Bilgisayar Programlama 2

Ders Notu: 9

Fonksiyonlar

- Fonksiyonlar, büyük bir programı daha kolay okunabilir ve bakımı kolay hale getirmek için parçalara ayırmak için kullanışlıdır. Ayrıca, programınızın birkaç farklı noktasında aynı kodu yazıyorsanız da kullanışlıdırlar. Bu kodu bir fonksiyona koyabilir ve bu kodu çalıştırmak istediğiniz her an fonksiyonu çağırabilirsiniz. Ayrıca fonksiyonları kendi yardımcı programlarınızı, matematiksel fonksiyonlarınızı vb. oluşturmak için de kullanabilirsiniz.

- **Fonksiyon Temelleri**
- Fonksiyonlar ’def’ ifadesiyle tanımlanır. İfade iki nokta üst üste ile biter ve fonksiyonun parçası olan kod ‘def’ ifadesinin altına girintili olarak yazılır. Aşağıda ekranda bir şeyler yazdıran basit bir fonksiyon ve çıktısı oluşturulmuştur:

```
1    def print_hello():
2        print('Hello!')
3
4    print_hello()
5    print('1234567')
6    print_hello()
```

```
Hello!
1234567
Hello!
```

- İlk iki satır fonksiyonu tanımlar. Son üç satırda ise fonksiyonu iki kez çağırıyoruz.

Fonksiyonlar

- Fonksiyonların bir kullanım alanı, programınızın çeşitli bölümlerinde aynı kodu tekrar tekrar kullanıyorsanız, kodu bir fonksiyona koyarak programınızı daha kısa ve anlaşılması daha kolay hale getirebilmenizdir.
- Örneğin, programınızın çeşitli noktalarında aşağıdaki gibi bir yıldız kutusu yazdırmanız gerektiğini varsayalım.

```
***************
*             *
*             *
***************
```

- Kodu bir fonksiyona yerleştirin ve bir kutuya ihtiyaç duyduğunuzda, birkaç satır gereksiz kod yazmak yerine sadece fonksiyon çağrılabilir:

```
1   def kutu_ciz():
2       print('*' * 15)
3       print('*', ' '*11, '*')
4       print('*', ' '*11, '*')
5       print('*' * 15)
6
7   kutu_ciz()
```

- Bunun bir avantajı, kutunun boyutunu değiştirmeye karar verirseniz, yalnızca fonksiyondaki kodu değiştirmeniz yeterli olmasıdır; oysa kutu çizme kodunu ihtiyaç duyduğunuz her yere kopyalayıp yapıştırsaydınız, hepsini değiştirmeniz gerekirdi.

Parametreler (Arguments)

- Fonksiyonlara değerler aktarılabilir:

```
1   def print_hello(n):
2       print('Hello ' * n)
3       print()
4
5   print_hello(3)
6   print_hello(5)
7
8   tekrar_sayisi = 2
9   print_hello(tekrar_sayisi)
```

```
Hello Hello Hello
Hello Hello Hello Hello Hello

Hello Hello
```

- print_hello fonksiyonunu 3 değeriyle çağırdığımızda, bu değer n değişkenine kaydedilir. Daha sonra fonksiyonumuzun kodunda bu n değişkenine başvurabiliriz.
- Bir fonksiyona birden fazla değer iletilebilir:

```
1   def multiple_print(kelime, n):
2       print(kelime * n)
3       print()
4
5   multiple_print('Hello', 5)
6   multiple_print('A', 10)
```

```
HelloHelloHelloHelloHello
AAAAAAAAAA
```

Fonksiyondan Sonuç Döndürme

- Hesaplamalar yapan ve sonuç döndüren fonksiyonlar yazılabilir:
- **Örnek 1**: Sıcaklıkları Celsius'tan Fahrenheit'e dönüştüren basit bir fonksiyon.

```
1   def convert(t):
2       return t*9/5+32
3
4   print(convert(20))
```

```
68
```

- `return` ifadesi, bir fonksiyonun hesaplamalarının sonucunu çağırana geri göndermek için kullanılır.

- Fonksiyonun kendisinin herhangi bir yazdırma işlemi yapmadığına dikkat edin. Yazdırma işlemi fonksiyonun dışında yapılır. Bu sayede, aşağıdaki gibi sonuçla matematiksel işlemler yapabiliriz.

```
print(convert(20))
```

- Fonksiyonun içinde sonucu döndürmek yerine doğrudan ekrana yazdırıldığında, sonuç ekrana yazdırılıp unutulur ve tekrar kullanımı mümkün olmazdı.

