

# Introduction to Python

## Table of Contents

- [I.1. What is Python.2](#)
  - [I.1.1 how to use jupyter notebook](#)
- [II. 2. Basic Python](#)
  - [I.2.1 사칙연산](#)
  - [II. 2.2 변수에 숫자 대입하고 계산하기](#)
  - [III. 2.3 변수에 문자 대입하고 출력하기](#)
- [III. 3. Data types](#)
  - [I. 3.1 int](#)
  - [II. 3.2 float](#)
  - [III. 3.3 bool](#)
  - [IV. 3.4 string](#)
  - [V. 3.5 list](#)
  - [VI. 3.6 dictionary](#)
- [IV. 4. Condition](#)
- [V. 5. Loop](#)
- [VI. 6. Function](#)
- [VII. 7. Packages](#)
  - [I. 7.1 numpy](#)
  - [II. 7.2 matplotlib](#)
  - [III. 7.3 tensorflow](#)

## 1. What is Python ?

```
In [1]: if 4 in [1,2,3,4]: print("4 is in here!")  
4 is in here!
```

위 코드는 다음과 같다.  
"만약 4가 [1,2,3,4] 안에 있으면 '4 is in here!'를 출력한다."  
이처럼 Python을 사용하면 직관적으로 프로그래밍을 할 수 있다.

## 1.1 how to use jupyter notebook

- Esc 키를 누른 후 h 키를 누르면 jupyter notebook의 cheat sheet가 나온다.
- jupyter notebook에는 Command Mode, Edit Mode, 중 두 가지 모드가 있다. 셀을 클릭하면 셀 내에 코드를 짤수 있는 Edit Mode가 되고, 그 상태에서 ESC를 누르면 셀 자체를 수정할수 있는 Command Mode가 된다.
- 많이 사용하는 단축키
  - Esc: Command Mode로 전환
  - Command Mode
    - a: 현재 셀 위에 셀 추가
    - b: 현재 셀 밑에 셀 추가
    - x: 현재 셀 삭제
    - Enter : 현재 셀에서 Edit Mode로 전환
  - Edit Mode
    - Tab: 들여쓰기
    - Shift + Tab: 커서가 위치한 라인, 혹은 드래그된 라인 들여쓰기 취소
    - Esc: Command Mode로 전환
  - 혼용
    - Ctrl + Enter: 현재 셀 실행
    - Shift + Enter: 현재 셀 실행 후 밑의 셀로 이동, 밑에 셀이 없을 경우 셀 추가

## 2. Basic Python

### 2.1 사칙연산

- 계산기를 사용하듯 작성한다.

```
In [2]: 1 + 2  
Out[2]: 3  
  
In [3]: 3 / 2.4  
Out[3]: 1.25  
  
In [4]: 3 * 9  
Out[4]: 27
```

### 2.2 변수에 숫자 대입하고 계산하기

```
In [5]: a = 1  
        b = 2  
        a + b  
  
Out[5]: 3
```

### 2.3 변수에 문자 대입하고 출력하기

Collected by Kwangho Lee  
iSystems Design Lab  
<http://isystems.unist.ac.kr/>  
UNIST

Reference  
[Wikidocs \(https://wikidocs.net/6\)](https://wikidocs.net/6)  
[TensorFlow Essential \(https://livebook.manning.com/#/book/machine-learning-with-tensorflow/chapter-2\(1\)\)](https://livebook.manning.com/#/book/machine-learning-with-tensorflow/chapter-2(1))

```
In [6]: a = 'Python'
        print(a)

Python
```

### 3. Data types

- int
- float
- bool
- string
- list
- dictionary

Python에는 여러 가지 자료형이 있지만, 특정 자료형을 맞추기 위해 변수 설정을 따로 신경쓸 필요는 없다.

