Introduction to Python and Programming Language

Koscom Algorithm Lecture

신승우

Tuesday 4th September, 2018

Outline

Hello World!

```
1 print('Hello World!')
```

세미콜론이 없음에 주의해주세요. C나 자바와는 다르게 세미콜론이 없습니다.

Indentation

파이썬 언어는 다른 언어에서의 괄호를 indent로 대신합니다. 표준적으로, 인덴트는 space 4개로 하며, 이를 위해서 대부분의 개발 도구에서는 tab을 space 4개로 대체하여 사용합니다. 만약 인터넷이나 다른 곳에서 소스를 사용할 때 indentation 관련 에러가 난다면, tab/space 여부를 체크해보세요.

= vs == vs is

```
1 a = 1
  a == 1
3 a is 1 # True or False?
```

- = : assignment
- == : equality check
- is : address check

if-elif-else

if-elif-else 구문에서는 항상 indent를 유지해야 합니다.

```
if cond1:
    .... # when cond1 is True
elif cond2:
    .... # when cond1 is False and cond2 is True
else:
    .... # when cond1 and cond2 is False
```

while

```
while(cond):
    ....  # while cond remains True
```

다른 대부분의 언어들과 비슷하게, break문을 통해서 벗어날 수 있습니다.

for loop

```
for i in range(10):
    ....
```

문법은 for *1 in *2: 형태를 취하며, 이 때

- *1 변수에 *2에 있는 원소를 *차례로* 대입하며,
- indent로 명시된 코드블럭 내 구문을 수행합니다.

즉, *2에 있는 부분은 순서가 있는 자료형이여야 합니다. 이에 대해서는 조금 후에 더 자세하게 다루겠습니다.

switch

파이썬에서는 switch문을 지원하지 않습니다. 하지만, 비슷하게 만드는 방법이 있습니다. 이에 대해서는 파이썬 사전을 배운 후 다루겠습니다.

assert

```
assert False
assert cond, ...
```

Boolean

많은 다른 언어에서 나오는 Boolean과 같으며, 다음의 operation들을 지원합니다.

```
assert (True and True) == True
assert (True and False) == False
assert (False and True) == False
assert (False and False) == False
assert (True or True) == True
assert (True or False) == True
assert (False or True) == True
assert (False or False) == False
assert (not True) == False
assert (not False) == True
```

Listing 1: Boolean Example (example_bool1.py)

Numeric

```
a = 1
print(type(a)) # <type 'int'>
b = 1.0
print(type(b)) # <type 'float'>
c = 1.0 + 1j # j 알파벳은 그냥 변수지만 , 숫자 뒤에 붙어 오면 허수로 인식합니다 .
print(type(c)) # <type 'complex'>
print('hello world!')
```

Listing 2: Numeric Types (example_numeric1.py)

list

list 데이터형은 다음의 연산들을 지원합니다.

- a in d : 배열(d) 안에 특정 원소(a)가 있는지를 검사합니다.
- $+$: 배열 두 개를 이어서 새로운 배열을 만듭니다. 같은 데이터형이여야 합니다.
- d[i] : d의 i번째 원소를 반환합니다.
- d[i:j:k] : d의 i번째 원소부터 j번째 원소까지, k번째 원소마다 선택하여 리스트를 만들어 반환합니다.
- d.index(elem) : d에서 elem 이 처음으로 나오는 위치를 반환합니다.
- d.append(elem) : elem을 d 맨 뒤에 추가합니다.

list

```
1 from example_sequence1 import *

3 print('e' in d) # True
  print([1,2,3] + [4,5,6]) # [1,2,3,4,5,6]
5 print(d[1]) # 'e'
  print(d[1:3]) # 'el'
7 print(d[1:6:2]) # 'el '
  print(d[::-1]) # !dlrow olleh'
9 print(len(d)) # 12
  for idx, elem in enumerate(d):
11     print(idx, elem)
```

