FONKSİYONLAR

Fonksiyonlar programlama dillerinde ne işe yarar?

- Bir kodu birden fazla kez yazmamak için kullanılır. Kod tekrarını önler.
- Karmaşıklaşan kodlarda hata yapma olasılığı arttığı için tek bir yerde bunu çözmek daha kolaydır.
- Birden fazla yerde aynı kod kullanımının olması okunurluğu azaltır.
- Bir problemi ayrı ayrı parçalara ayırıp çözmek işimizi kolaylaştırır.
- İş paylaşımında da kolaylık sağlar.
- Nesneye yönelik programlamada fonksiyonlar önemli bir yere sahiptir.

Fonksiyon Oluşturma

- def => burada fonksiyon için kullandığımız keyword
- deneme => fonksiyon ismi
- (parametre):
- a değerini yazdırmak istersek bir çıktı elde edemeyiz. Atama geçerli olur ama a kendi başına yazdırılamaz.

```
def deneme():
    """docstring fonksiyon için açıklama: Fonksiyonun ne işe yaradığını anlatır."""
    print("deneme")
    print("deneme")
    #return None var aslında!
```

```
deneme()
```

```
deneme
deneme
```

```
a = deneme()
```

```
deneme
deneme
```

```
a
```

NoneType

- return değeri olmadığı için a gibi bir değere fonksiyonu atarsak fonksiyonun tipini NoneType olarak görüntüleriz.
- Biz return için fonksiyona bir şey yazmasak bile return None değerini otomatik olarak alır.

```
type(a)
```

```
NoneType
```

Tanımladığımız topla() fonksiyonu:

- topla fonksiyonuna girilen her değere koddaki işlem yapılarak sonuç return ettirilir.
- topla fonksiyonu girilen string karakterleri de birleştirir.
- topla fonksiyonunu istediğimiz yerde kullanabiliyoruz.
- math sınıfından math.sqrt() ile karekök alma işlemi yapılarak çıkan değer ile girilen ikinci değer topla fonksiyonu ile toplanır. Sonuç return edilir.
- Parametre kısıtlaması yok.

```
def topla(x,y):
    z = x + y
    return z
```

```
topla(10,5)
```

```
15
```

```
topla("a","b")
```

```
'ab'
```

```
print("deneme", topla(5,6))
```

```
deneme 11
```

```
import math
topla(math.sqrt(16),4)
```

```
8.0
```

topla? kullanımı ne işe yarar?

- Bu şekilde kodumuzu çalıştırırsak topla fonksiyonunun imzasını görebiliriz.

```
topla?
```

topla?? kullanımı ne işe yarar?

- Bu şekilde kodumuzu çalıştırırsak kendi yazdığımız kod olduğu için topla fonksiyonunun kodunu da görebiliriz.

```
topla??
```

Tanımladığımız us() fonksiyonu:

- range(2, y+1) şeklinde tanımlamamamızın sebebi:
  - sonuc değerine başta ilk değer olan x'i atadık.
  - 2. değerden başlayıp (y+1). değere kadar olan değerleri çarpar.
  - (y+1)'e kadar (y+1). değer dahil değildir.

```
def us(x,y):
    sonuc = x
    for i in range(2, y+1):
        sonuc *= x
    return sonuc
```

```
us(2,5)
```

```
32
```