#### 3.1 int

- int 는 정수형(integer)의 줄임말로, 정수를 저장하는 자료형이다.

```
In [7]: a = 123
        a = -56
        a = 0
        type(a)

Out[7]: int
```

#### 3.2 float

- float 는 실수형 (Floating-point) 을 표현하며 소수점이 포함된 숫자를 말한다.

```
In [8]: a = 1.2
        a = -3.45
        type(a)

Out[8]: float
```

#### 3.3 bool

- bool 은 참, 거짓 (boolean) 두가지 값을 표현한다.

```
In [9]: a = True
        a = False
        type(a)

Out[9]: bool
```

#### 3.4 string

- string 은 문자열을 나타내며 단어, 문자 등으로 구성된 문자들의 집합을 의미한다. 예시는 다음과 같다.

```
In [10]: a = 'Hello World!'
         a = 'a'
         a = '123'
         type(a)

Out[10]: str
```

문자열을 사용하기 위해서는 문자열로 만들고 싶은 부분 양 끝을 따옴표(' ')로 둘러싸면 된다.

1) 큰따옴표

```
In [11]: "Hello World!"

Out[11]: 'Hello World!'
```

2) 작은따옴표

```
In [12]: 'Python is fun!'

Out[12]: 'Python is fun!'
```

string 과 함께 변수를 출력하고 싶을때는 .format()을 사용하면 편리하다.  
사용 방법은 다음과 같다.

'사용하고 싶은 문자열 { }'.foramt(변수)

문자열 내에 중괄호({})를 포함시키고 문자열 맨 끝에 .format(변수)를 입력하면  
문자열 내에 있는 중괄호 안에 변수가 대입된다.

```
In [13]: a = 24
         b = 'My age: {}'.format(a)
         print(b)

My age: 24
```

#### 3.5 list

- 1보다 크고 10보다 작은 자연수 중 홀수인 숫자 집합 {1, 3, 5, 7, 9}를 생각해 보자. 이런 집합은 int나 string으로 표현하기 어렵다. Python에서는 이런 숫자 모음을 담아둘 수 있는 자료형 list가 존재한다.

```
In [14]: odd = [1, 3, 5, 7, 9]
```

리스트를 만들 때는 요소들을 대괄호([])로 감싸 주고 각 요소들을 침표(.)로 구분해준다.

리스트명 = [요소1, 요소2, 요소3, ...]

리스트 안에는 어떠한 자료형도 포함시킬 수 있다.

```
In [15]: a = []
        b = [1, 2, 3]
        c = ['I', 'love', 'deep', 'learning']
        d = [1, 2, 'Combination', 'of', 'int and str']

        print(d)

[1, 2, 'Combination', 'of', 'int and str']
```

리스트명[숫자] 를 사용하여 각 요소에 접근 가능하다.  
주의할 점은 순서가 0번부터 시작한다는 점이다.

```
In [16]: d[0]
Out[16]: 1

In [17]: d[1]
Out[17]: 2

In [18]: d[4]
Out[18]: 'int and str'
```

리스트의 요소를 여러개 갖고 오고 싶은 경우엔 슬라이싱 (slicing) 이란 방법을 사용하면 된다.

리스트명[시작:끝]

주의할 점은 마지막 순서는 포함되지 않는다는 점이다.

```
In [19]: d[0:1]
Out[19]: [1]

In [20]: d[0:2]
Out[20]: [1, 2]

In [21]: d[1:4]
Out[21]: [2, 'Combination', 'of']

In [22]: d[0:0]
Out[22]: []
```

### 3.6 dictionary

- '이름'='홍길동'이고 '생일'='1970년 1월 1일'인 사람에 대한 변수를 만든다고 생각해 보자.  
파이썬에는 이런 대응관계를 나타내는 자료형이 있고 이를 dictionary 라 부른다.
- dinctionary 는 list 처럼 순차적으로 해당 요소값을 구하지 않고 key 를 통해 Value 를 읽는다. 이것이 dictionary 의 큰 특징이다.  
baseball이란 단어 뜻을 찾기 위해 사전의 내용을 순차적으로 모두 검색하는 것이 아니라 baseball이라는 단어가 있는 곳만 펼쳐 보는 것이다.
- 맨 위의 예시에선 '이름'이 Key가 되고 '홍길동'이 Value가 될것이다.

