

# Python Programlama Dili

## Değişkenler

Bir veriyi bellek hücrelerinde saklayan yani depolayan birimlere değişken denir. Kullanıcıdan alınan ya da programcı tarafından gönderilen verinin bellekte tutulması için değişkenleri kullanırız. Değişkenler belirlenen veri tipine göre hafızada (RAM) yer tutar. Değişkenler istenildiği zaman çağrılabilir, değiştirilebilir ya da silinebilir.<sup>9</sup>

### Değişken Tanımlama Kuralları

Python'da değişken tanımlanırken değişkenin türünü belirtmeye gerek kalmaz. Değişken ismi verilerek tanımlanabilir. Değişkenin türünü Python kendisi algılar. Değişken içindeki verinin tipi değiştiğinde de dinamik olarak tip değişecektir. Değişkenlere tam sayı, ondalık veya karakter değerleri atayabiliriz.

Python programlama dilinde değişken tanımlama kuralları aşağıda belirtilmiştir.

- ✚ Değişken isimleri rakamla veya aritmetiksel operatörlerle (+,-,\*,/) başlayamaz. Arada boşluk kullanılamaz.
- ✚ Çıkarma işlemi gibi algılandığı için "-" tire işareti kullanılamaz. "\_" alt tire işareti ile başlayabilir.
- ✚ Python programlama diline ait anahtar kelimeler değişken adı olarak kullanılamaz. (elif,as,false,and vs.)
- ✚ Değişken isimleri büyük-küçük harfe karşı duyarlıdır. LEYLEK=1 ve leylek=2 iki farklı değişkendir.

### Değişkenlere Değer Atama

Bir değişkene değer ataması yapmak için eşittir (=) operatörü kullanılır. Değişken o anda tanımlanır ve değeri atanır. Aşağıdaki örnekte sayı adında ve 5 değeri atanan değişken tanımlanmıştır.

```
sayi = 5
```

Python ile aynı anda birden fazla değişkene tek bir değer ataması yapılabilir.

```
s1 = s2 = s3 = 100
```

Python ile aynı anda değişken tanımlaması ve değer ataması aşağıdaki gibi yapılabilir.

```
not1,not2,isim = 50,75,"hayri"
```

### Değişkenler Örnek-1

```
In [1]: isim="ali"  
not1,not2 = 1,2  
print(isim+ ":" + not1 + not2)
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-1-3a33e7998e19> in <module>  
    1 isim="ali"  
    2 not1,not2 = 1,2  
----> 3 print(isim+ ":" + not1 + not2)  
  
TypeError: can only concatenate str (not "int") to str
```

```
In [2]: print(isim, ":", not1 + not2)  
ali : 3
```

Farklı tipteki verilerin çıktısı birleştirilerek alınmaz. Tip hatası verecektir. Ancak aynı tipteki veriler birleştirilebilir.

<sup>9</sup><http://roboturka.com/python/python-degiskenler/>  
Python Anaconda ve Pycharm Kurulumu

## Değişkenler Örnek-2

```
In [1]: x , y = 1, 2
```

```
In [2]: print("x:", x , "y", y)
```

```
x: 1 y 2
```

```
In [3]: x , y = y, x
```

```
In [4]: print("x:", x , "y", y)
```

```
x: 2 y 1
```

Değişkenlerin değerleri karşılıklı olarak takas edilebilir. Bunun için aşağıdakine benzer bir kullanım yapılabilir.

```
x,y=1,2
```

```
x,y = y, x
```

## Değişkenler Örnek-3

```
In [1]: tamsayi = 7  
print(tamsayi)
```

```
7
```

```
In [2]: ondalik = 7.0  
print(ondalik)  
ondalik = float(7)           # yada float metodu ile tanımlama  
print(ondalik)
```

```
7.0
```

```
7.0
```

```
In [3]: sayi = 15                # tam sayı  
litre = 2.5                     # ondalıklı sayı  
isim = "Öğr.Gör. Serkan KORKMAZ" # string ifade  
print ("sayı:",sayi)  
print ("litre:",litre)  
print ("isim:",isim)
```

```
sayı: 15
```

```
litre: 2.5
```

```
isim: Öğr.Gör. Serkan KORKMAZ
```

```
In [4]: #aynı türde iki değişken yazdırılırken  
#toplama yada birleştirme işlemi yapılabilir.  
not1,not2 = 1,2  
print(not1 + not2)
```

```
3
```

## Değişkenler Örnek-4

```
In [1]: yazi = 'merhaba'  
print(yazi)  
yazi = "hello"      # " yada ' kullanılabilir.  
print(yazi)  
# tırnaklar iç içe kullanılabilir.  
mesaj = "1071'de Anadolu Tamamen Türklerin Oldu"  
print(mesaj)
```

```
merhaba  
hello  
1071'de Anadolu Tamamen Türklerin Oldu
```

```
In [2]: isim,soyisim = "Mustafa Kemal","ATATÜRK"  
print(isim + " " + soyisim)
```

```
Mustafa Kemal ATATÜRK
```

## Etkileşimli Kabuk Hafızası

Etkileşimli kabukta alt tire işareti (`_`), yapılan son işlemin veya girilen son öğenin değerini tutma işlevi görür.

`_` komutunun etkileşimli kabuk ortamı dışında herhangi bir geçerliliği yoktur.

### Etkileşimli Kabuk Hafızası Örnek-1

```
In [1]: 2345 + 54355
```

```
Out[1]: 56700
```

```
In [2]: _
```

```
Out[2]: 56700
```

```
In [3]: _ + 15
```

```
Out[3]: 56715
```

```
In [4]: _
```

```
Out[4]: 56715
```

```
In [5]: "www"
```

```
Out[5]: 'www'
```

```
In [6]: _
```

```
Out[6]: 'www'
```

```
In [7]: _ + ".python.org"
```

```
Out[7]: 'www.python.org'
```

Gördüğünüz gibi, `_` komutu son girilen öğeyi hafızasında tutuyor.

Bu özellikten çeşitli şekillerde yararlanabiliriz:

```
2345+54355  
56700
```

Eğer bu işlemin ardından `_` komutunu verirsek şöyle bir çıktı alırız:56700

`_` komutu yalnızca sayıları değil, karakter dizilerini de hafızasında tutabilir:

```
"www"
```

Eğer bu işlemin ardından `_ + ".python.org"` komutunu verirsek şöyle bir çıktı alırız:

```
"www.python.org"
```

## Açıklama (Yorum) Satırları

Yorum satırları derleme aşamasında (kodların çalıştırılıp çıkış üretilmesi) dikkate alınmaz.

Tek satırlık açıklama (yorum satırı) eklemek için ilgili satırın başına # işareti konulmalıdır.

Çok satırlık açıklama (yorum satırı) eklemek için ilgili satırın başına ve sonuna ''' işareti konulmalıdır.

### Açıklama Satırı Örnek-1

```
In [1]: # Bu bir açıklama satırıdır. Bu satır görüntülenmez.  
print("Bu satır görüntülenir.")
```

Bu satır görüntülenir.

```
In [2]: # TEK YORUM SATIRI  
...  
Her Satıra  
# İşareti Yazmak Yerine  
Üç Tırnak İşareti ile Birden Fazla Yorum Satırı Oluşturabilirsiniz  
...  
print("Bu satır görünür.")
```

Bu satır görünür.

## Versiyon Kontrolü

Python 2 ve 3 olmak üzere iki temel sürüm mevcuttur.

Kullandığınız Python sürümünü öğrenmek için komut satırından python --version komutunu çalıştırabilirsiniz.

```
!python --version
```

```
In [1]: !python --version
```

Python 3.7.3

## Dil Kodlaması

Varsayılan olarak, Python kaynak dosyaları UTF-8'de kodlanmış olarak değerlendirilir. Bu kodlamada, standart kitaplık tanımlayıcılar için yalnızca ASCII karakterlerini kullanmasına rağmen dünyadaki çoğu dilin karakterleri aynı anda string hazırlayıcılarında, tanımlayıcılarında ve yorumlarında kullanılabilir.

Tüm bu karakterleri doğru görüntülemek için, editörünüz dosyanın UTF-8 olduğunu anlamalıdır ve dosyadaki tüm karakterleri destekleyen bir yazı karakteri kullanmalıdır.

Varsayılandan farklı bir kodlama bildirmek için, dosyanın ilk satırı olarak özel bir yorum satırı eklenmelidir.

Sözdizimi aşağıdaki gibidir:

```
# -*- coding: encoding -*-
```

Örneğin, Windows-1252 kodlamasının kullanılacağını ilan etmek için kaynak kod dosyanızın ilk satırı şöyle olmalıdır:

```
# -*- coding: cp1252 -*-
```

## Operatörler

Python programlama dilinde kullanılan operatörler aşağıdaki tabloda belirtilmiştir.

Operatör	Anlamı
<b>Aritmetiksel Operatörler</b>	
+	Toplama
-	Çıkarma
*	Çarpma
/	Bölme
//	Tamsayı Bölme
%	Bölümden Kalan (Mod)
**	Kuvvet (Üs) Alma
<b>Karakter Kümesi Operatörleri</b>	
+	Karakter kümesi birleştirme
*	Karakter kümesi çoğaltma
/	String kaçış operatörü
<b>Birleşik Atama Operatörleri</b>	
=	Atama Operatörü
+=	Ekleyerek Atama Operatörü
-=	Çıkartarak Atama Operatörü
*=	Çarparak Atama Operatörü
/=	Bölerek Atama Operatörü
<b>Karşılaştırma Operatörleri</b>	
==	Eşit mi?
!=	Farklı mı?
>	Büyük mü?
<	Küçük mü?
>=	Büyük veya eşit mi?
<=	Küçük veya eşit mi?
<b>Mantıksal Operatörler</b>	
and	ve
or	veya
not	değil
<b>Boolean Operatörler</b>	
True	Doğru
False	Yanlış

## Aritmetiksel Operatörler

Python programlama dilinde kullanılan aritmetiksel operatörler aşağıdaki tabloda belirtilmiştir.

Operatör	Anlamı
<b>Aritmetiksel Operatörler</b>	
+	Toplama
-	Çıkarma
*	Çarpma
/	Bölme
//	Tamsayı Bölme
%	Bölümden Kalan (Mod)
**	Kuvvet (Üs) Alma

## Aritmetiksel Operatörler Örnek-1

Python programlama dilinde kullanılan aritmetiksel operatörlerin kullanım örnekleri

```
In [1]: 1 / 2
```

```
Out[1]: 0.5
```

```
In [2]: 2 ** 3
```

```
Out[2]: 8
```

```
In [3]: 17 / 3
```

```
Out[3]: 5.666666666666667
```

```
In [4]: 17 // 3
```

```
Out[4]: 5
```

```
In [5]: 3 - 5
```

```
Out[5]: -2
```

```
In [6]: 7 + 12
```

```
Out[6]: 19
```

```
In [7]: 2 * 5
```

```
Out[7]: 10
```

```
In [8]: 12 % 5
```

```
Out[8]: 2
```

## Aritmetiksel Operatörler Örnek-2

```
In [1]: a=5  
b=10  
c=a+2*b  
print(c)
```

```
25
```

Kod	Ekran Çıktısı
a=5 b=10 c=a+2*b print(c)	25

## Karakter Kümesi Operatörleri

Python programlama dilinde kullanılan karakter kümesi operatörleri aşağıdaki tabloda belirtilmiştir.

Operatör	Anlamı
<b>Karakter Kümesi Operatörleri</b>	
+	Karakter kümesi birleştirme
*	Karakter kümesi çoğaltma
/	String kaçış operatörü

Karakter kümelerini birleştirmek için "+" operatörü kullanılır.

Karakter kümeleri aralarında boşluk bırakılmadan birbirlerine bitiştirilir.

Karakter dizilerini birleştirmek için mutlaka + operatörü kullanmak zorunda değiliz.

Python + operatörü kullanılsa bile karakter dizilerini birleştirecektir.

Python programlama dilinde string kaçış operatörü '/' operatördür.

'/' kullanıldıktan sonra ' veya " gibi özel karakterler kullanılabilir.

### Karakter Kümesi Operatörleri Örnek-1

```
In [1]: a="Python"
        b="Programlama"
        c="Dili"
        d=a+b+c
        print(d)

PythonProgramlamaDili
```

### Karakter Kümesi Operatörleri Örnek-2

```
In [1]: "Serkan" + "Korkmaz"
Out[1]: 'SerkanKorkmaz'

In [2]: "Serkan" + " " + "Korkmaz"
Out[2]: 'Serkan Korkmaz'

In [3]: "Öğr." + "Gör." + " " + "Serkan" + " " + "Korkmaz"
Out[3]: 'Öğr.Gör. Serkan Korkmaz'

In [4]: # + operatörü kullanılsa bile birleştirme yapar
        "Öğr." "Gör." " " "Serkan" " " "Korkmaz"
Out[4]: 'Öğr.Gör. Serkan Korkmaz'
```

Kod	Ekran Çıktısı
"Serkan" + "Korkmaz"	'SerkanKorkmaz'
"Serkan" + " " + "Korkmaz"	'Serkan Korkmaz'
"Öğr." + "Gör." + " " + "Serkan" + " " + "Korkmaz"	'Öğr.Gör. Serkan Korkmaz'



### Karakter Kümesi Operatörleri Örnek-3

```
In [1]: "w" * 3
```

```
Out[1]: 'www'
```

```
In [2]: "yavaş " * 2
```

```
Out[2]: 'yavaş yavaş '
```

```
In [3]: "-" * 10
```

```
Out[3]: '-----'
```

```
In [4]: "uzak" + " " * 5 + "çok uzak..."
```

```
Out[4]: 'uzak     çok uzak...'
```

### Karakter Kümesi Operatörleri Örnek-4

```
In [1]: 'spam eggs' # single quotes
```

```
Out[1]: 'spam eggs'
```

```
In [2]: 'doesn\'t' # use \' to escape the single quote...
```

```
Out[2]: "doesn't"
```

```
In [3]: "doesn't" # ...or use double quotes instead
```

```
Out[3]: "doesn't"
```

```
In [4]: '"Yes," they said.'
```

```
Out[4]: '"Yes," they said.'
```

```
In [5]: "\"Yes,\" they said."
```

```
Out[5]: '"Yes," they said.'
```

```
In [6]: '"Isn\'t," they said.'
```

```
Out[6]: '"Isn\'t," they said.'
```

```
In [7]: print('"Isn\'t," they said.')
```

```
"Isn't," they said.
```

```
In [8]: s = 'First line.\nSecond line.' # \n means newline  
s # without print(), \n is included in the output
```

```
Out[8]: 'First line.\nSecond line.'
```

```
In [9]: print(s) # with print(), \n produces a new line
```

```
First line.  
Second line.
```

## Birleşik Atama Operatörleri

Python programlama dilinde kullanılan birleşik atama operatörleri aşağıdaki tabloda belirtilmiştir.

Operatör	Anlamı
<b>Birleşik Atama Operatörleri</b>	
=	Atama Operatörü
+=	Ekleyerek Atama Operatörü
-=	Çıkartarak Atama Operatörü
*=	Çarparak Atama Operatörü
/=	Bölerek Atama Operatörü

### Birleşik Atama Operatörleri Örnek-1

```
In [1]: a=1  
b=3  
c=5  
d=10
```

```
In [2]: a+=b  
print("a:",a,"b:",b,"c:",c,"d:",d)  
a: 4 b: 3 c: 5 d: 10
```

```
In [3]: c-=b  
print("a:",a,"b:",b,"c:",c,"d:",d)  
a: 4 b: 3 c: 2 d: 10
```

```
In [4]: b*=a  
print("a:",a,"b:",b,"c:",c,"d:",d)  
a: 4 b: 12 c: 2 d: 10
```

```
In [5]: d/=c  
print("a:",a,"b:",b,"c:",c,"d:",d)  
a: 4 b: 12 c: 2 d: 5.0
```

### Birleşik Atama Operatörleri Örnek-2

```
In [1]: a,b,x,y = 5,3,10,15  
a += b  
b +=2  
x *= y  
y -=1  
print("a:", a , "b:", b, "x:", x, "y:", y )  
a: 8 b: 5 x: 150 y: 14
```

## Mantıksal Operatörler

Python programlama dilinde kullanılan mantıksal operatörler aşağıdaki tabloda belirtilmiştir.

Operatör	Anlamı
<b>Mantıksal Operatörler</b>	
and	ve
or	veya
not	değil

### Mantıksal Operatörler Örnek-1

```
In [1]: a=3  
b=4
```

```
In [2]: if a==3 and b==4:  
        print("Evet")  
else:  
        print("Hayır")
```

Evet

```
In [3]: if a==3 and b==5:  
        print("Evet")  
else:  
        print("Hayır")
```

Hayır

```
In [4]: if a==3 or b==5:  
        print("Evet")  
else:  
        print("Hayır")
```

Evet

```
In [5]: if not (a==b):  
        print("Evet")  
else:  
        print("Hayır")
```

Evet

```
In [6]: if not (a!=b):  
        print("Evet")  
else:  
        print("Hayır")
```

Hayır

## Giriş ve Çıkış Fonksiyonları

Python programlama dilinde giriş ve çıkış işlemlerinde kullanılan fonksiyonlar print() ve input() fonksiyonlarıdır.

### print() fonksiyonu

print() fonksiyonu ekrana mesaj yazdırmak için kullanılır. print() fonksiyonuna herhangi bir değişken parametre olarak yollanabilir ve bu değişkenin değeri yazdırılabilir.

#### print() Örnek-1

```
In [1]: print("Merhaba Dünya!")  
Merhaba Dünya!
```

print() fonksiyonu ile ekrana "Merhaba Dünya!" mesajı yazdırılmıştır.

#### print() Örnek-2

```
In [1]: i = 256*256  
print("i:" , i)  
i: 65536
```

print() fonksiyonuna gönderilen i değişkeni ve bu değişkenin değeri ekrana yazdırılabilir.

#### print() Örnek-3

```
In [1]: print("*" * 1)  
print("*" * 2)  
print("*" * 3)  
print("*" * 4)  
print("*" * 5)  
  
*  
**  
***  
****  
*****
```

print() fonksiyonunda kullanılan "\*" operatörü ile istenilen karakter dizilerini dilediğimiz kadar çoğaltılabiliriz.

#### print() Örnek-4

```
In [1]: print('#'*5, '$'*3)  
##### $$$
```

### print() Örnek-5

```
In [1]: print("Python")
        print("Naber?")

Python
Naber?
```

print() fonksiyonu ile ekrana alt alta gelecek şekilde "Python" ve "Naber?" mesajları yazdırılmıştır.

### print() Örnek-6

```
In [1]: print("Öğr.Gör.", "Serkan", "Korkmaz")

Öğr.Gör. Serkan Korkmaz

In [2]: print("Öğr.Gör.\nSerkan Korkmaz")

Öğr.Gör.
Serkan Korkmaz
```

### print() Örnek-7

```
In [1]: merhaba = 'Merhaba' # tek tırnak da kullanılabilir

        # önce veri tipi sonra değişkenler tanımlanabilir
        m7 = '%s kelimesi %d karakterdir.' % (merhaba, len(merhaba))

        print(m7) # ekrana "Merhaba kelimesi 7 karakterdir." yazdırır

Merhaba kelimesi 7 karakterdir.
```

merhaba = 'Merhaba' # tek tırnak da kullanılabilir

# önce veri tipi sonra değişkenler tanımlanabilir

m7 = '%s kelimesi %d karakterdir.' % (merhaba, len(merhaba))

print (m7) # ekrana "Merhaba kelimesi 7 karakterdir." yazdırır.

Kod	Ekran Çıktısı
merhaba = 'Merhaba'	
len(merhaba)	7
m7='%s kelimesi %d karakterdir.' % (merhaba, len(merhaba))	
print (m7)	Merhaba kelimesi 7 k arakterdir.

## format() metodu

Python programlama dilinde format() metodu kullanılarak karakter dizilerinde biçimlendirme işlemleri kolay ve etkili bir biçimde yapılabilir.

### format metodu() Örnek-1

```
In [1]: metin = "{} ve {} iyi bir ikilidir"
        metin.format("Python", "Django")
```

```
Out[1]: 'Python ve Django iyi bir ikilidir'
```

```
In [2]: metin = "{} {}'yi seviyor!"
        metin.format("Ali", "Ayşe")
```

```
Out[2]: "Ali Ayşe'yi seviyor!"
```

```
In [3]: metin = "{} {} yaşında bir {}dur"
        metin.format("Ahmet", "18", "futbolcu")
```

```
Out[3]: 'Ahmet 18 yaşında bir futbolcudur'
```

Öncelikle bir karakter dizisi tanımlıyoruz. Bu karakter dizisi içindeki değişken değerleri ise {} işaretleri ile gösteriyoruz. Daha sonra format() metodunu alıp bu karakter dizisine bağlıyoruz. Karakter dizisi içindeki {} işaretleri ile gösterdiğimiz yerlere gelecek değerleri de format() metodunun parantezleri içinde sırasıyla gösteriyoruz.

Yalnız burada şuna dikkat etmemiz lazım: Karakter dizisi içinde kaç tane {} işareti varsa, format() metodunun parantezleri içinde de o sayıda değer olması gerekiyor.

Bu yapının, yazdığımız programlarda işimizi ne kadar kolaylaştıracağını tahmin edebilirsiniz. Kısa karakter dizilerinde pek belli olmayabilir, ama özellikle çok uzun ve boşluklu karakter dizilerini biçimlendirirken format() metodunun hayat kurtardığına kendiniz de şahit olacaksınız.

### format metodu ( ) Örnek-2

```
In [1]: a=2
        b=3
        print("a={} ve b={}".format(a,b))
```

```
a=2 ve b=3
2 + 3 = 5
```

```
In [2]: print("{} + {} = {}".format(a,b,a+b))
```

```
2 + 3 = 5
```

### format metodu() Örnek-3

```
In [1]: ad = input('Adınızı giriniz : ')
        soyad = input('Soyadınızı giriniz : ')
        memleket = input('Memleketinizi giriniz : ')

        print("Adı : {}\nSoyadı : {}\nMemleketi : {}".format(ad, soyad, memleket))

Adınızı giriniz : Serkan
Soyadınızı giriniz : Korkmaz
Memleketinizi giriniz : Adana
Adı : Serkan
Soyadı : Korkmaz
Memleketi : Adana
```

Yukarıdaki örnekte girilen adı, soyadı ve memleketi bilgileri öncelikle ilgili değişkenlerde tutulur. Değişkenlerde tutulan bu bilgiler format metodu kullanılarak ekrana yazdırılmıştır.

### format metodu() Örnek-4

```
In [1]: print("{0} {1} {2}".format("Serkan", "Korkmaz", "Adana"))

Serkan Korkmaz Adana
```

Küme parantezlerini, önceki örneklerde görüldüğü şekilde içi boş olarak kullanabilirsiniz. Böyle bir durumda Python, karakter dizisi içindeki küme parantezleriyle, karakter dizisi dışındaki değerleri teker teker ve sırasıyla eşleştirecektir.

Ancak yukarıdaki örnekte olduğu gibi küme parantezleri içine birer sayı yazarak, karakter dizisi dışındaki değerlerin hangi sırayla kullanılacağını belirleyebiliriz.

## print parametreleri

Python programlama dilinde print fonksiyonunda parametreler bu kısımda anlatılacaktır.

### sep parametresi

**sep** ifadesi, İngilizcede separator (ayırıcı, ayraç) kelimesinin kısaltmasıdır. Dolayısıyla print() fonksiyonundaki bu sep parametresi, ekrana basılacak öğeler arasına hangi karakterin yerleştirileceğini gösterir.

Bu parametrenin **ön tanımlı değeri** bir adet **boşluk karakteri**dir ("""). Yani siz bu özel parametrenin değerini başka bir şeyle değiştirmezseniz, Python bu parametrenin değerini bir adet boşluk karakteri olarak alacak ve ekrana basılacak öğeleri birbirinden birer boşlukla ayıracaktır.

Ancak eğer biz istersek bu sep parametresinin değerini değiştirebiliriz. Böylece Python, karakter dizilerini birleştirirken araya boşluk değil, bizim istediğimiz başka bir karakteri yerleştirebilir.

Eğer amacınız parametreleri birbirine bitişirmekse, yani sep parametresinin öntanımlı değeri olan boşluk karakterini ortadan kaldırmaksa, sep parametresine boş bir karakter dizisi vermeniz gerekmektedir.

print('a', 'b', sep="") komutunun ekran çıktısı "ab" şeklinde olacaktır.

sep parametresine None değeri verildiğinde, print() fonksiyonu bu parametre için öntanımlı değeri (yani bir adet boşluk) kullanır.

print('a', 'b', sep=None) komutunun ekran çıktısı "ab" şeklinde olacaktır.

### sep parametresi Örnek-1

```
In [1]: print("29","04","2019")
```

```
29 04 2019
```

```
In [2]: print("29","04","2019",sep="/")
```

```
29/04/2019
```

```
In [3]: print("www","python","org")
```

```
www python org
```

```
In [4]: print("www","python","org",sep=".")
```

```
www.python.org
```

```
In [5]: print("www","python","org",sep="\n")
```

```
www
python
org
```

```
In [6]: print("www","python","org",sep="\t")
```

```
www    python    org
```

```
In [7]: print("p","y","t","h","o","n",sep=".")
```

```
p.y.t.h.o.n
```



### *end parametresi*

end parametresi kullanıldığında ise, end parametresinde gönderilen ifade ile mesaj bitirilir.

print() fonksiyonunun sep adlı özel bir parametresi bulunur. Bu parametre print() fonksiyonunda görünmese bile her zaman oradadır. Aynı bu şekilde, print() fonksiyonunun end adlı özel bir parametresi daha bulunur. Tıpkı sep parametresi gibi, end parametresi de print() fonksiyonunda görünmese bile her zaman oradadır.

sep parametresi print() fonksiyonuna verilen parametreler birleştirilirken araya hangi karakterin gireceğini belirtirken, end parametresi ise bu parametrelerin **sonuna** neyin geleceğini belirler.

print() fonksiyonu öntanımlı olarak, parametrelerin sonuna 'satır başı karakteri' ekler.

print("birinci satır", "ikinci satır", "üçüncü satır", sep="\n") komutunun ekran çıktısı aşağıdaki gibi olacaktır.

birinci satır

ikinci satır

üçüncü satır

end parametresi de, tıpkı sep parametresi gibi, her zaman ismiyle birlikte kullanılması gereken bir parametredir. Yani eğer end parametresinin ismini belirtmeden sadece değerini kullanmaya çalışırsak Python ne yapmaya çalıştığımızı anlayamaz. Yine tıpkı sep parametresi gibi, end parametresinin değeri de sadece bir karakter dizisi veya None olabilir.

