ITU ACM Student Chapter Course Program

Python Course

Week 3

Instructor
Hüseyin Averbek

Assistants
Serra Bozkurt
Gökalp Akartepe

Week 3

Döngüler

Python'da if-elif-else gibi ifadeleri kullanırken program, yapmasını istediğimiz görevi bir kez yaptıktan sonra sonlanır. Aynı şekilde input-print fonksiyonlarını kullanırken de almak istediğimiz her girdi için ayrı bir input fonksiyonu, yaptığımız her ayrı işlemin çıktısını görmek için de ayrı bir print fonksiyonu kullanılır.

Örneğin verilen 4 ayrı sayının toplamı istendiğinde Python'da aşağıdaki gibi bir kod kullanılabilir:

```
birinci_sayi = int(input("Lütfen 1. sayıyı giriniz: "))
ikinci_sayi = int(input("Lütfen 2. sayıyı giriniz: "))
ucuncu_sayi = int(input("Lütfen 3. sayıyı giriniz: "))
dorduncu_sayi = int(input("Lütfen 4. sayıyı giriniz: "))

print("Sayıların toplamı: ", birinci_sayi + ikinci_sayi + ucuncu_sayi +␣
 ↪dorduncu_sayi)
```

```
Lütfen 1. sayıyı giriniz: 1
Lütfen 2. sayıyı giriniz: 2
Lütfen 3. sayıyı giriniz: 3
Lütfen 4. sayıyı giriniz: 4
Sayıların toplamı:  10
```

Buradaki örnekte yapılacak işlem sayısı belli ve azdır. Ancak yapılacak işlem sayısı arttıkça (örneğin 100 sayı verilip toplamı istendiğinde) aynı işlemleri bu şekilde yapmak programcı için vakit alır.

Veya örneğin kullanıcıdan tuttuğumuz sayıyı tahmin etmesini isteyen bir program yazalım:

```
tutulan_sayi = 5
tahmin = int(input("Lütfen 1 ile 10 arasında bir sayı giriniz: "))


if tahmin == tutulan_sayi:
    print("Tebrikler, doğru tahmin!")
else:
    print("Üzgünüz, tekrar deneyin!")
```

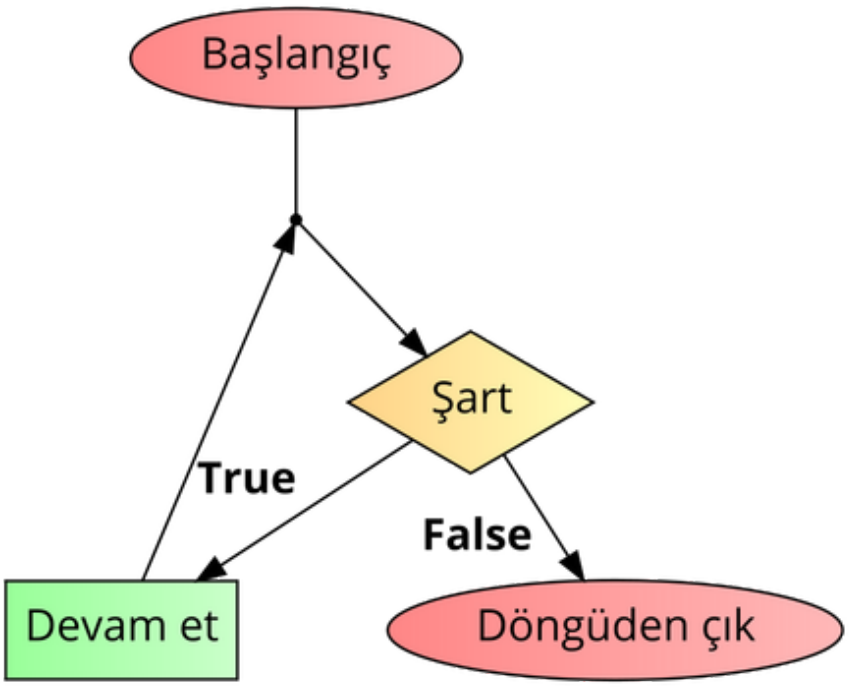
```
Lütfen 1 ile 10 arasında bir sayı giriniz: 3
Üzgünüz, tekrar deneyin!
```

Burada ise kullanıcı doğru tahmini girerse program tebrik mesajı verecek fakat yanlış girdiğinde ikinci bir tahmin hakkı olmayacaktır. Program tekrar kullanılmak istenirse baştan başlatılmalıdır. Halbuki program tahmin sonucu doğru gelene kadar çalışabilse baştan başlatmaya gerek kalmaz.

