

Python

1. Python

❖ 파이썬의 장점

- ✓ Guido가 생각했던 Python 문법적 특징은 들여쓰기를 철저하게 지키도록 언어를 설계
- ✓ 코드의 가독성이 좋음
- ✓ C 언어에서처럼 { } 등의 괄호를 넣지 않기 때문에 프로그램을 좀더 깔끔함
- ✓ 파이썬 코드는 재사용하기가 쉬움
- ✓ 코드의 분석이 쉽기 때문에 다른 사람이 작성한 코드를 받아서 작업하는 사람들이 훨씬 더 작업이 편리
- ✓ 생태계가 좋음

1. Python

❖ 파이썬의 구현

- ✓ C파이썬: C로 작성된 인터프리터를 사용하는 일반적인 파이썬으로 ipython이라고도 합니다.
- ✓ 스택리스 파이썬 : C 스택을 사용하지 않는 인터프리터
- ✓ 자이썬 : 자바 가상 머신 용 인터프리터. 과거에는 제이파이썬(Jpython)이라고 불리기도 함
- ✓ IronPython : .NET 플랫폼 용 인터프리터
- ✓ PyPy : 파이썬으로 작성된 파이썬 인터프리터.

❖ 사용 가능 플랫폼

- ✓ 마이크로소프트 윈도우(9x/NT 계열은 최신판, 3.1 및 MS-DOS는 옛 버전만)
- ✓ 매킨토시(맥 OS 9 이전, 맥 OS X 이후 포함)
- ✓ 각종 유닉스
- ✓ 리눅스
- ✓ 팜 OS
- ✓ 노키아 시리즈 60

1. Python

❖ 파이썬의 활용 분야

- ✓ GUI Programming: 기본 모듈인 Tkinter 이용
- ✓ Web Programming: django 프레임워크
- ✓ Game Programming: PyOpenGL,
- ✓ Database Programming
- ✓ Text 처리
- ✓ 수치 연산: Programming: Numeric Python 모듈이나 nextworkx모듈
- ✓ 병렬 연산 Programming
- ✓ 사물인터넷 – 라즈베리 파이
- ✓ C, C++, Java 와 결합한 프로그래밍
- ✓ 데이터 분석 분야: NumPy, Pandas, Scipy
- ✓ 시각화: Matplotlib..

2. 파이썬 설치

❖ 파이썬의 버전

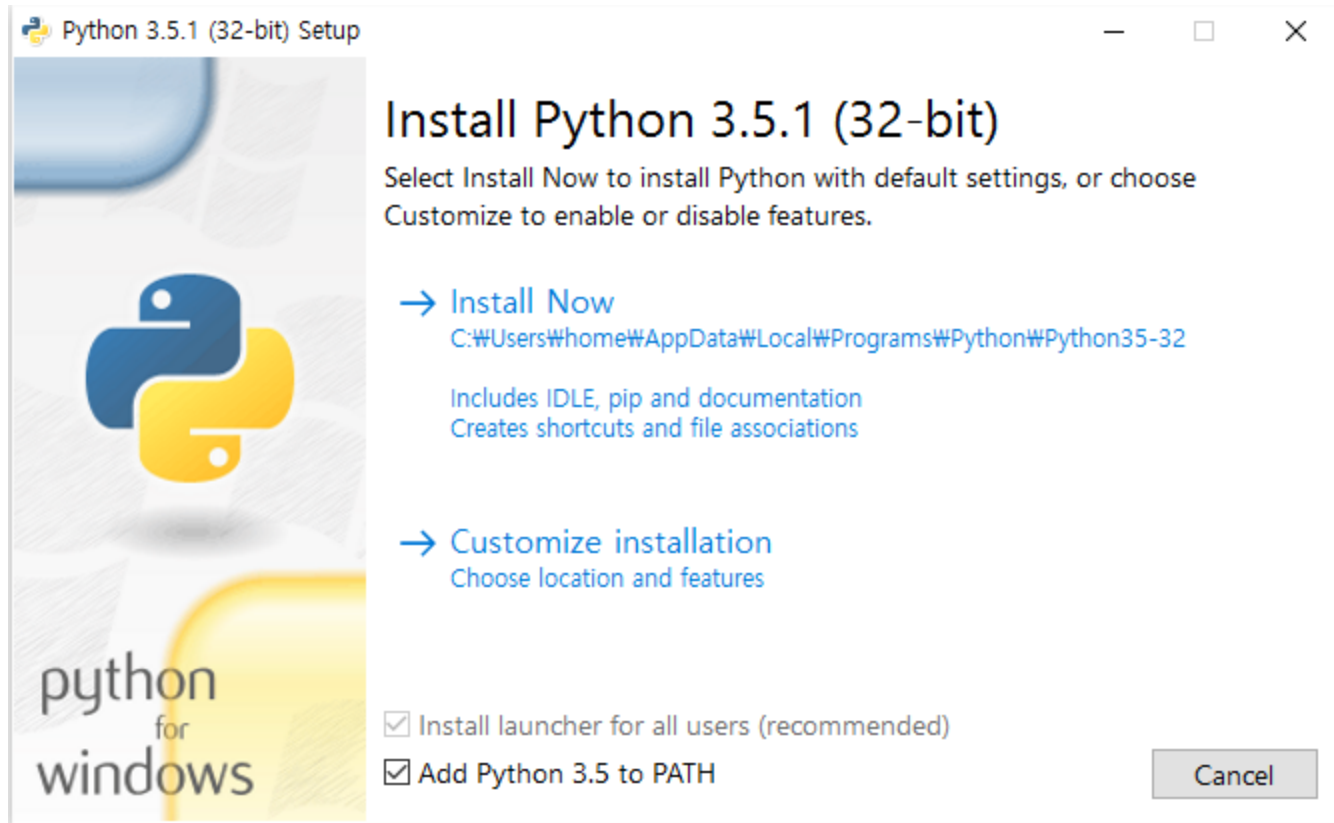
- ✓ 파이썬은 2.x 버전과 3.x 버전이 있는데 파이썬은 2.x버전 과 3.x버전 간의 호환성을 유지하고 있지 않습니다.
- ✓ 변화
 - ◆ print가 함수로 변경
 - ◆ long 형이 없어지고 정수는 int로 통일
 - ◆ int / int 의 결과는 float
 - ◆ 문자열이 string과 unicode로 구분되었는데 string 과 bytes로 구분됩니다.
- ✓ 2to3.py를 이용해서 2.x 버전을 3 버전으로 변환가능(파이썬 설치 디렉토리의 Tools\Scripts 디렉토리에 존재)

❖ 파이썬 다운로드 및 설치

<https://www.python.org/downloads/>

윈도우 용을 설치할 때 파이썬 명령어가 있는 디렉토리를 path에 추가하는 옵션을 체크해야 이클립스에서 파이썬 설정을 자동으로 할 수 있습니다.

2. 파이썬 설치



2. 파이썬 설치

파이썬 배포판 - 아나콘다: www.continuum.io/downloads
<https://repo.continuum.io/archive/.winzip>

- 파이썬 배포판: lpython, Jupyter notebook, Qt Console, Python 포함
- 데이터 분석, 수학, 과학 관련 다양한 라이브러리 포함

Get Superpowers with Anaconda

Anaconda is the leading open data science platform powered by Python. The open source version of Anaconda is a high performance distribution of Python and R and includes over 100 of the most popular **Python**, R and Scala packages for data science. Additionally, you'll have access to over 720 packages that can easily be installed with conda, our renowned package, dependency and environment manager, that is included in Anaconda. Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution. See [the packages included with Anaconda](#) and [the Anaconda changelog](#).

Which version should I download and install?

Because Anaconda includes installers for Python 2.7 and 3.5, either is fine. Using either version, you can use Python 3.4 with the conda command. You can create a 3.5 environment with the conda command if you've downloaded 2.7 — and vice versa.

If you don't have time or disk space for the entire distribution, try [Miniconda](#), which contains only conda and Python. Then install just the individual packages you want through the conda command.

Anaconda for Windows

PYTHON 2.7	PYTHON 3.5
WINDOWS 64-BIT GRAPHICAL INSTALLER 341M	WINDOWS 64-BIT GRAPHICAL INSTALLER 352M
Windows 32-bit Graphical Installer 286M	Windows 32-bit Graphical Installer 293M

Behind a firewall? Use these [zipped Windows installers](#).

cloudera

GET NOW

VIEW OUR
WEBINARS

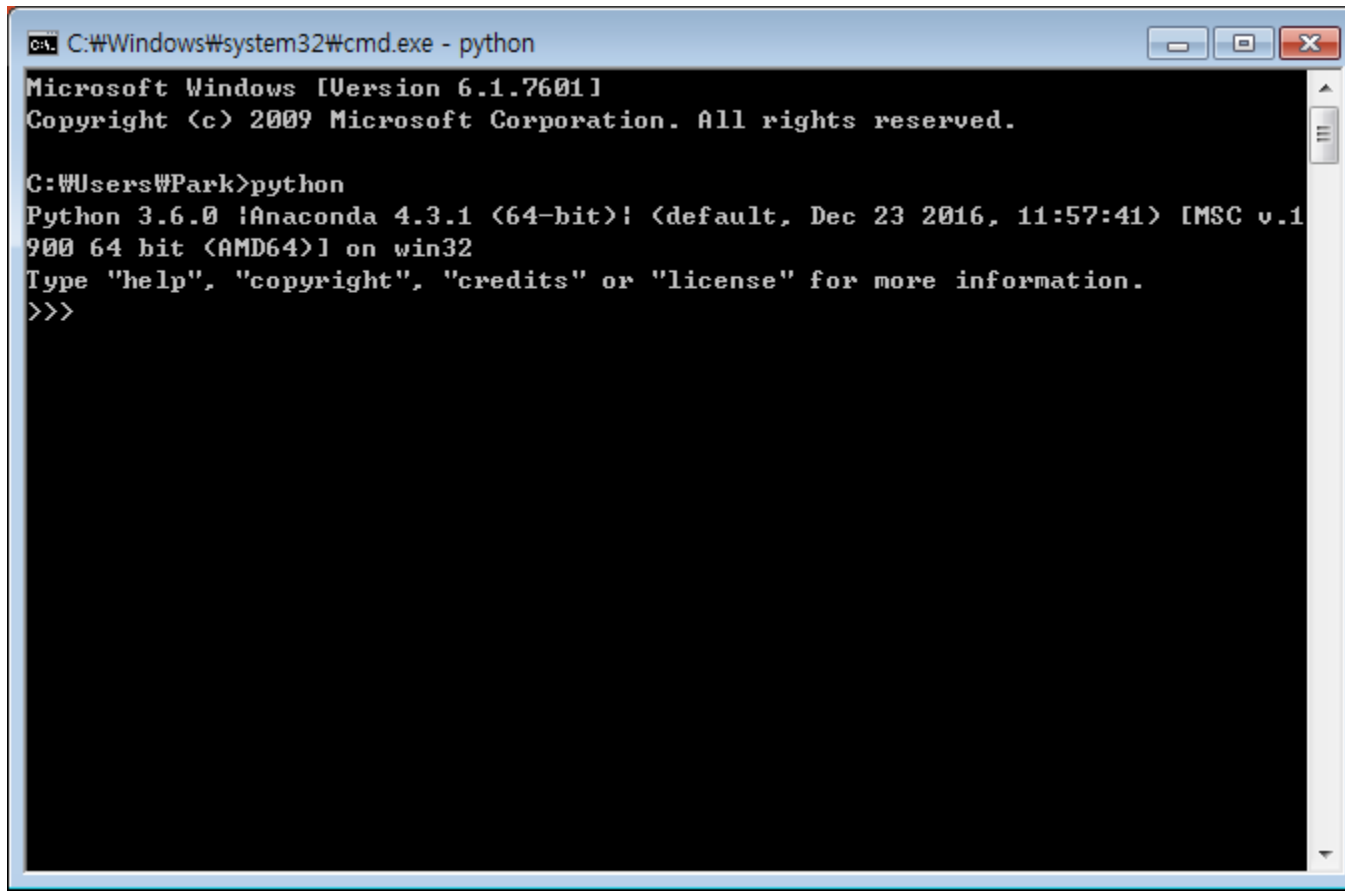
See upcoming and previous
webinars from our experts.

VIEW NOW

2. 파이썬 설치

❖ 설치 확인

- ✓ cmd를 실행
- ✓ python 이라고 입력 한 후 버전 확인



```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Park>python
Python 3.6.0 |Anaconda 4.3.1 (64-bit)| (default, Dec 23 2016, 11:57:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```


3. IDE

| IDE

- ❖ ipython: 아나콘다를 설치하면 같이 설치
- ❖ PyDev 플러그 인을 이용한 이클립스
- ❖ Visual Studio의 파이썬 도구
- ❖ PyCharm



3. IDE

❖ anaconda와 ipython 업데이트

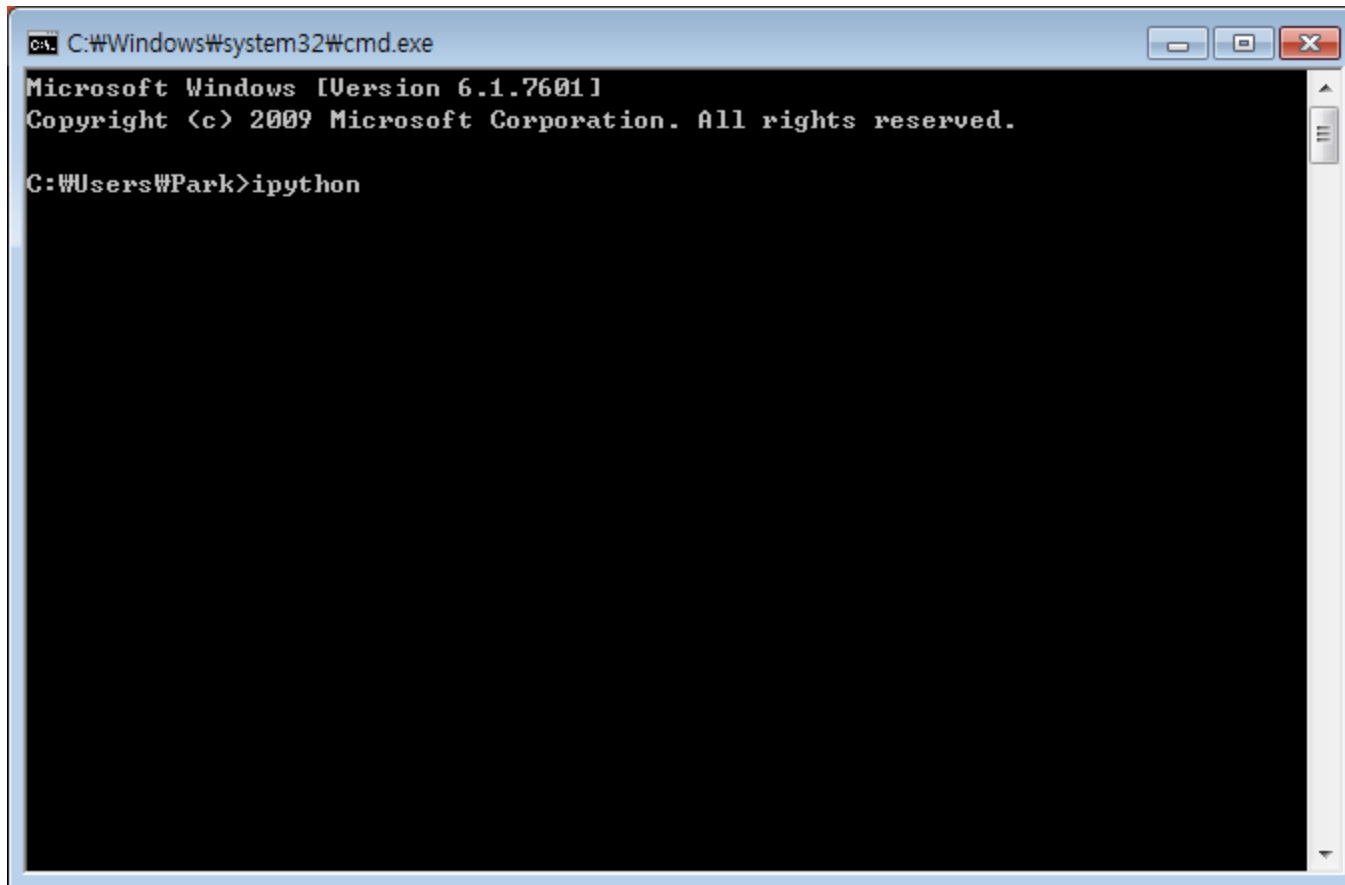
conda update conda

conda update ipython



3. IDE

- ❖ ipython을 이용한 코딩 및 실행

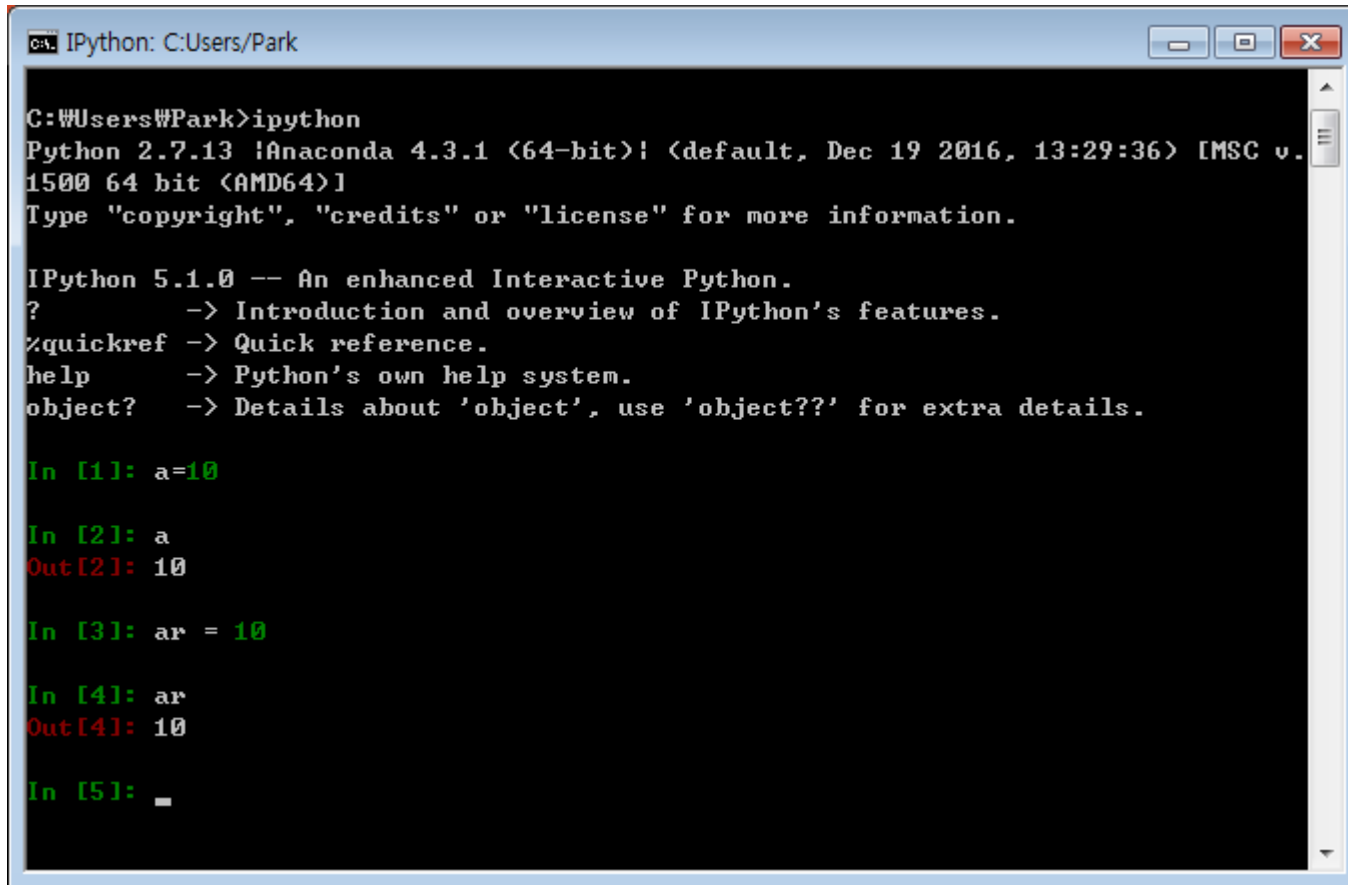


```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Park>ipython
```

3. IDE

- ❖ 객체의 이름만 입력하면 바로 출력



```
C:\Users\Park>ipython
Python 2.7.13 |Anaconda 4.3.1 (64-bit)| (default, Dec 19 2016, 13:29:36) [MSC v.
1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]: a=10

In [2]: a
Out[2]: 10

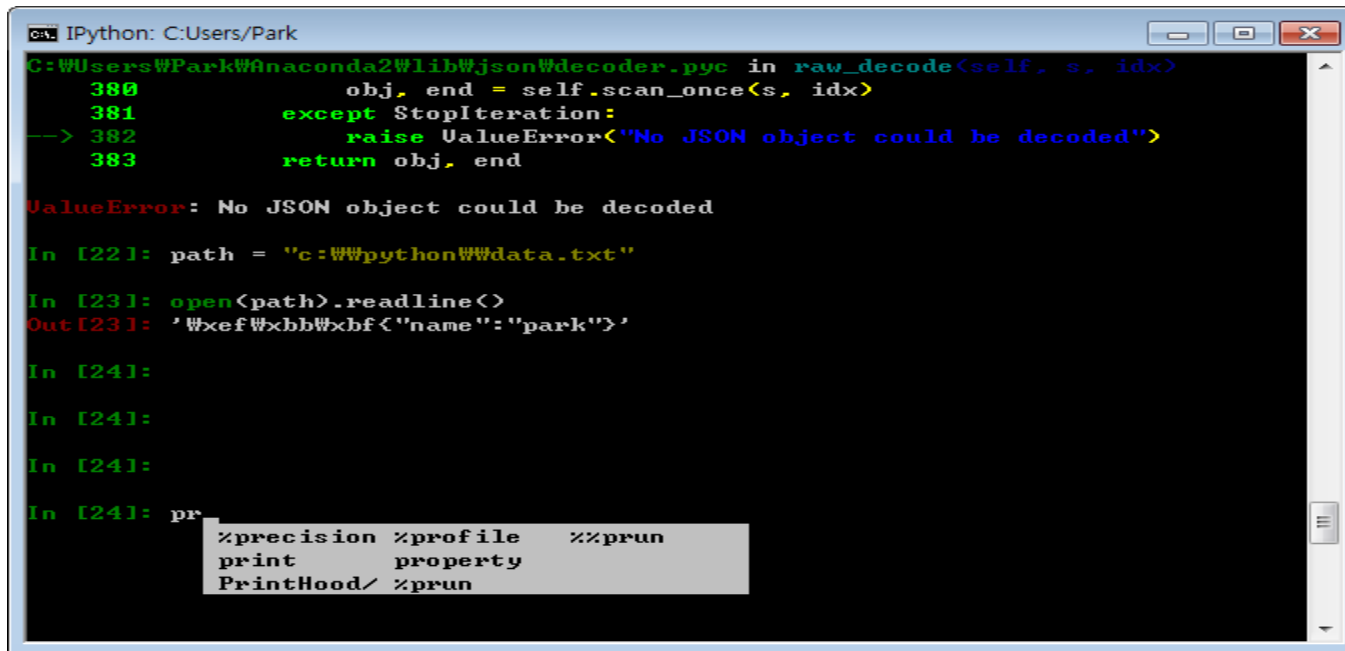
In [3]: ar = 10

In [4]: ar
Out[4]: 10

In [5]: _
```

4.ipython

- ❖ 탭을 이용한 자동완성 기능 제공: 명령을 입력하는 동안 Tab을 누르면 현재 위치에서 사용할 수 있는 변수를 화면에 출력
- ❖ 변수 이름 앞이나 뒤에 ?를 입력하면 변수에 대한 정보를 출력
- ❖ 함수 이름 뒤에 ??를 입력하면 함수의 소스 코드 출력



```
IPython: C:\Users\Park
C:\Users\Park\Anaconda2\lib\json\decoder.py in raw_decode(self, s, idx)
    380         obj, end = self.scan_once(s, idx)
    381     except StopIteration:
--> 382         raise ValueError("No JSON object could be decoded")
    383     return obj, end

ValueError: No JSON object could be decoded

In [22]: path = "c:\\python\\data.txt"

In [23]: open(path).readline()
Out[23]: 'b\xef\xbb\xbf{"name":"park"}'

