



**파이썬**



## 파이썬

발행일	2018년 10월 31일
지은이	홍순구
발행처	KGITBANK

본 책 내용의 전부 또는 일부를 재사용하려면 반드시 저작권자의 동의를 받으셔야 합니다.

## 〈차례〉

1장. 파이썬 개요 및 개발환경 구성 .....	1
2장. 자료형과 연산자 .....	23
3장. 데이터 구조 .....	79
4장. 제어문 .....	129
5장. 함수 .....	149
6장. 모듈과 패키지 .....	185
7장. 객체와 클래스 .....	209
8장. 예외처리 .....	227
9장. 입/출력 프로그래밍 .....	235
10장. 데이터베이스 연동 .....	241
11장. 웹 데이터 수집 .....	259





## 1장. 파이썬 개요 및 개발환경 구성

## 1. 파이썬 개요

파이썬(Python)<sup>1)</sup>은 1989년 암스테르담에서 프로그래머인 귀도 반 로섬(Guido van Rossum)에 의해 개발되기 시작한 프로그래밍 언어입니다. 파이썬은 플랫폼 독립적이며 인터프리터식이고, 객체지향적이며, 동적 타이핑(dynamically typed) 대화형 언어입니다.

파이썬은 언어 자신의 기능은 작게 하며 사용자가 언제나 필요로 하는 최소한의 기능만을 제공하도록 만들어져 있습니다. 이로 인해 파이썬의 코드들은 다른 유저가 코딩 했더라도 동일한 작업을 하는 프로그램은 대체로 비슷한 코드로 수렴합니다.

파이썬은 기본 기능에 없는 많은 기능이 라이브러리에 의해서 제공되고 있습니다. 파이썬에서는 프로그램의 문서화가 매우 중시되고 있어 언어의 기본 기능에 포함되어 있습니다. 예를 들면 코드 블록을 들여쓰기를 이용하도록 강제화 합니다.

파이썬은 원래 교육용으로 설계되었기 때문에 읽기 쉽습니다. 그래서 효율적인 코드를 가능한 간단하게 쓸 수 있도록 하려는 철학이 있기 때문에 파이썬 커뮤니티에서도 알기 쉬운 코드를 선호하는 경향이 강합니다.

다음은 파이썬의 특징을 몇 가지 요약한 것입니다.

- 가독성 - 문법이 간결하고 들여쓰기를 기반으로 가독성이 좋습니다.
- 확장성 - 풍부한 라이브러리를 바탕으로 무궁한 확장성이 있습니다.
- 접착성 - C로 구현된 모듈을 쉽게 만들어 붙일 수 있습니다. 물론 그 반대 기능도 가능합니다.
- 유니코드 - 문자열이 모두 유니코드로 나타납니다.(3.x이상에서)
- 동적 타이핑 - 동적 언어(Dynamic Language)이며, 인터프리터 형 언어(Interpreted Language)입니다.

파이썬은 데이터 분석을 위한 머신러닝 라이브러리를 제공하는 패키지들과 텐서플로우가 파이썬으로 구현되면서 데이터를 분석하려는 개발자들에게 더욱 더 인기가 높아졌습니다.



그림 1. 파이썬 로고

1) 파이썬이라는 이름은 귀도 반 로섬이 좋아하는 코미디 <Monty Python's Flying Circus>의 이름에서 따온 것입니다.

## 2. 파이썬 설치

### 2.1. 파이썬 인터프리터<sup>2)</sup>

파이썬의 공식 사이트는 <https://www.python.org/>입니다. 이곳에서 파이썬 인터프리터를 다운로드 받을 수 있습니다.

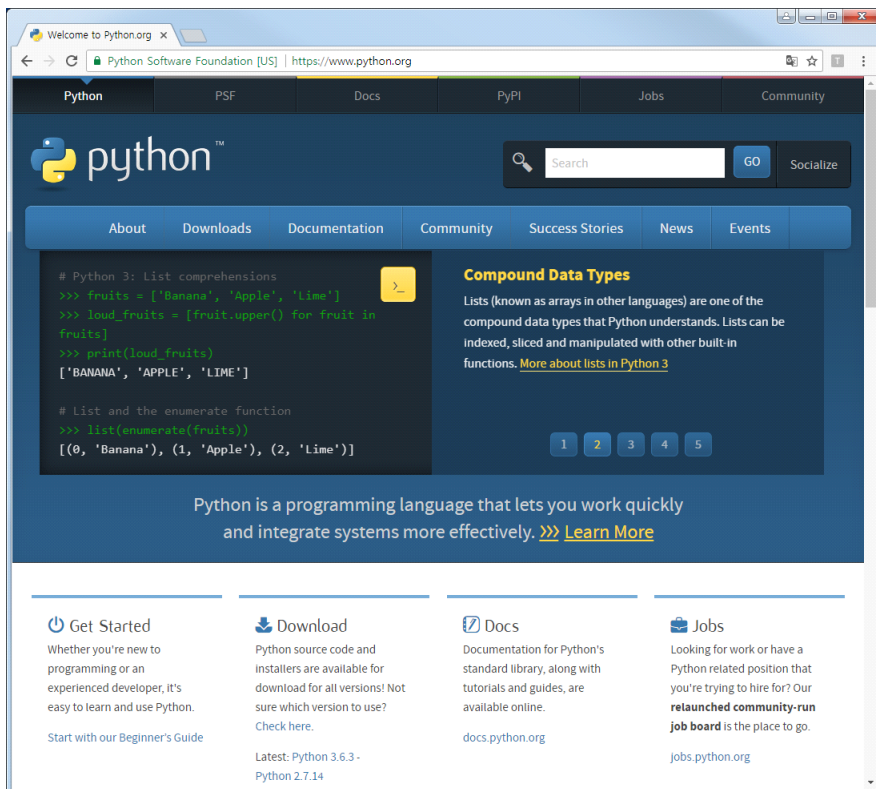


그림 2. 파이썬 공식 사이트

여러분은 <https://www.python.org/downloads/>에서 가장 최신의 파이썬을 다운로드 받을 수 있습니다.

파이썬은 2.x 버전과 3.x 버전을 다운로드 받을 수 있습니다. 이 교재는 3.x 버전을 기준으로 설명합니다. 필요에 따라서 2.x 버전코드를 같이 제공하므로 여러분이 2.x버전을 사용해도 이 책을 이용하여 공부하는데 지장이 없습니다.

2) 다음 절에서 설명하는 아나콘다 배포판을 사용할 거라면 파이썬 인터프리터는 설치하지 않아도 됩니다.

본 교재는 윈도우용 파이썬 3.6.3버전을 다운로드 받아 설치하였습니다. 파이썬 설치경로는 C:\Python\Python36-32 로 하였습니다.<sup>3)</sup> 다운로드 받은 윈도우용 설치파일을 더블클릭 합니다. 파일 열기 보안 경고 창이 보인다면 [실행] 버튼을 클릭하세요.

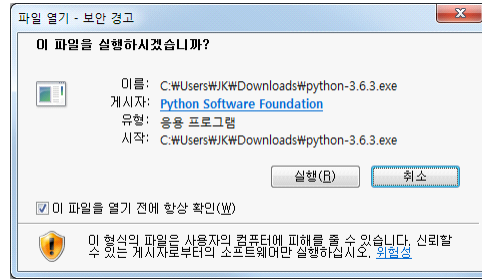


그림 3. 보안 경고

파이썬 설치 프로그램이 실행되면 다음 그림에서 보이는 것처럼 파이썬이 설치될 경로를 PATH에 추가해 주라는 [Add Python 3.6 to PATH]를 체크하고 [Customize installation]을 선택하세요.



그림 4. 설치 시작

Optional Features 창에서는 모든 항목이 선택되어 있는 기본값으로 두고 [Next] 버튼을 클릭합니다.

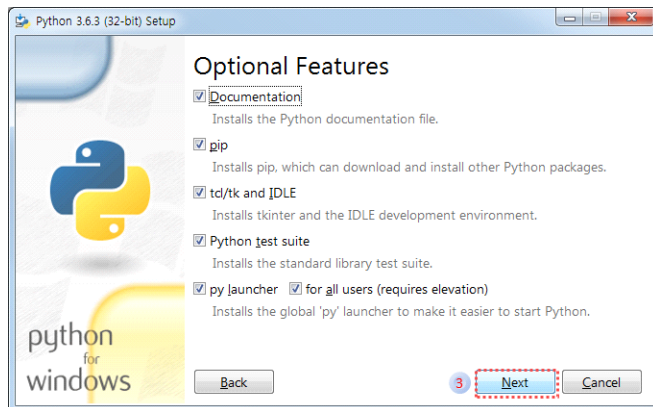


그림 5. Optional Features에서 [Next]

3) 설치 경로는 PATH 환경변수에 추가되어야 합니다. PATH 환경변수에 설치 디렉토리를 추가하는 것은 설치 시 [Add Python 3.6 to PATH]를 체크해주면 됩니다.



Advanced Options 창에서 파이썬이 설치될 위치를 선택하세요. 이 책에서는 설치 위치를 C:\Python\Python36-32로 합니다.

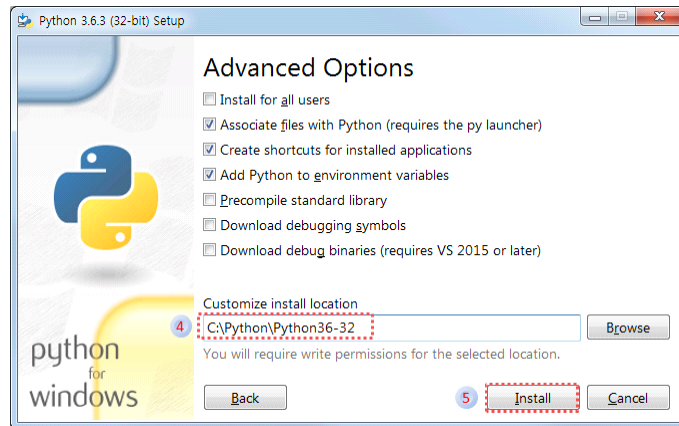


그림 6. Advanced Options에서 설치 위치 선택 후 [Install]

잠시만 기다리면 설치가 완료됩니다.

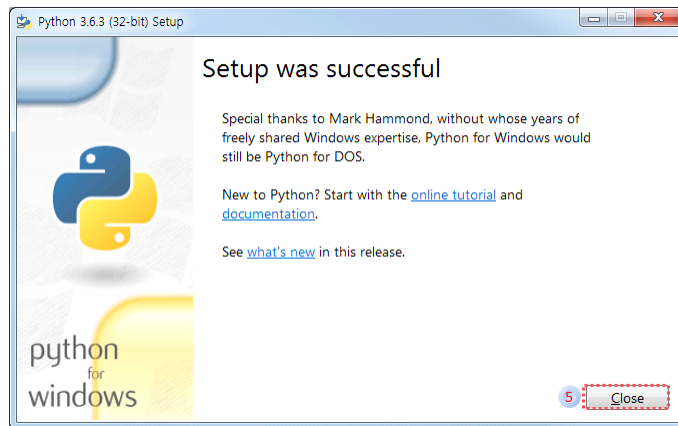


그림 7. 파이썬 설치 완료

설치 완료 창에 온라인 튜토리얼의 링크와 도큐먼트의 링크를 볼 수 있습니다.

- The Python Tutorial 링크 : <https://docs.python.org/3.6/tutorial/index.html>
- Python 3.6.x documentation 링크 : <https://docs.python.org/3.6/index.html>

파이썬이 정상 설치되었다면 C:\Python\Python36-32 디렉토리에서 Python.exe 실행파일을 더블클릭해 보세요. 그러면 파이썬 인터프리터 창이 실행되어야 합니다.

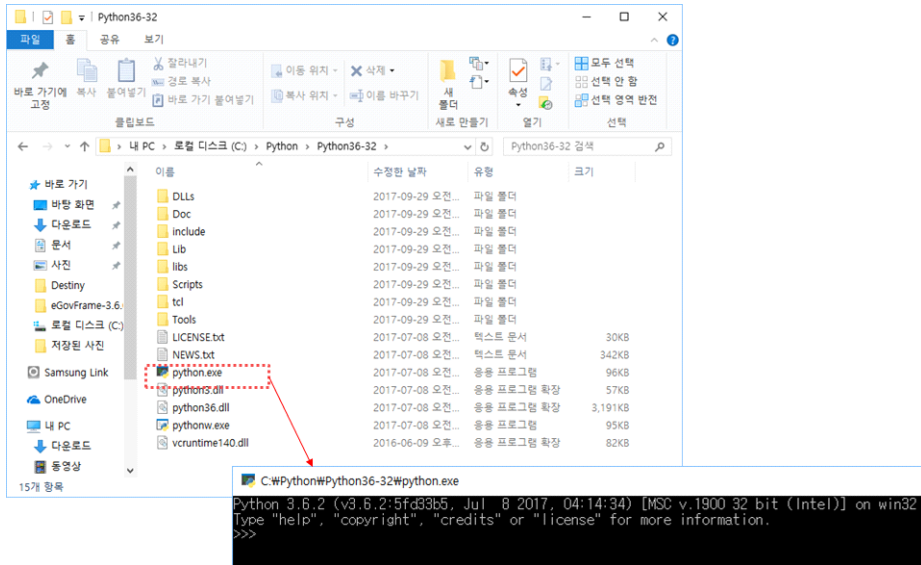


그림 8. 파이썬 인터프리터 실행

## 2.2. Hello World

파이썬 인터프리터를 실행시키고 “Hello World”를 출력해 보세요.

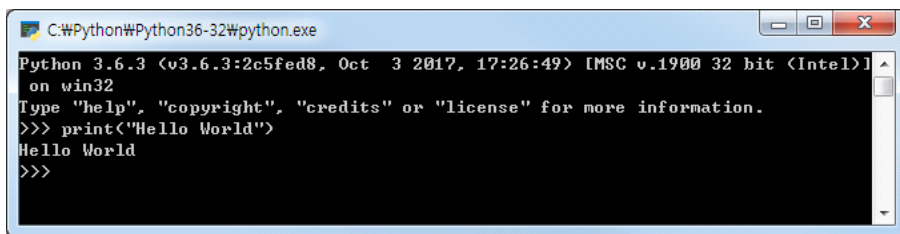


그림 9. Hello World

이제 여러분은 파이썬의 세계로 들어온 것입니다.

## 3. 아나콘다를 이용한 개발 환경 설정

### 3.1. 파이썬 배포판

우리가 실무에서 개발을 할 때 주요 기능들이 구현된 많은 패키지<sup>4)</sup>들을 사용합니다. 여러분이 만일 리눅스나 MacOS 운영체제를 사용한다면 pip<sup>5)</sup>로 패키지를 설치하는데 큰 문제가 없습니다. 그러나 Windows에서는 파이썬 인터프리터를 설치하고 pip를 이용해 여러 패키지를 설치하려 할 때에 의존성 등의 문제로 인해 필요한 패키지들을 일일이 설치하는 것이 상당히 불편합니다.

여러분이 윈도우에서 파이썬을 이용해 개발하기 위해 개발 환경을 구성하려면 파이썬 인터프리터를 설치하고, 개발 도구를 설치하고, pip를 설치하고, pip로 패키지를 설치하는 등 많은 복잡한 일들을 해야 합니다. 그러다보니 이러한 사용자들의 불편을 덜어주기 위해 윈도우 운영체제에서는 주요 패키지와 개발 환경이 포함된 파이썬 배포판을 사용합니다. 이런 파이썬 배포판은 데이터 처리 및 분석에 필요한 패키지가 모두 들어 있어서 데이터 과학 분야에서 널리 사용되고 있습니다. 파이썬 배포판에는 Anaconda, Winpython, python(x,y) 등이 있습니다.

### 3.2. 아나콘다 설치

파이썬 배포판은 여러 가지 종류가 있지만 이 책에서는 아나콘다(Anaconda)를 사용하겠습니다.

아나콘다 배포판(Anaconda Distribution)<sup>6)</sup>은 Continuum Analytics라는 곳에서 만든 파이썬 배포판으로, 1000개 이상의 데이터 사이언스 패키지들과 의존 패키지들을 포함하고 있습니다. 또한 상업용으로도 무료로 사용할 수 있으므로 회사 내에서도 사용할 수 있다는 장점이 있습니다.

웹 브라우저를 실행하고 다음 주소로 이동합니다.

● <https://www.anaconda.com/download/>

4) 패키지(Package)는 특정 기능을 수행하기 위하여 만들어 놓은 모듈 또는 라이브러리들의 모음입니다.

5) PIP(Pip Installs Packages)란 파이썬으로 작성된 패키지 소프트웨어를 설치, 관리하는 시스템입니다.

6) <https://www.anaconda.com/distribution/>

아나콘다 다운로드 페이지에서 자신의 플랫폼(운영체제)에 맞는 설치파일을 다운로드 받을 수 있습니다. 이 책에서는 윈도우용 아나콘다를 설치하겠습니다.

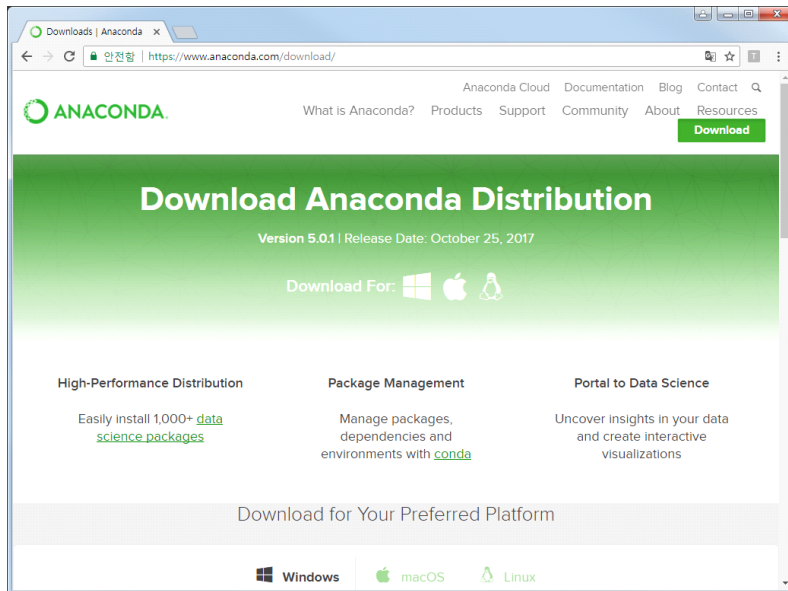


그림 10. <https://www.anaconda.com/download/>

다운로드 페이지에서 [Windows]를 클릭하면 파이썬 2.x 버전과 3.x 버전의 아나콘다를 다운로드 할 수 있습니다. 이 책에서는 파이썬 3.x 용 아나콘다를 다운로드 하겠습니다.

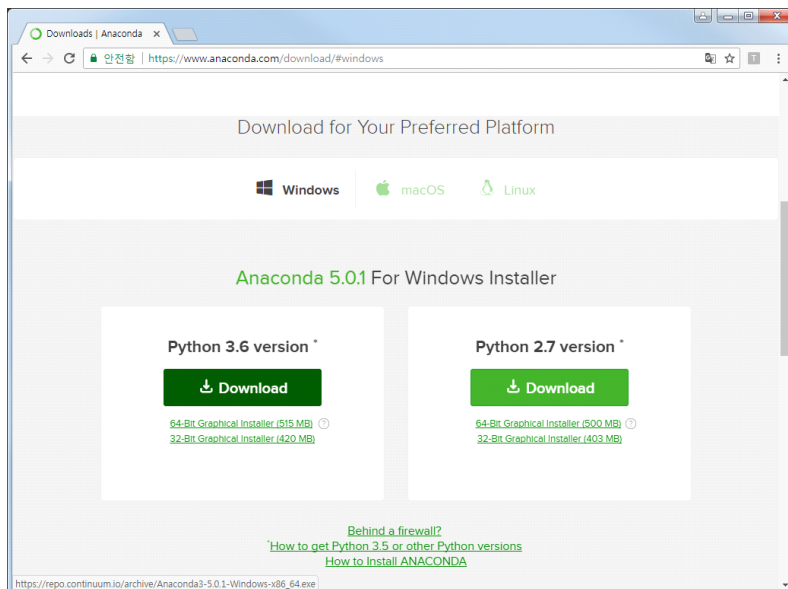


그림 11. Anaconda For Windows Installer

다운로드 받은 아나콘다 설치파일을 더블클릭하면 설치를 시작합니다.



그림 12. 윈도우용 아나콘다 설치파일

실행 시 보안 경고 창이 뜨면 [실행(R)] 버튼을 클릭하세요.

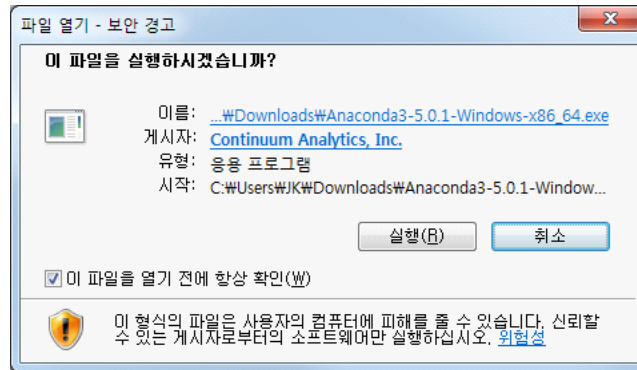


그림 13. 파일 열기 - 보안 경고

설치 시작 화면에서는 Next를 클릭하여 설치를 시작합니다.

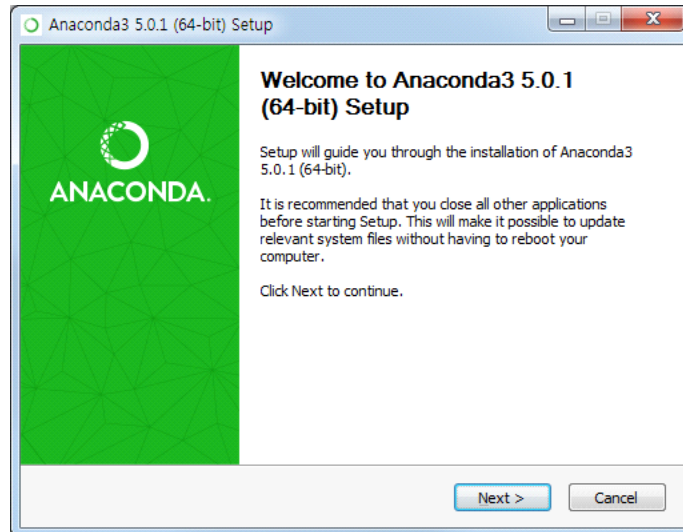


그림 14. 설치 시작

Anaconda End User License Agreement 문서를 읽고 [I Agree]를 클릭하여 라이선스를 동의합니다.

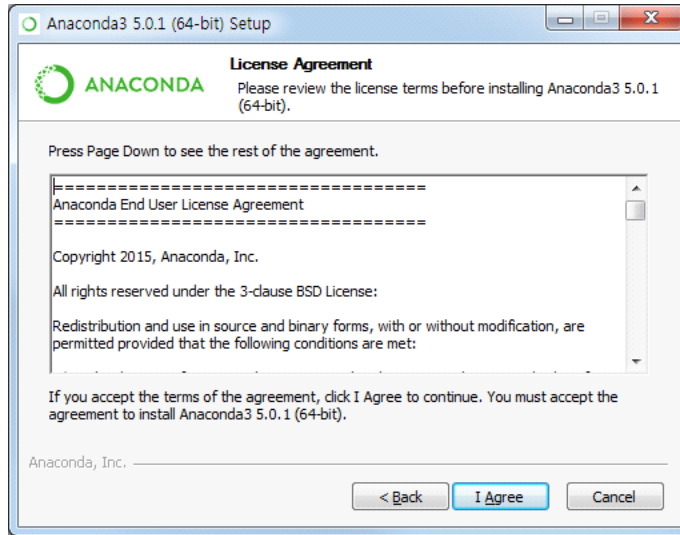


그림 15. 라이선스 동의

설치 타입을 선택하고 Next를 클릭합니다. Select type은 어느 것을 선택해도 상관없습니다.

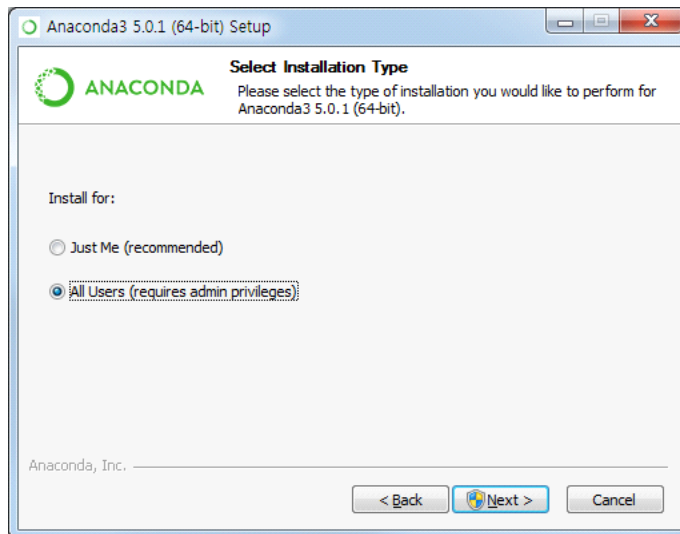


그림 16. 설치 타입 선택

설치 디렉토리 경로를 입력하고 Next를 클릭합니다. 이 책에서 기본 설치 경로는 C:\ProgramData\Anaconda3 입니다.

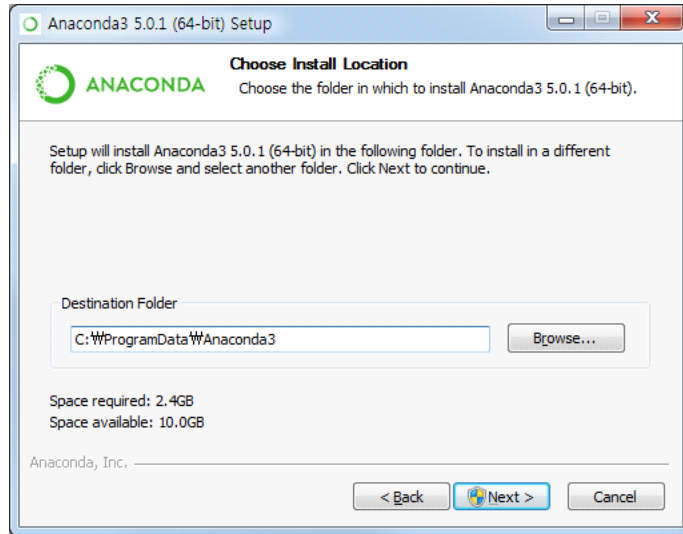


그림 17. 설치 위치 지정

고급 설치 옵션에서 [Register Anaconda as the system Python 3.6]을 선택(기본값)하고 [Install]버튼을 클릭하면 설치를 시작합니다.

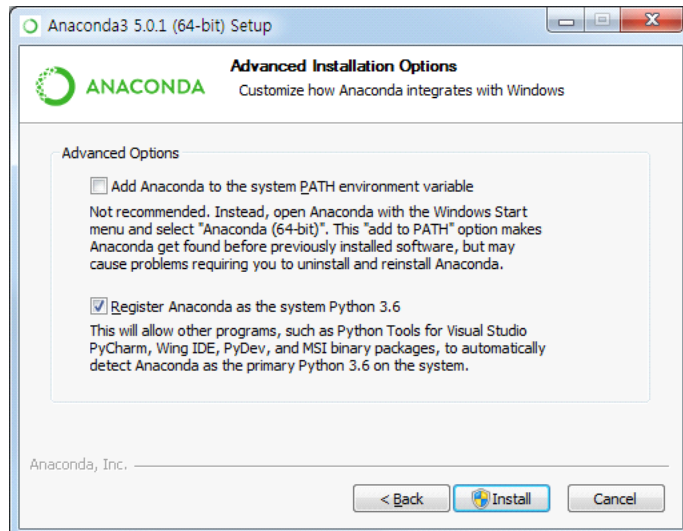


그림 18. 고급 설치 옵션

잠시만 기다리면 설치가 완료됩니다. [Next] 버튼을 클릭하세요.

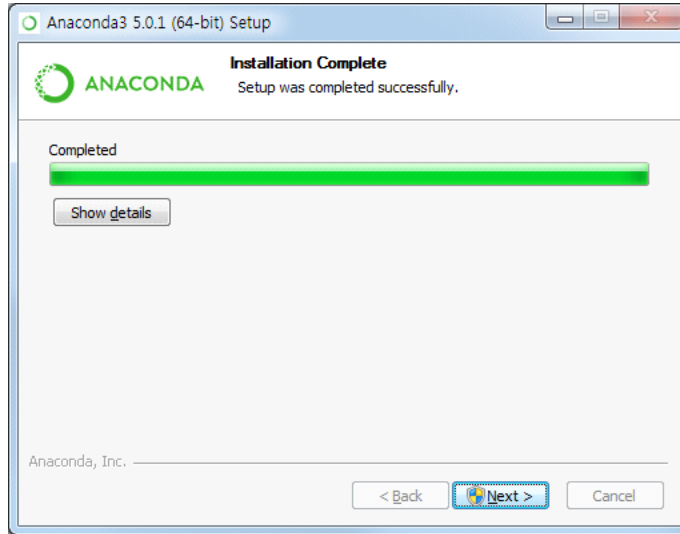


그림 19. 설치 완료

[Thanks for installing Anaconda3!] 창에서 [Finish] 버튼을 클릭하면 웹브라우저에 아나콘다 클라우드 홈페이지<sup>7)</sup>와 아나콘다 지원 홈페이지<sup>8)</sup>를 볼 수 있습니다.

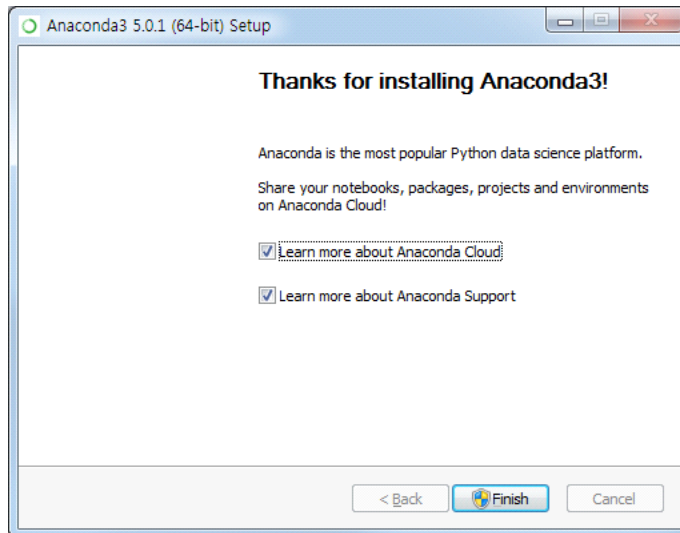


그림 20. 아나콘다 설치 종료, Finish

7) <https://anaconda.org/>

8) <https://www.anaconda.com/support/>



설치가 완료되면 윈도우의 시작메뉴에 Anaconda3 프로그램들이 추가됩니다.

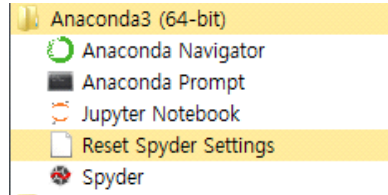


그림 21. 아나콘다 시작 메뉴

아나콘다의 메뉴들에 대한 설명은 다음과 같습니다.

- Anaconda Navigator : 아나콘다의 환경과 프로젝트 및 설치 구성요소들을 관리할 수 있는 윈도우 애플리케이션입니다.
- Anaconda Prompt : 아나콘다 명령을 직접 실행시킬 수 있는 명령행 프롬프트입니다.
- Jupyter Notebook : 주피터 노트북을 실행시킵니다.
- Spyder : 파이썬 애플리케이션을 개발하기 위한 IDE입니다.

### 3.3. 아나콘다 네비게이터

아나콘다 시작 메뉴에서 아나콘다 네비게이터(Anaconda Navigator)를 실행시키면 아나콘다를 설치할 때 함께 설치되는 구성요소들을 확인할 수 있고 원하는 구성 요소를 실행시킬 수 있습니다.

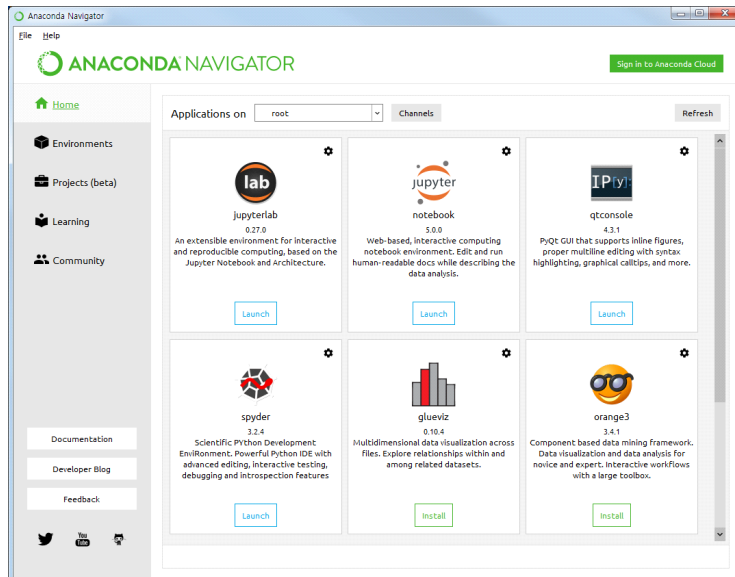


그림 22. 아나콘다 네비게이터

### 3.4. 주피터 노트북

주피터 프로젝트<sup>9)</sup>의 주피터 노트북(Jupyter Notebook)은 라이브 코드(live code), 방정식 작성(equation), 시각화(visualization) 그리고 설명문(explanatory text) 작성 등을 할 수 있는 웹 애플리케이션입니다. IPython Notebook과 유사하지만 서버를 두고 작성하기 때문에 공유가 가능하다는 장점이 있습니다. 주피터 노트북은 데이터 정제(data cleaning), 변환(transformation), 수치 시뮬레이션(numerical simulation), 통계 모델링(statistical modeling), 기계학습(machine learning) 외에도 많은 용도로 사용이 가능한 도구입니다.

주피터 노트북은 <https://try.jupyter.org> 를 통해 어떻게 동작하는지 확인이 가능합니다. 이곳에서 언어별로 어떻게 코드를 작성하고 실행할 수 있는지를 볼 수 있습니다.

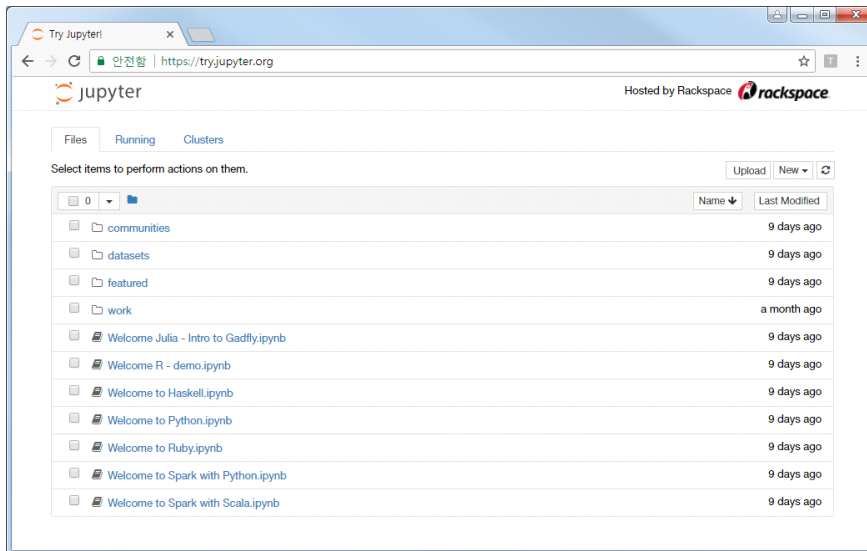


그림 23. <https://try.jupyter.org>

#### 1) 주피터 노트북

주피터 노트북은 아나콘다를 이용해서 설치할 수 있으며, pip를 이용해서도 설치할 수 있습니다. 이미 아나콘다를 설치했다면 주피터 노트북도 같이 설치되어 있습니다. 그러므로 주

9) 주피터 프로젝트(Project Jupyter)는 2014 년에 IPython Project에서 태어난 비영리 오픈 소스 프로젝트로서 모든 프로그래밍 언어에서 대화식 데이터 과학 및 과학 컴퓨팅을 지원하기 위해 발전했습니다. 주피터는 주피터 커뮤니티의 합의를 통해 깃허브(GitHub)에서 공개적으로 개발되었으며, 항상 100% 오픈소스 소프트웨어이며, 수정 된 BSD 라이선스의 자유주의적 조건하에 모든 사람이 무료로 사용할 수 있습니다.

피터 노트북을 다시 설치할 필요는 없습니다.

만일 파이썬이 설치되어 있는 상태에서 주피터 노트북을 설치하려면 다음처럼 pip 명령을 이용해 설치할 수 있습니다.

```
C:\Users\COM>pip install jupyter10)
```

```
Collecting jupyter
  Downloading https://files.pythonhosted.org/packages/83/df/0f5dd132200728a86190
397e1ea87cd76244e42d39ec5e88efd25b2abd7e/jupyter-1.0.0-py2.py3-none-any.whl
Collecting qtconsole (from jupyter)
  Downloading https://files.pythonhosted.org/packages/37/22/0d8474f78a8c421d485a
c2339de7c871d535160f09f170de90c8185b87c4/qtconsole-4.4.2-py2.py3-none-any.whl (1
12kB)
    100% |████████████████████████████████████████| 112kB 337kB/s
... 생략 ...
Installing collected packages: ipython-genutils, decorator, traitlets, pygments,
jupyter-core, tornado, pyzmq, jupyter-client, backcall, colorama, simplegeneric
, pickleshare, wcwidth, prompt-toolkit, parso, jedi, ipython, ipykernel, qtconso
le, jsonschema, nbformat, Send2Trash, pywinpty, terminado, MarkupSafe, jinja2, t
estpath, mistune, defusedxml, pandocfilters, entrypoints, webencodings, bleach,
nbconvert, prometheus-client, notebook, widgetsnbextension, ipywidgets, jupyter-
console, jupyter
  Running setup.py install for backcall ... done
  Running setup.py install for simplegeneric ... done
  Running setup.py install for MarkupSafe ... done
  Running setup.py install for pandocfilters ... done
  Running setup.py install for prometheus-client ... done
Successfully installed MarkupSafe-1.0 Send2Trash-1.5.0 backcall-0.1.0 bleach-3.0
.2 colorama-0.4.0 decorator-4.3.0 defusedxml-0.5.0 entrypoints-0.2.3 ipykernel-5
.1.0 ipython-7.0.1 ipython-genutils-0.2.0 ipywidgets-7.4.2 jedi-0.13.1 jinja2-2.
10 jsonschema-2.6.0 jupyter-1.0.0 jupyter-client-5.2.3 jupyter-console-6.0.0 jup
yter-core-4.4.0 mistune-0.8.4 nbconvert-5.4.0 nbformat-4.4.0 notebook-5.7.0 pand
ocfilters-1.4.2 parso-0.3.1 pickleshare-0.7.5 prometheus-client-0.4.2 prompt-too
lkit-2.0.6 pygments-2.2.0 pywinpty-0.5.4 pyzmq-17.1.2 qtconsole-4.4.2 simplegene
ric-0.8.1 terminado-0.8.1 testpath-0.4.2 tornado-5.1.1 traitlets-4.3.2 wcwidth-0
.1.7 webencodings-0.5.1 widgetsnbextension-3.4.2
You are using pip version 9.0.3, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' comm
and.
```

```
C:\Users\COM>
```

10) `python -m pip install jupyter` 명령으로 실행시킬 수도 있습니다. 만일 파이썬 2.x 버전과 파이썬 3.x 버전을 같이 사용하고 있다면 파이썬 3.x의 인터프리터 이름은 `python3`일 수 있습니다.