Örneklerde gördüldüğü gibi şimdiye kadar yazdığımız programlar bir defa çalışıyor ve sona eriyordu. Ancak biz çoğu zaman programlarımızın belli koşullarda çalışmasını sürekli devam ettirmesini ve işlemlerini tekrar etmesini isteriz.

Bu gibi belirli bir şarta kadar (sayı veya durum) çalışması istenen programlar yazmak için Python'da döngüler kullanılır.

while Döngüsü



While döngüsü, içine aldığı kod bölümünün tekrarlanmasını sağlar. While döngüsünde kod istenen şart sağlandığı sürece döner. While'ın yapısı:

```
#while (koşul):
      #işlem1
      #İşlem2
      #...
```

şeklindedir. Basit bir örnek olarak:

```
n = 1
toplam = 0

while n < 5:  #döngü; n'in 1, 2, 3 ve 4 değerleri için döner.
    toplam += n
    n += 1

print("1'den 5'e kadar olan rakamların toplamı: "+str(toplam))
```

```
1'den 5'e kadar olan rakamların toplamı: 10
```

Bu örnekte döngünün şartı n değişkeninin 5'ten küçük olmasıdır. İlk olarak döngü şartının sağlanıp sağlanmadığını kontrol edilir ve şart sağlanıyorsa döngünün içine girilir, döngüde n değişkeni önce toplam değişkeniyle toplanır sonrasında da değeri 1 arttırılır ardından başa dönülür ve şart sağlanmayana kadar bu şekilde devam eder.

While döngülerinde sağlanması istenen şarta ve şartı sağlayan değişkenin değerinin değişip değişmediğine dikkat edilmelidir. Bunlardaki bir yanlışlık yüzünden sonsuz döngüye girilebilir veya döngü hiç çalışmayabilir:

```
n = 0

while n < 5:
    print(n)
```

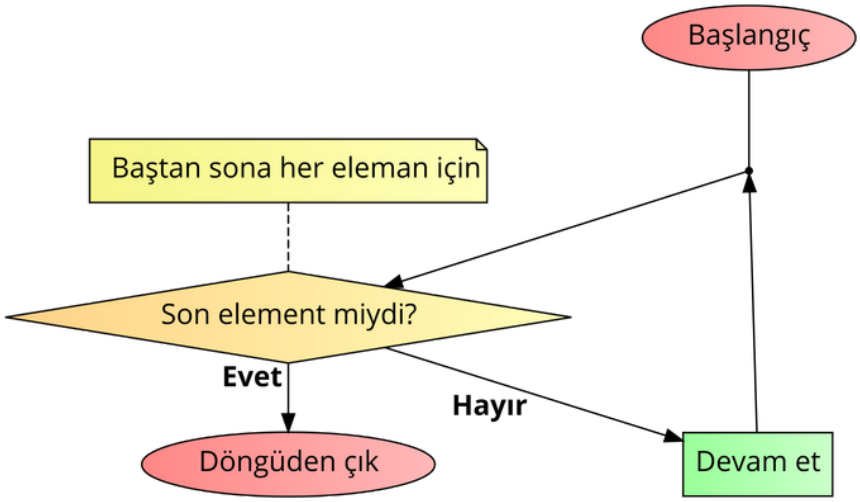
```
tutulan_sayi = 5
tahmin = int(input("Lütfen 1 ile 10 arasında bir sayı giriniz: "))

while tahmin != tutulan_sayi:
    print("Üzgünüz, tekrar deneyin!")
    tahmin = int(input("Lütfen 1 ile 10 arasında bir sayı giriniz: "))

print("Tebrikler, doğru tahmin!")
```

```
Lütfen 1 ile 10 arasında bir sayı giriniz: 3
Üzgünüz, tekrar deneyin!
Lütfen 1 ile 10 arasında bir sayı giriniz: 7
Üzgünüz, tekrar deneyin!
Lütfen 1 ile 10 arasında bir sayı giriniz: 9
Üzgünüz, tekrar deneyin!
Lütfen 1 ile 10 arasında bir sayı giriniz: 5
Tebrikler, doğru tahmin!
```

for Döngüsü



in operatörü

Pythondaki in operatörü, bir elemanın başka bir listede, demette veya karakter dizilerinde bulunup bulunmadığını kontrol eder ve sonuç olarak Boolean veri tipinde bir değer döndürür:

```
print("o" in "hello")
print(5 in [1,3,5])
```

```
True
True
```

