Fonksiyonlar

Table of contents

Bölüm 4: Fonksiyonlar - Kodunuzu Organize Edin

4.1 Fonksiyonlar: Kod Tekrarından Kurtulun

Fonksiyonlar, belirli bir görevi yerine getiren, yeniden kullanılabilir kod bloklarıdır. Aynı işlemi tekrar tekrar yazmak yerine, bu işlemi bir fonksiyon içinde tanımlayabilir ve gerektiğinde çağırabiliriz. Bu, kod tekrarını azaltır, kodun okunabilirliğini artırır ve programların bakımını kolaylaştırır. Fonksiyonlar, bir programın yapı taşlarıdır ve büyük ve karmaşık bir programı daha küçük ve yönetilebilir parçalara (fonksiyonlara) ayırarak programı daha kolay anlamak, geliştirmek ve hata ayıklamak mümkün olur.

Fonksiyon Kullanmanın Faydaları:

- **Kod Tekrarını Azaltır:** Aynı işlemi tekrar tekrar yazmak yerine, bir kez fonksiyon olarak tanımlayabilir ve gerektiğinde çağırabiliriz.

- **Kod Okunabilirliğini Artırır:** Fonksiyonlar, kodun daha düzenli ve anlaşılır olmasını sağlar.
- **Kod Bakımını Kolaylaştırır:** Fonksiyonlar, kodun daha kolay değiştirilmesini ve güncellenmesini sağlar.

4.2 Fonksiyon Tanımlama

Python'da bir fonksiyon tanımlamak için `def` anahtar kelimesini kullanırız. Fonksiyon tanımının sözdizimi şu şekildedir:

```
def fonksiyon_adi(parametre1, parametre2, ...):
    """Fonksiyonun açıklaması (docstring)"""
    # Fonksiyonun gövdesi
    # Yapılacak işlemler
    return deger  # İsteğe bağlı
```

- `def`: Fonksiyon tanımı olduğunu belirten anahtar kelime.
- `fonksiyon_adi`: Fonksiyonun adı. Fonksiyon adları, değişken isimleriyle aynı kurallara tabidir.
- `parametre1, parametre2, ...`: Fonksiyona verilecek girdiler (isteğe bağlı).
- `"""Fonksiyonun açıklaması (docstring)"""`: Fonksiyonun ne yaptığını açıklayan bir metin (isteğe bağlı).
- `Fonksiyonun gövdesi`: Fonksiyonun yapacağı işlemleri içeren kod bloğu.
- `return deger`: Fonksiyonun döndüreceği değer (isteğe bağlı).

Örnekler:

```
def selamla():
    """Ekrana bir selamlama mesajı yazdırır."""
    print("Merhaba!")

def selamla(isim):
    """Verilen isimle bir selamlama mesajı yazdırır."""
    print(f"Merhaba {isim}!")

def topla(sayi1, sayi2):
    """İki sayıyı toplar ve sonucu döndürür."""
    toplam = sayi1 + sayi2
    return toplam

def find_largest_number(number1, number2, number3):
    """Üç sayıdan en büyüğünü döndürür."""
```

```
    if number1 >= number2 and number1 >= number3:
        return number1
    elif number2 >= number1 and number2 >= number3:
        return number2
    else:
        return number3
```

4.3 Fonksiyon Çağırma

Bir fonksiyonu çağırmak için, fonksiyonun adını ve ardından parantez içinde argümanları (eğer varsa) yazarız. Argümanlar, fonksiyona verilen girdilerdir.

Örnekler:

```
selamla()  # selamla() fonksiyonunu çağırır

selamla("Ahmet")  # selamla() fonksiyonunu "Ahmet" argümanıyla çağırır

sonuc = topla(5, 3)  # topla() fonksiyonunu 5 ve 3 argümanlarıyla çağırır ve sonucu sonuc değ
print(sonuc)  # Çıktı: 8
```

4.4 Geri Dönüş Değerleri (Return Values)

Fonksiyonlar, işlemlerini tamamladıktan sonra bir değer döndürebilir. Bu değer, `return` ifadesi ile belirtilir. Eğer `return` ifadesi kullanılmazsa, fonksiyon `None` değerini döndürür.

