ÖĞRENME BİRİMİ 6

FONKSİYONLAR

Neler Öğreneceksiniz?

Bu öğrenme birimi ile;

- Fonksiyon mantığını kavrayacak,
- Gömülü fonksiyonları programlarınızda kullanabilecek,
- Kendi fonksiyonlarınızı yazabilecek, parametre gönderebilecek ve kullanabileceksiniz.

Anahtar Kelimeler:

Fonksiyon, parametre, veri döndürme, özyineleme.

Hazırlık Çalışmaları

1. İnternet üzerinden "Nesne Yönelimli Programlama" konusunu araştırınız.
2. Fonksiyonel programlama ve işlevsel programlama konularını araştırıp aralarındaki temel farkı yazınız.

6. FONKSİYONLAR

6.1. Fonksiyon

Bir bilgisayar programı yazılırken bazı işlemlerin programın farklı yerlerinde sürekli tekrarlanması gerekebilir. Örneğin arazi hesapları ile ilgili bir program yazılıyorsa sık sık geometrik şekillerin alanı hesaplanmak zorunda kalınabilir. Her gerektiğinde alan hesabı işlemini yerine getiren kodları yazmak hem programcının iş yükünü hem de hata yapma olasılığını arttırır. Bu nedenle programcılar, sık tekrarlanan işler için aynı kodu defalarca yazmak yerine, işlemi yerine getiren kod bloğunu yazıp adlandırarak ihtiyaç hâlinde bu adla basit bir şekilde çağırıp kullanmayı tercih ederler.

İhtiyaç duyulduğunda çağrılıp çalıştırılabilen bu kod paketlerine **fonksiyon** adı verilir. Fonksiyon farklı programlama dillerinde **prosedür** veya **yordam** olarak da adlandırılabilir.

Sık tekrarlanan işlemleri gerektiğinde kullanılacak küçük kod parçalarına bölüp yazma yani "fonksiyon" yaklaşımı bütün programlama dillerinde kullanılabilecek bir yöntemdir.

Fonksiyonlar sayesinde;

- Programcı aynı kodları defalarca yazma yükünden kurtulur.
- Daha az kod yazılacağı için hata yapma olasılığı azalır.
- Fonksiyon sadece çağrıldığında kullanılacağı için bilgisayarın bellek kullanımından tasarruf edilir.
- Kod okunabilirliğini arttırır ve kod analizini daha kolay hâle getirir.
- Karmaşık problemlerin daha basit küçük parçalara ayrılarak çözülmesini kolaylaştırır.

Fonksiyonlar çağrıldıklarında, barındırdıkları kod kümelerini işleyerek oluşan sonuçları döndürebilir. Ayrıca istenirse kendilerine parametre olarak gönderilen verileri işleyip ürettikleri sonucu da döndürebilir.

6.1.1. Fonksiyonların Kullanımı

Bu bölüme kadar yazılan örnek programlarda programlama dilinin bazı hazır fonksiyonları kullanıldı. Örneğin ekrana veri yazdırmak için kullandığınız print() bir fonksiyondur. Programlama dilleri yazılımcının gerektiğinde kullanabileceği birçok hazır fonksiyonla beraber gelir. Bunlara **built-in** yani **gömülü fonksiyonlar** denir.

Bir fonksiyonu çağırıp çalıştırmak için fonksiyona verilen ismi yazmak gerekir. Eğer fonksiyon parametre alıyorsa isminin yanına parantez içinde fonksiyona gönderilecek parametreleri de yazmak gerekir.

Örnek 1:

```
print("Merhaba, ben bir gömülü fonksiyonum!")
```

Yukarıdaki komut çalıştırıldığında programlama dili ile beraber gelen print isimli fonksiyon, ekrana getireceği metin fonksiyona parantez içinde parametre olarak gönderilerek çağrılmış olur.

Bu komut çalıştığında

```
Merhaba, ben bir gömülü fonksiyonum!
```

çıktısını verir ve girilen parametre ekrana metin olarak gelir.

Tanımladığımız veya programlama dili ile hazır gelen fonksiyonlar çağrılmadıkça o fonksiyon bloğu içinde yer alan kodlar çalıştırılmaz. Fonksiyonu başka bir fonksiyondan ya da doğrudan programdan ismi ile birlikte parantez içinde parametre bilgilerini yazarak çağırabilirsiniz.

6.1.2. Gömülü Fonksiyonların ve Modüllerin Kullanımı

Programlama dili ile temel işlemleri yerine getiren birçok fonksiyon hazır ve tanımlanmış olarak gelir. Şu ana kadar kullanılmış olan **print, input, type, int, float, str** gibi fonksiyonlar programlama dili içinde gömülüdür. Gömülü fonksiyonlar, geliştiricileri tarafından programlama dili içine gömülmüş ve tanımlamaya gerek kalmadan kullanılabilen fonksiyonlardır. Gömülü fonksiyonlarda tek yapılması gereken fonksiyonu çağırmak ve kullanmaktır.

Bu gömülü fonksiyonlar haricinde farklı işlevler için geliştirilmiş fonksiyon kütüphaneleri vardır. Örneğin matematik işlemlerinde ihtiyaç duyabileceğiniz tüm fonksiyonlar, hazır olarak programlama dili ve "**Math**" isimli bir kütüphane ile gelir. İhtiyaç duyduğunuzda makine öğrenmesi, oyun geliştirme, ağ işlemleri gibi alanlarda size gerekli işlevleri sağlayacak kütüphaneler programlama diline eklenip kullanılabilir.