### end parametresi Örnek-1

```
In [1]: print("www.python.org",end=" :)")  
www.python.org :)
```

print() fonksiyonunda kullanılan end parametresi ile "www.python.org" ifadesinin sonuna ":" karakteri eklenmiştir.

### file parametresi

print() fonksiyonunun sep ve end dışında üçüncü bir özel parametresi file parametresidir. file parametresi, print() fonksiyonuna verilen karakter dizisi ve/veya sayıların, yani parametrelerin nereye yazılacağını belirtir.

Bu parametrenin öntanımlı değeri sys.stdout'tur.sys.stdout, 'standart çıktı konumu' anlamına gelir.Standart çıktı konumu; bir programın, ürettiği çıktıları verdiği yerdir.

Standart çıktı konumu = çıktıların standart olarak verildiği konum.

Mesela Python öntanımlı olarak, ürettiği çıktıları ekrana verir.Eğer o anda etkileşimli kabukta çalışıyorsanız, Python ürettiği çıktıları etkileşimli kabuk üzerinde gösterir.Eğer yazdığımız bir programı komut satırında çalıştırıyorsanız, üretilen çıktılar komut satırında görünür.Dolayısıyla Python'ın standart çıktı konumu etkileşimli kabuk veya komut satırındadır.Yani print() fonksiyonu yardımıyla bastığımız çıktılar etkileşimli kabukta ya da komut satırında görünecektir.

Tıpkı sep ve end parametreleri gibi, file parametresi de, siz görmesenez bile her zaman print() fonksiyonunun içinde vardır. Yani diyelim ki şöyle bir komut verdik:

```
print("Tahir olmak da ayıp değil", "Zühre olmak da")
```

Python bu komutu şöyle algılar:

```
print("Tahir olmak da ayıp değil", "Zühre olmak da", sep=" ", end="\n", file=sys.stdout)
```

Yani kendisine parametre olarak verilen değerleri ekrana yazdırırken sırasıyla şu işlemleri gerçekleştirir:

1. Parametrelerin arasına birer boşluk koyar (sep=" "),
2. Ekrana yazdırma işlemi bittikten sonra parametrelerin sonuna satır başı karakteri ekler (end="\n")
3. Bu çıktıyı standart çıktı konumuna gönderir (file=sys.stdout).

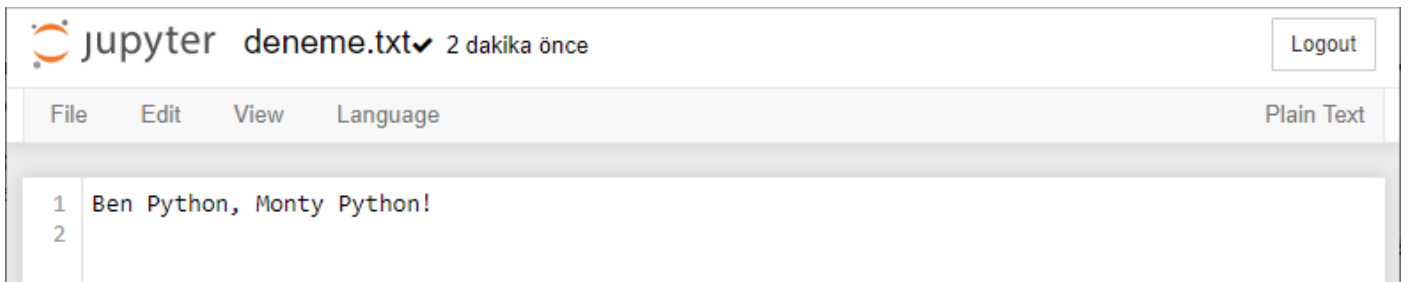
İşte biz burada file parametresinin değeri olan standart çıktı konumuna başka bir değer vererek bu konumu değiştiriyoruz.

### file parametresi Örnek-1

```
In [1]: # -*- coding: utf-8 -*-
dosya = open("deneme.txt", "w")
print("Ben Python, Monty Python!", file=dosya)
dosya.close()
```

```
In [2]: import os
os.getcwd()
```

```
Out[2]: 'C:\\Users\\Serkan Korkmaz\\SERKAN\\print'
```



The screenshot shows a Jupyter Notebook window titled "deneme.txt" with a "Logout" button in the top right corner. The menu bar includes "File", "Edit", "View", "Language", and "Plain Text". The code cell contains the following Python code:

```
1 Ben Python, Monty Python!
2
```

The output of the code cell is displayed below the code:

```
'C:\\Users\\Serkan Korkmaz\\SERKAN\\print'
```

“deneme.txt” dosyasının içeriği

Gelin isterseniz örnekteki kodları satır satır inceleyelim:

```
1.dosya = open("deneme.txt", "w")
```

Öncelikle deneme.txt adlı bir dosya oluşturduk ve bu dosyayı dosya adlı bir değişkene atadık. Open() fonksiyonun ilk parametresi "deneme.txt" adlı bir karakter dizisi. İşte bu karakter dizisi bizim oluşturmak istediğimiz dosyanın adını gösteriyor. İkinci parametre ise "w" adlı başka bir karakter dizisi. Bu da deneme.txt dosyasının yazma modunda açılacağını gösteriyor.

2. Oluşturduğumuz bu deneme.txt adlı dosya, o anda bulunduğunuz dizin içinde oluşacaktır. Bu dizinin hangisi olduğunu öğrenmek için şu komutları verebilirsiniz:

```
In [2]: import os
        os.getcwd()

Out[2]: 'C:\\Users\\Serkan Korkmaz\\SERKAN\\print'
```

Bu komutun çıktısında hangi dizinin adı görünüyorsa, deneme.txt dosyası da o dizinin içindedir.

```
3. print("Ben Python, Monty Python!", file=dosya)
```

Ardından da normal bir şekilde print() fonksiyonumuzu çalıştırdık. Ama gördüğümüz gibi print() fonksiyonu bize herhangi bir çıktı vermedi. Çünkü, daha önce de söylediğimiz gibi, print() fonksiyonunu biz ekrana değil, dosyaya çıktı verecek şekilde ayarladık. Bu işlemi, file adlı bir parametreye, biraz önce tanımladığımız dosya değişkenini yazarak yaptık.

```
4. dosya.close()
```

Bu komut yardımıyla da, yaptığımız değişikliklerin dosyada görünebilmesi için ilk başta açtığımız dosyayı kapatıyoruz.

Şimdi "deneme.txt" adlı dosyayı açın. Biraz önce print() fonksiyonuyla yazdırdığımız "Ben Python, Monty Python!" karakter dizisinin dosyaya işlenmiş olduğunu göreceksiniz.

Böylece print() fonksiyonunun standart çıktı konumunu değiştirmiş olduk. Yani print() fonksiyonunun file adlı parametresine farklı bir değer vererek, print() fonksiyonunun etkileşimli kabuğa değil dosyaya yazmasını sağladık.

### *flush parametresi*

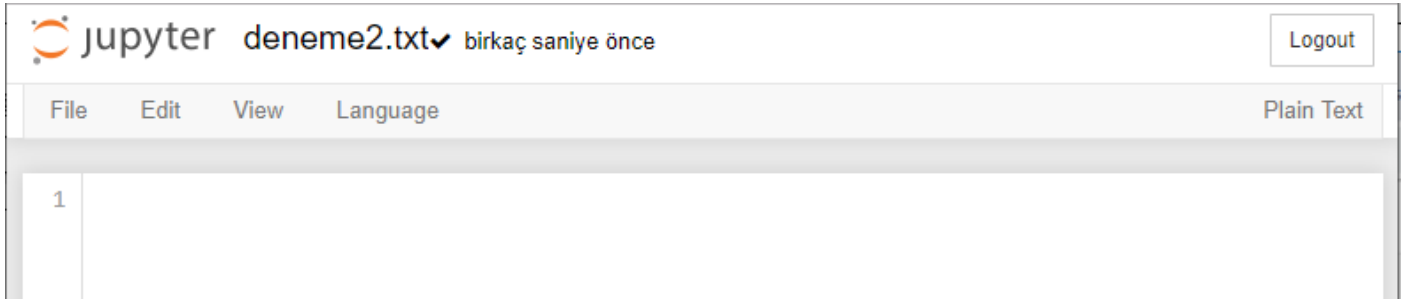
Python programlama dilinde print() gibi bir komut verdiğimizde, yazdırmak istediğimiz bilgi standart çıktı konumuna gönderilir. Ancak Python'da bazı işlemler standart çıktı konumuna gönderilmeden önce bir süre tamponda bekletilir. Daha sonra bekleyen bu işlemler topluca standart çıktı konumuna (örneğin deneme.txt dosyasına) gönderilir. Dosyaya yazma işlemleri sona erdiğinde yani dosya kapatıldığında ise Python, tamponda bekleyen bütün bilgileri standart çıktı konumuna (ilgili dosyaya) gönderir.

#### flush parametresi Örnek-1

```
In [1]: f=open("deneme2.txt", "w")  
print("Serkan Korkmaz", file=f)
```

```
In [2]: print("Adana", file=f)  
print("Gaziantep", file=f)
```

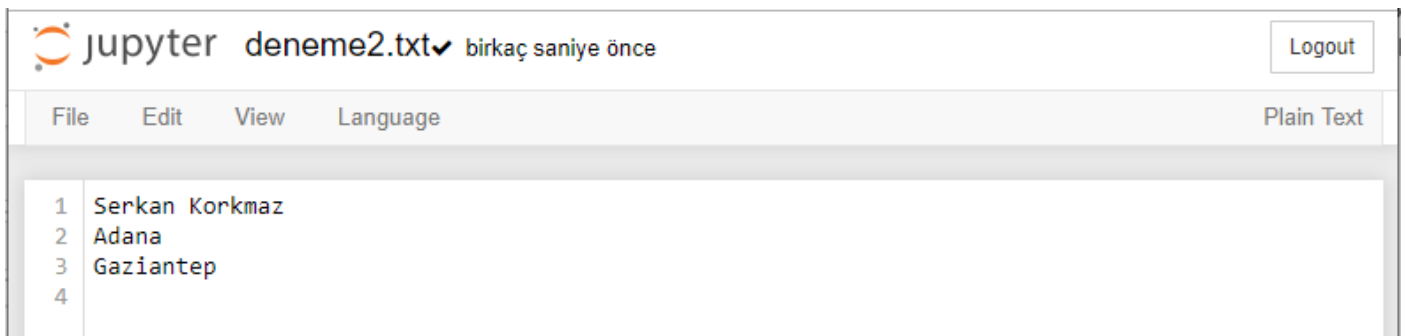
```
In [3]: f.close()
```



Dosya kapatılmadan önce "deneme2.txt" dosyasının içeriği

```
In [3]: f.close()
```

Yukarıdaki komutla dosya kapatılır.



Dosya kapatıldıktan sonra "deneme2.txt" dosyasının içeriği

Gelin isterseniz örnekteki kodları satır satır inceleyelim:

```
1. f = open("deneme2.txt", "w")
```

Bu komutla deneme2.txt adlı bir dosyayı yazma kipinde açtık.

```
2. print("Serkan Korkmaz", file=f)
```

Bu komutla "deneme2.txt" adlı dosyaya "Serkan Korkmaz" diye bir satır eklenmiştir.

Oluşturulan "deneme2.txt" dosyasını açıldığında dosyada hiçbir bilgi olmadığı görünecektir. Dosya şu anda boş görünecektir.

Python bizim bu dosyaya eklemek istediğimiz satırı tampona kaydetmiştir. Dosyaya yazma işlemleri sona erdiğinde ise Python, tamponda bekleyen bütün bilgileri standart çıktı konumuna (yani bizim durumumuzda f adlı değişkenin tuttuğu "deneme2.txt" adlı dosyaya) aktaracaktır.

```
3. print("Adana", file=f)
```

Bu komutla "deneme2.txt" adlı dosyaya "Adana" diye bir satır eklenmiştir.

```
4. print("Gaziantep", file=f)
```

Bu komutla "deneme2.txt" adlı dosyaya "Gaziantep" diye bir satır eklenmiştir.

Dosyaya yazacağımız şeyler bu kadar.

```
5. f.close()
```

Bu komutla "deneme2.txt" adlı dosya kapatılır. Artık yazma işleminin sona erdiği Python'a bildirilmiş olur.

Böylece dosyamızı kapatmış olduk.

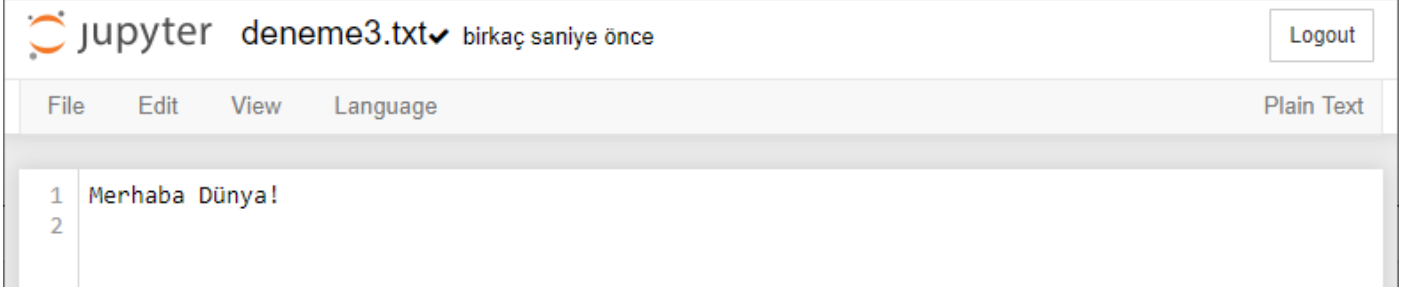
"deneme2.txt" dosyası tekrar açıldığında 'Serkan Korkmaz, 'Adana' ve 'Gaziantep' satırları görünecektir.

Gerçekten de Python dosyaya yazdırmak istediğimiz bütün verileri önce tamponda bekletmiştir.

Daha sonra dosya kapatılınca tamponda bekleyen bütün veriler dosyaya aktarılmıştır.

## flush parametresi Örnek-2

```
In [1]: # -*- coding: utf-8 -*-
f=open("deneme3.txt", "w", encoding="utf-8")
print("Merhaba Dünya!", file=f, flush=True)
```



Şimdi dikkatlice inceleyin:

```
f=open("deneme3.txt", "w", encoding="utf-8")
```

“deneme3.txt” isimli dosyamızı yazma modunda oluşturduk.

Open fonksiyonunun encoding parametresine verilen “utf-8” değeri ile Türkçe karakterlerin düzgün görüntülenmesi sağlanır. Eğer Türkçe karakterler hala düzgün görüntülenmiyorsa “utf-8” yerine “cp1254” dil kodlaması kullanılabilir.

Şimdi bu dosyaya bazı bilgiler ekleyelim:

```
print("Merhaba Dünya!", file=f, flush=True)
```

Gördüğünüz gibi, burada flush adlı yeni bir parametre kullandık. Bu parametreye verdiğimiz değer True.

Şimdi dosyaya çift tıklayarak dosyayı açın.

Gördüğünüz gibi, henüz dosyayı kapatmadığımız halde bilgiler dosyaya yazıldı.

Bu durum, tahmin edebileceğiniz gibi, flush parametresine True değeri vermemiz sayesinde.

Flush parametresi iki değer alabilir: True ve False.

Flush parametresinin öntanımlı değeri False’tur. Yani eğer biz bu parametreye herhangi bir değer belirtmezsek Python bu parametrenin değerini False olarak kabul edecek ve bilgilerin dosyaya yazılması için dosyanın kapatılmasını bekleyecektir.

Flush parametresine True değerini verdiğimizde ise veriler tamponda bekletilmeksizin standart çıktı konumuna gönderilecektir.

Yazdığınız bir programda, yapmak istediğiniz işin niteliğine göre, bir dosyaya yazmak istediğiniz bilgilerin bir süre tamponda bekletilmesini veya hiç bekletilmeden doğrudan dosyaya yazılmasını isteyebilirsiniz.

İhtiyacınıza bağlı olarak da flush parametresinin değerini True veya False olarak belirleyebilirsiniz.

### \* parametresi

Bir parametrenin başına yıldız eklediğimizde, o parametreyi oluşturan bütün öğeler tek tek fonksiyona gönderilir.

Yıldızlı parametreleri bir fonksiyona uygulayabilmemiz için o fonksiyonun birden fazla parametre alabilmesinin yanı sıra, yapısının da yıldızlı parametre almaya uygun olması gerekir.

open(), type() ve len() fonksiyonlarının yapısı yıldızlı parametre almaya uygun değildir.

Örneğin; \* parametresi kullanılarak len() fonksiyonuna 11 karakterli bir ifade gönderilirse, len() fonksiyonuna 1 değil de, 11 ayrı parametre verilmiş gibi bir sonuç ortaya çıkmaktadır.

Dolayısıyla yıldızlı parametreleri her fonksiyonla birlikte kullanamayız.

Ancak print() fonksiyonu yıldızlı parametreler için son derece uygun bir fonksiyondur.

<i>Python Kodu</i>	<i>Ekran Çıktısı</i>	<i>sep parametresi</i>	<i>Anlamı</i>
<code>print("www","python","org")</code>	wwwpython org	sep="."	. karakteri ile ayırır.
<code>print("www","python","org",sep="\n")</code>	www.python.org	sep="\n"	Alt alta yazar.
<code>print("www","python","org",sep=":",end=" :)")</code>	www.python.org :)	sep=":"	Tab ile ayırır.
<code>print("p","y","t","h","o","n",sep=".")</code>	p.y.t.h.o.n		
<code>print(* "python",sep=".")</code>	p.y.t.h.o.n		

### \* parametresi Örnek-1

```
In [1]: print("L", "i", "n", "u", "x", sep=".")
```

```
L.i.n.u.x
```

```
In [3]: print(* "Linux", sep=".")
```

```
L.i.n.u.x
```

```
In [4]: print(* "Galatasaray")
```

```
G a l a t a s a r a y
```

```
In [5]: print(* "TBMM", sep=".")
```

```
T.B.M.M
```

```
In [6]: print(* "abccdefgğh", sep="/")
```

```
a/b/c/ç/d/e/f/g/ğ/h
```

Python programlama dilinde print fonksiyonunda parametreler aşağıdaki tabloda belirtilmiştir.

Parametre	İşlevi
<i>Print Parametreleri</i>	
sep	Ekranında yazdırılan ifadeler, belirtilen karakter ile ayrılır.
end	Ekranında yazdırılan ifadeler, belirtilen karakter ile sonlandırılır.
file	Yazılan ifadenin yazılacağı dosyaya belirtir.
flush	Tamponda bekletilen ifadeler yazdırılır.
*	Parametreyi oluşturan bütün öğeler tek tek fonksiyona gönderilir.

## input() fonksiyonu

input() fonksiyonu kullanıcıdan bilgi girişi almak için kullanılır.

### input() Örnek-1

```
In [*]: ad=input("Lütfen adınızı giriniz:")
print("Merhaba",ad)

Lütfen adınızı giriniz: Serkan Korkmaz
```

```
ad=input("Lütfen adınızı giriniz:")
```

Yukarıdaki python kodunda kullanıcının girdiği bilgiler (isim bilgileri) öncelikle ad isimli değişkene aktarılır.

```
In [1]: ad=input("Lütfen adınızı giriniz:")
print("Merhaba",ad)

Lütfen adınızı giriniz:Serkan Korkmaz
Merhaba Serkan Korkmaz
```

```
print("Merhaba",ad)
```

Yukarıdaki python kodunda ad isimli değişkenin değeri önüne "Merhaba" öneki getirilerek ekrana yazdırılır.

### input() Örnek-2

```
In [*]: yas=input("Yaşınızı giriniz:")
print("Yaşınız",yas)

Yaşınızı giriniz: 40
```

Yukarıdaki python kodunda kullanıcının girdiği yaş bilgisi öncelikle yas isimli değişkene aktarılır.

```
In [1]: yas=input("Yaşınızı giriniz:")
print("Yaşınız",yas)

Yaşınızı giriniz:40
Yaşınız 40
```

```
print("Yaşınız",yas)
```

Yukarıdaki python kodunda yas isimli değişkenin değeri önüne "Yaşınız" öneki getirilerek ekrana yazdırılır.



### input() Örnek-3

```
In [1]: print("Hello, I'm Python!")
```

```
Hello, I'm Python!
```

```
In [*]: name = input('What is your name?\n')
print('Hi, %s.' % name)
```

```
What is your name?
```

```
Serkan Korkmaz
```

```
name = input('What is your name?\n')
```

Yukarıdaki python kodunda kullanıcının girdiği isim bilgisi öncelikle name isimli değişkene aktarılır.

```
In [1]: print("Hello, I'm Python!")
```

```
Hello, I'm Python!
```

```
In [2]: name = input('What is your name?\n')
print('Hi, %s.' % name)
```

```
What is your name?
```

```
Serkan Korkmaz
```

```
Hi, Serkan Korkmaz.
```

```
print('Hi, %s.' % name)
```

Yukarıdaki python kodunda name isimli değişkenin değeri önüne "Hi," öneki ve sonuna "." işareti konularak ekrana yazdırılır.

print()fonksiyonundaki %s ifadesi yerine name değişkeninin değeri getirilir.

%s karakter kümesi değişken türündeki değişkenlerin değerini almak için kullanılır.

## Tip Dönüştürme İşlemleri

Python programlama dilinde tip öğrenme ve tip dönüştürme işleminde kullanılan fonksiyonlar bu kısımda anlatılacaktır.

### type() Fonksiyonu

type() fonksiyonu, parametre olarak girilen değerin hangi veri tipine ait olduğunu gösterir.

#### type() Örnek-1

type() fonksiyonu, parametre olarak girilen değerin hangi veri tipine ait olduğunu gösterir.

```
In [1]: type('erik')
```

```
Out[1]: str
```

```
In [2]: type(5)
```

```
Out[2]: int
```

```
In [1]: type(3.14)
```

```
Out[1]: float
```

#### type() Örnek-2

```
In [1]: #Kullanıcıdan herhangi bir veri girmesini istiyoruz
sayı = input("Herhangi bir veri girin: ")

#Kullanıcının girdiği verinin tipini bir
#değişkene atıyoruz
tip = type(sayı)

#Son olarak kullanıcının girdiği verinin tipini
#ekrana basıyoruz.
print("Girdiğiniz verinin tipi: ", tip)

Herhangi bir veri girin: 3
Girdiğiniz verinin tipi: <class 'str'>
```

input fonksiyonu ile girilen verinin hangi türde olduğunu gösteren bir uygulamadır.

Uygulama çalıştırıldığında hangi türden veri girilirse girilsin, type() fonksiyonu girilen bu verileri str (karakter dizisi) olarak değerlendirmektedir.

### type() Örnek-3

```
In [1]: a=3
        b=3.14
        c="Python"
        d=False
        e=3+15j

        f=[1,2,3,4,5,"Python"]
        g=(1,2,3,4,5,"Python")
        h={"Elma":3, "Armut":4, "Kiraz":5}

        print(type(a),a)
        print(type(b),b)
        print(type(c),c)
        print(type(d),d)
        print(type(e),e)

        print(type(f),f)
        print(type(g),g)
        print(type(h),h)

<class 'int'> 3
<class 'float'> 3.14
<class 'str'> Python
<class 'bool'> False
<class 'complex'> (3+15j)
<class 'list'> [1, 2, 3, 4, 5, 'Python']
<class 'tuple'> (1, 2, 3, 4, 5, 'Python')
<class 'dict'> {'Elma': 3, 'Armut': 4, 'Kiraz': 5}
```

type() fonksiyonu ile farklı veri türlerinde değere sahip olan değişkenlerin türünü gösteren bir uygulamadır.

## Tip Dönüştürme Fonksiyonları

input() fonksiyonundan gelen veri her zaman bir karakter dizisidir.

Girilen verilerin karakter dizisi şeklinde değil de farklı bir türde (int tamsayı gibi) değerlendirilmesi için tip dönüşümü yapılması gerekmektedir.

Aritmetik işlemler yapmak istediğimizde karakter dizilerini sayıya çevirmemiz gerekir.

Tabi ki bu durumun tersi için de aynı şeyleri ifade etmek mümkündür.

Her tamsayı ve/veya kayan noktalı sayı bir karakter dizisine dönüştürülebilir.

Ama her karakter dizisi tamsayıya ve/veya kayan noktalı sayıya dönüştürülemez.

Örneğin, 5654 gibi bir tamsayıyı veya 543.34 gibi bir kayan noktalı sayıyı str() fonksiyonu yardımıyla karakter dizisine dönüştürebiliriz.

Ancak “elma” gibi bir karakter dizisini ne int() ne de float() fonksiyonuyla tamsayıya veya kayan noktalı sayıya dönüştüremezsiniz. Çünkü “elma” verisi sayı değerli değildir.

Aşağıda verilen tabloda tip dönüştürme işlemlerinde kullanılan fonksiyonlar ve işlevleri açıklanmıştır.

Fonksiyon	İşlevi
<b>Tip Dönüşüm Fonksiyonları</b>	
str( )	Sayı değerli bir karakter dizisini veya tamsayıyı kayan noktalı sayıya ( <i>float</i> ) çevirir.
int( )	Sayı değerli bir karakter dizisini veya kayan noktalı sayıyı tamsayıya ( <i>integer</i> ) çevirir.
bool( )	Herhangi bir ifadeyi mantıksal veri türüne ( <i>bool</i> ) çevirir.
float( )	Bir tamsayı veya kayan noktalı sayıyı karakter dizisine ( <i>string</i> ) çevirir.
complex( )	Herhangi bir sayıyı veya sayı değerli karakter dizisini karmaşık sayıya ( <i>complex</i> ) çevirir.
ord()	Verilen bir karakteri, sayısal olarak ASCII değerine dönüştürür.
chr()	Ascii değeri sayısal olarak verilen bir değeri karaktere dönüştürür.

## str()

Python'daki tip dönüştürücüleri elbette sadece int() fonksiyonuyla sınırlı değildir. int() fonksiyonu sayı değerli verileri (mesela karakter dizilerini) tam sayıya dönüştürüyor. Bunun bir de tersi mümkündür.

