

LABORATUVAR ÇALIŞMASI 11 – Sözlükler**Bu Çalışmanın Amacı**

Bu çalışmadaki amacımız, sözlükler konusunda öğrendiklerimizi pekiştirmektir.

Sözlük

Sözlükleri, “**indeks değerleri tamsayılardan oluşabileceği gibi, karakter dizilerinden de oluşabilen listeler**” olarak tanımlayabiliriz. Sözlüklerde de listelerde olduğu gibi, herhangi bir indekste herhangi bir türde (karakter dizisi, tamsayı, ondalıklı sayı vs.) veri tutulabilir.

Bir sözlüğün tanımlanmasını ve kullanılmasını gösteren örnek aşağıdadır:

```
>>> tur_ing = dict()
>>> tur_ing['bir'] = 'one'
>>> tur_ing['iki'] = 'two'
>>> tur_ing['uc'] = 'three'
>>> tur_ing['bes'] = 'five'
>>> tur_ing['iki']
'two'
>>> type(tur_ing['uc'])
<type 'str'>
>>> tur_ing['dort']
***Hata Mesajı (olmayan bir indekse erişimden dolayı)***
>>> tur_ing
{'bes': 'five', 'iki': 'two', 'bir': 'one', 'uc': 'three'}
>>> type(tur_ing)
<type 'dict'>
>>> len(tur_ing)
4
```

Sözlüğü tek bir satırda tanımlamak da mümkündür:

```
>>> tur_ing = dict()
>>> tur_ing = {'bir': 'one', 'iki': 'two', 'uc': 'three',
               'bes': 'five'}
>>> tur_ing['uc']
'three'
```

Sözlükte yer alan indislerin isimlerini (indisler içerisine atanan değerleri değil) kontrol etmek için **in** komutu kullanılabilir. Aranılan indeks isminin sözlükte bulunması durumunda **True**, diğer durumda ise **False** döndürür:

```
>>> tur_ing = dict()
>>> tur_ing = {'bir': 'one', 'iki': 'two', 'uc': 'three'}
>>> 'iki' in tur_ing
True
>>> 'two' in tur_ing
False
>>> 'ik' in tur_ing
False
```

Eğer değerlerin bulunup bulunmadığını kontrol etmek istiyorsak, “**values**” isimli fonksiyondan şu şekilde yararlanabiliriz:

```
>>> tur_ing = dict()
>>> tur_ing = {'bir': 'one', 'iki': 'two', 'uc': 'three'}
>>> degerler = tur_ing.values()
>>> 'two' in degerler
True
>>> 'iki' in degerler
False
>>> 'tw' in degerler
False
```

Sözlüklerde indeks değerlerleri tamsayılardan da oluşabilir:

```
>>> kare = dict()
>>> kare = {0 : 0, 1 : 1, 2 : 4, 3 : 9, 4 : 16, 5 : 25}
>>> kare[0]
0
>>> kare['0']
***Hata Mesajı***
>>> kare[4]
16
```

Tuple Veri Türü

Sıralanmış değerler topluluklarıdır. Gösterimi aşağıdaki gibi olup, tek bir elemana sahip olsa bile tanımlama esnasında bu elemandan sonra virgül konmalıdır (**t1** ve **t2**’ de olduğu gibi):

```
>>> t = ('a', 'b', 'c', 'd', 'e')
('a', 'b', 'c', 'd', 'e')
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
>>> p = tuple('python')
>>> print p
('p', 'y', 't', 'h', 'o', 'n')
>>> print p[1]
y
>>> print p[3 : 5]
('h', 'o')
>>> p[4] = 'j'
***Hata Mesajı (Tuple veri türü, atamayı desteklemiyor.)***
>>> u = ('f',) + p[1:]
>>> print u
('f', 'y', 't', 'h', 'o', 'n')
```

Yukarıdaki kutuya bakarak, tuple veri türünün atamayı desteklemediğini görebiliriz. İki değişkenin değerlerini **değiş - tokuş** yapmak için tuple yapısı büyük kolaylık sağlar:

```
>>> a = 3
>>> b = 7
>>> a, b = b, a
>>> a
7
>>> b
3
```

“**split**” fonksiyonunu kullanarak, örneğin, bir elektronik posta adresini “@” işaretinden öncesi ve sonrası olmak üzere ikiye ayırma işlemini rahatlıkla yapabiliriz:

```
>>> adres = 'python-help@python.org'
>>> kullanıcı_adi, alan_adi = adres.split('@')
>>> print kullanıcı_adi
python-help
>>> print alan_adi
python.org
```

NOT: **split** fonksiyonu değer olarak liste döndürmektedir. Buradaki atama biçimi, tuple ataması mantığına uyduğu için bu örnek verilmiştir.

