yashiradi, shuning uchun Windows tizimida Python-da dastur yozib, siz uni, masalan, GNU / Linux-da ishlatishingiz va bir xil natijaga erishishingiz mumkin.

1.3. Python dasturlash tilini o`rnatish.

Agar siz biror GNU/Linux distributivini ishlatayotgan bo'lsangiz ko'p xollarda sizning tizimingizda **python** o'rnatilgan bo'ladi. Buni tekshirib ko'rish uchun terminalingizdan quyidagi buyruqni ishga tushirib ko'ring. **python -V**

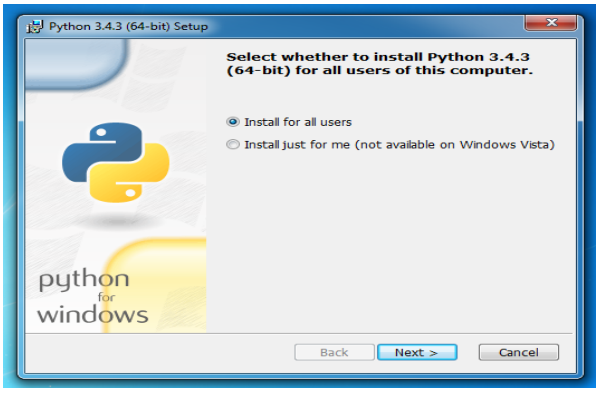
Agar sizda **Python 3.4.3** yozuvi yoki shunga o'xshash yozuv hosil bo'lsa unda hammasi joyida.

Windows operatsion tizimiga o'rnatish uchun www.python.org/downloads web sahifasiga o'tamiz va u yerdan oxirgi python versiyasini yuklab olamiz. Pythonni o'rnatish odatiy dasturlarni o'rnatish kabi kechadi. Hech qanday qiyin joyi yo'q.



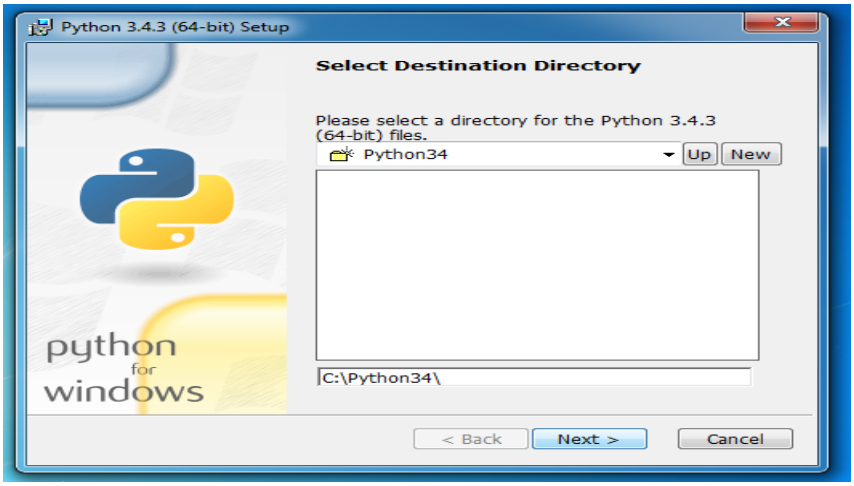
1.1.1-chizma. Python dasturining o`rnatiluvchi fayli.

Python dasturlash tilining o`rnatuvchi paketini ustiga sichqoncha ko`ratkichini 2 marta bosamiz va bizga quyidagi oyna hosil bo`ladi.



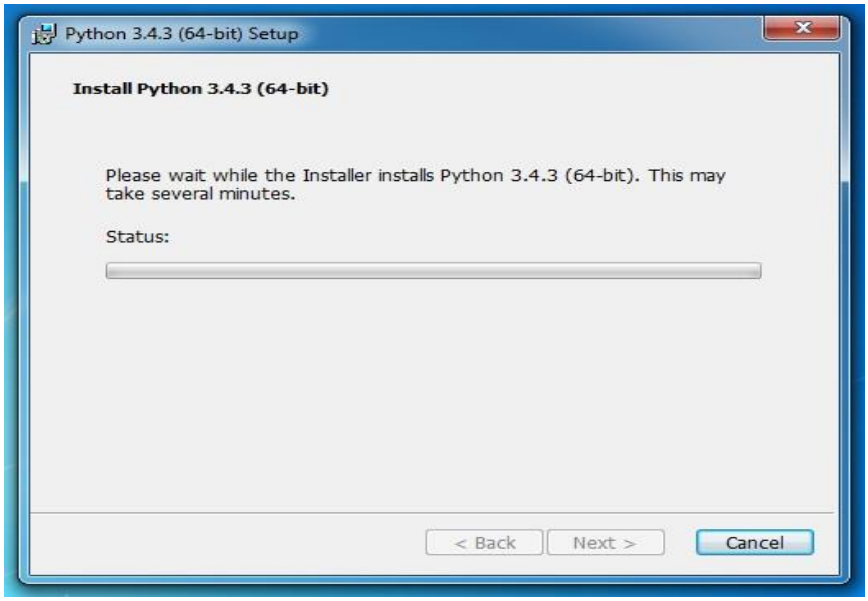
1.1.2-chizma. Python dasturini o`rnatishni boshlashni ko`rsatuvchi oyna.

Bu yerda **Install for all users**-barcha foydalanuvchilar uchun. **Install just for me**- faqat siz uchun, agar buni tanlab istalyatsiya qilsak ya'ni o`rnatsak Windows Vista operatsion sistemasida xatolik yuz beradi va dastur ishlamaydi. Shuning uchun **Install for all users** ni tanlaganimiz maqul. Keyin next tugmasi bosamiz.



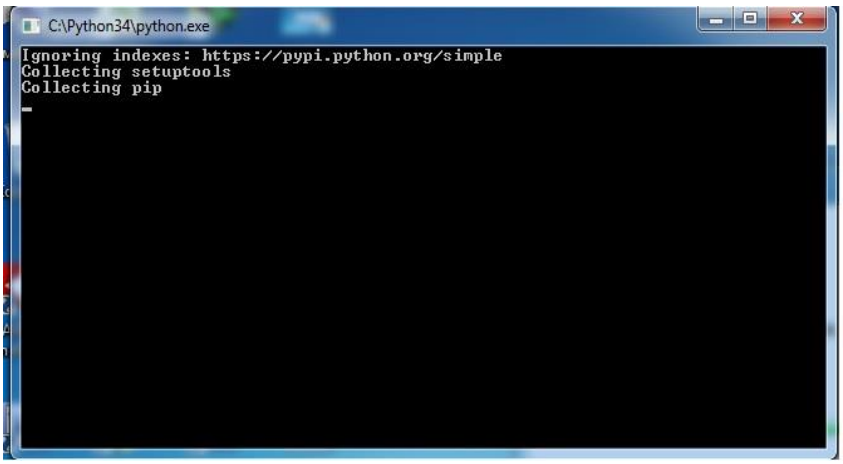
1.1.3-chizma. Python dasturini o`rnatilish joyini ko`rsatish oynasi.

Bu yerda esa Python dasturlash tilini qayerda o`rnatilishi ko`rsatilayapti.



1.1.4-chizma. Python dasturini o`rnatilish jarayoni.

Python o`rnatilyapti va bir necha sekunddan so`ng quyidagi oyna namoyon bo`ladi:



1.1.5-chizma. PIP kutubxonasini qo`shish jarayonida hosil boladigan oyna.

Bunda Console rejimida dastur ishga tushib **pip** kutubxonasini qo`shadi.

1.1.6-chizma. Python dasturini o`rnatish tugallanganligi haqidagi muloqot oynasi.

Va dasturni o`rnatish muofaqqiyatli tugallandi.

1.4. Dastur tuzilishi

Pythonda har bir instursiya alohida satraga yoziladi. Ko’pgina dasturlash tillari (masalan, PHP, Perl va boshqa)da har bir instruksiya nuqtali vergul bilan yakunlanadi. Pythonda ham instruksiya ohiriga nuqtlil vergul qo’yish mumkin, biroq majburiy emas.

Har bir satr boshidagi bo`sh joy(отступ) muhim ahamiyatga ega. Kiritilgan amallar bo`sh joylarning kattaligiga qarab **blok**larga birlashadi. Bo`sh joy istalgancha bo`lishi mumkin asosiysi bitta kiritilgan blok chegarasida bo`sh joy bir xil bo`lishi

kerak. Noto`g`ri qo`yilgan bo`sh joylar xatolik yuz berishiga olib kelishi mumkin. Bitta probel bilan bo`sh joy hosil qilish yaxshi qaror emas uni o`rniga to`rtta probel yoki *Tab* belgisini ishlatish kerak.

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

Pythonda nuqtali vergul quyish tavsiya etilmaydi. Satr yakuni instruksiya yakuni hisoblanadi. Shunga qaramay, bir necha instruksiyani bir satrga yozish kerak bo'lsa nuqta verguldan foyadalanish mumkin va quyidagi misolda keltiriladi.

1-misol. Bir nechta instrukisyani bir satrda qo'llash

```
>>> x = 5;  y  =  10;  z  =  x  +  y  #  uch instruksiya bir satrda
>>> print (z)
```

Pythonda dastur yozilishinig yana bir muhim husisiyati shundaki kodlar blokini ajratsih uchun alohiada belgi ishlatilmaydi. Faqat bok tekislanishi bilan ajratiladi.

Masalan, PHPda while sikli tanasi figurali qavs yordamida quyidagicha yoziladi:

```
$i = 1;
while ($ i < 11) {
echo $i.  "\ n" ;
$i ++ ;
}
echo " Dastur tamom ";
```

Python tilida esa boshqacha yoziladi:

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

Diqqat qiling, blok ichidan tashqari barcha instruksiyalar bir hil joylashgan. Blok tanasi esa alohida probellar satrida joylashgan. Misoliizda ikkita instruksiya o'n marta

bajariladi. Blok yakunlangach yana instruksiya dastlabki probel tekisligiga joylashtiriladi. Bu print("Dastur tamom") instruksiyasidir.

Agar bir blok ichidagi probeller turlicha bo'lsa, xatolik kelib chiqadi. Odatda to'rt probel yoki bitta tabulyatsiya belgisi bilan ajratiladi.

Agar blok bitta instruksiyadan ibotar bo'sa uni asosiy instruksiya bilan bir satrga joylashtirish mumkin. Masalan:

```
for i in range (l, 11) :
print (i)
print ("Dastur tamom" )
```

yuqoridfagi dasturni quyidagicha ham yozish mumkin:

```
for i in range (l, 11) : print (i)
print ("Dastur tamom" )
```

Agar konstruksiya juda uzun bo'lsa quyidagi usullardan birida uni yangi satrga bo'lib yozish mumkin.

- ✓ satr oxiriga "\" belgisini joylashtirish yordamida va undan keingi belgi navbatdagi satrdan yoziladi. Masalan:

Пример:

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

- ✓ ifodani aylanma qavslar yordamida yozish. Bu ancha yaxshiroq usul bo'lib, ixtiyoriy ifodani satrlarga joylashtirish mumkin bo'ladi. Masalan:

```
x = (15 + 20 # bu izoh
+ 30)
print (x)
```

- ✓ ro'yxat va lug'atlarni aniqlashda ularni bir nechta satrga yozish mumkin. Bunda mos ravishda kvadrat va aylama qavslardan foydalaniladi. Ro'yxatni aniqlashga doir misol:

```
arr = [15, 20, # bu izoh
       30]
```

```
print (arr)
```

Lug’atni aniqlashga doir misol:

Пример определения словаря:

```
arr  =  {" x" :  15,  "y":  20,   # bu izoh
          "z" :  30 }
print (arr)
```

1.5. Izohlar

Izohlar dastur matnini tushintirish uchun foydalaniladi va dasturni izohlaydi. Izohlar ihtiyoriy matn, ko’rsatalardan iborat bo’lishi mumkin va ular bajarilmaydi. Ular dastur kodini o'qiyotganlar uchun foydali bo'ladi va dastur nima qilishini oson tushunishga yordam beradi. Izohlarga yechimdagi muhim joylarni, muhim bo'lgan qismlarni yozish mumkin. Pythonda bir satrli izoh ishlatiladi va u # belgisidan boshlanadi. Masalan: *# bu izoh*

```
# print — bu funksiya
print ('salom dunyo! ')
```

bir satrli izoh satr boshidan balki instruksiyaning ihtiyoriy joyidan boshlanishi mumkin va unadan keying belgilar izoh sifatida qabul qilinadi. Maslan:

```
print ('salom dunyo!')   #  print — bu funksiya
```

Agar # belgisi aylanma qavs yoki apastrof ichida joylashgan bo’lsa, u izoh belgisi hisoblanmaydi:

```
print ("# Bu izoh emas")
```

Python da ko’p satrli izohlar mavjud bo’lmagani uchun, bazan ko’p satrli izohlar uchtalik qo’shtirnoq (yoki uchtalik apostrof) orqali berilishi mumkin:

```
"""
Bu instruksiya bajarilmaydi
Faqat izoh sifatida foydalanish mumkin
print ("Bu ham izohning uchinchi satri!")
"""
```

Yuqoridagi satrlar bajarilmaydi ular faqat dasturda o’qish uchun holos.

1.6. Dastur natijasini chop etish

Dastur natijasini ekranga chiqarish uchun *print()* funksiyasidan foydalaniladi. Funksiya quyidagi formatda beriladi:

print ([<Obyektlar>] [, sep= ' '] [, end=' \n'] [, file=sys.stdout] [, flush=False])

print() funksiyasi obyektni satrga o'tkazadi va uni stdout standart chiqarishga uzatadi. *file* parametri yordamida chiqarish boshqa joyga masalan, faylga yo'naltirilishi mumkin. Bunda *flush* paramtri qiymati False bo'lsa chiqariladigan qiymat faylga yozilishi majburiy bo'ladi. Chop etishlarni yo'naltirish bo'yicha fayllar mavzusida batafsil tanishib chiqamiz.

Satr chop etilgach avtomatik satr keying satrga o'tkaziladi:

```
print ("1-satr")
print ("2-satr ")
```

Natija:

```
1-satr
2-satr
```

Agar bir nechta qiymatni bitta satrda chiqarish zarur bo'lsa, qiymatlar vergul yordamida print() funksiyasidagi birinchi parametrda beriladi:

```
print ("1-satr", "2-satr")
```

Natija:

```
1-satr 2-satr
```

Misoldan ko'rinib turganidek, chop etiladigan satrlar orasiga avtomati probel qo'yiladi. **sep** parametri yordamida boshqa belgini ajratgich sifatifda ko'rsatish mumkin.

Masalan satrlar probelsiz chiqishi uchun quyidagicha yoziladi:

```
print ("1-satr","2-satr", sep="")
```

Natija:

```
1-satr2-satr
```

Obyektni chiqargandan keyin ohirida keying satrga o'tish belgisi qo'shiladi. Agar keying chiqariladigan satr oldingisi bilan bir satrga joylashishi zarur bo'lsa end

parametrida quyidagicha probel belgisi ko’rsatiladi:

```
print ("1-satr", "2-satr", end=" ")
print ("3-satr)
```

Natija:

1-satr 2-satr 3-satr

Agar aksincha bolishini ya’ni keying chop etish natijasi yangi satrdan chiqishini hohlasangiz *print*() funksiyasida end parametrini ko’rsatmang. Masalan:

```
for n in range (l, 5):
print (n, end=" ")
print()
print ("Bu matn yangi satrda chop etiladi")
```

Natija quyidagicha chop etiladi:

1 2 3 4

Bu matn yangi satrda chop etiladi

Bu yerda biz for siklidan foydalanib sonlar ketma ketlini chop etdik. Sikldan keyingi print() operatori navbatdagi print ("Bu matn yangi satrda chop etiladi") operatorida chop etiladigan satrni yangi satrdan chop etish uchun parametrsiz shaklda yozildi.

1.7. Ma’lumotlarni kiritish

Python 3 da ma’lumotlarni kiritish uchun stdin standartidan ma’lumot oluvchi input () funksiyasi dan foydalaniladi. Funksiya quyidagi umumiy ko’rinishga ega:

[<Qiymat> =] input ([<habar>])

Misol sifatida foydalanuvhci bilan salomlashish dasturini ko’rib chiqamiz.

2-misol:

```
# -*- coding: utf-8 -*-
name = input("Ismingizni kiriting: ")
print("Salom,", name)
input("Dasturdan chiqish uchin <Enter> tugmasini bosing")
```

Dastur natijasi:

```
==== RESTART: C:/Users/user/AppData/Local/Programs/Python/Python37/14.py ====
Ismingizni kiriting: Umidjon
Salom, Umidjon
Oynani yopish uchun <Enter> tugmasini bosing
```

Muhokama uchun savollar va topshiriqlar

1. Python qanday til?
2. Python yaratuvchilarini bilasizmi?
3. Python qanday imkoniyatlarni taqdim etadi?
4. Python muharrirlarni bilasizmi?
5. Python qanday o’rnatiladi?
6. Pythonda operatorlar qanday ajratiladi?
7. Pythonda blok qanday beriladi?
8. Python shell muhiti vazivasi nima?
9. Pythonda izohlar qanday beriladi?
10. Pythonda dastur chop etish operatori?
11. Pythonda ma’lumot kiritish operatori.

2-BOB. O’ZGARUVCHILAR

O`quv modullari

Python, o’zgaruvchi, obyektlar, dinamik tiplashtirish, Turlar, Instruksiya, Interpretator kodlar

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

Python dasturlash tilida barcha ma’lumotlar obyektlar sifatida beriladi. Har bir obyekt ma’lumot tipiga va qiymatiga ega. Obyektga ruxsat o’zgaruvchi orqali aniqlashtiriladi. O’zgaruvchi insalizasiyasida obyekt (kompyuter xotirasidagi obyekt manzili) ga havola saqlanadi. Keyinchalik shu havola sababli dasturdan obyektni o’zgartirish mumkin bo’ladi.

2.1. O’zgaruvchini nomlash

Har bir o’zgaruvchi lotin alifbosi, raqamlar, ostga chizish belgisidan iborat bo’lgan va raqmdan boshlanamaydigan unikal nomga ega bo’lishi kerak. Bundan tashqari o’zgaruvchini nomlashda ostgi chiziqni nom boshida kelishi tavsiya etilmaydi, chunki ko’p bunday nomlangan identifikator tilda maxsus vazifani bajaradi. Ular ustma-ust tushib qolib, dasturda xatolik kelib chiqishi mumkin. Bundan tashqari kalit so’zlardan o’zgaruvchi nomi sifatida foydalanish ta’qiqlanadi.

Qisqa qilib aytganda o'zgaruvchilarni nomlashda quyidagi qoidalarga amal qilish kerak:

- O'zgaruvchining birinchi belgisi alifbo harfi (ASCII simvollari katta va kichik registrda) yoki “_” (ostki chiziq) simvoli bo'lishi mumkin(lekin odatda tavsiya etilmaydi).

- O'zgaruvchilarning qolgan qismi harflardan (ASCII simvollari katta va kichik registrda), “_" (ostki chiziq) simvoli va raqamlardan(0-9) tashkil topishi mumkin.
- O'zgaruvchilar nomlashda katta va kichik registrlar farqlanadi. Masalan, *myname va myName – bular boshqa-boshqa o'zgaruvchi hisoblanadi.*
- O'zgaruvchilarni to'g'ri nomlashga misollar*: i, _my_name, name_23, a1b2_c3*
- O'zgaruvchilarni noto'g'ri nomlashga misollar: 2things, ' ', my-name, >a1b2_c3 va “o'zgaruvchi qo'shtirnoqda"

Kalit so’zlar

False – yolg`on.

True - rost.

None - “bo`sh” obyekt.

and – mantiqiy VA amali.

with / as – konteks menejeri.

break –tsikldan chiqish.

class – metod va atributlarda iborat.

continue – tsikldan keyingi iteratsiyaga o`tish.

def – funksiyani aniqlash.

del – obyektni yo`qotish.

elif – aks holda, agar.

else – for/else yoki if/elsega qarang.

for – for tsikli.

from – moduldan bir nechta funksiyani import qilish.

if - agar.

import – moduldan import.

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

lambda –yashirin funksiyani aniqlash.

not –mantiqiy inkor amali.

or –mantiqiy Yoki amali.

while – while tsikli.

2.2. Ma’lumot turlari

Python 3 da obyektlar quydagi turlarda bo’lishi mumkin:

- bool – ma’lumotlar mantiqiy turi. True yoki False (1 yoki 0 ga teng kuchli) qiymatiga ega bo’lihi mumkin:

```
>>> type (True),  type (False)
(< class  'bool'>,  <class  'bool '> )
>>> int (True), int (False)
        (1, 0)
```

- NoneType – None qiymatli obyekt(qiymati yo’qligini anglatadi):

```
>>> type (None)
<class 'NoneType'>
```

Mantiqiy ifodada None qiymati False kabidir:

```
>>> bool (None )
False
```

- int – butun son. Son qiymati chegarasi faqat operativ xotira sig’imiga bog’liq:

```
>>>  type (214 7 48 364 7) ,  type (9 9999999999 9999999999 999 )
(< class  'int'> ,  <class  'int'> )
```

- float – suzuvchi vergulli son:

```
>>>  type( 5.1) ,  type (8 .5e- 3)
(< class 'float '> ,  <class  'float ' >)
```

- complex – kompleks son:

```
>>>  type (2 +2j)
<class 'complex'>
```

- str - Unicode-satr:

```
>>>  type(“Satr”)
<class 'str '>)
```

- bytes – o’zgarmas baytlar ketma-ketligi:

>>> type (bytes (“Satr”, "ut f- 8"))

<class 'bytes ' >)

- bytearray – o’zgaruvchan baytlar ketma-ketligi:

  >>> type (bytearray ("Satr", "utf-8"))

  <class 'bytearray'>)

- list – ro’yxat. list boshqa dasturlash tillaridagi massivlarning analogidir:

  >>> type ([1 , 2, 3])

  <class 'list'>)

- tuple - kortej:

  >>> type ((1, 2, 3)

  <class 'tuple'>)

- range -diapazon:

>>> type (range (l , 10))

<class 'range '>

- dict - lug’at. dict ma’lumot turi bohsqa dasturlash tillariadagi assosiativ massivning analogidir:

>>> type ({" x ": 5, " y": 20})

<class 'dict '>

- set – to’plam (unikal obyektlar to’plami):

>>> type ({"a", "b", "c"})

<class 'set '>

- frozenset – o’zgarmas toplam:

>>> type (frozenset (["a", "b", "c"]))

<class 'frozenset '>

- ellipsis – uch nuqta shakli yoki Ellipsis so’zini anglatadi. ellipsis turi qirqim olish kengaytmasi sifatida ham ishlatiladi:

>>> type (…) , … , ... is Ellipsis

(<class 'ellipsis'> , Ellipsis, True)

>>>class C ():

def __getitem__(self, obj) : return obj

```
>>> c = C()
>>> c[…, 1:5 , 0: 9:1 , 0]
(Ellips is, slice (l, 5, None) , slice (0, 9, 1) , 0)
```

- function - funksiya:

```
>>> def func() : pass
>>> type (func)
<class 'function'>
```

- module - modul:

```
>>> import sys
>>> type (sys )
<class 'module '>
```

- type – ma’lumotar turi va sinfi. Hayron bo’lmang! Pythonda hamma ma’lumotlar hatto ma’lumot turlarining o’zi ham obyekt hisoblanadi!

```
>>> class C: pass
>>> type (C)
<class 'type'>
>>> type (type( "") )
<class 'type '>
```

Asosiy ma’lumot turari o’zgaruvchan va o’zgarmaslarga bo’linadi. O’zgaruvchan turlarga ro’yxatlar, lug’atalar, va bytearray tegishli. Ro’yxat elementini o’zgartirish misoli:

```
>>> arr = [1, 2, 3]
>>> arr[0] = 0 # Ro’yxatning birinchi elementi o’zgaradi
>>> arr
[0, 2, 3]
```

O’zgarmas turlarga sonlar, satrlar, kortejlar, diapazonlar va bytes turlari tegishlidir. Masalan, agar ikkita boshqa satrdan satr olmoqchi bo’lsak konkatenasiya operasiyasiadan foydalanish kerak va yangi obyektga havola o’zgaruvchig ao’zlashtiriladi:

```
>>> str1 = "Gultoji"
```

```
>>> str2 = "xo’roz"
>>> str3 = str1 + str2 # Konkatenasiya
>>> print(str3)
Gultojixo’roz
```

Bundan tashqari, ma’lumotlar turlari izchillikdagi va aks etuvchilarga bo’linadi. Izchillikga satrlar, ro’yxatlar, kortejlart, diapazonlar, byte va bytearray turlari aks etuvchiga lug’at turlari tegishlidir.

Izchillikdagi va aks etuvchi iterasiya mexanizmini qo’llab-quvvatlaydi va _next_ ( ) yoki next ( ) funksiyalari yordamida elementlar bo’yicha yurib chiqishi mumkin. Masalan, ro’yxat elementini chiqarish quyidagicha:

```
>>> arr = [1, 2]
>>> i = iter(arr)
>>> i.__next__()   # __next__() metodi
1
>>> next(i)   # next() funksiyasi
2
```

Agar lug’atdan foydalanilsa, har bir iterasiya kalit qaytaradi:

```
>>> d = {"x": 1, "y": 2}
>>> i = iter(d)
>>> i.__next__() # kalit qaytariladi
'y'
>>> d[i.__next__()] # kalit bo’yicha qiymat qaytaradi
1
```

Amaliyotda bun usullardan keng foydalanilmaydi. Buning o’rniga ko’pincha for sikli operatoridan foydalaniladi. Masalan, ro’;yxat elementlari quyidagicha chop etiladi:

```
>>>  for i in [1, 2]:
print (i)
```

So’zdagi harflarni chop etish misolini ko’ramiz. Misol uchun so’zdagi har bir harfga tire qo’yamiz:

```
>>> for i in "Salom":
print(i + " -", end=" ")
```

Natija:

S-a-l-o-m

Lug’at elementlarini saralash maslasi:

```
>>> d = {"x": 1, " y":  2}
>>> for key in d:
print (d[key])
```

2.3. O’zgaruvchiqa qiymat o’zlashtirish

Python dasturlash tilida *dinamik tiplashtirishdan* foydalaniladi. Bu shuni anglatadiki, o’zgaruvchi o’zlashtirgan qiymatiga qarab interpretator avtomatik ravishda ma’lumotlar turidan biriga ajratadi. O’zgaruvchiga qiymat “=” operatori yordamida o’zlashtiriladi (beriladi). Misollar yordamida ko’ramiz:

```
>>> x  =  7            #  int tipli
>>> y=  7. 8           #  float tipli
>>> sl  =  "Satr "     #  sl  o’zgaruvhcisi “Satr ” qiymatini o’zlashtirdi
>>> s2  =  ' Satr '    #  s2  o’zgaruvhcisi ham “Satr ” qiymatini o’zlashtirdi
>>> b = True                #  b o’zgaruvchisi mantiqiy True qiymatini
```

o’zlashtirdi

Bir satrda birdaniga bir nechta o’zgaruvhciga qiymat o’zlashtirish mumkin:

```
>>> x  =  y  =  10
>>> x,  y
(10, 10 )
```

Qiymat o’zlashtirilgach o’zgaruvchida obyekt emas balki, obyektga havola saqlanadi.

Buni albatta gruhli o’zlashtirishda ham hisobga olinadi. Guruxli o’zlashtiurish son, satr va kortej tiplarida qo’llanilishi mumkin, biroq o’zgaruvchan obyektlarda buni amalga oshirish mumkin emas. Masalan:

```
>>> x  =  y  =  [ 1 ,   2 ]  #  Go’yoki ikkita obyekt yaratildi
```

```
>>> x,  y
([1, 2], [1, 2])
```

Bu misolda biz ikkita elementdan iborat ro'yxat yaratdik hamda x va y o'zgaruvchilarga qiymatini o'zlashtirdik. Endi y o'zgaruvchisi qiymatini o'zgartirib ko'ramiz:

```
>>> y[l]  =  100  #  Ikkinchi elementni o'zgartirdik
>>> x, y
([1,  100] ,  [1,  100 ] )
```

Misoldan ko'rinib turibdiki, y o'zgaruvchisi qiymatining o'zgarishi x o'zgaruvchisi qiymatini o'zgarishiga ham olib kelmoqda. Demak, har ikki o'zgaruchida bitta obyektga havola ko'rsatilmoqda. Agar ikkita obyektni ko'rsatmoqchi bo'lsak guruxli oz'lashtirishdan emas, alohid ao'zlashtirishdan foydalanish kerak:

```
>>>  x  =  [1,  2]
>>>  y  =  [1,  2]
>>>  y[ l]  =  100          #  Ikkinchi element o'zgarmoqda
>>>  x,  y
([1,  2],  [1,  100 ]  )
```

Ikkita o'zgaruvchi bir obyektga havola ko'rsatayorganini tekshirishga **is** operatori imkon beradi.

Agar o'zgaruvchilar bitta obyektga havola ko'rsatayortgan bo'lsa, is operatori True qiymatini qaytaradi:

```
>>> x  =  y  =  [1,  2]      #  bir obyekt

>>> x  is  y
True
>>> x  =  [1,  2]    #  turli obyekt
>>> y  =  [1,  2]    #  turli obyekt
>>> x is y
```

Fa lse

Interpretator kodlar samaradorligini oshirish maqsadida kichik sonlar va uzun bo'lmagan satrlarni keshlashni amalga oshiradi. Agar ikkita o'zgaruvchiga 2 sono o'zlashtirilsa, bu o'zgaruvchilar bitta obyektga havola ko'rsatadi degani. Masalan:

```
>>> x = 2; y = 2; z = 2
>>> x is y, y is z
(True, True)
```

Obyektga ko'rsatilgan havolalar miqdorini aniqlashga sys modulidagi getrefcount () metodi imkon beradi:

```
>>> import sys          # sys modulini ulaymiz
>>> sys.getrefcount (2)
304
```

Qachonki obyektga havolalar soni nol ga teng bo'lib qolsa obyekt operativ xotiradan avtomatik o'chiriladi.

2.4. Ma'lumot tipini aniqlash

Python ixtiyoriy vaqtda o'zgaruvchi turini unda saqlanadigan qiymat ma'lumot turiga mufofiq o'zgartira oladi. Masalan:

```
>>> a = "Satr "   # str turida
>>> a = 7         # Endi shu o'zgaruvchi int turiga o'zgardi
```

O'zgaruvchi qaysi turga havola o'rsatayotganini aniqlashga type(<o'zgaruvchi_nomi>) funksiyasi imkon beradi:

```
>>> type(a)
<class 'int'>
```

O'zgarvchilarda saqlanadigan ma'lumot turlarini quyidagi usiullardan birida tekshish mumkin:

- type () funksiyasi qaytaradigan qiymatni tur nomi bilan taqqoslash:

```
>>> x = 10
>>> if type (x) == int :
```

print ("Bu tur – int")

- isinstance () funksiyasi yordamida turni tekshirish:

```
>>> s = "Satr "
>>> if isinstance (s, str) :
print ("Bu tur str")
```

2.5. Ma’lumot tipini o’zgartirish

Oldingi mavzudan bilamizki, Pythonda *turlar dinamikdir*. O’zgaruvchiga qiymat o’zlashtirilgach unda obyekt emas balki, aniqlagan turdagi obyektga havola saqlanadi. Agar keyin o’zgaruvchiga boshqa turdagi qiymat o’zlashtiriladigan bo’lsa, o’zgaruvchi endi unga mos turdagi boshqa obyektga havola saqlaydi. Pythondagi shu tarzdagi ma’lumot turi bu o’zgaruvchining emas, obyektning tavsifidir. O’zgaruvchi faqat o’zida obyektga havloni saqlaydi.

O’zgaruvchiga qiymat o’zlashtirilgach o’zgaruvchi o’zlashtirilgan qiymat turiga mos amallarni bajarishi mumkin bo’ladi. Masalan, satr o’zgaruvchisiga son o’zgaruvchisini qo’shish mumkin emas va bu quyidagicha xatolik keltirib chiqaradi:

```
>>> 2 + "2511"
Traceback (most recent call last) :
File 11 <pyshe ll #0>11, line 1, in <module>
2 + “25”
TypeError: unsupported operand type (s) for +: 'int' and 'str '
```

Quyidagi funksiyalar ma’lumotlar turii o’zgartirish uchun mo’ljallangan:

- bool([<Obyekt>]) - obyektni mantiqiy turga o’zgartiradi. Masalan :

```
>>> bool(0) , bool (l) , bool (" ") , bool (“Satr”), bool ([ l, 2] ), bool ([ ])
(False, True, False, True, True, False)
```

- int ([<Obyekt> [, <Sanoq sistemasi>]]) – obyektni songa o’zgartiradi. Ikkinchi parameter sifatida sanoq sistemasii ko’rsatish mumkin. (agar ko’rsatilmasa 10 lik sanoq sistemasi deb hisoblanadi). Masalan:

```
>>> int(7. 5), int ("71")
```

(7, 71)

>>> int ("71", 10) , int ("71", 8) , int ("0071", 8) , int ("A" , 16)

(71, 57, 57, 10)

Agar o’tkazishga imkon bo’lmasa, istisni haqida habar beriladi:

>>> int ("7ls")

Traceback (most recent call last) :

File "<pyshell#9>" , line 1, in <module>

int ("71s")

ValueError: invalid literal for int () with base 10: '7ls'

- float ([<Son yoki satr>]) - butu son yoki satrni haqiyqiy (suzuvchi vergulli) songa o’tkazadi. Masalan:

>>> float (7) , float ("7. l")

(7.0, 7.1)

>>> float ("Infinity"), float ("-inf")

(inf, -inf)

>>> float ("Infinity") + float ("-inf")

nan

- str ([<Obyekt>]) - Obyektni satrga o’tkazadi. Masalan:

>>> str (125), str([l, 2, 3])

('125', '[1, 2, 3] ')

>>> str((l , 2, 3)), str ({" x" : 5, "y" : 10 })

(' (1, 2, 3) ', "{ 'y ' : 10, 'x' : 5) ")

>>> str (bytes("satr" , "utf-8 "))

"b'\\xd1\\x81\\xdl\\x82\\xdl\\x80\\xd0\\xbe\\xd0\\xba\\xd0\\xb0'"

>>> str(bytearray("satr", "utf-8 "))

"bytearray (b'\\xdl \\x81\\xdl \\x82\ \xdl \\x80\ \xd0\ \xbe\ \xd0\\xba\\xd0\\xb0 ')"

- str (<Oyekt>[, <Kodirovka>[, <Xatolik tahlili>]]) - bytes yoki bytearray turidagi obyektni satrga o’tkazish. Uchnchi parameter sifatida "strict", "replace " yoki "ignore " lardan birini berish mumkin. Masalan:

>>> objl = bytes ("1-satr", "utf- 8 ")

```
>>> obj2 = bytearray ("2-satr", "utf- 8")
>>> str (objl, "utf- 8") , str (obj2, "utf- 8")
('1-satr', '2-satr ')
>>> str(objl, "ascii ", "strict" )
Traceback (most recent call last):
File "<pyshel l#16>" , line 1, in <module>
str (objl, "ascii ", "strict" )
Uni code DecodeError: 'ascii' codec can 't decode byte
Oxdl in position 0: ordinal not in range (1 28)
>>> str (objl, "ascii", "ignore")
'1'
```

- bytes (<Satr>, <Kodirovka>[, <Xatolik tahlili>]) - satrni bytes tutidagi obyektga o'tkazish. Uchnchi parameter sifatida "strict " (odatiy qiymaat), "replace" yoki "ignore" ni ko'rsatish mumkin.

Masalan:

```
>>> bytes ("satr", "cp1251")
b' \xfl \xf2\xf0 \xee\xea\xe0'
>>> bytes ("satr123", "ascii", "ignore" )
b'123 '
```

Misol sifatida foydalanuvchi tomonidan kiritilgan ikki soni qo'shish masalasini ko'rib chiqaylik

2.4-misol. Ma'lumot kiritish va natija olish

```
# -*- coding: utf-8 -*-
x = input ("x=")          # 5 ni kiritamiz
y = input ("y=")          # 12 ni kiritamiz
print (x + y)
input ()
```

Ushbu dastur natijasini olsak “512” chiqadi. Chunki input() funksiyasi natija sifatida satri qaytaradi. Agar bu sonlar yig’indisini olishni hoxlasak ularni son turiga o’tkazish kerak (2.5-misol)

2.5-misol. Satrni songa o’tkazish

```
# -*- coding: utf-8 -*-
x = int (input ("x="))          # 5 ni kiritamiz
y = int (input ("y="))          # 12 ni kiritamiz
print (x + y)
input ()
```

Bu dastur natijasi 17 bo’ladi. Biroq foydalanuvchi son o’rniga satr kiritsa, xatolik kelib chiqadi. Bu xatolikni qayta ishlashni biz keyinroq ko’rib chiqamiz.

2.6. O’zgaruvchini o’chirish

O’zgaruvchini del instruksiyasi yordamida o’chirish mumkin:

del <o’zgaruvchi>[, …, <o’zgaruvchi_N>]

Bitta o’zgaruvchini o’chirishga misol:

```
>>> x = 10; x
10
>>> del  x;  x
Traceback (most  recent  call  last) :
Fi le  "<pyshell# l>" ,  line 1,  in  <module>
del x; x
NameError:  name  'x'  is  not  defined
```

Bir nechta o’zgaruvchini o’chirishga doir misol:

```
>>> x,  y = 10,  20
>>> del x,  y
```

Muhokama uchun savollar va topshiriqlar

1. Pythonda oz’garuvchi nima?
2. Pythonda qanday ma’lumot turlari bor?
3. Pythonda o’zlashtirish amali.
4. Pythonda qanday ma’lumot turlari bor?
5. Pythonda ma’lumot turlarini o’zgartirish
6. Pythonda o’zgaruvchini o’chirish.

3-BOB. OPERATORLAR

O`quv modullari

Matematik operatorlar, konketenasiya, ikkilik (bitli) operatorlar, o’zlashtirish operatori, unar minus, unar plyus, inversiya

Operatorlar ma’lumotlar ustida biror amal bajarishga imkon beradi. Masalan, o’zlashtirish operatori o’zgaruvchi qiymatlarini saqlash, matematik operatorlar arifmetik amallar bajarish, konketenasiya operatori satrlarni ulash kabi amalllarni bajarishga hizmat qiladi.

Python 3 dagi operatorlarni batafsil ko’rib chiqamiz.

3.1. Matematik operatorlar

Sonlar ustida amallar bajarish uchun quyidagi operatorlardan foydalaniladi:

- **+** – qo’shish:

```
>>> 10 + 5              # butun son
15
>>> 12 .4 + 5.2   # haqiyqiy son
17.6
>>> 10 + 12 .4          # butun va haqiyqiy sonlar
22 .4
```

- **-** – ayirish:

```
>>> 10 - 5          # butun son
5
>>> 12 .4 - 5.2          # haqiyqiy son
7.2
>>> 12 - 5.2             # butun va haqiyqiy son
6.8
```

- * – ko’paytirish:

```
>>> 10 * 5              # butun son
50
```

```
>>>  12 .4  *  5.2  #  haqiyqiy son
64 .4 8
>>>  10  *  5.2              #  butun va haqiyqiy son
52 .0
```

➢ / – bo’lish. Bo’lish natijasi doimo haqiyqiy son bo’ladi xatto, bo’linma va bo’luvchi butun son bo’lsa ham. Masalan:

```
>>> 10 / 5                    #  bo’linma qoldiqsiz
2.0
>>> 10 / 3                    #  bo’linma qoldiqli
3. 33333333 33333335
>>> 10 .0 / 5.0               #  bo’linuvchi va bo’luvchi haqiyqiy sonlar
2.0
>>> 10.0 / 3.0                #  bo’linuvchi va bo’luvchi haqiyqiy sonlar
3. 3333333333333335
>>>  10 / 5.0                 #  butun son haqiyqiy songa bo’linsa
2.0
>>> 10.0 / 5                  #  haqiyqiy son butun songa bo’linsa
2. 0
```

➢ // – kami bilan yaxlitlanadigan bo’lish. Barcha qoldiq tashlab yuboriladi. Maslan:

```
>>> 10 // 5   # bo’linma qoldiqsiz
2
>>>10 // 3   # bo’linma qoldiqli
3
>>>10.0 // 5.0      # haqiyqiy sonli bo’linma
2.0
>>> 10 .0 // 3.0     # haqiyqiy sonli bo’linma (qoldiqli)
3.0
>>> 10 // 5.0              # butun son haqiyqiy songa bo’linganda (qoldiqsiz)
2.0
```

```
>>>  10 // 3.0        # butun son haqiyqiy songa bo'linganda(qoldiqli)
3.0
>>>  10 .0 // 5       # haqiyqiy son butun songa bo'linganda(qoldiqsiz)
2.0
>>>  10 .0  11  3    # haqiyqiy son butun songa bo'linganda(qoldiqli)
3.0
```

- % - qoldiqli bo'lish (bo'linma qoldiq qismi):

```
>>> 10 % 5                  # butun sonli qoldiqsiz bo'linma
0
>>> 10 % 3           #  butun sonli qoldiqli bo'linma
1
>>> 10.0 % 5.0     #  haqiyqiy sonlar ustida amal
0.0
>>> 10.0 % 3.0     #  haqiyqiy sonlar ustida amal
1.0
>>>  10  %  5.0     #  butun va haqiyqiy sonlar ustida amal
0.0
>>> 10 % 3.0        #  butun va haqiyqiy sonlar ustida amal
1.0
>>>  10.0 % 5       #  haqiyqiy va butun sonlar ustida amal
0.0
>>>  10.0  %  3     #  haqiyqiy va butun sonlar ustida amal
1.0
```

- ** – darajaga ko'tarish:

```
>>>  10**2,  10.0**2
(100,  100.0)
```

- Unar minus (-) va unar plyus (+) :

```
>>>+10,  +10.0,  -10,  -10 .0 ,  -( - 10) ,  -( -10 .0)
(10,  10.0,  -10,  - 10.0,  10, 10 .0)
```

3.2. Ikkilik operatorlar

Pythonda quyidagi ikkilik (bitli) operatorlar ishlatiladi:

- ~ ikkilik inversiya. Har bir bit qarama qarshisiga o'zgaradi:

```
>>> x = 100              #  011 001 00
>>> x= ~x                #  100 110 11
```

- & – ikkilik VA:

```
>>> x  = 100             #  01100100
>>> y  = 75              #  01001011
>>> z  = x & y           # 01000000
>>> "{0 :b } & {l: b}={2:b}".format (x, y , z)
'1100100 & 1001011 = 1000000 '
```

- | – ikkilik YOKI:

```
>>>  x  =  100                  #  01100100
>>>  y  =  75                   #  01001011
>>>  z  =  x | y                #  01101111
>>> "{0:b } | {l: b } = {2: b }".format (x,  y,  z )
'1100100 | 1001011  =  11011 11 '
```

- ^ - ikkilik istisnoli YOKI:

```
>>>  x  =  100           # 01100100
>>>  y  =  250           # 11111010
>>>  z  =  x ^ y         # 10011110
>>>  "{ 0: b} ^{l :b}  =  {2 :b} ".format (x,  y,  z )
'1100100  ^ 11111010  = 1001111 0 '
```

- << – chapga siljitish –ikkilik ko'rinishdagi soni chapgabir yoki bir nechta o'ringa siljitish va o'ng tomonni nol bilan to'ldirish:

```
>>>  x  =  100           #  01100100
>>>  y  =  x  << 1       #  11001000
>>>  z  =  y  << 1       #  10010000
>>>  k  =  z  << 2       #  01000000
```

➢ >> – o’ngga siljitish – ikkilik ko’rinishdagi sonni o’ngga bir yoki bir nechta o’ringa siljitish va agar son musbat bo’lsa, chap tomonni nol bilan to’ldirish:

```
>>> x  =  100              #  01100100
>>> y  =  x >> 1           #  00110010
>>> z  =  y >> 1           #  00011001
>>> k  =  z >>  2          #  00000110
```

Agar son manfiy bo’lsa, chap tomondagi razryadar bir bilan to’ldiriladi:

```
>>> x = -127               # 10000001
>>> y = x  >> 1            #  11000000
>>> z = y  >> 2            #  11110000
>>> k = z  << 1            #  11100000
>>> m = k  >> 1            #  11110000
```

3.3. Sonli ketma-ketliklar bilan ishlash operatorlari

Sonli ketma-ketliklar bilan ishlash operatorlari quyidagicha:

➢ + - konkatenasiya:

```
>>> print ("1-satr"  +  "2-satr")  #  satrlar konkatenasiyasi
1-satr2-satr
>>> [1,  2,  3]  +  [4,  5,  6 )       #  ro’yxatlar
[1,  2,  3,  4,  5,  6]
>>>  (1,  2,  3)  +  ( 4,  5,  6)      #  Kortejlar
(1,  2,  3,  4,  5,  6)
```

➢ * – takrorlash:

```
>>>  "s"  *  20              #  Satr
'sssssssssssssssssss'
>>> [1,  2]* 3              # Ro’yxat
[1, 2, 1, 2, 1, 2]
>>>  (1,  2) * 3             #  Kortej
(1,  2,  1,  2,  1,  2)
```

- in – tegishlilikka tekshirish. Agar element sonli satr tarkibida bo'lsa natija sifatida True ni qaytaradi:

```
>>> "satr" in "Qidirish uchun satr" # Satr
True
>>> "2-satr" in "Qidirish uchun satr" # Satr
False
>>> 2 in [1, 2, 3], 4 in [1, 2, 3] # Ro'yxat
(True , False)
>>> 2 in (1 , 2, 3), 6 in (1, 2, 3) # Kortej
(True , False)
```

- not in – tegishli emaslikka tekshirish. Agar element sonli ketma-ketlik tarkibida bo'lmasa True natija qaytaradi:

```
>>> "satr" not in "Qidirish uchun satr"          # Satr
False
>>> "2-satr" not in "Qidirish uchun satr"        # Satr
True
>>> 2 not in [1, 2, 3], 4 not in [1, 2,]        # Ro'yxat
(False, True )
>>> 2 not in (1, 2, 3), 6 not in (1 , 2, 3)     # Kortej
(False, True)
```

3.4. O'zlashtirish operatorlari

O'zlashtirish operatorlari qiymatlarni o'zgaruvchilarga saqlash uchun foydalaniladi. Pythonda foydalaniladigan o'zlashtirish operatorlarini ko'rib chiqamiz:

= – o'zgaruvhciga qiymat o'zlashtirish:

```
>>> x = 5; x
5
```

- += – o'zgaruvchi qiymatini ko'rsatilgan kattalikka oshirish:

```
>>> x = 5;
>>> x += 10              # x=x+10 ga teng kuchli
>>> x
```

Konkatenasiya amalini o’zlashtirish uchun ham foydalanish mumkin:

```
>>> s = "Ol" ;
>>> s += "mos"
>>> print (s)
Olmos
```

- - = – o’zgaruvchi qiymatini ko’rsatilgan kattalikka kamaytirish:

```
>>> x = 10; x - = 5   # x = x - 5 ga teng kuchli
>>> x
5
```

- * = – o’zgaruvchi qiymatini ko’rsatilgan kattalikka ko’paytirish:

```
>>> x = 10; x * = 5   # x = x * 5 ga teng kuchli
>>> x
50
```

Sonli ketma ketliklar uchun ham * = amali qo’llaniladi:

```
>>> s = "* "; s * = 20
>>> s
' ********************* '
```

- / = – o’zgaruvchi qiymatini ko’rsatilgan kattalikka bo'lish:

```
>>> x = 10 ; x /= 3            # x = x / 3 ga teng kuchli
>>> x
3. 3333333333333335
>>> y = 10 .0 ; y /= 3.0              # y = y / 3.0 ga teng kuchli
>>> y
3. 3333333333333335
```

- / /= – o’zgaruvchi qiymatini ko’rsatilgan songa bo’linadi, kami bo’yicha yaxlitlanadi va o’zlashtiriladi:

```
>>> x = 10; x //= 3   # x = x // 3 ga teng kuchli
>>> x
3
```

```
>>> y= 10 .0 ;  y //=  3.0          #  y=  y  //  3.0 ga teng kuchli
>>>  y
3.0
```

➢ % = – o’zgaruvchi qiymatini ko’rsatilag songa bo’ladi va qoldiq qismi o’zlashtiriladi:

```
>>>  x  =  10;  x  %=  2          #  x  =  x  %  2  ga teng kuchli
>>>  x
0
>>>  y  =  10;  y  %=  3          #  y  =  y  %  3 ga teng kuchli
>>>  y
1
```

➢ ** = – o’zgaruvchi qiymatini ko’rsatilgan darajaga ko’taradi va o’zlashtiriladi:

```
>>>  x  =  10 ;  x  * * =  2                #  x  =  x  * *  2 ga teng kuchli
>>>  x
100
```

3.5. Operatorlarni bajarishda ustuvorlik

Operatorlar bajarilish ustuvorligini kamayish tartibida yozamiz:

1. -x, +x, -x, ** – unar minus, unar plyus, ikkilik inversiya, darajaga ko’tarish. Agar unar operator * * operatoridan chapda joylashgan bo’lsa, darajaga ko’tarish katta ustuvorlikka ega, agar o’ngda joylashgan bo’lsa, kichik ustuvorlikka ega. Maslan:

- 10 * * -2 ifodasi quyidagi qavsli ifodaga teng kuchli:

- (10 ** (-2))

2. *, %, /, // – ko’paytirish(takrorlash), qoldiqli bo’lish, bo’lish, kami bo’yicha yaxlitlab bo’lish.

