Python 3: Control Flow

금융공학 프로그래밍 I

Iterating

- Data
  - us_cities.txt

```
new york: 8244910
los angeles: 3819702
chicago: 2707120
houston: 2145146
philadelphia: 1536471
phoenix: 1469471
san antonio: 1359758
san diego: 1326179
dallas: 1223229
```

```
1  data_file = open('us_cities.txt', 'r')
2  for line in data_file:
3      city, population = line.split(':')              # Tuple unpacking
4      city = city.title()                             # Capitalize city names
5      population = '{0:,}'.format(int(population))    # Add commas to numbers
6      print(city.ljust(15) + population)
7  data_file.close()
```

  - Looping without explicit indexing

Looping without Indices

```
for x in x_values:
    print(x * x)
```

is preferred to

```
for i in range(len(x_values)):
    print(x_values[i] * x_values[i])
```

- range

```
In [352]: range(10)
Out[352]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [353]: range(0, 20, 2)
Out[353]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

for statement example

```
In [455]: words = ['apple', 'bat', 'bar', 'atom', 'book']

In [456]: by_letter = {}

In [457]: for word in words:
   .....:     letter = word[0]
   .....:     if letter not in by_letter:
   .....:         by_letter[letter] = [word]
   .....:     else:
   .....:         by_letter[letter].append(word)
   .....:

In [458]: by_letter
Out[458]: {'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

zip

- Pairs from two sequences

```
countries = ('Japan', 'Korea', 'China')
cities = ('Tokyo', 'Seoul', 'Beijing')
for country, city in zip(countries, cities):
    print('The capital of {0} is {1}'.format(country, city))
```

- Also useful for creating dict

```
In [1]: names = ['Tom', 'John']

In [2]: marks = ['E', 'F']

In [3]: dict(zip(names, marks))
Out[3]: {'John': 'F', 'Tom': 'E'}
```

enumerate

- If we actually need the index from a list,

```
letter_list = ['a', 'b', 'c']
for index, letter in enumerate(letter_list):
    print("letter_list[{0}] = '{1}'".format(index, letter))
```

The output of the loop is

```
letter_list[0] = 'a'
letter_list[1] = 'b'
letter_list[2] = 'c'
```

Iterator

- Iterator: an object with a next() method

```
In [40]: f = open('us_cities.txt', 'r')

In [41]: next(f)
Out[41]: 'new york: 8244910\n'

In [42]: next(f)
Out[42]: 'los angeles: 3819702\n'
```

```
f = open('somefile.txt', 'r')
for line in f:
    # do something
```

```
In [43]: e = enumerate(['foo', 'bar'])

In [44]: next(e)
Out[44]: (0, 'foo')

In [45]: next(e)
Out[45]: (1, 'bar')
```

Itrerable

- List is not a iterator

```
for i in range(2):
    print('foo')
```


```
In [59]: x = ['foo', 'bar']

In [60]: type(x)
Out[60]: list

In [61]: y = iter(x)

In [62]: type(y)
Out[62]: listiterator

In [63]: next(y)
Out[63]: 'foo'

In [64]: next(y)
Out[64]: 'bar'
```

Other loops

- While

```
In [57]: total = 0
         while total < 100:
             total += 1
         print total

Out[57]: 100
```

```
x = 256
total = 0
while x > 0:
    if total > 500:
        break
    total += x
    x = x // 2
```

- List comprehensions

```
In [58]: m = [i ** 2 for i in range(5)]
         m

Out[58]: [0, 1, 4, 9, 16]
```

Comprehensions

- Lists

```
In [477]: strings = ['a', 'as', 'bat', 'car', 'dove', 'python']

In [478]: [x.upper() for x in strings if len(x) > 2]
Out[478]: ['BAT', 'CAR', 'DOVE', 'PYTHON']
```

- Sets

```
In [479]: unique_lengths = {len(x) for x in strings}

In [480]: unique_lengths
Out[480]: set([1, 2, 3, 4, 6])
```

- Dicts

```
In [481]: loc_mapping = {val : index for index, val in enumerate(strings)}

In [482]: loc_mapping
Out[482]: {'a': 0, 'as': 1, 'bat': 2, 'car': 3, 'dove': 4, 'python': 5}
```

continue & break

- continue

```
sequence = [1, 2, None, 4, None, 5]
total = 0
for value in sequence:
    if value is None:
        continue
    total += value
```

- break

```
sequence = [1, 2, 0, 4, 6, 5, 2, 1]
total_until_5 = 0
for value in sequence:
    if value == 5:
        break
    total_until_5 += value
```