이렇게 설치된 주피터 노트북은 jupyter notebook 명령으로 실행시킬 수 있습니다.

```
C:\Users\COM>jupyter notebook
```

```
[I 12:23:43.871 NotebookApp] Serving notebooks from local directory: C:\Users\COM
[I 12:23:43.872 NotebookApp] The Jupyter Notebook is running at:
[I 12:23:43.872 NotebookApp] http://localhost:8888/?token=21412f724ae354a654a583
fc89a123a9b551c3878fd5b0a3
[I 12:23:43.872 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[C 12:23:43.873 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
    http://localhost:8888/?token=21412f724ae354a654a583fc89a123a9b551c3878fd5b0a3
[I 12:23:43.919 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

- 주피터 노트북을 실행시킨 디렉토리가 주피터 노트북의 루트 디렉토리가 됩니다. 주피터 노트북을 실행할 때에는 소스코드가 있는 디렉토리에서 실행시키세요.

## 2) 주피터 노트북 실행

아나콘다를 설치한 윈도우 환경에서 주피터 노트북을 실행시키려면 아나콘다 네비게이터에서 Jupyter notebook의 [Launch] 버튼을 클릭하세요.

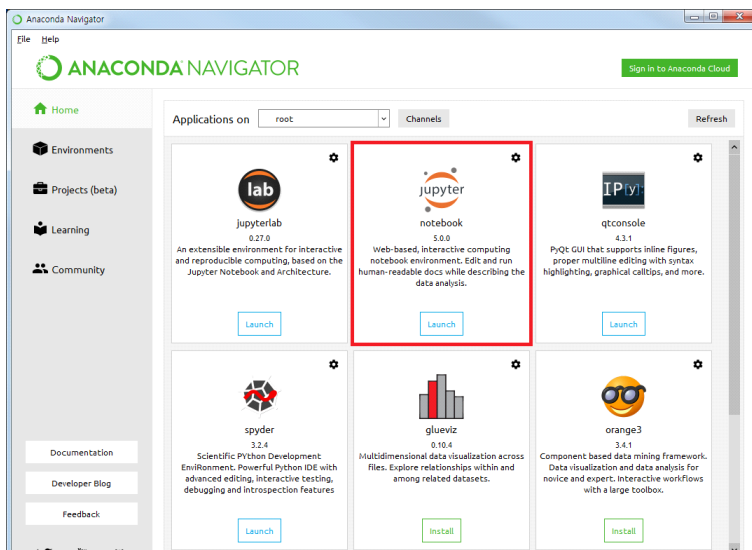


그림 24. 아나콘다 네비게이터에서 주피터 노트북 실행

아나콘다 네비게이터를 이용하지 않고 아나콘다 시작메뉴 아래에 있는 [Jupyter Notebook] 메뉴를 선택해도 주피터 노트북을 실행시킬 수 있습니다.

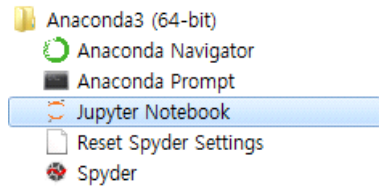


그림 25. Jupyter Notebook 실행 메뉴

주피터 노트북을 실행시키면 아래의 그림처럼 브라우저에 의해 <http://localhost:8888/tree>로 연결됩니다. 브라우저는 사용자의 홈 디렉토리(윈도우는 [내 문서])의 파일 목록을 보여줍니다.

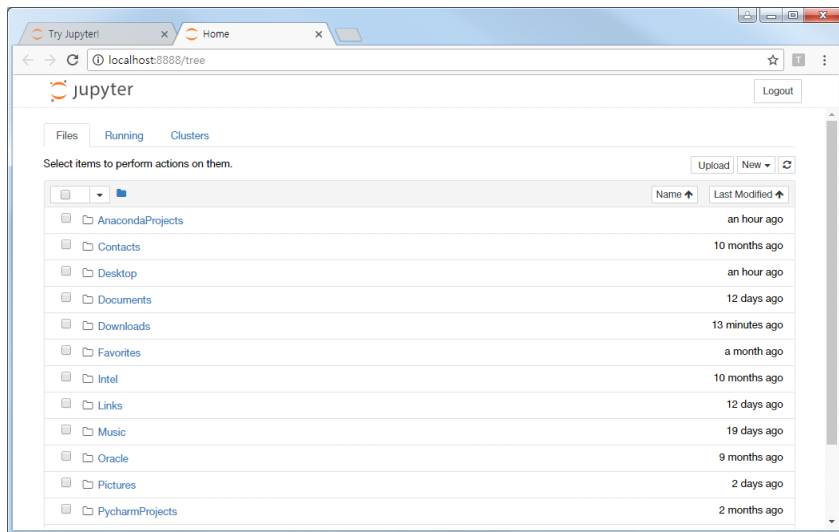


그림 26. 주피터 노트북 실행 화면

명령행 터미널(Command Prompt)에서 주피터 노트북 실행은 아래의 명령을 이용합니다. 주피터 노트북을 실행시키면 브라우저에 의해 주피터 노트북 페이지를 볼 수 있습니다.

```
jupyter notebook
```

주피터 노트북을 특정 포트로 실행시키고 싶다면 다음처럼 실행시키면 됩니다. 다음 예는 9999번 포트를 이용합니다.

```
jupyter notebook --port 9999
```

브라우저 실행 없이 서버만 실행하기 위해서 `--no-browser` 옵션을 추가하세요.

```
jupyter notebook --no-browser
```

주피터 노트북의 도움말을 보기 위해서 `--help` 옵션을 사용할 수 있습니다.

```
jupyter notebook --help
```

### 3.5. 주피터 노트북에서 코드 작성 및 실행

주피터 노트북을 실행시키고 소스코드를 작성하기 위해 AnacondaProjects 폴더 링크를 클릭합니다. 그리고 [New] 버튼의 하위 메뉴에서 [Python3] 메뉴를 선택하세요.

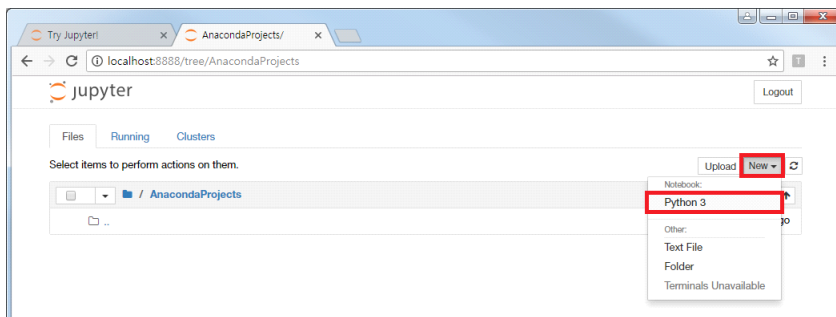


그림 27. 파이썬 코드 생성

브라우저 탭에 다음 그림과 같이 입력양식<sup>11)</sup>이 있는 화면이 만들어집니다.

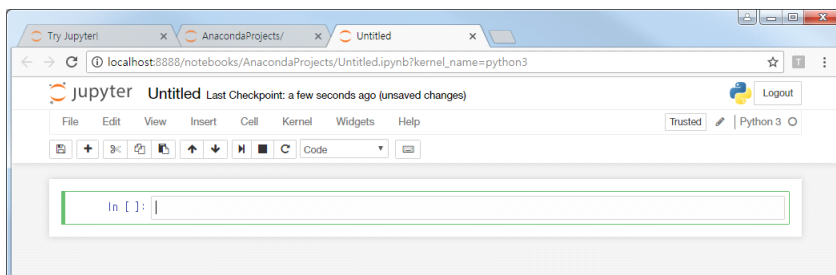



그림 28. 파이썬 코드 작성 및 실행을 위한 주피터 화면

11) 주피터 노트북에서 코드를 입력하는 곳을 셀(Cell)이라고 부릅니다.

주피터 노트북에서 파이썬 코드를 실행시키기 위해서는 3가지 방법이 있습니다.

- Shift+Enter 또는 메뉴에서  버튼 클릭 : 쉬프트키를 누른 상태에서 엔터키를 누르거나 현재 선택된 셀을 실행시키고 포커스를 다음 셀로 이동시킵니다. 다음 셀이 없을 경우 셀을 추가합니다.
- Ctrl+Enter : 컨트롤키를 누른 상태에서 엔터키를 누르면 현재 선택한 셀을 실행 시킵니다. 포커스는 다음 셀로 이동하지 않습니다.
- Alt+Enter : 알트(Alt)키를 누른 상태에서 엔터키를 누르면 현재 선택한 셀을 실행 시킵니다. 무조건 아래에 새로운 셀을 생성하고 포커스를 이동시킵니다.

코드를 실행시키면 입력 양식 옆의 In[ ]에는 실행시키는 순서에 따라 숫자가 차례대로 부여됩니다. 실행중일 경우에는 In[\*]로 표시됩니다.

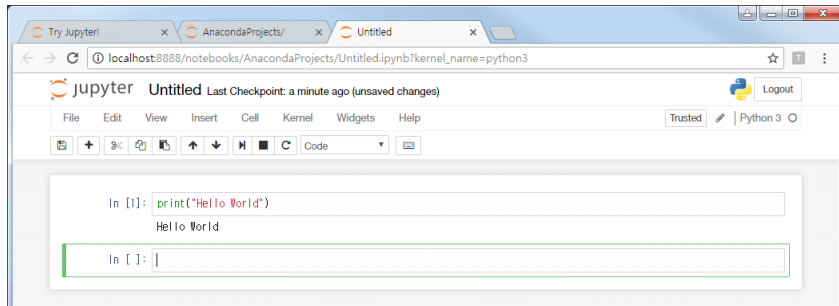


그림 29. 주피터 노트북에서 코드 실행

주피터 노트북으로 작성한 코드와 실행 결과를 저장할 수 있습니다. 파일의 이름은 화면에서 Untitled를 클릭하면 파일 이름을 변경할 수 있습니다. 새로운 이름을 입력하고 [Rename] 버튼을 클릭하면 새로운 이름으로 변경됩니다.

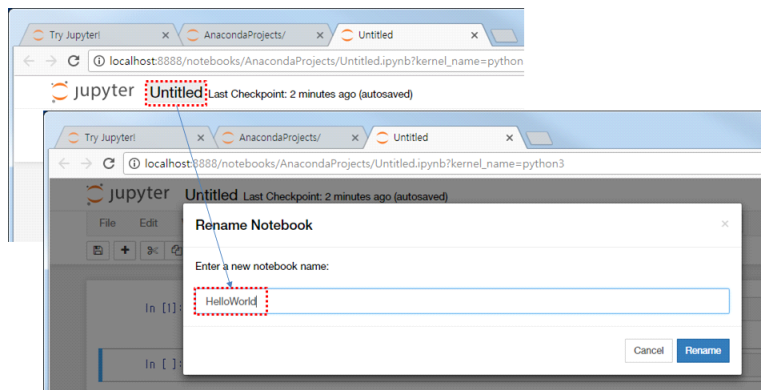


그림 30. 주피터 노트북 저장 파일이름 변경

파일이 저장되면 주피터 노트북의 [Files] 탭에서 확장자가 ipynb인 파일을 볼 수 있습니다.

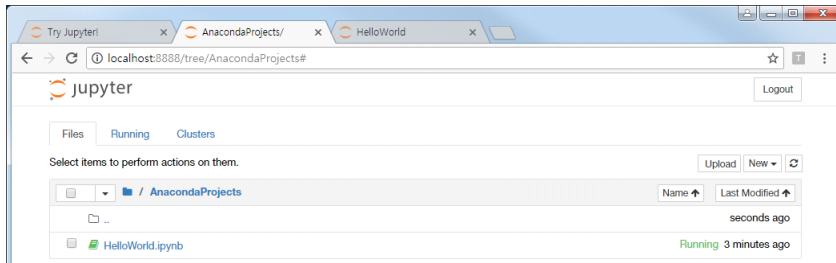


그림 31. 확장자가 .ipynb인 파일

### 3.6. 스파이더에서 코드 작성 및 실행

#### 1) 스파이더 실행화면

스파이더(Spyder) IDE의 실행 화면은 다음 그림처럼 Editor, Variable Explorer, IPython console 로 구분되어 있습니다.

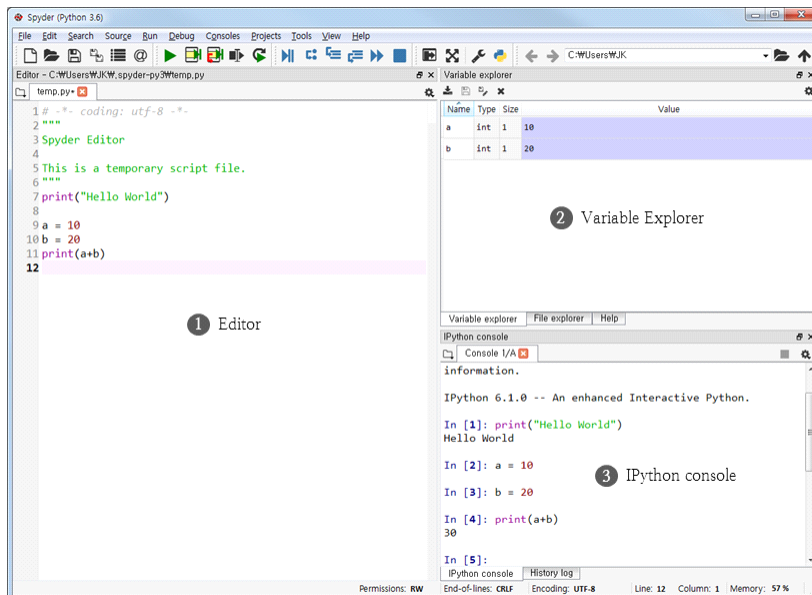


그림 32. Spyder 실행 화면

1. Editor : 파이썬 코드를 입력하는 곳입니다. 파이썬의 주석은 #으로 시작합니다. 그런데 스파이더 에디터에서는 독특한 주석인 '%%'를 사용합니다. 이 주석은 스파이더에서 코드들을 '셀(Cell)' 이란 것으로 구분하고 실행 시 셀 단위로 실행시키기 위한 주석



입니다.

2. Variable Explorer : 파이썬에서 실행한 코드에 의해 할당된 변수들의 목록과 값을 볼 수 있는 영역입니다.
3. IPython console : Editor에서 실행시킨 코드가 이곳에서 실제로 실행되며 결과를 출력합니다.
4. Spyder의 Editor 창과 IPython console 창에서 폰트의 크기 변경은 Ctrl키와 +(플러스)를 이용해 키우고 Ctrl키와 -(마이너스)를 이용해 줄일 수 있습니다.

## 2) 스파이더에서 코드 실행

스파이더는 코드를 실행시키기 위해 4가지 방법을 제공합니다.

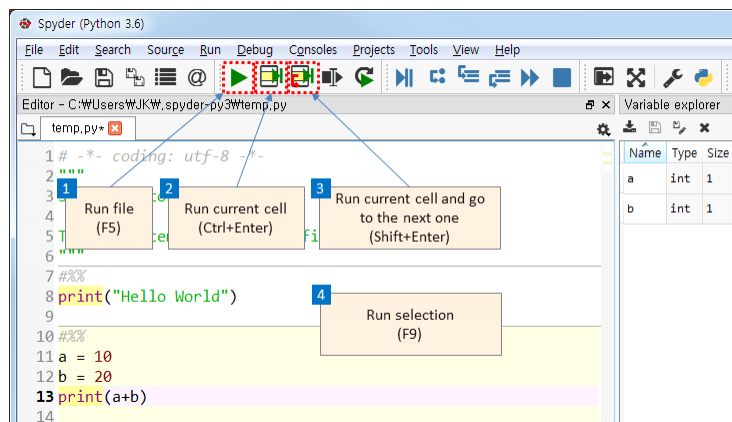


그림 33. 스파이더에서 코드 실행

- 1) 저장한 파일 전체를 처음부터 끝까지 한꺼번에 실행하는 것은 [Run file] 아이콘을 클릭하거나 F5 단축키를 누르면 됩니다.
- 2) [Run current cell] 아이콘 또는 Ctrl + Enter 단축키는 현재 커서가 위치한 셀 전체를 실행하고 커서를 현재 셀에 그대로 둡니다. 셀은 '%%'를 입력하여 코드 위/아래로 '선(line)'으로 구분이 된 코드들의 모음을 말합니다.
- 3) [Run current cell and go to the next one] 아이콘 또는 Shift + Enter 단축키는 현재 커서가 위치한 셀 전체를 실행 후 다음 셀로 커서를 이동시킵니다.
- 4) 단축키 F9는 현재 커서가 위치한 행(row) 또는 선택(블록 설정)한 행 전체를 실행 (Run selection) 합니다.

## 4. 연습문제

---

- 1) 파이썬을 여러분의 윈도우 환경에 설치해 보세요.
- 2) 아나콘다를 여러분의 윈도우 환경에 설치하고 주피터 노트북에서 'Hello World'를 출력해 보세요.

정답 및 풀이

1)

<https://www.anaconda.com/download/>

2)

jupyter notebook

## 2장. 자료형과 연산자

## 1. 파이썬 기본

### 1.1. 소스코드 인코딩

기본적으로, 파이썬 소스 파일은 utf-8로 인코딩되어 처리됩니다. 이 인코딩에서는 세계 대부분의 언어 문자를 문자열 리터럴(literal)<sup>12)</sup>, 식별자(identifier)<sup>13)</sup> 및 주석(comment)<sup>14)</sup>에서 동시에 사용할 수 있지만 표준 라이브러리는 이식 가능한 코드가 따라야 하는 규칙인 식별자에만 ASCII 문자를 사용 합니다. 이러한 모든 문자를 제대로 표시하려면 편집기에서 해당 파일이 utf-8인지 확인하고 파일의 모든 문자를 지원하는 글꼴을 사용해야 합니다.

우리는 한글을 처리하기 위해 utf-8을 사용하므로 별도로 인코딩을 지정하지 않아도 됩니다. 만일 인코딩을 지정하려면 다음 형식으로 지정합니다.

```
# -*- coding: utf-8 -*-
```

- 이 코드는 파이썬 파일의 맨 위에 입력합니다.

### 1.2. 변수

‘변수(variable)’는 모든 프로그래밍 언어에서 사용되는 기본 개념 중 하나입니다. 변수는 프로그램이 실행되는 동안 변할 수 있는 값을 저장하기 위해 사용합니다.

#### 1) 변수 이름 만드는 규칙

- 변수의 이름은 문자, 숫자, 밑줄(\_(underscore))문자를 포함
- 변수는 숫자로 시작할 수 없음
- 변수에 사용하는 문자는 대소문자를 구분
- 공백, 문장부호, 특수문자(밑줄문자만 유일하게 가능) 등은 사용할 수 없음
- 파이썬의 예약어(예: class, def)는 사용할 수 없음
- 이미 사용 중인 내장 함수<sup>15)</sup>나 모듈 이름(예: id, list, print)등은 사용 지양

12) 리터럴(literal)이란 소스 코드의 고정된 값을 대표하는 용어입니다.

13) 식별자(identifier)는 어떤 대상을 유일하게 식별 및 구별할 수 있는 이름을 뜻합니다. 프로그래밍 언어에서 식별자는 코드에 존재하는 변수, 자료형, 함수 등을 가리키는 이름입니다.

14) 주석(comment)은 컴퓨터 프로그래밍 환경에서 소스 코드에 도움이 되는 정보를 삽입하기 위해 사용합니다.

변수의 이름은 문자, 숫자, 밑줄(`_`(underscore))문자를 포함할 수 있지만 숫자는 변수의 맨 처음에 올 수 없습니다. 그리고 파이썬의 예약어는 변수 이름으로 사용할 수 없습니다. 변수에 사용하는 문자는 대소문자를 구분합니다. 변수 이름에 한글을 사용할 수는 있지만 권장하지 않습니다. 변수 이름에 공백, 문장부호, 특수문자(밑줄문자만 유일하게 가능) 등은 사용할 수 없습니다. 그리고 변수 이름에 파이썬의 예약어(예: `class`, `def`)는 사용할 수 없습니다. 이미 사용 중인 내장함수<sup>16)</sup>나 모듈 이름(예: `id`, `list`, `print`)등은 변수 이름에 사용을 지양합니다. 만일 변수 이름에 사용하고 싶다면 밑줄문자를 맨 뒤에 붙여서(예: `id_`, `list_`, `print_`) 사용하세요.

## 2) 변수에 값 할당

할당은 변수에 값을 저장하는 것을 의미합니다. 파이썬에서 변수에 값을 저장하기 위해서는 할당연산자<sup>17)</sup>를 사용합니다. 가장 많이 사용하는 할당 연산자는 “=” 입니다.

다음 코드는 변수 `a`에 정수 값 10을 할당하고 출력합니다.

```
a = 10
print(a)
```

```
10
```

변수는 = 연산자의 오른쪽에 올 수 없습니다. 다음 구문은 실행되지 않습니다.

```
10 = a
```

```
File "<ipython-input-1-a1ebfc217b9f>", line 1
  10 = a
      ^
SyntaxError: can't assign to literal
```

15) 프로그래밍 언어에서 함수(function)는 수학에서의 함수의 개념과 비슷합니다. 함수는 어떤 입력값을 받아 다른 값을 출력하도록 미리 만들어져 있는 것을 의미합니다. 함수를 사용하려면 ‘함수명()’ 형식으로 사용합니다. 한 쌍의 소괄호 안에는 함수가 실행되기 위해 필요한 값을 입력합니다.

16) `help()`, `print()`, `type()`, `input()` 등 자주 사용하는 함수들은 파이썬이 기본적으로 가지고 있는 함수들로 정의되어 있습니다.

17) 할당연산자는 변수에 값을 저장하는 연산자들을 의미합니다. `=`, `+=`, `-=`, `*=`, `/=`, `//=`, `**=` 등이 있으며 자세한 내용은 2장 4절 연산자에서 다룹니다.

파이썬에서는 변수를 선언하여 값을 저장할 때 변수의 타입을 지정하지 않습니다. 자바 또는 C언어 등에서 변수 이름 앞에 사용하는 데이터의 타입을 사용하지 않습니다.

```
int a = 10
```

```
File "<ipython-input-1-60c5c054435a>", line 1
    int a = 10;
      ^
SyntaxError: invalid syntax
```

### 3) id()

다음 코드는 변수 `x`와 `y`를 선언하고 각각 100과 200을 할당합니다. `print()` 함수로 `x`를 출력하면 `x`의 값이 출력됩니다. 변수는 파이썬에서 가장 많이 사용되는 객체입니다<sup>18)</sup>. 만일 객체의 주소 값을 알고 싶다면 `id()` 함수를 이용하세요.

```
x = 100
print(x)
```

```
100
```

```
print(id(x))
```

```
1566469376
```

### 4) 변수 삭제

변수를 삭제하려면 ‘`del 변수명`’을 이용하세요. 예를 들어 `x` 변수를 커널에서 삭제하고 싶다면 `del x` 하세요.

```
del x
```

구문에서...

- `x` : 삭제하고 싶은 변수명을 입력합니다. 여러 개 변수를 삭제하려면 콤마(,)를 이용해 변수들을 나열하면 됩니다.

18) 다른 언어와 달리 파이썬의 변수는 모두 객체입니다.

다음 코드는 변수 a, b에 각각 10, 20을 할당한 후 출력해 봅니다. print()함수는 인수의 값들을 공백으로 연결해서 출력합니다.

```
a = 10
b = 20
print(a, b)
```

```
10 20
```

```
del a, b
```

그런데 커널의 모든 변수를 삭제하려면 반복문과 전역객체에 대해 알아야 합니다. 다음 코드를 이해하지 못해도 됩니다. 다음 코드는 현재 커널의 모든 변수를 삭제하기 위한 구문일 뿐입니다. 이 책을 계속 보면 아래 구문에 대해 이해하게 됩니다.

```
for name in dir():
    if not name.startswith('_'):
        del globals()[name]
```

- for ~ in 구문은 동일한 구문을 반복 실행시키기 위한 구문입니다.
- globals()는 현재 커널에 선언되어 있는 전역변수 목록을 출력해 줍니다.
- del은 변수를 삭제합니다.

## 5) 다중 변수 선언

한 라인에 여러 개 변수를 선언해 사용할 수 있습니다.

```
a, b = 10, 20
print(a+b)
```

```
30
```

이때에 변수에 값의 할당 작업은 순차적으로 발생하는 것은 아닙니다. 특히 다중 변수 선언 및 할당을 할 때에 할당 연산자의 오른쪽에 변수가 올 경우...  
예를 들면 다음 구문이...

```
a, b = b, a+b
```

다음 구문과 같은 구문을 의미 하지는 않습니다.

```
a = b
b = a+b
```

\* 할당연산자(=)의 오른쪽 변수는 왼쪽의 변수 값이 바뀐다고 해서 같이 바뀌지는 않습니다. = 연산자가 실행되는 동안에 오른쪽 변수의 값은 변하지 않습니다.

실제 코드를 작성하고 실행하면 다음과 같습니다.

```
a = 10
b = 20
a, b = b, a+b
print(a, b)
```

```
20 30
```

```
a = 10
b = 20
a = b
b = a+b
print(a, b)
```

```
20 40
```

### 1.3. 도움말 기능

도움말은 help() 함수를 이용할 수 있습니다.

```
help([x])
```

구문에서...

- x : 도움말을 얻을 함수입니다.

다음 코드는 print 함수에 대한 도움말을 얻습니다.

```
help(print)
```

```
Help on built-in function print in module builtins:
print(...)
```



```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

help() 함수에 인수를 포함하지 않고 호출하면 도움말 유틸리티가 실행됩니다. 도움말 유틸리티프롬프트(help>)에서 원하는 단어를 입력해서 도움말을 볼 수 있습니다.

```
help()
```

Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely check out the tutorial on the Internet at <http://docs.python.org/3.6/tutorial/>.

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam".

```
help> input
```

Help on method raw\_input in module ipykernel.kernelbase:

```
input = raw_input(prompt='') method of
spyder.utils.ipython.spyder_kernel.SpyderKernel instance
    Forward raw_input to frontends
```

```
    Raises
```

```
    -----
```

```
    StdinNotImplemtenError if active frontend doesn't support stdin.
```

```
help> q
```

You are now leaving help and returning to the Python interpreter.

If you want to ask for help on a particular object directly from the interpreter, you can type "help(object)". Executing "help('string')" has the same effect as typing a particular string at the help> prompt.

- 도움말 유틸리티 종료 명령은 quit 또는 q입니다.

## 2. 화면 입출력

### 2.1. 사용자 입력 : input()

프로그램 실행 시 사용자의 입력을 받기 위해 input() 메서드를 사용합니다.

```
read_message = input("prompt_message")
```

구문에서...

- *prompt\_message* : input() 함수의 인자는 데이터를 입력받기 위한 프롬프트(prompt) 메시지를 지정합니다.

다음 코드는 두 수를 입력받아 두 수의 합을 출력합니다.

```
first_number = input('첫 번째 숫자:')
```

첫 번째 숫자:20

```
second_number = input('두 번째 숫자:')
```

두 번째 숫자:30

```
print(int(first_number) + int(second_number))
```

50

input() 함수를 이용해 입력받은 데이터의 타입은 모두 문자 타입입니다. 그러므로 덧셈 연산을 하려면 숫자 타입으로 변환해야 합니다. int() 함수는 인수를 숫자로 바꿔줍니다.

위 코드는 다음처럼 작성해도 됩니다.

```
first_number = int(input('첫 번째 숫자:'))
```

첫 번째 숫자:20

```
second_number = int(input('두 번째 숫자:'))
```

두 번째 숫자:30

```
print(first_number + second_number)
```

50

## 2.2. 화면으로의 출력 : print()

print는 파이썬 버전 3.x에서 함수로 바뀌었습니다. print() 함수의 인자 개수는 상관없으며, 출력하려는 값들을 공백으로 구분해서 출력합니다<sup>19)</sup>.

2.x의 print 문은 다음처럼 작성합니다.

```
print "welcome to", "python"
```

```
welcome to python
```

3.x의 print 문장은 다음처럼 작성합니다.

```
print("welcome to", "python")
```

```
welcome to python
```

\* 만일 여러분이 2.x를 사용 중 이라면 print문장은 3.x 코드를 2.x에서 그대로 사용할 수 있습니다. 그 이유는 3.x 버전의 print() 함수의 괄호가 2.x 버전에서는 출력할 문자열을 묶어주는 용도로 사용되기 때문입니다. 그러나 3.x에서는 2.x 버전의 print 구문을 그대로 사용할 수 없습니다.<sup>20)</sup>

print() 함수는 구분자(sep), 끝라인(end), 출력(file)을 키워드 파라미터로 지원합니다.

```
print('message', sep=' ', end='\n', file=sys.stdout)
```

구문에서...

- sep : 출력되는 문자열들의 구분자를 지정합니다.
- end : 행의 마지막에 포함될 문자를 지정합니다. 기본값은 개행(\n)<sup>21)</sup>입니다.
- file : 출력 대상을 지정합니다. 기본값(sys.stdout)은 표준출력(모니터)을 의미합니다. 이 속성의 값을 파일포인터<sup>22)</sup>로 설정하면 데이터를 파일에 저장할 수 있습니다.

19) print() 함수 대신에 sys.stdout.write('test') 로 출력이 가능합니다.

20) 파이썬 2.x 버전의 test.py 파일을 3.x 버전의 코드로 변경하길 원한다면 다음 코드처럼 2to3 모듈을 사용하면 됩니다.

```
C:\Python\Tools\Scripts>2to3.py -w test.py
```

21) 대한민국과 일본의 컴퓨터 문자 체계에서는 역슬래시 기호에 원 기호(₩)와 엔 기호(¥)를 각각 배정해 놓고 있기 때문에 이들 기호로 표시될 수 있습니다.

22) 파일포인터는 [9.1. 파일 입출력]에서 설명합니다.

print() 함수는 여러 개 데이터를 공백을 이용해 연결해 출력해 줍니다. sep 속성을 이용하면 데이터를 연결하는 문자를 지정할 수 있습니다.

```
print("Hello", "World")
```

```
Hello World
```

```
print("Hello", "World", sep=",")
```

```
Hello,World
```

end 속성은 데이터를 출력한 후 행의 마지막에 포함될 문자를 지정합니다. 다음코드는 두 개 행을 출력합니다.

```
print("Hello")
print("world")
```

```
Hello
world
```

end 속성을 \t(또는 Wt)로 지정하면 다음 행이 탭 되어 출력됩니다. Wt는 탭 키<sup>23)</sup>를 의미합니다.

```
print("Hello", end="\t")
print("world")
```

```
Hello   world
```

23) 탭 키는 커서를 한꺼번에 여러 칸씩 움직일 수 있도록 만든 것으로, 이 키를 한 번 누를 때마다 보통 8칸씩 오른쪽으로 커서가 움직입니다.

### 3. 기본 자료형

자료형(데이터 타입, data type)은 프로그래밍 언어에서 정수, 실수, 논리 등 여러 종류의 자료(데이터)가 어떤 값을 가질 수 있는지에 대해 알려주는 속성입니다. 자료형으로 인해 변수에 저장할 수 있는 가능한 값, 해당 자료형에서 수행을 마칠 수 있는 명령들, 데이터의 의미, 해당 자료형의 값을 저장하는 방식을 결정할 수 있습니다.

대부분의 프로그래밍 언어들은 명시적으로 자료형의 개념을 포함하고 있으며, 변수를 선언 할 때 자료형을 지정하는 키워드가 존재합니다. 예를 들면 다음과 같습니다.

- 정수(integer)를 지정하는 키워드는 int
- 부동소수점(실수, floating-point)를 지정하는 float 또는 double
- 논리(boolean)을 지정하는 boolean
- 문자 한 개(character)를 지정하는 char
- 문자와 숫자로 이루어진 문자열(string)을 지정하는 string

파이썬도 자료형의 개념을 포함하고 있습니다. 그러나 파이썬은 변수를 선언할 때 자료형을 지정하는 키워드가 존재하지는 않습니다. 파이썬의 기본 자료형은 숫자형, 문자형, 논리형이 있습니다.

#### 3.1. 숫자형

파이썬의 숫자형 타입(Numbers)은 정수(int), 실수(float), 복소수(complex) 형이 있습니다.

- 소수점이 없는 정수는 int 형입니다. 파이썬 3.x 버전에서는 2.x 버전에 있었던 long 형이 없어지고 모든 정수를 int 형으로 인식합니다.
- 소수점이 있는 숫자는 float 형입니다.
- 허수부를 포함하는 복소수는 complex 형입니다. 허수부를 표현하는 문자는 i가 아니고 j입니다.

##### 1) 정수

많은 프로그래밍 언어들은 정수를 크기별로 나누어 다루고 있습니다. 예를 들면 자바 언어는 정수 자료형을 byte, short, int, long으로 나누어 각각 1, 2, 4, 8바이트를 할당해 사용

합니다. 예를 들면 int 자료형은 4바이트 크기를 갖는데 그렇게 하면 int 자료형으로 표현할 수 있는 정수의 범위는  $-2^{31} \sim 2^{31} - 1$  ( $-2,147,483,648 \sim 2,147,483,647$ )입니다. 파이썬 2.x 버전은 정수를 int형과 long형<sup>24)</sup>으로 구분해서 다루었습니다. 그래서 파이썬 2.x 버전에서 int 자료형의 범위는  $-2^{31} \sim 2^{31} - 1$ 입니다. 2.x에서 int 자료형의 범위를 넘어선 크기의 정수는 long형입니다. type()은 자료형을 확인하는 함수입니다.

다음 코드는 파이썬 2.x에서 실행했을 경우의 결과입니다. 2.x에서  $2^{31}$  ( $2,147,483,648$ )은 long형을 알려줍니다.

```
type(2**31)
```

```
<type 'long'>
```

```
sys.maxint
```

```
2147483647
```

파이썬 3.x 버전에서는 long 자료형이 없어졌습니다. 숫자의 크기가 얼마가 되든 int 자료형으로 데이터를 저장합니다.

다음은 3.x 버전에서 자료형을 확인하는 코드입니다.

```
type(10)
```

```
<class 'int'>
```

```
type(2**31)
```

```
<class 'int'>
```

```
type(2**40)
```

```
<class 'int'>
```

24) 파이썬 2.x 버전에서 int 형은 4바이트 크기이며, long 형은 8바이트 크기로 저장했습니다.

다음 코드는 변수에 값을 저장하고 변수를 이용해 자료형을 확인합니다. 변수에 값을 저장하고 자료형 확인해도 변수에 저장되어 있는 값의 자료형을 알 수 있습니다.

```
a = 10
type(a)
```

```
int
```

```
c = 2147483648 # 2**31
print(c)
```

```
2147483648
```

```
type(c)
```

```
int
```

## 2) 실수

실수는 부동소수점이라고도 부릅니다. 소수점 이하를 갖는 수를 의미하며 파이썬에서 실수는 float 형으로 처리합니다.

```
b = 3.5
type(b)
```

```
float
```

파이썬 2.x 버전의 정수와 정수의 연산 결과는 정수 자료형였습니다. 예를 들면 다음 코드에서 보여주는 것처럼 나눗셈 연산  $3/2$ 의 결과는 파이썬 2.x에서는 1입니다.

```
3/2
```

```
1
```

파이썬 3.x 버전은 정수형의 연산 결과는 실수형(float)이 될 수 있습니다. 정수형과 정수형의 나눗셈 연산은 실수형(float)입니다. 나눗셈(/) 연산의 결과는 항상 부동소수점 형식입니다.

```
3/2
```

```
1.5
```

```
type(2/2)
```

```
<class 'float'>
```

### 3) 복소수

복소수는 허수를 갖습니다. 허수는 j문자를 붙여 표현합니다.

```
c = 3 + 2j
type(c)
```

```
complex
```

허수는 제공하면 음수가 되는 수입니다.

```
d = 1j
print(d**2)
```

```
(-1+0j)
```

복소수를 표현할 때 j 문자는 단독으로 사용될 수 없습니다.

```
c = 3 + j
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-24-ae86b6a564e7> in <module>()
----> 1 c = 3 + j

NameError: name 'j' is not defined
```

- 위의 구문은  $c = 3 + 1j$ 처럼 j 앞에 숫자를 붙여야 합니다.



## 4) \_

대화식 모드에서는 마지막으로 인쇄 된 표현식이 변수 \_에 지정 됩니다.

```
a = 10
b = 20
a + b
```

```
30
```

\_ 변수를 이용해 이전 마지막 행인 a + b 결과를 사용할 수 있습니다.

```
c = 100
print(c + _)
```

```
130
```

\* \_ 변수는 숫자의 연산 결과뿐만 저장하는 것은 아닙니다. 마지막으로 인쇄 된 표현식이 변수 \_에 지정 됩니다.

## 3.2. 문자형

파이썬은 단일문자와 문자열을 구분하지 않습니다<sup>25)</sup>.

## 1) 문자형의 표현

파이썬의 문자열은 겹따옴표("와 ") 또는 홑따옴표('와 ') 로 묶어서 사용합니다.

```
name = "JinKyoung"
address = '서울시 강남구'
print(name, address)
```

```
JinKyoung 서울시 강남구
```

25) 자바 언어에서는 문자(char)와 문자열(String)을 다르게 처리합니다. 자바에서 문자는 홑따옴표('와 ') 로 묶어서 표현하고 문자열은 겹따옴표("와 ")로 묶어서 표현합니다.

## 2) 여러 줄 문자 표현

여러 줄의 문자열을 작성하려면 겹따옴표 3개(''') 또는 홑따옴표 3개(''')를 사용합니다.

```
text = '''이렇게 작성하면
줄 바꿈도 그대로 적용해서
여러 줄의 문자를 작성할 수 있습니다.'''
```

```
print(text)
```

이렇게 작성하면  
줄 바꿈도 그대로 적용해서  
여러 줄의 문자를 작성할 수 있습니다.

## 3) 소스코드 줄 바꿈

아래 코드 첫 라인의 맨 마지막에 있는 역슬래시(\ 또는 ₩)는 입력하는 코드를 줄 바꿈하겠다는 의미입니다. 이것은 반드시 필요한 것은 아닙니다. 그러나 소스코드를 줄 바꿈 해서 작성하면 가독성이 높은 코드를 작성할 수 있습니다.

```
text = '''\
이렇게 작성하면
줄 바꿈도 그대로 적용해서
여러 줄의 문자를 작성할 수 있습니다.\
'''
```

```
print(text)
```

이렇게 작성하면  
줄 바꿈도 그대로 적용해서  
여러 줄의 문자를 작성할 수 있습니다.