Karakter dizisi olmayan verileri karakter dizisine dönüştürmemiz de mümkündür. Bu işlem için str() adlı başka bir tip dönüştürücüden yararlanılmaktadır.

### str() Örnek-1

```
In [1]: sayı = 15
        karakter = str(sayı)
        print(karakter)

15
```

```
In [2]: print(type(karakter))

<class 'str'>
```

Tam sayı olan 23'ü str() adlı bir fonksiyondan yararlanarak karakter dizisi olan "23" ifadesine dönüştürülmüştür. Son satırda da, elde ettiğimiz değer bir karakter dizisi olduğundan emin olmak için type() fonksiyonu kullanılmıştır.

### str() Örnek-2

```
In [1]: len(1234567890)
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-1-91d9757a72ea> in <module>
----> 1 len(1234567890)

TypeError: object of type 'int' has no len()
```

```
In [2]: sayı = 1234567890
        >>> karakter = str(sayı)
        >>> len(karakter)
```

```
Out[2]: 10
```

len()fonksiyonu sayılarla birlikte kullanılamaz.

Bu fonksiyon, parametre olarak girilen değer uzunluğunu gösterir.

## int()

Bu fonksiyonun görevi, bir veriyi tamsayıya dönüştürmektir.

input() fonksiyonundan gelen veri her zaman bir karakter dizisidir. Dolayısıyla bu fonksiyondan gelen veriyle herhangi bir aritmetik işlem yapabilmek için öncelikle bu veriyi bir sayıya dönüştürmemiz gerekir. Bu dönüştürme işlemi için int() adlı özel bir dönüştürücü fonksiyondan yararlanırız.

### int() Örnek-1

```
In [1]: veri = input("Lütfen bir sayı girin: ")
        sayı = int(veri)
        print("Girdiğiniz sayının karesi: ", sayı ** 2)

Lütfen bir sayı girin: 5
Girdiğiniz sayının karesi: 25
```

Kullanıcıdan aldığı sayının karesini hesaplayan kodlamada girilen 5 sayısının karesini alarak ekrana "Girdiğiniz sayının karesi: 25" yazdırılmıştır.

### int() Örnek-2

Ancak bazen öyle durumlarla karşılaşabilirsiniz ki, programınız hiçbir hata vermez. Ama elde edilen sonuç aslında tamamen beklentinizin dışındadır. Mesela şu basit örneği inceleyelim:

```
In [1]: sayı1 = input("Toplama işlemi için ilk sayıyı girin: ")
        sayı2 = input("Toplama işlemi için ikinci sayıyı girin: ")

        print(sayı1, "+", sayı2, "=", sayı1 + sayı2)

Toplama işlemi için ilk sayıyı girin: 12
Toplama işlemi için ikinci sayıyı girin: 34
12 + 34 = 1234

In [2]: "23" + "23"

Out[2]: '2323'
```

Bu kodları çalıştırdığımızda böyle bir manzara ile karşılaşırız.

Gördüğümüz gibi yukarıdaki program herhangi bir hata vermedi. Ama beklediğimiz çıktıyı da vermedi.

Zira biz programımızın iki sayıyı toplamasını istiyorduk. O ise kullanıcının girdiği sayıları yan yana yazmakla yetindi.

Yani bir aritmetik işlem yapmak yerine, verileri birbiriyle bitiştiirdi.

Çünkü input() fonksiyonunun kullanıcıdan aldığı şey bir karakter dizisidir.

Dolayısıyla bu fonksiyon yukarıdaki gibi bir durumla karşılaştığı zaman karakter dizileri arasında bir birleştirme işlemi gerçekleştirir.

Yazdığımız bir programın herhangi bir hata vermemesi o programın doğru çalıştığı anlamına gelmeyebilir.

Dolayısıyla bu tür durumlara karşı her zaman uyanık olmanızda fayda var.

Böylesi bir durumla karşılaşmamak için uygun tip dönüştürücü fonksiyonun kullanılması gerekmektedir.

### int() Örnek-3

```
In [1]: a=int(input("a:"))
        b=int(input("b:"))
        c=int(input("c:"))
        print("Toplam:",a+b+c)

a:3
b:4
c:5
Toplam: 12
```

### int() Örnek-4

```
In [1]: karakter_dizisi = "23"
        sayı = int(karakter_dizisi)
        print(sayı)

23
```

```
In [2]: karakter_dizisi = "elma"
        sayı = int(karakter_dizisi)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-2-bf955c83aede> in <module>
      1 karakter_dizisi = "elma"
----> 2 sayı = int(karakter_dizisi)

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

Yukarıdaki kodları satır satır açıklayalım.

```
karakter_dizisi = "23"
```

Öncelikle "23" adlı bir karakter dizisi tanımladık.

```
sayı = int(karakter_dizisi)
```

Ardından int() fonksiyonunu kullanarak bu karakter dizisini bir tamsayıya (integer) dönüştürdük. İsminden de anlayacağınız gibi int() fonksiyonu İngilizce integer (tamsayı) kelimesinin kısaltmasıdır ve bu fonksiyonun görevi bir veriyi tamsayıya dönüştürmektir.

Ancak burada dikkat etmemiz gereken bir şey var. Herhangi bir verinin sayıya dönüştürülebilmesi için o verinin sayı değerli bir veri olması gerekir. Örneğin "23", sayı değerli bir karakter dizisidir.

```
karakter_dizisi = "elma"
```

Ama mesela "elma" sayı değerli bir karakter dizisi değildir. Bu yüzden "elma" karakter dizisi sayıya dönüştürülemez.

```
sayı = int(karakter_dizisi)
```

Sayı değerli olmayan bir veriyi sayıya dönüştürmeye çalıştığımızda Python bize bir hata mesajı verecektir.

Yazılan programlarda bu tür durumlara özellikle dikkat etmek gerekecektir.

## int() Örnek-5 Vize Final Ortalama

```
In [1]: vize=int(input("Vize notunu giriniz:"))  
Vize notunu giriniz:60
```

```
In [2]: final=int(input("Final notunu giriniz:"))  
Final notunu giriniz:80
```

```
In [3]: vizesonuc=vize*0.4
```

```
In [10]: print("Vizenin %40'ı",vizesonuc)  
Vizenin %40'ı 24.0
```

```
In [4]: finalsonuc=final*0.6
```

```
In [7]: print("Finalin %60'ı",finalsonuc)  
Finalin %60'ı 48.0
```

```
In [8]: ortalama=vizesonuc+finalsonuc
```

```
In [9]: print("Ortalama:",ortalama)  
Ortalama: 72.0
```

```
vize=int(input("Vize notunu giriniz:"))
```

Yukarıdaki ifade ile klavyeden girilen vize notu (60) tamsayı tipine dönüştürülerek "vize" değişkenine aktarılır.

```
final=int(input("Final notunu giriniz:"))
```

Yukarıdaki ifade ile klavyeden girilen final notu (80) tamsayı tipine dönüştürülerek "final" değişkenine aktarılır.

```
vizesonuc=vize*0.4
```

Yukarıdaki ifade ile "vize" değişkeninin değerinin %40'ı alınarak "vizesonuc" değişkenine aktarılır.

```
print("Vizenin %40'ı",vizesonuc)
```

Yukarıdaki ifade ile "vizesonuc" değişkeninin değeri ekrana yazdırılır. "Vizenin %40'ı 24.0"

```
ortalama=vizesonuc+finalsonuc
```

Yukarıdaki ifadesi ile "vizesonuc" ve "finalsonuc" değişkenlerinin değeri toplanır.

```
print("Ortalama:",ortalama)
```

 ifadesi ile de ortalama değeri ekrana yazdırılır. "Ortalama: 72.0 "



## *bool()*

Herhangi bir verinin mantıksal karşılığını elde etmek için kullanılır.

bool() fonksiyonu, değişkeni Mantıksal olarak true (doğru) veya false (yanlış) olarak ifade eder.

### bool() Örnek-1

```
In [1]: bool(1)
```

```
Out[1]: True
```

```
In [2]: bool(0)
```

```
Out[2]: False
```

```
In [3]: bool("a")
```

```
Out[3]: True
```

```
In [4]: bool("")
```

```
Out[4]: False
```

Python dilinde değeri boş olan veya 0 olan veriler mantıksal olarak False değerine karşılık gelir.

Aynı şekilde değeri boş olmayan veya 0 olmayan veriler mantıksal olarak True değerine karşılık gelir.

### bool() Örnek-2

```
In [1]: x = "True"  
y=bool(x)  
print(y)  
type(y)
```

```
True
```

```
Out[1]: bool
```

Python'da mantıksal değişkenlerin tipi 'bool' olarak ifade edilir.

### bool() Örnek-3

```
In [1]: t, f = True, False
print (type(t))
print (type(f))

<class 'bool'>
<class 'bool'>
```

t, f = True, False

t mantıksal değişkeni tanımlanmış ve değeri True olarak belirlenmiştir.

f mantıksal değişkeni tanımlanmış ve değeri False olarak belirlenmiştir.

### bool() Örnek-4

```
In [6]: t, f = True, False
print ("True and False:", t and f )
print ("True or False:", t or f)
print ("not True:", not t)
print ("not False:", not f)
print ("True equal False:", t == f)
print ("True not equal False:", t != f)

True and False: False
True or False: True
not True: False
not False: True
True equal False: False
True not equal False: True
```

t mantıksal değişkeni tanımlanmış ve değeri True olarak belirlenmiştir.

f mantıksal değişkeni tanımlanmış ve değeri False olarak belirlenmiştir.

t ve f mantıksal değişkenleri ile mantıksal operatörler (and, or, not) kullanılmıştır.

t ve f mantıksal değişkenleri ile karşılaştırma operatörleri (== , !=) kullanılmıştır.

## float()

Tamsayıların (integer) dışında kalan sayılar kayan noktalı sayılar (float) olarak ifade edilir. Bir tamsayıyı veya sayı değerli bir karakter dizisini kayan noktalı sayıya dönüştürmek için float() fonksiyonu kullanılır.

### float() Örnek-1

```
In [1]: a = 23  
        type(a)
```

```
Out[1]: int
```

```
In [2]: float(a)
```

```
Out[2]: 23.0
```

### float() Örnek-2

```
In [1]: b = "23"  
        type(b)
```

```
Out[1]: str
```

```
In [2]: float(b)
```

```
Out[2]: 23.0
```

### float() Örnek-3

```
In [1]: num1 = 10  
        num2 = 8.5  
  
        # iki sayının toplamı  
        sum = int(num1) + float(num2)  
  
        # sonucu yazdırma  
        print('{0} + {1} = {2}'.format(num1, num2, sum))  
  
10 + 8.5 = 18.5
```

### float() Örnek-4

```
In [1]: num1 = int(input("birinci sayıyı giriniz:"))  
        num2 = int(input("ikinci sayıyı giriniz:"))  
  
        # iki sayının toplamı  
        sum = int(num1) + float(num2)  
  
        # sonucu yazdırma  
        print('{0} + {1} = {2}'.format(num1, num2, sum))  
  
10 + 8.5 = 18.5
```

Python'da kullanıcı tarafından girilen iki sayının toplamını yazdırma

### *complex()*

Herhangi bir sayıyı karmaşık sayıya dönüştürmeniz gerekirse `complex()` adlı bir fonksiyondan yararlanabilirsiniz.

#### `complex()` Örnek-1

```
In [1]: type(12+0j)
Out[1]: complex
```

Karmaşık sayılar Python'da 'complex' ifadesiyle gösterilir.

12+0j ifadesinin bir karmaşık sayı olduğunu anlıyoruz.

#### `complex()` Örnek-2

Herhangi bir sayıyı karmaşık sayıya dönüştürmeniz gerekirse `complex()` adlı bir fonksiyondan yararlanabiliriz.

```
In [1]: complex(15)
Out[1]: (15+0j)
```

### *ord()*

Verilen bir karakteri, sayısal olarak ASCII değerine dönüştürür.

#### ord() Örnek-1

```
In [1]: ord('A')  
Out[1]: 65
```

### *chr()*

Sayısal olarak verilen ASCII değerini, karaktere dönüştürür.

#### chr() Örnek-1

```
In [1]: chr(65)  
Out[1]: 'A'
```

```
In [2]: chr(97)  
Out[2]: 'a'
```

#### chr() Örnek-2

```
In [1]: for i in range(65,91):  
        print(i, chr(i), sep="-")  
  
65-A  
66-B  
67-C  
68-D  
69-E  
70-F  
71-G  
72-H  
73-I  
74-J  
75-K  
76-L  
77-M  
78-N  
79-O  
80-P  
81-Q  
82-R  
83-S  
84-T  
85-U  
86-V  
87-W  
88-X  
89-Y  
90-Z
```

### chr() Örnek-3

```
In [1]: for i in range(48,58):  
        print(i, chr(i), sep="-")
```

```
48-0  
49-1  
50-2  
51-3  
52-4  
53-5  
54-6  
55-7  
56-8  
57-9
```

### chr() Örnek-4

```
In [1]: for i in range(97,123):  
        print(i, chr(i), sep="-")
```

```
97-a  
98-b  
99-c  
100-d  
101-e  
102-f  
103-g  
104-h  
105-i  
106-j  
107-k  
108-l  
109-m  
110-n  
111-o  
112-p  
113-q  
114-r  
115-s  
116-t  
117-u  
118-v  
119-w  
120-x  
121-y  
122-z
```

## Metinsel İşlemler

Python programlama dilinde metinsel ifadeler için kullanılan karakter indeks değerleri ve metinsel işlem fonksiyonları bu kısımda anlatılacaktır.

### Karakter İndeksleri

Python programlama dilinde metinsel ifadeler için kullanılan karakter indeks değerleri aşağıdaki tabloda belirtilmiştir.

Karakter	İndeks Değeri
<i>Pozitif indeks</i>	
İlk Karakter	0
2. Karakter	1
3. Karakter	2
<i>Negatif indeks</i>	
Son Karakter	-1
Sondan 2. Karakter	-2
<i>Aralıklı indeks</i>	
2. Karakterden itibaren 5. Karaktere kadar (5 dahil değil)	2:5

Karakter İndeksleri Örnek-1

```
In [1]: word = 'Python'
        word[0] # character in position 0
Out[1]: 'p'

In [2]: word[5] # character in position 5
Out[2]: 'n'

In [3]: word[-1] # last character
Out[3]: 'n'

In [4]: word[-2] # second-last character
Out[4]: 'o'

In [5]: word[-6]
Out[5]: 'P'

In [6]: word[0:2] # characters from position 0 (included) to 2 (excluded)
Out[6]: 'Py'

In [7]: word[2:5] # characters from position 2 (included) to 5 (excluded)
Out[7]: 'tho'

In [8]: word[:2] + word[2:] # s[:i] + s[i:] is always equal to s:
Out[8]: 'Python'

In [9]: word[:4] + word[4:] # s[:i] + s[i:] is always equal to s:
Out[9]: 'Python'
```

## Metinsel İşlem Fonksiyonları

Python programlama dilinde metinsel ifadeler için kullanılan metinsel işlem fonksiyonları aşağıda belirtilmiştir.

Fonksiyon	İşlevi
<i>Metinsel İşlem Fonksiyonları</i>	
capitalize()	İlk harfi büyük yapar
upper()	Metinde geçen karakterlerin tümünü büyük harf yapar.
lower()	Metinde geçen karakterlerin tümünü küçük harf yapar.
replace()	Metinde geçen bir ifadeyi başka bir ifadeyle değiştirir.
split()	Metni boşluklara göre veya belirli bir karaktere göre ayırır.
strip()	Metinde geçen boşlukları veya belirli bir karakteri metinden kaldırır.
startswith()	Parametrede verilen değerle başlıyor mu?
endswith()	Parametrede verilen değerle bitiyor mu?
len()	Metinde geçen karakterlerin sayısını (uzunluğunu) gösterir.
rjust(x)	X genişlikli bir alanda sağa hizalar.
ljust(x)	X genişlikli bir alanda sola hizalar.
center(x)	.X genişlikli bir alanda ortalar.

### Metinsel İşlem Fonksiyonları Örnek-1

```
In [1]: kelime = "yapay zeka"
print (kelime.capitalize()) # İlk harf büyük "Yapay zeka"
print (kelime.upper())     # Hepsi büyük "YAPAY ZEKA"
print (kelime.rjust(15))   # Sağa yasla
print (kelime.center(15)) # Merkeze ortala
print (kelime.replace('zeka', 'zekalar')) # "zeka"ları "zekalar" ile değiştirelim
print (' yapay zeka önemlidir :').strip() # boşlukları silmek için strip() kullanılır

Yapay zeka
YAPAY ZEKA
 yapay zeka
 yapay zeka
 yapay zekalar
 yapay zeka önemlidir :)
```

### Metinsel İşlem Fonksiyonları Örnek-2

```
In [1]: metin="araba"
print(metin.startswith("a"))
```

True

```
In [2]: print(metin.startswith("b"))
```

False

```
In [3]: print(metin.startswith("ar"))
```

True

```
In [4]: print(metin.endswith("a"))
```

True

```
In [5]: print(metin.endswith("s"))
```

False

```
In [6]: print(metin.replace("a", "e"))
```

erebe



### *split() fonksiyonu*

split() fonksiyonu cümle içerisinde geçen ifadeyi boşluklara göre ayırır.

Bu metodun görevi karakter dizilerini belli noktalardan bölmektir. Zaten split kelimesi Türkçede 'bölmek, ayırmak' gibi anlamlara gelir. İşte bu metod, üzerine uygulandığı karakter dizilerini parçalarına ayırır.

split() metodunu herhangi bir parametre içermeyecek şekilde kullandığımızda, yani metodun parantezleri içine herhangi bir şey eklemediğimizde split() metodukarakter dizilerini bölerken boşluk karakterini ölçüt alacaktır. Yani karakter dizisi içinde karşılaştığı her boşluk karakterinde bir bölme işlemi uygulayacaktır.

Ama bazen istediğimiz şey, bir karakter dizisini boşluklardan bölmek değildir. split() metoduna hangi parametreyi verirseniz bu metod ilgili karakter dizisini o karakterin geçtiği yerlerden bölecektir. Yani mesela siz split() metodunda “,” parametresini verirseniz, split() metodu ‘,’ karakteri geçen yerden karakter dizisini bölecektir.

#### split Örnek-1

```
In [1]: for kelime in "Yapay Zeka".split():  
        print(kelime)
```

```
Yapay  
Zeka
```

Yukarıdaki kodda “Yapay Zeka” içerisindeki tüm sözcükler alt alta gelecek şekilde ekrana yazdırılır.

#### split Örnek-2

```
In [1]: metin="İstanbul Büyükşehir Belediyesi"  
        metin.split()
```

```
Out[1]: ['İstanbul', 'Büyükşehir', 'Belediyesi']
```

```
In [2]: for i in metin.split():  
        print(i)
```

```
İstanbul  
Büyükşehir  
Belediyesi
```

```
In [3]: for x in metin.split():  
        print(x[0],end="")
```

```
İBB
```

```
In [4]: kurum = input("Kısaltılacak metni girin: ")  
  
        for k in kurum.split():  
            print(k[0], end="")
```

```
Kısaltılacak metni girin: Türkiye Büyük Millet Meclisi  
TBMM
```

```
In [5]: kardiz = "Gaziantep, Kilis, Şanlıurfa, Adana, Kahramanmaraş"  
        kardiz = kardiz.split(",")  
        print(kardiz)
```

```
['Gaziantep', ' Kilis', ' Şanlıurfa', ' Adana', ' Kahramanmaraş']
```

### *strip()* fonksiyonu

Bazı durumlarda kullanıcıdan ya da başka kaynaktan gelen karakter dizilerinde bu tür istenmeyen boşluklar olabilir. Ama sizin kullanıcıdan veya başka bir kaynaktan gelen o karakter dizisini düzgün kullanabilmeniz için öncelikle o karakter dizisinin sağında ve solunda bulunan boşluk karakterlerinden kurtulmanız gerekebilir. İşte böyle anlarda strip() metodu yardımınıza yetişecektir.

strip() metodunu kullanarak, karakter dizisinin orijinalinde bulunan sağlı sollu boşluk karakterlerini ortadan kaldırırız.

strip() metodu parametresiz olarak kullanıldığında, bir karakter dizisinin sağında veya solunda bulunan belli başlı karakterleri kırpar. strip() metodunun ön tanımlı olarak kırptığı karakterler şunlardır:

- ‘ ‘ boşluk karakteri
- \t sekme (TAB) oluşturan kaçış dizisi
- \n satır başına geçiren kaçış dizisi
- \r imleci aynı satırın başına döndüren kaçış dizisi
- \v düşey sekme oluşturan kaçış dizisi
- \f yeni bir sayfaya geçiren kaçış dizisi

strip() metodunu kullanarak, karakter dizisinin orijinalinde bulunan sağlı sollu boşluk karakterlerini ortadan kaldırır. Ancak eğer biz istersek strip() metoduna bir parametre vererek bu metodun istediğimiz herhangi başka bir karakteri kırpmasını da sağlayabiliriz.

#### strip Örnek-1

```
In [1]: kardiz = " istihza "  
print(kardiz)
```

```
istihza
```

```
In [2]: metin=" python "  
metin.strip()  
metin
```

```
Out[2]: ' python '
```

```
In [3]: metin = " python "  
metin=metin.strip()  
metin
```

```
Out[3]: 'python'
```

```
In [4]: kardiz = "kazak"  
kardiz=kardiz.strip("k")  
kardiz
```

```
Out[4]: 'aza'
```

## *len() fonksiyonu*

len() fonksiyonu, parametre olarak girilen değerin karakter sayısını (uzunluğunu) gösterir.

### len() Örnek-1

```
In [1]: a="araba"  
print(len(a))
```

5

```
In [2]: liste=[1,2,3,4,5,6]  
print(len(liste))
```

6

### len() Örnek-2

```
In [*]: ad=input("Adınızı Giriniz:")  
print("Adınız:",ad,len(ad),"karakterlidir.")
```

Adınızı Giriniz:

```
In [1]: ad=input("Adınızı Giriniz:")  
print("Adınız:",ad,len(ad),"karakterlidir.")
```

Adınızı Giriniz:Serkan Korkmaz  
Adınız: Serkan Korkmaz 14 karakterlidir.

## Matematiksel İşlemler

Python programlama dilinde matematiksel işlem fonksiyonları bu kısımda anlatılacaktır.

### Matematiksel İşlem Fonksiyonları

Python programlama dilinde kullanılan bazı özel amaçlı fonksiyonlar bu kısımda anlatılmıştır.

math modülü matematiksel işlemler yapmanızı kolaylaştırmak için yazılmış bir modüldür.<sup>10</sup>

#### *min()* fonksiyonu

min() fonksiyonu, parametre olarak verilen değerler içerisinde en küçük değeri hesaplar.

##### min () Örnek-1

```
In [2]: # min() fonksiyonu verilen değerler içindeki en küçük değeri hesaplar  
min(5,10,25)
```

```
Out[2]: 5
```

```
In [3]: min(3,4,1,-2,15)
```

```
Out[3]: -2
```

```
In [4]: min(5,10,25,1)
```

```
Out[4]: 1
```

```
In [5]: min(3.5,1.5,2,4,6.8)
```

```
Out[5]: 1.5
```

#### *max()* fonksiyonu

max() fonksiyonu, parametre olarak verilen değerler içerisinde en büyük değeri hesaplar.

##### max () Örnek-1

```
In [6]: # max() fonksiyonu verilen değerler içindeki en büyük değeri hesaplar  
max(5,10,25)
```

```
Out[6]: 25
```

```
In [7]: max(3,4,1,-2,15)
```

```
Out[7]: 15
```

```
In [8]: max(5,10,25,35)
```

```
Out[8]: 35
```

```
In [9]: max(3.5,1.5,2,4,6.8,8.3)
```

```
Out[9]: 8.3
```

<sup>10</sup>[https://belgeler.yazbel.com/python-istihza/standart\\_moduller/math.html](https://belgeler.yazbel.com/python-istihza/standart_moduller/math.html)

## *pow()* fonksiyonu

pow() fonksiyonu, ilk parametrede girilen sayı değerinin ikinci parametredeki sayı değeri kadar kuvvetini hesaplar.

### pow ( ) Örnek-1

```
In [1]: pow(3,2)
```

```
Out[1]: 9
```

```
In [2]: pow(4,1/2)
```

```
Out[2]: 2.0
```

```
In [3]: # 16 sayısının 2. kuvvetini hesapla.  
# çıkan sayının 3'e bölümünden kalanı göster  
pow(16, 2, 3)
```

```
Out[3]: 1
```

```
In [4]: # 11 sayısının 3. kuvvetini hesapla.  
# çıkan sayının 4'e bölümünden kalanı göster  
pow(11, 3, 4)
```

```
Out[4]: 3
```

pow() fonksiyonu ile yapılan ilk örnekte  $3^2$  işlemi gerçekleştirilmiştir.

pow() fonksiyonu ile yapılan ikinci örnekte  $4^{1/2}$  işlemi gerçekleştirilmiştir.

pow() fonksiyonu, pek kullanılmayan üçüncü bir parametre daha alır.

Bu fonksiyonun üçüncü parametresi şöyle kullanılır.

# 16 sayısının 2. kuvvetini hesaplar. Çıkan sayının 3'e bölümünden kalanı gösterir.

```
pow(16, 2, 3)
```

# 11 sayısının 3. kuvvetini hesaplar. Çıkan sayının 4'e bölümünden kalanı gösterir.

```
pow(11, 3, 4)
```

# 25 sayısının 2. kuvvetini hesaplar. Çıkan sayının 5'e bölümünden kalanı gösterir.

