Mantıksal Değerler ve Karşılaştırma Operatörleri

Bu konuda, koşullu durumları incelemeden önce bilmemiz gereken mantıksal değerleri ve karşılaştırma operatörlerini öğrenmeye çalışacağız.

Mantıksal Değerler (Boolean)

Mantıksal değerler ya da ingilizce ismiyle *boolean* değerler aslında Pythonda bir veri tipidir ve iki değere sahiplerdir: **True ve False**. Şimdi bu değerlerden değişkenler oluşturalım.

In [1]:

```
a =  True
print(type(a))
```

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

In [ ]:

```
b = False
print(type(b))
```

Pythonda bir sayı değeri eğer 0'dan farklıysa **True**, 0 ise **False** olarak anlam kazanır. Bunu *bool()* fonksiyonuyla dönüştürme yaparak görebiliriz.

In [3]:

```
bool(12.4)
```

Out[3]:

```
True
```

In [4]:

```
bool(0.0)
```

Out[4]:

```
False
```

In [5]:

```
bool(121212)
```

Out[5]:

```
True
```

In [6]:

```
bool(-1)
```

Out[6]:

```
True
```

In [7]:

```
bool(0)
```

Out[7]:

```
False
```

Bool değerleri ayrıca birazdan göreceğimiz bir karşılaştırma operatöründen sonra ortaya çıkan sonuç değeridir

In [8]:

```
1 > 2 # 1 2 den büyük olmadığı için sonuç False olacaktır.
```

Out[8]:

```
False
```

In [10]:

```
1 < 2 # 1 2 den küçük olduğu için sonuç True olacaktır.
```

Out[10]:

```
True
```

Ayrıca Pythonda eğer bir değişkenin değerini sonradan belirlemek isterseniz geçici olarak bu değişken **None** (atanmamış anlamında) değerine eşitleyebilirsiniz.

In [11]:

```
a = None # Henüz değer atamadık
print(a)
```

```
None
```

In [12]:

```
a = 4 # Şimdi değer atıyoruz.
print(a)
```

```
4
```

Karşılaştırma Operatörleri

Operatör	Görevi	Örnek
==	İki değer birbirine eşitse True, değilse False değer döner.	2 == 2 (True) , 2 == 3 (False)
!=	İki değer birbirine eşit değilse True, diğer durumda False döner.	2 != 2 (False), 2 != 3 (True)
>	Soldaki değer sağdaki değerden büyükse True, değilse False döner.	3 > 2 (True), 2 > 3 (False)
<	Soldaki değer sağdaki değerden küçükse True, değilse False döner.	2 < 3 (True) , 3 < 2 (False)
>=	Soldaki değer sağdaki değerden büyükse veya sağdaki değere eşitse True, değilse False döner.	3 >= 2 (True),3 >= 3 (True) , 2 >= 3 (False)
<=	Soldaki değer sağdaki değerden küçükse veya sağdaki değere eşitse True, değilse False döner.	3 <= 2 (False),3 <= 3 (True) , 2 <= 3 (True)

Örneklere bakalım.

In [13]:

```
"Mehmet" == "Mehmet"
```

Out[13]:

True

In [14]:

```
"Mehmet" == "Murat"
```

Out[14]:

False

In [15]:

```
"Mehmet" != "Murat"
```

Out[15]:

True

In [16]:

```
"Oğuz" < "Murat" # Alfabetik olarak bakar.
```

Out[16]:

False

In [17]:

```
2 < 3
```

Out[17]:

True

In [18]:

```
54 >= 54
```

Out[18]:

True

In [19]:

```
98 > 32
```

Out[19]:

True

In [20]:

```
34 <= 45
```

Out[20]:

True

Mantıksal Bağlaçlar

and Operatörü

Bu mantıksal bağlaç, bütün karşılaştırma işlemlerinin sonucunun **True** olmasına bakar. Bağlanan karşılaştırma işlemlerinin hepsinin kendi içinde sonucu **True** ise genel sonuç **True** , diğer durumlarda ise sonuç **False** çıkar. Kullanımı şu şekildedir.

In [1]:

```
1 < 2 and "Murat" == "Murat"
```

Out[1]:

```
True
```

In [2]:

```
2 > 3 and "Murat" == "Murat"
```

Out[2]:

```
False
```

In [3]:

```
2 == 2 and 3.14 < 2.54 and "Elma" != "Armut"
```

Out[3]:

```
False
```

İşlemlerin birinin bile sonucu **False** ise genel işlemin sonucu **False** çıkmaktadır.

or Operatörü

Bu mantıksal bağlaç, bütün karşılaştırma işlemlerinin sonuçlarından **en az birinin True** olmasına bakar. Bağlanan karşılaştırma işlemlerinin **en az birinin True** olmasında genel sonuç **True** , diğer durumlarda ise sonuç **False** çıkar. Kullanımı şu şekildedir.