## 4) 탈출 문자(escape character)

줄 바꿈을 위해 겹따옴표 3개(''') 또는 홑따옴표 3개(''')를 사용할 수 있지만 코드 내에서 줄 바꿈을 위해 주로 사용하는 것은 \n입니다. 역슬래시(\)<sup>26)</sup>는 통화기호(₩)로

26) 대한민국과 일본의 컴퓨터 문자 체계에서는 역슬래시 기호에 원 기호(₩)와 엔 기호(¥)를 각각 배정

표시될 수 있습니다.

```
text = '\\n을 이용하면\n줄 바꿈을 \n적용 시킬 수 있습니다.'
```

```
print(text)
```

```
\n을 이용하면
줄 바꿈을
적용 시킬 수 있습니다.
```

문자열 내에서 특별한 의미를 갖는 문자들은 역슬래시(\)를 이용하여 이스케이프(escape) 문자를 사용합니다. 파이썬의 이스케이프 문자는 다음과 같습니다.

표 1. 탈출 문자

문자	의미
\n	줄 바꿈
\t	탭
\r	리턴(행의 첫 번째 열로 돌아옴)
\0	널(null)
\\	₩ 문자 표시
\'	'(홀따옴표) 문자 표시
\"	“(겹따옴표) 문자 표시

```
print("Hello\nWorld")
```

```
Hello
World
```

```
print("Hello\tWorld")
```

```
Hello  World
```

```
print("Hello\rWorld")
```

```
World
```

```
print("Hello\0World")
```

해 놓고 있기 때문에 이들 기호로 표시될 수 있습니다.

```
HelloWorld
```

```
print("Hello\\World")
```

```
Hello\World
```

```
print("Hello\'World")
```

```
Hello'World
```

```
print("Hello\"World")
```

```
Hello"World
```

## 5) 문자열 연결하기

문자열과 문자열을 +(덧셈)하면 문자열을 연결합니다.

```
"Hello"+"World"
```

```
'HelloWorld'
```

문자열을 공백으로 연결해도 두 문자열을 연결합니다.

```
"Hello" "World"
```

```
'HelloWorld'
```

문자열과 숫자를 \*(곱셈) 연산 하면 문자열을 곱셈함 숫자만큼 반복합니다.

```
2 * "Hello"
```

```
'HelloHello'
```

## 6) 문자열 인덱싱

[*index*] 형식으로 문자열에서 지정한 위치(*index*)의 문자를 뽑아낼 수 있습니다. 파이썬의 문자열은 첫 문자의 인덱스가 0입니다. 음수는 맨 뒤의 문자부터 의미합니다. 아래의 “Python” 문자열에서 -1은 맨 뒤의 문자 ‘n’을 의미하며 이는 양수 인덱스 5의 위치와 같습니다.

```

+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1

```

```
"Python"[0]
```

```
'p'
```

```
"Python"[3]
```

```
'h'
```

인덱스를 벗어나는 참조는 에러를 발생시킵니다.

```
"Python"[6]
```

```

-----
IndexError                                Traceback (most recent call last)
<ipython-input-19-3f68d322bb09> in <module>()
----> 1 "Python"[6]

IndexError: string index out of range

```

[*from\_index*:*to\_index*]를 이용하면 부분 문자열을 빼 낼 수 있습니다. *from\_index* 위치의 문자는 포함하지만 *to\_index* 위치의 문자는 포함하지 않습니다.

```
str_ = 'Python'
```

*from\_index*와 *to\_index*는 값을 지정하지 않을 수 있습니다. `[:m]` 형식이면 처음부터 *m* 인덱스 위치(*m* 위치 포함 안 됨)까지, 그리고 `[n:]` 이면 *n* 위치부터 마지막까지 부분 문자열을 추출해 줍니다.

```
str_[0:6]
str_[:]
str_[0:]
str_[6:]
```

```
'Python'
```

`[:]` 형식에서도 인덱스를 음수로 지정할 수 있습니다.

인덱스에 음수를 사용하면 맨 뒤에서부터 순서를 찾습니다. 이 경우에도 *to* 위치의 항목은 포함되지 않습니다.

```
str_[-6:-1]
```

```
'Pytho'
```

만일 음수를 이용해 맨 마지막 문자까지 빼내고 싶다면 *to\_index*를 포함하지 않으면 됩니다.

```
str_[-6:]
```

```
'Python'
```

*from* 항목을 포함하지 않을 경우 맨 처음 항목이 됩니다. 이 경우는 *to*도 마찬가지입니다.

```
str_[:-1]
```

```
'Pytho'
```

다음처럼 인덱스를 잘 못 사용할 경우 아무것도 출력되지 않을 수 있습니다

```
str_[-1:-5]
```

```
''
```

[*start\_index*:*stop\_index*:*step*] 형식은 매 *step*번째 아이템을 추출해 줍니다. *start\_index*, *stop\_index*, *step* 은 생략될 수 있습니다. 만일 *start\_index*와 *end\_index*가 생략되면 [*::step*] 형식이 됩니다. 이 형식은 문자열뿐만 아니라 리스트, 튜플 등에서도 사용할 수 있습니다.

```

+---+---+---+---+---+---+---+---+---+
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
+---+---+---+---+---+---+---+---+---+
0  1  2  3  4  5  6  7  8  9 10
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1

```

```
'0123456789'[:2]
```

```
'02468'
```

```
'0123456789'[:3]
```

```
'0369'
```

```
'0123456789'[2:8:2]
```

```
'246'
```

*step*이 음수를 가질 수 있습니다. 그렇게 하면 문자열을 역순으로 뽑아낼 수 있습니다. *step*이 역순이면 *start\_index*는 *stop\_index* 보다 더 뒤의 인덱스여야 합니다. 다음 코드는 모두 같은 결과를 출력합니다.

```

'0123456789'[9::-2]
'0123456789'[::-2]
'0123456789'[-1::-2]

```

```
'97531'
```

```
'0123456789'[-1:-5:-2]
```

```
'97'
```

## 7) raw 문자열

문자열 앞에 *r*을 붙여 raw 문자열을 선언해 사용하기도 합니다. *r*을 붙이면 역슬래시 문자

를 해석하지 않고 남겨둡니다. 탈출 문자를 사용하지 않고 역슬래시 등의 문자를 그대로 표현해서 사용할 때 유용합니다. 특히 정규표현식(Regular Expression)<sup>27)</sup>과 같은 곳에 유용합니다.

```
print("\n \t \r \\ \\' \'"")
```

```
\ ' "
```

```
print(r"\n \t \r \\ \\' \'"")
```

```
\n \t \r \\ \\' \'
```

정규표현식을 이용하면 패턴을 이용해 해당 패턴과 일치하는 문자열을 찾거나 위치를 알 수 있습니다. 다음 코드는 문자열에서 전화번호 형식을 갖는 문자열을 찾습니다.

```
import re
data = "이름:홍길동, 주소:서울시, 전화번호:010-2345-6789, 특징:동해번쩍서해번쩍"
phone_pattern = r'[\d]{3,4}-[\d]{4}-[\d]{4}'
phone = re.findall(phone_pattern, data)
print(phone)
```

```
['010-2345-6789']
```

정규표현식에 대해서는 [Regular expression operations<sup>28)</sup>]를 참고하세요.

## 8) 유니코드 처리

파이썬 2.x에서는 한글을 사용하기 위해서는 유니코드임을 알려주기 위해서 u를 문자열 앞에 표기해야 했습니다.

```
type('가')
```

```
<type 'str'>
```

```
type(u'가')
```

27) 프로그래밍에서 사용하는 일종의 형식 언어입니다. 주로 문자열(string) 관련 프로그래밍에 많이 사용되는데, 일정한 규칙을 가진 텍스트 문자열을 찾거나 변경할 경우가 많은데, 이럴 때 정규 표현식을 사용합니다. 정규표현식에 대한 더 자세한 내용은 <https://namu.wiki/w/정규표현식> 을 참고하세요.

28) <https://docs.python.org/3/library/re.html>



```
<type 'unicode'>
```

파이썬 3.x부터는 모든 문자를 유니코드로 처리하기 때문에 별도로 u 문자를 문자열 앞에 붙일 필요 없습니다.

```
type('가')
```

```
<class 'str'>
```

만일 유니코드를 여러분이 원하는 인코딩으로 변경하길 원한다면 encode() 함수를 사용할 수 있습니다. UTF-8은 유니코드를 위한 가변 길이 문자 인코딩 방식 중 하나입니다. UTF-8을 이용한 인코딩 시 한글 1자당 3바이트씩 저장됩니다.

```
type('가'.encode('utf-8'))
```

```
<class 'bytes'>
```

```
print('가')
```

```
가
```

```
print('가'.encode('utf-8'))
```

```
b'\xea\xba\x80'
```

EUC-KR은 한글 완성형 인코딩이며 한글 1자당 2바이트씩 저장됩니다.

```
print('가'.encode('euc-kr'))
```

```
b'\xb0\xa1'
```

### 3.3. 논리형

파이썬의 논리형(Bool)은 True 또는 False 값을 갖습니다. true 또는 TRUE를 논리형 값으로 사용할 수 없습니다.

```
a = True
print(type(a))
```

```
<class 'bool'>
```

```
a = true      # 자바에서는 true와 false를 소문자로 사용 합니다.
```

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

- 실행하는 환경에 따라 출력되는 에러 메시지가 이 책의 에러 메시지와 다를 수 있습니다.

## 1) False인 경우

조건을 비교하기 위해 반드시 논리형만 필요한 것은 아닙니다. 정수형의 경우에는 0, 실수는 0.0, 복소수는 0+0j, 문자열이 널스트링(“ 또는 ”)이면 False로 판단됩니다. 숫자가 0이 아니거나 문자열이 널스트링이 아니면 True로 판단됩니다.

다음은 False로 판단되는 값들입니다.

- None
- False
- 숫자 타입 0에 해당하는 것(예: 0, 0L, 0.0, 0j).
- 빈 문자(예: “, ”).
- 빈 튜플 또는 리스트(예: (), []).
- 빈 딕셔너리(예: {}).

숫자 0으로 판별되는 것은 모두 False입니다.

```
if 0:
    print(True)
else:
    print(False)
```

```
False
```

```
if 0.0:
    print(True)
else:
    print(False)
```

```
False
```

```
if 0+0j:
    print(True)
else:
    print(False)
```

```
False
```

문자열 널스트링(“ 또는 “”)도 False로 판별됩니다.

```
if '':
    print(True)
else:
    print(False)
```

```
False
```

## 2) True인 경우

파이썬에서 True로 판별되는 경우는 False로 판별되는 경우를 제외하고 모든 경우입니다.

```
b = 3
```

```
if b:
    print(True)
```

```
True
```

## 4. 포매팅

문자, 숫자, 날짜 데이터에 형식을 지정하는 것을 포매팅(formatting)이라고 합니다.

### 4.1. 문자열에 형식 지정

print() 함수를 이용해 변수의 값과 문자열을 섞어 출력하는 경우가 많습니다. 예를 들면 다음 코드처럼 name 변수에 '홍길동' age 변수에 20 값이 저장되어 있을 경우...

'홍길동님의 나이는 20세입니다.' 형식으로 출력하길 원한다면 다음 코드처럼 작성할 것입니다.

```
name = "홍길동"
age = 20
print(name, '님의 나이는 ', age, '세입니다.', sep='')
```

홍길동님의 나이는 20세입니다.

### 4.2. format() 함수

format() 함수는 문자열에 출력할 값의 형식을 지정합니다.

```
'{인덱스:정렬방법 자릿수 타입}'.format(변수명)
```

구문에서...

- 인덱스 : format() 함수의 인수 중에서 해당 자리에 출력할 인수의 인덱스입니다.
- 정렬방법 : 정렬할 방법을 지정합니다. < 기호는 왼쪽 정렬, > 기호는 오른쪽 정렬 그리고 ^ 기호는 가운데 정렬해서 출력합니다.
- 자릿수 : 변수의 값을 출력할 최대 자릿수를 지정합니다.
- 타입 : 출력 형식을 지정합니다. 'd'는 10진 정수, 'f'는 실수(부동소수점), 's'는 문자열을 의미합니다. 숫자 형식 'b'는 2진수, 'o'는 8진수, 'x'는 16진수로 출력합니다.

앞의 print 문장을 format() 함수를 이용하면 다음처럼 작성됩니다.

```
print('{}님의 나이는 {}입니다.'.format(name, age))
```

```
홍길동님의 나이는 20세입니다.
```

## 1) 순서 지정

{ }에 출력할 변수의 인덱스를 지정할 수 있습니다. 만일 인덱스를 지정하지 않을 경우 format() 함수의 인수 순서대로 매핑됩니다.

```
a, b, c = 10, 20, 30
```

```
print('출력 : {}, {}, {}'.format(a, b, c))
```

```
출력 : 10, 20, 30
```

```
print('출력 : {0}, {1}, {2}'.format(a, b, c))
```

```
출력 : 10, 20, 30
```

```
print('출력 : {2}, {1}, {0}'.format(a, b, c))
```

```
출력 : 30, 20, 10
```

만일 { }의 수가 모자라면 해당 위치의 변수는 출력되지 않지만 { }의 수가 변수의 수보다 더 많으면 에러가 발생합니다.

```
print('출력값 : {}, {}, {}'.format(a, b))
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-5-fbec15bbeb71> in <module>()
----> 1 print('출력값 : {}, {}, {}'.format(a, b))

IndexError: tuple index out of range
```

## 2) 숫자 출력

숫자에 자릿수를 지정하면 숫자가 지정한 자릿수보다 작을 경우 남은 자리는 공백으로 채워집니다. 숫자의 기본 정렬은 오른쪽 정렬입니다. 숫자가 자릿수보다 더 크면 숫자의 길이만큼 늘어 출력됩니다. 아래의 예제코드에서 [ ]는 출력결과에서 자릿수 이해를 돕기 위해 입력한 것입니다.

```
a = 12345
print('출력: [{:}], [{:10}], [{:3}]'.format(a, a, a))
```

출력: [12345], [        12345], [12345]

{ } 안에 데이터의 타입을 지정할 수 있습니다. 'd'는 10진수 정수, 'f'는 실수(부동소수점), 'o'(8진수, octal), 'x'(16진수, hexadecimal)을 의미합니다. 'f'는 소수점 이하 6자리까지만 출력합니다. 정수는 실수, 2진수, 8진수, 16진수로 출력할 수 있지만 실수는 'f'만 가능합니다.

```
a = 12345
print('출력: [{:d}], [{:f}], [{:b}], [{:o}], [{:x}]'.format(a, a, a, a, a))
```

출력: [12345], [12345.000000], [11000000111001], [30071], [3039]

\* 'f'의 소수점 이하 자릿수 기본값은 6자리입니다.

실수는 정수부와 소수부의 자릿수를 각각 지정할 수 있습니다. 자릿수는 {width.precision} 형식으로 지정합니다. width는 소수점(.)을 포함한 전체 자릿수, precision은 소수점 이하 자릿수를 의미합니다.

```
a = 12345.123456789
print('출력: [{:}], [{:f}], [{:15f}], [{:10.2f}], [{:20.10f}]'\
      .format(a, a, a, a, a))
```

출력: [12345.123456789], [12345.123457], [    12345.123457], [    12345.12], [    12345.1234567890]

\* 위 코드에서 print 문장의 라인 맨 끝에 있는 \ (역슬래시)는 한 라인 코드를 여러 라인으로 작성할 경우 사용하는 줄바꿈 문자입니다. \는 \w로 표시될 수 있습니다.

### 3) 문자열 출력

문자열에 자릿수를 지정하면 문자열의 길이가 자릿수보다 작을 경우 남은 자리는 공백으로 채워집니다. 문자열의 기본 정렬 방법은 왼쪽 정렬입니다. 문자열의 포맷 코드는 's'이지만 입력하지 않는 것이 더 편할 수 있습니다.

```
b = 'Hello World'
print('출력: [{ }], [{:s}], [{:20}], [{:5}]'.format(b, b, b, b))
```

출력: [Hello World], [Hello World], [Hello World           ], [Hello World]

{:width.precision} 형식으로 크기를 지정하면 전체 자릿수와 출력할 문자열의 개수를 지정할 수 있습니다.

```
print('출력: [{ }], [{:.5}], [{:10.5s}]'.format(b, b, b))
```

출력: [Hello World], [Hello], [Hello       ]

\* {:.5}는 문자열의 앞 5자리만 출력합니다.

\* {:10.5}는 전체 10자리로 출력하며 문자열의 처음 5자리를 출력하고 나머지는 공백으로 채웁니다.

### 4) 정렬방법 지정

{ }의 콜론(:)뒤에 '<', '>', '^'을 이용하면 정렬 상태를 지정할 수 있습니다. '<'는 왼쪽정렬, '>'는 오른쪽 정렬 그리고 '^'는 왼쪽 정렬합니다.

```
print('출력: [{:>5d}], [{:<5d}], [{:^5d}]'.format(a, b, c))
```

출력: [   10], [20   ], [ 30 ]

```
print('출력: [{:>10d}], [{:<10d}], [{:^10d}]'.format(a, b, c))
```

출력: [       10], [20       ], [   30   ]

## 5) 공백 대체 문자

자릿수가 더 클 경우 출력되는 공백 대신 채울 문자를 지정할 수 있습니다.

```
a = 10
b = 'Hello'
print("출력: {:$>10}, {:*<20}, {:_<10.5s}".format(a, b, b))
```

출력: \$\$\$\$\$\$\$\$10, Hello World\*\*\*\*\* , Hello\_\_\_\_\_

숫자 타입의 경우 크기 앞에 0(zero)을 붙이면 남은 자리를 0으로 채워줍니다. 음수의 경우에는 맨 앞에 -가 표시되며, 부호(sign) 표시도 전체 자릿수에 포함됩니다.

```
a = 123
b = -123
print("출력: [{:05}], [{:05}]".format(a, b))
```

출력: [00123], [-0123]

‘+’를 붙이면 양수의 경우 숫자 앞에 + 부호를 붙여줍니다.

```
print("출력: [{:5}], [{:+5}], [{:+05}]".format(a, a, a))
```

출력: [ 123], [ +123], [+0123]

‘=’를 붙이면 전체 자릿수만 큼 출력하는 문자열의 맨 앞에 부호를 표시해줍니다.

```
print("출력: [{:=10}], [{:=+10}], [{:=+010}]" .format(a, a, a))
print("출력: [{:=10}], [{:=+10}], [{:=+010}]" .format(b, b, b))
```

출력: [ 123], [+ 123], [+00000123]

출력: [- 123], [- 123], [-00000123]

‘=’는 채울 문자와 같이 사용할 수 있습니다.

```
print("출력: [{:=10}], [{:=+010}], [{:=>+010}]" .format(a, a, a))
```



```
print("출력: [{:=10}], [{:=+010}], [{:=>+010}"].format(b, b, b))
```

```
출력: [*****123], [+*****123], [*****+123]
```

```
출력: [-*****123], [-*****123], [*****-123]
```

## 6) 날짜 출력

날짜를 출력하기 위한 포맷 문자열은 %를 이용합니다. %Y는 연도(Year), %m은 월(Month), %d는 일(Day), %H는 시간(Hour), %M(Minute)은 분, %S는 초(Second)를 의미합니다.

```
from datetime import datetime
print('{:%Y-%m-%d %H:%M}'.format(datetime(2001, 2, 3, 4, 5)))
```

```
2001-02-03 04:05
```

\* 날짜와 시간을 갖는 객체를 만들려면 datetime 모듈을 import 해야 합니다. 모듈과 모듈의 import에 대해서는 6장에서 설명합니다.

## 7) 파라미터를 갖는 format

{ } 안에 {}를 이용하면 format() 함수의 인수를 이용해 포맷 형식을 지정할 수 있습니다.

```
a = 2.7182818284
```

```
print("출력: [{:}{:}.{:}"].format(a, '>', 10, 3))
```

```
출력: [      2.72]
```

format 함수의 인수에 이름을 지정할 수 있습니다. 이 경우 이름을 갖는 인수(키워드 인수)는 함수 인수 목록에서 이름이 없는 인수(위치 인수) 뒤에 와야 합니다.

```
print("출력: [{:}{sign}{:}.{:}"].format(a, '>', 10, 3, sign='+'))
```

```
출력: [      +2.72]
```

## 5. 연산자

### 5.1. 산술 연산자

산술 연산자(Arithmetic operator) 덧셈, 뺄셈, 곱셈, 나눗셈은 수학에서의 연산과 동일하고, 나머지 연산자는 피제수를 젓수로 나눈 나머지 값을 구하는 연산입니다. 나머지 연산의 실제 계산은 피제수에서 젓수를 계속 빼서 피제수의 값이 젓수의 값보다 작아질 때의 피제수의 값이 최종적으로 나머지 연산의 값이 됩니다.

표 2. 산술 연산자

연산자	설명
+	덧셈
-	뺄셈
*	곱셈
/	나눗셈
//	몫
%	나머지
**	제곱

다음 코드는 숫자의 사칙연산 결과를 출력합니다.

```
a=8
b=3
print(a+b)
```

11

```
print(a-b)
```

5

```
print(a*b)
```

24

```
print(a/b)
```

```
2.6666666666666665
```

2.x 버전의 정수와 정수의 연산 결과는 정수 자료형였습니다. 예를 들면 다음 코드에서 보여주는 것처럼 나눗셈 연산 3/2의 결과는 파이썬 2.x에서는 1입니다.

```
3/2
```

```
1      # Python 2.x 버전에서 실행한 결과
```

3.x 버전은 정수형의 연산 결과는 실수형(float)이 될 수 있습니다. 정수형과 정수형의 나눗셈 연산은 실수형(float)입니다.

```
3/2
```

```
1.5
```

```
type(2/2)
```

```
<class 'float'>
```

몫은 '/' 연산자를 이용합니다. 다음 코드는 8/3의 결과 2.6666666666666665 에서 몫 2 만 출력된 것을 확인할 수 있습니다.

```
a=8
b=3
print(a//b)
```

```
2
```

나머지는 '%' 연산자를 이용합니다.

```
print(a%b)
```

```
2
```

제곱 결과를 얻으려면 ‘\*\*’를 이용합니다. 다음 구문은  $3^8$ 을 출력합니다.

```
print(a**b)
```

```
512
```

\* 자바 또는 C언어에서 사용하던 증가연산자(++), 감소연산자(--)는 사용할 수 없습니다.

```
print(a++)
```

```
File "<stdin>", line 1
  print(a++)
        ^
SyntaxError: invalid syntax
```

```
print(++a)    # 이것은 증가연산자가 아니고 부호 연산자 +가 두 개입니다.
```

```
8
```

```
print(a--)
```

```
File "<stdin>", line 1
  print(a--)
        ^
SyntaxError: invalid syntax
```

```
print(-a)
```

```
-8
```

```
print(--a)    # 이것은 감소연산자가 아니고 부호 연산자 -가 두 개입니다.
```

```
8
```

## 5.2. 논리 연산자

파이썬의 논리연산자는 &, |, and, or, not이 있습니다.

표 3. 논리 연산자

연산자	설명
&	AND
	OR
and	AND(short circuit)
or	OR(short circuit)
not	NOT

논리연산자(Logical operator)들 중에서 &와 and 연산자는 양쪽 항의 값이 모두 True 인 경우에만 True를 반환합니다. |와 or 연산자는 양쪽 항 중에서 어느 한쪽만 True 이면 True를 반환합니다. 그런데 and와 or 연산자는 왼쪽항의 결과에 따라서 오른쪽 항을 수행하지 않을 수 있습니다.<sup>29)</sup>

‘and’ 연산자는 AND 연산을 수행하여 양쪽 항이 모두 true일 때만 true를 반환합니다. 만일 왼쪽 항이 false일 때는 오른쪽 항을 수행하지 않고 무조건 false를 반환합니다.

‘or’ 연산자는 OR 연산을 수행하여 양쪽 중 한쪽만 true를 만족해도 true를 반환합니다. 만일 왼쪽 항이 true이면 오른쪽 항을 수행하지 않고 무조건 true를 반환합니다.

자바 또는 C 언어에서 보던 ! 부호를 논리 반전(not) 연산자로 사용할 수 없습니다.

```
a = False
if not a:
    print(True)
```

```
True
```

```
if !a:           #!는 not 연산자가 아니다.
    print(True)
```

```
File "<stdin>". line 1
    if !a:
      ^
SyntaxError: invalid syntax
```

29) 그래서 and와 or를 short circuit operator라고 부릅니다.

### 5.3. 비교 연산자

파이썬의 비교연산자들은 크기를 비교해 결과를 True 또는 False로 반환합니다.

표 4. 비교 연산자

연산자	설명
$a < b$	b가 a보다 크면 true
$a \leq b$	b가 a보다 크거나 같으면 true
$a > b$	a가 b보다 크면 true
$a \geq b$	a가 b보다 크거나 같으면 true
$a == b$	a와 b가 같으면 true(조건문에서 유용)
$a != b$	a와 b가 같지 않으면 true

비교연산자를 이용하면 크기를 비교할 수 있습니다. 결과는 True 또는 False입니다.

```
3 == 3
```

```
True
```

```
3 != 3
```

```
False
```

```
3 < 3
```

```
False
```

```
3 > 3
```

```
False
```

```
3 <= 3
```

```
True
```

```
3 >= 3
```

```
True
```

크기 비교는 문자열 타입도 가능합니다. 소문자가 대문자보다 큰 값입니다.

```
'Hello' > 'World'
```

```
False
```

```
'Hello' == 'Hello'
```

```
True
```

```
'Hello' == 'hello'
```

```
False
```

```
'a' > 'A'
```

```
True
```

논리 타입도 크기 비교가 가능합니다. True가 False 보다 큽니다.

```
True > False
```

```
True
```

```
True < False
```

```
False
```

비교 연산자는 if 구문 또는 while 구문<sup>30)</sup> 등에서 사용해서 프로그램의 논리적 흐름을 결정 하는데 사용됩니다. 예를 들면 점수에 따라 합격 또는 불합격 여부를 판단해야 될 경우 비교 연산자가 사용됩니다.

```
jumsu = 80
if(jumsu > 60):
    print("합격입니다.")
else:
    print("불합격입니다.")
```

```
합격입니다.
```

30) if, while 구문은 4장 제어문에서 설명합니다.

```
jumsu = 50
if(jumsu > 60):
    print("합격입니다.")
else:
    print("불합격입니다.")
```

불합격입니다.

## 5.4. 비트 연산

비트 연산자들은 숫자를 2진수로 변환하여 연산합니다.

표 5. 비트 연산자

연산자	설명
$a \& b$	AND 연산, 두 비트가 모두 1이면 1
$a   b$	OR 연산, 두 비트중 하나 이상 1이면 1
$a \wedge b$	XOR 연산, 두 비트가 같으면 0, 다르면 1
$\sim a$	NOT 연산, 0을 1로, 1을 0으로 변환
$a \gg n$	Shift 연산, a를 n비트만큼 오른쪽으로 이동, $2^n$ 으로 나눈 결과와 같음
$a \ll n$	Shift 연산, a를 n비트만큼 왼쪽으로 이동, $2^n$ 을 곱한 결과와 같음

비교연산자는 숫자들을 2진수 비트로 변환하여 연산하는 연산자들입니다. 다음 표는 비트연산자들의 연산 결과를 나타낸 것입니다.

표 6. 비트 AND, OR, XOR, NOT 연산자

X	Y	$\sim X$	$\sim Y$	$X \& Y$	$X   Y$	$X \wedge Y$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0



## 1) & 연산자

&(AND)는 두 비트가 모두 1일 1입니다. 두 비트 중 하나이상 0이면 결과는 0입니다.

$$\begin{array}{r}
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline \end{array} \\
 \& \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline \end{array} \\
 \hline
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}
 \end{array}$$

그림 1. 비트 AND 연산

다음은 11(0b00001011)과 6(0b00000110)을 & 연산한 결과입니다.

```
print(bin(11), bin(6))
```

```
0b1011 0b110
```

```
0b00001011 & 0b00000110
```

```
2
```

& 연산자는 특정 비트를 지우기 위해 사용됩니다. & 연산자는 지우고 싶은 자리에 0, 남기고 싶은 자리에 1을 이용해서 & 연산을 하면 됩니다. 예를 들면 숫자 11에서 홀수 번째 비트를 지우고 싶다면 다음과 같이 0b10101010으로 & 연산을 하면 됩니다.

```
0b00001011 & 0b10101010
```

```
10
```

이렇게 특정 비트를 & 연산으로 삭제하는 것을 마스크(mask) 연산 또는 마스크링(masking)이라고 부릅니다. 이때 연산에 사용되는 비트를 마스크비트라고 부릅니다. 마스크하기 위한 비트를 2진수로 지정하는 것은 매우 번거로울 수 있습니다. 2진수 4자리는 16진수 1자리로 표현이 가능합니다. 16진수는 0부터 15까지 값을 갖는데 0부터 9까지는 10진수의 수를 그대로 사용하며, 11부터 15까지는 각각 A부터 F까지 알파벳으로 표현합니다.

앞의 코드에서 마스크 비트 0b10101010은 0xAA로 표현할 수 있습니다.

```
0b00001011 & 0xAA
```

```
10
```

만일 숫자 2,147,483,636에서 가장 오른쪽 8비트 값을 알기 위해서는 다음과 같이 마스크 연산을 할 수 있습니다.

2147483636 & 0x000000FF

244

```
print(bin(2147483636), bin(244))
```

```
0b111111111111111111111111111111110100 0b11110100
```

파이썬 3.x 버전은 모든 정수를 int형으로 다룹니다. 그러므로 32비트 64비트 정수의 구분이 없습니다. 만일 여러분이 정수를 32비트 크기로 출력하길 원한다면 `format()` 함수와 `&` 연산자를 이용해서 나타낼 수 있습니다.

```
print('{:032b}'.format(11 & 0xFFFFFFFF))
```

[illegible]

```
print('{:032b}'.format(-12 & 0xFFFFFFFF))
```

```
11111111111111111111111111111111110100
```

## 2) | 연산자

1(OR)는 두 비트 중 하나 이상이 1일 1입니다. 두 비트 모두 0 일 때만 결과는 0입니다.

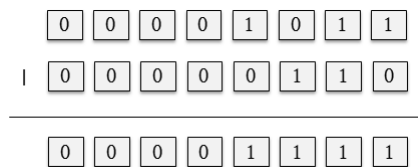


그림 2. 비트 OR 연산

다음은 11(0b00001011)과 6(0b00000110)을 | 연산한 결과입니다.

```
print(bin(11), bin(6))
```

```
0b1011 0b110
```

```
0b00001011 | 0b00000110
```

```
15
```

### 3) ^ 연산자

^(XOR)는 두 비트 값이 다를 때 1입니다. 두 비트가 같은 값(모두 1이거나 모두 0)일 때 결과는 0입니다.

	0	0	0	0	1	0	1	1
^	0	0	0	0	0	1	1	0
	<hr style="border: 0; border-top: 1px solid black; margin: 5px 0;"/>							
	0	0	0	0	1	1	0	1

그림 3. 비트 XOR 연산

다음은 11(0b00001011)과 6(0b00000110)을 ^ 연산한 결과입니다.

```
print(bin(11), bin(6))
```

```
0b1011 0b110
```

```
0b00001011 ^ 0b00000110
```

```
13
```

### 4) ~ 연산자

~(NOT) 연산자는 1은 0으로, 0은 1로 바꿉니다. 이것은 2진수의 1의 보수<sup>31)</sup>를 만드는 것과 같습니다.

31) 보수는 더해서  $n$ 이 되도록 하는 값을 말합니다.  $n$ 진법에서  $n$ 의 보수와  $n-1$ 의 보수가 있습니다. 10진법을 사용한다면 10의 보수와 9의 보수가 있습니다. 2진수는 2의 보수와 1의 보수가 있습니다. 보수는 뺄셈 연산을 할 때 주로 사용합니다.

다음은 숫자 11에 ~ 연산자를 했을 경우입니다.

~11

12

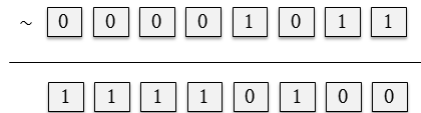


그림 4. 비트 NOT 연산

다음 코드는 실제로 비트가 어떻게 바뀌는 지 확인해 보기위해서 format() 함수를 사용해 32비트만 출력해 봅니다.

```
print('{:032b}'.format(11 & 0xFFFFFFFF))
```

```
000000000000000000000000000000001011
```

```
print('{:032b}'.format(~11 & 0xFFFFFFFF))
```

```
1111111111111111111111111111110100
```

## 5) >> 와 << 비트 이동 연산자

>>는 모든 비트들을 오른쪽으로 이동시킵니다. 왼쪽에 채워지는 비트는 부호비트입니다. 즉, 부호를 그대로 유지한 상태에서 오른쪽으로 비트를 이동시킵니다. X >> n의 경우 X를  $2^n$ 으로 나누는 결과와 같습니다.

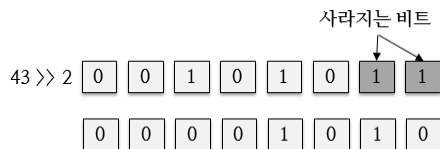
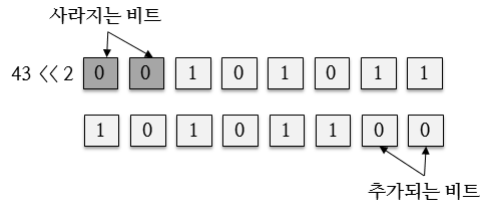


그림 5. 비트 이동 연산자(>>)

```
43 >> 2
```



$\ll$ 는 모든 비트들을 왼쪽으로 이동시킵니다. 오른쪽에 채워지는 비트는 0입니다.  $X \ll n$ 의 경우  $X$ 에  $2^n$ 을 곱하는 결과와 같습니다. 아래 그림에서 왼쪽 비트는 쉬프트 연산자에 의해 사라지는 부분입니다. 이것은 전체 비트가 8비트라고 가정한 것입니다. 실제로는 가장 왼쪽의 비트 두 개가 사라집니다.

그림 6. 비트 이동 연산자( $\ll$ )

```
43 << 2
```

```
172
```

```
print('{:08b}'.format(43 & 0xFF))
```

```
00101011
```

```
print('{:08b}'.format(43 << 2 & 0xFF))
```

```
10101100
```

```
256 << 3
```

```
2048
```

```
print('{:032b}'.format(256 & 0xFFFFFFFF))
```

```
00000000000000000000000000000000
```

```
print('{:032b}'.format(256 << 3 & 0xFFFFFFFF))
```

```
00000000000000000000000000000000
```

```
-256 << 3
```

```
-2048
```

```
print('{:032b}'.format(-256 << 3 & 0xFFFFFFFF))
```

```
1111111111111111111111110000000000
```

## 5.5. isinstance

`instanceof(data, type)` 함수는 데이터의 타입을 확인하는 함수입니다. 이 함수를 이용하면 입력한 값의 타입을 비교해서 이에 맞는 처리를 할 수 있습니다.

```
isinstance(10, int)
```

```
True
```

```
isinstance(3.5, float)
```

```
True
```

```
isinstance(2+3j, complex)
```

```
True
```

```
isinstance('hello', str)
```

```
True
```

```
isinstance(r"hello", str)
```

```
True
```

```
isinstance(True, bool)
```

```
True
```

## 6. 문자열 다루기

파이썬의 변수에 문자열을 할당할 수 있습니다. 문자열 할당은 겹따옴표("와 ") 또는 홑따옴표('와 ')를 이용할 수 있습니다.

문자열을 다루는 여러 방법들이 있습니다.

표 7. 문자열 다루기

방법	설명
" " 또는 ' '	문자열을 만듭니다.
+	문자열을 연결합니다.
len("문자열")	문자열의 길이를 반환합니다.
[from : to]	문자열을 from 위치부터 to 위치까지 자릅니다. to는 포함 안 됩니다.
[from : to : step]	문자열을 from 위치부터 to 위치 step 마다 반환합니다. to는 포함 안 됩니다.
split('delimiter')	문자열을 delimiter로 잘라 리스트로 반환합니다.
'delimiter'.join(["str", ...])	문자열 리스트를 delimiter로 연결합니다.
capitalize()	첫 문자를 대문자로, 나머지 문자를 소문자로 바꿔줍니다.
upper, lower()	문자열을 모두 대문자(upper) 또는 소문자(lower)로 바꿉니다.
startswith(), endswith()	특정 문자로 시작하는지와 끝나는지를 식별해서 논리(True/False)값을 반환합니다.
find(), index()	특정 문자의 인덱스를 반환합니다.
isalnum(), isalpha(), isnumeric(), isdecimal()	이 문자열이 숫자인지, 문자인지 판별해 줍니다.
replace(old, new)	old 문자를 new 문자로 치환합니다.

print 함수를 이용하지 않고 변수를 직접 출력하게 되면 문자열의 경우 ' ' 표시를 해줘 문자열임을 알 수 있게 해 줍니다. print 함수를 이용하면 ' ' 없이 할당되어있는 값만 출력해 줍니다.

```
temp = 'python is easy'
temp
```

```
'python is easy'
```

```
print(temp)
```

```
python is easy
```



문자열의 길이는 len() 함수를 이용합니다.

```
temp = 'python is easy'
len(temp)
```

```
14
```

문자열을 + 연산자로 더하면 문자열을 연결해 줍니다.

```
'python'+ 'is easy'
```

```
'pythonis easy'
```

```
'python ' + 'is easy'
```

```
'python is easy'
```

print() 함수는 인수들을 공백으로 연결해 출력해 줍니다.

```
print('python' , 'is easy')
```

```
python is easy
```

변수가 문자열을 저장하고 있을 경우 문자열 변수를 + 로 더할 수 있습니다.

```
a='python '
b='is easy'
a + b
```

```
'python is easy'
```

[ ] 기호와 문자의 위치를 지정하면 특정 위치의 문자를 뽑을 수 있습니다. 문자열에서 첫 문자의 위치는 0입니다.

```
temp = 'python is easy'
temp[3]
```

```
'h'
```

[*from\_index*:*to\_index*]를 이용하면 부분 문자열을 빼 낼 수 있습니다. *from\_index* 위치의 문자는 포함하지만 *to\_index* 위치의 문자는 포함하지 않습니다. [ : ] 안에 값을 지정하지 않을 수 있습니다. [:*m*] 형식이면 처음부터 *m* 인덱스 위치(*m* 위치 포함 안 됨)까지, 그리고 [*n*:] 이면 *n* 위치부터 마지막까지 부분 문자열을 추출해 줍니다.

```
temp = 'python is easy'
temp[0:6]
temp[:]
```

```
'python is easy'
```

```
temp[6:]
```

```
' is easy'
```

```
temp[:6]
```

```
'python'
```

문자열을 나누기 위해 `split()` 함수를 사용할 수 있습니다. `split()` 함수는 문자열을 나누기 위한 구분자를 지정할 수 있습니다. 문자열 구분을 위한 기본 구분자는 공백입니다. `split()` 함수의 결과는 문자열들을 갖는 배열입니다.

```
temp='python is easy'
temp.split()
```

```
['python', 'is', 'easy']
```

```
temp2 = 'python,java,R,SAS'
temp2.split(',')
```

```
['python', 'java', 'R', 'SAS']
```

`join()` 함수는 배열의 문자열들을 연결해 줍니다.

```
temp3 = ['python', 'java', 'R', 'SAS']
','.join(temp3)
```

```
'python,java,R,SAS'
```

대/소문자를 변환할 수 도 있습니다.

```
"helloworld".capitalize()
```

```
'Helloworld'
```

```
"helloworld".upper()
```

```
'HELLOWORLD'
```

```
"HelloWorld".lower()
```

```
'helloworld'
```

find와 index는 특정 문자의 위치를 찾을 수 있습니다.

```
"helloworld".find('w')
```

```
5
```

```
"helloworld".find('o')
```

```
4
```

```
"helloworld".find('o', 5)
```

```
6
```

```
"helloworld".index('o', 5)
```

```
6
```

문자열의 내용이 숫자 또는 알파벳 등으로 이루어졌는지 확인할 수 있습니다.

```
"1234567890".isnumeric()
```

```
True
```

```
"helloworld".isalpha()
```

```
True
```

replace()는 부분 문자열을 다른 문자열로 대체합니다.

```
"helloworld".replace("hello", "nice")
```

```
'niceworld'
```

dir(str) 은 문자열을 다룰 수 있는 함수들의 목록을 출력합니다.

```
dir(str)
```

```
['_add_', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattr__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum',
'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

## 7. 날짜 다루기

날짜와 시간을 얻기 위해서는 `datetime` 패키지<sup>32)</sup>의 `date`, `time`, `datetime` 클래스들을 사용합니다.