Bazı fonksiyonların geri döndürdüğü değerler **tuple** türünden olabilir. Örneğin “**divmod**” fonksiyonu bir tamsayıyı diğerine böldüğünde sonuç olarak hem bölümü hem de kalanı döndürür:

```
>>> sonuc = divmod(7, 2)
>>> print sonuc
(3, 1)
>>> bolum, kaLan = divmod(7, 2)
>>> print boLum
3
>>> print kaLan
1
```

Peki, bu şekilde değer döndüren fonksiyonların tanımlanması nasıl olmalı? Aşağıdaki betik dosyasında yer alan ve verilen bir tamsayının karesini ve karekökünü döndüren fonksiyonu inceleyelim:

```
def kare_ve_karekok(tamsayi):
    if ((type(tamsayi) == int) & (tamsayi >= 0)):
        return tamsayi ** 2, tamsayi ** 0.5
    else:
        print 'Lutfen pozitif bir tamsayi giriniz.'
```

F5 ile Python Shell ekranına geldiğimizde:

```
>>> kare_ve_karekok(16)
(256, 4.0)
>>> kare, karekok = kare_ve_karekok(25)
>>> print kare
625
>>> print karekok
5.0
```

Fonksiyonların değişken sayıda argüman almasını sağlamak için “*” işareti kullanılır.

```
def hepsini_yazdir(* argumanlar):
    print argumanlar
```

Bu betik dosyasını çalıştırsak:

```
>>> hepsini_yazdir(5, 'klm', 7.0, "pqr")
(5, 'klm', 7.0, 'pqr')
```

Yukarıdaki fonksiyona daha fazla sayıda argüman verilse de hepsini ‘tuple’ şeklinde ekrana yazdıracaktır.

Benzer biçimde, birden fazla sayıda argüman alan bir fonksiyona, içerisinde o fonksiyonun alacağı sayıda değer bulunduran bir ‘tuple’ ı argüman olarak vermek için gene “*” işareti kullanılır. İki argüman alan **divmod** fonksiyonuna argüman olarak ‘tuple’ geçirilmesi ile ilgili örneği inceleyelim:

```
>>> t = (7, 3)
>>> divmod(t)
***Hata Mesajı (divmod, tuple'ı tek argüman gibi görüyor)***
>>> divmod(*t)
(2, 1)
```

tuple’ lar, büyüklük-küçüklük, eşitlik yönünden karşılaştırılabilirler. İki ‘tuple’ karşılaştırılırken ilk önce birinci elemanlarına bakılır. Birinci elemanı diğerinden büyük olan ‘tuple’, diğerinden daha büyüktür (Geri kalan elemanlara bakılmaz.). Eşitlik halinde ise 2. elemana, 3. elemana... bakılır:

```
>>> (0, 1, 99) < (0, 2, 0)
True
>>> (1, 2, 3) < (1, 1, 77)
False
```

Yazdığımız programlarda; **karakter dizisi**, **liste**, **tuple** gibi veri türlerinden birini bir diğerine dönüştürme ya da tamsayı, ondalıklı sayı gibi türlerdeki verileri bu yapılardan herhangi birinde saklama ihtiyacı duyabiliriz. Aşağıdaki fonksiyonu inceleyelim:

```
def sarmala(kaynak, yapi):
    # yapi nin turu karakter dizisi ise
    if type(yapi) == type(""):
        # degeri karakter dizisi olarak dondur
        return str(kaynak)
    # yapi nin turu liste ise
    if type(yapi) == type([]):
        # degeri liste olarak dondur
        return [kaynak]
    # yapi nin turu ikisi de degilse degeri tuple olarak dondur
    return (kaynak,)
```

Burada “kaynak” isimli argümanın taşıdığı değer, “yapi” argümanının ait olduğu türe dönüştürülmektedir. Aşağıdaki örnek kullanımları inceleyelim:

```

>>> sarmala(7,"")
'7'
>>> sarmala(7,[])
[7]
>>> sarmala(7,())
(7,)
>>> sarmala(7,0)
(7,)
>>> sayi = 21
>>> st = sarmala(sayi, "")
>>> st
'21'
>>> type(st)
<type 'str'>
>>> Ls = sarmala(sayi, [])
>>> Ls
[21]
>>> type(Ls)
<type 'list'>
>>> tp = sarmala(sayi, ())
>>> tp
(21,)
>>> type(tp)
<type 'tuple'>
>>> sarmala((2, 3, 4, 5), [])
[(2, 3, 4, 5)]
>>> sarmala([2, 3, 4, 5], ())
([2, 3, 4, 5],)
>>> sarmala([2, 3, 4, 5], "")
'[2, 3, 4, 5]'

```

Görüldüğü gibi, **sarmala** fonksiyonunun 2. argümanı olan **yapi**' ya karakter dizisi türünden herhangi bir değer (örnekteki "") verilirse, 1. argüman olan **kaynak** ile taşınan değer karakter dizisine dönüştürülmektedir. Eğer **yapi**' ya liste türünden herhangi bir değer (örnekteki boş liste) verilirse 1. argüman olan **kaynak** ile taşınan değer **liste** içerisine atılarak elde edilen bu yeni liste döndürülmektedir. Eğer **yapi**' ya bunların haricinde başka türden herhangi bir değer verilirse, 1. argüman olan **kaynak** ile taşınan değer **tuple** içerisine atılarak elde edilen bu yeni tuple döndürülmektedir.

