PYTHON STATEMENTS

(BAYONNOMALAR)

MANTIQ ELEMENTLARI VA OPERATORLARI

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

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

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

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

```
a = 100
b = 30

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

Qiymatlarni tekshirish

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

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

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

Funksiyada mantiq elementlari

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

```
def myfunction():
    return True

print(myFunktion())
```

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

```
def myfunc():
    return False

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

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

```
x = 300

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

Mantiq operatorlari

Mantiq operatorlar shartlarni birlashtirib ishlatish uchun kerak:

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

```
a = 5

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

```
True
True
False
```

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

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

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

PYTHONDA SHART OPERATORLARI

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

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

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

IF

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

```
a = 50
b = 30

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

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

```
a = 50
b = 30

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

else

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

```
a = 50
b = 90

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

elif

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

```
a = 50
b = 30

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

Pass

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

```
a = 33
b = 99

if b>a:
    pass

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

PYTHONDA SIKLLAR

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

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

WHILE SIKLI

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

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

While sikl operatorining ishlash tartibi

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

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

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

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

break

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

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

continue

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

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

else

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

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

FOR SIKLI

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

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

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

Satr bo'ylab sikl

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

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

break

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

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

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

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

continue

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

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

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

range() va xrange()

range() funksiyasi biror amalni belgilangan marta bajarish yoki biror oraliqdagi sonlarga murojaat qilsh uchun qo'llaniladi. Bunda **range()** ichiga kerakli son qo'yiladi va sanoq avtomatik tarzda o dan

boshlanib ko’rsatilgan songacha davom etadi. Ammo uning o’zi hisobga kirmaydi.

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

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

```
0
1
2
3
4
```

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

Masalan 1 dan 5 gacha bo’lgan sonlarni ekranga chiqaramiz. Bunda sanoq 1 dan boshlanishi uchun1 sonini ham kiritamiz. Demak, biz 1 dan 6 gacha bo’lgan oraliqni kiritamiz:

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

```
1
2
3
4
5
```

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

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

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

```
2
4
6
8
10
```

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

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

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

else

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

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

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

Sikl ichida sikl

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

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

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

pass

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

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