```
datetime.date(year, month, day)
```

```
datetime.time(hour, minute, second, microsecond, tzinfo)
```

```
datetime.datetime(year, month, day, hour, minute, second)
```

다음 표는 날짜 관련 용어들에 대한 설명입니다.

표 8. 날짜 관련 용어

용어	설명
타임스탬프(timestamp)	컴퓨터에서 시간을 측정하는 방법으로 1970년 1월 1일 자정 이후로 초단위로 측정한 절대 시간임. 이렇게 절대적인 시간을 용도에 맞도록 변환해서 사용.
협정세계시(UTC, Universal Time Coordinated)	1972년 부터 시행된 국제 표준시. 세슘원자의 진동수에 의거한 초의 길이가 기준이 됨
그리니치 평균시(GMT, Greenwich Mean Time)	런던 그리니치 천문대의 자오선상에서의 평균 태양시. 1972년부터 협정 세계시를 사용하지만, 동일한 표현으로 널리 쓰임.
지방표준시(LST, Local Standard Time)	UTC를 기준으로 경도 15도마다 1시간 차이가 발생하는 시간임. 한국은 동경 135도를 기준으로 UTC보다 9시간 빠름.
일광절약 시간제(Daylight Saving Time, DST)	흔히 서머타임으로 알고 있는 것으로 에너지 절약을 목적으로 시간을 한 시간씩 앞당기거나 뒤로 미루는 제도임.

다음 코드는 `datetime.date`를 이용해 날짜를 지정하거나 출력하는 예입니다.

```
from datetime import date
today = date(2015, 11, 26)
today2 = date.today()
```

```
print(today2.year)
```

32) 패키지는 미리 만들어진 모듈들의 모음입니다. 모듈들은 미리 정의된 함수들의 모음입니다. 파이썬은 내장된 패키지더라도 사용할 때에는 `import` 해야 사용할 수 있는 패키지들이 있습니다. 모듈과 패키지에 대한 자세한 내용은 5장에서 설명합니다.

```
2018
```

```
print(today2.month)
```

```
5
```

```
print(today2.day)
```

```
31
```

```
print(today2)
```

```
2018-05-31
```

datetime.time은 시간을 지정하거나 알 수 있습니다.

```
from datetime import time  
t = time(7)  
print(t)
```

```
07:00:00
```

```
t2 = time(8, 14, 20, 3000)  
print(t2)
```

```
08:14:20.003000
```

```
t3 = time(hour=3, second=12)  
print(t3)
```

```
03:00:12
```

datetime.datetime을 날짜와 시간을 지정하거나 알 수 있습니다.

```
from datetime import datetime, date, time  
dt = datetime.now()  
print(dt)
```

```
2018-05-31 12:30:20.292817
```

```
dt2 = datetime(2015, 11, 26, 16, 24, 15)  
print(dt2)
```

```
2015-11-26 16:24:15
```

```
dt3 = datetime.utcnow()  
print(dt3)
```

```
2018-05-31 03:30:21.468884
```

```
d = date(2015,12,25)  
t = time(15,20,15)  
dt4 = datetime.combine(d,t)  
print(dt4)
```

```
2015-12-25 15:20:15
```

## 8. 연습문제

---

1) 다음 파이썬 표현 중에서 정상적으로 화면에 값이 출력되는 것을 고르세요.

```
print("I love 'you'")
```

```
print("I like you")
```

```
print('Korea')
```

```
print('Ko"r"ea')
```

```
print{Hello}
```

```
print[Hello]
```

2) 두 수를 입력받아 두 수의 덧셈, 뺄셈, 곱셈, 나눗셈, 몫, 나머지를 출력하세요.



정답 및 풀이

1)

```
print("I love 'you'")
```

```
I love 'you'
```

```
print("I like you")
```

```
I like you
```

```
print('Korea')
```

```
Korea
```

```
print('Ko"r"ea')
```

```
Ko"r"ea
```

```
print{Hello}
```

```
File "<stdin>", line 1
    print{Hello}
        ^
SyntaxError: invalid syntax
```

```
print[Hello]
```

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

2)

```
num1 = int(input("첫 번째 수를 입력하세요."))
num2 = int(input("두 번째 수를 입력하세요."))
```

```
print(num1 + num2)
print(num1 - num2)
print(num1 * num2)
print(num1 / num2)
print(num1 // num2)
print(num1 % num2)
```

```
첫 번째 수를 입력하세요.10
두 번째 수를 입력하세요.2
12
8
20
5.0
5
0
```

```
첫 번째 수를 입력하세요.23
두 번째 수를 입력하세요.4
27
19
92
5.75
5
3
```

## 3장. 데이터 구조

파이썬의 데이터 구조(집합 데이터형)는 리스트(list), 튜플(tuple), 딕셔너리(dict, 사전), 그리고 중복을 허락하지 않는 집합(set)이 있다. 리스트와 딕셔너리는 내부의 값을 나중에 바꿀 수 있지만 튜플은 한 번 값을 정하면 내부의 값을 바꿀 수 없습니다. 그리고 많은 객체 지향 언어들처럼 사용자가 새롭게 자신의 형을 정의하는 것도 가능합니다. 파이썬은 동적 언어이기 때문에, 값이 타입을 가지고 있고, 변수는 모두 값의 참조(reference)입니다.

## 1. 리스트

리스트(list)는 어떤 종류의 값도 담을 수 있습니다. 리스트를 만들려면 대괄호('[ '와 ' ]')를 이용합니다. 리스트는 인덱스를 이용해 읽기와 쓰기를 지원하고, 본 데이터셋을 뽑아내는 슬라이싱(slicing)을 지원합니다. 파이썬의 인덱스는 0부터 시작합니다.

표 1. 리스트 다루기

방법	설명
<code>listData = [ ]</code>	리스트를 만들어 줍니다.
<code>len(listData)</code>	리스트의 항목의 수를 반환합니다.
<code>min(listData), max(listData)</code>	리스트에서 가장 작은(min) 항목과 가장 큰(max) 항목을 반환합니다.
<code>listData[from:to]</code>	리스트의 from 위치부터 to 위치까지 부분 데이터를 추출합니다.(to 위치의 항목은 포함 안 됩니다.)
<code>listData.append(value)</code>	list에 value를 추가합니다.
<code>listData.clear()</code>	list의 모든 항목을 삭제합니다.
<code>listData.count(value)</code>	리스트에서 value의 개수를 반환합니다.
<code>listData.extend(newList)</code>	list에 newList를 추가합니다.
<code>+</code>	두 리스트를 연결합니다. 이 경우 연결된 리스트를 변수에 저장해야 합니다.
<code>listData.index(value, position=0)</code>	position위치 이후에서 value의 값이 있는 인덱스를 반환합니다.
<code>listData.insert(index, value)</code>	list의 index위치에 value를 삽입합니다.
<code>listData.remove(value)</code>	리스트에서 해당 값을 삭제합니다.
<code>del listData[index]</code>	리스트에서 인덱스를 이용해 항목을 삭제합니다. del() 함수 형으로 사용할 수 있습니다.
<code>listData.pop()</code>	리스트에서 가장 마지막 항목을 반환하고 삭제합니다.
<code>listData.reverse()</code>	리스트의 항목들의 순서를 반대로 합니다.
<code>listData.sort(reverse=False)</code>	리스트의 항목들을 정렬합니다. reverse 속성을 True로 하면 내림차순으로 정렬합니다.

## 1.1. 리스트 만들기

리스트는 `[]`를 이용해 만듭니다. `[]`안에 `[]`를 넣으면 차원이 증가합니다.

- 변수명 = `[ ... ]` : 1차원 리스트
- 변수명 = `[ [ ... ], ... ]` : 2차원 리스트
- 변수명 = `[ [ [ ... ], ... ], ... ]` : 3차원 리스트

### 1) 1차원 리스트

다음 코드는 리스트를 다루는 예입니다. 리스트는 `[]`를 이용합니다. 이 예는 문자열을 이용해 리스트를 만들었습니다.

```
fruits = ["banana", "apple", "orange", "grape"]
print(fruits)
```

```
['banana', 'apple', 'orange', 'grape']
```

리스트에 문자열뿐만 아니라 숫자도 담을 수 있습니다.

```
numbers = [1,2,3,4,5]
print(numbers)
```

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

이 예제에서 문자열을 이용해 리스트의 여러 기능을 테스트 할 수 있습니다. 그러면 아마도 더 그럴싸해 보일 수 있습니다. 그런데 여러분이 더 적은양의 타자 입력으로 보다 쉽게 기능을 사용해 볼 수 있도록 이 책에서는 숫자를 이용하겠습니다.

### 2) 다차원 리스트

리스트를 만들기 위해 `[]`를 한 쌍만 이용하면 1차원 리스트 구조를 갖습니다. 이것은 데이터를 한 줄로 나열한 것이라고 생각하면 됩니다. 그런데 `[]` 안에 `[]`를 이용해 리스트가 다른 리스트를 갖게 할 수 있습니다. 그러면 이 리스트는 하나의 차원이 더 생겨 2차원 구

조가 됩니다. 예를 들면 숫자를 여러 줄로 나열한 것이라고 생각하면 됩니다.

```
numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
print(numbers_2d)
```

```
[[1, 2, 3, 4, 5], [10, 20, 30, 40, 50], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
```

numbers\_2d 변수를 표 형식으로 표현하면 다음 구조로 되어 있는 2차원 리스트입니다.

표 2. 2차원 리스트의 구조

행 \ 열	0	1	2	3	4
0	1	2	3	4	5
1	10	20	30	40	50
2	1	3	5	7	9
3	2	4	6	8	10

리스트는 2차원뿐만 아니라 3차원 이상도 가능합니다. 리스트 안에 리스트를 더 넣으면 n 차원 리스트도 가능합니다. 그러나 3차원 이상 리스트가 자주 사용될 일은 없습니다. 2차원 리스트 정도만 알아도 대부분의 문제를 해결할 수 있습니다.

리스트 안의 부분 리스트들은 항목의 수가 같을 필요는 없습니다.

```
numbers_2d_v = [[1,2,3], [10,20,30,40], [1,3,5,7,9], [2,4,6,8,10,12]]
print(numbers_2d_v)
```

```
[[1, 2, 3], [10, 20, 30, 40], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10, 12]]
```

다음 표는 위의 리스트 데이터 구조를 표현한 것입니다. 모든 행이 열의 수가 같이 없을 수 있습니다.

표 3. 길이가 다른 2차원 리스트의 구조

행 \ 열	0	1	2	3	4	5
0	1	2	3			
1	10	20	30	40		
2	1	3	5	7	9	
3	2	4	6	8	10	12

## 1.2. 기본 정보 조회

- `len()` : 길이(항목의 수)
- `min()`, `max()` : 최댓값, 최솟값

실습을 위해 다음 데이터를 이용합니다.

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

```
numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

### 1) 항목 수

`len()` 함수는 리스트의 항목 수를 반환합니다. `len()` 함수는 리스트의 함수가 아닙니다. 그러므로 리스트 객체를 함수의 인수에 전달해야 합니다.

```
len(numbers)
```

```
5
```

2차원 리스트는 행의 수를 출력합니다.

```
len(numbers_2d)
```

```
4
```

2차원 리스트에서 행별 항목의 수는 `[ ]`를 이용합니다.

```
len(numbers_2d[0])
```

```
5
```

### 2) 최솟값과 최댓값

`min()` 함수는 리스트에서 가장 작은 항목을 반환하며, `max()` 함수는 가장 큰 항목을 반환합니다.

```
min(numbers)
```

```
1
```

```
max(numbers)
```

```
5
```

2차원 리스트에서 `max()` 함수는 부분 리스트들 중에서 첫 항목이 가장 큰 값을 갖는 리스트를 반환합니다. 첫 항목의 크기가 같으면 다음 항목을 비교합니다.

```
max(numbers_2d)
```

```
[10, 20, 30, 40, 50]
```

### 1.3. 항목 추가하기

- `+` : 두 리스트를 연결
- `*` : 리스트를 곱한 수만큼 반복
- `append()` : 항목을 맨 뒤에 추가
- `extend()` : 리스트를 항목별로 맨 뒤에 추가
- `insert()` : 리스트를 지정한 인덱스 위치에 삽입

실습을 위해 다음 데이터를 이용합니다.

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

```
numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

#### 1) `+`에 의한 연결

덧셈(`+`) 연산자는 두 리스트를 연결합니다.

```
numbers + numbers
```

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



두 리스트를 + 연산자로 합칠 수 있습니다. 그러나 이 경우에는 결과를 받아 저장하는 할당문이 있어야 합니다. 아래 구문처럼 할당문이 없을 경우에는 두 리스트를 더한 결과를 출력할 뿐 결과는 어디에도 저장되지 않습니다.

```
numbers
```

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

아래 구문처럼 +한 결과를 변수에 저장하도록 할당하는 구문입니다.

```
new_number = number + number
```

```
new_number
```

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

## 2) \*에 의한 반복

곱셈(\*) 연산자는 곱한 수만큼 리스트를 반복합니다. 리스트에 곱하는 값은 정수(int)여야 합니다.

```
3*numbers
```

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

곱셈(\*) 연산자도 반복한 결과를 저장하도록 할당해야 다시 사용할 수 있습니다.

## 3) 맨 뒤에 항목 추가하기

append() 함수는 리스트의 맨 뒤에 새로운 항목을 추가합니다.

```
numbers.append(10)
numbers
```

```
[1, 2, 3, 4, 5, 10]
```

`append()` 함수로 리스트에 리스트를 추가할 수 있습니다. 그러나 이 경우 리스트 안에 새로운 리스트가 추가되어 차원이 바뀔 수 있습니다.

```
numbers.append([20, 30, 40, 50])
numbers
```

```
[1, 2, 3, 4, 5, 10, [20, 30, 40, 50]]
```

#### 4) 리스트로 항목 추가하기

리스트에 리스트를 항목별로 추가하려면 `extend()` 함수를 이용해야 합니다.

```
numbers = [1,2,3,4,5]
numbers.extend([10,20,30,40,50])
numbers
```

```
[1, 2, 3, 4, 5, 10, 20, 30, 40, 50]
```

`extend()` 함수를 이용할 때 문자열을 이용하면 문자열이 리스트로 바뀌고 문자 하나씩 리스트에 추가됩니다.

```
numbers = [1,2,3,4,5]
numbers.extend('hello')
numbers
```

```
[1, 2, 3, 4, 5, 'h', 'e', 'l', 'l', 'o']
```

#### 5) 중간에 항목 삽입하기

`insert(index, data)`는 `index` 위치에 `data` 항목을 삽입합니다.

```
numbers.insert(5, 100)
numbers
```

```
[1, 2, 3, 4, 5, 100, 10, 20, 30, 40, 50]
```

## 1.4. 인덱싱

- `count()` : 리스트에서 데이터의 개수를 반환
- `index()`
- `[index]`

### 1) 데이터 수 세기

`count()` 함수는 리스트에서 데이터의 개수를 셉니다.

```
numbers = [1,3,5,7,9,1,2,3,4,5]
numbers.count(1)
```

2

- 파이썬에서 인덱스는 0부터 시작합니다. 그러므로 맨 처음 항목의 인덱스는 0입니다.

### 2) 항목의 위치 반환하기

`index()` 함수는 해당 항목의 위치를 반환합니다.

```
numbers = [1,3,5,7,9]
numbers.index(5)
```

2

`index()` 함수의 두 번째 인수에 인덱스를 추가하면 해당 위치 이후에서 찾습니다. 다음 코드는 데이터 5를 인덱스 3위치 이후에서 찾습니다.

```
numbers = [1,3,5,7,9,1,2,3,4,5]
numbers.index(5,3)
```

9

리스트에서 해당 항목을 찾지 못하면 에러가 발생합니다.

```
numbers = [1,3,5,7,9,1,2,3,4,5]
numbers.index(6,3)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-49-d9f9352672c9> in <module>()
      1 numbers = [1,3,5,7,9,1,2,3,4,5]
----> 2 numbers.index(6, 3)

ValueError: 6 is not in list
```

### 3) 인덱스를 이용한 직접 접근

[*index*] 형식을 이용하면 리스트에서 해당 인덱스 위치의 항목을 찾을 수 있습니다.

```
numbers = [1,3,5,7,9,1,2,3,4,5]
numbers[2]
```

```
5
```

0번 인덱스는 맨 첫 항목을 찾습니다.

```
numbers[0]
```

```
1
```

인덱스를 벗어난 참조는 에러를 발생시킵니다.

```
numbers[20]
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-1-3eb814297aa4> in <module>()
      1 numbers = [1,3,5,7,9,1,2,3,4,5]
----> 2 numbers[20]

IndexError: list index out of range
```

#### 4) 다차원 리스트에서 인덱스 사용

다차원 리스트에서 인덱스를 이용해 데이터를 참조하려면 차원 별로 [index]를 지정해야 합니다. 다음 예에서 'hello'를 찾으려면 a에서 a[2]로 [ 'a', 'b', [ 'hello', 'world' ] ]을 찾고 다시 a[2][2]로 [ 'hello', 'world' ]를 찾은 다음 a[2][2][0]을 이용해 'hello'를 찾습니다.

```
a = [ 1, 2, [ 'a', 'b', [ 'hello', 'world' ] ] ]
print(a[2][2][0])
```

```
hello
```

### 1.5. 슬라이싱

- [ from\_index : to\_index ] : from\_index부터 to\_index까지 부분 리스트 추출
- [ from\_index : to\_index : step ] : from\_index부터 to\_index까지 매 step 번째 아이템을 추출

실습을 위해 다음 데이터를 이용합니다.

```
numbers = [1,2,3,4,5,6,7,8,9,10]
```

```
numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

#### 1) [ from\_index : to\_index ]

[ from\_index : to\_index ]는 인덱스를 이용해서 부분 리스트를 추출합니다.

from\_index 위치에서 to\_index 위치(to\_index 포함 안함)까지 부분 리스트를 추출합니다.

```
numbers[2:4]
```

```
[3, 4]
```

to를 넣지 않으면 끝 항목까지 추출합니다.

```
numbers[2:]
```

```
[3, 4, 5, 6, 7, 8, 9, 10]
```

from을 넣지 않으면 첫 항목부터 추출합니다.

```
numbers[:4]
```

```
[1, 2, 3, 4]
```

from과 to 둘 다 포함하지 않으면 모든 데이터를 추출합니다.

```
numbers[:]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

인덱스에 음수를 사용하면 맨 뒤에서부터 순서를 찾습니다. 이 경우에도 to 위치의 항목은 포함되지 않습니다.

```
numbers[-5:-1]
```

```
[6, 7, 8, 9]
```

from 항목을 포함하지 않을 경우 맨 처음 항목이 됩니다. 이 경우는 to도 마찬가지입니다.

```
numbers[:-1]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

다음처럼 인덱스를 잘 못 사용할 경우 아무것도 출력되지 않습니다

```
numbers[-1:-5]
```

```
[]
```

2차원 리스트도 인덱싱을 통해 부분 리스트를 추출할 수 있습니다.

```
numbers_2d[1:3]
```

```
[[10, 20, 30, 40, 50], [1, 3, 5, 7, 9]]
```

numbers\_2d[1]의 결과는 [10, 20, 30, 40, 50]이므로 다음처럼 인덱싱을 통한 부분 리스트에서 다시 부분 리스트를 추출할 수 있습니다.

```
numbers_2d[1][1:4]
```

```
[20, 30, 40]
```

## 2) [ from\_index : to\_index : step ]

[ *from\_index* : *to\_index* : *step* ] 형식은 매 *step* 번째 아이템을 추출해 줍니다.

슬라이싱을 위해서 *from\_index*, *to\_index*, *step*은 생략될 수 있습니다. 만일 *from\_index*와 *to\_index*가 생략되면 `::step` 형식이 됩니다.

```
numbers[::2]
```

```
[1, 3, 5, 7, 9]
```

step이 음수이면 역순으로 출력할 수 있습니다.

```
numbers[::-1]
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
numbers[-5::-1]
```

```
[6, 5, 4, 3, 2, 1]
```

## 1.6. 항목 수정하기

### 1) 인덱싱으로 데이터 수정하기

리스트 데이터는 인덱싱 방법으로 수정할 수 있습니다. 지정한 인덱스 위치의 데이터를 다른 데이터로 바꿉니다.

```
numbers = [1,3,5,7,9,1,2,3,4,5]
numbers[2] = 2
numbers
```

```
[1, 3, 2, 7, 9, 1, 2, 3, 4, 5]
```

### 2) 슬라이싱으로 데이터 수정하기

지정한 범위의 리스트 항목을 다른 항목들로 한꺼번에 바꿉니다. 이 경우에는 바꾸려는 항목이 더 많거나 더 적어도 항목들을 일괄적으로 변경됩니다.

```
numbers[0:5] = [2,4,6,8,10]
numbers
```

```
[2, 4, 6, 8, 10, 1, 2, 3, 4, 5]
```

step을 이용해서도 일정 구간마다 항목을 바꿀 수 있습니다.

```
numbers[::2] = [10, 20, 30, 40, 50]
numbers
```

```
[10, 4, 20, 8, 30, 1, 40, 3, 50, 5]
```

step을 이용할 경우에는 구간 내에서 항목의 개수와 변경하려는 새로운 데이터의 개수가 일치해야 합니다.

```
numbers = [1,3,5,7,9,1,2,3,4,5]
numbers[::2] = [10, 20, 30, 40]
numbers
```



```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-90-3265c4ba8f22> in <module>()
      1 numbers = [1,3,5,7,9,1,2,3,4,5]
----> 2 numbers[::2] = [10, 20, 30, 40]
      3 numbers

ValueError: attempt to assign sequence of size 4 to extended slice of size 5
```

## 1.7. 삭제하기

- `pop()` : 맨 뒤의 항목 반환 및 삭제
- `remove()` : 해당 항목 삭제
- `del` : 지정한 위치 항목 삭제, 변수 삭제
- `clear()` : 모든 항목 삭제

실습을 위해 다음 데이터를 이용합니다.

```
numbers = [1,2,3,4,5,6,7,8,9,10]
```

```
numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

### 1) `pop()`

`pop()` 함수는 맨 뒤의 항목을 반환하고 해당 데이터를 삭제합니다.

```
numbers.pop()
```

```
10
```

```
numbers
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2차원 리스트에서 `pop()` 함수의 동작은 맨 마지막 리스트를 뽑아내는 것입니다.

```
numbers_2d.pop()
```

```
[2, 4, 6, 8, 10]
```

```
numbers_2d
```

```
[[1, 2, 3, 4, 5], [10, 20, 30, 40, 50], [1, 3, 5, 7, 9]]
```

## 2) remove()

remove() 함수는 해당 항목을 삭제합니다.

```
numbers.remove(3)  
numbers
```

```
[1, 2, 4, 5, 6, 7, 8, 9]
```

2차원 리스트에서 부분 리스트를 삭제하려면 삭제할 리스트를 인수로 넣어야 합니다.

```
numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]  
numbers_2d.remove([1,2,3,4,5])  
numbers_2d
```

```
[[10, 20, 30, 40, 50], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
```

## 3) del

리스트의 특정 항목을 삭제할 때 del 을 사용할 수도 있습니다.

```
numbers = [1,2,3,4,5,6,7,8,9,10]  
del numbers[3]  
numbers
```

```
[1, 2, 3, 5, 6, 7, 8, 9, 10]
```

슬라이싱을 이용해 삭제도 가능합니다.

```
numbers = [1,2,3,4,5,6,7,8,9,10]
del numbers[3:8]
numbers
```

```
[1, 2, 3, 9, 10]
```

```
numbers = [1,2,3,4,5,6,7,8,9,10]
del numbers[:2]
numbers
```

```
[2, 4, 6, 8, 10]
```

del은 remove() 함수와는 다르게 변수 자체를 삭제할 수 있습니다.

```
del numbers
```

삭제 된 변수를 참고하려 하면 에러가 발생합니다.

```
print(numbers)
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-62-6a54518a0c2c> in <module>()
----> 1 numbers

NameError: name 'numbers' is not defined
```

#### 4) clear()

clear() 함수는 리스트의 모든 항목을 삭제합니다.

```
numbers = [1,2,3,4,5,6,7,8,9,10]
numbers.clear()
numbers
```

```
[]
```

## 1.8. 정렬하기

- `sort()` : 정렬(`reverse=True` 속성을 이용하면 내림차순 정렬)
- `reverse()` : 역순으로 나열(내림차순 정렬이 아님)
- `[::-1]` :

`sort()` 함수는 리스트를 정렬합니다. 기본 정렬은 오름차순입니다.

```
numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
numbers.sort()
numbers
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

`sort()` 함수에 `reverse` 속성을 사용하면 내림차순으로 정렬합니다.

```
numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
numbers.sort(reverse=True)
numbers
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

`reverse()` 함수는 리스트를 역순으로 변경합니다. 내림차순 정렬을 의미하는 것이 아닙니다.

```
numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
numbers.reverse()
numbers
```

```
[5, 1, 9, 3, 8, 10, 4, 7, 2, 6]
```

[ *from\_index* : *to\_index* : *step* ] 형식의 슬라이싱에서 *step*이 음수이면 역순으로 출력할 수 있습니다.

```
numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
numbers[::-1]
```

```
[5, 1, 9, 3, 8, 10, 4, 7, 2, 6]
```

이 경우에는 `numbers` 변수의 값이 바뀌지는 않습니다. 역순으로 바뀐 결과를 다시 사용하

려면 새로운 변수에 할당해야 합니다.

```
numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
new_numbers = numbers[::-1]
new_numbers
```

```
[5, 1, 9, 3, 8, 10, 4, 7, 2, 6]
```

## 1.9. 리스트 복제

- `copy()` : 복제된 새로운 객체를 생성
- `=` : 주소를 복사해 같은 객체를 참조

### 1) `copy()`

`copy()` 함수를 사용하면 리스트를 복사합니다. 이 경우 새로운 리스트가 만들어집니다.

```
numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
new_numbers = numbers.copy()
print(numbers)
print(new_numbers)
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

`copy()`에 의해 만들어진 리스트는 이전 리스트와는 다른 리스트입니다.

```
numbers.remove(6)
print(numbers)
print(new_numbers)
```

```
[2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

### 2) 할당문(=)

할당문(=)을 이용해 새로운 변수에 리스트를 할당하면 두 변수는 같은 객체를 참조합니다.

이 경우 어느 한 리스트의 항목을 변경하면 다른 리스트도 같은 결과가 됩니다.

```
numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
new_numbers = numbers
print(numbers)
print(new_numbers)
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

다음처럼 numbers 리스트에서 항목을 삭제하면 new\_numbers 리스트도 삭제된 결과가 조회됩니다.

```
numbers.remove(5)
print(numbers)
print(new_numbers)
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1]
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1]
```

## 2. 튜플

튜플은 소괄호('('와 ')')를 이용해 만듭니다. 튜플(tuple)은 리스트와 유사하지만 읽기 전용입니다. 튜플은 속도가 빨라 수정이 필요 없는 배열 형태의 데이터 타입에 사용합니다. 튜플은 데이터를 수정할 수 없기 때문에 제공되는 함수가 많지 않습니다. 제공되는 함수도 count(), index() 정도입니다.

표 4. 튜플 다루기

방법	설명
tupleData = ( )	튜플을 만들어 줍니다.
len(tupleData)	튜플의 항목 수를 반환합니다.
min(tuple), max(tuple)	튜플에서 가장 작은 값(min)과 가장 큰 값(max)을 반환합니다.
tupleData.count(value)	튜플에서 value의 개수를 반환합니다.
tupleData.index(value, position)	position위치 이후에서 value의 값이 있는 인덱스를 반환합니다.

### 2.1. 튜플 만들기

#### 1) 1차원 튜플

다음 코드는 튜플을 만듭니다. 기본적으로 튜플은 소괄호 ( )를 이용해 만들지만 소괄호 없이 만들 수 있습니다

```
numbers1 = (1,2,3,4,5)
type(numbers1)
```

```
tuple
```

소괄호 안의 항목이 하나일 경우에는 콤마(,)가 있어야 튜플이 됩니다.

```
numbers2 = (1,)
type(numbers2)
```

```
tuple
```

```
numbers3 = 1,
type(numbers3)
```

```
tuple
```

```
print(numbers1)
print(numbers2)
print(numbers3)
```

```
(1, 2, 3, 4, 5)
(1,)
(1,)
```

튜플에 저장한 데이터의 항목이 하나인 경우 기본 타입(예에서는 정수)과 혼동하지 않도록 값뒤에 콤마(,)를 붙여 출력합니다. 그러나 데이터 항목이 두 개 이상인 경우 맨 뒤에 콤마(,)를 붙이지 않아도 됩니다.

```
numbers3 = (1,2,3)
numbers5 = 1,2,3
print(numbers3)
print(numbers5)
```

```
(1, 2, 3)
(1, 2, 3)
```

## 2) 다차원 튜플

소괄호 안에 소괄호를 이용해서 2차원 이상 구조를 갖는 튜플을 만들 수 있습니다. 다차원 튜플은 리스트에서의 개념과 같습니다.

```
numbers_2d = ((1,2,3), (4,5,6))
numbers_2d
```

```
((1, 2, 3), (4, 5, 6))
```



## 2.2. 기본 정보 조회<sup>33)</sup>

항목의 수를 반환하는 `len()` 함수, 가장 작은 값을 반환하는 `min()` 함수, 그리고 가장 큰 값을 반환하는 `max()` 함수는 리스트에서의 함수와 같은 의미를 갖습니다.

- `len()` : 길이(항목의 수)
- `min()`, `max()` : 최댓값, 최솟값

실습을 위해 다음 데이터를 이용합니다.

```
numbers = (1,2,3,4,5)
```

```
numbers_2d = ((1,2,3,4,5), (10,20,30,40,50), (1,3,5,7,9), (2,4,6,8,10))
```

### 1) 항목 수

`len()` 함수는 튜플에서 항목 수를 반환합니다. `len()` 함수는 튜플의 함수가 아닙니다. 그러므로 튜플 객체를 함수의 인수에 전달해야 합니다.

```
len(numbers)
```

```
5
```

2차원 튜플은 행의 수를 출력합니다.

```
len(numbers_2d)
```

```
4
```

2차원 튜플에서 행별 항목의 수는 인덱싱한 튜플을 이용해 조회합니다.

```
len(numbers_2d[0])
```

```
5
```

33) [2. 튜플]에서 [2.2 기본 정보 조회], [2.3 인덱싱], [2.4 슬라이싱]은 리스트에서의 개념과 같습니다. 리스트에서 이해했다면 이 부분은 읽지 않고 건너뛰어도 됩니다.

## 2) 최소값과 최대값

`min()` 함수는 리스트에서 가장 작은 항목을 반환하며, `max()` 함수는 가장 큰 항목을 반환합니다.

```
min(numbers)
```

```
1
```

```
max(numbers)
```

```
5
```

2차원 튜플에서 `max()` 함수는 부분 리스트들 중에서 첫 항목이 가장 큰 값을 갖는 튜플을 반환합니다.

```
max(numbers_2d)
```

```
(10, 20, 30, 40, 50)
```

## 2.3. 인덱싱

- `count()` : 리스트에서 데이터의 개수를 반환
- `index()`
- `[index]`

### 1) 데이터 수 세기

`count()` 함수는 데이터의 개수를 셉니다.

```
numbers = (1,3,5,7,9,1,2,3,4,5)
numbers.count(1)
```

```
2
```

- 파이썬에서 인덱스는 0부터 시작합니다. 그러므로 맨 처음 항목의 인덱스는 0입니다.

## 2) 항목의 위치 반환하기

index() 함수는 해당 항목의 위치를 반환합니다.

```
numbers = (1,3,5,7,9)
numbers.index(5)
```

2

index() 함수의 두 번째 인수에 인덱스를 추가하면 해당 위치 이후에서 찾습니다. 다음 코드는 데이터 5를 인덱스 3위치 이후에서 찾습니다.

```
numbers = (1,3,5,7,9,1,2,3,4,5)
numbers.index(5,3)
```

9

튜플에서에서 해당 항목을 찾지 못하면 예외가 발생합니다.

```
numbers = (1,3,5,7,9,1,2,3,4,5)
numbers.index(6,3)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-49-d9f9352672c9> in <module>()
      1 numbers = (1,3,5,7,9,1,2,3,4,5)
----> 2 numbers.index(6, 3)

ValueError: 6 is not in tuple
```

## 3) 인덱스를 이용한 직접 접근

[index] 형식을 이용하면 리스트에서 해당 인덱스 위치의 항목을 찾을 수 있습니다.

```
numbers = (1,3,5,7,9,1,2,3,4,5)
numbers[2]
```

5

0번 인덱스는 맨 첫 항목을 찾습니다.

```
numbers[0]
```

```
1
```

인덱스를 벗어난 참조는 에러를 발생시킵니다.

```
numbers[20]
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-1-3eb814297aa4> in <module>()
      1 numbers = (1,3,5,7,9,1,2,3,4,5)
----> 2 numbers[20]

IndexError: tuple index out of range
```

#### 4) 다차원 튜플에서 인덱스 사용

다차원 튜플에서 인덱스를 이용해 데이터를 참조하려면 차원 별로 [index]를 지정해야 합니다. 다음 예에서 'hello'를 찾으려면 a에서 a[2]로 [ 'a', 'b', [ 'hello', 'world' ] ]을 찾고 다시 a[2][2]로 [ 'hello', 'world' ]를 찾은 다음 a[2][2][0]을 이용해 'hello'를 찾습니다.

```
a = ( 1, 2, ( 'a', 'b', ( 'hello', 'world' ) ) )
print(a[2][2][0])
```

```
hello
```

## 2.4. 슬라이싱

- [ from\_index : to\_index ] : from\_index부터 to\_index까지 부분 튜플 추출
- [ from\_index : to\_index : step ] : from\_index부터 to\_index까지 매 step 번째 아이템을 추출

실습을 위해 다음 데이터를 이용합니다.

```
numbers = (1,2,3,4,5,6,7,8,9,10)
```

```
numbers_2d = ((1,2,3,4,5), (10,20,30,40,50), (1,3,5,7,9), (2,4,6,8,10))
```

### 1) [ from\_index : to\_index ]

[ *from\_index* : *to\_index* ]는 인덱스를 이용해서 부분 튜플을 추출합니다.

*from\_index* 위치에서 *to\_index* 위치(*to\_index* 포함 안함)까지 부분 튜플을 추출합니다.

```
numbers[2:4]
```

```
(3, 4)
```

to를 넣지 않으면 끝 항목까지 추출합니다.

```
numbers[2:]
```

```
(3, 4, 5, 6, 7, 8, 9, 10)
```

from을 넣지 않으면 첫 항목부터 추출합니다.

```
numbers[:4]
```

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

from과 to 둘 다 포함하지 않으면 모든 데이터를 추출합니다.

```
numbers[:]
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

인덱스에 음수를 사용하면 맨 뒤에서부터 순서를 찾습니다. 이 경우에도 to 위치의 항목은 포함되지 않습니다.

```
numbers[-5:-1]
```

```
(6, 7, 8, 9)
```

from 항목을 포함하지 않을 경우 맨 처음 항목이 됩니다. 이 경우는 to도 마찬가지입니다.

```
numbers[: -1]
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

다음처럼 인덱스를 잘 못 사용할 경우 아무것도 출력되지 않습니다

```
numbers[-1:-5]
```

```
()
```

2차원 리스트도 인덱싱을 통해 부분 튜플을 추출할 수 있습니다.

```
numbers_2d[1:3]
```

```
((10, 20, 30, 40, 50), (1, 3, 5, 7, 9))
```

numbers\_2d[1]의 결과는 (10, 20, 30, 40, 50)이므로 다음처럼 인덱싱을 통한 부분 튜플에서 다시 부분 튜플을 추출할 수 있습니다.

```
numbers_2d[1][1:4]
```

```
(20, 30, 40)
```

## 2) [ from\_index : to\_index : step ]

[ *from\_index* : *to\_index* : *step* ] 형식은 매 *step* 번째 아이템을 추출해 줍니다.

슬라이싱을 위해서 *from\_index*, *to\_index*, *step*은 생략될 수 있습니다. 만일 *from\_index*와 *to\_index*가 생략되면 `::step` 형식이 됩니다.

```
numbers[::2]
```

```
(1, 3, 5, 7, 9)
```

*step*이 음수이면 역순으로 출력할 수 있습니다.

```
numbers[::-1]
```

```
(10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
```

```
numbers[-5::-1]
```

```
(6, 5, 4, 3, 2, 1)
```

## 2.5. 튜플 연결과 반복

- + : 두 리스트를 연결
- \* : 리스트를 곱한 수만큼 반복

덧셈(+) 기호로 튜플을 연결하거나 곱셈(\*)을 이용해 튜플을 반복시킬 수 있습니다. 그러나 이 경우 새로운 변수로 변경된 결과를 할당받아 저장해야 합니다. 리스트에 있던 `append()`, `extend()`, `insert()` 함수는 튜플에 없습니다.

### 1) +에 의한 연결

덧셈(+) 연산자는 두 리스트를 연결합니다.

```
numbers = (1,2,3,4,5)
```

```
numbers + numbers
```

```
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

두 리스트를 + 연산자로 합칠 수 있습니다. 그러나 이 경우에는 결과를 받아 저장하는 할당문이 있어야 합니다. 아래 구문처럼 할당문이 없을 경우에는 두 튜플을 더한 결과를 출력할 뿐 결과는 어디에도 저장되지 않습니다.

```
numbers
```

```
(1, 2, 3, 4, 5)
```

아래 구문처럼 +한 결과를 변수에 저장하도록 할당하는 구문입니다.

```
new_number = number + number
```

```
new_number
```

```
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

## 2) \*에 의한 반복

곱셈(\*) 연산자는 곱한 수만큼 튜플을 반복합니다. 튜플에 곱하는 값은 정수(int)여야 합니다.

```
numbers = (1,2,3,4,5)
```

```
3*numbers
```

```
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

곱셈(\*) 연산자도 반복한 결과를 저장하도록 할당해야 다시 사용할 수 있습니다.



## 2.6. 튜플에 항목 추가, 수정 및 삭제는 안 됨

튜플은 새로운 항목을 추가하거나, 기존 항목을 변경하거나 삭제할 수 없습니다.