3. +, - – qo’shish (konkatenasiya), ayirish.

4. <<, >> – ikkilik siljitish.

5. & – ikkilik VA.

6. ^ – ikkilik istisnoli YOKI.

7. | – ikkilik YOKI.

8. =, +=, -=, *=, /=, //=, %=, ** = – o’zlashtirish

Eslatib o’tamiz! Qavslar yordamida ifoda hisoblanishdagi ustuvorlik tartibini o’zgartirish mumkin.

Muhokama uchun savollar va topshiriqlar

1. Pythonda operatorlar?
2. Pythonda matyematik operatorlar
3. Pythonda munosabat operatorlari.
4. Pythonda ketma-ketliklar bilan ishlash operatorlari.
5. Pythonda o’zlashtirish operatorlari.
6. Pythonda operatorlarni bajarish ketma-ketligi.

4-BOB. SHARTLI OPERATORLAR

O`quv modullari

Python, shartli opearatorlar, mantiqiy ifoda, taqqoslash, mantiqiy qiymat, interpretator.

Shartli opearatorlar tegishli mantiqiy ifodaning qiymatiga ko'ra dasturning alohida bir qismini bajarilish yoki bajarilmasligini ta'minlaydi. Mantiqiy ifoda faqat ikkita qiymat qabul qiladi: *true* (rost) yoki *false* (yolg'on), hisoblashda 1 yoki 0 sifatida qabul qilinadi.

```
>>> True + 2            # 1 + 2 ga teng kuchli
3
>>> False + 2      # 0 + 2 ga teng kuchli
2
```

Mantiqiy qiymat yoki ifodani o'zgaruvchida saqlash mumkin:

```
>>> x = True ; y = False
>>> x, y
(True , False )
```

Ixtiyoriy obyekt mantiqiy kontekstda rost (true) yoki yolg'on(False) shaklida interpretatsiya qilinadi. Mantiqiy qiymatni aniqlah uchun bool() funksiyasidan foydalanish mumkin. Quyidagi obyektlar True qiymatni qaytaradi:

- Nolga teng bo'lmagan ixtiyoriy son:

```
>>> bool (l), boo1 (20) , bool (- 20)
(True , True , True )
>>> bool (l.0), bool(0.1), bool (-20.0)
(True, True, True )
```

- Bo'sh bo'lmagan obyekt:

```
>>> bool ("0") , bool ([0, None] ), bool( (None, ) ), bool ({"x": 5})
(True, True, True, True)
```

Quyidagi obyektlar False kabi interpretasiyalanadi:

- Nolga teng son :

```
>>>  bool (0) ,  bool (0.0)
(False,  False)
```

- Bo’sh obyekt:

```
>>> bool (" "),  bool ([ ]),  bool (( ))
(False,  False ,  False )
```

- None qiymati:

```
>>> bool (None )
False
```

4.1. Taqqoslash operatorlari

Taqqoslash operatorlaridan mantiqiy ifodalarda foydalaniladi. Ularni sanab o’tamiz:

- = = – teng:

```
>>>  1 = = 1,  1 = = 5
(True ,  False)
```

- ! = - teng emas:

```
>>>  1  ! =  5,    1 ! = 1
(True ,  False )
```

- < – kichik :

```
>>>  1  <  5,  1  <  0
(True ,  False)
```

- > – katta :

```
>>> 1 >  0,  1 > 5
(True ,  False )
```

- <= - kichik yoki teng:

```
>>>  1 <= 5,   1 <= 0,   1  <=  1
(True ,  False,  True )
```

- >= - katta yoki teng:

```
>>>  1  >=  0,  1 >=  5,  1  >=  1
(True ,  False,  True )
```

- in – sonli ketma-ketlikka tegishlilikni tekshirish:

```
>>> "satr" in "qidirish uchun satr" # Satr
True
>>> 2 in [1 , 2, 3) , 4 in [1, 2, 3] # Ro'yxat
(True , False)
>>> 2 in (1, 2, 3) ' 4 in (1, 2, 3) # Kortejlar
(True , False )
```

in operatoridan shuningdek, lug'at kalitiga tegishlilikni tekshirishda ham foydalaniladi:

```
>>> "x" in {" x" : 1, "y" : 2}, "z" in {" x" : 1, "y" : 2}
(True , False )
```

- not in - sonli ketma-ketlikka tegishli emaslikni tekshirish:

```
>>> "satr" not in "qidirish uchun satr" # Satr
False
>>> 2 not in [1, 2, 3] , 4 not in [1, 2, 3]      # Ro'yxat
(False, True)
>>> 2 not in (1 , 2, 3) , 4 not in (1 , 2 , 3) # Kortej
(False , True )
```

- is – ikkita o'zgaruchi bir obyektga havola qilinganini tekshiradi. Agar o'zgaruvchilar bior obyektga havola qilingan bo'lsa, is operatori True qiymat qaytaradi:

```
>>> x = y = [1, 2)
>>> x is y
True
>>> x = [1, 2] ; y= [1, 2)
>>> x is y
False
```

Sonlar va uzun bo'lmagan satrlar bilan ishlashda interpretator samaradorligini oshirish uchun keshlashdan foydalanishi seziladi. Turli o'zgaruvchilarda saqlanadigan

ikkita bir xil qiymat bitta obyektda saqlanishini va o’zgaruvchilarda bir obyektga havola saqlanishini anglatadi. Masalan:

```
>>> x = 2; y = 2; z = 2
>>> x is y, y is z
(True , True )
```

- is not – ikki o’zgaruvchi turli obyektlarga havola qilinishini tekshiradi.

Agar shunday bo’lsa, True qiymat qaytaradi:

```
>>> x = y = [1, 2 ]
>>> x is not y
False
>>> x = [1 , 2] ; y= [1, 2)
>>> x is not y
True
```

Mantiqiy ifoda not operatori yordamida invertirlanadi (teskari qiymatga o’giriladi):

```
>>> x = 1; y = 1
>>> x = y
True
>>> not (x == y) , not x == y
(False , False )
```

Agar x va y o’zgaruvchilar teng bo’lsa True qiymat qaytariladi, not operatori qo’llanilgach esa ifoda False qaytaradi. Aylanama qavslarni qo’yish shart emas.

Mantiqiy ifodada birdaniga bir nechta shartni ko’rsatish mumkin:

```
>>> x = 10
>>> 1 < x < 20, 11 < x < 20
(True, False )
```

Bir nechta mantiqiy ifodalar bitta katta mantiqiy ifodaga quyidagilar operatorlar yordamida birlashtirilishi mumkin:

- and – mantiqiy VA. Agar x and y ifodadagi x False bo’lsa x ni, aks holda y qaytaradi:

```
>>> 1 < 5 and 2 < 5          # True and True == True
True
>>> 1 < 5 and 2 > 5          # True and False == False
False
>>> 1 > 5 and 2 < 5          # False and True == False
False
>>> 10 and 20 , 0 and 20, 10 and 0
( 20, 0, 0)
```

➢ or – mantiqiy YOKI. Agar x or y ifodada x False bo'lsa, y ni qolgan hollarda x ni qaytaradi:

```
>>> 1 < 5 or 2 < 5           # True or True == True
True
>>> 1 < 5 or 2 > 5                  # True or False == True
True
>>> 1 > 5 or 2 < 5           # False or True == True
True
>>> 1 > 5 or 2 > 5            # False or False == False
False
>>> 10 or 20, O or 20, 10 or o
(1 0, 20, 10)
>>> 0 or "" or None or [] or "s"
's'
```

Taqqoslash operatorlarni ustuvorligini kamayish tartibida sanaymiz:

1. <, >, <=, >=, =, ! =, <>, is, is not, in, not in.
2. not – mantiqiy inkor.
3. and – Mantiqiy VA.
4. or – mantiqiy YOKI.

4.2. if...else tarmoqlanish operatori

if...else tarmoqlanish operatori mantiqiy ifoda natijasi bog'liq holda dastur qismining bajarilishi yoki bajarilmasligini ta'minlaydi. Operator umumiy holda quyidagi ko'rinishda bo'ladi:

if <mantiqiy ifoda>:

<agar mantiqiy ifoda rost qiymat qabul qilsa bajariladigan operatorlar to'plami>

[elif <mantiqiy ifoda>:

<agar mantiqiy ifoda rost qiymat qabul qilsa bajariladigan operatorlar to'plami>

]

[else:

<agar mantiqiy ifoda yolg'on qiymat qabul qilsa bajariladigan operatorlar to'plami>]

Kodlar blokini yozishda probellar ahamiyatini unutmaslik kerak. Quyidagi misolni ko'rib chiqamiz:

4.1-misol. Sonni juftlikka tekshirish

```
#  -*- coding :  ut f-8  - *-
x  =  int(input("Sonni kiriting:"))
if x % 2 == 0:
print (x, " – juft son")
else:
print (x, "  - toq son")
input ()
```

Agar blok bitta insturuksiyadan iborat bo'lsa, uni bir satrga yozish mumkin:

```
# -*-coding:utf-8-* -
x  = int(input("Sonni kiriting: ") )
if x%2 == 0: print (x , " – Juft son" )
else: print (x, " – Toq son ")
input ( )
```

Bu holda satr ohiri instruksiya ohiri hisoblanadi. Blok ichida bir nechta operator bo’lsa ularni bir satrga nuqtali vergul bilan ajratib yozish mumkin:

```
- * - coding:  utf-8 - * -
x = int(input ("Sonni kiriting:"))
if x%2 == 0: print(x , end=" " ) ; print (" – juft son" )
else: print (x, end=" " ) ; print ( " – toq son" )
input( )
```

Eslatma! Amalda operatorlarni bir satrga nuqtali vergul bilan ajratib yozish tavsiya etilmaydi.

if ..else operatori birdaniga bir nechat shartlarni tekshirishga imkon beradi. Misol orqali ko’rib chiqamiz (4.2-misol)

4.2-misol. Bir nechta shartlarni tekshirish

```
#  -*- coding : utf-8 - * -
print ("""Qaysi operatson tizimdan foyadalanasiz?
1 - Windows 8
2 - Windows 7
3 -Windows Vista
4 - Windows XP
5 - boshqa""")
os = input ("Javobingizga mos sonni kiriting:")
if os == "1":
print ("Siz Windows 8 ni tanladingiz")
elif os == "2":
print (" Siz Windows 7 ni tanladingiz")
elif os == "3":
print ("Siz Windows Vistani tanladingiz" )
elif  os == "4":
print ("Siz Windows XP ni tanladingiz")
elif os = = "5":
print ("Siz boshqa ni tanladingiz")
```

```
elif not os :
print ("Son kiritmadingiz")
else:
print ("Siz tanlagan OT ni aniqlay olmadik")
input( )
```

if ... else operatori yana bir boshqa formatda ham qo’llaniladi:

<O’zgaruvchi> = <rost> if <shart> else <yolg’on>

Misol:

```
>>>print ("Ha" if 10%2 == 0 else  "Yo’q" )
Ha
>>> s  =  "Ha"  if 10 % 2 == 0 else  "Yo’q"
>>> s
'Ha'
>>> s = "Ha" if 11% 2 ==0 else "Yo’q"
>>> s
'Yo`q'
```

Muhokama uchun savollar va topshiriqlar

1. Pythonda shartli operatorlar.
2. Pythonda taqqoslash operatorlari.
3. Pythonda if...else operatori.

5-BOB. SIKL OPERATORLARI

O`quv modullari

Sikl, iterasiya mexanizmi, break, funksiya**,** instruksiya, mantiqiy ifoda.

5.1. For sikli

for sikl operatoridan takrorlanishlarni hisoblash va ketma-ketliklar bilan ishlash uchun foydalaniladi. Umumiy ko’rinishi quyidagicha:

for <Joriy element> in <Ketma-ketlik> :

<Sikl ichki instruksiyalari>

[else:

<break operatoridan foydalanilmaganda bajariluvchi blok>

]

Bu yerda:

<Ketma-ketlik> – iterasiya mexanizmini qo’llovchi obyekt. Masalan: satr, ro’yxat, kortej, diapason, lug’at va boshqalar;

<Joriy element> – har bir iterasiyada ushbu parameter orqali ketma-ketlik yoki lug’atning joriy elementiga yo’l ko’rsatiladi;

<Sikl ichki instruksiyalari> – ko’p marta bajariluvchi blok;

Agar sikl ichida break operatori ishlatilmasa, sikl bajarilishi tugashi bilan else instruksiyasi bajarilishiga o’tiladi. Bu blok majburiy emas.

So’zdagi harflarni saralsh misolini ko’rib chiqamiz:

4.4-misol. So’zdagi harflarni saralash

```
for s in "Assalomu alaykum":
  print(s, end=" ")
else:
  print ("\nsikl bajarildi")
```

Natija:

```
A s s a l o m u  a l a y k u m
```

sikl bajarildi

Endi ro’yxat va korej har bir elementini alohida satrda chop etamiz (4.5-misol)

4.5-misol. Ro’yxat va kortejni saralash

```
for x in [1,2,3]:
  print(x)
for y in (1,2,3):
  print(y)
```

4.6-misol. Lug’at elementlarini saralash

```
>>> arr  =  {"x" :  1,  "y" :  2,  " z" :  3}
>>> arr.keys ()
dict_keys ([ 'y ',  'x',  'z'))
>>>for key in  arr.keys():          # keys () metodian foydalanilmoqda
print (key,  arr [key] )
y 2
x 1
z 3
>>>  for  key  in  arr:              # lug’atada iterasiyadan foydalanilmoqda
print (key, arr [key])
y 2
x 1
z 3
```

5.2. range() va enumerate() funksiyalari

Shu paytgacha biz sonli ktma ketliklar (satrlar) elementlarini chop etdik. Endi ro’yxat har bir elementini 2 ga ko’paytirib chop etamiz:

```
arr  =  (1,  2,  3)
for  i  in  arr:
i  =  i  *  2
print (arr)              #  Natija:  (1,  2,  3)
```

Ko’rinib turibdiki, ko’payrtma elementlarga ta’sir etmadi, ya’ni elementni o’ziga ta’sir etib bo’lmadi. Har bir elementga kirish range( ) funksiyasi yordamida amalga oshiriladi. U quyidagi formatga ega:

range ([<boshlang’ich qiymat>,] <Tugash> [, <Qadam>])

Birichi parametr boshlang’ich qiymatni aniqlaydi. Agar u ko’rsatilmasa, boshlang’ich qiymat 0 dan boshlanadi. Ikkinchi parameter tugash cheagarasini bildiradi. Bu parameter majburiydir. <Qadam> parametri ko’rsatilmasa qadam 1 deb qabul qilinadi. Misol sifatida ro’yxat har bir elementini 2 ga ko’paytitrishni ko’rib chiqamiz:

4.8-misol. range() funksiyasidan foydalanish misoli

```
arr=[1,2,3]
for i in range(len(arr)):
  arr[i]*=2
print(arr ) # Natija:  [2,  4,  6]
```

range() funksiyasidan foydalanisahga doir bir nechta misolarni ko’rib chiqamiz. 1 danm 10 gacha bo’lgan butun sonlarni chop etamiz:

```
for i in range (l, 101):  print (i)
```

Qiymatni nafaqat oshirib boorish balki, kamaytirish ham mumkin. 100 dan 1 gacha barcha sonlarni chop etamiz:

```
for  i  in  range (100,0, -1) :  print (i)
```

Shuningdek qadam faqat bittaga emas hohlagan songa o’zgarib borishi mumkin. 1 dan 100 gahca juft sonlarni chop etamiz:

```
for  i in range (2,  101,  2) : print(i)
```

5.3. While sikli

while siklidagi instruksiya mantiqiy ifoda rost bo’lganda bajariladi. While sikli quyidai formatga ega:

```
<Boshlang’ich qiymat>
while  <Shart>:
<Instruksiya>
```

< Orttirma >

[else:

<agar break operatori ishlatilmasa, bajariluvchi blok>

]

While siklining bajarilish tartibi:

1. O’zgaruvchi- hisoblagich boshlang’ich qiymat o’zlashtiradi.

2. Shart tekshiriladi, agar rost bo’lsa sikl ichidagi instruksiya bajariladi, aks holda sikl yakunlanadi.

3. O’zgaruvchi- hisoblagich <Orttirma> dagi parameter bo’yicha o’zgaradi.

4. 2 qadamga o’tiladi.

5. Agar sikl ichida break operatoridan foydalailmasa, sikl yakunlangach else bloki bajariladi. Bu blok majburiy emas.

1 dan 100 gacha sonlarni while siklodan foydalnib chop etamiz (4.11-misol).

4.11-misol. 1 dan 100 gacha sonlarni chop etish

```
i = 1 # <Boshlang’ich qiymat>
while i < 101 : # <Shart>
print(i) # <Instuksiya>
i += 1 # <Orttirma>
```

Eslatma! Agar orttirma ko’rsatilmasa, sikl cheksiz davom etadi. Cheksiz siklni to’xtatish uchun esa <Ctrl>+<C> klavish kombinasiayasdan foydalaniladi.

100 dan 1 gacha barcha butun sonlarni chop etamiz (4.12-misol).

4.12-misol. 100 dan 1 gacha sonlarni chop etish

```
i = 100
while i:
print (i)
i - = 1
```

E’tibor bergan bo’lsangiz, sikl shartida mantiqiy ifodadan foydalinmadi. Siklning har bir iterasiyasida sanagich-o’zgaruvchi qiymati bittaga kamayib boradi. Oqibatda u qachondir 0 ga teng bo’ladi. Bilamizki, 0 mantiqiy amal natijasi sifatida False ga teng kuchli. Undan boshqa ixtiyoriy son esa True ga teng kuchli.

While strukturasi yordamida turli struktura elementlari ustida ishlash mumki. Biroq bu holda while sikli for sikliga nisbatan sekinroq ishlaydi. Misol sifatida ro'yxatning har bir elementini 2 ga ko'paytirishni ko'rib chiqamiz (4.13-misol).

4.13-misol. Ro'yxaty elementlarini saralsh

```
arr = [1, 2, 3]
i, count = 0, len(arr)
while  i<count:
  arr[i]*=2
  i+=1
print(arr)              #  Natija:  [2,  4,  6]
```

5.4. continue operatori. Siklning navbatdagi itersiyasiga o'tish

continue operatori sikldagi barcha instruksiyalar bajarilmay, siklning navbatdagi iterasiyasiga o'tish imkonini beradi. Misol sifatida 1 dan 100 gacha bo'lagan butun sonlardan 5 dan 10 gacha oraliqqa kirmaydigan butun sonlar chop etishni ko'rib chiqamiz (4.14-misol).

4.14-misol. continue operatori

```
for i in range (l, 101):
if  4 < i < 11:
continue      # siklning keying iterasiyasiga o'tamiz
print (i)
```

5.5. break operatori. Siklni to'xtatish

break operatori sikl bajarilishini zudlik bilan to'xtatish imkonini beradi. 1 dan l00 butun sonlarni chop etishning yana bir usulini ko'rib chiqamiz (4.15-misol).

4.15-misol. Break operatori

```
i  = 1
while True:
if i > 100:
break #  siklni to'xtatish
```

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

Bu yerda biz shart qiymati sifatida True ni ko’rsatdik. Bu holda sikl cheksiz davom etadi. Biroq break operatoridan foydalanilgani sababli faqat 100 gacha sonlar chop etiladi.

Eslatma! break operatori dastur bajarilishi emas, sikl bajarilishini to’xtatadi va sikldan keyingi ko’rsatma bajariladi.

While siklini break operatori bilan birgalikda ishlatish oldindan ma’lumot miqdori noaniq bo’lgan hollarda ishlatish qulay. Misol sifatida noaniq miqdordagi sonlar yig’indisini hisoblasha dasturini ko’rib chiqamiz (4.16-misol).

4.16-misol. Noaniq miqdordagi sonlar yig’indisini hisoblash

```
print("Natiaja olish uchun 'stop'  so’zini kiriting")
summa=0
while True:
  x=input("Sonni kiriting:")
  if x == "stop":
    break
  x=int(x)
  summa+=x
print("Sonlar yig’indisi: ",  summa)
```

Natija:

```
Natiaja olish uchun 'stop'  so’zini kiriting
Sonni kiriting:10
Sonni kiriting:15
Sonni kiriting:30
Sonni kiriting:stop
Sonlar yig’indisi:  55
```

Muhokama uchun savollar va topshiriqlar

1. Pythonda sikl operatorlari.
2. For sikli.
3. range() va enumerate() funksiyalari
4. Pythonda while sikli.
5. Pythonda continue operatori.
6. Pythonda break operatori

6-BOB. SONLAR

O`quv modullari

modul, generasiya, kompleks son, obyekt, Fraction, funksiya**,** tasodifiy sonlar.

6.1. Sonlar bilan ishlashning tashqi funksiya va metodlari.

Python 3 dasturlash tili quyidagi sonli turlarni qo’llaydi:

- int - butun son (45, 6, -7, 0, 124). Son kattaligi faqat tezkor xotira sig’imiga bog’liq;
- float – haqiyqiy son (2.2101, 0.0025, 7.987456);
- complex – kompleks son (2+5j, 3+2j).

Butun son eng sodda tip, haqiyqiy son murakkabroq va eng murakkabi kompleks sondir. Shu tarzda agar amalda butun va haqiyqiy son ishtirok etsa, butun son avtomatik haqiyqiy songa o’tkaziladi so’ng haqiyqiy sonlar ustida amallar bajariladi. Bu amal natijasi haqiyqiy son bo’ladi.

Butun sonli obyektni oddiy usulda yarish mumkin:

```
>>> x = 0; y = 10; z = -80
>>> x, y, z
(0, 10, -80)
```

Bundan tashqari sonni ikkilik, sakkizlik yoki o’n oltilik shaklda ham ko’rsatish mumkin. Bunday sonlar avtomatik o’nlik butun songa o’zgartiriladi.

- 0b (yoki 0B) belgisidan boshlanuvchi hamda 0 va 1 raqamlari kombinatsiyasidan iborat ikkilik son:

```
>>> 0b11111111, 0b101101
(255, 45)
```

- 0 va o belgisidan boshlanuvchi hamda 0 dan 7 gacha raqamlar kombinatsiyasidan iborat sakkizlik son:

```
>>> 0o7 , 0o12, 0o777, 0o7 , 0o12, 0o777
(7, 10, 511, 7, 10, 51 1)
```

- 0x(yoki 0X) belgisidan boshlanuvchi hamda 0 dan 9 gacha raqamlar, A dan F gacha harflar kombinatsiyasidan iborat o’n oltilik son:

```
>>> 0x9, 0xA, 0x10, 0xFFF, 0xfff
(9, 10, 16, 4095, 40 95)
```

- Haqiyqiy son nuqta va (yoki) eksponsial ifodalash shaklidan iborat sonlar kombinasiyasi orqali ifodalanadi:

```
>>> 10., .14, 3.14, 11E20, 2.5e-12
(10.0, 0.14, 3.14, 1.le+21, 2.5e-12 )
```

Haqiyqiy sonlar ustida amallarni bajarishda hisob aniqligi chegarasini e’tiborga olish kerak bo’ladi.

Masalan, keyingi amal natijasi g’ayritabiy ko’rinishi mumkin:

```
>>> 0.3 - 0.1 - 0.1 - 0.1
-2 . 77555 75615 62891 4e-17
```

Biz aslida 0.0 natijani kutgan edik, lekin biz kurib kurganimizdek boshqa natijaga ega boldik. Agar fiksirlangan natijaga erishmoqchi bo’lsak decimal modulidan foydalanishimiz mumkin:

```
>>> from decimal import Decimal
>>> Decimal("0.3")-Decimal("0.1")-Decimal("0.1")-Decimal("0.1")
Decimal('0.0')
```

Bundan tashqari fractions moduli yordamida kasr sonni ham ifodalash mumkin. Kasr soni yaratishda ikki sonni yani surat va mahrajni bitta son yoki satr sifatida ko’rsatish mumkin.

Misol sifatida 4/5 kasr sonini yaratamiz:

```
>>> from fractions import Fraction
>>> Fraction (4, 5)
Fraction (4, 5)
```

½ kasr sonini quyidagi uchta usul bilan yaratish mumkin:

```
>>> Fraction (l, 2)
Fraction (l, 2)
>>> Fraction ("0.5" )
```

Fraction (l, 2)

>>> Fraction (0 .5)

Fraction (l, 2)

Kasr sonlar ustida oddiy sonlar kabi arifmetik amallar bajarish mumkin:

>>> Fraction (9, 5) - Fraction (2, 3)

Fraction (l7, 15)

>>> Fraction ("0.3") - Fraction ("0.1") - Fraction (" 0.l ") - Fraction (" 0.1 ")

Fraction (0,1)

>>> float (Fraction (0, 1))

0.0

<Haqiyqiy qism> + <Mavhum qism>J

Bu yerda J ixtiyoriy registrda (katta yoki kichik) bo'lishi mumkin. Kompleks sonlarga doir misollar:

>>> 2+ 5J, 8j

((2+ 5j), 8j)

6.2. Sonlar bilan ishlashning tashqi funksiya va metodlari

Sonlar bian ishlash uchun quyidagi tashqi funksiyalar mo'ljallangan:

- int ([<Obyekt>[, <Sanoq sistemasi>]]) – obyektni butun sonaga o'tkazish. Ikkinchi parametr sanoq sistemasini anglatib, ko'rsatish majburiy emas (odatiy holda 10 lik deb qabul qilinadi). Masalan:

>>> int(7. 5), int ("71", 10) , int ("0o71", 8) , int ("0xA", 16)

(7, 71, 57, 10)

>>> int(), int ("0b11111111", 2)

(0, 255)

- float ([<Son yoki satr>]) - son yoki satrni haqiyqiy songa o'tkazish. Masalan:

>>> float (7), float ("7.1") , float ("12.")

(7 .0, 7.1, 12.0)

>>>float ("inf") , float ("-Infinity"), float ("nan")

(inf, -inf, nan)

>>> float ()

0.0

- bin (<Son>) – o'nlik sonni ikkilikka o'tkazish. Sonlarni satr ko'rinishida taqdim qilinadi. Masalan:

>>> bin(255), bin(l) , bin(- 45)

('0b1111111' , '0b1' , '-0b101101')

- oct (<Son>) - o'nlik sonni sakkizlik songa o'tkazish. Sonlarni satr ko'rinishida taqdim qilinadi. Masalan:

>>> oct(7), oct(8), oct(64)

('0o7 ' , '0ol0 ' , '0ol00')

- hex (<Son>) - o'nlik sonni o'n oltilik songa o'tkazish. Sonlarni satr ko'rinishida taqdim qilinadi. Masalan:

>>> hex (l0) , hex(l6) , hex (255)

('0xa' , '0xl0' , '0xff ')

- round (<Son> [, <Nuqtadan ketyingi belgilar soni>]) – kasr qismi 0.5 dan kichik bo'lsa sonni o'zi, katta bo'lsa eng yaqin katta songa aylantiriladi. Agar kasr qismi 0.5 ga teng bo'lsa, eng yaqin juft songa aylantiriladi. Masalan:

>>> round (0.49), round (0.50), round(0.51)

(0, 0, 1)

>>> round (1.49), round(l.50), round (l.51)

(1, 2, 2)

>>> round (2.49), round(2.50), round(2.51)

(2, 2, 3)

>>> round(3.49), round(3.50), round(3.51)

(3, 4, 4)

Ikkinchi parametrverguldan keying raqamlar sonini olish uchun ko'rsatiladi. Biroq majburiy emas, ko'rsatilmasa 0 deb qabul qilinadi (son butunga yahlitlanadi):

>>> round (l.524, 2), round (l.525, 2), round (l.5555, 3)

(1.52, 1.52, 1.556)

- abs (<Son>) - sonning absolyut qiymatini qaytaradi:

>>> abs (-10) , abs (l0) , abs (-12.5)

(1 0, 10, 12 . 5)

- pow (<Son>, <Daraja> [, <Bo'luvchi>]) - <Son>ni <Daraja>ga ko'tarish:

>>> pow (l0, 2), 10**2, pow (3, 3), 3**3

(100, 100, 27, 27)

Agar vuchinchi parametr ko'rsatilsa, olingan natijani shu parametrga bo'lishda qoldiq qismi qaytariladi:

>>> pow(l0, 2, 2) , (10 ** 2)%2, pow(3, 3, 2), (3**3) % 2

(0, 0, 1, 1)

- max (<Vergul bilan ajratilga sonlar ro'yxati>) - ro'yxatdagi sonlarning eng kattasini aniqlash:

>>> max (l, 2, 3) , max (3, 2, 3, 1), max (l, 1.0) , max (l.0, 1)

(3, 3, 1, 1.0)

- min (<Vergul bilan ajratilga sonlar ro'yxati>) - ro'yxatdagi sonlarning eng kichigini aniqlash:

>>> min(l, 2, 3) , min(3, 2, 3, 1) , min (l, 1.0) , min (l.0, 1)

(1, 1, 1, 1.0)

- sum (<Ketma-ketlik> [, <Boshlang'ich qiymat>]) - ketma-ketlik (masalan: ro'yxat, kortej) elementlari yig'indisini aniqlash plyus <Boshlang'ich qiymat>. Agar ikkinchi parametr ko'rsatilmasa, u 0 ga teng deb qabul qilinadi. Agar ketma-ketlik bo'sh bo'lsa, ikkinchi parametr qiymati olinadi. Masalan:

>>> sum ((10, 20, 30, 40)), sum ([10, 20, 30, 40])

(100 , 100)

>>> sum ([10, 20, 30 , 40] , 2) , sum ([ ], 2)

(102 , 2)

- divmod (x, y) - ikki qiymatli (birinchi qiymat –bo'linma, ikkinchi qiymat–qoldiq (x//y, x%y)) kortej qaytaradi:

>>> divmod (l3, 2) # 13 == 6 * 2 + 1

(6, 1)

>>> 13//2, 13%2

(6, 1)

>>> divmod (13.5, 2.0) # 13 .5 == 6.0 * 2.0 + 1.5

(6 .0, 1.5)

>>> 13.5 // 2.0, 13 .5%2.0

(6.0, 1.5)

6.3. math moduli. Matematik funksiyalar

math moduli sonlar bilan ishlash uchun qo'shimcha funksiyalarni o'z ichiga oladi. Bu modulan foydalanishdan oldin quyidagi instruksiya bo'yicha chaqirib olinadi:

import math

Kompleks sonlar bilan ishlashda cmath kutubxonasidan foydalanish zarur. Math moduli quyidagi standart konstantalarni taqdim qiladi:

- pi – pi sonini qaytaradi:

>>> import math

>>> math.pi

3.141592653589793

- e – e konstantasi qiymatini qaytaradi:

>>> math.e

2.718281828459045

Sonlar bilan ishlashning asosiy funksiyalarni sanab o'tamiz:

- sin(), cos(), tan() – standart trigonoometrik funksiyalar (sinus, kosinus, tangens). Qiymatlar radianda beriladi;
- asin (), acos (), atan () - teskari trigonometric funksiyalar (arksinus, arkkosinus,arktangens). Natija radianda qaytariladi;
- degrees () – radiandan gradusga o'tkazish:

>>> math.degrees (math.pi)

180 .0

- radians () – gradusdan radianga o’tkazish:

>>> math.radians(l80.0)

3.141592653589793

- exp() - eksponenta;
- log (<Son> [, <Asos>]) - ko’rsatilga asos bo’yicha logarifm. Agar asos ko’rsatilmasa natural logarifm deb hisoblanadi (e asosga ko’ra);
- log10 () – o’nlik logarifm ;
- log2 () – 2 asosga ko’ra logarifm;
- sqrt () – kvadrat ildiz:

>>> math.sqrt(100), math.sqrt(25)

(10.0, 5.0)

- ceil () – eng yaqin katta songa yaxlitlash:

>>> math.ceil(5.49), math.ceil(5.50), math.ceil(5.51)

(6, 6, 6)

- floor () - – eng yaqin kichik songa yaxlitlash:

>>> math.floor(5.49), math.floor (5.50), math.floor(5.51)

(5, 5, 5)

- pow (<Son>, <Daraja>) –<Sonni> ni <Daraja>ga ko’tarish:

>>> math.pow(10, 2), 10**2, math.pow(3, 3), 3**3

(100.0, 100, 27.0, 27)

- fabs () – absolyut qiymat:

>>> math.fabs(10), math.fabs(-10) , math.fabs(-12.5)

(10.0, 10.0, 12.5)

- fmod() – qoldiqli bo’lish:

>>> math.fmod (l0, 5) , 10%5, math.fmod (l0, 3), 10%3

(0.0, 0, 1. 0, 1)

- factorial () – son faktoriali:

>>> math.factoria1(5), math.factorial(6)

(120, 720)

➢ fsum (<Sonlar ro'yxati>) - berilgan ro'yxatdagi barcha sonlar yig'indisini aniqlash:

> > > sum ([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])

0. 9999999999999999

>>> fsum ([.1, .1 , .1 , .1 , .1, .1 , .1 , .1 , .1 , .1])

1.0

Eslatma! Biz bu bo'limda faqat asosiy funksiyalarni ko'rib o'tamiz. Agar funksilarning to'liq ro'yxati zarur bo'lsa, math moduli dokumentasiyasiga murojaat qiling.

6.4. random moduli. random moduli. Tasodifiy sonlar generasiyasi

random moduli tasodifiy sonlarni generasiya qilishga imkon beradi. Moduldan foydalanishdan oldin quyidagi instruksiya yordamida bog'lanish amalga oshiriladi:

import random

Uning asosiy funksiyalarini sanab o'tamiz:

➢ random() – 0.0 dan 1.0 gacha tasodifiy sonlarni qaytaradi:

>>> import random

>>> random.random ()

0. 9753144027290991

>>> random.random ()

0.5468390487484339

>>> random.random()

0. 13015058054767736

➢ seed ([<Parametr>] [, version=2]) - tasodifiy sonlar generatorini yangi ketmaketlikka sozlash. Agar birinchi element ko'rsatilmasa, tasodifiy sonlar bazasi sifatida tizim vaqti olinadi. Agar birinchi parametr qiymati bitta bo'lsa, bitta songacha generasiyalanadi.

>>> random.seed (10)

>>> random.random ()

0. 5714025946899135

>>> random.seed(l0)

>>> random.random()

0. 5714025946899135

- uniform (<Boshlanish> , <Tugash>) - <Boshlanish> dan <Tugash> gacha oraliqdagi tasodifiy haqiyqiy sonni qaytaradi:

>>> random.uniform (0, 10)

9. 965569925394552

>>> random.uniform (0, 10)

0.4455638245043303

- randint (<Boshlanish> , <Tugash>) - <Boshlanish> dan <Tugash> gacha oraliqdagi tasodifiy butun sonni qaytaradi:

>>> random.randint (0, 10)

4

>>> random.randint (0, 10)

10

- randrange ([<Boshlanish>,] <Tugash> [, <Qadam>]) - sonli ketma-ketlikdan tasodifiy elementni qaytaradi. Parametrlari range() funksiyalari analogidir. Ya'ni, <Boshlanish> dan <Tugash>gacha bo'lgan sonlar orlaig'idan <Qadam> farqi bilan tasodifiy sonni qaytarardi:

>>> random. randrange(10)

5

>>> random.randrange (0, 10)

2

>>> random.randrange (0, 10, 2)

6

- choice (<Ketma-ketlik>) – berilgan ketma-ketlik (satr, ro'yxat, kortej) dagi tasodifiy elementni qaytaradi:

>>> random.choice ("string") # Satr

'i'

>>> random.choice (["s", "t", "r"]) # Ro'yxat

```
'r'
>>> random.choice((" s", "t", "r"))                    # Kortej
't '
```

- shuffle (<Ro'yxat> [, <0.0 dan 1.0 gacha sonlar>]) – ro'yxat elementlarini tasodifiy aralashtirish. Agar ikkinchi parametr ko'rsatilmasa, random() funksiyasi qiymatidan foydalaniladi. Bu holda hech qanday natija qaytarilmaydi. Faqat ro'yxatdagi elementlar tartibi o'zgaradi. Masalan:

```
>>> arr = [1, 2, 3,  4,  5,  6,  7,  8,  9, 10]
>>> random.shuffle(arr)
>>> arr
[8, 6, 9, 5, 3, 7, 2, 4, 10, 1]
```

- sample (<Ketma-ketlik>, <Elementlar soni>) –berilgan ketma-ketlikdan ko'rsatilgan miqdordagi tasodifiy holdagi elementlar ro'yxatini qaytaradi. Bunda obyekt sifatida iterasiyani qo'llovchi ixtiyoriy obektni ko'rsatish mumkin. Masalan:

```
>>> random.sample ("string",  2)
['i',  'r' ]
>>> arr =  [1,  2,  3,  4,  5,  6,  7,  8,  9,  10]
>>> random.sample(arr,  2)
[7, 10]
>>>  arr                                    # Ro'yxatning o'zi o'zgarmaydi
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>  random.sample (( l,  2,  3,  4,  5,  6,  7) ,  3)
[6 ,  3,  5]
>>>  random .sample (range(300), 5)
[126, 194, 272, 46, 71]
```

Berilgan uzunlikdagi parol genaratorini yaratish misolini ko'rib chiqamiz (5.1-misol). Buning uchun arr ro'yxatiga barcha ruxsat etilgan belgilarni qo'shamiz. So'ng choice() funksiyasi yordamida tasodifiy elementni olamiz. Odatiy holda parol uzunligini 8 belgidan iborat deb olinadi.

5.1-misol. Parol generator

```
#-*-coding:utf-8-*-
import  random # random modulini ulaymiz
def  passw_generator(count_char=8):
  arr = ['a', 'b',  'c', 'd','e','f','g', 'A', 'B' , 'C', 'D', 'E', 'F','G', 'H', 'I','J', 'K', 'L', 'M', 'N',  'O',  'P',  'Q', 'R' ,  'S' ,  'T',  'U', 'V', 'W', 'X', 'Y',  'Z', '1', '2','3', '4', '5', '6', '7', '8', '9', '0']
  passw = []
  for i in range(count_char):
    passw.append(random.choice(arr))
  return "".join(passw)
# funksiyani chaqiramiz
print (passw_generator(20)) #ngODHE8J8x ko’rinishada chiqishi mumkin
print (passw_generator())          #ZxcpkF50 ko’rinishada chiqishi mumkin
input ()
```

Muhokama uchun savollar va topshiriqlar

1. Pythonda sonlar.
2. Pythonda sonlar bilan ishlashning tashqi funksiya va metodlari.
3. Pythonda math moduli.
4. Pythonda matematik funksiyalar.
5. Pythonda random moduli. Tasodifiy son generasiyasi

7-BOB. SATRLAR VA ULAR USTIDA AMALLLAR

O`quv modullari

Satr, ulash(+operatori), kesish, takrorlash (* operatori), indeks bo'yicha murojaat, kombinasiaya, Satrlarni formatlash, Maydon, Satrni qidirish, almashtirish, Belgilar bilan ishlash.

Satr belgilarni ketma-ket berilishidan hosil bo'ladi. Satr uzunligi faqatgina konpyuter tezkor xotirasiga bog'liq cheklanadi.

Ketma-ketliklarga doir indeks bo'yicha murojaat, ulash(+operatori), kesish, takrorlash (* operatori), tegishlilikka tekshirish (in va not in operatori) kabi barcha amallar satrlar uchun ham amal qiladi.

Bundan tashqari, satrlar o'zgarmas ma'lumot tiplarga tegisli. Shuning uchun amaliy jihatdan barcha strlar bilan ishlash metodlari yangi satr sifatida qiymat qaytaradi.

Kichik satrlarni qayta ishlashda hech qanday muammo bo'lmasligi mumkin, biroq katta hajmli satrlar bilan ishlashda xotira yetishmovchiligi bilan bog'liq muammolar kelib chiqadi. Bir so'z bilan aytganda, belgini indeks bo'yicha o'qish mumkin lekin o'zgartirish mumkin emas (7.1-misol).

7.1-misol. Belgini indeks bo'yicha o'zgartirishga urinish

```
>>> s  =  "Python"
>>> s[0]              #  Indeks bo'yicha belgini o'qib olish mumkin
'P'
>>> s[0]  =  "J"   # Indeks bo'yicha satrni o'zgartirish mumkin emas
Traceback  (most  recent  call  last ) :
File  "<pyshell#2> ",  line  1,  in  <module>
s[0] = "J"              # Indeks bo'yicha satrni o'zgartirish mumkin emas
TypeError :  'str'  ob ject  does  not  support  item  assignment
```

Python 3 dasturlash tilida quyidagi satr turlari qo'llaniladi:

- str – Unicode - satr. Diqqat qiling aniq ko'dirovkalar: UTF-8, UTF-16 yoki UTF-32 – buyerda ko'rsatilmaydi.

```
>>> type ("Satr" )
<class 'str' >
>>> "Satr".encode (encoding = "cpl251")
b' \xfl \xf2 \xf0\xee\xea\xe0 '
>>> "satr".encode(encoding="utf- 8")
b' \xdl\x81\xdl\x82\xdl\x80\xd0\xbe\xd0\xba \xd0\xb0'
```

- bytes – o'zgarmas baytlar ketma-ketligi. Ketma-ketliknign har bir elementida belgi kodini ifodalovchi 0 dan 255 gacha butun son saqlanishi mumkin. bytes obyekt turi katta miqdordagi satrlarga tegishli metodlarni go'yoki belgilar ketma-ketligi kabi qo'llashi mumkin. Biroq indeks murojaat belgi emas butun son qaytaradi. Masalan:

```
>>> s =  bytes ("стр str", "cp1251")
>>> s[0],  s[5],  s[0:3],  s[4:7]
(241, 116, b'\xfl\xf2\xf0', b'str')
>>> s
b' \xfl\xf2\xf0  str'
```

bytes tipidagi ob'yekt bir baytli yoki ko'p baytli simvollardan iborat bo'lishi mumkin.

```
>>>  len ("строка ")
6
>>> len (bytes ("строка" ,  "cp125 1") )
6
>>>  len (bytes ("строка",  "utf-8") )
12
```

- bytearray – o'zgaruvchan baytlar ketma-ketligi. bytearray tipi bytes turi analogidir. Lekin elementni indeks bo'yicha o'zgartirish va qo'shimcha yaratish, o'chirish imkonini beradi. Maslan:

```
>>> s = bytearray ("str", "cp1251")   # Belgini o'zgartirish mumkin
```

```
>>> s[0] = 49; s
bytearray (b'1tr')
>>> s. append(55) ;  s
bytea rray  (b'1tr7')                    #  Belgi qo’shish mumkin
```

7.1. Satr yaratish

Satr yaratish quyidagi usullarda amalga oshiriladi:

- str funksiyasi yordamida([<Obyekt> [, <Kodirovka> [, <Xatoliklarni qayta ishlash>]]]). Agar faqat birinchi parameter ko’rsatilsa, satr ko’rinishdagi ixtiyoriy obyektni qaytaradi. Agar parametr umuman ko’rsatilmasa, bo’sh satr qaytariladi.

Masalan:

```
>>> str(),  str([l, 2]),  str((3, 4)),  str ({"x":1})
(' ', '[1, 2] ',  '(3, 4)',  "{'x':1}")
>>> str(b"\xfl \xf2\xf0\xee\xea\xe0")
"b'\\xfl\\xf2\\xf0\\xee\\xea\\xe0' "
```

- satrni apostrof yoki qo’shtirnoq orasida keltirish orqali:

```
>>> 'satr',  '"satr" ,  ' "x" :  5' ,  " 'x': 5"
('satr',  '"satr" ,  ' "x" :  5' ,  " 'x': 5" )
>>>  print ('1-satr\n2-satr ' )
1-satr
2-satr
>>> print ("1-satr\n2-satr " )
1-satr
2-satr
```

- satrni uchlangan apostrof yoki uchlangan qo’shtirnoq orasida keltirish orqali. Bunday obyektlar satrni bir nechta satrlarda joylashtirish va ular ichida qo’shtirnoq va apostroflardan foydalanish va chop etish imkonini beradi. Masalan:

```
>>>  print (' ' '1-satr
2-satr ' ' ')
```

```
1-satr
2-satr
>>>  print (" " "1-satr
2-satr " " ")
1-satr
2-satr
```

7.2. Maxsus belgilar

Maxsus belgilar – bu oddiy yo’ bilan qo’yib bo’lmaydigan xizmatchi yoki chop etilmaydigan belgilar kombinasiayasidir.