Örnek:

```
def mutlak_deger(sayi):
    """Verilen sayının mutlak değerini döndürür."""
    if sayi < 0:
        return -sayi
    else:
        return sayi

sonuc = mutlak_deger(-10)
print(sonuc)  # Çıktı: 10
```

4.5 Fonksiyonların Kapsamı (Scope)

Bir değişkenin **kapsamı (scope)**, o değişkenin programın hangi bölümlerinde erişilebilir olduğunu belirler. Fonksiyonlar, kendi **yerel kapsamlarına (local scope)** sahiptir. Bu, fonksiyon içinde tanımlanan değişkenlerin, fonksiyon dışında erişilemeyeceği anlamına gelir.

Örnek:

```
x = 10  # Global değişken x

def function():
    x = 5   # Local değişken x
    print("Fonksiyon içinde x: ", x)

function()  # Fonksiyon çağrısı
print("Fonksiyon dışında x: ", x)
```

Çıktı:

```
Fonksiyon içinde x: 5
Fonksiyon dışında x: 10
```

Global Değişkenleri Değiştirme

Bir fonksiyon içinde global bir değişkeni değiştirmek için, `global` anahtar kelimesini kullanmamız gerekir.

> **Warning**
>
> `global` anahtar kelimesini dikkatli kullanın! Fonksiyonlar içinde global değişkenleri değiştirmek, kodun okunabilirliğini ve bakımını zorlaştırabilir ve beklenmedik sonuçlara yol açabilir. Mümkün olduğunca, fonksiyonların yan etkilerinden kaçınmak ve değerleri `return` ifadesi ile döndürmek daha iyi bir uygulamadır.

Örnek:

```
x = 10 # global x

def fonksiyon():
    global x
    x = 20  # global x'i güncelle

print(x)
```

```
fonksiyon()
print(x)
```

Çıktı:

```
10
20
```

İç İçe Fonksiyonlar ve Kapsamlar

Bir fonksiyonun içinde başka bir fonksiyon tanımlayabiliriz. İçteki fonksiyon, dıştaki fonksiyonun yerel kapsamına ve global kapsama erişebilir. İçteki fonksiyon, dıştaki fonksiyonun yerel değişkenini değiştirmek için `nonlocal` anahtar kelimesini kullanabilir.

Örnek:

```
def dis_fonksiyon():
    a = 10
    print(f"Dış fonksiyon içinde a: {a}")
    def ic_fonksiyon():
        nonlocal a
        a = 20
        print(f"İç fonksiyon içinde a: {a}")

    ic_fonksiyon()
    print(f"Dış fonksiyon içinde a: {a}")

dis_fonksiyon()
```

Çıktı:

```
Dış fonksiyon içinde a: 10
İç fonksiyon içinde a: 20
Dış fonksiyon içinde a: 20
```

4.6 Varsayılan Parametre Değerleri

Fonksiyon tanımlarında parametrelere varsayılan değerler atayabiliriz. Bu sayede, fonksiyon çağrılırken bu parametreler için bir argüman verilmezse, varsayılan değer kullanılır.

Örnekler:

```python
def selamla(isim="Kullanıcı"):
    """Verilen isimle bir selamlama mesajı yazdırır.
    İsim verilmezse, "Kullanıcı" ismi kullanılır.
    """
    print(f"Merhaba {isim}!")

selamla()        # Çıktı: Merhaba Kullanıcı!
selamla("Ahmet")   # Çıktı: Merhaba Ahmet!

def us_al(sayi, us=2):
    """Verilen sayıyı verilen üsse yükseltir.
    Üs belirtilmezse, varsayılan olarak 2 kullanılır.
    """
    return sayi ** us

print(us_al(3))     # Çıktı: 9 (3^2)
print(us_al(3, 3))  # Çıktı: 27 (3^3)
```

4.7 Değişken Sayıda Argüman (`*args` ve `**kwargs`)

Fonksiyonlar, değişken sayıda argüman alabilir. `*args` ile değişken sayıda konumsal argüman, `**kwargs` ile de değişken sayıda anahtar kelime argümanı alabiliriz.