In [4]:

```
1 < 2 or "Murat" != "Murat"
```

Out[4]:

```
True
```

In [5]:

```
2 > 3 or "Murat" != "Murat"
```

Out[5]:

```
False
```

In [6]:

```
2 > 3 or "Murat" != "Murat" or 3.14 < 4.32
```

Out[6]:

True

not operatörü

not operatörü aslında bir mantıksal bağlaç değildir. Bu operatör sadece bir mantıksal değeri veya karşılaştırma işlemininin tam tersi sonuca çevirir. Yani, not operatörü **True** olan bir sonucu **False** , **False** olan bir sonucu **True** sonucuna çevirir.

In [7]:

```
not 2 == 2
```

Out[7]:

False

In [9]:

```
not "Python" == "Php"
```

Out[9]:

True

Operatörleri Beraber Kullanma

In [20]:

```
not (2.14 > 3.49 or ( 2 != 2 and "Murat" == "Murat"))
```

Out[20]:

True

In [22]:

```
"Araba" < "Zula" and ( "Bebek" < "Çoçuk" or (not 14 ))
```

Out[22]:

True

Koşullu Durumlar - if-else koşullu durumları

Python Programlarının çalışma mantığı ¶

Python programları çalışmaya başladığı zaman kodlarımız yukardan başlayarak teker teker çalıştırılır ve çalıştıracak kod kalmayınca programımız sona erer.

In [1]:

```
a = 2
b = 3
c = 4
print(a+b+c)
```

```
9
```

Yukarıdaki basit kodda program teker teker her bir satırı ve ifadeyi çalıştırır ve çalıştıracak kod kalmayınca program sona erer.Ancak Python'da her program bu kadar basit olmayabilir. Çoğu zaman Python programlarımız belirli bloklardan oluşur ve bu bloklar her zaman çalışmak zorunda olmaz.Peki bu bloklar nasıl tanımlanır ? Pythonda bir blok tanımlama işlemi **Girintiler** sayesinde olmaktadır. Örnek olması açısından, Pythonda bloklar şu şekilde oluşabilir.

In [2]:

```
a = 2 # Blok 1 'e ait kod

if (a == 2):
    print(a) # Blok 2'ye ait kod
print("Merhaba") # Blok 1 ' ait kod
```

```
2
Merhaba
```

Dikkat ederseniz burada daha önce görmediğimiz bir şey yaptık ve **if** in bulunduğu satırdan sonraki **print** işlemini bir tab kadar girintili yazdık. Burada gördüğümüz gibi, **girintiler(tab)** Pythonda bir blok oluşturmak için kullanılıyor ve her bloğunun çalıştırılması gerekmiyor.

Koşullu Durumlar

Koşullu durumlar aslında günlük yaşamda sürekli karşılaştığımız durumlardır. Örneğin havanın yağmurlu olma koşuluna göre şemsiyemizi alırız veya uykumuzun gelme koşuluna göre uyuruz. Aslında programlamada da birçok koşullu durumla karşılaşırız. Örneğin , belli koşullara göre belli işlemleri yaparız , belli koşullara göre yapmayız. İşte bunlar koşullu durumların temeli oluşturur.

if Bloğu

if bloğu programımızın içinde herhangi bir yerde belli bir koşulu kontrol edecek isek kullanılan bloklardır.Yazımı şu şekildedir;

```
          if (koşul):
              # if bloğu - Koşul sağlanınca (True) çalışır. Bu hizadaki her işlem bu if bloğuna ai
    t.
              # if bloğu - Girintiyle oluşturulur.
              Yapılacak İşlemler
```

if bloğu *eğer koşul sağlanırsa anlamı taşır*. Eğer if kalıbındaki koşul sağlanırsa (True) if bloğu çalıştırılır, koşul sağlanmazsa (False) if bloğu çalıştırılmaz.

Hemen bir örnek ile koşullu durumları anlamaya çalışalım.