```
pow(25, 2, 5)
```

## *round() fonksiyonu*

round() fonksiyonu, parametre olarak girilen ondalıklı sayı değerini yuvarlar.

round() fonksiyonuna iki parametre verilebilir. Verilen ilk parametre yuvarlanacak sayıyı, ikinci parametre ise yuvarlanacak sayının virgülden sonraki hassasiyetini belirtir.

### round() Örnek-1

```
In [1]: round(2.55)
```

```
Out[1]: 3
```

```
In [2]: round(2.55, 1)
```

```
Out[2]: 2.5
```

```
In [3]: round(2.68, 1)
```

```
Out[3]: 2.7
```

```
In [4]: round(2.68, 2)
```

```
Out[4]: 2.68
```

round(2.55)

2.55 sayısı yuvarlanır ve 3 sonucunu döndürülür.

round(2.55, 1)

2.55 sayısı virgülden sonra 1 basamak olacak şekilde yuvarlanır ve 2.5 sonucunu döndürülür.

round(2.68, 1)

2.68 sayısı virgülden sonra 1 basamak olacak şekilde yuvarlanır ve 2.7 sonucunu döndürülür.

round(2.68, 2)

2.68 sayısı virgülden sonra 2 basamak olacak şekilde yuvarlanır ve 2.68 sonucunu döndürülür.

round() fonksiyonunun çalışma prensibini daha iyi anlamak için kendi kendinize örnekler yapabilirsiniz.

### *divmod() fonksiyonu*

divmod() fonksiyonu, parametre olarak girilen ilk sayı değerinin, parametre olarak girilen ikinci sayı değerine bölünmesinden sonra tam kısmı ve kalan kısmı verir.

#### divmod() Örnek-1

```
In [1]: #43 7'e bölündüğünde tam ve kalanı verir.|\n        divmod(43,7)\n\nOut[1]: (6, 1)
```

43'ün 7'ye bölünmesi işleminin ardından işlemin sonucu olarak tam kısım 6 ve kalan 1 olarak bulunur.

### *abs() fonksiyonu*

abs() fonksiyonu, parametreye girilen sayı değerinin mutlak değerini alır.

Mutlak değer hesaplanırken girilen sayı değeri negatif de olsa pozitif de olsa sonuç her zaman pozitif olacaktır.

#### abs() Örnek-1

```
In [1]: abs(-5)
```

```
Out[1]: 5
```

```
In [2]: abs(5)
```

```
Out[2]: 5
```

```
In [3]: abs(-3.7)
```

```
Out[3]: 3.7
```

```
In [4]: abs(3.7)
```

```
Out[4]: 3.7
```

## math() modülü

Python programlama dilinde matematiksel işlemler yapmayı kolaylaştırmak için yazılmış math modülü kullanılır.

math modülünü kullanmak için öncelikle modülü import etmek gerekecektir.

```
import math
```

Yukarıdaki işlemin ardından math modülünün sahip olduğu tüm fonksiyonlar kullanılabilir.

### *fabs() fonksiyonu*

fabs() fonksiyonu, abs() fonksiyonu gibi parametrede girilen sayı değerinin mutlak değerini alır.

Mutlak değer hesaplanırken girilen sayı değeri negatif de olsa pozitif de olsa sonuç her zaman pozitif olacaktır.

abs'den küçük bir farkı var. Çıktısını tam sayı olarak değil ondalıklı sayı olarak döndür.

### fabs () Örnek-1

```
In [1]: import math  
math.fabs(-28)
```

```
Out[1]: 28.0
```

```
In [2]: math.fabs(-15)
```

```
Out[2]: 15.0
```

```
In [3]: abs(-15)
```

```
Out[3]: 15
```

### *sqrt() fonksiyonu*

sqrt() fonksiyonu, parametre olarak girilen sayı değerinin karekökünü alır.

### sqrt() Örnek-1

```
In [1]: import math  
math.sqrt(4)
```

```
Out[1]: 2.0
```

```
In [2]: math.sqrt(225)
```

```
Out[2]: 15.0
```

```
In [3]: math.sqrt(10)
```

```
Out[3]: 3.1622776601683795
```



### *fmod() fonksiyonu*

fmod() fonksiyonu, parametre olarak girilen ilk sayı değerinin, parametre olarak girilen ikinci sayı değerine bölünmesinden kalan kısmı verir. Çıktısını tam sayı olarak değil ondalıklı sayı olarak döndürüyor.

% operatöründen farkı negatif sayılarda ortaya çıkıyor. fmod() fonksiyonu negatif sayılarda sayının mutlak değerine göre işlem yapar.

#### fmod() Örnek-1

```
In [1]: #math matematiksel işlemler modülü
import math
math.fmod(43,7)
```

```
Out[1]: 1.0
```

```
In [2]: math.fmod(45,2)
```

```
Out[2]: 1.0
```

```
In [3]: math.fmod(45,14)
```

```
Out[3]: 3.0
```

```
In [4]: math.fmod(45,-14) # negatif değerlerlilerde mutlak değerli işlem yapar
```

```
Out[4]: 3.0
```

```
In [5]: 45%-14 # negatif değerlerlilerde beklenen değer
```

```
Out[5]: -11
```

43'ün 7'ye bölünmesi işleminin ardından kalan 1.0 olarak bulunur.

45'in 2'ye bölünmesi işleminin ardından kalan 1.0 olarak bulunur.

45'in 14'e bölünmesi işleminin ardından kalan 3.0 olarak bulunur.

### *copysign() fonksiyonu*

copysign () fonksiyonu, aldığı 2 parametreden ikincisinin işaretini birincisine verir.

#### copysign() Örnek-1

```
In [1]: import math
math.copysign(25,-12)
```

```
Out[1]: -25.0
```

```
In [2]: math.copysign(-12,-15)
```

```
Out[2]: -12.0
```

```
In [3]: math.copysign(-245,54)
```

```
Out[3]: 245.0
```

### *math.pow() fonksiyonu*

math.pow() fonksiyonu, ilk parametrede girilen sayı değerinin ikinci parametredeki sayı değeri kadar kuvvetini hesaplar.

pow() fonksiyonu çıktı olarak tamsayı değer döndürürken, math.pow() fonksiyonu ondalıklı sayı değeri döndürür.

#### math.pow ( ) Örnek-1

```
In [1]: import math  
math.pow(2,5) # math.pow() fonksiyonu ondalıklı sayı değeri döndürür.
```

```
Out[1]: 32.0
```

```
In [2]: pow(2,5) # pow() fonksiyonu tamsayı değeri döndürür.
```

```
Out[2]: 32
```

```
In [3]: math.pow(2,0)
```

```
Out[3]: 1.0
```

```
In [4]: pow(2,0)
```

```
Out[4]: 1
```

```
In [5]: math.pow(3,2)
```

```
Out[5]: 9.0
```

### *factorial() fonksiyonu*

factorial() fonksiyonu, parametre olarak verilen pozitif tamsayının faktoriyelini hesaplar.

Parametre olarak verilen değer pozitif tamsayı değilse ValueError hatası verir.

#### factorial ( ) Örnek-1

```
In [1]: import math  
math.factorial(5)
```

```
Out[1]: 120
```

```
In [2]: math.factorial(-5)
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-2-a46d876612ec> in <module>  
----> 1 math.factorial(-5)
```

```
ValueError: factorial() not defined for negative values
```

### *ceil() fonksiyonu*

ceil() fonksiyonu, parametre olarak verilen ondalıklı sayıyı bir üst sayıya çevirir.

Parametre olarak verilen değer tamsayı ise sonuç değişmez olduğu gibi kalır.

#### ceil ( ) Örnek-1

```
In [1]: import math  
        math.ceil(32.05)
```

```
Out[1]: 33
```

```
In [2]: math.ceil(2.98)
```

```
Out[2]: 3
```

```
In [3]: math.ceil(2)
```

```
Out[3]: 2
```

```
In [4]: math.ceil(32)
```

```
Out[4]: 32
```

### *floor() fonksiyonu*

floor() fonksiyonu, parametre olarak verilen ondalıklı sayıyı bir alt sayıya çevirir.

Parametre olarak verilen değer tamsayı ise sonuç değişmez olduğu gibi kalır.

#### floor ( ) Örnek-1

```
In [1]: import math  
        math.floor(25.42)
```

```
Out[1]: 25
```

```
In [2]: math.floor(-12.25)
```

```
Out[2]: -13
```

```
In [3]: math.floor(25)
```

```
Out[3]: 25
```

```
In [4]: math.floor(-12)
```

```
Out[4]: -12
```

### *fsum() fonksiyonu*

fsum() fonksiyonu, sum() fonksiyonu gibi parametre olarak verilen sayıların toplamını alır.

sum() fonksiyonu ondalıklı sayılarla çalışırken sorun çıkarabilir.

fsum() fonksiyonu, sum() fonksiyonundaki bu açığı kapatır.

#### fsum ( ) Örnek-1

```
In [1]: import math  
math.fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
```

```
Out[1]: 1.0
```

```
In [2]: sum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
```

```
Out[2]: 0.9999999999999999
```

### *gcd() fonksiyonu*

gcd() fonksiyonu, parametre olarak verilen iki sayının OBEB'ini hesaplar.

#### gcd ( ) Örnek-1

```
In [1]: import math  
math.gcd(45,70)
```

```
Out[1]: 5
```

```
In [2]: math.gcd(24,12)
```

```
Out[2]: 12
```

```
In [3]: math.gcd(24,-24)
```

```
Out[3]: 24
```

### *trunc() fonksiyonu*

trunc() fonksiyonu, int() fonksiyonu ile aynı işi yaparak parametre olarak verilen sayının tam kısmını alır.

#### trunc ( ) Örnek-1

```
In [1]: import math  
math.trunc(15.12)
```

```
Out[1]: 15
```

```
In [2]: math.trunc(-15.12)
```

```
Out[2]: -15
```

```
In [3]: int(-15.12)
```

```
Out[3]: -15
```

### *radians()* fonksiyonu

radians() fonksiyonu parametre olarak girilen sayıyı dereceden radyana çevirir.

$$\frac{\text{Derece}}{180} = \frac{\text{Radyan}}{\pi}$$

#### radians() Örnek-1

```
In [1]: import math  
        math.radians(90)
```

```
Out[1]: 1.5707963267948966
```

```
In [2]: math.radians(180)
```

```
Out[2]: 3.141592653589793
```

### *degrees()* fonksiyonu

degrees() fonksiyonu, parametre olarak girilen sayıyı radyandan dereceye çevirir.

#### degrees() Örnek-1

```
In [1]: import math  
        math.degrees(1.5707963267948966)
```

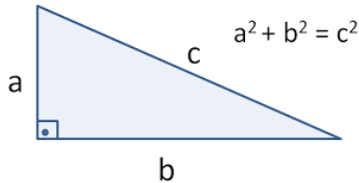
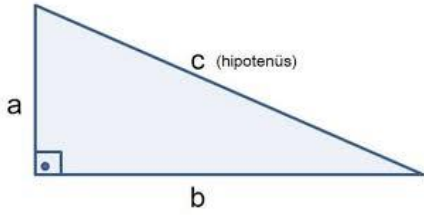
```
Out[1]: 90.0
```

```
In [2]: math.degrees(3.141592653589793)
```

```
Out[2]: 180.0
```

### *hypot()* fonksiyonu

hypot() fonksiyonu, 2 dik kenarı parametre olarak verilen dik üçgenin hipotenüsünü hesaplar.



#### hypot() Örnek-1

```
In [1]: import math  
        math.hypot(3,4)
```

```
Out[1]: 5.0
```

```
In [2]: math.hypot(5,12)
```

```
Out[2]: 13.0
```

### *sin() fonksiyonu*

sin() fonksiyonu, radyan cinsinden verilen sayının sinüsünü hesaplar.

#### sin() Örnek-1

```
In [1]: import math  
        math.radians(30)
```

```
Out[1]: 0.5235987755982988
```

```
In [2]: math.sin(0.5235987755982988)
```

```
Out[2]: 0.49999999999999994
```

```
In [3]: math.sin(math.radians(30))
```

```
Out[3]: 0.49999999999999994
```

### *cos() fonksiyonu*

cos() fonksiyonu, radyan cinsinden verilen sayının kosinüsünü hesaplar.

```
In [1]: import math  
        math.radians(30)
```

```
Out[1]: 0.5235987755982988
```

```
In [2]: math.cos(0.5235987755982988)
```

```
Out[2]: 0.8660254037844387
```

```
In [3]: math.cos(math.radians(30))
```

```
Out[3]: 0.8660254037844387
```

### *tan() fonksiyonu*

tan() fonksiyonu, radyan cinsinden verilen sayının tanjantını hesaplar.

```
In [1]: import math  
        math.radians(30)
```

```
Out[1]: 0.5235987755982988
```

```
In [2]: math.tan(0.5235987755982988)
```

```
Out[2]: 0.5773502691896257
```

```
In [3]: math.tan(math.radians(30))
```

```
Out[3]: 0.5773502691896257
```

### *e ve pi sabitleri*

Pi sabitini ve e (euler) sabitini kullanabilmek için math modülünü import etmek gerekir.

math.pi # pi sabitinin değerini döndürür.

math.e # e sabitinin değerini döndürür.

#### e ve pi sabiti Örnek-1

```
In [1]: import math #math matematiksel işlemler modülü
```

```
In [2]: math.pi # pi sabiti
```

```
Out[2]: 3.141592653589793
```

```
In [3]: math.e # e sabiti
```

```
Out[3]: 2.718281828459045
```

### *exp() fonksiyonu*

exp() fonksiyonu, e (euler) sabitinin parametre olarak girilen sayı değeri kadar kuvvetini alır.

math.exp(x) # e sabiti için  $e^x$  değerini döndürür.

#### exp() Örnek-1

```
In [1]: import math  
math.exp(1)
```

```
Out[1]: 2.718281828459045
```

```
In [2]: math.exp(2)
```

```
Out[2]: 7.38905609893065
```

```
In [3]: math.exp(3)
```

```
Out[3]: 20.085536923187668
```

### *expm1() fonksiyonu*

expm1() fonksiyonu, e (euler) sabitinin parametre olarak girilen sayı değeri kadar kuvvetini aldıktan sonra üretilen değerden 1 çıkarılır.

math.expm1(x) # e sabiti için  $e^x-1$  değerini döndürür.

#### expm1() Örnek-1

```
In [1]: import math  
math.expm1(1)
```

```
Out[1]: 1.718281828459045
```

```
In [2]: math.e
```

```
Out[2]: 2.718281828459045
```

```
In [3]: math.exp(1)
```

```
Out[3]: 2.718281828459045
```

```
In [4]: math.exp(1)-1
```

```
Out[4]: 1.718281828459045
```

```
In [5]: math.exp(2)
```

```
Out[5]: 7.38905609893065
```

```
In [6]: math.exp(2)-1
```

```
Out[6]: 6.38905609893065
```

```
In [7]: math.expm1(2)
```

```
Out[7]: 6.38905609893065
```



### *log() fonksiyonu*

log() fonksiyonunu kullanabilmek için math modülünü import etmek gerekir.

log() fonksiyonu, parametre olarak girilen ilk sayı değerinin, parametre olarak girilen ikinci sayı değerine göre logaritmasını alır.

#### log() Örnek-1

```
In [1]: import math  
        math.log(10,10)
```

```
Out[1]: 1.0
```

```
In [2]: math.log(25,5)
```

```
Out[2]: 2.0
```

```
In [3]: math.log(5,25)
```

```
Out[3]: 0.5
```

### *log2() fonksiyonu*

log2() fonksiyonunu kullanabilmek için math modülünü import etmek gerekir.

log2() fonksiyonu, parametre olarak girilen sayı değerinin 2 tabanında logaritmasını hesaplar.

#### log2() Örnek-1

```
In [1]: import math  
        math.log2(2)
```

```
Out[1]: 1.0
```

```
In [2]: math.log2(8)
```

```
Out[2]: 3.0
```

```
In [3]: math.log2(42)
```

```
Out[3]: 5.392317422778761
```

### *log10() fonksiyonu*

log10() fonksiyonunu kullanabilmek için math modülünü import etmek gerekir.

log10() fonksiyonu, parametre olarak girilen sayı değerinin 10 tabanında logaritmasını hesaplar.

#### log10() Örnek-1

```
In [1]: import math  
        math.log10(1000)
```

```
Out[1]: 3.0
```

```
In [2]: math.log10(100)
```

```
Out[2]: 2.0
```

```
In [3]: math.log10(10)
```

```
Out[3]: 1.0
```

### *log1p() fonksiyonu*

log1p() fonksiyonunu kullanabilmek için math modülünü import etmek gerekir.

log1p() fonksiyonu, parametre olarak girilen sayı değerinin 1 fazlasının e tabanında logaritmasını hesaplar.

#### log1p() Örnek-1

```
In [1]: import math  
        math.log1p(0)
```

```
Out[1]: 0.0
```

```
In [2]: math.log1p(2)
```

```
Out[2]: 1.0986122886681098
```

```
In [3]: math.log1p(math.e - 1)
```

```
Out[3]: 1.0
```

```
In [4]: math.log1p(math.exp(2)-1)
```

```
Out[4]: 2.0
```

```
In [5]: math.log1p(math.expm1(5))
```

```
Out[5]: 5.0
```

## Rastgele Sayılar

Python programlama dilinde belirli aralıkta rastgele sayıların üretilmesi işlemleri bu kısımda anlatılacaktır.

### random() modülü

Python programlama dilinde belirli aralıkta rastgele sayılar üretmek için random modülü kullanılır.

Random modülünü kullanmak için öncelikle modülü import etmek gerekecektir.

```
import random
```

Yukarıdaki işlemin ardından random modülünün sahip olduğu tüm fonksiyonlar kullanılabilir.

### random() fonksiyonu

random() fonksiyonu kullanılarak 0.0 ile 1.0 arasında rastgele ondalıklı sayı üretebiliriz.

random() fonksiyonu her çalıştırıldığında farklı bir ondalıklı sayı üretilir.

### random () Örnek-1

```
In [1]: import random  
        random.random()
```

```
Out[1]: 0.51521006720371
```

```
In [2]: import random  
        random.random()
```

```
Out[2]: 0.353646577697887
```

### random () Örnek-2

```
In [1]: import random  
        for i in range(10):  
            print("{:.4f}".format(random.random()))
```

```
0.1956  
0.7452  
0.1671  
0.3016  
0.4297  
0.8127  
0.8834  
0.0251  
0.5040  
0.9515
```

Yukarıdaki örnekte random() fonksiyonu ile 0 ile 1 arasında 10 adet sayı üretilmiştir. Üretilen sayılar 4 basamaklı olarak gösterilmektedir.

### *uniform() fonksiyonu*

uniform() fonksiyonu kullanılarak istenilen aralıkta rastgele ondalıklı sayı üretebiliriz.

#### uniform ( ) Örnek-1

```
In [1]: import random
random.uniform(0.5 , 1.5)

Out[1]: 1.2816232273176293
```

```
In [2]: import random
random.uniform(0.5 , 1.5)

Out[2]: 0.6573159936278538
```

```
In [3]: import random
random.uniform(0.5 , 1.5)

Out[3]: 0.7219779886759995
```

Yukarıdaki örnekte 0.5 ile 1.5 arasında rastgele ondalıklı sayılar üretilmiştir.

### *randint() fonksiyonu*

randint() fonksiyonu kullanılarak istenilen aralıkta rastgele tamsayı üretebiliriz.

#### randint ( ) Örnek-1

```
In [1]: import random
random.randint(45,500)

Out[1]: 220
```

```
In [2]: import random
random.randint(45,500)

Out[2]: 336
```

Yukarıdaki örnekte randint() fonksiyonu kullanılarak 45 ile 500 arasında rastgele tamsayılar üretilir.

### *choice() fonksiyonu*

choice() fonksiyonu kullanılarak dizi niteliği taşıyan veri tiplerinden (listeler, karakter dizileri) rastgele öğeler seçilir.

#### choice () Örnek-1

```
In [1]: import random
        liste=['Serkan','Yasemin','Nisanur','Suden']
        random.choice(liste)
```

```
Out[1]: 'Suden'
```

```
In [3]: import random
        liste=['Serkan','Yasemin','Nisanur','Suden']
        random.choice(liste)
```

```
Out[3]: 'Serkan'
```

Yukarıdaki örnekte dizi niteliği taşıyan veri tiplerinden listeler kullanılmıştır.

Örnekte liste adı verilen listeden rastgele elemanlar seçilmiştir.

#### choice () Örnek-2

```
In [1]: import random
        karakter_dizisi='python'
        random.choice(karakter_dizisi)
```

```
Out[1]: 'p'
```

```
In [2]: import random
        karakter_dizisi='python'
        random.choice(karakter_dizisi)
```

```
Out[2]: 'n'
```

Yukarıdaki örnekte dizi niteliği taşıyan veri tiplerinden karakter dizileri kullanılmıştır.

Örnekte karakter\_dizisi adı verilen ve değeri 'python' olan karakter dizisinden rastgele elemanlar seçilmiştir.

### *sample() fonksiyonu*

sample() fonksiyonu kullanılarak dizi niteliği taşıyan veri tiplerinden (listeler, karakter dizileri) belirli sayıda numune (örnek) alınmasını sağlar.

#### sample ( ) Örnek-1

```
In [1]: import random
        liste = list(range(20))
        liste
```

```
Out[1]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
In [2]: random.sample(liste,5)
```

```
Out[2]: [0, 2, 17, 7, 3]
```

Yukarıdaki örnekte dizi niteliği taşıyan veri tiplerinden listeler kullanılmıştır.

Örnekte ilk olarak değerleri sıralı şekilde 0'dan 19'a kadar tamsayılar olan liste isimli bir liste oluşturulmuştur.

Daha sonra liste isimli listenin elemanlarından rastgele 5 tanesi sample() fonksiyonu ile örnek olarak alınmıştır.

### *shuffle() fonksiyonu*

shuffle() fonksiyonu kullanılarak dizi niteliği taşıyan veri tiplerindeki (listeler, karakter dizileri) öğeler karıştırılır.

Burada dikkat edilmesi gereken önemli nokta, shuffle() fonksiyonunun özgün liste üzerinde değişiklik yapıyor olmasıdır. Yani liste üzerinde shuffle() fonksiyonu uygulandıktan sonra özgün liste kaybedilir. Liste üzerinde shuffle() fonksiyonu her kullanıldığında özgün listenin öğeleri bir kere daha karıştırılır.

#### shuffle ( ) Örnek-1

```
In [1]: import random
        liste = list(range(10))
        liste
```

```
Out[1]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [2]: random.shuffle(liste)
        liste
```

```
Out[2]: [2, 8, 3, 7, 6, 5, 0, 1, 4, 9]
```

Yukarıdaki örnekte dizi niteliği taşıyan veri tiplerinden listeler kullanılmıştır.

Örnekte ilk olarak değerleri sıralı şekilde 0'dan 9'a kadar tamsayılar olan liste isimli bir liste oluşturulmuştur.

Daha sonra liste isimli listenin elemanları shuffle() fonksiyonu ile karıştırılmıştır.

### *randrange() fonksiyonu*

randrange() fonksiyonu, randint() fonksiyonu ile aynı işi yapar. Yani her iki fonksiyon da belirli aralıkta rastgele tamsayılar üretir.

#### randrange ( ) Örnek-1

```
In [1]: import random  
random.randrange(10)
```

```
Out[1]: 2
```

```
In [2]: random.randint(10)
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-2-33f011be1854> in <module>  
----> 1 random.randint(10)
```

```
TypeError: randint() missing 1 required positional argument: 'b'
```

```
In [3]: random.randint(0,10)
```

```
Out[3]: 1
```

- ✚ randrange() fonksiyonu tek parametre ile kullanılabilirken randint() fonksiyonu iki parametre ile kullanılabilir.
- ✚ randint() fonksiyonu tek parametre ile kullanılırsa hata verecektir.

#### randrange ( ) Örnek-2

```
In [15]: import random  
random.randrange(10,20)
```

```
Out[15]: 19
```

```
In [33]: random.randint(10,20)
```

```
Out[33]: 20
```

- ✚ randrange() fonksiyonunda rastgele sayı üretilecek aralığın son değeri ikinci parametrede verilen değerden bir küçük sayıdır.
- ✚ randint() fonksiyonunda rastgele sayı üretilecek aralığın son değeri ikinci parametrede verilen değer kendisidir.
- ✚ random.randrange(10 , 20) komutu ile üretilecek en büyük sayı 19 olacaktır.
- ✚ random.randint(10 , 20) komutu ile üretilecek en büyük sayı 20 olacaktır.

## Özel Amaçlı Fonksiyonlar

### pass deyimi

pass kelimesi İngilizcede 'geçmek, pas geçmek' gibi anlamlara gelir.

Python'daki kullanımı da bu anlama oldukça yakındır.

Biz bu deyimi Python'da 'görmezden gel, hiçbir şey yapma' anlamında kullanırız.

Pass deyimi hiçbir şey yapmaz. Bir deyimin söz dizimsel (syntax) olarak gerekmesi durumunda kullanılabilir. Ancak program herhangi bir işlem gerektirmez.

Örneğin:

```
In [*]: while True:
        pass # Busy-wait for keyboard interrupt (Ctrl+C)
```

pass deyimi genellikle minimal sınıflar oluşturmak için kullanılır:

```
In [*]: class MyEmptyClass:
        pass
```

pass deyiminin kullanılabileceği başka bir yer, yeni kod üzerinde çalışırken, daha soyut bir seviyede düşünmeye devam etmenize izin veren bir fonksiyon veya koşullu gövde için yer tutucudur.

pass deyimi sessizce göz ardı edilir:

```
In [*]: def initlog(*args):
        pass # Remember to implement this!
```



## help() fonksiyonu

help() fonksiyonu kodlama esnasında bir python nesnesi hakkında bilgi almak için kullanılabilir.

```
In [ ]: help()
```

help yazılıp enter tuşuna basılması yeterli olacaktır.

Gelen çıktı ekranında help> yazan kısma yardım alınacak araştırılması istenilen kavram yazılabilir.

### help() Örnek-1

print komutu ile ilgili yardım alalım.

```
In [*]: help()
```

```
Welcome to Python 3.7's help utility!
```

```
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at https://docs.python.org/3.7/tutorial/.
```

```
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".
```

```
To get a list of available modules, keywords, symbols, or topics, type  
"modules", "keywords", "symbols", or "topics". Each module also comes  
with a one-line summary of what it does; to list the modules whose name  
or summary contain a given string such as "spam", type "modules spam".
```

```
help> print
```

Ayrıca doğrudan doğruya yardım alınacak komut help fonksiyonuna parametre olarak da verilebilir.

```
In [ ]: help(print)
```

```
In [*]: help()
```

```
Welcome to Python 3.7's help utility!
```

```
If this is your first time using Python, you should definitely check out the tutorial on the Internet at https://docs.python.org/3.7/tutorial/.
```

```
Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".
```

```
To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam".
```

```
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.
```

```
help> 
```

help( ) fonksiyonu kullanılarak print komutu ile ilgili yardım alınmıştır.

## eval() fonksiyonu

İngilizcede evaluate kelimesi, 'değerlendirmeye tabi tutmak, işleme sokmak, işlemek' gibi anlamlar taşır. Bu fonksiyonun görevi, kendisine verilen karakter dizilerini değerlendirmeye tabi tutmak ya da işlemektir.

### eval() Örnek-1

```
In [1]: print("""
Basit bir hesap makinesi uygulaması.