튜플은 `append()`, `insert()` 함수가 없습니다.

```
numbers = (1,2,3,4,5)
numbers.append(6)
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-93-13f33d177890> in <module>()
      1 numbers = (1,2,3,4,5)
----> 2 numbers.append(6)

AttributeError: 'tuple' object has no attribute 'append'
```

```
numbers = (1,2,3,4,5)
numbers.insert(1,6)
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-94-f21c0c703dcd> in <module>()
      1 numbers = (1,2,3,4,5)
----> 2 numbers.insert(1,6)

AttributeError: 'tuple' object has no attribute 'insert'
```

튜플은 항목 재 할당이 불가능합니다.

```
numbers[0] = 6
```

```
-----
TypeError                                    Traceback (most recent call last)
<ipython-input-96-749f0bd79a35> in <module>()
----> 1 numbers[0] = 6

TypeError: 'tuple' object does not support item assignment
```

튜플은 항목을 삭제할 수 없습니다.

```
del numbers[0]
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-98-8028f96cbc8d> in <module>()  
----> 1 del numbers[0]  
  
TypeError: 'tuple' object doesn't support item deletion
```

튜플은 remove() 함수도 없습니다.

```
numbers.remove(3)
```

```
-----  
AttributeError                            Traceback (most recent call last)  
<ipython-input-99-5c45eb4335bb> in <module>()  
----> 1 numbers.remove(3)  
  
AttributeError: 'tuple' object has no attribute 'remove'
```

### 3. 딕셔너리

딕셔너리(dict) 타입은 사전(dictionary) 타입 이라고도 부릅니다. 딕셔너리를 만들기 위해서는 중괄호('{와 '}')를 이용합니다. 딕셔너리(Dictionary) 타입은 키(key)와 값(value)의 쌍으로 구성된 자료 구조입니다. 딕셔너리의 키는 중복이 없이 유일한 값이어야 합니다. 값은 중복이 가능하며 모든 타입이 가능합니다. 키에는 리스트 타입을 사용할 수 없습니다. 그러나 튜플 타입은 사용할 수 있습니다.

딕셔너리는 인덱스를 이용한 데이터의 참조는 지원하지 않으며 키를 이용하여 데이터를 참조합니다. 만일 딕셔너리 키 목록에 없는 데이터를 사용하여 참조하면 에러가 발생합니다.

표 5. 딕셔너리 다루기

방법	설명
<code>dictData = {"key": "value", ... }</code>	딕셔너리를 만들어 줍니다.
<code>len(dictData)</code>	딕셔너리의 항목의 수를 반환합니다.
<code>dictData.items()</code>	딕셔너리의 각 항목들을 (key, value) 형식의 튜플들로 반환합니다.
<code>dictData.keys()</code>	딕셔너리의 키(key)들을 반환합니다.
<code>dictData.values()</code>	딕셔너리의 값(value)들을 반환합니다.

#### 3.1. 딕셔너리 만들기

다음 코드는 딕셔너리를 다루는 예입니다.

```
my_favorite = {"fruit": "apple", "number": 1, "sports": "golf"}
```

```
type(my_favorite)
```

```
dict
```

`get()` 함수는 딕셔너리의 키를 이용해 값을 가져옵니다.

```
print(my_favorite.get("number"))
```

```
1
```

```
print(my_favorite.get("sports"))
```

```
golf
```

### 3.2. 키 타입과 값 타입

- 키 타입 : 문자, 숫자, 논리, 튜플, 중복 허용 안 됨
- 값 타입 : 모든 타입, 중복 가능

딕셔너리의 키 타입은 기본 데이터 타입(문자, 숫자, 논리) 모두 가능합니다. 데이터 구조 중에서 리스트 타입은 키로 사용할 수 없습니다. 그러나 튜플 타입은 사용할 수 있습니다. 딕셔너리의 키는 중복이 없이 유일한 값이어야 합니다. 값은 중복이 가능하며 모든 타입이 가능합니다.

```
my_dic1 = {1:"Hello", 2:"World", "KEY": "VALUE"}
print(my_dic)
```

```
{1: 'Hello', 2: 'World', 'KEY': 'VALUE'}
```

```
print(my_dic1.get(1))
```

```
Hello
```

```
print(my_dic1.get("KEY"))
```

```
VALUE
```

딕셔너리의 키가 논리 타입일 경우 True는 숫자 1과 같은 값으로 간주됩니다. 그러므로 숫자 1과 논리값 True는 같은 키로 간주됩니다. 다음 예제에서는 True:“World”가 1:“Hello”를 덮어씁니다.

```
my_dic2 = {1:"Hello", True:"World", "KEY": "VALUE"}
```

```
print(my_dic2.get(True))
```

```
World
```

```
print(my_dic2)
```

```
{1: 'World', 'KEY': 'VALUE'}
```

논리타입 False를 키로 사용하면 숫자 0과 같은 키로 사용됩니다.

```
my_dic3 = {1:"Hello", False:"World", "KEY": "VALUE"}
```

```
print(my_dic3)
```

```
{1: 'Hello', False: 'World', 'KEY': 'VALUE'}
```

```
print(my_dic3.get(False))
```

```
World
```

```
print(my_dic3.get(0))
```

```
World
```

키는 대/소문자를 구분합니다.

```
my_dic4 = {"Key": "Value", "key": "value", "KEY": "VALUE"}
print(my_dic4)
```

```
{'Key': 'Value', 'key': 'value', 'KEY': 'VALUE'}
```

### 3.3. 추가하기

- dict\_data[key] = value : 기존 값 변경 또는 새로운 key:value 쌍 추가
- append, extend, insert : 지원하지 않음

기존 딕셔너리에 새로운 key:value 쌍을 추가하려면 dict\_data[key] = value 형식을 사용합니다.

이미 있는 키를 이용하면 값을 변경합니다.

```
my_favorite = {"fruit": "apple", "number": 1, "sports": "golf"}
```

```
my_favorite["fruit"] = "banana"
my_favorite
```

```
{'fruit': 'banana', 'number': 1, 'sports': 'golf'}
```

그러나 없는 키를 이용하면 새로운 key:value 쌍을 추가합니다.

```
my_favorite["language"] = "python"
my_favorite
```

```
{'fruit': 'banana', 'number': 1, 'sports': 'golf', 'language': 'python'}
```

덕셔너리는 append, extend, insert 기능을 지원하지 않습니다.

```
my_fav.append({"number": 1})
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-11-f146ab508252> in <module>()
----> 1 my_fav.append({"number": 1})

AttributeError: 'dict' object has no attribute 'append'
```

```
my_fav.extend({"number": 1})
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-12-2bc1c551e241> in <module>()
----> 1 my_fav.extend({"number": 1})

AttributeError: 'dict' object has no attribute 'extend'
```

```
my_fav.insert(1, {"number": 1})
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-13-a1d26e9d1f85> in <module>()
----> 1 my_fav.insert(1, {"number": 1})
```

```
AttributeError: 'dict' object has no attribute 'insert'
```

### 3.4. 삭제하기

딕셔너리 항목을 삭제하려면 `del` 을 이용합니다.

```
my_favorite = {"fruit": "apple", "number": 1, "sports": "golf"}
del my_favorite['fruit']
my_favorite
```

```
{'number': 1, 'sports': 'golf'}
```

`pop()` 함수는 지정한 키로 값을 가져오고 해당 키와 값을 삭제합니다.

```
my_favorite = {"fruit": "apple", "number": 1, "sports": "golf"}
```

```
print(my_favorite.pop("number"))
```

```
1
```

```
print(my_favorite)
```

```
{'fruit': 'apple', 'sports': 'golf'}
```

### 3.5. 조회하기

딕셔너리에서 데이터를 조회하는 방법은 `[]` 를 이용하는 방법뿐만 아니라 `반복문`과 `keys()`, `values()`, `items()` 등을 이용할 수 있습니다.

다음 데이터를 이용해 예를 들었습니다.

```
my_favorite = {'fruit': 'apple', 'sports': 'golf', 'number': 1, 'food': '라면'}
```

딕셔너리 변수를 for 반복문에 사용하면 키 목록을 반환합니다.

```
for fav in my_favorite:
    print(fav)
```

```
fruit
sports
number
food
```

키 목록으로 값을 얻으려면 [ ]를 사용합니다.

```
for fav in my_favorite:
    print(my_favorite[fav])
```

```
apple
golf
1
라면
```

[ ] 대신 get() 메서드를 사용해도 됩니다.

```
for fav in my_favorite:
    print(my_favorite.get(fav))
```

```
apple
golf
1
라면
```

keys() 함수는 키 목록을 반환하며, values() 함수는 값 목록을 반환합니다.

```
for key in my_favorite.keys():
    print(key)
```

```
fruit
sports
number
food
```



```
for value in my_favorite.values():
    print(value)
```

```
apple
golf
1
라면
```

items() 함수는 키/값을 튜플로 반환합니다. 튜플의 [0]번째는 키, [1]번째는 값입니다.

```
for item in my_favorite.items():
    print(item)
```

```
('fruit', 'apple')
('sports', 'golf')
('number', 1)
('food', '라면')
```

```
for item in my_favorite.items():
    print(item[0], item[1])
```

```
fruit apple
sports golf
number 1
food 라면
```

items() 함수의 결과에서 키와 값을 각각 다른 변수에 받을 수 있습니다.

```
for key, value in my_favorite.items():
    print(key, value)
```

```
fruit apple
sports golf
number 1
food 라면
```

### 3.6. in 연산자

in 연산자는 딕셔너리의 키에 있는지 확인합니다.

다음 코드는 my\_favorite 딕셔너리에 "sports"키가 있는지 확인합니다.

```
my_favorite = {"fruit": "apple", "number": 1, "sports": "golf"}
```

```
"sports" in my_favorite
```

```
True
```

```
"golf" in my_favorite
```

```
False
```

### 3.7. 딕셔너리 복제

copy() 함수는 딕셔너리를 복제합니다.

```
my_favorite = {"fruit": "apple", "number": 1, "sports": "golf"}
```

```
my_fav = my_favorite.copy()
print(my_fav)
print(my_favorite)
```

```
{'fruit': 'apple', 'number': 1, 'sports': 'golf'}
{'fruit': 'apple', 'number': 1, 'sports': 'golf'}
```

이렇게 복제된 딕셔너리는 같은 값을 가졌지만 다른 객체이므로 한 객체의 항목을 변경하더라도 다른 객체에 영향을 주지 않습니다.

```
del my_fav['fruit']
print(my_fav)
print(my_favorite)
```

```
{'number': 1, 'sports': 'golf'}
{'fruit': 'apple', 'number': 1, 'sports': 'golf'}
```

= 연산자로 할당된 경우에는 두 변수가 같은 객체를 가리키므로 한 변수의 항목을 변경하면 다른 변수를 통해서 참조하는 값도 변경된 값을 보게 됩니다.

```
my_favorite = {"fruit":"apple", "number":1, "sports":"golf"}
my_fav = my_favorite
print(my_fav)
print(my_favorite)
```

```
{'fruit': 'apple', 'number': 1, 'sports': 'golf'}
{'fruit': 'apple', 'number': 1, 'sports': 'golf'}
```

```
del my_fav['fruit']
print(my_fav)
print(my_favorite)
```

```
{'number': 1, 'sports': 'golf'}
{'number': 1, 'sports': 'golf'}
```

## 4. 셋

셋(set)은 순서가 정해지지 않고, 중복을 허용하지 않는 집합을 의미합니다. 셋은 중괄호('{ 와 }')를 이용하여 정의합니다. 다른 데이터 구조(리스트, 튜플, 딕셔너리)를 셋으로 바꾸려면 set() 함수를 이용합니다.

### 4.1. 셋 만들기

다음 코드는 셋 데이터를 정의하는 예입니다.

```
fruits = {"apple", "orange", "banana", "apple", "bear"}
```

```
print(fruits)
```

```
{'orange', 'banana', 'apple', 'bear'}
```

set() 함수는 리스트, 튜플, 딕셔너리를 셋으로 바꿉니다. 딕셔너리는 키(key)만 셋에 추가됩니다.

```
s1 = set([1,2,3])
```

```
s1
```

```
{1, 2, 3}
```

```
s2 = set((10,20))
```

```
s2
```

```
{10, 20}
```

```
s3 = set({'key' : 'value'})
```

```
s3
```

```
{'key'}
```

## 4.2. 추가하기

- 중복된 값은 추가 안 되며, 순서를 보장하지 않음
- `add()` : 기본 데이터 타입과 튜플 추가. 리스트와 딕셔너리는 `add()` 함수로 추가할 수 없음
- `update()` : 리스트, 튜플, 딕셔너리, 문자를 추가

셋은 중복된 값을 가질 수 없습니다. 이미 들어 있는 값과 같은 값을 `add` 할 수 없습니다.

```
fruits.add("apple")
```

```
print(fruits)
```

```
{'orange', 'banana', 'apple', 'bear'}
```

셋은 입력한 순서대로 데이터의 순서를 유지하지 않습니다. 다음 코드는 “mango”를 맨 마지막에 `add` 했지만 출력은 중간에 된 것을 볼 수 있습니다.

```
fruits.add("mango")
```

```
print(fruits)
```

```
{'banana', 'orange', 'mango', 'bear', 'apple'}
```

`add()` 함수는 기본 데이터 타입(숫자, 문자, 논리)과 튜플을 추가합니다. 리스트와 딕셔너리는 `add()` 함수로 추가할 수 없습니다. `update()` 함수는 리스트, 튜플, 딕셔너리, 그리고 문자를 추가할 수 있습니다. 딕셔너리는 키(key)만 추가됩니다.

```
s1 = {1,2}
s1.add('Hello')
s1.add(False)
s1.add((10, 20))
s1.add('abc')
s1
```

```
{(10, 20), 1, 2, False, 'Hello', 'abc'}
```

셋의 항목 중에서 1값이 있을 경우 True가 추가되지 않으며, 0값이 있을 경우 False가 추가되지 않습니다.

```
s1 = {0, 2}
s1.add(False)
s1
```

```
{0, 2}
```

add() 함수로 리스트는 추가할 수 없습니다.

```
s1 = {1,2}
s1.add([10,20,30])
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-14-7533bad45f21> in <module>()
      1 s1 = {1,2}
----> 2 s1.add([10,20,30])

TypeError: unhashable type: 'list'
```

리스트, 튜플, 딕셔너리에서 항목 하나씩 셋에 추가하려면 update() 함수를 이용합니다. 딕셔너리는 update() 함수에 의해 키만 추가됩니다.

```
s2 = {1,2}
s2.update([10,20,30])
s2
```

```
{1, 2, 10, 20, 30}
```

```
s2.update((100, 200, 300))
s2
```

```
{1, 2, 10, 20, 30, 100, 200, 300}
```

```
s2.update({"city":"seoul", "age": 20})
s2
```

```
{1, 10, 100, 2, 20, 200, 30, 300, 'age', 'city'}
```

### 4.3. 합집합, 교집합, 차집합

- $&$  : 두 셋의 교집합
- $|$  : 두 셋의 합집합
- $-$  : 두 셋의 차집합

$&$ 를 이용하면 두 집합의 교집합을 구하며,  $|$ 를 이용하면 두 집합의 합집합을 구합니다. 마이너스(-) 기호를 이용하면 차집합을 구할 수 있습니다. 리스트, 튜플, 딕셔너리 타입은 이들 연산자를 사용할 수 없습니다.

```
s1 = set([1, 2, 3, 4, 5, 6])
s2 = set([4, 5, 6, 7, 8, 9])
print(s1, s2)
```

```
{1, 2, 3, 4, 5, 6} {4, 5, 6, 7, 8, 9}
```

$&$ 는 교집합을 구합니다.

```
s1 & s2
```

```
{4, 5, 6}
```

$|$ (수직바)는 합집합을 구합니다.

```
s1 | s2
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

마이너스(-)는 차집합을 구합니다.

```
s1 - s2
```

```
{7, 8, 9}
```

## 4.4. 삭제하기

셋에서 항목의 삭제는 `remove()` 함수를 이용합니다. 함수의 인수는 셋에서 삭제할 값을 입력합니다.

```
s1 = {(10, 20), 1, 2, False, 'Hello', 'abc'}
s1.remove(1)
s1
```

```
{(10, 20), 2, False, 'Hello', 'abc'}
```

만일 없는 값을 삭제하려면 `KeyError`가 발생합니다.

```
s1.remove(200)
```

```
-----
KeyError                                Traceback (most recent call last)
<ipython-input-57-2f0b1326ae86> in <module>()
----> 1 s1.remove(200)

KeyError: 200
```

셋에 포함되어 있는 문자 또는 튜플 타입도 삭제할 수 있습니다.

```
s1.remove('Hello')
s1
```

```
{(10, 20), 2, False, 'abc'}
```

```
s1.remove((10, 20))
s1
```

```
{2, False, 'abc'}
```

논리값 `False`는 0으로 삭제할 수 있으며, `True`는 1로 삭제가 가능합니다.

```
s1.remove(0)
s1
```

```
{2, 'abc'}
```



## 5. enumerate

for 반복문을 사용하면 리스트 안의 데이터를 하나씩 꺼내서 처리할 수 있습니다. 예를 위해 리스트 데이터를 선언합니다.

```
fruits = ['watermelon', 'orange', 'mango', 'grape', 'banana', 'apple']
```

아직 for 반복문을 배우지는 않았지만 아래의 예제를 작성하고 실행하는데 어려움을 없을 것입니다. for 반복문은 4장 제어문에서 더 자세하게 배우므로 여기에서는 리스트 안의 데이터를 처리하는 예제 구문으로만 이해하세요.

```
for fruit in fruits:
    print("데이터는 {}".format(fruit))
```

```
데이터는 watermelon입니다.
데이터는 orange입니다.
데이터는 mango입니다.
데이터는 grape입니다.
데이터는 banana입니다.
데이터는 apple입니다.
```

그런데 리스트 데이터를 처리할 때 리스트 항목들의 인덱스와 데이터를 사용하기 위해서는 어떻게 해야 할까요? 아마도 다음처럼 len() 함수와 range() 함수, 그리고 [index] 형식을 이용할 수 있을 것입니다.

```
for i in range(len(fruits)):
    print("{}번째 데이터는 {}".format(i+1, fruits[i]))
```

```
1번째 데이터는 watermelon입니다.
2번째 데이터는 orange입니다.
3번째 데이터는 mango입니다.
4번째 데이터는 grape입니다.
5번째 데이터는 banana입니다.
6번째 데이터는 apple입니다.
```

내장 함수 enumerate()는 특정 루프를 조금 더 명확하게 만들어 줍니다. enumerate() 함수의 인수는 반복자(iterator) 또는 순서(sequence) 객체가 될 수 있습니다. enumerate(thing)

이라고 사용했을 경우 이 함수는 thing 반복자 객체를 (0, thing[0]), (1, thing[1]), (2, thing[2]), ... 이런 형식으로 리턴합니다.

```
enumerate(fruits)
```

```
<enumerate at 0x4d26af8>
```

다음처럼 enumerate() 함수를 이용하면 for 반복문을 이용해 처리할 때 리스트의 인덱스와 값을 동시에 받아 처리할 수 있습니다.

```
for index, value in enumerate(fruits):  
    print("{}번째 데이터는 {}입니다.".format(index+1, value))
```

1번째 데이터는 watermelon입니다.

2번째 데이터는 orange입니다.

3번째 데이터는 mango입니다.

4번째 데이터는 grape입니다.

5번째 데이터는 banana입니다.

6번째 데이터는 apple입니다.

## 6. 연습문제

---

1) 원금이 350만원 이율이 연3.5%로 3년간 복리로 예금했을 때 이자를 구하세요.

- $M = B(1+R)^Y$  R: 이자, B: 원금, Y: 년수
- pow() 함수 이용

2) 오늘의 온도를 화씨로 변환하는 프로그램을 작성하세요.

- $C = (F-32)/1.8$
- $F = (C*1.8) + 32$

3) 문자열의 슬라이싱과 합치기 기능을 이용하여 'Hello World'를 'World Hello'로 변경하세요.

4)  $x = \text{'abcdef'}$  를  $\text{'bcdefa'}$ 로 변경하세요.

- 문자열 슬라이싱 기능을 이용하세요.

5)  $x = \text{'abcdef'}$  를  $\text{'fdecba'}$ 로 출력하세요.

정답 및 풀이

1)

```
m = 350*pow((1+0.035),3)
print(m)
```

```
388.05125624999994
```

2)

```
f = (21*1.8) + 32
print(f)
```

```
69.800000000000001
```

3)

```
str = "Hello World"
print(str.split(' ')[1], str.split(' ')[0], sep=" ")
```

```
World Hello
```

4)

```
x = 'abcdef'
print(x[1:]+x[:1])
```

```
bcdefa
```

5)

```
print(x[::-1])
```

```
fedcba
```

```
print(''.join(reversed(x)))
```

```
fedcba
```



## 4장. 제어문

## 1. 조건문

### 1.1. if

조건문에는 if 라는 키워드를 사용합니다.

if 다음에는 '조건식'이 존재하는데 이 '조건식'이 참(True)이면 들여쓰기 한 문장이 실행됩니다.

if 문장 끝에는 콜론(:) 을 입력해야 합니다. 콜론은 블록의 시작을 의미합니다.

if 문의 '조건식'이 참(True)일 때 실행되는 문장은 들여쓰기를 해야 합니다.

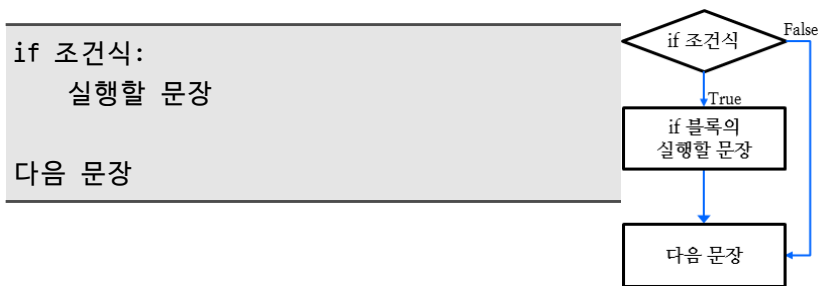


그림 1. if 구문 순서도

다음 코드는 if 블록이 있는 예입니다.

```
score = 92
if score >= 90:
    print('grade A')
```

```
grade A
```

다음 코드는 if 블록만 있는 구문은 if 블록의 조건식이 거짓일 경우 아무것도 하지 않습니다.

```
score = 80
if score >= 90:
    print('grade A')
```

\* 조건식이 거짓이므로 아무것도 실행하지 않았습니다.

## 1.2. if~else

else는 if 문의 조건식이 False일 경우 실행하는 블록을 정의합니다. else는 단독으로 사용될 수 없으며 반드시 if와 같이 사용돼야 합니다.

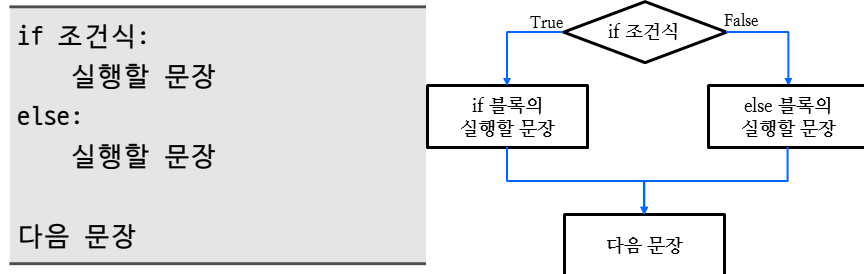


그림 2. if~else 구문 순서도

다음 코드는 if와 else를 사용한 예입니다.

```

score = 59
if score >= 60:
    print('합격')
else:
    print('불합격')
  
```

불합격

if 문의 조건식이 거짓일 경우 아무것도 실행하지 않으려면 else 문을 작성하지 않아도 되지만 다음 코드처럼 pass를 이용해도 됩니다.

```

score = 59
if score >= 60:
    print('합격')
else:
    pass
  
```

### 1.3. if~elif~else

elif는 if와 같이 사용되어서 여러 개 조건문을 지정하기 위해 사용합니다. elif 블록과 조건식은 여러 개 작성할 수 있습니다.

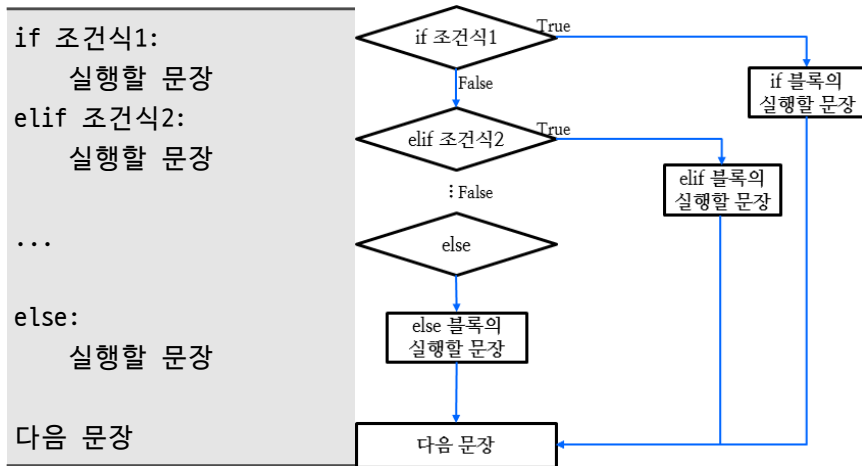


그림 3. if~elif~else 구문 순서도

구문에서...

- else 블록은 반드시 필요한 것은 아닙니다. 선택적으로 사용할 수 있습니다.

다음 코드는 점수에 따라 등급을 표시하는 예입니다.

```

score = 79
if score >= 90:
    print('A')
elif score >= 80:
    print('B')
elif score >= 70:
    print('C')
elif score >= 60:
    print('D')
else:
    print('F')
  
```

- 조건식의 결과는 반드시 True 또는 False 일 필요는 없습니다. 조건식의 결과가 0이면 False로 간주되고, 0 이외의 값이면 True로 간주됩니다.

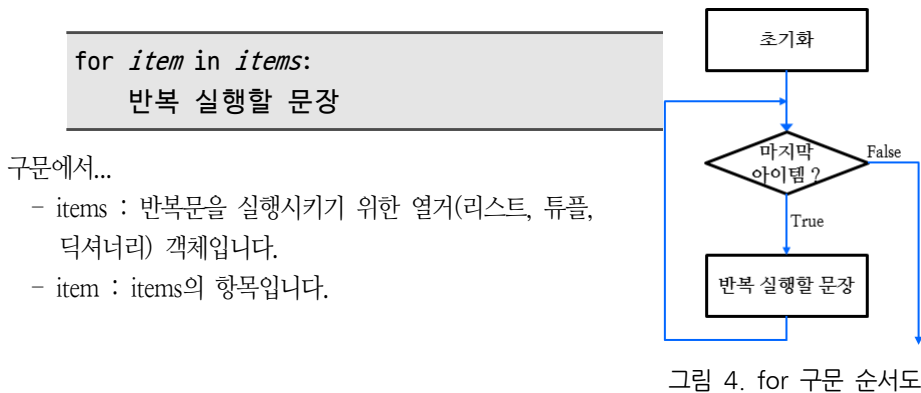


## 2. 반복문

반복문에는 for 문과 while 문이 있습니다.

### 2.1. for

for 문장은 동일한 코드를 반복 실행시키기 위해 사용합니다. for 문장은 주로 반복횟수가 정해져 있을 경우 사용합니다.



다음 코드는 정수값을 갖는 리스트를 이용하여 반복문을 실행시키는 예입니다.

```
for i in [0,1,2,3,4,5,6,7,8,9]:
    print(i, end="\t")
```

```
0  1  2  3  4  5  6  7  8  9
```

for 문장의 items 객체 위치에 `range(from_index, to_index, by)`<sup>34)</sup> 함수를 이용하여 반복문을 실행시킬수도 있습니다. `range()` 함수를 이용하면 인덱스 위주의 반복을 실행시킬 수 있습니다.

```
for i in range(0, 10):
    print(i, end="\t")
```

```
0  1  2  3  4  5  6  7  8  9
```

34) `from_index`는 포함하지만 `to_index`는 포함하지 않습니다.

- range() 함수의 from\_index가 생략되면 0부터 시작합니다. 그러므로 위의 range(0, 10)은 range(10)과 같습니다. range() 함수의 by 파라미터는 얼마씩 증가시킨 값을 갖게 할 것인지 결정합니다. range(0, 10, 3)으로 수정하면 출력되는 결과는 [0, 3, 6, 9]입니다.

다음 구문은 리스트를 반복문에 사용한 예입니다. 반복문을 리스트의 모든 항목을 처리합니다.

```
favorite_hobby = ['reading', 'fishing', 'shopping']
for hobby in favorite_hobby :
    print('%s is my favorite hobby' % hobby)
```

```
reading is my favorite hobby
fishing is my favorite hobby
shopping is my favorite hobby
```

다음 구문은 딕셔너리를 이용하여 반복문을 처리합니다. for 반복문은 딕셔너리의 items()는 딕셔너리의 아이템 하나(key와 value)를 튜플 형식으로 꺼내 for 절에 선언한 변수에 할당해줍니다.

```
color = {"apple": "green", "banana": "yellow", "cherry": "red" }
```

```
for c in color.items() :
    print(c)
```

```
('apple', 'green')
('banana', 'yellow')
('cherry', 'red')
```

```
for k, v in color.items() :
    print(k, v)
```

```
apple green
banana yellow
cherry red
```

```
for k in color.keys() :
    print(k)
```

```
apple
banana
```

```
cherry
```

```
for v in color.values() :  
    print(v)
```

```
green  
yellow  
red
```

## 2.2. while

while 문장은 조건식이 참일 동안 실행됩니다.

```
while 조건식 :  
    반복 실행할 문장
```

- 조건식 : 반복문을 실행시키기 위한 조건식입니다.

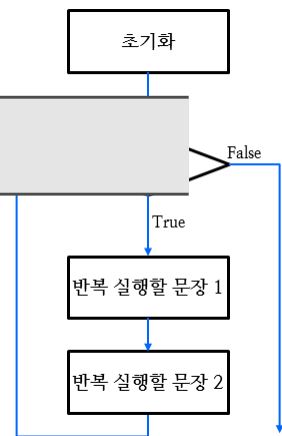


그림 5. while 구문 순서도

while 문은 반복 횟수가 정해져 있지 않을 경우에 사용합니다. while문이 시작되기 전에 while문의 조건식에 사용할 변수의 초기화를 선언하며, while문에는 조건식에 사용되는 변수를 증가시키거나 감소시키는 문장을 포함합니다. while 절의 조건은 반드시 bool 형식일 필요는 없습니다.<sup>35)</sup>

```
i = 0  
while i <= 10:  
    print(i, end=" ")  
    i = i + 1
```

```
0 1 2 3 4 5 6 7 8 9 10
```

```
num = 0
```

35) while 1: 표현식은 무한루프를 실행시킵니다.

```
while num <= 10:
    if num % 2 == 1:
        print(num, end=" ")
    num += 1
```

```
1 3 5 7 9
```

### 2.3. break와 continue

반복문 내에서 break를 만나면 break를 포함하는 반복문을 완전히 탈출합니다. 그러나 continue는 반복문 내에서 continue이후의 문장을 실행하지 않고 건너뜁니다.

다음 코드는 break 문과 continue 문의 예입니다.

```
num = 0
while num < 10:
    num += 1
    if num == 5:
        break
    print(num, end=" ")
```

```
1 2 3 4
```

```
num = 0
while num < 10:
    num += 1
    if num == 5:
        continue
    print(num, end=" ")
```

```
1 2 3 4 6 7 8 9 10
```

### 3. 중첩 루프

중첩 루프(Nested Loop)는 2차원 데이터를 다룰 때 사용합니다.

```
list_ = [[1,2,3,4,5], [11,12,13,14,15], [21,22,23,24,25]]
print(list_)
```

```
[[1, 2, 3, 4, 5], [11, 12, 13, 14, 15], [21, 22, 23, 24, 25]]
```

2차원 리스트에 대한 인덱싱은 ‘변수명[행][열]’ 형식으로 지정합니다. 위 데이터에서 13을 출력하길 원한다면 다음 코드처럼 작성합니다.

```
list_[1][2]
```

```
13
```

인덱스를 벗어난 데이터를 참조하면 예외가 발생합니다.

```
list_[3][2]
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-11-de1c74831325> in <module>()
----> 1 list_[3][2]

IndexError: list index out of range
```

2차원 리스트를 하나의 반복문으로 처리하면 리스트 안의 리스트를 받습니다.

```
for data in list_:
    print(data)
```

```
[1, 2, 3, 4, 5]
[11, 12, 13, 14, 15]
[21, 22, 23, 24, 25]
```

2차원 리스트에서 데이터를 조회하려면 중첩 루프를 이용해야 합니다.

```
for row in list_:
    for data in row:
        print(data, end=" ")
```

```
1 2 3 4 5 11 12 13 14 15 21 22 23 24 25
```

그러면 구구단을 출력하는 예를 중첩 루프를 이용해 만들어 보겠습니다.

```
for i in range(2, 10):
    for j in range(1, 10):
        print(i*j, end=" ")
    print()
```

```
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

위의 코드는 단이 가로방향으로 출력됩니다. 단이 세로방향으로 출력되게 하려면 반복문의 인덱스 실행 범위만 바꿔주면 됩니다.

```
for i in range(1, 10):
    for j in range(2, 10):
        print(i*j, end=" ")
    print()
```

```
2 3 4 5 6 7 8 9
4 6 8 10 12 14 16 18
6 9 12 15 18 21 24 27
8 12 16 20 24 28 32 36
10 15 20 25 30 35 40 45
12 18 24 30 36 42 48 54
14 21 28 35 42 49 56 63
16 24 32 40 48 56 64 72
18 27 36 45 54 63 72 81
```

i와 j 값을 출력해서 제대로 된 구구단을 출력해 보겠습니다. 다음 구문은 구구단을 출력하는 예입니다.

```
for i in range(1, 10):
    for j in range(2, 10):
        print("{}x{}={:>2}".format(j, i, i*j), end=' ')
    print()
```

```
2x1= 2 3x1= 3 4x1= 4 5x1= 5 6x1= 6 7x1= 7 8x1= 8 9x1= 9
2x2= 4 3x2= 6 4x2= 8 5x2=10 6x2=12 7x2=14 8x2=16 9x2=18
2x3= 6 3x3= 9 4x3=12 5x3=15 6x3=18 7x3=21 8x3=24 9x3=27
2x4= 8 3x4=12 4x4=16 5x4=20 6x4=24 7x4=28 8x4=32 9x4=36
2x5=10 3x5=15 4x5=20 5x5=25 6x5=30 7x5=35 8x5=40 9x5=45
2x6=12 3x6=18 4x6=24 5x6=30 6x6=36 7x6=42 8x6=48 9x6=54
2x7=14 3x7=21 4x7=28 5x7=35 6x7=42 7x7=49 8x7=56 9x7=63
2x8=16 3x8=24 4x8=32 5x8=40 6x8=48 7x8=56 8x8=64 9x8=72
2x9=18 3x9=27 4x9=36 5x9=45 6x9=54 7x9=63 8x9=72 9x9=81
```

- `print("{:>2}".format(i*j))`는 `i*j`를 2자리로 출력하며 오른쪽 정렬하라는 의미입니다.

## 4. 중첩 루프 탈출

break 문을 사용하면 프로그래머가 루프를 일찍 종료 할 수 있으며 continue 문을 사용하면 프로그래머가 루프의 다음 반복으로 일찍 이동할 수 있습니다. 현재 파이썬에서 break 와 continue는 가장 안쪽에 있는 루프에만 적용 할 수 있습니다.

다음 구문은 중첩 for 문의 안에서 break문을 사용한 예입니다.

```
for a in range(0, 3):
    for b in range(1, 3):
        if a==b:
            break
        print(a, b)
```

```
0 1
0 2
1 2
```

파이썬은 다양한 방법으로 중첩 루프에서의 break 또는 continue를 이용한 탈출을 지원합니다. 다음의 예는 break를 사용하지만 continue를 사용할 경우에도 동일한 개념이 적용됩니다.

### 4.1. 플래그 이용

프로그래머가 바깥 쪽 둘러싸는 루프의 다음 반복으로 이동하거나 한 번에 여러 루프를 종료하려는 경우에는 레이블이 지정된 break를 모방하는 일반적인 방법으로 플래그 값을 지정하는 것입니다.

다음 코드는 바깥 반복문에 break\_out\_of\_a 변수의 값을 False로 지정한 후 안쪽 반복문을 실행합니다. 만일 안쪽 반복문에서 바깥 반복문을 종료시키면 break\_out\_of\_a 변수의 값을 True로 합니다. 그리고 안쪽 반복문을 종료시키면 이후 조건문에 의해 바깥 반복문도 종료시킬 수 있습니다.

```
for a in range(0, 3):
    break_out_a = False
    for b in range(1, 3):
```



```

    if a==b:
        break_out_a = True
        break
    print(a, b)
if break_out_a:
    break;

```

```

0 1
0 2

```

## 4.2. 예외 이용

예외처리는 아직 우리가 배우지 않았습니다. 이 예제는 7장의 예외처리를 배우고 다시 보셔도 됩니다.

다음 코드는 바깥 반복문에서 안쪽 반복문을 실행시키기 위해 예외처리 코드를 작성했습니다.

```
class BreakOutOfALoop(Exception): pass
```

```

for a in range(0, 3):
    try:
        for b in range(1, 3):
            if a==b:
                raise BreakOutOfALoop
            print(a, b)
    except BreakOutOfALoop:
        break

```

```

0 1
0 2

```

- 안쪽 반복문에서 바깥 반복문을 종료시키려고 예외를 발생(raise)시킵니다.
- 그러면 안쪽 반복문에서 예외가 발생할 것이고 이것은 try 블록에서 예외가 발생하는 것을 의미합니다.
- try 블록에서 예외가 발생하면 이것을 처리하기 위해 제어는 except 블록으로 이동합니다.
- except 블록에서 break는 자신을 포함하는 반복문(여기서는 바깥 반복문)을 종료시킵니다.

### 4.3. 레이블을 갖는 반복문

레이블 및 `continue` 문에 대한 지원을 추가하면 기존 동작을 논리적으로 확장 할 수 있습니다. 레이블을 붙여서 사용하는 중첩 루프는 복잡한 코드의 가독성과 유연성을 향상시킵니다. 이러한 내용은 PEP 3136<sup>36)</sup>에 제안되었습니다. 그러나 파이썬은 제안서의 모든 내용을 구현하지 않았습니다. 파이썬은 이 절에서 설명하는 `as` 또는 `label`을 지원하지 않습니다. 이 절의 내용은 제안 내용이므로 참고용으로만 보시길 바랍니다. 여기에는 PEP 3136 제안서의 모든 내용을 담지 않았습니다. 더 자세한 내용은 PEP 3136 문서를 참고하세요.

PEP 3186문서에는 `for` 및 `while` 루프 구문 뒤에는 `as` 또는 `label` 키워드를 이용하여 반복문에 레이블을 지정하고 이 레이블은 `break` 또는 `continue`에서 종료시킬 반복문을 식별하는 데 사용할 수 있습니다.

```
for item in items [ (as | label) outer_label ] :
```

\* 파이썬은 이 구문을 실제로 지원하지는 않습니다.

`break` 또는 `continue` 문 뒤에는 중단 할 루프를 나타내는 선택적 레이블(식별자)이 옵니다.

```
break outer_label
```

\* 이 구문도 지원하지 않습니다.