For döngüsü de while döngüsü gibi içindeki kodu döndürür ancak while döngüsünden farkı belli bir şartı sağladığı sürece dönmek yerine listelerin, demetlerin, karakter dizilerinin ve hatta sözlüklerin üzerinde dolaşmamızı sağlar. Yapısı şu şekildedir:

```
club = "ITUACM"

for harf in club:
    print(harf)
```

```
I
T
U
A
C
M
```

```
liste = [4,3,2,1]

for eleman in liste:
    print("Eleman",eleman)
```

```
Eleman 4
Eleman 3
Eleman 2
Eleman 1
```

```
meyveler = ["elma", "armut", "kiraz"]
for meyve in meyveler:
    print(meyve)
```

```
elma
armut
kiraz
```

range() fonksiyonu

range() fonksiyonu; başlangıç, bitiş ve opsiyonel olarak aralığı arttırma değeri alarak bir sayı dizisi oluşturur. Örneğin:

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

```
1
2
3
4
5
```

```
for sayi in range(1,9): # kod 1,2,3,4,5,6,7,8 değerleri için dönecek
    print("* " * sayi)
```

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

```
for x in range(0, 50, 10):   # 0'dan 50'ye kadar 10'ar artarak giden bir dizi␣
 ↪oluşturacak.
    print(x)
```

```
0
10
20
30
40
```

break ve continue

a) break

break ifadesi döngülerde programcılar tarafından en çok kullanılan ifadedir. İşlevini tanımlamak gerekirse:

Döngü herhangi bir yerde ve herhangi bir zamanda break ifadesiyle karşılaştığı zaman çalışmasını bir anda durdurur. Böylelikle döngü hiçbir koşula bağlı kalmadan sonlanmış olur.

break ifadesi sadece ve sadece içinde bulunduğu döngüyü sonlandırır. Eğer iç içe döngüler bulunuyorsa ve en içteki döngüde break kullanılmışsa sadece içteki döngü sona erer. Örneklerle break ifadesini anlamaya çalışalım.

```
fiyat = 40              # alınacak ürünün fiyatı 40
cuzdan = 30             # cüzdanımızda 30 var
extra = 5               # extra olarak 5 birim paraya kadar çıkabiliriz

while cuzdan < fiyat: # cüzdanımızdaki para alınnacak ürünün fiyatından küçük␣
 ↪olduğu sürece cüzdanımıza para lazım

    if extra > 0:       # eğer ekleyebileceğimiz para tükenmediyse
        extra -= 1      # ekleyeceğimiz kadar parayı extradan alıp
        cuzdan += 1     # cüzdanımıza ekliyoruz

    else:               # eğer extra paramız kalmadıysa
        print("Yeterli bakiyeniz bulunmamaktadır!")
        break           # döngüden çıkıyoruz


if cuzdan == fiyat:   # döngü bittikten sonra cüzdanımızda yeterli para varsa␣
 ↪ürünü alabiliyoruz
    print("Güle güle kullanın!")
```

```
Yeterli bakiyeniz bulunmamaktadır!
```

Burada görüldüğü gibi while programımız cüzdanda yeterli para olmadığı sürece parayı arttırmak için çalışır. Eğer cüzdandaki para fiyata eşitlenirse while satırındaki şart sağlanır ve döngüden kendiliğinden çıkar. Ancak ekstra olarak yeterli birikimimiz kalmadığında break satırına ulaşır ve döngünün içindeyken bitirmiş oluruz.

b) continue

continue ifadesi break'e göre biraz daha az kullanılan bir ifadedir. İşlevini tanımlamak gerekirse:

Döngü herhangi bir yerde ve herhangi bir zamanda continue ifadesiyle karşılaştığı zaman geri kalan işlemlerini yapmadan direk bloğunun başına döner. continue ifadesinin görevini daha iyi kavrayabilmek için aşağıdaki örneği inceleyebiliriz:

```
toplam = 0

while True:                                                    # bu döngü dışarıdan␣
↪bir etkiyle durdurulmadığı sürece çalışır
    x = int(input("Bir sayı girin (bitirmek için -1): "))

    if x == -1:                                                # durdurmak için -1␣
↪girmemiz break komutunu çalıştırır
        break

    if x < 0 or x > 100:                          # istediğimiz aralıkta sayı␣
↪gelmediyse
        print("0-100 arası olmalı.")              # istediğimiz şartı belirtiriz
        continue                                  # continue komutunu okuyan python␣
↪tekrardan while kısmındaki şarta döner

    toplam += x                                   # eğer continue da takılmadıysa␣
↪buraya gelip girilen sayıyı ekler

print( "Toplam:", toplam)
```

```
Bir sayı girin (bitirmek için -1): 15
Bir sayı girin (bitirmek için -1): 20
Bir sayı girin (bitirmek için -1): 150
0-100 arası olmalı.
Bir sayı girin (bitirmek için -1): 90
Bir sayı girin (bitirmek için -1): -1
Toplam: 125
```

Python'da Hata Tipleri

Python programlarında bazen bir değişkenin tanımlanmadan kullanılmaya çalıştırılması , bazen de yapılamayacak bir aritmetik işlemin yapılması Pythonda hatalara yol açar. Ancak bu istisnai durumlarda hataların türüne göre programlarımızı daha güvenli bir şekilde yazabiliriz.Yani hata çıkarabilecek kodlarımızı öngörerek bu hataları programlarımızda yakalayabiliriz. Pythondaki bazı hatalara şunlar örnek verilebilir:

```
print(a) #Tanımlı değil, NameError hatası
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-c245efe0a5a7> in <module>
----> 1 print(a) #Tanımlı değil, NameError hatası

NameError: name 'a' is not defined
```

```
int("abc123") #ValueError Hatası
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-2e3c099de6de> in <module>
----> 1 int("abc123") #ValueError Hatası

ValueError: invalid literal for int() with base 10: 'abc123'
```

```
2 / 0  #Bir sayı 0'a bölünemez, ZeroDivisionError
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-18-89551b39432c> in <module>
----> 1 2 / 0  #Bir sayı 0'a bölünemez, ZeroDivisionError

ZeroDivisionError: division by zero
```

```
print('itü'acm) #Syntax Error
```

```
  File "<ipython-input-19-53bf08b790db>", line 1
    print('itü'acm) #Syntax Error
               ^
SyntaxError: invalid syntax
```

Hata Yakalama

Şimdiye kadar yazdığımız programlarda programın kullanıcı tarafından nasıl kullanılmasını istiyorsak her zaman o şekilde kullanılacağını varsaydık ancak pratikte işler her zaman beklediğimiz gibi gitmeyebilir. Buna bizim sayı girilmesini istediğimiz yere karakter dizisi girilmesi gibi durumları örnek verebiliriz.

```
ilk_sayı = input("ilk sayı: ")
ikinci_sayı = input("ikinci sayı: ")

ilk_sayı = int(ilk_sayı)
ikinci_sayı = int(ikinci_sayı)

print(ilk_sayı, "/", ikinci_sayı, "=", ilk_sayı / ikinci_sayı)
```

```
ilk sayı: x
ikinci sayı: y
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-41d2cc6e456b> in <module>
      2 ikinci_sayı = input("ikinci sayı: ")
      3
----> 4 ilk_sayı = int(ilk_sayı)
      5 ikinci_sayı = int(ikinci_sayı)
      6

ValueError: invalid literal for int() with base 10: 'x'
```

try-except: Python'da hata yakalama işlemleri için **try. . . except. . .** bloklarından yararlanılır.

Basit bir örnek olarak:

```
ilk_sayı    = input("ilk sayı: ")
ikinci_sayı = input("ikinci sayı: ")

try:
    sayı1 = int(ilk_sayı)
    sayı2 = int(ikinci_sayı)
    print(sayı1, "/", sayı2, "=", sayı1 / sayı2)
except ValueError:
    print("Lütfen sadece sayı girin!")
```

```
ilk sayı: x
ikinci sayı: y
Lütfen sadece sayı girin!
```

Bu örnekte programın çökme sebebi bir veriyi sayıya dönüştürmek istediğimizde kullanıcının sayı yerine bir harf girerse programımız çöker. Bu yüzden hata alma ihtimalimiz en yüksek olan int(ilk_sayı) ve int(ikinci_sayı) kodlarını **try...** bloğu içine aldık.