İşleçler:

+ toplama
- çıkarma
* çarpma
/ bölme

Yapmak istediğiniz işlemi yazıp ENTER
tuşuna basın. (Örneğin 23 ve 46 sayılarını
çarpmak için 23 * 46 yazdıktan sonra
ENTER tuşuna basın.)
""")

veri = input("İşleminiz: ")
hesap = eval(veri)

print(hesap)
```

Basit bir hesap makinesi uygulaması.

İşleçler:

```
+ toplama
- çıkarma
* çarpma
/ bölme
```

Yapmak istediğiniz işlemi yazıp ENTER tuşuna basın. (Örneğin 23 ve 46 sayılarını çarpmak için 23 \* 46 yazdıktan sonra ENTER tuşuna basın.)

```
İşleminiz: 23 * 46
1058
```

Bu programı çalıştırarak, "İşleminiz:" ifadesinden sonra, 23 \* 46 yazarsak programımız bize 1058 çıktısı verecektir.

Yani programımız hesap makinesi işlevini yerine getirip 23 sayısı ile 46 sayısını çarpacaktır.

Dolayısıyla, yukarıdaki programı kullanarak her türlü aritmetik işlemi yapabilirsiniz.

Hatta bu program, son derece karmaşık aritmetik işlemlerin yapılmasına dahi müsaade eder.

Eğer programımızı yukarıdaki gibi, eval() fonksiyonu olmadan yazarsak, kullanıcıımız 23 \* 46 gibi bir komut girdiğinde alacağı cevap dümdüz bir 23 \* 46 çıktısı olacaktır.

## eval() Örnek-2

```
In [1]: # eval fonksiyonu parametre olarak verilen değeri değerlendirir.  
eval("23*46")
```

```
Out[1]: 1058
```

```
In [2]: veri=input("işleminizi giriniz:")  
eval(veri)
```

```
işleminizi giriniz:15*3
```

```
Out[2]: 45
```

```
In [3]: veri=input("işleminizi giriniz:")  
eval(veri)
```

```
işleminizi giriniz:16**2
```

```
Out[3]: 256
```

```
In [4]: a=input("a değişkeninin değerini giriniz:")  
b=input("b değişkeninin değerini giriniz:")  
operator=input("Operatörü giriniz:")  
eval(a+operator+b)
```

```
a değişkeninin değerini giriniz:6  
b değişkeninin değerini giriniz:4  
Operatörü giriniz:+
```

```
Out[4]: 10
```

```
In [5]: a=input("a değişkeninin değerini giriniz:")  
b=input("b değişkeninin değerini giriniz:")  
operator=input("Operatörü giriniz:")  
eval(a+operator+b)
```

```
a değişkeninin değerini giriniz:80  
b değişkeninin değerini giriniz:7  
Operatörü giriniz:/
```

```
Out[5]: 11.428571428571429
```

eval() fonksiyonu her ne kadar son derece yetenekli ve güçlü bir araç da olsa yanlış ellerde yıkıcı sonuçlar doğurabilir.

Program yazarken, eğer eval() kullanmanızı gerektiren bir durumla karşı karşıya olduğunuzu düşünüyorsanız, bir kez daha düşünün. Ayrıca eval() fonksiyonu kullanılacağı zaman, kullanıcıdan gelen veri bu fonksiyona parametre olarak verilmeden önce sıkı bir kontrolden geçirilmelidir.

Yani kullanıcının girdiği veri eval() aracılığıyla doğrudan değerlendirmeye tabi tutulmamalıdır. Araya bir kontrol mekanizması yerleştirilmelidir.

Örneğin, yukarıdaki hesap makinesi programında kullanıcının gireceği verileri sadece sayılar ve işlemlerle sınırlandırabilirsiniz.

Yani kullanıcının, izin verilen değerler harici bir değer girmesini engelleyebilirsiniz.

## exec() fonksiyonu

Python'da eval() fonksiyonuna çok benzeyen exec() adlı başka bir fonksiyon daha bulunur. eval() ile yapamadığımız bazı şeyleri exec() ile yapabiliriz. Bu fonksiyon yardımıyla, karakter dizileri içindeki çok kapsamlı Python kodlarını işletebilirsiniz.

Örneğin eval() fonksiyonu bir karakter dizisi içindeki değişken tanımlama işlemini yerine getiremez. Yani eval() ile şöyle bir şey yapamazsınız:

```
eval("a = 45")
```

Ama exec() ile böyle bir işlem yapabilirsiniz:

```
exec("a = 45")
```

Böylece a adlı bir değişken tanımlamış olduk. Kontrol edelim:

```
print(a)
```

```
45
```

eval() ve exec() fonksiyonları özellikle kullanıcıdan alınan verilerle doğrudan işlem yapmak gereken durumlarda işinize yarar. Örneğin bir hesap makinesi yaparken eval() fonksiyonundan yararlanabiliriz.

Aynı şekilde mesela insanlara Python programlama dilini öğreten bir program yazıyorsanız exec() fonksiyonunu şöyle kullanabilirsiniz:

```
d1="""
```

```
Python'da ekrana çıktı verebilmek için print() fonksiyonu kullanılır. Bu fonksiyonu  
şöyle kullanabilirsiniz:
```

```
>>> print("Merhaba Dünya")
```

```
Şimdi de aynı kodu siz yazın!
```

```
>>> """
```

```
girdi=input(d1)
```

```
exec(girdi)
```

```
d2="""
```

```
Gördüğünüz gibi print() fonksiyonu, kendisine parametre olarak verilen değerleri ekrana  
basıyor. Böylece ilk dersimizi tamamlamış olduk. Şimdi bir sonraki dersimize  
geçebiliriz."""
```

```
print(d2)
```

eval() fonksiyonunu anlatırken güvenlik ile ilgili olarak söylediğimiz her şey exec() fonksiyonu için de geçerlidir. Dolayısıyla bu iki fonksiyonu çok dikkatli bir şekilde kullanmanız ve bu fonksiyonların doğurduğu güvenlik açığının bilincinde olmamız gerekir.

## iskeyword() fonksiyonu

iskeyword() fonksiyonu parametre olarak verilen değer anahtar kelime ise True, değilse False değerini döndürür.

Python'daki anahtar kelimeler özel bir anlamı olan özel kelimelerdir. Tanımlayıcılar bir değişkeni, işlevi, bir sınıfı veya modülü temsil eden kullanıcı tanımlı isimlerdir. Anahtar kelimeler tanımlayıcı olarak kullanılamaz. Mesela, anahtar kelimeler, değişken adı olarak kullanılamaz. Python programlama diline özel anahtar kelimeler (keyword) aşağıda listelenmiştir. Python programlama diline ait anahtar kelime sayısı 35 tane dir.

### iskeyword() Örnek-1

```
In [1]: import keyword  
keyword.kwlist
```

```
Out[1]: ['False',  
        'None',  
        'True',  
        'and',  
        'as',  
        'assert',  
        'async',  
        'await',  
        'break',  
        'class',  
        'continue',  
        'def',  
        'del',  
        'elif',  
        'else',  
        'except',  
        'finally',  
        'for',  
        'from',  
        'global',  
        'if',  
        'import',  
        'in',  
        'is',  
        'lambda',  
        'nonlocal',  
        'not',  
        'or',  
        'pass',  
        'raise',  
        'return',  
        'try',  
        'while',  
        'with',  
        'yield']
```

```
In [2]: len(keyword.kwlist)
```

```
Out[2]: 35
```

```
In [3]: keyword.iskeyword("pass")
```

```
Out[3]: True
```

```
In [4]: keyword.iskeyword("serkan")
```

```
Out[4]: False
```

## dir() fonksiyonu

dir() fonksiyonu, sistemin o anda tanıdığı tüm değerleri gösterir. Bu metot bize Python'daki bir nesnenin özellikleri hakkında bilgi edinme imkânı verir. Mesela karakter dizilerinin bize hangi metotları sunduğunu görmek için bu fonksiyonu şöyle kullanabiliriz:

dir(str)

### dir() Örnek-1

```
In [1]: print(dir(str),sep="")
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

```
In [2]: print(dir(int),sep="")
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattr__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

```
In [3]: print(dir(float),sep="")
```

```
['__abs__', '__add__', '__bool__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floordiv__', '__format__', '__ge__', '__getattr__', '__getformat__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__int__', '__le__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__pos__', '__pow__', '__radd__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rmod__', '__rmul__', '__round__', '__rpow__', '__rsub__', '__rtruediv__', '__set_format__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', 'as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'imag', 'is_integer', 'real']
```

```
In [4]: print(dir(complex),sep="")
```

```
['__abs__', '__add__', '__bool__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floordiv__', '__format__', '__ge__', '__getattr__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__int__', '__le__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__pos__', '__pow__', '__radd__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rmod__', '__rmul__', '__rpow__', '__rsub__', '__rtruediv__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', 'conjugate', 'imag', 'real']
```

## Karar Yapıları

### if else

**if** ifadesi Python programlama dilinde karşılaştırma yapmak için kullanılır.

if deyimi bir şarta bağlı olarak işlem yapmak için kullanılır. Eğer koşul sağlanıyorsa işlem/işlemler gerçekleştirilecektir. Koşul sağlanmıyorsa işlem/işlemler gerçekleştirilemeyecektir.

**else** deyimi "değilse" anlamına gelmektedir. Eğer koşul yanlış ise gerçekleştirilmesi istenilen işlem/işlemler bu kısımda belirtilir.

### if else Örnek-1

```
In [1]: parola=12345
        if parola==12345:
            print("Parolayı Doğru Girdiniz!")
```

```
Parolayı Doğru Girdiniz!
```

```
In [2]: parola=12345
        sayi=int(input("Lütfen parolayı giriniz:"))
        if sayi==parola:
            print("Parolayı Doğru Girdiniz!")
```

```
Lütfen parolayı giriniz:12345
Parolayı Doğru Girdiniz!
```

```
In [3]: parola=12345
        sayi=int(input("Lütfen parolayı giriniz:"))
        if sayi==parola:
            print("Parolayı Doğru Girdiniz!")
        else:
            print("Parolayı Yanlış Girdiniz!")
```

```
Lütfen parolayı giriniz:111
Parolayı Yanlış Girdiniz!
```

```
In [4]: parola=12345
        sayi=int(input("Lütfen parolayı giriniz:"))
        if sayi==parola:
            print("Parolayı Doğru Girdiniz!")
        else:
            print("Parolayı Yanlış Girdiniz!")
```

```
Lütfen parolayı giriniz:12345
Parolayı Doğru Girdiniz!
```



### if else Örnek-2

```
In [1]: a, b = 10, 20

if a != b:
    if a > b:
        print("a, b'den büyüktür!")
    else:
        print("b, a'dan büyüktür!")
else:
    print("a ve b eşittir!")

b, a'dan büyüktür!
```

a ve b değişkenlerine değer olarak verilen iki tamsayıyı birbiriyle kıyaslar. Ekranı uygun mesajı yazar.

Bu program çalıştırıldığında ekrana aşağıdaki mesajı yazar.

b, a'dan büyüktür!

### if else Örnek-3

```
In [1]: yas=int(input("Yaşınızı giriniz:"))
if yas < 18:
    print("Mekana Giremezsiniz!")
else:
    print("Hoşgeldiniz...")

Yaşınızı giriniz:17
Mekana Giremezsiniz!
```

```
In [2]: yas=int(input("Yaşınızı giriniz:"))
if yas < 18:
    print("Mekana Giremezsiniz!")
else:
    print("Hoşgeldiniz...")

Yaşınızı giriniz:18
Hoşgeldiniz...
```

#### if else Örnek-4

```
In [1]: a=input("enter sequence:")
b=a[::-1]
if a==b:
    print("palindrome")
else:
    print("not palindrome")
```

```
enter sequence:232
palindrome
```

```
In [2]: a=input("enter sequence:")
b=a[::-1]
if a==b:
    print("palindrome")
else:
    print("not palindrome")
```

```
enter sequence:321
not palindrome
```

```
In [3]: a=input("enter sequence:")
b=a[::-1]
if a==b:
    print("palindrome")
else:
    print("not palindrome")
```

```
enter sequence:kazak
palindrome
```

```
In [4]: a=input("enter sequence:")
b=a[::-1]
if a==b:
    print("palindrome")
else:
    print("not palindrome")
```

```
enter sequence:tas sat
palindrome
```

Yukarıda verilen örnek girilen bir karakter dizisinin palindrom (tersi düzüne eşit) olup olmadığını bulur.

Girilen karakter dizisinin tersi düzüne eşit ise "palindrome" değilse "not palindrome" mesajı yazdırılır.

"232" karakter dizisinin tersi "232" olduğu ve her iki karakter dizisi de birbirine eşit olduğu için palindromdur.

"321" karakter dizisinin tersi "123" olduğu ve her iki karakter dizisi birbirinden farklı olduğu için palindrom değildir.

"kazak" karakter dizisinin tersi "kazak" olduğu ve her iki karakter dizisi de birbirine eşit olduğu için palindromdur.

"tas sat" karakter dizisinin tersi "tas sat" olduğu ve her iki karakter dizisi de birbirine eşit olduğu için palindromdur.

### if elif else

Python programlama dilinde if bloğundan sonra tekrar bir if bloğunun geldiği art arda gelen if blokları kullanılabilir.

Bu durumda Python olası bütün sonuçları değerlendirebilmek için if bloklarını okumaya devam edecektir.

Sonraki blok yine bir if bloğu olduğu için Python kodları okumaya devam edecektir.

if deyimlerini art arda sıralayacak olursak, Python doğru olan bütün sonuçları listeleyecektir.

Eğer if deyiminden sonra **elif** deyimini kullanırsak, Python doğru olan ilk sonucu listelemekle yetinecektir.

### if elif else Örnek-1

```
In [1]: sayi=int(input("Lütfen bir sayı giriniz:"))
if sayi>0:
    print("Girilen sayı pozitiftir!")
elif sayi<0:
    print("Girilen sayı negatiftir!")
else:
    print("Girilen sayı sifıra eşittir!")
```

```
Lütfen bir sayı giriniz:5
Girilen sayı pozitiftir!
```

```
In [2]: sayi=int(input("Lütfen bir sayı giriniz:"))
if sayi>0:
    print("Girilen sayı pozitiftir!")
elif sayi<0:
    print("Girilen sayı negatiftir!")
else:
    print("Girilen sayı sifıra eşittir!")
```

```
Lütfen bir sayı giriniz:-5
Girilen sayı negatiftir!
```

```
In [3]: sayi=int(input("Lütfen bir sayı giriniz:"))
if sayi>0:
    print("Girilen sayı pozitiftir!")
elif sayi<0:
    print("Girilen sayı negatiftir!")
else:
    print("Girilen sayı sifıra eşittir!")
```

```
Lütfen bir sayı giriniz:0
Girilen sayı sifıra eşittir!
```

## if elif else Örnek-2

```
In [1]: print("İşlem türleri: 1 , 2 , 3 ")
islem=int(input("İşlemi giriniz:"))
if islem==1:
    print("İşlem 1 seçildi.")
elif islem==2:
    print("İşlem 2 seçildi.")
elif islem==3:
    print("İşlem 3 seçildi.")
else:
    print("Geçersiz işlem!")
```

İşlem türleri: 1 , 2 , 3  
İşlemi giriniz:1  
İşlem 1 seçildi.

```
In [2]: print("İşlem türleri: 1 , 2 , 3 ")
islem=int(input("İşlemi giriniz:"))
if islem==1:
    print("İşlem 1 seçildi.")
elif islem==2:
    print("İşlem 2 seçildi.")
elif islem==3:
    print("İşlem 3 seçildi.")
else:
    print("Geçersiz işlem!")
```

İşlem türleri: 1 , 2 , 3  
İşlemi giriniz:2  
İşlem 2 seçildi.

```
In [3]: print("İşlem türleri: 1 , 2 , 3 ")
islem=int(input("İşlemi giriniz:"))
if islem==1:
    print("İşlem 1 seçildi.")
elif islem==2:
    print("İşlem 2 seçildi.")
elif islem==3:
    print("İşlem 3 seçildi.")
else:
    print("Geçersiz işlem!")
```

İşlem türleri: 1 , 2 , 3  
İşlemi giriniz:3  
İşlem 3 seçildi.

```
In [4]: print("İşlem türleri: 1 , 2 , 3 ")
islem=int(input("İşlemi giriniz:"))
if islem==1:
    print("İşlem 1 seçildi.")
elif islem==2:
    print("İşlem 2 seçildi.")
elif islem==3:
    print("İşlem 3 seçildi.")
else:
    print("Geçersiz işlem!")
```

İşlem türleri: 1 , 2 , 3  
İşlemi giriniz:4  
Geçersiz işlem!

## Üçlü İfadeler (Ternary Expressions)

Python'daki kullanılan üçlü ifade, bir değer üreten if-else bloğunu tek bir satır veya ifadede birleştirmenize izin verir.

Python'da bunun sözdizimi şöyledir:

değer=doğru-ifade if koşul else yanlış-ifade

Burada doğru-ifade ve yanlış-ifade herhangi bir Python ifadesi olabilir.

Daha ayrıntılı olarak aşağıdaki yapı ile aynı etkiye sahiptir:

if condition:

```
value = true-expr
```

else:

```
value = false-expr
```

Bu daha somut bir örnek:

```
x = 5
```

```
'Non-negative' if x >= 0 else 'Negative'
```

```
'Non-negative'
```

if-else bloklarında olduğu gibi, ifadelerden yalnızca biri yürütülür. Bu nedenle, üçlü ifadenin “if” ve “else” tarafları maliyetli hesaplamalar içerebilir, ancak yalnızca doğrudan değerlendirilir.

Kodunuzu yoğunlaştırmak için her zaman üçlü ifadeler kullanmak cazip gelebilir. Ancak, koşulun yanı sıra doğru ve yanlış ifadelerin çok karmaşık olması durumunda kodun okunabilirliği azalacaktır.

### Üçlü İfade Örnek-1

```
In [1]: x=5  
'Pozitif' if x>=0 else 'Negatif'
```

```
Out[1]: 'Pozitif'
```

```
In [2]: is_nice = True  
state = "nice" if is_nice else "not nice"  
state
```

```
Out[2]: 'nice'
```

```
In [3]: a=int(input("Birinci sayıyı giriniz:"))  
b=int(input("İkinci sayı giriniz:"))  
buyuk = a if a>=b else b  
buyuk
```

```
Birinci sayıyı giriniz:5  
İkinci sayı giriniz:15
```

```
Out[3]: 15
```

### Üçlü İfade Örnek-2

```
In [1]: # Şartlı operatörü gösteren program
a, b = 10, 20

# min değişkeninin değeri: a < b ise a değilse b olacaktır
min = a if a < b else b

print(min)

10
```

Şartlı operatörlerin kullanımını gösteren yukarıdaki uygulamada aşağıdaki işlemler gerçekleştirilmiştir.

a, b = 10, 20 # a=10 , b=20

Öncelikle a ve b isimli 2 değişken tanımlamış ve bu değişkenlere sırasıyla 10 ve 20 değerleri atanmıştır.

min = a if a < b else b

Daha sonra a ve b değişkenlerinin değerlerine göre tanımlanan min değişkeninin değeri belirli olacaktır.

min değişkeninin değeri: a < b ise a değilse b olacaktır.

### Üçlü İfade Örnek-3

```
In [1]: # üçlü ifadeyi gösteren program
a, b = 10, 20

print ("a ve b eşit" if a == b else "a büyük b"
       if a > b else "b büyük a")

b büyük a
```

```
In [2]: # üçlü ifadeyi gösteren program
a, b = 10, 20

if a != b:
    if a > b:
        print("a büyük b")
    else:
        print("b büyük a")
else:
    print("a ve b eşit")

b büyük a
```

Üçlü operatör iç içe geçmiş if-else olarak yazılmıştır.

İlk önce verilen koşul (a < b) değerlendirilir. Ardından koşulun döndürdüğü mantıksal değere göre a veya b döndürülür.

Operatördeki argümanların sırası, C / C ++ gibi diğer dillerden farklıdır. Tüm Python işlemlerinde koşullu ifadeler en düşük önceliğe sahiptir.<sup>11</sup>

<sup>11</sup><https://www.geeksforgeeks.org/cc-ternary-operator-some-interesting-observations/>  
Python Anaconda ve Pycharm Kurulumu

### Üçlü İfade Örnek-3

```
In [1]: a=int(input("a değerini giriniz:"))
        b=int(input("b değerini giriniz:"))
        minimum=a if a <= b else b
        maximum=a if a >= b else b
        print("min={} max={}".format(minimum,maximum))
```

```
a değerini giriniz:5
b değerini giriniz:5
min=5 max=5
```

```
In [2]: a=int(input("a değerini giriniz:"))
        b=int(input("b değerini giriniz:"))
        c=int(input("c değerini giriniz:"))
        minimum = a if a <= b and a<=c else b if b <= a and b <= c else c
        maximum = a if a >= b and a>=c else b if b >= a and b >= c else c
        print("min={} max={}".format(minimum,maximum))
```

```
a değerini giriniz:10
b değerini giriniz:8
c değerini giriniz:6
min=6 max=10
```

```
In [3]: a=int(input("a değerini giriniz:"))
        b=int(input("b değerini giriniz:"))
        c=int(input("c değerini giriniz:"))
        minimum = a if a <= b and a<=c else b if b <= a and b <= c else c
        maximum = a if a >= b and a>=c else b if b >= a and b >= c else c
        print("min={} max={}".format(minimum,maximum))
```

```
a değerini giriniz:-3
b değerini giriniz:5
c değerini giriniz:-3
min=-3 max=5
```

```
In [4]: a=int(input("a değerini giriniz:"))
        b=int(input("b değerini giriniz:"))
        c=int(input("c değerini giriniz:"))
        minimum = a if a <= b and a<=c else b if b <= a and b <= c else c
        maximum = a if a >= b and a>=c else b if b >= a and b >= c else c
        print("min={} max={}".format(minimum,maximum))
```

```
a değerini giriniz:2
b değerini giriniz:2
c değerini giriniz:2
min=2 max=2
```

```
In [5]: a=int(input("a değerini giriniz:"))
        b=int(input("b değerini giriniz:"))
        c=int(input("c değerini giriniz:"))
        minimum = a if a <= b and a<=c else b if b <= a and b <= c else c
        maximum = a if a >= b and a>=c else b if b >= a and b >= c else c
        print("min={} max={}".format(minimum,maximum))
```

```
a değerini giriniz:-1
b değerini giriniz:-5
c değerini giriniz:-10
min=-10 max=-1
```

## Döngüler

### for Döngüsü

Döngüler programlarımızın birden fazla sayıda çalışmasını sağlar.

Herhangi bir string, tuple, sözlük veya liste üzerinde oldukça kolay bir şekilde gezilebilir.

Etrafta yazılmış Python programlarının kaynak kodlarını incelediğinizde, içinde for döngüsü içermeyen bir program kolay kolay bulamazsınız.

Fakat while döngüsünün kullanılmadığı programlar vardır.

Ancak for döngüsü Python'da o kadar yaygındır ve o kadar geniş bir kullanım alanına sahiptir ki, hemen hemen bütün Python programları bu for döngüsünden en az bir kez yararlanır.

for da tıpkı while gibi bir döngüdür. Ancak for döngüsü while döngüsüne göre biraz daha yeteneklidir.

while döngüsü ile yapamayacağınız veya yaparken çok zorlanacağınız şeyleri for döngüsü yardımıyla çok kolay bir şekilde halledebilirsiniz.

for döngüsünün söz dizimi şöyledir:

```
for değişken_adi in değişken:
    yapılacak_işlem
```

Python döngüleri opsiyonel **else** kullanımını destekler. Break ifadesine rağmen döngünün terk edilmediği durumlarda döngüye ait else ifadesi kullanılabilir.

```
for değişken_adi in değişken:
    yapılacak_işlemler1
else:
    yapılacak_işlemler2
```

### for Örnek-1

```
In [1]: sayılar = "123456789"
        for sayı in sayılar:
            print(int(sayı) * 2)
2
4
6
8
10
12
14
16
18
```

Burada sayılar adlı değişkenin her bir ögesi sayı olarak adlandırılır. Daha sonra, int() fonksiyonu yardımıyla bu öğeleri tek tek sayıya çevrilir ve her bir öge 2 ile çarpılır.



### for Örnek-2

```
In [1]: iller = ['İstanbul', 'Edirne', 'Ankara','Adana','İzmir']
        for il in iller:
            print ('Sıradaki İl :', il)

Sıradaki İl : İstanbul
Sıradaki İl : Edirne
Sıradaki İl : Ankara
Sıradaki İl : Adana
Sıradaki İl : İzmir
```

iller = ['İstanbul', 'Edirne', 'Ankara','Adana','İzmir']

Yukarıdaki ifade ile iller isimli elemanları 'İstanbul', 'Edirne', 'Ankara','Adana','İzmir' olan liste üzerinde for döngüsü kullanılmıştır.

Liste elemanlarının her biri sırası ile yazdırılmıştır.

### for Örnek-3

```
In [1]: numbers=[2,4,6,8]
        product=1
        for number in numbers:
            product = product * number
        print('The product is:',product)

The product is: 384
```

numbers=[2,4,6,8]

Yukarıdaki ifade ile numbers isimli elemanları “2,4,6,8” olan liste üzerinde for döngüsü kullanılmıştır.

Liste elemanların her biri birbiriyle çarpılmıştır.

### for Örnek-4

```
In [1]: for harf in 'Yapay Zeka':
        print(harf)

Y
a
p
a
y

Z
e
k
a
```

Yukarıdaki kodda “Yapay Zeka”içerisindeki tüm harfler alt alta gelecek şekilde ekrana yazdırılır.

### for Örnek-5

```
In [1]: for kelime in "Yapay Zeka".split():  
        print(kelime)
```

```
Yapay  
Zeka
```

Yukarıdaki kodda "Yapay Zeka" içerisindeki tüm sözcükler alt alta gelecek şekilde ekrana yazdırılır.

`split()` fonksiyonu cümle içerisinde geçen ifadeyi boşluklara göre ayırır.