다음은 `as` 키워드를 사용하는 예제입니다.

```
for a in a_list as a_loop:
    # ...
    for b in b_list as b_loop:
        # ...
        if condition_one(a, b):
            break b_loop # same as plain old break
        # ...
        if condition_two(a, b):
            break a_loop
        # ...
    # ...
```

36) <https://www.python.org/dev/peps/pep-3136/>

as 대신에 label을 사용할 수 있습니다.

```
for a in a_list label a_loop:
    # ...
    for b in b_list label b_loop:
        # ...
        if condition_one(a, b):
            break b_loop # same as plain old break
        # ...
        if condition_two(a, b):
            break a_loop
        # ...
    # ...
```

- break 또는 continue에 로컬 범위에 정의되지 않은 식별자를 사용하면 NameError가 발생합니다.

## 5. 연습문제

- 1) 숫자(양의 정수)를 입력받아 홀수인지 짝수인지를 판별하는 프로그램을 작성하세요. 양의 정수가 아니면 숫자를 다시 입력받아야 합니다.
- 2) 1부터 50까지 자연수 중에서 3의 배수의 총합을 출력하는 프로그램을 작성하세요.
- 3) 아래 패턴의 별을 출력하는 프로그램을 작성하세요.  
 - `print('*', end='')` 사용하면 줄 바꿈 없이 화면 출력이 가능합니다.

```
*****
```

그림 6. 줄 바꿈 없이 출력하기

- 4) 아래와 같은 패턴의 별(\*)을 출력하는 프로그램을 작성하세요(이중 루프 사용)

*****	*	*****	*	*****	*
*****	**	*****	**	*****	***
*****	***	*****	***	*****	*****
*****	****	**	****	**	*****
*****	*****	*	*****	*	*****

그림 7. 별 출력하기

정답 및 풀이

1)

```
while(True):
    num = eval(input('숫자(양의 정수)를 입력하세요. :'))
    if(isinstance(num,int) and num>=0):
        if num%2 == 0:
            print('짝수입니다.')
        else:
            print('홀수입니다.')
        break;
    else:
        print('양의 정수가 아닙니다.')
```

```
숫자(양의 정수)를 입력하세요. :0.5
양의 정수가 아닙니다.
숫자(양의 정수)를 입력하세요. :-4
양의 정수가 아닙니다.
숫자(양의 정수)를 입력하세요. :5
홀수입니다.
```

2)

```
sum = 0
for i in range(1,50):
    if i%3==0:
        sum = sum + i
    else:
        pass

print(sum)
```

408

3)

```
for i in range(0,5) :
    print('*', end='')

*****
```

4)

```
for i in range(0, 5) :
```

```

for j in range(0, 5) :
    print('*', end='')
print()

```

```

*****
*****
*****
*****
*****

```

```

for i in range(0, 5) :
    for j in range(0, i+1) :
        print('*', end='')
    print()

```

```

*
**
***
****
*****

```

```

for i in range(0, 5) :
    for j in range(i, 5) :
        print('*', end='')
    print()

```

```

*****
****
***
**
*

```

```

for i in range(0, 5) :
    for j in range(i+1, 5) :
        print(' ', end='')
    for j in range(0, i+1) :
        print('*', end='')
    print()

```

```

*
**
***
****
*****

```

```
for i in range(0, 5) :  
    for j in range(i+1, 5) :  
        print(' ', end='')  
    for j in range(0, (i*2)+1) :  
        print('*', end='')  
    print()
```

```
  *  
 ***  
*****  
*****  
*****
```







## 5장. 함수

## 1. 함수의 정의 및 사용

프로그래밍 언어에서 함수(function)는 수학에서의 함수의 개념과 비슷합니다. 함수는 어떤 입력값을 받아 다른 값을 출력하도록 미리 만들어져 있는 것을 의미합니다. 함수는 반복해서 사용한 코드들을 묶어 놓고 그것에 이름을 붙인 것입니다. 반복해서 사용할 코드는 함수를 이용하면 훨씬 구조적이고 간결한 코드를 작성할 수 있습니다.

함수를 사용하려면 먼저 함수가 정의(define)되어 있어야 합니다. 정의되어 있는 함수를 사용하려면 '함수명()' 형식으로 사용합니다. 한 쌍의 소괄호 안에는 함수가 실행되기 위해 필요한 값을 입력합니다.

함수는 입력 집합  $x$ 를 출력 집합  $f(x)$ 에 대응시킵니다.

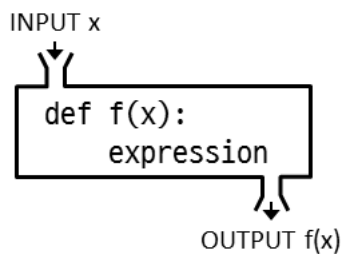


그림 1. 함수

### 1.1. 함수 정의하기

키워드 `def`는 함수를 정의합니다. `def`에는 함수 이름과 괄호로 묶인 매개변수(Parameter variable)<sup>37)</sup>의 목록이 따라 와야 합니다. 함수의 본문을 구성하는 명령문은 다음 행에서 시작하며 들여 쓰기 되어야 합니다.

```
def function_name([param1, param2, ...]):
    expression
```

구문에서...

- `def` : 함수를 정의하기 위해 사용하는 키워드입니다.

37) 매개변수(Parameter variable, 파라미터)는 함수가 실행될 때 함수 안에서 사용할 값을 받는 변수입니다. 인자라고 부르기도 합니다.

- *function\_name* : 함수의 이름입니다. 변수 이름 만드는 것처럼 함수를 구분하는 이름입니다. 함수 이름은 문자, 숫자, \_를 포함할 수 있으며, 숫자로 시작할 수 없습니다.
- *param1, param2, ...* : 함수의 매개변수(Parameter variable)입니다. 인자라고 부르기도 합니다. 함수가 실행될 때 필요로 하는 값을 받기 위해 사용합니다. 매개변수는 선택사항입니다.
- *expression* : 함수가 실행할 구문입니다. 반드시 들여쓰기가 되어 있어야 합니다.

## 1) 매개변수가 없는 함수 정의

함수 안에서 실행하는 값을 사용하지 않을 때 매개변수를 선언하지 않는 함수를 정의합니다. 함수를 만드는 것을 ‘함수를 정의(define)한다’라고 표현합니다.

다음 함수는 매개변수가 없는 함수입니다.

```
def my_hello():
    print("Hello World")
```

함수를 사용하는 것을 ‘함수를 호출(call)한다’라고 표현합니다. 다음코드는 앞에서 선언한 함수를 호출합니다. 함수를 호출할 때에는 반드시 함수명 뒤에 ( )를 붙여야 합니다.

```
my_hello()
```

```
Hello World
```

## 2) 매개변수가 있는 함수

함수가 실행하기 위해 어떤 값을 받아야 한다면 매개변수를 선언해야 합니다. 매개변수는 일반 변수와 동일한 사용법과 특징을 가지고 있습니다. 그러나 매개변수는 함수 안에서 사용할 값을 받는 변수입니다.

매개변수의 이름은 변수 이름 만드는 규칙에만 맞으면 자유롭게 선언할 수 있습니다. 함수 정의 시 ( ) 안에 선언한 이름으로 함수 안에서 사용할 수 있습니다. 매개변수는 여러 개 사용할 수 있습니다. 그러나 함수 호출 시 매개변수의 개수는 정의 시 개수와 일치<sup>38)</sup>해야 합니다.

38) 일반적으로는 함수 정의 시 매개변수와 호출 시 인수의 개수는 일치해야 합니다. 그러나 다양한 인자들에 대해 배우고 나면 반드시 일치하지 않을 수 있다는 것을 알게 됩니다.

다음 코드는 두 수를 입력받아 두 수의 합을 계산/출력하는 함수입니다.

```
def my_add(num1, num2):
    print(num1 + num2)
```

```
my_add(3, 5)
```

```
8
```

## 1.2. docstring

함수 본문의 첫 번째 문장에 문자열 리터럴을 포함할 수 있습니다. 이 문자열 리터럴은 함수의 설명서 문자열 또는 docstring<sup>39)</sup>입니다. docstring을 사용하여 온라인 또는 인쇄 된 문서를 자동으로 생성하거나 사용자가 코드를 대화식으로 검색 할 수 있게 해주는 도구가 있습니다. 우리가 작성하는 코드에 문서화 문자열을 포함시키는 것은 좋은 습관입니다. 문자열 리터럴을 여러 줄에 작성하려면 겹따옴표 3개(“”와 “”) 또는 홑따옴표 3개(“”와 “”)를 이용하면 됩니다.

```
def my_function():
    """함수의 첫 라인에 독스트링을
    포함시킬 수 있습니다.
    """
    pass
```

```
print(my_function.__doc__)
```

```
함수의 첫 라인에 독스트링을
포함시킬 수 있습니다.
```

- 문자열 스트링이 여러 개 일 경우 맨 처음의 문자열만 독스트링이 됩니다.
- 한 줄로 작성할 경우에는 겹따옴표 1개("와 ") 또는 홑따옴표 1개('와 ')를 이용할 수 있습니다.
- 독스트링은 함수의 설명서를 달아주는 역할을 합니다. 주석보다 더 많은 기능을 합니다.
- 독스트링은 함수 안에 선언할 수 있지만 파이썬 파일의 맨 위에 설정해 모듈의 설명서를 만들 수 있습니다.

39) <https://docs.python.org/3/tutorial/controlflow.html#tut-docstrings>

### 1.3. 함수 정의하고 호출하기 예

다음 코드는 피보나치수열<sup>40)</sup>을 출력하는 함수입니다.

```
def fibonacci(n):
    "n값 미만까지 피보나치수열을 출력합니다."
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

다음 코는 함수의 인수를 200을 주고 피보나치 수열을 2000 미만의 값 까지 출력합니다.

```
fibonacci(200)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144
```

이렇게 정의되어 있는 함수는 다음 사용 시 인수의 값만 바꾸면 결과를 얻을 수 있습니다. 다음 코드는 피보나치 수열을 2000 미만의 값 까지 출력합니다.

```
fibonacci(2000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

```
print(fibonacci.__doc__)
```

```
n값 미만까지 피보나치수열을 출력합니다.
```

- fibonacci.\_\_doc\_\_은 독스트링(docstring)을 출력해 줍니다.

### 1.4. 변수의 참조

40) 첫 번째 두 개 이후의 모든 숫자는 앞의 두 개를 합한 것입니다.

함수의 실행은 함수의 지역 변수에 사용되는 새로운 심볼 테이블을 도입합니다. 보다 정확하게는 함수의 모든 변수 할당은 로컬 심볼 테이블에 값을 저장합니다. 변수 참조는 먼저 로컬 심볼 테이블을 검색 한 다음, 둘러싸는 함수의 로컬 심볼 테이블에서 찾은 다음 전역 심볼 테이블에서 찾은 다음 마지막으로 내장 된 이름 테이블에서 찾습니다. 따라서 전역 변수는 전역 변수에 이름이 지정되지 않은 한 함수 내에서 직접 값을 할당 할 수는 있지만 참조 할 수는 있습니다.

다음 코드는 함수 내에서 전역변수를 참조하고 있습니다. func1() 함수 내에 global\_var 변수가 선언되어 있지 않으므로 로컬 심볼 테이블에서 값을 찾을 수 없습니다. 그러므로 전역 심볼 테이블에서 global\_var 변수를 찾아 출력합니다.

```
global_var = 100
```

```
def func1():
    print(global_var)
```

```
func1()
```

```
100
```

그러나 반대로 함수 안에 선언한 변수를 함수 밖에서 참조할 수 없습니다.

```
def func2():
    local_var = 200
    print(local_var)
```

```
func2()
```

```
200
```

```
print(local_var)
```

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

```
File "<stdin>", line 1, in <module>
    print(local_var)
```

```
NameError: name 'local_var' is not defined
```

## 1.5. 렉시컬(Lexical) 특성

파이썬은 함수를 어휘적(Lexically)으로 검색하여 변수가 로컬 변수인지 여부를 확인합니다. 만일 이름이 로컬에 없으면 이름이 전역으로 간주됩니다. 그러나 이름이 로컬에 있을 경우 로컬의 변수를 참조합니다.

다음 코드는 func1() 함수 안에서 global\_var 변수에 값을 할당하고 있습니다. 그런데 global\_var 변수가 전역 심볼 테이블에 있지만 인터프리터는 로컬 심볼 테이블에 새로운 변수를 추가하고 200이라는 값을 할당합니다. 그런데 #1의 print 문장을 실행하는 인터프리터는 로컬 심볼 테이블에서 global\_var 변수를 찾습니다. 그런데 #1이 실행되는 시점에는 아직 global\_var 변수는 초기화 되어 있지 않기 때문에 에러가 발생합니다.

```
global_var = 100
```

```
def func1():
    print(global_var) #1
    global_var = 200
    print(global_var) #2
```

```
func1()
```

Traceback (most recent call last):

```
File "<stdin>". line 1. in <module>
    func1()
```

```
File "<stdin>". line 2. in func1
    print(global_var)
```

UnboundLocalError: local variable 'global\_var' referenced before assignment

다음 그림은 로컬 심볼 테이블에 선언된 변수와 글로벌 심볼 테이블에 선언된 변수를 나타낸 것입니다. +는 공용(public), -는 전용(private)을 의미합니다.

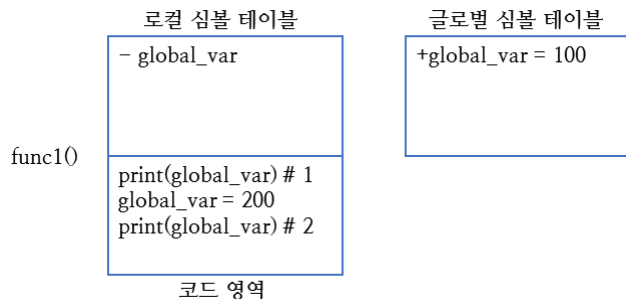


그림 2. 렉시컬 특성

## 1.6. 전역변수 수정

함수 내에서 전역변수를 참조하고 싶다면 함수 안에서 변수 선언 시 `global` 키워드로 지정해 주면 됩니다.

```
global_var = 100
```

```
def func1():
    global global_var
    print(global_var) # 1
    global_var = 200
    print(global_var) # 2
```

```
func1()
```

```
100
```

```
200
```

```
print(global_var)
```

```
200
```

`global` 키워드로 글로벌 심볼 테이블의 변수를 사용하도록 설정하면 로컬 심볼 테이블에 변수가 추가되지 않습니다.

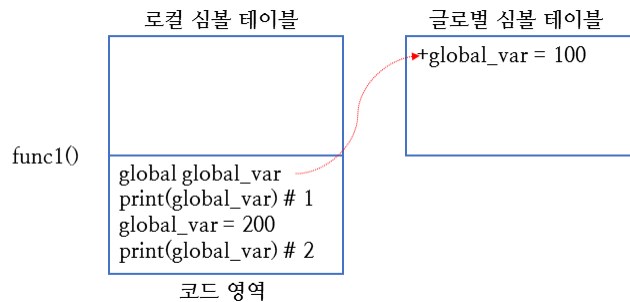


그림 3. 전역변수 참조

오류를 줄일 수 있는 가장 좋은 방법은 함수 내에서 변수를 초기화하기 전에 사용하지 않는 것입니다. 그리고 글로벌 심볼 테이블의 변수와 로컬 심볼 테이블에서 사용하는 변수의 이름을 명확히 하기 위해 함수 안의 지역변수는 `_`를 변수 앞에 붙여 주는 것도 좋은 방법입니다.



## 1.7. 값에 의한 호출

함수 호출에 대한 실제 매개 변수(인수)는 호출 될 때 호출 된 함수의 로컬 기호 테이블에 전달됩니다. 따라서 인수는 값에 의한 호출(call by value)을 사용하여 전달됩니다.(여기서 값은 항상 객체 참조가 아니라 객체 값입니다). 함수가 다른 함수를 호출하면 해당 호출에 대해 새 로컬 심볼 테이블이 만들어집니다.

```
var = 100
```

```
def func1(var):  
    var = 200  
    print(var)
```

```
func1(var)
```

```
200
```

```
print(var)
```

```
100
```

- 함수를 실행할 때의 var 변수는 200입니다. 그런데 함수 실행이 완료되고 나면 var 변수는 100을 출력합니다.

func1() 함수의 인수로 전달되는 전역변수 var의 값 100은 func1() 함수의 인수에 복사됩니다. 이후 함수 안에서 파라미터변수 var의 값을 200으로 변경하면 함수 영역 안의 var 변수의 값이 200으로 변경되므로 함수를 실행해도 전역변수 var의 값 100은 바뀌지 않습니다.

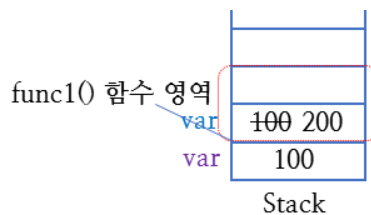


그림 4. 값에 의한 호출

그러면 함수의 인자로 전달된 변수의 값을 함수 안에서 변경할 때 호출하는 곳에서의 변수도 같이 변경되게 하려면 어떻게 해야 할까요?

## 1.8. 참조에 의한 호출

리스트 또는 튜플 등의 경우에는 참조 연산자(.)를 이용해 객체를 변경하면 참조의 호출(call by reference) 효과를 얻을 수 있습니다.

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

```
def func1(var):
    var.append(6)
    print(var)
```

```
func1(L)
```

```
[1, 2, 3, 4, 5, 6]
```

```
print(L)
```

```
[1, 2, 3, 4, 5, 6]
```

다음은 참조에 의한 호출의 이해를 돕기 위한 그림입니다. L과 변수 var는 같은 객체를 참조합니다. L이 var와 같은 데이터를 가리키고 있으므로 var의 데이터를 수정하면 L의 데이터를 수정하는 것과 같습니다.

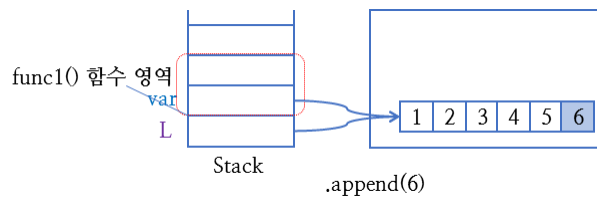


그림 5. 참조에 의한 호출

그러나 다음 예처럼 리스트를 함수의 인수에 전달하였더라도 할당 연산자(=)를 이용해 새로운 리스트를 할당하면 인수로 전달한 원본 리스트는 변경되지 않습니다.

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

```
def func1(var):
    var = 200
    print(var)
```

```
func1(L)
```

```
200
```

```
print(L)
```

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

## 1.9. 함수 이름 변경

함수 정의는 현재 심볼 테이블에 함수 이름을 지정합니다. 함수 이름의 값은 인터프리터가 사용자 정의 함수로 인식하는 유형입니다. 이 값은 다른 이름으로 지정 될 수 있으며, 이 이름을 함수로 사용할 수도 있습니다. 이것은 일반적인 이름 바꾸기 메커니즘으로 사용됩니다.

다음 코드는 앞에서 정의한 fibonacci 함수는 다음과 같이 f라는 이름으로 바꾸어 사용할 수 있습니다.

```
def fibonacci(n):
    "n값 미만까지 피보나치 수열을 출력합니다."
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

```
fibo = fibonacci
```

```
fibonacci(2000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

```
fibo(2000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

```
id(fibonacci)
```

```
178158720
```

```
id(fibo)
```

```
178158720
```

```
type(fibo)
```

```
function
```

- 함수 f와 fibonacci는 사실상 이름만 다를 뿐 같은 객체입니다.

다음 코드는 함수를 다른 이름으로 할당 할 때와 함수의 결과를 할당할 때를 비교합니다.

```
fibo1 = fibonacci
```

```
type(fibo1)
```

```
function
```

```
print(fibo1)
```

```
<function fibonacci at 0x00000000A9E7C80>
```

```
fibo1(2000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

- 위의 예는 함수를 다른 이름으로 할당합니다.
- fibo1은 함수입니다.

다음 코드는 리턴값을 갖지 않는 함수를 호출할 결과를 변수에 할당하는 예입니다. 다음 코드의 fibo2는 fibonacci() 함수가 아무것도 리턴하지 않기 때문에 아무 값도 가지고 있지 않는 값(None)이 됩니다.

```
fibo2 = fibonacci(2000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

```
type(fibo2)
```

```
NoneType
```

```
print(fibo2)
```

```
None
```

```
fibo2(2000)
```

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

```
File "<ipython-input-142-96dd13014279>". line 1. in <module>
    fibo2(2000)
```

```
TypeError: 'NoneType' object is not callable
```

- 위의 코드는 함수의 실행 결과를 할당합니다. fibonacci() 함수는 return 구문을 가지고 있지 않으므로 실행 결과는 아무것도 저장하지 않습니다.
- fibo2는 NoneType 값입니다. 그러므로 함수 형식으로 사용할 수 없습니다.
- 만일 fibonacci() 함수가 리턴 구문을 가지고 있다면 fibo2는 함수의 리턴값을 가집니다.

## 2. 리턴 값

함수의 리턴 값은 함수가 실행한 결과를 함수를 호출한 곳에 전달하기 위해 사용합니다.

```
def function_name(param) :
    # code
    return return_value
```

구문에서 :

- `return` : 함수의 결과값을 반환하기 위한 키워드입니다.
- `return_value` : 함수가 실행한 결과를 반환하는 값입니다. 변수이름을 사용할 수 있으며 파이선의 어떤 타입 이더라도 사용 가능합니다.

### 2.1. 리턴 값이 없는 함수

앞에서 선언한 `fibonacci()` 함수를 이용하여 리턴 값에 대하여 설명하겠습니다.

만일 여러분이 다음과 같이 `fibonacci()` 함수에 인자를 포함해 할당한다면 이 경우에는 함수를 실행한 결과가 할당되는 것입니다. 이 예에서는 `fibonacci()` 함수는 리턴 값이 없기 때문에 `f200` 변수에는 아무것도 저장되지 않을 것입니다.

```
f200 = fibonacci(200)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144
```

```
f200
```

```
f200(10)
```

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

```
File "<stdin>", line 1, in <module>
    f200(10)
```

```
TypeError: 'NoneType' object is not callable
```

fibonacci() 함수는 함수 안에서 값을 출력만 합니다. 만일 다음과 같이 print() 함수의 인자에서 fibonacci() 함수를 호출한다면 실행 결과에 None을 보게 됩니다. 이러한 실수는 종종 있게 됩니다. 화면에서 None을 보았다면 아무것도 리턴하지 않는 함수를 print() 등의 함수로 출력한 것입니다.

```
fibonacci(0)
```

```
print(fibonacci(0))
```

```
None
```

사실 함수에서 데이터를 출력하는 것은 바람직하지 못합니다. 함수는 어떤 기능을 실행하는 것이기 때문에 데이터를 출력하는 코드는 함수를 호출하는 곳에서 하는 것이 바람직합니다<sup>41)</sup>. 다음은 함수가 리턴 값을 갖는 경우에 대한 설명입니다.

## 2.2. 리턴 값이 있는 함수

이제 fibonacci() 함수를 조금 수정해 보겠습니다. fibonacci2() 함수는 결과를 저장할 지역 변수를 선언하고 그곳에 함수의 실행 결과를 저장합니다. 그리고 함수의 마지막 문장에 return 키워드를 이용하여 함수의 실행결과를 리턴합니다.

```
def fibonacci2(n):
    """n값 미만까지 피보나치수열을 출력합니다."""
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

```
f100 = fibonacci2(100)
```

```
f100
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

41) 반드시 그래야 하는 것은 아닙니다. 함수가 비즈니스로직을 구현했다면 프리젠테이션 로직은 함수 안에 두지 않고 함수를 호출하는 곳에 두자는 것입니다. print() 함수로 출력하는 코드는 프리젠테이션 로직에 해당하기 때문입니다.

```
print(fibonacci2(100))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

```
print(f100)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

### 2.3. 여러 개 값 리턴

파이썬은 여러 개 값을 리턴할 수 있습니다. 여러 개 값을 리턴하면 그 값들은 튜플에 저장되어 리턴됩니다. 함수가 여러 개 값을 리턴하면 하나의 튜플 변수에 값을 저장할 수 있고, 리턴하는 값의 개수만큼 변수를 선언하여 각각의 변수에 리턴 값이 저장되도록 할 수 있습니다.

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

```
a = swap(1,2)
type(a)
```

```
tuple
```

```
print(a)
```

```
(2, 1)
```

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

```
2 1
```

사실 위의 코드는 이상할 것이 없습니다. 파이썬은 다음 구문처럼 여러 개의 변수를 한 번에 초기화 할 수 있습니다.

```
a, b = (1,2)
print(a)
```

```
1
```

```
print(b)
```

```
2
```

```
c, d = [3,4]  
print(c)
```

```
3
```

```
print(d)
```

```
4
```

```
e, f = {5,6}  
print(e)
```

```
5
```

```
f
```

```
6
```

```
g, h = {'num': 100, 'msg':'hello'}  
print(g)
```

```
num
```

```
print(h)
```

```
msg
```



## 3. 함수 파라미터

파이썬은 인수의 개수가 다르게 적용되도록 함수를 정의할 수 있습니다. 이것을 가변 인수라고 하는데, 함수의 가변 인수에 결합될 수 있는 3가지 형식이 존재합니다.

### 3.1. 기본 인수 값

함수 인수 정의 시 가장 유용한 형식은 하나 이상의 인수에 대한 기본값을 지정하는 것입니다. 이렇게 하면 함수를 정의할 때에 허용하도록 정의 인수의 수 보다 적은 인수로 호출할 수 있는 함수가 생성됩니다.

다음 코드는 기본 값을 갖는 함수를 정의한 예입니다.

```
def make_url(ip, port=80):
    return "http://{}:{ {}".format(ip, port)
```

다음 코드는 위의 함수를 사용하는 예입니다.

```
make_url("localhost")
```

```
'http://localhost:80'
```

```
make_url("localhost", 8080)
```

```
'http://localhost:8080'
```

```
make_url("coderby")
```

```
'http://coderby:80'
```

- 기본 인수값을 갖는 port 인수는 값을 지정하지 않아도 됩니다.

### 3.2. 디폴트 변수

함수의 인자가 기본값으로 변수이름을 가질 수 있습니다. 이때 기본값은 함수가 정의되는 지점에 평가됩니다.

다음 함수는 인수인 기본값이 변수입니다. 함수가 정의될 때 func2() 함수의 arg 변수에 전달되는 값은 5입니다. 나중에 i 값이 바뀌더라도 함수를 호출할 때에 arg 인수를 지정하지 않는다면 arg의 기본값은 5가 됩니다.

```
i = 5
```

```
def func2(arg=i):
    print(arg)
```

```
i = 6
```

```
func2()
```

```
5
```

기본값은 한 번만 평가됩니다. 이것은 디폴트가 리스트, 딕셔너리 또는 대부분의 클래스의 인스턴스와 같은 변경 가능한 객체 일 때 차이를 만듭니다. 예를 들어, 다음 함수는 후속 호출에서 전달 된 인수를 누적합니다.

```
def func3(a, L=[]):
    L.append(a)
    return L
```

```
print(func3(1))
```

```
[1]
```

```
print(func3(2))
```

```
[1, 2]
```

```
print(func3(3))
```

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

만일 기본값을 함수들 호출 사이에 공유하지 않으려면 다음과 같은 형식으로 함수를 작성할 수 있습니다.

```
def func4(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

```
print(func4(1))
```

```
[1]
```

```
print(func4(2))
```

```
[2]
```

### 3.3. 키워드 인자

함수를 호출 할 때에 kwarg=value 형식의 인수를 사용하여 호출 할 수도 있습니다.

함수를 정의할 때에 기본값을 지정한 파라미터들은 호출 시 필수 인자는 아닙니다. 변수명만 있는 파라미터는 함수 호출 시 반드시 인수를 지정해야 합니다.

```
def function_name(변수명1, 변수명2=기본값, ...):
    pass
```

필수 인자                  선택 인자

그림 6. 필수 인자와 선택 인자

함수를 호출할 때 함수의 인수는 파라미터 이름을 가질 수 있습니다. 인수에 파라미터 이름을 지정해 선택적으로 사용하는 인자를 키워드 인자(keyword argument)라고 부릅니다.

```
function_name(값1, 변수명=값2, ...)
```

순서 인자                  키워드 인자  
(positional argument)      (keyword argument)

그림 7. 순서 인자와 키워드 인자

앞에서 정의했던 func4() 함수를 이용해 키워드 인자에 대해 설명하겠습니다.

```
def func4(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

```
list_ = []
```

다음 함수 호출처럼 인수를 선언한 순서대로 포함시킬거라면 인수가 파라미터이름을 가질 필요가 없습니다.

```
func4(10, list_)
```

```
[10]
```

당연히 인수가 순서대로 입력되더라도 파라미터 이름을 가질 수 있습니다.

```
func4(20, L=list_)
```

```
[10, 20]
```

```
func4(30, L=list_)
```

```
[10, 20, 30]
```

그런데, 파라미터 이름을 포함하면 인수의 순서를 바꿔 함수를 호출할 수 있습니다.

```
func4(L=list_, a=40)
```

```
[10, 20, 30, 40]
```

기본값을 갖는 파라미터는 함수 호출 시 인수를 생략할 수 있습니다.

```
func4(a=60)
```

```
[60]
```

다음의 경우들은 에러를 발생시킵니다.

필수 인자를 포함해야 하는데 그렇지 않은 경우 에러가 발생합니다.

```
func4()
```

Traceback (most recent call last):

```
File "<ipython-input-158-0e6ad11a93c1>", line 1, in <module>
    func4()
```

TypeError: func4() missing 1 required positional argument: 'a'

필수 인자가 파라미터 이름을 지정하지 않은 것과 파라미터 이름을 지정한 것을 동시에 사용할 수 없습니다.

```
func4(50, a=60)
```

Traceback (most recent call last):

```
File "<ipython-input-159-4a36985e9362>", line 1, in <module>
    func4(50, a=60)
```

TypeError: func4() got multiple values for argument 'a'

함수 정의 시 존재하지 않는 파라미터 이름을 사용할 수 없습니다.

```
func4(b=70)
```

Traceback (most recent call last):

```
File "<ipython-input-161-e445497b0d28>", line 1, in <module>
    func4(b=70)
```

TypeError: func4() got an unexpected keyword argument 'b'

키워드 인자는 순서 인자 뒤에 와야 합니다.

```
func4(L=list_, 70)
```

```
File "<ipython-input-162-79b4f6d43508>", line 1
```

```
    func4(L=list , 70)
```

```
            ^
```

```
SyntaxError: positional argument follows keyword argument
```

### 3.4. 튜플인자를 이용한 가변인자 설정

가변인자(Arbitrary Argument Lists<sup>42)</sup>)는 임의의 수의 인수로 함수를 호출 할 수 있도록 지정하는 것입니다. 함수의 매개 변수 이름 앞에 \*를 붙이면 함수 호출 시 인수들이 튜플에 저장되어 전달됩니다. 가변 인수 수 앞에 0 개 이상의 일반 인수가 올 수 있습니다.

```
def add(*args):
    sum = 0
    for num in args:
        sum = sum + num
    return sum
```

```
add(10, 20)
```

```
30
```

```
add(10, 20, 30)
```

```
60
```

```
add(10, 20, 30, 40)
```

```
100
```

일반적으로 이 가변 인수는 함수에 전달되는 나머지 모든 입력 인수를 스쿠핑<sup>43)</sup>하기 때문에 형식 인자 목록의 마지막에 옵니다. \*args 매개 변수 다음에 나오는 형식적 매개 변수

42) Variable Arguments라고 부르기도 합니다.

43) 'scoop up' 또는 'scooping'이라고 하며 '떠내다' 또는 '퍼내다'의 의미를 갖습니다.

는 '키워드 전용' 인수입니다. 그러므로 위치 인수는 튜플 인수 뒤에 올 수 없습니다.

```
def concat(*args, sep="/"):
    return sep.join(args)
```

```
concat("earth", "mars", "venus")
```

```
'earth/mars/venus'
```

```
concat("earth", "mars", "venus", sep=".")
```

```
'earth.mars.venus'
```

- join() 함수는 문자열 리스트를 구분자를 이용하여 연결하는 함수입니다.

### 3.5. 딕셔너리 인자

\*\*name 형식의 최종 형식 매개 변수가 있으면 형식 매개 변수에 해당하는 것을 제외하고 모든 키워드 인수가 들어있는 딕셔너리를 받습니다.

```
def func5(**data):
    for item in data.items():
        print(item)
```

```
func5(name="KilDong", age=30, address="서울시 강남구")
```

```
('name', 'KilDong')
```

```
('age', 30)
```

```
('address', '서울시 강남구')
```

딕셔너리 인자는 형식 매개 변수 목록 이외의 위치 인수를 포함하는 튜플을 받는 \*name 형식의 형식 매개 변수와 결합 될 수 있습니다. (\*name은 \*\*name 앞에 나와야합니다.)

예를 들어, 다음과 같은 함수를 정의하면 :

```
def func6(a, *b, **c):
    print(a)
    print(b)
    print(c)
```

이 함수는 다음처럼 호출할 수 있습니다.

```
func6(10, 1, 2, 3, name='KilDong', age=20)
```

```
10
(1, 2, 3)
{'name': 'KilDong', 'age': 20}
```

- 키워드 인수가 인쇄되는 순서는 함수 호출에서 제공된 순서와 일치하도록 보장됩니다.

- 딕셔너리 인자 또는 튜플 인자는 키워드 인수로 사용할 수 없습니다. 즉 함수 정의 시 초기값을 지정할 수 없습니다.

*Instructor Note: 함수 선언 시 파라미터의 순서는 어떻게 되나요?*

함수 선언시 파라미터 순서는 순서 인자, 튜플 인자, 키워드 인자, 딕셔너리 인자 순으로 가능합니다.

```
def func(a, b, c, *d, e=10, **f) :
    print(a, b, c, d, e, f)
```

```
func(10, 20, 30, 4, 5, e=20, name="JinKyoung", age=30)
```

```
10 20 30 (4, 5) 20 {'name': 'JinKyoung', 'age': 30}
```

### 3.6. 인수 언패킹

#### 1) 튜플 언패킹

인수가 이미 목록이나 튜플에 있지만 별도의 위치 인수가 필요한 함수 호출에 대해 압축을 풀어야하는 경우에는 반대 상황이 발생합니다. 예를 들어 내장 된 range() 함수는 별도의 start 및 stop 인수를 필요로 합니다. 별도로 사용할 수 없는 경우에는 \* 연산자로 함수 호출을 작성하여 인수를 목록 또는 튜플에서 언 패킹(Unpacking)합니다.

```
def add(*args):
    sum = 0
    for num in args:
        sum = sum + num
    return sum
```



```
numbers = (1,2,3,4,5,6,7,8,9,10)
```

numbers 변수를 add() 함수의 인자로 직접 넣을 수 없습니다.

```
add(numbers)
```

Traceback (most recent call last):

```
File "<ipython-input-183-9b08f8078530>", line 1, in <module>
    add(numbers)
```

```
File "<ipython-input-181-5c4df42c92eb>", line 4, in add
    sum = sum + num
```

TypeError: unsupported operand type(s) for +: 'int' and 'tuple'

이럴 경우에 다음과 같이 변수 앞에 \*를 붙여 튜플을 언패킹 하여 전달합니다.

```
add(*numbers)
```

```
55
```

## 2) 딕셔너리 언패킹

동일한 방식으로 딕셔너리 데이터는 \*\* 연산자로 키워드 인수를 전달할 수 있습니다.

다음 코드는 앞에서 정의했던 딕셔너리 인자를 받는 func5() 함수입니다.

```
def func5(**data):
    for item in data.items():
        print(item)
```

함수의 인자로 전달할 딕셔너리를 정의합니다.

```
custInfo = {"name":"KilDong", "age":30, "address":"서울시 강남구"}
```

딕셔너리 인자에 딕셔너리 데이터를 직접 넣을 수 없습니다.

```
func5(custInfo)
```

Traceback (most recent call last):

```
File "<ipython-input-189-61d1b1e9a82c>", line 1, in <module>  
    func5(custInfo)
```

TypeError: func5() takes 0 positional arguments but 1 was given

딕셔너리 데이터를 이용해 함수의 인수를 한꺼번에 전달하고 싶다면 \*\*을 이용하세요.

```
func5(**custInfo)
```

```
('name', 'KilDong')  
( 'age', 30)  
( 'address', '서울시 강남구')
```

## 4. 람다 식

### 4.1. Lambda Expressions

작은 익명 함수는 lambda 키워드로 만들 수 있습니다.

```
lambda variable_define : statement
```

구문에서...

- variable\_define : 함수의 인수를 정의합니다.
- statement : 함수가 실행할 문장을 작성합니다. 한 문장만 작성할 수 있습니다. return 구문이 없어도 statement의 결과를 반환합니다.

람다식을 설명하기 전에 기존의 함수를 사용해 봅니다. 다음 코드는 두 수를 더하는 함수입니다.

```
def add(a, b):
    return a+b
```

```
add(1,2)
```

```
3
```

위 코드는 다음처럼 람다식을 이용해서 간결하게 표현할 수 있습니다.

```
add2 = lambda a, b : a+b
```

```
add2(1,2)
```

```
3
```

람다식은 함수 객체가 필요한 곳이면 어디서든지 사용할 수 있습니다. 람다식은 한 개의 문장(표현식)만 작성할 수 있습니다. 사실 람다식은 의미론적으로 정상적인 함수 정의를 위한 문법적 단축 구문 일뿐입니다. 중첩 된 함수 정의와 마찬가지로 람다 함수는 포함 된 범위(scope)의 변수들을 참조 할 수 있습니다.

## 4.2. 리턴문에 람다식 사용

리턴문에 람다식을 사용하기 전에 함수의 리턴문에 함수를 사용할 수 있는지 알아봐야 합니다. 파이썬에서 함수 또한 객체이므로 리턴문에 함수를 사용할 수 있습니다.

다음 코드는 리턴문에 함수의 이름이 있습니다. 리턴문의 함수는 반드시 지역함수 일 필요는 없습니다.

```
def make_incrementor1(n):  
    def add(a):  
        return a+n  
    return add
```

```
f2 = make_incrementor1(10)
```

```
f2(5)
```

```
15
```

위의 구문은 다음처럼 리턴문에 람다식을 사용하여 표현할 수 있습니다.

```
def make_incrementor2(n):  
    return lambda x: x+n
```

```
f = make_incrementor2(10)
```

```
f(5)
```

```
15
```

- 위의 예제는 람다 표현식 반환합니다. 함수의 실행결과는 새로운 함수를 정의합니다.

### 4.3. 함수 인수에 람다식 사용

다른 용도는 작은 함수를 인수로 전달하는 것입니다.

```
pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
pairs.sort(key=lambda pair: pair[1])
pairs
```

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

다음 구문처럼 람다 함수는 다른 함수의 인수로 전달할 때 사용할 수 있습니다.

```
def apply(data, fun=None) :
    if fun==None :
        return data
    else :
        return fun(data)
```

```
apply(10, lambda x : x*x)
```

```
100
```

```
apply(10, lambda x : 2*x)
```

```
20
```

```
apply(10, lambda x : x)
```

```
10
```

### 4.4. 함수 인수

함수를 호출할 때 인수로 다른 함수가 실행하고 반환한 결과를 전달할 수 있습니다. 이것은 당연한 것입니다. 그런데 파이썬은 함수의 인수로 함수객체를 전달할 수도 있습니다.

다음 예를 보겠습니다. 만일 데이터에서 짝수만 뽑아내는 함수를 구현해야 한다고 생각해 보겠습니다.

첫 번째는 데이터에서 짝수만 뽑아 반환하는 함수를 만들 수 있습니다. 다음은 그런 예입니다.

