LABORATUVAR ÇALIŞMASI 8 – Listeler

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, listeler konusunda öğrendiklerimizi pekiştirmektir.

Listeler

Python' da aynı veya farklı türde bir ya da birden çok değişkeni içerisinde bulunduran yapılara "liste" adı verilir. Listeler köşeli parantezler ile ("[", "]") ifade edilirler ve birden çok elemana sahip olan listelerde köşeli parantezler arasına yazılan elemanlar birbirinden virgülle ayrılır. Bir listenin elemanı bir ya da birden çok elemana sahip olan bir ya da birden fazla sayıda liste olabilir. Örnekler:

```
>>> ondan_kucuk_asal_sayilar = [2, 3, 5, 7]
>>> ondan_kucuk_asal_sayilar
[2, 3, 5, 7]
>>> type(ondan_kucuk_asal_sayilar)
<type 'list'>
>>> ondan_kucuk_asal_sayilar[3]
7
>>> ondan_kucuk_asal_sayilar[0]
2
>>> ondan_kucuk_asal_sayilar[4]
***Hata Mesajı (olmayan bir indekse erişimden dolayı)***
>>> sehir_bilgileri = ['Samsun', 55, "Karadeniz"]
>>> sehir_bilgileri
['Samsun', 55, "Karadeniz"]
>>> sehir_bilgileri[0]
'Samsun'
>>> sehir_bilgileri[1]
55
>>> type(sehir_bilgileri[0])
<type 'str'>
>>> type(sehir_bilgileri[1])
<type 'int'>
>>> bolge = ["Karadeniz", ['Samsun', 55], ['Zonguldak', 67]]
>>> bolge
["Karadeniz", ['Samsun', 55], ['Zonguldak', 67]]
>>> bolge[0]
'Karadeniz'
>>> bolge[1]
['Samsun', 55]
>>> bolge[2][0]
'Zonguldak'
>>> bolge[2][1]
67
>>> type(bolge[2][1])
<type 'int'>
```

Örnekte de gördüğümüz gibi listelerin elemanlarına, hatta liste içerisinde yer alan listelerin elemanlarına, indekslerini belirterek (köşeli parantezler içerisinde) ulaşmamız mümkündür. Herhangi bir değerden (örneğin **0**) birden fazla (örneğin 10 tane) içeren bir listeyi şu şekilde tanımlayabiliriz:

```
>>> on_tane_sifir = [0] * 10
>>> on_tane_sifir
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Herhangi bir listenin herhangi bir elemanını sonradan değiştirmek mümkündür:

```
>>> sehir = ['Tokat', 60, 'Marmara']
>>> sehir
['Tokat', 60, 'Marmara']
>>> sehir[2]
'Marmara'
>>> sehir[2] = 'Karadeniz'
>>> sehir
['Tokat', 60, 'Karadeniz']
>>> sehir[2]
'Karadeniz'
```

Herhangi bir değerin liste içerisinde yer alıp - almadığını kontrol etmek için "**in**" komutu kullanılır. Sonuç olarak, **bool** türünde **True** ya da **False** dönecektir. Örnek:

```
>>> sehir = ['Tokat', 60, 'Karadeniz']
>>> 'Tokat' in sehir
True
>>> 60 in sehir
True
>>> 'Toka' in sehir
False
>>> 6 in sehir
False
>>> 600 in sehir
False
```

"***for*** *[eleman]* **in** *[liste]*" kalıbı kullanılarak liste içerisinde gezilebilir:

```
>>> sehir = ['Izmir', 35, ['Konak', 'Buca', 'Karsiyaka']]
>>> sehir
['Izmir', 35, ['Konak', 'Buca', 'Karsiyaka']]
>>> for her_bir_eleman in sehir:
        Print her_bir_eleman


Izmir
35
['Konak', 'Buca', 'Karsiyaka']
```

for döngüsü ile listeler üzerinde gezilerek işlem yapılabilir:

```
>>> asal_sayilar = [2, 3, 5, 7]
>>> asal_sayilar
[2, 3, 5, 7]
>>> for i in range(0, len(asal_sayilar)):
        asal_sayilar[i] = asal_sayilar[i] * 2


>>> asal_sayilar
[4, 6, 10, 14]
```

Bir listenin içerisinde yer alan diğer listelerden her biri tek bir eleman olarak değerlendirilir:

```
>>> dizi1 = [1, 2, 3]
>>> dizi2 = [1, [2, 3, 4, 5], ['a', 'b', 'c', 6]]
>>> len(dizi1)
3
>>> len(dizi2)
3
```

"+" operatörü iki listeyi arka arkaya birleştirirken "*" operatörü de listeyi, kendisinin sağındaki tamsayı kadar tekrar ettirerek arka arkaya birleştirir:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 'k']
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 'k']
>>> d = b * 3
>>> print d
[4, 5, 'k', 4, 5, 'k', 4, 5, 'k']
```

Listeler, dilimlere ayrılabilirler:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1 : 3]
['b', 'c']
>>> t[: 4]
['a', 'b', 'c', 'd']
>>> t[3 :]
['d', 'e', 'f']
>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1 : 3] = ['x', 'y']
>>> print t
['a', 'x', 'y', 'd', 'e', 'f']
```

"**append**" fonksiyonu, listenin en sonuna yeni bir eleman eklemede kullanılır:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print t
['a', 'b', 'c', 'd']
```

"**extend**" fonksiyonu, bir listenin sonuna diğer bir listeyi eklemek için kullanılır:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print t1
['a', 'b', 'c', 'd', 'e']
>>> print t2
['d', 'e']
```

Listenin elemanlarını küçükten büyüğe doğru sıralamak için "**sort**" fonksiyonu kullanılır:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print t
['a', 'b', 'c', 'd', 'e']
>>> u = ['d', 4, 'c', -1, 7, 'e', 0, 'b', 1.7, 'a']
>>> u.sort()
>>> print u
[-1, 0, 1.7, 4, 7, 'a', 'b', 'c', 'd', 'e']
```

Listenin herhangi bir indeksindeki elemanı çıkartıp bir değişkene atmak için "**pop**" fonksiyonu kullanılır:

```
>>> t = ['a', 'b', 'c']
>>> print t[1]
b
>>> x = t.pop(1)
>>> print t
['a', 'c']
>>> print t[1]
c
>>> print x
b
```

Eğer listeden çıkarılmak istenen eleman artık kullanılmayacaksa "**del**" komutu ile silinebilir. Bu komut, dilimleme ile birlikte de kullanılabilir:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> print t
['a', 'c']
>>> u = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del u[1 : 5]
>>> print u
['a', 'f']
```

Listeden silinecek elemanın indeksi bilinmiyor ama değeri biliniyorsa "**remove**" fonksiyonu kullanılabilir:

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> print t
['a', 'c']
```

Bir karakter dizisini bir listeye karakter karakter atmak için "**list**" fonksiyonu kullanılır:

```
>>> s = 'python'
>>> t = list(s)
>>> print t
['p', 'y', 't', 'h', 'o', 'n']
```

Boşluklar içeren bir karakter dizisinin boşluklarla ayrılmış olan her bir harf grubunu bir listenin indekslerine sırasıyla atmak için "**split**" fonksiyonu kullanılır:

```
>>> s = 'Python ile programlama yapmak'
>>> t = s.split()
>>> print t
['Python', 'ile', 'programlama', 'yapmak']
```

Yukarıdaki işlemde ayırıcı olarak boşluk karakterini (" ") kullanmıştık. Aynı içlem için başka karakterler kullanmak da mümkündür. Örnek kullanım ("-" karakteri için) şu şekildedir:

```
>>> s = 'Python2.4.2-Python2.6.4-Python3.1.1'
>>> delimiter = '-'
>>> s.split(delimiter)
['Python2.4.2', 'Python2.6.4', 'Python3.1.1']
```

split fonksiyonunun tersi göreve sahip olan fonksiyon ise "**join**" dir:

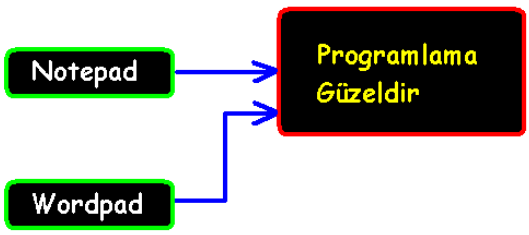
```
>>> t = ['Python2.4.2', 'Python2.6.4', 'Python3.1.1']
>>> delimiter = '+'
>>> delimiter.join(t)
'Python2.4.2+Python2.6.4+Python3.1.1'
```

Python' da **değişken**ler, bellekte belli bir adrese sahip olan **nesne**lerin adreslerini tutan **tutacaklardır**. Örneğin, Windows' ta bilgisayarımızın masaüstünde bir **metin belgesi** oluşturup içerisine bir şeyler kaydedip kapattıktan sonra bunu hem **Notepad** metin editörü ile, hem de **Wordpad** metin editörü ile açabiliriz. Burada metin belgesini **nesne**, her iki metin editörünü de **iki ayrı tutacak** gibi düşünebiliriz. Belgeyi **Notepad** ile açıp bir değişiklik yapıp kaydedersek, **Wordpad** ile açtığımızda değişiklik yapılmış hali ile karşılaşırız. Oysa iki ayrı metin dosyası belirleyerek birisine "**N.txt**", diğerine ise "**W.txt**" isimlerini verip, **N.txt**' yi sadece **Notepad** ile, **W.txt**' yi ise sadece **Wordpad** ile kullanıyor olsaydık, her bir metin editörü ile yapılan değişiklik, sadece o editörle açtığımız belgelerde karşımıza çıkacaktı.

Yukarıda açıklanan durumun mantığını göz önünde bulundurarak aşağıdaki kodları inceleyelim:

```
>>> Notepad = 'Programlama guzeldir.'
>>> Wordpad = Notepad
>>> Notepad
'Programlama guzeldir.'
>>> Wordpad
'Programlama guzeldir.'
>>> Notepad is Wordpad
True
```

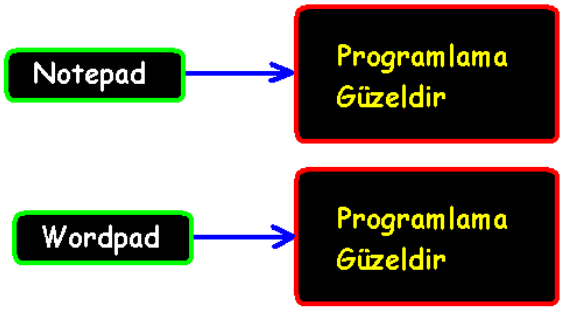
İki değişkenin (yani **tutacağın**) gösterdiği yerdeki nesnelerin **aynı nesne** olup - olmadığını öğrenmek için "**is**" komutu kullanılır.



Şimdi ise aşağıdaki kodun yukarıdakinden farklılığını irdeleyelim:

```
>>> Notepad = 'Programlama guzeldir.'
>>> Wordpad = 'Programlama guzeldir.'
>>> Notepad
'Programlama guzeldir.'
>>> Wordpad
'Programlama guzeldir.'
>>> Notepad is Wordpad
False
```

Burada ise iki farklı **tutacak** için **iki farklı nesne** bulunmaktadır. Tutacakların gösterdikleri yerlerdeki nesneler farklı farklı nesnelerdir (her ne kadar içerikleri aynı olsa da).