```
for kelime in "Yapay Zeka".split():
```

```
    print(kelime)
```

### for Örnek-6

```
In [1]: a=[1,2,3,4,5,6]  
        for eleman in a:  
            print(eleman)
```

```
1  
2  
3  
4  
5  
6
```

### for Örnek-7

```
In [1]: liste1=[99,98,97]  
        liste2=[1, 3.14, True, 1+3j, "çiçek", "böcek", liste1]  
        for n in liste2:  
            print(n,type(n),sep="\t")
```

```
1      <class 'int'>  
3.14   <class 'float'>  
True   <class 'bool'>  
(1+3j) <class 'complex'>  
çiçek  <class 'str'>  
böcek  <class 'str'>  
[99, 98, 97] <class 'list'>
```

Farklı veri türlerini barındıran liste2 listesinin elemanlarını ve liste elemanlarının türünü ekrana yazdırır.

## range() fonksiyonu

range kelimesi İngilizcede 'aralık' anlamına gelir.

Biz Python'da range() fonksiyonunu belli bir aralıkta bulunan sayıları göstermek için kullanıyoruz.

Verdiğimiz değere göre bir liste oluşturulur. Bu oluşturulan liste üzerinde rahatça gezinebiliriz.

range() fonksiyonunun formülü şöyledir:

```
range(ilk_sayı, son_sayı)
```

Bu arada, range(ilk\_sayı, son\_sayı) kodunun verdiği çıktıya ilk\_sayının dahil olduğuna, ama son\_sayının dahil olmadığına dikkat ediniz.

range metoduna üçüncü parametre daha ekleyip saymanın yönünü ve miktarını belirtebiliriz.

```
range(ilk_sayı, son_sayı, artış/azalış miktarı)
```

range() fonksiyonu geriye range nesnesi döndürür. Fonksiyonun döndürdüğü range nesnesi, verdiğimiz argümanlara uygun aralıktaki tamsayıları içerir.

range() fonksiyonu 1, 2 veya 3 parametre alabilir.

### range() Örnek-1

```
In [1]: for i in range(0, 10):  
        print(i)  
  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

range(0, 10) kodu sayesinde ve for döngüsünü de kullanarak, 0 ile 10 (10 hariç) aralığındaki sayıları ekrana yazdırdık.

Yukarıdaki kodda range() fonksiyonuna 0 ve 10 olmak üzere iki adet parametre verdiğimizizi görüyorsunuz.

Burada 0 sayısı, aralıktaki ilk sayıyı, 10 sayısı ise aralıktaki son sayıyı gösteriyor.

**Not:** Python2 serisinde range() fonksiyonunun yanı sıra xrange() fonksiyonu da bulunuyordu. xrange fonksiyonu geriye xrange nesnesi döndürüyordu. Bu fonksiyon Python3 serisinde range şekline dönüşmüştür. Yani artık xrange() kullanılmamaktadır.

### *range()* Örnek-2

Eğer range() fonksiyonunun ilk parametresi 0 olacaksa, bu parametreyi belirtmesek de olur.

Yani mesela 0'dan 10'a kadar olan sayıları listeleyeceksek range() fonksiyonunu şöyle yazmamız yeterli olacaktır.

`for i in range(10):`

```
In [1]: for i in range(10):  
        print(i)
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

0'dan başlayarak 10'a kadar (10 dahil değil) olan sayıları ekrana yazar.

### *range()* Örnek-3

```
In [1]: for sayi in range(20,30):  
        print(sayi)
```

```
20  
21  
22  
23  
24  
25  
26  
27  
28  
29
```

20'den başlayarak 30'a kadar (30 dahil değil) olan sayıları ekrana yazar.

### *range()* Örnek-4

range metoduna bir parametre daha ekleyip saymanın yönünü ve miktarını belirtebiliriz.

```
In [1]: for sayi in range(10,0,-1):  
        print('Sıradaki Sayı:',sayi)
```

```
Sıradaki Sayı: 10  
Sıradaki Sayı: 9  
Sıradaki Sayı: 8  
Sıradaki Sayı: 7  
Sıradaki Sayı: 6  
Sıradaki Sayı: 5  
Sıradaki Sayı: 4  
Sıradaki Sayı: 3  
Sıradaki Sayı: 2  
Sıradaki Sayı: 1
```

10'dan başlayarak 0'a kadar (0 dahil değil) 1'er azaltarak yazdırır.

### *range()* Örnek-5

```
In [1]: for sayi in range(0,20,2):  
        print('Sıradaki Sayı:',sayi)
```

```
Sıradaki Sayı: 0  
Sıradaki Sayı: 2  
Sıradaki Sayı: 4  
Sıradaki Sayı: 6  
Sıradaki Sayı: 8  
Sıradaki Sayı: 10  
Sıradaki Sayı: 12  
Sıradaki Sayı: 14  
Sıradaki Sayı: 16  
Sıradaki Sayı: 18
```

0'dan başlayarak 20'ye kadar (20 dahil değil) 2'şer artırarak yazdırır.

### *range() Örnek-6*

```
In [1]: for sayi in range(0,20):  
        if sayi%2==0:  
            print('sıradaki sayı',sayi)
```

```
sıradaki sayı 0  
sıradaki sayı 2  
sıradaki sayı 4  
sıradaki sayı 6  
sıradaki sayı 8  
sıradaki sayı 10  
sıradaki sayı 12  
sıradaki sayı 14  
sıradaki sayı 16  
sıradaki sayı 18
```

### *range() Örnek-7*

```
In [1]: for sayi in range(10,100,10):  
        print(sayi)
```

```
10  
20  
30  
40  
50  
60  
70  
80  
90
```

10'dan başlayarak 100'e kadar (100 dahil değil)10'ar artırarak yazdırır.

### *range() Örnek-8*

```
In [1]: for i in range(1,51):  
        if i%5==0 or i%7==0:  
            print(i,end=" ")
```

```
5 7 10 14 15 20 21 25 28 30 35 40 42 45 49 50
```

1 ile 50 arasındaki sayılardan5 veya 7'ye tam bölünen sayıları yazdırır.

### range() Örnek-9

```
In [1]: birinciSayi =int(input("Başlangıç değerini girin:"));
        ikinciSayi=int(input("Bitiş Değerini girin :"));
        adım =int(input("Adım aralalığını girin :"));

        for i in range(birinciSayi,ikinciSayi,adım):
            print(i,end=" ")

Başlangıç değerini girin:5
Bitiş Değerini girin :100
Adım aralalığını girin :5
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
```

Belirtilen başlangıç, bitiş ve adım değerine göre, aralıktaki sayıları yazdırır.

5 'den başlayarak 100'e kadar arasında 5 artarak bulunan sayıları yazdırır.

### range() Örnek-10 Tam Kare

```
In [15]: from math import sqrt

        for sayi in range(100,0,-1):
            kok=sqrt(sayi)
            if kok == int(kok):
                print(sayi,int(kok),sep='\t')
```

100	10
81	9
64	8
49	7
36	6
25	5
16	4
9	3
4	2
1	1

```
In [16]: sqrt(99)
```

```
Out[16]: 9.9498743710662
```

```
In [17]: sqrt(100)
```

```
Out[17]: 10.0
```

```
In [18]: sqrt(64)
```

```
Out[18]: 8.0
```

100'den başlayarak 0'a kadar (0 dahil değil) birer azalarak bulunan sayılardan tam kare (bir tamsayının karesi) olan sayıları yazdırır.

### *range() Örnek-11 Faktöriyel Hesaplama*

```
In [1]: # kullanıcıdan sayı değerini alalım
sayi=int(input('Faktöriyeli alınacak sayıyı giriniz:'))

if sayi < 0:
    print("{} negatif sayı olduğundan faktöriyeli hesaplanamaz!".format(sayi))
else:
    carp=1

    for oku in range(sayi):
        carp = carp * (oku + 1)

    print("{} sayısının faktöriyeli: {}".format(sayi,carp))
```

Faktöriyeli alınacak sayıyı giriniz:4  
4 sayısının faktöriyeli: 24

```
In [2]: # kullanıcıdan sayı değerini alalım
sayi=int(input('Faktöriyeli alınacak sayıyı giriniz:'))

if sayi < 0:
    print("{} negatif sayı olduğundan faktöriyeli hesaplanamaz!".format(sayi))
else:
    carp=1

    for oku in range(sayi):
        carp = carp * (oku + 1)

    print("{} sayısının faktöriyeli: {}".format(sayi,carp))
```

Faktöriyeli alınacak sayıyı giriniz:-2  
-2 negatif sayı olduğundan faktöriyeli hesaplanamaz!

Girilen sayının faktöriyelini hesaplar ve ekrana yazdırır. Girilen sayı negatif bir sayı ise faktöriyel hesaplanmaz.



## while Döngüsü

while döngüsünde döngünün çalışmasını sürdürebilmesi için bir koşul belirlenir.

Bu koşul sağlandığı sürece döngü tekrarlanır.

Döngü koşulu doğru olduğu sürece döngü bloğu çalışır.

Döngü bloğunun sağlanmadığı durumda döngüden çıkarılır.

Genel olarak döngü mantığı böyle çalışmaktadır.

İngilizce bir kelime olan while, Türkçede '... iken, ... olduğu sürece' gibi anlamlara gelir.

Python'da while bir döngüdür. Döngüler sayesinde programlar sürekli olarak çalıştırılabilir.

while döngüsünün söz dizimi şöyledir:

```
while koşul:  
    yapılacak_işlem
```

Python döngüleri opsiyonel else kullanımını destekler. Break ifadesine rağmen döngünün terk edilmediği durumlarda döngüye ait else ifadesi kullanılabilir.

```
while koşul:  
    yapılacak_işlemler1  
else:  
    yapılacak_işlemler2
```

### while Örnek-1

```
In [1]: a = 1  
        while a == 1:  
            print("bilgisayar çıldırdı!")  
            a = a + 1  
  
bilgisayar çıldırdı!
```

Burada **a** adlı bir değişken oluşturduk.

Bu değişkenin değeri 1.

Bir sonraki satırda ise while a == 1: gibi bir ifade yazdık.

En başta da söylediğimiz gibi while kelimesi, '... iken, olduğu sürece' gibi anlamlar taşıyor.

Python programlama dilindeki anlamı da buna oldukça yakındır.

Burada while a == 1 ifadesi programımıza şöyle bir anlam katıyor:

a değişkeninin değeri 1 olduğu sürece, ekrana 'bilgisayar çıldırdı!' yazısını dök!

## while Örnek-2

```
In [1]: a = 1
        while a < 10:
            a += 1
            print(a)

2
3
4
5
6
7
8
9
10
```

Python öncelikle `a = 1` satırını görüyor ve `a`'nın değerini 1 yapıyor.

Daha sonra `a`'nın değeri 10'dan küçük olduğu müddetçe... (`while a < 10`) satırını görüyor.

Ardından `a`'nın değerini, 1 artırıyor (`a += 1`) ve `a`'nın değeri 2 oluyor.

`a`'nın değeri (yani 2) 10'dan küçük olduğu için Python ekrana ilgili çıktıyı veriyor.

İlk döngüyü bitiren Python başa dönüyor ve `a`'nın değerinin 2 olduğunu görüyor.

`a`'nın değerini yine 1 artırıyor ve `a`'yı 3 yapıyor.

`a`'nın değeri hâlâ 10'dan küçük olduğu için ekrana yine ilgili çıktıyı veriyor.

İkinci döngüyü de bitiren Python yine başa dönüyor ve `a`'nın değerinin 3 olduğunu görüyor.

Yukarıdaki adımları tekrar eden Python, `a`'nın değeri 9 olana kadar ilerlemeye devam ediyor.

`a`'nın değeri 9'a ulaştığında Python `a`'nın değerini bir kez daha artırınca bu değer 10'a ulaşıyor.

Python `a`'nın değerinin artık 10'dan küçük olmadığını görüyor ve programdan çıkıyor.

### while Örnek-3

```
In [1]: i=1
while i<1000:
    print("i:",i)
    i *= 2

i: 1
i: 2
i: 4
i: 8
i: 16
i: 32
i: 64
i: 128
i: 256
i: 512
```

### while Örnek-4

```
In [1]: a = 1
b = 10
while a <= b:
    print(a,end=" ")
    a=a+1

1 2 3 4 5 6 7 8 9 10
```

1'den 10' a kadar yan yana arada boşluk bırakarak yazar.

### while Örnek-5

```
In [1]: i=0
while i<10:
    print("i:",i)
    i+=1

i: 0
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9
```

```
In [2]: i=0
while i<10:
    print("i:",i)
    i+=3

i: 0
i: 3
i: 6
i: 9
```

### while Örnek-6

```
In [1]: sayi,toplam=0,0
while sayi<=10:
    toplam+=sayi
    sayi+=1
print("Sayıların Toplamı:",toplam)

Sayıların Toplamı: 55
```

1'den 10' a kadar sayıların toplamını yazar.

### while Örnek-7

```
In [1]: user = "admin"
password = "1234"

while (True):
    kullanıcı= input("Kullanıcı adını giriniz:")
    sifre= input("Şifreyi giriniz:")

    if (kullanıcı==user and sifre==password):
        print("Bilgileriniz doğru. Hoşgeldiniz", kullanıcı)
        break

    elif (kullanıcı!=user and sifre==password):
        print("Girilen kullanıcı adı hatalı")

    elif (kullanıcı==user and sifre!=password):
        print("hatalı şifre")
        print("şifrenizi değiştirmek ister misiniz ? (E/H)")
        cevap= input()
        if (cevap== "E"):
            cevap=input("Yeni şifre oluşturmak için eski şifreyi giriniz:")
            if cevap==password:
                yenisifre = input("Yeni şifreyi giriniz:")
                print("Lütfen bekleyiniz!")
                password = yenisifre
                print("Şifre başarıyla değiştirildi!")
        else:
            print("Hatalı giriş! Lütfen tekrar deneyiniz!")
```

```
Kullanıcı adını giriniz:aaaa
Şifreyi giriniz:1111
Kullanıcı adını giriniz:adminnnn
Şifreyi giriniz:1234
Girilen kullanıcı adı hatalı
Kullanıcı adını giriniz:admin
Şifreyi giriniz:11111
hatalı şifre
şifrenizi değiştirmek ister misiniz ? (E/H)
E
Yeni şifre oluşturmak için eski şifreyi giriniz:1234
Yeni şifreyi giriniz:12345
Lütfen bekleyiniz!
Şifre başarıyla değiştirildi!
Kullanıcı adını giriniz:admin
Şifreyi giriniz:12345
Bilgileriniz doğru. Hoşgeldiniz admin
```

## break (döngüyü bitir)

Python'da break özel bir deyimdir. Bu deyim yardımıyla, devam eden bir süreci kesintiye uğratabiliriz.

Herhangi bir döngüde çalıştırıldığı zaman döngüyü sona erdirir.

Bu deyim kullanıldığı basit bir örnek verelim.

### break Örnek-1

```
In [1]: while True:
        parola = input("Lütfen bir parola belirleyiniz:")
        if len(parola) < 5:
            print("Parola 5 karakterden az olmamalı!")
        else:
            print("Parolanız belirlendi!")
            break
```

```
Lütfen bir parola belirleyiniz:ali
Parola 5 karakterden az olmamalı!
Lütfen bir parola belirleyiniz:ahmet
Parolanız belirlendi!
```

Burada, eğer kullanıcının girdiği parolanın uzunluğu 5 karakterden az ise, Parola 5 karakterden az olmamalı! uyarısı gösterilecektir. Eğer kullanıcı 5 karakterden uzun bir parola belirlemişse, kendisine 'Parolanız belirlendi!' mesajını gösterip, break deyimi yardımıyla programdan çıkıyoruz.

### break Örnek-2

```
In [1]: i=0
        while i<10:
            if i==5:
                break
            print("i:",i)
            i+=1
```

```
i: 0
i: 1
i: 2
i: 3
i: 4
```

0'dan başlayarak 10'a kadar olan tamsayıları yazdırmak isteyelim. Ancak döngü normal bir şekilde adım adım ilerlemeye devam ederken döngü sayacı 5 olduğunda, break deyimi yardımıyla döngüden çıkılır.

## continue (döngüye devam)

Python'da continue özel bir deyimdir. Bu deyim yardımıyla mevcut döngü sonraki adımla devam ettirilir.

Herhangi bir döngüde çalıştırıldığı zaman döngü sonraki iterasyonla devam ettirilir.

Bu deyim kullanıldığı basit bir örnek verelim.

### continue Örnek-1

```
In [1]: while True:
        s = input("Bir sayı girin: ")
        if s == "iptal":
            break
        if len(s) <= 3:
            continue
        print("En fazla üç haneli bir sayı girebilirsiniz.")

Bir sayı girin: 12345
En fazla üç haneli bir sayı girebilirsiniz.
Bir sayı girin: 123
Bir sayı girin: 456
Bir sayı girin: iptal
```

Burada eğer kullanıcı klavyede iptal yazarsa programdan çıkılacaktır.

Kullanıcının girdiği sayı üç veya daha az haneli bir sayı ise döngünün en başına gidilecektir.

Kullanıcının girdiği sayıdaki hane üçten fazlaysa ekrana "En fazla üç haneli bir sayı girebilirsiniz." yazdırılacaktır.

Continue deyimi kendisinden sonra gelen her şeyi es geçerek döngünün başına dönülmesini sağlar.

Buna göre, eğer kullanıcı, uzunluğu üç karakterden az bir sayı girerse continue deyiminin etkisiyle programımız döngünün en başına geri gidiyor. Ama eğer kullanıcı, uzunluğu üç karakterden fazla bir sayı girerse, ekrana 'En fazla üç haneli bir sayı girebilirsiniz,' cümlesinin yazdırıldığını görüyoruz.

### continue Örnek-2

```
In [1]: i=0
        while i<10:
            if i==3 or i==5:
                i+=1
                continue
            print("i:",i)
            i+=1

i: 0
i: 1
i: 2
i: 4
i: 6
i: 7
i: 8
i: 9
```

Program 0'dan 10'a kadar sayıları ekrana yazdırır. Döngü değişkeni (i) 3 veya 5 olduğunda ise döngü sonraki adımla devam edecektir. Dolayısıyla program 3 ve 5 sayılarını pas geçerek geri kalan 0-10 arasındaki diğer sayıları ekrana yazdıracaktır.

## Liste (List)

Liste, farklı elemanları peş peşe saklayan birimlerdir. Listeler bir veya daha fazla veri tipini içerisinde barındırabilir.

Python'da yer alan listeler yeniden boyutlandırılabilir ve farklı tipte veri tiplerini saklayabilir.

Liste elemanları değiştirilebilir (mutable) özelliğindedir.

### Liste Oluşturma

Bir liste elde etmek için, listenin elemanlarını birbirinden virgülle ayırmak ve bunların hepsini köşeli parantezler içine almak gerekir. Yani, liste elemanları köşeli parantezler arasına, aralarına virgül konularak yazılır.

#### Liste Oluşturma Örnek-1

```
In [1]: # liste1 adında boş bir liste oluşturulmuştur.  
liste1 = [ ]  
liste1
```

```
Out[1]: [ ]
```

```
In [2]: # liste2 adında 5 elemanlı bir liste oluşturulmuştur.  
liste2 = [1,2,3,4,5]  
liste2
```

```
Out[2]: [1, 2, 3, 4, 5]
```

```
liste1 = [ ]          # liste1 adında boş bir liste oluşturulmuştur.
```

```
liste2 = [1,2,3,4,5] # liste2 adında 5 elemanlı bir liste oluşturulmuştur.
```

#### Liste Oluşturma Örnek-2

```
In [1]: squares = [1, 4, 9, 16, 25]  
squares
```

```
Out[1]: [1, 4, 9, 16, 25]
```

squares = [1, 4, 9, 16, 25] ifadesi ile squares (kareler) isimli tamsayıların karelerini içeren bir liste tanımlanmıştır.

#### Liste Oluşturma Örnek-3

```
In [1]: meyveler = ["elma","armut","çilek","kiraz"]  
meyveler
```

```
Out[1]: ['elma', 'armut', 'çilek', 'kiraz']
```

meyveler = ["elma","armut","çilek","kiraz"] ifadesi ile meyveler isimli liste oluşturulmuştur.

## Liste Elemanlarına Erişim

Tıpkı karakter dizilerinde olduğu gibi, listelerde de her elemanın bir indisi (sırası) vardır.

Karakter dizilerinde olduğu gibi, listelerde de ilk elemanın indisi 0 olarak belirtilmiştir.

liste[1] listenin ikinci elemanını belirtmektedir.

Listenin en son elemanının indisi ise -1'dir.

Listede olmayan bir indise ulaşmak istenirse **IndexError** hatası verecektir.

Liste elemanlarının hepsine tek tek ulaşmak için for döngüsü kullanılabilir.

### Liste Elemanlarına Erişim Örnek-1

```
In [1]: squares = [1, 4, 9, 16, 25] # squares listesi (sayıların kareleri)
```

```
In [2]: squares[0] # squares listesinin ilk elemanı
```

```
Out[2]: 1
```

```
In [3]: squares[-1] # squares listesinin son elemanı
```

```
Out[3]: 25
```

```
In [4]: squares[-3:] # squares listesinin son 3 elemanı
```

```
Out[4]: [9, 16, 25]
```

```
In [5]: squares[:] # squares listesinin başından sonuna tüm elemanlar
```

```
Out[5]: [1, 4, 9, 16, 25]
```

### Liste Elemanlarına Erişim Örnek-2

```
In [1]: meyveler = ["elma", "armut", "çilek", "kiraz"]
for meyve in meyveler:
    print(meyve)
```

```
elma
armut
çilek
kiraz
```

meyveler listesinin tüm elemanları ekrana yazdırılır.

### Liste Elemanlarına Erişim Örnek-3

```
In [1]: meyveler = ["elma", "armut", "çilek", "kiraz"]
meyveler[::-1] # meyveler listesini ters çevirir.
```

```
Out[1]: ['kiraz', 'çilek', 'armut', 'elma']
```

meyveler listesinin elemanları tersine çevirilir.



## Liste Elemanlarını Değiştirme

Bir liste yapısı içerisinde bulunan elemanların değerleri değiştirilebilir.

Liste elemanlarını değiştirmek için şöyle bir formül kullanılır:

Liste[değiştirilecek elemanın indis değeri] = yeni değer

Liste elemanlarını değiştirmeye çalışırken olmayan bir indise ulaşmaya çalışılırsa **IndexError** hatası verecektir.

### Liste Elemanlarını Değiştirme Örnek-1

```
In [1]: letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']  
letters
```

```
Out[1]: ['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

```
In [2]: len(letters)
```

```
Out[2]: 7
```

```
In [3]: letters[2:5] = ['C', 'D', 'E'] # bazı değerler değiştirilir  
letters
```

```
Out[3]: ['a', 'b', 'C', 'D', 'E', 'f', 'g']
```

```
In [4]: letters[2:5] = [] # değiştirilen değerler kaldırılır  
letters
```

```
Out[4]: ['a', 'b', 'f', 'g']
```

```
In [5]: # boş bir liste ile elemanların tümü yer değiştirilerek liste temizlenir  
letters[:] = []  
letters
```

```
Out[5]: []
```

```
In [6]: len(letters)
```

```
Out[6]: 0
```

letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g'] ifadesi ile letters isimli 'a', 'b', 'c', 'd', 'e', 'f', 'g' karakterlerden oluşan bir liste oluşturulmuştur.

len(letters) ifadesi ile letters listesinin eleman sayısına ulaşılır. letters listesinde 7 eleman bulunmaktadır.

```
letters[2:5] = ['C', 'D', 'E']
```

Yukarıdaki ifade ile listenin 2. indeks değerinden itibaren 5. indeks değerine kadar (5 dahil değil) olan bazı değerleri sırasıyla 'C', 'D', 'E' olarak değiştirilmiştir.

```
letters[2:5] = []
```

Yukarıdaki ifade ile listenin 2. indeks değerinden itibaren 5. indeks değerine kadar (5 dahil değil) olan bazı değerleri listeden kaldırılmıştır.

```
letters[:] = []
```

Yukarıdaki ifade ile listenin tüm elemanları kaldırılmıştır. Yani liste temizlenmiştir.

## Liste Elemanlarını Değiştirme Örnek-2

```
In [1]: renkler=["kırmızı","sarı","mavi","yeşil","beyaz"]  
renkler
```

```
Out[1]: ['kırmızı', 'sarı', 'mavi', 'yeşil', 'beyaz']
```

```
In [2]: # renkler listesinin "kırmızı" olan ilk elemanı "siyah" olarak değiştirilir  
renkler[0]="siyah"  
renkler
```

```
Out[2]: ['siyah', 'sarı', 'mavi', 'yeşil', 'beyaz']
```

```
In [3]: # renkler listesinin "mavi" olan 3. elemanı "mor" olarak değiştirilir  
renkler[2]="mor"  
renkler
```

```
Out[3]: ['siyah', 'sarı', 'mor', 'yeşil', 'beyaz']
```

```
In [4]: # renkler listesinde indisi 10 olan eleman olmadığından IndexError hatası verir  
renkler[10]="pembe"
```

```
-----  
IndexError                                Traceback (most recent call last)  
<ipython-input-4-fd5b3dad6720> in <module>  
----> 1 renkler[10]="pembe"  
  
IndexError: list assignment index out of range
```

## Liste Elemanlarını Değiştirme Örnek-3

```
In [1]: liste=[1,2,3]  
liste
```

```
Out[1]: [1, 2, 3]
```

```
In [2]: liste[:]=5,6,7 # listenin tüm elemanları 5,6,7 olarak değiştirilir  
liste
```

```
Out[2]: [5, 6, 7]
```

```
In [3]: liste[0:3]=5,6,7 # listenin tüm elemanları 5,6,7 olarak değiştirilir  
liste
```

```
Out[3]: [5, 6, 7]
```

```
In [4]: liste[0:len(liste)]=5,6,7 # listenin tüm elemanları 5,6,7 olarak değiştirilir  
liste
```

```
Out[4]: [5, 6, 7]
```

## Liste Elemanlarını Dilimleme

Bir liste yapısı içerisinde bulunan elemanlara dilimleme yöntemi ile ulaşabiliriz. Dilimleme genel yapısı şu şekildedir:

dilim = liste[ başlangıç:bitiş:adım ] Bu yapı içerisinde başlangıç, bitiş veya adım isteğe bağlı olarak yazılmayabilir.