- \n – satrga o’tkazish;
- \r – katetka qaytarish;
- \t – tabulyatsiya belgisi;
- \v - vertical tabulyatsiya;
- \a – qo’ng’iroq;
- \b - zaboy;
- \f - formatni surish;
- \0 – nolli belgi (satr ohiri hisoblanmaydi);
- \ “- qo’shtirnoq;
- \ ‘ -apostrof;
- \N –N sakkizlik qimat. Masalan, \74 < belgisiga mos keladi;
- \xN – N o’n oltililiq qiymat. Masalan, \x6a satri j belgisiga mos keladi;
- \ \ - teskari slesh;
- \uxxxx - Unicode -16 bitli belgi. Masalan, \u043a ga kiril xarfiga mos keladi;
- \Uxxxxxxxx – Unicode 32 -bitli belgi.

7.3. Satrlar bilan ishlash amallari

Satrlar ketma-ketliklar kabi belgilarig aindeks orqali murojat qilish, qirqib olish, ulash, qayta takrorlash, tegishlilikka tekshirish kabi amallarni qo'llaydi. Bu amallarni batafsil ko'rib chiqamiz.

Satrning ixtiyoriy belgisiga murojat qilish ucun uning indeksini kvadrat qavs ichida keltirish etarli. Raqamlash noldan boshlanadi:

```
>>> s = "Python"
>>> s[0] , s[1] , s[2] , s[3] , s[4] , s[5]
('P', 'y', 't', 'h', 'o', 'n' )
```

Agar ko'rsatilgan indeksga mos keluvchi satrda bo'lmasa, IndexError istisnosi yuzaga keladi:

```
>>> s = "Python"
>>> s[10]
Traceback (most recent call last):
File "<pyshell#90 >", line 1, in <module>
s[l0]
IndexError: string index out of range
```

Indeks sifatida manfiy sonni ko'rsatish mumkin. Bu holda hisob teskari tartibdaya'ni satr oxiridan boshlanadi, ya'ni -1 elemen satr ohirga belgi bo'ladi. Masalan:

```
>>> s = "Python"
>>> s[-1] , s[len(s)-1]
('n', 'n' )
```

Bunday turdagi satrlar ozgarmas turga tegishli bo'lib, indeks bo'yicha belgini o'zgartirish mumkin emas:

```
>>> s = "Python"
>>> s[0] = "J"    # satrni o'zgartirish mumkin emas
Traceback (most recent call last):
File "<pyshell#94>" , line 1, in <module>
s [0] = "J"          # satrni o'zgartirish mumkin emas
```

TypeError: 'str' object does not support item assignment

Agar o’zgartirmoqchi bo’lsak, qatroni biror parchasini qirqib olishmiz mumkin. Amal formati:

[<Boshlanish>:<Tamom>:<Qadam>]

Bu yerda barcha parametrlar majburiy emas. Agar <Boshlanish> parametri ko’rsatilmasa, qiymat 0 deb qabul qilinadi. Agar <Tamom> ko’rsatilmasa, satr ohirigacha olinadi.

Agar <Qadam> ko’rsatilmasa, qiymat 1 deb olinadi. Parameter qiymati sifatida manfiy sonni ko’rsatish mumkin.

Misol orqali ko’rib chiqaylik. Boshlanishiga satrni nusxasini oalamiz:

```
>>> s = "Python"
>>> s[:]  # 0 pozitsiyadan satr ohirigacha satr qismini qaytaradi
'Python'
```

Endi belgilarni teskari tartibda chiqaramiz:

```
>>> s[: : -1] # <Qadam> parametri sifatida manfiy sonni ko’rsatamiz
'nohtyP '
```

Satrdagi birinchi belgini almashtiramiz:

```
>>>"J" + s[l:]        # 1 dan satr oxirigacha belgilarni olamiz va “J” ga ualaymiz
'Jython'
```

Oxirgi belgini o’chiramiz:

```
>>> s[:- 1]   # 0 dan len(s) -1 gacha satr qismini qaytaradi
'Pytho '
```

Satrdagi birinchi belgini olamiz:

```
>>> s[0:l ]   # indeksi 1 gacha satr chop etiladi
'P'
```

Endi ohirgi belgini olamiz:

```
>>> s[-1:]    # len(s)-1 dan satr ohirigacha qismni olamiz
'n'
```

Va nihoyat, 2,3,4 indeksdagi belgilarni chop etamiz:

```
>>> s[2:5]  #  2,  3 va  4 belgilar chop etiladi
'tho'
```

Satrdagi belgilar sonini aniqlashga len () funksiyasi yordam beradi:

```
>>> len("Python" ),  len (" \r\n\t " ),  len (r"\r\n\t")
(6, 3, 6)
```

Endi, for sikli yordamida barcha belgilarni saralab chop etamiz:

```
>>> s = "Python"
>>> for i in range (len(s)): print (s [i], end=" ")
```

Natija:

P y t h o n

Satr iterasiyani qo’llashni nazarda tutib biz shunchaki satrni sikl paramitri sifatida ko’rsatishimiz mumkin:

```
>>> s = "Python"
>>> for i in s:  print (i,  end=" ")
```

Natija quyidagicha:

P y t h o n

+ operatoridan foydalanib ikki satrni bittaga ulaymiz:

```
>>> print("1-satr"  +  "2-satr")
1-satr2-satr
```

Bundan tashqari, oshkormas holda qotrni ulashimiz mumkin. Bu holda ikki satr ular orasidagi operatorsiz ko’rsatiladi:

```
>>> print ("1-satr" "2-satr")
1-satr2-satr
```

Diqqat qiling, biz agar satrlar orasida vergul keltirilasa, biz natija sifatida satr emas kotrej olamiz:

```
>>> s  =  "1-satr ",  "2-satr "
>>> type (s)  # satr emas kotrej olamiz
<class  'tuple '>
```

Agar o’zgaruvchi va satr ulanadigan bo’lsa, albatta ulash belgisini qo’llash kerak, aks holda xatolik kelib chiqadi:

```
>>> s  =  "1-satr"
>>> print (s  +  "2-satr") #  Xatosiz
1-satr2-satr
>>> print ("s 2-satr") #  Xato
SyntaxError:  invalid syntax
```

Agar satrni boshqa turdagi ma’lumot (amaslan, son) bilan ulash zarur bo’lsa str () funksiyasidan foydalanish va konketanasiya amalini qo’llash kerak bo’ladi:

```
>>> "string" + str(10)
'string10'
```

Ko’rib chiqilgan amallardan tashqari satrlar uchun tegishlilikka tekshirish (in va not in operatorlari), takrorlash (* operatori) amallarini ham qo’llash mumkin:

```
>>> "-" * 20
'--------------------'
>>> "yt" in "Python"
True
>>> "yt" in "Perl "
False
>>> "PHP" not in "Python"
True
```

7.4. Satrni formatlash

Satrlarni formatlashda quyidagi shablondan foydlaniladi:

<Maxsus formatdagi satr> % <Qiymat>

<Maxsus formatdagi satr> parametri ichida quyidagicha formatdagi sintaksis ishlatiladi:

%[(<KALIT>)][<bayroq>][<Kenglik>][.<Aniqlik>]<O’zgartirish turi>

Satrdagi tasniflash miqdori <Qiymat> parametridagi elementlar miqdoriga teng bo’lishi mumkin. Agar faqat bitta spesifikatordan foydalanilayotgan bo'lsa, <Qiymat> ham bitta bo’lishi kerak.

Misol:

```
>>> " %s" %  10                          #  Bitta element
'10'
>>> "%s - %s - %s" % (10, 20, 30)    #  Bir nechta element
'10 - 20 - 30'
```

Spesifikator ichoidagi parametrlar quyidagi ma’nolarga ega:

- <KALIT> - lug’at kaliti. Agar kalit berilsa, <Qiymat> kortejni emas lug’atni ko’rsatishi kerak. Masalan:

```
>>> "% (name)s  - % (year)s" % {"year":1978, "name": 'Nik'}
'Nik - 1978'
```

- <Bayroq> - o’zgartirish bayrog’i. Quyidagi qiymatlardan iborat bo’lishi mumkin:

- <Aniqlik>:

```
>>> print ("%#o %#o %#o" % (0o77, 10, 10.5))
0o77  0o12  0o12
>>>  print ("%#x %#x %#x" % (0xff,  10,  10 . 5) )
0xff 0xa 0xa
>>> print ("% #.0F %.0F" % (300, 300))
 %  (0o77 ,  10,  10 . 5) )
%  (Oxff,  10,  10 . 5) )
%  (Oxff ,  10,  10 .  5) )
(3 00,  300) )
```

0 – oldingi qismni belgilangan uzunlikka yetguncha nollarga to’ldirish:

```
>>> “ ' %d '  - '%05d' % (3, 3)” # 5  - maydon kengligi
“ ‘3’ – ‘00003’ ”
```

- – maydon chap tomonidan tekislanishni belgilaydi. Odatiy holda o’ng tomondan tekislanadi. Agar bayroq bir vaqtda 0 bayroq bilan ko’rsatilsa, 0 bayrog’og’I bekor qilinadi. Masalan:

```
>>> " ' %5d ' – '%- 5d ' " % (3, 3)  #  5 -maydon kengligi
" '      3'  –  '3     ' "
>>>  " ' %05d '  - '% -05d' "  %  (3,  3)
```

" ' 00003 ' - ' 3 ' "

• probel- musbat sondan oldin probel qo'yiladi. Manfiy sondan oldin minus qo'yiladi. Masalan:

>>> " ' % d ' - ' % d ' % (-3, 3) "

" - 3 ' - 13 ' "

• + - agar musbat yoki manfiy ishoralar chop etilishi zarur bo'lsa qo'yiladi. Agar + boyrog'i probel bayrog'i bilan bir vaqtda ko'rsatilsa probel bekor qilinadi. Masalan:

>>> " ' % +d ' - ' %+d ' " % (- 3, 3)

" ' -3 ' - ' +3 ' "

<Kenglik> - maydonning minimal kengligi. Agar satr belgilangan kenglikda jaoylashtirilmasa, satr to'liq yoyilib chop etiladi:

>>> "'%10d'-'%-10d"'%(3,3)

"' 3'-'3 '"

>>>"'%3s %10s"' %("string", "string")

" 'string string' "

Qiymat o'rniga "*" belgisini ko'rsatish mumkin. Bu holda qiymat kortej ichida beriladi:

>>> " '%*s' '% l0s' " % (10, "string", "str")

" ' string' ' str' "

<Aniqlik> - haqiyqiy sonda nuqtadan keying raqamlar soni. Bu parameter oldida albatta nuqta bo'lishi kerak. Masalan:

>>> import math

>>> " %s %f %.2f " % (math.pi, math.pi, math.pi)

'3.141592653589793 3.141593 3.14 '

Qiymat o'rniga «* » belgisini ko'rsatish mumkin. Bu holda qiymat kortej ichida beriladi:

>>> " ' %*.* f ' " % (8, 5, math .pi)

" ' 3.14159 ' "

• <Qayta tiplash> -O'zgartirish turini belgilaydi. Bu parameter majburiy hisoblanadi.

<Qayta tiplash> parametrida quyidagi belgilar ko’rsatilishi mumkin:

- s – hohlagan obyektni str() funksiyasi yordamida satrga o’zgartiradi:

```
>>>  print ("%s"  % ("Oddiy satr" ))
Oddiy satr
>>> print ("%s  %s  %s"  %  (10,  10.52,  (1, 2, 3)))
10  10.52  (1,  2,  3)
```

- r – hohlagan obyektni repr() funksiyasi yordamida satrga o’zgartiradi:

```
>>> print (" %r " % ("Oddiy satr"))
'Oddiy satr'
```

- a – obyektni ascii() funksiyasi yordamida satrga o’zgartiradi:

```
>>>  print ("%a"% ("satr" ) )
'\u0441\u0442\u04 40\u0 43e\u04 3a\u0430'
```

- c – bitta belgi yoki son qiymatini belgiga o’tkazamiz. Misol sifatida son qiymatini va bu qiymatga mos belgini chop etamiz:

```
>>> for i in range (33, 127): print ("%s => %c" % (i, i))
```

- d va i – sonning butun qismini qaytaradi:

```
>>> print ("%d %d  %d" % (10, 25.6, - 80))
10  25  -80
>>> print("%i %i %i" %(10,25.6,-80))
10  25  -80
```

- o – sakkizlik qiymat:

```
>>> print ("%o %o %o" % (0o77, 10, 10.5))
77  12  12
>>> print ("%#0  %#0  %# 0"  %  (0o77, 10, 10.5))
Oo77  Oo12  Oo12
```

- x – quyi registrdagi o’n oltilik qiymat:

```
>>> print (" %x  %x  %x"  %  (0xff,  10,  10.5))
ffaa
>>>  print (" %#x  %#x  %#x"  %  (Oxff,  10,  10 .5 ) )
0xff  0xa  0xa
```

• x – yuqori registrdagi o’n oltilik qiymat:

```
>>>  print("%X %X %X"%(0xff, 10, 10.5))
FFAA
>>>  print ("%#X  %#X  %#X" % (0xff,  10,  10.5))
OXFF  OXA  OXA
```

7.5. format() metodi

Bundan tashqari formatlash uchun format () operatoridan foydalanish mumkin. U quyidagi formatda beriladi:

<Satr> = <Maxsus formatdagi satr>. format (*args,**kwargs)

<Maxsus formatdagi satr> parametri { va } belgilari ichida ko’rsatiladi va quyidagi sintaksisga ega:

{[<Maydon>] [!<Funksiya>] [: <Format>] }

Qavs ichida joylashgan barcha belgilar probelsiz chiqariladi. Agar satr ichida { va } belgilaridan foydalanishga to’g’ri kelsa, ikki mattadan yozish kerak, aks holda ValueError hatolik kelib chiqadi:

```
>>>  print ("{ { va } } - maxsus {0} ". format (" belgilar" ) )
{ va } - maxsus belgilar
```

<Maydon> parametrida pozitsiya indeksini(tartiblash noldan boshlanadi) yoki kalitni ko’rsatish mumkin. Aytaylik nomlanagan parametrlar yoki pozitsiyalar kombinasiyasi. Bu holda fomat() metodi nomlangan parametri o’zining oxirida ko’rsatiladi. Masalan:

```
>>> "{0} - {1} - {2}".format(10, 12.3, "string")
'10-12.3-string'
>>> arr = [10, 12 .3, "string"]            # Indeks
>>> "{0} - {1} - {2}".format (*arr)    # Indeks
'10 - 12.3 - string'
>>> "{model} - {color}".format(color = "red",  model = "BMW") #  kalit
'BMW - red'
>>> d = {"color": "red", "model": "BMW"}
```

```
>>> "{model}"
'BMW - red'
>>> "{color}"
'red - 2015 '
{color}".format (**d)                    #  Kalit
{0}".format(2015, color= "red" )         #  Kombinatsiya
```

7.6. Satrlar bilan ishlash metod va funksiyalari

Satrlar bilan ishlashda foydalaniladigan asosiy funksiyalarini ko'rib chgiqamiz:

- str([obyekt]) – ixtiyoriy obyektni satrga o'tkazadi. Agar paarmetr ko'rsatilmasa, bosh satrni qaytaradi. Obyektni chop etish uchun print() funksiyasidan foydalaniladi. Masalan:

```
>>> str( ) ,  str([l, 2]),  str((3, 4)),  str ({"x" : 1})
(' ', '[1,  2]',  '(3,  4)',  "{ 'x ' :  1}")
>>>  print ("1-satr\n2-satr")
1-satr
2-satr
```

- repr (<Obyekt>) – obyektni satr ko'rinishda qaytaradi. Python Shell IDLE redaktori oynasidan foydalanish mumkin. Masalan:

```
>>>  repr ("Satr"),  repr ([1, 2, 3]),  repr ({"x":5})
(" 'Satr' ",  ' [1, 2, 3]',  "{'x':5}")
>>>  repr (" 1-satr\n2-satr ")
" '1-satr\\2-satr' "
```

- ascii (<Obyekt>) – obyektni satr formatida qaytaradi. Satr faqat ASCII formatida beriladi. Masalan:

```
>>> ascii([l, 2, 3]), ascii ({"x": 5})
(' [1, 2, 3] ', "{'x': 5}")
>>> ascii ("satr")
'"\ \u04 41\\u04 42 \\u0440\\u043e\\u043a\\u04 30 "'
```

- len (<Satr>) - satrdagi belgilar sonini qaytaradi:

```
>>> len("Python"),  len("\r\n\t") ,  len (r"\r\n\t")
(6,  3,  6)
>>> len ("satr")
6
```

Satrlar bilan ishlashning asosiy metodlarni sanaymiz:

- strip([<belgilar>]) – qatar boshidagi va ohiridagi belgini o'chiradi. Agar parameter ko'rsatilmasa, probel belgilari o'chiriladi: probel, satrga o'tkazish belgisi (\n), karetka qaytarish belgisi (\r), gorizontal (\t) va vertikal (\v) tabulyatsiyalar:

```
>>>  sl,  s2 = "str\n\r\v\t",  "strstrstrokstrstrstr"
>>> " '%s'  - '% s' " % (sl.strip(),  s2.strip ("tsr" ))
" 'str' - 'ok' "
```

- lstrip ([<Belgi>]) – probel yoki satr boshidagi ko'rsatilgan belgilar o'chiriladi:

```
>>> sl,  s2  =  "  str  " ,  "strstrstrokstrstrstr"
>>> " '%s' – '%s' " % (sl.lstrip(), s2.lstrip("tsr" ))
" 'str   ' – 'okstrstrstr' "
```

- rstrip([<Belgilar>]) – probel yoki satr ohiridagi ko'rsaatilgan belgini o'chiradi:

```
>>> sl, s2 = "  str   " ,  "strstrstrokstrstrstr"
>>> " '%s' - '% s' " % (sl.rstrip() ,  s2.rstrip ("tsr"))
" ' str ' - ' strstrstrok ' "
```

- split ([<Bo'luvchi> [, <Limit>]]) – satrni ko'rsatilgan satrlarga bo'ladi. Agar birinchi parametr ko'rsatilmasa yoki None bo'lsa, bo'lish sifatida probel belgisidan foydalaniladi.

Misollar:

```
>>>  s  =  "word1 word2 word3"
>>>  s.split (),  s.split(None, 1)
(['word1', 'word2', 'word3'], ['wordl', 'word2 word3'])
>>> s = "wordl\nword2\nword3"
>>> s.split ("\n" )
```

[' wordl ', 'word2', 'word3 ']

Agar satrda birdaniga bir nechta probel bop'lsa va bo'luvchi ko'rsatilmasa, bo'sh element ro'yxatga kiritilmaydi:

```
>>> s = "wordl word2 word3 "
>>> s.split ()
 ['wordl', 'word2', 'word3']
```

Boshqa bo'luvchidan foydalanishda bo'sh elementlar paydo bo'lishi mumkin:

```
>>>  s  =  " ,   ,  wordl,  , word2, , word3, ,"
>>>  s.split (", ")
[' ', '  ', ' wordl', ' ', 'word2', '', 'word3', ',']
>>> "1,, 2, , 3".split (",")
['1', '', ' 2', ' ', ' 3']
```

Agar satrda bo'luvchi topilmasa, oldingi butun satrni o'zi qaytariladi:

```
>>> "wordl word2 word3".split ("\n")
['wordl word2 word3']
```

7.7. Localni sozlash

Lokal (tizimning barcha local sozlanmalini sozlash)ni o'rnatish uchun locale modulining set locale () funksiyasidan foydalaniladi.

Funksiyadan foydalanishdan oldin modulni quyidagicha chaqirib olish kerak bo'ladi:

```
irnport  locale
```

set locale ( ) funksiyasi quyidagi formatga ega:

setlocale (<kategoriya> [, <Lokal>));

<kategoriya> parametri quyidagi qiymatlarni qabul qiladi:

- locale.LC_ ALL -barch arejim uchun lokalni o'rnatish;
- locale.LC_ COLLATE – qatirni taqqoslash uchun;
- locale.LC_ CTYPE – belgiini quyi yoki yuqori registrga o'tkazish uchun;
- locale.LC_MONETARY – pul birligini aks ettirish uchun;
- locale.LC_NUMERIC – sonni formatlash uchun;

- locale.LC_ TIME – sana va vaqtni formatlash uchun.

Lokalning joriy qiymatini olish uchun getlocale ([<категория>)) funksiyasi qo'llaniladi. Misol sifatida Windows tizimida dastlab Windows-1251 kodirovkasi keyin utf-8 va ihtiyoriy hol sozlashni ko'rib chiqamiz. So'ng joriy sozlanma holatini barcha kategoriya va locale.LC_COLLATE uchun aniqlaymiz(6.3-misol).

7.3-misol. Lokalni sozlash

```
>>> import locale
>>> # windows -1251 kodirovkasi uchun
>>> locale.setlocale(locale.LC_ALL,  "Russian_Russia .1251" )
'Russian_Russia.1251'
>>> # Lokalni oddiy hol uchun o'rnatamiz
>>> locale.setlocale(locale.LC_ ALL ,  "")
' Russian__Russia.1251 '
>>> # Barcha kategoriya uchun joriy qiymatni olamiz
>>> locale.getloca le ()
(' Russian_Russia' ,  '1251')
>>>  # locale.LC  COLLATE kategoriyasi uchun joriy qiymatni olamiz
>>> locale.getlocale(locale.LC_COLLATE )
(' Russian_Russia', '1251')
```

Lokal sozlanmasini olish uchun localeconv () funksiyasi imkon beradi. Funksiya sozlanma lug'atini qaytaradi.

```
>>>  locale . localeconv ()
```

{'int_curr_symbol': 'RUB', 'currency_symbol': '?', 'mon_decimal_point': ',', 'mon_thousands_sep': '\xa0', 'mon_grouping': [3, 0], 'positive_sign': '', 'negative_sign': '-', 'int_frac_digits': 2, 'frac_digits': 2, 'p_cs_precedes': 0, 'p_sep_by_space': 1, 'n_cs_precedes': 0, 'n_sep_by_space': 1, 'p_sign_posn': 1, 'n_sign_posn': 1, 'decimal_point': ',', 'thousands_sep': '\xa0', 'grouping': [3, 0]}

7.8. Belgilar registrini o'zgartirish

Belgilar registrini o'zgartirish uchun quyidagi metodlardan foydalaniladi:

- upper() – satrdagi barcha belgilarni katta harfga o'tkazish:

```
>>>  print ("satr".upper())
SATR
```

- lower() – satrdagi barcha belgilarni kichik harfga o’tkazish:

```
>>>  print("SATR".lower())
satr
```

- swapcase ( ) – satrdagi barcha kichik harflarni katta harfga katta harflarni kichik harfga o’tkazish:

```
>>> print("SATR satr". swapcase ())
satr SATR
```

- capitalize() – satrdagi birinchi harfni katta harfga o’tkazadi:

```
>>>  print("satr satr".capitalize())
Satr satr
```

- title() – satrdagi har bir so’z boshidagi harfni katta harfga aylantiradi:

```
>>> s  =  "har bir so’z birinchi harfi kattaga aylanadi"
>>> print(s.title())
Har Bir So’z Birinchi Harfi Kattaga Aylanadi
```

7.9. Belgilar bilan ishlash funksiyalari

Alohida belgilar bilan ishlash uchun quyidagi funksiyalar ishlatiladi:

- chr(<Belgi kodi>) – ko’rsatilagan kodga mos belgi qaytaradi:

```
>>> print (chr(l055))
п
```

- ord (<Belgi>) – ko’rsatilgan belgi kodi qaytariladi:

```
>>> print (ord ("п"))
1055
```

7.10. Satrni qidirish va almashtirish

Satrda qidirish va almashtirishlar uchun quyidagi metodlar ishlatiladi:

- find () – satrdan satr ostini qidirish. Satrgan qidirilayotgan satr osti joylashgan positsiya raqamini qaytaradi. Agar satr osti topolmasa, 1- qiymati qaytariladi. Metod belgilar aregistriga bog’liq. Metod formati quyidagicha:

<Satr>.find(<Satr_osti>[, <Boshlanish> [, <Tugash>]])

Agar boshlanish ko’rsatilmasa qidirish satr boshidan boshglanadi (ya’ni 0 deb olinadi).

Agar <Boshlanish > va < Tugash > parametrlari ko’rsatilsa, amal shu qirqim oralig’ida bajariladi:

<Satr> [<Boshlanish >:< Tugash>] va satrning ushbu qismidan satr osti qidiriladi. Masalan:

```
>>> s = "misol misol Misol"
>>> s.find ("mis"),  s.find("Mis"),  s.find ("test" )
(0, 12, -1)
>>> s.find ("mis",  9),  s.find ("mis", 0, 6) ,  s.find ("mis",  6, 12 )
(-1, 0, 6)
```

- index() - find() metodining analogi hisoblanadi, biroq satr osti topilmasa ValueError xatoligi ro’y beradi. Metod formati quyidagicha:

<Satr>.index(<Satr_osti> [-, <Boshlanish> [, <Tugash>]])

Misol:

```
>>> s = "misol misol Misol"
>>> s.index ("mis "), s.index("mis",  7,  12),  s.index("Mis" , 1)
(0, 6, 12)
>>> s.index ("test" )
Traceback (most recent call last):
File "<pyshell#16>", line 1, in <module>
s.index("test")
ValueError: substring not found
```

- rfind() – satrdan satr ostini qidirish. Satrdan eng oxirida uchragan satr osti pozitsiyani qaytaradi. Agar satr osti topilmasa, -1 qaytariladi. Metod belgi registriga bog’liq. Quyidagi formatga ega:

<Satr>.rfind (<Satr_osti> [, <Boshlanishi> [, <Tugashi>)])

Agar boshlanish ko’rsatilmasa qidirish satr boshidan boshglanadi (ya’ni 0 deb olinadi).

Agar <Boshlanish > va < Tugash > parametrlari ko’rsatilsa, amal shu qirqim oralig’ida bajariladi:

<Satr> [<Boshlanish >:< Tugash>] va satrning ushbu qismidan satr osti qidiriladi. Masalan:

```
>>> s = "misol misol Misol Misol"
>>> s.rfind("mis"), s.rfind("Mis"), s.rfind("test" )
(6, 18, -1)
>>> s. find ("mis", 0, 6), s.find("Mis", 10, 20)
(0, 12)
```

- rindex () - rfind () metodining analogi bo’lib, farqi agar satr osti topilmasa ValueError qiymati qaytariladi. Metod formati:

<Satr> .rindex (<Satr_osti>[, <Boshlanish> [, <Tugash>]))

Masalan:

```
>>> s = "misol misol Misol Misol "
>>> s. rindex ("mis" ), s. ri ndex ("Mis" ), s. rindex ("mis", 0, 6)
(6, 18, 0)
>>> s.rindex ("test")
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    s.rindex ("test")
ValueError: substring not found
```

- count () – satrdagi satr ostlar soni ni qaytaradi. Agar satr osti satrda uchramasa 0 qiymati qaytariladi. Metod belgi registriga bog’liq. Metod formati:

<Satr>.count (<Satr_osti> [, <Boshlanish> [, <Tugash>)])

Masalan:

```
>>> s = "misol misol Misol Misol"
>>> s.count ("mis" ), s.count ("mis" , 6) , s. count ("Mis" )
(2, 1, 2)
>>> s.count (" test" )
0
```

- startswith () – satrni ko’rsatilgan satr osti bilan boshlanganlikka tekshirish. Agar boshlansa True, aks holda False qaytarailadi. Metob belgi registriga bog’liq. Quyidagi frormatga ega:

<Satr> . startswith (<Satr_osti> [, <Boshlanish> [, <Tugash>]])

Agar boshlanish ko’rsatilmasa qidirish satr boshidan boshglanadi (ya’ni 0 deb olinadi).

Agar <Boshlanish > va < Tugash > parametrlari ko’rsatilsa, amal shu qirqim oralig’ida bajariladi.

Masalan:

```
>>> s = "misol misol Misol Misol "
>>> s.startswith("mis"),  s. startswith("Mis" )
(True,False )
>>> s.startswith ("mis", 6), s.startswith("Mis",  14)
(False, True)
```

- replace () - qotordagi satr ostini boshqa ko’rsatilgan satr ostiga almashtirish va natijani yangi satr ko’rinishida ko’rsatish. Metod belgi registriga bog’liq. Metod formati:

<Satr>.replace(<Almashtirish uchun satr osti>, <Yangi satr osti> [, <Almashtirishlarning maksimal soni>])

Agar almashtirishlar soni ko’rsatilmasa, almastirish barcha topilgan satr ostilar uchun bajariladi.

Masalan:

```
>>> s = "Salom,  Anvar"
>>> print(s.replace("Anvar", "Umid"))
Salom, Umid
>>> print(s.replace ("anvar",  "umid" ))  # registrga bog’liq
Salom, Anvar
>>> s = "strstrstrstrstr"
>>> s.replace ("str" ,  "") ,  s. replace ("str" ,  "" , 3)
```

(' ', 'strstr')

7.11. Satr tarkibini tekshirish

Satr tarkibini tekshirish uchun quyidagi metodlardan foydalaniladi:

- isalnum() – agar satr faqat harf va (yoki) harflardan iborat bo'lsda True, aks holda False qaytaradi. Agar satr bo'sh bo'lsa, False qaytaradi. Masalan:

```
>>> "0123".isalnum(), "123abc" .isalnum(), "abcl23".isalnum ()
(True, True, True)
>>>"Satr".isalnum ()
True
>>> " ".isalnum(), "123 abc".isalnum() , "abc, 23".isalnum()
(False , False , False)
```

- isalpha ( ) - satrni faqat harfdan iboratlikka tekshirish. Agar satr faqat harflardan iborat bo'lsa True, aks holda False qaytaradi. Aagar satr bo'sh bo'lsa, False qaytaradi. Misol:

```
>>> "string".isalpha(),"satr".isalpha(), " ".isalpha ()
(True , True , False )
>>> "123abc" .isalpha(), "strstr" . isalpha() , "st, st".isalpha ()
(False, False , False)
```

- isdigit() – satrni faqat raqamdan iboratlikka tekshirish. Agar satr faqat raqamlardan iborat bo'lsa True qaytarialdi, aks holda False:

```
>>> "0123".isdigit(), "123abc".isdigit () , "abcl23".isdigit()
(True, False, False)
```

- isdecimal () – agar satr faqat o'nlik raqamdan iborat bo'lsa True, aks holda False qaytaradi. Masalan:

```
>>> "123".isdecimal() , "123str".isdecimal ()
(True, False )
```

- isnumeric () – agar satr faqat raqmlei belgidan iborat bo'lsaTrue qaytariladi, aks holda False. Diqqat qiling, raqmli belgi faqat ASC II kodirovkasidagi o'nli raqamlar balki, Rim raqamlari, kasr sonlar va boshqalarga ham taa'luqli. Masalan:

```
>>>  "\u2155".isnumeric () ,  "\u2155".isdigit ()
(True, False)
>>> print("\u2155")                # "1 /5" belgisi chop etiladi
```

- isupper () – satrdagi harflar faqat yuqori registrda bo'lsa True, aks holda False qaytariladi:

```
>>> "STRING".isupper(),  "SATR".isupper () ,  " ".isupper()
(True,  True ,  False )
>>> "STRINGl".isupper () ,  "SATR, 123" .isupper () ,  "123".isupper()
(True ,  True ,  False )
>>> "string".isupper(),  "STRing".isupper ()
(False,  False)
```

- islower () – agar satr faqat quyi registrdagi harflardan iborat bo'lsa True, aks holda false qaytaradi:

```
>>> "srting".islower(), "satr".islower(),  "".islower ()
(True, True, False)
>>> "stringl".islower() ,  "str, 123". islower(), "123".islower()
(True, True, False)
>>>"STRING".islower(),  "Satr".islower()
(False,  False )
```

- istitle ( ) - agar satrdagi barcha so'zlasgh bosh harfdan boshlansa True, aks holda False qaytaradi. Agar satr bo'sh bo'lsa ham False qaytariladi. Masalan:

```
>>>  "Str Str".istitle(), "Satr Satr" .istitle ()
(True ,  True )
>>> "Str Str 123" . istitle() ,  "Satr Satr 123" . istitle ()
(True, True )
>>> "Str  str" . istitle() ,  "Satr Satr" .istitle ()
(False, False )
>>> "".istitle(),  "123".istitle ()
(False, False)
```

- isprintable() – agar satr faqat chop etiluvchi belgilardan iborat bo’lsa True, aks holda False qaytaradi. Qayd etish kerakki, probel chop etiluvchi belgilarga tegishli.

Misollar:

```
>>> "123".  isprintable()
True
>>> "PHP  Python".isprintable()
True
>>> "\n".isprintable ()
False
```

- isspace () – agar satr faqat probelli belgilardan iborat bo’lsa True, aks holda False qaytaradi:

```
>>> "".isspace (), "\n\r\t".isspace(),  "str  str".isspace()
(False ,  True ,  False )
```

- isidentifier () – agar satr Pythonda o’zgaruvchi nomi siftida qo’llanilishi mumkin bo’lsa True, aks holda False qaytaradi:

```
>>> "s".isidentifier()
True
>>> "func". isidentifier()
True
>>> "123func".isidentifier()
False
```

isidentifier() metodi faqat til qoidasiga ko’ra nom sifatida qo’llash mumkinligini tekshiradi, U Python dagi kalit so’zlarga mosligini tekshirmaydi.

Buni aniqlash uchun keyword modulini e’lon qilish va iskeyword () funksiyasidan foydalanish kerak. U satrni kalit so’zlardan biriga mosligini tekshiaradi, tegishli bo’lsa True, aks holda False qaytaradi:

```
>>> keyword.iskeyword ("else" )
True
>>> keyword.iskeyword ("elsewhere" )
False
```

7.4-misol. Noma’lum miqdordagi sonlar yig’indisini aniqlash

```
#-*-coding:utf-8-*-
print("Natija olish uchun 'stop'  so’zini kiriting")
summa=0
while True:
  x = input("Sonni kiriting:")
  if x=="stop":
    break      # sikldan chiqish
  if x=="":
    print("Siz qiymat kiritmadingiz!")
    continue
  if x[0]== "-": # agar birinchgu belgi minus bo’lsa
    if not x[1:].isdigit(): # agar satrda raqmdan boshqa belgi qatnashsa
      print("Son kiritish kerak, satr emas!")
      continue
  else: #Agar minus bo’lmasa, to’liq satr tekshiriladi
      if not x.isdigit(): #  agar satrda raqmdan boshqa belgi qatnashsa
        print("Son kiritish kerak, satr emas!")
        continue
  x=int(x)            # satrni songa o’tkazamiz
  summa+=x
print ("Sonlar yig’indisi: ",  summa)
input()
```

```
Natija olish uchun 'stop'  so’zini kiriting
Sonni kiriting:5
Sonni kiriting:7
Sonni kiriting:8
Sonni kiriting:15
Sonni kiriting:25
Sonni kiriting:stop
```

Sonlar yig’indisi: 60

7.12. Bytes ma’lumot turi

str ma’lumot turi matnli ma’lumotni saqlash uchun juda yaxshi qo’l keladi. Biroq tasvirni qayta ishlash uchun nima qilish kerak?

Tasvir kodirovkadan iboprat emas, demak u Unicode satr sifatida akslanmaydi. Pythonda bu muammoni yechish uchun 0 dan 255 gacha butun sonlar ketma-ketligini saqlash imkonini beruvchi bytes va bytearray turlari kiritilgan. Har bir bunday son belgi kodini aniqlaydi. bytes ma’lumot turi satr singari o’zgarmas va ro’yxat kabiu o’zgaruvhcan bytearray turlarga ajratiladi.

bytes obyekt turini quyidagi usullardan birida yaratish mumkin:

- bytes ([<Satr>, <Kodirovka>[, <Xatoliklarni qayta ishlash>]]) funksiyais yordamida. Agar parametr ko’rsatilmasa, bo’sh obyekt qaytariladi. Satrni bytes turiga o’tkazish uchun kamida ikkita birinchi parametr berilishi zarur. Agar satr faqat birinchi parametr ko’rsatilsa xatolik yuz beradi. Masalan:

```
>>> bytes ()
b' '
>>>bytes("satr", "cp1251")
b' \xfl\xf2\xf0\xee\xea\xe0 '
>>> bytes("satr")
Traceback (most recent call last):
  File "<pyshell#51>", line 1, in <module>
    bytes("satr")
TypeError: string argument without an encoding
```

- apostraf, qo’shtirnoq, uch qo’shtirnoq yoki uch apastorf da satr oldidan b harfini ko’rsatish yordamida(registr ahamiyatsiz). Diqqat qiling, satr faqat ASCII kodirovkasida bo’lishi kerak. Boshqa hamma belgilar maxsus ketma-ketlikni tashkil etishi kerak:

```
>>>  b" string",  b' str ing ' ,  b" ""string" "" ,  b' " string ' ' '
(b 'string',  b'string' ,  b'string',  b'string')
>>>  b"қатор "
```

SyntaxError: bytes can only contain ASC II literal characters .

>>> b"\xfl\xf2\xf0\xee\xea \xe0"

b' \xfl\xf2\xf0 \xee\xea\xe0 '

- bytes turidagi bytes 0 dan 255 gacha butun sonlar ketma-ketligi orqali beriladigan bytes (<Ketma-ketlik>) funksiyasi yordamida. Agar sonlar diapazonga tegishli bo'lmasa valueError istisnosi paydo bo'ladi. Masalan:

>>> b = bytes ([225, 226, 224, 174, 170,160])

>>> b

b'\xel\xe2\xe0\xae\xaa\xa0'

7.13. Bytearray ma'lumot turi

bytearray ma'lumot turi bytes turning bir ko'rinishi hisoblanadi va undagi barcha metod va amallarni qo'llaydi. bytes turidan farqi bytearray turi obyektlarni o'zgartirishga yo'l qo'yib beradi.

7.14. Obyektlarni baytlar ketma-ketligiga o'tkazish

Obyektni baytlar ketma-ketligiga va yana obyektga o'tkazish uchun pickle moduli imkon beradi. Bu modulni quyidagicha biriktirib olish kerak:

import pickle

O'zgartirish uchun quyidagi ikkita funksiya ishlatiladi:

- **dumps (<Obyekt>(, <Protokol>] [, fix_imports=True])** - obyektni serializatsiya qiladi va maxsus formatdagi baytlar ketma-ketligini qaytaradi. Format ko'rsatilgan protokolga bog'liq: 0 dan 4 gacha;

Ro'yxat va kortejini o'zgartirish misolini:

>>> import pickle

>>> objl = (1, 2, 3, 4, 5] # Ro'yxat

>>> obj2 = (6 , 7, 8, 9, 10) # Kortej

>>> pickle.dumps (objl)

b' \x80\x03]q\x00 (K\x0 1K\x02 K\x03K\x0 4K\x05e.'

>>> pickle.dumps(obj2)

b'\x80\x03(K\x0 6K\x0 7K\x0 8K\ tK\ntq\x00.'

➢ **loads (<Baytlar ketma-ketligi>[, fix_imports=True] [, encoding="ASC II") [, errors="strict"])** - maxsus formatdagi baytlar ketma-ketligini obyektga qayta o'tkazadi. Ro'yxat va kortejni tiklash misolini ko'ramiz:

```
>>> pickle.loads (b'\x80\x03]q\x00(K\x01K\x02K\x0 3K\x0 4K\x05e.')
[1, 2, 3, 4, 5]
>>> pickle.loads (b' \x80\x03(K\x06K\x07K\x08K\tK\ntq\x00.')
(6, 7, 8,  9,  10)
```

7.15. Satrlarni shifrlash

Satrlarni shifrlash uchun hashlib modulidan foyfdalaniladi. Ushbu modul funksiyalaridan foydalanishdan oldin quyidagicha modulni bog'lab olish kerak:

```
import hashlib
```

Modul quyidagi funksiyalardan foydalanishni taqdim etadi: md5(), sha1(), sha224(), sha256(), sha384() va sha512(). Masalan:

```
>>> import hashlib
>>> h = hashlib.sha1(b"password" )
```

update() metodi yordamida baytlar ketma-ketligiga o'tkaziladi. Bu holda obyekt oldingi qiymatlari bilan bog'lanadi:

```
>>> h = hashlib.shal()
>>> h.update (b"password" )
```

Muhokama uchun savollar va topshiriqlar

1. Pythonda satrlar va ular ustida amallar.
2. Pythonda satr yaratish.
3. Pythonda maxusu belgilar.
4. Pythonda satrlar bilan ishlash amallari.
5. Pythonda satrlarni formatlash.
6. Pythonda format() metodi.
7. Pythonda satrlar bilan ishlash metod va funksiyalari.

8. Pythonda localni sozlash.
9. Pythonda belgila registrini o'zgartirish.
10. Pythonda belgilar bilan ishlash funksiyalari

8-BOB. MUNTAZAM IFODALAR

O`quv modullari

Muntazam ifoda, compile, modifikator, yangi qator belgisi, “nuqta” metasimvollari.

Muntazam ifoda satrdagi murakkab qidirish yoki almashtirishga imkon beradi. Python tilida muntazam ifodadan foydalanish uchun re modulidan foydalaniladi. Modul funksiyalaridan foydalanishdan oldin quyidagi ko’rsatma bo’yicha biriktiriladi:

import re

8.1. Muntazam ifoda sintaksisi

Muntazam ifodadan foydalanishga compile() funksiyasi shabloni imkon beradi. Funksiya quyidagi formatga ega:

<Shablon> = re.compile (< Muntazam ifoda> [, <Modifikator>])

<Modifikator> parametrida quyidagi bayroqlar (yoki ularning kombinasiayalari | operatori bilan beriladi) ko’rsatilishi mumkin:

- L yoki LOCALE – joriy lokalni sozlanmasini hisobga olish;
- r yoki IGNORECASE – registrni hisobga olmay qidirish. Masalan:

```
>>> import re
>>> p = re.compile(r"^[a -яе] +$",  re.I  |  re.U)
>>> print("Topildi" if p.search("АБВГ")  else  "Yo’q")
Topildi
>>> p = re.compile(r"^ [a-яе] +$", re.U)
>>> print ("Topildi" if p.search("АБВГДЕЕ") else "Yo’q" )
Yo’q
```

- M yoki MULTILINE – yangi qator belgisi ("\ n") bilan ajratilgan bir nechta pastki satrlardan iborat qatorda qidirish. ^ belgisi har bir kichik satr boshiga biriktirilagan, $ belgisi esa yangi satr belgisidan oldingi holatiga mos keladi.
- S yoki DOTALL - odatiy holda “nuqta” metasimvollari satr o’tkazish (\n) dan tashqari har qanday belgiga mos keladi. Satrga o’tkazish belgisi “nuqta”

metasimvoli qo'shimcha modifikator ishtirokida mos keladi. ^ belgisi langarni butun satr boshiga, $ belgisi esa butun satr oxiriga to'g'ri keladi. Misol:

```
>>> p = re.compile (r"^.$")
>>> print ("Topildi" if  p.search ("\n" )  else  "Yo’q" )
Yo’q
>>> p = re.compile (r"^.$", re.M)
>>> print ("Topildi" if  p.search ("\n" )  else  "Yo’q" )
Yo’q
>>> p = re.compile (r"^.$", re.S)
>>> print ("Topildi" if  p.search ("\n" )  else  "Yo’q" )
Topildi
```

- x yoki VERBOSE –agar bayroq ko'rsatilgan bo'lsa, probellar va yangi qatorga o’tkazish belgisi e'tiborga olinmaydi. Izohlardan muntazam ifoda ichida ham foydalanish mumkin. Misol:

```
>>>  p = re.compile(r """  #  Satr boshiga bog’lanadi
[0 -9] + # Satr bitta (yoki ko’proq) raqmdan iborat
$ # Satr ohiriga bog’lanadi
""", re.X | re.S)
>>> print ("Topildi" if  p.search ("1234567890")  else "Yo’q")
Topildi
>>> print ("Topiladi" if p.search ("abcdl23")  else "Yo’q" )
Yo’q
```

8.2. Shablonga birinchi moslikni qidirish

Birinchi moslikni qidirish uchun quyidagu funksiya va metodlardan foydalaniladi:

- match() – satr boshlanishi bo’yicha moslikka tekshiradi. Metod formati:

match(<Satr> [, <Boshlang’ich pozitsiya> [, <Ohirgi pozitsiya>]])

Agar moslik topilsa Match obyekti qiymat qaytaradi, aksincha None qaytaradi. Masalan:

```
>>> import re
>>> p = re.compile (r "[0-9] + ")
>>> print ("Topildi" if p.match ("strl23" )  else  "Yo'q" )
Yo'q
>>> print ("Topildi" if p.match ("str123", 3)  else  "Yo'q" )
Topildi
>>> print ("Topildi" if p.match ("l23str")  else  "Yo'q" )
Topildi
```

match() metodi o'rniga .match() funksiyasidan foydalanish mumkin. Funksiya formati:

re .match (<Shablon>, <Satr> [, <Modifikator>])

<Shablon> parametrida muntazam ifodali satr yoki komplyatsiyalangan muntazam ifoda ko'rsatiladi. <Modifikator> parametrida compile() funksiyasida foydalanilgan bayroq ko'rsatilishi mumkin. Agar moslik aniqlansa, Match obyekti qaytariladi, aks holda None qiymati qaytariladi. Masalan:

```
>>> p  =  r" [0 - 9] +"
>>> print ("Topildi"  if  re .match (p,  "strl23" )  else  "Yo'q")
Yo'q
>>> print ("Topildi"  if  re.match (p, "123str")  else  "Yo'q")
Topildi
>>> p =  re . com pile (r " [0 -9] + ")
>>> print ("Topildi"  if  re . match (p, "123str")  else  "Yo'q")
Topildi
```

- search () – satrninig ihtiyoriq qismi bilan mosligini tekshirish. Metod formati:

search (<Satr> [, <Boshlang'ich pozitsiya> [, <Ohirgi pozitsiya>]])

Agar moslik aniqlansa Match obyekti, aks holda None qiymati qaytariladi. Masalan:

```
>>> p = re.compile (r"[0-9]+")
```

```
>>> print ("Topildi" if p.search ("str123")  else "Yo’q")
Topildi
>>>  print ("Topildi"  if  p. search ("l23str" )  else  "Yo’q" )
Topildi
>>>  print ("Topildi"  if  p. search("1 23str" ,  3)  else  "Yo’q" )
Yo’q
```

search () metodi o’rniga search () funksiyasidan foydalanish mumkin. Funksiya formati:

re.search(<Shablon> , <Satr> [, <Modifikator>])

<Shablon> parametrida muntazam ifodali satr yoki komplyatsiyalangan muntazam ifoda ko’rsatiladi. <Modifikator> parametrida compile() funksiyasida foydalanilgan bayroq ko’rsatilishi mumkin. Agar moslik aniqlansa, Match obyekti qaytariladi, aks holda None qiymati qaytariladi. Masalan:

```
>>> p = r" [0 -9] +"
>>> print ("Topildi"  if  re . search (p,  "str 1 23" )  else  "Yo’q" )
Topildi
>>>  p = re . compi le (r "[ 0- 9] +" )
>>>  print("Topildi"  if  re . search (p,  "str l2 3" )  else  "Yo’q" )
Topildi
```

- fullmatch () – berilgan satrni muntazam ifodaga to’liq mosligini tekshiradi. Bu metod Python 3.4 versiyadan boshlab qo’llab-quvvatlanadi. Umumiy ko’rinishi:

fullmatch (<Satr> [, <Boshlang’ich pozitsiya> [, <Ohirgi pozitsiya>]])

Moslik aniqlansa Match obyekti,aks holda None qiymati qaytariladi. Masalan:

```
>>> p =  re.compile (" [Pp]ython")
>>> print ("Topildi"  if p.fullmatch("Python")  else  "Yo’q" )
Topildi
>>>  print ("Topildi"  if  p. fullmatch("py") else  "Yo’q" )
Yo’q
>>>  print ("Topildi"  if  p.fullmatch ("Python Ware" )  else  "Yo’q")
```

```
Yo’q
>>>  print ("Topildi"  if p.fullmatch ("PythonWare", 0, 6) else  "Yo’q")
Topildi
```

fullmatch () metodi o’rniga fullmatch () funksiyasidan ham foydalanish mumkin. Funksiya formati:

re.fullmatch (<Shablon> , <Satr> [, <Modifikator>])

<Shablon> parametrida muntazam ifoda satri yoki muntazam ifoda konpilyatsiyasi ko’rsatiladi. <Modifikator> parametrida compile() funksiyasida foydalanilgan bayroqni ko’rsatish mumkin. Agar satr to’liq shablon bilan mos kelsa, match obyekti, aks holda None qiymati qaytariladi. Misollar:

```
>>> p =  "( Pp]ython"
>>> print ("Topildi"  if  re.fullmatch (p,  "Python")  else  "Yo’q" )
Topildi
>>>  print ("Topildi"  if  re.fullmatch (p,  "py")  else  "Yo’q")
Yo’q
```

Oldinroq foydalanuvchi tomonidan kiritilgan ihtiyoriy miqdordagi sonlar yig’indisini hisoblash dasturini ko’rgan edik (4.16-misolga qarang). Shuningdek manfiy sonlar kiritilishini ham ko’zda tutilgan usulda ko’rib chiqamiz (8.1-misol).