Kendiniz de projenizde kullanmak için yazdığınız fonksiyonları bir kütüphane hâlinde toplayarak ihtiyacı olan programcılara dağıtabilirsiniz. Bu şekilde bir konuda belirli işlevleri yerine getiren fonksiyonların bir araya getirildiği Python dosyalarına **modül** denir. Hâlihazırda programlama dili kurulumu ile birlikte birçok modül bilgisayarınıza yüklenir. Bu modüller haricinde ihtiyaç duyabileceğiniz modüller de üçüncü parti sağlayıcılardan bulunabilir.

Programlama diline eklenmiş olan modülü ve içerdiği fonksiyonları kullanabilmek için önce yazılan kodun başına "import" komutu eklenerek modüle erişim sağlanır. Programlama dili kurulumu ile gelen, matematik fonksiyonlarını içeren "math.py" dosyasına yani **Math modülüne** erişmek için programın başına aşağıdaki gibi erişim ifadesi eklenmesi gerekir.

from modül_adi **import** fonksiyon_adi

Programın başlangıç kısmına bu ifade eklenerek **Math** modülünden istenilen fonksiyonlara erişilebilir.

Programa,

```
from math import sin
```

satırı eklendiğinde **Math** modülünden parametre olarak verilen sayının sinüs değerini veren fonksiyona erişim sağlanmış olur ve **sin()** tanımlı fonksiyon programda istediğiniz yerde kullanılabilir. Birden fazla fonksiyona erişim sağlamak isteniyorsa fonksiyonları aşağıdaki gibi beraber belirtmek yeterli olacaktır.

```
from math import sin, sqrt, cos, pow
```

Bu satırla programa modülden karekök bulan **sgrt()**, güç hesabı yapan **pow()** ve trigonometri işlemi yapan **sin()** ve **cos()** fonksiyonlarına erişim imkânı verilmiş olur. Bu fonksiyonlar aşağıdaki örnekle denenebilir.

Örnek 2:

```
from math import sin, sqrt, cos, pow # fonksiyonlara erişim sağlıyoruz.
print( sqrt(4) ) # 4 sayısının karekökünü buldurup ekrana yazdırıyoruz.
print( sin(30) ) # 30 sayısının sinüs değeri
print( cos(45) ) # 45 sayısının cosinüs değeri
print( pow(3,2) ) # 3'ün 2. kuvveti
```

Fonksiyonların yanında parantez içinde yer alan ifadeler parametredir. **sqrt(), sin() ve cos()** fonksiyonlarının tek parametre, **pow()** fonksiyonununsa iki parametre aldığına dikkat ediniz. Fonksiyonlar birden fazla parametre alabilir. Böyle durumlarda parametreler arasına virgül (,) konulur.

Çıktı:

```
2.0
0.9974949866040544
0.23523757330298942
9.0
```

Eğer aynı modülden çok sayıda fonksiyon kullanılması gerekiyorsa fonksiyon isimlerini spesifik olarak yazmak yerine programa **import modül_adı** satırı eklenerek modüldeki tüm fonksiyonlara erişim sağlanabilir.

```
import math
```

Aşağıdaki örnek programda Math modülündeki bütün fonksiyonlar erişime açılmış, bazıları kullanılmıştır.

Örnek 3:

```
import math
print( math.pow(3,12) )
print( math.sqrt(9) )
print( math.sin(math.pi/2) )
```

Çıktı:

```
531441.0
3.0
1.0
```

Fonksiyonlar isimleri ile erişime açılmadığı için programdan çağrılırken isimlerinin başlarına modül adları da eklendi. Bu sayede program hangi modülden hangi fonksiyona erişilmek istendiğini anlayarak fonksiyonun işlevini yerine getirmiştir.

6.2. Fonksiyon Tanımlama

Fonksiyonlar def komutu kullanarak tanımlanabilir. Fonksiyon tanımlarken izlenecek yol aşağıdaki gibidir:

1. def komutu yazılarak yeni bir fonksiyon tanımlanacağı programlama diline bildirilir.
2. Anahtar sözcükten sonra fonksiyon çağrılırken kullanılacak olan isim Python'un isimlendirme kurallarına uygun olarak belirlenmelidir. Burada fonksiyonun işlevi ile ilintili bir isim vermek kod okunabilirliği açısından mantıklı olacaktır.
3. Parantezler arasına fonksiyona gönderilecek parametreler yazılır, eğer fonksiyonumuz parametre almıyorsa parantez araları boş bırakılır. Tanım sonuna iki nokta üst üste konarak alt satırdan itibaren kod bloğunun başladığı belirtilir.
4. Tanım ve isim satırının altında bir sekme (tab) boşluk bırakılarak fonksiyon çağrıldığında çalışacak kodlar yazılır.

```
def fonksiyon_adi  ( varsa parametre listesi ) :
    Kod_blogu
    Kod_blogu
```

Bu sıralamayı izleyerek çağrıldığında ekrana "Merhaba Arkadaşlar" yazan **selamla** isimli bir fonksiyon örneği aşağıdadır:

Örnek 4:

```
def selamla():
    print("Merhaba Arkadaşlar!")
```

Yukarıdaki kod bloğu çalıştırıldığında ekrana hiçbir şey gelmeyecektir. Çünkü fonksiyon tanımlanmasına rağmen henüz çağrılmadı. Tanımlanan fonksiyonlar sadece çağrıldıklarında çalışır. Programa bir satır daha ekleyerek yazılan fonksiyon çağrılabilir.

