Fonksiyonlar

Fonkisyonlar, belirli bir görevi yerine getiren ifadelerdir. Programı organize bir hale getirir ve tekrarı önlerler.

Python'da iki tip fonksiyon bulunur: gömülü fonksiyonlar(built-in functions) ve kullanıcı tarafından tanımlanan fonksiyonlar(user-defined functions).

1.1.Fonksiyonların Tanımlanması ve Çağırılması

Fonksiyon tanımının yapısı şu şekildedir:

```
    def fonksiyon_adi(parametreler):
        #ifadeler
        #dönüş döğeri(opsiyonel)
```

def : fonksiyon başlığının başlangıcını belirten anahtar kelimedir.
parametreler : fonksiyona gönderdiğimiz değerlerdir.
iki nokta(:) : fonksiyon başlığının sonunu belirtir.
ifadeler : fonksiyonun gövdesini oluştururlar ve aynı girinti seviyesinde bulunurlar.
dönüş değeri : fonksiyondan bir değer döndürür, isteğe bağlıdır.

```
#Parametresiz fonksiyon örneği
```

```
<<< def selam_soyle():
        print("Selam! Nasılsın?")

<<< selam_soyle()
>>> Selam! Nasılsın?
```

```
#Parametreli fonksiyon örneği
```

```
<<< def selam_soyle(isim):
        print("Selam",isim)

<<< selam_soyle("Ahmet")
>>> Selam Ahmet
```

1.2.Fonksiyonlarda Return İfadesi

Return ifadesi fonksiyondan çıkmak ve çağrıldığı yere değer göndermek için kullanılır.

#Örnek

```
<<< def sayilari_carp(a,b,c):
            carpim = a*b*c
            return carpim

<<< def mutlak_deger(a):
            if a>=0:
                Return a
            else:
                return -a

<<< carpim = sayilari_carp(3,-4,5)
    print("Çarpımın mutlak değeri:",mutlak_deger(carpim))
>>> Çarpımın mutlak değeri: 60
```

Return ifadesinden sonra fonksiyon sona erer, return'den sonraki ifadeler okunmaz.

#Örnek

```
<<< def sayinin_karesi(a):
        karesi = a*a
        return karesi
        print("Sayıların karesi:",karesi)

<<< sayinin_karesi(5)
>>> 25
```

1.3. Fonksiyonlarda Parametreler

1.3.1. Varsayılan Parametre Değerleri(Default Parameter Value)

```
<<< def selam_soyle(isim):
        print("Selam " + isim)

<<< selam_soyle()
>>> Error
```

```
# Fonksiyonun bu hatayı döndürmesinin sebebi, isim parametresine bir değer gönderilmemesidir.
# Bu hatayı engellemek için parametrelere varsayılan değer atanmaldır.
```

```
<<< def selam_soyle(isim = "anonim"):
        print("Selam " + isim)

<<< selam_soyle()
>>> Selam anonim
```

```
#Örnek
<<< def kullanicinin_bilgileri(isim = "Bilgi Yok" , soyisim = "Bilgi Yok"  , dogum_tarihi = "Bilgi Yok"):
        print("Kullanıcı ismi:", isim, "\nSoyismi:", soyisim, "\nDoğum Tarihi:", dogum_tarihi)

<<< kullanicinin_bilgileri()
>>> Kullanıcı ismi: Bilgi Yok
    Soyismi: Bilgi Yok
    Doğum Tarihi: Bilgi Yok

<<< kullanicinin_bilgileri("Ahmet","Ahmetoğlu",1950)
>>> Kullanıcı ismi: Ahmet
    Soyismi: Ahmetoğlu
    Doğum Tarihi: 1950

<<< kullanicinin_bilgileri(isim = "Mehmet" , dogum_tarihi = 1970)
>>> Kullanıcı ismi: Mehmet
    Soyismi: Bilgi Yok
    Doğum Tarihi: 1970
```

```
#Örnek    (print fonksiyonunun parametreleri)
<<< print("Çatal", "Kaşık", "Bıçak")
    print("Kase", "Tabak", "Bardak")
>>> Çatal Kaşık Bıçak
    Kase Tabak Bardak

<<< print("Çatal", "Kaşık", "Bıçak", sep = '/', end = '-')
    print("Kase", "Tabak", "Bardak", sep = '&')
>>> Çatal/Kaşık/Bıçak-Kase&Tabak&Bardak
```

1.3.2.İsteğe Bağlı Sayıdaki Parametreler (Arbitrary Arguments)

Fonksiyona kaç argüman verileceği bilinmiyorsa, fonksiyon tanımında parametre isminden önce * koyulur. Böylelikle verilen argümanlar bir demet gibi kullanılabilir.

```
#Örnek
<<< def toplam(*a):
        toplamlar = 0
        For i in a:
            Toplamlar += i
        Return toplamlar

<<< print(toplam(5,6,7,8,9,10))
>>> 45
```

1.4.Fonksiyonlarda Global/Yerel Değişkenler

```
# Python'da her bir değişkenin Python tarafından tanımlanan bir kapsamı vardır. Bu kapsamlar, değişkenlerin nerede var olduğunu ve nerelerde kullanılabileceğini gösterir.
# Python'da en genel kapsama sahip değişkenler global değişkenler olarak tanımlanırlar ve global değişkenlere tanımlandığı andan itibaren programın her yerinden ulaşabiliriz.
# Python'da fonksiyonlarda tanımlanan değişkenler Python tarafından yerel değişkenler olarak tanımlanırlar. Bu değişkenler fonksiyona özgüdür, fonksiyon çalışmasını bitirdikten sonra bellekten silinip yok olurlar. Bu yüzden fonksiyon içinde tanımlanmış bir değişkene başka bir yerden erişilemez.
```

```
#Örnek
<<< def my_fonc2():
        m = 10
        return m

<<< print(my_fonc2)
    print(m)
>>> 10
    Error

# m harfi fonksiyonun içinde tanımlanan bir değişken olduğu için ikinci print fonksiyonu hata verdi.
```

```
#Örnek
<<< k = 7

<<< def my_fonc():
        k = 10
        return k

<<< print("fonksiyonun içinde:", my_fonc())
    print("fonksiyonun dışında:", k)
>>> fonksiyonun içinde: 10
    Fonksiyonun dışında: 7
```

1.4.1. Global ifadesi

```
<<< k = 7

    def my_fonc():
        global k
        k = 10
        return k

<<< print("fonksiyonun içinde:", my_fonc())
    print("fonksiyonun dışında:", k)
>>> fonksiyonun içinde: 10
    Fonksiyonun dışında: 10

# global ifadesini kullanarak, fonksiyonda yeni bir k değişkeni oluşturmak yerine, globaldeki k değişkeninin üstünde işlem yapıyoruz.
```

1.5.Lambda İfadeleri

Lambda fonksiyonları küçük fonksiyonlardır. Sayısız argüman alabilirler ama sadece bir işlem gerçekleştirirler.
Tanımlanması şu şekildedir:

```
    lambda argümanlar : işlem/ifade
```

#Örnek

```
<<< toplam = lambda a,b,c,d : a+b+c+d

<<< print(toplam(2,4,5,6))
>>> 17
```

#Örnek

```
<<< esitMi = lambda x,y : x==y

<<< esitMi(4,5)
>>> False
```