8.1-misol. Noaniq miqdordagi sonlar yig’indisi

```
#  -*- coding :  ut f-8  -* -
import re
print("Natija olish uchun 'stop '  so’zini kiriting")
summa = 0
p=re.compile(r"^[-]?[0-9]+$",re.S)
while True:
  x =input("Sonni kiriting: ")
  if x =="stop":
    break  #  Sikldan chiqish
  if not p.search(x):
    print("Satr emas, son kiritish kerak!" )
```

```
        continue  #  Siklning keying iterasiyasiga o’tish
      x=int(x) # Satrni songa o’tazamiz
      summa=summa+x
print("Sonlar yig’indisi:",  summa)
input ()
```

Muhokama uchun savollar va topshiriqlar

1. Pythonda muntazam ifodalar.
2. Pythonda muntazam ifoda sintaksisi.
3. Pythonda shablonga birinchi moslikni qidirish.
4. Pythonda shablonga barcha moslikni qidirish.
5. Pythonda satrni almashtirish

9-BOB. RO’YXATLAR, KORTEJLAR, TO’PALAMLAR VA DIAPAZONLAR (6-soat)

O`quv modullari

Ro’yxatlar, kortejlar, to’plamlar, diapazonlar, element pozitsiayasi, murojaat qilish, qirqim olish, konkatenasiya, takrorlash, kortej elementlari, obyekt.

9.1. Ro’yxatlar

Ro’yxatlar, kortejlar, to’plamlar va diapazonlar – bu obyektlarning tartiblangan to’plamidir. To’plamning har bir elementi faqatgini ixtiyoriy turdagi obyektga havola saqlay olishi sabab o’zida cheklanmagan darajadagi imkoniyat taqdim qiladi.

To’plamdagi element pozitsiayasi indeks orqali aniqlanadi. Elementlarni tartiblash 0 dan boshlanadi.

Ro’yxatlar va kortejlar shunchaki elementlarning tartiblangan ketma-ketligidir. Barcha ketma-ketliklar singari ular elementga indeks bo’yicha murojaat qilish, qirqim olish, konkatenasiya (+ operatori), takrorlash (* operatori), tegishlilikka (in operatori) yoki tegishli emaslikka (not in operatori) tekshirish amallarini qo’llaydi.

✓ Ro’yxatlar o’zgaruvchan tiplar toifasiga kiradi. Biz elementni nagfaqat indeks bo’yicha chop qilishimiz, balki uni o’zartira oilishimiz ham mumkinligini anglatadi:

```
>>> arr = [1, 2, 3]  # Ro’yxat yaratamiz
>>> arr [0]          # indeks bo’yicha elementni amiqlaymiz
1
>>> arr [0] = 50  #  indeks bo’yicha elementni o’zgartiramiz
>>> arr
[50, 2, 3]
```

Kortejlar o’zgarmas tiplar toifasiga mansub. Kortej elementlarini indeks bo’yicha aniqlash (olish, chop etish) mukin, biroq o’zgartirish mumkin emas:

```
>>> t = (1, 2, 3)            #  Kortej yaratamiz
```

```
>>> t [0]                    # Indeks bo'yicha elementni olamiz
1
>>> t[0] = 50 # Indeks bo'yicha elementni o'zgartirish mumkin emas
Traceback (most recent call last ) :
File "<pyshell#7 >", line 1, in <module>
t[0] = 50 # Indeks bo'yicha elementni olamiz
TypeError: 'tuple ' obj ect does not support item assignment
```

To'plam o'zgaruvchan kabi bo'lishi ham mumkin, o'zgarmas kabi ham mumkin. Uning asosiy farqi o'zida unikal qiymatlarni saqlashidadir (bir hil qiymatlar avtomatik yo'qotiladi). Masalan:

```
>>> set([0, 1, 1, 2, 3, 3, 4])
{0, 1, 2, 3, 4}
```

✓ Diapazonlar o'zida boshlang'ich va ohirgi qiymatlari hamda qadam kattaligi berilgan sonlar to'plamini aks ettiradi. Ular ning oldingi obyektlar to'plamidan muhim ustunligi tezkor xotiradan kam joy egallaydi. Masalan:

```
>>> r = range (0, 101, 10)
>>> for i in r: print (i, end = " ")
0 10 20 30 40 50 60 70 80 90 100
```

Yuqoridagi turlarni batafsil ko'rib chiqamiz.

9.2. Ro'yxat yaratish

Ro'yxat yaratish quyidagi usullarda amalga oshirilishi mumkin:

➢ list ([<Ketma-ketlik>]) funksiyasi yordamida. Funksiya ixtiyoriy ketma-ketlikni ro'yxatga o'tkazadi. Agar paraetr ko'rsatilmasa bo'sh ro'yxat yaratiladi. Masalan:

```
>>> list () # Bo'sh ro'yxat yaratamiz
[ ]
>>> list ("String" ) # Satrni ro'yxtga o'tkazamiz
[ 's' , 't', 'r', 'i', 'n ', 'g' ]
>>> list ( (1, 2, 3, 4, 5)) # Kortejni ro'yxatga o'giramiz
[1, 2, 3, 4, 5]
```

- barcha royxat elementlarini kvadrat qavs ichida keltirish orqali:

```
>>> arr = [1, "str", 3, "4"]
>>> arr
[1, 'str', 3, '4 ']
```

- append () metodi yordamida element bo'yicha to'ldirish:

```
>>> arr =  []  #  Bo'sh ro'yxat yaratamiz
>>> arr.append (l)  # Element yaratamiz  (0- indeks)
>>> arr.append (" str") # 2-element yaratamiz  (1- indeks)
>>>  arr
[1, 'str ']
```

Ro'yxat yaratishda o'zgaruvchida obyekt emas, obyektga havola saqlanadi. Buni guruhli o'zlashtirishda hisobga olish kerak. Gurugli o'zlashtirishdan sonlsr uchun, qatoralar uchun foydalanish mumkin. Ro'yxat uchun esa mumkin emas. Misol ko'rib chiqamiz:

```
>>>  x  =  y  =  [1,  2]                    #  ikkita obyekt yaratildi
>>>  x,  y
( [ 1,  2] ,  [ 1,  2] )
```

Biz bu misolda ikki elementli ro'yxat yaratdik va uni x va y o'zgaruvchilariga o'zlashtirdik. Endi y o'zgaruvchidagi ikkinchi element qiymatini o'zgartiramiz:

```
>>>  y[l]  =  100  #  ikkinchi elementni o'zgartiramiz
>>>  x,  y #  birdnaiga ikkita element qiymati o'zgarmoqda
([1,  100] ,  [1,  100])
```

Ko'rinib turibdiki har ikkala o'zgaruvchi bitta obyektni ko'rsatyapdi, alohida obyektni ko'rsatishi uchun boshqacharoq o'zlashtiramiz:

```
>>> x,  y= [1,  2],  [1,  2]
>>> y[l] = 100 #  Ikkinchi elenetni o'zgartirmiz
>>> x,  y
([1, 2], [1,  100] )
```

Ikkita o’zgaruvhci bitta obyektga havolanai ko’rsatayotganini aniqlash uchun is operatoridan foydalaniladi. Agar o’zgaruvchilar bitta obyektni ko’rsatsa, is operatori True qiymat qaytaradi:

```
>>> x = y = [1, 2] # Bu tavsiya etilmaydi
>>> x is y          # o’zgaruvchilar havolasini tekshirish
True                # demak bitta obyekni ko’rsatayapdi
>>> x, y = [1, 2], [1, 2]# to’g’ri
>>> x is y # bular farqli obyekt
False
```

9.3. Ro’yxatlar ustida amallar

Ro’yxat elementiga murojaat element indeksini kvadrat qavs orasiada ko’rsatish orqali amalga oshiriladi. Elementlar tartibi noldan boshalanadi. Masalan:

```
>>> arr = [1, "str", 3.2, "4 "]
>>> arr[0] , arr[1] , arr[2] , arr[3]
(1, 'str', 3.2, '4')
```

Pozitsion o’zlashtirishda elementlar qiymatini ihtiyoriy o’zgaruvchilarga o’zlashtirish mumkin. Bunda chap tomondagi o’zgaruvchilar soni o'ng tomondagi elementlar soniga o’zaro mos bo’lishi kerak, aks holda xatolik yuz beradi:

```
>>> x, y, z = [1, 2, 3] # Pozitsion o’zlashtirish
>>> x, y, z
(1, 2, 3)
>>> x, y = [1, 2, 3] # Elementlar soni mos bo’lishi kerak
Traceback (most recent call last):
Fi le "<pyshell#86>" , line 1, in <module>
x, y = [1, 2, 3] # Elementlar soni mos bo’lishi kerak
ValueError: too many values to unpack (expected 2)
```

Python 3 da pozitsion o’zlashtirishda chap tomondagi bitta o’zgaruvchi oldidan yulduzcha yozish va bu orqali ortiqcha elementlarni ro’yxat sifatida unga o’zlashtirish mumkin. Agar ortiqcha element bo’lmasa, bu o’zgaruvchi o’zida bosh ro’yxatni saqlaydi:

```
>>> x, y, *z = [1, 2, 3] ; x, y,
(1, 2, [3] )
>>> x, y, *z = [1, 2, 3, 4, 5] ; x, y, z
(1, 2, [3, 4, 5])
>>> x, y, *z = [1, 2] ;   x, y, z
( 1, 2, [ ] )
>>> *x, y, z = [1, 2] ;       x, y, z
( [ ], 1, 2 )
>>> x, *y, z = [1, 2, 3, 4, 5] ; x, y, z
(1, [2, 3, 4] , 5)
>>> *z = [1, 2, 3, 4, 5] ; z
[1, 2, 3, 4, 5]
```

Ro'yxat elementlarni tartiblash nol dan boshlangani uchun ohirgi element tartibi elementlar sonidan bitta kam bo'ladi. Elemntlar sonini aniqlash uchun len() funksiyasidan foydalaniladi:

```
>>> arr = [1, 2, 3, 4, 5]
>>> len (arr)  #  Elementlar sonini aniqlaymiz
5
>>>  arr [len (arr ) -1] #  Ohirgi element tartibini aniqlaymiz
5
```

Agar ko'rsatilgan indeksdagi element mavjud bo'lmasa, IndexError xatoligi yuz beradi:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr [5]  # Mavjud bo'lmagan elementga murojaat
Traceback (most recent call last):
File "<pyshell#99>", line 1, in <module>
arr [5]  # Mavjud bo'lmagan elementga murojaat
IndexError: list index out of range
```

Indeks sifatida manfiy qiymatdan foydalanih mumkin. Bu holda tartiblanish ro'yxat oxiridan boshlanadi. Agar indeksni musbatlashtirmoqchi bo'lsak, elementlar sonidan manfiy indeksni ayrladi. Masalan:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr [- 1] , arr [len (arr) -1] # Oxirgi elementga murojaat
(5, 5)
```

Ro'yxatlar o'zgaruvchi turlaga kirgani uchun indekslar yordamida ularning elementlarini o'zgartirish mumkin:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr [0] = 600 # indeks bo'yicha elementni o'zgartirish
>>> arr
[600, 2, 3, 4, 5]
```

Bundan tashqari ro'yxatlarda ko'rsatilgan qismdagi ro'yxat elementlarini nusxalab olish imkoniayati mavjud. Amal formati:

[<Boshlanish> :<Ohiri> :<Qadam>]

Barcha barametrlar majburiy emas. Agar <Boshlanish> ko'rsatilmasa qiymat siaftida 0 olinadi, <Ohiri> ko'rsatilmasa ohirigacha olinadi.

Agar :<Qadam> parametri kiritilmasa 1 qiymatdan foydlaniladi. Parametrlar qiymati sifatida manfiy son olinishi mumkin. Misol ko'rib chiqamiz. Dastlab ro'yxat yuzaki nusxasini olamiz:

```
>>> arr = [1, 2, 3, 4, 5 ]
>>> m = arr[ : ]; m # Yuzaki nusxa yaratamiz va qiymatni chop etamiz
[1, 2, 3, 4, 5 ]
>>> m is arr # is operatori turli obyekt ekanligini ko'rsatyapdi
False
```

Endi belgilarni teskari tartibda chop etamiz:

```
>>> arr = [1, 2, 3, 4, 5 ]
>>> arr [ : : -1 ] # - 1 qadam
[5, 4, 3, 2, 1]
```

Birinchi va ohirgi elementsiz chop etamiz:

```
>>> arr [1:]              # birinchi elementsiz
[2, 3, 4, 5]
>>> arr [ :- 1]           # Ohirgi elementsiz
[1, 2, 3, 4]
```

Ro'yxatning birinchi ikki elementini olamiz:

```
>>> arr [0:2]             # 2 indeksli belgi chop etilmaydi
[1, 2]
```

Endi ohirgi elementni chop etamiz

```
>>> arr[- 1:]       # ro'yxatning ohirgi elementi
[5]
```

Va nihoyat ikkinchidan to'rtinchigacha elemetlarni chop etamiz:

```
>>> arr[1:4]  # 1, 2 va 3 indeksli elementlar olinadi
[2, 3, 4]
```

Qirqim bo'yicha ro'yxat bo'lagini o'zgartirish mumkin. Agar qirqimda bo'sh ro'yxat o'zlashtirilsa, ushbu bo'lak asosiy ro'yxatdan o'chiriladi:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr[ l:3 ] = (6, 7) # 1 va 2 indeksli elementni o'zgartiramiz
>>> arr
[1, 6, 7, 4, 5 ]
>>> arr[l:3 ] = [ ] # 1 va 2 indeksli element o'chiriladi
>>> arr
(1, 4, 5]
```

Ikkita ro'yxatni bittaga birlashtirishda + operatoridan foydalaniladi:

```
>>> arrl = [1, 2, 3, 4, 5]
>>> arr2 = [ 6, 7, 8, 9]
>>> arr3 = arrl + arr2
>>> arr3
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

+ operatori o’rniga += operatoridan foydalanish mumkin. Bu holda elementlar joriy ro’yxatga qo’shiladi:

```
>>> arr = [1 , 2, 3, 4, 5]
>>> arr += [6 , 7, 8, 9]
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

Ko’rib chiqilgan amallardan tashqari ro’yxatlar uchun takrorlash(* operatori yordamida), tegishlilikka tekshirish (in operatori yordamida) amallarini qo’llash mumkin:

```
>>> [ 1, 2, 3] * 3 # takrorlash amali
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> 2 in [1, 2, 3, 4, 5], 6 in [1, 2, 3, 4, 5] #Tegishlilikka tekshirish
(True, False)
```

9.4. Ko’p o’lchamli ro’yxatlar

Ro’yxat elementi ixtiyoriy tipli obyekt bo’lishi mumkin. Masalan, ro’yxat elementi son, satr, ro’yxat, kortej, lug’at va boshqalar bo’lishi mumkin. Ko’p o’lchamli ro’yxatlarni quyidagicha yaratish mumkin:

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

Qavs ichida bir nechta satrlarni keltirish mumkin.

Demak, oldingi misolni quyidagicha yozish mumkin:

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

Ko’p o’lchamli ro’yxat elementini ko’rsatish uchun ikkita indeks ko’rsatiladi:

```
>>> arr [l] [1]
5
```

Ikki o’lchamli ro’yxat elementi sifatida yana ro’yatmi keltirish mumkin va bu ichma-ich joylshtirilib boriladi. Ko’p o’lchamli ro’yxatda elementni ko’rsatish uchun esa nechta o’lchamli ro’yxat bo’lsa ushancha indekslar yoziladi. Masalan:

```
>>>  arr = [[1, ("a",  "b"], 3],  [4,  5,  6] ,  [7,  8,  9] ]
>>>  arr  [0] [1] [0]
'a'
```

9.5. Ro’yxat elemantlarini saralash

Ro’yxatning barcha elementlari bo’ylab yurib chiqish uchun for operatoridan foydalanish mumkin:

```
>>>  arr [1, 2, 3, 4, 5]
>>>  for i in arr: print (i, end="  ")
1 2 3 4 5
```

Yodda tutish kerakki, ***i*** o’zgaruvchisi sikl ichida o’zgarishi mumkin, biroq u o’zgarnmas tipni ko’rsatsa (masalan son yoki satrni) elementlarni o’zgartira olmaydi:

```
>>> arr = [1, 2, 3, 4] # O’zgarmat tipli elementlar (son)
>>> for  i  in arr:  i +=  10
>>>  arr       #  Ro’yxat o’zgarmaydi
[1, 2,  3, 4]
>>>  arr [ [1, 2],  [3,  4]]  # O’zgaruvchan tipli elementlar (ro’yxat)
>>>  for  i  in  arr : i [0] + =  10
>>>  arr  #  Ro’yxat o’zgaradi
[[11, 2],  [13,  4] ]
```

Har bir elementni ko’rsatish masalan, indeks bo’yicha generasiyalash uchun range () funksiyasidan foydalaniladi. Funksiya obyekt-diapazon qaytaradi, iterasiyanni qo’llab-quvvatalaydi. For sikli ichida joriy iondeksni olish mumkin.

range () funksiyasi quyidagi formatga ega:

range ([<Boshlanish>,] <Ohiri> [, <Qadam>])

Birinchi parametrda boshlang’ich qiymat beriladi. Agar <Boshlanish> parametri ko’rsatilmasa, qiymat sifatida 0 olinadi.

Shuni esda saqlash kerakki, bunda diapazon indeks bo’ylab “yurib chiqadi”. Agar <Qadam> parametri ko’rasatilmasa, qiymat sifatida 1 olinadi. Ro’yxat har bir elementini 2 ga ko’paytirish misoli:

```
arr  =  [ 1 ,  2,  3,  4]
```

```
for i  in  range (len (arr) ):
arr[i]  *=  2
print (arr)
```

Natija: [2, 4, 6, 8]

Shuningdek joriy indeks va elementni ham qaytarish imkoniga ega bo'lgan enumerate (<obyekt> [, start=0]) funksiyasidan ham foydalanish mumkin. Ro'yxat har bir elementini 2 ga ko'paytiramiz:

```
arr  =  [1,  2,  3,  4]
for  i,  elem  in  enumerate(arr):
arr [i ]  *=  2
print (arr)
```

Natija: [2, 4, 6, 8]

Bundan tashqari elementlarni while siklidan foydalanib ham olish mumkin. Biroq bu holda for ga qaraganada sekinroq ishlaydi. Misol sifatida while yordamida ro'yxat elementlarini 2 ga ko'paytirishni ko'ramiz:

```
arr = [1,  2,  3,  4]
i= 0
c= len(arr )
while i < c:
arr [i]  *=  2
i  +=  1
print (arr )
```

Natija: [2, 4, 6, 8]

9.6. Ro'yxat generatorlari

Biz elementlani quyidagicha o'zgartirgan edik:

```
arr = [1, 2, 3, 4]
for i in range (len(arr)):
arr [i] *= 2
print (arr) #
```

Natija: [2, 4, 6, 8]

Ro’yxat generatori yordamida buni ixchamroq ifodlash mumkin:

```
arr = [1, 2, 3, 4]
arr = [i*2 for i in arr ]
print (arr) #
Natija: [2, 4, 6, 8]
```

Ro’yxat generatori murakkabroq strukturada ham bo’lishlari mumkin. Masalan, bir nechta for ichma-ich sikllar va unlarda joylashgan if tarmoqlanish operatoridan qatnashishgan holat. Misol uchun juft sonlarni 10 ga ko’paytirib chop etishni ko’raylik:

```
arr = [1, 2, 3, 4]
arr = [ i * 10 for i in arr if i % 2 == 0]
print (arr)
# Natija: [20, 40]
```

Ushbu kod bajarilish ketma-ketligi quyidagi kod bajarilish ketma-ketligiga teng kuchli:

```
arr = [ ]
for i in [1, 2, 3, 4] :
if i%2 == 0:          # Agar son juft bo’lsa
arr.append (i * 10)   # Element qo’shamiz
print(arr)
# Natija: [20, 40]
```

Misolni sal murakkablashtiramiz. Ro’yxatdagi juft elementlarni olamiz va ularni 10 ga ko’paytiramiz:

```
arr = [[1, 2], [3, 4], [5, 6] ]
arr = [ j * 10 for i in arr for j in i if j % 2 == 0
print (arr)                    # Natija: [20, 40, 60 ]
```

Ushbu kod bajarilish ketma-ketligi quyidagi kod bajarilish ketma-ketligiga teng kuchli:

```
arr = []
for  i  in  [[ 1,  2] ,  [3,  4] ,  [5,  6] ]:
for  j  in  i:
if  j  %  2  =  0:  #  Agar son chuft bo'lsa
arr.append (j * 10)  # Elementni qo'shamiz
print (arr)                    # Natija: [20,  40,  60 ]
```

Misol sifatida ro'yxatdagi juft sonlar yig'indisini hisoblaymiz:

```
>>> arr = [1, 4, 12, 45, 10]
>>> sum((i for i in arr if  i%2 == 0))
26
```

9.7. map(), zip(), filter() va reduce() funksiyalari

- **map ( )** funksiyasi ketma-ketlikning har bir elementiga qo'llash imkonini beradi. Funksiya quyidagi formatda beriladi:

map (<Funksiya>, <1-ketma-ketlik> [, ... ,<N-ketma-ketlik >])

map() funksiyasi iterasiyani qo'llab-quvvatlaydigan obyekt qaytaradi. Agar Python 3 versiyada ro'yxat olish uchun natijani list() funksiyasida berish kerak.

<Funksiya> parametri sifatida funksiyaga funksiya nomi (funksiya nomi aylanma qavslarsiz) chaqiriluvchi qiymatlari tartibiga mos holda beriladi. Funksiya ichida yangi qiymat qaytarish zarur. Misol sifatida ro'yxatning har bir elementiga 10 ni qo'shishni ko'rib chiqamiz (9.3-misol).

9.3-misol. map() funksiyasi

```
def  func (elem):
" " " Ro'yxatning har bir elementini oshirish" " "
return elem + 10              # yangi qiymat qaytaradi
arr  =  [1,  2,  3,  4,  5]
print (list (map (func, arr)))
#  Natija:  [11,  12 ,  13,  14 ,  15]
```

map () funksiyasi bir nechta ketma-ketlikni uzatadi. Bu holda teskari qaytar funksiya birdaniga bir siljishda bir nechta qiymatni uzatadi. Uchta ro'yxat elementlari yig'indisini aniqlaymiz (8. 4-misol).

8.4-misol. Uchta ro'yxat elementlari yig'indisini topish

```
def  func(el, e2,  e3 ) :
""" Uchta turli ro'yxat elementlari yig'indisini xisoblash"""
return  el  +  e2  +  e3  #  yangi qiymat qaytaradi
arrl = [1,  2,  3,  4,  5]
arr2 =  [1 0,  20 ,  30,  40 ,  50]
arr3  =  [100 ,  200 ,  300,  400,  500 ]
print (list (map (func ,  arrl,  arr2 ,  arr3 )  )  )
#  Natija :  [111,  2 2 2,  333,  444,  555 ]
```

Agar ketma-ketlikdagi elementlar soni turli hil bo'lsa, minimal elementlar sonicha ketma-ketlik qaytariladi:

```
def  func(e1,  e2,  e3 ):
  """Uchta turli ro'yxat elementlari yig'indisi """
  return e1 +  e2  + e3
arr1= [1, 2, 3, 4, 5]
arr2= [10,  20]
arr3  =  [100 ,  200 ,  300,  400,  500]
print (list (map (func,  arr1,  arr2,  arr3)))
#  Natija:  (111,  222 )
```

- **zip ()** funksiyasi ketma-ketliklardagi elementladan bittadan olib yangi ketma-ketlik hosil qiladi va natija sifatiada iterasiyani qo'llab-quvvatlovchi obyekt qaytaradi. Agar natijani ro'yxatga aylantirish zarur bo'lsa list() funksiyasi qo'llaniladi:

Funksiya formati:

zip (<Ketma-ketlik1> [, ... , < Ketma-ketlikN>])

Masalan:

```
>>>  zip([l,  2,  3],  [4,  5 ,  6],  [7,  8,  9])
< zip object at Ox00 FCA C88>
```

```
>>>  list (zip([l,  2,  3] ,  [4,  5,  6] ,  [7,  8,  9] ))
[(1, 4, 7),  (2,  5,  8) ,  (3,  6,  9)]
```

Agar ketma-ketlikdagi elementlar soni turlicha bo'lsa, natijaviy kortejlar eng kam sondagi elementlar sonicha bo'ladi:

```
>>>  list (zip( [l,  2,  3] ,  [4 ,  6] ,  [7 ,  8,  9,  10] ))
[(1, 4, 7), (2,  6,  8)]
```

Misol sifatida uchta ro'yxatdagi elementlarni map() funksiyasi o'rniga zip() funksiyasidan foydalanib yig'ish misolini ko'rib chiqamiz. (8.5-misol).

8.5-misol. zip() funksiyasidan foydalanib uchra ro'yxat elementlarini yig'ish

```
arr1 = [1, 2,  3,  4,  5]
arr2 = (10, 20, 30, 40, 50)
arr3 = [100,  200,  300,  400,  500]
arr = [ x + y + z for (x, y, z) in zip (arr1, arr2, arr3)]
print (arr)
#  Natija: [111, 222, 333, 444, 555]
```

9.8. Ro'yaxatga elementlar qo'shish va o'chirish

Ro'yxat elementlarini qo'shish va o'chirish uchun quyidagi metodlardan foydalaniladi:

- **append (<Obyekt>)** – ro'yxat oxiriga bitta obyekt qo'shadi. Metod joriy ro'yxatni o'zgartiradi va hech nima qaytarmaydi. Masalan:

```
>>> arr = [1, 2, 3]
>>> arr.append (4) ;                arr  #  Sonni qo'shish
[1, 2, 3, 4]
>>>  arr.append ([5, 6]) ;          arr  #  Ro'yxatni qo'shish
[1, 2, 3, 4,  [5,  6]]
>>>arr. append ((7, 8));  arr # Kortejni qo'shish
[1, 2, 3,  4, [5,  6], (7,  8)]
```

- **extend (<Ketma-ketlik>)** – ro'yxat oxiriga ketma-ketlik elementlarini qo'shadi. Metod joriy ro'yxatni o'zgartiradi va hech nima qaytarmaydi. Masalan:

```
>>> arr =  [1, 2, 3]
>>> arr.extend([4, 5, 6])  #  Ro'yxat qo'shamiz
>>> arr.extend((7, 8, 9))  #  Kortej qo'shamiz
>>> arr.extend("abc")  #  Satdan harf qo'shamiz
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9, 'a', 'b', 'c']
```

Bir nechta elementlarni qo'shish uchun konkatenasiya amali yoki += operatoridan foydalaniladi:

```
> >> arr =  [1,  2,  3]
>>> arr + [4,  5,  6]                #  Yangi ro'yxat qaytaradi
[1, 2, 3, 4, 5, 6]
>>>  arr +=  [4 ,  5,  6]    #  Joriy ro'yxat o'zgaradi
>>>  arr
[1,  2,  3,  4,  5,  6]
```

Bundan tashqari, qirqim olish amalidan ham foydalanish mumkin:

```
>>> arr  =  [1,  2,  3]
>>> arr [ len(arr):]  =  [4,  5,  6]  # Joriy ro'yxat o'zgaradi
>>> arr
[1,  2,  3,  4,  5,  6]
```

- **insert (<Indeks>, <Obyekt>)** – ko'rsatilgan pozitsiyaga bitta obyektni qo'shish. Qolgan elementlar suriladi. Metod joriy ro'yxatni o'zgartiradi va hech nima qayatarmaydi. Misollar:

```
>>>  arr =  [1,  2,  3]
>>>  arr.insert (0, 0) ;  arr  #  Ro'yxat boshiga 0 qiymatini qo'shadi
[0, 1, 2, 3]
>>> arr.insert (-1, 20) ;  arr # Manfiy sonni ham ko'rsatish mumkin
(0, 1, 2, 20, 3]
>>> arr.insert(2, 100);      arr      #  2 pozitsiyaga 100  ni qo'yamiz
[0, 1, 100, 2, 20, 3]
```

```
>>> arr.insert (10, [4, 5]) ;  arr  #  Ro'yxat qo'shamiz
[0, 1, 100, 2, 20, 3, [4, 5]]
```

insert() metodi faqat bitta obyekt yaratishga imkon beradi. Agar bir nechta obyekt qo'shilsa, qirqim olib o'zlashtirish amalidan foydalanish mumkin. Bir nechta elementlarni ro'yxat boshiga qo'shamiz:

```
>>> arr = [1, 2, 3]
>>> arr [:0] = [-2, -1, 0]
>>> arr
[-2, -1, 0, 1, 2, 3]
```

- **pop ([<Indeks>])** – ko'rsatilgan indeksda joylashagan elementni o'chiradi va uni qaytaradi. Agar indeks ko'rsatilamasa, ro'yxatning ohirgi elementi ochiriladi va qaytariladi.

Agar ko'rsatilgan indeksdagi element yo'q bo'lsa yoki ro'yxat bo'sh bo'lsa, IndexError xatoligi kelib chiqadi. Misollar:

```
>>>  arr = [1, 2, 3, 4, 5]
>>>  arr.pop()  #  Ro'yxat oxirgi elementi o'chiriladi
5
>>>  arr  #  Ro'yxat o'zgaradi
[1,  2,  3,  4]
>>>  arr.pop(0)  #  Ro'yxat birinchi elementi o'chiriladi
1
>>>  arr  #  Ro'yxat o'zgaradi
[2,  3,  4]
```

Shuningdek **del** operatori ham ro'yxat elementini o'chirishga imkon beradi:

```
>>>  arr  =  [1,  2,  3,  4,  5 ]
>>>  del  arr[4] ;  arr  # Ro'yxatning oxirgi elemntini o'chiramiz
[1,  2,  3,  4]
>>> del arr[:2]; arr #Ro'yxatning birinchi va ikkinchi elementini o'chiramiz
[3, 4]
```

remove (<Qiymat>) - ko’rsatilgan qiymatga teng birinchi elementni o’chirish. Agar element topilmasa, valueError xatoligi yuz beradi.

Metod joriy ro’yxatni o’zgartiradi va hech nima qaytarmaydi. Misollar:

```
>>> arr = [1, 2, 3, 1, 1]
>>> arr.remove (l) # faqat birinchi element o’chiriladi
>>> arr
[2, 3, 1, 1]
>>> arr . remove (5) # bunday element yo’q
Traceback (most recent call last ) :
File "<pyshell# 3>" , line 1, in <module>
arr.remove (5) # Bunday element yo’q
ValueError: list.remove (x): x not in list
```

- **clear ()** – ro’yxatning barcha elementlarini o’chirib uni tozalash. Hech qanday natija qaytarilmaydi. Ushbu metod Python 3.3 versiyadan boshlab ishlatilgan. Masalan:

```
>>> arr = [1, 2, 3, 1, 1]
>>> arr.clear ()
>>> arr
[]
```

9.9. Ro’yxat elementlarini qidirish va ro’yxatga kirivchi qiymatlari haqida ma’lumot olish

Oldingi mavzularda ro’yxatga elementlarni tegishliligini tekshirish uchun in operatoridan foydalanish mumkinligini aytib o’tgan edik. Agar element ro’yxataga tegishli bo’lsa True natija qaytaradi, aks holda False. Shu operatorning aynan teskarisi ham mavjud. Yani elementni tegishli emaslikka tekshiruvchi **not in.** Agar tegishli bo’lmasa True, aks holda False qaytaradi. Misollar:

```
>>> 2 in [1, 2, 3, 4, 5], 6 in (1, 2, 3, 4, 5] #tegishlilikka tekshirish
(True, False)
>>> 2 not in [1, 2, 3, 4, 5], 6 not in [1, 2, 3, 4, 5] #tegishli emaslikka tekshirish
```

(False, True)

Bu ikkala operator ham elementning ro'yxatda joylashgan o'rni haqida hech qanday ma'lumot bermaydi. Elementning ro'yxatda joylashgan o'rnini aniqlash uchun **index()** metodi ishlatiladi.

Metod formati:

index (<Qiymat> [, <Boshlanish> [, <Tugash>]))

index() ko'rsatilgan qiymataga teng ro'yxat elementi indeksini qaytaradi. Agar qiymat ro'yxatda topilmasa, ValueError xatoligi kelib chiqadi. Agar ikkinchi va uchinchio parametrlar ko'rsatilmasa, qidirish ro'yxat boshidan ohirigacha amalga oshiriladi. Misol:

```
>>> arr = [1, 2 , 1, 2, 1]
>>> arr.index (1) , arr.index (2)
(0, 1)
>>> arr.index (1, 1) , arr.index (1, 3, 5)
(2, 4)
>>> arr.index (3)
Traceback (most recent call last):
File "<pyshell#16>" , line 1, in <module>
arr.index (3)
ValueError: 3 is not in list
```

Ko'rsatilgan qiymatga teng elementlar sonini aniqlash uchun count (<Qiymat>) ishlatiladi. Agar element ro'yxatda chiqmasa, 0 qiymatini qaytaradi. Misol:

```
>>> arr = [1, 2, 1, 2, 1]
>>> arr . count (1), arr.count (2)
(3, 2 )
>>> arr.count (3) # Element ro'yxatda topilmadi
0
```

Ro'yxatdagi eng katta va eng kichik qiymatlarni aniqlash uchun mos ravishda max() va min() funksiyalaridan foydalaniladi. Masalan:

```
>>> arr = [1, 2 , 3, 4, 5]
```

```
>>> max (arr),  min (arr)
(5, 1)
```

9.10. Ro’yxatni teskarilash va aralshtirish

reverse () metodidan ro’yxat elementlarini teskari tartibda joylashtirish uchun foydalaniladi. Metod joriy ro’yxatni o’zgartiradi va hech nima qaytarmaydi. Masalan:

```
>>>  arr  = [1,  2,  3,  4,  5]
>>>  arr.reverse () #  Joriy ro’yxatni o’zgartiradi
>>>  arr
[5, 4, 3, 2, 1]
```

shuffle (<Ro’yxat> [, <0.0 dan 1.0 gacha son>]) random modulidan olingan elementlarni tasodifiy «joylashtirish» funksiyasi. Funksiya elementlar joyini o’zgartiradi holos va u qiymat qaytaramaydi.

Agar ikkinchi parametr ko’rsatilmasa, random () funksiyasi paarmetridan foydalaniladi.Masalan:

```
>>>  import  random  #  random modulini bog’laymiz
>>>  arr  =  [1,  2,  3,  4,  5, 6, 7,  8,  9,  10]
>>>  random. shuffle (arr ) #  Elelementlar tasodifiy joylashtiriladi
>>> arr
 [2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
```

9.11. Tasodifiy elementni tanlash

Ro’yxatdan tasodifiy elementni tanlab olish uchun random modulidan quyidagi funksiyalar ishlatiladi:

- choice (<Ketma-ketlik>) – ixtiyoriy ketma-ketlikdan(satr, ro’yxat, kortej) tasodifiy elemntni olish uchun:

```
>>>  import  random  #  random modulini ulaymiz
>>>  random.choice(["s", "t", "r"]) #  Ro’yxat
's'
```

- sample (<Ketma-ketlik>, <Elementlar soni>) – ro’yxatdan ko’rsatilgan miqdordagi elementlarni tasodifiy tanlab olish.

Ketma-ketlik sifatida iterasiyani qo'llab-quvvatlovchi ixtiyoriy obyektni ko'rsatish mumkin. Masalan:

```
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random. sample(arr, 2)
(7, 10)
>>> arr # Ro'yxatni o'zi o'zgarmaydi
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

9.12. Ro'yxatni saralash

Ro'yxatni saralash uchun sort() metodidan foydalaniladi. U quyidagi formatga ega:

```
sort ([key=None] [, reverse=False])
```

Barcha parametrlar majburiy emas. Metod joriy ro'yxatni o'zgartiradi va hech nima qaytarmaydi. Odatdagi holda saralanish o'sish tartibida bo'ladi:

```
>>> arr = (2, 7, 10 , 4, 6, 8, 9, 3, 1, 5]
>>> arr.sort () # Joriy ro'yxat o'zgaradi
>>> arr
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Agar kamayish tartibioda saralash zarur bo'lsa, reverse parametri qiymati sifatiada True ko'rsatilishi kerak:

```
>>> arr = [2 , 7, 10, 4, 6, 8, 9, 3, 1, 5]
>>> arr.sort(reverse = True ) # Kamayish tartibida saralash
>>> arr
(10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Yodda saqlash kerakki, standart saralash belgilar registriga bog'liq (8.8-misol).

8.8-misol. Standart saralash

```
arr = ["bir", "Bir1", "bir2"]
arr.sort ()
for i in arr:
print (i, end= " ")
# Natija: Bir1 bir bir2
```

Natijada esa noto’g’ri saralanish chop etiladi.

Agar belgilar registrini hisobga olmaslik kerak bo’lsa, key parametrida funksiyaga havola ko’rsatiladi (8.9-misol).

8.9-misol. Foydalanuvchi saralashi

```
arr = ["bir", "Bir1", "bir2"]
arr.sort (key=str.lower)   #  lower () metodini ko’rsatamiz
for i in arr:
print (i, end=" ")

# Natija:  bir Bir1 bir2
```

sort ( ) metodi ro’yxatni saralaydi va hech qanday qiymat qaytarmaydi. Bazi holatlarda saralanagan ro’yxatni olish va joriy ro’yxatni esa o’zgarishsiz qoldirish zarur bo’ladi. Bu holler uchun sorted() funksiyasidan foydalaniladi. Funksiya quyidagi formatga ega:

sorted (<Ketma-ketlik>[, key=None] [, reverse=False])

Birinchi parametrda saralanadigan ro’yxat ko’rsatiladi. Qolgan parametrlar sort() metodi parametrlari bilan bir hil. sorted() funksiyasidan foydalanishga doir misolni ko’rib chiqamiz (8.10-misol).

8.10-misol. sorted() funksiyasidan foydalanish

```
>>> arr = [2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
>>> sorted (arr ) # yangi ro’yxat qaytaradi!
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> sorted (arr, reverse=True ) # Yangi ro’yxat qaytaradi!
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> arr = ["bir 1" , "Bir" , "Bir2"]
>>> sorted (arr, key=str.lower)
['Bir', 'bir 1', 'Bir2']
```

9.13. Ro’yxatni sonlar bilan to’ldirish

Diapazon tarkibida bo’lgan sonlardan ro’yxat yaratish uchun range() funksiyasidan foydalanish mumkin. Bu funksiya list() funksiyasini ishlatib ro’yxatga o’tkazish mumkin bo’lgan diapazon qaytaradi.

range() funksiyasi quyidagi formatga ega:

range ([<Boshlang’ich qiymat>,] <Ohirgi qiymat> [, <Qadam>])

birinchoi parameter – <Boshlang’ich qiymat> ko’rsatilmasa 0 qiymat olinadi.

Ikkinchi parameter ohirgi qiymat bo’lib, bu qiymat qaytariluvchi diapazonni aniqlaydi. Agar <Qadam> parametri berilmasa, 1 qiymati foydalaniladi. Misol sifatida ro’yxatni 0 dan 10 gacha sonlar bilan to’ldirishni ko’ramiz:

```
>>> list (range (11))
[0, 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 1 0]
```

1 dan 15 gacha sonlardan iborat ro’yxat tuzamiz:

```
>>> list(range (l, 16) )
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 , 15]
```

Endi sonlar tartibini qarama-qarshisiga o’zgartiramiz:

```
>>> list (range(15, 0, -1 ) )
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Agar olinadigan sonlar (yoki boshqa ro’yxat elementlari) tasodifiy bo’lishi zarur bo’lsa, random modulidagi sample (<Ketma-ketlik>, <Elementlar soni>) funksiyasidan foydalaniladi. Masalan:

```
>>> import random
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.sample (arr, 3)
[1, 9, 5]
>>> random. sample(range (300) , 5)
[259 , 294, 142 , 292 , 245]
```

9.14. Ro’yxatni satrga o’tkazish

Ro’yxatni satrga o’tkazish uchun join () metodidan foydalaniladi.

Elementlar ko’rsatilagan bo’luvchi bilan ajtratilib qo’shiladi.Metod formati:

<Satr> = <Bo'luvchi>. join (<Ketma-ketlik>)

Misol:

```
>>> arr = [ "wordl", "word2", "word3"]
>>> "- ". join (arr)
'word1 – word2 - word3'
```

Diqqat qiling, ro'yxat elementlari ichida satr bo'lmasa, TypeError qaytariladi:

```
>>>  arr  =  ["word1",  "word2",  "word3" ,  2]
>>>  " - ". join (arr)
Traceback (most recent call last):
File "<pyshell# 69>", line 1, in <module>
11 - ".  join(arr)
TypeError:  sequence item  3:  expected  str instance ,  int  found
```

Bu xatodan qochish uchun ro'yxat elementlarida satrga o'girish funksiyasi str () dan foydalanish kerak:

```
>>> arr = ["wordl ",  "word2",  "word3",  2]
>>> "-".join ((str (i) for  i  in arr) )
'word1 - word2 - word3 - 2'
```

Bundan tashqari str () funksiyasi yordamida birdaniga ro'yxatni satr ko'rinishda tasvirlash mumkin:

```
>>> arr  =  ["wordl ",  "word2 ",  "word3 ",  2]
>>> str(arr)
"[ 'wordl ',  'word2 ',  'word3 ',  2]"
```

9.15. Kortejlar

Kotrejlar ro'yxatlar kabi elementlarning tartiblangan ketma-ketligidir. Ular ro'yxatlar bilan ko'p jixatalari bir hil, biroq bir juda muhim farqli tomoni – kortejni o'zgartirish mumkin emas. Aytish mumkinki, kortej – bu faqat o'qish uchun ruxsat etilgan ro'yxatdir.

Kortej yaratish quyidagi usullarda amalaga oshiriladi:

- tuple ([<Ketma-ketlik>]) funksiyasi yordamida. Funksiya ixtiyoriy ketma-ketlikni kortejga o'tkazishga imkon beradi. Agar parameter ko'rsatilmasa, bo'sh kortej yaratiladi. Masalan:

```
>>> tuple ( )                # Bo'sh kortej yaratamiz
>>> tuple ("String" ) #  Satrni kortejga o'tkazamiz
(' S' ,  ' t' ,  'r ' ,  'i',  'n' ,  'g' )
>>> tuple ([ l,  2,  3,  4,  5] )  #Ro'yxatni kortejga o'tkazamiz
(1,  2,  3,  4,  5)
```

- barcha elementlarni vergul bilan ajratilgan holda aylanma qavslar ichida (yoki qavssiz) ko'rsatish orqali:

```
>>> t1 = ()  # Bo'sh kortej yaratamiz
>>> t2  =  (5, )  #  Bir elementli kortej yaratamiz
>>> t3  = (1, "str" ,  (3,  4) )  #  Uch elementli kortej yaratamiz
>>> t4  = 1, "str" ,  (3,  4)  #  Uch elementli kortej yaratamiz
>>> t1,  t2 ,  t3 ,  t4
(( )  ,  (5, )  ,  (1,  ' str '  ,  (3,  4) )  ,  (1,  'str '  ,  (  3,  4)  )  )
```

Ikkinchi qatordagi misolga alohida e'tibor qarating. Agar bir elementli kortej yaratishda elemntdan so'ng albatta vergul qo'yish kerak. Aks holda bu boshqa turdagi obyekt yaratiladi. Masalan:

```
>>> t  =  (5) ;  type (t)
<class  'int'>  #  Kortej emas, son yaratildi!
>>>  t  =  ("str ") ;  type (t)
<class  'str '>  #  Kortej emas, satr yaratildi!
```

Berilgan misoldagi to'rtinchi qator qavssiz vergul yordamida ham kortej yaratish mumkinligigni ko'rasatmoqda. Shuni yodda saqlangki, Python dasturlash tilidaqavs ichida berilgan ifoda vergullar bilan ajratilsa u kortej tipiga o'zgarardi.

