break komutu: Döngüyü sonlandıran bir komuttur.

Ör:

```
while True:
    print("Bilgisayar çıldırdı!")
```

Çıktı: **Sonsuz döngü…**

Yukarıdaki kod aksi belirtilmediği sürece sürekli çalışır. Yani ekrana sürekli "Bilgisayar çıldırdı!" yazısı gelir. Ancak;

```
while True:
    print("Bilgisayar çıldırdı!")
    break
```

Çıktı: **Bilgisayar çıldırdı!**

break komutunu ilave edersek programa dur komutu vermiş oluruz. Böylece sonsuz döngüden kurtulmuş oluruz. Ekrana sadece bir kere "Bilgisayar çıldırdı!" yazısı çıkacak ve öyle kalacaktır.

Ör: Aşağıdaki program ilgili toplama işlemini sürekli yapacaktır.

```
while True:
   a=int(input("sayı gir: "))
   b=int(input("sayı gir: "))
   print("Sayıların toplamı:", a+b)
```

Ör: Kullanıcının belirli bir şey yazana kadar programın sürekli çalışması

```
while True:
   soru = input("Nasılsınız, iyi misiniz?")

   if soru == "yeter artık":
      print("Tamam kızma sormuyorum artık! ")
      break
```

Ör: Kullanıcıya bir parola belirletirken parolanın 6 karakterden kısa olmamasını sağlayalım.

```
while True:
    parola = input("Bir parola belirleyin: ")

    if not parola:
        print("Parola bölümü boş geçilemez!")

    elif len(parola) < 6:
        print("Parola 6 karakterden kısa olmamalı")

    else:
        print("Yeni parolanız: ", parola)
        break
```

continue komutu: continue komutunun görevi kendisinden sonra gelen her şeyin es geçilip döngünün başa dönmesini sağlamaktır. Break komutu kadar sık kullanılmaz. continue riskli bir komuttur. Bazen programımızın bir bölümünde sonsuz döngüye girmesine neden olabilir.

Ör:

```
a=0
while a<10:
     if a==3:
         continue
     a+=1
     print(a, end=” ”)
```

Çıktı: 1 2 3

pass komutu: Herhangi bir işlem yapmadan geçeceğimiz durumlarda kullanılır. Kısaca “Hiçbir şey yapmadan yola devam et!” anlamı taşır. Daha iyi anlamak için yukarıdaki örneği aynen yazalım. Tek değişiklik continue yerine pass yazmak olsun.

```
a=0
while a<10:
     if a==3:
         pass
     a+=1
     print(a, end=” ”)
```

Çıktı: 1 2 3 4 5 6 7 8 9 10

HATA ÇEŞİTLERİ:

1- Söz Dizimi Hataları (SyntaxError) :

Ör1: Hatalı atama işlemi:

```
y = 5
x + 2 = y
print(x)
```

Doğrusu:

```
y = 5
x=y-2
print(x)
```

Çıktı: 3

Ör2: Eşleşmeyen parantez:

```
print(5+(3*4)    #hata
print(5+(3*4))   #doğrusu
```

Ör3: Eşleşmeyen tırnak işareti:

```
print(‘hello”)     #hata
print(“hello”)     #doğrusu
print(‘hello’)     #doğrusu
```

Ör4: Hatalı girinti; if, while, for gibi döngü komutlarından sonra yazılan kodlar biraz içerden yazılır.

```
a=4
b=6
if a<b:
print("a, b'den küçüktür")
```

Doğrusu:

```
a=4
b=6
if a<b:
   print("a, b'den küçüktür")
```

2- Çalışma Zamanı Hataları:

Ör1: Atanmayan değişken

```
x=5
print(x+y)              #y diye bir değişken yok, yani atanmamış.
```

Doğrusu:

```
x=5
y=8
print(x+y)
```

Çıktı: 13

Ör2: Sıfıra bölme işlemi

```
32/0
```

Ör3: Sayıları string'e bölmek

```
print(5/"a")
```

Ör4: Hiç gerçekleşmeyecek koşullar

```
if 2>4:
        print("Merhaba Uzaylı")
```

3- Mantık Hataları (Anlam Bilimsel Hatalar) : Program genellikle hata mesajı vermeden çalışır ancak istenilen işlemi gerçekleştirmez. En zor hata ayıklama türüdür.