Aşağıdaki örnekte ise, nesnenin tutacaklar tarafından paylaşılması durumunda tutacaklardan biri ile nesnenin değiştirilmesi sonucunda değişikliğin diğer tutacağa da yansıdığı görülmektedir:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
>>> b[0] = 17
>>> print a
[17, 2, 3]
>>> print b
[17, 2, 3]
```

Listeler, tıpkı diğer değişkenler gibi (karakter dizisi, ondalıklı sayı vs.) fonksiyonlara **argüman** olarak verilebilirler. Aşağıda, kendisine argüman olarak verilen listenin ilk elemanını silen bir fonksiyon verilmiştir (betik dosyasında):

```
def ilk_elemani_sil(liste):
    del liste[0]
```

Bu fonksiyonun Python Shell' de kullanımını inceleyelim:

```
>>> kent = ['a', 'n', 'k', 'a', 'r', 'a']
>>> ilk_elemani_sil(kent)
>>> kent
['n', 'k', 'a', 'r', 'a']
```

Alıştırmalar

Alıştırma – 1

Görev

Bir sözcüğün içerisindeki karakterlerin soldan sağa alfabetik sıraya göre dizilip - dizilmediğini kontrol eden bir fonksiyon yazınız. Fonksiyona "**alfabetik_artan**" ismini veriniz ve "**lab08_alfabetik_artan.py**" isimli betik dosyasına kaydediniz. Fonksiyon argüman olarak sözcük almalı, değer olaraksa **bool** türünde True (sözcüğün alfabetik olarak dizildiği durumlar için) ya da False (diğer durumlar için) döndürmelidir. Fonksiyona sadece küçük harflerden oluşan ve Türkçe harf içermeyen sözcükler girilmelidir. Örnek kullanım aşağıdadır:

```
>>> alfabetik_artan('adem')
True
>>> alfabetik_artan('koop')
True
>>> alfabetik_artan('kaan')
False
>>> alfabetik_artan('emre')
False
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 2

Görev

Argüman olarak bir liste alan ve sonuç olarak ekrana, bu listenin elemanlarından kaçının karakter dizisi, kaçının tamsayı, kaçının ondalıklı sayı, kaçının "bool", kaçının liste olduğunu yazan, değer döndürmeyen bir fonksiyon yazınız. Fonksiyonunuza "**liste_analizi**" ismini veriniz ve "**lab08_liste_analizi.py**" isimli bir betik dosyasına kaydediniz. Fonksiyonun, argüman olarak verilen listenin içinde eleman olarak bulunan diğer listelerin içerikleriyle ilgilenmesine lüzum yoktur. Fonksiyonun çalıştırılmasına dair örnek ekran görüntüsü aşağıdadır:

```
>>> su = ['H2O', 18, 0, 100.0, 'Bilesik', "Kovalent",
['Hidrojen', 2], ['Oksijen', 1], 3, True]
>>> liste_analizi(Su)
Karakter dizisi sayisi : 3
Tamsayi sayisi         : 3
Ondalikli sayi sayisi  : 1
Boolean sayisi         : 1
Liste sayisi           : 2
>>> liste_analizi( [1, '1.1', [1.1, 1.1], [False, 7, True], -1] )
Karakter dizisi sayisi : 1
Tamsayi sayisi         : 2
Ondalikli sayi sayisi  : 0
Boolean sayisi         : 0
Liste sayisi           : 2
```

İpucu

"**lis**" isimli bir dizinin eleman sayısını bulmak için "**len(lis)**" komutu kullanılabilir.

Bir **a** değişkeninin değerinin tamsayı olup - olmadığını kontrol etmek için "**type(a)==type(7)**" ifadesi kullanılabilir.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 3

Görev

Elimizde "-" işaretleri ile ayrılmış, "6-19-4-60-22-7-4" şeklinde, pozitif tamsayılardan oluşan karakter dizileri var. **a**, bu dizideki tamsayıların ortalaması (tamsayıların toplamının tamsayı adedine bölümü [tamsayı bölme]) olsun. **b** ise, bu dizi küçükten büyüğe doğru sıralandığında, eğer tek sayıda eleman varsa en ortadaki (5 tane tamsayı varsa baştan 3.) sayı, çift sayıda eleman varsa ortadaki iki sayının ortalaması (6 tane tamsayı varsa 3. ve 4. sayının toplamının yarısı [tamsayı bölme ile]) olsun. **a * b** ise bu dizinin **anahtarı** olsun.