Listing 3: Operations for Sequential Types (example_sequence2.py)

list vs iterator

Dictionary

- d[key] : dict에서 key에 해당되는 value를 반환합니다.
- d[key] = val : dict에서 key에 해당되는 value를 val로 업데이트합니다.
- d.keys() : dict의 key들을 반환합니다.

```
1 from example_dict1 import *

3 a[3] # 'c'
  a.keys() # [1,2,3,...,26]
5 len(a) # 26
  a[27]='A'
7 1 in a # True
  del d[1]
9 1 in a # False
```

Listing 4: Operations for Mapping Types (example_dict2.py)

Function Definition Syntax

```
1 def function_name(function_arguments):
      ...
```

- def : 함수 정의 키워드입니다.
- function : 함수 이름을 나타냅니다.
- args : 함수 인자입니다. 아래와 같은 옵션이 있습니다.
  - arg
  - arg_default : 함수 인자의 기본값을 정해줄 때, =을 이용하여 기본값을 지정해줄 수 있습니다.
  - *arg_list : 정해지지 않은 수의 인자를 받고자 할 때, *을 하나 붙여서 들어온 인자를 배열로 받을 수 있습니다.
  - **arg_dict : 정해지지 않은 수의 이름이 명시된 인자를 받고자 할 때, *를 두개 붙여서 들어온 인자들을 dict 형태로 받을 수 있습니다.
- return None : 함수의 결과값으로 return 뒤의 구문을 반환합니다.

실습 : sorting a list

정수의 리스트가 주어졌을 때, 이를 크기순으로 배열한 리스트를 반환하는 함수 sort_list를 작성하세요.

Lambda Function

```
func1 = lambda x: x+2

def func2(x):
    return x+2

func3 = lambda x,y,z : x+y+z
func4 = lambda *args : sum(args)
```

Listing 5: Lambda Function Example (example_lambda1.py)

파이썬에서는 익명함수(lambda function)을 지원합니다. 이 때 구문은 다음과 같습니다.

- lambda : 람다함수 키워드. 람다함수 뒤의 구문 중 콜론 전에 있는 구문은 함수의 인자를, 뒤는 반환하는 값을 나타낸다.
- x,y,z(func3)/*args(func4) : 람다함수의 인자. 쉼표로 구분되며, 상기된 *args등도 똑같이 사용 가능함을 func4에서 확인할 수 있다.
- x+y+z(func3)/sum(args)(func4) : 람다함수의 반환값.

Functional Python : Function as a First-Class Citizen

파이썬에서 함수는 다른 객체와 똑같이 취급됩니다. 즉,

- 다른 함수에 인자로 넘길 수 있으며, [1]
- 함수의 반환값이 될 수 있고,
- 변수에 저장될 수 있습니다.

이는 자바나 C에서는 없는 특성이기 때문에, 예시를 들어서 살펴보도록 하겠습니다.

[1] 합성함수와는 다릅니다!

실습 : sorting a list using custom key

이번에는 정수의 리스트를, 정수를 4로 나눈 나머지를 기준으로 배열해보겠습니다.

Class Syntax

```
1  class MyString:
       class_name = 'MyString'
3      maker = 'Schin'

5      def __init__(self, datum = ''):
           self.datum = datum
7
       # instance method
9      def get_first_letter(self):
           return self.datum[0]
11
       def most_frequent_word(self):
13         bow = MyString._create_bow(self.datum)
           res = ''
15         tmp = 0
```

Listing 6: Making a MyString Class (example_class2.py)

Reference/examples/example_class2.py 파일을 참고하세요.

위 코드에서 파이썬 클래스의 각 요소들을 설명하면 다음과 같습니다.

- class : 클래스를 정의하는 키워드입니다. 이 키워드 뒤의 단어가 클래스의 이름이 됩니다. 여기서는 MyString입니다.
- maker = 'Schin' : 클래스의 **속성**을 지정합니다. 여기서는 class_name, maker, datum의 3개의 속성이 있습니다. 편의상 datum을 우리가 다루고자 하는 문자열로 생각하겠습니다.
- a = MyString : a라는 변수의 값으로 MyString이라는 **클래스**를 지정합니다.
- b = MyString() : b라는 변수의 값으로 MyString이라는 **인스턴스**를 지정합니다.