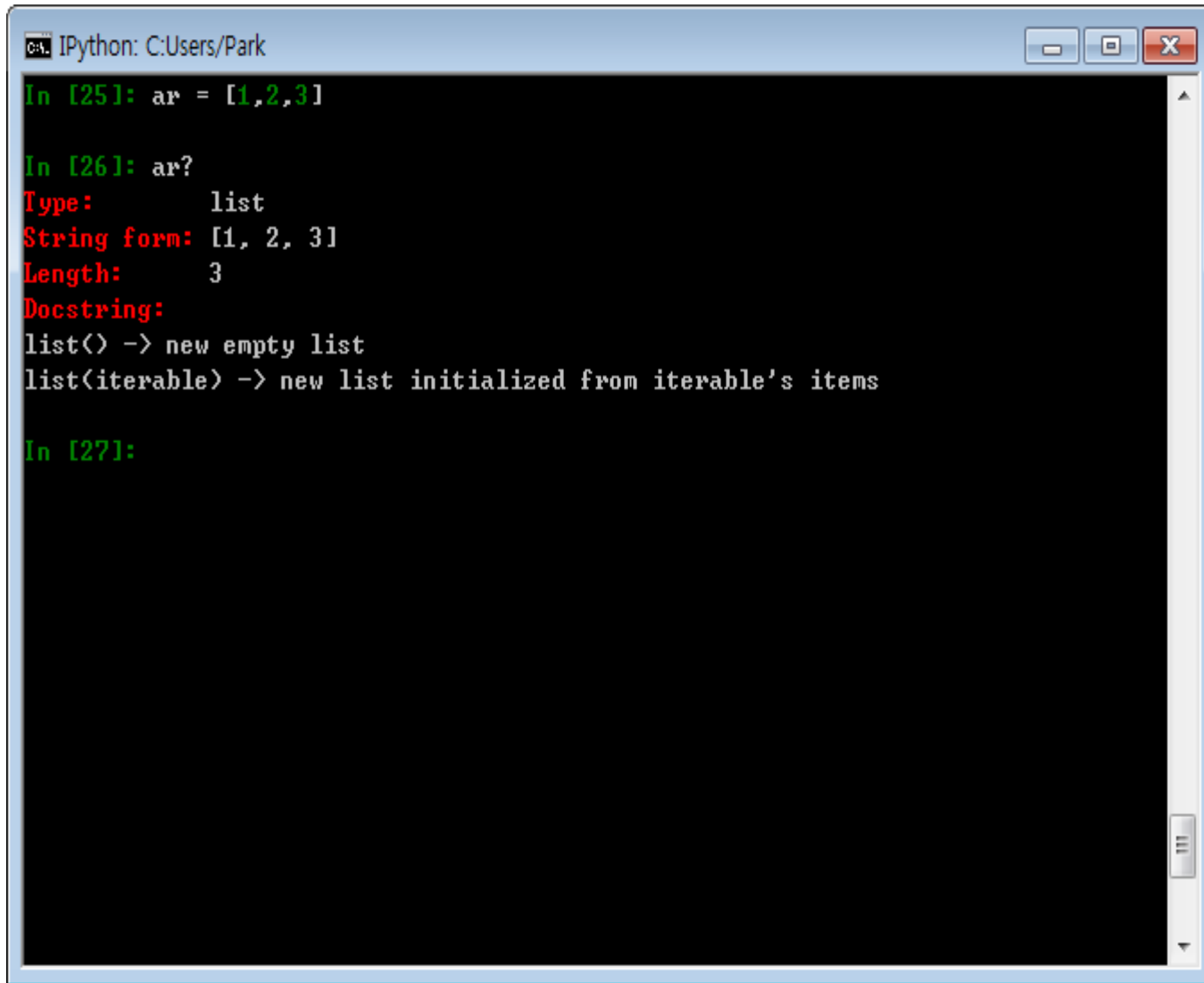
In [24]:

In [24]:

In [24]:

In [24]: pr
%precision %profile %prun
print      property
PrintHood/ %prun
```

4.ipython



The screenshot shows an IPython terminal window with the title bar 'IPython: C:\Users\Park'. The terminal has a black background with green text for input and red text for output. The user has entered three commands: 'In [25]: ar = [1,2,3]', 'In [26]: ar?', and 'In [27]:'. The output for the second command shows the type, string form, length, and docstring of the list 'ar'.

```
IPython: C:\Users\Park

In [25]: ar = [1,2,3]

In [26]: ar?
Type:      list
String form: [1, 2, 3]
Length:    3
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items

In [27]:
```

4.ipython

❖ 작업 디렉토리 확인

In [1]: `import os`

In [2]: `os.getcwd()`

Out[2]: 'C:\Users\WWPark'

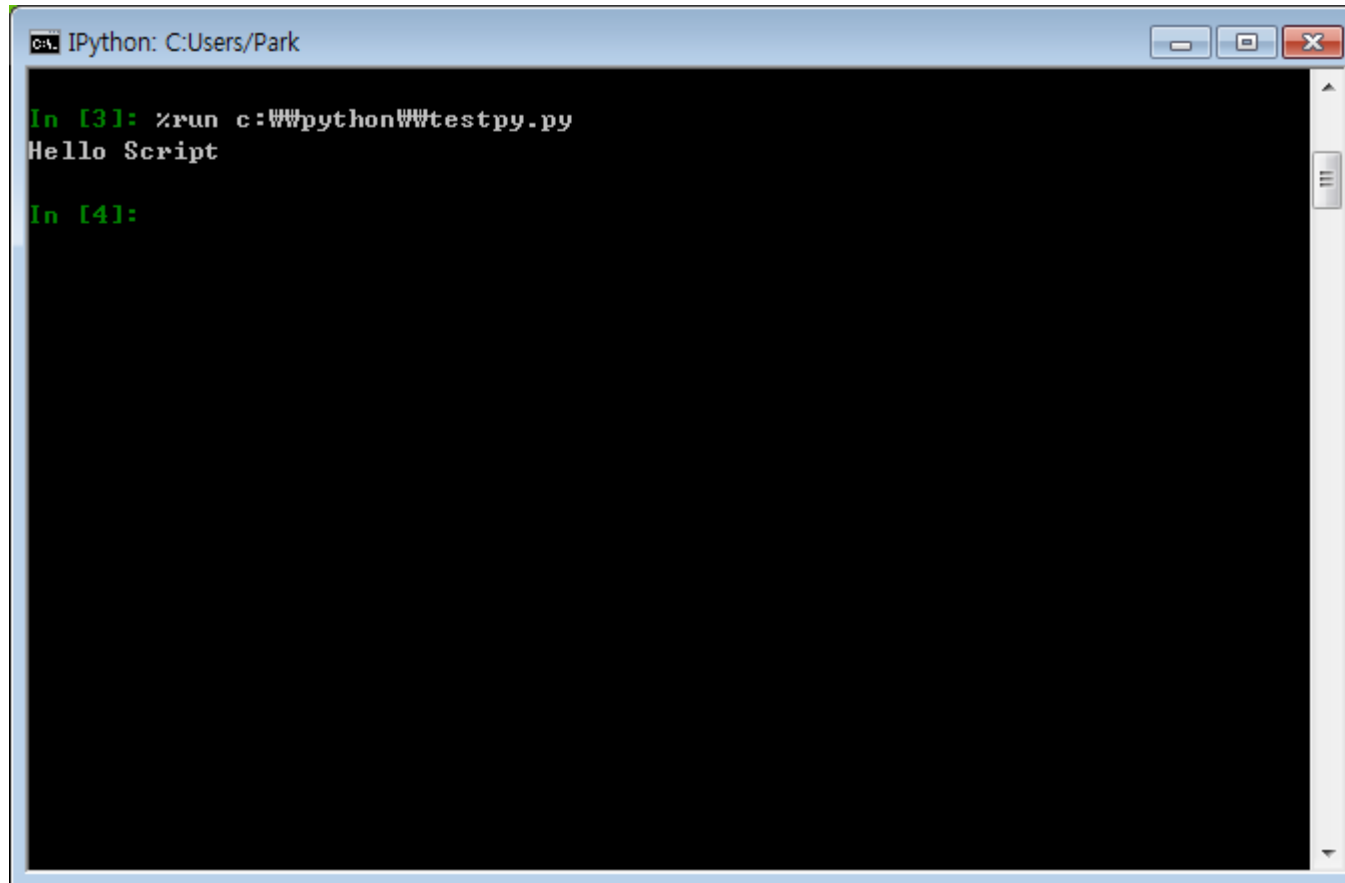
❖ 작업 디렉토리 변경

In [4]: `os.chdir('c:\python')`



4.ipython

- ❖ %run 파일경로: 파일 경로에 해당하는 파이썬 파일 실행
- ❖ 아래 코드를 갖는 파이썬 파일을 작성하고 실행
`print("Hello Script")`



The screenshot shows an IPython terminal window titled "IPython: C:\Users\Park". The terminal has a black background with green text. The first prompt is "In [3]:", followed by the command "%run c:\www\python\www\testpy.py". The output of this command is "Hello Script". The second prompt is "In [4]:", which is currently empty.

```
IPython: C:\Users\Park
In [3]: %run c:\www\python\www\testpy.py
Hello Script
In [4]:
```


4.ipython

❖ 단축키

- Ctrl+P 또는 위 화살표 키: 명령어 히스토리를 역순으로 검색하기
- Ctrl+N 또는 아래 화살표 키: 명령어 히스토리에서 최근 순으로 검색하기
- Ctrl+R: readline 명령어 형식의 히스토리 검색(부분 매칭)하기
- Ctrl+Shift+V: 클립보드에서 텍스트 붙여넣기
- Ctrl+C: 현재 실행 중인 코드 중단하기
- Ctrl+A: 커서의 줄의 처음으로 이동하기
- Ctrl+E: 커서의 줄의 끝으로 이동하기
- Ctrl+K: 커서가 놓은 곳부터 줄의 끝까지 텍스트 삭제하기
- Ctrl+U: 현재 입력된 모든 텍스트 지우기
- Ctrl+F: 커서를 앞으로 한 글자씩 이동하기
- Ctrl+B: 커서를 뒤로 한 글자씩 이동하기
- Ctrl+L: 화면 지우기

4.ipython

❖ 매직 명령어: ipython이 제공하는 특수 명령어

%quickref: ipython의 빠른 도움말 표시

%magic: 모든 매직 함수에 대한 상세 도움말 출력

%debug: 최근 예외 트레이스백의 하단에서 대화형 디버거로 진입

%hist: 명령어 입력(그리고 선택적 출력) history 출력

%pdb: 예외가 발생하면 자동으로 디버거로 진입

%paste: 클립보드에서 들여쓰기가 된 채로 파이썬 코드 가져오기

%cpaste: 실행 파이썬 코드를 수동으로 붙여 넣을 수 있는 프롬프트 표시

%reset: 대화형 네임스페이스에서 정의된 모든 변수와 이름을 삭제

%page OBJECT: 객체를 pager를 통해 보기 좋게 출력

%run script.py: ipython 내에서 파이썬 스크립트 실행

%prun statement: cProfile을 통해 statement를 실행하고 프로파일링 결과를 출력

%time statement: 단일 statement 실행 시간을 출력

%timeit statement: statement를 여러차례 실행한 후 평균 실행 시간을 출력. 매우 짧은 시간 안에 끝나는 코드의 시간을 측정할 때 유용

%who, %who_ls, %whos: 대화형 네임스페이스 내에서 정의된 변수를 다양한 방법으로 표시

%xdel variable: variable을 삭제하고 ipython 내부적으로 해당 객체에 대한 모든 참조를 제거

%%writefile filename: filename인 파일을 생성

4.ipython

```
In [10]: import numpy as np
```

```
In [11]: a = np.random.randn(100,100)
```

```
In [12]: %time np.dot(a, a)
```

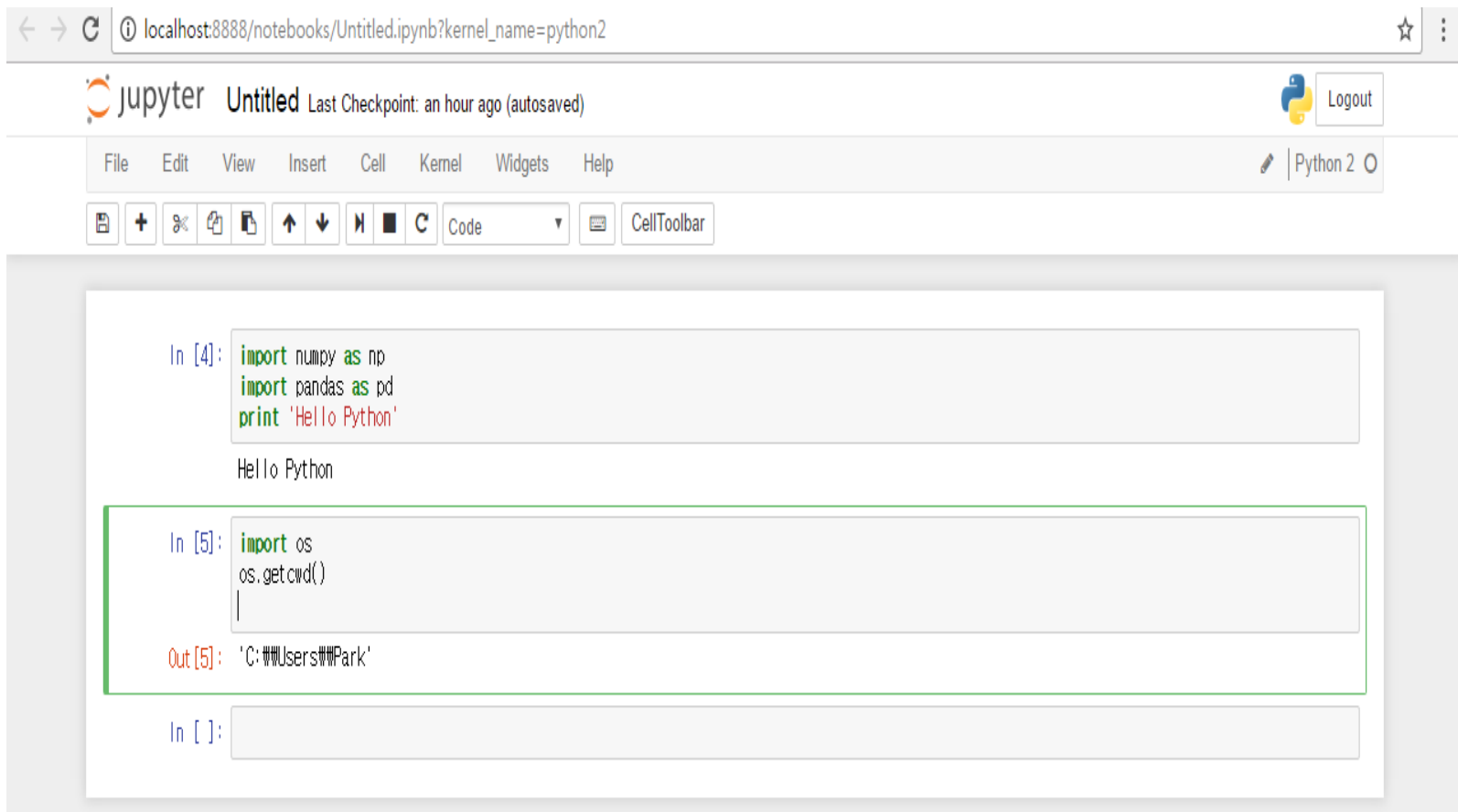
Wall time: 710 ms

```
Out[12]:
```

```
array([[ 19.58800137,  2.62837613, -8.28386112, ..., -4.0982812 ,
        -13.99239531,  5.67487065],
       [-5.55024439, 10.85218715,  6.14092077, ..., -13.73882312,
        -0.72756494,  2.39935928],
       [-3.60873191, -15.89902742,  8.33705822, ..., 10.37198751,
        -2.66928403, -4.6852677 ],
       ...,
       [-1.53904249, -1.2630197 , 25.4836311 , ...,  7.55673741,
         1.06642476,  3.21558587],
       [ 8.5091311 , -14.67747325,  9.04744578, ..., -9.93647063,
         4.96999403, -8.40284447],
       [-6.94798702, -5.71814232, 11.46917944, ...,  5.43738466,
        -8.7403922 , -6.61607408]])
```

5.ipython notebook

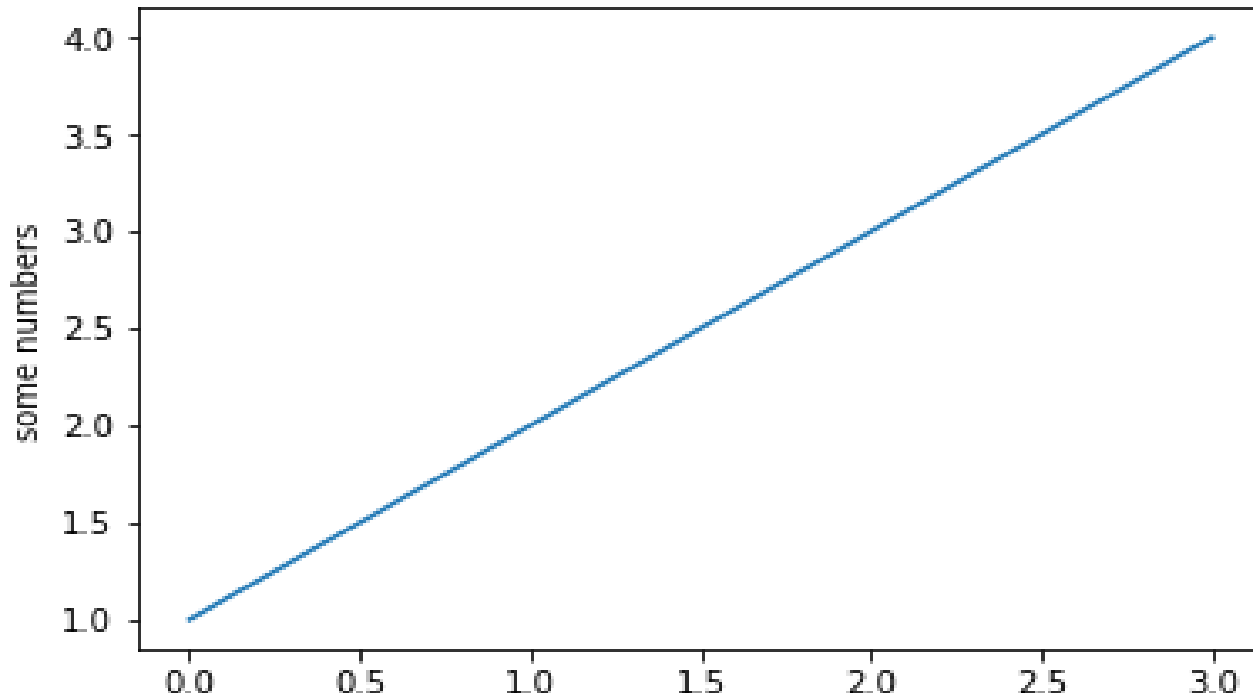
C:\Users\Park>>jupyter notebook



The screenshot displays the Jupyter Notebook web interface in a browser. The address bar shows the URL `localhost:8888/notebooks/Untitled.ipynb?kernel_name=python2`. The notebook title is "Untitled" with a subtext "Last Checkpoint: an hour ago (autosaved)". The top right corner features a "Logout" button. The main menu includes "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu is a toolbar with icons for saving, creating new cells, and other functions. The notebook content area shows two code cells. The first cell, labeled "In [4]:", contains the following code: `import numpy as np`, `import pandas as pd`, and `print 'Hello Python'`. The output of this cell is "Hello Python". The second cell, labeled "In [5]:", contains the code `import os` and `os.getcwd()`. The output of this cell is `Out[5]: 'C:\\Users\\Park'`. A third cell labeled "In []:" is partially visible at the bottom.

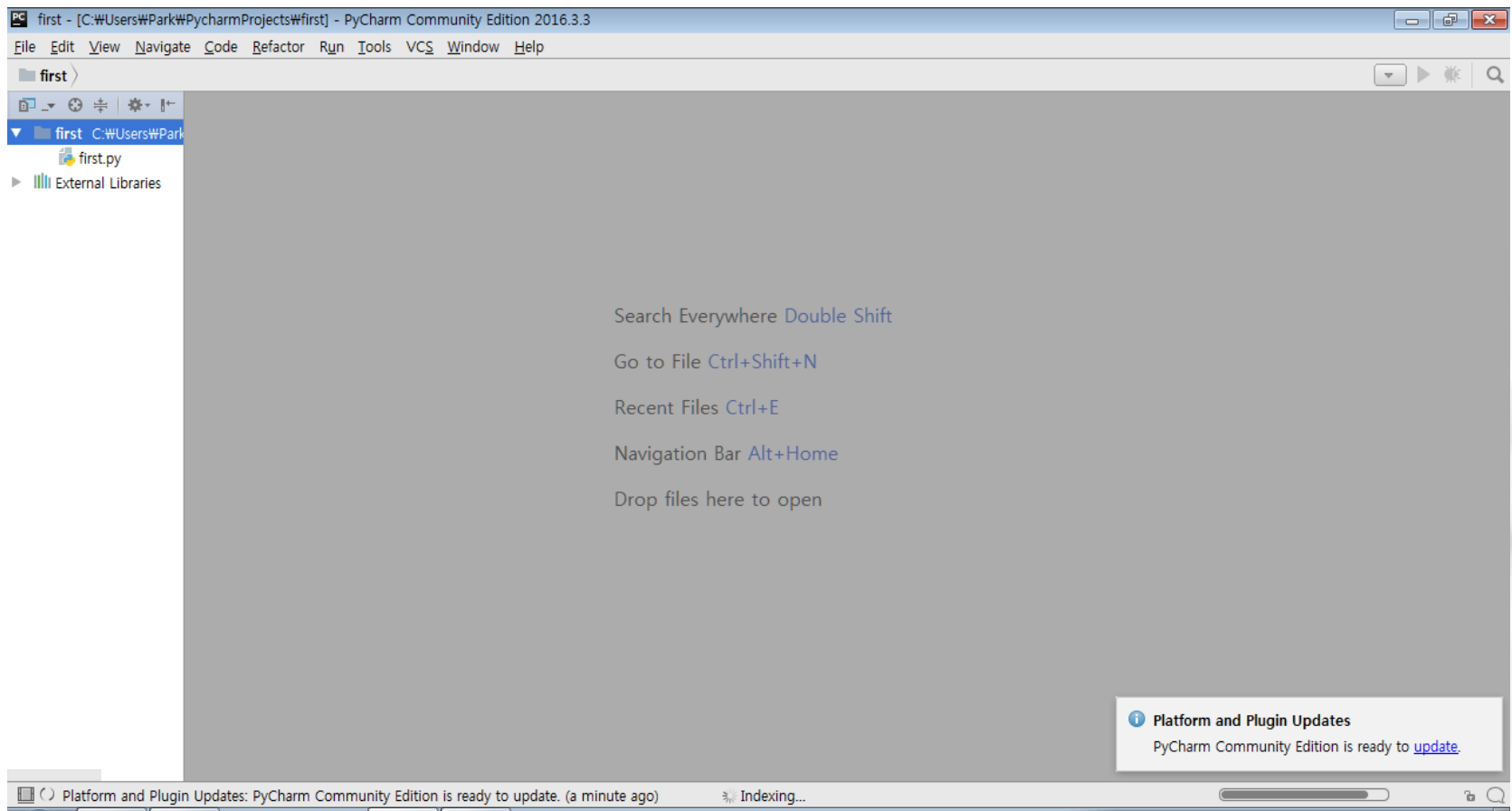
5.ipython notebook