Örnek:

```python
def topla(*args):
    """Değişken sayıda sayıyı toplar."""
    toplam = 0
    for sayi in args:
        toplam += sayi
    return toplam

print(topla(1, 2)) # Çıktı: 3
print(topla(10, 20, 30, 40))  # Çıktı: 100
```

Örnek:

```python
def ogrenci_bilgileri(**kwargs):
    """Öğrenci bilgilerini yazdırır."""
    for anahtar, deger in kwargs.items():
        print(f"{anahtar}: {deger}")
```

```
ogrenci_bilgileri(isim="Ahmet", soyisim="Yılmaz", numara=12345)
```

Çıktı:

```
isim: Ahmet
soyisim: Yılmaz
numara: 12345
```

Önemli:

Fonksiyon tanımlarında `*args` ve `**kwargs` sıralaması önemlidir. Önce `*args`, sonra `**kwargs` gelmelidir.

```
def my_func(*args, **kwargs):
    print("Args:", args)
    print("Kwargs:", kwargs)
```

Doğru Kullanım:

```
my_func(1, "another_value", some_key="value")
```

Çıktı:

```
Args: (1, 'another_value')
Kwargs: {'some_key': 'value'}
```

⚠ Yanlış Kullanım

```
my_func(1, some_key="value", "another_value")  # Bu hata verir.
```

4.8 Lambda Fonksiyonları

Lambda fonksiyonları, Python'da kısa ve basit fonksiyonları tek satırda tanımlamak için kullanılır. Özellikle, fonksiyonun karmaşık bir yapısı yoksa ve sadece bir ifadeyi hesaplayıp döndürüyorsa, lambda fonksiyonları kullanışlı olabilir. lambda fonksiyonlarının bir adı yoktur ve genellikle bir değişkene atanır.

Sözdizimi:

```
lambda argümanlar: ifade
```

Örnekler:

```
kare_al = lambda x: x*x
print(kare_al(5)) # Çıktı: 25
```

```
cift_mi = lambda x: x%2 == 0
topla = lambda x, y: x + y

print(cift_mi(6)) # Çıktı: True
print(cift_mi(5)) # Çıktı: False
print(topla(3, 5)) # Çıktı: 8
```

> ⚠ Warning
>
> **Lambda Fonksiyonları Ne Zaman Kullanılmamalı?**
> **Karmaşık İşlemler:** Eğer fonksiyonunuzda birden fazla ifade veya döngü varsa, lambda fonksiyonu kullanmak kodun okunabilirliğini azaltabilir. Bu durumlarda normal fonksiyon tanımını kullanmak daha iyidir.
> **Yan Etkiler:** Lambda fonksiyonları, genellikle sadece bir ifadeyi hesaplayıp döndürdükleri için yan etkiler (global değişkenleri değiştirmek, dosya işlemleri yapmak gibi) yaratmak için uygun değildir. Eğer fonksiyonunuzun yan etkileri olması gerekiyorsa, normal `def` ile fonksiyon tanımını kullanmalısınız.

4.9 `pass` İfadesi

`pass` ifadesi, Python'da hiçbir şey yapmayan bir ifadedir. Genellikle kod bloklarını henüz tamamlanmadığı veya geçici olarak boş bırakılması gereken durumlarda kullanılır.

Örnek:

```
def fonksiyon():
  pass # Bu fonksiyon daha sonra tamamlanacak.
```

4.10 Özyineli Fonksiyonlar (Recursive Functions)

Özyineli fonksiyonlar, kendini çağıran fonksiyonlardır. Özyineli fonksiyonlar, bazı problemleri çözmek için oldukça zarif ve etkili bir yöntemdir. Özyineli fonksiyonlar, problemi daha küçük

alt problemlere bölerek çözer ve her alt problemi aynı fonksiyonu kullanarak tekrar tekrar çözer.

> **Warning**
>
> Özyineli fonksiyon yazarken dikkatli olun! Eğer temel durumu doğru bir şekilde tanımlamazsanız, fonksiyon sonsuz döngüye girebilir ve programınız çökebilir.

Özyinelemenin Temel Mantığı

- **Temel Durum (Base Case):** Her özyineli fonksiyonun, özyinelemenin durduğu bir temel durumu olmalıdır. Temel durum, problemin en küçük ve çözümü bilinen halidir.
- **Özyineli Adım (Recursive Step):** Temel duruma ulaşılmadığı sürece, fonksiyon kendini daha küçük bir girdiyle tekrar çağırır. Her çağrıda, problem daha da basitleştirilir.

