Python Programlama

Ders 3

Ali Mertcan KOSE Msc.
amertcankose@ticaret.edu.tr

İstanbul Ticaret Üniversitesi

İterasyon/Yİneleme

Tekrarlama ifadeleri, bir ifadeyi birden çok kez yürütmemize izin verir.

Sıklıkla, bu işlem **loops(döngüler)** olarak adlandırılır.

Koşullu ifadeler gibi, Boolean ifadeleri tarafından denetlenebilir.

Python'un **while** ve **for** olmak üzere iki tür tekrarlama ifadesi vardır.

While ifadesi

while< *condition* >:

< *statment* >

< *statement* >

. . .

< *condition* >: bir boolean veya değeridir.

< *condition* >: doğruysa, while ifadesi altında girintili adımları yürütün.

< *condition* >: ifadeleriRepeat steps until is False yürütüldükten sonra, while tekrar değerlendirilir.

< *condition* >: yanlış olana kadar adımları tekrarlayın.

Not:

Koşullu ifadeler gibi, döngülerdeki kod bloklarının da girinti ile gösterildiğini unutmayın

While loops mantığı

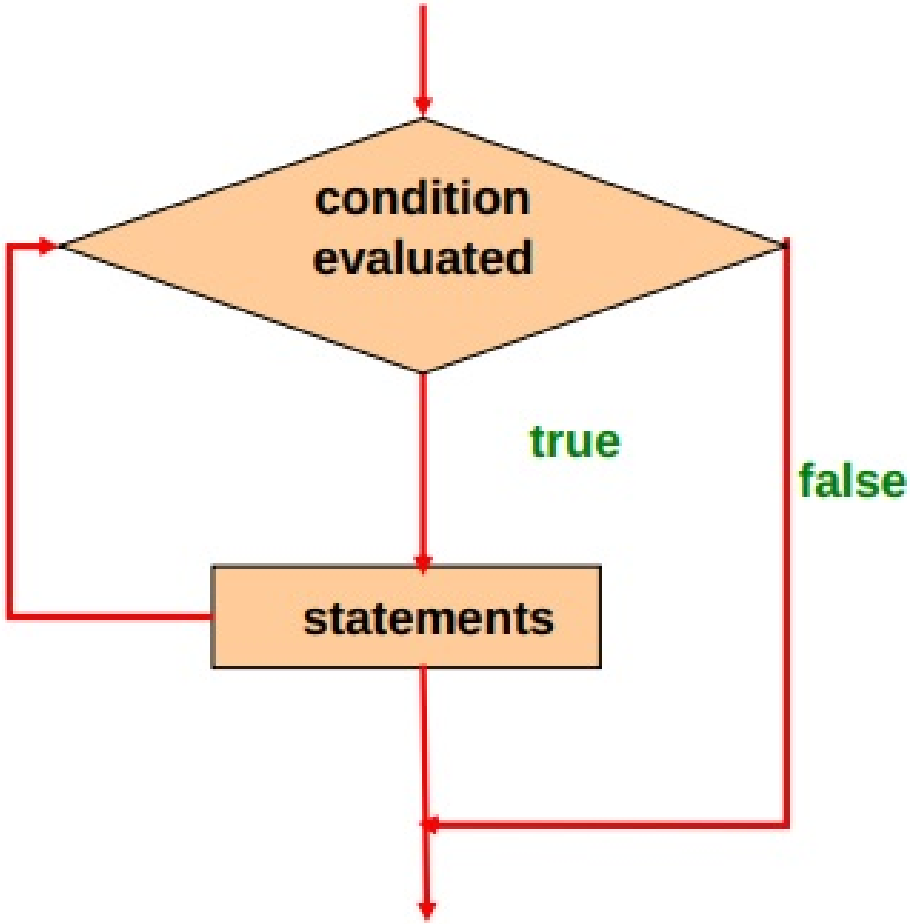

Figure 1: While-loop mantığı

Random Paketi

Python paketi, kendi kodunuzu yazmadan birçok eylem gerçekleştirmenize olarak tanıyan bir işlev ve yöntem koleksiyonudur.

Python, *random* paketinde rastgele değerler üretmek için işlevsellik içerir.

Pakeyye tanımlanan işlevlere erişmek için, önce içeren paketi içe aktarmanız gerekir.

Random Paketi

Syntax:

- import random

num=random.randint(1,10)

VEYA

- from random import * num = randint (1,10)

randint işlevi, iki değer arasında rastgele bir değer oluşturur (dahil).

0 ile 1 (hariç) arasında rastgele bir float değeri oluşturmak için random () fonksiyonu kullanılabilir.

For Loops

Python'daki başka bir döngü türü, kesin tekrarlamalar için uygun olan for döngüsüdür.