```
import matplotlib.pyplot as plt  
plt.plot([1,2,3,4])  
plt.ylabel('some numbers')  
plt.show()
```

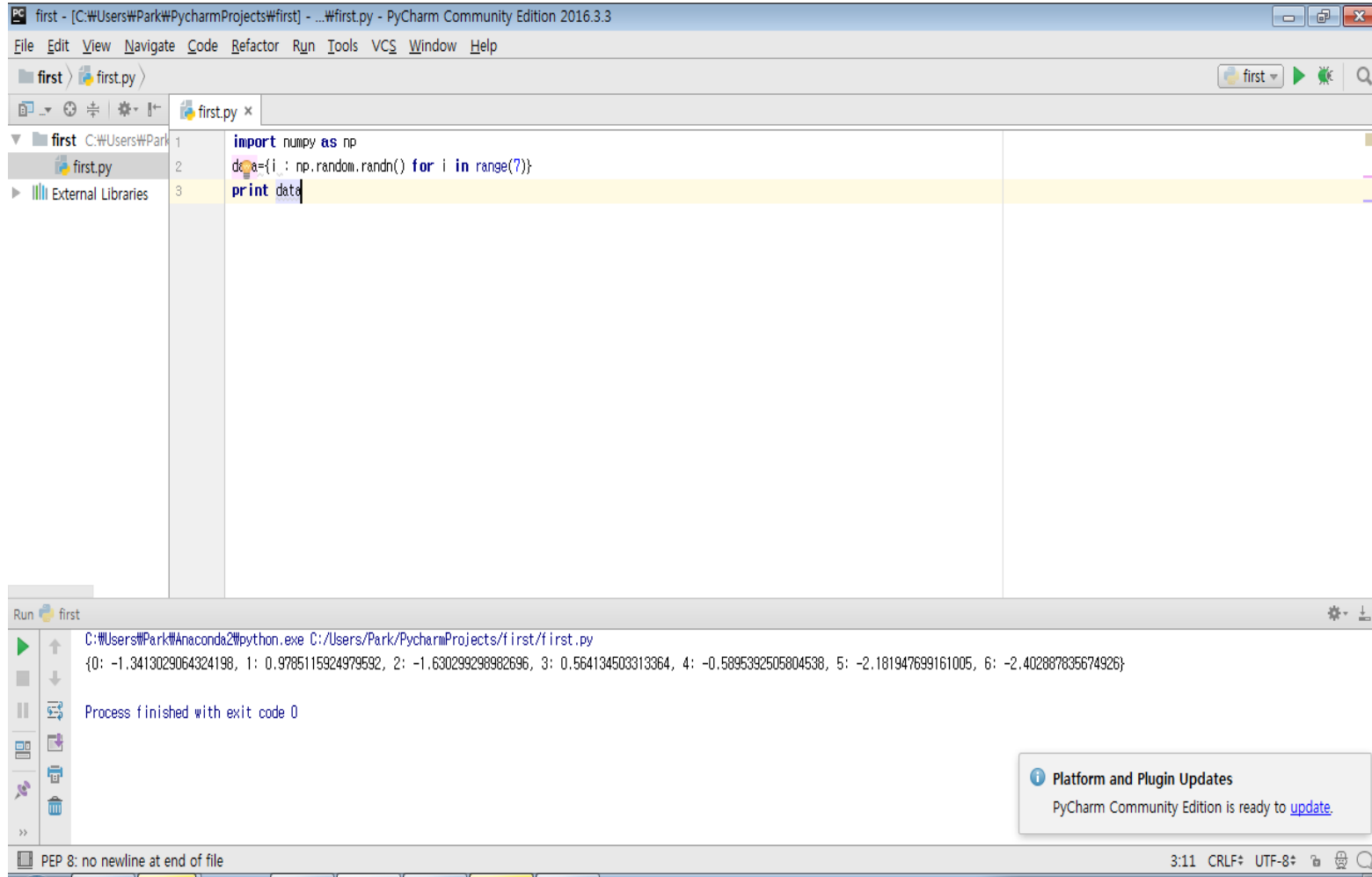


6. pycharm

- ❖ Pycharm은 IntelliJ, CLion 등을 제공하는 JetBRAINS에서 배포하는 프로그램입니다.
- ❖ <http://www.jetbrains.com/pycharm/download/#section=windows> 에서 Community 버전을 선택해서 다운로드 후 설치



6. pycharm

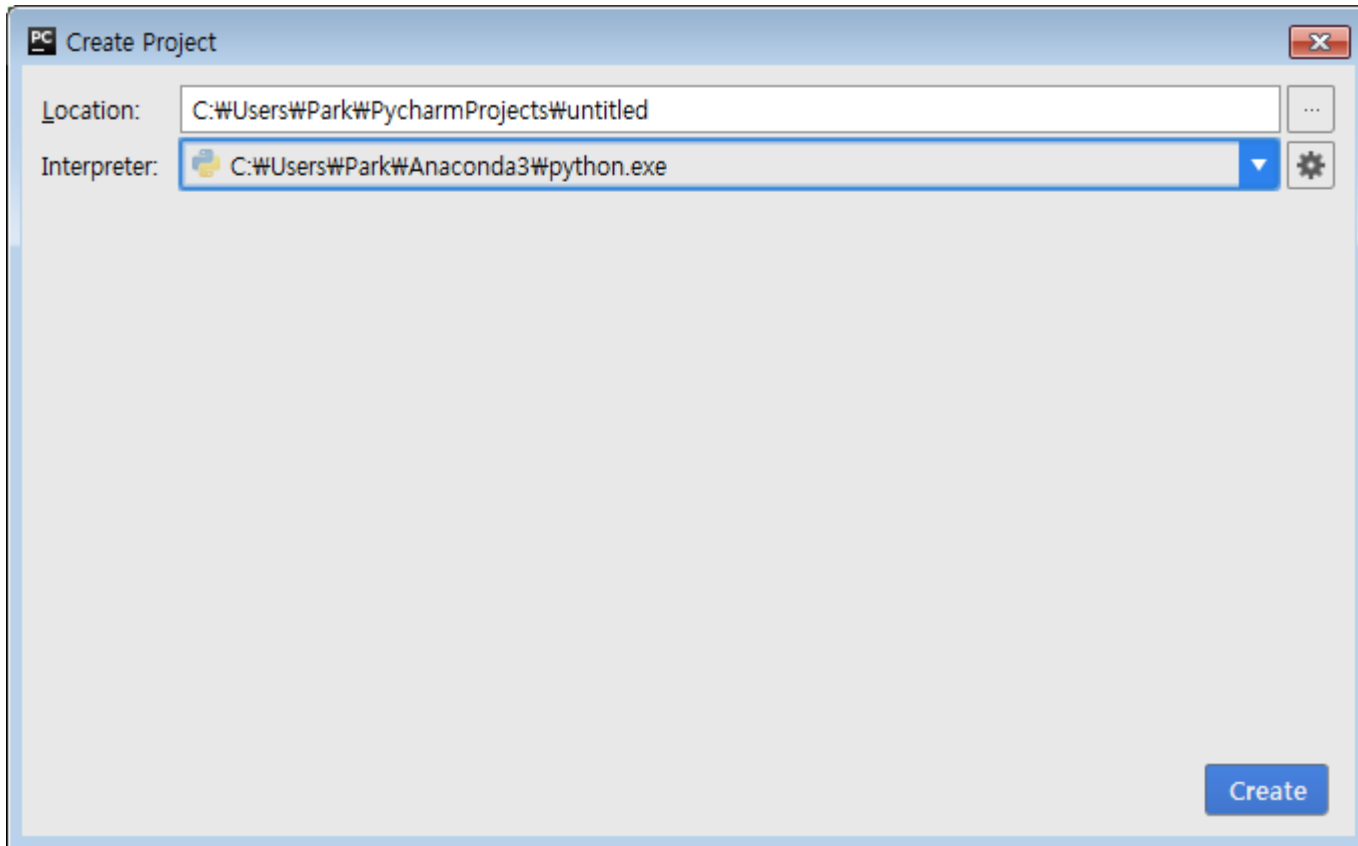


6. pycharm

- ❖ 가상 환경(Virtual Environment): 사용자가 정한 임의의 디렉토리 밑에 Python과 관련 패키지 등을 함께 넣어 그 안에서 독립적인 파이썬 개발 환경을 할 수 있도록 한 것으로 Lightweight, Self-contained 파이썬 개발 환경으로서 필요한 경우 한 개발 머신 안에 여러 개의 가상 환경을 만들고 각 가상 환경에서 다른 파이썬 버전이나 다양한 패키지들을 독립적으로 설치 사용할 수 있는 가상적 개발 환경
- ❖ Python 3 (3.4+)는 기본적으로 가상환경을 생성하는 유틸리티를 포함하고 있습니다.
- ❖ 가상 환경을 만들기 위해서는 pyvenv 라는 유틸리티를 사용하는데, 각 OS 마다 (윈도우즈, Mac, 리눅스) 사용법이 약간씩 다릅니다.
- ❖ Pycharm에서는 pyvenv 유틸리티가 번들로 포함되어 있으므로 가상환경 생성이 쉽습니다.

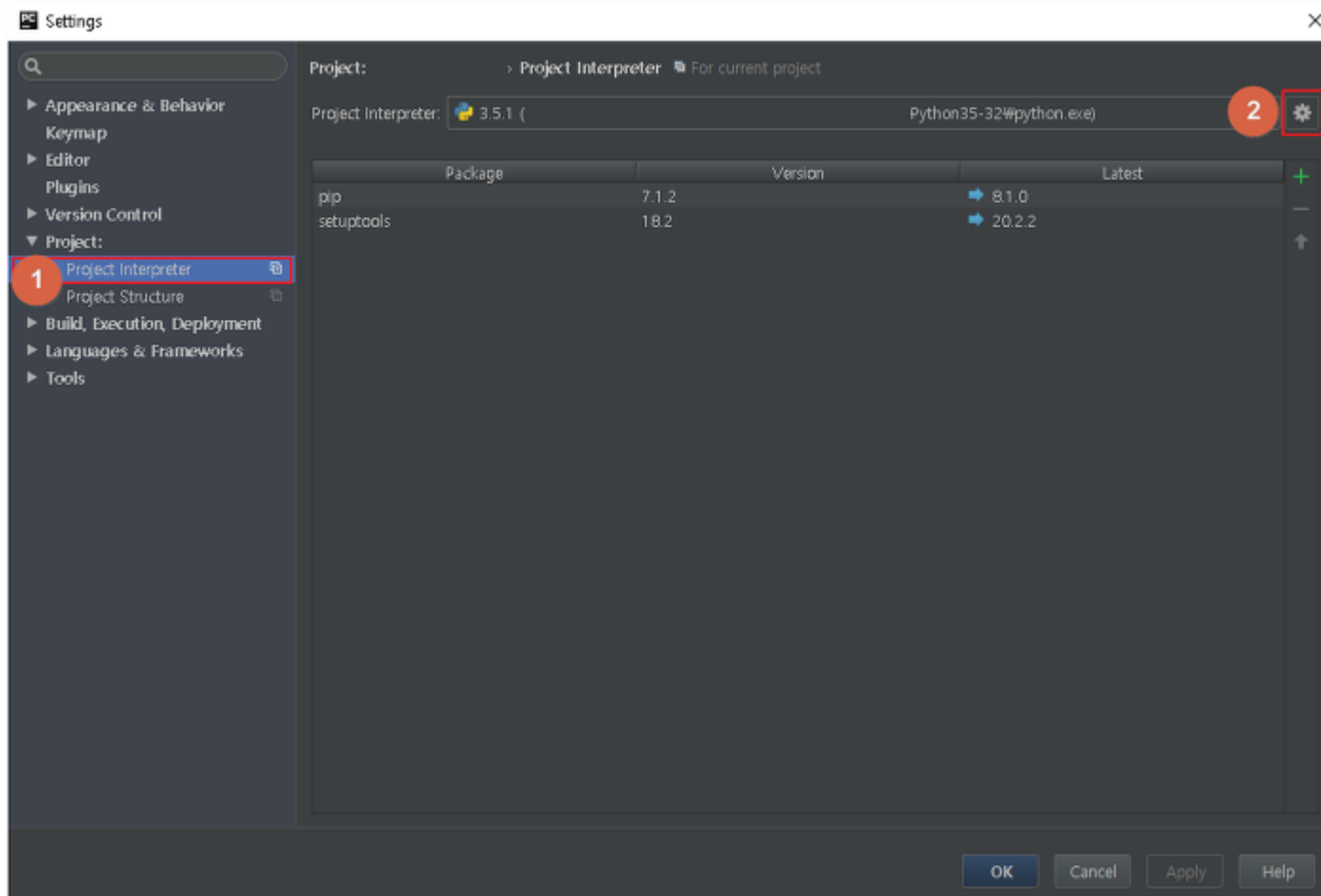
7. Pycharm 가상환경 구축

- ❖ 프로젝트 생성 [File] – [New Project]



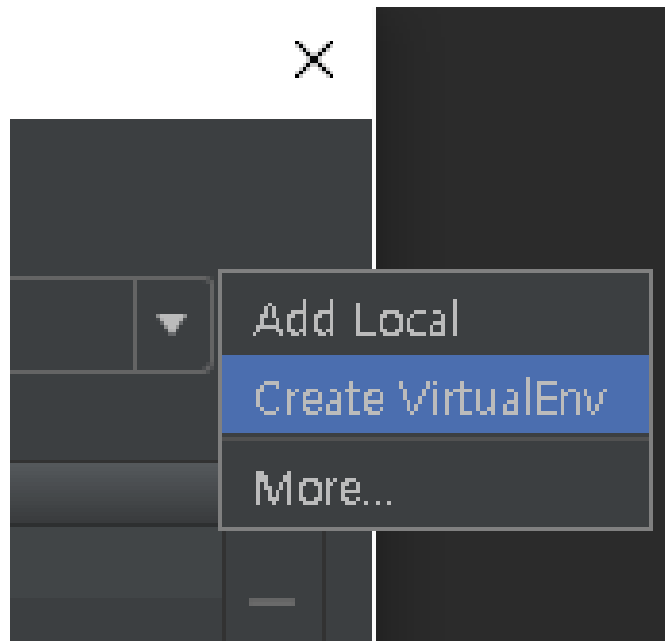
7. Pycharm 가상환경 구축

❖ 프로젝트 생성 [File] – [settings]



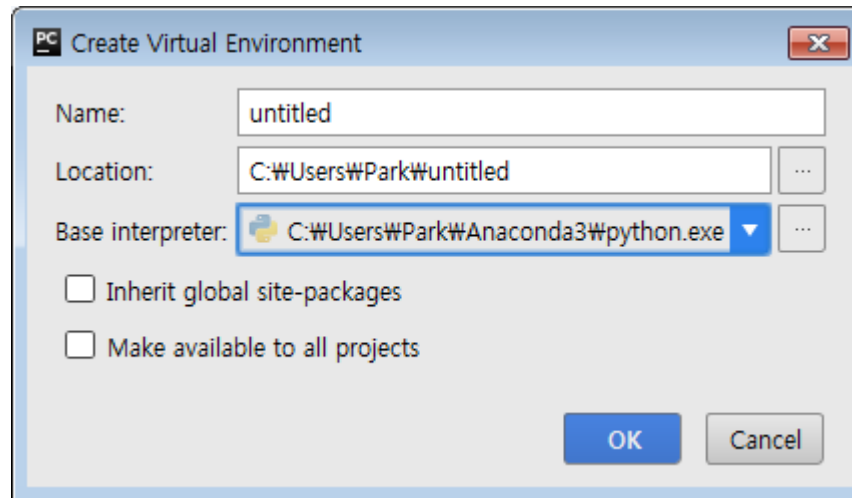
7. Pycharm 가상환경 구축

❖ Create VirtualEnv 선택



7. Pycharm 가상환경 구축

❖ 옵션 설정



- ✓ Name: 가상 환경 이름
- ✓ Location: 가상 환경이 생성될 경로
- ✓ Base interpreter: 사용할 파이썬 버전
- ✓ Inherit global site-packages: 기본 인터프리터의 site-packages를 상속받아 사용할 경우 체크
- ✓ Make available to all projects: 모든 프로젝트에서 사용할 경우 체크

8. Python Basic

❖ 파이썬과 다른 언어의 다른 차이점은 코딩 구조

- ✓ 구조적 프로그래밍 언어의 대부분은 특정한 부분을 구분 짓기 위해서 특정한 기호를 사용하여 그 시작과 끝을 표시하는 엄격한 구조를 따라야만 했는데 파이썬은 이러한 명령어가 없습니다.
- ✓ 파이썬에서는 코드의 구조를 정의하기 위해 기호 대신에 들여쓰기를 사용
- ✓ Java 나 C와 같은 언어에서 코드 블록을 여닫기 위하여 사용하는 중괄호 대신 파이썬에서는 공백을 사용하며 이는 코드를 읽기 쉽게하고 코드 내에서 불필요한 기호의 사용을 줄여줍니다.
- ✓ 코드의 정렬과 구성이 엄격하게 제한되는데, 네 칸를 들여쓰는 것이 표준이기는 해도 프로그래머가 들여쓰기의 규칙을 정할 수 있습니다.
- ✓ 하나의 문장을 종료할 때 종료기호는 없음
- ✓ 하나의 줄에 2개 이상의 명령어를 사용할 때 명령어를 구분하기 위한 용도로 ;을 사용

8. Python Basic

C 언어

```
#include <stdio.h>

int main()
{
    int i, sum=0;
    for(i=1; i<=10; i++)
    {
        sum+=i;
    }
    printf("%d", sum);
}
```

파이썬 언어

```
sum=0
for i in range(1,11):
    sum+=i
print(sum)
```

8. Python Basic

❖ 자바코드

```
int x = 100;
if(x > 0){
    System.out.println("x가 0보다 큽니다.");
} else {
    System.out.println("x가 0보다 작습니다.");
}
```

❖ 파이썬 코드(2.x에서는 #coding:utf-8을 상단에 추가)

```
x = 100
if x > 0:
    print('x가 0보다 큽니다.')
else:
    print('x가 0보다 크지 않습니다.')
```

8. Python Basic

- ❖ 도움말을 실행하고자 하는 경우에는 [F1] 키를 눌러도 되고 IDLE 창에서 help(도움말을 얻고자 하는 명령)
 - ❖ Python의 예약어
 - ✓예약어 (Reserved Words, keyword):파이썬에서 이미 문법적인 용도로 사용되고 있는 단어 또는 문자
 - ✓사용자 정의 식별자로 사용하면 예약어의 기능을 잃어버리는 단어들
 - ✓아래처럼 작성하면 예약어 확인 가능
- ```
import keyword
keyword.kwlist
```
- ['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']**



## 8. Python Basic

- ❖ 파이썬에서 주석을 달려면 다음과 같이 # 이런 샤프 기호를 사용하면 됩니다. 그러면 # 기호 뒤에 있는 모든 문자열들이 실행에서 무조건 무시됩니다.
- ❖ 어떤 특정 구역을 모두 주석화시키려면 """ 에서 """ 까지, 이렇게 큰따옴표(쌍따옴표) 3개를 사용하면 됩니다.
- ❖ 소스 첫줄의 #! 기호는 주석문이 아닙니다. "유닉스 Shebang"입니다.
- ❖ 둘째 줄의 # 는 한글 확장완성형 인코딩 지정문입니다.

# 8. Python Basic

## ❖ 파이썬 코드

```
#!/usr/bin/python
```

```
-*- coding: cp949 -*-
```

```
이줄은 라인 코멘트입니다
```

```
print("Hello World!")
```

```
print("Hello World!") # 이것도 라인 코멘트입니다
```

```
print("Hello World!") # 이것도 라인 코멘트입니다
```

```
print("Hello World!") # 이것도 라인 코멘트입니다
```

```
"""
```

```
이것은 블록 코멘트입니다.
```

```
그래서 여러 줄의 주석을
```

```
이렇게 달 수 있습니다.
```

```
큰따옴표 3개를 연속으로 적으면 됩니다.
```

```
"""
```

```
print("Hello World!") # 라인 코멘트
```

# Python 기본문법

# 1. 식별자

- ❖ 파이썬 식별자는 변수, 함수, 모듈, 클래스 또는 객체를 식별하는데 사용되는 이름
  - ✓ 대소문자 구별함
  - ✓ 식별자는 문자 A~Z 또는 a~z과 언더바(\_)로 시작
  - ✓ 식별자 첫 시작 문자를 제외하고 식별자 내에 숫자(0~9)를 사용할 수 있음
  - ✓ 특수문자 @, \$, %등은 식별자에 올 수 없음
  - ✓ 예를 들어 다음과 같은 것은 식별자가 될 수 없음 : 1abc, @file, %x

## 2. 변수와 상수

### ❖ 상수(Constant, Literal):값을 변경할 수 없는 데이터

#### ✓ number

- 정수:10진수(107), 8진수(0107), 16진수(0x107, 0X107), 2진수(0b101), long(107L)  
=>파이썬 2.x 버전에서는 L을 붙여서 long 형을 만들 수 있지만 3.x에서는 정수 상수는 전부 int 형으로 만들기 때문에 L을 붙일 수 없습니다.
- 실수:1.2, 0.12e1, 0.12E1 -> 주의: 지수부는 정수일 수 있으나 실수 일 수 없음
- 복소수: 허수부 뒤에 j 나 J 사용 4+5j, 7-2j

#### ✓ str:' 문자열' 또는 "문자열" 의 형태로 기재하며 여러 줄의 문자열을 하나의 문자열로 만들 때는 ''' 문자열''' 또는 """문자열"""

#### ✓ list:[데이터 나열]

#### ✓ tuple:(데이터 나열)

#### ✓ dict:{키:값...}

#### ✓ bool 상수:True(1), False(0)

#### ✓ 제어문자:\n(줄바꿈), \t(탭)

#### ✓ None: Types.NoneType 의 유일한 값으로 존재하지 않는 변수에 대입하여 이 변수에 아무런 값이 없다는 것을 나타내기 위해 활용

## 2. 변수와 상수

- ❖ 변수(Variable): 데이터를 저장할 수 있는 공간에 붙여놓는 이름
- ❖ Python은 변수를 선언할 때 자료형을 기재하지 않음
- ❖ 변수에 값을 할당 할 때 데이터 타입을 자동으로 설정
- ❖ 등호 (=)는 변수에 값을 할당하는 데 사용
- ❖ 파이썬에서 대입의 의미

a = 1 : 1이라는 값을 저장한 공간의 주소를 a에 대입

- ❖ 변수 명명 규칙
  - ✓ 식별자 규칙을 적용
  - ✓ 예약어, 내장함수, 모듈 이름을 변수명으로 만드는 일이 없도록 해야 하는데 예약어를 변수명으로 사용하게 되면 원래의 기능을 잃어버리게 됩니다.
- ❖ 변수의 삭제는 **del 변수명**

## 2. 변수와 상수

```
counter = 100 # 정수 할당
miles = 1000.0 # 부동 소수점
name = "홍길동" # 문자열
```

```
print(counter)
print(miles)
print(name)
```

### 3. 연산자

- ❖ 여러 줄을 하나의 문장으로 만들고자 할 때는 중간에 줄 연속 문자(**W**)를 추가해주면 됩니다.
- ❖ `[], { }`에 포함 된 문이나 `( )` 괄호는 줄 연속 문자를 사용할 필요가 없습니다.
- ❖ 하나의 줄에서 문장을 구분하는 연산자는 **;**
- ❖ 산술연산자
  - ✓ **+**연산자: 숫자의 경우는 덧셈을 하고 문자열이나 데이터의 모임은 결합을 수행
  - ✓ **\***연산자: 문자열의 경우는 반복을 나타내고 숫자의 경우는 곱셈
  - ✓ **\*\***연산자: 거듭제곱( $3^{**}3$  은 3의 3제곱)
  - ✓ **/**연산자: 나눗셈을 한 결과 - 결과는 실수
  - ✓ **//**연산자: 몫을 정수로 구해줍니다.
- ❖ 여러 개를 한번에 대입 가능  
 $a = b = 0$
- ❖ 연속해서 여러 개의 변수에 값 할당 가능한데 순서대로 대입  
 $c, d = 3, 4$



# 3. 연산자

❖ 샘플 코드