Örnek 5:

```
def selamla():
    print("Merhaba Arkadaşlar!")


selamla() #fonksiyonu çağırıyoruz.
```

Program çalıştırıldığında fonksiyonun çağrılıp yürütüldüğü ve ekrana "Merhaba Arkadaşlar!" yazısının geldiği görülür. İstenirse aynı fonksiyon birden fazla çağrılabilir.

Örnek 6:

```
def selamla():
    print("Merhaba Arkadaşlar!")


selamla()#fonksiyonu çağırıyoruz.
selamla()#fonksiyonu 2. defa çağırıyoruz.
selamla()#fonksiyonu 3. defa çağırıyoruz.
selamla()#fonksiyonu 4. defa çağırıyoruz.
```

Kod bloğunun ekran çıktısı aşağıdaki gibi olacaktır. Selamla() fonksiyonu her çağrıldığında çalışacak ve ekrana içeriğindeki kod bloğuna uygun davranarak "Merhaba Arkadaşlar!" metnini getirecektir.

Çıktı:

```
Merhaba Arkadaşlar!
Merhaba Arkadaşlar!
Merhaba Arkadaşlar!
Merhaba Arkadaşlar!
```

Bir programda fonksiyon kullanılmadan önce tanımlanmalıdır, aksi hâlde programınız hata verecektir.

Önceki selamla() fonksiyonu bir isim verisi isteyecek şekilde düzenlendiğinde aşağıdaki kod örneği elde edilir.

Örnek 7:

```
def selamla():
  ad = str(input("Adınızı giriniz: "))
  if ad:
    print ("Merhaba " + str(ad))
  else:
    print("Merhaba Arkadaşlar!")



selamla()
```

Çıktı:

```
Adınızı giriniz: Çağan
Merhaba Çağan
Adınızı giriniz:
Merhaba Arkadaşlar!
```

Daha önceki bölümlerde döngüler konusu işlenmişti. Aşağıdaki örnekte **while** döngüsü bir fonksiyon içinde kullanılarak ekrana 1'den 10'a kadar sayılar yazdırıldı.

Örnek 8:

```
def onakadar_say():
    a = 0
    while a<10:
        a += 1
        print(a)


onakadar_say()#fonksiyonu çağırıyoruz.
```

Program çalıştırılıp fonksiyon çağrıldığında fonksiyon içindeki kodlar çalışacak ve ekrana 1'den 10'a kadar sayılar yazdırılacaktır. Örnekte gördüğünüz gibi Python'un sekmeli akış yapısı fonksiyon içinde de devam etmektedir.

Aşağıda aynı örneğin for döngüsü ile gerçekleştirilmiş hâli gösterilmektedir.

Örnek 9:

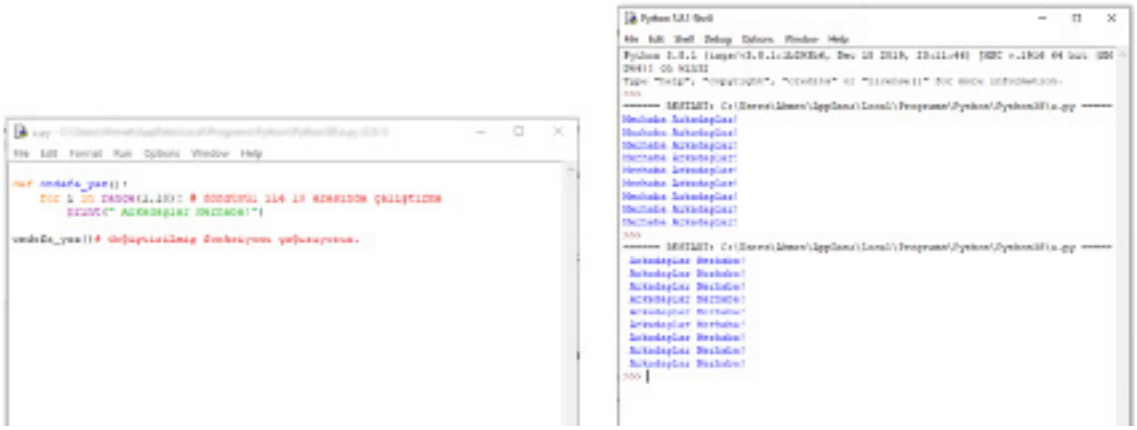
```
def ondefa_yaz():
    for i in range(1,10): # döngüyü1 ile 10 arasında çalıştırma
        print("Merhaba Arkadaşlar!")


ondefa_yaz()#fonksiyonu çağırıyoruz.
```

Kod derlendiğinde ekranda verilen metnin 10 defa yazdırıldığı görülür. Eğer **ondefa_yaz()** fonksiyonu bir daha çağrılırsa ekrana bir 10 satır daha eklenecektir.