```
data = [1, 2, 3, 4, 5]
def odd(data):
    result = []
    for d in data:
        if d%2 == 0:
            result.append(d)
    return result
```

```
odd(data)
```

```
[2, 4]
```

다음은 전달받은 데이터에서 짝수일 경우에 True를 리턴하는 함수를 구현하고 filter() 함수에 데이터와 짝수일 경우에 True를 리턴하는 함수를 전달하는 예입니다.

```
data = [1, 2, 3, 4, 5]
def odd2(data):
    if data%2 == 0:
        return True
    else:
        return False
```

```
r = filter(odd2, data)
type(r)
```

```
filter
```

```
list(r)
```

```
[2, 4]
```

위의 코드는 람다식으로 다음처럼 간단하게 작성할 수 있습니다.

```
data = [1, 2, 3, 4, 5]
list(filter(lambda x: x%2==1, data))
```

```
[2, 4]
```

## 5. 파이썬 내장 함수

파이썬 내장함수는 import 하지 않고 즉시 사용 가능한 함수들을 말합니다. 내장 함수명은 일종의 키워드로 간주되므로 식별자로 사용하는 것은 피하여야 합니다.

표 1. 입출력 관련 함수

함수명	기능
print(x)	객체를 문자열로 표시한다.
input([prompt])	사용자 입력을 문자열로 반환한다.
help([x])	x에 대한 도움말을 출력한다.
globals()	전역 변수의 리스트를 반환한다.
locals() 또는 vars()	지역 변수의 리스트를 반환한다. __dict__ 어트리뷰트 <sup>44)</sup> 를 반환한다.
del(x) 혹은 del x	객체를 변수 공간에서 삭제한다.
eval(expr)	값을 구한다.
exec(obj)	파이썬 명령을 실행시킨다.
open(filename[,mode]))	파일을 연다

표 2. 기본 자료형의 생성과 변환함수

함수명	기능
object()	새로운 object (모든 객체의 base)를 생성한다.
bool(obj)	객체의 논리값을 반환한다.
int(obj)	문자열 형태의 숫자나 실수를 정수로 변환한다.
float(obj)	문자열 형태의 숫자나 정수를 실수로 변환한다.
complex(re [, img])	문자열이나 주어진 숫자로 복소수를 생성한다.

44) 객체의 내부 변수가 저장된 딕셔너리입니다.

표 3. 기본 자료형의 정보를 얻는 함수

함수명	기능
type(obj)	객체의 형을 반환한다.
dir(obj)	객체가 가진 함수와 변수들을 리스트 형태로 반환한다.
repr(obj) ascii(obj)	eval()함수로 다시 객체를 복원할 수 있는 문자열 생성 repr()과 유사하나 non-ascii문자는 escape한다.
id(obj)	객체의 고유번호(int형)을 반환한다.
hash(obj)	객체의 해시값(int형)을 반환. (같은 값이면 해시도 같다.)
chr(num) ord(str)	ASCII 값을 문자로 반환 한 문자의 ASCII 값을 반환
isinstance(obj, className)	객체가 클래스의 인스턴스인지를 판단한다.
issubclass(class, classinfo)	class가 classinfo의 서브클래스 일때 True 반환

표 4. 열거형 정보를 얻는 함수

함수명	기능
len(seq)	시퀀스형을 받아서 그 길이를 반환한다.
iter(obj [,sentinel] ) next(iterator)	객체의 이터레이터(iterator)를 반환한다. 이터레이터의 현재 요소를 반환하고 포인터를 하나 넘긴다.
enumerate(iterable, start=0)	이터러블에서 enumerate 형을 반환한다. 입력값으로 시퀀스자료형(리스트, 튜플, 문자열)을 입력을 받는다.
sorted(iterable[,key][,reverse])	정렬된 *리스트*를 반환
reversed(seq)	역순으로 된 *iterator*를 반환한다.
filter(func, iterable)	iterable의 각 요소 중 func()의 반환 값이 참인 것만을 묶어서 이터레이터로 반환.
map(func, iterable)	iterable의 각 요소를 func()의 반환값으로 매핑해서 이터레이터로 반환.

- enumerate() 함수는 인덱스와 아이템을 하나씩 튜플 형식으로 리턴합니다.
- filter() 함수는 iterable 객체의 아이템들 중에서 func() 함수의 결과가 True인 경우의 아이템들만 묶어서 반환합니다.



표 5. 산술/논리 연산과 관련된 함수

함수명	기능
hex(n) oct(n) bin(n)	정수 n의 16진수 값을 구해서 '문자열'로 반환한다. 정수 n의 8진수 값을 구해서 '문자열'로 반환한다. 정수 n의 2진수 값을 구해서 '문자열'로 반환한다.
abs(n)	절대값을 구한다. 복소수의 경우 크기를 구한다.
pow(x,y[,z])	거듭제곱을 구한다. pow(x,y)은 $x**y$ 와 같다.
divmod(a,b)	a를 b로 나눈 (몫, 나머지)를 구한다. 튜플 반환.
all(iterable) any(iterable)	iterable의 모든 요소가 True 일 경우 True를 반환. iterable의 하나 이상의 요소가 True 일 경우 True를 반환.
max(iterable) max(arg1, arg2, ...)	최대값을 구한다.
min(iterable) min(arg1, arg2, ...)	최소값을 구한다.
round()	반올림을 한다.

## 6. 연습문제

---

- 1) 함수의 인자로 리스트를 받은 후 리스트 내에 있는 모든 정수 값에 대한 최댓값과 최솟값을 리턴하는 함수를 작성하세요.

- `def get_max_min(data_list):`

- 2) 체질량 지수 (Body Mass Index, BMI)는 체중과 키를 이용해 인간의 비만도를 나타내는 지수로 아래의 수식에 의해 계산됩니다. (체중은 kg, 키는 m)

-  $BMI = \text{체중(kg)} / \text{신장(m)}^2$

BMI < 18.5, 마른체형

18.5 ≤ BMI < 25.0, 표준

25.0 ≤ BMI < 30.0, 비만

BMI ≥ 30.0, 고도 비만

- 함수의 인자로 체중(kg)과 신장(cm)을 받은 후 BMI 값에 따라 '마른체형', '표준', '비만', '고도 비만' 중 하나의 상태를 출력하는 함수를 작성해 보세요.

- 3) 직각 삼각형의 밑변과 높이를 입력 받은 후 삼각형의 면적과 둘레를 계산하는 함수를 작성하세요.

- 리턴값은 면적과 둘레를 동시에 리턴하게 작성하세요.

- `def get_triangle(width, height)`

- 4) 함수의 인자로 시작과 끝 숫자가 주어질 때 시작부터 끝까지의 모든 정수 값의 합을 리턴하는 함수를 작성하세요. (시작 값과 끝 값을 포함)

- `def mysum2(start, end):`

- 5) 함수의 인자로 문자열을 포함하는 리스트가 입력될 때 각 문자열의 첫 세 글자만으로 구성된 리스트를 리턴하는 함수를 작성하세요.

- 예를 들어 함수의 입력으로 ['Seoul', 'Daegu', 'Kwangju', 'Jeju']가 입력될 때 함수의 리턴 값은 ['Seo', 'Dae', 'Kwa', 'Jeu'] 입니다.

- `def get_abbrs(list):`

정답 및 풀이

1)

```
def get_max_min(data) :  
    return (min(data), max(data))
```

```
data_list = [3,6,2,7,9,10,8]  
get_max_min(data_list)
```

(2, 10)

2)

```
def get_bmi(weight, height) :  
    bmi = weight / pow(height,2)  
    if(bmi < 18.5) :  
        return "마른체형"  
    elif(18.5 <= bmi < 25.0) :  
        return "표준"  
    elif(25.0 <= bmi < 30.0) :  
        return "비만"  
    else :  
        return "고도 비만"
```

```
get_bmi(69, 1.72)
```

'표준'

```
get_bmi(80, 1.72)
```

'비만'

```
get_bmi(90, 1.72)
```

'고도 비만'

```
get_bmi(50, 1.72)
```

'마른체형'

3)

```
import math
```

```
def get_triangle(width, height):  
    area = (width * height) / 2  
    perimeter = width + height + math.sqrt(pow(width,2) + pow(height,2))  
    return (area, perimeter)
```

```
get_triangle(3,4)
```

```
(6.0, 12.0)
```

4)

```
def mysum2(start, end) :  
    sum = 0  
    for num in range(start, end+1) :  
        sum = sum + num  
    return sum
```

```
mysum2(1,2)
```

```
3
```

```
mysum2(1,100)
```

```
5050
```

5)

```
def get_abbrs(list):  
    for i in range(0, len(list)) :  
        list[i] = list[i][:3]  
    return list
```

```
list_data = ['Seoul', 'Daegu', 'Kwangju', 'Jeju']  
get_abbrs(list_data)
```

```
['Seo', 'Dae', 'Kwa', 'Jej']
```

## 6장. 모듈과 패키지

## 1. 모듈 사용하기

### 1.1. 모듈

파이썬 인터프리터를 종료하고 다시 입력하면, 함수 및 변수 정의가 사라집니다. 따라서 다소 긴 프로그램을 작성하려면 텍스트 편집기를 사용하여 인터프리터의 입력을 준비하고 대신 해당 파일을 입력으로 실행하는 것이 좋습니다. 이것을 스크립트 작성이라 고합니다. 프로그램이 길어질수록 쉽게 유지 보수 할 수 있도록 여러 파일로 분할 할 수 있습니다. 또한 각 프로그램의 함수 또는 변수 정의를 복사하지 않고 여러 프로그램에서 이미 작성한 편리한 기능들을 사용할 수도 있습니다. 이것을 지원하기 위해 파이썬은 함수 또는 변수의 정의를 파일에 넣고 스크립트 또는 인터프리터의 대화형 인스턴스에서 사용하는 방법을 가지고 있습니다. 이러한 파일을 모듈이라고합니다.

모듈의 정의를 다른 모듈이나 기본 모듈(최상위 레벨과 계산기 모드에서 실행되는 스크립트에서 액세스 할 수 있는 변수의 모음)로 가져올 수 있습니다.

모듈은 파이썬 정의와 문장을 담고 있는 파일입니다. 파일 이름은 접미어 .py가 추가 된 모듈 이름입니다. 모듈 내에서 모듈의 이름(문자열)은 전역 변수 `__name__`의 값으로 사용할 수 있습니다.

### 1.2. 표준 모듈

파이썬은 별도의 문서인 파이썬 라이브러리 레퍼런스(Python Library Reference)<sup>45)</sup>에서 설명하는 표준 모듈 라이브러리를 제공합니다. 일부 모듈은 인터프리터에 내장되어 있습니다. 이것들은 언어의 핵심 부분은 아니지만 효율성이나 시스템 호출과 같은 운영 체제 기본 요소에 대한 액세스를 제공하기 위해 내장되어 있는 작업에 대한 액세스를 제공합니다. 이러한 모듈 세트는 기본 플랫폼에 종속 된 구성 옵션입니다. 예를 들어, winreg 모듈은 Windows 시스템에서만 제공됩니다. 그러므로 특정 모듈은 사용에 주의를 기울일 필요가 있습니다. sys는 모든 파이썬 인터프리터에 내장되어 있습니다. 변수 sys.ps1 및 sys.ps2는 기본 및 보조 프롬프트로 사용되는 문자열을 정의합니다.

파이썬은 여러 가지 모듈을 제공합니다. 문자열(string), 날짜(date), 시간(time), 수학(math), 분수(fractions), 랜덤(random), 파일(file), sqlite3, os, sys, threading, unittest, xml, email,

45) <https://docs.python.org/3/library/index.html>

http 등 200여개의 다양한 모듈이 존재합니다.

### 1.3. 모듈 import 방법

아래의 코드는 time 내장 모듈의 ctime 함수를 사용하기 위한 모듈 import 방법들입니다.

#### 1) import

다음은 일반적인 import 구문입니다.

```
import time
```

```
time.ctime()
```

```
'Mon Dec 25 22:49:33 2017'
```

- 이 코드에서 import 구문은 현재 심볼 테이블에 time 모듈에 정의된 함수의 이름을 직접 입력하지 않습니다. 모듈 이름만 입력하면 됩니다. 모듈 안의 함수들은 모듈 이름을 붙여 사용 할 수 있습니다.

모듈은 다른 모듈을 import 할 수 있습니다. 일반적으로 모듈(또는 스크립트)의 시작 부분에 모든 import 문을 배치 할 필요는 없습니다. import 된 모듈 이름은 import 모듈의 전역 심볼 테이블에 배치됩니다.

#### 2) from A import B

다음은 from A import B 구문입니다. A가 패키지이면 B는 모듈이 될 수 있고, A가 모듈이면 B는 함수 또는 변수가 될 수 있습니다. 이 경우 아래의 경우에서처럼 모듈이름을 이용해서 함수를 참조하며 안 됩니다.

```
del time
```

```
from time import ctime
```

```
ctime()
```

```
'Mon Dec 25 22:49:36 2017'
```

```
time.ctime()
```

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

```
File "<stdin>", line 1, in <module>
    time.ctime()
```

```
NameError: name 'time' is not defined
```

모듈이 정의하는 모든 이름을 가져올 수 있는 변형이 있습니다.

```
from time import *
```

```
ctime()
```

```
'Mon Dec 25 22:49:36 2017'
```

이렇게 하면 밑줄(\_)로 시작하는 이름을 제외한 모든 이름을 가져옵니다. 대부분의 경우 파 이전 프로그래머는 알려지지 않은 이름 집합을 인터프리터에 도입하여 이미 정의한 일부 항목을 숨길 수 있기 때문에 이 기능을 사용하지 않습니다.

- 일반적으로 모듈이나 패키지에서 \*를 가져 오는 연습은 거의 눈에 띄지 않습니다. 왜냐 하면 \*를 사용하면 종종 가독성이 낮은 코드가 생성되기 때문입니다. 그러나 대화 형 세션에서 타이핑을 줄이는 데 사용할 수 있습니다.
- 효율성을 위해 각 모듈은 인터프리터 세션 당 한 번만 가져옵니다. 따라서 모듈을 변경 하는 경우 인터프리터를 다시 시작해야 합니다. 대화식으로 테스트하려는 모듈이 하나 뿐인 경우에는 `importlib.reload()`를 사용하세요.  
예)  
`import importlib;`  
`importlib.reload(modulename)`
- `import` 한 패키지 또는 모듈의 해제는 `del` 구문을 사용합니다.  
예)  
`del package`



### 3) import A as B

다음은 import A as B 형식의 구문입니다. 이 경우에는 A 모듈 또는 패키지의 이름이 길 경우 별칭을 주어 짧게 쓰기 위한 용도로 사용합니다.

```
del ctime
```

```
import time as t
```

```
t.ctime()
```

```
'Mon Dec 25 22:49:39 2017'
```

```
ctime()
```

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

```
  File "<stdin>", line 1, in <module>
    ctime()
```

```
NameError: name 'ctime' is not defined
```

## 1.4. dir() 함수

내장 함수 dir()은 모듈이 정의한 이름을 찾는 데 사용됩니다. dir() 함수는 정렬된 문자열 목록을 반환합니다.

```
import math
```

```
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot',
'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

```
dir(sys)
```

```
['__displayhook__', '__doc__', '__excepthook__', '__interactivehook__',
 '__loader__', '__name__', '__package__', '__spec__', '__stderr__',
 '__stdin__', '__stdout__', '_clear_type_cache', '_current_frames',
 '_debugmallocstats', '_enablelegacywindowsfsencoding', '_getframe',
 '_git', '_home', '_xoptions', 'api_version', 'argv', 'base_exec_prefix',
 'base_prefix', 'builtin_module_names', 'byteorder', 'call_tracing',
 'callstats', 'copyright', 'displayhook', 'dllhandle',
 'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix',
 'executable', 'exit', 'flags', 'float_info', 'float_repr_style',
 'get_asyncgen_hooks', 'get_coroutine_wrapper', 'getallocatedblocks',
 'getcheckinterval', 'getdefaultencoding', 'getfilesystemencodeerrors',
 'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount',
 'getsizeof', 'getswitchinterval', 'gettrace', 'getwindowsversion',
 'hash_info', 'hexversion', 'implementation', 'int_info', 'intern',
 'is_finalizing', 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path',
 'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps2',
 'set_asyncgen_hooks', 'set_coroutine_wrapper', 'setcheckinterval',
 'setprofile', 'setrecursionlimit', 'setswitchinterval', 'settrace',
 'stderr', 'stdin', 'stdout', 'thread_info', 'version', 'version_info',
 'warnoptions', 'winver']
```

인수 없이 사용하는 `dir()`은 현재 정의된 이름을 나열합니다.

```
dir()
```

```
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'fibonacci', 'sys']
```

`dir()` 함수는 내장 함수와 변수의 이름을 나열하지 않습니다. 내장 함수와 변수들의 리스트는 표준 모듈 `builtins`에 정의되어 있습니다.

```
import builtins
```

```
dir(builtins)
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
```

```

'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError',
'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError',
'MemoryError', 'ModuleNotFoundError', 'NameError', 'None',
'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError',
'OverflowError', 'PendingDeprecationWarning', 'PermissionError',
'ProcessLookupError', 'RecursionError', 'ReferenceError',
'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '_',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__',
'__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin',
'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile',
'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod',
'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format',
'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id',
'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license',
'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object',
'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr',
'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod',
'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

```

## 2. 사용자 정의 모듈

### 2.1. 모듈 만들기

모듈(module)은 프로그램의 기능 단위입니다. 모듈은 파일 단위(\*.py 라는 확장자를 사용)로 작성된 파이썬 코드입니다. 모듈은 함수보다 상위 개념으로 코드의 재사용을 통한 품질을 향상시킵니다.

텍스트 편집기를 사용하여 다음 내용으로 D:/pylib<sup>46)</sup> 디렉토리에 fibonacci.py 파일을 작성하세요.

```
def fibo1(n):    # n까지 피보나치 수열 출력
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()
```

```
def fibo2(n):    # n까지 피보나치 수열 반환
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

### 2.2. 모듈의 경로

만일 fibonacci 라는 모듈을 import 하면 인터프리터는 먼저 해당 이름을 가진 내장 모듈을 검색합니다. 내장 모듈에서 찾지 못하면 sys.path 변수에 지정된 디렉토리 목록에서 fibonacci.py라는 파일을 검색합니다.

sys.path는 모듈의 경로를 가지고 있으며, 다음 위치에서 초기화됩니다.

- 입력 스크립트가 들어있는 디렉토리(또는 파일이 지정되지 않은 경우 현재 디렉토리).
- PYTHONPATH 환경변수에 지정한 디렉토리
- 표준 라이브러리 디렉토리. ex) C:\Python3\Lib

46) 디렉토리는 다르게 설정해도 됩니다.

sys.path는

- sys.path.insert(*index*, *path*) 또는 sys.path.append(*path*)로 추가하고,
- sys.path.remove(*path*)로 제거합니다.

만일 fibonacci.py 파일을 D:/pylib 폴더에 복사해 두었다면...

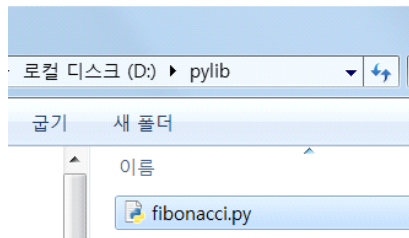


그림 1. 모듈 디렉토리

다음 구문처럼 모듈의 디렉토리를 시스템 패스에 추가할 수 있습니다.

```
import sys
sys.path.append('D:/pylib')
```

*Instructor Note:* 환경변수가 무엇입니까?

환경변수는 컴퓨터의 프로그램들이 실행하기 위해 설정하는 변수입니다. 주로 환경변수의 이름은 대문자로 지정합니다.

리눅스는 다음 형식으로 환경변수를 설정합니다. 환경변수 앞에 \$를 붙이면 기존 환경변수를 불러옵니다. 콜론(:)은 환경변수 구분자입니다. 아래 예에서 환경변수로 설정하는 디렉토리는 /home/user/mymodules입니다.

```
export PYTHONPATH=$PYTHONPATH:/home/user/mymodules
```

윈도우는 [내 컴퓨터] -> [속성] -> [고급 시스템 설정] -> [고급] 탭 -> [환경 변수]에서 환경변수를 추가합니다. 윈도우에서 기존 환경변수 값을 불러오기 위해서는 변수 앞/뒤에 %를 붙입니다. 세미콜론(; )은 환경변수 구분자입니다. 설정하는 디렉토리는 C:/mymodules입니다.

```
변수: PYTHONPATH, 값: %PYTHONPATH%;C:/mymodules
```

## 2.3. 모듈 실행

파이썬 모듈을 이용하여 실행할 때 다음 구문처럼 실행할 수 있습니다.

```
python module_name.py <arguments>
```

다음 구문은 파이썬 인터프리터를 이용하여 fibonacci.py 모듈을 실행시킵니다. \$는 직접 입력하는 것이 아닌 리눅스의 터미널을 의미합니다. \$를 제외한 명령을 리눅스의 터미널 또는 윈도우의 명령 프롬프트에서 실행을 의미합니다.

```
$ python fibonacci.py <arguments>
```

모듈의 끝에 다음 구문을 추가하면 모듈을 직접 가져온 것처럼 모듈의 코드가 실행됩니다.

```
if __name__ == "__main__":
    import sys
    fibo1(int(sys.argv[1]))
```

위 코드를 추가하면 모듈이 “주(main)” 파일로 실행되는 경우에만 명령 행을 구문 분석하는 코드가 실행되기 때문에 파일을 가져올 수 있는 모듈과 마찬가지로 스크립트로 사용할 수 있습니다.

```
$ python fibonacci.py 50
1 1 2 3 5 8 13 21 34
```

만일 모듈이 import 되면 모듈 코드는 실행되지 않습니다.

```
import fibo
```

이러한 방법은 모듈에 사용자 인터페이스를 제공하거나 테스트 목적으로 사용됩니다. 스크립트로 모듈 실행하는 경우 모듈의 테스트 슈트를 실행합니다.

## 2.4. 컴파일된 파이썬 파일

모듈 로드 속도를 높이기 위해 파이썬은 각 모듈의 컴파일 된 버전을 \_\_pycache\_\_ 디렉토리에 module.version.pyc라는 이름으로 캐시합니다. 여기서 버전은 컴파일 된 파일의 형식을 인코딩합니다. 그것은 일반적으로 파이썬 버전 번호를 포함합니다. 예를 들어, CPython

릴리스 3.3에서 컴파일 된 버전의 spam.py는 `__pycache__/spam.cpython-33.pyc`로 캐시됩니다. 이 명명 규칙은 다른 릴리스의 컴파일 된 모듈과 다른 버전의 Python을 함께 사용할 수 있도록 합니다.

파이썬은 소스의 수정 날짜를 컴파일 된 버전과 대조하여 날짜가 지났거나 다시 컴파일 해야 하는지 확인합니다. 이것은 완전 자동 프로세스입니다. 또한 컴파일 된 모듈은 플랫폼에 독립적이므로 동일한 라이브러리를 다른 아키텍처의 시스템간에 공유 할 수 있습니다.

파이썬은 두 가지 상황에서 캐시를 검사하지 않습니다.

- 첫째, 명령 행에서 직접로드 된 모듈에 대해 결과를 다시 컴파일하고 저장하지 않습니다.
- 둘째, 소스 모듈이 없는 경우 캐시를 확인하지 않습니다. 소스가 없는 배포판(컴파일 전용)을 지원하려면 컴파일 된 모듈이 소스 디렉토리에 있어야하며 소스 모듈은 있어서는 안됩니다.

- Python 명령에서 `-O` 또는 `-OO` 스위치를 사용하여 컴파일 된 모듈의 크기를 줄일 수 있습니다. `-O` 스위치는 `assert` 문을 제거하고 `-OO` 스위치는 `assert` 문과 `__doc__` 문자열을 모두 제거합니다. 일부 프로그램은 이러한 기능을 사용할 수 있기 때문에 수행 중인 작업을 알고 있는 경우에만이 옵션을 사용해야 합니다. “최적화 된”모듈에는 `opt` 태그가 있으며 일반적으로 더 작습니다. 향후 릴리스에서는 최적화 효과가 변경 될 수 있습니다.
- 프로그램이 `.py` 파일에서 읽을 때 보다 `.pyc` 파일에서 읽을 때 더 빨리 실행되지 않습니다. `.pyc` 파일에 대해 더 빠른 유일한 것은 로드되는 속도입니다.
- `compileall` 모듈은 디렉토리의 모든 모듈에 대해 `.pyc` 파일을 만들 수 있습니다.
- PEP 3147에는 의사 결정 과정을 포함하여 이 과정에 대한 자세한 내용이 나와 있습니다.

### 3. 패키지

패키지는 “점으로 구분 된 모듈 이름”을 사용하여 파이썬 모듈 네임 스페이스를 구조화하는 방법입니다. 예를 들어, 모듈 이름 A.B는 A라는 패키지에 B라는 서브 모듈을 지정합니다. 모듈을 사용하면 다른 모듈의 작성자가 서로의 전역 변수 이름을 신경 쓰지 않아도 되므로 점이 있는 모듈 이름을 사용하면 작성자(author)<sup>47)</sup>를 절약 할 수 있습니다. 넘파이(NumPy)나 파이썬 이미징 라이브러리(Imaging Library)와 같은 다중 모듈 패키지는 서로의 모듈 이름들이 중복될지에 대해 걱정할 필요가 없습니다.

#### 3.1. 샘플 패키지

패키지를 설명하기 위해서 임의 패키지 환경을 구성하겠습니다. helloworld 디렉토리를 만들고 그 안에 ab 디렉토리와 cd 디렉토리를 생성합니다. 그리고 각 디렉토리에 `__init__.py` 파일을 생성해 놓으세요. helloworld 디렉토리는 D:/pylib 디렉토리 안에 생성하세요.<sup>48)</sup>

##### 1) 패키지 디렉토리 구조 및 샘플 코드

패키지를 설명하기 위한 디렉토리와 파일들의 구조는 아래와 같습니다.

```
helloworld/
+ __init__.py
+ ab/
  + __init__.py
  + a.py
  + b.py
+ cd/
  + __init__.py
  + c.py
```

`__init__.py` 파일은 파이썬이 디렉토리를 패키지에 포함하도록 처리하기 위해 필요합니다. `__init__.py`는 가장 간단한 경우 빈(empty) 파일 일 수 있지만 패키지의 초기화 코드를 실행하거나 나중에 설명 할 `__all__` 변수를 설정할 수도 있습니다.

47) 모듈 사용 시 다른 모듈에서 같은 이름의 변수가 있을 동일한 사용자에게 의해 정의된 것인지 확인할 수 있어야 합니다. 그러므로 모듈 작성 시 author 속성을 추가해 모듈을 만든 사람의 이름을 입력해 놓는 것을 말합니다.

48) 이후의 예에서는 helloworld 디렉토리가 D:/pylib 디렉토리에 있는 것을 기준으로 설명합니다.



각 패키지 안의 `__init__.py` 파일들의 소스코드는 다음과 같습니다.

```
helloworld/__init__.py
```

```
# -*- coding: utf-8 -*-

print("helloworld 패키지를 import 합니다.")
```

```
helloworld/ab/__init__.py
```

```
# -*- coding: utf-8 -*-

__all__ = ["a"]

print("ab 패키지를 import 합니다.")
```

```
helloworld/cd/__init__.py
```

```
# -*- coding: utf-8 -*-

print("cd 패키지를 import 합니다.")
```

a.py, b.py 파일들의 소스코드는 다음과 같습니다. c.py 파일은 뒤에서 설명합니다.

```
helloworld/ab/a.py
```

```
# -*- coding: utf-8 -*-

def hello():
    print("Hello")
```

```
helloworld/ab/b.py
```

```
# -*- coding: utf-8 -*-

def world():
    print("World")
```

## 2) 패키지 경로 추가하기

패키지를 가져올 때, 파이썬은 `sys.path`의 디렉토리를 검색하여 패키지 하위 디렉토리를 찾습니다.

패키지의 경로는

- `sys.path.insert(index, path)` 또는 `sys.path.append(path)`로 추가하고,
- `sys.path.remove(path)`로 제거할 수 있습니다.

```
import sys
```

```
sys.path.append("D:/pylib")
```

## 3.2. 패키지 import 방법

### 1) import 패키지명

패키지 사용자는 패키지에서 개별 모듈을 가져올 수 있습니다. 예를 들면 다음과 같습니다.

```
import helloworld
```

`helloworld` 패키지를 import 합니다.

상위 패키지를 import 한다고 해서 하위 패키지까지 로드되지 않습니다. 하위 패키지를 사용하려면 하위 패키지까지 지정해야 합니다.

```
import helloworld.ab.a
```

`helloworld` 패키지를 import 합니다.

`ab` 패키지를 import 합니다.

- `helloworld` 패키지가 로드된 상태라면 [`helloworld` 패키지를 import 합니다.]는 출력되지 않습니다.

모듈의 함수들은 전체 이름으로 참조되어야합니다.

```
helloworld.ab.a.hello()
```

```
Hello
```

## 2) from 패키지명 import 모듈명

하위 모듈을 가져 오는 다른 방법은 다음과 같습니다.

```
from helloworld.ab import b
```

이것은 서브 모듈 b를 로드하고 패키지 접두사 없이 사용할 수 있게 하므로 다음과 같이 사용할 수 있습니다.

```
b.world()
```

```
World
```

## 3) from 패키지명.모듈명 import 함수명

또 다른 변형은 원하는 함수 또는 변수를 직접 가져 오는 것입니다.

```
from helloworld.ab.b import world
```

from ~ import ~ 로 함수를 직접 가져오면 패키지와 모듈 이름을 생략할 수 있습니다.

```
world()
```

```
World
```

- 패키지 가져 오기 항목을 사용할 때 항목은 패키지의 하위 모듈(또는 하위 패키지)이거나 함수, 클래스 또는 변수와 같이 패키지에 정의 된 다른 이름 일 수 있습니다. import 문은 항목이 패키지에 정의되어 있는지 먼저 테스트합니다. 그렇지 않다면 모듈

이라고 가정하고 모듈을 로드하려고 시도합니다. 만일 찾지 못하면 ImportError 예외가 발생합니다.

- 반대로 `import item.subitem.subsubitem`과 같은 구문을 사용할 때 마지막 항목을 제외한 각 항목은 패키지여야 합니다. 마지막 항목은 모듈이나 패키지가 될 수 있지만 이전 항목에서 정의된 클래스나 함수 또는 변수가 될 수 없습니다.

### 3.3. from 패키지명 import \*

사용자가 import 구문을 다음처럼 사용하면 어떻게 될까요?

아래 코드는 커널을 재시작 한 후 실행해 보세요.

```
import sys
```

```
sys.path.append("D:/pylib")
```

```
from helloworld.ab import *
```

helloworld 패키지를 import 합니다  
ab 패키지를 import 합니다.

```
a.hello()
```

```
Hello
```

```
b.hello()
```

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

```
File "<ipython-input-6-ffdd9487a12b>", line 1, in <module>
    b.hello()
```

```
NameError: name 'b' is not defined
```

이상적으로, 이것이 어떻게든 파일 시스템으로 나가서 패키지에 있는 서브 모듈을 찾아서 모두 가져 오기를 바랄 것이다. 이 작업에는 오랜 시간이 걸릴 수 있으며 하위 모듈을 가져 오면 명시 적으로 하위 모듈을 가져올 때만 발생하는 원치 않는 부작용이 발생할 수 있습니다.

유일한 해결책은 패키지 작성자가 패키지의 명시적 색인을 제공하는 것입니다. import 문은 다음 규칙을 사용합니다. 패키지의 `__init__.py` 파일의 코드가 `__all__`이라는 목록을 정의하면 `package import *`가 발생할 때 가져와야 하는 모듈 이름의 목록으로 간주됩니다. 새 패키지 버전이 릴리스 될 때 이 목록을 최신 상태로 유지하는 것은 패키지 작성자의 몫입니다. 패키지 작성자는 패키지에서 `*`를 가져 오는 용도로 사용하지 않으면 지원하지 않을 수도 있습니다. 예를 들어, `helloworld/ab/__init__.py` 파일에는 다음 코드가 포함되어 있습니다.

```
__all__ = ["a"]
```

이것은 `from helloworld.ab import *` 구문이 `ab` 패키지의 모듈 `a`를 임포트 한다는 것을 의미합니다.

만일 `__all__`이 정의되지 않은 경우 `from helloworld.ab import *` 구문은 `helloworld.ab` 패키지의 모든 하위 모듈을 현재 네임 스페이스로 가져 오지 않습니다. 패키지 `helloworld.ab`가 임포트 되었는지(`__init__.py`에 초기화 코드가 있을 가능성이 있음)만 확인한 다음 패키지에 정의 된 이름을 가져옵니다.

`import *`를 사용할 때에는 모듈이 특정 패턴을 따르는 이름 만 내보내도록 설계되었더라도 프로덕션 코드에서는 여전히 나쁜 습관으로 간주됩니다.

`from Package import specific_submodule`을 사용하는 것은 가져오는 모듈이 다른 패키지의 동일한 이름을 가진 서브 모듈을 사용해야하는 경우가 아니면 권장되는 표기법입니다.

### 3.4. 내부 패키지 참조

패키지가 하위 패키지로 구조화되면 형제 패키지의 하위 모듈을 참조하기 위해 절대 가져 오기를 사용할 수 있습니다. 예를 들어, `helloworld.cd.c` 모듈이 `helloworld.ab` 패키지의 `a` 모듈을 사용해야하는 경우 `from helloworld.ab import a`를 사용할 수 있습니다.

그런데 `from module import name` 형식의 import 문을 사용하여 상대적인 가져 오기를 작성할 수도 있습니다. 이러한 가져 오기는 선행 점( 또는 `..`)을 사용하여 상대 가져 오기에 관련된 현재 및 부모 패키지를 나타냅니다. 예를 들어 `c` 모듈에서 다음을 사용할 수 있습니다.

```
helloworld/cd/c.py

# -*- coding: utf-8 -*-
from .. ab import a

def nice():
    print("Nice")
    a.hello()
```

- 위 구문의 from ..은 현재 패키지의 상위 패키지를 의미합니다.

- 상대 가져 오기는 현재 모듈의 이름을 기반으로 합니다. 주 모듈의 이름은 항상 "\_\_main\_\_"이기 때문에 파이썬 응용 프로그램의 주 모듈로 사용되는 모듈은 항상 절대 가져 오기를 사용해야 합니다. 최상위 모듈에서는 상대 가져오기를 사용할 수 없습니다.

### 3.5. 모듈 디렉토리의 패키지

패키지는 \_\_path\_\_라는 특수 속성을 지원합니다. 이것은 파일의 코드가 실행되기 전에 패키지의 \_\_init\_\_.py를 담고 있는 디렉토리의 이름을 포함하는 리스트로 초기화됩니다. 이 변수는 수정할 수 있습니다. 그렇게 하면 패키지에 포함 된 모듈 및 하위 패키지에 대한 향후 검색에 영향을 미칩니다.

이 기능은 자주 필요하지는 않지만 패키지에서 찾은 모듈 세트를 확장하는 데 사용할 수 있습니다.

예를 들기 위해 다음과 같은 상황이 있다고 가정해 보겠습니다.

- mypkg 와 \_mypkg\_foo라는 패키지를 가지고 있습니다.
- \_mypkg\_foo 패키지는 mypkg라는 패키지와 foo.py 모듈을 포함하고 있습니다.
- 다운로드되어 설치된 패키지 mypkg는 foo.py를 포함하고 있지 않습니다.

이럴 경우에 mypkg 패키지의 \_\_init\_\_.py 파일은 다음과 같이 작성할 수 있습니다.

```
try:
    import _mypkg_foo
    __path__.append(os.path.abspath(os.path.dirname(_mypkg_foo.__file__)))
    import mypkg.foo
except ImportError:
    pass
```

- 이렇게 하면 만일 누군가가 `_mypkg.foo`를 설치했을때 `mypkg.foo` 모듈을 사용할 수 있고, 그렇지 않다면 `mypkg.foo` 모듈을 사용할 수 없습니다.

### 3.6. 패키지 설치

외부의 패키지를 사용하려면 패키지를 설치해야 합니다. 파이썬의 패키지 설치에는 여러 가지 방법으로 할 수 있습니다.

#### 1) pip를 이용한 설치

파이썬이 설치되어 있고 pip가 설치되어 있는 환경이라면 pip 명령을 이용해 쉽게 설치할 수 있습니다.

다음 구문은 PyMySQL 패키지를 설치하는 예입니다. \$ 기호는 명령 프롬프트를 의미합니다.

```
$ pip install PyMySQL
```

#### 2) whl 파일을 이용한 설치

휠(whl) 파일을 직접 다운로드 받아 설치할 수 있습니다. 다음 구문은 python 명령으로 다운로드 받은 패키지 설치파일을 직접 설치하는 구문입니다.

```
$ python -m pip install PyMySQL-0.8.0-py2.py3-none-any.whl
```

#### 3) conda를 이용한 설치

아나콘다를 사용하는 환경이라면 Anaconda Prompt에서 conda 명령으로 패키지를 설치할 수 있습니다.

다음 명령은 아나콘다 프롬프트에서 PyMySQL 패키지를 설치하는 예입니다.