```
x=y=z=0
print(x)
print(y)
print(z)
c,d=3,4
c,d=d,c
print(c)
print(d)
```



# 3. 연산자

## ❖ 비교 연산자: 연산의 결과가 True 또는 False

- ✓  $a = 10, b = 20$  이라 가정
- ✓  $==$ : 값이 동일 ( $a == b$ )  $\rightarrow$  False
- ✓  $!=$ : 값이 동일하지 않음 ( $a != b$ )  $\rightarrow$  True
- ✓  $>$ : 왼쪽 값이 오른쪽 값보다 크다 ( $a > b$ )  $\rightarrow$  False
- ✓  $<$ : 왼쪽 값이 오른쪽 값보다 작다 ( $a < b$ )  $\rightarrow$  True
- ✓  $>=$ : 왼쪽 값이 오른쪽 값보다 크거나 동일하다 ( $a >= b$ )  $\rightarrow$  False
- ✓  $<=$ : 왼쪽 값이 오른쪽 값보다 작거나 동일하다 ( $a <= b$ )  $\rightarrow$  True

## ❖ 비트 연산자: 정수 데이터끼리 비트 단위로 연산을 수행 한 후 결과를 10진 정수로 리턴하는 연산자

- ✓  $\&$ : AND 연산으로 둘 다 1일 때만 1  $\rightarrow (a \& b) = 12 \rightarrow 0000\ 1100$
- ✓  $|$ : OR 연산. 둘 중 하나만 1이어도 1  $\rightarrow (a | b) = 61 \rightarrow 0011\ 1101$
- ✓  $\wedge$ : XOR 연산. 두 개의 데이터가 다르면 1  $\rightarrow (a \wedge b) = 49 \rightarrow 0011\ 0001$
- ✓  $\sim$ : 1의 보수 연산. ( $\sim a$ ) = -61  $\rightarrow 1100\ 0011$
- ✓  $<<$ : 왼쪽 시프트 연산자. 변수의 값을 왼쪽으로 지정된 비트 수 만큼 이동  $a << 2 = 240 \rightarrow 1111\ 0000$
- ✓  $>>$ : 오른쪽 시프트 연산자. 변수의 값을 오른쪽으로 지정된 비트 수 만큼 이동  $a >> 2 = 15 \rightarrow 0000\ 1111$

# 3. 연산자

- ❖ 논리 연산자: bool 식을 연산해서 결과를 bool 타입으로 리턴하는 연산자
  - ✓  $a = \text{True}, b = \text{False}$  이라 가정
  - ✓ and: 논리 AND 연산. 둘 다 참 일 때 만 참 ( $a \text{ and } b$ ) = False
  - ✓ or: 논리 OR 연산. 둘 중 하나만 참 이여도 참 ( $a \text{ or } b$ ) = True
  - ✓ not: 논리 NOT 연산. 논리 상태를 반전  $\text{not}(a \text{ and } b) = \text{True}$
- ❖ 멤버 연산자: 데이터 모임의 멤버 인지 확인 가능한 연산자
  - ✓  $a = 10, b = 10, \text{list} = [1, 2, 3, 4, 5]$  라 가정
  - ✓ in: 컬렉션 내에 포함되어 있으면 참 ( $a \text{ in list}$ ) = False
  - ✓ not in: 컬렉션 내에 포함되어 있지 않으면 참 ( $b \text{ not in list}$ ) = True
- ❖ 식별 연산자: 가리키고 있는 곳이 같은지 여부를 확인하는 연산자
  - ✓  $a = 20, b = 20$  이라 가정
  - ✓ is: 개체메모리 위치나 값이 같다면 참 ( $a \text{ is } b$ ) = True
  - ✓ is not: 개체메모리 위치나 값이 같지 않다면 참 ( $a \text{ is not } b$ ) = False

# 3. 연산자

- ❖  $+=, -=, *=, /=, //=, \%=, **=, >>=, <<=, \&=, ^=, \&=, |=$ 
  - ✓  $+=$  왼쪽 변수에 오른쪽 값을 더하고 결과를 왼쪽변수에 할당  $c += a \rightarrow c = c + a$
  - ✓  $-=$  왼쪽 변수에서 오른쪽 값을 빼고 결과를 왼쪽변수에 할당  $c -= a \rightarrow c = c - a$
  - ✓  $*=$  왼쪽 변수에 오른쪽 값을 곱하고 결과를 왼쪽변수에 할당  $c *= a \rightarrow c = c * a$
  - ✓  $/=$  왼쪽 변수에서 오른쪽 값을 나누고 결과를 왼쪽변수에 할당  $c /= a \rightarrow c = c / a$
  - ✓  $\%=$  왼쪽 변수에서 오른쪽 값을 나눈 나머지의 결과를 왼쪽변수에 할당  $c \% = a \rightarrow c = c \% a$
  - ✓  $**=$  왼쪽 변수에 오른쪽 값만큼 제곱을 하고 결과를 왼쪽변수에 할당  $c ** = a \rightarrow c = c ** a$
  - ✓  $//=$  왼쪽 변수에서 오른쪽 값을 나눈 몫의 결과를 왼쪽변수에 할당  $c //= a \rightarrow c = c // a$
- ❖ `type(데이터)`: 데이터의 자료형을 문자열 리턴
- ❖ `id(데이터)`: 저장위치를 구분하기 위한 코드 리턴

# 4. 내장함수(Built-in Function)

- ❖ 별도의 모듈의 추가없이 기본적으로 제공되는 함수들
- ❖ 내장 함수: <https://docs.python.org/2/library/functions.html>

Python » 2.7.13 » Documentation » The Python Standard Library »

previous | next | modules | index

Table Of Contents

2. Built-in Functions

3. Non-essential Built-in Functions

Previous topic

1. Introduction

Next topic

4. Built-in Constants

This Page

Report a Bug

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

2. Built-in Functions

The Python interpreter has a number of functions built into it that are always available. They are listed here in alphabetical order.

| Built-in Functions |             |              |             |                |
|--------------------|-------------|--------------|-------------|----------------|
| abs()              | divmod()    | input()      | open()      | staticmethod() |
| all()              | enumerate() | int()        | ord()       | str()          |
| any()              | eval()      | isinstance() | pow()       | sum()          |
| basestring()       | execfile()  | issubclass() | print()     | super()        |
| bin()              | file()      | iter()       | property()  | tuple()        |
| bool()             | filter()    | len()        | range()     | type()         |
| bytearray()        | float()     | list()       | raw_input() | unichr()       |
| callable()         | format()    | locals()     | reduce()    | unicode()      |
| chr()              | frozenset() | long()       | reload()    | vars()         |
| classmethod()      | getattr()   | map()        | repr()      | xrange()       |
| cmp()              | globals()   | max()        | reversed()  | zip()          |
| compile()          | hasattr()   | memoryview() | round()     | __import__()   |
| complex()          | hash()      | min()        | set()       |                |
| delattr()          | help()      | next()       | setattr()   |                |
| dict()             | hex()       | object()     | slice()     |                |
| dir()              | id()        | oct()        | sorted()    |                |

In addition, there are other four built-in functions that are no longer considered essential: `apply()`, `buffer()`, `coerce()`, and `intern()`. They are documented in the Non-essential Built-in Functions section.

**abs(*x*)**

Return the absolute value of a number. The argument may be a plain or long integer or a floating point number. If the argument is a complex number, its magnitude is returned.

**all(*iterable*)**

Return `True` if all elements of the *iterable* are true (or if the iterable is empty). Equivalent to:

## 4. 내장함수(Built-in Function)

❖ `max(s)`: 시퀀스 자료형(문자열, 리스트, 튜플)을 입력받아 그 자료가 지닌 원소 중 최대값을 반환하는 함수

❖ `range([start,]stop[,step])`: 수치 자료형으로 `start`, `stop`, `step` 등을 입력받아 해당 범위에 해당하는 정수를 `range` 타입으로 반환하는 함수

- ✓ 인수가 하나(`stop`)인 경우: 0부터 `stop-1`까지의 정수 리스트를 반환한다.
- ✓ 인수가 두 개(`start`, `stop`)인 경우: `start`부터 `stop-1`까지의 정수 리스트를 반환한다.
- ✓ 인수가 세 개(`start`, `stop`, `step`)인 경우: `start`부터 `stop-1`까지의 정수를 반환하되 각 정수 사이의 거리가 `step`인 것들만 반환한다.

## 4. 내장함수(Built-in Function)

```
print(max(10,30, 50))
print(max([100,300,200]))
print(max('Hello World'))
print(range(10))
print(range(3, 10))
print(range(3, 10, 2))
```

```
50
300
r
range(0, 10)
range(3, 10)
range(3, 10, 2)
```

## 5. 콘솔 입출력

- ❖ 콘솔 (Console): Windows에서는 Command창, 리눅스/맥에서는 Terminal창 각 IDE (예, 이클립스)에서는 별도의 콘솔 창이 제공됨
- ❖ `print(데이터 나열)`: 화면으로 데이터를 출력할 때 가장 보편적으로 사용되는 함수로 데이터를 출력하고 줄 바꿈을 수행
- ❖ 여러 개의 데이터를 ,로 구분해서 출력할 수 있습니다.

```
print (1,2); print (3,4)
```

```
1 2
```

```
3 4
```

- ❖ `end`라는 매개변수를 이용해서 줄 바꿈을 대신할 문자열을 추가할 수 있음

```
print(1,2, end=' '); print(3,4)
```

```
1 2 3 4
```

- ❖ `sep` 매개변수를 이용해서 데이터를 구분하는 문자열을 추가할 수 있음

```
print(1,2,3,4,5, sep='\\t')
```

```
1 2 3 4 5
```



## 5. 콘솔 입출력

❖ `format(데이터, format_spec)`: `format`에 맞추어서 데이터를 문자열로 리턴

```
print(format(4, '10d'))
```

```
print(format(43.3, '10.3f'))
```

```
print(format('안녕하세요!', '10s'), format('반갑습니다.', '10s'))
```

❖ `{숫자}`와 `format()`을 이용한 데이터 매핑

```
print('{0} is {1}'.format('Python', 'fun'))
```

# 5. 콘솔 입출력

## ❖ input('메시지')

- ✓ 문자열을 입력받는 내장함수
- ✓ 매개변수로 하나의 문자열을 받는데 매개변수로 입력된 내용은 키보드의 입력을 알려주는 프롬프트
- ✓ Enter를 누를 때까지 입력을 받아서 하나의 문자열로 리턴합니다.
- ✓ 정수나 실수형으로 사용할 때는 문자열을 입력받고 난 후 수치형으로 변환해야 합니다.

```
name = input('이름:')
print(name)
age = int(input('나이:'))
print(age)
print(type(age))
```

이름:박문석  
박문석  
나이:34  
34

## 6. 제어문

- ❖ 프로그램은 일반적으로 위에서 아래로 한 문장씩 수행되는데 이것을 순차적인 제어 흐름이라고 합니다.
- ❖ 때로는 이러한 제어 흐름에서 벗어날 때도 있습니다.
- ❖ 건너뛰기도 해야하고 반복하기도 해야 합니다.
- ❖ 파이썬의 제어문은 if, for, while 만 존재합니다.
- ❖ 파이썬의 코드 블록 작성 규칙
  - ✓ 가장 바깥쪽에 있는 블록의 코드는 반드시 1열부터 시작
  - ✓ 내부 블록은 같은 거리만큼 들여쓰기
  - ✓ 블록은 {}, begin, end 등으로 구분하지 않고 들여쓰기만으로 구분
  - ✓ 탭과 공백을 함께 사용하는 것은 권장하지 않음
  - ✓ 들여쓰기 간격은 일정하기만 하면 됩니다.

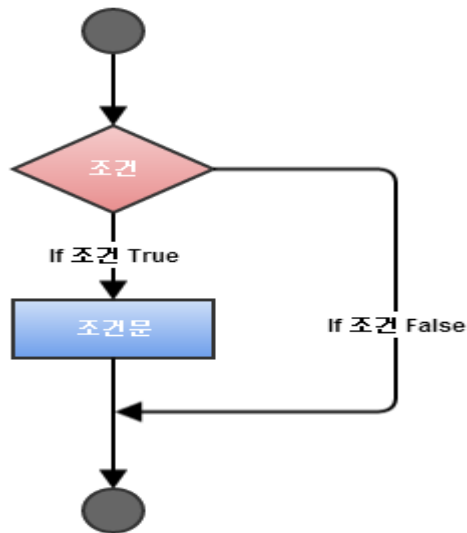
## 6. 제어문

### ❖ 단순if

if expression :  
code block;

### ❖ expression

- ✓boolean 경우 : 표현식(expression)이 사실이면 true를 리턴해 if문 안쪽 문(statement)이 실행되고, 거짓이면 false를 리턴해 다음으로 진행 한다.
- ✓value 경우: non-zero, non-null → true, zero, null → false 로 리턴



## 6. 제어문

### ❖예제

```
var1 = 100
```

```
if var1:
```

```
 print("1 - true 일 때 수행될 문장")
```

```
 print(var1)
```

```
var2 = 0
```

```
if var2:
```

```
 print("2 - true 일 때 수행될 문장")
```

```
 print(var2)
```

```
print("Good bye!")
```

### ❖결과

1 - true 일 때 수행될 문장

100

Good bye!

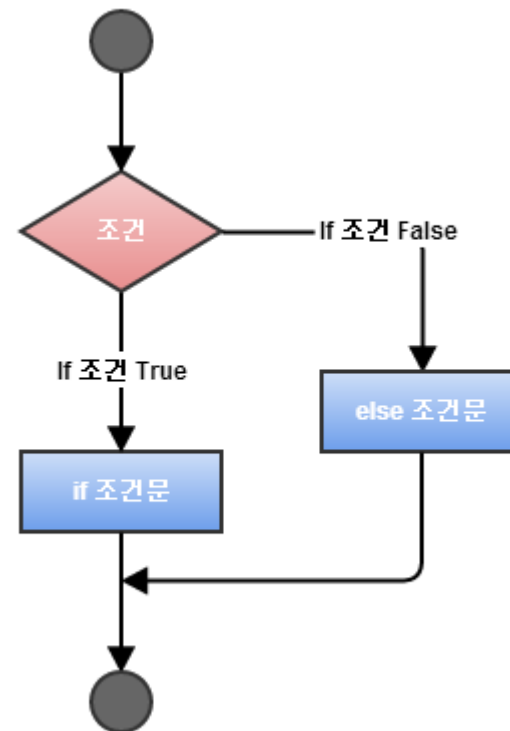
## 6. 제어문

### ❖if-else

if expression :  
    code block;  
else:  
    code block;

### ❖삼항 연산자

statement if expression else statement



## 6. 제어문

### ❖if-elif-else

if와 함께 elif 절 사용

```
if 조건1:
 코드블록
 ...
elif 조건2:
 코드블록
 ...
elif 조건3:
 코드블록
 ...
elif 조건4:
 코드블록
 ...
else:
 코드블록
 ...
```

첫 번째 조건은 항상 if로 시작합니다.

두 번째 조건부터는 elif를 이용합니다.

마지막의 else는 생략할 수 있습니다.

## 6. 제어문

### ❖연습

월-일 까지의 문자열을 입력하면 영문으로 요일을 표시하는 프로그램을 작성

```
dow = input('요일(월~일)을 입력하세요 : ')
```

```
if dow == '월':
 print('Monday')
elif dow == '화':
 print('Tuesday')
elif dow == '수':
 print('Wednesday')
elif dow == '목':
 print('Thursday')
elif dow == '금':
 print('Friday')
elif dow == '토':
 print('Saturday')
elif dow == '일':
 print('Sunday')
else:
 print('잘못 입력된 요일입니다.')
```



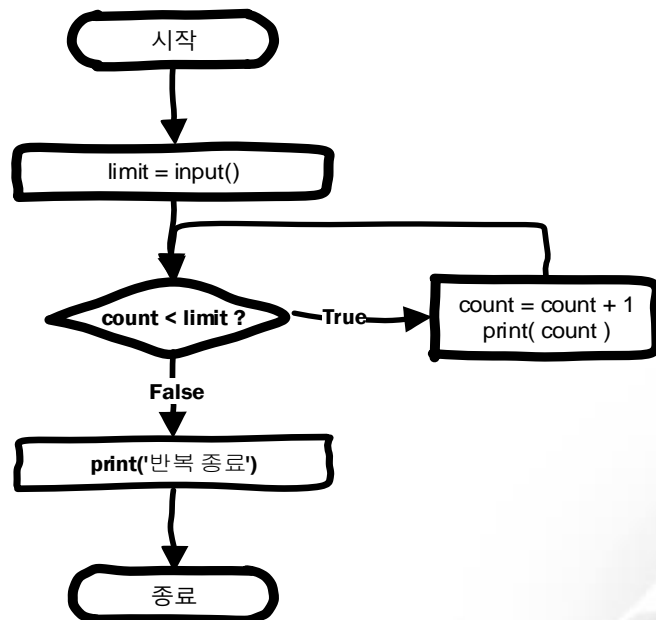
## 6. 제어문

❖ while – 반복문

while 조건:

코드블록

=> 조건이 참인 동안 코드 블록을 수행



```
print('몇 번 반복할까요? : ')
limit = int(input())

count = 0
while count < limit:
 count = count + 1
 print('{0}회 반복.'.format(count))

print('반복이 종료되었습니다.')
```

## 6. 제어문

❖ 구구단 2단 출력

```
i = 0;
```

```
while i < 9:
```

```
 i = i + 1
```

```
 print('2 * {0} = {1}'.format(i, 2*i))
```

## 6. 제어문

### ❖while – 반복문

```
while 조건:
 코드블록
else:
 코드블록
```

⇒조건이 참일 때 까지 코드 블록 1을 수행하고 반복문 수행을 종료하거나 조건에 맞지 않으면 코드 블록 2를 수행

### ❖무한반복

while문의 조건이 항상 참이 되는 루프

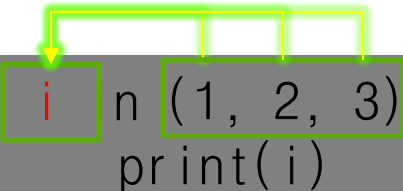
```
while True:
 코드블록
```

## 6. 제어문

### ❖ for

for문은 조건을 평가하는 대신 순서열을 순회하다가 순서열의 끝에 도달하면 반복을 종료함.

for 반복변수 in 순서열:  
    코드블록



```
for i in (1, 2, 3) :
 print(i)
```

변수 i에는 매 반복마다 튜플 (1, 2, 3)의 요소들이 차례대로 복사됩니다. 튜플의 길이는 3이므로 이 for문은 3번 반복을 수행합니다.

```
>for .py
1
2
3
```

## 6. 제어문

### ❖ continue

- 반복문이 실행하는 코드블록의 나머지 부분을 실행하지 않고 다음 반복으로 건너가도록 흐름을 조정

```
for i in range(10):
 if i % 2 == 1:
 continue

 print(i)
```

i가 홀수일 때 코드블록의 나머지 부분을 실행하지 않고 다음 반복으로 바로 건너갑니다.

continue가 실행되는 경우에는 이 코드는 실행되지 않습니다.

```
>continue.py
0
2
4
6
8
```

## 6. 제어문

### ❖ break

- 루프를 중단시키는 기능 수행

```
i = 0
while(True):
 i = i+1
 if i == 1000:
 print('i가 {0}이 되었습니다. 반복문을 중단합니다.'.format(i))
 break
 print(i)
```

```
>break.py
0
1
2
...
997
998
999
i가 1000이 되었습니다. 반복문을 중단합니다.
```

## 6. 제어문

❖제어문 안에 다른 제어문을 사용할 수 있음

❖구구단 전체 출력

```
print("★ 구구단을 출력★\n")
for x in range(1, 10):
 for y in range(2, 9):
 print(y,"X",x,"=",format(x*y,'2d'),end=" ")
 print("")
```

## 6. 제어문

❖ \*을 5개씩 5개 출력

```
for x in range(0, 5):
 for y in range(0, 5):
 print('{0}'.format('*'), end='')
 print('')
```

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*



## 6. 제어문

❖ \*을 1-5개씩 출력

```
for x in range(0, 5):
 for y in range(0, x+1):
 print('{0}'.format('*'), end='')
 print('')
```

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

# DataType

# 1. 자료형

|                     |                            |
|---------------------|----------------------------|
| bool                | True 또는 False를 저장          |
| int, float, complex | 정수, 실수, 복소수                |
| str                 | 유니코드 문자열 – 변경할 수 없는 데이터    |
| bytes               | 0-255 사이의 코드 모임            |
| list                | 순서가 있는 데이터의 집합 : 내용이 변경 가능 |
| tuple               | 순서가 있는 데이터의 집합 : 내용 변경 불가능 |
| set                 | 순서가 없는 데이터의 집합             |
| dict                | 키와 값을 쌍으로 만들어진 데이터의 집합     |

## 2. bool

- ❖ True 와 False를 저장할 수 있는 자료형
- ❖ 0이 아닌 값이나 데이터가 존재하는 시퀀스 자료형의 경우는 True로 간주하고 0이나 데이터가 존재하지 않는 시퀀스 자료형의 경우는 False
- ❖ bool 자료형끼리 산술연산 가능
- ❖ 식이나 연산의 결과를 bool 타입으로 변경하고자 하는 경우에는 bool(값이나 연산식)

```
print(bool([]))
```

```
print(bool(3))
```

```
print(bool(3+4))
```

**False**

**True**

**True**

# 3. 수치 데이터

## ❖ 정수형

- ✓ 음의 정수, 0, 양의 정수
- ✓ python에서는 메모리가 허용하는 한, 무한대의 정수를 다룰 수 있음.
- ✓ 하지만 CPU 레지스터로 표현할 수 있는 크기보다 큰 정수를 다루는 경우 연산 속도는 느려집니다.
- ✓ 일반적인 정수는 10진 정수로 간주
- ✓ 0o로 시작하는 정수는 8진수
- ✓ 0x, 0X로 시작하는 정수는 16진수
- ✓ 0b로 시작하는 정수는 2진수
- ✓ 문자열을 정수로 변환할 때는 int(문자열) 또는 int(문자열, 진법)를 이용
- ✓ 문자열 대신에 실수를 대입하면 소수를 버립니다.
- ✓ 실수로 된 문자열이나 복소수는 정수로 변환이 되지 않습니다.

# 3. 수치 데이터

## ❖ 예제

```
a = 10
print("a =", a)
a = 0o12
print("a =", a)
a = 0xA
print("a =", a)
a = 0b1010
print("a =", a)
a = int('10')
print("a =", a)
a = int('20', 5)
print("a =", a)
a = int(10.8)
print("a =", a)
```

## ❖ 결과

```
a = 10
a = 10
a = 10
a = 10
a = 10
a = 10
a = 10
```

# 3. 수치 데이터

## ❖ 실수형

- ✓ 파이썬에서는 실수를 지원하기 위해 부동 소수형을 제공
- ✓ 부동 소수형은 소수점을 움직여서 소수를 표현하는 자료형
- ✓ 부동 소수형은 8바이트를 이용해서 표현
- ✓ 한정된 범위의 수만 표현
- ✓ 디지털 방식으로 소수를 표현해야 하기 때문에 정밀도의 한계가 있음
- ✓ 소수점을 표현하면 실수(1.2)
- ✓ 지수(e)를 포함해도 실수(3e3, -0.2e-4)
- ✓ 실수의 가장 큰 값과 작은 값은 sys 모듈의 float\_info 또는 info.max, info.min으로 확인 가능
- ✓ 무한대의 수는 float('inf' | '-inf')
- ✓ is\_integer 메서드를 통해서 정수로 변환시의 오차 문제 확인 가능
- ✓ 실수를 정수로 변경할 때 사용할 수 있는 함수  
int, round, math.ceil, math.floor

# 3. 수치 데이터

## ❖ 예제

```
import sys
print("실수 정보:", sys.float_info)
a = 1.2
b = 3e3
c=-0.2e-4
print(a, b, c)
a = int(1.7)
print("a:", a)
a = round(1.7)
print("a:", a)
import math
a = math.ceil(1.7)
print("a:", a)
a = math.floor(1.2)
print("a:", a)
```

## ❖ 결과

실수 정보:

```
sys.float_info(max=1.79769313486231
57e+308, max_exp=1024,
max_10_exp=308,
min=2.2250738585072014e-308,
min_exp=-1021, min_10_exp=-307,
dig=15, mant_dig=53,
epsilon=2.220446049250313e-16,
radix=2, rounds=1)
```

1.2 3000.0 -2e-05

a: 1

a: 2

a: 2

a: 1



# 3. 수치 데이터

- ❖ 실수의 진법 변환 오류:소수 중에는 정확하게 2진수로 표현할 수 없는 수가 있으므로 실수 연산은 오차가 발생할 수 있습니다.