Fonksiyonda parametre sıraları

- Fonksiyona gönderdiğimiz parametreler sırayla çıktı verir.
- Fonksiyona yerleri farklı bir şekilde parametreleri girersek yine parametreleri fonksiyonda tanımladığımız sırada çıktı olarak verir.
- Son parametrede keyword argümanı tanımlanmazsa error ile karşılaşırız.
- Veriyi kendi sırasında göndermeliyiz.
- İlk girilen parametre kimden olarak kabul eder. Ama devamında kimden = 'parametre' olarak bir giriş yapılınca o sıraya uymadığı için error ile karşılaşılır.

```
def gonder(kimden, kime, konu, mesaj):
    print("Kimden:",kimden)
    print("Kime:",kime)
    print("Konu:",konu)
    print("Mesaj:",mesaj)
```

```
gonder('deneme@gmail.com', 'deneme2@gmail.com','ders','proje bilgileri')
```

```
Kimden: deneme@gmail.com
Kime: deneme2@gmail.com
Konu: ders
Mesaj: proje bilgileri
```

```
gonder(konu= 'ders',mesaj = 'proje bilgileri',kimden = 'deneme@gmail.com',kime = 'deneme2@gmail.com')
```

```
Kimden: deneme@gmail.com
Kime: deneme2@gmail.com
Konu: ders
Mesaj: proje bilgileri
```

```
gonder(konu='deneme@gmail.com',mesaj = 'deneme2@gmail.com',kimden = 'ders','proje bilgileri')
```

```
  File "C:\Users\user\AppData\Local\Temp\ipykernel_4260\2791227459.py", line 1
    gonder(konu='deneme@gmail.com',mesaj = 'deneme2@gmail.com',kimden =
'ders','proje bilgileri')

^
SyntaxError: positional argument follows keyword argument
```

```
gonder('deneme@gmail.com',kime = 'deneme2@gmail.com',kimden = 'ders', konu = 'proje bilgileri')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4260\2424375151.py in <module>
----> 1 gonder('deneme@gmail.com',kime = 'deneme2@gmail.com',kimden = 'ders',
konu = 'proje bilgileri')

TypeError: gonder() got multiple values for argument 'kimden'
```

SEARCH STACK OVERFLOW

Tanımladığımız soruUret() fonksiyonu:

- soruUret fonksiyonunda (soru, cevap=3)
  - soru: sorulan sorudur.
  - cevap=3: cevabın kaç defa kullanıcıdan alınacağını belirtir.
- İkinci değer girildiği takdirde kodda ikinci değer için tanımlananın yerine soruyu kullanıcıya girdiği değer kadar sorar.

```
def soruUret(soru,cevap=3):
    while cevap>0:
        x = int(input(""))
        if x =='1234':
            return True
        cevap-=1
    return False
```

```
soruUret("deneme")
```

```
12
85
67
False
```

```
soruUret("deneme",5)
```

```
12
54
98
10596
179
False
```

Fonksiyonda parametrelere değer atama

- Eğer fonksiyonda bir parametre için değer belirtimemişse fonksiyona hiç değer göndermeden çalıştıramayız.
- Sadece bir parametre için değer belirtilmemişse tek parametre ile çalıştırabiliriz. Girilen değeri o parametreye atar.
- Eğer bir parametre için değer belirtilmemişse iki parametre ile çalıştırabiliriz. İlk değeri, değeri belirtilmeyen parametreye ikinci değeri de ikinci parametredeki değeri silerek onun yerine atar.
- Eğer bir parametre için değer belirtilmemişse üç parametre ile çalıştırabiliriz. İlk değeri, değeri belirtilmeyen parametreye ikinci değeri ikinci parametredeki değeri silerek onun yerine atar, üçüncü değeri de üçüncü parametredeki değeri silerek onun yerine atar.
- Fonksiyon kaç parametreli tanımlandıysa daha fazla parametre değeri ile çalıştıramayız.

```
def f(a,b=2,c=10):
    print(a,b,c)
```

```
f()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4260\3782956317.py in <module>
----> 1 f()

TypeError: f() missing 1 required positional argument: 'a'
```

SEARCH STACK OVERFLOW

```
f(5)
```

```
5 2 10
```

```
f(5,7)
```

```
5 7 10
```

```
f(5,7,9)
```

```
5 7 9
```

```
f(5,7,9,12)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4260\2738637812.py in <module>
----> 1 f(5,7,9,12)

TypeError: f() takes from 1 to 3 positional arguments but 4 were given
```

SEARCH STACK OVERFLOW

generic fonksiyonlar:

- Yazmış olduğumuz sınıfların, fonksiyonların vb. belli bir türe göre değil, her türe göre çalışmasını sağlamaktadır.
- Örneğin; sum, len, int gibi fonksiyonlar

max(), min() ve sum() Fonksiyonları

- max() fonksiyonu, girilen değerlerin en büyüğünü döndürür.
- min() fonksiyonu, girilen değerlerin en küçüğünü döndürür.
- sum() fonksiyonu, girilen değerlerin toplamını döndürür.
    - max'a string değerler verdiğimizde en büyük karakteri verir. Her karakterin ASCII karşılığı vardır ve buna göre en büyük değeri döndürür.
    - Birden fazla string değeri max'a verdiğimizde onların arasından en büyük olanı döndürür. Yine ASCII'ye göre karşılaştırır.

```
max(4,5,6,8,9,7)
```

```
9
```

```
max("deneme")
```

```
'n'
```

```
max('a','f','def')
```

```
'f'
```

