1. Getting Started

1.1. Hello world!

프로그래머라면 새로운 프로그래밍 언어를 접할 때 마다 반드시 수행하는 의식이 있죠? 그것은 바로 컴퓨터로 하여금 `Hello, world!`를 외치게 하는 일이지요. 아래의 코드는 Python interpreter로 하여금 `Hello, world!`를 출력하도록 명령합니다.

```
print("Hello, world!")
```

```
Hello, world!
```

1.2. Comments

샵("#")을 이용하여 주석을 남길 수 있습니다. 샵은 interpreter로 하여금 해당 행의 나머지 문자열을 모두 무시하도록 합니다.

```
# This is a Python comment
print("This is not a comment")  # This is a comment
```

```
This is not a comment
```

1.3. Variables: Scalar Data Types

Python에서 모든 자료형은 객체(object)로 표현됩니다. 모든 객체는 하나의 자료형(type)을 갖고 있습니다. 자료형에는 scalar와 non-scalar라는 두 가지 종류가 있습니다. Scalar objects는 더 이상 나눌 수 없는 최소 단위의 자료형들을 지칭합니다. Python 자료 표현의 원자(atoms)라고 생각하면 쉽습니다. 혹시 Java에서의 primitive data types의 역할을 떠올렸다면 옳습니다.

Python에는 네 종류의 scalar 자료형이 있습니다.

- `int`: 정수를 표현하기 위해 사용합니다.
- `float`: 실수(real numbers)를 표현하기 위해 사용합니다.
- `bool`: `True`/`False`로 나타나는 Boolean 값을 표현하기 위해 사용합니다.
- `None`: null을 나타내는 자료형이며 `None`이라는 한 가지 값만을 가질 수 있습니다.

Python 내장 함수인 `type()`을 사용하여 객체의 자료형을 알아낼 수 있습니다.

Basic Operators

`int`와 `float`형의 변수에 대하여 다음의 연산자들을 사용할 수 있습니다.

- i+j: i와 j의 합을 계산합니다.
- i-j: i와 j의 차를 계산합니다.
- i*j: i와 j의 곱을 계산합니다.
- i/j: i를 j로 나눈 결과를 float로 반환합니다 (auto type casting). j가 0이면, 에러가 발생합니다.
- i//j: i를 j로 나눈 결과를 내림(truncate)하여 반환합니다.
- i%j: i를 j로 나눈 나머지를 계산합니다.
- i**j: i의 j제곱을 계산합니다.

`bool`형의 변수에 대하여 다음의 연산자들을 사용할 수 있습니다.

- i and j: i와 j 모두가 `True`일 때만 `True`며, 그렇지 않은 경우는 `False`입니다.
- i or j: i와 j 모두가 `False`일 때만 `False`며, 그렇지 않은 경우는 `True`입니다.
- not i: i가 `True`일 때 `False`며, i가 `False`일 때 `True`입니다.

```
# Variable declaration
int_a = 4
int_b = 5

float_c = 3.1
float_d = 2.7

bool_eq = (int_a != int_b)

var_none = None

print('int_a =', int_a)
print('int_b =', int_b)
print('int_a + int_b =', int_a+int_b)

print('float_c =', float_c)
print('float_d =', float_d)
print('float_c + float_d =', float_c+float_d)

print('bool_eq =', bool_eq)

print('var_none =', var_none)
```

```
print(type(int_a))
print(type(float_c))
print(type(bool_eq))
print(type(var_none))

print('int_a/int_b =', int_a/int_b)
print('int_a//int_b =', int_a//int_b)

print('float_c/float_d =', float_c/float_d)
print('float_c//float_d =', float_c//float_d)

print('int_a/int_b =', int_a/int_b)

bool_x = True

print('bool_x =', bool_x)
print('not bool_x =', not bool_x)
print('bool_x and bool_x =', bool_x and bool_x)
print('bool_x and not bool_x =', bool_x and not bool_x)
print('bool_x or bool_x =', bool_x or bool_x)
print('bool_x or not bool_x =', bool_x or not bool_x)
```

```
int_a = 4
int_b = 5
int_a + int_b = 9
float_c = 3.1
float_d = 2.7
float_c + float_d = 5.800000000000001
bool_eq = True
var_none = None
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'NoneType'>
int_a/int_b = 0.8
int_a//int_b = 0
float_c/float_d = 1.1481481481481481
float_c//float_d = 1.0
int_a/int_b = 0.8
bool_x = True
not bool_x = False
bool_x and bool_x = True
bool_x and not bool_x = False
bool_x or bool_x = True
bool_x or not bool_x = True
```

1.4. Variables: Common Non-Scalar Data Types

위에서 소개된 4개의 scalar 자료형을 제외한 나머지 데이터 타입은 모두 non-scalar 자료형으로 분류가 됩니다 (C에서 structures, Java나 C++에서 class를 떠올렸다면 옳습니다).