Demetler (Tuplelar) : Demetler listelere benzer ancak farkları demetlerin değiştirilemez oluşudur.

```
#Demet oluşturma:
demet = (1,2,3,4,5,6,7,8,9)
print(demet)
Çıktı: (1, 2, 3, 4, 5, 6, 7, 8, 9)


#Tek elemanlı bir demet tanımlama:
demet = (1,)
print(demet)
Çıktı: (1,)

#0. indekse ulaşma:
demet = (1,2,3,4,5,6,7,8,9)
print(demet[0])
Çıktı: 1

#Elemanların indexini yani kaçıncı sırada olduğunu bulma:
demet = (1,2,3,”python”,”kodlama”,”Ali”)
print(demet.index("python"))
Çıktı: 3

#Elemanların demette kaç defa geçtiğini bulma:
demet = (1,23,34,34,2,1,4,5,1,1,34)
print(demet.count(34))
Çıktı: 3

#Demetler değiştirilemez:
demet = ("Elma","Armut","Muz")
demet[0] = "Kiraz"
Çıktı: Hata

#Demetler değiştirilemez:
demet = ("Elma","Armut","Muz")
demet.remove = "Kiraz"   #Kiraz değerini silmeye çalıştık hata aldık
Çıktı: Hata
```

Sözlükler (Dictionary): Aynı gerçek hayattaki sözlükler gibi davranan bir veri tipidir. Sözlüğün içindeki her bir eleman indeks olarak değil, anahtar (key) ve değer (value) olarak tutulur. Örneğin elimize İngilizce-Türkçe bir sözlük alıp book (key) kelimesini aradığımızda karşılık olarak kitap (value) kelimesini görürüz. Sözlükleri de bu şekilde düşünebiliriz.

Sözlük oluşturma:

sözlük1 = {"book":”kitap”,"apple":”elma”,"printer":”yazıcı”}

Anahtara karşılık gelen değerleri bulma:

print(sözlük1["book"])

Çıktı: kitap

Sözlüğe eleman ekleme:

sözlük1 [“table”] = ”masa”

print (sözlük1)

Çıktı: {'book': 'kitap', 'apple': 'elma', 'printer': 'yazıcı', 'table': 'masa'}

Boş bir sözlük oluşturma:

sözlük2 = {}

Boş bir sözlük oluşturma; dict() fonksiyonu ile:

sözlük2 = dict ()

Anahtara karşılık gelen değerleri bulma, listeler örneği:

```
liste = { "a" : [ 1, 2, 3 ] , "b": [ [ 4, 5 ] , [ 6, 7 ] ] , "c" : 8 }

print( liste ["b"] [1] [1] )      # Çıktı: 7

liste [“c”] + = 5

print ( liste ["c"]  )            # Çıktı: 13
```

İç içe Sözlükler:

```
a = {"sayılar":{"bir":1,"iki":2,"üç":3},"meyveler":{"kiraz":"yaz","portakal":"k ış","erik":"yaz"}}

print ( a [ "meyveler"] [ "kiraz"] )
```

Çıktı: yaz

Değerleri listeye dönüştürme:

```
sözlük = { "bir": 1, "iki": 2, "üç": 3 }

print( sözlük.values () )
```

Çıktı: dict_values ([1, 2, 3])

Anahtarları listeye dönüştürme:

```
print( sözlük.keys () )
```

Çıktı: dict_keys (['bir', 'iki', 'üç'])

Anahtar ve değerlerini bir liste içinde demete dönüştürme:

```
print( sözlük.items () )
```

Çıktı: dict_items ([('bir', 1), ('iki', 2), ('üç', 3)])

FONKSİYONLAR:

Fonksiyonlar karmaşık işlemleri tek adımda yapar.

- Fonksiyonlar bir modül içinde bulunur.
- Modüller ise bir kütüphane içinde bulunur.
- Python standart kütüphanesinde yaklaşık 230 modül vardır.

Python'da kabaca iki tip fonksiyon bulunur.

1- **Gömülü fonksiyonlar (builtin functions)**
2- **Özel fonksiyonlardır (custom functions)**

Gömülü fonksiyonlar:

print(), type(), str(), int() gibi fonksiyonlardır. Gömülü fonksiyonlar **__builtins__** isimli modülde yer almaktadır. Gömülü fonksiyonları görmek için **dir(__builtins__)** komutunu yazmanız yeterlidir. Bu fonksiyonlar hazır tanımlandığı için herhangi bir işlem yapmadan direk kullanabiliriz.

