LABORATUVAR ÇALIŞMASI 5 – Değer Döndüren Fonksiyonlar

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, "Değer Döndüren Fonksiyonlar" ve "Bir Fonksiyonu Diğer Bir Fonksiyona Argüman Olarak Geçirme" konularında öğrendiklerimizi pekiştirmektir.

Fonksiyonlar

Fonksiyonları argüman alıp – almamasına ve değer döndürüp – döndürmemesine göre incelersek, şu örneklere bakmamız faydalı olacaktır:

- Argüman alan ve değer döndüren fonksiyon → *Arkadaşınıza, bozması için 1 TL veriyorsunuz ve o da size iki tane 50 Kuruş veriyor.*
- Argüman alan ancak değer döndürmeyen fonksiyon → *Arkadaşınıza önceden 1 TL borcunuz vardı, ödemek için 1 TL veriyorsunuz ama ondan herhangi bir şey almıyorsunuz.*
- Argüman almayan ancak değer döndüren fonksiyon → *Arkadaşınıza bir şey vermeden ondan 1 TL borç istiyorsunuz ve size 1 TL veriyor.*
- Argüman almayan ve değer döndürmeyen fonksiyon → *Arkadaşınıza önceden 1 TL borcunuz vardı, ondan bu borcu silmesini istiyorsunuz ve o da artık kendisine borçlu olmadığınızı söylüyor. Herhangi bir şey alıp – vermiyorsunuz.*

Yukarıdaki örneklerde bahsedilen paraları **argüman**, arkadaşınıza söylemiş olduğunuz "Bu parayı bozar mısın?", "Sana borcum vardı, bu parayı alır mısın?", "Bana borç verir misin?" ve "Sana olan borcumu siler misin?" ifadelerini ise **fonksiyon çağrısı** olarak düşünebiliriz.

Fonksiyon Çağrıları

Fonksiyon çağrısı, **fonksiyonun ismini** ve ismin sağ tarafında yer alan parantezlerin içerisine **fonksiyonun aldığı argümanları** sırasıyla yazarak yapılır (Eğer fonksiyon argüman almıyorsa parantezlerin içi boş bırakılır.). Bu fonksiyon değer döndüren bir fonksiyon ise, fonksiyonun isminin sol tarafına "=" işareti, onun da soluna herhangi isimde bir değişken konularak, fonksiyonun döndürdüğü değerin, bu değişkenin içerisine atılması sağlanmış olur.

Değer döndüren bir fonksiyonun sol tarafına "=" işareti ve değişken koyarak atama işlemi yapmazsak, **fonksiyonun döndürdüğü değeri ihmal etmiş oluruz**. Bu durumda fonksiyonun döndürdüğü değer de ekrana yazılacaktır. Aşağıdaki örneği inceleyerek bunu somutlaştırabiliriz:

```
>>> a = '77'
>>> type(a)
<type 'str'>
>>> deger = int(a)
>>> deger
77
>>> int(a)
77
```

Bu örnekte "a" değişkeninin içerisine, karakter dizisi olarak "'77'" değeri atılmıştır. Python' da, aldığı değerleri tamsayıya dönüştüren (dönüştürememesi durumunda uyarı veren) "**int()**" fonksiyonu var olup, değer olarak dönüşümün sonucunu döndürmektedir. Yukarıdaki kutuda dördüncü satıra baktığımız zaman "**int()**" fonksiyonunun, "a" yı tamsayıya dönüştürerek "deger" değişkeninin içerisine attığını görürüz. Bu durumda ekrana herhangi bir şey yazılmayacaktır ve "deger" isimli değişkenin değeri tamsayı olarak 77 olacaktır. Yedinci satırda ise "**int()**" fonksiyonunun döndürdüğü değer herhangi bir değişkenin içerisine atılmadığından ihmal edilmiştir. Bu durumda ise Python yorumlayıcısı dönen bu değeri ekrana yazdıracaktır.