6.2.1. Fonksiyon Düzenleme

Tanımlanan fonksiyonların işlevlerini değiştirmek için fonksiyon tanımı içerisindeki kod bloklarını yeniden düzenlemek yeterlidir. Örneğin yukarıdaki programda ekrana getirilen mesaj değiştirilmek istenirse içerikteki kodların düzenlenip yeniden çağrılması yeterli olacaktır.



Görsel 6.1: Fonksiyon düzenleme

Ekrana ilk olarak "Yıldız üçgen çiziliyor :" mesajını getiren, ardından her satıra bir döngü ile "*" işareti koyarak üçgen çizen fonksiyon örneği aşağıda gösterilmiştir.

Örnek 10:

```
def yildiz_ucgen_ciz():
    print("Yıldız üçgen çiziliyor:")
    for satir in range(1,10):
        print ("*" * satir)


yildiz_ucgen_ciz()# ücgen cizme fonksiyonu çağırılıyor.
```

Çıktı:

```
Yıldız üçgen çiziliyor:
*
**
***
****
*****
******
*******
********
*********
```

Sıra Sizde: Yukarıdaki örneği, aşağıdaki gibi bir ekran çıktısı verecek şekilde düzenleyiniz ve çalıştırınız.

```
Yıldız üçgen çiziliyor:
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
```

6.2.2. Parametre Kavramı ve Fonksiyonlar ile Parametre Kullanımı

Şu ana kadar fonksiyon tanımlarken parantez içleri hep boş bırakıldı. Bu şekilde parametre kullanmayan sadece içerdiği kod bloğunu işleyen fonksiyonlar yazıldı. Fonksiyonların en önemli işlevlerinden biri de verileri parametre olarak alıp işledikten sonra size sonucu bildirebilme yetenekleridir.

Parametre, yazılan fonksiyona işlemesi için gönderilen veridir. Metin, sayı, liste ve benzeri veriler fonksiyonlara işlenmeleri için parametre (bazen referans da denir) olarak gönderilebilir. Bunun için fonksiyon tanımlanırken parantez içerisine gönderilecek parametrenin hangi adla işleneceğinin belirtilmesi yeterlidir.

İlk yazılan **selamla()** isimli fonksiyon incelendiğinde parantez içine **"ad"** yazarak fonksiyona **"ad"** isimli bir parametre gönderileceği belirtilsin. Fonksiyon kodlarının da bu parametreyi ekrana yazdıracak şekilde düzenlenmesi gereklidir.

Örnek 11:

Çıktı:

```
Merhaba Arda
Merhaba Ali
Merhaba Çağan
```

Aldığı parametre sayesinde fonksiyon daha işlevsel hâle gelmiştir. Fonksiyona her çalıştığında farklı parametre verilerek ekran çıktısı değiştirilmiş olur.

```
def selamla(ad):
    print("Merhaba " + ad)


selamla("Arda")
selamla("Ali")
selamla("Çağan")
```

Eğer fonksiyon tanımlanırken parametre kullanacağı belirtilmiş ve parantez içine parametre tanımı yapılmışsa fonksiyon çağrılırken muhakkak parametre kullanılmalıdır. Aksi hâlde program çalışırken hata verecektir.

Programda fonksiyon ikinci defa çağrılırken parametre girilmesin.

```
def selamla(ad):
    print("Merhaba " + ad)

selamla("Arda")
selamla()
selamla("Çağan")
```

Program hatalı satıra kadar çalışacak ve hatalı satıra geldiğinde aşağıdaki gibi uyarı verecektir.

```
Merhaba Arda
Traceback (most recent call last):
  File "C:\Users\Ahmet\AppData\Local\Programs\Python\Python38\b6o1.py", line 5, in
<module>
    selamla()
TypeError: selamla() missing 1 required positional argument: 'ad'
```

Fonksiyon yazılırken alacağı parametrenin boş geçilmesi durumunda, parametre olarak varsayılan bir değer alacak şekilde düzenlenebilir.

Örnek 12:

```
def ulke_yaz(ulke = "Türkiye"): # varsayılan ülke değeri "Türkiye"
  print( ulke + " benim memleketim.")

ulke_yaz ("Türkiye")
ulke_yaz ("Azerbaycan")
ulke_yaz ()
ulke_yaz ("Almanya")
```

Çıktı:

```
Türkiye benim memleketim.
Azerbaycan benim memleketim.
Türkiye benim memleketim.
Almanya benim memleketim.
```

Parametre adlandırmaları biliniyorsa fonksiyon çağrılırken doğrudan parametrelere atama yapılabilir.

Örnek 13:

```
def en_kucuk_cocuk(cocuk3, cocuk2, cocuk1):
  print("En Genç olan cocuk " + cocuk3)

en_kucuk_cocuk (cocuk1= "Seyhan", cocuk2= "Ahmet", cocuk3= "Mehmet")
```

Çıktı:

```
En Genç olan cocuk Mehmet
```

Fonksiyona parametre olarak nesne, dizi ya da koleksiyon da gönderilebilir.

Örnek 14:

```
def mevsim_yaz(mevsim_dizisi):
  for x in mevsim_dizisi:
    print(x)

mevsimler = ["İlkbahar", "Yaz", "Sonbahar", "Kış"]

mevsim_yaz(mevsimler)
```

Çıktı:

```
İlkbahar
Yaz
Sonbahar
Kış
```

Aşağıda daha önce yazılan **yildiz_ucgen_ciz()** fonksiyonunun üçgenin satır sayısını parametre olarak alacak şekilde düzenlenmiş hâli gösterilmektedir.

