Fonksiyonlar

Bu derste fonksiyonların ne olduğunu , bir fonksiyonun nasıl tanımlanacağını ve nasıl kullanılacağını öğrenmeye çalışacağız.

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.

İsterseniz şimdi de fonksiyonların ne olduğunu gerçek hayattan benzetme yaparak anlamaya çalışalım. Örneğin evimize bir adet **katı meyve sıkacağı** alıyoruz ve canımız ne zaman meyve suyu isterse bu aleti kullanıyoruz. Yani aslında bu aletin görevi ve fonksiyonu **meyve suyu** hazırlamaktır.

Python geliştiricilerin yazdığı fonksiyonlara yani bizim hazır kullandığımız fonksiyonlara(print(),type() vs.) gömülü fonksiyonlar(built-in function) denilmektedir.Ancak bunlardan hariç olarak biz kendi özel fonksiyonlarımızı(user-defined functions) da tanımlayabiliriz.

Peki biz kendi fonksiyonlarımızı nasıl tanımlayacağız ? İsterseniz şimdi yavaştan fonksiyonların nasıl tanımlanacağını öğrenelim.

Fonksiyonların Tanımlanması

Fonksiyon tanımlamanın yapısı şu şekildedir;

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

İsterseniz şimdi bir tane "selamla" isimli bir fonksiyon tanımlayalım.

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
```

Fonksiyonumuzu tanımladık ve Python bunu bir fonksiyon olarak algıladı ? Ancak tıpkı katı meyve sıkacağını alıp kullanmazsak hiçbir işe yaramadığı için , bu fonksiyonu da tanımlayıp kullanmazsak hiçbir işe yaramayacaktır. O halde şimdi de fonksiyonların kullanılmasını öğrenelim.

Fonksiyonların Kullanılması veya Çağrılması (Function Call)

Tanımlanan bir fonksiyonun kullanılmasına programlama dillerinde *Fonksiyon Çağrısı* denmektedir. O halde **selamla** fonksiyonumuzu nasıl çağıracağımızı öğrenelim. Fonksiyon çağrısı şu şekilde yapılabilmektedir;

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

İsterseniz şimdi **selamla** fonksiyonumuzu çağıralım.

In [6]:
```
type(selamla) # Tanımlı
```

Out[6]:
```
function
```

In [7]:
```
selamla() # Fonksiyon parametre almadığında içine argümanlarımızı göndermiyoruz.
```

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

Burada gördüğünüz gibi, fonksiyonumuz çağrıldığı zaman, kendi bloğundaki işleri yaptı ve ekrana 2 tane değer yazdırdı. Bu fonksiyonu istediğimiz yerde tekrar tekrar çağırabiliriz.

In [8]:
```
selamla()
selamla()
selamla()
selamla()
```

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

selamla fonksiyonumuzu 4 defa çağırdık ve fonksiyonumuz işlevini 4 defa yerine getirdi. Peki fonksiyonumuzun içine bir tane değer verseydik ne olurdu ?

In [10]:
```
selamla("python") # Hata verdi çünkü fonksiyonumuz hiçbir değer almıyor.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-97be52311f74> in <module>()
----> 1 selamla("python") # Hata verdi çünkü fonksiyonumuz hiçbir değer almıyor.

TypeError: selamla() takes 0 positional arguments but 1 was given
```

Parametreler ve Argümanlar

Biliyorsunuz biz selamla fonksiyonunun içine herhangi bir değer göndermiyorduk ve fonksiyonumuz hep aynı işi yapıyordu. Ancak çoğu zaman fonksiyonlarımız içine gönderdiğimiz değerlerle farklı işlemler yaparlar. Örneğin **katı meyve sıkacağına** eğer "Elma" verirsek elma suyu, "Nar" verirsek nar suyu hazırlayacaktır. Fonksiyonlarda da **Parametreleri** bu şekilde düşünebilirsiniz. İsterseniz şimdi **selamlama** fonksiyonumuzu bir tane parametre alacak şekilde tanımlayalım.

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. Şimdi isterseniz farklı bir fonksiyon tanımlayalım ve 3 tane parametre alsın.

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
```

Şimdi de örnek olması açısından bir sayının faktoriyelini hesaplayan bir fonksiyon yazalım.

Eğer sayımız "5" ise faktoriyel 5 x 4 x 3 x 2 x 1 = 120 olacaktır

In [28]:
```
def faktoriyel(sayı):
    faktoriyel = 1
    if (sayı == 0 or sayı == 1):
        print("Faktoriyel",faktoriyel)
    else:
        while (sayı >= 1):
            faktoriyel *= sayı
            sayı -=1
        print("Faktoriyel", faktoriyel)
```

In [29]:
```
faktoriyel(5)
```

```
Faktoriyel 120
```

In [30]:
```
faktoriyel(6)
```

```
Faktoriyel 720
```

In [31]:
```
faktoriyel(1)
```

```
Faktoriyel 1
```

In [32]:
```
faktoriyel(0)
```

```
Faktoriyel 1
```

İşte bu kadar ! Bir sonraki dersimizde **fonksiyonlarda return** konusunu işleyeceğiz.

In [ ]:
```

```