Veri dönüştürme işlemi sırasında kullanıcının uygun olmayan bir veri girmesi halinde üretilecek hata bir *ValueError*'dır. Bu yüzden **except...** bloğuna ValueError değerini koyduk.

Aslında kodun anlamı gayet basit: Eğer try bloğu içinde belirtilen işlemler sırasında verilen error tipi ile karşılaşırsan bunu görmezden gel ve normal şartlar altında kullanıcıya göstereceğin hata

mesajı yerine kullanıcıya altta verilen mesajı göster:

try:

`hata verebileceğini bildiğimiz kodlar`

except *HataAdı*:

`hata durumunda yapılacak işlem`

Birden fazla hata durumunda:

Çoklu hata durumlarında en uygun çözüm her hata mesajına ayrı bir çıktı vermektir:

```
ilk_sayı = input("ilk sayı: ")
ikinci_sayı = input("ikinci sayı: ")

try:
    sayı1 = int(ilk_sayı)
    sayı2 = int(ikinci_sayı) #input olarak 0 rakamı verildiğinde program␣
 ↪ZeroDivisionError verir.
    print(sayı1, "/", sayı2, "=", sayı1 / sayı2)
except ZeroDivisionError:
    print("Bir sayıyı 0'a bölemezsiniz!")
except ValueError:
    print("Lütfen sadece sayı girin!")
```

```
ilk sayı: x
ikinci sayı: 0
Lütfen sadece sayı girin!
```

Bunun dışında hataları gruplayıp hepsi için tek bir error mesajı verdirebiliriz:

```
ilk_sayı    = input("ilk sayı: ")
ikinci_sayı = input("ikinci sayı: ")

try:
    sayı1 = int(ilk_sayı)
    sayı2 = int(ikinci_sayı)
    print(sayı1, "/", sayı2, "=", sayı1 / sayı2)
except (ValueError, ZeroDivisionError):
    print("Bir hata oluştu!")
```

```
ilk sayı: x
ikinci sayı: 0
Bir hata oluştu!
```

Python'da Output Düzeni ve Formatlama

Python'da print() fonksiyonu programın output vermesini sağlar. print(), bu programlama dilinde gömülü (built-in) bir fonksiyon olarak gelir. Python'da bu tarz fonksiyonların anlaşılması için help() fonksiyonu vardır.

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Burada bizim için önemli olan *print(value, . . . , sep=' ', end='', file=sys.stdout, flush=False)* satırıdır. Burada görülen; 1. sep=' ' ifadesi, print()'in içine yazılan çoklu değerlerin arasına default olarak boşluk koyduğunu gösterir. 2. end='' ifadesi, print()'in yazdırma işi her sonlandığında bir satır alta geçtiğini gösterir.

1. sep=' ' ifadesi

print()'in içine yazdırılacak çoklu elemanların arasında yer alacak ifadeleri belirler. Tırnak işaretlerinin arasına istenilen her şey koyulabilir. Default olarak bir adet space karakteri içerir.

```
print(1, 2, 3, 4, 5, 6, 7, sep=" :) ")
```

```
1 :)2 :)3 :)4 :)5 :)6 :)7
```

2. end=' ' ifadesi

print()'in yazdırma işi bittikten sonraki davranışını belirler. Tırnak işaretlerinin arasına istenilen her şey koyulabilir. Default olarak newline ('') işareti vardır, yani bir alt satıra geçişi sağlar.

```
print(1, 2, 3, 4, 5, 6, 7, end=" :) ")
```

```
1 2 3 4 5 6 7 :)
```

NOT: '' = new line, yani Python'da yeni satır bırakma işaretidir. Nasıl çalıştığını anlamak için aşağıdaki örneği inceleyelim.

```
print("\n", "İkinci satır", '\n', "\n", "Dördüncü satır")
```

```

 İkinci satır

 Dördüncü satır
```

Örnekte görüldüğü gibi print(): 1. İlk '' ifadesini gördüğünde yazdırma kısmında bir satır sonraya geçer. 2. İkinci değer olarak "İkinci satır" ifadesi şuan bulunulan kısımdan, yani ikinci satırın

başından yazdırılır. 3. Üçüncü değer olarak '' ifadesini gören print(), bulunulan kısımdan bir alt satıra, yani "İkinci satır" ifadesinin sonundan üçüncü satırın başına geçer. 4. Üçüncü satırın başında bulunurken değer olarak '' ifadesini gören print(), bulunulan kısımdan bir alt satıra, yani üçüncü satırın başından dördüncü satırın başına geçer. 5. Beşinci ve son değer olarak "Dördüncü satır" ifadesini gören print, bulunulan yerden bu ifadeyi yazdırır, yani dördüncü satırın başından itibaren bu ifadeyi yazdırır.