Tanımladığımız topla2() fonksiyonumuz:

- topla2() fonksiyonumuz *x şeklinde tanımlandığı için topla() fonksiyonumuzdaki gibi sadece tanımlanan parametre kadar değeri değil girilen değerlerin hepsi ile toplama işlemi yapar.
- topla2() aslında rastgele parametre alabilen argümandır.
- topla2(*x) burada * operatörü gereklidir.
- topla2() birçok parametre ile çalıştırılabilir. Hata vermez. Çünkü generic bir durum söz konusudur.
- topla2()'de değer girilmezse boş olarak yazdırır. Toplam da 0 olur.
- topla2()'de bir değer girilirse o sayı yazdırılır. Toplam da kendisine eşittir.
- topla2()'de iki değer girilirse o sayılar yazdırılır. Toplam da ikisinin toplamına eşittir.
- topla2()'de üç değer girilirse o sayılar yazdırılır. Toplam da üçünün toplamına eşittir.

```
def topla2(*x):
    print(x)
    toplam = 0
    for i in x:
        toplam += i
    return toplam
```

```
topla2()
```

```
()
0
```

```
1 topla2(1)
```

```
(1,)
1
```

```
1 topla2(1,2)
```

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

```
1 topla2(1,2,7)
```

```
(1, 2, 7)
10
```

```
1 topla2(1,2,7,7,9,6,45)
```

```
(1, 2, 7, 7, 9, 6, 45)
77
```

```
1 l = [i for i in range(10)]
2 topla2(*l)
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
45
```

for döngüsü içinde iterasyon kullanımı:

- for döngüsü içinde bir iterasyon kullanılmadığı takdirde error verir.
- iterasyon bir sayı girilirse sayıdaki rakamlar kadar fonksiyon çalışır.
    - Örneğin; '100' şeklinde iterasyonda, 100 sayısında 3 rakam yan yana olduğu için 3 kere çalışır.
- for dögüsü içinde liste de kullanılabilir, tanımlı listedeki eleman sayısı kadar fonksiyon çalışır.

```
1 for i in 100:
2     print("deneme")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4260\2736345579.py in <module>
----> 1 for i in 100:
      2     print("deneme")

TypeError: 'int' object is not iterable
```

SEARCH STACK OVERFLOW

```
1 for i in "100":
2     print("deneme")
```

```
deneme
deneme
deneme
```

```
1 for i in l:
2     print("deneme")
```

```
deneme
deneme
deneme
deneme
deneme
deneme
deneme
deneme
deneme
deneme
```

```
1 l = [i for i in range(10)]
2 topla2(*l)
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
45
```

- topla(*l) olarak l bir liste kendisi bir iterasyon tekrardan iterasyona dönüştürme demek için * işareti kullanılır.

Tanımladığımız islem1() fonksiyonumuz:

- Genelde fonksiyon parametrelerinde * operatörü kullanılıcaksa sona konulması tavsiye edilir.
- Girilen değerler operatöre göre işlem yapılır.

```
def islem1(operator,*operand):
    print(operand)
    if operator == '+':
        sonuc = 0
        for i in operand:
            sonuc += i
    elif operator == '*':
        sonuc = 1
        for i in operand:
            sonuc *= i
    return sonuc
```

```
islem1('+',10,2,15)
```

```
(10, 2, 15)
27
```

Tanımladığımız islem2() fonksiyonumuz:

- x = 10 atanır.
- y = 20 atanır.
- z = 30 atanır.
- d = 40 atanır.
- *e ise sonrasında girilen tüm değerlerdir. print(e) olarak yazdırdığımız için *e'deki değerler çıktı olarak alınır.

```
def islem2(x,y,z,d,*e):
    print(e)
```

```
islem2(10,20,30,40,10,2,0,2,5,6,4,8,5,6)
```

```
(10, 2, 0, 2, 5, 6, 4, 8, 5, 6)
```

Tanımladığımız fonk() fonksiyonumuz:

- fonk()'da ve topla2()'de de hata vermiyor çünkü ikisi de farklı veri tipi döndürüyor.
- {} Python'da dictionary (sözlük) veri tipi, () Python'da tupple (demet) veri tipidir.
- topla2() bir değerle de iki değerle de çalışır.
- Fakat fonk() bir değerle de iki değerle de çalışmaz.
- fonk() sözlük veri tipi yapısında çaılştığı için a = 5 gibi bir atama yapmak gereklidir.

```
def fonk(**kw):
    print(kw)
```

```
fonk()
```

```
{}
```

```
topla2()
```

```
()
0
```

```
topla2(5)
```

```
(5,)
5
```

```
topla2(5,2)
```

```
(5, 2)
7
```

```
fonk(5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4260\347560807.py in <module>
```

```
1 fonk(5,2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4260\1348281208.py in <module>
----> 1 fonk(5,2)

TypeError: fonk() takes 0 positional arguments but 2 were given
```

SEARCH STACK OVERFLOW

```
1 fonk(a=5, b='w')
```

```
{'a': 5, 'b': 'w'}
```

ÖRNEK BİR SORU:

Aşağıdaki kodun;

a) g(2) için,

b) g(2,3) için,

c) g(2,3,4,5,6,7) için,

d) g(2,3,4,5,6,7, a = 10, b='20') için

çıktıları ne olur?

```
1 def g(x ,y = 0, *a, **b):
2     print(x)
3     print(y)
4     print(a)
5     print(b)
```

```
1 g(2)
```

```
2
0
()
{}
```

```
1 g(2,3)
```

```
2
3
()
{}
```

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

```
2
3
(4, 5, 6, 7)
{}
```

```
1 g(2,3,4,5,6,7, a = 10, b = '20')
```

```
2
3
(4, 5, 6, 7)
{'a': 10, 'b': '20'}
```

```
1 topla(5,7)
```

```
12
```

Örnek Sorunun Çözümü:

- a) g(2) için x'e 2 değeri atanır, y = 0 olarak belirtilmişti, diğerlerinin çıktıları da sırayla () ve {} dir.
- b) g(2,3) için x'e 2 değeri atanır, y'ye 3 değeri atanır, (y için 0 değeri ortadan kalkar) diğerlerinin çıktıları da yine sırayla () ve {} dir.
- c) g(2,3,4,5,6,7) için x'e 2 değeri atanır, y'ye 3 değeri atanır, (y için 0 değeri ortadan kalkar) diğer girilen 4,5,6,7 sayıları (4,5,6,7) şeklinde yazılır, {} yine boş olarak çıktı oluşur.
- d) g(2,3,4,5,6,7, a = 10, b = '20') için x'e 2 değeri atanır, y'ye 3 değeri atanır, (y için 0 değeri ortadan kalkar) diğer girilen 4,5,6,7 sayıları (4,5,6,7) şeklinde yazılır, a = 10 ve b = '20' ataması {} içine yazılır çünkü sözlük veri yapısı tipindedir.

Bir fonksiyonunun iki farklı şekilde tanımlanması sonucunda ne olur?

- Python 2. deklarasyonda 1. deklarasyonu override etmiş olur. İlk deklarasyon kullanılmaz hale gelir.
- Arka tarafta ilk fonksiyon garbage collector tarafından silinir.
- topla() fonksiyonunda olduğu gibi.

```
1 def topla(x):
2     return x+x
```

```
1 topla(5)
```

```
10
```

Özyinelemeli (Recursive) Fonksiyon:

- Kendi kendini çağıran fonksiyonlara Özyinelemeli (Recursive) Fonksiyonlar denilir.
- Özyinelemeli fonksiyonlar, ileri bilgisayar uygulamalarında çok kullanılır.
- Bilgisayar biliminin zor sayılan konularından birisidir. Çoğunlukla, döngülerle çözülebilen problemler, özyinelemeli fonksiyonlarla çok daha kolay olarak çözülebilir.
- Biz de recursive şeklinde faktöriyel fonksiyonu tanımladık.

```
1 def fak(x):
2     sonuc = 1
3     if x == 0: #basecase
4         return 1
5     return x*fak(x-1)
```

```
1 fak(5)
```

```
120
```