Fonksiyondan Sonuç Döndürme

- **Örnek 2:** Python math modülü trigonometrik fonksiyonları yalnızca radyan cinsinden hesaplar. Derece cinsinden çalışan kendi sinüs fonksiyonumuzu yazabiliriz:

```
1   from math import pi, sin
2
3   def deg_sin(x):
4       return sin(pi*x/180)
5
6   print(deg_sin(30))
```

- **Örnek 3:** Bir fonksiyon birden fazla değer döndürebilir. ax + by = e ve cx + dy = f denklem sistemini çözen bir fonksiyon:
- x = (de − b f)/(ad − bc) ve y = (a f − ce)/(ad − bc)

```
1   def solve(a,b,c,d,e,f):
2       x = (d*e-b*f)/(a*d-b*c)
3       y = (a*f-c*e)/(a*d-b*c)
4       return [x, y]
5
6   xsol, ysol = solve(2,3,4,1,2,5)
7
8   print("x =", xsol)
9   print("y =", ysol)
```

- return ifadesi birden fazla değeri liste olarak döndürmek için kullanılabilir.

Fonksiyondan Sonuç Döndürme

- **Örnek 4:** return ifadesi tek başına kullanılarak fonksiyon erken sonlandırılabilir.

```
1   def multiple_print(string, n, bad_words):
2       if string in bad_words:
3           return
4       print(string * n)
5       print()
6
7   multiple_print("Hello", 3, ["bad", "words"])
8
9   multiple_print("bad", 3, ["bad", "words"])
```

```
HelloHelloHello
```

- string bad_words listesindeyse, fonksiyon hiçbir şey yazdırmadan return ile çıkar. Aynı etkiyi if/else ile de sağlayabiliriz, ancak return kullanmak kodu daha sade hale getirir.

Varsayılan Argümanlar

- Bir argümana varsayılan değer atanabilir. Bu sayede argüman isteğe bağlı hale gelir; çağrıda belirtilmezse varsayılan değer kullanılır.
- `def multiple_print(string, n=1):`
- `    print(string * n)`
- `    print()`
- `multiple_print('Hello', 5)  # HelloHelloHelloHelloHello`
- `multiple_print('Hello')      # Hello`
- Varsayılan argümanlar, fonksiyon tanımında varsayılan olmayan argümanların sonuna yazılmalıdır.

Anahtar Kelime Argümanları

- Varsayılan argümanlarla ilgili bir kavram da anahtar kelime argümanlarıdır. Aşağıdaki gibi bir fonksiyon tanımı olduğunu varsayalım:
- `def fancy_print(text, color, background, style, justify):`
- Bu fonksiyonu her çağırdığımızda argümanların doğru sırasını hatırlamamız gerekir. Python, aşağıdaki gibi argümanlara isim vererek çağrı yapmamıza izin verir:
- `fancy_print(text='Hi', color='yellow', background='black',`
- `style='bold', justify='left')`
- Anahtar kelime argümanları kullanıldığında sıra önemli değildir. Bu kavramı print() fonksiyonunda da görmüştük: sep, end ve file argümanları.
- Varsayılan değerler ile birlikte kullanıldığında, çağrıda yalnızca değiştirmek istediğimiz argümanları belirtmek yeterlidir.

Yerel Değişkenler

- Aynı i değişkenini kullanan iki fonksiyon düşünelim:

```
def func1():                    def func2():
    for i in range(10):             i = 100
        print(i)                    func1()
                                    print(i)
```

- Bir fonksiyon içinde tanımlanan değişken o fonksiyona **yerel**dir; bu değişken fonksiyon dışında var olmaz. func1 çağrıldığında func2'deki i değişkeni etkilenmez. func2'deki print(i) komutu 100 yazdırır.
- Bu sayede her fonksiyon kendi değişkenlerini tanımlayabilir; diğer fonksiyonlardaki aynı isimli değişkenlerle çakışma olmaz.