```
sum = 0
for i in range(0,1000):
 sum += 1
print("정수 1을 1000번 더한 결과:" , sum)
```

```
sum = 0.0
for i in range(0,1000):
 sum += 0.1
print("실수 0.1을 1000번 더한 결과:" , sum)
```

- ❖ 결과

정수 1을 1000번 더한 결과: 1000

실수 0.1을 1000번 더한 결과: 99.9999999999986

# 3. 수치 데이터

## ❖ 복소수

- ✓ 실수부와 허수부로 숫자를 표현
- ✓ 허수부는 j를 추가해서 표현
- ✓ 복소수는 출력될 때 ( ) 안에 출력됩니다.
- ✓ 복소수에서 real 속성을 호출하면 실수부만 리턴
- ✓ imag를 호출하면 허수부만 리턴
- ✓ 정수 2개를 가지고 복소수 생성 가능 - complex(실수부, 허수부)
- ✓ conjugate()를 호출해서 켄레 복소수 리턴

```
c = 3 + 4j
print(c)
a = 3;
b = 4;
print(complex(3,4))
d = c.conjugate();
print(d)
```

# 3. 수치 데이터

## ❖ fractions 모듈을 이용한 분수 표현

- ✓ fractions 모듈을 이용해서 분수 표현 가능
- ✓  $5/7$ 을 표현하고자 할 때 `Fraction('5/7')` 또는 `Fraction(5,7)`
- ✓ 그리고 만들어진 데이터의 `numerator` 와 `denominator`를 이용해서 분자와 분모를 따라 추출 가능

```
from fractions import Fraction
a = Fraction(5,7) + Fraction('2/5')
print(a)
```

# 3. 수치 데이터

## ❖ decimal 모듈을 이용한 숫자 표현

- ✓ Decimal(정수| 실수| 문자열)을 이용하면 정밀한 숫자를 표현 가능
- ✓ Decimal('0.1')을 이용하면 정확하게 숫자 표현 가능
- ✓ Decimal 객체는 Decimal 객체나 정수 데이터와 연산은 가능하지만 실수 객체와의 연산은 수행되지 않음

```
from decimal import Decimal
hap = Decimal(0.0)
delta = Decimal('0.1')
for i in range(0,1000):
 hap += delta
print("hap:" , hap)
```

# 3. 수치 데이터

- ❖ 수치 연산 함수
  - ✓ `abs(숫자)`: 절대 값 리턴
  - ✓ `divmod(x, y)`:  $x//y$ ,  $x\%y$ 의 값을 쌍으로 리턴
  - ✓ `pow(x, y)`:  $x$ 의  $y$ 승을 구합니다.
- ❖ `math` 모듈
  - ✓ `pi`, `e` 와 같은 상수
  - ✓ `sqrt(숫자)`, `sin(값)`과 같은 함수 내장

# 3. 수치 데이터

## ❖ and와 or 연산에서의 주의 사항

- ✓ True 또는 False 가 아닌 value의 and와 or 연산에서는 결과가 True 또는 False로 리턴되지 않습니다.
- ✓ and 연산의 경우 앞의 연산이 True이면 뒤의 연산의 결과가 리턴되며 False 인 경우 앞의 연산 결과가 리턴됩니다.
- ✓ or 연산의 경우는 and와 반대의 결과를 가져옵니다.

b = 1>2 and 3

print(b)

b = 0 and 3

print(b)

b = 1<2 and 3

print(b)

b = 0 or 3

print(b)

b = 1 or 3

print(b)

False

0

3

3

1

# 4. Sequential Type

## ❖ Sequential Type

- ✓ 객체를 순서대로 저장하는 자료형
- ✓ 문자열(str), list(값을 변경할 수 있음), tuple(값을 변경할 수 없음)
- ✓ 문자열은 " 또는 "" 로 묶인 문자들의 모임
- ✓ 리스트는 [ ] 안에 묶인 데이터의 모임
- ✓ 튜플은 ( )안에 묶인 데이터의 모임
- ✓ Sequential Type의 공통 연산
  - 인덱싱(특정 번째에 해당하는 데이터를 가져오는 것): 변수명[인덱스] – 인덱스 번째 데이터 리턴, 음수는 뒤에서부터 진행
  - 슬라이싱(특정 범위에 해당하는 데이터를 가져오는 것): [s:t:p] – s번째부터 t번째 바로 앞 까지 p 간격으로 가져오는 기능을 수행하게 되는데 t가 생략되면 s이후 전체, p를 생략하면 순서대로 하나씩
  - 연결하기: + 연산자
  - 반복하기: \* 연산자
  - 멤버 검사: in 연산자
  - 데이터 개수: len(SequentialType의 데이터)

# 4. Sequential Type

## ❖ 예제

```
str = "Korea"
print(str[0]) #0번째 글자
print(str[-2]) #뒤에서 2번째 글자
print(str[1:3]) #1번째부터 3번째 글자 앞까지
print(str[0:5:2]) #0부터 5번째 글자 앞까지 2개씩 건너뛰면서
print(str[:-1])#-1 번째까지 가져오기
print(str[::-1])#반대로 출력
str = "Hello" + "World"
print(str)
str = str * 4
print(str)
```

K  
e  
o  
r  
Kra  
Kore  
aeroK  
HelloWorld  
HelloWorldHelloWorldHelloWorld  
oWorld



# 5. str

## ❖ str

- ✓ 파이썬의 문자열 자료형
- ✓ 한 줄 문자열: ' ' 또는 " " 안에 기재하는데 단일 인용부호 안에 이중 인용부호를 이용해서 문자열을 대입할 수 있고 이중 인용부호 안에 단일 인용부호를 사용할 수 있습니다.
- ✓ 여러 줄 문자열:''' ''' 또는 """ """ 안에 기재

## ❖ 문자열은 데이터를 변경할 수 없음

## ❖ 문자열 내의 특정 문자의 값을 변경할 수 없습니다.

str = "hello"

str[0] = 'f'=> 이런 문장은 에러

## ❖ 문자열을 추가하거나 삭제할 때는 슬라이싱과 연결하기를 이용해야 합니다.

- Hardward and Shell 문자열에서 and 앞에 Kernel을 추가하기

```
str = "Hardware and Shell"
```

```
str = str[:8] + ' Kernel' + str[8:]
```

```
print(str)
```

- 위의 문자열에서 and 제거하기

```
str = str[:15] + str[20:]
```

```
print(str)
```

## 5. str

- ❖ ESCAPE 코드: 이스케이프 코드란 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 "문자 조합"으로 출력물을 보기 좋게 정렬하는 용도로 이용

| 코드   | 설명         |
|------|------------|
| \n   | 개행 (줄바꿈)   |
| \t   | 수평 탭       |
| \\   | 문자 "\"     |
| \'   | 단일 인용부호(') |
| \"   | 이중 인용부호(") |
| \r   | 캐리지 리턴     |
| \f   | 폼 피드       |
| \a   | 벨 소리       |
| \b   | 백 스페이스     |
| \000 | 널문자        |

이중에서 활용빈도가 높은 것은 \n, \t, \\, \', \"

## 5. str

### ❖ 문자열 포맷 코드 : 데이터 매핑

| 코드 | 설명                    |
|----|-----------------------|
| %s | 모든 데이터 가능             |
| %c | 문자 1개(character)      |
| %d | 정수 (Integer)          |
| %f | 부동소수 (floating-point) |
| %o | 8진수                   |
| %x | 16진수                  |
| %% | Literal % (문자 % 자체)   |

- ❖ 숫자 1개를 같이 사용하면 자릿수를 확보해서 생성하는데 양수를 사용하면 오른쪽 맞춤이고 음수를 대입하면 왼쪽 맞춤을 수행하며 앞에 0을 대입하면 남는 자리는 0으로 채워줍니다.

%10d : 10칸을 확보한 후 정수 데이터 매핑

- ❖ 실수는 숫자1.숫자2 형식으로 만들 수 있는데 이 경우 뒤의 숫자는 소수 자릿수입니다.

## 5. str

### ❖ 문자열의 서식 지정

- ✓ print() 함수를 사용하면 출력 문자열의 서식을 지정해서 출력 가능
- ✓ 단순히 문자열이나 데이터를 , 를 이용해서 출력할 수 있지만 양식을 만들어두고 빈칸에 필요한 내용을 채워서 문서를 자유롭게 채워나갈 수 있음
- ✓ 이전 방식은 문자열 안에 서식 문자를 지정하고 튜플을 이용해서 값을 채움
- ✓ 서식 문자는 %와 함께 표현

```
int_val = 23
```

```
float_val = 45.9876
```

```
print("int_val:%3d, float_val:%0.2f" % (int_val, float_val))
```

## 5. str

### ❖ 문자열의 서식 지정

- ✓ 문자열의 format 메서드를 이용
- ✓ 문자열 안에 {}를 사용하고 .format(데이터 나열)을 이용하면 나열된 데이터가 순차적으로 대입
- ✓ {인덱스}를 이용하면 format에서 인덱스 번째 데이터가 대입
- ✓ {인덱스:서식}을 이용하면 인덱스 번째의 데이터를 서식에 맞추어 출력

data = [1,3,4]

```
print('최대값:{0:5d}\t최소값:{1:5d}'.format(max(data), min(data)))
```

- ✓ 서식에 사용할 수 있는 보조 서식 문자
  - <: 왼쪽 맞춤
  - >: 오른쪽 맞춤
  - 공백: 양수 일 때 앞에 공백 출력
  - +: 양수 일 때 + 출력

# 5. str

## ❖ 문자열의 대소문자 변환 메서드

- ✓ upper(): 대문자로 변환
- ✓ lower(): 소문자로 변환
- ✓ swapcase(): 대소문자 스위치
- ✓ capitalize(): 첫 글자만 대문자로 변환
- ✓ title(): 각 단어의 첫 글자를 대문자로 변환

## ❖ 문자열의 검색 관련 메서드

- ✓ count(문자열): 문자열의 횟수
- ✓ find(문자열): 문자열이 있을 때 오프셋 반환
- ✓ find(문자열, 숫자): 숫자 위치부터 문자열을 검색해서 오프셋 반환
- ✓ 찾는 문자열이 없을 때는 -1을 반환
- ✓ rfind(문자열): 뒤에서부터 검색
- ✓ index(문자열): 문자열이 없을 경우 예외 발생
- ✓ rindex(문자열): 문자열을 뒤에서부터 검색하고 없으면 예외 발생
- ✓ startswith(문자열): 문자열로 시작하는지 여부를 검사해서 bool로 리턴
- ✓ endswith(문자열): 문자열로 끝나는지 여부를 검사
- ✓ startswith(문자열, 숫자): 숫자에 해당하는 위치가 문자열로 시작하는지 여부를 검사해서 bool로 리턴
- ✓ endswith(문자열, 숫자1, 숫자2): 숫자1부터 숫자2까지의 문자열이 문자열로 끝나는지 여부를 검사

# 5. str

## ❖ 문자열의 편집과 치환 메서드

- ✓ strip(): 좌우 공백 제거
- ✓rstrip(): 오른쪽 공백 제거
- ✓lstrip(): 왼쪽 공백 제거
- ✓ strip('문자열'): 좌우의 문자열을 제거
- ✓ replace(문자열1, 문자열2): 앞의 문자열을 뒤의 문자열로 치환

## ❖ 문자열의 분리와 결합 관련 메서드

- ✓ split(): 공백으로 문자열을 분리해서 리스트로 반환
- ✓ split('문자열'): 문자열로 분리해서 리스트로 반환
- ✓ split('문자열', 숫자): 문자열로 분리하는데 숫자만큼만 분리해서 리스트로 반환
- ✓ '문자열'.join(문자열 리스트): 문자열 리스트를 문자열을 각 요소 별로 문자열을 추가해서 결합
- ✓ splitlines(): 줄 단위로 분리

## ❖ 맞춤 관련 메서드

- ✓ center(숫자): 숫자 만큼의 자리를 확보하고 가운데 맞춤
- ✓ ljust(숫자): 왼쪽 맞춤
- ✓ rjust(숫자): 오른쪽 맞춤
- ✓ center(숫자, 문자열): 숫자 만큼의 자리를 확보하고 가운데 맞춤한 후 빈자리는 문자열로 채움

## 5. str

### ❖ 예제

```
msg = "c:\\\\data\\\\data.txt"
#test의 존재여부
print(msg.find('test'))
#위의 문자열에서 파일의 확장자만 추출하기
ar = msg.split(".")
print("확장자:", ar[len(ar)-1])
print(msg.replace('data', 'python'))
```

-1

확장자: txt

c:\python\python.txt



# 5. str

## ❖ 탭 문자 변환 메서드

- ✓ `expandtabs(숫자)`: 숫자를 생략하면 탭을 8자 공백으로 변경하고 숫자를 대입하면 숫자만큼의 공백으로 변경

## ❖ 문자열 확인 메서드

- ✓ `isdigit()`
- ✓ `isnumeric()`
- ✓ `isdecimal()`
- ✓ `isalpha()`
- ✓ `isalnum()`
- ✓ `islower()`
- ✓ `isupper()`
- ✓ `isspace()`
- ✓ `istitle()`
- ✓ `isidentifier()`
- ✓ `isprintable()`

## 5. str

### ❖ 문자열 매핑 메서드

- ✓ maketrans()와 translate()를 이용하면 문자열 매핑 가능
- ✓ maketrans()를 이용해서 치환할 문자열을 만들고 translate의 매개변수로 대입

```
instr = 'abcdef'
outstr = '123456'
trans = str.maketrans(instr, ostr)
str = 'hello world'
print(str.translate(trans))
```

# 5. str

## ❖ 문자열 인코딩

- ✓ 파이썬의 문자열은 기본적으로 utf-8 인코딩을 사용하는데 기본 아스키 코드는 1바이트를 사용하고 나머지는 2바이트를 사용하는 방식의 인코딩입니다.
- ✓ 문자열을 바이트 코드로 변환할 때는 encode()를 이용하고 특정 코드로 변환하고자 하는 경우에는 encode('인코딩방식') 메서드를 이용합니다.
- ✓ 바이트 코드는 문자열 함수를 거의 사용 가능하지만 문자열과 직접 연산은 불가능합니다.
- ✓ 바이트 코드를 문자열로 변환할 때는 decode('인코딩방식')을 이용해서 문자열로 변환할 수 있습니다.
- ✓ ord('문자'): 문자를 코드 값으로 변환
- ✓ chr(유니코드): 유니코드를 문자로 변환

```
b = '파이썬'.encode('utf-8')
print(b)
b = '파이썬'.encode('ms949')
print(b)
str = b.decode('ms949') #utf-8로 인코딩 하면 예외 발생
print(str)
```

## 5. str

### ❖ 유니코드에서 한글 자소 분리 및 자소를 글자로 변환

- ✓ 한글 유니코드는 초성 19개, 중성 21개, 종성 28개로 구성
- ✓ 한글 유니코드는 0xAC00에서 시작
- ✓ 한글 유니코드 구하기

$0xAC00 + ((\text{초성순서} * 21) + \text{중성순서}) * 28 + \text{종성순서}$

- ✓ 한글 한 글자의 초성 중성, 종성의 위치 찾아내기

`c = '한'`

`code = ord(c) - 0xAC00;`

`chosung = code // (21*28)`

`jungsung = (code - chosung*21*28)//28`

`jongsung = (code - chosung*21*28 - jungsung*28)`

`print('초성:', chosung, "번째 글자")`

`print('중성:', jungsung, "번째 글자")`

`print('종성:', jongsung, "번째 글자")`

## 5. str

### ❖ 영타로 입력한 내용 한글로 변경하기

```
cho_list_eng = ["r", "R", "s", "e", "E", "f", "a", "q", "Q", "t", "T", "d", "w", "W", "c", "z", "x", "v", "g"]
jung_list_eng = ["k", "o", "I", "O", "j", "p", "u", "P", "h", "hk", "hO", "hl", "y", "n", "nj", "np", "nl", "b",
 "m", "ml", "l"]
jong_list_eng = ["", "r", "R", "rt", "s", "sw", "sg", "e", "f", "fr", "fa", "fq", "ft", "fx", "fv", "fg", "a", "q",
 "qt", "t", "T", "d", "w", "C", "z", "x", "v", "g"]
```

```
str='wnd'
cho = cho_list_eng.index(str[0])
jung = jung_list_eng.index(str[1])
jong = jong_list_eng.index(str[2])
code = 0xac00 + ((cho*21) + jung)*28 + jong
print(chr(code))
```

## 6. 리스트

### ❖ list

- ✓ 순서를 갖는 객체의 집합
- ✓ 내부 데이터는 변경 가능한 자료형이고 시퀀스 자료형의 특성을 지원하며 데이터의 크기를 동적으로 변경할 수 있으며 내용을 치환하여 변경할 수 있습니다.
- ✓ 대괄호 [ ]로 표현
- ✓ 항목 교체
  - 리스트[인덱스] = 값
  - 리스트[시작위치:종료위치] = [데이터 나열]
  - 리스트[시작위치:종료위치] = 값
- ✓ 데이터 삭제
  - 리스트[시작위치:종료위치] = [ ]
  - del 리스트[인덱스]
- ✓ 데이터 중간에 삽입
  - 리스트[시작위치:종료위치] = [데이터]
- ✓ range()를 리스트로 변환
  - list(range(범위))

## 6. 리스트

### ❖ 예제

```
L = [1,2,3]
print ("자료형:", type(L))
print ("길이:" , len(L))
print ("두번째 데이터:", L[1])
print ("뒤에서 두번째:", L[-1])
print ("두번째 부터 4번째 앞 까지:",L[1:3])
print ("리스트 결합:", L + L)
print ("리스트 반복:",L * 3)
```

자료형: <class 'list'>

길이: 3

두번째 데이터: 2

뒤에서 두번째: 3

두번째 부터 4번째 앞 까지: [2, 3]

리스트 결합: [1, 2, 3, 1, 2, 3]

리스트 반복: [1, 2, 3, 1, 2, 3, 1, 2, 3]

## 6. 리스트

### ❖ 예제

```
L = list(range(4)) #range를 이용한 list 생성
print(L)
del L[::2] #짝수번째 데이터 삭제
print(L)
L[1:1] = [2] #1번째 자리에 2를 추가
print(L)
L[0:0] = [0] #0번째 자리에 0을 추가
print(L)
#리스트 내의 모든 객체 순회
for i in L:
 print(i)
```

```
[0, 1, 2, 3]
[1, 3]
[1, 2, 3]
[0, 1, 2, 3]
0
1
2
3
```



## 6. 리스트

### ❖ 중첩 list

- ✓ 리스트 안에 다른 리스트가 포함된 경우
- ✓ 리스트는 다른 객체를 직접 저장하지 않고 객체들의 참조만을 저장합니다.
- ✓ 따라서 다른 리스트를 참조하는 경우 참조된 리스트의 내용이 변경되면 포함하고 있는 리스트의 내용도 변경됩니다.

```
innerlist = list(range(4))
outerlist = ['begin', innerlist, 'end']
print(outerlist)
innerlist[0] = 200
print(outerlist)
```

```
['begin', [0, 1, 2, 3], 'end']
['begin', [200, 1, 2, 3], 'end']
```

## 6. 리스트

### ❖ list 메서드

- ✓ `append(값)`: 마지막에 데이터 추가
- ✓ `insert(인덱스, 데이터)`: 인덱스 위치에 데이터를 삽입
- ✓ `index(데이터)`: 데이터의 위치를 검색
- ✓ `count()`: 요소의 개수 리턴
- ✓ `sort()`: 리스트를 정렬
- ✓ `reverse()`: 리스트의 순서를 변경
- ✓ `remove(데이터)`: 데이터의 첫번째 것을 삭제
- ✓ `pop(인덱스)`: 인덱스를 생략하면 가장 마지막 데이터를 삭제하고 리턴하고 인덱스를 대입하면 인덱스 번째를 삭제하고 리턴
- ✓ `extend(리스트)`: 리스트를 추가

## 6. 리스트

### ❖ 정렬

- ✓ sort()를 이용하면 데이터를 내부적으로 정렬
- ✓ 값을 리턴하지 않음
- ✓ 매개변수로 reverse=True를 대입하면 내림차순 가능

```
data = [30,50,10,40,20]
data.sort()
print("오름차순:", data)
data.sort(reverse=True)
print("내림차순:", data)
```

## 6. 리스트

### ❖ 정렬

- ✓ 문자열을 정렬할 때 대소문자 구분없이 정렬 가능
- ✓ 매개변수로 key라는 속성에 비교할 함수이름을 전달

```
data = ['Morning', 'Afternoon', 'evening', 'Night']
data.sort()
print("대소문자 구분해서 정렬:", data)
data.sort(reverse=True)
print("내림차순 정렬:", data)
data.sort(key=str.lower, reverse=True)
print("내림차순 정렬:", data)
```

## 6. 리스트

### ❖ 정렬

- ✓ 사용자 정의 함수를 이용한 정렬

```
def strlen(st):
 return len(st)
```

```
data = ['Morning', 'Evening', 'Afternoon', 'Night']
data.sort(key=strlen)
print(data)
```

## 6. 리스트

### ❖ 정렬

- ✓ 리스트를 정렬하지 않고 정렬한 결과를 리스트로 리턴하는 함수는 `sorted()`
- ✓ `key`와 `reverse` 키워드를 동일하게 지원
- ✓ 이 메서드는 튜플이나 디셔너리에서도 사용 가능
- ✓ `reverse()`를 이용하면 데이터를 역순으로 배치

```
data = [30,50,10,40,20]
print("오름차순:", sorted(data))
print("내림차순:",sorted(data, reverse=True))
```

```
data = [30,50,10,40,20]
data = sorted(data)
print("오름차순:", data)
data.reverse()
print("내림차순:",data)
```

# 7. 튜플

## ❖ tuple

- ✓ 임의 객체들이 순서를 가지는 모임으로 리스트와 유사
- ✓ 차이점은 내부 데이터의 변경이 불가능하다는 것
- ✓ 튜플은 시퀀스 자료형이므로 인덱싱과 슬라이싱, 연결하기, 반복하기, 길이 정보 등의 연산 가능
- ✓ 튜플은 ()안에 표현
- ✓ ()를 사용하지 않고 ,로 데이터를 구분하면 튜플로 처리
- ✓ ()를 하는 경우 데이터가 1개 일 때는 ,를 마지막에 추가
- ✓ list()와 tuple()를 이용해서 서로 간에 변환 가능
- ✓ 포함된 데이터 개수를 리턴해주는 count 소유
- ✓ 포함된 데이터 위치를 리턴해주는 index 소유

```
t = (1,2,3,2,2,3)
print("2의 개수:", t.count(2))
print("2의 위치:", t.index(2))
print("2의 위치:", t.index(2,3))
```

# 7. 튜플

## ❖ tuple

- ✓ 여러 개의 데이터를 한꺼번에 대입하는 것이 가능
- ✓ 변수, 변수 = 데이터, 데이터
- ✓ 위의 방법을 이용하면 swap이 쉽게 가능

```
x,y=1,2
x,y=y,x
print(x, y)
```



# 8. set

## ❖ set

- ✓ 데이터를 순서와 상관없이 중복 없이 모아놓은 자료형으로 set과 frozenset 두 가지 자료형을 제공
- ✓ 빈 객체 생성은 set()
- ✓ 처음부터 데이터를 가지고 있는 경우에는 {데이터 나열}
- ✓ 데이터가 없는 경우에는 {} 대신 set()으로 생성하는 이유는 디셔너리와 혼동의 문제
- ✓ 튜플, 문자열, 리스트 및 디셔너리로부터 생성이 가능
- ✓ set의 원소로 변경이 가능한 데이터 타입은 불가능합니다.

```
hashset = set()
print(hashset)
hashset = {1,2,3}
print(hashset)
hashset = set((1,2,3,2))
print(hashset)
hashset = set('hello world')
print(hashset)
hashset = set([1,3,2])
print(hashset)
```

## 8. set

### ❖ set의 연산

- ✓ 데이터의 추가는 `add(데이터)`, `update[데이터 나열]`
- ✓ 데이터의 복사는 `copy()`
- ✓ 데이터의 전체 제거는 `clear()`
- ✓ 데이터의 한 개 제거는 `discard(데이터)` – 없으면 그냥 통과
- ✓ 데이터의 한 개 제거는 `remove(데이터)` – 없으면 예외
- ✓ `Pop()`: 1개 제거

### ❖ set의 집합 연산

- ✓ `union(다른 set)`, `intersection(다른 set)`, `difference(다른 set)`, `symmetric_difference(다른 set)`의 메서드가 제공되는데 결과를 리턴합니다.
- ✓ `update()`, `intersection_update()`, `difference_update()`, `symmetric_difference_update()`는 호출하는 객체의 데이터를 변경

## 8. set

### ❖ 포함 관계 연산자

- ✓ 데이터 in set : 앞의 요소가 포함되어 있는지 리턴
- ✓ 데이터 not in set: 앞의 요소가 포함되어 있지 않은지 리턴
- ✓ issuperset(다른 set): 다른 set을 포함하고 있는지 여부 리턴
- ✓ issubset(다른 set): 다른 set의 서브셋인지 여부 리턴
- ✓ isdisjoint(다른 set): 교집합이 공집합인지 여부 리턴

### ❖ 예제

```
a = {1,2,3,4,5}
b = {4,5,6,7,8}
print(a.union(b))
print(a.intersection(b))
print(a.difference(b))
print(a.symmetric_difference(b))
```

# 9. 사전

## ❖ dict

- ✓ Key와 Value를 쌍으로 저장하는 자료형
- ✓ {키:데이터, 키:값...}로 생성가능
- ✓ Key의 순서는 없으며 Key의 값은 중복될 수 없습니다.
- ✓ Key의 자료형에 제한은 없지만 거의 str
- ✓ Key 값에 해당하는 데이터를 가져올 때는 디셔너리[키]
- ✓ 디셔너리[키] = 데이터 를 이용하면 키의 데이터가 없으면 삽입이 되고 있으면 수정
- ✓ 없는 키의 값을 호출하면 에러
- ✓ len(디셔너리)는 키의 개수
- ✓ del 디셔너리[키]는 키의 데이터 삭제

```
member = {'baseball':9, 'soccer':11, "volleyball":6}
print(member)
print(member['baseball'])
print(member['basketball'])
```

## 9. 사전

### ❖ dict

- ✓ 생성하는 다른 방법은 dict(키=데이터, 키=데이터)로 가능
- ✓ 키의 리스트나 값의 리스트가 존재하는 경우 dict(zip(키의 리스트, 값의 리스트))로 생성 가능
- ✓ 사전을 for에 사용하면 키의 모든 리스트가 순서대로 대입됩니다.

```
keys = ['one', 'two', 'three']
values = (1,2,3)
dic = dict(zip(keys, values))
for key in dic:
 print(key,":", dic[key])
```

# 9. 사전

## ❖ dict

- ✓ keys(): 키에 대한 모든 데이터를 리턴
- ✓ values(): 값에 대한 모든 데이터를 리턴
- ✓ items(): 모든 데이터 리턴
- ✓ list()의 매개변수로 위 3개의 메서드 리턴값을 넣으면 list로 변환
- ✓ clear(): 모두 삭제
- ✓ copy(): 복사
- ✓ get(key [,x]): 값이 존재하면 값을 리턴하고 없으면 x 리턴
- ✓ setdefault(key [,x]): get 과 동일하지만 값이 존재하지 않으면 x로 설정
- ✓ update(디셔너리): 디셔너리의 모든 항목을 설정
- ✓ popitem(): 마지막 하나의 튜플을 반환하고 제거
- ✓ pop(key): key에 해당하는 데이터를 반환하고 제거

# 9. 사전

## ❖ Ordereddict

- ✓ 키의 순서를 유지하는 디셔너리를 만들고자 할 때는 collections 모듈의 Ordereddict를 사용
- ✓ dict와 동일하게 동작하지만 데이터가 추가된 순서를 기억해서 데이터를 순서대로 처리할 수 있도록 해주는 자료형

```
from collections import OrderedDict
keys = ['one', 'two', 'three']
values = (1,2,3)
dic = dict(zip(keys, values))
for key in dic:
 print(key,":", dic[key])
print("=====")
dic = OrderedDict(zip(keys, values))
for key in dic:
 print(key,":", dic[key])
```



# Copy



# 1.복사

## ❖ 복사

- ✓ Python에서의 복사는 2가지가 있습니다.
- ✓ 하나는 객체의 참조를 복사하는 것이고 다른 하나는 객체 자체를 복사하는 것입니다.
- ✓ 파이썬에서는 상수를 대입하는 경우 상수의 참조를 복사

a = 1

b = 1

print(id(a))

Print(id(b))

**1674327824**

**1674327824**

위의 문장은 1이라는 객체를 저장하고 그 주소를 a라는 변수에 대입하고 동일한 객체의 주소를 b에 저장했으므로 동일한 id가 출력

# 1.복사

✓ 변수의 값을 대입해도 결과는 동일합니다.