Yukarıdaki paragrafta incelemiş olduğumuz ve bir **tür dönüştürme fonksiyonu** olan "**int()**" fonksiyonundan başka "**float()**" ve "**str()**" fonksiyonlarından da bahsedebiliriz. "**float()**" fonksiyonu da parantezleri arasına girilen değeri ondalıklı sayıya dönüştürmeyi dener, başarılı olursa bu dönüşümün sonucunu döndürür, başarısız olursa hata verir. "**str()**" fonksiyonu ise aldığı değeri karakter dizisine dönüştürür. **Bütün sayıların, harflerin ve işaretlerin birer karakter karşılığı olduğu için "str()" fonksiyonu her durumda başarılı olur.** Örneğin, "'j'" karakterini bir tamsayıya ya da ondalıklı sayıya dönüştürmek mümkün değilken; gerek "3.277" ondalıklı sayısını, gerekse "814578" tamsayısını bir karakter dizisine dönüştürmek mümkün olabilmektedir. Aşağıdaki örneği inceleyelim:

```
>>> a = 'karakter'
>>> b = 500
>>> c = 12.34
>>> d = '500'
>>> e = '12.34'
>>> int(a)
***Hata Mesajı***
>>> float(a)
***Hata Mesajı***
>>> str(a)
'karakter'
>>> int(b)
500
>>> float(b)
500.0
>>> str(b)
'500'
>>> int(c)
12
>>> float(c)
12.34
>>> str(c)
'12.34'
>>> int(d)
500
>>> float(d)
500.0
>>> str(d)
'500'
>>> int(e)
***Hata Mesajı***
>>> float(e)
12.34
>>> str(e)
'12.34'
```

İçerisinde karakter bulunan değerlerin tamsayıya ya da ondalıklı sayıya dönüştürülmesi sırasında hata ile karşılaşmamızın nedeni, karakterlerin sayı değeri taşımamasıdır. "**int()**" fonksiyonu ise ondalıklı bir sayıyı tamsayıya dönüştürürken yuvarlama yapmasına karşın, ondalıklı sayı formatında yazılmış olan karakter dizilerinin tamsayıya dönüştürülmesini desteklememektedir. Diğer dönüşümlerde ise sorun bulunmamaktadır.

Değer Döndüren Fonksiyonlar

Yazdığımız fonksiyonun çeşitli işlemler ve hesaplamalar sonucunda bir değer üreterek bunu döndürmesini istiyorsak "**return**" komutunu kullanmamız gerektiğine değinmiştik. Şimdi bunu örnek kod üzerinde görelim:

```
def MutlakDeger(tamsayi):
    if tamsayi < 0:
        sonuc = tamsayi * (-1)
    else:
        sonuc = tamsayi
    return sonuc
```

Yukarıda, kendisine argüman olarak verilen tamsayıların mutlak değerini alan bir fonksiyon verilmiştir. "**tamsayi**" değerine göre şartlı işlem yapılarak her şart için hesaplanan değer "**sonuc**" değişkeni içerisine atılmakta ve fonksiyon da bu **sonuc** değişkeninin değerini döndürmektedir. Örnek kullanım:

```
>>> MutlakDeger(6)
6
>>> MutlakDeger(-9)
9
>>> MutlakDeger(0)
0
```

Bir Fonksiyonu Diğer Bir Fonksiyona Argüman Olarak Geçirme

Python' da fonksiyonlar; tıpkı tamsayılar, karakter dizileri, sözlükler gibi ayrı birer türdür (tür adı: **function**). Bunu, aşağıdaki örnek Python Shell ekranına bakarak da anlayabiliriz:

```
>>> def kare_aL(sayi):
        return sayi * sayi

>>> type(kare_aL)
<type 'function'>
```

Nasıl ki içerisinde tamsayı bulunan bir değişkenin ismi aslında o tamsayının bellekte bulunduğu yerin adresi ise aynı durum fonksiyon isimleri için de geçerlidir. Fonksiyonlar da aslında bellekte saklanan program parçacıklarıdır ve bellek üzerinde belli bir adresleri vardır. Yukarıdaki kutucukta yer alan "**kare_aL**" fonksiyonunun bellekte yer aldığı adres, bir bakıma, "**kare_aL**" isimli bir değişkende tutulmaktadır.

