Ardışık Veri Tipleri

Prof.Dr. Bahadır AKTUĞ
JFM212 Python ile Mühendislik Uygulamaları

*Kaynakça bölümünde verilen kaynaklardan derlenmiştir.

Ardışık Veri Tipleri

- Ardışık veri tipleri, Python programlarında önemli bir yer tutmaktadır
- Python 6 adet ardışık veri tipi bulunmaktadır:
  - strings
  - byte sequence
  - byte arrays
  - list
  - tuple
  - range nesnesi
- Bu ardışık tipleri farklı gibi görünse de, önemli bir ortak yönleri bulunmaktadır: Bu veri tiplerinde elemanlar ardışık olarak bulunmaktadır.

Ardışık Veri Tipleri

- Ardışık veri tiplerinin elemanlarına indisler ile ulaşılabilir.
- Metin tipi değişkeninin elemanlarına indisler ile erişim:

```
>>> metin = "Python ile Programlama"
>>> print(metin[0], metin[11])
PP
```

- Liste tipi değişkeninin elemanlarına indisler ile erişim:

```
>>> l = ["Ankara", "İstanbul", "İzmir", "Adana"]
>>> print(l[1], l[2])
İstanbul İzmir
```

Ardışık Veri Tipleri

- Python'da veri tipi farklı olsa da (string, list, tuple vb.), ardışık veri tipleri üzerindeki fonksiyonlar ortakdır.
- Örneğin, bir ardışık veri tipinin uzunluğu "len()" fonksiyonu ile bulunabilir:

```
>>> metin = "Python ile Programlama"
>>> l = ["Ankara", "İstanbul", "İzmir", "Adana"]
>>> print(len(metin),len(l))
22 4
```

Listeler

- Listeler genel olarak C, C++ veya Java gibi programlama dillerindeki "dizi"lere benzetilebilir.
- Bununla birlikte, Python listeleri klasik dizilere göre çok daha esnek ve güçlüdür.
- Öncelikle, bir "liste" içindeki elemanlar aynı türden (integer, string, float vb.) olmak zorunda değildir.
- Listeler çalışma zamanında genişletilebilir, daraltılabilir. Klasik "dizi (array)"lerde boyut derleme zamanında sabittir.
- Python'da listeler, sıralı nesneler dizisidir. Bu nesneler, farklı türde olabildiği gibi bazı liste'ler de olabilir.

Listeler

- Python'da listelerin genel özellikleri:
  - Elemanlar sıralı olarak bulunur
  - Elemanlar herhangi bir nesne türünde olabilir
  - Liste elemanlarına indisler ile erişilir
  - Başka listeler, başka listeler içeren başka listeler de elemanlar olabilir
  - Boyutları sabit değildir
  - Değiştirilebilir (mutable) türdendirler

Listeler

Python'da listelerin gösterimi için bazı örnekler:

Gösterim	Açıklama
[]	Boş liste
[1,1,2,3,5,8]	Tamsayılardan oluşan bir liste
[42, "JFM212", 3.1415]	Farklı veri tiplerinden oluşan bir liste
["Ankara", "Adana", "Bursa", "İzmir", "Gaziantep", "Antalya","Konya", "Samsun"]	Metin tipi veri tiplerinden oluşan bir liste
[["Ankara","Konya", 7556900], ["New York","Londra",2193031], ["Antalya", "Samsun", 123466]]	Eleman olarak liste(ler) içeren bir liste
["İller", ["ilçeler", ["beldeler", ["köyler", "mezralar", 1021]]]]	Birden fazla iç içe liste içeren bir liste

Listeler

- Listenin elemanlarına ve alt elemanlarına erişim:
  - Elemanlara indisler ile erişilir.
  - Erişilen elemanın kendisi de bir liste ise yine indis kullanılabilir.

```
>>> bilgi = [["Ali","Demir"],[[["Atatürk Cad.", "24"],
"06100"],"Ankara"]]
>>> print(bilgi[0])
['Ali','Demir']
>>> print(bilgi[0][1])
Demir
>>>  print(bilgi[1][0][1])
06100
```

Demetler (Tuples)

- Bir "demet" değiştirilemez (immutable) türde bir listedir.
- Demetler listelere benzer şekilde tanımlanırlar ancak "[]" yerine "()" kullanılır
- Elemanlar erişim aynı listelerdeki gibi yapılır.
- Liste yerine demet kullanmanın avantajları:
  - Demetler genel olarak listelere göre daha hızlıdır
  - Değiştirilemez olmalı programlama hatalarını azaltır
  - Demetler, listelerin aksine, daha sonra göreceğimiz "sözlük" veri tiplerinde "anahtar" olarak kullanılabilir.

Demetler (Tuples)

- Demetin elemanları değiştirilemez
- Dilimleme (slicing) işlemi aynı listelerdeki gibi yapılır

```
>>> t = ("Listeler", "ve", "demetler")
>>> t[0]
'Listeler'

>>> t[1:3]
('ve', 'demetler')

>>> t[0]="yeni değer"
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
```

Ardışık Veri Tiplerinde Birleştirme (concatenation) ve Tekrarlama (repetition)

- Python'da ardışık veri tipleri tıpkı metin tipi değişkenlerde (string) olduğu gibi "+" operatörü ile birleştirilebilirler:

```
>>> a = [1,2,5,4]
>>> b = [8,14,9]
>>> c = [45,10,6]
>>> a + b + c
[1, 2, 5, 4, 8, 14, 9, 45, 10, 6]
```

- Benzer şekilde "*" ile operatörü ile tekrarlama yapılabilir:

```
>>> a*4
[1, 2, 5, 4, 1, 2, 5, 4, 1, 2, 5, 4, 1, 2, 5, 4]
```