1.4.1. Strings

Python은 따옴표(')나 쌍따옴표(")를 사용하여 스트링을 나타냅니다. 따옴표나 쌍따옴표를 세 번 반복하여 여러줄에 걸친 스트링을 생성할 수도 있습니다 (multi-line string).

String Operators

Python은 연산자 오버로딩을 이용하여 일반적인 수식 연산자들을 스트링 타입의 변수에도 사용 가능하도록 해두었습니다.

- str1 + str2: + 연산자를 사용하여 두 개의 스트링을 합칠 수 있습니다 (string concatenation).
- str1 * n: * 연산자는 스트링을 반복합니다.
- str1[n]: 스트링 안에서 인덱스를 지정할 수 있습니다.
- str1[n:m]: 스트링 안에서 인덱스의 범위를 지정할 수도 있습니다. Python에서는 이러한 범위 지정을 가리켜 slicing이라고 합니다.

```
s = "a string"
print(s)

s = 'another string'
print(s)


sss = """
This is a multi-line string
that spans over
a single line
"""
print(sss)


s = "Pohang"
print('s =', s)
print('s + s =', s+s)
print('s * 3 =', s*3)
print('s[2] =', s[2])
print('s[0:2] =', s[0:2])
print('s[2:] =', s[2:])
```

```
print('s[:3] =', s[:3])
```

```
a string
another string

This is a multi-line string
that spans over
a single line

s = Pohang
s + s = PohangPohang
s * 3 = PohangPohangPohang
s[2] = h
s[0:2] = Po
s[2:] = hang
s[:3] = Poh
```

1.4.2. Tuples

튜플은 스트링과 같이 일련의 요소들이 순서대로 나열된 것입니다. 튜플과 스트링의 큰 차이점은 튜플의 요소(element)들은 문자가 아니어도 된다는 것입니다. 어떤 data type도 튜플의 요소가 될 수 있으며, 심지어 모든 요소들이 같은 자료형이 아니어도 됩니다.

튜플은 괄호 ( )를 사용하여 표현됩니다. 각 요소들은 쉼표(,)로 구분되어집니다. 스트링에서와 같이 +, *, [ ] 등의 연산자들이 사용될 수 있습니다.

```
# tuples
t1 = () # an empty tuple
t2 = (1, 'two', 3)

print(t1)
print(t2)
print(t2+t2)
print(t2[1])
print((t2+t2)[3])
print((t2+t2)[3:])
print((t2+t2)[:2])
print((t2+t2)[1:4])
```

```
()
(1, 'two', 3)
(1, 'two', 3, 1, 'two', 3)
two
1
(1, 'two', 3)
(1, 'two')
('two', 3, 1)
```

1.4.3. Lists

튜플과 마찬가지로 리스트는 순서가 있는 일련의 값들을 표현하며, 인덱스를 사용하여 각 값을 구분할 수 있습니다. 리스트는 대괄호 [ ]를 사용하여 표현됩니다.

The Mutability of Lists

리스트는 앞 선 두 종류의 데이터 타입과 중요한 차이를 갖습니다. 바로 리스트는 변경이 가능하다는 것입니다. 반명 튜플과 스트링은 한 번 선언이 되면 변경이 불가능합니다.

```
# lists
l1 = [] # an empty list
l2 = ['I did', 'it', 'all', 4, 'love']

print(l2)
print(l2 + l2)
print(l2[2])
print(l2[1:2])
print(l2[2:4])
print(l2[3:])
print(l2[:3])
```

```
['I did', 'it', 'all', 4, 'love']
['I did', 'it', 'all', 4, 'love', 'I did', 'it', 'all', 4, 'love']
all
['it']
['all', 4]
[4, 'love']
['I did', 'it', 'all']
```

1.4.4. Dictionaries

Python에서는 hashtable 형태의 데이터 타입을 기본으로 제공합니다. 딕셔너리 타입의 변수를 사용하면 튜플이나 리스트에서와 달리 정수가 아닌 인덱스를 사용할 수 있습니다. 딕셔너리의 요소(element)들은 key-value의 짝으로 표현됩니다. 즉, 각 요소는 value를 갖고 있으며 key를 통해 인덱스 될 수 있습니다.