Özel fonksiyonlar:

Özel fonksiyonlar bizim tarafımızdan tanımlanmıştır. Bu yüzden bu fonksiyonları kullanabilmemiz için import komutunu kullanmak şarttır. Örneğin gömülü olmayan math modülünü kullanmak istiyorsak import komutunu yazmamız gerekecektir. O halde math modülünde neler var bir bakalım.

```
import math
dir(math)
```

Ör: Karekök Fonksiyonu

```
import math
sayi = float(input("Sayı Giriniz: "))
kok = math.sqrt(sayi)
print(sayi," sayısının karekökü" "=", kok)
```

Ör: İç içe karekök Fonksiyonu

```
import math
y = math.sqrt(math.sqrt(256))
print(y)
```

Çıktı: **4**

Fonksiyon Tanımlamak ve Çağırmak: Bir kayıt işlemi yapalım.

```
isim    = "Ali"
soyisim = "Yılmaz"
d_yeri  = "İzmir"
yas     = "28"

print("isim           : ", isim)
print("soyisim        : ", soyisim)
print("doğum yeri     : ", d_yeri)
print("yaş            : ", yas)
```

Bu programı çalıştırdığımızda şöyle bir çıktı alırız:

```
isim           :  Ali
soyisim        :  Yılmaz
doğum yeri     :  İzmir
yaş            :  28
```

Peki başka bir kişi için de kayıt oluşturmak isteseydik ne yapacaktık? Programa her yeni kayıt eklenişinde benzer satırları tekrar tekrar yazmak zorunda kalacaktık. Oysa bundan kurtulmak mümkündür. Örnek olarak kayıt_oluştur() adlı bir fonksiyon tanımlayalım.

```
def kayıt_oluştur(isim, soyisim, d_yeri, yas):

    print("isim           : ", isim)
    print("soyisim        : ", soyisim)
    print("doğum yeri     : ", d_yeri)
    print("yaş            : ", yas)
```

Artık elimizde bizim oluşturduğumuz kayıt_oluştur() adlı bir fonksiyon var. Dolayısıyla bu yeni fonksiyonumuzu daha önceki fonksiyonları nasıl kullanıyorsak aynı şekilde kullanabiliriz. Yukarıdaki kodların devamında aşağıdaki gibi komutlar yazalım:

```
kayıt_oluştur("Berk", "Kılıç", "Edirne", "25")
kayıt_oluştur("Tarkan", "Öztürk", "Giresun", "41")
```

Çıktımız aşağıdaki gibi olacaktır.

```
isim            :  Berk
soyisim         :  Kılıç
doğum yeri      :  Edirne
yaşı            :  25

isim            :  Tarkan
soyisim         :  Öztürk
doğum yeri      :  Giresun
yaşı            :  41
```

Ör: Kare bulma fonksiyonu

```
def kare_bul(i):
    print(i**2)
kare_bul(2)
```

Çıktı: 4

Ör: Sayıların 10 katını alan fonksiyon

```
def onkat(n):
   return n*10
x = onkat(3)
print(x)
```

Çıktı: 30

Ör: Karenin alanını hesaplayan fonksiyon

```
def kare(n):

     print(n*n)

kare(3)
```

Çıktı: 9

Ör: 1'den 10'a kadar yazdıran fonksiyon

```
def say():
     for i in range(1, 11):
          print(i, end=" ")
say()
```

Çıktı: `1 2 3 4 5 6 7 8 9 10`

time fonksiyonları: Zamanla ilgili işlemlerin yer aldığı modüldür. Bu modüle ait **clock** ve **sleep** fonksiyonları bulunur.

clock(): Programın başladığı andan itibaren geçen süreyi saniye olarak hesaplar.

Ör: Bilginin kaç saniyede girildiğini hesaplayan program:

```
import time
baslangicZamani = time.clock()
ad=input("İsminizi Giriniz: ")
zaman = time.clock() – baslangicZamani
print("isminizi tam", zaman, "saniyede girdiniz")
```

Çıktı:

```
İsminizi Giriniz:  Onur
Sevgili Onur isminizi tam 13.000000000001e-06 saniyede girdiniz
```

sleep(): Programın çalışması sırasında belirtilen süre kadar durmasını sağlar.

Ör: Bilgi girilmeden önce 5 saniye bekleten program:

```
import time
time.sleep(5)
ad=input("İsminizi Giriniz: ")
```

Ör: Belli aralıklarda rastgele bir tamsayı üretme

```
import random
for i in range(1):          #Rastgele 1 adet sayı üretilecek
   print(random.randrange(1, 100)
```

Çıktı: 56

Ör: Sayısal loto Programı:

```
import random
for i in range(6):          #Rastgele 6 adet sayı üretilecek
   print(random.randrange(1, 50), end=" ")
```

Çıktı: 35, 41, 3, 45, 12, 8

sample() fonksiyonu: random modülü içinde herhangi bir dizi içinden istediğimiz sayıda rastgele numune almamızı sağlar.

Ör:

```
import random
liste = ["ahmet", "mehmet", "sevgi", "sevim", "selin", "zeynep"]
print(random.sample(liste, 2))
```

Çıktı: ['sevgi', 'zeynep']