### Liste Elemanlarını Dilimleme Örnek-1

```
In [1]: # Python'da ön-tanımlı 'range' fonksiyonu
# ile 0'dan 5'e kadar (5 hariç) tam sayı listesi oluşturulmuş
sayilar = list(range(5))

# Listenin tamamını yazdıralım = "[0, 1, 2, 3, 4]"
print (sayilar)

# Listenin 2. indisinden 4. indisine kadar olan dilimini yazdıralım = "[2, 3]"
print (sayilar[2:4])

# Listenin 2. indisinden en sona kadar olan dilimini yazdıralım = "[2, 3, 4]"
print (sayilar[2:])

# Listenin en baştan 2. indise kadar olan dilimini yazdıralım = "[0, 1]"
print (sayilar[:2])

# Listenin tamamını dilim olarak yazdıralım = "[0, 1, 2, 3, 4]"
print (sayilar[:])

# Liste indisleri negatif olabilir. -1 listenin son elemanını verdiğinden,
# en baştan listenin son elemanına kadar olan dilimi yazdıralım:
print (sayilar[:-1])

# Dilime yeni bir alt liste atayalım
sayilar[2:4] = [8, 9]

# Listenin yeni hali = "[0, 1, 8, 9, 4]"
print (sayilar)

[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

### Liste Elemanlarını Dilimleme Örnek-2

```
In [1]: metin="İstanbul Büyükşehir Belediyesi"
metin.split()
# split() fonksiyonu metinde geçen ifadeyi boşluk karakterine göre ayırır.
```

```
Out[1]: ['İstanbul', 'Büyükşehir', 'Belediyesi']
```

```
In [2]: metin="İstanbul Büyükşehir Belediyesi"
liste = metin.split()
liste[0]
```

```
Out[2]: 'İstanbul'
```

## Listeye Eleman Ekleme

Listeler büyüyüp küçülebilen bir veri tipidir.

Listelere dilediğimiz kadar eleman ekleyebiliriz. Ancak eklenecek elemanın liste olması gerekmektedir.

'+' operatörü kullanılarak bir listeye elemanlar eklenebilir.

'+' operatörü kullanılarak eklenecek elemanlar liste olmazsa **TypeError** hatası verecektir.

Bir listeye doğrudan karakter dizileri veya sayılar eklenemez.

### Listeye Eleman Ekleme Örnek-1

```
In [1]: liste = [2, 4, 5, 7]
        liste
```

```
Out[1]: [2, 4, 5, 7]
```

```
In [2]: liste + [8]
```

```
Out[2]: [2, 4, 5, 7, 8]
```

```
In [3]: liste + 8
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-3-114334947ffc> in <module>
----> 1 liste + 8
```

```
TypeError: can only concatenate list (not "int") to list
```

liste = [2, 4, 5, 7]      # liste isimli liste oluşturulmuştur.

liste + [8]              # listeye [8] elemanı eklenmiştir.

liste + 8                # listeye doğrudan sayı eklenemez.

### Listeye Eleman Ekleme Örnek-2

```
In [1]: squares = [1, 4, 9, 16, 25] # squares listesi (sayıların kareleri)
        squares
```

```
Out[1]: [1, 4, 9, 16, 25]
```

```
In [2]: # squares listesinin sonuna "36, 49, 64, 81, 100" elemanları eklenir.
        squares + [36, 49, 64, 81, 100]
```

```
Out[2]: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

squares = [1, 4, 9, 16, 25]      # squares listesi (sayıların kareleri) oluşturulmuştur.

squares + [36, 49, 64, 81, 100] # squares listesinin sonuna "36, 49, 64, 81, 100" elemanları eklenir.

## Listeleri Birleřtirme

Bazı durumlarda farklı kaynaklardan farklı listeler gelebilir. Böyle bir durumda, bu farklı listeleri tek bir liste halinde birleřtirmek gerekebilir.

Karakter dizilerinde olduđu gibi listelerde de birleřtirme iřlemi için '+' operatörü kullanılır.

Bir verinin listeye parça parça deđil de bir bütün olarak eklenmesi için [] operatörleri kullanılır.

### Listeleri Birleřtirme Örnek-1

```
In [1]: derlenen_diller=["C","C++","C#","Java"]
        yorumlanan_diller=["Python","PHP","Perl","Ruby"]

In [2]: # listeleri birleřtirmek için '+' operatörü kullanılır
        programlama_dilleri = derlenen_diller + yorumlanan_diller

In [3]: print(programlama_dilleri)

['C', 'C++', 'C#', 'Java', 'Python', 'PHP', 'Perl', 'Ruby']
```

derlenen\_diller=["C","C++","C#","Java"]

# derlenen\_diller listesi oluşturulmuřtur.

yorumlanan\_diller=["Python","PHP","Perl","Ruby"]

# yorumlanan\_diller listesi oluşturulmuřtur.

programlama\_dilleri = derlenen\_diller + yorumlanan\_diller

# derlenen\_diller ve yorumlanan\_diller adlı listelerin elemanları, programlama\_dilleri isimli tek bir listenin içerisinde birleřtirilmiřtir.

### Listeleri Birleřtirme Örnek-2

```
In [1]: # Girilen 5 sayıyı ve ortalamasını ekrana yazdırır
        toplam=0 # sayıların toplamı
        notlar = [] # girilen notlar

        for i in range(5):
            veri = int(input("{} not: ".format(i+1)))
            toplam += veri
            notlar += [veri] # Kullanıcıdan gelen veri liste haline getirilir

        print("Girilen Notlar: ", *notlar)
        print("Not Ortalaması:", toplam/5)

1. not: 40
2. not: 60
3. not: 80
4. not: 70
5. not: 60
Girilen Notlar: 40 60 80 70 60
Not Ortalaması: 62.0
```

Bu program, kullanıcının 5 adet sayı girmesine izin verip, program çıkışında girilen sayıların ortalamasını verecektir.

notlar = []

# Girilen notların tutulduđu notlar isimli boş bir liste oluşturulur.

[veri]

# Kullanıcıdan gelen veri, liste haline getirilir.

notlar += [veri]

# Kullanıcıdan gelen veri listesi notlar listesiyle birleřtirilir.

## Listeleri Kopyalama

Listenin bir kopyasını almak için dilimleme yöntemi veya list() fonksiyonu kullanılabilir.

Liste isimli bir listenin kopyasını almak için kullanılacak yöntemler şunlardır:

1. kopya = liste[ : ]
2. kopya = list(liste)

### Listeleri Kopyalama Örnek-1

```
In [1]: liste = ["Ayşe", "Fatma", "Zeki"] # liste listesi oluşturulur
liste
Out[1]: ['Ayşe', 'Fatma', 'Zeki']

In [2]: kopya = liste[:] # liste listesinin kopyası oluşturulur
kopya
Out[2]: ['Ayşe', 'Fatma', 'Zeki']

In [3]: id(kopya) # kopya listesinin kimlik numarası
Out[3]: 2108321374536

In [4]: id(liste) # liste listesinin kimlik numarası
Out[4]: 2108321248968

In [5]: id(kopya) == id(liste) # iki listenin kimlik numarası eşit değildir
Out[5]: False

In [6]: liste[0] = "Veli" # liste listesinde değişiklik yapılır
liste
Out[6]: ['Veli', 'Fatma', 'Zeki']

In [7]: kopya # liste listesinde yapılan değişiklik kopya listesine yansımamıştır
Out[7]: ['Ayşe', 'Fatma', 'Zeki']
```

```
liste = ["Ayşe", "Fatma", "Zeki"] # liste listesi oluşturulur.
kopya = liste[:] # liste listesinin kopyası oluşturulur.
id(kopya) # kopya listesinin kimlik numarası # 2108321374536
id(liste) # liste listesinin kimlik numarası # 2108321248968
id(kopya) == id(liste) # iki listenin kimlik numarası eşit değildir. # False
liste[0] = "Veli" # liste listesinde değişiklik yapılır.
liste # ['Veli', 'Fatma', 'Zeki']
kopya # ["Ayşe", "Fatma", "Zeki"]
```

# liste listesinde yapılan değişiklik kopya listesine yansımamıştır.

## Listeleri Kopyalama Örnek-2

```
In [6]: iller = ["Adana", "Mersin", "Gaziantep"] # iller listesi oluşturulur  
iller
```

```
Out[6]: ['Adana', 'Mersin', 'Gaziantep']
```

```
In [7]: kopya = list(iller) # iller listesinin kopyası oluşturulur  
kopya
```

```
Out[7]: ['Adana', 'Mersin', 'Gaziantep']
```

```
In [8]: id(kopya) # kopya listesinin kimlik numarası
```

```
Out[8]: 2477344410312
```

```
In [9]: id(iller) # iller listesinin kimlik numarası
```

```
Out[9]: 2477340124616
```

```
In [10]: id(kopya) == id(iller) # iki listenin kimlik numarası eşit değildir
```

```
Out[10]: False
```

```
In [11]: iller[2] = "Kahramanmaraş" # iller listesinde değişiklik yapılır  
liste
```

```
Out[11]: ['Ayşe', 'Fatma', 'Zeki']
```

```
In [12]: kopya # iller listesinde yapılan değişiklik kopya listesine yansımaz
```

```
Out[12]: ['Adana', 'Mersin', 'Gaziantep']
```

iller = ["Adana", "Mersin", "Gaziantep"] # iller listesi oluşturulur.

**kopya = list(iller)** # iller listesinin kopyası oluşturulur.

id(kopya) # kopya listesinin kimlik numarası # 2477344410312

id(iller) # iller listesinin kimlik numarası # 2477340124616

id(kopya) == id(iller) # iki listenin kimlik numarası eşit değildir. # False

iller[2] = "Kahramanmaraş" # iller listesinde değişiklik yapılır.

iller # ['Ayşe', 'Fatma', 'Zeki']

kopya # ['Adana', 'Mersin', 'Gaziantep']

# iller listesinde yapılan değişiklik kopya listesine yansımamıştır.

## Listeleri Eşitleme

İki liste karşılıklı olarak birbirine eşitlenebilir. Bu durumda listelerin herhangi birinde yapılan bir değişiklik doğrudan doğruya diğer listeye de yansiyacaktır.

kopya ve listem isimli 2 listeyi karşılıklı olarak birbirine eşitlemek için kullanılacak ifade şu şekilde olmalıdır:

kopya = listem

### Listeleri Eşitleme Örnek-1

```
In [1]: liste = ["elma", "armut", "erik"]  
liste
```

```
Out[1]: ['elma', 'armut', 'erik']
```

```
In [2]: kopya = liste
```

```
In [3]: id(kopya) # kopya listesinin kimlik numarası
```

```
Out[3]: 1849184229320
```

```
In [4]: id(liste) # liste listesinin kimlik numarası
```

```
Out[4]: 1849184229320
```

```
In [5]: kopya
```

```
Out[5]: ['elma', 'armut', 'erik']
```

```
In [6]: liste
```

```
Out[6]: ['elma', 'armut', 'erik']
```

```
In [7]: kopya[0] = "karpuz"
```

```
In [8]: kopya
```

```
Out[8]: ['karpuz', 'armut', 'erik']
```

```
In [9]: liste
```

```
Out[9]: ['karpuz', 'armut', 'erik']
```

```
In [10]: liste[1] = "çilek"
```

```
In [11]: liste
```

```
Out[11]: ['karpuz', 'çilek', 'erik']
```

```
In [12]: kopya
```

```
Out[12]: ['karpuz', 'çilek', 'erik']
```

Eşitleme işleminden sonra kullanılan `id(kopya)` ifadesi ile kopya listesinin kimlik numarası (1849184229320) ve `id(liste)` ifadesi ile liste listesinin kimlik numarası (1849184229320) olmuştur. Bu iki değer birbirine eşit olduğu için her iki liste de bellekte aynı yeri işaret etmektedir.



## İç içe Listeler (Nested Lists)

Bir liste yapısı içerisinde başka bir liste/listeler de kullanılabilir.

İç içe geçmiş listelerin elemanlarına ulaşmak için önce gömülü listenin ana listedeki konumunu, ardından da ulaşılmak istenilen elemanın gömülü listedeki konumu belirtilmelidir.

### İç içe Listeler Örnek-1

```
In [1]: a = ['a', 'b', 'c']
        n = [1, 2, 3]
        x = [a, n]
        x

Out[1]: [['a', 'b', 'c'], [1, 2, 3]]

In [2]: x[0]

Out[2]: ['a', 'b', 'c']

In [3]: x[0][1]

Out[3]: 'b'
```

a = ['a', 'b', 'c'] ifadesi ile a isimli elemanları karakterlerden oluşan liste oluşturulur.

n = [1, 2, 3] ifadesi ile n isimli tamsayı elemanlardan oluşan liste oluşturulur.

x=[a, n] ifadesi ile ilk elemanı a ve ikinci elemanı n olacak şekilde, içerisinde a ve n listelerini içeren x isimli iç içe liste oluşturulur.

### İç içe Listeler Örnek-2

```
In [1]: liste=["Ali","Veli",["Ayşe","Nazan","Zeynep"],34,65,33,5.6]
        type(liste)

Out[1]: list

In [2]: for oge in liste:
        print("{} adlı elemanın veri tipi: {}".format(oge,type(oge)))

Ali adlı elemanın veri tipi: <class 'str'>
Veli adlı elemanın veri tipi: <class 'str'>
['Ayşe', 'Nazan', 'Zeynep'] adlı elemanın veri tipi: <class 'list'>
34 adlı elemanın veri tipi: <class 'int'>
65 adlı elemanın veri tipi: <class 'int'>
33 adlı elemanın veri tipi: <class 'int'>
5.6 adlı elemanın veri tipi: <class 'float'>
```

liste=["Ali","Veli",["Ayşe","Nazan","Zeynep"],34,65,33,5.6] # liste adında bir liste oluşturulmuştur.

type(liste) # listenin tipi sorgulanmıştır.

for oge in liste: # listenin her bir elemanına ulaşabilmek için döngü oluşturulmuştur.

print("{} adlı elemanın veri tipi: {}".format(oge,type(oge))) # listenin her bir elemanı ve tipini ekrana yazdırılır.

## Liste Üreteçleri (List Comprehension)

Bir liste oluşturmak için, köşeli parantezler arasında, virgüllerle ayrılmış ifadeler sıralarız.

Aynı zamanda liste üreteçleri ile liste oluşturabilirsiniz ki bunlara Python jargonunda "List Comprehension" denir.<sup>12</sup>

Yazımı da şöyledir:

```
listem = [ifade for degisken in sequence if conditional]
```

İfade olarak geçerli herhangi bir Python ifadesi yazılabilir. Bu ifade sequence ile belirtilen objenin her elemanı için hesaplanır ve sonuç listeye dahil edilir.

Liste üreteçleri (list comprehension), listeleri oluşturmak için kısa bir yol sağlar. Liste üreteçleri, her zaman bir sonuç listesi döndürür. Sonuç, ifadeyi for bağlamında ve onu izleyen cümlecikler bağlamında değerlendirmekten kaynaklanan yeni bir liste olacaktır.<sup>13</sup>

### Liste Üreteçleri Örnek-1

```
In [1]: listem = [x**2 for x in (1,2,3,4,5)]
listem
Out[1]: [1, 4, 9, 16, 25]
```

### Liste Üreteçleri Örnek-2

```
In [1]: list1 = [3,4,5]
multiplied = [item*3 for item in list1]
print(multiplied)
[9, 12, 15]
```

### Liste Üreteçleri Örnek-3

```
In [1]: [x.lower() for x in ["A","B","C"]]
Out[1]: ['a', 'b', 'c']
In [2]: [x.upper() for x in ["a","b","c"]]
Out[2]: ['A', 'B', 'C']
```

### Liste Üreteçleri Örnek-4

```
In [1]: string = "Hello 12345 World"
numbers = [x for x in string if x.isdigit()]
print(numbers)
['1', '2', '3', '4', '5']
```

<sup>12</sup><https://ysar.net/python/list-dersleri-tutorial.html>

<sup>13</sup><https://www.pythonforbeginners.com/basics/list-comprehensions-in-python>

## Liste Fonksiyonları

### *list()*

list fonksiyonu üstünde döngü kurulabilen nesnelere üzerinde çalışabilir. list() fonksiyonunun kullanım amaçları şunlardır:

1. Liste tipinde bir veri oluşturmak # listem = list() # listem adında boş liste oluşturulur.
2. Farklı veri tiplerini liste tipine dönüştürmek # list('python') # ['p', 'y', 't', 'h', 'o', 'n']
3. range() ifadesini listeye dönüştürür. #list(range(10,20)) # [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

### list() Örnek-1

```
In [1]: list('python')
Out[1]: ['p', 'y', 't', 'h', 'o', 'n']
```

list() fonksiyonu ile 'python' karakter kümesi listeye dönüştürülmüştür.

### list() Örnek-2

```
In [1]: alfabe="abcçdefgğhıijklmnoöprştuüvyz"
        len(alfabe)
Out[1]: 29

In [2]: harfler=list(alfabe)
        print(harfler)
['a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'ğ', 'h', 'ı', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'ö', 'p', 'r', 's', 'ş', 't', 'u', 'ü', 'v', 'y', 'z']
```

alfabe isimli karakter dizisini öğelerine ayırmak için list fonksiyonu kullanılmıştır.

### list() Örnek-3

```
In [1]: list(range(10,20))
Out[1]: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

list() fonksiyonu, range() ifadesini listeye dönüştürür.

### list() Örnek-4

```
In [1]: # bir sözlüğü, list fonksiyonu yardımıyla listeye dönüştürebiliriz.
        s = {'elma': 44, 'armut': 10, 'erik': 100}
        list(s)
Out[1]: ['elma', 'armut', 'erik']

In [2]: # sözlüğün anahtarlarından değil de değerlerinden bir liste oluşturabiliriz.
        list(s.values())
Out[2]: [44, 10, 100]
```

### append()

append() fonksiyonu ile listenin sonuna yeni eleman eklenir. append kelimesi İngilizcede 'eklemek, ilave etmek, iliştiirmek' gibi anlamlara gelir. append() metodunun görevi de kelime anlamıyla uyumludur. Bu metodu, bir listeye öge eklemek için kullanıyoruz.

#### append() Örnek-1

```
In [1]: cubes = [1, 8, 27, 65, 125] # something's wrong here
```

```
In [2]: 4 ** 3 # the cube of 4 is 64, not 65!
```

```
Out[2]: 64
```

```
In [3]: cubes[3] = 64 # replace the wrong value
```

```
In [4]: cubes
```

```
Out[4]: [1, 8, 27, 64, 125]
```

```
In [5]: cubes.append(216) # add the cube of 6
```

```
In [6]: cubes.append(7 ** 3) # and the cube of 7
```

```
In [7]: cubes
```

```
Out[7]: [1, 8, 27, 64, 125, 216, 343]
```

cubes = [1, 8, 27, 65, 125] # cubes (küpler) isimli tamsayıların küplerini içeren bir liste tanımlanmıştır.

cubes[3] = 64 ifadesi ile listede hatalı olan değer değiştirilir.

cubes.append(216) ifadesi ile listenin sonuna 216 ( $6^3$ ) değeri eklenir.

cubes.append(7 \*\* 3) ifadesi ile listenin sonuna 343 ( $7^3$ ) değeri eklenir.

#### append() Örnek-2

```
In [1]: liste=[1,2,3,4,5,6]
        liste.append("Python")
        print(liste)
        [1, 2, 3, 4, 5, 6, 'Python']
```

Listenin sonuna "Python" elamanı eklenir.

## del

Bir listeden eleman silmek için del ifadesinden yararlanılır.

del ifadesi ile liste tamamen silinebilir veya listede indeks değerine göre belirlenen eleman listeden çıkarılabilir.

### del Örnek-1

```
In [1]: liste = [10, 5, 3, 15, 20]
del liste[2]
print(liste)

[10, 5, 15, 20]
```

del liste[2] komutu ile listenin 2 numaralı indeks değerine sahip elemanı olan "3" silinir.

### del Örnek-2

```
In [1]: liste = [1, 2, 3, 4, 5]
liste
```

```
Out[1]: [1, 2, 3, 4, 5]
```

```
In [2]: del liste[-1]
liste
```

```
Out[2]: [1, 2, 3, 4]
```

del liste[-1] komutu ile listenin son elemanı olan "5" silinir.

### del Örnek-3

```
In [1]: liste = [1, 5, 3, 2, 9]
liste
```

```
Out[1]: [1, 5, 3, 2, 9]
```

```
In [2]: del liste # Liste tamamen silinir
```

```
In [3]: liste # Liste silindiği için hata verecektir.
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-3-6eb0c56ccd55> in <module>
----> 1 liste

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

del liste ifadesi ile liste tamamen silinir.

Liste silindiği için hata verecektir.

### *remove()*

remove() fonksiyonu ile parametre olarak gönderilen ve listede bulunan eleman listeden çıkarılır, yani silinir.

#### remove() Örnek-1

```
In [1]: uyeler=["Ali","Ahmet","Mehmet"]  
uyeler
```

```
Out[1]: ['Ali', 'Ahmet', 'Mehmet']
```

```
In [2]: uyeler.remove("Ahmet")  
uyeler
```

```
Out[2]: ['Ali', 'Mehmet']
```

uyeler listesinden "Ahmet" isimli elemanı çıkarır.

### *sort()*

sort() fonksiyonu ile listede bulunan elemanlar küçükten büyüğe veya alfabetik olarak a'dan z'ye doğru sıralanır.

#### sort() Örnek-1

```
In [1]: liste2 = [10, 5, 3, 15, 20, 45, 30, 20, 17]  
liste2.sort()  
print(liste2)
```

```
[3, 5, 10, 15, 17, 20, 20, 30, 45]
```

### *reverse()*

reverse() fonksiyonu ile listede bulunan elemanların sıralaması ters çevrilir.

#### reverse() Örnek-1

```
In [1]: liste = [1,2,3,4,5]  
liste.reverse()  
print(liste)
```

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

### *count()*

count() fonksiyonu ile aradığımız bir elemanın listede kaç adet olduğunu bulur.

#### count() Örnek-1

```
In [7]: liste = [1, 1, 1, 2, 2, 3]
        liste.count(1)
```

```
Out[7]: 3
```

```
In [6]: liste.count(2)
```

```
Out[6]: 2
```

```
In [8]: liste.count(3)
```

```
Out[8]: 1
```

### *index()*

index() fonksiyonu liste içerisinde arama yapmayı sağlar.

count() fonksiyonu ile benzer olsa da; bu metot ile öğenin listede bulunduğu indis değeri bulunur.

#### index() Örnek-1

```
In [1]: liste = ["Ahmet", "Mehmet", "Selim", "Canan", "Ayşe"]
        liste.index("Canan")
```

```
Out[1]: 3
```

```
In [2]: liste.index("Ahmet")
```

```
Out[2]: 0
```

Listede ilk eleman olarak yer alan "Ahmet" elemanının indeks değeri 0'dır. Listede 4. eleman olarak yer alan "Canan" elemanının indeks değeri ise 3'tür.

### *pop()*

pop() fonksiyonu ile listenin son elemanını listeden çıkarılır. pop() fonksiyonuna parametre verilirse, parametre olarak verilen indeks değerine sahip olan eleman listeden çıkarılır.

#### pop() Örnek-1

```
In [1]: liste=[1,2,3,4,5,6]
        liste.pop()
        print(liste)
```

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

```
In [2]: liste.pop(0)
        print(liste)
```

```
[2, 3, 4, 5]
```

### enumerate

Liste elemanlarının indeks değerlerine de ihtiyaç duyulduğunda enumerate() fonksiyonu ile bu ihtiyaç karşılanabilir.

Döngü içerisinde her bir liste elemanının indeks değerine ulaşmak için enumerate fonksiyonunu kullanabilirsiniz.