In [7]:

```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
```

```
Yaşınızı giriniz:17
Bu mekana giremezsiniz.
```

In [9]:

```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
```

```
Yaşınızı giriniz:25
```

```
# Negatif mi değil mi ?
sayı = int (input("Sayıyı giriniz:"))

if (sayı < 0):
    print("Negatif Sayı")
```

```
Sayıyı giriniz:-3
Negatif Sayı
```

In [11]:

```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
print("Mekana hoşgeldiniz.")
```

```
Yaşınızı giriniz:25
Mekana hoşgeldiniz.
```

In [12]:

```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
print("Mekana hoşgeldiniz.")
```

```
Yaşınızı giriniz:17
Bu mekana giremezsiniz.
Mekana hoşgeldiniz.
```

else Bloğu

else blokları **if koşulu** sağlanmadığı zaman (False) çalışan bloklardır. Kullanımı şu şekildedir;

```
          else:
              # else bloğu - Yukarısındaki herhangi bir if bloğu (veya ilerde göreceğimiz elif blo
    ğu) çalışmadığı
              # zaman çalışır.
              # else bloğu - Girintiyle oluşturulur.
                  Yapılacak İşlemler
```

In [13]:

```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
else:
    # else bloğu -  if koşulu sağlanmazsa çalışacak.
    print("Mekana hoşgeldiniz.")
```

```
Yaşınızı giriniz:25
Mekana hoşgeldiniz.
```

In [14]:

```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
else:
    # else bloğu -  if koşulu sağlanmazsa çalışacak.
    print("Mekana hoşgeldiniz.")
```

```
Yaşınızı giriniz:17
Bu mekana giremezsiniz.
```

Koşullu Durumlar - if - elif - else koşullu durumları

if-elif-else Blokları

Önceki konumuzda koşullu durumlarımızla sadece tek bir koşulu kontrol edebiliyorduk. Ancak programlamada bir durum bir çok koşula bağlı olabilir. Örneğin bir hesap makinesi programı yazdığımızda kullanıcının girdiği işlemlere göre koşullarımızı belirleyebiliriz. Bu tür durumlar için if-elif-else kalıplarını kullanırız. Kullanımı şu şekildedir;

```
if koşul:
    Yapılacak İşlemler
elif başka bir koşul:
    Yapılacak İşlemler
elif başka bir koşul:
    Yapılacak İşlemler

     //
     //
else:
    Yapılacak İşlemler
```

Programlarımızda, Kaç tane koşulumuz var ise o kadar elif bloğu oluşturabiliriz. Ayrıca **else** in yazılması zorunlu değildir. if - elif - else kalıplarında, hangi koşul sağlanırsa program o bloğu çalıştırır ve if-elif-blokları sona erer.

In [1]:

```
işlem =  int(input("İşlem seçiniz:")) # 3 tane işlemimiz olsun.

if işlem == 1:
    print("1. işlem seçildi.")
elif işlem == 2:
    print("2. işlem seçildi.")
elif işlem == 3:
    print("3. işlem seçildi.")
else:
    print("Geçersiz İşlem!")
```

```
İşlem seçiniz:1
1. işlem seçildi.
```

In [2]:

```
işlem =  int(input("İşlem seçiniz:")) # 3 tane işlemimiz olsun.

if işlem == 1:
    print("1. işlem seçildi.")
elif işlem == 2:
    print("2. işlem seçildi.")
elif işlem == 3:
    print("3. işlem seçildi.")
else:
    print("Geçersiz İşlem!")
```

```
İşlem seçiniz:3
3. işlem seçildi.
```

In [3]:

```
işlem =  int(input("İşlem seçiniz:")) # 3 tane işlemimiz olsun.

if işlem == 1:
    print("1. işlem seçildi.")
elif işlem == 2:
    print("2. işlem seçildi.")
elif işlem == 3:
    print("3. işlem seçildi.")
else:
    print("Geçersiz İşlem!")
```

```
İşlem seçiniz:4
Geçersiz İşlem!
```

In [4]:

```
işlem =  int(input("İşlem seçiniz:")) # 3 tane işlemimiz olsun.

if işlem == 1:
    print("1. işlem seçildi.")
elif işlem == 2:
    print("2. işlem seçildi.")
elif işlem == 3:
    print("3. işlem seçildi.")
```

```
İşlem seçiniz:4
```

if-if-if Blokları

In [ ]:

```
note = float(input("Notunuzu giriniz:"))

if note >= 90:
    print("AA")
elif note >= 85:
    print("BA")
elif note >= 90:
    print("BA")
elif note >= 80:
    print("BB")
elif note >= 75:
    print("CB")
elif note >= 70:
    print("CC")
elif note >= 65:
    print("DC")
elif note >= 60:
    print("DD")
else:
    print("Dersten Kaldınız")
```

In [1]:

```
note = float(input("Notunuzu giriniz:"))

if note >= 90:
    print("AA")
if note >= 85:
    print("BA")
if note >= 90:
    print("BA")
if note >= 80:
    print("BB")
if note >= 75:
    print("CB")
if note >= 70:
    print("CC")
if note >= 65:
    print("DC")
if note >= 60:
    print("DD")
else:
    print("Dersten Kaldınız")
```

```
Notunuzu giriniz:93
AA
BA
BA
BB
CB
CC
DC
DD
```

