PYTHON PROGRAMLAMA DİLİ

FONKSİYONLAR

Dr. Öğr. Üyesi M. Ozan AKI

FONKSİYONLAR - TANIMLAMA

Bir program içinde sıkça kullanılan bir kod bloğunu fonksiyon olarak tanımlamak ve ihtiyaç duyulduğu yerlerde bu fonksiyonu çağırmak daha basit, anlaşılabilir ve temiz bir kodlama yöntemidir.

```
def fonksiyon_adı:
    fonksiyon kodları
    fonksiyon kodları
    ...
    return
```

FONKSİYONLAR - PARAMETRELER

Fonksiyonlar gerektiğinde bir ya da daha çok parametre alarak, bu parametrelere göre işlemi yapar, ve geriye bir sonuç döndürürler.

Fonksiyonun parametrelerine dışarıdan gönderilen değere (değişken yada sabit bir değer olabilir) argüman denir.

```
def fonksiyon_adı(a,b):
    fonksiyon kodları
    fonksiyon kodları
    ...
    return r
```

```
f=fonksyion_adı(3,5)
```

FONKSİYONLAR - PARAMETRELER

Fonksiyon çağrılarında, parametreleri sırasıyla yazmamız gerekir.

Eğer fonksiyon parametreleri (a,b) sırasıyla tanımlanmışsa, ilk yazılan parametre a, ikinci yazılan parametre b olur.

```
def fonksiyon_adı(a,b):
    fonksiyon kodları
    fonksiyon kodları
    ...
    return r
```

Ancak fonksiyon çağırılırken, parametre adı belirtilerek sırasından bağımsız olarak parametreye değer geçilebilir.

```
f=fonksyion_adı(3,5)

f=fonksyion_adı(a=3,b=5)

f=fonksyion_adı(b=5,a=3)
```

FONKSİYONLAR - PARAMETRELER

Eğer fonksiyonun bazı parametreleri kullanıcı tarafından girilmediğinde varsayılan bir değer atanmak istenirse, parametre listesindeki değişkene atama operatörü ile varsayılan bir değer atanabilir.

Bu durumda, girilmeyen parametre yerine varsayılan değer geçerli olacaktır.

Ancak parametre girilirse, girilen değer geçerli olur.

```
def fonksiyon_adı(a,b,c=0):
    fonksiyon kodları
    fonksiyon kodları
    ...
    return r
```

```
x=fonksyion_adı(3,5)
```

```
y=fonksyion_adı(3,5,7)
```

FONKSİYONLAR - PARAMETRELER

Eğer bir fonksiyona değişken sayıda bir parametre geçilmek istenirse, parametre adının soluna * karakteri eklenir.

Bu şekilde değişken sayıdaki tüm parametreler bir liste olarak fonksiyona geçilir ve parametrelere tek tek liste üzerinden erişilir.

```
def fonksiyon_adı(*params):
    fonksiyon kodları
    fonksiyon kodları
    ...
    return r
```

```
x=fonksyion_adı(3,5)

y=fonksyion_adı(3,5,7)

z=fonksyion_adı(3,5,7,9,11,13)
```

FONKSİYONLAR - PARAMETRELER

Eğer bir fonksiyona **dictionary** tipinde bir parametre geçilmek istenirse, parametre sının soluna ** karakteri eklenir.

Bu şekilde her bir parametreye, **dictionary** içinde tanımlanan (**key**) üzerinden erişilebilir.

```
def fonksiyon_adı(**args):
    fonksiyon kodları
    fonksiyon kodları
    ...
    return r
```

```
x=fonksyion_adı(x=12, y=34)
```

```
y=fonksyion_adı(a=12, b=34, c=56)
```

```
z=fonksyion_adı(ad='Ali', yas=22)
```

FONKSİYONLAR - PARAMETRELER

Eğer hem tek değişken hem liste, hem **dictionary** şeklinde parametre tanımlanırsa;

Soldan sağa doğru önce sabit parametreler atanır.

Kalan diğer parametreler liste elemanı olarak atanır.

key=value çifti olarak verilen parametreler ise **dictionary** olarak atanır.

```
def fonksiyon_adı(a,b,*args, **kwargs):
    fonksiyon kodları
    fonksiyon kodları
    ...
    return r
```

```
x=fonksyion_adı(1,2,3,4,ad='Ali', yas=22)
a: 1
b: 2
*args: (3,4)
**kwargs: {'ad':'ali', 'yas':22}
```

FONKSİYONLAR - AÇIKLAMA

Def fonksiyon tanımı satırından hemen sonra bir string ifade yazılırsa, bu ifade fonksiyonun açıklama satırı olarak Kabul edilir.

Fonksiyonu kullanmak istediğimiz erlerde, fonksiyon ismini yazdığımızda bu açıklama satırı yardımcı bilgi olarak görüntülenir.

```
def fonksiyon_adı(x,y):
    "Fonksiyon için açıklama satırı"
    fonksiyon kodları
    fonksiyon kodları
    ...
    return z
```

FONKSİYONLAR - RETURN

Bir fonksiyon kod bloğu içinde **return** çalıştığı anda fonksiyon sonlanır ve çağırıldığı noktaya geri döner.

return yanında bir değer varsa bu değer bir dönüş değeri olarak döndürülür.

return yanında bir değer yoksa herhangi değer döndürmeden fonksiyon sonlanır.

return ifadesi hiç yoksa, fonksiyon altındaki tüm kod bloğunun sonuna gelindiğinde fonksiyon sonlanır.

fonksiyon içinde birden fazla **return** ifadesi bulunabilir. İlk hangisi çalışırsa fonksiyon sonlanır. Değer belirtildiyse o değer döndürülür.

```
def fonksiyon_adı(x,y):
    "Fonksiyon için açıklama satırı"
    fonksiyon kodları
    fonksiyon kodları
    ...
    return z
```

FONKSİYONLAR - LAMBDA

Lambda fonksiyonlar, isimsiz ve satır içi tanımlanabilen küçük ve basit bir işlevi olan fonksiyonlardır. Diğer adıyla anonim fonksiyon adıyla da anılır.

lambda arg1,arg2,…argn**:**kodlar

Şeklinde tanımlanır. Lambda fonksiyonların ismi olmadığından herhangi yerden çağırılamaz.