Örnek Problem: Faktöriyel Hesaplama

```python
def faktoriyel(n):
    """Verilen sayının faktöriyelini hesaplar."""
    if n == 0:
        return 1
    else:
        return n * faktoriyel(n-1)

print(faktoriyel(5)) # Çıktı: 120
```

Adım Adım Özyineleme

`faktoriyel(5)` çağrısı nasıl çalışır?

1. `faktoriyel(5)` çağrılır. 5, 0'a eşit olmadığı için `else` bloğuna girilir.
2. `faktoriyel(5) = 5 * faktoriyel(4)` olarak hesaplanır.
3. `faktoriyel(4) = 4 * faktoriyel(3)` olarak hesaplanır.
4. Bu işlem `faktoriyel(1)` ve `faktoriyel(0)`'a kadar devam eder.
5. `faktoriyel(0)`'ın değeri 1'dir.
6. Döndürülen değerler zinciri geriye doğru işlenir:
    - `faktoriyel(1) = 1`
    - `faktoriyel(2) = 2 * faktoriyel(1) = 2 * 1 = 2`
    - `faktoriyel(3) = 3 * faktoriyel(2) = 3 * 2 = 6`
    - `faktoriyel(4) = 4 * faktoriyel(3) = 4 * 6 = 24`
    - `faktoriyel(5) = 5 * faktoriyel(4) = 5 * 24 = 120`
    - Sonuç: 120

Örnek Problem: Fibonacci Dizisi

`1, 1, 2, 3, 5, 8, 13, 21, 34, ...`

```
def fibonacci(n):
    """Fibonacci dizisinin n'inci elemanını döndürür."""
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(6)) # Çıktı: 8
print(fibonacci(10)) # Çıktı: 55
```

Adım Adım Özyineleme

`fibonacci(6)` çağrısı nasıl çalışır?

1. `fibonacci(6)` çağrılır. 6, 1'den küçük olmadığı için `else` bloğuna girilir.
2. `fibonacci(6) = fibonacci(5) + fibonacci(4)` olarak hesaplanır.
3. `fibonacci(5) = fibonacci(4) + fibonacci(3)` olarak hesaplanır.
4. Bu işlem `fibonacci(1)` ve `fibonacci(2)`'ye kadar devam eder.
5. `fibonacci(1)` ve `fibonacci(2)`'nin değerleri sırasıyla 1 ve 1'dir.
6. Döndürülen değerler zinciri geriye doğru işlenir:
    - `fibonacci(2) = fibonacci(1) + fibonacci(0) = 1 + 0 = 1`
    - `fibonacci(3) = fibonacci(2) + fibonacci(1) = 1 + 1 = 2`
    - `fibonacci(4) = fibonacci(3) + fibonacci(2) = 2 + 1 = 3`
    - `fibonacci(5) = fibonacci(4) + fibonacci(3) = 3 + 2 = 5`
    - `fibonacci(6) = fibonacci(5) + fibonacci(4) = 5 + 3 = 8`

4.11 Alıştırmalar

1. Kullanıcıdan iki sayı alan ve bu sayıların toplamını, farkını, çarpımını ve bölümünü hesaplayan 4 ayrı fonksiyon yazın.
2. Kullanıcıdan bir metin alan ve bu metnin içindeki sesli harflerin sayısını döndüren bir fonksiyon yazın.
3. Kullanıcıdan bir liste alan ve bu listedeki en büyük sayıyı bulan bir fonksiyon yazın.
4. Girilen bir sayının asal olup olmadığını kontrol eden bir fonksiyon yazın.
5. Kullanıcıdan bir cümle alın ve bu cümledeki kelimeleri bir liste olarak döndüren bir fonksiyon yazın.
6. Bir listenin elemanlarını toplayan bir fonksiyon yazın, ancak liste boş ise 0 döndürsün.

7. Bir metni tersine çeviren bir fonksiyon yazın. Örneğin, “Merhaba” metni “abaleM” olarak döndürülsün.
8. Bir metindeki her kelimenin ilk harfini büyük harfe dönüştüren bir fonksiyon yazın. Örneğin, “python programlama dili” metni “Python Programlama Dili” olarak döndürülsün.
9. Girilen bir sayının 2’nin kuvveti olup olmadığını kontrol eden bir fonksiyon yazın.