Örnek 15:

```
def yildiz_ucgen_ciz(satirSayisi):
    print( str(satirSayisi) + " satırlık yıldız üçgen çiziliyor:")
    for satir in range(1,satirSayisi+1):
        print ("*" * satir)

yildiz_ucgen_ciz(6)# ücgen cizme fonksiyonu 6 satır parametresi ile çağrılıyor.
```

Satır sayısı dışarıdan fonksiyona "**satirSayisi**" adı gönderildi. Dışarıdan alınan bu değer **str()** gömülü fonksiyonu ile metne çevrilip ekrana yazdırıldı. Aynı değer döngüde de kullanıldı.

Düzenlenen yeni fonksiyonun ekran çıktısı aşağıdaki şekilde olacaktır.

```
6 satırlık yıldız üçgen çiziliyor:
*
**
***
****
*****
******
```

İstenirse bir fonksiyona virgül ile ayırarak birden fazla parametre gönderilebilir. Aşağıda bir dörtgenin alanını kenar uzunluklarını parametre alarak hesaplayan fonksiyon gösterilmiştir.

Örnek 16:

```
def dortgen_alan(kenar1, kenar2):
    alan = kenar1 * kenar2
    print("Verilen dörtgenin alanı = "+ str(alan) + " metre karedir.")



dortgen_alan(3,7)# 3 metreye 7 metre bir dörtgenin alanı içi fonksiyonu çağıralım.
```

Çıktı:

```
Verilen dörtgenin alanı = 21 metre karedir.
```

Gönderilen sayının tek ya da çift sayı mı olduğunu bulan fonksiyonu içeren program aşağıda gösterilmiştir.

Örnek 17:

```
# girilen sayı çift mi tek mi bulan fonksiyon
def cift mi_tek mi( sayi ):
    if (sayi % 2 == 0):
        print("çift")
    else:
        print("tek")
# fonksiyonu deniyoruz
Cift mi_tek mi(5)
Cift mi_tek mi(8)
```

Çıktı:

```
tek
çift
```

Sıra Sizde: Yıldızlarla üçgen oluşturan örnekten faydalanarak satır sayısını dışarıdan parametre olarak alan ve aşağıdaki gibi bir desen çıktısı verecek fonksiyonu yazınız ve çalıştırınız.

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

6.2.3. Değer Döndürme ve Return İfadesi

Daha önce yazılan fonksiyonlar, içerdikleri kodları işleyerek çalıştırıp aldıkları parametreleri kullanarak bazı işlemler yaptı. Çalışırken dışarıya herhangi bir veri göndermedi. Bu şekilde çalışıp dışarıya veri döndürmeyen fonksiyonlara **void fonksiyon** denir.

Örnek 18:

```
def ortalama_hesapla(sinav1,sinav2,sozlu):
    ortalama = (sinav1+sinav2+sozlu)/3.0  # ortalama hesaplanması
    print(ortalama) # ortalamanin yazdirilmasi
```

Yukarıdaki fonksiyonda sınav ve sözlü notlarını parametre olarak alan fonksiyon not ortalamasını bulup ekrana yazdırır. Bu fonksiyonu daha dinamik hâle getirmek için bulduğu ortalama değerini ekrana yazdırmak yerine veri olarak döndürmesi sağlanabilir.

Veri döndürme işlevi için **return** ifadesi kullanılır. Bulunan ortalama ekrana yazdırılmak yerine return ifadesi ile fonksiyon sonucu oluşan veri olarak programa geri döndürülebilir.

Ortalama sonucunu geri döndüren fonksiyon aşağıdaki örnekte gösterilmiştir.

```
def ortalama_hesapla(sinav1,sinav2,sozlu):
    ortalama = (sinav1+sinav2+sozlu)/3.0  # ortalama hesaplanması
    return ortalama # ortalamanin geriye değer olarak döndürülmesi
```

Örnek 19:

```
def ortalama_hesapla(sinav1,sinav2,sozlu):
    ortalama = (sinav1+sinav2+sozlu)/3.0  # ortalama hesaplanması
    return ortalama # ortalamanin geriye değer olarak döndürülmesi


if (ortalama_hesapla(35,60,40) < 50):
    print("Dersten kaldı")
else:
    print("Geçti")
```

Örnekte değerlendirilmek istenen notlar (35, 60 ve 40) karar cümlesi içinden parametre olarak fonksiyona gönderilmiştir. Fonksiyon bulduğu sonucu (örneğe göre 45 sonucunu bulacaktır.) geri döndürecektir. Kodlardaki karar cümlesi de gelen 45 değerine göre karşılaştırma operatörünü işletecek ve ekrana "**Dersten kaldı.**" ifadesini yazacaktır.

Aşağıdaki örnekte verilen sayının tam bölenlerini bulup liste hâlinde gönderen bir fonksiyon gösterilmiştir.