In [2]:

```
note = float(input("Notunuzu giriniz:"))

if note >= 90:
    print("AA")
if note >= 85:
    print("BA")
if note >= 90:
    print("BA")
if note >= 80:
    print("BB")
if note >= 75:
    print("CB")
if note >= 70:
    print("CC")
if note >= 65:
    print("DC")
if note >= 60:
    print("DD")
else:
    print("Dersten Kaldınız")
```

```
Notunuzu giriniz:80
BB
CB
CC
DC
DD
```

For Döngüleri

in Operatörü

Pythondaki *in* operatörü , bir elemanın başka bir listede,demette veya stringte (karakter dizileri) bulunup bulunmadığını kontrol eder.

In [3]:

```
"a" in "merhaba"
```

Out[3]:

```
True
```

In [4]:

```
"mer" in "merhaba"
```

Out[4]:

```
True
```

In [5]:

```
"t" in "merhaba"
```

Out[5]:

```
False
```

In [6]:

```
4 in [1,2,3,4]
```

Out[6]:

```
True
```

In [7]:

```
10 in [1,2,3,4]
```

Out[7]:

```
False
```

In [8]:

```
4 in (1,2,3)
```

Out[8]:

```
False
```

for Döngüsü

```
            for eleman in veri_yapısı(liste,demet vs):
                Yapılacak İşlemler
```

Bu yapı bize şunu söyler;

```
            eleman değişkeni her döngünün başında listenin,demetin vs. her bir elemanına eşit olaca
    k ve her döngüde
            bu elemanla işlem yapılacak.
```

Listeler Üzerinde Gezinmek

In [1]:

```
liste = [1,2,3,4,5,6,7]

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

```
Eleman 1
Eleman 2
Eleman 3
Eleman 4
Eleman 5
Eleman 6
Eleman 7
```

In [2]:

```
# Liste elemanlarını toplama
liste = [1,2,3,4,5,6,7]
toplam = 0
for eleman in liste:
    toplam += eleman
print("Toplam",toplam)
```

```
Toplam 28
```

In [3]:

```
# Çift elemanları bastırma
liste = [1,2,3,4,5,6,7,8,9]

for eleman in liste:
    if eleman % 2 == 0:
        print(eleman)
```

```
2
4
6
8
```

Karakter Dizileri Üzerinde Gezinmek (Stringler)

In [4]:

```
s =  "Python"
for karakter in s:
    print(karakter)
```

```
P
y
t
h
o
n
```

In [5]:

```
# Her bir karakterleri 3 ile çarpma
s = "Python"

for karakter in s:
    print(karakter * 3)
```

```
PPP
yyy
ttt
hhh
ooo
nnn
```

Demetler üzerinde gezinmek (Demetler)

In [6]:

```
# Listelerle aynı mantık
demet =  (1,2,3,4,5,6,7)

for eleman in demet:
    print(eleman)
```

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

In [9]:

```
# Listelerin için 2 boyutlu demetler

liste = [(1,2),(3,4),(5,6),(7,8)]

for eleman in liste:
    print(eleman) # Herbir elemanın  demet olduğu görebiliyoruz.
```

```
(1, 2)
(3, 4)
(5, 6)
(7, 8)
```

While Döngüleri

```
while (koşul):
    İşlem1
    İşlem2
    İşlem3
      //
      //
```

In [1]:

```
# Döngüde i değerlerini ekrana yazdırma

i = 0

while (i < 10):
    print("i'nin değeri",i)
    i += 1 # Koşulun bir süre sonra False olması için gerekli - Unutmayalım
```

```
i'nin değeri 0
i'nin değeri 1
i'nin değeri 2
i'nin değeri 3
i'nin değeri 4
i'nin değeri 5
i'nin değeri 6
i'nin değeri 7
i'nin değeri 8
i'nin değeri 9
```

In [4]:

```
# Döngüde i değerlerini ekrana yazdırma

i = 0

while (i < 20):
    print("i'nin değeri",i)
    i += 2 # Koşulun bir süre sonra False olması için gerekli -
```

```
i'nin değeri 0
i'nin değeri 2
i'nin değeri 4
i'nin değeri 6
i'nin değeri 8
i'nin değeri 10
i'nin değeri 12
i'nin değeri 14
i'nin değeri 16
i'nin değeri 18
```

In [2]:

```
# Ekrana 40 defa "Python Öğreniyorum" yazdıralım.
i = 0

while (i < 40):
    print("Python Öğreniyorum")
    i +=1
```

```
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
```

In [5]:

```
# Liste üzerinde indeks ile gezinme
liste = [1,2,3,4,5]

a = 0

while (a < len(liste)):
    print("Indeks:",a,"Eleman:",liste[a])
    a +=1
```

```
Indeks: 0 Eleman: 1
Indeks: 1 Eleman: 2
Indeks: 2 Eleman: 3
Indeks: 3 Eleman: 4
Indeks: 4 Eleman: 5
```

Sonsuz Döngü Olayları

while döngüsü kullanırken biraz dikkatli olmamızda fayda var. Çünkü, while döngü koşulunun bir süre sonra **False** olması gerekecek ki döngümüz sonlanabilsin. Ancak eğer biz **while** döngülerinde bu durumu unutursak , döngümüz sonsuza kadar çalışacaktır. Biz buna **sonsuz döngü** olayı diyoruz.

In [ ]:

```
# Bu kodu çalıştırmayalım. Jupyter sıkıntı çıkarabilir :)
i = 0
while (i < 10):
    print(i)
    # i değişkenini artırma işlemi yapmadığımız için i değişkeninin değeri sürekli 0 kalıyor
    # ve döngü koşulu sürekli True kalıyor.
```

range() Fonksiyonu

In [11]:

```
range(0,20) # 0'dan 20' a kadar (dahil değil) sayı dizisi oluşturur.
```

Out[11]:

```
range(0, 20)
```

In [13]:

```
print(*range(0,20)) # Yazdırmak için başına "*" koymamız gerekiyor.
```

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

In [17]:

```
liste = list(range(0,20)) # list fonksiyonuyla listeye dönüştürebilir.
```

In [16]:

```
liste
```

Out[16]:

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

In [18]:

```
print(*range(5,10))
```

```
5 6 7 8 9
```

In [20]:

```
print(*range(15)) # Başlangıç değeri vermediğimiz 0'dan başlar
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

In [22]:

```
print(*range(5,20,2)) # 5'ten 20'ye kadar olan sayıları 2 atlayarak oluşturur.
```

```
5 7 9 11 13 15 17 19
```

In [23]:

```
print(*range(5,100,5)) # 5'ten 100'e kadar olan ve 5 ile bölünebilen sayılar
```

```
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
```

In [28]:

```
print(*range(20,0)) # 20'den geri gelen sayıları oluşturmaz.
```

In [29]:

```
print(*range(20,0,-1)) # 20'den geri gelen sayıları oluşturur.
```

```
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

In [31]:

```
for sayı in range(0,10):
    print(sayı)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [32]:

```
for sayı in range(1,20):
    print("* " * sayı)
```

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

Döngülerde kullanılan ifadeler : break ve continue

break ifadesi

break ifadesi döngülerde programcılar tarafından en çok kullanılan ifadedir. Anlamı şu şekildedir;

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

break ifadesi **sadece ve sadece** içindeki bulunduğu döngüyü sonlandırır. Eğer iç içe döngüler bulunuyorsa ve en içteki döngüde break kullanılmışsa sadece içteki döngü sona erer.

In [2]:

```
i = 0

while (i < 20):
    print(i)
    i +=1
```

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

In [5]:

```
i = 0

while (i < 20):
    print(i)
    if (i == 10):
        break # i'nin değeri 10 olunca bu koşul sağlanıyor ve  break
    i +=1
```

```
0
1
2
3
4
5
6
7
8
9
10
```

In [3]:

```
# for döngüsüyle break kullanalım.
liste = [1,2,3,4,5,6,7,8,9]
for i in liste:
    if (i == 5):

        break
    print(i)
```

```
1
2
3
4
```

In [2]:

```
while True: # Sonsuz döngü. Nasıl sonlandırabiliriz ?
    isim = input("İsminiz(Çıkmak için q tuşuna basın.):")
    if (isim == "q"): # break ile tabii ki.
        print("Çıkış yapılıyor...")
        break
    print(isim)
```

```
İsminiz(Çıkmak için q tuşuna basın.):q
Çıkış yapılıyor...
```

continue ifadesi

continue ifadesi *break*'e göre biraz daha az kullanılan bir ifadedir.

```
    Döngü herhangi bir yerde ve herhangi bir zamanda continue ifadesiyle karşılaştığı zaman ger
i kalan işlemlerini
    yapmadan direk bloğunun başına döner.
```

In [7]:

```
liste = [1,2,3,4,5,6,7,8,9,10]

for i in liste:
    print("i:",
```

```
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9
i: 10
```