Ardışık Veri Tiplerinde Eleman Kontrolü

- Herhangi bir elemanın bir ardışık veri tipinde olup olmadığını anlamak için "in" operatörü kullanılır:

```
>>> a = [1,2,5,4]
>>> 2 in a
True
>>> 7 not in a
True
>>> b = ("Ankara","İzmir","İstanbul")
>>> 'Ankara' in b
True
>>> 'Adana' not in b
True
```

Ardışık Veri Tiplerindek Kopyalama Sığ/derin Kopyalama (shallow copy) ve Derin (deep copy)

- **Python'da bir değişkene yeni bir değer atandığında, değişkeninin daha önce sahip olduğu adresteki veriler değiştirilmek yerine yeni bir adres verilir.**

```
>>> x = 3
>>> y = x
>>> print(id(x), id(y))
1616756784 1616756784
>>> y = 4
>>> print(id(x), id(y))
1616756784 1616756800
>>> print(x,y)
3 4
```

Ardışık Veri Tiplerindek Kopyalama Sığ/derin Kopyalama (shallow copy) ve Derin (deep copy)

- Bu durumda ardışık veri tiplerinde de geçerlidir.

```
>>> renkler = ["kırmızı","mavi","yeşil"]
>>> renkler2 = renkler
>>> print(id(renkler),id(renkler2))
4317096 4317096
>>> renkler2 = ["turuncu","kahverengi"]
>>> print(renkler,renkler2)
['kırmızı', 'mavi', 'yeşil'] ['turuncu', 'kahverengi']
>>> print(id(renkler),id(renkler2))
4317096 33918808
```

Hafızadaki yeni adres alıyor!

Ardışık Veri Tiplerindek Kopyalama Sığ/derin Kopyalama (shallow copy) ve Derin (deep copy)

- Ancak, yeni bir atama yapmadan ardışık veri tipinin bir elemanını değiştirdiğimizde önceki ardışık veri değişkeninin de içeriği değiştirilir.

```
>>> renkler = ["kırmızı","mavi","yeşil"]
>>> renkler2 = renkler
>>> print(id(renkler),id(renkler2))
4317096 4317096
>>> renkler2[0] = "turuncu"
>>> print(renkler,renkler2)
['turuncu', 'mavi', 'yeşil'] ['turuncu', 'mavi', 'yeşil']
>>> print(id(renkler),id(renkler2))
4317096 4317096
```

renkler değişkenin İçeriği de değişiyor!

Hafızadaki yerleri aynı!

Ardışık Veri Tiplerindek Kopyalama Sığ/derin Kopyalama (shallow copy) ve Derin (deep copy)

- Bu nedenlerle, ardışık tipteki bir değişkeni kopyalamak özel yöntemler kullanmak gerekir. Bunlardan biri de "**sığ kopyalama**" dır:

```
>>> renkler = ["kırmızı","mavi","yeşil"]
>>> renkler2 = renkler[:]
>>> print(id(renkler),id(renkler2))
2678696 10456760
>>> renkler2[0] = "turuncu"
>>> print(renkler,renkler2)
['kırmızı', 'mavi', 'yeşil'] ['turuncu', 'mavi', 'yeşil']
```

Dilimleme Operatörü! (sığ kopyalama)

aynı adrese sahip iki değişken yerine yeni adres veriliyor!

Ardışık Veri Tiplerindek Kopyalama Sığ/derin Kopyalama (shallow copy) ve Derin (deep copy)

- Ardışık tipteki değişken içerisinde yine ardışık tipte bir değişken varsa, bu durumda (:) operatörü de istenilen kopyalamayı yapamaz:

```
>>> renkler = [["kırmızı","mavi"],"yeşil"]
>>> renkler2 = renkler[:]
>>> print(id(renkler),id(renkler2))
10473840 2678696
>>> renkler2[0][0] = "turuncu"
>>> print(renkler,renkler2)
[['turuncu', 'mavi'], 'yeşil'] [['turuncu', 'mavi'], 'yeşil']
```

Farklı adresler veriliyor!

renkler değişkeninin elemanı değişiyor!

Ardışık Veri Tiplerindek Kopyalama Sığ/derin Kopyalama (shallow copy) ve Derin (deep copy)

- Ardışık tipteki değişken içerisinde yine ardışık tipte bir değişken varsa bu durumda kopyalama işlemi için **derin kopyalama** yapmak gerekir.
- Derin kopyalamada copy modülünden deepcopy fonksiyonu kullanılır.

```
>>> from copy import deepcopy
>>> renkler = [["kırmızı","mavi"],"yeşil"]
>>> renkler2 = deepcopy(renkler)
>>> renkler2[0][0] = "turuncu"
>>> print(renkler,renkler2)
[['kırmızı', 'mavi'], 'yeşil'] [['turuncu', 'mavi'], 'yeşil']
```

Kaynakça

1 Wentworth, P., Elkner, J., Downey, A.B., Meyers, C. (2014). How to Think Like a Computer Scientist: Learning with Python (3nd edition).
2 Pilgrim, M. (2014). Dive into Python 3 by. Free online version: DiveIntoPython3.org ISBN: 978-1430224150.
3 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
4 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
5 Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/.
6 Millman, K.J., Aivazis, M. (2011). Python for Scientists and Engineers, Computing in Science & Engineering, 13, 9-12.
7 John D. Hunter (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95.
8 Travis E. Oliphant (2007). Python for Scientific Computing, Computing in Science & Engineering, 9, 10-20.
9 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
10 http://www.diveintopython.net/
11 https://docs.python.org/3/tutorial/
12 http://www.python-course.eu
13 https://developers.google.com/edu/python/
14 http://learnpythonthehardway.org/book/