For döngüsü ile yazılabilecek her şey while döngüsü ile de yazılabilir, ancak while döngüleri, for döngülerinin kolayca çözemediği bazı sorunları çözebilir.

Genel olarak; yapmanız gereken yineleme sayısını biliyorsanız for döngülerini kullanın- örneğin, 500 deneme için.

Mümkün olduğunda bir for döngüsü kullanmak, sonsuz döngü yazma riskini azaltır ve genellikle artan sayaç değişkenleriyle hatalarla karşılaşmanızı önler.

Örnek: 1 den 10 kadar sayıların karelerini yazdıralım.

```
for i in range (1,11): print (i**2)
```

For Loops

for$<$ *condition* $>$: in range():

$<$ *statment* $>$

$<$ *statement* $>$

...

Döngü boyunca $<$ *variable* $>$ her seferinde bir değer alır.

ilk iterasyon, $<$ *variable* $>$ en küçük değerde başlar.

Bir sonraki iterasyon, $<$ *variable* $>$ bir önceki değişken değerine eklenerek $(+1)$ değişkenin değeri elde edilir.

some_num-1 e ulușıldığında döngü sona erer.

range (start,stop,step)

range fonksiyonu, genellikle for döngüleriyle yinelmek için kullanılan bir sayı listesi oluşturur.

Bu fonksiyonun 3 parametresi vardır.

- `start` : sıra numarasını başlatır.
- `stop` : Durma değerine kadar olan ancak durdurma değeri dahil olmayan sayılar üretir.
- `step` : Dizideki her sayı arasındaki güncelleme/farktır.

default değerler `start` $= 0$ ve `step` $= 1$, döngü `stop` $- 1$ (dahil) değerine kadardır.

For Loops

```
mysum = 0
for i in range (5,11,2):
mysum +=i
print(mysum)
```

Output: 21

```
mysum = 0
for i in range (7,10):
mysum += i
print (mysum)
```

Output: 24

Örnek

```
x= 4

for i in range(x):

print(i)
```

Output

```
0

1

2

3
```

Örnek

```
x=4

for i in range(0,x):

print(i)
```

Output

0

1

2

3

Örnek (string)

```
for ch in 'abcdf':
print (ch)
```

Output

```
a
b
c
d
e
f
```

Örnek (string)

```
letters =‘abcdef’

for i in range (0,len(letters)):

print(letters[i])
```

Output

a

b

c

d

e

f

Örnek (string)

Aşağıdaki kod segmentleri de aynı şeyleri yapar:

ikincisi daha çok *"pythonic"* olarak ifade edilir.

s="abcdef"

for index in range(len(s)):

if s[index] == 'i' or s[index] == 'u':

```
print ("There is an i or u")
```

for char in s:

if char =='i' or char == 'u':

```
print("There is an i or u")
```

break İfadesi

break ifadesi, içinde bulunduğu her döngüden hemen çıkmak için kullanılır.

break, kalan ifadeleri kod bloğunda atlar.

Sadece döngü içeren kırılma çıkışlarıdır!

while

$< statement >$

break

$< statement >$

break İfadesi Örneği

```
mysum=0
for i in range (5,11,2):
mysum +=i
if mysum == 5:
break
mysum +=1
print(mysum)
```

For döngüsü örneği

Durumu (F-Full time, P-Part time) ve 10 eğitmenin maaşını girmek içi nbir Python programını yazınız.

- Full time eğitmen sayısı
- bütün eğitmenlerin ortalama maaşı

12 derece sıcaklık değeri (her ay için bir tane) girmek ve en yüksek sıcaklığa sahip ayın sayısını görüntülemek için bir Python programı yazınız.

Bir Python tahmin oyunu programı yazın. Program 1 ile 10 arasında rastegele bir int üretmelidiri. Kullanıcın doğru tahmin etmesi için 3 tahmini vardır. Kullanıcı doğru tahmin ederse doğru yoksa yanlış olarak mesaj versin.

For vs While Döngüsü

`for` döngüsü:

- iterasyonların sayısı biliniyor
- break ile erken den sona erebilir.
- bir sayaç kullanılır.
- while döngüsü kullanaraj for döngüsü yeniden yazılabilir.

For vs While Döngüsü

`while` döngüsü:

- sınırsız yineleme sayısı
- break ile erken den sona erebilir.
- bir sayaç kullanabilir, ancak döngüden önce başlatılması ve döngü içinde artırılması içinde artma işlemi gerekir.
- for döngüsü kullanarak bir bir while döngüsünü yeniden yazabiliriz.

İç İçe Döngüler

İç içe döngü, başka bir döngü içindeki bir döngü anlamına gelir.

