Python Essentials

Created by 이 윤 준 (yoonjoon.lee@gmail.com)

May, 2019

Data Types

Primitive Data Types

- Boolean
- Integer
- Real
- Complex
- String

Container

- tuple
- list
- set
- dictionary

Tuple

One-dimensional, fixed-length, immutable sequence of Python objects

```
>>> tuple = 4, 5, 6
>>> tuple
(4, 5, 6)
>>> a, b, c = tuple
>>> a
4
>>> tuple[1]
5
>>> tuple.append(7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

```
>>> tuple+(7, 8)
(4, 5, 6, 7, 8)
>>> tuple
(4, 5, 6)
```

List

- Variable-length
- Contents can be modified.
- Defined using square brackets [] or using the list type function.

```
>> a_list = [2, 3, 7, None]
>>> tup = ('foo', 'bar', 'bar')
>>> b_list = list(tup)
>>> b_list
['foo', 'bar', 'bar']
>>> b_list[1] = 'peekaboo'
>>> b_list
['foo', 'peekaboo', 'bar']
```

```
>>> b = ['saw', 'small', 'He', 'foxes', 'six']
>>> b.sort(); b
['He', 'foxes', 'saw', 'six', 'small']
```

```
>>> seq = [7, 2, 3, 7, 5, 6, 0, 1]
>>> seq[1:5]
[2, 3, 7, 5]
>>> seq[3:4] = [6,3]
>>> seq
[7, 2, 3, 6, 3, 5, 6, 0, 1]
```

Set

- An unordered collection of unique elements
- Created in two ways: via the set function or using a set literal with curly braces

```
>>> a = set([2, 2, 2, 1, 3, 3])
>>> a
set([1, 2, 3])
>>> b = {3, 4, 5, 6, 7, 8}
```

Dict

- hash map or associative array
- A flexibly-sized collection of key-value pairs, where key and value are Python objects

```
>>> d1 = {'a': 'some values', 'b':[1, 2, 3, 4]}
>>> d1
{'a': 'some values', 'b': [1, 2, 3, 4]}
>>> d1[7]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 7
>>> d1[7] = 'an integer'
>>> d1
{'a': 'some values', 'b': [1, 2, 3, 4], 7: 'an integer'}
```

```
>>> mapping = dict(zip(range(5), reversed(range(5)))); map
{0: 4, 1: 3, 2: 2, 3: 1, 4: 0}
```

List, Set, and Dict Comprehensions

List comprehensions are one of the most-loved Python language features. Concisely form a new list by filtering the elements of a collection and transforming the elements passing the filter in one concise expression.

```
>>> strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
>>> [x.upper() for x in strings if len(x) > 2]
['BAT', 'CAR', 'DOVE', 'PYTHON']
```

```
[expr for val in collection if condition]
{key-expr : value-expr for value in collection if conditio
{expr for value in collection if condition}
```

Nested list comprehensions

```
>>> all_data = [['Tom', 'Billy', 'Jefferson',
...              'Andrew', 'Wesley', 'Steven', 'Joe'],
...             ['Susie', 'Casey', 'Jill',
...              'Ana', 'Eva', 'Jennifer', 'Stephanie']]
>>> result = [name for names in all_data for name in names
...           if name.count('e') >= 2] ; result
['Jefferson', 'Wesley', 'Steven', 'Jennifer', 'Stephanie']
```

Import

부가 기능을 별도의 라이브러리 또는 모듈로 구성

```python
import math

math.sqrt(4)
```

Input and Output

```
f = open('newfile.txt', 'w')    # Open 'newfile.txt' for wr
f.write('Testing\n')            # Here '\n' means new line
f.write('Testing again')
f.close()
```

Loop

```
new york: 8244910

los angeles: 3819702

chicago: 2707120

houston: 2145146

philadelphia: 1536471

phoenix: 1469471

san antonio: 1359758

san diego: 1326179
```

```python
data_file = open('Data/us_cities.txt', 'r')
for line in data_file:
    city, population = line.split(':')            # Tuple
    city = city.title()                           # Capita
    population = '{0:,}'.format(int(population))  # Add co
    print(city.ljust(15) + population)
data_file.close()
```

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

Comparison and Logical Operators

Comparison

```
x = 1      # Assignment
x == 2     # Comparison
```

```
x = 'yes' if 42 else 'no'
```

Combining Expression

```
1 < 2 and 'f' in 'foo'
```

Functions

내장 함수 (embeded functions)

```
bools = False, True, True
all(bools)  # True if all are True and False otherwise
```

사용자 정의 함수 (user defined functions)

Why Function?

코드를 명쾌하게 작성하기 위하여 사용자 정의 함수 사용은

- 프로그램 로직의 다른 가닥과 분리하고,
- 코드 재사용 촉진한다

Python 함수는 아래와 같은 유연성을 갖고 있다.

- 동일 파일에서 필요한 함수들을 함께 정의하여 작성할 수 있다.
- 함수 내에서 다른 함수를 정의할 수 있다.
- 함수를 포함하여 어떤 객체도 인수로 사용할 수 있다.

Doctrings

Python은 함수와 모듈에 주석을 다는 docstring이라는 시스템을 갖추고 있다. docstring은 함수와 모듈을 실행할 때에도 이용할 수 있다.

```
# Filename: temp.py
def f(x):
    """
    This function squares its argument.
    """
    return x**2
```

```
f.__doc__
```

```
f?
```

One-Line Functions: `lambda`

한줄로 간단한 함수를 `lambda`를 사용하여 정의

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

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

Keywords Arguments

```
plt.plot(x, 'b-', label="white noise")
```

Coding Style & PEP8

```
import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
```

See Also