Örnek 20:

```
def tam_bolenleri_bul(sayi):
    tam_bolenler = []

    for i in range(2, sayi):
        if (sayi % i == 0):
            tam_bolenler.append(i)

    return tam_bolenler

print(tam_bolenleri_bul(15))
```

Çıktı:

```
[3, 5]
```

UYGULAMA FAALİYETİ 1

Öğrenilen fonksiyon ve değer döndürme kavramlarını kullanarak basit bir hesap makinesi programı yazınız.

```
def toplama(sayi1,sayi2):
    return sayi1+sayi2

 def cikarma(sayi1,sayi2):
    return sayi1-sayi2

 def carpma(sayi1,sayi2):
    return sayi1*sayi2

 def bolme(sayi1,sayi2):
    return sayi1/sayi2

print("Hesap Makinesi 1.0")
print("Toplama : 1, Çıkarma :2, Çarpma :3, Bölme :4, Çıkış :q")
while True:
    secim = input("İşleminiz :")
    if secim == "q":
        break

         sayi1 = int(input("Birinci sayıyı girin :"))
    sayi2 = int(input("İkinci sayıyı girin :"))
    if secim=="1":
        print("Sonuç :", toplama(sayi1, sayi2))
    elif secim =="2":
        print("Sonuç :", cikarma(sayi1,sayi2))
    elif secim =="3":
        print("Sonuc :",carpma(sayi1,sayi2))
    elif secim=="4":
        print("Sonuç :",bolme(sayi1,sayi2))
    else:
       print("Yanlış seçim lütfen tekrar deneyin")
        break
```

Çıktı:

```
Hesap Makinesi 1.0
Toplama : 1, Çıkarma :2, Çarpma :3, Bölme :4, Çıkış :q
İşleminiz :2
Birinci sayıyı girin :445
İkinci sayıyı girin :41
Sonuç : 404
İşleminiz :
```

Sıra Sizde: Yıldızlarla üçgen oluşturan örnekten faydalanarak satır sayısını dışarıdan parametre olarak alan ve aşağıdaki gibi bir desen çıktısı verecek fonksiyonu yazınız ve çalıştırınız.

Örnek kullanım:

```
#fonksiyon tanımını burada yapacaksınız.
sayi=int(input("sayi giriniz:"))
if (sayi_asalmi(sayi)):
    print("Sayı asal")
else:
    print("Sayı asal değil")
```

Örnek çıktı:

```
sayi giriniz:5
Sayı asal
```

6.3. Lambda Fonksiyonlar

Python programlama dilinde ihtiyaç hâlinde **def** ifadesi kullanmadan isimsiz, tek satırlık, küçük fonksiyonlar da tanımlanabilir. Bu fonksiyonlara **anonim fonksiyonlar** veya tanımlarken kullanılan **lambda** ifadesi nedeniyle **lambda fonksiyonlar** denir.

Lambda fonksiyon tanımı lambda [arg1 [,arg2,.....argn]]:ifade şeklinde yapılabilir.

Lambda ile tanımlanan fonksiyonlar herhangi bir sayıda argüman alabilir ancak ifade şeklinde tek bir değer döndürebilir. Ayrıca anonim olduklarından direk çağrılamaz, değişkene atanarak kullanılabilir.

Atandığında ürünün KDV'li fiyatını veren bir lambda fonksiyon tanımlanması ve kullanılması aşağıda gösterilmiştir.

Örnek 21:

```
kdvli_fiyat = lambda fiyat : fiyat * 1.18 # lambda fonksiyon tanımı


print(kdvli_fiyat(100)) #lambda fonksiyon kullanımı
```

Çıktı:

```
118.0
```

Birden fazla argüman alarak atandığında verilen üç sayıyı toplayan lambda fonksiyon aşağıda gösterilmiştir.

Örnek 22:

```
toplam = lambda a, b, c : a + b + c # lambda fonksiyon tanımı
print(toplam(4, 8, 15)) #lambda fonksiyon kullanımı
```

Çıktı:

```
27
```

Lambda fonksiyonu bu kullanımları ile pek de gerekli gözükmüyor olabilir ancak lambda fonksiyonlarının asıl gücü, başka bir fonksiyonun içinde isimsiz bir fonksiyon olarak kullanıldıklarında ortaya çıkar.

Lambda kullanarak bir parametre alıp onu henüz bilinmeyen bir sayı ile çarpan bir fonksiyon tanımlayabiliriz.

```
def carpan(n):
  return lambda a : a * n
```

Bu fonksiyonun kullanımı aşağıda gösterilmiştir.

Örnek 23:

```
def carpan(n):
  return lambda a : a * n


carpilan = carpan(5)


print(carpilan(10))
```

Çıktı:

```
50
```

Bu kullanımla fonksiyona gönderilen sayı ile çarpan belirlenebilir.

```
def carpan(n):
  return lambda a : a * n

carpilan = carpan(2)
print(carpilan(20))
```

Çıktı:

```
40
```

Örnek 24:

```
def carpan(n):
  return lambda a : a * n

ikikat = carpan(2)
uckat = carpan(3)
dortkat = carpan(4)

print(ikikat(10))
print(uckat(10))
print(dortkat(10))
```

Çıktı:

```
20
30
40
```

6.4. Özyinelemeli Fonksiyonlar

Bir fonksiyon, çözmek için tasarlandığı problemi çözerken kendi içinde yeniden kendini çağırabilir. Fonksiyonun kendi kendini çağırmasına **özyineleme (recursion),** bu tip fonksiyonlara ise **özyinelemeli (recursive) fonksiyon** denir. Bu tarz çözüm algoritmalarında döngüler, fonksiyonun kendini kopyalayıp çağırması ile oluşur. Bu kopya, fonksiyonların işleri bittiğinde yok olur. Özyinelemeli fonksiyon kendi kendini çağırabilen fonksiyon türüdür (Görsel 6.2).

