Veri Bilimi için Python Programlama

Merih Palandöken

Elektrik Elektronik Mühendisliği Bölümü

İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ

TÜBİTAK

4005 TÜBİTAK

Sunum İçeriği

- Temel Veri Tipleri (Karakter Dizileri, Listeler, Demetler, Sözlükler)
- Akışın denetlenmesi (for döngüsü, while döngüsü, range fonksiyonu if, if-else, if-elif-else)
- Operatörler (Aritmetik İşlem Operatörleri, İkili İşlem Operatörleri, Kıyaslama Operatörleri, Atama Operatörleri)
- Fonksiyonlar

String - 1

- Diziler, tek veya çift tırnak işaretleri ile eşleştirilerek sınırlandırılabilir.

```
single_quoted_string = 'data science'
double_quoted_string = "data science"
escaped_string = 'Isn\'t this fun'
another_string = "Isn't this fun"

real_long_string = 'this is a really long string. \
It has multiple parts, \
but all in one line.'
```

- Çok satırlı dizeler için üçlü tırnak kullanın

```
multi_line_string = """This is the first line.
and this is the second line
and this is the third line"""
```

String - 2

- Ters eğik çizgi çıktısı almak için ham dizileri kullanın

```
tab_string = "\t" # sekme karakterini temsil eder.
len(tab_string) # 1

not_tab_string = r"\t" # '\' ve 't' karakterlerini temsil eder
len(not_tab_string) # 2
```

- Diziler + operatörüyle birleştirilebilir (birbirine yapıştırılabilir) ve * ile tekrarlanabilir

```
s = 3 * 'un' + 'ium' # s : 'ununununium'
```

String -2

- Yan yana iki veya daha fazla dizi değişmezi (yani tırnak içine alınmış olanlar) otomatik olarak birleştirilir:

```
s1 = 'Py' 'thon'
s2 = s1 + '2.7'

real_long_string = ('this is a really long string. '
'It has multiple parts, '
'but all in one line. ')
```

Listeler -1

```
integer_list = [1, 2, 3]
heterogeneous_list = ["string", 0.1, True]
list_of_lists = [ integer_list, heterogeneous_list, [] ]
list_length = len(integer_list) # 3
list_sum = sum(integer_list) # 6
```

- Bir listenin i-inci öğesini alın:

```
x = [i for i in range(10)] # [0, 1, ..., 9]
zero = x[0] # 0'a eşittir, listeler 0 dizinlidir
one = x[1] # 1
nine = x[-1] # son eleman
eight = x[-2] # 8
```

Listeler -1

- Listeden bir dilim alın:

```
one_to_four = x[1:5] # [1, 2, 3, 4]
first_three = x[:3] # [0, 1, 2]
last_three = x[-3:] # [7, 8, 9]
three_to_end = x[3:] # [3, 4, ..., 9]
without_first_and_last = x[1:-1] # [1, 2, ..., 8]
copy_of_x = x[:] # [0, 1, 2, ..., 9]
another_copy_of_x = x[:3] + x[3:] # [0, 1, 2, ..., 9]
```

Listeler -2

- Üyelik kontrolü:

```
1 in [1, 2, 3] # True
0 in [1, 2, 3] # False
```

- Listeleri birleştir:

```
x = [1, 2, 3]
y = [4, 5, 6]
x.extend(y) # x şimdi [1,2,3,4,5,6]

x = [1, 2, 3]
y = [4, 5, 6]
z = x + y  # z : [1,2,3,4,5,6]; x değişmedi.
```

Listeler -2

- Liste açma (Çoklu atama)

```
x, y = [1, 2] # x 1 ve y 2
[x, y] = 1, 2 # yukarıdakiyle aynı
x, y = [1, 2] # yukarıdakiyle aynı
x, y = 1, 2   # yukarıdakiyle aynı
_, y = [1, 2] # y 2, ilk elemanı umursamadı!
```

Listeler -3

- Listenin içeriğini değiştirme:

```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8]
x[2] = x[2] * 2      # x [0, 1, 4, 3, 4, 5, 6, 7, 8]
x[-1] = 0     # x [0, 1, 4, 3, 4, 5, 6, 7, 0]
x[3:5] = x[3:5] * 3 # x [0, 1, 4, 3, 4, 3, 4, 3, 4, 5, 6, 7, 0]
x[5:6] = []   # x [0, 1, 4, 3, 4, 4, 3, 4, 5, 6, 7, 0]
del x[:2]     # x [4, 3, 4, 4, 3, 4, 5, 6, 7, 0]
del x[:]      # x []
del x         # bundan sonra x'e atıfta bulunmak bir NameError
```