다음은 기본적인 dictionary 의 모습이다.

```
{key1 : value1, key2 : value2, key3 : value3}
```

각 요소는 Key : Value 형태로 이루어져 있고 쉼표(.)로 구분되어 있다.  
Key 에는 변하지 않은 값을 사용하고, Value 에는 변하는값, 변하지 않는 값 모두 사용 가능하다.

```
In [23]: dic = {
        'name' : 'iSystems',
        'website' : 'http://isystems.unist.ac.kr/teaching/machine-learning/',
        1 : 'Hello World',
        32 : [1,2,3,4]
        }
```

dictionary 에서 Key 를 사용해 Value 를 읽는 방법은 다음과 같다.

딕셔너리명[키]

```
In [24]: dic['name']
Out[24]: 'iSystems'

In [25]: dic['website']
Out[25]: 'http://isystems.unist.ac.kr/teaching/machine-learning/'

In [26]: dic[1]
Out[26]: 'Hello World'

In [27]: dic[32]
Out[27]: [1, 2, 3, 4]
```

### 4. Condition

조건문은 프로그램에서 주어진 조건을 판단하여 상황에 맞게 처리해야 할 때 쓰이는 것이다. Python에서는 조건문에 if와 else를 사용한다.

if 문의 기본 구조는 다음과 같다.

```
if 조건:
    수행할 문장 1
    수행할 문장 2
else:
    수행할 문장 A
    수행할 문장 B
...
```

조건문이 참(True)이면 if문 바로 다음의 문장들 (if 블록)을 수행하고, 거짓이면 else 문 다음 문장들 (else 블록)을 수행하게 된다.  
이 때 else는 if 없이 독립적으로 사용할 수 없다.

```
In [28]: weekday = True
         if weekday:
             print("Go to the LAB")
         else:
             print("Wake up at 11:00 a.m and Go to the LAB")
```

Go to the LAB

```
In [29]: weekday = False
         if weekday:
             print("Go to the LAB")
         else:
             print("Wake up at 11:00 a.m and Go to the LAB")
```

Wake up at 11:00 a.m and Go to the LAB

#### • 주의할 점

if 문을 만들 때는 if 조건: 바로 아래 문장부터 if문에 속하는 모든 문장에 들여쓰기를 해 주어야 한다.  
그렇지 않으면 오류가 발생한다.

```
In [30]: if True:
          print('Hello')
          print('World')
          print('!!!')

File "<ipython-input-30-1bea1ba66701>", line 4
    print('!!!')
    ^
IndentationError: unexpected indent
```

## 5. Loop

- 반복적으로 수행해야 할 일이 있을 경우 반복문을 사용한다.
- Python에서는 여러 반복문을 제공해 주는데, 그 중 for문을 자주 사용한다.

for문의 기본적인 구조는 다음과 같다.

```
for 변수 in 리스트 (또는 튜플, 문자열):
    수행할 문장 1
    수행할 문장 2
    ...
```

리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 for문 블록 안의 문장들이 수행된다.

```
In [31]: test_list = ['one', 'two', 'three']
         for i in test_list:
             print(i)
```

one  
two  
three

일정 횟수만큼 반복하는 코드를 짜고 싶은 경우 내장 함수인 range()를 이용하면 된다.

```
In [32]: for i in range(3):
          print(i)
          print("Hi")
```

0  
Hi  
1  
Hi  
2  
Hi

## 6. Function

- 프로그래밍을 하면서 같은 내용을 반복해서 작성한다면, 함수를 사용하는 것이 좋다.
- '반복적으로 사용되는 가치 있는 부분'을 한 덩어리로 묶어서 '어떤 입력값이 주어졌을 때 어떤 결과 값을 돌려주'는' 것을 함수라 한다.