Elementlar o'rni indekslar orqali aniqlanadi. Ro'yxat kabi elementlar tartibi 1 dan emas 0 dan boshlanadi. Barcha ketma-ketliklar kabi kortejlar ham elementga indeks orqali murojaat qilish, qirqib olish, konkatenasiya(+), takrorlash(*) va

tegishlilikka (in) yoki tegishli emaslikka (not in) tekshirish amallarini qo'llay oladi. Masalan:

```
>>> t = (1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> t[0] # Kortejning birinchi elementi qiymatini olamiz
1
> > > t [ :, : -1 ]          # Elemetlar tartibini o'zgartiramiz
(9, 8, 7, 6, 5, 4, 3, 2, 1)
>>> t[ 2:5 ]          # Qirqim olamiz
(3, 4, 5)
>>> 8 in t, 0 in t # Tegishlilikka tekshiramiz
(True , False)
>>> (1, 2, 3) * 3 # Takrorlash
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> (1, 2, 3) + (4, 5, 6) # Kokatenasiya
(1, 2, 3, 4, 5, 6)
```

Kortej o'zgarmas ma'lumot tipiga tegishli. Bu indeks bo'yicha elementni olish mumkinligini, ammo o'zgartirish mumkin emasligini anglatadi:

```
>>> t = (1, 2, 3)          # Kortej yaratamiz
>>> t[0]                   #Indeks bo'yicha elementni olamiz
1
>>> t[0] = 50         #Indeks bo'yicha elementni o'zgartirish mumkin emas
Traceback (most recent call last):
File "<pyshell#95>" , line 1, in <module>
t[0] = 50 #Indeks bo'yicha elementni o'zgartirish mumkin emas
TypeError : 'tuple ' object does not support item assignment
```

Kortejlar len( ), min( ), max( ) funksiyalarni hamda index( ) va count( ) metodlarini qo'llay oladi. Masalan:

```
>>> t = (1, 2, 3) #Kortej yaratamiz
>>> len (t) #Elementlar sonini aniqlaymiz
3
```

```
>>> t = (1, 2, 1, 2, 1 )
>>> t. index(l) , t. index (2) # KOrtejadan elementlarni qidiramiz
(0, 1)
```

9.16. To’plamlar

To’plam bu tartiblanmagan unikal elementlar ketma-ketligidir. To’plamni e’lon qilish uchun set() funksoiyasidan foydalaniladi:

```
>>> s = set ()
>>> s
set ([ ])
```

set() funksiyasi shuningdek, elementlar ketma-ketligini to’plamaga o’tkazishga ham imkon beradi:

```
>>> set ("string" ) # Satrni o’tkazamiz
set (['g', ' i ', 'n ', 's' , 'r', 't'])
>>> set([l, 2, 3, 4, 5]) # Ro’yxatni o’tkazamiz
set ([1, 2, 3, 4 , 5] )
>>> set( (l, 2, 3, 4, 5)) # Kortejni o’tkazamiz
set ([1, 2, 3, 4, 5] )
>>> set([l, 2, 3, 1, 2, 3]) #Faqat unikal elementlar qoldiriladi
set ([1, 2, 3])
```

for sikli to’plam elementlarini olish (o’tkazish, ko’rsatish) uchun imkon beradi:

```
>>> for i in set([l, 2, 3] ): print i
1 2 3
```

To’plamdai elementlar sonini aniqlash uchun len() funksiyasi ishlatiladi:

```
>>> len (set ([l, 2, 3]))
3
```

To’plamlar bilan ishlashda quyidagi operator va metodlar ishlatiladi::

- | va union() - ikki to’plamni birlashtirish:

```
>>> s = set ([1, 2, 3])
>>> s.union (set ((4, 5, 6])), s | set ((4, 5, 6] )
(set ([1, 2, 3, 4, 5, 6]), set ([1, 2, 3, 4, 5, 6]))
```

Agar element allqachon to’plam tarkibida bo’lsa, qayta to’palamga qo’shilmaydi:

```
>>> set ((1, 2, 3] ) | set ((1, 2, 3])
set ([1, 2, 3])
```

- a |=b va a. update (b) – b to’plam elementlarini a to’plamaga qo’shish:

```
>>> s= set ([l, 2, 3])
>>> s.update (set([ 4, 5, 6] ))
>>> s
set ([1, 2, 3, 4, 5, 6 ])
>>> s | = set ([7, 8, 9] )
>>> s
set ([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- – va difference () – to’plam farqini aniqlash:

```
>>> set ([1, 2, 3]) - set ([1, 2, 4])
set ([3] )
>>> s = set ([1, 2, 3])
>>> s.difference (set ([l, 2, 4] ))
set ([3])
```

- a -= b va a.difference_update(b) – a va b to’plamdan elementlarni удаляют элементы из множества a, которые существуют и во множестве a, и во множестве b:

```
>>> s= set ([l, 2, 3])
>>> s.difference_update (set ([l, 2, 4]))
>>> s
set ([3])
>>> s - = set ([3, 4, 5])
>>> s
set ([ ])
```

- & va intersection() – to’plamalar kesishmasi. Har ikkala to’plamdagi ham mavjud elementlarni olishga imkon beradi:

>>> set ([1, 2, 3]) & set ([1, 2, 4])

set ([1, 2])

>>> s == set ([l, 2, 3])

>>> s. intersection(set ([l, 2, 4]))

set ([l, 2])

- a & = b va a.intersection_update(b) - во множестве a останутся элементы, которые существуют и во множестве a, и во множестве b:

>>> s == set([1, 2, 3])

>>> s.intersection_update (set ([l, 2, 4]))

>>> s

set ([1, 2])

>>> s &= set ([1, 6, 7])

>>> s

set ([1])

^ va symmetric_difference() - возвращают все элементы обоих множеств, исключая элементы, которые присутствуют в обоих этих множествах:

>>> s = set ([1, 2, 3])

>>> s^set ([l, 2, 4]), s.syrnmetric_difference (set ([l, 2, 4]))

(set([3, 4]), set([3, 4]))

>>> s^set ([l, 2, 3]), s. symmetric_difference (set ([l, 2, 3]))

(set ([ ]) , set ([ ]))

>>> s^set([4, 5, 6]), s.s ymm etri c_ di fference (set ([4, 5, 6]))

(set ([1, 2, 3, 4, 5, 6]), set ([1, 2, 3, 4, 5, 6]))

- a ^ = b va a.symmetric_difference_update(b) – во множестве a будут все элементы обоих множеств, исключая те, что присутствуют в обоих этих множествах:

>>> s = set([l, 2, 3])

>>> s. symmetric _difference_update (set ([l, 2, 4]))

>>> s

set ([3, 4])

>>> s ^ = set ([3, 5, 6])

>>> s

set ([4, 5, 6])

To’plamalrni taqqoslash operatorlari:

- in – to’plamda elementni mavjudlikka tekshirish:

>>> s = set([l, 2, 3, 4, 5])

>>> 1 in s, 12 in s

(True, False)

- not in - to’plamda elementni mavjud emaslikka tekshirish:

>>> s= set([l, 2, 3 ; 4, 5])

>>> 1 in s, 12 in s

(False, True)

- = = – tenglikka tekshirish:

>>> set ([1, 2, 3]) == set ([1, 2, 3])

True

>>> set ([1, 2, 3]) == set ((3, 2, 1])

True

>>> set ([1, 2, 3]) = set ([1, 2, 3, 4])

False

- a <= b va a.issubset (b) – a to’plamning barcha elementlari b to’plam tarkibiga kirishini tekshirish:

>>> s = set ([l, 2, 3])

>>> s <= set ([l, 2]), s <= set([l, 2, 3, 4])

(False, True)

>>> s. issubset (set ([l, 2])), s.issubset (set([l, 2, 3, 4]))

(False , True)

- a < b – a to’plamning barcha elementlari b to’plam tarkibiga kirishini va b to’plamga teng bo’lmasligini tekshirish:

>>> s = set([l, 2, 3])

>>> s < set([l, 2, 3]), s < set([l, 2, 3, 4])

(False, True)

- a > = b va a.issuperset (b) – b to’plamning barcha elementlari a to’plam tarkibiga kirishini tekshirish:

```
>>> s= set( [l, 2, 3] )
>>> s>= set( [l, 2) ), s >= set( [l, 2, 3, 4) )
(True, Fa lse )
>>> s.issuperset (set ([l, 2])), s.issuperset (set([l, 2, 3, 4]))
(True, False)
```

- a > b – b to’plamning barcha elementlari a to’plam tarkibiga kirishini va a to’plam b to’plamga teng bo’lmasligini tekshiradi:

```
>>> s = set([ l, 2, 3])
>>> s > set([l, 2]), s > set ([ l, 2, 3] )
(True, False)
```

- isdisjoint (b) – a va b to’plamlarda birorta mos element bor yoki yo’qlikka ya’ni butunlay farqlilikka tekshiradi:

```
>>> s = set([ l, 2, 3])
>>> s. isdisjoint (set ([4, 5, 6]))
True
>>> s. isdisjoint (set([l, 3, 5]))
False
```

To’plamlar bilan ishlash uchun quyidagi metodlardan foydalaniladi:

- copy () – to’plam nusxasini yaratadi. Diqqat qiling, = operatori bilan o’zlashtirishda faqatgini shu obyektning o’ziga havola o’zlashtiriladi, u nusxalanmaydi. Masalan:

```
>>> s = set ( [l, 2, 3] )
>>> c = s; s is c          # = yoramida olinganda nusxa olish mumkin emas!
True
>>> c = s.copy()          # Obyekt nusxasini yaratamiz
>>> c
set ( [1, 2, 3] )
```

```
>>> s  is  c                    #  Endi bular turli obyekt
False
```

add (<Element>) – To'plamaga < Element> qo'shadi:

```
>>> s  =  set ( [1, 2, 3] )
>>> s.add(4);  s
set ([1, 2, 3, 4] )
```

- remove (<Element>) – to'plamdan <Element>ni o'chiradi. Agar element topilmasa, KeyError istisnosi paydo bo'ladi:

```
>>> s  =  set([ l,  2,  3] )
>>> s.remove (3) ;  s
set ([1, 2])
>>> s.remove (5)
```

- discard(<Element>) –to'plamdan <Element> ni agar u mavjud bo'lsa o'chiradi. Agar ko'rsatilgan element mavjud bo'lmasa, hech qanday xatolik ro'y beramaydi:

```
>>>  s =  set ( [1, 2, 3] )
>>>  s.discard(3) ;  s              #  Element mavjud
set  ( [1,  2] )
>>> s.discard(5);  s               # Element mavjud emas
set ([1, 2])
```

- pop () – to'plamdan ixtiyoriy elementni o'chiradi va uni qaytaradi. Agar elementlar bo'lmasa, KeyError xatoligi paydo bo'ladi:

```
>>> s = set ([1, 2])
>>> s.pop (),  s
(1, set ([2]) )
>>> s.pop() ,  s
(2, set ([ ]) )
>>>  s.pop( )  #  Agar element bo'lmasa, xatolik ro'y beradi
Traceback  (most  recent  call  last) :
```

File "<pyshell #89>" , line 1, in <module>

s. pop() # Agar element bo'lmasa, xatolik ro'y beradi

KeyError : 'pop from an empty set '

- clear () – to'plamdan barcha elementlarni o'chirish:

>>> s = set ([l, 2, 3])

>>> s. clear ( ) ; s

set ([ ])

9.17. Diapazonlar

Diapazon bu boshlang'ich va ohirchi chegaralari hamda qadami (qo'shni sonlar orasidagi farq) berilgan butun sonlardir. Ro'yxatlar, kortejlar, va to'plamlar kabi ketma-ketlik bo'lib, kortejlarga o'xshab o'zgarmas hisoblanadi.

Diapazonlar asosan qandaydir tegishlilikni tekshirish uchun va sikllarni tashkillash uchun foydalaniladi.

Diapazon yaratish uchun range () funksiyasidan foydalaniladi:

range ([<Boshlang'ich qiymat> ,] <Ohirgi qiymat> [,<Qadam>])

Birinchi parametr – Boshlang'ich qiymat ko'rsatilmasa, qiymat sifatida 0 olinadi.

Ikkinchi parametr diapazon ohirgi qiymatni anglatadi va o'zi shu diapazonga kirmaydi. Agar <Qadam> parametri ko'rsatilmasa, qiymat sifatida 1 dan foydlaniladi. Misollar:

>>> r = range (l, 10)

>>> for i in r:print (i, end = " ")

1 2 3 4 5 6 7 8 9

>>> r = range (10, 110, 10)

>>> for i in r: print (i, end = " ")

10 20 30 40 50 60 70 80 90 100

>>> r = range (10, 1, - 1)

>>> for i in r : print (i, end = " ")

10 9 8 7 6 5 4 3 2

Diapazondan ro’yxatga, kortejga, oddiy yoki o’zgarmas top’lamga o’tkazish uchun mos ravishda list (), tuple (), set( ) yoki frozenset () funksiyalaridan foydalanish mumkin:

```
>>> list (range (l, 10))          # Ro’yxatga o’zgartiramiz
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> tuple (range (l, 10))         # Kortejga o’zgartiramiz
(1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> set (range (l, 10))           # To’plamga o’zgartiramiz
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Diapazon elementga indeks bo’yicha murojaat qilish, qirqim olish, (natija sifatida yana diapazon qaytariladi), tegishli yoki tegishli emaslikka tekshirish len (), min (), max () funksiyalari, index () va count () metodlari. Misollar:

```
>>> r = range (l,  10)
>>> r[2] ,  r [-1]
(3,  9)
>>> r[2:4]
range (3,  5)
>>> 2  in r,  12 in  r
(True,  False )
>>> 3 not in  r,  13  not  in  r
(False, True)
>>> len(r),  min(r) ,  max (r)
(9,  1,  9)
>>> r.index( 4),  r.count (4)
(3, 1)
```

Python 3.3 versiyasida ikki diapazonni taqqoslash imkoniyati paydo bo’ldi:

- == – agar diapazonlar teng bo’lsa true qaytariladi aks holda False. Agar diapazonlar bir hil ketma-ketlikdagi sonlardan iborat bo’lsa ular teng hisoblanadi. Misollar:

```
>>> range (l, 10) == range (l, 10, 1)
True
>>> range (l, 10, 2) == range (l, 11,  2)
True
>>> range (l, 10, 2)  == range (l,  12 ,  2)
False
```

- **!** **=** – agar diapazonalar teng bo'lmasa True, aks holda False qaytariladi:

```
>>> range (l, 10, 2) != range (l,  12,  2)
True
>>> range (l, 10) != range (l,  10,  1)
False
```

Python 3.4 versiyadan boshlab esa diapazonlar mos ravishda boshlang'ich, ohirgi chegarani va qadamni aniqlovchi start, stop va step hususiyatalarni qo'llay boshladi:

```
>>> r = range (l, 10)
>>> r.start,  r.stop,  r.step
(1, 10, 1)
```

9.18. Itertools moduli

itertools moduli turli ketma-ketliklar asosidagi ketma-ketliklarni generasiya qilishga imkon beruvchi funksiyalarni o'z ichiga oladi.

Barcha funksiyalar takrorlashni qo'llab-quvvatlaydigan obyektlarni qaytaradi. Funksiyalarni ishlatishdan oldin quyidagicha modulni ulash kerak:

```
import itertools
```

Noaniq qiymatlar generatsiyasi

Noaniq sondagi qiymatlarni generasiya qilsh uchun quyidagi funksiyalar mo'ljallangan:

- count ([start=0] [, step=1]) – cheksiz o'suvchi qiymatlar ketma-ketligini yaratadi. Boshlang'ich qiymat start, qadam step parametrlarida belgilanadi. Misol:

```
>>> import itertools
```

```
>>> for i in itertools.count( ):
if i >10: break
print (i, end=" ")
0 1 2 3 4 5 6 7 8 9 10
>>> list (zip (itertools.count( ), "абвгд"))
[(0, 'a'), (1, 'б '), (2, 'в'), (3, ' г '), (4, 'д')]
>>> list (zip (itertools.count (start=2, step=2), "абвгд"))
[(2,  'a'),  (4,  'б'),  (6,  'в' ),  (8,  ' г ') , (10 ,  ' д ')]
```

➢ **cycle (<Ketma-ketlik >)** – har bir iterasiya ketma-ketlikning navbatdagi elementini qaytaradi. Qachonki ketma-ketlik tugasa, saralash yana boshidan boshlanadi. Masalan:

```
>>> n =  1
>>> for i in itertools.cycle ("абв"):
if  n > 10 : break
print (i, end=" ")
n += 1
а б в а б в а б в а
>>>  list (zip( itertools.cycle ([0, 1]), "абвгд" ))
[(0,  'a' ),  (1,  'б'),  (0, 'в' ),  (1,  'г'),  (0,  'д')]
```

➢ repeat (<Obyekt> [, <Qaytarilish soni>]) – obyektni ko’rsatilgan sonda takrorlanadi. Agar takrorlanish soni ko’rsatilmasa, obyekt cheksiz takrorlanadi. Masalan:

```
>>>  list (itertools.repeat (l,  10))
[1,  1,  1,  1,  1,  1,  1,  1,  1,  1]
>>>  list (zip (itertools. repeat (5), "абвгд" ))
[( 5,  ' а '),  (5,  ' б '),  (5,  'в '),  (5,  'г '),  (5,  'д' )]
```

Qiymatlar kombinasiyasi generasiyasi

Turli hil qiymatlar kombinasiyasini olish uchun quyidagi funksiyalardan foydalaniladi:

- **combinations ()** –ko’rsatilgan ketma-ketlikdan elemantlar sonicha yangi ketma-ketliklarni kortej sifatiada qaytariladi. Kortejdagi elementlar turlicha generasiyalanadi. Funksiya umumiy formati:

combinations (<Ketma-ketlik>, <Elementlar soni>)

Misollar:

```
>>> import itertools
>>> list (itertools.combinations ('абвг', 2) )
[(' а ', 'б' ), (' а' , 'в '), (' а ' , 'г' ), ( 'б' , ' в' ), (' б' , 'г' ),
( 'в ' , 'г' )]
>>> ["" . join(i) for i in itertools.combinations ('абвг' , 2)]
['аб', 'ав', 'аг', 'бв', 'бг', 'вг' ]
>>> list (itertools.combinations ('вгаб', 2))
[('в', 'г'), ('в', 'а'), ('в', 'б'), ('г', 'а'), ('г ', 'б'), ('а', 'б')]
>>> list (itertools. combinations ('абвг', 3) )
[('а', 'б' , 'в' ), (' а' , ' б' , 'г' ), (' а ', 'в' , 'г' ), ( 'б', 'в', 'г' )]
```

combinations_with _replacement() – har bir iterasiyada ko’rsatilgan sondagi elemntlar kombinasiyasidagi kotrejni qaytaradi. Bu kortej yagona element bo’lishi mumkin. Funksiya formati:

combinations_with _replacement (<Ketma-ketlik>, <Elementlar soni>)

Misollar:

```
>>> list(itertools.combinations_with_replacement('абвг', 2))
[( ' а ' , 'а ' ), (' а' , 'б ' ), (' а' , ' в' ), (' а' , 'г '), (' б ' , ' б' ),
('б', 'в '), ('б', 'г ' ), ('в' , 'в ' ), (' в' , 'г '), ( 'г ' , 'г ' ))
>>> list(itertools.combinations_ with_replacement ('вгаб', 2) )
[(' в ' , 'в ' ), ( ' в' , 'г' ), ( ' в ' , ' а ' ), (' в' , 'б '), ( ' г' , 'г' ),
(' г', 'а ' ), ( 'г' , 'б '), (' а' , ' а' ), ( ' а', 'б' ), (' б' , 'б ')]
```

- **permutations ()** – har bir iterasiyada ko’rsatilgan sondagi elementlar kombinasiyasidan iborat kortej qaytariladi. Agar elementlar soni ko’rsatilmasa, ketma-ketlik uzunligidan foyadalaniladi. Funksiya formati:

permutations (<Ketma-ketlik>[, <Elementlar soni>])

Misollar:

>>> list (itertools.permutations ('абвг', 2))

[('а', 'б'), ('а' , 'в'), (' а ' , 'г '), (' б ' , 'а '), (' б ' , 'в '), ('б' , 'г'), (' в' , ' а'), (' в' , 'б'), ('в' , ' г'), (' г' , ' а '), (' г ' , 'б '), (' г' , 'в ')]

>>> [''.join (i) for i in itertools. permutations ('абвг ')]

['абвг', 'абгв ' , ' авбг ' , 'авгб' , 'агбв ' , 'агвб' , 'бавг' ,

' багв ' , ' бваг ' , ' бвга ' , 'бгав ' , 'бгва ' , ' вабг ' , 'вагб ' ,

'вбаг ' , 'вбга ' , 'вгаб ', 'вгба' , 'габв' , 'гавб' , 'гбав ',

'гбва' , 'гваб' , 'гвба']

- **product ( )** - har bir iterasiayada bir yoki bir nechta ketma-ketlik elementlaridan hosil qilingan kombinasiayalar kortejini qaytaradi. Funksiya formati:

product (<Ketma-ketlik l> [, ... , < Ketma-ketlik N>) [, repeat=l])

Misollar:

>>> from itertools import product

>>> list (product ('абвг', repeat=2))

[('а', 'а'), ('а', 'б'), ('а', 'в'), ('а', 'г'), ('б', 'а'), ('б', 'б'), ('б', 'в'), ('б', 'г'), ('в', 'а'), ('в', 'б'), ('в', 'в'), ('в', 'г'), ('г', 'а'), ('г', 'б'), ('г', 'в'), ('г', 'г')]

>>> [''' .j o in (i) for i in product (' аб ', 'вг' , repeat=1)]

['ав', 'аг', 'бв', 'бг']

>>> [''' .j oin (i) for i in product (' аб ', 'вг ' , repeat=2)]

['авав', 'аваг', 'авбв', 'авбг', 'агав', 'агаг', 'агбв', 'агбг', 'бвав', 'бваг', 'бвбв', 'бвбг', 'бгав', 'бгаг', 'бгбв', 'бгбг']

9.19. Elementlar izchilligi filtrasiyasi

Ketma-ketlik elementlari filtra uchun quyidagi fuksiyalar mo'ljallangan:

- **filterfalse (<Funksiya>, <Ketma-ketlik>)** – birinchi parametr – <Funksiya> False qiymat qaytaradigan ketma-ketlik (har bir iterasiyada bittadan) elementlarini qaytaradi. Misollar:

>>> import itertools

>>> def func (x): return x > 3

>>> list (itertools.filterfalse (func, [4, 5, 6, 0, 7, 2, 3]))

[0, 2, 3]

>>> list (filter (func, [4, 5, 6, 0 , 7, 2, 3]))

[4, 5, 6, 7]

Agar birinchi farametrdagi funksiya nomi o’rniga None ko’rsatilsa, ketma-ketlikdagi har bir element False qiymatiga muvofiqligi tekshiriladi. Agar mantiqiy kontekstdagi element "True" qiymatini qaytarsa, u qaytarilgan natijaga kiritilmaydi. Misollar:

>>> list (itertools.filterfalse(None, [0, 5, 6, 0, 7, 0, 3]))

[0, 0, 0]

>>> list (filter (None , (0, 5, 6, 0, 7, 0, 3)))

[5, 6, 7, 3]

- dropwhile (<Funksiya>, <Ketma-ketlik>) – birinchi parametrda ko'rsatilgan funktsiya False qaytadigan elementdan boshlab ketma-ketlik elementlarini (har bir takrorlashda bittadan) qaytaradi. Misollar:

>>> def func(x) : return x > 3

>>> list (itertools.dropwhile (func , [4, 5, 6, 0, 7, 2, 3]))

[0, 7, 2, 3]

>>> list(itertools . dropwhile (func , [4, 5, 6, 7, 8]))

[ ]

>>> list (itertools.dropwhile(func, (1, 2, 4, 5, 6, 7, 8)))

(1, 2, 4, 5, 6, 7, 8)

Muhokama uchun savollar va topshiriqlar

1. Pythonda ro’yxatlar
2. Pythonda Ro’yxat yaratish.
3. Pythonda Ro’yxatlar ustida amallar.
4. Pythonda Ko’p o’lchamli ro’yxatlar.
5. Pythonda Ro’yxat elemantlarini saralash.

6. Pythonda Ro’yxat generatorlari
7. Pythonda map(), zip(), filter() va reduce() funksiyalari.
8. Pythonda Ro’yaxatga elementlar qo’shish va o’chirish.
9. Pythonda ro’yxat elementlarini qidirish
10. Pythonda ro’yxatga kirivchi qiymatlari haqida ma’lumot olish.
11. Pythonda ro’yxatni teskarilash va aralshtirish.
12. Pythonda tasodifiy elementni tanlash
13. Pythonda ro’yxatni saralash.
14. Pythonda ro’yxatni sonlar bilan to’ldirish.
15. Pythonda ro’yxatni satrga o’tkazish
16. Pythonda kortejlar.
17. Pythonda to’plamlar.
18. Pythonda diapazonlar.
19. Pythonda itertools moduli.
20. Pythonda noaniq qiymatlar generatsiyasi.
21. Pythonda qiymatlar kombinasiyasi generasiyasi.
22. Pythonda elementlar izchilligi filtri.

10-BOB. LUG’ATLAR

O`quv modullari

Lug’atlar, dict, dict.fromkeys, o'zgaruvchi ob'ekt, copy, traceback, get, metodi, lug'atdagi kalitlar, lug’at elementlari, metodlar.

Lug’atlar – bu elementlarga murojaat indekslar orqali emas kalitlar orqali murojaat qilinadigan obyektlar to’plamidir. Kalitlar sifatida o’zgarmas obyektlar – masalan sonlar, satrlar yoki kortejlarni ko’rsatish mumkin. Lug’at elementlari ixtiyoriy ma’lumot turidagi obyektlardan iborat. Yana shuni yodda shaqlash kerakki, lug’atdagi bu elementlar ixtiyoriy tartibda joylashishi mumkin. Agar elementni olish kerak bo’lsa qiymat saqlangan kalitdan foydalaniladi.

Lug’atlar ketma-ketliklarga tegishli emas. Shu sabab ketma-ketliklar uchun mo’ljallangan amallar qirqim olish, konkatenasiya, takrorlash va boshqalar lu’g’atlarda qo’llanilmaydi. Ro'yxatlar singari, lug'atlar ham o'zgaruvchan ma'lumot turi hisoblanadi. Boshqacha qilib aytganda, biz qiymatni nafaqat kalit orqali olishimiz, balki uni o'zgartirishimiz ham mumkin.

10.1. Lug’at yaratish

Lug’atni quyidagi usullar yordamida yaratish mumkin:

- dict() funksiyasi yordamida. Funksiya formati:

dict (<Kalit1>=<Qiymat1> [, ... , < KalitN>=<QiymatN>])

dict (<Lug’at>)

dict (<Ikki elementli kortej ro’yxati (Kalit, Qiymat)>)

dict (<Ikki elementli ro’yxat ro’yxati[Kalit, Qiymat]>)

Agar parametrlar ko’rsatilmasa, bo’s ro’yxat yaratiladi. Misollar:

```
>>> d = dict () ; d # Bo’sh lug’at yaratamiz
{}
>>> d = dict (a=1 , b=2 ); d
{' a' : 1, ' b': 2 }
>>> d = dict ({"a": 1, "b" : 2}) ; d # Lug’at
```

```
{ 'a' : 1,  'b' : 2}
>>>  d  =  dict ( [( "a",  1)  ,  ("b" ,  2) ] )  ;  d  #  Kortejlar ro'yxati
{' a ' :  1,  ' b' :  2}
>>>  d  =  dict ( [[ " a" ,  1]  ,  ["b ",  2] ]  )  ;  d  #  Ro'yxatlar ro'yxati
{'a': 1, 'b': 2}
```

Ikki ro'yxatni kortejlar ro'yxatida birlashtirish uchun zip () funksiyasi imkon beradi:

```
>>>  k  =  [" a" ,  "b"]            #  Kalitlar ro'yxati
>>>  v  =  [1,  2]                  #  Qiymatla ro'yxati
>>>  list (zip( k,  v) )            #  Kortejlar ro'yxatini yaratish
[( 'a',  1) ,  ('b' ,  2) ]
>>>  d = dict (zip( k,  v) ); d      # Lug'at yaratish
{'a' : 1,   'b' :  2}
```

- barcha lug'at elementlarini figurali (gulli) qavs ichida ko'rsatish orqali. Bu lug'at yaratishda ko'propq foydalaniladigan usul hisoblanadi. Kalitar va qiymatlar orasida ikki nuqta qo'yilishui va "kalit/qiymat" juftligi vergullar bilan ajratib yozilishi kerak. Masalan:

```
>>> d = {} ;  d      # Bo'sh lug'at yaratish
{}
>>> d = { "a" : 1, "b" : 2 } ;  d
{'a' : 1,  'b' : 2}
```

- Lug'atni elementlar bo'yicha to'ldirish. Bunda kalit kvadrat qavs ichida ko'rsatiladi:

```
>>>  d  =  {}                 # Bo'sh lug'at ayaratamiz
>>>  d[" a"]  =  1   # 1-elementni qo'shamiz (kalit "a")
>>>  d["b"] =2                # 2-elementni qo'shamiz (kalit "b")
>>>  d
{ 'a'  :  1,  'b ' :  2}
```

- dict.fromkeys (<Ketma-ketlik> [, <Qiymatalar>]) metodi yordamida. Metod kalitlari birinchi parametrda berilgan ketma-ketlik elementlaridan, qiymatlari ikkinchi parametrda berilgan qiymatlardan iborat yangi lug'at yaratadi. Agar ikkinchi parametrt ko'rsatilmasa, lug'at elementlari qiymati None bo'ladi. Masalan:

```
>>>  d = dict.fromkeys ([ "a", "b", "c"] )
>>>  d
{ 'a' : None ,  ' c ' : None,  ' b ' : None }
>>>  d  =  dict . fromkeys ([ "a", "b", "c "], 0)      #Ro'yxatni ko'rsatish
>>>  d
{' a': 0,  ' c' :  0,  'b': 0}
>>>  d =  dict . fromkeys (  (" a" ,  "b" ,  "c " ) ,  0)  #  Kortejni ko'rsatish
>>>  d
{' a' :  0,  'c' : 0,  'b' : 0}
```

Lug'at yaratishda o'zgaruvchi ob'ektga emas, balki ob'ektga havolani saqlaydi. Bu guruhni tayinlashda hisobga olinishi kerak. Siz raqamlar va satrlar uchun guruh topshirig'idan foydalanishingiz mumkin, ammo ro'yxatlar va lug'atlar uchun buni qila olmaysiz. Keling, bir misolni ko'rib chiqaylik:

```
>>> dl = d2 {"a": 1, "b": 2}              # Jakobs ikkita ob'ekt yaratdi
>>> d2 ["b"] = 10
>>> dl, d2                                # Ikki o'zgaruvchida qiymat o'zgartirildi! ! !
({'a': 1, 'b': 10}, {'a': 1, 'b': 10})
```

Misoldan ko'rinib turibdiki, d2 o'zgaruvchisidagi qiymatni o'zgartirish dl o'zgaruvchisidagi qiymatni ham o'zgartirdi. Ya'ni, har ikkala o'zgaruvchi ikki xil ob'ektga emas, balki bitta ob'ektga tegishli. Ikkita ob'ektni olish uchun siz alohida topshiriq qilishingiz kerak:

```
>>>  d1,  d2  =  {  "a" :  1,  "Ь" :  2  },  {  "a" :  1,  "Ь" :  2  }
>>>  d2 [" b" ]  =  10
>>>  d1,  d2
```

({' a ': 1, 'b': 2}, {'a': 1, 'b':10})

dict () funktsiyasi lug'atning yuzaki nusxasini yaratishga imkon beradi (9.1-misol).

9.1-misol. dict() funktsiyasidan foydalangan holda lug'atning sayoz nusxasini yarating

```
>>> d1 = {"a": 1, "b": 2} # Lug'at yarataimz
>>> d2 = dict (d1)          # Yuzaki nusxasini yaratamiz
>>> d1 - d2          # Operator bularning turli xil ob'ektlar ekanligini ko'rsatadi
False
>>> d2 ["b"] = 10
>>> d1, d2 # Faqat d2 dagi qiymat o'zgardi
({'a': 1, 'b': 2}, {'a': 1, 'b': 10})
```

Bundan tashqari, yuzaki nusxasini yaratish uchun litter () usulidan foydalanishingiz mumkin (10.2-misol)

10.2-misol. copy () usuli yordamida lug'atning yuzaki nusxasini yaratish

```
>>> d1= {"a": 1, "b": 2} # Lug'at yarating
>>> d2 = d1.copy ()                # Yuzaki nusxasini yarating
>>> dl - d2          # Operator bularning turli xil ob'ektlar ekanligini ko'rsatadi
False
>>> d2 ["b"] = 10
>>> d1, d2          # Faqat d2 dagi qiymat o'zgardi
({'a': 1, 'b': 2}, {'a': 1, 'b': 10})
```

Lug'atning to'liq nusxasini yaratish uchun nusxa ko'chirish () funktsiyasidan copy modulidan foydalaning (9.3 -misol).

9.3 -misol. Lug'atning to'liq nusxasini yaratish

```
>>> d1 = {"a": 1, "b":[20, 30, 40)}
>>> d2 = dict (d1) # Sayoz nusxasini yarating
```

```
>>> d2 ["b"] [0] = "test"
>>> dl, d2 # Ikki o'zgaruvchidagi qiymatlar o'zgardi !!!
({'a': 1, 'b': ['test', 30, 40]}, {'a': 1, 'b': ['t est', 30, 40]})
>>> impo rt soru
>>> d3 = nusxa ko'chirish. deepcopy (d l) # To'liq nusxasini yarating
>>> d3 ["b"] [1] = 800
>>> d1, d2 # O'zgargan qiymat faqat d2 o'zgaruvchida
({'a': 1, 'b': ['test', 30, 40]}, {'a': 1, 'b': ('test', 80 0, 40]})
```

10.2. Lug'atlar bo'yicha amallar

Lug'at elementlariga murojaat ichida kalit ko'rsatilgan to'rtburchak qavslar yordamida amalga oshiriladi. Kalit sifatida siz o'zgarmas ob'ektni belgilashingiz mumkin. Masalan: raqam, satr yoki katak. Lug'atning barcha elementlarini namoyish qilaylik:

```
>>> d = {1: "int", "a": "str", (1, 2): "tuple"}
>>> d [1], d ["a"], d [(1, 2)]
('int', 'str', 'tuple')
```

Agar ko'rsatilgan kalitga mos keladigan element lug'atda bo'lmasa, u holda KeyError qaytariladi:

```
>>> d = {"a": 1, "b": 2}
> >> d ["c"] # Mavjud bo'lmagan elementga murojaat qilish
Traceback (most recent call last ) :
File "<pyshell#4 9>", line 1, in <module>
d ["c"] # Mavjud bo'lmagan elementga murojaat qilish
KeyError: 'c'
```

in operatoridan foydalanib siz kalit mavjudligini tekshirishingiz mumkin. Agar kalit topilsa, u holda True qaytariladi, aks holda -False. Misollar:

```
>>> d = { "a" : 1, "b" : 2
```

```
>>> "a" in d # Kalit mavjud
True
>>> "c" in d # Kalit mavjud emas
False
```

not in operatori lug'atda kalit yo'qligini tekshirishga imkon beradi. Agar kalit yo'q bo'lsa, u True, aks holda, False qaytaradi. Misollar:

```
>>> d = { "a" : 1, "b" : 2 }
>>> "c" not in d              # Kalit mavjud emas
True
>>> "a" not in d              # Kalit mavjud
False
```

get (<Kalit> [, <Default>]) metodi agar belgilangan kalit lug'atda yo'q bo'lsa, KeyError istisnosini qaytarishning oldini olishga imkon beradi. Agar kalit lug'atda mavjud bo'lsa, unda usul ushbu kalitga mos keladigan qiymatni qaytaradi. Agar kalit mavjud bo'lmasa, u holda None yoki ikkinchi parametrda ko'rsatilgan qiymat qaytariladi. Misol:

```
>>> d = { "a" : 1, "b" : 2 }
>>> d.get (" a ") , d.get( "c ") , d.get (" c" , 800 )
(1, None , 800)
```

Shu bilan bir qatorda, setdefault (<Kalit> [, <Default>]) usulidan foydalanishingiz mumkin. Agar kalit lug'atda mavjud bo'lsa, unda usul ushbu kalitga mos keladigan qiymatni qaytaradi. Agar kalit yo'q bo'lsa, unda ikkinchi parametrda ko'rsatilgan qiymat bilan lug'atda yangi element yaratiladi. Agar ikkinchi parametr ko'rsatilmagan bo'lsa, yangi elementning qiymati None.

Misol:

```
>>> d = {"a": 1, "b": · 2}
```

```
>>> d. setdefault ("a"), d. setdefault ("c"), d. o'rnatilgan xato ("d", 0)
(1, yo'q, 0)
>>> d
{'a': 1, 'c': yo'q, 'b': 2, 'd': 0}
```

Lug'atlar o'zgaruvchan ma'lumotlar turlari bo'lgani uchun biz elementni kalit orqali o'zgartirishimiz mumkin. Agar ko'rsatilgan kaliti bo'lgan element lug'atda yo'q bo'lsa, u lug'atga qo'shiladi:

```
>>> d = {"a": 1, "b": 2}
>>> d ["a"]= 800                    # Elementni kalit yordamida o'zgartirish
>>> d ["c"] = "string"                    # Yangi element qo'shiladi
>>> d
{'a': 800, 'c': 'string', 'b': 2}
```

len () funktsiyasi lug'atdagi kalitlar sonini aniqlash imkonini beradi:

```
>>> d = {"a": 1, "b": 2}
>>> len (d)                  # Lug'atdagi kalitlar sonini aniqlash
2
```

del operatori yordamida lug'atdan elementni o’chirib tashlashingiz mumkin:

```
>>> d = {"a": 1, "b": 2}
>>> del d ["b"]; d  # "b" kalitli elementni o’chirib tashlaymiz va lug’atni chop etamiz
{'a': 1}
```

10.3. Lug’at elementlarini saralash

For sikli yordamida lug'atlar ketma-ketlik bo’lmasada uning barcha elementlarini ko’rib chiqish mumkin. Masalan, lug'at elementlarini ikki usul bilan chop etaylik. Birinchi usulda ob'ektni lug'at kalitlari bilan qaytaradigan key () usuli qo'llaniladi. Ikkinchi usulda biz lug'atni faqat parametr sifatida belgilaymiz. Har bir siklning takrorlanishi sikl ichidagi ushbu kalitga mos keladigan qiymatni olish uchun ishlatilishi mumkin bo'lgan kalitni qaytaradi (9.4-misol).

9.4-misol. Lug'at elementlarini ko'rib chiqish

```
d = {" x" : 1, " y" : 2, "z" : 3}
for key in d.keys( ) : # keys () metodidan foyadalanish
print( "({0} => {i}) ".format (key, d[key] ), end=" ")
# Natija: (y => 2) (x => 1) (z => 3)
print ()                # Satr tashlash
for key in d:       # Lug’at ham iterasiyani qo’llaydi
print ("({0} => {1 }) ".format (key, d[ key] ), end=" ")
# Natija: (y => 2) (x => 1) (z => 3)
```

Lug'atlar tartibsiz tuzilmalar bo'lgani uchun lug'at elementlari alohida tartibda namoyish etilmaydi. Kalitlar bo'yicha saralangan elementlarni ko'rsatish uchun tugmalar ro'yxatini olish va keyin sort () usulidan foydalanish kerak.

Masalan:

```
d = { "x" : 1, "y" : 2, "z" : 3}
k = lis t (d. keys () ) # Kalitlar ro’yxatini olamiz
k. sort () # Kalitlar ro’yxatini saralaymiz
for key in k:
print ("({0} => {1} )" .format (key, d[key] ), end=" ")
# Natija: (z => 3 ) (y => 2 ) (x => 1 )
```

Sort () usuli o'rniga sorted () funktsiyasidan foydalanib kalitlarni saralash mumkin.

Misol:

```
d = {" x" : 1, "y " : 2, "z " : 3}
for key in sorted (d. keys () ):
print( " ({ O} => {1} )" .format (key, d[ key] ) , end=" ")
# Natija: ( x => 1 ) (y => 2 ) ( z => 3)
```

Lug'at tugmachasi har bir takrorlashda qaytarilganligi sababli, sorted () funktsiyasini key () usulini bajarish natijasi o'rniga darhol lug'at ob'ekti uzatilishi mumkin:

```
d = {" x" : 1, "y" : 2, "z" : 3}
for key in sorted (d) :
print( " ({ O} => {1 }) ".format (k ey, d[key ] ), end=" ")
# Natija: (x => 1) (y => 2) (z => 3)
```

10.4. Lug'atlar bilan ishlash metodlari

Lug'atlar bilan ishlash uchun quyidagi metodlardan foydalaniladi:

- keys() – lug'atning barcha tugmachalarini o'z ichiga olgan dict_keys ob'ektini qaytaradi. Ushbu ob'ekt takrorlashni qo'llab-quvvatlaydi, shuningdek operatsiyalarni o'rnatadi. Misol:

```
>>> d1, d2 = { "a" : 1 , "b" : 2}, { "a" : 3 , "c" : 4, "d" : 5 }
>>> d1.keys() , d2 . keys () # dict_keys obyektini olamiz
( dict _ keys ( [ 'a' , 'b' ]) , dict _ keys ( [ 'a' , ' c ' , ' d' ] ) )
>>> list (d1.keys( )), list (d2. keys() ) # Kalitlar ro’yxatini olamiz
([' a', 'b' ], [' a' , 'c' , 'd ' ])
>>> for k in d1.keys( ) : print (k, end=" ")
a b
>>> d1.keys ( ) | d2 . keys () # Birlashtirish
{'a', 'c', 'b', 'd'}
>>> d1.keys () - d2.keys () # Farq
{'b'}
>>> d2.keys () - d1.keys () # Farq
{'c' 'd'}
>>> dl . keys () & d2 . keys () # Bir hil kalit
{'a'}
> > > d1.keys() ^ d2.keys ( ) # Unikal kalit
{ 'c', 'b', 'd' }
```

- values () – lug’atning barcha qiymatlaridan iborat dict_values obyektini qaytaradi. Bu obyekt iterasiyani qo’llaydi. Misollar:

```
>>> d = { "a" : 1, "Ь" : 2 }
>>> d.values ()                    # dict_values obyektini olamiz
dict_values ([1, 2) )
>>> list (d.values () )            #Ro'yxat qiymatlarni olamiz
[1, 2]
>>> [ v for v in d.values () ]
[1, 2]
```

- items () – kortej ko'rinishdagi barcha kalit va uning qiymatlarini dict_items obyektida qaytaradi. Bu obyekt iterasiyani qo'llab-quvvatlaydi. Misollar:

```
>>> d = { "a" : 1, "b" : 2 }
>>> d.items () # dict_items obyektini olamiz
dict_items ( [ ('a' , 1 ), ('b', 2) ] )
>>> list (d. items ())             # Kortejlar ro'yxatini olamiz
[( ' a ' , 1) , (' b' , 2) ]
```

- <Kalit> in <Lug'at> - lug'atda ko'rsatilgan kalit mavjudligini tekshiradi. Agar kalit topilsa True, aks holda False qaytariladi. Misollar:

```
>>> d = { "a ": 1, "b" : 2 }
>>> "a" in d                       # Kalit mavjud
True
>>> "c" in d                       # Kalit mavjud emas
False
```

- <Kalit> not in <Lug'at> - lug'atda ko'rsatilgan kalit mavjud emasligini tekshiradi. Agar kalit topilmasa True, aks holda False qaytariladi. Misollar:

```
>>> d = { " a" : 1, "b" : 2 }
>>> "c" not in d                   # Kalit mavjud emas
True
>>> "a" not in d                   # Kalit mavjud
False
```

- get (<Kalit> [, <Odatiy qiymat>]) - agar kalit lug'atda mavjud bo'lsa, u holda ushbu elementga mos keladigan qiymat qaytariladi. Agar kalit yo'q bo'lsa, u holda None yoki ikkinchi parametrda ko'rsatilgan qiymat qaytariladi. Misol:

```
>>> d = { "a" : 1, "b" : 2}
>>> d. get (" a ") , d.get (" c" ) , d.get (" c" , 800)
(1, None, 800)
```

- setdefault (<Key> [, <Default value>]) - agar kalit lug'atda mavjud bo'lsa, u holda usul ushbu kalitga mos keladigan qiymatni qaytaradi. Agar kalit yo'q bo'lsa, u ikkinchi parametrda ko'rsatilgan qiymat bilan lug'atda yangi element yaratadi. Agar ikkinchi parametr ko'rsatilmagan bo'lsa, yangi elementning qiymati None bo'ladi. Misol:

```
>>> d = {"a":1, "b": 2}
>>> d.setdefault ("a") ,d . setdefault (" c "), d.setdefault ("d", 0)
(1, None , 0)
>>> d
{'a' : 1, 'c' : None , 'b' : 2 , 'd' : 0}
```

- pop (<Kalit> [, <Odatiy qiymat>]) –elementni ko'rsatilgan kalit bilan olib tashlaydi va uning qiymatini qaytaradi. Agar kalit yo'q bo'lsa, unda ikkinchi parametrdan qiymat qaytariladi. Agar kalit yo'q bo'lsa va ikkinchi parametr ko'rsatilmagan bo'lsa, KeyError istisnosi qaytariladi. Misollar:

```
>>> d = { "a" : 1, "b" : 2, "c" : 3 }
>>> d.pop(" a ") , d.pop("n" , 0)
(1, 0)
>>> d.pop("n" ) # kalit va ikkinchi parametr
Traceback (most recent ca ll last ) :
File "<pyshell#40>" , line 1, in <module>
d.pop(" n" )                # kalit va ikkinchi parametr
KeyError : ' n '
>>> d
{ 'c' : 3, 'b' : 2}
```

- popitem () - ixtiyoriy elementni olib tashlaydi, kalit va uning qiymatnidan iborat kortejni qaytaradi.