Ancak **map**, **filter**, **reduce** ve benzeri fonksiyonlara parametre olarak girilir ya da bir değişkene atanarak kullanılırlar.

```
def topla(x,y):
    "x ve y sayılarını toplar"
    z=x+y
    return z
```

```
topla=lambda x,y:x+y
```

FONKSİYONLAR – GLOBAL/LOKAL DEĞİŞKENLER

Fonksiyon içinde bir değişken tanımlandığında bu değişken sadece fonksiyon kod bloğu içinde geçerlidir ve buna **lokal** değişken adı verilir.

Fonksiyonların dışında tanımlanan, tüm modül için geçerli ve erişilebilir değişkenlere **global** değişken denir.

Eğer fonksiyon içinde tanımlanan **lokal** değişken ile **global** değişken aynı isimde ise, **lokal** değişken önceliklidir.

Eğer **global** değişkene bir atama yapmak gerekirse, değişkenin adının önüne global ifadesi eklenerek **global** değişkene erişilir.

```
sayi=22                 # global degisken


def yazdir:
    global sayi=55   # global degisken
    sayi=55             # lokal degisken
    print(sayi)



print(sayi)
yazdir()
print(sayi)
```

FONKSİYONLAR – GLOBAL/LOKAL DEĞİŞKENLER

İç içe birden fazla fonksiyon tanımlanmışsa ve aynı isimde hem global hem lokal değişken mevcutsa, içteki fonksiyonun kapsamı bir üst fonksiyon olduğundan bir üst fonksiyondaki lokal değişken öncelikli olur.

```
sayi=22             # global degisken

def yazdir:
    sayi=55         # lokal degisken
    def arttir():
        sayi+=5
    arttir()

print(sayi)
yazdir()
print(sayi)
```

FONKSİYONLAR – RECURSIVE

Özyinelemeli (**Recursive**) fonksiyonlar, kendi kendini çağıran fonksiyonlardır. Fonksiyon her kendini çağırdığında bir kopyası oluşturulur. Son çağrı sonlandığında, tüm kopyalar dönüş yaparak sonlanır.

Bazı problemlerin çözümünde kullanılması kaçınılmazdır.

Fonksiyon içinde mutlaka bir koşula bağlı sonlandırma (**return**) olmalıdır. Aksi halde sonsuz döngüye girer ve hata verir.

```
def faktoriyel(x):
    if x==1:
        return 1
    else:
        return x*faktoriyel(x-1)
```

FONKSİYONLAR – FONKSİYON DÖNDÜRMEK

Bir fonksiyondan bir değer değil de başka bir fonksiyon döndürülebilir.

Bunun için fonksiyon içinde tanımlanan iç fonksiyonun ismi **return** ifadesinin yanında döndürülür.

Dönen değer, parantezler kullanılarak bir fonksiyon olarak çağırılabilir.

Bu çeşitli koşullara bağlı olarak dinamik olarak değişen bir fonksiyon yapısı elde edilebilir.

```
def kuvvet(k):
    def toplama(x):
        return x**2
    def kup(x):
        return x**3
    if k==2:
        return kare
    if k==3:
        return kup
```

```
fonk=kuvvet(2)
fonk(5) # 25
fonk=kuvvet(3)
fonk(5) # 125
```

FONKSİYONLAR – PARAMETRE FONKSİYON

Bir fonksiyon, başka bir fonksiyona parametre olarak gönderilebilir.

Parametre olarak gönderilen fonksiyon, çağırılan fonksiyonun içinde, isminin yanın parantez kullanılarak kullanılabilir.

```
def toplama(a,b):
    return a+b
def carpma(a,b):
    return a*b


def islemyap(x,y,f):
      sonuc=f(x,y)
      return sonuc


toplam=islemyap(3,5,toplama) # 8
carpim=islemyap(3,5,carpma)  # 15
```

FONKSİYONLAR – DECORATOR FONKSİYONLAR

Decorator fonksiyonlar, bir fonksiyona bir özellik eklenmek istendiği zaman kullanılır.

Özellik eklediğimiz **decorator** fonksiyonu, kullanmak istediğimiz bir çok fonksiyon ile ilişkilendirebiliriz.

Bunun için, bir **decorator** fonksiyon tanımlayıp, kullanmak istediğimiz fonksiyon tanımının üzerine **@** sembolü ile birlikte **decorator** fonksiyon adı yazılır.

```
def decorator_fonk(x):
    def wrapper():
        print("önceki işlemler")
        func()
        print("sonraki işlemler")
    return wrapper

@decorator_func
def merhaba():
    print("Merhaba")

# merhaba=decorator_func(merhaba)
# ile ayni islemi yapar
```