Global Değişkenler

- Birden fazla fonksiyonun erişebileceği bir değişkene **global değişken** denir. Bir fonksiyon global değişkeni değiştirmek istiyorsa global ifadesini kullanmalıdır:

```
def reset():                def print_time():
    global time_left            print(time_left)
    time_left = 0
time_left = 30
```

- time_left değişkeni birden fazla fonksiyonun erişmesini istediğimiz global bir değişkendir. reset() fonksiyonu bu değişkeni değiştirdiğinden global ifadesini kullanır. print_time() sadece okumak istediği için global ifadesine gerek duymaz.

Örnekler

- **Örnek 1:** İki tam sayı m ve n alan ve m x n boyutunda yıldız kutusu yazdıran rectangle() fonksiyonunu yazınız.

```
def rectangle(m, n):
    for i in range(m):
        print('*' * n)
```

- **Örnek 2:** Bir tam sayının basamaklarının toplamını döndüren sum_digits() fonksiyonunu yazınız.

```
def sum_digits(num):
    return sum(int(d) for d in str(num))
```

Sorular

- **Soru 1:** İki tam sayı m ve n alan ve ekrana m x n boyutunda yıldız (*) kutusu yazdıran rectangle adlı bir fonksiyon yazınız. rectangle(2, 4) çağrısının çıktısı aşağıdaki gibi olmalıdır:
- ****
- ****
- **Soru 2:** add_excitement adlı bir fonksiyon yazınız. Fonksiyon bir string listesi alır ve her string'in sonuna ünlem işareti (!) ekler. Orijinal listeyi değiştirmeli, hiçbir şey döndürmemelidir.
- **Soru 3:** sum_digits adlı bir fonksiyon yazınız. Fonksiyon num adlı bir tam sayı alır ve num'un basamaklarının toplamını döndürür.

Sorular

- **Soru 4:** digital_root adlı bir fonksiyon yazınız. n sayısının basamak toplamını bulun, yeni sayının basamak toplamını bulun; tek basamaklı bir sayı elde edene kadar devam edin. Bu sayı dijital köktür. Örneğin n=45893 için: 4+5+8+9+3=29, 2+9=11, 1+1=2 (dijital kök=2).
- **Soru 5:** first_diff adlı bir fonksiyon yazınız. Fonksiyon iki string alır ve stringlerin ilk farklılaştığı konumu döndürür. Stringler aynıysa -1 döndürür.
- `first_diff('abc', 'abd')  # 2 döndürür`
- `first_diff('abc', 'abc')  # -1 döndürür`

Sorular

- **Soru 6:** binom adlı bir fonksiyon yazınız. Fonksiyon iki tam sayı n ve k alır ve binom katsayısını (n k) döndürür. Tanım: (n k) = n! / (k! * (n-k)!)
- `binom(5, 2)  # 10 döndürür`
- **Soru 7:** number_of_factors adlı bir fonksiyon yazınız. Fonksiyon bir tam sayı alır ve bu sayının kaç adet çarpanı olduğunu döndürür.
- `number_of_factors(12)  # 6 döndürür (1,2,3,4,6,12)`

Sorular

- **Soru 8:** factors adlı bir fonksiyon yazınız. Fonksiyon bir tam sayı alır ve bu sayının tüm çarpanlarını liste olarak döndürür.
- `factors(12)  # [1, 2, 3, 4, 6, 12] döndürür`
- **Soru 9:** closest adlı bir fonksiyon yazınız. Fonksiyon bir sayı listesi L ve bir sayı n alır; L'de n'den büyük olmayan en büyük elemanı döndürür. Örneğin L=[1,6,3,9,11] ve n=8 ise, n'den büyük olmayan 8'e en yakın eleman 6'dır.
- `closest([1,6,3,9,11], 8)  # 6 döndürür`

Sorular

- **Soru 10:** Tam sayı n alan ve tam olarak n basamaklı rastgele bir tam sayı döndüren bir fonksiyon yazınız. Örneğin n=3 ise 100-999 arasında bir sayı döndürülmelidir (093 gibi iki basamaklı değil).
- `import random`
- `def random_n_digit(n):`
- `    return random.randint(10**(n-1), 10**n - 1)`

Sorular

- **Soru 11:** Aşağıdaki programda global değişken kullanımını inceleyin ve sonucu tahmin edin. Ardından programı çalıştırarak tahmininizi doğrulayın.
- `count = 0`
- `def increment():`
- `    global count`
- `    count += 1`
- `def show_count():`
- `    print(count)`
- `increment(); increment(); show_count()  # ?`