Agar lug'at bo'sh bo'lsa, KeyError istisnosi qaytariladi. Misollar:

```
>>> d = { "a" : 1, "b" : 2 }
>>> d. popitem () # Ixtiyoriy elementni o’chiramiz
(' a ' , 1)
>>> d. popitem() # Ixtiyoriy elementni o’chiramiz
( ' b ', 2)
>>> d. popitem () # Lug’at bo’sh. Istisno qaytariladi
Traceback (most recent call last ) :
File "<pyshell#45>", line 1, in <module>
d.popitern () # Lug’at bo’sh. Istisno qaytariladi
KeyError : 'popitem() : dictionary is empty'
```

- clear () - lug'atning barcha elementlarini o’chirib tashlaydi. Usul hech narsani qiymat sifatida qaytarmaydi. Misol:

```
>>> d = { "a" : 1, "b" : 2 }
>>> d.clear ()          # Barcha elementni o’chiramiz
>>> d                   # Lug’at endi bo’sh
{ }
```

- update () - lug'atga elemntlar qo'shadi. Metod joriy lug'atni o'zgartiradi va hech narsa qaytarmaydi. Metod formatlari:

update (<Kalit1>=<Qiymatl> [, ... , <KalitN>=<QiymatN>])

update (<Lug’at>)

update (<Ikki elementli kortejlar ro'yxati>)

update (<Ikki elementli ro’yxatlar ro'yxati>)

Agar ko'rsatilgan kalit element lug'atda allaqachon mavjud bo'lsa, unda uning qiymati qayta yoziladi. Misollar:

```
>>> d = { "a" : 1, "b" : 2 }
>>> d. update (c =3 , d=4 )
>>> d
```

```
{' a': 1 ,  'c ' :  3 ,  'b' :  2,  'd' :  4}
>>>  d.  update ({"c":10,  "d": 20 })          #  Lug'at
>>>  d                                          #     Qayta    yozilgan    elementlar
```
ro'yxati
```
{ ' a ':  1,  ' c ':  10,  'b' :  2,  'd ' :  20 }
>>>  d.update ([("d" ,  80) ,  ("e",  6)])       #  Kortejlar ro'yxati
>>>  d
{'a':1, 'c':  10,  'b': 2, 'e': 6, ' d':80 }
>>>  d. update (  [[ "a" ,  "str" ],  ["i",  " t"] ])       #  Ro'yxatlar ro'yxati
>>>  d
{'a' :  'str ' ,  'c ' :  10,  ' b':  2,  ' e ' :  6,  ' d' :  80 ,  'i' :  't' }
```

- copy () – lug'atning yuzaki nusxasini yaratish:

```
>>>  d1  =  {  "a" :  1,  "b" :  [1 0,  20]  }
>>>  d2  =  d1 .copy()            #  Yuzaki nusxa yaratamiz
>>>  d1  is  d2                   #  Bular turlicha obyektlar
False
>>>  d2 ["a"]  =  800             #  Qiymatni o'zgartiramiz
>>>  dl ,  d2                     #  Faqat d2 dagi qiymat o'zgaradi holos
({ 'a':1,  'b' :  [10,  20] } ,  {'a' :  800,  'b' :  [10,  20] })
>>>  d2 [" b" ] [0]  =  "new"  #  ichki ro'yxatni o'zgartiramiz
>>>  d1, d2          #  d1va d2 qiymatlarni o'zgartiramiz  !
({ ' a' :  1,  'b':  ( ' new' ,  20 ] },  {' a' :  800 ,  'b' :  [ ' new ' ,  20) })
```

Lug'atning to'liq nusxasini yaratish uchun copy modulidagi deepcopy() funksiyasidan foydalaniladi.

10.5. Lug'at generatorlari

Ro'yxat generatorlaridan tashqari Python 3 lug'at generatorlarini ham qo'llab-quvvatlaydi. Lug'at generatorlari uchun sintaksis ro'yxat generatorlari sintaksisiga o'xshash bo'lib, farqi ikkita:

- ifoda to'rtburchak qavsga emas, balki jingalak qavslarga kiritilgan;

- for sikli oldidagi ifoda ichida ikkita emas, balki bitta nuqta bilan ajratilgan ikkita qiymat ko'rsatilgan. Ifoda ikki nuqta chap tomonida kalit bilan, o'ng tomonidagi element element qiymatiga joylashadi.

Misollar:

```
>>>  keys  =  ["a" ,  "b" ]                # Kalitlar ro’yxati
>>>  values  =  [1,  2]                    # Qiymatlar ro’yxati
>>> {k: v for (k, v) in zip (keys, values )}
{'a':  1,  'b' :  2}
>>> {k : 0 for k in keys }
{'a' : 0,  'b' : 0}
```

Lug'at generatorlari murakkab strukturali bo'lishi mumkin. Masalan, bir nechta ichma ish joylashgan for va (yoki) if operatorlari kombinasiyasidan iborat bo’lishi mumkin. Lug'atdan faqat juft qiymatli elementlarni o'z ichiga olgan yangi lug'at yarataylik:

```
>>> d = {"a":1, "b" : 2,  "c" : 3,  "d" :  4}
>>> { k : v  for  (k ,  v)  in  d. items()  if  v % 2 == 0 }
{'b' : 2,  'd' :  4}
```

Muhokama uchun savollar va topshiriqlar

1. Pythonda lug’atlar.
2. Pythonda lug’at yaratish.
3. Pythonda lug’atlar ustida amallar.
4. Pythonda lug’at elementlarini saralash.
5. Pythonda lug’atlar bilan ishlash metodlari.

   Pythonda lug’atlar generatori

11-BOB. SANA VA VAQT BILAN ISHLASH

O`quv modullari

Sava va vaqt modullari, «uxlab qolish» skripti, datetime moduli, time, datetime, calendar, timeit, gmtime, localtime, mktime, strftime, strptime, ctime.

Sava va vaqt modullari

Python tilida sana va vaqt bilan ishlash uchun quyidagi modullar mo'ljallangan:

- ✓ **time** - joriy sana va vaqtni olishga, shuningdek ularning formatlangan natijalarini chiqarishga imkon beradi;
- ✓ **datetime** - sana va vaqtni manipulyatsiya qilishga imkon beradi. Masalan, arifmetik amallarni bajarish, sanalarni taqqoslash, sana va vaqtni turli formatlarda ko'rsatish va boshqalar;
- ✓ **calendar** - kalendarni oddiy matnda yoki HTML formatida aks ettirishga imkon beradi;
- ✓ **timeit** - dasturni optimallashtirish uchun kichik kod qismlarining bajarilish vaqtini aniqlashga imkon beradi.

Joriy sana va vaqtni chop etish

Joriy sana va vaqtni olish uchun **time** modulidagi quyidagi funktsiyalar imkon brishi mumkin:

- ➢ **time () –** davr boshlangandan beri (odatda 1970 yil 1 yanvardan boshlab) soniya sonini aks ettiruvchi haqiqiy sonni qaytaradi:

```
>>>  import  time                # Modulni bog’laymiz
>>>  time.time ()                # Sekundlar miqdorini olamiz
1428057929.227704
```

- ➢ **gmtime ([<Soniyalar soni>])** – Universal vaqt(UTC)ni ifodalaydigan struct_time ob'ektini qaytaradi. Agar parametr ko'rsatilmagan bo'lsa, joriy vaqt qaytariladi. Agar parametr ko'rsatilgan bo'lsa, unda vaqt davr boshidan ko’rsatilgan vaqtgacha o'tgan soniyalar soniga to'g'ri keladi. Misollar:

```
>>> time.gmtime (0)              # Era boshidan beri
```

time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=1, tm_isdst=0)

>>> time.gmtime() #Joriy sana va vaqt

time.struct_time(tm_year=2020, tm_mon=9, tm_mday=8, tm_hour=12, tm_min=55, tm_sec=59, tm_wday=1, tm_yday=252, tm_isdst=0)

>>> time.gmtime(1428057929.0) # 03-04-2015

time.struct_time(tm_year=2015, tm_mon=4, tm_mday=3, tm_hour=10, tm_min=45, tm_sec=29, tm_wday=4, tm_yday=93, tm_isdst=0)

Ob'ekt ichida uning nomini yoki indeksini ko'rsatib, ma'lum bir atributning qiymatini olishingiz mumkin:

>>> d = time.gmtime ()

>>> d.tm_year, d[0]

(2020, 2020)

>>> tuple (d) # Kortejga o’tkazish

(2020, 9, 8, 12, 53, 35, 1, 252, 0)

- **localtime ([<Soniyalar soni>])** - mahalliy vaqtni ifodalovchi struct_time ob'ektini qaytaradi. Agar parametr ko'rsatilmagan bo'lsa, u holda joriy vaqt qaytariladi. Misollar:

>>> time.localtime ()

time.struct_time(tm_year=2020, tm_mon=9, tm_mday=8, tm_hour=19, tm_min=26, tm_sec=32, tm_wday=1, tm_yday=252, tm_isdst=0)

>>time.localtime(l428057929.0)

time.struct_time (tm_year=2 015, tm_m on=4 , tm_m day=3 , tm_hour=l 3, tm_min=45, tm _sec=2 9, tm_w day=4 , tm_yda y=93 , tm_i sdst=0)

- **mktime (<struct_time obyekti>)** - davr boshidan beri o'tgan soniyalar sonini aks ettiruvchi haqiqiy sonni qaytaradi. Parametr struct_time ob'ekti yoki to'qqizta elementli kortejdir. Agar belgilangan sana noto’g’ri bo'lsa, OverflowError istisnosi qaytariladi. Misol:

>>> d=time.localtime(1428057929.0)

>>> time.mktime(d)

```
1428057929.0
>>> tuple(time.localtime(1428057929.0))
(2015, 4, 3, 15, 45, 29, 4, 93, 0)
>>> time.mktime((2015,4,3,13,45,29,4,93,0))
1428050729.0
>>> time.mktime((1940,0,31,5,23,43,5,31,0))
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    time.mktime((1940,0,31,5,23,43,5,31,0))
OverflowError: mktime argument out of range
```

gmtime () va localtime () funktsiyalarida qaytarilgan struct_t ime ob'ekti quyidagi atributlarni o'z ichiga oladi ("Hususiyat nomi - indeks - tavsif" shaklidagi juftliklar ko'rsatilgan):

- tm_year – 0 - yil;
- tm_mon – 1 – oy (1 dan 12 gacha);
- tm_mday – 2 – oy kunlari (1 dan 31 gacha);
- tm_hour – 3 – soatlar (0 dan 23 gacha);
- tm_min – 4 – daqiqalar (0 dan 59 gacha);
- tm_sec – 5– soniyalar (0 dan 59 gacha);
- tm _wday – 6 – hafta kunlari (0 dushanbadan 6 yakshanbagacha);
- tm_yday – 7 – yil boshidan beri kunlar soni (1 dan 366 gacha);
- tm _isdst –8 – yozgi vaqtni to'g'rilash (0, 1 yoki -1 qiymatlari).

Biz joriy sana va vaqtni shu tarzda namoyish qilamizki, hafta va oy kuni o’zbek tilida yoziladi (11.1-misol).

11.1-misol. Hafta va oy kuni o’zbek tilida yozish

```
import time                          #  time modulini ulaymiz
d=["dushanba" ,  "seshanba" ,  "chorshanba",  "payshanba","juma",  "juma",
"shanba"]
```

```
m =["",  "yanvar",  "fevral" ,  "mart",  "aprel" ,  "may", "iyun",  "iyul",
"avgust",  "sentyabr",  "oktayabr", "noyabr",  "dekabr"]
t=time.localtime()                  #  joriy vaqtni olamiz
print("Bugun:\n%s %s %s %s
%02d:%02d:%02d\n%02d.%02d.%02d"%(d[t[6]],t[2],m[t[1]],
t[0],  t[3], t[4], t[5],t[2], t[1], t[0]))
input ()
```

Dastur natijasi:

Bugun:

seshanba 15 sentyabr 2020 17:45:04

15.09.2020

Sana va vaqt formati

Sana va vaqtni formatlash time modulidan quyidagi funktsiyalar bilan amalga oshiriladi:

- **strftime (<Qator formati> [, <struct_time obyekti>])** – satr formatiga muvofiq sana satrini qaytaradi. Agar ikkinchi parametr ko'rsatilmagan bo'lsa, joriy sana va vaqt ko'rsatiladi. Agar ikkinchi parametr struct_time ob'ekti yoki to'qqiz elementli kortejni aniqlasa, u holda sana belgilangan qiymatga mos keladi. Funksiya mahalliy sozlamalarga bog'liq. Misollar:

```
>>> import time
>>> time.strftime("%d.%m.%Y")       #Formatlangan sana
'15.09.2020'
>>> time.strftime("%H.%M.%S")      #Fornatlanagan vaqt
'18.01.26'
```

- **strptime (<Sanali satr > [, <Satr formati>])** - birinchi satrda ko'rsatilgan satrni format satriga mos ravishda ajratib beradi. struct_time ob'ektini qaytaradi. Agar satr formatga mos kelmasa, ValueError istisnosi qaytariladi. Agar satr formati ko'rsatilmagan bo'lsa, "%a %b %d %H: %M: %S %Y" formatidan fouydalaniladi. Misollar:

```
>>> time.strptime("Fri Apr 03 14:01:34 2015")
```

time.struct_time (tm_year=2 015, tm_ mon=4, tm_m day=3 , tm_hour= l4, tm_min=l, tm_sec=3 4, tm_wday=4 , tm _yday=94 , tm_i sdst=-1)

>>> time.strptime("03.04.2015","%d.%m.%Y")

time.struct_time(tm_year=2015, tm_mon=4, tm_mday=3, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=4, tm_yday=93, tm_isdst=-1)

>>> time.strptime("03-04-2015","%d.%m.%Y")

…format mos emas…

ValueError: time data '03-04-2015' does not match format '%d.%m.%Y'

- asctime ([<struct_time obyekti>]) - "% a% b% d% N:% M:% C% Y" formatidagi qatorni qaytaradi. Agar parametr ko'rsatilmagan bo'lsa, joriy sana va vaqt chiqariladi. Agar parametr struct_time ob'ekti yoki to'qqizta elementdan iborat kortejni aniqlasa, u holda sana ko'rsatilgan qiymatga mos keladi. Misollar:

>>> time.asctime()

'Tue Sep 15 18:49:20 2020'

>>> time.asctime(time.localtime(1321954972.0))

'Tue Nov 22 14:42:52 2011'

- **ctime ([<Soniyalar soni>])** - funktsiya asctime() ga o'xshaydi, lekin parametr sifatida struct_time ob'ektini emas, balki davr boshidan beri o'tgan soniyalar sonini oladi. Misol:

>>> time.ctime()

'Tue Sep 15 18:51:32 2020'

>>> time.ctime(456485564)

'Tue Jun 19 14:32:44 1984'

strftime() va strptime () funktsiyalaridagi <Satr formati> parametrida quyidagi maxsus belgilar kombinatsiyasilaridan foydalanish mumkin:

- % y – ikki xonali yil ("00" dan "99" gacha);
- % Y – to'rt xonali yil (masalan, "2020");

- % m – oldida nolga ega bo'lgan oy raqami ("01" dan "12" gacha);
- % b – mahalliy sozlamalarga qarab oyning qisqartmasi (masalan, yanvar uchun "Yan");
- % B – mahalliy sozlamalarga qarab oy nomi (masalan, "Yanvar");
- % d – oldida noli bo’lgan bo'lgan oy kun soni ("01" dan "31" gacha);
- % j – yil boshidan kun soni ("001" dan "366" gacha);
- % U – yilning hafta raqami ("00" dan "53" gacha). Hafta yakshanba kuni boshlanadi. Yil boshidan birinchi yakshanbagacha bo'lgan barcha kunlar o hafta soni;
- % W – yilning hafta raqami ("00" dan "53" gacha). Hafta dushanba kuni boshlanadi. Yil boshidan birinchi haftagacha bo'lgan barcha kunlar o raqamli haftaga yo'naltiriladi;
- % w – haftaning kun soni (yakshanba kuni "0", shanba kuni "6");
- % a –mahalliy sozlamalarga qarab haftaning kun qisqartmasi (masalan, dushanba uchun "dushanba");
- % A – mahalliy sozlamalarga qarab haftaning kun nomi (masalan, "dushanba");
- % H – 24 soatlik formatda soat ("00" dan "23" gacha)
- % I – 12 soatlik formatda soat ("01" dan "12" gacha);
- % M – daqiqa ("00" dan "59" gacha);
- % s – soniya ("00" dan "59" gacha, vaqti-vaqti bilan "61" gacha);
- % p – joriy tilda AM va PM qiymatlariga teng;
- % s – sana va vaqtni joriy tilda aks ettiradi;
- % x – sana joriy tilda aks etishi;
- % X – joriy tilda vaqtni aks ettirish. Misol:

```
>>> import  locale
>>> locale.setlocale(locale.LC_ALL, "Uz")
'Uz'
>>> print(time.strftime("%x"))                # Sanani ko’rsatish
15/09/2020
>>> print(time.strftime("%X"))                # Vaqtni ko’rsatish
```

20:26:08

>>> print(time.strftime("%c")) # Sanani va vaqtni ko’rsatish

15/09/2020 20:27:21

- % Z - vaqt zonasi nomi yoki bo'sh satr (masalan, "Toshkent vaqti", "UTC");
- %% - "%" belgisi.

Misol tariqasida strftime () funktsiyasidan foydalangan holda joriy sana va vaqtni namoyish qilaylik (10.2-misol).

10.2-misol. Sana va vaqtni formatlash

```
import time
import locale
locale.setlocale(locale.LC_ALL,"Uz")
s="Bugun: %A \n%d %b %Y %H:%M:%S\n%d.%m.%Y"
print(time.strftime(s))
input()
```

Dastur natijasi:

Bugun: seshanba

15 Sen 2020 20:40:30

15.09.2020

«Uxlab qolish» skripti

time modulidagi sleep(<soniyalardagi vaqt>) funktsiyasi belgilangan vaqt davomida skriptning bajarilishini to'xtatadi, shundan so'ng skript ishlashni davom ettiradi. Parametr sifatida butun sonni yoki haqiqiy sonni belgilashingiz mumkin. Misol:

```
>>>  import  time          # time modulini ulaymiz
>>>  time. sleep (5)       # 5 sekundga "Uxlaymiz"
```

datetime moduli. Sana va vaqt monipulyatsiyasi

datetime moduli sana va vaqtni boshqarishga imkon beradi. Masalan, arifmetik amallarni bajarish, sanalarni taqqoslash, sana va vaqtni har xil formatlarda namoyish

etish va hokazo. Ushbu moduldagi sinflarni ishlatishdan oldin ushbu modulni quyidagicha ulash kerak:

import datetime

Modul beshta sinfdan iborat:

- timedelta – kunlar, soniyalar va mikrosaniyalar ko'rinishidagi sana. Ushbu sinfning nusxasini date va datetime sinflari nusxalari bilan biriktirilishi mumkin. Bundan tashqari, ikkita sanani ayirish natijasi timedelta sinfining nusxasi bo'ladi;
- date – sanani ob'ekt sifatida aks ettirish;
- time – vaqtni ob'ekt sifatida aks ettirish;
- datetime – sana va vaqt kombinatsiyasini ob'ekt sifatida aks ettirish;
- tzinfo – vaqt zonasi uchun javobgar mavhum sinf. Ushbu sinf haqida qo'shimcha ma'lumot olish uchun datetime moduli ma’lumotnomasiga qarang.

Timedelta sinfi

datetime modulidagi **timedelta** sinfi sanalar bo'yicha amallarni bajarishga imkon beradi. Masalan: qo'shish, ayirish, taqqoslash vahokazo. Sinf konstruktori quyidagi formatga ega:

timedelta ([days] [, seconds] [, microseconds] [, milliseconds] [, minutes][, hours] [, weeks])

Barcha parametrlar ixtiyoriy va odatiy holda 0 ga teng. Dastlabki uchta parametr asosiy hisoblanadi:

- days – kunlar (-999999999 <= days <= 999999999 orlaig’ida) ;
- seconds – sekundlar(0 <= seconds < 3600 *24 orlaig’ida);
- microseconds – mikrosekundlar (0 <= microseconds < 10 00000 oralig’ida).

Boshqa barcha parametrlar avtomatik ravishda quyidagi qiymatlarga o'tkaziladi:

- milliseconds - millisekund (bir millisekund 1000 mikrosekundga aylantiriladi):

>>> import datetime

```
>>> datetime.timedelta (milliseconds=l )
datetime.timedelta(microseconds=1000)
```

- minutes - minutlar (bir minut 60 sekundga aylantiriladi):

```
>>>  datetime.timedelta (minutes=l)
datetime.timedelta(seconds=60)
```

- hours -soatlar (bir soat 3600 sekundga aylantiriladi):

```
>>>  datetime.timedelta (hours=l )
datetime.timedelta(seconds=3600)
```

- weeks – haftalar (bir hafta 7 kunga aylantiriladi):

```
>>>  datetime.timedelta(weeks=1)
datetime.timedelta(days=7)
```

Parametr qiymatlari o’z o’rniga muvofiq tartibida vergul bilan ajratilgan holda ko'rsatilishi yoki parametr nomiga qiymat berilishi mumkin. Misol tariqasida bir soatni ishlataylik:

```
>>> datetime.timedelta(0,0,0,0,0,1)
datetime.timedelta(seconds=3600)
>>> datetime.timedelta(hours=1)
datetime.timedelta(seconds=3600)
```

Natijani quyidagi atributlardan foydalanib olishingiz mumkin:

- days – kunlar;
- seconds – sekundlar;
- microseconds – mikrosekundlar.

Misol:

```
>>> d=datetime.timedelta(hours=1, days=2,  milliseconds=1)
>>> d
datetime.timedelta(days=2, seconds=3600, microseconds=1000)
>>> d.days,  d.seconds, d.microseconds
(2, 3600, 1000)
>>> repr(d), str(d)
```

('datetime.timedelta(days=2, seconds=3600, microseconds=1000)', '2 days, 1:00:00.001000')

total _seconds () metodi natijalarni soniyalarda olish imkonini beradi:

```
>>> d=datetime.timedelta(minutes=1)
>>> d.total_seconds()
60.0
```

timedelta sinfining obyektlarida arifmetik amallarni +, -, / , !, /, %, *, unar +, unar - operatorlaridan foydalanish va abs () funksiyasi yordamida absolyut qiymatni olish mumkin. Misollar:

```
>>> dl = datetime.timedelta(days=2)
>>> d2 = datetime.timedelta(days=7)
>>> dl+d2, d2-dl                    # Qo'shish, ayirish
(datetime.timedelta(days=9), datetime.timedelta(days=5))
>>> d2/dl                           # Bo'lish
3.5
>>>  d1/ 2,  d2 / 2.5               # Bo'lish
(datetime.timedelta (1), datetime.timedelta (2, 69120 ))
>>>  d2  //  dl                    # Bo'lish
3
>>>  dl // 2,  d2 // 2              # Bo'lish
(datetime.timedelta (1) ,  datetime.timedelta (3, 43200) )
>>>  d2  %  d1                      #  Qoldiq olish
datetim e.t imedelta (l)
>>>  dl  *  2,  d2  *  2            #  Ko'paytirish
(datet ime .t imede lta (4) ,  datet ime .t imedelta (l 4) )
>>>  2  *  dl ,  2  *  d2           #  Ko'paytirish
(datetime.timedelta (4) ,  datetime.timedelta (l4))
>>>  d3  =  - d1
>>>  d3 ,  abs (d3)
(datetime.timedelta (-2),  datetime.timedelta (2))
```

Bundan tashqari == ,! =, <, <=,>, va > = taqqoslash operatorlaridan foydalanish mumkin. Misol:

```
>>> dl = datet ime .t imedelta (days=2 )
>>> d2 = datet ime .t imedelta (d ays=7 )
>>> d3 = datet ime .t imede lta (w eeks=l )
>>> dl == d2 , d2 == d3 # Tenglikka tekshirish
(False, True )
>>> dl! = d2, d2 != d3 # Teng emaslikka tekshirish
(True, False )
>>> dl < d2 , d2 <= d3 # Kichik, kichik yoki teng
(True , True )
>>> dl > d2, d2 >= d3 # Katta, katta yoki teng
(False , True )
```

timedelta ob'ektining str () va repr () funktsiyalari yordamida satrli ko’rinishini olish mumkin ko'rinishini ham olishingiz mumkin:

```
>>> d = datetime.timedelta (hours = 25, minutes = 5, seconds = 27 )
>>> str (d)
'1 day, 1:0 5:2 7'
>>> repr (d)
'datetime.timedelta(days=1, seconds=3927)'
```

Quyidagi sinf xususiyatlari qo'llab-quvvatlanadi:

- min - timedelta ob'ekti bo'lishi mumkin bo'lgan minimal qiymat;
- max - timedelta ob'ekti bo'lishi mumkin bo'lgan maksimal qiymat;
- rezolyutsiyasi - timedelta qiymatlari orasidagi mumkin bo'lgan minimal farq.

Keling, ushbu xususiyatlarning qiymatlarini chop etamiz:

```
>>> datetime.timedelta.min
datetime.timedelta ( -99999 9999)
>>> datetime.timedelta.max
datetime.timedelta ( 99999 9999 , 86399, 999999)
```

```
>>>  datetime.timedelta.resolution
datetime.timedelta (0,  0,  1)
```

date sinfi

datetime modulidagi date sinfi sanalar ustida turli amallarni bajarishga imkon beradi. Sinf konstruktori quyidagi formatga ega:

date (<yil>, <oy>, <kun>)

Barcha parametrlar majburiy talab qilinadi. Parametrlarda quyidagi qiymatlar oralig'i ko'rsatilishi mumkin:

- <Yil> - datetime sinfining MINYEAR va MAHUEAR konstantalarida saqlanadigan qiymatlar oralig'ida joylashgan raqam sifatida (bu haqda keyinroq gaplashamiz). Ushbu doimiylarning qiymatlari:

```
>>>  import  datetime
>>> datetime.MINYEAR,  datetime.MAXYEAR
(1, 9999)
```

- <Oy> –1 dan 12 gacha;
- <Kun> – 1 dan oydagio kunlar sonigacha.

Agar qiymatlar doiradan tashqarida bo'lsa, valueError istisnosi qaytariladi. Misol:

```
>>> datetime.date(2015, 4, 3)
datetime.date(2015, 4, 3)
>>> datetime.date(2015, 14, 3)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    datetime.date(2015, 14, 3)
ValueError: month must be in 1..12
>>>  d  =  datetime.date (2015,  4,  3)
>>>  repr (d) ,  str (d)
(' datetime.date (2015,  4,  3)' ,  '2015-04-03 ')
```

Sinf ob'ektini yaratish uchun siz ushbu sinfning quyidagi usullaridan foydalanishingiz mumkin:

- today () – joriy sanani qaytaradi:

```
>>> datetime.date. today () # joriy sanani olamiz
datetime.date (2 015, 4, 3)
```

- fromtimestamp (< Количество секунд>) - era boshlangandan beri o’tgan sekundlarga mos sananai qaytaradi:

```
>>> import datetime, time
>>> datetime.date. fromtimestamp(time.time()) # Joriy sana
datetime.date(2020, 9, 24)
>>> datetime.date.fromtimestamp(1321954972) # Sana 22-11-2011
datetime.date(2011, 11, 22)
```

- fromordinal (< 1-yildan boshlab yil soni>) - birinchi yildan beri o'tgan kunlar soniga mos keladigan sanani qaytaradi. Parametr sifatida 1 dan datetime.date.max.toordinal() gacha raqam ko'rsatiladi. Misollar:

```
>>> datetime.date.max.toordinal()
3652059
>>> datetime.date.fromordinal (3652059)
datetime.date(9999, 12, 31)
>>> datetime.date.fromordinal(1)
datetime.date(1, 1, 1)
```

Natijani quyidagi atributlardan foydalanib olishingiz mumkin:

- year - yil (MINYEAR dan MAXYEARgacha soat);
- month- oy (1 dan 12 gacha raqam);
- day - kun (1 dan bir oydagi kunlar sonigacha raqam)

Misol:

```
>>> d = datetime.date.today () # Joriy sana (24 -09-2020)
>>> d.year, d.month, d.day
(2020, 9, 24)
```

Sana sinfi misollarida quyidagi operatsiyalarni bajarish mumkin:

- date2 = date1 + timedelta - sanaga kunlar bilan belgilangan muddatni qo'shadi. timedelta.seconds va timedelta.microseconds atributlari uchun qiymatlar e'tiborga olinmaydi;
- date2 = date1 - timedelta - sanadan kunlar bilan belgilangan muddatni olib tashlaydi. timedelta.seconds va timedelta.microseconds atributlari uchun qiymatlar e'tiborga olinmaydi;
- timedelta = datel - date2 - sanalar orasidagi farqni qaytaradi (davr kunlarda ifodalanadi). timedelta.seconds va timedelta.microseconds qiymatlari 0 ga teng bo'ladi;
- shuningdek, taqqoslash operatorlari yordamida ikkita sanni taqqoslash mumkin.

Misollar:

```
>>> dl = datetime.date (2015, 4,  3)
>>> d2 = datetime.date (2015, 1, 1)
>>> t = datetime.timedelta (days = 10 )
>>> d1+ t, d1 - t                          # 10 kunni qo’shamiz va ayriymiz
(datetime.date (2015,  4,  13) ,  datetime.date (2015,  3,  24) )
>>> dl - d2                          #  Sanalar orasidagi farq
datetime.timedelta (92)
>>>  dl  <  d2 ,  dl  >  d2 ,  dl  <=  d2 ,  dl  >=  d2
(False,  True,  False,  True )
>>>  dl  = =  d2,  dl  !=  d2
(False,  True )
```

Sana sinfining namunalari quyidagi usullarni qo'llab-quvvatlaydi:

- replace ([yil] [, oy] [, kun]) - sanani ko’rsatilgan qiymatlar bilan yangilash. Qiymatlar ko'rsatilishi parametrlar tartibida vergul bilan ajratilgan holda yoki parametr nomiga qiymat berilish orqali amalga oshirilishi mumkin. Misollar:

```
>>>  d  =  datetime.date (2015,  4,  3)
>>>  d. replace(2014, 12)                    # Yil va oyni yangilaymiz
```

datetime.date (2014, 12, 3)

>>> d. replace (year=2015, month=l, day=31)

datetime.date (2015, 1, 31)

>>> d. replace(day=30) # Faqat sanani yangilaymiz

datetime.date(2015, 1, 30)

- strftime (<Format string>) - Formatlangan qatorni qaytaradi. Format satrida vaqt modulidan strftime () funktsiyasida ishlatiladigan maxsus belgilar birikmalarini belgilashingiz mumkin. Misol:

>>> d = datet ime .date (2 01 5, 4, 3)

>>> d. strft ime (" %d. %m.%Y")

'0 3.0 4.2 015 '

- isoformat () - yyyy-oo-kk formatidagi sanani qaytaradi:

>>> d = datetime.date (2020, 4, 3)

>>> d.isoformat ()

'2020-04- 03'

- ctime() - " %a %b % d %H: %M : % S %Y" qator formatini qaytaradi:

>>> d = datetime.date (2015, 4, 3)

>>> d.ctime ( )

'Wed Sep 9 00:00:00 2020'

- timetuple ( ) - sana va vaqt li struct_time obyektini qaytaradi:

>>> d = datetime.date (2015, 4 , 3)

>>> d.timetuple( )

time.struct_time(tm_year=2015, tm_mon=4 , tm _ mday=3, tm_hour=0 , tm _min=0,

tm_sec=0, tm _ wday=4 , tm _yday=93 , tm_i sdst=- 1)

- toordinal ( ) – 1-yildan boshlab o’tgan kunlar sosnini qaytaradi:

>>> d = datetime.date(2015 , 4 , 3)

>>> d.toordinal ( )

735691

>>> datetime.date.fromordinal(735691)

datetime.date(2015 ,4 , 3)

- weekday ( ) – haftaning kun tartib tartibini qaytaradi (dushanba uchun 0, yakshanba uchun 6):

```
>>> d = datetime.date (2015, 4, 3)
>>>  d.weekday( )     #  4 – bu juma
4
```

- isoweekday ( ) – haftaning kun tartib tartibini qaytaradi (dushanba uchun 1, yakshanba uchun 7):

```
>>>  d = datetime.date (2015, 4, 3)
>>> d.isoweekday ( )              # 5 – bu juma
5
```

- isocalendar ( ) – uchta elementli kortej (yil, hafta va kun tartibini) qaytaradi:

```
>>> d = datetime.date (2015,  4, 3)
>>> d.isocalendar( )
(2015, 14, 5)
```

Va nihoyat, quyidagi sinf xususiyatlarini qo'llab-quvvatlaydi:

- min – mumkin bo'lgan minimal sana qiymati;
- max – mumkin bo'lgan maksimal sana qiymati;
- rezolyutsiya – bu sana qiymatlari orasidagi eng kichik farq.

Ushbu xususiyat qiymatlarini choop etamiz:

```
>>>  datetime.date.min
datetime.date (l, 1, 1)
>>>  datetime.date.max
datetime.date (9999, 12,  31)
>>>  datetime.date.resolution
datetime.timedelta (l)
```

Calendar moduli. Calendarni ko’rsatish

Calendar moduli oddiy matn yoki HTML kod shaklida taqvim (kalendar) hosil qiladi. Modulni ishlatishdan oldin uni quyidagicha ulash kerak:

import calendar

Modul quyidagi sinflarni taqdim etadi:

- Calendar – bu boshqa barcha sinflar meros qilib olgan asosiy sinf. Konstruktor formati:

Calendar ([<Haftaning birinchi kuni>])

Misol tariqasida, keling, haftaning kunlari bo'yicha taqsimlangan 2015 yil aprel oyidagi barcha kunlarning ikki o'lchovli ro'yxatini olaylik:

```
>>>  import  calendar
>>> c =  calendar.Calendar (0)
>>> c.monthdayscalendar(2020,10)                    # 10 – bu oktyabr oyi
```

[[0, 0, 0, 1, 2, 3, 4], [5, 6, 7, 8, 9, 10, 11], [12, 13, 14, 15, 16, 17, 18], [19, 20, 21, 22, 23, 24, 25], [26, 27, 28, 29, 30, 31, 0]]

- textcalendar - taqvimni oddiy matnda aks ettirishga imkon beradi. Konstruktor formati:

TextCalendar ([<Haftaning birinchi kuni>])

Keling, butun 2020 yil uchun taqvimni namoyish qilaylik:

```
>>> c = calendar.TextCalendar(0)
>>> print(c.formatyear(2020))
```

Natija:

```
                                  2020

      January                   February                   March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                         1  2                        1
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       2  3  4  5  6  7  8
13 14 15 16 17 18 19      10 11 12 13 14 15 16       9 10 11 12 13 14 15
20 21 22 23 24 25 26      17 18 19 20 21 22 23      16 17 18 19 20 21 22
27 28 29 30 31            24 25 26 27 28 29         23 24 25 26 27 28 29
                                                    30 31

      April                     May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                   1  2  3       1  2  3  4  5  6  7
 6  7  8  9 10 11 12       4  5  6  7  8  9 10       8  9 10 11 12 13 14
13 14 15 16 17 18 19      11 12 13 14 15 16 17      15 16 17 18 19 20 21
20 21 22 23 24 25 26      18 19 20 21 22 23 24      22 23 24 25 26 27 28
27 28 29 30               25 26 27 28 29 30 31      29 30
```

```
                July                August              September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                         1  2                1  2  3  4  5  6
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       7  8  9 10 11 12 13
13 14 15 16 17 18 19      10 11 12 13 14 15 16      14 15 16 17 18 19 20
20 21 22 23 24 25 26      17 18 19 20 21 22 23      21 22 23 24 25 26 27
27 28 29 30 31            24 25 26 27 28 29 30      28 29 30
                          31

               October             November             December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
          1  2  3  4                            1                1  2  3  4  5  6
 5  6  7  8  9 10 11       2  3  4  5  6  7  8       7  8  9 10 11 12 13
12 13 14 15 16 17 18       9 10 11 12 13 14 15      14 15 16 17 18 19 20
19 20 21 22 23 24 25      16 17 18 19 20 21 22      21 22 23 24 25 26 27
26 27 28 29 30 31         23 24 25 26 27 28 29      28 29 30 31
                          30
```

- LocaleTextcalendar - taqvimni oddiy matnda aks ettirishga imkon beradi.

Haftaning oylari va kunlari belgilangan joyga qarab ko'rsatiladi. Konstruktor formati:

LocaleTextCalendar ([<Haftaning birinchi kuni> [, <Mahalliy nomi>]])

Keling, butun yil 2020 yil taqvimini o’zbek tilida namoyish qilaylik:

```
>>> c=calendar.LocaleTextCalendar(0,  "Uz")
>>> print(c.formatyear(2020))
```

```
                                2020

                Yanvar                Fevral                 Mart
Du Se Ch Pa Ju Sh Ya      Du Se Ch Pa Ju Sh Ya      Du Se Ch Pa Ju Sh Ya
       1  2  3  4  5                         1  2                         1
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       2  3  4  5  6  7  8
13 14 15 16 17 18 19      10 11 12 13 14 15 16       9 10 11 12 13 14 15
20 21 22 23 24 25 26      17 18 19 20 21 22 23      16 17 18 19 20 21 22
27 28 29 30 31            24 25 26 27 28 29         23 24 25 26 27 28 29
                                                    30 31

                Aprel                  May                  Iyun
Du Se Ch Pa Ju Sh Ya      Du Se Ch Pa Ju Sh Ya      Du Se Ch Pa Ju Sh Ya
       1  2  3  4  5                      1  2  3       1  2  3  4  5  6  7
 6  7  8  9 10 11 12       4  5  6  7  8  9 10       8  9 10 11 12 13 14
13 14 15 16 17 18 19      11 12 13 14 15 16 17      15 16 17 18 19 20 21
20 21 22 23 24 25 26      18 19 20 21 22 23 24      22 23 24 25 26 27 28
27 28 29 30               25 26 27 28 29 30 31      29 30
```

```
        Iyul
Du Se Ch Pa Ju Sh Ya
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
      Avgust
Du Se Ch Pa Ju Sh Ya
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

```
      Sentabr
Du Se Ch Pa Ju Sh Ya
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

```
       Oktabr
Du Se Ch Pa Ju Sh Ya
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

```
      Noyabr
Du Se Ch Pa Ju Sh Ya
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

```
      Dekabr
Du Se Ch Pa Ju Sh Ya
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

Muhokama uchun savollar va topshiriqlar

1. Pythonda sana va vaqt bilan ishlash.
2. Pythonda joriy sana va vaqtni chop etish.
3. Pythonda sana va vaqt formati.
4. Pythonda «Uxlash» skripti
5. Pythonda datetime moduli.
6. Pythonda sana va vaqt monipulyatsiyasi.
7. Pythonda calendar moduli.
8. Pythonda calendarni ko’rsatish

12-BOB. FOYDALANUVCHI FUNKSIYALARI

O`quv modullari

Funksiya, anonim funksiyalar, funksiya-generatorlar, dekorator, rekursiya, faktorial, global va local o’zgaruvchilar, ilova funksiyalar, funksiya annotasiyasi, parametrsiz funksiya, paramertli funksiya.

Funksiya - bu dasturning istalgan joyidan chaqirib ishlatish mumkin bo’lgan, biror muayyan vazifani bajaradigan qism dasturidir. Funksiyalar dasturni ixchamlashtiradi va ishlash tezligini oshirishga hizmat qiladi. Oldingi boblarda biz Python tilining ichki yaratilgan funktsiyalaridan bir necha bor foydalanganmiz - masalan, len() funktsiyasidan foydalanib, biz ketma-ketlik elementlari sonini aniqladik.

Ushbu bobda dasturning maxsus funksiyalarni yaratishni ko’rib chiqamiz.

12.1. Funksiyai aniqlanishi va uni chaqirish

Funktsiya **def** kalit so’zidan foydalanib quyidagi ko’rsatma bo’yicha yaratiladi (yoki dasturchilar tili bilan aytganidek, aniqlanadi):

def <Funktsiya nomi> ([<Parametrlar>]):

[“” “Hujjat qatori” “”]

<Funktsiya tanasi>

[return <Natija>]

Funksiya nomi lotin harflari, raqamlar va pastki chiziqlardan tashkil topgan, raqam bilan boshlanmaydigan, takrorlanmas identifikator bo’lishi kerak. Kalit so’zlar va o’rnatilgan identifikatorlarning nomlari funksiya nomi sifatida ishlatilmasligi kerak. Funksiya nomidagi belgilarning registri ham muhimdir.

Funktsiya nomidan keyin qavslar ichida vergul bilan ajratilgan bir yoki bir nechta parametrlarni belgilash mumkin, agar funksiya parametrlarni qabul qilmasa, faqat qavslar ko'rsatiladi. Qavslardan keyin ikki nuqta qo‘yiladi.

Funktsiya tanasida asosiy amallar ko’rsatiladi. Har qanday birikma dastur konstruktsiyasida bo'lgani kabi, funktsiya ichidagi qatorlarda ham chap tomonda bir

xil sonli bo'shliqlar bilan ajralib turadi. Funktsiyaning oxiri oldinroq kamroq bo'shliqlar qo'yilgan gap hisoblanadi. Agar funktsiya tanasida ko'rsatmalar mavjud bo'lmasa, unda uning ichiga biron bir amalni bajarmaydigan pass bayonoti joylashtirilishi kerak.

Ushbu operator dasturni nosozliklarini tuzatish bosqichida, funktsiyani aniqlab, tanani keyinroq qo'shishga qaror qilganimizda foydalanish uchun qulaydir. Hech narsa bajarmaydigan funksiyaga misol:

```
def func() :
pass
```

Majburiy bo’lmagan return operatori funktsiyadan qiymatni funksiya tashqarisiga qaytarishga imkon beradi. Ushbu ko'rsatmani bajargandan so'ng, funktsiyani bajarish to'xtatiladi. Bu shuni anglatadiki, return operatoridan keyingi operatorilar hech qachon bajarilmaydi. Misol:

```
def  func() :
print ("return operatorigacha bo’lgan matn" )
return "Qaytariluvchi qiymat"
print ("Bu operatorlar hech qachon bajarilmaydi" )
print (func ())  #  Funksiyani chaqiramiz
```

Dastru natiajasi quyidagicha:

```
return operatorigacha bo’lgan matn
Qaytariluvchi qiymat
```

return operatori funksiya ichida umuman bo'lmasligi mumkin. Bunday holda, funksiya ichidagi barcha ko'rsatmalar bajariladi va natijada None qaytariladi.

Masalan, uchta funksiyani yarataylik (11.1-misol)

12.1-misol. Funksiya aniqlanishi

```
def print_ok () :
"""Parametrsiz funksiyaga misol"""
print ("Amalni muvaffaqiyatli bajarilishi haqida xabar")
def echo (m) :
"""Paramertli funksiyaga misol"""
```

```
print (m)
def summa (x,  y) :
""" Ikki son yig’indisini hisoblovchi
parametrli funklsiayaga doir misol """
return x  +  y
```

Funksiyani chaqirishda qiymatlar vergul bilan ajratilgan holda qavs ichida beriladi. Agar funksiya parametrlarni qabul qilmasa, faqat qavslar ko'rsatiladi. Shuni ham ta'kidlash kerakki, funksiya ta'rifidagi parametrlarning soni chaqirilganda parametrlar soniga mos kelishi kerak, aks holda xato xabari ko'rsatiladi. 11.1-misolda ko'rsatilgan funktsiyalardan 11.2-misoldagi ko’rinishda chaqirib foydalanish mumkin.

12.2-misol. Funksiyani chaqirish

```
print_ok ()          # Parametrsiz funksiyani chaqirish
echo ("Habar")             # Funksiya habar matnini chop etadi
x = summa (5,  2)          #  x o’zgaruvchisi 7 qiymatini o’zlashtiradi
a, b = 10, 50
y= summa(a,  b)    #  y o’zgaruvchisi 60 qiymatini o’zlashtiradi
```

Oxirgi misoldan ko'rinib turibdiki, funksiya chaqiruvidagi o'zgaruvchi nomi funksiya ta'rifidagi o'zgaruvchiga mos kelmasligi mumkin. Bundan tashqari, global o'zgaruvchilar x va y funksiya ta'rifida bir xil nomdagi o'zgaruvchilarga zid kelmaydi, chunki ular har xil ko’rinish sohasida joylashgan. Funktsiya ta'rifida ko'rsatilgan o'zgaruvchilar lokal va faqat funktsiya ichida amal qilinadi. Biz keyiongi bo'limda buni batafsilroq ko'rib chiqamiz.

summa() funktsiyasida ishlatiladigan + operatori nafaqat sonlarni qo'shish uchun, balki ketma-ketlikni birlashtirish uchun ham ishlatiladi. Shunday bo’lsada, summa() funksiyasidan faqat sonlarni qo'shishda ko'proq foydalanish mumkin. Misol tariqasida satrlarni ulash va ro'yxat birlashtirishni ko’rib chiqamiz:

```
def  summ a (x,  y ) :
return  x  +  y
print (summa("ona",  "jon") )          # Natija:  onajon
```

```
print (summa ([1, 2],  [3, 4]))          #  Natija:   [1,  2,  3,  4  ]
```

Siz allaqachon bilganingizdek, Pythonda hamma narsa: satrlar, ro'yxatlar va hatto ma'lumotlar turlari ham ob'ektlar sifatida tasvirlanadi. Funktsiyalar ham bundan istisno emas. Def bayonoti function turidagi ob'ektni yaratadi va unga havolani def bayonotidan keyin ko'rsatilgan identifikatorda saqlaydi. Shunday qilib, biz funktsiyaga havolani boshqa o'zgaruvchida saqlashimiz mumkin - buning uchun funktsiya nomi qavssiz ko'rsatiladi. Ma'lumotni o'zgaruvchiga saqlaylik va u orqali funktsiyani chaqiramiz (11.3-misol).

12.3-misol. Funktsiya ma'lumotlarini o'zgaruvchida saqlash

```
def  summa (x,  y)  :
return  x  +  y
f  =  summa #  f o’zgaruvchida havolani saqlaymiz
v  =  f (10,  20)      #  f o’zgaruvchi orqali funksiyani chaqiramiz
```

Bundan tashqari, funksiya nomini boshqa funksiyada parametr sifatida parametrlar ro’yxatida berish ham mumkin. Havola orqali uzatiladigan funktsiyalar odatda qayta chaqiriluvchi funktsiyalari deb ataladi (11.4-misol).

12.4-misol. Qayta chaqiriluvchi funksiya

```
def  summ a (x,  y) :
return  x  +  y
def  func (f ,  a,  b) :
"""f o’zgaruvchisi orqali
summa( )  funksiyasiga havola """
return f(a,  b)                  #  summa() funksiyasin ichaqiramiz
#  funksiyaga parameter sifatiada havola beramiz
v  =  func(summa,  10,  20 )
```

12.2. Funktsiya ta'riflarining joylashishi

Dasturdagi barcha ko'rsatmalar yuqoridan pastgacha ketma-ket bajariladi. Bu shuni anglatadiki, dasturda identifikatorni ishlatishdan oldin, avval unga qiymat berish

orqali aniqlanishi kerak. Shuning uchun funktsiya ta'rifi funktsiya chaqirig'idan oldin joylashtirilishi kerak.

To'g'ri:

```
def  s uппп a( x,  y ):
return  x  +  y
v = summa(10,  20 )  #  Aniqlanishdan keyin chaqirilyapdi.  Hammasi joyida
```

Noto’g;ri:

```
v  =  summa(10,  20)  #  Identifikator hali aniqlanmagan.  Bu xato!!!
def  summa (x,  y )  :
return  x  +  y
```

Muhokama uchun savollar va topshiriqlar

1. Funksiya yaratishdan ko’zlangan asosiy maqsad nima?
2. Pythonda funksiyaning aniqlanishi qanday amalga oshiriladi?
3. Funktsiya ma'lumotlarini o'zgaruvchida saqlash qanday amalga oshiriladi?
4. Pythonda tiplarni o’zgartiruvchi standart funksiyalarga misollar keltiring?
5. Funksiyaga qiymat uzatish qanday amalga oshiriladi?
6. Funksiyaga murojaat qilishda qanday xatoliklar yuzaga kelishi mumkin?
7. Foydalanuvchi ismi va yoshini so'rab, uning tug'ilgan yilini hisoblaydigan funksiya yozing?
8. Foydalanuvchidan son olib, son juft yoki toqligini konsolga chiqaruvchi funksiya yozing?
9. Foydalanuvchidan son qabul qilib, sonni 2 dan 10 gacha bo'lgan sonlarga qoldiqsiz bo'linishini tekshiruvchi funksiya yozing. Natijalarni konsolga chiqaring?
10. Foydalanuvchidan son olib, uning kvadrati va kubini konsolga chiqaruvchi funksiya yozing?

13-BOB. FAYL VA KATALOGLAR BILAN ISHLASH

O`quv modullari

Faylni ochish, buffering, Os moduli, Fayl tarkibini o’qish, StringIO va BytesIO sinfi, fayl va kataloglarga kirish huquqi, faylarni monipulyasiyalash funksiyasi, fayl yoki kataloglarga yo’lni o’zgartirish, kiritish/chiqarishni qayta yo'naltirish

Biz tez-tez ma'lumotlarni xotirada saqlashimizga to’g’ri keladi. Buning ikkita usuli bor: faylga yozish va ma'lumotlar bazasiga saqlash. Birinchi usul oz miqdordagi ma'lumotni saqlashda qo'llaniladi. Agar ma’lumot hajmi katta bo'lsa, ma'lumotlar bazasidan foydalanish yaxshiroq va qulayroq.

10.1. 13.1. Fayllarni ochish.

Fayl ustida ishlashdan oldin open() funktsiyasidan foydalangan holda fayl ob'ekti yaratishingiz kerak. Funktsiya quyidagi formatga ega:

open (<Faylga yo’l> [, mode= ' r '] [, buffering=- 1] [, encoding=None] [,

errors=None] [, newline=None] [, closefd=True])

Birinchi parametr faylga yo'lni belgilaydi. Yo'l absolyut yoki nisbiy bo'lishi mumkin.

Windows OTda absolyut yo'lni belgilashda, Pythonda chiziq(slesh)ning maxsus belgi ekanligini unutmang. Shu sababli, yotiq chiziq(slesh) ikki baravarga ko'paytirilishi yoki oddiy satrlar o'rniga formatlanmagan satrlardan foydalanish kerak. Misol:

```
>>> "C:\\temp\\new\\file.txt"          #To’g’ri
'C:\\temp\\new\\file.txt'
>>> r"C:\temp\new\file.txt"            #To’g’ri
'C:\\temp\\new\\file.txt'
>>> "C : \temp\new\file.txt"           #Noto’g’ri
'C: \temp\new\xOcile.txt'
```

Oxirgi misolga e’tibor qarating. Shu tarzda, chiziqlar ikki baravar ko'paymaganligi sababli, uchta maxsus belgining mavjudligi darhol paydo bo'ldi: \ t, \ n va \ f (\ xOc sifatida ko'rsatiladi). Ushbu maxsus belgilarni o'zgartirgandan so'ng, yo'l quyidagicha ko'rinadi:

C : <Tab>emp<Satrga o’tkazish>ew<Formatga o’tkazish>ile.txt

Bunday satrni open () funktsiyasiga o'tkazishda OSError istisnoga olib keladi:

>>> open ("C : \temp\new\file . txt")

Traceback (most recent call last) :

File "<pyshell#O> " , line 1 , in <module>

open ("C : \temp\new\file . txt ")

OSError: [Errno 22] Invalid argument : ' C :\temp\new\xOcile . txt '

Absolyut fayl yo'li o'rniga nisbiy yo'lni belgilashingiz mumkin. Bunday holda, yo'l joriy ishlaydigan katalog joylashgan joyga qarab belgilanadi. Nisbiy yo'l os.path modulidagi abspath () funktsiyasi yordamida avtomatik ravishda absolyut yo'lga aylantiriladi. Natijalar quyidagicha bo’lishi mumkin:

- agar ochilayotgan fayl joriy ishchi katalogda bo'lsa, unda faqat fayl nomi ko'rsatilishi mumkin. Misol:
  >>> import os.path # Modulni bog’laymiz
  >>> # Joriy ish katalogidagi fayl (C : \book\ )
  >>> os.path . abspath (r"file.txt")
  'C: \\book\\file.txt'
- agar ochiladigan fayl ichki papkada joylashgan bo'lsa, fayl nomidan oldin chiziq bilan ajratilgan pastki papkalarning nomlari berilgan. Misollar:
  >>> # C:\book\folderl\ da ochiladigan fayl
  >>> os.path.abspath(r" folderl/file.txt")
  'C:\\ book\\ folderl\\file.txt'
  >>> # C:\book\folderl\folder2\ da ochiladigan fayl

>>> os.path.abspath(r" folderl/folder2/file.txt")

'C:\\book\\folderl\\folder2\\file.txt '

- agar faylga ega papka darajadan pastroq bo'lsa, unda fayl nomidan oldin ikkita nuqta va chiziq (slesh) ko'rsatiladi. (" .. /"). Misol:

  >>> # C:\ da ochiladigan fayl

  >>> os.path.abspath (r" .. /file.txt ")

  'C: \\file. txt

- agar yo'lning boshida yotiq chiziq (slesh) bo'lsa, u holda yo'l diskning ildizidan boshlab hisoblanadi. Bunday holda, joriy ishlaydigan katalogning joylashuvi muhim emas. Misollar:

  >>> # C:\book\ folderl\ da ochiladigan fayl

  >>> os.path.abspath (r" /book/folderl/file.txt")

  'C:\\ book\\folderl\\file.txt '

  >>> # C:\book\folderl\folder2\ da ochiladigan fayl

  >>> os.path.abspath (r" /book/folderl/folder2 /file.txt")

  'C: \\ book\\ folderl\\folder2\\file.txt '

Ko'rib turganimizdek, oldinga va orqaga yotiq chiziqlar (sleshlar) absolyut va nisbiy yo'llarda belgilanishi mumkin. Ularning barchasi os.path modulidagi sep atributi qiymati asosida avtomatik ravishda o'zgartiriladi. Ushbu atributning qiymati ishlatilayotgan operatsion tizimga bog'liq. Windows operatsion tizimida sep atributining qiymatini ko'rsatib o’tamiz:

>>> os.path.sep

'\\'

>>> os.path.abspath(r"C: /book/folderl /file.txt ")

'C: \\book\\folderl\\file.txt '

Nisbiy yo'ldan foydalanganda, amaldagi ishchi katalog joylashgan joydan xabardor bo'ling, chunki ishchi katalog har doim ham bajariladigan dastur joylashgan katalog bilan bir xil bo’lmaydi. Agar fayl uning belgisini ikki marta bosish orqali ishga

tushirilsa, u holda kataloglar mos keladi. Agar fayl buyruq satridan ishga tushirilsa, u holda ishchi katalog fayl ishga tushiriladigan katalog bo'ladi.

Keling, bularning barchasini misol bilan ko'rib chiqamiz, buning uchun C: \ book katalogida quyidagi fayl tuzilishini yaratamiz:

C: \book\

test .py

folderl\

__init__ . py

Module1.py

C:\book\test.py faylining tarkibi quyidagi ro'yxatda keltiriladi:

```
# -*- coding : utf-8 -*

import os , sys

print ( "%-25s%s" % ( "Fayl : ", os . path . abspath( file )))

print ( " %-25s%s" % ( "Joriy ishchi katalog : ", os . getcw d() ))

print ( " %-25s%s" % ( " Katalogni import qilish : ", sys . path [O ] ))

print ( " %-25s%s" % ( "Faylga yo’l: "; os . path . abspath ( "file.txt " )))

print ( "-" * 40)

import folderl . module1 as m

m. get_cwd ()
```

C: \ book \ folder1 \ __ init__ .py fayli bo'sh holatda yaratiladi. Siz avval bilganingizdek, ushbu fayl Python tarjimoniga ushbu katalog modullar to'plami ekanligini aytadi.

C:\book\folder1\ module1 .py faylining tarkibi quyidagi ro'yxatda keltiriladi:

```
# -*- coding : utf-8
```

```
import os , sys

def get_cwd ( ) :

    print ( "%-25s%s" % ( "Fayl : " , os.path.abspath (__file__)))

    print ( "%-25s%s" % (" Joriy ishchi katalog:" , os.getcwd ()))

    print ( "%-25s%s" % ("Katalogni import qilish :" , sys.path [O] ))

    print ( "%-25s%s " % (" Faylga yo’l:", os.path.abspath ("file.txt ")))
```

Buyruqning satrini ishga tushiramiz va undan C:\book katalogiga o'tamiz va test.py faylini ishga tushiramiz:

```
C:\>cd C:\book

C:\book>test.py

Fayl:                    C:\book\test.py

Joriy ishchi katalog:    C:\book

Katalogni import qilish: C:\book

Faylga yo’l:             C:\book\file.txt

---------------------------------------

Fayl:                    C:\book\folder1\module1.py

Joriy ishchi katalog:    C:\book

Katalogni import qilish: C:\book

Faylga yo’l:             C:\book\file.txt
```

Ushbu misolda joriy ishchi katalog test.py fayli joylashgan katalog bilan bir xil. Biroq, module1.py moduli ichidagi joriy ishchi katalogga e'tibor qarating. Agar siz ushbu modul ichida open () funktsiyasida fayl nomini yo'lsiz ko'rsatadigan bo'lsangiz, u holda fayl C:\book\folder1 katalogidan emas, balki C:\book katalogidan qidiriladi.

Endi C: diskining ildiziga o'tamiz va yana test.py faylini ishga tushiramiz:

```
C:\book>cd C:\

C:\>C:\book\test.py

Fayl:                    C:\book\test.py

Joriy ishchi katalog:    C:\

Katalogni import qilish: C:\book

Faylga yo’l:             C:\file.txt

---------------------------------------

Fayl:                    C:\book\folder1\module1.py

Joriy ishchi katalog:    C:\

Katalogni import qilish: C:\book

Faylga yo’l:             C:\file.txt
```

Bunday holda, joriy ishchi katalog test.py fayli joylashgan katalog bilan bir xil emas. Agar open () funktsiyasida test.py va module1.py fayllari ichida siz fayl nomini yo'lsiz ko'rsatsangiz, u holda fayl ushbu fayllar katalogidan emas, balki C: diskining ildizidan qidiriladi. Ishga tushiriladigan fayl bilan katalogdagi faylni har doim qidirish uchun siz os modulidagi chdir () funktsiyasidan foydalanib ushbu katalogni joriy qilishingiz kerak. Masalan, test.py fayli tarkibini o'zgartiraylik:

```
# - * - coding : utf-8 -*

import os, sys

# Ishga tushiriladigan fayl bilan katalogni joriy qilish

os.chdir (os.path.dirnarne(os.path.abspath (file)))

print ( "% -25s%s" % ( "Fayl:", file))

print ( " %-25s%s" % ( "Joriy ishchi katalog:", os.getcwd() ))

print ( " %-25s%s" % ( "Katalogni import qilish: ", sys.path [ O ]))
```

print (" %-25s%s" % ("Faylga yo’l: ", os.path. abspath ("file.txt ")))

To'rtinchi qatorga e'tibor bering. _File_ atributidan foydalanib, fayl nomi bilan birga bajariladigan faylga yo'l olamiz. _File_ atributi har doim ham faylga to'liq yo'lni o'z ichiga olmaydi. Masalan, agar faylni ishga tushirish quyidagi tarzda amalga oshirilsa: C:\book>C : \Python37\python test.py, unda atributda yo'lsiz faqat fayl nomi bo'ladi.

Har doim faylga to'liq yo'l olish uchun atribut qiymatini os.path modulidan abspath () funktsiyasiga o'tkazishimiz kerak bo’ladi. Keyinchalik, dirname() funktsiyasi yordamida yo'lni (fayl nomisiz) chiqaramiz va uni chdir() funktsiyasiga o'tkazamiz. Endi open () funktsiyasida yo'lsiz fayl nomini ko'rsatsangiz, qidiruv ushbu fayl bilan katalogda amalga oshiriladi. Buyruqlar satri yordamida test.py faylini ishga tushiramiz:

C : \>C : \book\test. py

Fayl: C:\book\test.py

Joriy ishchi katalog C:\book

Katalogni import qilish: C:\book

Faylga yo’l: C:\book\file.txt

Kataloglar bilan ishlash funktsiyalarini keyingi bo'limlarda batafsil muhokama qilib chiqamiz. Hozircha esda tutish kerakki, bizni misollarda joriy katalog bajarilayotgan fayl joylashgan katalog emas, balki fayl ishga tushirilgan katalog bo'ladi. Bundan tashqari, fayllarni qidirish yo'llari va modul qidirish yo'llari bir biri bilan hech qanday bog’liq emas. Faylni ochishda Open() funktsiyasidagi ixtiyoriy rejim parametri quyidagi qiymatlarni qabul qilishi mumkin:

- r – faqat o’qish uchun ochish(standart xolda foydalaniladi). Faylni ochgandan so'ng, ko'rsatkich faylning boshiga o'rnatiladi. Agar fayl mavjud bo'lmasa, FileNotFoundError xatolik qaytaradi.
- r+ – o’qish va yozish uchun ochish. Faylni ochgandan so'ng, ko'rsatkich faylning boshiga o'rnatiladi. Agar fayl mavjud bo'lmasa, FileNotFoundError istisno qaytaradi.

- w – yozish uchun ochish. Agar fayl mavjud bo'lmasa, u yaratiladi. Agar fayl mavjud bo'lsa, uning ustiga yoziladi. Faylni ochgandan so'ng, ko'rsatkich faylning boshiga o'rnatiladi;
- w+ – yozish va o’qish uchun ochish. Agar fayl mavjud bo'lmasa, u yaratiladi. Agar fayl mavjud bo'lsa, uning ustiga yoziladi. Faylni ochgandan so'ng, ko'rsatkich faylning boshiga o'rnatiladi;
- a – yozish uchun ochish. Agar fayl mavjud bo'lmasa, u yaratiladi. Yozish faylning oxiridan amalga oshiriladi. Avvalgi faylning tarkibi o'chirilmaydi;
- a+ – yozish uchun fayl yaratish. o’qish va yozish uchun ochish. Agar fayl mavjud bo'lmasa, u yaratiladi. Yozish faylning oxiridan amalga oshiriladi. Avvalgi faylning tarkibi o'chirilmaydi;
- x – yozish uchun fayl yaratish. Agar fayl mavjud bo’lsa, FileExistsError xatolik qaytaradi;
- x+ – o’qish va yozish uchun fayl yaratish. Agar fayl mavjud bo’lsa, FileExistsError xatolik qaytaradi.

Eslatma:

Oxirgi ikkita rejimni qo'llab-quvvatlash Python 3.3 dan keyingi versiyalardan boshlab taqdim etilgan.

Rejimni ko'rsatgandan so'ng, modifikator quyidagilarni bajarishi mumkin:

- b – fayl ikkilik(binar) rejimda ochiladi. Fayl metodlari ***bayt*** tipidagi obyektlarni qabul qiladi va qaytaradi.
- t – fayl matn rejimida ochiladi (Windows OTda standart xolda shunday). Fayl metodlari ***str*** tipidagi ob'ektlarni qabul qiladi va qaytaradi. Ushbu rejimda satr oxiridagi belgini qayta ishlash avtomatik ravishda amalga oshiriladi - masalan, Windows OTda o'qish paytida \ r \ n belgilar \ n belgisi bilan almashtiriladi. Masalan, file.txt faylini yaratib, unga ikkita qator yozamiz:

```
>>> f = open ( r"file . txt", "w")        # Yozish uchun faylni ochamiz
>>> f.write( "Stringl\nString2")          # Faylga ikkita satr yozamiz
```

```
>>> f.close()                          # Faylni yopamiz
```

w rejimini belgilaganimiz uchun, agar fayl mavjud bo'lmasa, u yaratiladi va agar mavjud bo'lsa, uning ustiga yoziladi.

Endi faylning tarkibini ikkilik(binar) va matnli rejimlarda ko’rib chiqamiz:

```
>>> # Ikkilik(binar) rejimi (\ r belgisi qoladi)
>>> with open (r"file . txt", "rb") as f:
        for line in f:
                print( repr ( line ) )
b ' Stringl\r\n '
b ' String2 '
>>> # Matn rejimi ( \ r belgisi o’chiriladi)
>>> with open ( r"file . txt", "r") as f:
        for line in f:
                print (repr ( line ) )
' Stringl\n '
' String2 '
```

Ishni tezlashtirish uchun yozilgan ma'lumotlar buferlanadi. Buferdan olingan ma'lumotlar faylga faqat faylni yopish paytida yoki funktsiya yoki flush() metodi chaqirilgandan so'ng to'liq yoziladi. Bufer hajmining ixtiyoriy parametrlarini ***buffering***dan foydalanib belgilash mumkin. Agar 0 qiymat sifatida ko'rsatilgan bo'lsa, ma'lumotlar darhol faylga yoziladi (qiymat faqat ikkilik rejimda amal qiladi). Faylga satrma-satr yozishda 1 qiymati ishlatiladi (qiymat faqat matn rejimida amal qiladi), boshqa musbat raqam buferning taxminiy hajmini belgilaydi va manfiy qiymat (yoki qiymat yo'q) tizimning standart o'lchamini belgilaydi. Odatda, matnli fayllar satrma-satr buferlanadi va ikkilik fayllar qismlarga bo'linadi, ularning o'lchamlarini interpretator o'zi 4096 dan 8192 baytgacha diapozoni bo’yicha tanlaydi. Matn rejimidan foydalanganda (standart ravishda), faylni o'qiyotganda ma'lumotlarni Unicode kodlashiga o'tkazishga harakat qilinadi va yozishda qarama-qarshi operatsiya amalga oshiriladi - satr ma'lum bir kodlashda baytlar ketma-ketligiga aylantiriladi.

Odatda, tizimda ishlatiladigan kodlash standart belgilanadi. Agar o’tkazishning iloji bo'lmasa, xatolik qaytariladi. ***encoding*** parametri sizga faylni yozishda va o'qishda foydalaniladigan kodlarni belgilashga imkon beradi. Masalan, ma'lumotlarni UTF-8 da kodlashni ko’raylik:

```
>>> f = open(r"file.txt" , "w", encoding="utf -8 ")
>>> f.write( "Satr")                    # Satrni faylga yozamiz
6
>>> f.close ()                          # Faylni yopamiz.
```

Ushbu faylni o'qish uchun faylni ochishda kodlashni aniq belgilashingiz kerak:

```
>>> with open ( r"file . tx t ", "r", encoding="utf-8 ") as f:
        for line in f:
            print ( line)
Satr
```

UTF-8, UTF-16 va UTF-32 kodlashlaridagi fayllar bilan ishlashda, faylning boshida qisqacha BOM (Byte Order Mark – baytlar tartibi belgisi) deb nomlangan xizmat belgilari bo'lishi mumkin. Ushbu belgilar UTF-8 da kodlash uchun ixtiyoriydir va oldingi misolda ular yozish paytida faylga qo'shilmagan. Belgilar qo'shilishi uchun ***encoding*** parametrida utf-8-sig ni ko'rsatishingiz kerak. Qatorni BOM bilan UTF-8 kodlashdagi faylga yozamiz:

```
>>> f = open (r"file . txt " , "w" , encoding="utf-8-sig" )
>>> f.write( "Satr" )                   # Satrni faylga yozamiz
6
>>> f.close() # Faylni yopamiz.
```

Endi encoding parametridagi har xil qiymatdagi faylni o'qiylik:

```
>>> with open ( r"file . txt" , "r" , encoding="utf-8" ) as f :
```

```
    for line in f :
        print(repr(line) )
' \u feffSatr'
>>> with open (r"file . tx t", "r" , encoding="utf-8-sig") as f:
    for line in f:
        print(repr(line) )
‘Satr’
```

Birinchi misolda biz utf-8 qiymatini ko'rsatdik, shuning uchun ma'lumotlar bilan birga BOM markeri fayldan o'qildi. Ikkinchi misolda utf-8-sig qiymati ko'rsatilgan, shuning uchun BOM markeri natijaga kiritilmagan. Agar siz faylda marker mavjudligini bilmasangiz va siz markersiz ma'lumotlarni olishingiz kerak bo'lsa, u holda faylni UTF-8 kodlashda o'qiyotganda har doim utf-8-sig qiymatini belgilashingiz kerak. UTF-16 va UTF-32 kodlashlari uchun BOM markeri albatta kerak. ***encoding*** parametrida utf -16 va utf-32 qiymatlarini belgilashda belgi avtomatik ravishda qayta ishlanadi. Ma'lumotlarni yozishda marker avtomatik ravishda faylning boshiga kiritiladi va o'qiyotganda natijada ko'rinmaydi. Satrni faylga yozamiz, keyin uni fayldan o'qiymiz:

```
>>> with open ( r"file.txt ", "w" , encoding="utf-16" ) as f:
    f. write ( "Satr" )
6
>>> with open (r"file.txt", "r", encoding="utf-16") as f:
    for line in f:
        print (repr(line))
‘Satr’
```

Utf-16-le, Utf-16-be, Utf-32-le va Utf-32-be qiymatlaridan foydalanganda faylning boshiga BOM markerini qo'shish kerak va o'qiyotganda uni o'chirish kerak.

errors parametrida siz xatolar bilan ishlash darajasini belgilashingiz mumkin. Bu parametr qo’llashi mumkin bo'lgan qiymatlar:

- "strict" (xatolik ro’y berganda ValueError istisnosini qaytaradi - standart xolda)
- "replace" (noma'lum belgi so’roq belgisi bilan yoki \ufffd kodli belgi bilan almashtiriladi)
- " ignore" (noma'lum belgilar e'tiborga olinmaydi)
- "xmlcharrefreplace" (noma'lum belgi &#xxxx ketma-ketlik bilan almashtiriladi)
- "backslashreplace" (noma'lum belgi \uxxxx ketma-ketligi bilan almashtiriladi)

newline parametri satr oxiridagi belgilar uchun ishlov berish rejimini o'rnatadi. Bu parametr qo’llashi mumkin bo'lgan qiymatlar:

- None (standart xolda) – satr oxiridagi belgilar uchun standart ishlov berishni amalga oshiradi. Masalan, Windows OTda faylni o'qish paytida \r\n belgisi \n ga aylanadi, teskarisi esa yozishda amalga oshiriladi;
- “” (bo’sh satr) – satr oxirigacha ishlov berish amalga oshirilmaydi;
- "<Maxsus belgi>" – ko’rsatilgan maxsus belgi satr oxirini belgilash uchun ishlatiladi va qo'shimcha ishlov berish amalga oshirilmaydi. Faqat \r\n, \r va \n lar maxsus belgilar sifatida ko'rsatilishi mumkin.

13.2. Fayllar bilan ishlash metodlari

Faylni ochganimizdan so'ng open() funktsiyasi fayl bilan keyingi ish olib boriladigan ob'ektni qaytaradi. Ob'ekt turi faylni ochish va buferlash rejimiga bog'liq bo’ladi. Asosiy metodlar bilan tanishtirib o’tamiz:

- ✓ close() – faylni yopadi. Interpretator ob'ektni havolasi bo'lmaganida avtomatik ravishda o'chirib tashlaganligi sababli, kichik dasturlarda siz faylni aniq yopolmaysiz. Biroq, faylni aniq yopish dasturlash uslubining

yaxshi belgilaridan biri hisoblanadi. Bundan tashqari, agar fayl yopilmay qolgan bo'lsa, quyidagicha ogohlantirish xabari hosil bo'ladi:

"Resourcewarning : unclosed file".

Python tili kontekst menejeri protokolini qo'llab-quvvatlaydi. Ushbu protokol kod blokida istisno ro'y berganidan yoki hech qanday istisno yo'qligidan qat'i nazar, fayl yopilishini kafolatlaydi. Misol:

```
with open ( r"file.txt", "w", encoding="cpl251 ") as f:
        f.write ( "Satr")                    # Satrni faylga yozamiz
# Bu yerda fayl avtomatik ravishda yopiladi.
```

- ✓ write(<Ma'lumotlar>) – satrlarni yoki baytlar ketma-ketligini faylga yozadi. Agar parametr sifatida satr ko'rsatilgan bo'lsa, unda fayl matn rejimida ochilishi kerak. Baytlar ketma-ketligini yozish uchun esa faylni ikkilik(binar) rejimda ochish kerak. Ikkilik(binar) rejimda qatorni va matn rejimida baytlar ketma-ketligini yozish mumkin emasligini unutmaslik kerak. Bu metod yozilgan belgilar yoki baytlar sonini qaytaradi. Faylga yozishga misol ko'rib chiqaylik:

```
>>> # Matn rejimi
>>> f = open (r"file.txt", "w", encoding="cp1251 ")
>>> f.write ( "Satrl\n Satr2 ")                    # Satrni faylga yozamiz
15
>>> f . close () # Faylni yopamiz
>>> # Ikkilik (binar) rejimi
>>> f = open (r" file.txt", "wb")
>>> f.write (bytes( "Satrl\nSatr2 ", "cp1251") )
15
>>> f.write (bytearray ( " \nSatr3", "cp1251") )
8
>>> f.close ()
```

- ✓ writelines(<Ketma-ketlik>) – ketma-ketlikni faylga yozadi. Agar ketma-ketlikning barcha elementlari satrlar bo'lsa, unda fayl matn rejimida

ochilishi kerak. Agar barcha elementlar baytlar ketma-ketligi bo'lsa, unda faylni ikkilik (binar) rejimda ochish kerak. Ro'yxat elementlarini yozishga misol:

```
>>> # Matn rejimi
>>> f = open (r"file.txt ", "w", encoding="cp 1251" )
>>> f.writelines ( ["Satrl\n", "Satr2 " ] )
>>> f. close ()
>>> # Ikkilik (binar) rejimi
>>> f = open (r"file . tx t", "wb")
>>> arr = [bytes ( "Satrl \n" , "cp1251" ) , bytes ( "Satr2 ", " cp125 1 " )]
>>> f .writelines(arr)
>>> f.close ()
```

- ✓ writable() – agar faylga yozish mumkin bo'lsa, True (rost), aks holda False (yolg'on) qiymat qaytaradi:

```
>>> f = open ( r"file.txt", "r")          # Faylni o'qish uchun ochamiz
>>> f.writable ()
False
>>> f = open ( r"file.txt", "w" )         # Faylga yozish uchun ochamiz
>>> f .writable ()
True
```

- ✓ read ([<miqdor>]) – fayldan ma'lumotlarni o'qiydi. Agar fayl matn rejimida ochilsa, u holda satr, ikkilik (binar) rejimida esa baytlar ketma-ketligi qaytariladi. Agar parametr ko'rsatilmagan bo'lsa, faylning tarkibi ko'rsatgichning joriy holatidan fayl oxiriga qaytariladi:

```
>>> # Matn rejimi
>>> with open ( r"file.txt" , "r", encoding="cp l251") as f:
f.read ()
'Satrl\nSatr2 '
>>> # Ikkilik (binar) rejimi
>>> with open ( r"file.txt", " rb") as f:
```

f. read ()

b ' \xdl \xf2\xf0\xee\xea\xe01\n\xdl\xf2\xf0\xee\xea\xe02 '

Agar siz parametr sifatida raqamni ko'rsatsangiz, unda har bir murojaatda ko’rsatilgan belgilar yoki baytlar soni qaytariladi. Fayl tugagandan so'ng, metod bo'sh satrni qaytaradi. Misol:

>>> # Matn rejimi

>>> f = open (r"file.txt", "r", encoding="cp1251")

>>> f . read (8) # 8 belgisini sanaymiz

' Satr1\n '

>>> f.read (8) # 8 belgisini sanaymiz

'Satr2 '

>>> f.read (8) # Faylni tugatish

"

>>> f.close ()

- ✓ readline ([<miqdor>]) – har bir murojaatda fayldan bitta qatorni o'qiydi. Agar fayl matn rejimida ochilsa, u holda satr, ikkilik (binar) rejimida esa baytlar ketma-ketligi qaytariladi. Qaytarilayotgan satr uzatilayotgan satr belgilarini o'z ichiga oladi. Oxirgi satr istisno hisoblanadi, agar u satrga o’tkazish belgisi bilan tugamasa, u holda qo’shilmaydi. Fayl tugagandan so'ng, bo'sh satr qaytariladi. Misol:

  >>> # Matn rejimi

  >>> f = open (r"file.tx t", "r", encoding="cp l251")

  >>> f . readline ( ) , f . readline ()

  (' Satrl \n ' , 'Satr2 ')

  >>> f.readline () # Faylni tugatish

  ' '

```
>>> f . close ()
>>> # -Ikkilik (binar) rejimi
>>> f = open (r"file.tx t ", "rb" )
>>> f . readline ( ) , f.readline ()
(b ' \xdl\xf2\xf0\xee\xea\xe01\n ' , b ' \xdl\xf2\xf0\xee\xea\xe02 ')
>>> f.readline ()                    # Faylni tugatish
b ' '
>>> f.close ()
```

Agar ixtiyoriy parametrda raqam ko'rsatilgan bo'lsa, yangi satrli belgi (\ n)ni o'qish, fayl oxiridagi belgi yoki ko’rsatilgan sonli belgi fayldan o'qilguncha amalga oshiriladi. Boshqacha qilib aytadigan bo'lsak, agar satrdagi belgilar soni parametr qiymatidan kam bo'lsa, unda ko’rsatilgan belgilar soni emas, balki bitta satr o'qiladi va agar satrdagi belgilar soni ko'proq bo'lsa, unda ko’rsatilgan belgilar soni qaytariladi. Misol:

```
>>> f = open (r"file.txt ", "r", encoding="cpl251 ")
>>> f . readline (2), f.readline (2)
( ‘S’ , ‘a’ )
>>> f . readline (100) # Bu 100 ta belgini emas, balki bitta satrni qaytaradi.
‘trl\n’
>>> f . close ()
```

- ✓ readlines () – ro'yxatdagi fayl tarkibini to'liq o'qiydi. Ro’yxatning har bir elementi bitta satrni, shu jumladan satrga o’tkazish belgisini ham o'z ichiga oladi. Oxirgi satr esa istisno hisoblanadi. Agar satrni oxiri satrga o’tkazish belgisi bilan tugamagan bo’lsa, u belgi avtomatik qo’shilib qolmaydi. Agar fayl matn rejimida ochilgan bo'lsa, unda satrlar ro'yxati, ikkilik rejimda esa baytlar ro'yxati qaytariladi. Misol:

```
>>> # Matn rejimi
>>> with op en ( r" file . tx t" , " r", encoding="cpl251") as f:
f. readlines ()
[ ' Satrl \n ' , ' Satr2 ' ]
```

```
>>> # Ikkilik rejimi
>>> with open ( r" file.tx t" , "rb") as f:
f. readlines ()
[b ' \xdl \xf2\xf0\xee\xea\xe01\n ' , b ' \xdl\xf2\xf0\xee\xea\xe02 ']
```

- ✓ _next_ ( ) – har bir murojaatda bitta qatorni o'qiydi. Agar fayl matn rejimida ochilgan bo'lsa, unda satr, ikkilik rejimda esa baytlar ketma-ketligi qaytariladi. Faylni oxiriga yetkanda Stopiteration istisnosini qaytaradi. Misol:

```
>>> # Matn rejimi
>>> f = open(r"file.txt", "r", encoding="cp1251")
>>> f.__next__(), f.__next__()
('Satr1\n', 'Satr2')
>>> f.__next__()                    # Faylni tugatish
Traceback (most recent call last):
 File "<pyshell#26>", line 1, in <module>
 f.__next__()                    # Faylni tugatish
StopIteration
>>> f.close()
```

__Next__ () metodi tufayli biz for operatori yordamida fayl satrlari bo'yicha satrlarni takrorlashimiz mumkin. For operatori avtomatik ravishda har bir takrorlashda __next__ () metodini chaqiradi. Barcha satrlarni qabul qilishi uchun satrga o’tkazish belgisi (\n)ni olib tashlashimiz kerak bo’ladi:

```
>>> f = open (r"file . txt ", "r", encoding="cp1251 ")
>>> for line in f : print ( line.rstrip ( " \n" ) , end=" " )
Satr1 Satr2
>>> f.close ()
```

- ✓ flush () – buferdan ma'lumotlarni diskka majburiy ravishda yozadi.
- ✓ fileno () – faylning butun deskriptorini qaytaradi. Qaytaradigan qiymati har doim 2 dan katta bo'ladi, chunki 0 *stdin* kiritish standartiga

bog’langan, 1 *stdout* chiqarish standartiga bog’langan va 2 *stderr* xatolik haqida habar chiqarish standartiga o'rnatiladi. Misol:

```
>>> f = open ( r" file . txt ", "r", encoding="cp l251" )
>>> f. fileno ()              # Fayl deskriptori
3
>>> f . close ()
```

- ✓ truncate ([<Miqdor>]) - faylni ko’rsatilgan miqdordagi sonli belgilarga (agar matn rejimi ko'rsatilgan bo'lsa) yoki baytlarga (ikkilik rejimda) qisqartiradi. Ushbu metod yangi fayl hajmini qaytaradi. Misol:

```
>>> f = open (r"file . txt ", "r+", encoding="cpl251 ")
>>> f.read ()
'Satrl\nSatr2 '
>>> f . truncate (4)
4
>>> f.close ()
>>> with open ( r"file . txt", " r", encoding="cp1251") as f:
f. read ( )
'Sat'
```

- ✓ tell () – faylning boshiga nisbatan ko'rsatgich o'rnini butun son sifatida qaytaradi. Shuni e’tiborgaolish kerakki, Windows OTda tell () metodi \ r belgisini qo'shimcha bayt sifatida qabul qiladi, agar bu fayl matn rejimida ochilgan bo’lsa o'chiriladi. Misol:

```
>>> with open ( r"file . tx t", "w", encoding="cpl251 ") as f:
f.write ( "Stringl\nString2")
15
>>> f = open (r"file . txt", "r", encoding="cp1251")
>>> f.tell () # Ko'rsatkich faylning boshida joylashgan
0
>>> f.readline () # Ko'rsatkichni joyini o’zgartirish
' Stringl\n '
```

```
>>> f.tell () # Qaytgan natija 9 ( 8 + ' \ r' ) ,  8 emas ! ! !
9
>>> f close ()
```

Ushbu nomuvofiqlikni oldini olish uchun faylni matnda emas, ikkilik rejimda oching:

```
>>> f = open ( r"file . txt ", "rb")
>>> f. readline () # Ko'rsatkichni joyini o’zgartirish
b ' Stringl\r\n '
>>> f. tell () # Endi qiymat mos keladi
9
>>> f . close ()
```

Fayl ob'ektlari metodlardan tashqari bir nechta atributlarni qo'llab-quvvatlaydi:

- ✓ name – faylning nomi
- ✓ mode – faylni ochilgan rejimi
- ✓ closed – agar fayl yopilgan bo’lsa True(rost) qiymat, aks holda False(yolg’on) qiymat qaytaradi. Misollar:

```
>>> f = open (r"file . txt ", "r+b")
>>> f.name, f.mode, f.closed
('file.txt', 'rb+' , False)    # faylni yopmaganimiz uchun False qiymat qaytardi
>>> f.close ()                 # faylni yopamiz
>>> f.closed
True
```

- ✓ encoding - faylga yozishdan oldin yoki o'qiyotganda satrlarni konvertatsiya qilish uchun ishlatiladigan kodlash nomi. Atribut faqat matn rejimida ishlaydi. Shuni ham unutmangki, atributning qiymatini o'zgartira olmaysiz, chunki uni faqat o'qish mumkin. Misol:

```
>>> f = open (r"file . txt", "a", encoding="cpl25 1")
>>> f . encoding
```

' cp1251 '

>>> f . close ()

- ✓ buffer – buferga kirishga imkon beradi. Atribut faqat matn rejimida ishlaydi. Ushbu ob'ekt yordamida siz matn oqimiga baytlar ketma-ketligini yozishingiz mumkin. Misol:

  >>> f = open (r"file . txt ", "w", encoding="cp1251")

  >>> f.buffer . write (bytes ("С трока", "cp125 1"))

  6

  >>> f . close ()

13.3. Os moduli yordamida fayllarga kirish

Os moduli fayllar bilan ishlaydigan quyi darajali funksiyalardan tashkil topgan. Ushbu modulning ishlashi foydalanilayotgan operatsion tizimga bog'liq bo’ladi. Biz foydalanayotgan versiyasining nomini modulning name atributidan foydalanib olishimiz mumkin. Windows XP va undan keyingi versiyalarida atribut "nt" belgisini qaytaradi. Buni misolda ko’rib o’tsak:

```
>>> import os
>>> os.name                    # Operatsion tizim Windows7
'nt'
```

OS modulidagi quyidagi funktsiyalar fayllarga kirish uchun mo'ljallangan:

- ✓ open (<faylga yo’l>,<ochish rejimi>[, mode = 0o777]) – faylni ochadi va butun qiymatli deskriptor qaytaradi, uning yordamida fayl bilan qo'shimcha ishlash amalga oshiriladi. Agar faylni ochib bo'lmaydigan bo'lsa, OSError istisnosi yoki OSError sinfining sinfosti bo'lgan istisnolaridan birini qaytaradi. Windows operatsion tizimidagi <ochish rejimi (Mode)> parametrida quyidagi belgilarni ko'rsatish mumkin:
  - os.O_RDONLY – faqat o’qish
  - os.O_WRONLY – faqat yozish
  - os.O_RDWR – o’qish va yozish

- os.O_APPEND – faylni oxiriga qo’shish
- os.O_CREAT – Agar fayl mavjud bo’lmasa va os.O_EXCL belgisi ko’rsatilmagan bo’lsa, fayl yaratadi.
- os.O_EXCL – bu os.O_CREAT belgisi bilan birgalikda yangi fayl yaratish uchun qo’llaniladi. Agar fayl mavjud bo’lsa FileExistsError xatolik qaytaradi.
- os.O_TEMPORARY - bu ham os.O_CREAT belgisi bilan birgalikda yopilgandan so'ng darhol avtomatik ravishda o'chiriladigan yangi vaqtinchalik fayl yaratish uchun qo’llaniladi.
- os.O_SHORT_LIVED – bu belgi os.O_TEMPORARY belgisi kabi foydalaniladi. Faqat yaratilgan fayl operativ xotirada saqlanib qoladi.
- os.O_TRUNC - fayl tarkibini tozalash uchun ishlatiladi.
- os.O_ BINARY – faylni ikkilik(binar) rejimida ochish.
- os.O_TEXT – faylni matn rejimida ochish. Windows operatsion tizimida fayl odatda matn rejimida ochiladi.

Keling bir necha misollarni ko’rib chiqaylik. Faylni yozish uchun ochamiz va unga bitta qator yozamiz. Agar fayl mavjud bo’lmasa uni yaratamiz, mavjud bo’lsa uni tozalaymiz:

```
>>> import os # Modulni bog’laymiz
>>> mode = os.O_WRONLY | os.O_CREAT | os.O_TRUNC
>>> f = os . open ( r" file.txt", mode)
>>> os.write (f, b"String\n" )          # Ma’lumotlarni yozamiz
8
>>> os.close ( f)                       # Faylni yopamiz
```

Faylni oxiriga yana bitta qator qo’shamiz:

```
>>> mode = os.O_WRONLY | os.O_CREAT | os.O_APPEND
```

```
>>> f = os.open ( r" file.txt", mode )
>>> os . write (f, b"String2 \n" )        # Ma’lumotlarni yozamiz
8
>>> os.close (f)                          # Faylni yopamiz
```

Fayl tarkibini matn rejimida o’qiymiz:

```
>>> f = os.open ( r" file . txt", os.O_RDONLY)
>>> os.read ( f, 50)                      #  Fayldan 50 bayt o’qiymiz
b ' Stringl\nString2\n'
>>> os.close (f)                          # Faylni yopamiz
```

Endi fayl tarkibini ikkilik rejimida o’qiymiz:

```
>>> f = os . open (r"file.txt", os.O_RDONLY | os.O_BINARY)
>>> os.read (f, 50)                       # Fayldan 50 bayt o’qiymiz
b ' Stringl\r\nString2\r\n '
>>> os.close (f)                          # Faylni yopamiz
```

- ✓ read (<descriptor>, <baytlar miqdori>) - fayldan ko’rsatilgan miqdordagi bayt sonini o'qiydi. Fayl tugagandan so'ng, bo'sh satr qaytariladi. Misol:

  ```
  >>> f = os . open ( r" file.txt", os.O_RDONLY)
  >>> os.read ( f, 5 ) , os.read (f, 5 ) , os.read (f, 5) , os.read ( f, 5)
  (b' Strin ', b' gl \nS ', b ' tring', b' 2 \n ' )
  >>> os.read (f, 5)                    # Faylni tugatish
  b ' '
  >>> os . close (f)                    # Faylni yopamiz
  ```

- ✓ write (<descriptor>, <baytlar ketma-ketligi>) – faylga baytlar ketma-ketligini yozadi. Yozilgan baytlar sonini qaytaradi.
- ✓ Close (<descriptor>) – faylni yopadi.

13.4. StringIO va BytesIO sinfi

IO modulidagi StringIO sinfi fayl ob'ekti sifatida satrlar bilan ishlashga imkon beradi. Ushbu fayl ob'ekti bilan barcha amallar (keyingi qatorlarda biz uni "fayl" deb ataymiz) operativ xotirada bajariladi. Sinf konstruktorining formati quyidagicha:

StringIO ([<Boshlang'ich qiymat >] [, newline=None])

Agar birinchi parametr ko'rsatilmagan bo'lsa, unda boshlang'ich qiymat bo'sh satr bo'ladi. Ob'ekt yaratilgandan so'ng, hozirgi holat ko'rsatkichi "fayl" ning boshiga o'rnatiladi. Sinf konstruktori tomonidan qaytarilgan ob'ekt quyidagi metodlarga ega:

- ✓ close () – “fayl”ni yopadi. "fayl" ochiq yoki yopiq ekanligini tekshirish uchun closed atributidan foydalanadi. Atribut agar “fayl” yopilgan bo’lsa True qiymat qaytaradi, aks holsa esa False qiymat qaytaradi.

- ✓ getvalue () – fayl tarkibini satr ko’rinishida qaytaradi.