- Karakter dizileri de dilimlenebilir, ama değiştirilemezler!

```
s = 'abcdefg'
a = s[0]      # 'a'
x = s[:2]     # 'ab'
y = s[-3:]    # 'efg'
s[:2] = 'AB'  # hataya yol açar
s = 'AB' + s[2:]    # str artık ABcdefg
```

Range() fonksiyonu

```
for i in range(5):
        print (i) # 0, 1, 2, 3, 4 (ayrı satırlarda) yazdırılacak
for i in range(2, 5):
        print (i) # 2, 3, 4 yazdırılacak
for i in range(0, 10, 2):
        print (i)  # 0, 2, 4, 6, 8 yazdırılacak
for i in range(10, 2, -2):
        print (i)  # 10, 8, 6, 4 yazdırılacak
```

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
2 a
3 little
4 lamb
```

Range() in Python 2 and 3

- Python 2'de range(5), [0, 1, 2, 3, 4]'e eşittir
- Python 3'te range(5) yinelenebilen, ancak [0, 1, 2, 3, 4] ile aynı olmayan bir nesnedir

```
print (range(3)) # python 3'te "range(0, 3)" ifadesi görülecek

print (range(3)) # python 2'de "[0, 1, 2]"
print (list(range(3))) # Python 3'te [0, 1, 2] yazdırılacak

x = range(5)
print (x[2])  # python 2'de «2» yazdırılacak
print (x[2])  # python 3'te de «2» yazdırılacak

x[2] = 5      # Python 2: [0, 1, 5, 3, 4, 5]
x[2] = 5      # Python 3: Hata!
```

İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ

TÜBİTAK

4005 TÜBİTAK

Listelere Erişim

Aşağıdaki kod için beklenen çıktı nedir?

```
a = list(range(10))
b = a
b[0] = 100
print(a)

a = list(range(10))
b = a[:]
b[0] = 100
print(a)
```

Tuples -1(Demetler)

- Listelere benzer, ancak değiştirilemez

```
a_tuple = (0, 1, 2, 3, 4)
Other_tuple = 3, 4
Another_tuple = tuple([0, 1, 2, 3, 4])
Hetergeneous_tuple = ('john', 1.1, [1, 2])
```

- Dilimlenebilir, birleştirilebilir veya tekrarlanabilir

```
a_tuple[2:4]    #  (2, 3)
```

- Değiştirilemez!

```
a_tuple[2] = 5
```

TypeError: 'tuple' nesnesi öğe atamasını desteklemiyor

Tuples -2

- Fonksiyonlardan birden çok değer döndürmek için kullanışlıdır
- Tuple'lar ve listeler ayrıca birden fazla atama için kullanılabilir

```
def sum_and_product(x, y):
      return (x + y),(x * y)
sp = sum_and_product(2, 3)     # equals (5, 6)
s, p = sum_and_product(5, 10) # s is 15, p is 50


x, y = 1, 2
[x, y] = [1, 2]
(x, y) = (1, 2)
x, y = y, x
```

Dictionaries -1 (Sözlükler)

- Bir sözlük, değerleri benzersiz anahtarlarla ilişkilendirir

```
empty_dict = {}                          # Pythonic
empty_dict2 = dict()                     # daha az Pythonic
grades = { "Joel" : 80, "Tim" : 95 }
```

- Anahtarla değere erişin/değiştirin

```
joels_grade = grades["Joel"]             # 80
grades["Tim"] = 99                       # eski değeri değiştirir
grades["Kate"] = 100                     # 3. bir içerik ekler
num_students = len(grades)               # 3'e eşittir
```

```
try:
        kates_grade = grades["Kate"]
except KeyError:
        print "no grade for Kate!"
```

İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ

TÜBİTAK

4005 TÜBİTAK

Dictionaries -2 (Sözlükler)

- Anahtarın varlığını kontrol edin

```
joel_has_grade = "Joel" in grades      # True
mary_has_grade = "Mary" in grades      # False
```

- keyError'dan kaçınmak ve varsayılan değer eklemek için "get" kullanın

```
joels_grade = grades.get("Joel", 0)    # 80'e eşit
mary_grade = grades.get("Mary", 0)    # 0'a eşit
no_ones_grade = grades.get("No One")  # default
default is None
```

Dictionaries -3 (Sözlükler)

- Tüm ögeleri alma:

```
all_keys = grades.keys()   # tüm anahtarların bir listesini
döndür
all_values = grades.values() # tüm değerlerin bir listesini
döndür
all_pairs = grades.items() # (anahtar, değer) demetlerinin
bir listesi
```

Kontrol Akışı -1

- if-else

```
if 1 > 2:
    message = "if only 1 were greater than two..."
elif 1 > 3:
    message = "elif stands for 'else if'"
else:
    message = "when all else fails use else (if you want to)"
print (message)
parity = "even" if x % 2 == 0 else "odd"
```

- python 2 ve python3 arasındaki fark; Python 2'de print bir ifadedir
- print(mesaj) ve print mesajının ikisi de geçerlidir
- Python 3'te print bir fonksiyondur. Yalnızca print(mesaj) geçerlidir

Karşılaştırma Operatörleri

Operasyon	Anlamı
<	küçüktür
<=	Küçük eşittir
>	büyüktür
>=	Büyük eşittir
==	eşittir
!=	Eşit değildir
is	Nesne özdeşliği
is not	Olumsuz nesne özdeşliği

```
a = [0, 1, 2, 3, 4]
b = a
c = a[:]

a == b
Out[129]: True

a is b
Out[130]: True

a == c
Out[132]: True

a is c
Out[133]: False
```

Bit değerleri üzerine işlem yapan operatörler: & (AND), | (OR), ^ (XOR), ~(NOT), << (Left Shift), >> (Right Shift)

Kontrol Akışı- 2

- Döngüler

```
x = 0
while x < 10:
        print (x, "is less than 10 ")
        x += 1

for x in range(10):
        pass

for x in range(10):
        if x == 3:
                continue # hemen sonraki yinelemeye git
        if x == 5:
                break # döngüden tamamen çık
        print (x)
```

Döngülerde pass anahtar sözcüğü:
Hiçbir şey yapmaz, boş ifade yer tutucusu

İstisna Durumları

```
try:
      print (0 / 0)
except ZeroDivisionError:
      print ("cannot divide by zero")
```

Fonksiyonlar - 1

- *def* kullanılarak tanımlanır:

```
def double(x):
    """Bu, isteğe bağlı bir belge dizisini koyduğunuz
yerdir. Bu, işlevin ne yaptığını açıklar.
     örneğin, bu işlev girdisini 2""" ile çarpar.
return x * 2
```

- Tanımlandıktan sonra bir fonksiyonu çağırabilirsiniz.

```
z = double(10) # z : 20
```

- Parametrelere varsayılan değerler verebilirsiniz

```
def my_print(message="my default message"):
      print (message)

my_print("hello") # 'hello' yazdırır
my_print() # 'my default message' yazdırır
```

İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ

TÜBİTAK

4005 TÜBİTAK

Fonksiyonlar - 2

- Bazen argümanları ada göre belirtmek yararlıdır.

```
def subtract(a=0, b=0):
        return a - b

subtract(10, 5) # 5 döndürür
subtract(0, 5) # -5 döndürür
subtract(b = 5) # yukarıdakiyle aynı
subtract(b = 5, a = 0) # yukarıdakiyle aynı
```

Fonksiyonlar - 3

```
In [12]: def double(x): return x * 2
    ...: DD = double;
    ...: DD(2)
    ...:
Out[12]: 4
```

```
In [16]: def apply_to_one(f):
    ...: return f(1)
    ...: x=apply_to_one(DD)
    ...: x
    ...:
Out[16]: 2
```

İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ

TÜBİTAK

4005 TÜBİTAK

Fonksiyonlar – lambda ifadeleri

- **Lambda** anahtar sözcüğü ile küçük anonim **fonksiyonlar** oluşturulabilir.

```
In [18]: y=apply_to_one(lambda x: x+4)

In [19]: y
Out[19]: 5
```

```
In [104]: def small_func(x): return x+4
     ...: apply_to_one(small_func)
Out[104]: 5
```

lambda ifadeleri - 2

```
In [22]: pairs = [(2, 'two'), (3, 'three'), (1, 'one'), (4, 'four')]
    ...: pairs.sort(key=lambda pair: pair[0])
    ...: pairs
Out[22]: [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
```

```
In [107]: def getKey(pair): return pair[0]
    ...: pairs.sort(key=getKey)
    ...: pairs
Out[107]: [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')
```

Teşekkürler

İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ

TÜBİTAK

4005 TÜBİTAK