In [6]:

```
liste = [1,2,3,4,5,6,7,8,9]


for i in liste:
    if (i == 3 or i == 5):
        continue
    print("i:",i)
```

```
i: 1
i: 2
i: 4
i: 6
i: 7
i: 8
i: 9
```

List Comprehension

In [1]:

```
# Listelerdeki append metodunu hatırlayalım.
liste = [1,2,3,4]
liste.append(5)
```

In [2]:

```
liste
```

Out[2]:

```
[1, 2, 3, 4, 5]
```

In [4]:

```
# liste1'den liste2'yi oluşturalım.

liste1 = [1,2,3,4,5]

liste2 = list() # veya liste2 = [] ikisi de boş liste oluşturur.


for i in liste1:
    liste2.append(i) # liste2 'ye liste1 in elemanları for döngüsü yardımıyla atıyoruz.

print(liste2)
```

```
[1, 2, 3, 4, 5]
```

In [5]:

```
liste1 = [1,2,3,4,5] # Örnek 1

liste2 = [i for i in liste1] # List Comprehension

print(liste2)
```

```
[1, 2, 3, 4, 5]
```

In [6]:

```
liste1 = [1,2,3,4,5] # Örnek 2

liste2 = [i*2 for i in liste1] # List Comprehension

print(liste2)
```

```
[2, 4, 6, 8, 10]
```

In [11]:

```
liste1 = [(1,2),(3,4),(5,6)] # Örnek 3

liste2 = [i*j for (i,j) in liste1] # List Comprehension

print(liste2)
```

```
[2, 12, 30]
```

In [13]:

```
liste1 = [1,2,3,4,5,6,7,8,9,10] # Örnek 4

liste2 = [i for i in liste1 if not (i == 4 or i == 9)] # List Comprehension

print(liste2)
```

```
[1, 2, 3, 5, 6, 7, 8, 10]
```

In [19]:

```
s = "Python"  # Örnek 5

liste = [i * 3 for i in s] # List Comprehension

print(liste)
```

```
['PPP', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']
```

Metodlar

Metod nedir ?

Şimdiye kadar Pythonda kullanabildiğimiz bir çok veri tipi gördük ve bazı veritipleri üzerinde bu veritiplerinin metodlarını kullandık. Aslında bu veritiplerin oluşturulan her bir değişken Pythonda **obje(object)** olarak düşünülür ve Python geliştiricileri bu objelere birçok metod tanımlamıştır.

Metodlar bir obje üzerinde belli işlemleri gerçekleştiren objelere özgü fonksiyonlardır ve objelerin üzerinde metodlar şu şekilde kullanılır.

```
obje.herhangi_bir_metod(değerler(opsiyonel))
```

Örneğin bir **liste objesi** oluşturduğumuz zaman bu objenin üzerinde belli metodları uygulayabiliriz.

In [4]:

```
liste =  [1,2,3,4,5,6]

liste.insert(1,"Murat")
```

In [5]:

```
liste
```

Out[5]:

```
[1, 'Murat', 2, 3, 4, 5, 6]
```

In [6]:

```
liste.pop()
```

Out[6]:

```
6
```

In [7]:

```
liste
```

Out[7]:

```
[1, 'Murat', 2, 3, 4, 5]
```

Örneğin liste metodlarına erişmek için Jupyterde **Tab** tuşuna basabiliriz. Ayrıca bir metodun ne iş yaptığını anlamak için **help** fonksiyonunu kullanabiliriz.

Fonksiyonlar

Fonksiyonlar programlamada belli işlevleri olan ve tekrar tekrar kullandığımız yapılardır. Örneğin kursumuzun başlarından beri kullandığımız **print()** fonksiyonunun görevi **içine gönderdiğimiz değerleri** ekrana yazdırdırmaktır. Bu fonksiyon Python geliştiricileri tarafından bir defa yazılmış ve biz de bu fonksiyonu programlarımızın değişik yerlerinde tekrar tekrar kullanıyoruz. İşte fonksiyonların kullanım amacı tam olarak budur. Fonksiyonlar bir defa tanımlanır ve programlarda ihtiyacımız olduğu zaman kullanırız. Ayrıca fonksiyonlar kod tekrarını engeller ve kodlarımız daha derli toplu durur.