```
a = [1,2,3]
```

```
b = a
```

```
a[0]=100
```

```
print("a의 id:", id(a));
```

```
print("b의 id:", id(b));
```

```
print("a의 첫번째 요소:",a[0])
```

```
print("b의 첫번째 요소:",b[0])
```

**a의 id: 44029232**

**b의 id: 44029232**

**a의 첫번째 요소: 100**

**b의 첫번째 요소: 100**

## 2. 얇은 복사

### ❖ 얇은 복사

- ✓ python은 copy 모듈의 copy와 deepcopy 메서드를 이용해서 얇은 복사와 깊은 복사를 지원합니다.
- ✓ copy는 얇은 복사를 지원하는 메서드
- ✓ 얇은 복사를 하게되면 객체를 복제해서 다른 곳에 복사 한 후 대입하게 됩니다.
- ✓ 얇은 복사는 재귀적으로 객체를 복제해서 복사 작업을 수행해주지는 않습니다.
- ✓ 단 한번의 복사만 수행하게 됩니다.
- ✓ list의 슬라이싱과 dict 의 copy 메서드도 얇은 복사

## 2. 얇은 복사

```
a = [1,2,3]
b = a[0:3]
a[0]=300
print("a[0]:", a[0])
print("b[0]:", b[0])
```

**a[0]: 300**

**b[0]: 1**

=> list의 슬라이싱은 얇은 복사를 수행하므로 결과는 그대로 1

## 2. 얇은 복사

```
import copy
a = [1,2,3]
x = [a, 100]
y = copy.copy(x)
```

```
#copy를 했으므로 2개의 아이디는 다름
print("x의 아이디:", id(x));
print("y의 아이디:", id(y));
```

```
#두 번째 데이터는 일반적인 데이터이므로 복제가 이루어짐
x[1] = 200
print("x[1]:",x[1])
print("y[1]:",y[1])
```

```
#첫 번째 데이터는 다시 데이터의 모임이므로 주소의 복제를 수행
x[0][0] = 300
print("x[0][0]:",x[0][0])
print("y[0][0]:",y[0][0])
```

x의 아이디: 2491496  
y의 아이디: 46087544  
x[1]: 200  
y[1]: 100  
x[0][0]: 300  
y[0][0]: 300

# 3. 깊은 복사

## ❖ 깊은복사

- ✓ deepcopy는 깊은 복사를 지원하는 메서드
- ✓ 깊은 복사를 하게되면 객체를 복제해서 다른 곳에 복사 한 후 대입하게 됩니다.
- ✓ 깊은 복사는 재귀적으로 객체를 복제해서 복사 작업을 수행합니다.



### 3. 깊은 복사

```
import copy
a = [1,2,3]
x = [a, 100]
y = copy.deepcopy(x)
```

x의 아이디: 6882408  
y의 아이디: 45429304  
x[0][0]: 300  
y[0][0]: 1

```
#copy를 했으므로 2개의 아이디는 다름
print("x의 아이디:", id(x));
print("y의 아이디:", id(y));
```

```
#첫번째 데이터는 다시 데이터의 모임이므로 재귀적으로 복제를 수행
x[0][0] = 300
print("x[0][0]:",x[0][0])
print("y[0][0]:",y[0][0])
```

# 형변환



# 1. 수치 데이터 변환

## ❖ 정수로의 변환

- ✓ 다른 자료형의 데이터를 정수로 변환할 때는 int(데이터)
- ✓ 변환할 수 없다면 ValueError 에러 발생
- ✓ 실수의 경우는 int 내장함수를 이용해도 되지만 round(), math.floor(), math.ceil() 이용해서 변환하기도 함
- ✓ 임의의 진수를 10진수로 변환하는 경우에도 int() 함수 이용
- ✓ 16진수 변환은 hex(), 8진수 변환은 oct() 함수를 이용해도 되고 format 함수를 이용가능

```
dec = "{0:x}".format(100)
```

```
print(dec)
```

## ❖ 실수로의 변환은 float() 이용

- ❖ 복소수로의 변환은 complex()를 이용하는데 매개변수를 1개 주면 실수부로 처리하고 2개를 주면 첫번째 값을 실수부로 처리하고 두번째 값을 허수부로 처리

## 2.시퀀스로의 변환

❖ 시퀀스 자료형 사이의 변환은 자료형 이름으로 된 함수가 있어서 자료형 이름으로 된 함수를 이용하면 됩니다.

❖ list(), tuple()

❖ 2개의 list가 있다면 dict로 변환가능

dict(zip(list, list))

❖ dict에서 list로 변환을 수행할 때는 keys(), values(), items() 메서드 이용

```
dict = {1:'one', 2:'two', 3:'three'}
```

```
print(list(dict.keys()))
```

```
print(list(dict.values()))
```

```
print(list(dict.items()))
```

```
[1, 2, 3]
```

```
['one', 'two', 'three']
```

```
[(1, 'one'), (2, 'two'), (3, 'three')]
```



# 3. 문자열 변환

- ❖ 객체를 문자열로 변환에는 `str()`, `repr()` 함수를 이용
- ❖ 문자열로 된 객체를 원래의 타입으로 되돌릴때는 `eval()`
- ❖ 유니코드를 문자로 변환할 때는 `chr()`
- ❖ 문자를 유니코드로 변환할 때는 `ord()`
- ❖ 문자열에서 바이트로의 변환은 `encode()`
- ❖ 바이트 열에서 문자열로의 변환은 `decode()`



# 실습 예제 확인



# Function

# 1.함수의 정의

## ❖ 함수(Function)

- 자주 사용하는 문장들을 하나의 이름으로 묶어서 호출해서 사용하기 위한 개념입니다.
- 반복적으로 호출해서 실행이 가능
- 코드의 일정 부분을 별도의 논리적인 개념으로 분리하기 위해서 사용하기도 합니다.
- 함수는 위처럼 코드를 재사용할 수 있게 해주고 프로그램을 논리적으로 구성할 수 있도록 해줍니다.
- 파이썬의 함수는 전부 일급 함수(First Class Function)라서 함수를 다른 함수의 인수로 대입할 수 있고 함수의 반환 값을 다른 함수에 전달할 수 있고 변수에 대입하는 것도 가능
- 함수 안에 함수를 만드는 것도 가능



# 1.함수의 정의

## ❖ 정의

- 정의(Definition)란, 어떤 이름을 가진 코드가 구체적으로 어떻게 동작하는지를 “구체적으로 기술”하는 것

## ❖ 파이썬에서는 함수나 메서드를 정의할 때 definition(정의)를 줄인 키워드인 **def**를 사용

## ❖ def 다음에 함수이름과 인수들을 나열하고 :

## ❖ 함수의 몸체는 그 다음 줄에 들여쓰기를 하고 시작해야 하며 파이썬은 어떤 형식의 데이터도 인수로 전달할 수 있기 때문에 인수의 자료형은 기재하지 않습니다.

## ❖ return 은 결과를 돌려주고자 할 때 결과를 함수를 호출한 곳으로 돌려줄 때 사용

## ❖ def 키워드를 이용한 함수 정의

def 함수이름(인수들):

문장을 나열

return <값>



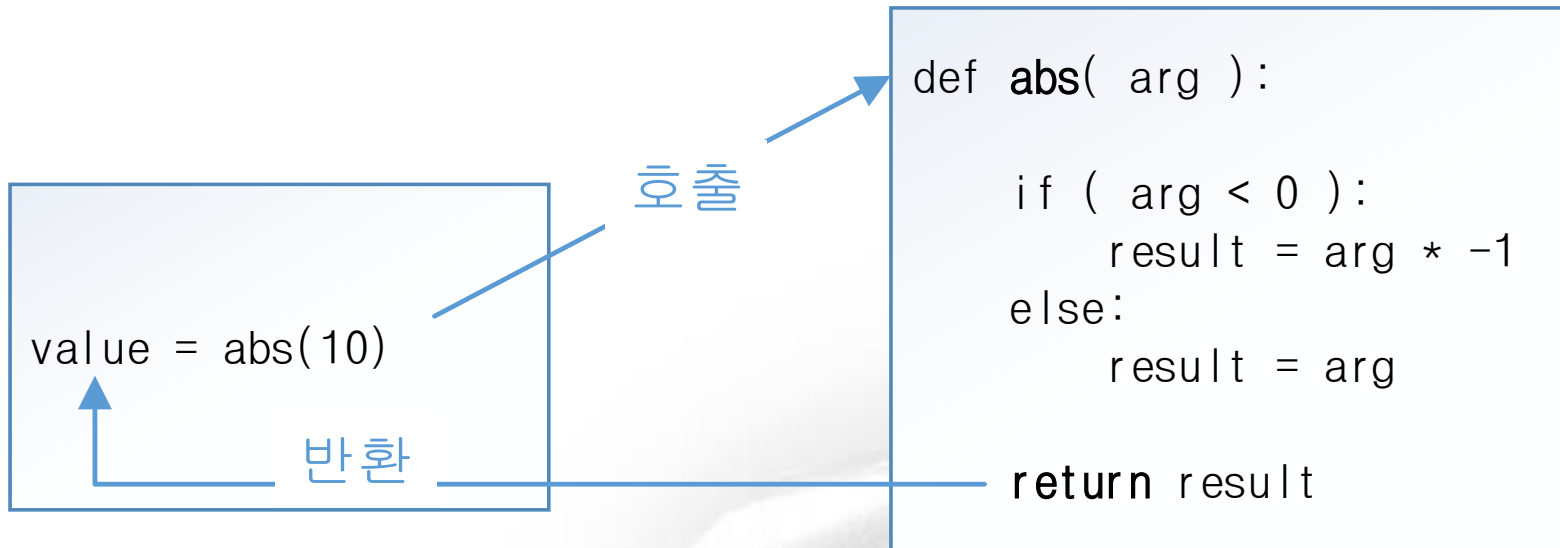
# 1.함수의 정의

## ❖ 호출(Call)

- 모든 함수는 이름을 갖고 있으며, 이 이름을 불러주면 파이썬은 그 이름 아래 정의되어 있는 코드를 실행.

## ❖ 반환(Return)

- 함수가 자신의 코드를 실행하고 나면 결과가 나오는데, 그 결과를 자신의 이름을 부른 코드에게 돌려줌.





# 1.함수의 정의

- ❖ return 문장 다음에 값이 없으면 파이썬은 None 객체를 반환합니다.
- ❖ return 문장이 없어도 None 객체를 반환한 것으로 간주합니다.
- ❖ return을 할 때 여러 개의 값을 return 하면 튜플을 만들어서 리턴하게 됩니다.

```
def nothing():
```

```
 return
```

```
print(nothing())
```

```
def swap(a, b):
```

```
 return b, a
```

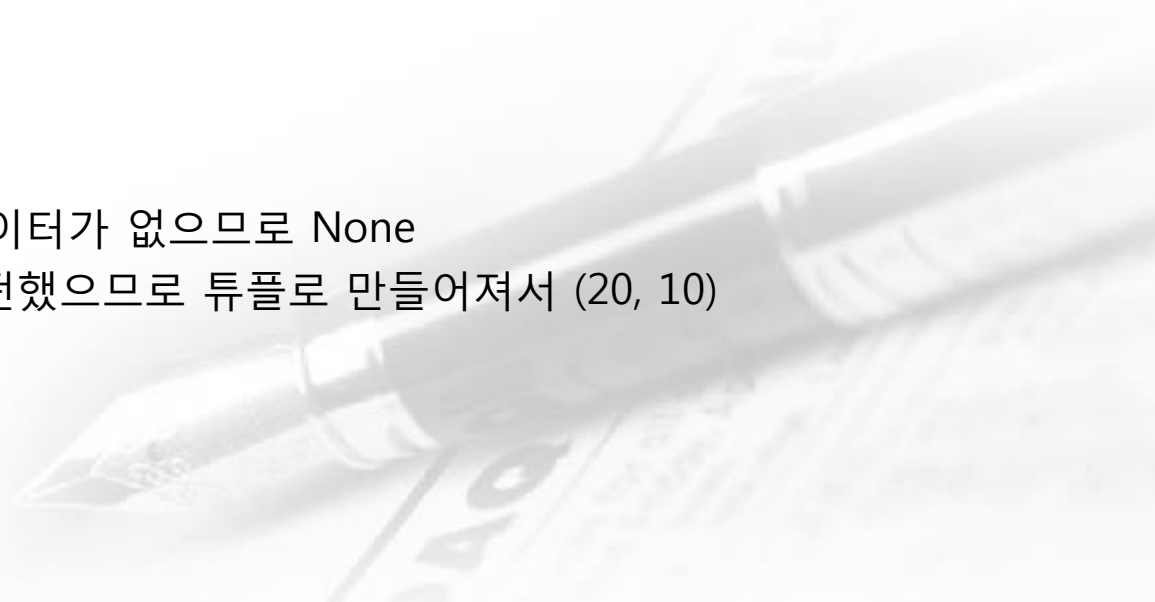
```
a = 10
```

```
b = 20
```

```
print(swap(a,b))
```

=>첫번째 출력문은 return 뒤에 데이터가 없으므로 None

=>두번째 출력문은 2개의 값을 리턴했으므로 튜플로 만들어져서 (20, 10)



## 2.매개변수 전달

- ❖ 파이썬에서는 매개변수(argument)에 데이터를 넘겨주면 객체의 참조를 넘겨줍니다.
- ❖ 변경이 불가능한 데이터를 넘겨주면 복제를 해서 넘겨주고 변경이 가능한 객체를 넘겨주면 참조를 넘겨줍니다.
- ❖ 매개변수가 있는 함수를 호출할 때 매개변수를 대입하지 않으면 에러 발생

```
def f(t):
```

```
 t = 10
```

```
a = 20
```

```
f(a)
```

```
print(a)
```

=>a는 정수 상수를 대입했기 때문에 변경이 불가능하므로 20



## 2.매개변수 전달

```
def f(t):
 t[0] = 100
li = [1,2,3]
f(li)
print(li[0])
```

=>list가 저장하고 있는 주소를 넘겨주었기 때문에 함수 내에서 리스트의 내용을 변경하면 원본의 데이터도 변경됩니다.

=>따라서 li[0]의 값은 100



## 2.매개변수 전달

- ❖ 매개변수의 기본값: 함수의 매개변수가 있는 경우 매개변수를 넘겨주지 않으면 에러가 발생합니다.
- ❖ 파이썬에서는 매개변수에 기본값을 할당할 수 있는데 **매개변수이름=값**의 형식을 취하게 되면 매개변수에 값을 대입하지 않을 때 기본적으로 값이 대입되면 매개변수에 값이 있으면 그 값을 취하게 됩니다.
- ❖ 기본값이 있는 매개변수 뒤에는 기본값이 없는 매개변수가 올 수 없습니다.

```
def f(n, step=1):
 return n+step;
a = 1
a = f(a,10)
print(a)
b = 1
b = f(b)
print(b)
```

=> 첫번째의 경우는 매개변수가 있으므로 step에 10이 대입되서 11

=> 두번째의 경우는 매개변수가 생략되어 있으므로 step에 1이 대입되서 2

## 2.매개변수 전달

- ❖ 매개변수를 대입할 때는 기본적으로 순서대로 대입되지만 파이썬에서는 키워드를 이용해서 순서와 상관없이 대입이 가능합니다.
- ❖ 매개변수를 대입할 때 매개변수의 이름과 함께 대입하면 됩니다.

```
def f(height, weight):
```

```
 print("키는", format(height, 'd'))
```

```
 print("몸무게는 ", format(weight, 'd'))
```

```
f(180, 67)
```

```
f(weight=80, height=190)
```



## 2.매개변수 전달

### ❖ 가변 매개변수(Arbitrary Argument List)

- 입력 개수가 달라질 수 있는 매개변수
- \*를 이용하여 정의된 가변 매개변수는 튜플이고 \*\*이용하여 정의된 매개변수는 dic

```
def 함수이름(*매개변수):
 코드블록
```

매개변수 앞에 \*를 붙이면 해당매개변수는 가변으로 지정됩니다.

### ❖ 실습 (가변 매개변수)

```
def merge_string(*text_list):
 result = ''
 for s in text_list:
 result = result + " " + s
 return result
print(merge_string('안녕하세요', '반갑습니다'))
print(merge_string('아버지가', '방에', '들어가신다.'))
```

## 2.매개변수 전달

### ❖ 실습 (가변 매개변수)

```
def f(width, height, **kw):
 print(width, height)
 print(kw)
f(width=10, height=5, depth=20, dimension=40)
```



## 2. 매개변수 전달

❖ 일반 매개변수와 함께 사용하는 가변매개변수

```
def print_args(argc, *argv):
 for i in range(argc):
 print(argv[i])
```

```
>>> print_args(3, "argv1", "argv2", "argv3")
argv1
argv2
argv3
```

가변 매개변수 앞에 정의된 일반 매개변수는 키워드 매개변수로 호출할 수 없습니다.

```
>>> print_args(argc=3, "argv1", "argv2", "argv3")
SyntaxError: non-keyword arg after keyword arg
```



## 2. 매개변수 전달

❖ 가변 매개변수와 함께 사용하는 일반 매개변수

```
def print_args(*argv, argc):
 for i in range(argc):
 print(argv[i])
```

```
>>> print_args("argv1", "argv2", "argv3", argc=3)
argv1
argv2
argv3
```

가변 매개변수 뒤에 정의된 일반 매개변수는 반드시 키워드 매개변수로 호출해야 합니다.

```
>>> print_args("argv1", "argv2", "argv3", 3)
Traceback (most recent call last):
 File "<pyshell#15>", line 1, in <module>
 print_args("argv1", "argv2", "argv3", 3)
TypeError: print_args() missing 1 required keyword-only
argument: 'argc'
```

# 3.유효범위

❖ “함수 밖에서 변수 a를 정의하여 0을 대입하고, 함수 안에서 변수 a를 또 정의하여 1을 대입했다. 이 함수를 실행(호출)하고 나면 함수 밖에서 정의된 변수 a의 값은 얼마일까?”

- 답 : 0
- 함수 밖에 있는 a와 안에 있는 a는 이름은 같지만 사실은 완전히 별개의 변수

❖ 실습 1

```
def scope_test():
 a = 1
 print('a:{0}'.format(a))
```

함수를 정의하는 시점에서는 a가 메모리에 생성되지 않습니다. 함수를 호출하면 그제서야 함수의 코드가 실행되면서 a가 메모리에 생성됩니다.

함수 밖에서 a를 정의하고 0으로 초기화 합니다.

```
>>> a = 0
>>> scope_test()
a:1
>>> print('a:{0}'.format(a))
a:0
```

scope\_test()가 호출되면 함수 내부에서 a를 정의하고 1로 초기화합니다.

하지만 함수 밖에서 정의한 a를 출력해보면 여전히 0을 갖고 있음을 확인할 수 있습니다.

# 3.유효범위

- ❖ 변수는 자신이 생성된 범위(코드블록) 안에서만 유효
- ❖ 함수 안에서 만든 변수는 함수 안에서만 살아있다가 함수 코드의 실행이 종료되면 그 생명이 다함 → 이것을 **지역변수(Local Variable)**라고 함
- ❖ 이와는 반대로 함수 외부에서 만든 변수는 프로그램이 살아있는 동안에는 함께 살아있다가 프로그램이 종료될 때 같이 소멸됨.
- ❖ 파이썬은 함수 안에서 사용되는 모든 변수를 지역변수로 간주
- ❖ 외부 변수를 사용하기 위해서는 **global** 키워드를 이용
- ❖ 함수 내에서 자신을 포함하는 함수의 변수를 사용할 때는 **nonlocal** 키워드를 사용
- ❖ global 이나 nonlocal 변수를 사용할 때는 선언을 먼저 하고 사용을 해야 합니다.

# 3.유효범위

- 변수의 유효범위

```
def outer():
 a = 1
 def inner():
 nonlocal a
 print("함수의 외부 함수에 있는 a:{0}".format(a))
 a = 10
 inner()
 print("내부함수에서 변경한 경우의 a:{0}".format(a))

a = 0
outer()
print("a:{0}".format(a))
```

nonlocal 키워드는 지정한 변수의 유효범위가 함수를 포함하고 있는 함수임을 알리고 지역변수의 생성을 막습니다. 따라서 결과는 1



# 3.유효범위

- 변수의 유효범위

```
def outer():
 a = 1
 def inner():
 global a
 print("함수의 외부에 있는 a:{0}".format(a))
 a = 10
 inner()

a = 0
outer()
print("a:{0}".format(a))
```

global 키워드는 지정한 변수의 유효범위가 함수를 포함하고 있는 외부임을 알리고 지역변수의 생성을 막습니다. 따라서 결과는 0



## 4.재귀 함수(recursion)

- ❖ 재귀함수(Recursive Function)는 자기 스스로를 호출하는 함수
- ❖ 함수가 자기 자신을 부르는 것을 재귀호출(Recursive Call)이라 함.
- ❖ 재귀 함수의 예

```
def some_func(count):
 if count > 0:
 some_func(count-1)
 print(count)
```



## 4.재귀함수(recursion)

◆ 팩토리얼

```
def factorial(n):
 if n == 0:
 return 1
 elif n > 0:
 return factorial(n-1)*n
```

```
>>> factorial(5)
```

120

```
>>> factorial(10)
```

3628800

```
>>> factorial(100)
```

9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146  
39761565182862536979208272237582511852109168640000000000000000000000

# 4.재귀 함수(recursion)

- ❖ 재귀 호출의 단계가 깊어질 수록 메모리를 추가적으로 사용하기 때문에 재귀 함수가 종료될 조건을 분명하게 만들어야 함.

```
>>> def no_idea():
 print("나는 아무 생각이 없다.")
 print("왜냐하면")
 no_idea()
```

```
>>> no_idea()
나는 아무 생각이 없다.
왜냐하면
나는 아무 생각이 없다.
왜냐하면...
```

```
Traceback (most recent call last):
 File "<pyshell#10>", line 1, in <module>
 no_idea()
 File "<pyshell#8>", line 4, in no_idea
 no_idea()
```

...

```
File "<pyshell#8>", line 2, in no_idea
 print("나는 아무 생각이 없다.")
File "C:\Python34\lib\idlelib\PyShell.py", line 1342, in write
 return self.shell.write(s, self.tags)
```

```
RuntimeError: maximum recursion depth exceeded while calling a Python object
```

종료할 조건도 지정해주지 않은 채 무조건 재귀호출을 수행하면 스택 오버플로우가 발생합니다.

스택 오버 플로우가 발생하면 파이썬에서 지정해놓은 최대 재귀 단계를 초과했다는 에러가 출력됩니다.



## 5. 함수를 변수에 담아 사용

```
def print_something(a):
 print(a)
```

() 없이 함수의 이름만을 변수에 저장합니다.

```
p = print_something
>>> p(123)
123
>>> p('abc')
abc
```

변수의 이름 뒤에 ()를 붙여 함수처럼 호출하면 됩니다.

```
>>> def plus(a, b):
 return a+b
```

```
>>> def minus(a, b):
 return a-b
```

plus() 함수와 minus() 함수를 리스트의 요소로 집어넣습니다.