Comparison

```
In [49]: x = 1    # Assignment

In [50]: x == 2   # Comparison
Out[50]: False
```

```
In [52]: x = 'yes' if 42 else 'no'

In [53]: x
Out[53]: 'yes'

In [54]: x = 'yes' if [] else 'no'

In [55]: x
Out[55]: 'no'
```

```
In [56]: 1 < 2 and 'f' in 'foo'
Out[56]: True

In [57]: 1 < 2 and 'g' in 'foo'
Out[57]: False

In [58]: 1 < 2 or 'g' in 'foo'
Out[58]: True

In [59]: not True
Out[59]: False

In [60]: not not True
Out[60]: True
```

Conditional execution

- if / elif / else

```
In [56]: for i in range(1, 10):
             if i % 2 == 0:  # % is for modulo
                 print "%d is even" % i
             elif i % 3 == 0:
                 print "%d is multiple of 3" % i
             else:
                 print "%d is odd" % i

Out[56]: 1 is odd
         2 is even
         3 is multiple of 3
         4 is even
         5 is odd
         6 is even
         7 is odd
         8 is even
         9 is multiple of 3
```

Ternary Expressions

```
value = true-expr if condition else false-expr
```

```
if condition:
    value = true-expr
else:
    value = false-expr
```

```
In [354]: x = 5

In [355]: 'Non-negative' if x >= 0 else 'Negative'
Out[355]: 'Non-negative'
```

Functions

- Python functions are very flexible
  - Any number of functions can be defined in a given file
  - Any object can be passed to a function as an argument, including other functions
  - Functions can be (and often are) defined inside other functions
  - A function can return any kind of object, including functions

Function definition

❖ Function definition

```
In [59]: def f(x):
             return x ** 2
         f(2)
Out[59]: 4
```

❖ map

```
In [60]: def even(x):
             return x % 2 == 0
         even(3)

Out[60]: False

In [61]: map(even, range(10))
Out[61]: [True, False, True, False, True, False, True, False, True, False]
```

pass

- if statement

```
if x < 0:
    print 'negative!'
elif x == 0:
    # TODO: put something smart here
    pass
else:
    print 'positive!'
```

- function

```
def f(x, y, z):
    # TODO: implement this function!
    pass
```

Lambda function

One-Line Functions: `lambda` The `lambda` keyword is used to create simple functions on one line

For example, the definitions

```
def f(x):
    return x**3
```

and

```
f = lambda x: x**3
```

$$\int_0^2 x^3 dx$$

```
In [68]: from scipy.integrate import quad

In [69]: quad(lambda x: x**3, 0, 2)
Out[69]: (4.0, 4.440892098500626e-14)
```

Lambda Function Example

❖ Summation function

```
In [65]: def cumsum(l):
             total = 0
             for elem in l:
                 total += elem
             return total
         cumsum(range(10))
Out[65]: 45
```

❖ reduce

- Python 3.5에서는 "from functools import reduce" 필요

```
In [64]: reduce(lambda x, y: x + y, range(10))
Out[64]: 45
```

Default argument value

```
def my_function(x, y, z=1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

```
def f(x, coefficients=(1, 1)):
    a, b = coefficients
    return a + b * x
```

```
In [71]: f(2, coefficients=(0, 0))
Out[71]: 0

In [72]: f(2)  # Use default values (1, 1)
Out[72]: 3
```

Returning multiple values

- Returning tuple

```
def f():
    a = 5
    b = 6
    c = 7
    return a, b, c

a, b, c = f()
```

- Returning dict

```
def f():
    a = 5
    b = 6
    c = 7
    return {'a' : a, 'b' : b, 'c' : c}
```

Recursive Function Calls

$$x_{t+1} = 2x_t, \quad x_0 = 1$$

Obviously the answer is 2^t

❖ Recursive Function Calls

```
def x(t):
    if t == 0:
        return 1
    else:
        return 2 * x(t-1)
```

Closures: functions that return functions

```
def make_watcher():
    have_seen = {}

    def has_been_seen(x):
        if x in have_seen:
            return True
        else:
            have_seen[x] = True
            return False

    return has_been_seen
```

```
In [496]: watcher = make_watcher()

In [497]: vals = [5, 6, 1, 5, 1, 6, 3, 5]

In [498]: [watcher(x) for x in vals]
Out[498]: [False, False, False, True, True, True, False, True]
```

Q & A

khhwang78@gmail.com