Fonksiyonların Tanımlanması

Fonksiyon tanımlamanın yapısı şu şekildedir;

```
def fonksiyon_adı(parametre1,parametre2..... (opsiyonel)):
    # Fonksiyon bloğu
    Yapılacak işlemler
    # dönüş değeri - Opsiyonel
```

In [2]:

```
type(selamla) # Henüz tanımlamadık.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-02f6a6f95c12> in <module>()
----> 1 type(selamla)

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

In [3]:

```
def selamla():
    print("Selam arkadaşlar...")
    print("Nasılsınız?")
```

In [4]:

```
type(selamla) # Fonksiyonumuz tanımlandı.
```

Out[4]:

```
function
```

Fonksiyonların Kullanılması veya Çağrılması (Function Call)

```
fonksiyon_adı(Argüman1,Argüman2....)
```

In [6]:

```
type(selamla) # Tanımlı
```

Out[6]:

```
function
```

In [7]:

```
selamla() # Fonksiyon parametre almadığında içine argümanlarımızı göndermiyoruz.
```

```
Selam arkadaşlar...
Nasılsınız?
```

In [8]:

```
selamla()
selamla()
selamla()
selamla()
```

```
Selam arkadaşlar...
Nasılsınız?
Selam arkadaşlar...
Nasılsınız?
Selam arkadaşlar...
Nasılsınız?
Selam arkadaşlar...
Nasılsınız?
```

Parametreler ve Argümanlar

In [13]:

```
def selamla(isim): # isim değişkenimiz burada parametre olmaktadır
    print("Merhaba:",isim)
```

In [16]:

```
selamla("Kemal") # "Kemal" değeri burada  argüman olmaktadır.
```

```
Merhaba: Kemal
```

In [17]:

```
selamla("Ayşe") # "Ayşe" değeri burada  argüman olmaktadır.
```

```
Merhaba: Ayşe
```

Bizim *fonksiyon tanımlarken* tanımladığımız herbir değişken birer **Parametre** , *fonksiyon çağrısı* yaptığımız zaman içine gönderdiğimiz değerler ise **Argüman** olmaktadır. Burada fonksiyonu çağırırken gönderdiğimiz "Kemal" değeri "isim" isimli parametreye eşit oluyor ve fonksiyonumuz bu değere göre işlem yapıyor. "Ayşe" değerini gönderdiğimizde ise fonksiyonumuz bu değere göre işlem yaparak ekrana farklı bir değer yazdırıyor.

In [18]:

```
# Toplama fonksiyonu
def toplama(a,b,c):
    print("Toplamları:",a+b+c)
```

In [19]:

```
toplama(3,4,5)
```

```
Toplamları: 12
```

In [20]:

```
toplama(10,11,29)
```

```
Toplamları: 50
```

In [22]:

```
toplama(4,9,40)
```

```
Toplamları: 53
```

Fonksiyonlarda Return

return ifadesi fonksiyonun işlemi bittikten sonra **çağrıldığı yere** değer döndürmesi anlamı taşır. Böylelikle, fonksiyonda aldığımız değeri bir değişkende depolayabilir ve değeri programın başka yerlerinde kullanabiliriz.

In [10]:

```
def toplama(a,b,c): # Birinci fonksiyon
    print("Toplamları",a+b+c)

def ikiyleçarp(a): # İkinci fonksiyon
    print("2 ile çarpılmış hali", a * 2)
```

In [11]:

```
toplam = toplama(3,4,5)

ikiyleçarp(toplam)
```

```
Toplamları 12

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-da4bfe0bd65f> in <module>()
      1 toplam = toplama(3,4,5)
      2
----> 3 ikiyleçarp(toplam)

<ipython-input-10-5128afde7d6b> in ikiyleçarp(a)
      3
      4 def ikiyleçarp(a): # İkinci fonksiyon
----> 5     print("2 ile çarpılmış hali", a * 2)
      6

TypeError: unsupported operand type(s) for *: 'NoneType' and 'int'
```

Burada hata almamızın sebebi fonksiyonları herhangi bir değer döndürmemesi yani **return** kullanmamasıdır. İsterseniz **toplam** değişkeninin tipine bakalım.

In [12]:

```
type(toplam)
```

Out[12]:

```
NoneType
```

In [13]:

```
def toplama(a,b,c):
    return a+b+c # return'un kullanımı
def ikiyle_çarp(a):
    return a*2
```

In [14]:

```
toplam = toplama(3,4,5)
print(ikiyle_çarp(toplam))
```

```
24
```

In [32]:

```
def üçleçarp(a):
    print("1.fonksiyon çalıştı")
    return a*3
```

In [33]:

```
def ikiyletopla(a):
    print("2.fonksiyon çalıştı")
    return a + 2
```

In [34]:

```
def dördeböl(a):
    print("3.fonksiyon çalıştı")
    return a / 4
```

In [35]:

```
# 3 ünü beraber kullanalım.