Fonksiyonların adreslerini bilmemiz, bize bir fonksiyona diğer bir fonksiyonun ismini argüman olarak geçirme şansını verir. Aşağıdaki betik örneğini inceleyelim:

```
def kok_aL(sayi):
    return sayi**0.5

def kare_aL(sayi):
    return sayi**2

def kup_aL(sayi):
    return sayi**3

def isLem_yap(fonk_adi, sayi):
    return fonk_adi(sayi)
```

F5 ile Python Shell ekranına gelelim:

```
>>> isLem_yap(kok_aL, 9)
3.0
>>> isLem_yap(kare_aL, 9)
81
>>> isLem_yap(kup_aL, 9)
729
```

Görüldüğü gibi "**isLem_yap**" fonksiyonu, diğer fonksiyonların isimlerini argüman olarak alıp, bu fonksiyonları çağırmak suretiyle değer hesaplamakta ve döndürmektedir.

Alıştırmalar

Alıştırma – 1

Görev

Python' da gerek değişkenlerin, gerekse değişkenlerin içerisine atanan değerlerin **tür**lerini döndüren "**type**" fonksiyonunu daha önceki bölümlerde incelemiştik. Aşağıdaki kutuyu inceleyiniz ve "**type**" fonksiyonunu **argüman alma** ve **değer döndürme** bakımından gözlemleyiniz. Değer döndürüp–döndürmediğini, döndürüyor ise döndürdüğü değerin türünü belirtiniz.

```
>>> a = 60
>>> type(a)
<type 'int'>
>>> b = type(a)
>>> b
<type 'int'>
>>> c = type(b)
>>> c
<type 'type'>
```

İpucu

"Fonksiyonlar" ve "Fonksiyon Çağrıları" bölümlerini inceleyiniz.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 2

Görev

"**math**" modülünde yer alan "**exp**", "**pow**" ve "**sqrt**" fonksiyonlarının prototiplerinden yararlanarak, üssel fonksiyonların yer aldığı "UsselFonksiyonlar.py" isimli bir betik dosyası oluşturunuz.

NOT: Matematikte bir **e** sabit sayısı vardır ve yaklaşık değeri **2.71828**' dir. **math** modülündeki **exp(x)** fonksiyonu da e^x değerini döndürür. **math** modülündeki **pow(x,y)** fonksiyonu x^y değerini döndürürken **sqrt(x)** fonksiyonu da $\sqrt{x}$ değerini döndürür. Sizden, bu fonksiyonların prototiplerini kullanarak yeni bir betik dosyası oluşturmanız istenmektedir. **Bunu yapmaktaki amacımız, pek çok matematiksel fonksiyonun yer aldığı math sınıfından, sadece üssel olanları ayıklayarak üssel işlemler için özelleşmiş bir sınıf oluşturmaktır.**

Bu dosya **F5** ile çalıştırıldığında örnek kullanım aşağıdaki gibi olmalıdır (e^x değerini hesapladığımız fonksiyona "e_ussu", x^y değerini hesapladığımız fonksiyona "us_hesapla", $\sqrt{x}$ değerini hesapladığımız fonksiyona ise "karekok_al" isimlerini verdiğimizi farz edersek):

```
>>> e_ussu(3)
20.085536923187668
>>> us_hesapla(7, 2)
49.0
>>> karekok_al(169)
13.0
```

İpucu

"Ara Yüz Kullanımı" ve "Değer Döndüren Fonksiyonlar" bölümlerini inceleyiniz. Betik dosyasının en başında (fonksiyon tanımlamadan hemen önce) ihtiyaç duyacağınız modülü / modülleri çağırmayı unutmayınız.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 3