Görsel 6.2: Özyinelemeli fonksiyon

Özyinelemeli fonksiyonlar, bazı durumlarda çok kullanışlı ve bazı problemlerin çözümünde elzem olsalar da fonksiyonlarda karmaşıklığı ve programlarda bellek kullanımını arttırır. Bu nedenle Python programlama dili özyineleme derinliğini varsayılan olarak 1000 ile sınırlandırır. Bu sınır aşıldığında fonksiyonunuz aşağıdaki gibi bir **RecursionError** (özyineleme hatası) verir.

```
Traceback (most recent call last):
   [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
>>>
```

6.4.1. Özyinelemeli Fonksiyonların Çalışma Şekli

Daha önce yazdığınız fonksiyonlardan birinin içinden tekrar kendini çağırmayı denerseniz programınız sonsuz döngüye girer ve yukarıdaki hatayı alırsınız.

Özyinelemeli fonksiyonlar her zaman kodun başlangıcına yazılan ve yinelemenin sınırlarını belirleyen şart (Base Case) ve fonksiyonun kendisini geri çağıran döngü (Recursive Case) kısmından oluşur. Bu sayede özyinelemeli fonksiyon hata vermeden çalışır.

Tümevarım işlemi yapan yani verilen sayıdan küçük tüm sayıları toplayan bir fonksiyonun özyinelemeli olarak yazılmış hâli aşağıdaki örnekte gösterilmiştir.

Örnek 25:

```
def tumevarim(sayi):
    # Base Case(Şart Durumu)
    if sayi == 1:
        # azalan sayinin 1'e eşit olması
        return 1
    # Recursive Case(Döngü)
    else:
        return sayi + tumevarim(sayi - 1)

print(tumevarim(5))
```

Çıktı:

```
15
```

Fibonacci dizisi, 0 ve 1 ile başlayan ve her sayının kendisinden önce gelen iki sayının toplanması ile elde edilen bir sayı dizisidir. İtalyan matematikçi Leonardo Fibonacci'den adını alır. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946,...

Fibonacci dizisinin ilk 20 elemanını yazdıran bir özyinelemeli fonksiyon aşağıda gösterilmiştir.

Örnek 26:

```
def fibonacciler(n):
    if n==1:
        return 1
    elif n==2:
        return 1
    else:
        return fibonacciler(n-1)+fibonacciler(n-2)

for i in range(1,21):
    print(fibonacciler(i), end=" ")

print()
```

Çıktı:

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

Sıra Sizde: Öğrendiklerinizi kullanarak parametre olarak girilen sayı ile faktöriyel hesabı yapan özyinelemeli fonksiyonu yazınız ve deneyiniz.

6.5. Fonksiyonlarda Kullanılan Değişkenlerin Kapsamı

Python programlama dilinde değişkenler yerel (local) ve genel (global) değişken olmak üzere iki farklı şekilde tanımlanabilir. Bir program içinde tanımlanmış tüm değişkenlere program içindeki tüm konumlardan erişilemeyebilir. Örneğin bir fonksiyon içinde tanımladığınız değişken yerel olarak konumlanır ve kapsamı sadece o fonksiyon içindedir. Yani bir fonksiyon içinde tanımladığınız bir değişkene fonksiyon dışından veya bir başka fonksiyon içinden erişilemez.

Fonksiyon dışında tanımlanmış değişkenlerin ise genel kapsamı vardır ve program içinde herhangi bir konumdan ulaşabilirsiniz.

Aynı isimli bir yerel bir de genel değişken tanımlanmış ise öncelik yerel değişkenindir.

Örnek 27:

```
sonuc = 0;
# iki sayıyı toplayan fonksiyon
def toplama( sayi1, sayi2 ):
   # Toplama
   sonuc = sayi1 + sayi2 # Yerel değişken tanımı Genel ile aynı isimlendirilmiş
   print("Fonksiyonun içinde toplam: ", sonuc)
   return sonuc

# Fonksiyonu çağırma
toplama( 5, 15 )
print("Fonksiyonun dışında toplam: ", sonuc)
```

Çıktı:

```
Fonksiyonun içinde toplam:  20
Fonksiyonun dışında toplam:  0
```

Fonksiyon içinde tanımlanan **sonuc** değişkeni 20 değeri almış olmasına rağmen öncelik yerelde olduğu için genel olarak tanımlanmış **sonuc** değişkeni 0 değerinde kalmıştır.

Program aşağıdaki gibi düzenlendiğinde:

```
sonuc = 10;
# iki sayıyı toplayan fonksiyon
def toplama( sayi1, sayi2 ):
   # Toplama

   #sonuc = sayi1 + sayi2 # Yerel değişken tanımı Genel ile aynı isimlendirilmiş
   print("Fonksiyonun içinde toplam: ", sonuc)
   return sonuc

# Fonksiyonu çağırma
toplama( 5, 15 )
print("Fonksiyonun dışında toplam: ", sonuc)
```

Çıktı:

```
Fonksiyonun içinde toplam:  10
Fonksiyonun dışında toplam:  10
```

Fonksiyon içinde herhangi bir atama veya tanımlama yapmadan genel (global) değişkene erişildi ve içerdiği değer ekrana yazdırıldı. Ama değer ataması yapılınca "**sonuc**" adında yeni bir yerel değişken tanımlandı ve atama önceliği onun oldu.