```
>>> flist = [plus, minus]
>>> flist[0](1, 2)
3
>>> flist[1](1, 2)
-1
```

flist[0]은 plus() 함수를 담고 있습니다. 따라서 이 요소 뒤에 괄호를 열고 매개변수를 입력하여 호출하면 plus() 함수가 호출됩니다.

flist[1]는 minus()를 담고 있으므로 이 코드는 minus(1, 2)와 같습니다.

# 5. 함수를 변수에 담아 사용

## ❖ 함수를 변수에 담을 수 있는 이유?

- 파이썬이 함수를 일급 객체(First **Class** Object)로 다루고 있기 때문
- 일급 객체란 프로그래밍 언어 설계에서 매개변수로 넘길 수 있고 함수가 반환할 수도 있으며 변수에 할당이 가능한 개체를 가리키는 용어
- 파이썬에서는 함수를 "매개변수"로도 사용할 수 있고 함수의 결과로 "반환"하는 것도 가능

```
>>> def hello_korean():
 print('안녕하세요.')
```

```
>>> def hello_english():
 print('Hello.')
```

```
>>> def greet(hello):
 hello()
```

```
>>> greet(hello_korean)
안녕하세요.
```

```
>>> greet(hello_english)
Hello.
```

## 5. 함수를 변수에 담아 사용

### ❖ 함수를 결과로써 반환하기

```
>>> def hello_korean():
 print('안녕하세요.')
```

```
>>> def hello_english():
 print('Hello.')
```

```
>>> def get_greeting(where):
 if where == 'K':
 return hello_korean
 else:
 return hello_english
```

```
>>> hello = get_greeting('K')
>>> hello()
안녕하세요.
>>> hello = get_greeting('E')
>>> hello()
Hello.
```

매개변수 where가 'K'인 경우 hello\_korean() 함수를 반환합니다.

그 외의 경우 hello\_english() 함수를 반환합니다.

get\_greeting() 함수가 반환하는 결과를 변수 hello 에 담아 “호출”합니다.

## 6. 중첩 함수

- ❖ 중첩 함수(Nested Function) : 함수 안에 정의된 함수
  - 중첩 함수는 자신이 소속되어 있는 함수의 매개변수에 접근할 수 있음.

```
>>> import math
>>> def stddev(*args):
 def mean():
 return sum(args)/len(args)
 def variance(m):
 total = 0
 for arg in args:
 total += (arg - m) ** 2
 return total/(len(args)-1)
 v = variance(mean())
 return math.sqrt(v)
```

중첩 함수

중첩 함수

```
>>> stddev(2.3, 1.7, 1.4, 0.7, 1.9)
0.6
```

## 6. 중첩함수

- ❖ 중첩함수의 자신이 소속되어 있는 함수 외부에서는 보이지 않음.

```
>>> mean()
Traceback (most recent call last):
 File "<pyshell#2>", line 1, in <module>
 mean()
NameError: name 'mean' is not defined
```



# 7.Lambda

## ❖ lamdba

- 이름이 없는 한 줄 짜리 함수
- 작성 방법

lambda <인수 나열>:<반환할 내용>

- 인수가 없으면 생략 가능

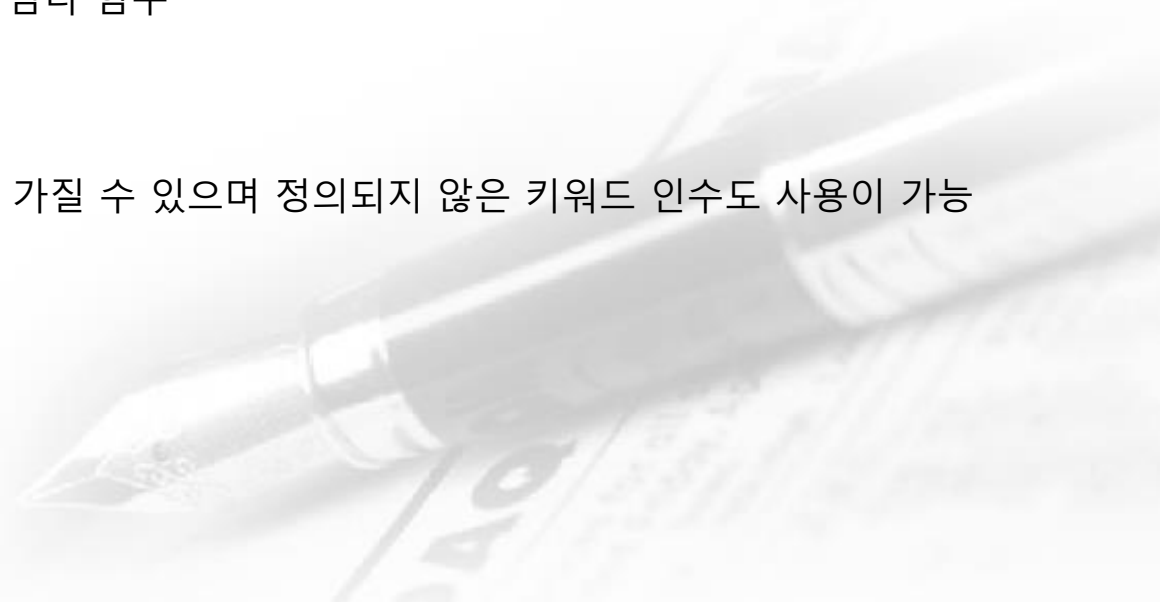
항상 1을 리턴하는 람다 함수

f = lambda:1

2개의 인수를 받아서 합을 구해주는 람다 함수

f = lambda x , y : x+y

- 람다 함수도 매개변수의 기본값을 가질 수 있으며 정의되지 않은 키워드 인수도 사용이 가능



## 8. 함수적 프로그래밍

### ❖ map 내장 함수

- 컬렉션과 함수를 매개변수로 받아서 컬렉션의 모든 데이터를 함수의 매개변수로 대입해서 결과를 리턴하는 함수

```
def f(x):
```

```
 return x*x
```

```
li = [1,2,3,4,5]
```

```
print('반복문을 이용한 실행')
```

```
for imsi in li:
```

```
 print(f(imsi), end=" ")
```

```
print('\nmap을 이용한 실행')
```


```
result = list(map(f,li))
```

```
print(result)
```

```
print('lambda와 map을 이용한 실행')
```

```
result = list(map(lambda x : x*x,li))
```

```
print(result)
```



## 8. 함수적 프로그래밍

### ❖ filter 내장 함수

- 컬렉션과 함수를 매개변수로 받아서 컬렉션의 모든 데이터를 함수의 매개변수로 대입해서 결과가 참인 경우만 리턴하는 함수

```
li = [1,2,3,4,5]
```

```
print('lambda와 filter을 이용한 실행')
```

```
result = list(filter(lambda x : x%2==0, li))
```

```
print(result)
```





# 모듈과 패키지

# 1. 모듈

❖ 같은 경로에 있는 example.py 의 hello() 모듈을 불러올 경우

```
import example
example.hello()
```

❖ from 파일이름 import 모듈이름

```
from example import sum
```

hello() # 위와 비교했을 때 'example.' 을 쓰지 않아도 되는 장점이 있습니다.

❖ 모듈에 별명 붙여서 사용하기

```
import example as ex
```

example 대신에 ex를 사용

❖ 다른 폴더에 있는 모듈 import 하기

```
sys.path.append("C:\Python\Mymodules")
```

# 위 디렉토리는 import 하고자 하는 모듈의 디렉토리

# 1.모듈

## ❖ 모듈

- 일반적으로는 “독자적인 기능을 갖는 구성 요소”를 의미
- 서로 연관된 작업을 하는 코드들의 모임으로 작성 중인 모듈의 크기가 어느 정도 커지게 되면 일반적으로 관리 가능한 작은 단위로 다시 분할
- 파이썬에서는 개별 소스 파일을 일컫는 말
- 파이썬 프로그램으로 작성된 파일도 가능하고 C나 Fortran 등으로 만든 파이썬 확장파일도 모듈이 될 수 있음

## ❖ 모듈의 종류

- 표준 모듈 : 파이썬과 함께 따라오는 모듈
  - 사용자 생성 모듈 : 프로그래머가 직접 작성한 모듈
  - 서드 파티(3rd Party) 모듈 : 파이썬 재단도 프로그래머도 아닌 다른 프로그래머, 또는 업체에서 제공한 모듈
- ❖ 외부 모듈을 사용할 수 있도록 추가하는 방법은 import 파일명
- ❖ 확장자는 생략하고 파일에 있는 변수나 메서드는 파일명.변수 또는 파일명.함수()로 호출

# 1.모듈

❖ mymath.py  
mypi = 3.14

```
def area(r):
 return mypi * r * r
```

❖ test.py  
import mymath

```
print(mymath.mypi)
```

```
print(mymath.area(5))
```



# 1. 모듈

- ❖ 자격(Qualified) 변수: 앞의 예에서 처럼 소속을 정확히 밝혀서 사용하는 변수
- ❖ 무자격(Unqualified) 변수: 소속을 밝히지 않고 사용하는 변수
- ❖ 파이썬은 모듈 가져오기를 수행하면 특별히 지정한 폴더에서 찾아가기 시작합니다.
- ❖ sys.path 변수에 그 순서가 기재되어 있습니다.

```
import sys
print(sys.path)
```

결과

```
['C:\\Users\\Administrator\\python\\test',
 'C:\\Users\\Administrator\\python\\test',
 'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35-
32\\DLLs',
 'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35-
32\\lib',
 'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35-
32',
 'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35-
32\\lib\\site-packages',
 'C:\\Users\\Administrator\\AppData\\Local\\Programs\\Python\\Python35-
32\\python35.zip']
```

# 1.모듈

❖ 직접 검색할 위치를 추가: `sys.path.append("검색할 경로")`

❖ 윈도우의 환경 변수에 추가

`PYTHONPATH = 검색경로;`

❖ 리눅스의 환경 변수에 추가

C Shell 인 경우: `setenv PYTHONPATH 검색경로`

bash Shell 인 경우: `export PYTHONPATH =검색경로`

❖ 절대 경로를 이용한 가져오기

- `import math` : `math` 모듈처럼 모듈의 이름만 가져온 경우로 `math.해서 사용`
- `from math import sin, cos, pi`: `math` 모듈에서 `sin, cos, pi` 만 가져온 경우로 `math.을 생략하고 사용 가능`
- `from math import *` : `math` 모듈의 모든 이름을 현재 위치로 가져옵니다.
- `import math as ma`: `math`라는 모듈 이름 대신에 `ma` 사용
- `from math import pi as py`: `pi` 대신에 `py` 사용
- 하나의 모듈에서 여러 개의 이름을 가져올 때는 괄호 가능
- 모듈 이름이 문자열로 되어 있는 경우 `__import__(모듈이름)` 으로 가져오는 것이 가능
- 모듈은 한 번 가져오면 메모리에 적재된 상태가 되므로 다른 모든 모듈에서 사용이 가능

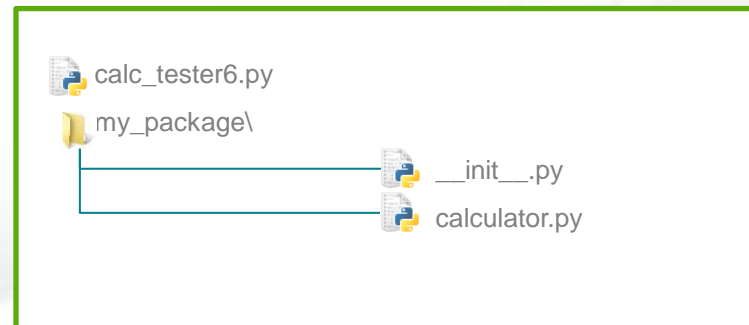
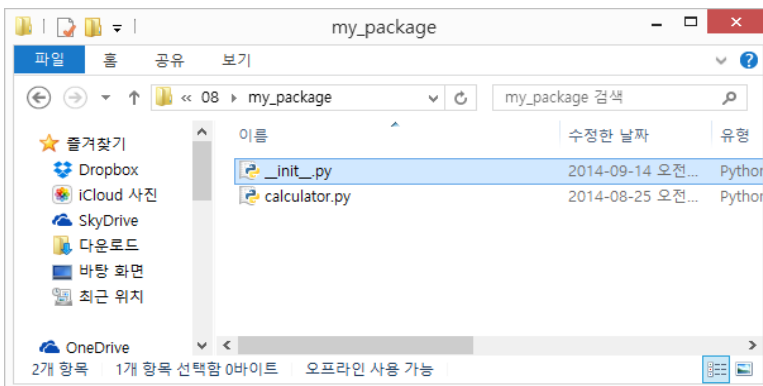
## 2. 패키지

### ❖ 패키지

- 모듈을 모아놓는 디렉토리
- 모듈 꾸러미로 해석하면 이해하기 편함
- 디렉토리가 "파이썬의 패키지"로 인정받으려면 `__init__.py` 파일을 그 경로에 갖고 있어야 함
- `__init__.py` 파일에는 패키지를 초기화하는 어떠한 파이썬 코드도 포함할 수 있습니다.
- 패키지에서 특정한 모듈을 가져올 때는 아래 그림과 같은 경우

**from my\_package import calculator**

•



## 2. 패키지


- ❖ 프로젝트에 my\_package를 생성
- ❖ 위에서 생성한 패키지 안에 calculator.py 파일을 생성하고 내용을 작성

```
def plus(x, y):
 return x+y
```

```
def minus(x, y):
 return x-y
```

- ❖ test.py 파일을 생성하고 위의 패키지를 사용하는 코드를 작성

```
from my_package import calculator
result = calculator.plus(10, 30)
print("결과:{0}".format(result))
result = calculator.minus(10, 30)
print("결과:{0}".format(result))
```





## 2.패키지

### ❖ site-packages

- 파이썬의 기본 라이브러리 패키지 외에 추가적인 패키지를 설치하는 디렉토리
- 각종 서드 파티 모듈을 바로 이 곳에 설치함

### ❖ site-packages 확인

```
import sys
sys.path
['', 'C:\\Python34\\Lib\\idlelib',
'C:\\WINDOWS\\SYSTEM32\\python34.zip', 'C:\\Python34\\DLLs',
'C:\\Python34\\Lib', 'C:\\Python34',
'C:\\Python34\\Lib\\site-packages']
```

site-package는 파이썬이 기본적으로 모듈을 탐색하는 경로에 포함되어 있습니다.

# 3.파이썬 실행파일

- ❖ 디렉토리에 `__main__.py` 파일이 존재하는 경우 파이썬 프로그램 실행 `python 디렉토리이름`
- ❖ zip 파일로 압축되어 있는 경우에도 가능 `python zip파일이름`



# pip

❖외부 모듈 설치

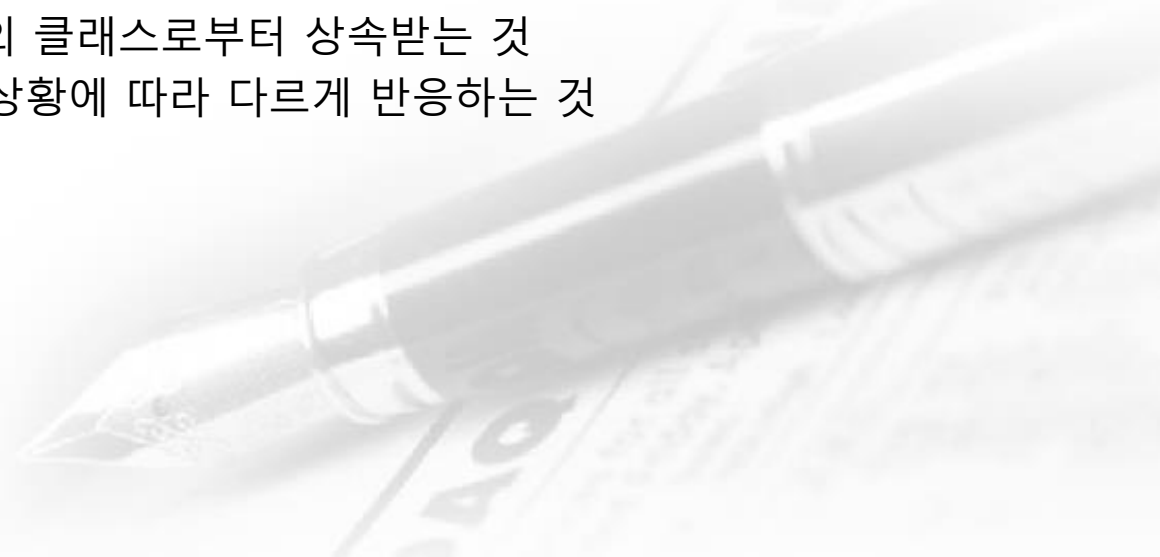
C:₩>pip install 외부모듈이름



# 클래스

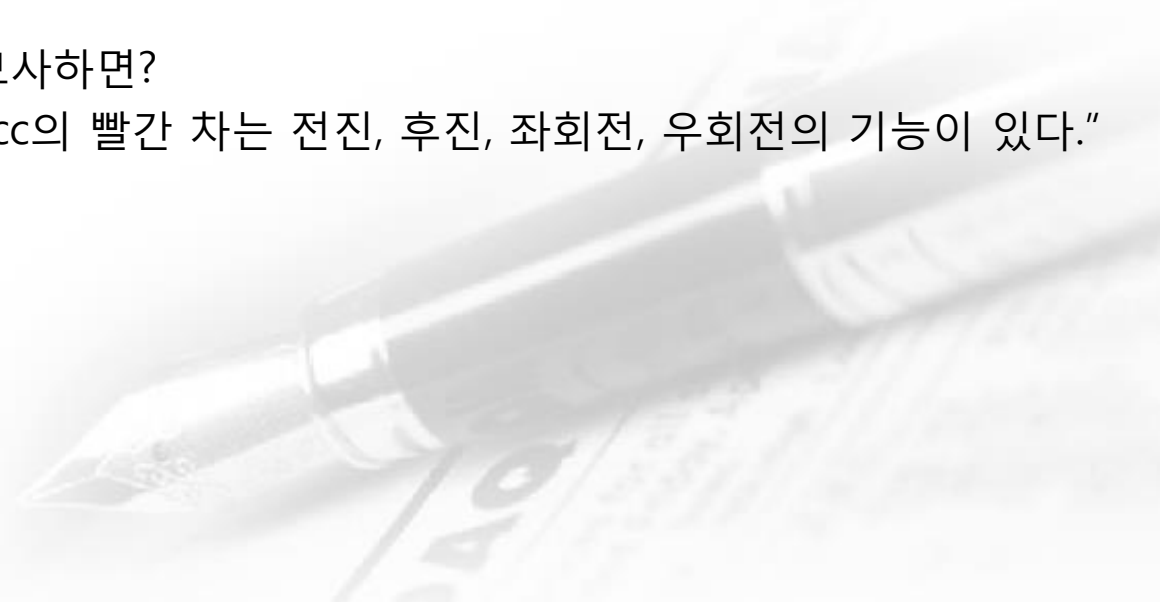
# 1. 클래스

- ❖ class: 관련있는 저장공간과 기능을 하나로 묶은 것 - Encapsulation
- ❖ class Object: 클래스와 동일한 의미로 사용하는데 어떤 클래스를 구체적으로 지정하기 위해 사용
- ❖ class instance: 클래스를 호출하여 생성된 객체
- ❖ method: 클래스 안에 정의된 함수
- ❖ member variable: 클래스 안에 정의된 변수
- ❖ attribute: 클래스 안에 있는 모든 것
- ❖ Inheritance: 하위 클래스가 상위 클래스의 모든 속성을 물려받는 것
- ❖ Super Class: Base Class라고도 하는데 다른 클래스의 상위 클래스
- ❖ Sub Class: Derived Class라고도 하는데 다른 클래스로부터 속성을 물려받는 클래스
- ❖ Multiple Inheritance: 2개 이상의 클래스로부터 상속받는 것
- ❖ Polymorphism: 동일한 코드가 상황에 따라 다르게 반응하는 것



# 1. 클래스

- ❖ 객체(Object) = 속성(Attribute) + 기능(Method)
- ❖ 파이썬의 클래스는 클래스도 하나의 객체
- ❖ 속성은 사물의 특징
  - 예) 자동차의 속성 : 바디의 색, 바퀴의 크기, 엔진의 배기량
- ❖ 기능은 어떤 것의 특징적인 동작
  - 예) 자동차의 기능 : 전진, 후진, 좌회전, 우회전
- ❖ 속성과 기능을 들어 자동차를 묘사하면?
  - "18인치 바퀴를 가진 2,000cc의 빨간 차는 전진, 후진, 좌회전, 우회전의 기능이 있다."



# 1. 클래스

- ❖ 다음과 같이 묘사한 자동차를 코드로 표현
  - "18인치의 바퀴를 가진 2,000cc의 빨간 차는 전진, 후진, 좌회전, 우회전의 기능이 있다."

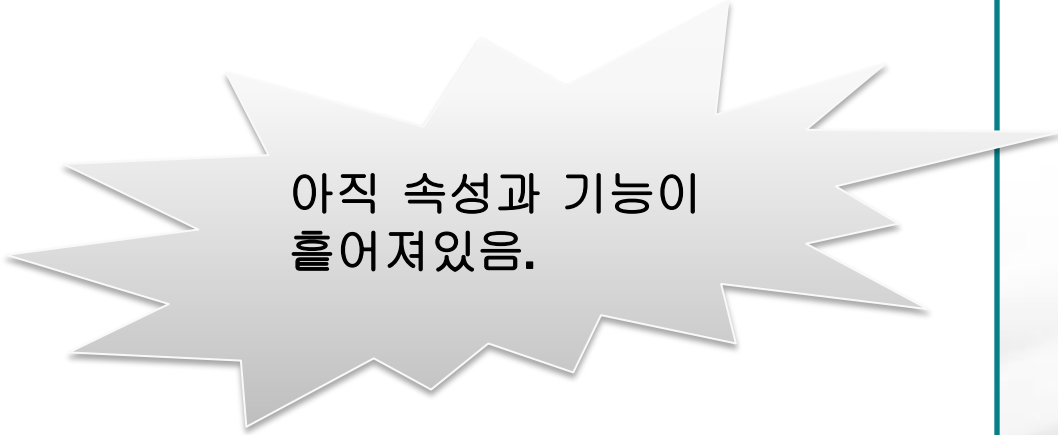
```
color = 0xFF0000 # 바디의 색
wheel_size = 16 # 바퀴의 크기
displacement = 2000 # 엔진 배기량
```

```
def forward(): # 전진
 pass
```

```
def backward(): # 후진
 pass
```

```
def turn_left(): # 좌회전
 pass
```

```
def turn_right(): # 우회전
 pass
```



아직 속성과 기능이  
흘어져있음.

# 1. 클래스

❖ 다음과 같이 묘사한 자동차를 코드로 표현하면... (2)

- "18인치의 바퀴를 가진 2,000cc의 빨간 차는 전진, 후진, 좌회전, 우회전의 기능이 있다."

```
class Car:
```

Car 클래스의 정의 시작을 알립니다.

```
 def __init__(self):
```

```
 self.color = 0xFF0000 # 바디의 색
 self.wheel_size = 16 # 바퀴의 크기
 self.displacement = 2000 # 엔진 배기량
```

Car 클래스 안에 차의 색, 바퀴 크기, 배기량을 나타내는 변수를 정의합니다.

```
 def forward(self): # 전진
 pass
```

```
 def backward(self): # 후진
 pass
```

```
 def turn_left(self): # 좌회전
 pass
```

```
 def turn_right(self): # 우회전
 pass
```

Car 클래스 안에 전진, 후진, 좌회전, 우회전 함수를 정의합니다.



# 1. 클래스

- ❖ 앞에서 만든 Car 클래스는 자료형
- ❖ Car 클래스의 객체는 다음과 같이 정의함

```
num = 123 # 자료형:int, 변수: num
my_car = Car() # 자료형: Car 클래스, 객체: my_car
```

- ❖ 객체 대신 인스턴스(Instance)라는 용어를 사용하기도 함.
  - 클래스가 설계도, 객체는 그 설계를 바탕으로 실체화한 것이라는 뜻에서 유래한 용어
  - 객체뿐 아니라 변수도 인스턴스라고 부름. 자료형을 메모리에 실체화한 것이 변수이기 때문임.



# 1. 클래스

- ❖ 결합도는 한 시스템 내의 구성 요소간의 의존성을 나타내는 용어.
  - 소프트웨어에서도 결합도가 존재함.
  - 예) A() 함수를 수정했을 때 B() 함수의 동작에 부작용이 생긴다면 이 두 함수는 **강한 결합도**를 보인다고 할 수 있음.
  - 예) A() 함수를 수정했는데도 B() 함수가 어떤 영향도 받지 않는다면 이 두 함수는 **약한 결합**으로 이루어져 있다고 할 수 있음.
- ❖ 클래스 안에 같은 목적과 기능을 위해 묶인 코드 요소(변수, 함수)는 객체 내부에서만 강한 응집력을 발휘하고 객체 외부에 주는 영향은 줄이게 작성해야 합니다.
- ❖ 클래스를 잘 설계하면 유지보수가 편리해지고 재사용성이 높아집니다.



# 1. 클래스

- ❖ 클래스는 다음과 같이 `class` 키워드를 이용하여 정의.

```
class 클래스이름:
 코드블록
```

- ❖ 클래스의 코드블록은 변수와 메서드(Method)로 이루어짐.
  - 기능(Method) : 객체 지향 프로그래밍에서 사물의 동작을 나타냄.
  - 메서드(Method) : 객체 지향 프로그래밍의 기능에 대응하는 파이썬 용어. 함수와 거의 동일한 의미이지만 메서드는 클래스의 멤버라는 점이 다름.
  - 함수(Function) : 일련의 코드를 하나의 이름 아래 묶은 코드 요소.
- ❖ 객체의 멤버(메서드와 데이터 속성에 접근하기)
  - 객체의 멤버에 접근할 때는 점(.)을 이용.