Görev

"**Lab05_kiyasLa.py**" isimli betik dosyası oluşturarak içerisinde "**kiyasLa**" isminde; "**sayi_1**" ve "**sayi_2**" isminde iki argüman alan bir fonksiyon hazırlayınız (***kiyasLa(sayi_1, sayi_2)***). Fonksiyonunuz, '**sayi_1 > sayi_2**' olması durumunda '**1**', '**sayi_1 < sayi_2**' olması durumunda '**-1**', '**sayi_1 == sayi_2**' olması durumunda ise '**0**' değerini döndürmelidir. Ayrıca, fonksiyonunuza doctest kullanımı da ekleyiniz. Fonksiyonunuzda doctest kullanımı sırasında test etmeniz gereken değerler ve sonuçları şöyle olmalıdır:

	sayi_1	**sayi_2**	**sonuç**
1. Test	9	4	1
2. Test	6	6	0
3. Test	3	13	-1

İpucu

Üç tırnak işaretleri arasında doctest için gereken test işlemlerini belirttikten sonra betik dosyasının başına "**import doctest**", sonuna da "**doctest.testmod()**" (girintisiz biçimde) yazılarak test işlemi gerçekleştirilebilir:

```
import doctest

def fonksiyon(arg_1, arg_2, ...):
    """
    ...
    ...
    """
    ......
    ......

doctest.testmod()
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 4

Görev

"**Lab05_isaret_kontroL.py**" isimli betik dosyası oluşturarak içerisinde "**isaret_kontroL**" isminde; "**sayi_1**" ve "**sayi_2**" isminde iki argüman alan bir fonksiyon hazırlayınız (***isaret_kontroL(sayi_1, sayi_2)***). Fonksiyonunuz, '**sayi_1 * sayi_2**' çarpımının **pozitif** olması durumunda '**1**', **negatif** olması durumunda '**-1**', **sıfır (0)** olması durumunda ise '**0**' değerini döndürmelidir. Ayrıca, fonksiyonunuza doctest kullanımı da ekleyiniz. Fonksiyonunuzda doctest kullanımı sırasında test etmeniz gereken değerler ve sonuçları şöyle olmalıdır:

	sayi_1	**sayi_2**	**sonuç**
1. Test	9	4	1
2. Test	6	0	0
3. Test	3	-13	-1

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 5

Görev

"**Lab05_fonk_geneL.py**" isimli betik dosyası oluşturarak içerisine '*Alıştırma - 3*' ve '*Alıştırma - 4*' te hazırlamış olduğumuz "**kiyasLa**" ve "**isaret_kontroL**" fonksiyonlarınızı kopyalayınız (doctest kullanmadan). Daha sonra, betik dosyanızın içerisine "**fonk_geneL**" isminde bir fonksiyon ekleyiniz. Bu fonksiyon, "Bir Fonksiyonu Diğer Bir Fonksiyona Argüman Olarak Geçirme" başlığı altındaki örnektekine benzer biçimde, 1. argüman olarak diğer iki fonksiyondan herhangi birinin adını alabilmeli, 2. ve 3. argümanlarla verilen iki sayı değerini de alarak, ilk argümanda hangi fonksiyonun adı verilmişse aldığı iki sayıya o fonksiyon üzerinde işlem yaptırarak sonucu döndürmelidir. Betik dosyasında iken F5 tuşuna basılarak Python Shell ekranına gelindiğinde örnek kullanım aşağıdaki gibi olacaktır:

```
>>> fonk_geneL(kiyasLa, 3, 13)
-1
>>> fonk_geneL(isaret_kontroL, 3, 13)
1
>>> fonk_geneL(kiyasLa, 3, -13)
1
>>> fonk_geneL(isaret_kontroL, 3, -13)
-1
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.