#### enumerate Örnek-1

```
In [2]: meyveler = ["elma","armut","çilek","kiraz"]
        for sıra, meyve in enumerate(meyveler):
            print("{}- {}".format(sıra,meyve))

0. elma
1. armut
2. çilek
3. kiraz
```

#### enumerate Örnek-2

```
In [1]: iller = ['İstanbul', 'Ankara', 'İzmir', 'Malatya']

        for sıra, iller in enumerate(iller):
            print ('%d: %s' % (sıra + 1, iller))

1: İstanbul
2: Ankara
3: İzmir
4: Malatya
```

#### enumerate Örnek-3

```
In [1]: fruits = ['Banana', 'Apple', 'Lime']
        loud_fruits = [fruit.upper() for fruit in fruits]
        print(loud_fruits)

['BANANA', 'APPLE', 'LIME']

In [2]: list(enumerate(fruits))

Out[2]: [(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

### len()

len() fonksiyonu liste içerisindeki elemanların sayısını bulur.

#### len() Örnek-1

```
In [1]: diller=["Türkçe","İngilizce","Fransızca","Rusça","Almanca","İtalyanca"]
        len(diller)

Out[1]: 6
```



## Toplu Halde Liste Fonksiyonları

Fonksiyon	İşlevi
list	Boş bir liste oluşturmak veya karakter dizilerinden bir liste oluşturmak için kullanılır.
append	Listenin sonuna eleman eklemek için kullanılır.
del	Listeyi silmek veya listede indeks değerine göre belirlenen elemanı listeden çıkarmak için kullanılır.
remove	İsmi ile belirtilen bir elemanı listeden kaldırmak için kullanılır.
sort	Listeyi küçükten büyüğe veya a'dan z'ye sıralamak için kullanılır.
reverse	Listeyi tersine çevirmek için kullanılır.
count	Belirtilen elemanın listede kaç tane olduğu bulmak için kullanılır.
index	İstenilen elemanın listedeki indisini bulmak için kullanılır.
pop	Listenin sonundaki elemanı listeden kaldırmak için kullanılır.
enumerate	Liste elemanlarının indeks değerlerine de ulaşmak için kullanılır.
len	Liste içindeki elemanların sayısını bulmak için kullanılır.
extend	Listenin sonuna başka bir listeyi eklemek için kullanılır.
copy	Listeyi başka bir listeye kopyalamak için kullanılır.
insert	Listenin istenilen bir indisine eleman eklemek için kullanılır.
clear	Listenin içeriğini boşaltmak yani listeyi temizlemek için kullanılır.

Aşağıda verilen tabloda liste fonksiyonları toplu halde verilmiştir.



## karma Örnek-1

```
In [1]: listemiz = [3, 1, 2, 3] # Liste oluşturma

print (listemiz) # Listenin tamamını ekrana yazdıralım

print (listemiz[0]) #Listenin ilk elemanını yazdıralım
print (listemiz[1]) #Listenin ikinci elemanını yazdıralım

print (listemiz[-1]) #Listenin son elemanının yazdıralım
print (listemiz[-3]) #Listenin sondan iki önceki elemanının yazdıralım

[3, 1, 2, 3]
3
1
3
1
```

## karma Örnek-2

```
In [1]: listemiz = [3, 1, 2, 3] # Liste oluşturma
listemiz[2] = 'yapay' # Listelerde farklı veri tipleri tutulabilir
print (listemiz)
```

```
[3, 1, 'yapay', 3]
```

```
In [2]: # Listenin en sonuna `append` ile veri eklenebilir
listemiz.append('zeka')
print (listemiz)
```

```
[3, 1, 'yapay', 3, 'zeka']
```

```
In [3]: # Listenin son elemanını at ve atılan elameni ekrana yazdır
son_eleman = listemiz.pop()
print(son_eleman)
print(listemiz)
```

```
zeka
[3, 1, 'yapay', 3]
```

```
In [4]: # index() ile kontrol edilen elemanın listedeki indisini görebiliriz.
listemiz.index('yapay')
```

```
Out[4]: 2
```

```
In [5]: # count() ile kontrol edilen elemandan listede kaç tane olduğunu görebiliriz.
listemiz.count(3)
```

```
Out[5]: 2
```

```
In [6]: # remove() ile istediğimiz bir elemanı listeden silebiliriz.
listemiz.remove('yapay')
print(listemiz)
```

```
[3, 1, 3]
```

### karma Örnek-3

```
In [1]: listemiz = [3, 1, 2, 3]
# sort() ile listeyi sıralayabiliriz.
listemiz.sort()
listemiz
```

```
Out[1]: [1, 2, 3, 3]
```

```
In [2]: # reverse() ile listeyi tersine çevirebiliriz.
listemiz.reverse()
listemiz
```

```
Out[2]: [3, 3, 2, 1]
```

```
In [3]: # extend() ile listenin sonuna başka bir listeyi ekleyebiliriz.
ek_liste = [1, 11, 111, 1111]
listemiz.extend(ek_liste)
listemiz
```

```
Out[3]: [3, 3, 2, 1, 1, 11, 111, 1111]
```

```
In [4]: # copy() ile listemizi başka bir listeye kopyalayabiliriz.
yeni_liste = listemiz.copy()
yeni_liste
```

```
Out[4]: [3, 3, 2, 1, 1, 11, 111, 1111]
```

```
In [5]: # insert() ile listemizin istediğimiz bir indisine eleman ekleyebiliriz.
yeni_liste.insert(5,555) # 5. indise 555 ekle
yeni_liste
```

```
Out[5]: [3, 3, 2, 1, 1, 555, 11, 111, 1111]
```

```
In [6]: # clear() ile listemizin içeriğini boşaltabiliriz.
yeni_liste.clear()
yeni_liste
```

```
Out[6]: []
```

#### karma Örnek-4

```
In [1]: B= [[1,2,3], [4,5,6] , 'third element', (2,4,5)]
```

```
In [2]: len(B)
```

```
Out[2]: 4
```

```
In [3]: print(B[0])
```

```
[1, 2, 3]
```

```
In [4]: print(B[1])
```

```
[4, 5, 6]
```

```
In [5]: print(B[1][2])
```

```
6
```

```
In [6]: print(B[2])
```

```
third element
```

```
In [7]: print(B[3])
```

```
(2, 4, 5)
```

B= [[1,2,3], [4,5,6] , 'third element', (2,4,5)]	# B isimli liste oluşturulur.
len(B)	# listenin eleman sayısı bulunur. # 4
print(B[0])	# listenin 1. elemanı yazdırılır. # [1, 2, 3]
print(B[1])	# listenin 2. elemanı yazdırılır. # [4, 5, 6]
print(B[1][2])	# listenin 2. elamanının 3. elemanı yazdırılır. # 6
print(B[2])	# listenin 3. elemanı yazdırılır. # third element
print(B[3])	# listenin 4. elemanı yazdırılır. # (2, 4, 5)

### karma Örnek-5

```
In [1]: list_a = [1, 2, 3, 4]
list_b = [2, 3, 4, 5]

common_num = [a for a in list_a for b in list_b if a == b]

print(common_num)

[2, 3, 4]
```

Liste üreteçleri ile iki listedeki ortak numaralar bulunmuştur.

### karma Örnek-6

```
In [1]: stack = [3, 4, 5]
```

```
In [2]: stack.append(6)
stack.append(7)
stack
```

```
Out[2]: [3, 4, 5, 6, 7]
```

```
In [3]: stack.pop()
```

```
Out[3]: 7
```

```
In [4]: stack
```

```
Out[4]: [3, 4, 5, 6]
```

```
In [5]: stack.pop()
```

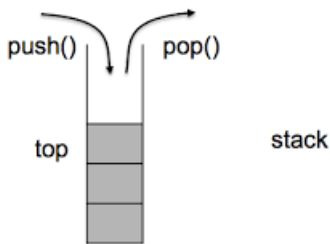
```
Out[5]: 6
```

```
In [6]: stack.pop()
```

```
Out[6]: 5
```

```
In [7]: stack
```

```
Out[7]: [3, 4]
```



Liste yöntemleri, bir listeyi yığın(stack) olarak kullanmayı çok kolaylaştırır.

Yığın(stack) işlemlerinde eklenen son eleman ilk alınan elemandır (“son giren ilk çıkar”).

Yığının en üstüne bir öğe eklemek için `append ()` fonksiyonu kullanılır.

Yığının en üstünden bir öğe almak için, `pop()` fonksiyonu kullanılır.

## karma Örnek-7

```
In [1]: # elemanları iki öğeli liste olan sayılar isimli liste
sayılar=[0,10],[30,50],[55,60],[67,79]
sayılar
```

```
Out[1]: [[0, 10], [30, 50], [55, 60], [67, 79]]
```

```
In [2]: # sayılar listesinin elemanlarını sırayla yazdıralım.
for i in sayılar:
    print(i)
```

```
[0, 10]
[30, 50]
[55, 60]
[67, 79]
```

```
In [3]: # listenin ilk elemanı range fonksiyonunun ilk parametresi olsun.
# listenin ikinci elemanı range fonksiyonunun ikinci parametresi olsun.
for i in sayılar:
    print(range(i[0],i[1]))
```

```
range(0, 10)
range(30, 50)
range(55, 60)
range(67, 79)
```

```
In [4]: for i in sayılar:
        print(range(*i)) # * işareti ile öğelerden oluşan diziler ayrıştırılır.
```

```
range(0, 10)
range(30, 50)
range(55, 60)
range(67, 79)
```

```
In [5]: for i in sayılar:
        print(*range(*i))
        # range fonksiyonunda çıktı alırken döngü kurmak için * kullanılabilir.
```

```
0 1 2 3 4 5 6 7 8 9
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
55 56 57 58 59
67 68 69 70 71 72 73 74 75 76 77 78
```

Yukarıdaki örnekte range() fonksiyonunun listelerle birlikte kullanıldığı bir uygulama yapılmıştır.

Öncelikle iki öğeli elemanlardan oluşan sayılar isimli bir liste oluşturulmuştur. Daha sonra sayılar listesinin tüm elemanları sırayla yazdırılmıştır.

Listenin ilk elemanının range fonksiyonunun ilk parametresi, listenin ikinci elemanının range fonksiyonunun ikinci parametresi olması istenilmiştir.

```
for i in sayılar:
```

```
    print(range(*i))    # * işareti ile öğelerden oluşan diziler ayrıştırılır.
```

```
for i in sayılar:
```

```
    print(*range(*i))    # range fonksiyonunda çıktı alırken döngü kurmak için * kullanılabilir.
```

```
In [1]: liste = []  
  
while True:  
    sayi = input("Bir sayı giriniz: (çıkamak için q)")  
  
    if sayi == "q":  
        break  
  
    sayi = int(sayi)  
  
    if sayi not in liste:  
        liste += [sayi]  
        print(liste)  
    else:  
        print("Bu sayıyı daha önce girdiniz!")
```

```
Bir sayı giriniz: (çıkamak için q)12  
[12]  
Bir sayı giriniz: (çıkamak için q)20  
[12, 20]  
Bir sayı giriniz: (çıkamak için q)7  
[12, 20, 7]  
Bir sayı giriniz: (çıkamak için q)6  
[12, 20, 7, 6]  
Bir sayı giriniz: (çıkamak için q)8  
[12, 20, 7, 6, 8]  
Bir sayı giriniz: (çıkamak için q)30  
[12, 20, 7, 6, 8, 30]  
Bir sayı giriniz: (çıkamak için q)3  
[12, 20, 7, 6, 8, 30, 3]  
Bir sayı giriniz: (çıkamak için q)q
```

Yukarıdaki örnekte olduğu gibi, kullanıcı tarafından aynı verinin birden fazla girilmesini önlemek için de listelerden yararlanabiliriz.

## Sözlük (Dictionary)

Süslü parantezler arasına, sırasız anahtar:değer çiftleri olarak yazılır.

Eleman erişiminde indis yerine anahtar kullanılır. Bu yüzden anahtarlar benzersizdir (unique).

Elemanları değiştirilebilir (mutable) özelliktedir.

### Sözlük Örnek-1

```
In [1]: notlar={'ali':90,'neriman':93,'ismet':91,'emre':97}
notlar
```

```
Out[1]: {'ali': 90, 'neriman': 93, 'ismet': 91, 'emre': 97}
```

```
In [2]: notlar['emre']=99
notlar
```

```
Out[2]: {'ali': 90, 'neriman': 93, 'ismet': 91, 'emre': 99}
```

```
In [3]: del notlar['ismet']
notlar
```

```
Out[3]: {'ali': 90, 'neriman': 93, 'emre': 99}
```

Yukarıda verilen örneği detaylı bir şekilde inceleyelim.

```
In [1]: notlar={'ali':90,'neriman':93,'ismet':91,'emre':97}
notlar
```

```
Out[1]: {'ali': 90, 'neriman': 93, 'ismet': 91, 'emre': 97}
```

notlar isimli bir sözlük tanımlanmıştır.

```
In [2]: notlar['emre']=99
notlar
```

```
Out[2]: {'ali': 90, 'neriman': 93, 'ismet': 91, 'emre': 99}
```

notlar isimli sözlüğün 'emre' anahtar değerine sahip elemanının değeri 99 olarak değiştirilmiştir.

```
In [3]: del notlar['ismet']
notlar
```

```
Out[3]: {'ali': 90, 'neriman': 93, 'emre': 99}
```

notlar isimli sözlüğün 'ismet' anahtar değerine sahip elemanı sözlükten silinmiştir.



## Sözlük Örnek-2

```
In [1]: meyveler={"Elma":3, "Armut":4, "Kiraz":5}
print(meyveler["Elma"])
```

3

```
In [2]: print(meyveler["Armut"])
```

4

```
In [3]: print(meyveler["Kiraz"])
```

5

```
In [4]: print(meyveler["Çilek"])
```

```
-----
KeyError                                Traceback (most recent call last)
<ipython-input-4-73ae1934eb2f> in <module>
----> 1 print(meyveler["Çilek"])

KeyError: 'Çilek'
```

Yukarıdaki örnekte meyveler isimli bir sözlük tanımlanmıştır.

meyveler isimli sözlüğün 'Elma' anahtar değerine sahip elemanın değeri 3 olarak atanmıştır.

meyveler isimli sözlüğün 'Armut' anahtar değerine sahip elemanın değeri 4 olarak atanmıştır.

meyveler isimli sözlüğün 'Kiraz' anahtar değerine sahip elemanın değeri 5 olarak atanmıştır.

```
In [4]: print(meyveler["Çilek"])
```

```
-----
KeyError                                Traceback (most recent call last)
<ipython-input-4-73ae1934eb2f> in <module>
----> 1 print(meyveler["Çilek"])

KeyError: 'Çilek'
```

meyveler isimli sözlüğün 'Çilek' anahtar değerine sahip elemanı olmadığı için **KeyError** hatası verecektir.

## Fonksiyonlar

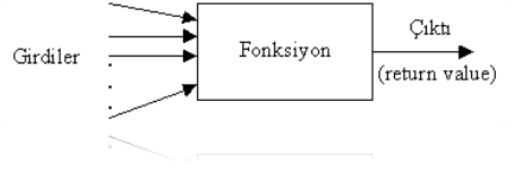
Fonksiyonlar, bir dizi işlemi tek adımda yapmamızı sağlayan, yani verileri işleyen ve sonuç üreten komutlardır.

Fonksiyonlar bizi tekrarlı bir şekilde kod yazmaktan kurtarır.

Fonksiyon, yalnızca çağrıldığında çalışan bir kod bloğudur.

Parametreler olarak bilinen verileri bir fonksiyona aktarabilirsiniz.

Bir fonksiyon, sonuç olarak bir veriyi geri döndürebilir.



## Fonksiyon Oluşturma

Python'da **def** anahtar sözcüğü kullanılarak bir fonksiyon tanımlanabilir.

Python programlama dilinde fonksiyon tanımlamak için **def** yapısı kullanılır.

```
In [1]: def my_function():
        print("Hello from a function")
```

## Fonksiyon Çağırma

Fonksiyonlar ihtiyaç duyulan yerde çağrılarak programcının işini kolaylaştırır.

Bir fonksiyonu çağırarak için, fonksiyonun adı yazılır ve ardından parantez kullanılır.

```
In [1]: def my_function():
        print("Hello from a function")

        my_function()

        Hello from a function
```

Örnekte ilk olarak, `my_function()` isimli bir fonksiyon tanımlanmıştır.

Bu fonksiyon "Hello from a function" ifadesini mesaj olarak yazdırmaktadır.

Fonksiyonu çağırarak için aşağıdaki ifade kullanılmıştır.

```
my_function()
```

## Fonksiyon Örnek-1

```
In [1]: def selamla():
        print("Merhaba!")
        print("Nasılsın?")
        selamla()

        Merhaba!
        Nasılsın?
```

`selamla()` isimli bir fonksiyon tanımlanmıştır. Bu fonksiyon çağırıldığında, ekrana alt alta gelecek şekilde "Merhaba!" ve "Nasılsın?" mesajlarını yazdırmaktadır.

## Parametreler

Fonksiyonlara herhangi bir deęer parametre olarak gnderilebilir. Parametreler, fonksiyonların parantezleri arasında yazılan gelerdir. Parametreler, fonksiyon adından sonra parantez iinde belirtilir. Fonksiyonlar farklı sayıda parametre alabilir. İstedięimiz kadar parametre ekleyebiliriz. Eklenecek parametreleri virgl ile ayırmak gerekecektir.

### Parametreler rnek-1

```
In [1]: def my_function(fname):
        print("Hello " + fname + "!")

        my_function("Sinem")
        my_function("Arda")
        my_function("Demet")

        Hello Sinem!
        Hello Arda!
        Hello Demet!
```

Yukarıdaki rnekte my\_function() isimli bir fonksiyon tek parametrelili (fname) olarak tanımlanmıştır. Fonksiyon aęırıldığında, fonksiyon iinde kullanılan parametreyi isim olarak alıp ekrana "Hello Sinem!" gibi mesaj yazdırılır.

### Varsayılan Deęerli Parametreler

Fonksiyona gnderilen herhangi bir parametreye varsayılan deęer atandığında, fonksiyona bir deęer parametre olarak gnderilmezse, fonksiyon bu varsayılan deęere gre alıřacaktır. Fonksiyonu parametre gndermeden aęırırsak, varsayılan deęeri kullanacaktır.

### Varsayılan Deęerli Parametreler rnek-1

```
In [1]: def my_function(country = "Norway"):
        print("I am from " + country)

        my_function("Turkey")
        my_function()

        I am from Turkey
        I am from Norway
```

my\_function() isimli fonksiyon ilk kez aęırılırken "Turkey" parametre deęeri olarak aęırılmıştır.

my\_function() isimli fonksiyon ikinci kez aęırılırken herhangi bir parametre deęeri olmadan aęırılmıştır. Dolayısıyla gnderilmeyen parametre deęeri varsayılan deęer olarak "Norway" kabul edilmiştir.

### Varsayılan Deęerli Parametreler rnek-2

```
In [1]: def selamla(isim="İsimsiz"):
        print("Merhaba", isim)

        selamla()
        selamla("Serkan")

        Merhaba İsimsiz
        Merhaba Serkan
```

selamla() isimli fonksiyon ilk kez aęırılırken herhangi bir parametre deęeri olmadan aęırılmıştır. Dolayısıyla gnderilmeyen parametre deęeri varsayılan deęer olarak "İsimsiz" kabul edilmiştir.

## Geri Dönüş Değeri

Bir fonksiyonun bir değer döndürmesini sağlamak için return ifadesini kullanılır. Eğer fonksiyon return ifadesiyle herhangi bir çıktı vermiyorsa, fonksiyon çağırıldığında üretilecek return değeri **None** olur.

### Geri Dönüş Değeri Örnek-1

```
In [1]: def my_function(x):
        return 5 * x

        print(my_function(3))
        print(my_function(5))
        print(my_function(9))

        15
        25
        45
```

## Parametre Olarak Liste Göndermek

Herhangi bir parametre türünü bir fonksiyona (karakter kümesi, sayı, liste, sözlük vb.) gönderebilirsiniz. Parametreler, aynı fonksiyonun içindeki veri türü olarak değerlendirilir. Bir listeyi parametre olarak gönderirseniz, liste fonksiyona ulaştığında o yine bir liste olarak değerlendirilir.

### Parametre Olarak Liste Göndermek Örnek-1

```
In [1]: def my_function(food):
        for x in food:
            print(x)

        fruits = ["apple", "banana", "cherry"]

        my_function(fruits)

        apple
        banana
        cherry
```

### Parametre Olarak Liste Göndermek Örnek-2

```
In [1]: # double() fonksiyonu tanımlama
        def double(x):
            return 2*x

        [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
In [2]: # double() fonksiyonu üzerinde hesaplanmış liste kullanabiliriz
        list1=[double(x) for x in range(10)]
        print(list1)

        [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
In [3]: # double() fonksiyonu üzerinde hesaplanmış liste kullanırken koşul eklenebilir
        list2=[double(x) for x in range(10) if x%2==0]
        print(list2)

        [0, 4, 8, 12, 16]
```

## Toplama Örnek

Fonksiyonun kendisine tamsayı olarak verilen 3 parametresi girilen bir toplama( ) isimli fonksiyon tanımlayalım.

toplama( ) isimli fonksiyon, tamsayı olarak girilen 3 değeri parametre olarak alsın.

Fonksiyon geri dönüş değeri olarak parametre olarak gönderilen 3 değerın toplamını göndersin.

```
In [1]: def toplama(a,b,c):
        print("Toplam:", a+b+c)

        a=toplama(3,4,5)

        print("Sayıların toplamı:", a)

        Toplam: 12
        Sayıların toplamı: None
```

Fonksiyonda return değeri tanımlanmadığı için fonksiyonun ürettiği değer **None** olmuştur.

```
In [1]: def toplama(a,b,c):
        return a+b+c

        a=toplama(3,4,5)

        print("Sayıların toplamı:",a)

        Sayıların toplamı: 12
```

Fonksiyonda **return** değeri tanımlandığı zaman ise fonksiyonun ürettiği değer 12 olmuştur.

## Yıldız Piramidi Örnek

Kendisine tamsayı olarak verilen parametre sayısı kadar yıldız piramidi oluşturan yıldiz\_piramit( ) isimli fonksiyon tanımlayalım.

yildiz\_piramit ( ) isimli fonksiyon, 1'den başlayarak girilen değer sayısınca ekrana alt alta gelecek şekilde yıldız (\*) karakterini yazdırsın.

```
In [2]: def yildiz_piramit(r):
        for x in range(r+1):
            print('*' *x)

        sayi=int(input('sayıyı giriniz:'))

        yildiz_piramit(sayi)

        sayıyı giriniz:5

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

### Dairenin Alanı Örnek

Yarıçapı girilen bir dairenin alanını oluşturulan `daire_alan()` isimli fonksiyonla hesaplatalım.

`daire_alan()` isimli fonksiyon, tamsayı olarak girilen yarıçap değerini parametre olarak alsın.

Fonksiyon geri dönüş değeri olarak hesaplanan dairenin alanını göndersin.

Dairenin alanı aşağıdaki formüle göre hesaplanır.

Dairenin alanı:  $\pi * r^2$  ( $\pi=3.14$ ) ( $r$ =yarıçap,  $r$  cm olarak tamsayı)

```
In [1]: #yarıçapı girilen dairenin alanını hesaplar
def daire_alan(r):
    return 3.14 * pow(r,2)

yaricap=int(input("Yarıçapı Giriniz:"))
print("Dairenin Alanı:",daire_alan(yaricap))

Yarıçapı Giriniz:5
Dairenin Alanı: 78.5
```

### Dikdörtgenin Alanı Örnek

Genişliği ve yüksekliği girilen bir dikdörtgenin alanını tanımlayacağımız `dikdortgen_alan()` isimli fonksiyonla hesaplatalım.

`dikdortgen_alan()` isimli fonksiyon, tamsayı olarak girilen genişlik ve yükseklik değerlerini parametre olarak alsın.

Fonksiyon geri dönüş değeri olarak hesaplanan dikdörtgenin alanını göndersin.

Dikdörtgenin alanı aşağıdaki formüle göre hesaplanır.

Dikdörtgenin alanı: Genişlik \* Yükseklik

```
In [1]: #genişliği ve yüksekliği girilen dikdörtgenin alanını hesaplar
def dikdortgen_alan(genislik,yukseklk):
    return genislik * yukseklik

genislik=int(input("Genişlik değerini giriniz:"))
yukseklk=int(input("Yükseklik değerini giriniz:"))

print("Dikdörtgenin Alanı:",dikdortgen_alan(genislik,yukseklk))

Genişlik değerini giriniz:8
Yükseklik değerini giriniz:12
Dikdörtgenin Alanı: 96
```

Genişlik değeri 8, yükseklik değeri 12 girilen bir dikdörtgenin alanı 96 olarak hesaplanmıştır.

## Selamla Örnek

selamla() fonksiyonunu parametre almayacak şekilde aşağıdaki gibi tanımlayabiliriz.

```
In [1]: def selamla():
        print("Merhaba!")
        print("Nasılsın?")
        selamla()

        Merhaba!
        Nasılsın?
```

Parametre olarak kendisine gelen ismi kullanarak selamlamak için kullanılacak selamla() fonksiyonunu Python programlama dilinde aşağıdaki gibi tanımlayabiliriz.

```
In [1]: def selamla(isim):
        print("Merhaba", isim, "Nasılsın?")
        selamla("Serkan")
        selamla("Ahmet")

        Merhaba Serkan Nasılsın?
        Merhaba Ahmet Nasılsın?
```

## Büyük Sayı Bulma Örnek

Parametre olarak kendisine gelen iki sayıdan büyük olanı bulmak için kullanılacak buyuksayi() fonksiyonunu Python programlama dilinde aşağıdaki gibi tanımlayabiliriz.

```
In [1]: def buyuksayi(sayi1,sayi2):
        buyuk=sayi1 if sayi1>=sayi2 else sayi2
        return buyuk

        a=float(input("Birinci sayıyı giriniz:").replace(',','.'))
        b=float(input("İkinci sayı giriniz:").replace(',','.'))

        print("Büyük Sayı:",buyuksayi(a,b),sep='')

        Birinci sayıyı giriniz:5
        İkinci sayı giriniz:15
        Büyük Sayı:15.0
```

```
In [1]: def buyuksayi(sayi1,sayi2):
        buyuk=sayi1 if sayi1>=sayi2 else sayi2
        return buyuk

        a=float(input("Birinci sayıyı giriniz:").replace(',','.'))
        b=float(input("İkinci sayı giriniz:").replace(',','.'))

        print("Büyük Sayı:",buyuksayi(a,b),sep='')

        Birinci sayıyı giriniz:3,545
        İkinci sayı giriniz:4,8546
        Büyük Sayı:4.8546
```

### Sıralama Örnek

Bir sayı dizisini küçükten büyüğe (ya da tam tersi) sıralamak için kullanılacak sırala() fonksiyonunu Python programlama dilinde aşağıdaki gibi tanımlayabiliriz.

```
In [1]: def sirala(dizi):
        if len(dizi) <= 1:
            return dizi
        referans = dizi[len(dizi) // 2]
        sol = [x for x in dizi if x < referans]
        orta = [x for x in dizi if x == referans]
        sag = [x for x in dizi if x > referans]
        return sirala(sol) + orta + sirala(sag)

liste=[7,30,16,8,10,15,27,1,5]
print (sirala(liste))

[1, 5, 7, 8, 10, 15, 16, 27, 30]
```

### Fibonacci Serileri Örnek

```
In [2]: def fibonacci(n):
        a,b=0,1
        while a<n:
            print(a,end=' ')
            a,b=b,a+b
        print()
        fibonacci(1000)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Fibonacci serilerinin genel formülü

$$F_n = F(n) = \begin{cases} 0 & n = 0; \\ 1 & n = 1; \\ F(n-1) + F(n-2) & n > 1. \end{cases}$$



## Recursion (Özyineleme)

Python programlama dili fonksiyon özyinelemesini de kabul eder. Bu tanımlanmış bir fonksiyonun kendisini çağırabileceği anlamına gelir.

Özyineleme ortak bir matematiksel ve programlama kavramıdır. Bir fonksiyonun kendisini çağırdığı anlamına gelir. Bunun bir sonuca ulaşmak için verilerde dolaşabileceğiniz anlam avantajına sahiptir.

Geliştiricinin özyinelemeye çok dikkat etmesi gerekir. Çünkü hiç bitmeyen ya da aşırı miktarda bellek ya da işlemci gücü kullanan bir fonksiyonu yazmak oldukça kolay olabilir. Ancak, doğru yazıldığında özyineleme programlama için çok verimli ve matematiksel olarak zarif bir yaklaşım olabilir.

### Recursion Örnek-1

```
In [1]: def tri_recursion(k):
        if(k>0):
            result = k+tri_recursion(k-1)
            print(result)
        else:
            result = 0

        return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

Recursion Example Results

1

3

6

10

15

21

Out[1]: 21

Bu örnekte, `tri_recursion ()`, kendisini çağırmak için tanımladığımız ("özyinelemeli") bir fonksiyondur.

`k` değişkenini, her tekrarlama işleminde 1 azaltan veri olarak kullanırız.

Özyineleme, koşul 0'dan büyük olmadığına (yani 0 olduğunda) sona erer.

Kodlamaya yeni başlayan birisine bunun tam olarak nasıl çalıştığını anlatmak zor olacaktır.

Hesaplama mantığını anlamak için değerleri değiştirerek kodu test etmek gerekecektir.

Öğrenmenin en iyi yolu onu test etmek ve değiştirmektir.