Python 함수의 구조는 다음과 같다.

```
def 함수명(인력 인수):
    수행할 문장 1
    수행할 문장 2
    ...
```

def 는 함수를 만들 때 사용하는 키워드이며 함수명은 함수를 만드는 사람이 임의로 정할 수 있다.  
다음과 같이  $x$ 와  $y$ 를 입력 인수로 받아 더한 값을 돌려주는 함수를 만든다면,

$$f(x,y) = x + y$$

```
In [33]: def my_sum(x, y):
        return x + y
```

위와 같이 코드를 작성할 수 있다. 여기에서, return은 함수의 결과값을 돌려주는 명령어이다. return이 없으면 함수는 아무 값도 반환하지 않는다.

```
In [34]: a = 3
        b = 11
        result = my_sum(a, b)
        print(result)

14
```

함수의 인수에 초기값을 미리 설정할 수도 있다.

```
In [35]: def my_sum2(x=1, y=2):
        return x + y

In [36]: print(my_sum2(4, 5)) # x == 4, y == 5
        print(my_sum2(x=4, y=5)) # x == 4, y == 5
        print(my_sum2(x=4)) # x == 4, y == 2
        print(my_sum2()) # x == 1, y == 2

9
9
6
3
```

## 7. Packages

- 모듈, 또는 패키지란 함수나 변수 또는 클래스들을 모아 놓은 파일을 말한다. 즉, 다른 Python 프로그램에서 불러와 사용할 수 있게끔 만들어진 Python 파일이라고도 할 수 있다.
- Python에는 굉장히 유용한 여러 패키지가 있는데 그중 많이 쓰이는 numpy, matplotlib, 그리고 tensorflow에 대해 간략히 소개하겠다.

모듈을 코드 내에 추가하는 방법은 다음과 같다.

```
import 모듈명
import 모듈명 as 임의의 축약 이름
from 모듈명 import 모듈 내 변수, 함수 혹은 클래스
```

각각 의미하는 바는,

- 1) 모듈을 전부 코드 내에 추가한다.
- 2) 모듈을 전부 코드 내에 추가하고 모듈명을 임의의 축약 이름으로 바꿔 사용한다.
- 3) 모듈 내에 있는 특정 변수, 함수 혹은 클래스만 추가한다.

## 7.1 numpy

- numpy는 거의 모든 Python 프로그램에서 사용되는 패키지이다.
- array라는 데이터 타입을 이용해서 matrix나 tensor를 표현하기도 하고, 수학적 계산에서 기초가 되는 여러 함수를 제공한다.

```
In [37]: import numpy
        a = numpy.array([1, 2, 3])
        print(a)

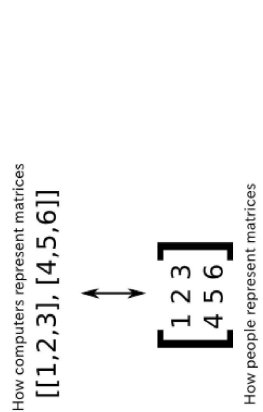
[1 2 3]
```

```
In [38]: import numpy as np
        a = np.array([1, 2, 3])
        print(type(a))
        print(a.shape)
        print(a)

<class 'numpy.ndarray'>
(3,)
[1 2 3]
```

```
In [39]: b = np.array([[1,2,3],[4,5,6]])
        print(type(b))
        print(b.shape)
        print(b)

<class 'numpy.ndarray'>
(2, 3)
[[1 2 3]
 [4 5 6]]
```



np.array의 각 요소에 접근하는 방법은 list에서 사용하던 방법과 같다. 어레이명[숫자]를 사용하여 각 요소에 접근 가능하다. 차원이 늘어나면 어레이명[숫자, 숫자, ...] 혹은 어레이명[숫자][숫자]로 접근한다. **순서가 0번부터 시작한다는 점에 주의한다.**

```
In [40]: a[0]

Out[40]: 1

In [41]: a[2]

Out[41]: 3
```

```
In [42]: b[0]
Out[42]: array([1, 2, 3])

In [43]: b[0,1]
Out[43]: 2

In [44]: b[0][1]
Out[44]: 2
```

어레이의 요소를 여러개 갖고 오고 싶은 경우엔 리스트와 같이 슬라이싱 (slicing) 이란 방법을 사용하면 된다.  
어레이명[시작:끝]  
차원이 늘어나면 다음과 같이 슬라이싱이 가능하다.