Syntactic Sugar : Magic Methods

2 by 2 행렬 클래스를 만들어 봅시다. 이 때, 클래스의 두 인스턴스를 더하거나 빼는 메소드를 만들어 봅시다.
파이썬에서는 더하기, 빼기 등의 자주 쓰이는 operation들을 우리가 만든 클래스에서 편하게 커스터마이징하는 방법을 제공합니다. 이를 magic method라고 합니다. 많이 쓰이는 magic method와 대응되는 파이썬 문법은 다음과 같습니다.

- __cmp__ : 대소비교할 때 쓰입니다.
- __eq__ : == 에서 쓰입니다.
- __add__, __sub__, __mul__, __div__ : 사칙연산
- __iter__ : 인스턴스가 여러 값을 포함하고 있을 경우, 그 값들을 traverse 할 수 있도록 합니다. a in b에서 쓰입니다.
- __str__ : str(a)에서 쓰입니다.

Variables I

위 질문에 답하기 위해서는 먼저 변수의 bind-bound와 Namespace의 개념에 대해서 알아야 합니다. 일반적으로 변수에는 3가지 종류가 있습니다.

- Bound variable : 어떤 값이나 다른 변수에 의해서 값이 결정되는지 정해진 변수
- Binding variable : Bound variable의 값을 결정하는 변수
- Free variable : Bound되지 않은 변수

Namespace란 variable과 값의 bind-bound를 나타내는 구조입니다. 여기에 없는 variable에 대한 접근은 NameError을 반환합니다.

To Bind a Variable

파이썬에서 Variable을 bind하기 위해서는

- =을 이용한 assignment
- class/function 이름
- 루프에서 루프 카운터의 bind
- 함수 argument

등[2]이 있습니다.

[2]이 외에는 try-except나 with 구문이 있습니다.

Loading Namespaces

다른 코드 블럭의 Namespace를 가져오기 위해서는 2 가지 방법이 있습니다.

- .의 이용 : AAA.bb는 AAA 코드 블럭 내에서 bind된 bb를 찾는 구문입니다.
- from ... import * : ... 모듈이나 파일 안에서 정의된 모든 name을 bind 합니다.
- Syntactic Sugar : a.some_method()는 a의 클래스 정의부 내에서 bind 된 some_method를 찾습니다. 즉, 이는 A.some_method(a)와 같습니다.

Explicit Checking of Namespace : locals()

locals() 함수를 사용하여 특정 시점에서의 namespace를 체크할 수 있습니다. 어떠한 종류의 디버거던 사용해 보신 분들은 익숙한 개념일 것입니다.

Execution Flow of the Interpreter

파이썬 코드는 인덴트로 구분된 코드 블럭 단위로 실행됩니다. 한 코드 블럭 안에서는 Flow Control 구문이 있을 시 그 구문을 따르고, 그렇지 않으면 라인 하나하나씩 실행됩니다.
라인 중 =가 있는 라인과 없는 라인은 다르게 작동합니다. =가 있는 라인은, =의 왼쪽에는 variable, 오른쪽에는 expression이 있습니다. expression은 계산 가능한 코드나, 기존에 값이 assign된 변수입니다. 없는 라인은 기존에 assign된 변수의 값을 변화시키거나, 다른 함수를 실행합니다.
그렇다면, expression의 계산이나 함수의 실행은 어떻게 이루어질까요?

Substitution

파이썬 인터프리터는 단순합니다. 그 시점에서의 Namespace에서 variable을 찾고, 찾지 못할 경우 NameError를 리턴합니다. 만약 찾을 수 있는 경우, 그 코드의 해당 부분을 namespace에서 찾은 값으로 바꿔쓰기하여 계산을 다시 수행합니다.

Example on Sorting

위에서 custom key를 가진 sorting함수를 다시 살펴보며 어떤 식으로 bind-bound가 이루어지는지 살펴보겠습니다.