딕셔너리는 중괄호 { }를 사용하여 표현되며 key와 value는 콜론(:)으로 구분됩니다.

```
Months = {'Jan':1, 'Feb':2, 'Mar':3, 'Apr':4, 'May': 5, 1:'January', 2:'February', 3:'March', 4:'April', 5:'May'}

print("The third month is " + Months[3])
print("April is the %d-th month" % Months['Apr'])

```

```
The third month is March
April is the 4-th month
```

딕셔너리의 키를 통해 변수에 저장된 각 요소들을 열람해볼 수도 있습니다. 딕셔너리가 제공하는 keys() 함수를 사용하면 저장된 모든 key들을 받아볼 수 있습니다.

```
for key in Months.keys():
  print(Months[key])
```

```
1
2
3
4
5
January
February
March
April
May
```

1.5. Python Controls

Python에서는 분기(branch)와 반복(iteration)을 제공하여 flow를 정의할 수 있게 합니다. 분기를 정의하기 위해 if, elif, else 등의 키워드를 사용하며, 반복을 정의하기 위해 while과 for 등의 키워드를 사용합니다. Python에서 switch 구문은 제공하지 않습니다.

Python에서 **들여쓰기**(indentation)는 코드의 scope을 정의합니다. C나 Java 등의 언어에서 중괄호 { }와 그 역할이 같습니다.

1.5.1. `if/elif/else`

```
# if / else if / else
a = 0

if a == 1:
  print("a == 1")
elif a == 0:
  print("a == 0")
else:
  print("a == something else")
```

```
a == 0
```

```
# Nested if / else if / else
a = 9

if a > 0:
  if a % 3 == 0:
    if a % 9 == 0:
      print('a is positive and divisible by both 3 and 9')
    else:
      print('a is positive and divisible by 3')
  else:
    print('a is a positive number')
elif a == 0:
  print('a is zero')
else:
  print('a is a negative number')
```

```
a is positive and divisible by both 3 and 9
```

1.5.2. `for`

for문의 control block의 표현을 위해 range() 함수가 자주 사용됩니다. range() 함수는 argument의 갯 수에 따라 다음과 같은 범위를 생성합니다.

- range(stop): arugment가 1개 일 때엔, 0부터 stop까지 1씩 증가하며 진행합니다.
- range(start, stop): arugment가 2개 일 때엔, start부터 stop까지 1씩 증가하며 진행합니다.
- range(start, stop, step): arugment가 3개 일 때엔, start부터 stop까지 step씩 증가하며 진행합니다.

```
# for
for i in range(5):
  print(i)
```

```
0
1
2
3
4
```

```
# for
for i in range(2, 7):
  print(i)
```

```
2
3
4
5
6
```

```
# for
for i in range(-5, 5, 2):
  print(i)
```

```
-5
-3
-1
1
3
```

리스트나 튜플을 사용하여 for loop을 정의할 수도 있습니다.

```
l1 = ['A','B','C']
for item in l1:
  print(item)

t1 = ('hana', 'one', 1)
for item in t1:
  print(item)
```

```
A
B
C
hana
one
1
```

1.5.3. `while`

```
# while
x = 10
sum_to_ten = 0
while x > 0:
  sum_to_ten += x
  x -= 1

print(sum_to_ten)
```

```
55
```

1.6. Functions

Python에서 함수는 def라는 키워드로 정의됩니다. Control에서 처럼 들여쓰기를 사용하여 scope이 표현됩니다.

```
def say_ho():
  print('ho-')

print('Are you with me?')
say_ho()
```

```
Are you with me?
ho-
```

def statement의 괄호를 사용하여 함수의 arguments를 정의할 수 있습니다. 또한 return 키워드를 사용하여 원하는 지점에서 return을 정의할 수도 있습니다. return 값을 위한 함수의 타입을 정의할 필요는 없습니다.

```
def maxVal(x, y):
  if x > y:
    return x
```

```
  return y

print('maxVal(3,8)?', maxVal(3,8))
```

```
maxVal(3,8)? 8
```

Python에서는 여러 개의 변수를 동시에 return할 수도 있습니다. 다음 예제에서는 %s의 스트링 placeholder를 사용하여 변수를 대입하고 있습니다.

```
def getName():
  firstname = 'Mike'
  lastname = 'Stockton'

  return firstname, lastname

name_first, name_last = getName()
print('Hello, Mr. %s. I got your firstname %s as well.' % (name_last, name_first))
```

```
Hello, Mr. Stockton. I got your firstname Mike as well.
```

Recursion

함수를 정의하는 기본 요소들을 사용하여 recursive 함수를 정의할 수도 있습니다. 다음 예제는 1부터 n까지 정수의 합을 보여줍니다.

```
def sum_to_ten(n):
  if n < 1:
    return 0
  return sum_to_ten(n-1) + n

print('sum_to_ten(10) =', sum_to_ten(10))
print('sum_to_ten(9) =', sum_to_ten(9))
print('sum_to_ten(5) =', sum_to_ten(5))
```

```
sum_to_ten(10) = 55
sum_to_ten(5) = 15
```