Genel bir değişkene fonksiyon içinden nasıl atama yapılır? Burada devreye **global** ifadesi girer. Fonksiyon içinde **global genel_degisken_adi** şeklinde global değişken tanımlaması yapılabilir.

Örnek 28:

```
sonuc = 0;
# iki sayıyı toplayan fonksiyon
def toplama( sayi1, sayi2 ):
   # Toplama
   global sonuc  #kullanacağımız genel değişkeni bildiriyoruz.
   sonuc = sayi1 + sayi2
   print("Fonksiyonun içinde toplam: ", sonuc)
   return sonuc

# Fonksiyonu çağırma
toplama( 5, 15 )
print("Fonksiyonun dışında toplam: ", sonuc)
```

Çıktı:

```
Fonksiyonun içinde toplam:  20
Fonksiyonun dışında toplam:  20
```

Genel değişkene fonksiyon içinden erişilmiş olur.

Sıra Sizde:

1. Parametre olarak girilen sayıdan 0'a kadar olan sayıların çarpımını yapan aşağıdaki özyinelemeli fonksiyonda bir hata bulunmaktadır. Bu hatayı belirleyip hata içermeyen programı yazınız.

```
def sifira_kadar_carp(sayi):
    if sayi == 0:
        return 0
    return sayi * sifira_kadar_carp(sayi-1)

print sifira_kadar_carp(4)

#İşleyişte sıkıntımız nedir, neden sonuc 0 geliyor?
```

UYGULAMA FAALİYETİ 2

Adam Asmaca Oyunu

```
   import random


   kelime_listesi = ["türkiye" , "gaziantep" , "istanbul" , "programlama" , "bilgisa-
yar" , "bilişim" , "okul" , "deniz"]
   secili_kelime = random.choice(kelime_listesi) #rastgele bir kelime seçiliyor
   tahmin_sayisi = 5
   harfler = [] #kullanicinin girdigi harleri saklayacağımız liste
   x = len(secili_kelime)
   z = list('_' * x)
   print(' '.join(z), end='\n')
   while tahmin_sayisi > 0:
       harf = input("Bir harf giriniz : ")
       if harf in harfler:
           print("Lütfen daha önce tahmin ettiginiz harfleri tekrar girmeyiniz...")
           continue


       elif len(harf) > 1:
           print("Sadece bir harf girilebilir.")
           continue


       elif harf not in secili_kelime:   #girilen harf kelime icinde yoksa
           tahmin_sayisi -= 1
          print("Harf kelimede yok!. {} tane tahmin hakkınız kaldı.".format(tahmin_sa-
yisi))
   else:
           for i in range(len(secili_kelime)):
               if harf == secili_kelime[i]:
                   print("Doğru Tahmin")
                   z[i] = harf
                   harfler.append(harf)
           print(' '.join(z), end='\n')


       cevap = input("Kelimenin tamamını tahmin etmek istiyor musunuz? ['e' veya 'h']
: ")
```

```
      if cevap == "e":
          tahmin = input("Kelimenin tamamını tahmin edebilirsiniz : ")
          if tahmin == secili_kelime:
              print("Tebrikler bildiniz...")
              break
          else:
              tahmin_sayisi -= 1
              print("Yanlis tahmin ettiniz. {} tane tahmin hakkınız kaldı.".format(-
tahmin_sayisi))



      if tahmin_sayisi == 0:
          print("Tahmin hakkınız kalmadı. Kaybettiniz! Adam Asıldı.")
          break
```

ÖLÇME VE DEĞERLENDİRME 6

1. Belirli işlevleri yerine getiren fonksiyonların bir arada bulunduğu Python dosyalarına ..... denir.
2. **sayi** isimli parametre alan **kare_al** fonksiyonunun tanımı ..... satırı ile yapılır.
3. Kendi kendini çağırabilen fonksiyonlara ..... denir.
4. **Aşağıdakilerden hangisi fonksiyon kullanmanın faydalarından biri değildir?**

   A) Kod tekrarından kurtulunur.
   B) Bellek kullanımından tasarruf edilir.
   C) Karmaşık problemlerin çözümü kolaylaşır.
   D) Kod güzel görünür.
   E) Hata yapma olasılığı azalır.
5. **Programlama dili ile beraber gelen fonksiyonlar hangisidir?**

   A) Void fonksiyonlar
   B) Özyinelemeli fonksiyonlar
   C) Gömülü fonksiyonlar
   D) Modüller
   E) Math modülündeki fonksiyonlar
6. **Aşağıdakilerden hangisi bir gömülü fonksiyon değildir?**

   A) input()
   B) int()
   C) print()
   D) str()
   E) selamla()
7. **Fonksiyon tanımlamaya yarayan komut aşağıdakilerden hangisidir?**

   A) return
   B) def
   C) print
   D) while
   E) global
8. **Aşağıdaki hangi ifade değişken tanımlama ile ilintilidir?**

   A) global
   B) return
   C) def
   D) print
   E) while

NOT: Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları veya faaliyetleri geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme birimine geçiniz.