Dış döngünün her yinelemesi yürütülürse, iç döngü tamamen yürütülür.

for i in range (1,6,2):

for j in range (6,0,-3):

```
print ("i=" +str(i)+"\tj="+str(j))
```

İç İçe Döngüler

Output

i=1 j=6

i=1 j=3

i=3 j=6

i=3 j=3

i=5 j=6

i=5 j=3

Yıldız Üçgeni Yazdıran Program

```
MAX_ROWS = 10;
for row in range (1,MAX_ROWS+1,1):
for star in range in range (1,row+1,1):
print ("*",end=' ')
print ()
```

Output

```
*
**
* * *
* * **
* * * * *
* * * * **
* * * * * * **
* * * * * * * * **
* * * * * * * * * * **
* * * * * * * * * * * *
* * * * * * * * * * * * **
```

İç İçe For Döngüsü

```
                Multiplication Table
          1    2    3    4    5    6    7    8    9
------------------------------------------------------------
1 |       1    2    3    4    5    6    7    8    9
2 |       2    4    6    8   10   12   14   16   18
3 |       3    6    9   12   15   18   21   24   27
4 |       4    8   12   16   20   24   28   32   36
5 |       5   10   15   20   25   30   35   40   45
6 |       6   12   18   24   30   36   42   48   54
7 |       7   14   21   28   35   42   49   56   63
8 |       8   16   24   32   40   48   56   64   72
9 |       9   18   27   36   45   54   63   72   81
```

Figure 2: İç içe For Döngüsü

İç İçe For Döngüsü - Stars

Aşağıdaki yıldız üçgenini görüntülemek için bir program yazın.

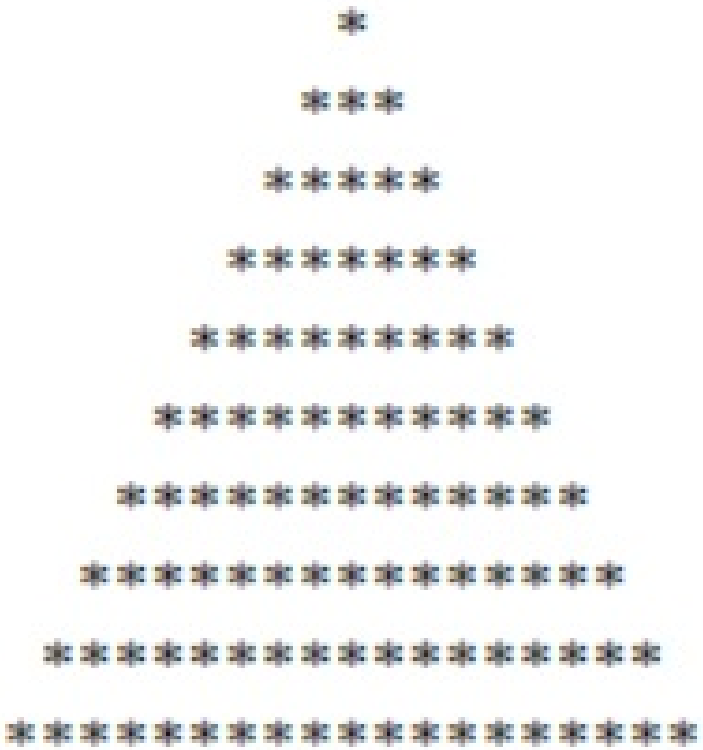

Figure 3: Stars

Örnek

s1 = “Ticaret u JaZZ”

s2 = “i rule Ticaret”

if len(s1) == len(s2):

for char1 in s1:

```
for char2 in s2:

  if char1 == char2:

    print("common letter")

    break
```

Kapsamlı Numaralandırma

Kapsamlı numaralandırma, yalnızca aranan değer kümesi yanıtı içeriyorsa çalışan bir arama tekniğidir.

```
#Find the cube root of a perfect cube
```

Kapsamlı Numaralandırma

```
x = int(input('Enter an integer:'))
ans = 0
while ans**3 < abs(x):
ans = ans + 1
if ans**3 != abs(x):
  print(x, 'is not a perfect cube')
else:
if x < 0:
ans = -ans
print('Cube root of', x, 'is', ans)
```

Program Tekniği

Bu programda, doğru cevabı bulana veya tüm olasılıkların alanını tüketene kadar tüm olasılıkları sıralayınız.

Bu, bir sorunu çözmenin etkili bir yolu gibi görünmeyebilir, ancak kapsamlı numaralandırma algoritmeları genellikle en pratik çözümdür.

Bu tür algoritmaların uygulanması ve anlaşılması kolaydır.

Çoğu durumda, tüm pratik amaçlar için yeterince hızlı çalışabilir.