Argüman olarak yukarıdaki gibi bir tamsayı dizisi alan, sonuç olarak ise dizinin anahtarını döndüren bir fonksiyon hazırlayınız. Fonksiyonunuza "**dizi_anahtari**" ismini vererek "**lab08_dizi_anahtari.py**" isimli bir betik dosyasına kaydediniz.

NOT: Fonksiyonunuza yukarıdaki formatta ve en az 3 tane pozitif tamsayıdan oluşan karakter dizilerinin **doğru** bir biçimde girileceği garanti edilmektedir. Aksi durumlar için hata kontrolü yapmanıza **lüzum yoktur**.

Örnek olarak **'3-10-7-5'** dizisinde çift sayıda eleman vardır. Bunları küçükten büyüğe doğru sıraladığımızda **'3-5-7-10'** elde ederiz. **a=(3 + 5 + 7 + 10) / 4 = 25 / 4 = 6**,
b = (5 + 7) / 2 = 12 / 2 = 6 olacağından anahtar değeri, **6 * 6 = 36** bulunur.

'3-10-7' dizisinde ise tek sayıda eleman vardır. Bunları küçükten büyüğe doğru sıraladığımızda **'3-7-10'** elde ederiz. **a=(3 + 7 + 10) / 3 = 20 / 3 = 6**, **b = 7** olacağından anahtar değeri, **6 * 7 = 42** olarak bulunur. Örnek kullanım, aşağıdadır:

```
>>> dizi_anahtari('3 – 5 – 7 - 10')
36
>>> dizi_anahtari('3 – 10 – 7 - 5')
36
>>> dizi_anahtari('3 – 10 - 7')
42
>>> dizi_anahtari('7 – 3 - 10')
42
```

İpucu

'5' karakterini tamsayıya dönüştürmek için **int('5')** yazılabilir. **a='5'** için ise **int(a)** yazılmalıdır. **int**, bize dönüşümün sonucunu tamsayı olarak verir.

Bir **a** tamsayısının tek mi çift mi olduğunu öğrenmek için **(a/2)** ile **(a/2.0)** ifadelerinin eşit olup - olmadığına bakılır. Bu ikisi eşit ise sayı çifttir, değilse sayı tektir.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.

Alıştırma – 4

Ön Bilgi

11 haneden oluşan TC kimlik numaraları, aslında belirli bir kurala göre oluşturulmuştur, şöyle ki:

- Sol baştan 1., 3., 5., 7. ve 9. hanelerin toplamının **7 katından**, sol baştan 2., 4., 6. ve 8. hanelerin toplamını çıkardığımızda elde edeceğimiz sayının son basamağı ("mod 10" işlemine göre değeri) bize TC kimlik numarasının soldan 10. hanesini vermelidir.
- Sol baştan ilk 10 hanenin toplamının son basamağı ("mod 10" işlemine göre değeri) ise, bize TC kimlik numarasının soldan 11. hanesini vermelidir.

Yukarıdaki iki kural sağlanıyorsa TC kimlik numarası geçerli, diğer durumda ise geçersizdir.

Örnek verecek olursak Atatürk' ün TC kimlik numarası, 10000000146 olarak belirlenmiştir (Aynı zamanda ilk sıradaki TC kimlik numarasıdır.). Bu numaranın sağlamasını yapalım:

10000000146 sayısının 1., 3., 5., 7. ve 9. hanelerin toplamı:
1 + 0 + 0 + 0 + 1 = 2 *(1. toplam)*

10000000146 sayısının 2., 4., 6. ve 8. hanelerin toplamı:
0 + 0 + 0 + 0 = 0 *(2. toplam)*

1. toplamın 7 katından 2. toplamı çıkarıp son basamağını alırsak:
[(2 * 7) - 0] mod 10 = 14 mod 10 = 4

buluruz; bu bize soldan 10. haneyi vermelidir. 10000000146 sayısının soldan 10. hanesinde **4** rakamı bulunduğu için bu koşul sağlanmıştır.