어레이명[시작:끝, 시작:끝]

**마지막 인덱스는 포함되지 않는다.**

```
In [45]: a[0:1]
Out[45]: array([1])

In [46]: a[0:2]
Out[46]: array([1, 2])

In [47]: a[0:0]
Out[47]: array([], dtype=int64)
```

```
In [48]: b[0:2]
Out[48]: array([[1, 2, 3],
               [4, 5, 6]])
```

```
In [49]: b[0:2, 0:2]
Out[49]: array([[1, 2],
               [4, 5]])
```

시작, 끝을 정하지 않고 ':' 만 사용할 시, 해당 dimension의 모든 요소가 포함된다.

어레이명[ : ]

```
In [50]: b[0:2, :]
Out[50]: array([[1, 2, 3],
               [4, 5, 6]])
```

**어레이명[시작:끝][시작:끝]은 기대하는 값을 반환하지 않는다.**

```
In [51]: b[0:2][0:2]
Out[51]: array([[1, 2, 3],
               [4, 5, 6]])
```

이 외에도, array내에 가장 큰 값을 반환하는 `numpy.argmax()`, `matrix` 연산에 용이한 `np.matrix` 등 다양한 클래스들이 있다.

참고 링크

- [numpy official tutorial \(https://docs.scipy.org/doc/numpy-dev/user/quickstart.html\)](https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)
- [numpy reference \(https://docs.scipy.org/doc/numpy/reference/\)](https://docs.scipy.org/doc/numpy/reference/)

## 7.2 matplotlib

matplotlib는 이미지나 데이터를 시각화하거나 그래프를 그릴 때 유용한 패키지이다.

import 할 땐 다음과 같이 쓴다.

```
In [52]: import matplotlib.pyplot as plt
```

그리고 싶은 이미지나 데이터를 figure 위에 올리고 이를 show한다.  
간단한 사용 방법은 다음과 같다.

```
plt.plot(x, y)
plt.show()
```

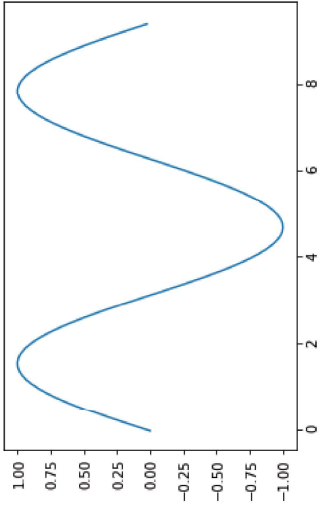
이미지를 출력할 때는 다음과 같이 작성한다.

```
plt.imshow(이미지)
plt.show()
```

이 때 이미지는 RGB값을 갖고있는 2차원 데이터일 수도 있고 단순한 2차원 array일 수도 있다.

```
In [53]: # Compute the x and y coordinates for points on a sine curve
x = np.arange(0,3*np.pi,0.1)
y = np.sin(x)
```

```
# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```



아래에서 사용된 cPickle은 Python 자료형으로 데이터를 저장하고 불러오는 패키지이다.

```
In [54]: from six.moves import cPickle
input_image = cPickle.load(open('./image_files/lena.pkl', 'rb'))
```

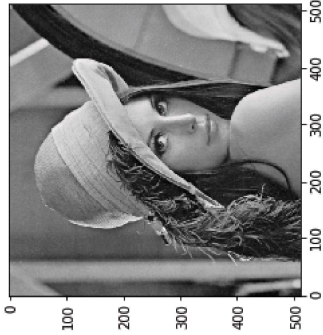
```
In [55]: type(input_image)
```

```
Out[55]: numpy.ndarray
```

```
In [56]: input_image.shape
```

```
Out[56]: (512, 512)
```

```
In [57]: plt.imshow(input_image, 'gray')
plt.show()
```



pyplot에는 여러 기능들이 있으므로, 잘 사용하면 간결하게 시각화가 가능하다.

참고 링크

- [pyplot official tutorial \(https://matplotlib.org/users/pyplot\\_tutorial.html\)](https://matplotlib.org/users/pyplot_tutorial.html)
- [pyplot reference \(https://matplotlib.org/api/pyplot\\_summary.html\)](https://matplotlib.org/api/pyplot_summary.html)

## 7.3 tensorflow

- tensorflow는 google에서 제공하는 deep learning 패키지이다.
- Python 코드는 한 줄씩 실행되어 가며 진행된다면 tensorflow는 모든 계산을 그래프로 그린 뒤, 마지막에 Session을 열어 한꺼번에 계산한다는 것이 특징이다.

tensorflow에는 크게 세 가지 구조가 있다.

- 상수를 뜻하는 tf.constant(임의의 데이터)
- 변수를 뜻하는 tf.Variable(임의의 데이터 혹은 random value)
- 처음엔 아무것도 없는 빈 그릇이지만 들어오는 데이터를 담을 수 있는 tf.placeholder(데이터 타입, 데이터 모양)