```
my_car = Car()
print(my_car.color)
```

my\_car의 멤버에 접근하게 해줍니다.

## 2. 메서드

### ❖ 메서드 생성

- 메서드를 클래스 내부에 선언할 때는 **첫번째 매개변수** 는 무조건 현재 클래스의 객체가 되어야 합니다.
- 관습적으로 **self** 라는 단어를 이용합니다.
- 두번째 매개변수부터는 사용자가 정의할 수 있습니다.
- 메서드를 호출하는 방법
  - 클래스 이름을 이용한 호출(언바운드 호출)  
클래스이름.메서드이름(인스턴스이름, 매개변수)
  - 인스턴스 이름을 이용한 호출(바운드 호출)  
인스턴스이름.메서드이름(매개변수) ex) car01.speedUp()
  - 클래스 **내부**에서 자신의 클래스에 속한 메서드를 호출  
self.메서드이름(매개변수)

# 3. 변수

❖ Student.py 파일을 생성하고 작성

```
class Student:
```

```
 def start(self):
```

```
 print("안녕하세요")
```

```
 def printName(self, name):
```

```
 print("이름은 {}".format(name))
```

```
 def call(self):
```

```
 self.start() ## 내부에 있는 메서드 호출.
```

Student.start() - 언바운스 호출

st01 = Student ()

st01.start() - 바운스 호출

st01.printName("홍길동")



# 3. 변수

❖ \_\_main\_\_.py 파일을 생성하고 작성

```
from Student import student
stu = student()
student.start(stu)
stu.printName("중앙")
```



# 3. 변수

## ❖ 클래스 내의 변수 생성

- 메서드 외부에서 선언하면 클래스 변수: 클래스 내부에 1개만 만들어지며 클래스 이름으로 접근할 수 있고 객체 이름으로도 접근해서 사용할 수 있게 됩니다.
- 메서드 내부에서 self와 함께 선언되면 인스턴스 변수: 각 객체 내부에 각각 만들어지면 객체 이름으로만 접근해서 사용할 수 있습니다.
- 메서드 내부에서 self. 과 함께 변수를 선언하지 않으면 그 변수는 메서드의 지역변수가 됩니다.



# 3. 변수

❖ Student.py 파일을 수정 – 클래스 변수 생성

```
class student:
```

```
 schoolName = "Korea"
```





# 3. 변수

❖ \_\_main\_\_.py 파일을 수정

```
from Student import student
```

```
stu1 = student()
```

```
stu2 = student()
```

```
print("stu1의 주소:{0}".format(id(stu1)))
```

```
print("stu2의 주소:{0}".format(id(stu2)))
```

#위 2개의 객체는 각각 생성자를 호출해서 만들어 진 것이므로 서로 다른 영역을 차지하고 만들어집니다.

```
student.schoolName="Seoul"
```

```
'''
```

schoolName은 메서드 외부에 만들어진 것이므로 class 변수가 되고

클래스 변수는 1개만 만들어서

클래스와 클래스로부터 만들어진 객체 모두가 공유해서 사용합니다.

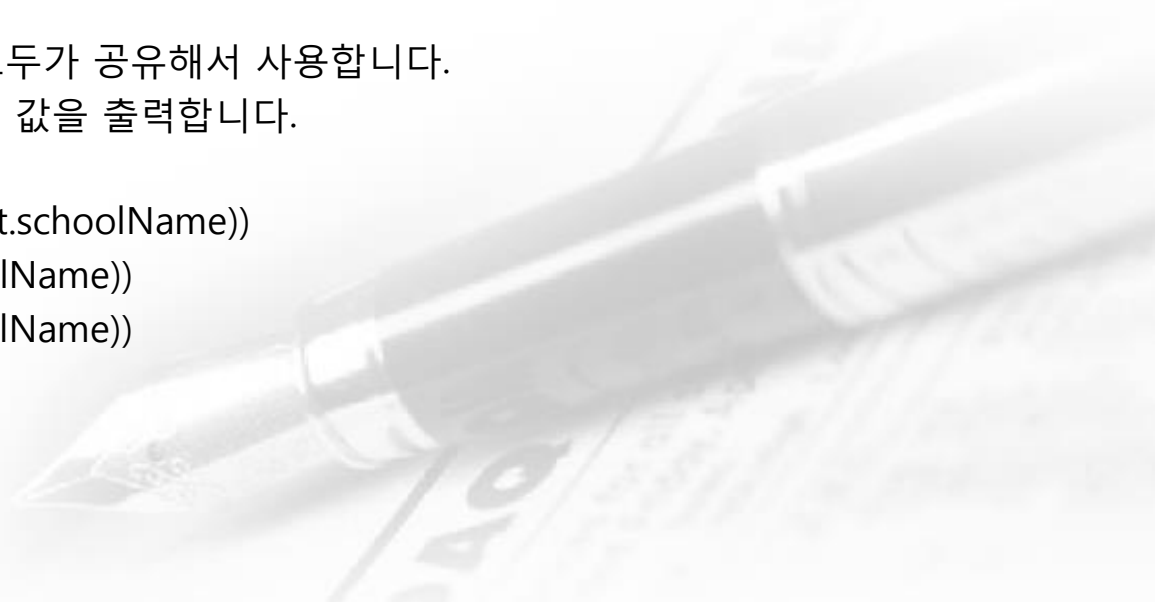
따라서 아래 3개의 출력문은 모두 동일한 값을 출력합니다.

```
'''
```

```
print("Student의 학교:{0}".format(student.schoolName))
```

```
print("stu1의 학교:{0}".format(stu1.schoolName))
```

```
print("stu2의 학교:{0}".format(stu2.schoolName))
```



# 3. 변수

❖ Student.py 파일을 수정 – 인스턴스 변수 생성

class student:

    schoolName = "Korea"

    def setName(self, name):

        self.name = name

    def getName(self):

        return self.name



# 3. 변수

❖ \_\_main\_\_.py 파일을 수정

```
from Student import student
stu1 = student()
stu2 = student()
stu1.setName("Steve Jobs")
stu2.setName("Steve wozniak")
print("stu1의 이름:{0}".format(stu1.getName()))
print("stu2의 이름:{0}".format(stu2.getName()))
```



## 4. 생성자와 소멸자

### ❖ 생성자: `__init__()`

- 객체가 생성된 후 가장 먼저 호출되는 메서드
- "초기화하다"는 뜻의 initialize를 줄여서 붙여진 이름
- 첫번째 매개변수는 `self` 이며 이후에 매개변수 추가 가능
  - `__init__(self, name, age)`
- `self` 이외의 매개변수가 있는 생성자를 만들면 인스턴스를 생성할 때 매개변수를 넘겨 주어야 합니다.
- 멤버 변수의 초기화 코드를 주로 작성합니다.

### ❖ 소멸자: `__del__()`

- 객체가 소멸될 때 호출되는 메서드
- 외부 자원을 사용하는 경우 해제하는 코드를 주로 작성
- `self` 이외의 매개변수를 받지 않습니다.

## 4. 생성자와 소멸자

❖ \_\_main\_\_.py 파일을 수정

```
from Student import student
stu1 = student()
stu2 = student()
print("stu1의 이름:{0}".format(stu1.getName()))
print("stu2의 이름:{0}".format(stu2.getName()))
```

위처럼 프로그램을 실행시키면 에러

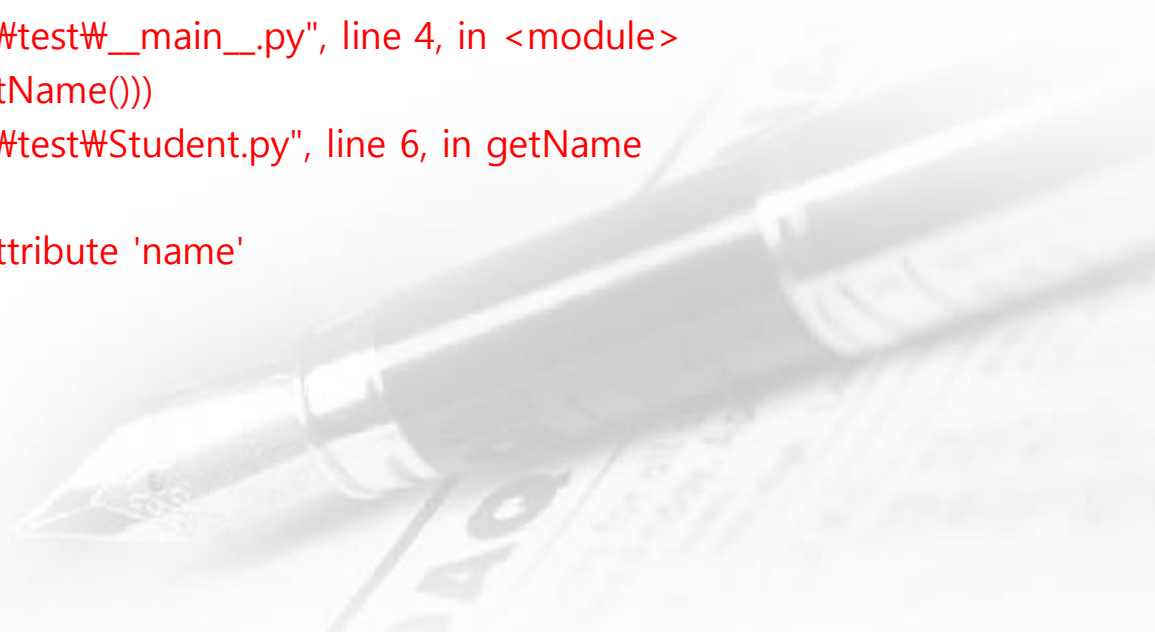
setName을 호출하기 전에 getName을 호출해서 아직 name이 만들어지기 전 상태

Traceback (most recent call last):

File "C:\Users\Administrator\python\test\\_\_main\_\_.py", line 4, in <module>  
 print("stu1의 이름:{0}".format(stu1.getName()))

File "C:\Users\Administrator\python\test\Student.py", line 6, in getName  
 return self.name

AttributeError: 'student' object has no attribute 'name'



## 4. 생성자와 소멸자

❖ Student.py 파일을 수정

```
from time import ctime
```

```
class student:
```

```
 #생성자
```

```
 def __init__(self, name="noname"): ## 2번째개값이 없을 default 설정
 print("{0}에 객체가 생성되었습니다.".format(ctime()))
 self.name=name
```

```
 schoolName = "Korea"
```

```
 def setName(self, name):
```

```
 self.name = name
```

```
 def getName(self):
```

```
 return self.name
```

```
 #소멸자
```

```
 def __del__(self):
```

```
 print("{0}에 객체가 소멸되었습니다.".format(ctime()))
```



## 4. 생성자와 소멸자

❖ \_\_main\_\_.py 파일을 수정

```
from Student import student
stu1 = student("중앙")
stu2 = student()
print("stu1의 이름:{0}".format(stu1.getName()))
print("stu2의 이름:{0}".format(stu2.getName()))
del stu1
del stu2
```



# 5.정적 메서드&클래스 메서드

## ❖ 정적 메서드

- 인스턴스를 생성하지 않고 클래스를 이용해서 직접 호출할 수 있는 메서드
  - ex) 클래스명.정적메서드명()
- 메서드 내에서 멤버 변수를 호출할 수 없습니다.
- 클래스 변수는 호출할 수 있습니다.
- @staticmethod 데코레이터로 수식
- self 키워드 없이 정의

```
class 클래스이름:
```

```
 @staticmethod
```

```
 def 메서드이름(매개변수):
```

```
 pass
```

@staticmethod데코레이터로 수식합니다.

self 매개변수는 사용하지 않습니다.



# 5.정적 메서드&클래스 메서드

## ❖ 클래스 메서드

- @classmethod 데코레이터로 수식
- 정적 메서드와 유사하지만 첫번째 매개변수로 클래스 객체가 전달되는 것이 다릅니다.
- cls 매개변수 사용

```
class 클래스이름:
 # ...
```

```
 @classmethod
 def 메서드이름(cls):
 pass
```

클래스 메서드를 정의하기 위해서는...  
1. @classmethod 데코레이터를 앞에 붙여줍니다.

2. 메서드의 매개변수를 하나 이상 정의합니다.

# 5.정적 메서드&클래스 메서드

❖ Student.py 파일을 수정

```
class student:
```

```
 @classmethod
```

```
 def cmethod(cls):
```

```
 print("클래스 메서드")
```

```
 print(cls)
```

```
 @staticmethod
```

```
 def smethod():
```

```
 print("정적 메서드")
```



## 5.정적 메서드&클래스 메서드

❖ \_\_main\_\_.py 파일을 수정  
from Student import student  
student.cmethod()  
student.smethod()



# 6.Property

## ❖ private 멤버 명명 규칙

- private 멤버는 클래스 내부에서는 접근이 가능하지만 클래스 외부(인스턴스)에서 접근이 되지 않는 멤버
- 이와 반대되는 개념으로 public이 있는데 public 멤버는 클래스 외부(인스턴스)에서 접근이 가능한 멤버입니다.
- python의 모든 멤버는 기본적으로 public
- 문법적으로 지원하지는 않고 이름을 이용해서 private 이라는 의미만 전달
- **두 개의 밑줄 \_ 이 접두사** 여야 합니다.
- 접미사는 **밑줄이 한 개** 까지만 허용됩니다.
- 접미사의 밑줄이 두 개 이상이면 public으로 간주합니다.

## ❖ Property

- 변수를 호출하는 것 처럼 사용하지만 실제로는 getter 와 setter 메서드를 호출하는 것으로 처리되는 속성
- 변수명 = property(fget=None, fset=None, fdel=None, doc=None)
- fget은 getter, fset은 setter, fdel은 삭제할 때 사용할 메서드

# 6.Property

❖ Student.py 파일을 수정

class student:

```
def __init__(self, name="noname"):
```

```
 self.__name = name
```

```
def setName(self, name):
```

```
 print("setter 호출")
```

```
 self.__name = name
```

```
def getName(self):
```

```
 print("getter 호출")
```

```
 return self.__name
```



# 6.Property

❖ \_\_main\_\_.py 파일을 수정

```
from Student import student
```

```
stu = student()
```

#아래 문장은 변수를 호출한 것으로 메서드 호출문은 아닙니다.

```
stu.name = "kim"
```

```
print(stu.name)
```



# 6.Property

- ❖ Student.py 파일을 수정: 아래처럼 수정하면 변수를 호출하는 코드를 작성해도 getter와 setter 메서드를 호출합니다.

class student:

```
def __init__(self, name="noname"):
 self.__name = name
```

```
def setName(self, name):
 print("setter 호출")
 self.__name = name
```

```
def getName(self):
 print("getter 호출")
 return self.__name
```

```
name = property(getName, setName)
```



# 예외처리



# Exception

- ❖ 예외(Exception)는 문법적으로는 문제가 없는 코드를 실행하는 중에 발생하는 오류

```
def func(y):
 x = input("숫자를 입력하세요:")
 return int(x) ** y
print(func(2))
```

숫자를 입력하세요:t

**Traceback (most recent call last):**

**File "C:\Users\Administrator\python\test\\_\_main\_\_.py", line 4, in  
<module>**

**print(func(2))**

**File "C:\Users\Administrator\python\test\\_\_main\_\_.py", line 3, in func  
return int(x) \*\* y**

**ValueError: invalid literal for int() with base 10: 't'**

# try ~ except

- ❖ try 절 안에 예외 발생 가능성이 있는 코드 블록을 배치
- ❖ except 절에는 예외가 발생했을 때 처리를 하는 코드 블록 배치

```
print(1/0)
```

```
Traceback (most recent call last):
 File "C:\Users\Administrator\python\test__main__.py",
 line 1, in <module>
 print(1/0)
ZeroDivisionError: division by zero
```

```
try:
 print(1/0)
except:
 print("예외가 발생했습니다.")
```



# try ~ except

- ❖ try 블록 안에서 여러 종류의 예외가 발생하는 경우에는 except에 예외형식을 기재해서 분리해서 처리 가능

**try:**

# 예외가 발생할 가능성이 있는 코드

**except 예외형식1:**

# 예외형식1에 해당하는 예외가 발생했을 때 수행될 코드

**except 예외형식2:**

# 예외형식2에 해당하는 예외가 발생했을 때 수행될 코드

```
li = [1, 2, 3]
```

```
try:
```

```
 index = int(input("첨자를 입력하세요: "))
```

```
 print(li[index]/0)
```

```
except ZeroDivisionError:
```

```
 print("0으로 나눌 수 없습니다.")
```

```
except IndexError:
```

```
 print("잘못된 첨자입니다.")
```

index가 0~2사이로 입력된다면 ZeroDivisionError가 일어납니다.

index가 0~2를 벗어나면 li[index]에서 IndexError가 발생합니다.

# try ~ except

❖ 예외의 인스턴스를 활용하는 방법 : as 문 사용

try:

# 문제가 없을 경우 실행할 코드

except 예외형식1 as err:

# 문제가 생겼을 때 실행할 코드

except 예외형식2 as err:

# 문제가 생겼을 때 실행할 코드

```
li = [1, 2, 3]
```

```
try:
```

```
 index = int(input("첨자를 입력하세요:"))
```

```
 print(li[index]/0)
```

```
except ZeroDivisionError as err:
```

```
 print("0으로 나눌 수 없습니다. ({0})".format(err))
```

```
except IndexError as err:
```

```
 print("잘못된 첨자입니다. ({0})".format(err))
```

# try ~ except

- ❖ except 뒤에 else가 오면 try에 대한 else가 아닌 "except절에 대한 **else**"

```
try:
 # 실행할 코드블록
except:
 # 예외 처리 코드블록
else:
 # except절을 만나지 않았을 경우 실행하는 코드블록
```

```
li = [1, 2, 3]

try:
 index = int(input("첨자를 입력하세요:"))
 print(li[index])
except IndexError as err:
 print("잘못된 첨자입니다. ({0})".format(err))
else:
 print("리스트의 요소 출력에 성공했습니다.")
```

# try ~ except

- ❖ Finally는 예외 발생 여부에 상관없이 수행되는 문장을 작성하는 영역
- ❖ finally가 else는 함께 사용하는 것도 가능함.

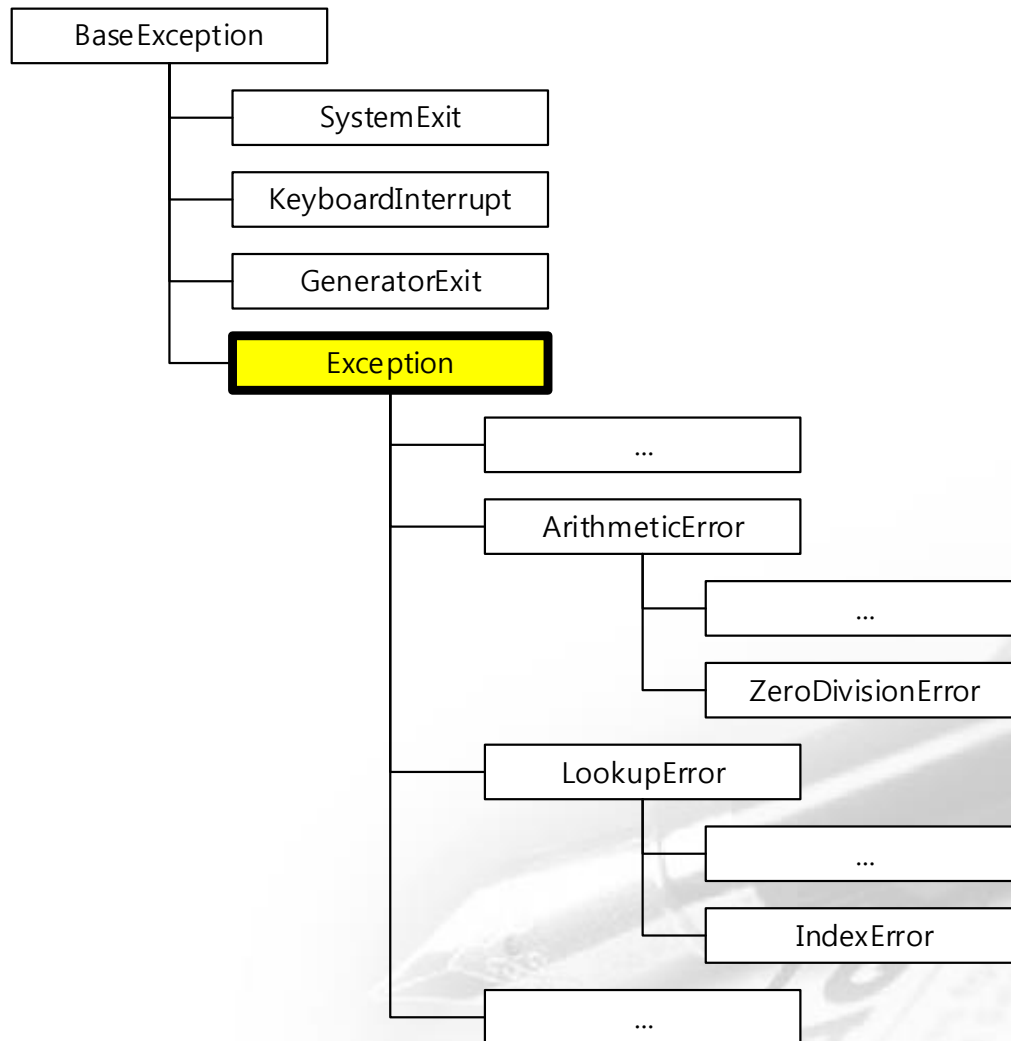
```
try:
 # 코드 블록
except:
 # 코드블록
else:
 # 코드블록
finally:
 # 코드블록
```

```
li = [1, 2, 3]

try:
 index = int(input("첨자를 입력하세요:"))
 print(li[index])
except IndexError as err:
 print("잘못된 첨자입니다. ({0})".format(err))
else:
 print("리스트의 요소 출력에 성공했습니다.")
finally:
 print("무조건 수행")
```

# Exception 클래스

- ❖ 파이썬에서 제공하는 예외 형식들은 거의 모두 Exception 클래스로부터 파생



# 내장 예외의 강제 발생

- ❖ raise문을 이용하면 내장 예외를 강제로 일으킬 수 있음.

```
text = input()
if text.isdigit() == False:
 raise Exception("입력받은 문자열이 숫자로 구성되어 있지 않습니다.")
```

raise문을 통해 예외를 일으킵니다.

```
>>> raise Exception("예외를 일으킵니다.")
Traceback (most recent call last):
 File "<pyshell#1>", line 1, in <module>
 raise Exception("예외를 일으킵니다.")
Exception: 예외를 일으킵니다.
```

예외를 처리하는 곳이 없다 보니 파이썬 인터프리터가 받아 예외 정보를 출력했습니다.

```
>>> try:
 raise Exception("예외를 일으킵니다.")
except Exception as err:
 print("예외가 일어났습니다. : {0}".format(err))
```

예외가 일어났습니다. : 예외를 일으킵니다.



# 예외의 외부 처리

```
def some_function():
 print("1~10 사이의 수를 입력하세요:")
 num = int(input())
 if num < 1 or num > 10:
 raise Exception("유효하지 않은 숫자입니다.: {0}".format(num))
 else:
 print("입력한 수는 {0}입니다.".format(num))

try:
 some_function()
except Exception as err:
 print("예외가 발생했습니다. {0}".format(err))
```

함수 안에서 일어난 예외가 except문으로 처리되지 않으면 함수 밖으로 다시 던져집니다.



# 사용자 정의 예외

- ❖ 파이썬이 제공하는 내장 예외 형식만으로 충분하지 않을 때 직접 예외 클래스를 정의할 수 있음.
- ❖ 사용자 정의 예외 클래스는 Exception 클래스를 상속하여 정의함.

```
class MyException(Exception):
 pass
```

- ❖ 필요에 따라 다음과 같이 데이터 속성이나 메서드를 추가 가능.

```
class MyException(Exception):
 def __init__(self):
 super().__init__("MyException이 발생했습니다.")
```



# 사용자 정의 예외

```
class InvalidIntException(Exception):
 def __init__(self, arg):
 super().__init__('정수가 아닙니다.: {0}'.format(arg))

def convert_to_integer(text):
 if text.isdigit(): # 부호(+, -) 처리 못함.
 return int(text)
 else:
 raise InvalidIntException(text)

if __name__ == '__main__':
 try:
 print('숫자를 입력하세요:')
 text = input()
 number = convert_to_integer(text)
 except InvalidIntException as err:
 print('예외가 발생했습니다 ({0})'.format(err))
 else:
 print('정수 형식으로 변환되었습니다 : {0}({1})'.format(number, type(number)))
```

# assert

- ❖ 예외가 발생할 상황이 아님에도 사용자가 강제로 예외를 만들 수 있습니다.
- ❖ `assert <중단 조건>, <에러 메시지>`

```
a = 30
margin = 2 * 0.2
assert margin > 10, '마진이 작습니다.'
```

Traceback (most recent call last):

File "C:\Users\Administrator\python\test\\_\_main\_\_.py", line 3, in <module>

assert margin > 10, '마진이 작습니다.'

AssertionError: 마진이 작습니다.