10000000146 sayısının sol baştan ilk 10 hanenin toplamının son basamağını alırsak:

[1 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 1 + 4] mod 10 = 6 mod 10 = 6

buluruz; bu da bize soldan 11. haneyi vermelidir. 10000000146 sayısının soldan 11. hanesinde **6** rakamı bulunduğu için bu koşul sağlanmıştır. **Bu durumda TC kimlik numarasının geçerli olduğuna hükmedebiliriz.**

Görev

Sizden, "**tc_no_dogrula**" isminde, argüman olarak 11 haneli (0 ile başlamayan) pozitif bir tamsayı alan (Bu sayı, TC kimlik numarasıdır.) ve yukarıda verilen bilgiler doğrultusunda bu sayının geçerli bir TC kimlik numarası olup-olmadığını kontrol ederek geçerli ise ekrana "**TC kimlik numarasi gecerlidir.**", geçerli değilse de ekrana "**TC kimlik numarasi gecerli degildir.**" yazdıran bir fonksiyon hazırlayarak, "**lab08_tc_no_dogrula.py**" isimli betik dosyasına kaydetmeniz beklenmektedir.

Fonksiyonunuza 11 haneli pozitif tamsayı girilmediği zaman (karakter dizisi, farklı sayıda basamağa sahip tamsayı vs.) uyarı mesajı vermelidir. Programın çalıştırılmasına dair örnek ekran görüntüleri aşağıdadır:

```
>>> tc_no_dogrula(10000000146)
TC kimlik numarasi gecerlidir.
>>> tc_no_dogrula(10000000135)
TC kimlik numarasi gecerli degildir.
>>> tc_no_dogrula(-10000000146)
Lutfen 11 basamakli bir tamsayi giriniz!
>>> tc_no_dogrula(-1000000014)
Lutfen 11 basamakli bir tamsayi giriniz!
>>> tc_no_dogrula(10146)
Lutfen 11 basamakli bir tamsayi giriniz!
>>> tc_no_dogrula('10000000146')
Lutfen 11 basamakli bir tamsayi giriniz!
>>> tc_no_dogrula(10000000.146)
Lutfen 11 basamakli bir tamsayi giriniz!
```

Fonksiyonunuzu, kendi TC kimlik numaranızı kullanarak da test ediniz.

İpucu

Python' da tamsayıların **int** veri türü ile ifade edildiklerini öğrenmiştik. Python' da **-2147483648** ile **2147483647** sayıları arasında kalan sayılar (bu sayılar da dahil olmak üzere) **int** veri türündendir (Sayıların işâretlerinin (negatiflik-pozitiflik) de saklanabildiği ikilik gösterimde 32 bit ile ifade edilebilirler.). Ancak, **-2147483648**

sayısından daha küçük ve **2147483647** sayısından daha büyük sayıları ikilik gösterimde ifade etmek için 32 bit yeterli olmayacağından bu tür sayılar **long** isminde başka bir türe ait olmaktadırlar. Aşağıdaki örneği inceleyelim:

```
>>> type(1358)
<type 'int'>
>>> type(-2147483647)
<type 'int'>
>>> type(2147483647)
<type 'int'>
>>> type(2147483648)
<type 'long'>
>>> type(10000000146)
<type 'long'>
```

Bir sayının **long** türünden olup-olmadığını kontrol etmeniz gerekirse aşağıdaki yöntemi kullanmak oldukça pratik olacaktır:

```
>>> tc_no = 10000000146
>>> type(tc_no) == type(2147483648) # long turunden mi?
True
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri raporunuza yazınız.