Alıştırmalar

Alıştırma – 1

Görev

Argüman olarak bir sözlük ve değer alan, sonuç olaraksa o değerin yer aldığı **tüm indeksleri** ekrana yazdıran (eğer o değer hiçbir indekste yer almıyorsa ekrana “İndeks bulunamadı.” yazdıran) ve değer döndürmeyen bir fonksiyon hazırlayınız. Fonksiyonunuza “**ters_arama**” adını vererek “**Lab10_ters_arama.py**” adlı betik dosyasına kaydediniz. Fonksiyonun kullanılmasına dair örnek aşağıdadır:

```
>>> sehirler = {'Samsun': 'Karad.', 'Antalya': 'Akd.',
               'Tokat': 'Karad.', 'Manisa': 'Ege'}
>>> ters_arama(sehirler, 'Karad.')
Samsun
Tokat
>>> ters_arama(sehirler, 'Akd.')
Antalya
>>> ters_arama(sehirler, 'Ege')
Manisa
>>> ters_arama(sehirler, 'Manisa')
İndeks bulunamadı.
```

Sonuç

Gerçekleştirmenizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 2**Görev**

“0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 ...” şeklinde devam eden ve her elemanın, kendisinden önceki iki elemanın toplamından oluştuğu kurallı diziye “**Fibonacci dizisi**” denir (Bu dizinin sıfırıncı elemanını **0**, yedinci elemanını **13**, onuncu elemanını **55** olarak düşünelim.). “**fibonacci**” adında bir fonksiyonumuz olduğunu ve argüman olarak negatif olmayan bir tamsayı alıp, sonuç olarak Fibonacci dizisinin o tamsayıya karşılık gelen indeksteki elemanı döndürdüğünü düşünürsek, “**fibonacci(11)**” bize **89** değerini verecektir. **fibonacci** fonksiyonunun sonucu hesaplamak için yapacağı işlemler, argüman olarak aldığı tamsayı büyüdükçe artacaktır. **Bunun için, performans açısından belli bir değere kadarki tamsayıların Fibonacci karşılıklarının program içerisinde tutulmasında yarar vardır (küçük değerlerin daha sık kullanıldığını düşünürsek).**

10 ve 10’ dan küçük tamsayılar (argüman olarak fonksiyona verilen) için toplama ve benzeri hesaplama işlemi yapmadan direkt sonuç döndüren, 10’ dan büyük tamsayılar için hesaplama yaparak sonuç döndüren bir Fibonacci fonksiyonu oluşturunuz. Fonksiyonunuza “**hizli_fibonacci**” ismini vererek “**Lab10_hizli_fibonacci.py**” isimli betik dosyasına kaydediniz. Fonksiyonun çalıştırılmasına ait örnek aşağıdadır (**doctest** olarak ekleyiniz.):

```
>>> hizli_fibonacci(7)
13
>>> hizli_fibonacci(11)
89
>>> hizli_fibonacci(19)
4181
>>> hizli_fibonacci(0)
0
>>> hizli_fibonacci(-7)
Lutfen pozitif bir tamsayi giriniz.
>>> hizli_fibonacci(12.7)
Lutfen pozitif bir tamsayi giriniz.
```

İpucu

“Sözlük” başlığını inceleyiniz.

Sonuç

Gerçekleştirmenizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 3

Görev

Sınırsız sayıda sayının tek bir fonksiyon çağırısı ile toplanabilmesi için, sınırsız sayıda argüman kabul ederek aldığı sayıları toplayan ve ekrana toplamı yazdıran bir fonksiyon hazırlayınız. Fonksiyonunuza **“sinirsiz_toplam”** ismini vererek **“Lab10_sinirsiz_toplam.py”** isimli betik dosyasına kaydediniz. Fonksiyonun çalıştırılmasına ait örnek aşağıdadır:

NOT: Fonksiyona, sadece tamsayı ve ondalıklı sayı girileceği garanti edilmiştir. Ayrıca hata kontrolü yapmanız gerekmemektedir.

```
>>> sinirsiz_toplam(3, 5.0, 8, -0.1, 0)
15.9
>>> sinirsiz_toplam(4, -5, 6)
5.0
```

İpucu

“Tuple Veri Türü” bölümünü inceleyiniz.

Sonuç

Gerçekleştirmenizi ve / veya karşılaştığınız problemleri raporunuza yazınız.