```
In [58]: import tensorflow as tf
```

```
a = tf.constant([1, 2, 3])
b = tf.constant([4, 5, 6])
```

```
c = tf.constant([[1, 2],[3, 4]])
d = tf.constant([[1,-1], [-1, 1]])
```

```
w = tf.Variable([1, 1])
```

```
x = tf.placeholder(tf.float32, [2, 2])
```

```
summation = a + b
```

```
multiplication = c * d
```

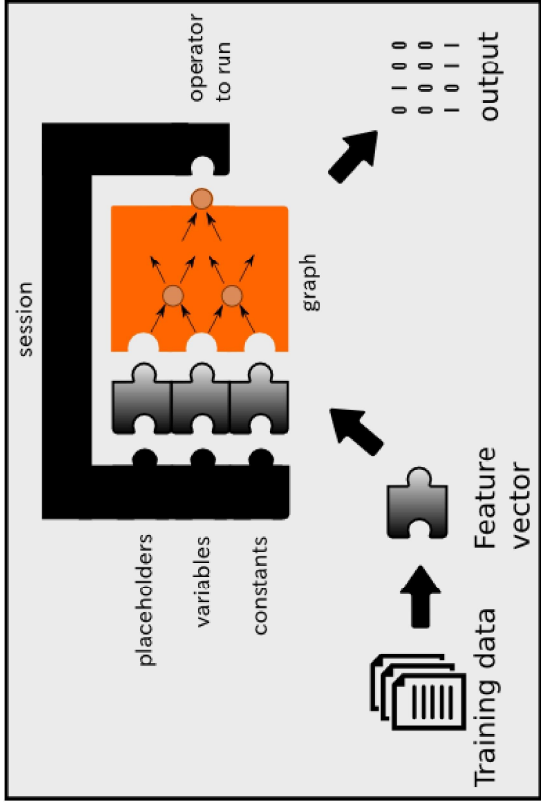
```
In [59]: summation
```

```
Out[59]: <tf.Tensor 'add:0' shape=(3,) dtype=int32>
```

```
In [60]: multiplication
```

```
Out[60]: <tf.Tensor 'mul:0' shape=(2, 2) dtype=int32>
```

위와 같이 결과값이 아직 없다.  
 계산기에 비유하자면 3 + 4를 적어 놓기만 한 상태로 아직 계산기에 입력하기 전 단계인 것이다.  
 이 값을 계산하기 위해서는 `tf.Session()` 안에 수식을 집어넣어 주면 된다.



```
In [61]: sess = tf.Session()
         sess.run(summation)
```

```
Out[61]: array([5, 7, 9], dtype=int32)
```

```
In [62]: sess.run(multiplication)
```

```
Out[62]: array([[ 1, -2],
                [-3,  4]], dtype=int32)
```

`tf.Variable` 을 사용하기 위해서는 변수를 먼저 초기화해 주는 작업이 필요하다.

```
In [63]: init = tf.global_variables_initializer()
         sess.run(init)
```

```
In [64]: sess.run(w)
```

```
Out[64]: array([1, 1], dtype=int32)
```

`tf.placeholder` 를 사용하기 위해서는 `sess.run()` 을 할 때 `feed_dict`라는 인수를 이용하여 데이터를 넣어주면 된다.  
`feed_dict`에 dictionary 형태로 값을 넣는다.  
 이 때 Key 는 `placeholder`의 변수명이 되고 Value는 넣고 싶은 데이터가 된다.

```
In [65]: sess.run(x, feed_dict={x : [[1,2],[3,4]]})

Out[65]: array([[ 1.,  2.],
                [ 3.,  4.]], dtype=float32)
```