```
>>> import io                    # Modulni bog’laymiz
>>> f = io.StringIO("String1\n")
>>> f.getvalue()                 # Fayl tarkibini olamiz
'String1\n'
>>> f.close()                    # Faylni yopamiz
```

- ✓ tell () – “fayl” boshiga nisbatan ko'rsatgichning joriy holatini qaytaradi.
- ✓ write (<satr>) – “fayl” ga satrni yozadi:

```
>>> f = io . StringIO ( "S tringl \n")
>>> f.seek (O, 2)          # ko’rsatkichni “fayl”ning oxiriga o’tkazamiz
8
>>> f.write ( "String2\n")      # satrni “fayl”ga yozamiz
8
>>> f.getvalue ()               # “fayl” tarkibini olamiz
'Stringl \nString2 \n '
>>> f.close ()                  # “fayl”ni yopamiz
```

- ✓ writelines (<ketma – ketlik>) – “fayl”ga ketma-ketlikni yozadi:

```
>>> f = io.StringIO ()
>>> f.writelines ( [" Stringl\n", "String2\n" ] )
>>> f.getvalue ()          # “fayl” tarkibini olamiz
' Stringl \nString2\n '
>>> f.close ()                    # “fayl”ni yopamiz
```

- ✓ read (<belgilar miqdori>) - "fayl" dan ma'lumotlarni o'qiydi. Agar parametr ko'rsatilmagan bo'lsa, u holda "fayl" ning tarkibi ko'rsatgichning joriy holatidan "fayl" oxirigacha qaytariladi. Agar siz parametr sifatida raqamni ko'rsatsangiz, unda har bir murojaatda ko’rsatilgan belgilar soni qaytariladi. "fayl" tugagandan so'ng, metod bo'sh satrni qaytaradi. Misol:

```
>>> f = io . String IO ( "Stringl \nString2\n" )
>>> f.read ()
' Stringl\nString2\n '
>>> f . seek (0)                  # ko’rsatkichni “fayl”ning boshiga o’tkazamiz
0
>>> f.read (S) , f.read (5) , f.read (S) , f.read (S) , f.read (S)
( ' Strin ' , ' gl \nSt ' , ' ring2 ' , ' \n ' , ' ' )
>>> f.close ()                    # “fayl”ni yopamiz
```

- ✓ readline (<belgilar miqdori>) - har bir murojaatda "fayl" dan bitta satrni o'qiydi. Qaytarilayotgan satr uzatilayotgan satr belgilarini o'z ichiga oladi. Oxirgi satr istisno hisoblanadi, agar u satrga o’tkazish belgisi bilan tugamasa, u holda qo’shilmaydi. “fayl” tugagandan so'ng, bo'sh satr qaytariladi. Misol:

```
>>> f = io . StringIO ( "Stringl \nString2 ")
>>> f.readline () , f.readline () , f . readline ()
( ' Stringl\n ', ' String2 ', ' ' )
>>> f . close ()                  # “fayl”ni yopamiz
```

Agar ixtiyoriy parametrda raqam ko'rsatilgan bo'lsa, yangi satrli belgi (\ n)ni o'qish, fayl oxiridagi belgi yoki ko’rsatilgan sonli belgi fayldan o'qilguncha

amalga oshiriladi. Boshqacha qilib aytadigan bo'lsak, agar satrdagi belgilar soni parametr qiymatidan kam bo'lsa, unda ko’rsatilgan belgilar soni emas, balki bitta satr o'qiladi va agar satrdagi belgilar soni ko'proq bo'lsa, unda ko’rsatilgan belgilar soni qaytariladi. Misol:

```
>>> f = io.StringIO ( "Stringl \nString2\nString3\n")
>>> f . readline (S) , f.readline (S)
( ' Strin ' , 'gl \n ' )
>>> f . readline ( 100 )          # Bu 100 ta belgini emas, balki bitta satrni qaytaradi.
' String2 \n '
>>> f. close ()                   # “fayl”ni yopamiz
```

- ✓ readlines (<Belgilarning taxminiy soni>)– ro'yxatdagi fayl tarkibini to'liq o'qiydi. Ro’yxatning har bir elementi bitta satrni, shu jumladan satrga o’tkazish belgisini ham o'z ichiga oladi. Oxirgi satr esa istisno hisoblanadi. Agar satrni oxiri satrga o’tkazish belgisi bilan tugamagan bo’lsa, u belgi avtomatik qo’shilib qolmaydi. Misol:

```
>>> f = io.String IO ( "Stringl\n String2\nString3" )
>>> f.readlines ()
[ ' Stringl\n ', ' St ring2\n ', ' String3 ']
>>> f.close ()                    # “fayl”ni yopamiz
```

Agar ixtiyoriy parametrda raqam ko'rsatilgan bo'lsa, u holda ko’rsatilgan belgilar soni va satrning oxiridagi \n fragmentigacha o’qiydi. Keyin ushbu satr bo'linadi va ro'yxatning yangi satriga qo'shiladi. Misol:

```
>>> f = io.String IO ( "Stringl \nString2\nString3" )
>>> f . readlines(14)
[ ' S tringl\n ', ' String2 \n ']
>>> f.seek (0)                    # ko’rsatkichni “fayl”ning boshiga o’tkazamiz
0
>>> f.readlines (17)
[ ' Stringl\n ', ' String2\n ' , ' String3 ']
```

```
>>> f . close ()                # “fayl”ni yopamiz
```

- ✓ flush - buferdan ma'lumotlarni "fayl"ga chiqaradi.
- ✓ truncate (<belgilar miqdori>) – “fayl” ni ko’rsatilgan belgilar soniga qisqartiradi. Misol:

```
>>> f = io . StringIO ( "Stringl\nString2 \nString3")
>>> f . truncate (15)           # “fayl”ni qirqamiz
15
>>> f. getvalue ()              # “fayl” tarkibini olamiz
' Stringl\nString2 '
>>> f.close ()                  # “fayl”ni yopamiz
```

Ko’rib turganingizdek, StringIO sinfi faqat satrlar bilan ishlaydi. Baytlar ketma-ketligi bo'lgan "fayllar" da shunga o'xshash operatsiyalarni bajarish uchun IO modulining BytesIO sinfidan foydalanamiz. Sinf konstruktorining formati quyidagicha:

BytesIO ([<Boshlang'ich qiymat >])

BytesIO sinfi StringIO sinfi bilan bir xil metodlarni qo'llab-quvvatlaydi, ammo metodlar satrlarni emas, balki baytlarning ketma-ketligini qiymat sifatida qabul qiladi va qaytaradi. Keling, misol yordamida asosiy operatsiyalarni ko'rib chiqaylik:

```
>>> import io                   # Modulni bog’laymiz
>>> f = io.BytesIO (b"Stringl \n")
>>> f.seek(0, 2)                # ko’rsatkichni faylning oxiriga o’tkazamiz
8
>>> f.write (b"String2 \n")     # faylga yozamiz
8
>>> f.getvalue ()               # fayl tarkibini olamiz
b ' Stringl\nString2 \n'
>>> f.seek (0)                  # ko’rsatkichni faylning boshiga o’tkazamiz
```

```
0

>>> f.read()                    # ma’lumotlarni o’qiymiz

b ' Stringl \nString2\n '

>>> f.close ()                  # faylni yopamiz
```

BytesIO sinfi, shuningdek, *memoryview* ob'ektiga havolani qaytaradigan getbuffer() metodini qo'llab-quvvatlaydi. Ushbu ob'ekt yordamida siz ma'lumotni indeks yoki qism bo'yicha olishingiz va o'zgartirishingiz, ma'lumotlarni butun sonlar ro'yxatiga (tolist() metodi yordamida) yoki baytlar ketma-ketligiga (tobytes () metodi yordamida) o'zgartirishingiz mumkin. Misol:

```
>>> f = io.BytesIO (b" P ython" )

>>> buf = f.getbuffer ()

>>> buf [0]                     # Indeks bo’yicha belgini olamiz

b 'P'

>>> buf [0] = b"J"              # Indeks bo’yicha belgini o’zgartiramiz

>>> f.getvalue ()               # Tarkibini olamiz

b ' Jython '

>>> buf.tolist ()               # Raqamlar ro’yhatiga o’tkazamiz

[74 , 12 1, 116, 104 , 111 , 110]

>>> buf.tobytes ()              # bytes tipiga o’tkazamiz

b ' Jython '

>>> f . close ()                # faylni yopamiz
```

13.5. Fayl va kataloglarga kirish huquqi

UNIX oilasiga tegishli operatsion tizimlarida har bir ob'ektga (faylga yoki katalogga) ma'lum bir foydalanuvchi turiga mos kirish huquqlari beriladi. Masalan, tizim egasiga, gruppaga va hokazolarga. Quyidagi kirish huquqlari berilishi mumkin:

- o’qish;
- yozish;
- bajarish;

Kirish huquqlarini harflarda quyidagicha ko'rsatilgan bo’ladi:

- r – faylni o'qish va katalog tarkibini ko'rish mumkin;
- w – faylni o'zgartirish, o'chirish va nomini o'zgartirish, katalogda fayllarni yaratish yoki o'chirish mumkin. Katalog qayta nomlanishi yoki o'chirilishi mumkin.
- x - faylni bajarish mumkin va katalogda fayllar bilan ishlash, shu jumladan undagi fayllarni izlashni amalga oshirish mumkin.

Faylga kirish huquqi shunga o'xshash yozuv bilan belgilanadi:

    -rw-r--r--

Birinchi – belgisi faylga hech qanday kirish huquqini bermaydi. Keyingi 3 ta belgi

(rw-) fayl egasiga kirish, o’qish va yozish huquqini taqdim etadi. – belgisi ijro etishga ruxsat yo'qligini anglatadi. Keyingi uchta (r--) belgi guruhning faqat o'qish uchun ruxsatlarini aniqlaydi. So'nggi uchta belgi (r- -) boshqa barcha foydalanuvchilar uchun faqat o'qish huquqlarini o'rnatadi.

Katalogga kirish huquqlari quyidagi qator bilan belgilanadi:

drwxr-xr-x

Birinchida turgan d harfi buni katalog ekanligini ifodalaydi. Keyingi rwx esa katalog egasi katalogda hamma amalni bajara olishini bildiradi va guruh va boshqa barcha foydalanuvchilar katalogni faqat o'qish va qidirishni amalga oshiradi, ya’ni (r-x).

Katalogni ko'rish uchun ruxsatni (x) o'rnatish kerak. Kirish huquqlari raqam bilan ham ko'rsatilishi mumkin. Bunday raqamlarga ruxsat maskasi deyiladi. Bunday raqamlar 0 dan 7 gacha bo’lgan uchta raqamdan iborat bo’ladi. Birinchi raqam egasiga, ikkinchisi guruhga, uchinchisi esa boshqa barcha foydalanuvchilarga huquqlarni belgilaydi. Masalan, -rw-r-r-- kirish huquqi 644 raqamiga to'g'ri keladi. Keling, kirish huquqlari maskasiga, ikkilik va harfiy yozuvlarga kiritilgan raqamlarni moslashtiramiz. Uni quyidagi jadvalda ko’rsatsak:

Sakkizlik raqam	**Ikkilik yozuv**	**Harfli yozuv**	**Sakkizlik raqam**	**Ikkilik yozuv**	**Harfli yozuv**
0	000	---	4	100	r--
1	001	--x	5	101	r-x
2	010	-w-	6	110	rw-
3	011	-wx	7	111	rwx

Endi ushbu jadvaldagi ma'lumotlarga ko'ra rw-r--r-- kirish huquqlari quyidagicha yozilishi mumkin: 110 100 100, bu esa o’z navbatida sakkizlikda 644 raqamiga aylanadi. Shunday qilib, agar huquq berilsa, unda tegishli pozitsiya 1 ga teng, agar bo'lmasa 0 ga teng bo'ladi.

Fayl yoki katalogga kirish huquqini aniqlash uchun os modulidan access() funktsiyasidan foydalish kerak. Bu funktsiya quyidagi formatga ega:

access (<Fayl yoki katalogga yo’l>, <Rejim>)

Tekshiruv muvaffaqiyatli o'tgan bo'lsa, funktsiya True(Rost) qiymat, aks holda False(yolg’on) qiymat qaytaradi. <Rejim> parametri tekshirish turini belgilaydigan quyidagi doimiylarni o'z ichiga olishi mumkin:

- os.F_OK – yo’l yoki fayl mavjudligini tekshirish:

```
>>> import os                          # os modulini bog’laymiz
>>> os.access( r" file.txt", os.F_OK)  # Fayl mavjud
True
```

>>> os.access(r"C : \book", os.F_OK) # Katalog mavjud

True

>>> os.access(r"C : \book2", os.F_ OK) # Katalog mavjud emas

False

- os.R_OK – fayl yoki katalogning o'qish mumkinligini tekshirish;
- os.W_OK – fayl yoki katalogga yozish mumkinligini tekshirish;
- os.X_OK – fayl yoki katalogning bajarilishini aniqlash.

Dasturdan ruxsatlarni o'zgartirish uchun os modulidan chmod() funktsiyasidan foydalanish kerak. Funktsiya quyidagi formatga ega:

chmod (<Yo'l>, < Kirish huquqlari >)

Kirish huquqi 0o (bu sakkizlik yozuvni bildiradi) belgilar ketma-ketligidan keyin raqamlar ko'rinishida beriladi. Masalan:

>>> os.chmod (r"file.txt", 0o777) # Faylga to'liq ruxsatni ta'minlaydi.

13.6. Faylarni monipulyasiyalash funksiyasi

Fayllarni nusxalash va ko'chirish uchun ***shutil*** modulidan va uning quyidagi funksiyalaridan foydalaniladi:

- ✓ copyfile (<nusxa olinayotgan fayl>, <qayerga nusxalash>) – fayl tarkibini boshqa faylga ko'chirishga imkon beradi. Hech qanday huquqiy ma'lumot (masalan, ruxsatnomalar) ko'chirilmaydi. Agar fayl mavjud bo'lsa, uning ustiga yoziladi. Agar faylni nusxalash mumkin bo'lmasa, OSError istisnosi yoki ushbu sinfning sinfostisi bo'lgan istisnolardan biri qaytariladi. Misol:

  >>> import shutil # Modulni bog'laymiz

  >>> shutil.copyfile (r" file.txt", r" file2.tx t ")

  >>> # Faylga yo'l mavjud emas :

  >>> shutil.copyfile (r"file.txt", r"C : \book2\file2.txt ")

  ... Parcha chiqarib tashlandi ...

  FileNotFoundError : [Errno 2] No such file or directory :

  'C: \\book2\\file2.txt'

FileNotFoundError istisnosi OSError sinfining kichik sinfostisi bo'lib, ko'rsatilgan fayl topilmasa qaytariladi.

- ✓ copy (<nusxa olinayotgan fayl>, <qayerga nusxalash>) – faylni ruxsatnomalar bilan birga nusxalashga imkon beradi. Agar fayl mavjud bo'lsa, uning ustiga yoziladi. Agar faylni nusxalash mumkin bo'lmasa, OSError istisnosi yoki ushbu sinfning sinfostisi bo'lgan istisnolardan biri qaytariladi. Misol:

  >>> shutil.copy(r" file.txt", r" file3.txt ")

- ✓ copy2 (<nusxa olinayotgan fayl>, <qayerga nusxalash>) – barcha ma’lumotlar bilan birga faylni nusxalashga imkon beradi. Agar fayl mavjud bo'lsa, uning ustiga yoziladi. Agar faylni nusxalash mumkin bo'lmasa, OSError istisnosi yoki ushbu sinfning sinfostisi bo'lgan istisnolardan biri qaytariladi. Misol:

  >>> shutil.copy2 (r" file.txt", r" file4.txt")

- ✓ move (<faylga yo’l>, <qayerga ko’chirish>) – asl faylni o'chirib, faylni belgilangan joyga ko'chiradi. Agar fayl mavjud bo'lsa, uning ustiga yoziladi. Agar faylni ko’chirish mumkin bo'lmasa, OSError istisnosi yoki ushbu sinfning sinfostisi bo'lgan istisnolardan biri qaytariladi. File4.txt faylini C:\book\test katalogiga ko'chirishga misol:
  >>> shutil . move (r" file4.txt" , r" C:\book\test")

Fayllarning nomini o'zgartirish va o'chirish uchun ***os*** modulidan quyidagi funksiyalar qo'llaniladi:

- ✓ rename (<eski nom>, <yangi nom>) – fayl nomini o'zgartiradi. Agar faylni nomini o’zgartirish mumkin bo'lmasa, OSError istisnosi yoki ushbu sinfning sinfostisi bo'lgan istisnolardan biri qaytariladi. Istisno yordamida ishlov berish orqali fayl nomini o'zgartirishga misol:
  import os # Modulni bog’laymiz
  try :
  os . rename (r" file3.txt ", "file4.txt ")

except OSError :

print (" Fayl nomini o'zgartirib bo'lmadi ")

else :

print (" Fayl qayta nomlandi ")

- ✓ remove (<faylga yo'l>) va unlink (<faylga yo'l>) – faylni o'chirish imkonini beradi. Agar faylni o'chirish mumkin bo'lmasa, OSError istisnosi yoki ushbu sinfning sinfostisi bo'lgan istisnolardan biri qaytariladi.

>>> os.remove (r" file2.txt ")

>>> os.unlink (r" file4.txt ")

Os.path moduli tarkibida fayl mavjudligini tekshirish, fayl hajmini olish va h.k. uchun qo'shimcha funksiyalar mavjud. Ushbu funksiyalarni tavsiflaymiz:

- ✓ exists (<yo'l>) –belgilangan yo'lning mavjudligini tekshiradi. Agar yo'l mavjud bo'lsa funksiya belgisi True, aks xolda False qaytaradi.

>>> irnport os.path

>>> os . path.exists (r" file.txt ") , os.path.exists (r" file2.txt ")

(True, False)

>>> os.path.exists (r"C : \book") , os.path.exists (r"C : \book2 ")

(True, False)

- ✓ getsize (<faylga yo'l>) – fayl hajmini baytda qaytaradi. Agar fayl mavjud bo'lmasa, OSError istisnosi qaytariladi:

>>> os.path.getsize (r" file.txt ") # Fayl mavjud

18

>>> os.path.getsize (r"file2.txt ") # Fayl mavjud emas

... Parcha chiqarib tashlandi ...

OSError : [Error 2] Ko'rsatilgan faylni topish imkoni bo'lmadi : ' file2.txt '

- ✓ getatime (<faylga yo'l>) – faylga oxirgi kirish vaqtini aniqlashga xizmat qiladi. Funksiya davr boshidan beri soniya sonini qiymat sifatida qaytaradi. Agar fayl mavjud bo'lmasa, OSError istisnosi qaytariladi. Misol:

```
>>> import time # time modulini bog’laymiz
>>> t = os.path. getatime ( r" file.txt " )
>>> t
1304 111982 . 96875
>>> time . strft ime ("%d.%m. %Y %H : %M:%S", time.localtime (t) )
' 30.04 . 2 011 01 : 19:4 2'
```

- ✓ getctime (<faylga yo’l>) – fayl yaratilgan sana haqida sizga ma'lumot beradi. Funksiya davr boshidan beri soniya sonini qiymat sifatida qaytaradi. Agar fayl mavjud bo'lmasa, OSError istisnosi qaytariladi. Misol:

```
>>> t = os . path.getctime ( r" file.txt")
>>> t
1304028509.015625
>>> time. strftime ("%d . % m. %Y %H : %M : %S", time.localtime (t) )
' 2 9.04 . 2011 02 : 08 : 29 '
```

- ✓ getmtime (<faylga yo’l>) – fayl oxirgi marta o'zgartirilgan vaqtni qaytaradi. Funksiya davr boshidan beri soniya sonini qiymat sifatida qaytaradi. Agar fayl mavjud bo'lmasa, OSError istisnosi qaytariladi. Misol:

```
>>> t = os.path.getmtime( r" file.txt")
>>> t
1304 044731 .265 625
>>> time.strftime ("%d.%m. % Y %H : %M : %S", time.localtime (t) )
' 2 9.04 . 2011 06 : 38:51'
```

Os modulidagi ***stat()*** funksiyasi sizga faylning hajmini va faylni yaratish, o'zgartirish va unga kirish vaqtini, shuningdek boshqa ma’lumotlar qiymatlarini olish imkonini beradi. Funksiya stat_result obyektini qiymati sifatida o'nta xususiyatni o'z ichiga oladi: st _ mode, st _ ino, st _dev, st_nlink, st_uid, st_gid, st_size, st_atime, st_mtime va st _ctime.

13.7. Fayl yoki kataloglarga yo’lni o’zgartirish

Os.path modulidan quyidagi funksiyalar fayl yoki katalogga yo'lni o'zgartirishi mumkin:

- ✓ abspath (<Nisbiy yo'l >) – joriy ishchi katalog joylashishini hisobga olgan holda nisbiy yo'lni absolyut yo'lga o'zgartiradi. Misollar:

  >>> import os.path

  >>> os.path.abspath (r"file.txt ")

  'C:\ \ book\ \file.txt '

  >>> os.path.abspath (r" folderl /file.txt ")

  ' C : \ \ book\\folderl\\file.txt '

  >>> os.path.abspath (r" .. /file.txt ")

  'C:\ \file.txt '

  Yuqorida ta'kidlab o'tilganidek, nisbiy yo'lda oldinga va orqaga yotiq chiziqlarni ishlatishimiz mumkin. Ularning barchasi os.path modulidagi ***sep*** atributi qiymati asosida avtomatik ravishda konvertatsiya qilinadi. Ushbu atributning qiymati ishlatilayotgan operatsion tizimga bog'liq. Windows operatsion tizimida ***sep*** atributining qiymatini ko'rsatamiz:

  >>> os.path.sep

  ' \ \'

  Windows OTda yo'lni ko'rsatayotganda, yotiq chiziq(slesh) maxsus belgi ekanligini unutmaslik lozim. Shu sababli yotiq chiziq ikki baravarga ko'paytirilishi yoki oddiy chiziqlar o'rniga formatlanmagan satrlardan foydalanish kerak. Misollar:

  >>> "C : \ \ temp\ \new\\file.txt " # to’g’ri

  'C: \ \temp\\new\ \file.txt '

  >>> r"C : \t emp\new\file.txt " # to’g’ri

  'C:\\ tem p\ \new\ \file.txt '

  >>> "C : \temp\new\file.txt " # noto’g’ri

  ' C : \temp\new\x0cile.txt '

  Bundan tashqari, agar chiziq satrning oxirida joylashgan bo'lsa, unda formatlanmagan satrlardan foydalanganda ham uni ikki baravar oshirish kerak.

```
>>> r"C: \temp\new\ "                    # noto'g'ri !!!
SyntaxError : EOL while scanning string literal
>>> r"C : \temp\new\\"
'C:\\ temp\ \new\ \\\ '
```

- ✓ isabs (<yo'l>) – agar yo'l absolyut bo'lsa, True, aks xolda esa False qaytaradi.

```
>>> os.path.isabs (r"C: \book\ file.txt " )
True
>>> os.path.isabs ("file.txt")
False
```

- ✓ basename (<yo'l>) – fayl nomini uning yo'lisiz qaytaradi.

```
>>> os.path.basename ( r"C : \book\folderl\file.txt " )
' file.txt'
>>> os.path.basename ( r"C: \book\folder" )
' folder'
>>> os.path.basename ( "C:\ \book\\folder\\")
'  '
```

- ✓ dirname (<yo'l>) – fayl saqlanadigan papkaga yo'lni qaytaradi.

```
>>> os.path.dirname ( r"C : \book\folder\file . txt")
'C:\\ book\\folder '
>>> os.path.dirname (r"C :\book\folder" )
'C:\ \book'
>>> os.path.dirname ( "C : \\book\\folder\\ " )
'C:\\ book\\folder '
```

- ✓ split (<yo'l>) – ikkita elementdan iborat kortej qaytaradi: fayl saqlanadigan papkaga yo'l va fayl nomi:

```
>>> os.path.split (r"C : \book\folder\file.txt")
( 'C:\ \book\ \ folder', 'file.txt')
>>> os.path.split ( r"C : \book\ folder")
( ' C : \ \book ' , 'folder' )
>>> os.path.split ( "C : \ \book\ \folder\ \" )
```

(' C : \ \book\ \ folder ', ' ')

- ✓ splitdrive (<yo'l>) – yo'lni disk nomiga va yo'lning qolgan qismiga ajratadi. Ikki elementdan iborat kortej qiymat sifatida qaytariladi.

  >>> os.path.splitdrive (r"C : \book\folder\ file.txt")

  (' C: ', '\ \book\ \ folder\ \file.txt ')

- ✓ splitext (<yo'l>) – ikkita elementdan iborat kortej qaytaradi: fayl nomi bilan, lekin kengaytmasiz va fayl kengaytmasi bo'lgan yo'llar (oxirgi nuqtadan keyingi qism):

  >>> os.path.splitext (r"C :\book\folder\file.tar.gz ")

  (' C : \\book\\folder\\file.tar ', '.gz')

- ✓ join (<Yo'l 1> [, . . . , <Yo'l N>]) - belgilangan yo'llar elementlarini birlashtiradi, agar kerak bo'lsa, ular orasiga ajratgichlarni kiritish mumkin:

  >>> os.path.join("C: \\ ", "book\\folder", " file.txt")

  ' C:\\book\\ folder\\file.txt '

  >>> os.path.join (r"C:\\ ", "book/folder/", "file.txt ")

  ' C : \\\\book/folder/file.txt'

  Oxirgi misolga e'tibor qarating – faylga yo'lni ko'rsatishda turli xil yotiq chiziqlar (sleshlar) ishlatiladi va natijada yo'l noto'g'ri bo'ladi. Ushbu yo'lni to'g'ri bajarish uchun normpath () funksiyasidan foydalanishingiz kerak:

  >>> p = os.path.join (r"C: \\" , "book/folder/ " , " file.txt ")

  >>> os.path.normpath(p)

  'C: \\book\\folder\\ file.txt '

13.8. Kiritish/chiqarishni qayta yo’naltirish

Fayllar bilan ishlash usullarini ko’rib chiqishda, ***fileno()*** usuli bilan qaytariladigan qiymat har doim 2 dan katta bo’ladi, chunki 0 raqami ***stdin*** standart kiritish uchun, 1 raqami ***stdout*** standart chiqarish va 2 raqami ***stderr*** standart xatolar haqida xabar chiqishi uchun berilgan bo’ladi. Ushbu oqimlarning barchasi fayl obyektlari bilan bir oz o’xshashdir. Masalan, ***stdout*** va ***stderr*** oqimlari xabarlarni ko’rsatish uchun write() metodini, ***stdin*** oqimi esa foydalanuvchi ma’lumotlarini qabul qilish uchun readline() metodini qo’llab-quvvatlaydi. Agar ushbu oqimlarga fayl metodlarini qo’llab- quvvatlaydigan obyektga havola berilgan bo’lsa, unda siz standart oqimlarni mos keladigan faylga yo’naltirishingiz mumkin. Misol uchun quyidagicha bajarishimiz mumkin:

Faylni chiqarishni qayta yo’naltirish:

```
>>> import sys                          # sys modulini bog’laymiz
>>> tmp_out = sys.stdout                # Havolani sys.stdoutga saqlaymiz
>>> f = open (r” file.txt “ , “a” )     # Yozish uchun faylni ochamiz
>>> sys.stdout = f                      # Chiqarishni faylga qayta yo’naltiramiz
>>> print ( “Satrni faylga yozamiz” )
>>> sys.stdout = tmp_out                # Standart chiqarishni tiklaymiz
>>> print ( “Standart chiqarishga satrni yozamiz” )
Standart chiqarishga satrni yozamiz
>>> f.close ()                          # Faylni yopamiz
```

Ushbu misolda biz avval tmp_out o’zgaruvchisida standart chiqarishga havolani saqladik. Ushbu o’zgaruvchidan foydalanib, keyinchalik chiqarishni standart oqimga qaytarishingiz mumkin. Print() funksiyasi to’g’ridan-to’g’ri chiqarishni qayta yo’naltirishni qo’llab-quvvatlaydi. Buning uchun odat bo’yicha standart chiqarish oqimiga taalluqli fayl parametridan foydalanish kerak.

Masalan, faylga satrni quyidagicha yozish mumkin:

```
>>> f = open (r “ file.txt”, “a”)
>>> print ( “Satrni faylga yozamiz” , file=f)
>>> f.close ( )
```

Python 3.3. da qo’llab-quvvatlangan flush parametri ma’lumotlarni qidiruv buferdan faylga qachon to’g’ridan-to’g’ri saqlash kerakligini belgilashga imkon beradi. Agar uning qiymati False bo’lsa (odatga ko’ra), saqlash faqat faylni yopgandan so’ng yoki flush () usulini chaqirgandan so’ng amalga oshiriladi. Belgilash uchun Python interpretatori har safar print() funksiyasiga murojaat qilganda, saqlashni amalga oshiradi va ushbu parametrni True ga o’tkazadi.

Masalan:

```
>>> f = open (r” file.txt “ , “a” )
>>> print (“Satrni faylga yozamiz” , file = f, flush = True)
>>> print (“Boshqa bir satrni faylga yozamiz” , file = f, flush = True )
>>> f.close()
```

stdin standart kiritish buyrug’ini ham qayta yo’naltirish mumkin. Bunday holatda input() funksiyasi har bir murojaatda fayldan bitta qatorni o’qiydi. Fayl yakunlangandan so’ng EOFError istisnosini qaytaradi. Masalan, kiritish oqimini qayta yo’naltirish usuli yordamida fayl tarkibini ko’rib chiqaylik:

Faylni kiritishni qayta yo’naltirish:

```
# -*- coding: utf-8 -*
import sys                          # sys modulini bog’laymiz
tmp_in = sys.stdin                  # Havolani sys.stdinga saqlaymiz
f = open (r” file.txt “, “r”)       # Faylni o’qish uchun ochamiz
sys.stdin = f                       # Kiritishni qayta yo’naltiramiz
while True :
    try :
        line = input ( )            # Fayldan satrlarni sanaymiz
```

```
            print (line)          # Satrlarni chiqaramiz
        except EOFError :         # Agar fayl oxiriga yetsa
            break                 # Sikldan chiqib ketish bajariladi
    sys.stdin = tmp_in            # Standart kiritishni qayta tiklaymiz
    f.close()                     # Faylni yopamiz
    input()
```

Agar standart kiritish terminalga tegishli yoki yo'qligini bilishingiz kerak bo'lsa, siz isatty() metodidan foydalanishingiz mumkin bo'ladi. Agar obyekt terminalga murojaat qilayotgan bo'lsa metod **True** qiymatni aks holda **False** qiymatni qaytaradi.

Masalan:

```
>>> tmp_in = sys.stdin              # Havolani sys.stdinga saqlaymiz
>>> f = open (r " file.txt", "r")
>>> sys.stdin = f                   # Kiritishni qayta yo'naltiramiz
>>> sys.stdin.isatty ( )            # Terminalga murojaat qilinmaydi
False
>>> sys.stdin = tmp_in              # Standart kiritishni qayta tiklaymiz
>>> sys.stdin.isatty ( )            # Terminalga murojaat qilinadi
True
>>> f.close ( )                     # Faylni yopamiz
```

Shuningdek, standart kiritish/chiqarishni buyruqlar satri (командная строка) yordamida qayta yo'naltirishingiz mumkin. Masalan, C: \book papkasida tests.py faylini buyruqlar satri orqali kod bilan yarataylik. Deylik, tests.py faylining tarkibi quyidagicha bo'lsin:

```
# -*- coding : utf-8 -*
```

```
while True :
    try :
        line = input ()
        print (line)
    except EOFError:
        break
```

Biz buyruq satrini ishga tushiramiz va cd c:\book skripti orqali papkaga o’tamiz. Endi buyruqni bajarib, avval yaratilgan file.txt matnli faylining tarkibini namoyish etamiz (uning tarkibi har qanday bo’lishi mumkin):

```
C:\Python34\python.exe tests.py < file.txt
```

Siz shunga o’xshash tarzda standart chiqishni faylga yo’naltirishingiz mumkin. Faqat bu holatda < belgisi > belgisi bilan almashtirilishi kerak. Tests.py faylini quyidagicha o’zgartiramiz:

```
# -*- coding : utf-8
print ( “ String” )                    # Ushbu satr faylga yoziladi
```

Endi buyruqni bajarish orqali chiqarishni file.txt-ga yo’naltiramiz:

```
C:\Python34\python.exe tests.py > file.txt
```

Ushbu rejimda file.txt fayli ustiga yoziladi. Agar natijani fayl oxiriga qo’shishingiz kerak bo’lsa, >> belgilaridan foydalanish kerak. Masalan:

```
C:\Python34\python.exe tests.py >> file.txt
```

Stdoutdan foydalanib, to’g’ridan-to’g’ri konsol oynasida jarayonni bajarilish indikatorini yaratishingiz mumkin bo’ladi.

Muhokama uchun savollar va topshiriqlar

1. Pythonda fayllarni ochishda “mode” nimani belgilab beradi?
2. Windows OT da faylga absolut yo’lni belgilash qanday amalga oshiriladi?
3. Faylga yo’lni absolut va nisbiy belgilash orasidagi farqlar nima?
4. Yangi faylga yozishni tushuntirib bering
5. Fayllar bilan ishlashda qanday xatoliklar kuzatilishi mumkin?
6. Faylni qatorma-qator o’qish qanday?
7. Pythonda qaysi formatdagi fayllar bilan ishlash mumkin?
8. JSON formatida ma’lumotlar qanday ko’rinishda saqlanadi?
9. Fayllar bilan ishlashda xatoliklarni oldini olish uchun nima qilish kerak?
10. Fayllarga qayta yozishga bir necha misollar keltiring?

FOYDALANILGAN ADABIYOTLAR

1. Бейдер Д. Чистый Python. Тонкости программирования для профи. 2018 г.
2. Воган Л.Непрактичный Python: занимательные проекты для тех, кто хочет поумнеть. 2021 г.
3. Седер Наоми. Python. Экспресс-курс. 2019 г.
4. Райан Митчелл Современный скрапинг веб-сайтов с помощью Python. 2021
5. R. H. Ayupov. PYTHON dasturlash tili asoslari. O’quv-uslubiy qo‘llanma. Nizomiy nomidagi TDPU nashriyoti, 2020, 118 bet.
6. Н. А. Прохоренок, В. А. Дронов. Python 3 и PyQt 5. Разработка приложений. — СПб.: БХВ-Петербург, 2016. — 832 с.: ил.

MAVZULAR BO’YICHA TESTLAR TO’PLAMI

1-dars. Python dasturlash tili bilan tanishish.

Python 3.0 versiayasi qaysi yilda chiqarilgan?

a) 2008
b) 2004
c) 1996
d) 2005

Python 1.0 versiayasi qaysi yilda chiqarilgan?

a) 1990
b) 2008
c) 1996
d) 2005

Python yaratuvchisi kim?

a) Gvido
b) Mante Python
c) Tim Berners Li
d) Yan Guk

Pythonga hos bo’lmagan hususiyat qaysi?

a) Komplyatsiya qilinadi.
b) Ko'plab platformalarda hech qanday o'zgartirishlarsiz ishlay oladi.
c) Dasturni yozish davomida quyi darajadagi detallarni, misol uchun xotirani boshqarishni hisobga olish shart emas.
d) Interpretatsiya (Интерпретируемый) qilinadigan til.

2-dars: Dastur tuzilishi. Izoxlar. Dastur natijasini chop etish. Ma’lumotlarni kiritish

Pythonda bloklarni belgilashda gulli (figurali) qavslardan foydalanish mumkinmi?

a) Mumkin, lekin tavsiya qilinmaydi
b) Ha, mumkin
c) Mumkin emas

d) Bazan mumkin, bazan mumkin emas

Pythonda bir qatorli izoh to’g’ri berilgan javobni aniqlang.

a) # bu izox;
b) % bu izox;
c) * bu izox;
d) $ bu izox;

Pythonda dastur natijasini chop etish uchun qaysi operator ishlatiladi?

a) print()
b) cout()
c) echo()
d) writeln()

Pythonda ma’lumotlarni kiritish uchun qaysi operator ishlatiladi?

a) input ()
b) print()
c) cin()
d) edit()

3-dars. O’zgaruvchilar. O’zgaruvchini nomlash

Pythonda o’zgaruvchilar qaysi kalit so’z bilan e’lon qilinadi?

a) Pythonda o’zgaruvchi e’lon qilinmaydi
b) var
c) let
d) input

Pythonda mantiqiy amal natijasi qanday boladi?

a) True yoki False(1 yoki 0)
b) Begin yoki end
c) int
d) float

Pythonda suzuvchi vergulli son turi qaysi so’z bilan ifoda qilinadi?

a) float
b) decimal
c) int
d) bool

Pythonda ro’yxat turi qaysi so’z bilan ifoda qilinadi?

a) list
b) dict
c) tuple
d) range

Pythonda lug’at turi qaysi so’z bilan ifoda qilinadi?

a) dict
b) list
c) tuple
d) range

Pythonda o’zgaruvchida qanday qiymat saqlanadi?

a) Obyektga havola
b) Son
c) Yacheyka
d) Satr

4-dars: Ma’lumot tipini aniqlash. Ma’lumot tipini o’zgartirish. O’zgaruvchini o’chirish

Pythonda ma’lumotni butun son tipiga o’zgartirish tug’ri ko’rsatilag javobni aniqalang.

a) int(<ma’lumot>)
b) float(ma’lumot)
c) str(ma’lumot)
d) bool(ma’lumot)

Pythonda o’zgaruvchini o’chirish konstruksiyasi qanday bo’ladi?

a) del <o’zgaruvchi>[, …, <o’zgaruvchi_N>]
b) delete <o’zgaruvchi>[, …, <o’zgaruvchi_N>]
c) rd <o’zgaruvchi>[, …, <o’zgaruvchi_N>]
d) clear <o’zgaruvchi>[, …, <o’zgaruvchi_N>]

5-dars: Operatorlar. Matematik operatorlar.

Pythonda “ / / ” amali qanaday ma’no anglatadi?

a) kami bilan yaxlitlanadigan bo’lish
b) qoldiqli bo’lish
c) darajaga ko’tarish

d) bo’lish

Pythonda “ % ” amali qanaday ma’no anglatadi?

a) qoldiqli bo’lish
b) kami bilan yaxlitlanadigan bo’lish
c) darajaga ko’tarish
d) bo’lish

Quyidagilardan qaysi biri o’z;ashtirish amali bo’lolmaydi?

a) **
b) +=
c) -=
d) /=

Mantiqiy (ikkilik) VA amali qaysi belgi bilan abelgilanadi?

a) &
b) |
c) ^
d) <<

6-dars: Shartli operatorlar. Taqqoslash operatorlari. if...else operatori

Pythonda >>> True + 2 amal natijasi nimaga teng?

a) 3
b) “True+2”
c) 2
d) 0

Pythonda bool (20) funksiya qanday qiymat qaytaradi?

a) True
b) 0
c) False
d) 20

Pythonda bool (“0”) funksiya qanday qiymat qaytaradi?

a) True
b) False
c) 0
d) 20

Pythonda qaysi operator ikkita o’zgaruchi bir obyektga havola qilinganini tekshiradi?

a) is
b) in
c) if
d) not

Pythonda qaysi operator qiymatni qator yoki ketma-ketlikka tegishlilikka tekshiradi?

a) in
b) is
c) if
d) not

Pythonda “VA” mantiqiy amalai qaysi so’z bilan ifoda qilinadi?

a) AND
b) OR
c) NOT
d) IF

Pythonda “YOKI” mantiqiy amalai qaysi so’z bilan ifoda qilinadi?

a) OR
b) AND
c) NOT
d) IF

7-dars. Sikl operatorlari. For sikli. range() va enumerate() funksiyalari

Javoblar orasidan qaysi operator Pythonda takrorlanishni ta’minlaydi?

a) while
b) if
c) switch
d) numerate

Pythonda range funksiyasi formati to’g’ri berilgan javobni belgilang?

a) range ([<boshlang’ich qiymat>,] <Tugash> [, <Qadam>])
b) range ([<boshlang’ich qiymat>,] [<Qadam>], <Tugash>)
c) range (<Qadam>)
d) range ([<Qadam>,] [<boshlang’ich qiymat>,] <Tugash>)

Pythonda range() funksiyasi qanday vaziafa bajaradi?

a) Ketma-ketlikni ifodalaydi
b) Sonlarni yigindisini qayataradi
c) Darajaga ko’taradi
d) Kvadrat ildiz chiqaradi

Pythonda for operatori qanaday maqsadda qo’llaniladi?

a) takrorlanishlarni hisoblash va ketma-ketliklar bilan ishlash uchun foydalaniladi;
b) tarmoqlanishlarni amaliga oshirishda va sonlar bilan ishlashda qo’llaniladi;
c) murakkab ifodalarni yozishda foydalaniladi;
d) ketma-ketliklarni ifodalsh uchun va tarmoqlarga ajratsish uchun ishlatiladi;

Pythonda ushbu dastur natijasi nimaga teng?

```
s=0
for i in range(1,10,2):
    s=s+i
print(s)
```

a) 25
b) 10
c) 45
d) 0

8-dars. While sikli. continue operatori. break operatori

Pythonda qaysi operator sikldagi barcha instruksiyalar bajarilmay, siklning navbatdagi iterasiyasiga o’tish imkonini beradi

a) continue
b) break
c) for
d) while

Pythonda qaysi operator sikl bajarilishini zudlik bilan to’xtatish imkonini beradi

a) break
b) continue
c) for
d) while

Pythonda while operatori qanday maqsadda qo’llaniladi?

a) takrorlanishlarni tashkil etish uchun
b) tarmoqlanishi tashkil etish uchun
c) ichma-ich takrorlanishni tashkil etish uchun
d) shartsiz takrorlanishlarni tashkil etish uchun

Pythonda ushbu dastur natijasi nimaga teng?

```
i=1
s=1
while i<10:
    s+=i
    i+=1
print(s)
```

a) 46
b) 45
c) 1
d) 0

Pythonda ushbu dastur natijasi nimaga teng?

```
i=1
s=0
while i<10:
  if i==5: break
  s+=i
  i+=1
print(s)
```

a) 10
b) 45
c) 46
d) 0

9-dars. Sonlar. Sonlar bilan ishlashning tashqi funksiya va metodlari.

Python 3 butun son tipi qaysi so’z bilan belgilanadi?

a) int
b) float
c) decimal
d) complex

Python 3 kasr sonni qanday ifodalash mumkin?

a) Fraction() funksiyasi yordamida
b) complex() funksiyasi yordamida
c) float xizmatchi so’zi yordamida
d) ifodalab bo’lmaydi

Pythonda kompleks son yozilish formati to’g’ri berilgan javobni tping?

a) <Haqiyqiy qism> + <Mavhum qism>J
b) <Haqiyqiy qism>J + <Mavhum qism>
c) <Haqiyqiy qism>+J + <Mavhum qism>
d) <Haqiyqiy qism>+ <Mavhum qism>+J

Pythonda o’nlik sonni ikkilik sanoq sistemasiga o’tkazish uchun qayis funksiya ishlatiladi?

a) bin (<Son>)
b) oct (<Son>)
c) hex (<Son>)
d) abs (<Son>)

Pythonda o’nlik sonni sakkizlik sanoq sistemasiga o’tkazish uchun qayis funksiya ishlatiladi?

a) oct (<Son>)
b) bin (<Son>)
c) hex (<Son>)
d) abs (<Son>)

Pythonda o’nlik sonni o’n oltilik sanoq sistemasiga o’tkazish uchun qayis funksiya ishlatiladi?

a) hex (<Son>)
b) oct (<Son>)
c) bin (<Son>)
d) abs (<Son>)

Pythonda divmod (x, y) funksiyasi qanday amal bajkaradi?

a) Sonning butun va qoldiq qismini qaytaradi

b) Sonning butun qismini qaytaradi
c) Sonning qoldiq qismini qaytaradi
d) Sonni yaxlitlaydi

Pythonda sonni radiandan gradusga o'tkazish uchunqaysi funksiya qo'llaniladi?

a) degrees()
b) radians()
c) exp()
d) ceil()

Pythonda sonni eng yaqin katta songa yaxlitlash funksiyasi qaysi?

a) ceil ()
b) floor ()
c) fabs ()
d) round ()

10-dars. math moduli. Matematik funksiyalar. random moduli. Tasodifiy son generasiyasi.

Pythonda random() funksiyasining vazasi nima?

a) dan 1.0 gacha tasodifiy sonlarni qaytaradi
b) Tasodifiy sonlar diapazonini aniqlaydi
c) Tasodifiy sonlar modulini chaqirib beradi
d) 1 dan 10 gacha tasodifiy sonlarni qaytaradi}

Pythonda uniform () funksiyasining vazasi nima?

a) Ko'rsatilgan diapazondagi haqiyqiy tasodifiy sonlar qaytaradi
b) dan 1.0 gacha tasodifiy sonni qaytaradi
c) Tasodifiy sonlar modulini chaqirib beradi
d) 1 dan 10 gacha tasodifiy sonlarni qaytaradi

Pythonda randint () funksiyasining vazasi nima?

a) Ko'rsatilgan diapazondagi butun tasodifiy sonni qaytaradi
b) 0.0 dan 1.0 gacha tasodifiy sonni qaytaradi
c) Tasodifiy sonlar modulini chaqirib beradi
d) 1 dan 10 gacha tasodifiy sonlarni qaytaradi

Pythonda qaysi funksiya berilgan ketma-ketlik (satr, ro'yxat, kortej) dagi tasodifiy elementni qaytaradi?

a) choice (<Ketma-ketlik>)
b) shuffle (<Ro'yxat> [, <0.0 dan 1.0 gacha sonlar>])
c) sample (<Ketma-ketlik>, <Elementlar soni>)
d) randrange ([<Boshlanish>,] <Tugash> [, <Qadam>])

11-dars. Satrlar va ular ustida amallar. Satr yaratish. Maxusu belgilar. Satrlar bilan ishlash amallari. Satrlarni formatlash. format() metodi.

Joriy sana vaqtning xar hil formatlangan natijalarini chiqarishga imkon beruvchi modul qaysi?

a) datetime
b) timelimit
c) timeit
d) calendar

Qaysi funksiya satr formatiga muvofiq sana satrini qaytaradi

a) strftime
b) stringdate
c) stringtime
d) struct_time

Datetime moduli nechta sinfdan iborat bo'ladi?

a) 5 ta
b) 3 ta
c) 4 ta
d) 6 ta

Vaqtni ob'ekt sifatida aks ettirish datetime modulining qaysi sinfida amalga oshiriladi?

a) time
b) timedelta
c) tzinfo
d) timelimit

Qaysi funktsiya belgilangan vaqt davomida skriptning bajarilishini to'xtatadi?

a) sleep
b) turn_off
c) stoptime
d) localtime

12-dars. Funksiyai aniqlanishi va uni chaqirish, Fiunksiyani aniqlanishi joylashuvi

Dasturda yozilgan funksiyalar nima uchun xizmat qiladi?

a) dasturni ixchamlashtiradi va ishlash tezligini oshiradi
b) dasturni murakkabligini ta’minlaydi
c) dasturni mukammalligini ta’minlaydi
d) dasturni tezlashtirib, natijani formatlaydi

Python dasturlash tilida funksiya qaysi kalit so’z yordamida hosil qilinadi?

a) def
b) function
c) functional
d) steff

Funksiya natijasini qaytarish qaysi kalit so’z orqali amalga oshiriladi?

a) return
b) restart
c) rework
d) label

Parametrsiz funksiyani chaqirish qanday amalga oshiriladi?

a) def print_ok():
   print ("Amalni muvaffaqiyatli bajarilishi haqida xabar")
b) def echo (m) :
   print (m)
c) def summa (x, y) :
   return x + y
d) def unparametric(alpha):
   print()

Funksiya yaratishda parametrlar bir nechta bo’lsa ular bir-biridan qanday ajratib ko’rsatiladi?

a) Vergul
b) Ikki nuqta
c) Nuqtali vergul
d) Yotiq chiqiz (slesh)

Jakbarov Odiljon Otamirzaevich

Namangan muhandislik - qurilish institutining Informatika va AT kafedrasi mudiri, texnika fanlari nomzodi, dotsent. Matematik modellashtirish, Axborot texnologiyalari, Dasturlash hamda Web dasturlash bo`yicha mutaxassis. 2 ta o`quv qo`llanma, 70 dan ortiq ilmiy – uslubiy ishlar muallifi, dasturiy mahsulotlar yaratish bo`yicha 6 dan ortiq guvohnomalari mavjud. 12 dan ortiq Kasb ta`limi (Informatika va AT) mutaxassislik bo`yicha magistrlik dissertasiyalariga rahbarlik qilgan.

Goyipov Umidjon Gulomjonovich

Namangan muhandislik - qurilish institutining Informatika va AT kafedrasi o'qituvchisi. Axborotlarga ishlov berishni algoritmlash hamda Web dasturlash bo`yicha mutaxassis. 30 dan ortiq ilmiy teziz va maqolalar, 10 dan ortiq uslubiy ishlar muallifi. Dasturiy mahsulotlar yaratish bo`yicha 6 dan ortiq guvohnomalari mavjud.

Akbarov Bahriddin Xusniddin o'g'li

Namangan muhandislik-qurilish institutining "Informatika va AT" kafedrasi o'qituvchisi. Algoritmlash asoslari, Tizimlar nazariyasi asoslari hamda Python dasturlash tili bo'yicha mutaxasis. 10 dan ortiq ilmiy tezis va maqolalar, 5 dan ortiq uslubiy ishlar muallifi.

Terishga __.__.20__ yilda berildi. Bosishga __.__.20__ yilda ruxsat etildi. Shrift garniturasi Times New Roman Uzb. Oq qog’ozga ofset usulida bosildi. Bichimi 60x84 1/16. Shartli bosma tabog’i 8.5. Nashriyot bosma tabog’i 8.5 Adadi 200 nusxa.

Buyurtma №___. Bahosi kelishilgan narxda. «Namangan» nashriyoti. (Namangan shahri, Navoiy ko‘chasi,36.) Nashriyot lisenziya raqami AN-156

2009 yil 14 avgustda berilgan.