Formatlama

Programlamada bazı yerlerde bir stringin içinde daha önceden tanımlı string, float, int vs. değerleri yerleştirmek isteyebiliriz. Böyle durumlar için Pythonda format() fonksiyonu bulunmaktadır. Örneğin, programımızda 2 tane tamsayı değerimiz var ve biz bunları ve toplamlarını bir string içinde ekrana bastırmak istiyoruz. Bunun için format() fonksiyonunu kullanabiliriz.

format() metodunda süslü parantezlerle sabit veya değişkenlerin sahip oldukları değerlerinin yeri tutulur.

```
n = input("Bir sayı giriniz: ")

print("Girdiğiniz sayı: {}".format(n))
```

```
Bir sayı giriniz: 8
Girdiğiniz sayı: 8
```

```
ilk_sayi = int(input("Lütfen bir sayı giriniz: "))
ikinci_sayi = int(input("Lütfen ikinci sayıyı giriniz: "))

print("Girdiğiniz ilk sayı = {} {}İkinci sayı = {} {}Toplam = {}".
 ↪format(ilk_sayi,"\n", ikinci_sayi,
                                                                "\n",␣
 ↪ilk_sayi + ikinci_sayi))
```

```
Lütfen bir sayı giriniz: 4
Lütfen ikinci sayıyı giriniz: 5
Girdiğiniz ilk sayı = 4
İkinci sayı = 5
Toplam = 9
```

Burada görüldüğü üzere print fonksiyonuyla bastırılacak herhangi bir stringin içinde, sırayla konulan süslü parantezlerin ({}) içine sırayla yazılacak olan değerler, string.format(değerler) şeklinde sırayla yazılır, virgülle ayrılır.

Değişkenlerin içine de köşeli parantezler konup sonradan formatlanabilir.

```
n = input("Bir sayı giriniz: ")

çıktı = "Girdiğiniz sayı: {}"
print(çıktı.format(n))
```

```
Bir sayı giriniz: 7
Girdiğiniz sayı: 7
```

Değişkenler, print fonksiyonu dışında formatlanıp da ekrana yazdırılabilir.

```
n = input("Bir sayı giriniz: ")

çıktı_formatsız = "Girdiğiniz sayı: {}"
çıktı_formatlı = çıktı.format(n)
print(çıktı_formatlı)
```

```
Bir sayı giriniz: 8
Girdiğiniz sayı: 8
```

Değişken yerine direkt olarak sabit bir değer de yazılabilir.

```
print("{}".format(12))
```

```
12
```

Sayıların virgülden sonraki basamak sayısı aşağıdaki şekilde formatlanabilir.

```
print("{sayı:.2f}".format(sayı = 12))
```

```
12.00
```

```
ad = "Ahmet"
boy = 187

print(f"{ad}'in boyu {boy}'dir")
```

```
Ahmet'in boyu 187'dir
```

```
ad = "Hüseyin"
soyad = "Averbek"
yaş = 21
kilo = 88.351295

print(f"{ad} {soyad} {yaş} yaşındadır ve {kilo:.2f} kilodur.")
```

```
Hüseyin Averbek 21 yaşındadır ve 88.35 kilodur.
```

Yüzde işareti yöntemi

Yüzde işareti yönteminde yüzde işaretleriyle sabit veya değişkenlerin sahip oldukları değerlerinin yeri tutulur.

- String değerlerinin yerini tutmak için %s
- Integer değerlerinin yerini tutmak için %d
- Float değerlerinin yerini tutmak için %f

kullanılır.

```
n = int(input("Bir sayı giriniz: "))

print("Girdiğiniz sayı: %d"%n)
```

```
Bir sayı giriniz: 5
Girdiğiniz sayı: 5
```

Yukarda da görüldüğü üzere, çıktıdan sonra tekrar bir yüzde işareti koyulup değişkenin adı yazılır.

```
a = float(input())
b = float(input())

print("%.1f sayısının %.1f sayısına bölümü %.1f'dir"%(a,b,a/b))
```

```
3.25
1.41
3.2 sayısının 1.4 sayısına bölümü 2.3'dir
```