Tahmin ve Kontrol-cube root

```
cube = 8
for guess in range (abs(cube)+1):
if guess**3 >= abs (cube):
    print (cube, 'is not a perfect cube')

else:
if cube < 0:
guess = -guess
print('Cube of'+ str (cube) + " "+'is' +" "+ str(guess))
```

Uygun Çözümler

- Yeterince iyi bir çözüm
- Bir tahminle başlayın ve küçük bir değer kadar arttırın.
- Küçük bir epsilon sayısı için $|guess^3 - cube| >= epsilon$ olup olmadığını tahmin edelim.
- Artış boyutu azalır => program yavaşlar
- epsilon artar => daha az doğru yanıt olur.

Uygun Çözümler - cube root

cube = 27

epsilon = 0.01

guess = 0.0

increment = 0.0001

num_guesses = 0

Bisection araması- cube root

```
while abs(guess**3 - cube) >= epsilon and guess <= cube:
guess+= increment
num_guesses +=1
print('num_guesses =', num_guesses)
if abs(guess**3-cube) >= epsilon:
print('Failed on cube root of', cube)
else:
print(guess, 'is close to the cube root of', cube)
```

Bisection araması

- Her yinelemede yarım aralık
- Yeni tahmin yarısının arasında
- Açıklmak için, kodla yazalım

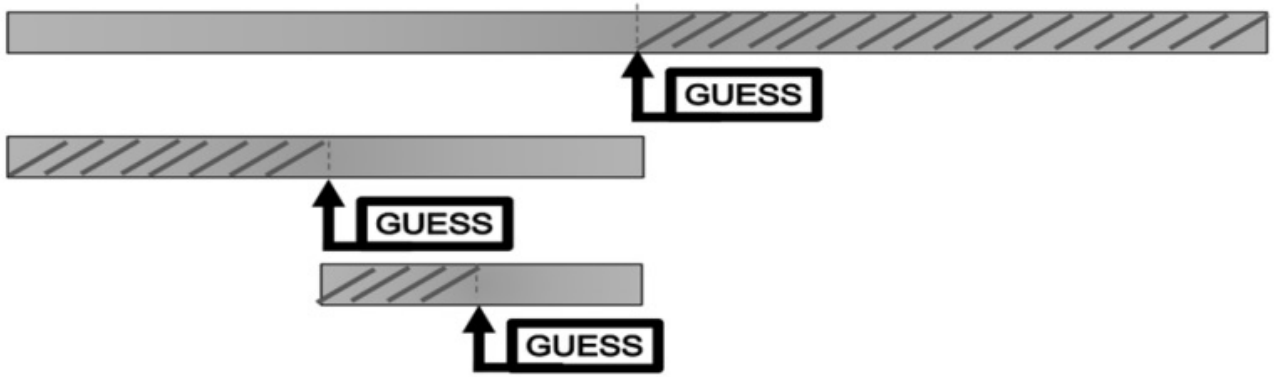

Figure 4: Stars

Bisection araması- cube root

cube= 27

epsilon = 0.01

num_guesses = 0

low = 0

high = cube

guess = (high + low)/2.0

Bisection araması- cube root

while abs (guess**-cube) >= epsilon:

if guess**3 < cube :

```
low= guess
```

else:

```
high = guess
```

guess = (high+low)/2.0 num_guesses +=1

print('num_guesses=',num_guesses)

print(guess,'is close to the cube root of', cube)

Bisection araması- Program Tekniği

Programdaki döngünün her yinelemesinde, aranacak alanın boyutu yarıya indirilir.

Her adımda arama alanını ikiye böldüğü için buna **Bisection araması** denir.

Bisection araması kapsamlı numaralandırmaya karşı büyük bir gelişmedir, çünkü kapsamlı numaralandırma arama alanını her yinelemede sadece küçük bir miktar azaltmıştır.

Alıştırmalar

1. Aşağıdakileri yapan bir Python komut dosyasını yazın. Bir banka hesabına 10000 TL yatırdığınızı varsayalım, yıllık %15'lik bir faiz oranı varsayarak orjinal yatırımı ikiye katlamak için gereken yıl sayısını veriniz.
2. Bir biyoloji deneyinde, bir mikroorganizma popülasyonu her 10 saatte bir ikiye katlanıyor. İlk mikroorganizma sayısını girmek ve 1000000'den fazla organizmaya sahip olmanın ne kadar süreceğini (günler ve kalan saatleri) çıkarmak için bir Python programı yazınız.

Alıştırmalar

3. Negatif bir değer girilene kadar pozitif tamsayılar giren bir Python komut dosyası yazın. Program, değer girişinin ortalamasını vermelidir.
4. Bir Python tahmin oyunu programı yazın. Program 1 ile 10 arasında rastgele bir int oluşturmalıdır. Kullanıcı, bulana kadar sayıyı tahmin etmeye çalışacaktır. Program, yapılan tahminlerin sayısını vermelidir.