print(dördeböl(ikiyletopla(üçleçarp(5))))
```

```
1.fonksiyon çalıştı
2.fonksiyon çalıştı
3.fonksiyon çalıştı
4.25
```

return ifadesinden sonra fonksiyonumuz tamamıyla sona erer. Yani, **return** ifadesinden sonra yapılan herhangi bir işlem çalıştırılmaz.

In [38]:

```
def toplama(a,b,c):
    return a + b + c
    print("Toplama fonksiyonu") # Çalıştırılmadı.
```

In [39]:

```
print(toplama(1,2,3))
```

```
6
```

In [40]:

```
def toplama(a,b,c):
    print("Toplama fonksiyonu") # Çalıştırıldı.
    return a + b + c
```

In [41]:

```
toplama(1,2,3)
```

```
Toplama fonksiyonu
```

Out[41]:

```
6
```

Fonksiyonlarda Parametre Türleri

Parametrelerin Varsayılan Değerleri

In [3]:

```
def selamla(isim):
    print("Selam",isim)
```

In [4]:

```
selamla("Murat")
```

```
Selam Murat
```

In [5]:

```
selamla("Serhat")
```

```
Selam Serhat
```

In [6]:

```
selamla() # Böyle bir kullanım hata verecektir.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-8fe4760a9706> in <module>()
----> 1 selamla() # Böyle bir kullanım hata verecektir.

TypeError: selamla() missing 1 required positional argument: 'isim'
```

Ancak biz eğer bir parametrenin değerini varsayılan olarak belirlemek istersek, bunu varsayılan değerler ile yapabiliriz. Varsayılan değerleri anlamak için **selamla** fonksiyonunu varsayılan parametre değeri ile yazalım.

In [7]:

```
def selamla(isim = "İsimsiz"):
    print("Selam",isim)
```

In [9]:

```
selamla() # Hiç bir değer göndermedik. "isim" parametresinin değeri varsayılan olarak "İsimsiz" olarak
```

```
Selam İsimsiz
```

In [10]:

```
selamla("Serhat") # Değer verirsek varsayılan değerin yerine bizim verdiğimiz değer geçer.
```

```
Selam Serhat
```

In [12]:

```
def bilgilerigöster(ad = "Bilgi Yok",soyad = "Bilgi Yok",numara =  "Bilgi Yok"):
    print("Ad:",ad,"Soyad:",soyad,"Numara:",numara)
```

In [13]:

```
bilgilerigöster() # Bütün parametreler varsayılan değerle ekrana basılacak.
```

```
Ad: Bilgi Yok Soyad: Bilgi Yok Numara: Bilgi Yok
```

Esnek Sayıda Değerler

In [23]:

```
def toplama(a,b,c):
    print(a+b+c)
```

In [24]:

```
toplama(3,4,5)
```

```
12
```

In [26]:

```
toplama(3,4,5,6) # 4 tane argüman veremeyiz.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-aeb23284c8f3> in <module>()
----> 1 toplama(3,4,5,6) # 4 tane argüman veremeyiz.

TypeError: toplama() takes 3 positional arguments but 4 were given
```

Eğer bu fonksiyonu 4 argüman alacak şekilde tanımlamak istersek, tekrardan tanımlamamız gerekiyor.

In [28]:

```
def toplama(a,b,c,d):
    print(a+b+c+d)
```

In [29]:

```
toplama(3,4,5,6)
```

```
18
```

In [30]:

```
toplama(3,4,5) # Ancak bu sefer de 3 argüman veremiyoruz.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-35c9fb4ed348> in <module>()
----> 1 toplama(3,4,5) # Ancak bu seferde 3 argüman veremiyoruz.

TypeError: toplama() missing 1 required positional argument: 'd'
```

In [39]:

```
def toplama(*parametreler): # Artık parametreler değişkenini bir demet gibi kullanabilirim.
    toplam =  0
    print("Parametreler:",parametreler)
    for i in parametreler:
        toplam += i
    return toplam
```

In [40]:

```
print(toplama(3,4,5,6,7,8,9,10))
```

```
Parametreler: (3, 4, 5, 6, 7, 8, 9, 10)
52
```

In [41]:

```
print(toplama())
```

```
Parametreler: ()
0
```

In [38]:

```
print(toplama(1,2,3))
```

```
Parametreler: (1, 2, 3)
6
```

print fonksiyonunu tekrar hatırlayacak olursak aslında **print** fonksiyonu bu şekilde tanımlanmış bir fonksiyondur. Çünkü biz print fonksiyonuna istediğimiz sayıda argüman gönderebiliyorduk.

In [42]:

```
print(3,4,5,6)
```

```
3 4 5 6
```

In [43]:

```
print("Elma","Armut")
```

```
Elma Armut
```