```
(base) C:\Users\COM>conda install PyMySQL
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.4.10
  latest version: 4.5.4

Please update conda by running

  $ conda update -n base conda

## Package Plan ##

environment location: C:\ProgramData\Anaconda3

added / updated specs:
- pymysql

The following packages will be downloaded:

  package                        |      build
  -----|-----
  pymysql-0.8.1                  |      py36_0      135 KB

The following NEW packages will be INSTALLED:

  pymysql: 0.8.1-py36_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
pymysql 0.8.1: ##### |
100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```



다음 명령은 tensorflow를 설치합니다.

```
(base) C:\Users\COM>conda install tensorflow
Solving environment: done

==> WARNING: A newer version of conda exists. <==
current version: 4.4.10
latest version: 4.5.4

Please update conda by running

$ conda update -n base conda

## Package Plan ##

environment location: C:\ProgramData\Anaconda3

added / updated specs:
- tensorflow

The following packages will be downloaded:
```

package	build	
grpcio-1.11.0	py36he025d50_0	1.3 MB
certifi-2018.4.16	py36_0	143 KB
gast-0.2.0	py36_0	15 KB
libprotobuf-3.5.2	he0781b1_0	2.0 MB
tensorboard-1.8.0	py36he025d50_0	3.1 MB
openssl-1.0.2o	h8ea7d77_0	5.4 MB
tensorflow-1.8.0	0	4 KB
ca-certificates-2018.03.07	0	155 KB
termcolor-1.1.0	py36_1	8 KB
protobuf-3.5.2	py36h6538335_0	512 KB
markdown-2.6.11	py36_0	122 KB
tensorflow-base-1.8.0	py36h1a1b453_0	36.5 MB
bleach-1.5.0	py36_0	22 KB
astor-0.6.2	py36_0	43 KB
html5lib-0.9999999	py36_0	178 KB
absl-py-0.2.2	py36_0	136 KB
Total:		49.6 MB

The following NEW packages will be INSTALLED:

```

absl-py:      0.2.2-py36_0
astor:        0.6.2-py36_0
gast:         0.2.0-py36_0
grpcio:       1.11.0-py36he025d50_0
libprotobuf:  3.5.2-he0781b1_0
markdown:     2.6.11-py36_0
protobuf:     3.5.2-py36h6538335_0
tensorboard:  1.8.0-py36he025d50_0
tensorflow:   1.8.0-0
tensorflow-base: 1.8.0-py36h1a1b453_0
termcolor:    1.1.0-py36_1

```

The following packages will be REMOVED:

```

anaconda:      5.1.0-py36_2

```

The following packages will be UPDATED:

```

ca-certificates: 2017.08.26-h94faf87_0 --> 2018.03.07-0
certifi:         2018.1.18-py36_0      --> 2018.4.16-py36_0
openssl:         1.0.2n-h74b6da3_0      --> 1.0.2o-h8ea7d77_0

```

The following packages will be DOWNGRADED:

```

bleach:         2.1.2-py36_0      --> 1.5.0-py36_0
html5lib:       1.0.1-py36h047fa9f_0 --> 0.9999999-py36_0

```

Proceed ([y]/n)? y

Downloading and Extracting Packages

```

grpcio 1.11.0: ##### | 100%
certifi 2018.4.16: ##### | 100%
gast 0.2.0: ##### | 100%
libprotobuf 3.5.2: ##### | 100%
tensorboard 1.8.0: ##### | 100%
openssl 1.0.2o: ##### | 100%
tensorflow 1.8.0: ##### | 100%
ca-certificates 2018.03.07: ##### | 100%
termcolor 1.1.0: ##### | 100%
protobuf 3.5.2: ##### | 100%
markdown 2.6.11: ##### | 100%
tensorflow-base 1.8.0: ##### | 100%
bleach 1.5.0: ##### | 100%
astor 0.6.2: ##### | 100%

```

```
html5lib 0.9999999: ##### | 100%
absl-py 0.2.2: ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Users\COM>
```

### 3.7. 프락시 서버를 사용하는 환경에서 패키지 설치를 위한 인증서 파일

프락시 서버를 사용하는 환경에서 패키지를 설치하려면 아나콘다의 .condarc 파일에 설정을 추가해 줘야 합니다.

다음 예는 S전자의 인재개발원에서 사용하는 인증서 파일의 예입니다. 이 인증서 파일은 사용자 폴더에 있어야 합니다. 리눅스라면 사용자 홈디렉토리이며 윈도우라면 C:\WUsers\₩윈도우계정 폴더입니다. 필자는 C:\WUsers\WCOMW 디렉토리가 사용자 홈디렉토리입니다.

```
.condarc

channels:
  - defaults

proxy_servers:
  http: http://10.241.3.7:8080
  https: https://10.241.3.7:8080

ssl_verify: False
```

- 위의 프락시 서버 주소는 S전자의 연수원에서 사용하는 프락서 서버의 주소입니다.

## 4. 연습문제

---

- 1) 다음과 같은 디렉토리 및 파일 구조를 가지고 있고 각 파일의 내용이 다음과 같을 때 각 패키지와 모듈을 작성하고 실행 결과를 확인하세요.

```
helloworld/
+ __init__.py
+ ab/
  + __init__.py
  + a.py
    >>> def hello():
    ...     print("Hello")
    ...
  + b.py
    >>> def world():
    ...     print("World")
    ...
+ cd/
  + __init__.py
  + c.py
    >>> from .. ab import a
    >>> def nice():
    ...     print("Nice")
    ...     a.hello()
    ...

untitled0.py
>>> from helloworld.cd import c
>>> c.nice()
Nice
Hello
```

## 7장. 객체와 클래스

## 1. 객체와 클래스

### 1.1. 객체와 클래스 개념

- 클래스 : 객체를 만들기 위한 틀(template, blueprint)
- 객체 : 클래스의 인스턴스<sup>49)</sup>
- 객체의 명사적 특성 : 데이터(Data), 변수(Variable), 필드(Field), 속성(Attribute)
- 객체의 동사적 특성 : 행위(Behavior), 함수(Function), 메서드(Method), 기능(Operation)

객체는 객체지향 프로그래밍에서 가장 기본이 되는 개념입니다. 객체의 사전적 의미로는 물건, 물체를 의미합니다. 객체지향의 기본 원리는 현실 세계의 모든 것을 객체로 간주하고, 이러한 객체를 속성(attribute)과 행위(behavior)를 갖는 소프트웨어적 개념으로 표현하는 것이라고 할 수 있습니다. 속성은 사물의 특징을 나타내는 정보를 말하며, 행위는 사물을 이용해 할 수 있는 행동입니다.

‘연필’ 객체를 예로 들면, 연필의 의미 있는 속성에 해당하는 것은 연필의 ‘색상’, ‘굵기’ 등이 있고, 연필의 행위는 ‘쓰다’가 있습니다. ‘연필’이라는 이름은 파이썬에서 ‘클래스의 이름’이 됩니다. 그리고 ‘색상’과 ‘굵기’ 속성은 ‘변수’가 되며, ‘쓰다’라는 기능은 ‘함수’라는 것이 됩니다.

객체들 중에서 연필 객체처럼 우리 눈으로 직접적으로 볼 수 있는 것도 있지만, 우리가 직접 눈으로 볼 수 없는 것도 있습니다. 은행계좌 또는 학생정보 등이 눈으로 볼 수 없는 객체의 대표적인 예입니다. 이러한 눈에 보이지 않는 객체들도 속성과 행위를 가지고 있습니다. ‘은행계좌’라는 객체는 ‘계좌번호’, ‘잔고’, ‘이율’ 등 속성을 나타내는 것들이 있고, ‘입금하다’, ‘출금하다’, ‘이체하다’ 등의 행위가 있습니다.

클래스는 객체를 만들기 위한 설계도(blueprint) 또는 명세서(specification)라고 할 수 있습니다. 클래스는 하나 이상의 유사한 객체를 묶어 하나의 공통된 특성으로 표현한 추상적 자료 형입니다. 클래스가 자기 자신의 구조를 기술하고 정보를 유지하는데 사용하는 데이터를 변수 또는 필드라고 하며, 필드에 작용하는 연산을 함수 또는 메서드라 부릅니다. 객체의 명사적 특징들이 클래스에서는 속성(attribute) 즉, 변수(variable)에 해당하고, 객체의 동사적 특징은 메서드(method)에 해당합니다.

49) 인스턴스(instance)는 객체를 부르는 또 다른 단어입니다. 보통 객체를 충칭해 부를 때 사용합니다.

## 1.2. 클래스 선언과 객체 생성

### 1) 클래스 선언

클래스를 선언할 때 `class` 키워드를 이용합니다. 클래스의 이름은 보통 카멜 표기법(Camel Case)을 사용합니다. 카멜 표기법은 클래스 이름의 첫 문자는 대문자로 시작하며 두 단어 이상으로 만들어질 때 각 단어의 첫 문자를 대문자로 표기하고 나머지는 소문자로 표기합니다.

```
class ClassName:
    class_body
```

다음 코드는 `Person` 클래스를 정의하는 예입니다.

```
class Person:
    pass
```

### 2) 객체 생성

이렇게 정의한 클래스는 객체를 만들기 위해 사용합니다. `type()` 함수로 객체의 타입을 확인해 볼 수 있습니다.

```
p1 = Person()
```

```
type(p1)
```

```
__main__.Person
```

## 1.3. 객체에 멤버 추가

그런데 우리가 클래스를 사용하는 가장 큰 이유는 두 가지입니다. 첫 번째는 객체를 이용해 데이터를 저장하기 위한 용도이며, 두 번째는 객체 고유의 기능을 갖기 위해서입니다.

## 1) 변수 추가

다음 코드는 Person 클래스에 name과 gender 변수를 추가했습니다.

```
class Person:
    name = "홍길동"
    gender = "남자"
```

```
p1 = Person()
```

```
print(p1.name)
```

```
홍길동
```

## 2) 메서드 추가

다음은 클래스에 메서드를 추가해 보겠습니다. 클래스 안에 선언한 함수를 메서드라고 부릅니다. 클래스 안의 메서드는 객체를 이용해 실행시키기 위해서 만듭니다.

```
class Person:
    def print_info():
        print("Person 객체입니다.")
```

위 클래스의 인스턴스를 만들고 메서드를 호출해 보겠습니다.

```
p1 = Person()
p1.print_info()
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-9-cb5c9961f685> in <module>()
      1 p1 = Person()
----> 2 p1.print_info()

TypeError: print_info() takes 0 positional arguments but 1 was given
```

## 3) 인스턴스 메서드



객체를 이용해 참조할 수 있는 메서드를 인스턴스 메서드라고 부릅니다. 그런데 인스턴스 메서드의 첫 번째 인자는 `self`여야 합니다. `self` 인자는 객체의 멤버(변수 또는 메서드)에 접근하기 위해 사용합니다. 위의 `Person` 클래스의 `print_info()` 메서드를 다음과 같이 수정해야 합니다.

```
class Person:
    def print_info(self):
        print("Person 객체입니다.")
```

```
p1 = Person()
p1.print_info()
```

Person 객체입니다.

그러면 인스턴스 메서드의 `self`는 어디에 사용될까요? 다음 클래스를 통해 `self`의 용도를 알아보겠습니다.

```
class Person:
    name = "홍길동"
    gender = "남자"
    def print_info(self):
        print("{}님은 {}입니다.".format(name, gender))
```

위 클래스의 객체를 만들고 `print_info()` 메서드를 호출하면 `name`변수와 `gender` 변수에서 에러가 발생합니다.

```
p1 = Person()
```

```
p1.print_info()
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-4-123b1e7cbd59> in <module>()
----> 1 p1.print_info()
```

50) 키워드가 아니라면 다른 단어도 가능합니다. 그러나 현재 자신 객체임을 알리기 위해 `self`를 쓸 것을 권장합니다.

```
<ipython-input-2-7dd31de1f5cb> in print_info(self)
      5     def print_info(self):
----> 6         print("{}님은 {}입니다.".format(name, gender))
```

```
NameError: name 'name' is not defined
```

인스턴스 메서드는 객체를 통해 참조하는 메서드입니다. 그러므로 객체가 갖는 멤버에 접근하려면 객체 자신을 참조할 수 있는 그 무엇이 있어야 합니다. 그래서 자신 객체를 의미하는 `self`를 인스턴스 메서드의 인자로 정의해서 자신 객체의 멤버를 참조할 수 있도록 하는 것입니다. `self` 대신에 파이썬의 키워드가 아니라면 다른 단어를 사용할 수 있습니다. 그러나 자신 객체임을 명백히 알리기 위해 `self`를 사용할 것을 권장합니다.

```
class Person:
    name = "홍길동"
    gender = "남자"
    def print_info(self):
        print("{}님은 {}입니다.".format(self.name, self.gender))
```

```
p1 = Person()
```

```
p1.print_info()
```

```
홍길동님은 남자입니다.
```

## 1.4. 클래스 메서드와 스태틱 메서드

클래스는 객체를 생성하지 않고 참조할 수 있는 메서드를 가질 수 있습니다.

- 클래스(class) 메서드 : 메서드 첫 번째 인자는 `cls`<sup>51)</sup>여야 하며, 메서드 위에 `@classmethod` 아노테이션을 선언합니다.
- 스태틱(static) 메서드 : 메서드 위에 `@staticmethod` 아노테이션을 선언합니다.

함수 위에 `@classmethod` 아노테이션(annotation)을 표기하면 클래스 메서드가 됩니다. 이 메서드의 첫 번째 인자는 `cls` 여야 합니다. `@staticmethod` 아노테이션을 표기하면 정적(static) 메서드가 됩니다. 클래스 메서드는 메서드의 첫 번째 인자인 `cls`를 이용하여 변수를 참조할 수 있으며, 정적 메서드는 클래스 이름을 이용해서 참조할 수 있습니다.

51) `cls` 인자도 키워드가 아니라면 다른 단어도 가능합니다. 그러나 클래스의 멤버를 참조하기 위한 객체임을 알리기 위해 `cls`를 사용한 것입니다. `class`는 클래스를 정의할 때 사용하는 키워드이기 때문에 `cls` 또는 `clazz` 등으로 사용합니다.

```
class Person:
    name = "홍길동"
    gender = "남자"
    def print_info(self):
        print("{}님은 {}입니다.".format(self.name, self.gender))

    @classmethod
    def do_(cls):
        print("이름 : {}, 성별 : {}".format(cls.name, cls.gender))

    @staticmethod
    def that_():
        print("이름은 {}이고, 성별은 {}입니다.".format(Person.name,
        Person.gender));
```

다음 코드는 객체를 생성하지 않고 클래스 메서드와 정적 메서드를 클래스 이름으로 직접 참조하는 예입니다.

```
Person.do_()
```

```
이름 : 홍길동, 성별 : 남자
```

```
Person.that_()
```

```
이름은 홍길동이고, 성별은 남자입니다.
```

클래스 메서드와 스테틱 메서드는 인스턴스를 통해서도 참조할 수 있습니다. 그러나 인스턴스 메서드가 호출 될 때 참조하는 변수는 인스턴스 변수가 아님을 주의하세요. 클래스 메서드 또는 스테틱 메서드에서 참조하는 변수는 클래스 변수입니다.

```
p1 = Person()
p1.do_()
```

```
이름 : 홍길동, 성별 : 남자
```

p1 객체의 변수 값을 수정하더라도 클래스 메서드와 스테틱 메서드가 참조하는 변수는 클래스 변수입니다.

```
p1.name = "이순신"
p1.do_()
```

```
이름 : 홍길동, 성별 : 남자
```

```
p1.that_()
```

이름은 홍길동이고, 성별은 남자입니다.

## 1.5. 생성자와 소멸자

생성자는 객체가 생성될 때 자동으로 실행되며, 생성 시 필요한 코드를 포함할 수 있습니다. 생성자의 이름은 `__init__()`입니다.

소멸자는 객체가 소멸될 때 자동으로 실행되며, 소멸 시 필요한 코드를 포함할 수 있습니다. 소멸자의 이름은 `__del__()`입니다. 인스턴스 객체의 레퍼런스 카운트가 0이 될 때 소멸됩니다.

```
class Person:
    def __init__(self):
        print("Person 객체를 생성합니다.")
        self.name = "홍길동"
        self.gender = "남자"

    def __del__(self):
        print("Person 객체를 소멸시킵니다.")

    def print_info(self):
        print("{}님은 {}입니다.".format(self.name, self.gender))
```

객체 생성시 `__init__()` 함수가 실행됩니다.

```
p1 = Person()
```

Person 객체를 생성합니다.

```
p1.print_info()
```

홍길동님은 남자입니다.

`__del__()` 함수는 객체의 레퍼런스 카운트가 0이 될 때 실행됩니다.

```
p1 = Person()
```

```
p1 = None
```

Person 객체를 생성합니다.  
Person 객체를 소멸시킵니다.

```
p1 = Person()
del p1
```

Person 객체를 생성합니다.  
Person 객체를 소멸시킵니다.

## 1.6. 생성자를 이용한 인스턴스 변수 초기화

생성자를 이용해 클래스의 인스턴스가 생성될 때 인스턴스 변수를 초기화 할 수 있습니다.

```
class Person:
    def __init__(self, name, gender):
        print("Person 객체를 생성합니다.")
        self.name = name
        self.gender = gender

    def __del__(self):
        print("Person 객체를 소멸시킵니다.")

    def print_info(self):
        print("{}님은 {}입니다.".format(self.name, self.gender))
```

Person 클래스는 인스턴스 생성 시 Person(이름, 성별) 형식으로 사용할 수 있습니다.

```
p1 = Person("홍길서", "여자")
```

Person 객체를 생성합니다.

```
p1.print_info()
```

홍길서님은 여자입니다.

파이썬은 함수를 중복 정의<sup>52)</sup> 할 수 없습니다. 그러므로 생성자도 중복 정의할 수 없습니

다. 만일 Person 클래스에 다음과 같이 생성자를 중복 정의하면 먼저 정의한 `__init__(self, name)` 생성자는 무시됩니다.

```
class Person:
    def __init__(self, name):
        print("Person 객체를 생성합니다.")
        self.name = name
        self.gender = "여자"

    def __init__(self, name, gender):
        print("Person 객체를 생성합니다.")
        self.name = name
        self.gender = gender

    def __del__(self):
        print("Person 객체를 소멸시킵니다.")

    def print_info(self):
        print("{}님은 {}입니다.".format(self.name, self.gender))
```

먼저 정의된 `Person(self, name)`은 `Person(self, name, gender)`에 의해 무시됩니다.

```
p1 = Person("홍길동")
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
  p1 = Person("홍길동")
```

```
TypeError: __init__() missing 1 required positional argument: 'gender'
```

나중에 정의된 `Person(self, name, gender)`는 사용할 수 있습니다.

```
p1 = Person("홍길서", "여자")
```

```
Person 객체를 생성합니다.
홍길서님은 여자입니다.
```

---

52) 메서드(또는 함수) 중복 정의를 Overloading 이라고 합니다.

## 2. 상속과 재정의

### 2.1. 상속

상속(Inheritance)은 객체 재사용의 한 방법입니다. 상속을 이용하면 부모 클래스의 모든 속성들을 자식 클래스로 물려줄 수 있습니다.

```
class SubClassName(SuperClassName) :
```

구문에서...

- SubClassName : 정의할 자식 클래스 이름
- SuperClassName : 상속받을 부모 클래스 이름

다음 구문은 Person 클래스를 정의하고 Student 클래스에서 Person 클래스를 상속받습니다.

```
class Person:
    def __init__(self, name, gender):
        self.name = name
        self.gender = gender

    def print_info(self):
        print("{}님은 {}입니다.".format(self.name, self.gender))

    def to_string(self):
        return "name: {0}, gender: {1}".format(self.name, self.gender)
```

Student 클래스는 Person 클래스의 하위 클래스입니다.

```
class Student(Person):
    def __init__(self, name, gender, major):
        self.name = name
        self.gender = gender
        self.major = major

    def __del__(self):
        pass
```

클래스 간의 상속 관계를 확인하려면 ‘issubclass(SubClass, SuperClass)’ 함수를 사용합니다.

```
issubclass(Student, Person)
```

```
True
```

## 2.2. 부모클래스의 생성자 사용

Student 클래스의 생성자는 Person 클래스의 생성자를 이용하여 인스턴스 변수를 초기화할 수 있습니다. 다음 코드는 Student 클래스의 생성자를 수정한 것입니다.

```
class Student(Person):
    def __init__(self, name, gender, major):
        Person.__init__(self, name, gender)
        self.major = major
```

- 이렇게 하면 부모클래스의 생성자를 호출하여 자식클래스의 변수들을 쉽게 초기화할 수 있습니다.

상속관계가 있을 경우 자식 클래스는 부모 클래스의 변수와 메서드를 물려받습니다.

```
s1 = Student("홍길남", "남자", "경제학")
s1.print_info()
```

```
홍길남님은 남자입니다.
```

위의 예에서 print\_info() 함수를 호출하면 name과 gender 정보만 출력됩니다. 그 이유는 Student클래스에서 상속받은 Person 클래스의 print\_info() 함수는 name 정보와 gender 정보만 출력하도록 구현되어 있기 때문입니다.

## 2.3. 재정의

부모 클래스에서 정의한 함수를 자식클래스에서 다시 정의할 수 있습니다. 이름과 인수의 목록이 같아야 하며 구현부는 다를 수 있습니다.



다음 코드는 Student 클래스가 Person 클래스의 print\_info() 함수를 재정의(Overriding)하는 예입니다.

```
class Student(Person):
    def __init__(self, name, gender, major):
        Person.__init__(self, name, gender)
        self.major = major

    def __del__(self):
        pass

    def print_info(self):
        print("{}님은 {}이며, 전공은 {}입니다."\
              .format(self.name, self.gender, self.major))
```

```
issubclass(Student, Person)
```

```
True
```

```
Student.__dict__
```

```
mappingproxy({'__doc__': None,
              '__init__': <function __main__.Student.__init__>,
              '__module__': '__main__',
              'print info': <function __main__.Student.print info>,
              'to_string': <function __main__.Student.to_string>})
```

```
s2 = Student("홍길서", "여자", "컴퓨터공학")
s2.print_info()
```

```
홍길서님은 여자이며, 전공은 컴퓨터공학입니다.
```

## 2.4. super()

super()는 부모 클래스의 멤버를 참조합니다. Student 클래스에 to\_string() 함수를 재정의(Overriding)해서 추가하세요. to\_string()에서 super().to\_string()을 통해 Person의 to\_string()을 호출합니다.

```
class Student(Person):
    def __init__(self, name, gender, major):
```

```

    Person.__init__(self, name, gender)
    self.major = major

def __del__(self):
    pass

def print_info(self):
    print("{}님은 {}이며, 전공은 {}입니다."\
          .format(self.name, self.gender, self.major))

def to_string(self):
    return super().to_string() + ", major:{}".format(self.major)

```

```
s3 = Student("홍길복", "남자", "심리학")
```

```
print(s3.to_string())
```

```
name: 홍길복, gender: 남자, major:심리학
```

## 2.5. 정적 변수

클래스의 변수이름 앞에 \_\_(under score 두 개)를 붙이면 내부적으로 `_클래스명.__변수명` 형식으로 참조할 수 있습니다. 이는 클래스 내의 어떤 메서드에서도 동일한 이름으로 참조할 수 있도록 합니다.

다음 클래스는 객체가 만들어지는 개수를 저장하기 위해서 `__count` 변수를 정의했습니다.

```

class Student(Person):
    __count = 0

    def __init__(self, name, gender, major):
        Student._Student__count += 1
        Person.__init__(self, name, gender)
        self.major = major

    def __del__(self):
        Student._Student__count -= 1

    def print_info(self):
        print("{}님은 {}이며, 전공은 {}입니다."\
              .format(self.name, self.gender, self.major))

```

```
@classmethod
def get_count(cls):
    return Student._Student__count
```

다음 코드는 Student 객체를 생성하고 get\_count() 메서드를 통해서 \_\_count 변수를 조회합니다.

```
s1 = Student("홍길동", "남자", "컴퓨터공학")
Student.get_count()
```

1

```
s2 = Student("홍길서", "여자", "컴퓨터공학")
Student.get_count()
```

2

객체에 None을 할당하거나 del을 이용해 객체를 삭제하면 소멸자가 호출됩니다.

```
s1 = None
Student.get_count()
```

1

```
del s2
Student.get_count()
```

0

### 3. 연습문제

---

#### 1) 다음의 조건을 만족하는 클래스를 작성하세요.

- 도형(Shape) 클래스와 삼각형(Triangle) 클래스를 만들어야 합니다.
- 도형 클래스는 생성자를 통해 x, y좌표를 정할 수 있습니다. x, y 좌표의 기본 값은 0, 0입니다.
- 도형 클래스는 x, y좌표를 이동시킬 수 있어야 합니다.
- 도형 클래스는 x, y좌표 정보를 문자열로 리턴하는 to\_string() 메서드가 있어야 합니다.
- 모든 도형 클래스는 도형의 면적을 구하는 메서드(cal\_area())가 있어야 합니다. 다만 도형 클래스의 메서드는 구현되어 있지 않으므로 호출하면 에러가 발생합니다.
- 도형 클래스에 정적 메서드를 하나 이상 구현해야 합니다.
- '삼각형 클래스는 도형 클래스이다' 관계가 성립해야 합니다.
- 삼각형 클래스는 삼각형의 밑변의 길이와 높이를 생성자를 통해 정할 수 있어야 합니다. 삼각형 객체를 만들 때 좌표 정보 x, y는 기본값 0, 0을 가질 수 있습니다.
- 삼각형의 면적을 구하는 메서드를 추가해야 합니다.
- 삼각형의 정보를 문자열로 반환하는 to\_string() 메서드가 있어야 합니다.
- 삼각형 객체가 몇 개가 만들어져 있는지 정보를 저장하는 클래스 변수를 추가하고 생성자에서 객체를 만들 때마다 증가시키며, 소멸자에서 객체를 소멸시킬 때마다 감소시켜야 합니다.

정답 및 풀이

```
class Shape:
    def __init__(self, x=0, y=0) :
        self.x = x
        self.y = y

    def move(self, x_, y_):
        self.x = self.x + x_
        self.y = self.y + y_

    def to_string(self):
        return ("x: {}, y: {}".format(self.x, self.y))

    @staticmethod
    def info():
        print("도형 클래스 입니다.")

    def calc_area(self):
        raise Exception("이 메서드는 반드시 구현되어야 합니다.")
```

```
class Triangle(Shape):
    __count = 0 #_Triangle__count

    def __init__(self, width, height, x=0, y=0):
        Triangle._Triangle__count = Triangle._Triangle__count + 1
        Shape.__init__(self, x, y)
        self.width = width
        self.height = height

    def __del__(self):
        Triangle._Triangle__count = Triangle._Triangle__count - 1

    def calc_area(self):
        return (self.width + self.height) / 2

    def to_string(self):
        return Shape.to_string(self) + ", width: {}, height: {}".format(self.width, self.height)

    @classmethod
    def get_count(cls):
        return (cls.__count)
```

```
t1 = Triangle(10,20)
print(t1.calc_area())
```

15.0

```
print(t1.to_string())
```

x: 0, y: 0, width: 10, height: 20

```
t1.move(5, 10)
print(t1.to_string())
```

x: 5, y: 10, width: 10, height: 20

```
print(Triangle._Triangle__count)
```

1

```
t2 = Triangle(20,40)
print(Triangle._Triangle__count)
```

2

```
print(Triangle.get_count())
```

2

```
t1 = None
print(Triangle._Triangle__count)
```

1



## 8장. 예외처리

## 1. 예외처리

예외 처리(Exception Handling)는 오류발생에 대한 대처 방법 중의 하나입니다. 예외 처리는 시스템 스스로 오류를 복구 하는 것이 아니고 오류발생 가능성이 있는 부분에 대한 처리를 미리 프로그래밍 하는 것입니다.

만일 여러분이 데이터를 입력받는 상황이라면 사용자의 입력 실수로 인해 프로그램이 종료되는 것을 원하지 않을 것입니다. 사용자가 입력 값을 잘 못 입력했을 경우 프로그램은 안내 메시지를 출력하고 계속 실행되어야 할 것입니다. 예외처리는 에러 또는 예외가 발생할 상황을 가정하고 이를 미리 처리해 프로그램의 안정성을 높이는 것이 목적입니다.

다음과 같은 상황들은 예외 처리를 해야 합니다.

- 파일을 다룰 때 파일이 없거나 쓰기금지로 인한 오류
- 데이터베이스 프로그래밍 시 제약조건 등에 의한 데이터베이스 서버 오류
- 네트워크 관련 코드에서 네트워크 연결 실패로 인한 오류

다음 코드는 divide() 함수를 실행할 때에 잘 못된 입력 값에 의해 프로그램이 종료되지 않도록 예외처리를 한 예입니다.

```
def divide(a, b):
    return a/b
```

```
try:
    c = divide(5, 'string')
except ZeroDivisionError:
    print('두 번째 인자는 0이어서는 안됩니다.')
except TypeError:
    print('모든 인자는 숫자이어야 합니다.')
except:
    print('무슨 에러인지 모르겠어요~')
```

모든 인자는 숫자이어야 합니다.

- 사실 이 코드는 divide() 함수의 인자를 수정해야 하는 것이 맞습니다. 여기서는 예외를 설명하기 위한 예 일뿐입니다.



파이썬에서 주로 발생하는 예외들은 exception 모듈에 미리 정의되어 있습니다. exception 모듈은 내장 모듈이므로 import할 필요가 없습니다.

다음은 예외 상황에 따른 미리 정의되어 있는 예외 클래스들입니다.

```
f = open('없는파일.txt', 'r')
-> FileNotFoundError: [Errno 2] No such file or directory: '없는파일.txt'
```

```
4 / 0
-> ZeroDivisionError: division by zero
```

```
a = [1,2,3]
a[3]
-> IndexError: list index out of range
```

다음은 파이썬은 예외클래스 구조입니다. 파이썬의 예외 클래스들은 BaseException의 하위 클래스들입니다. BaseException의 하위 클래스에는 SystemExit, KeyboardInterrupt, GeneratorExit 그리고 Exception 클래스가 있으며 우리가 실제로 처리해야 하는 예외 클래스들은 Exception 클래스의 하위 클래스들입니다.

```
BaseException
+---SystemExit
+---KeyboardInterrupt
+---GeneratorExit
+---Exception
    +---StopIteration
    +---StopAsyncIteration
    +---ArithmeticError
    |   +---FloatingPointError
    |   +---OverflowError
    |   +---ZeroDivisionError
    +---AssertionError
    +---AttributeError
    +---BufferError
    +---EOFError
    +---ImportError
        +---ModuleNotFoundError
    +---LookupError
    |   +---IndexError
    |   +---KeyError
    +---MemoryError
```

```
+--NameError
|   +--UnboundLocalError
+--OSError
|   +--BlockingIOError
|   +--ChildProcessError
|   +--ConnectionError
|       |   +--BrokenPipeError
|       |   +--ConnectionAbortedError
|       |   +--ConnectionRefusedError
|       |   +--ConnectionResetError
|   +--FileExistsError
|   +--FileNotFoundError
|   +--InterruptedError
|   +--IsADirectoryError
|   +--NotADirectoryError
|   +--PermissionError
|   +--ProcessLookupError
|   +--TimeoutError
+--ReferenceError
+--RuntimeError
|   +--NotImplementedError
|   +--RecursionError
+--SyntaxError
|   +--IndentationError
|       +--TabError
+--SystemError
+--TypeError
+--ValueError
|   +--UnicodeError
|       +--UnicodeDecodeError
|       +--UnicodeEncodeError
|       +--UnicodeTranslateError
+--Warning
    +--DeprecationWarning
    +--PendingDeprecationWarning
    +--RuntimeWarning
    +--SyntaxWarning
    +--UserWarning
    +--FutureWarning
    +--ImportWarning
    +--UnicodeWarning
    +--BytesWarning
    +--ResourceWarning
```

## 2. try~except로 예외 처리하기

### 2.1. try~except

try는 예외를 처리할 때 예외가 발생할 가능성이 있는 문장을 작성하기 위해 사용하는 블록의 시작 부분입니다. try 절(또는 블록)에서 예외가 발생하면 except 절을 찾습니다. except 절은 예외가 발생했을 경우 실행되어야 하는 코드를 작성합니다.

```
try:
    4 / 0
except ZeroDivisionError as e:
    print(e)
```

### 2.2. else

try 절과 함께 사용되는 else 절은 예외가 발생하지 않을 경우에 실행됩니다. else 절은 except 절 다음에 와야 합니다.

```
try:
    f = open('foo.txt', 'r')
except FileNotFoundError as e:
    print(str(e))
else:
    data = f.read()
```

### 2.3. finally

finally 절은 예외 발생과 상관없이 항상 실행됩니다. 이 절에는 주로 try절에서 열어놓은 리소스를 해제하기 위해 사용합니다.

```
f = open('foo.txt', 'w')
try:
    # 실행 코드
finally:
    f.close()
```

### 3. raise로 예외 발생시키기

raise는 강제로 예외를 발생해야 할 필요가 있을 때 사용합니다.

다음 코드에서 Shape 클래스의 calcArea() 함수는 raise를 이용하여 강제로 예외를 발생시킵니다. 만일 Shape 클래스를 상속받는 클래스가 있다면 반드시 calcArea() 함수를 구현해야 합니다. 부모클래스에 선언되어 있는 함수를 자식클래스에서 다시 정의하는 것을 재정의(Override)라고 합니다.

```
class Shape:
    def calcArea(self):
        raise NotImplementedError
```

```
class Circle(Shape):
    pass
```

```
myCircle = Circle()
```

```
myCircle.calcArea()
```

```
-----
NotImplementedError                                Traceback (most recent call last)
<ipython-input-6-b1f6ef9b90f4> in <module>()
----> 1myCircle.calcArea()

<ipython-input-2-28d5b59c31dc> in calcArea(self)
      1class Shape:
      2    def calcArea(self):
----> 3        raise NotImplementedError

NotImplementedError:
```

사실 raise의 주 용도는 예외를 직접 처리하는 것이 아닌 예외가 발생한 원인을 제공한 곳으로 예외를 던져주기 위한 것입니다.

다음 코드는 피호출자(callee)와 호출자(caller) 사이에서 예외 처리를 어떻게 처리해야 하는지를 보여줍니다.

```
# 피호출자(callee)
def insert(data):
    if len(data)==0:
        raise Exception("길이가 0입니다.")
    print(data, "을(를) 입력합니다.", sep=" ")
```

```
# 호출자(caller)
# data = [1, 2, 3]
data = []
try:
    insert(data)
except Exception as e:
    print(e.args[0])
else:
    print("정상 실행되었습니다.")
```

길이가 0입니다.

- callee의 insert() 함수는 data의 길이가 0이면 예외를 발생시킵니다. 그리고 실제 예외를 처리하는 것은 caller에서 합니다. 데이터의 길이가 0인 것으로 인해 발생하는 예외의 처리는 data의 근원지인 caller에서 처리하는 것이 마땅하기 때문입니다.

## 4. 사용자 정의 예외

사용자가 예외 클래스를 만들려면 Exception 클래스 또는 그 하위 클래스를 상속받아 만들면 됩니다.

다음 코드는 사용자 정의 예외 클래스의 예입니다. 생성자가 인자를 갖게 할지 여부는 여러분이 선택할 수 있습니다. 다만 상속받는 예외 클래스(예에서는 Exception)의 생성자를 호출 할 때 예외 메시지를 전달해야 한다는 것만 잊지 않으면 됩니다.

```
class ZeroLengthException(Exception):
    def __init__(self):
        Exception.__init__(self, "사용자 정의 예외 : 길이가 0입니다.")
```

다음 코드는 사용자 정의 예외 클래스를 이용해 예외를 발생시킵니다.

```
def insert(data):
    if len(data)==0:
        raise ZeroLengthException()
    print(data, "을(를) 입력합니다.", sep=" ")
```

다음 코드는 데이터의 길이가 0인 리스트를 insert() 함수의 인수로 전달하기 때문에 예외가 발생합니다. 이 때의 예외 메시지는 사용자 정의 예외 클래스에서 정의한 내용입니다.

```
# data = [1, 2, 3]
data = []
try:
    insert(data)
except Exception as e:
    print(e.args[0])
else:
    print("정상 실행되었습니다.")
```

사용자 정의 예외 : 길이가 0입니다.

## 9장. 입/출력 프로그래밍

## 1. 파일 입출력

파일로 데이터를 입출력할 때 `open()` 함수를 이용하여 파일을 열어야 합니다. `open()` 함수는 파일 객체를 반환합니다.

```
file_pointer = open(file_name, mode)
```

구문에서...

- `file_pointer` : 열린 파일 객체입니다. 파일 객체의 `readline()` 또는 `readlines()` 함수를 이용하여 파일로부터 데이터를 읽습니다. `write()` 함수는 데이터를 씁니다.
- `mode` : 파일 열기 모드를 의미합니다.
  - `r` : 읽기 모드 - 파일을 읽기만 할 때 사용
  - `w` : 쓰기 모드 - 파일의 내용을 쓸 때 사용
  - `a` : 추가 모드 - 파일의 마지막에 추가할 때 사용
  - `b` : 바이너리 모드 - 피클 등을 사용하여 저장하거나 불러올 때는 바이너리 모드로 지정해야 합니다.

다음 코드는 파일 객체를 쓰기 모드로 엽니다.

```
f = open("sample.txt", "w")
```

```
print(f.writable())
```

```
True
```

`write()` 함수는 파일에 데이터를 기록합니다. `write()` 함수는 기록한 바이트 수를 반환합니다. 개행 문자(`\n`)은 1바이트로 처리됩니다. `\n`은 시스템에 따라 `\r\n`으로 보일 수도 있습니다.

```
f.write("Helle\nWorld\n")
```

```
12
```



파일포인터를 닫아야 데이터가 저장됩니다.

```
f.close()
```

다음 코드는 방금 쓴 sample.txt 파일을 데이터 추가 모드로 엽니다.

```
f = open("sample.txt", "a")
```

파일 포인터 객체의 write() 함수를 이용해서 데이터를 쓸 수 있지만 print() 함수를 이용해서도 파일에 데이터를 쓸 수 있습니다.

```
print("프린트 함수로 쓸 수 있습니다.", file=f)
```

```
f.close()
```

다음 코드는 sample.txt 파일을 읽기 모드로 엽니다.

```
f = open("sample.txt", "r")
```

readline()함수는 한 줄 씩 읽고 readlines() 함수는 파일의 모든 라인을 한 번에 읽습니다. 이렇게 읽은 데이터는 for 반복문을 이용해 처리할 수 있습니다.

```
lines = f.readlines()
for line in lines:
    print(line)
```

```
Helle
```

```
World
```

```
프린트 함수로 쓸 수 있습니다.
```

```
f.close()
```

## 2. 피클을 이용한 객체 직렬화

데이터를 텍스트 형식으로 읽고 써야 하는 것은 생각보다 번거로운 일입니다. 만일 여러분이 고객의 정보를 저장하기 위해 아래와 같이 데이터들을 콤마(,)로 연결하여 하나의 문자열을 만들어 저장한 후...

```
홍길동,20,kildong@hong.com,서울시 강동구
홍길서,25,kilseo@hong.com,서울시 강서구
```

이것을 아래처럼 한 라인씩 읽어 split() 함수를 이용해 분리하고 형 변환 하는 등의 작업은 데이터를 읽고 쓰는 사람에겐 매우 불편한 일입니다.

```
line = "홍길동,20,kildong@hong.com,서울시 강동구"
line.split(",")
```

```
['홍길동', '20', 'kildong@hong.com', '서울시 강동구']
```

피클(Pickle) 모듈은 리스트나 객체들을 파일에 저장 할 때 유용하게 사용됩니다.

피클로 데이터를 파일에 쓸 때는 쓰기 바이너리 모드(wb)로 열어야 하고 읽을 때는 읽기 바이너리 모드(rb)로 파일을 열어야 합니다.

```
pickle.dump(data, file_pointer)
```

```
data = pickle.load(file_pointer)
```

```
import pickle
```

```
colors = ['red', 'green', 'black']
```

```
f = open('pickle.dat', 'wb')
pickle.dump(colors, f)
```

```
f.close()
```

다음 그림은 피클로 저장한 데이터를 메모장에서 열어본 것입니다. 피클로 저장한 데이터파일은 바이너리 파일이기 때문에 텍스트 에디터로 파일을 열어도 내용을 알아볼 수 없습니다.

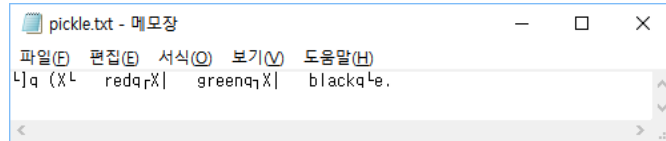


그림 1. pickle로 저장한 데이터

```
import pickle
```

```
del colors
```

```
f = open('pickle.dat', 'rb')
colors = pickle.load(f)
```

```
print(colors)
```

```
['red', 'green', 'black']
```



## 10장. 데이터베이스 연동

## 1. SQLite

### 1.1. SQLite와 파이썬

SQLite는 구글 안드로이드에 탑재된 파일 기반의 DBMS로써 설치가 필요 없이 사용 가능합니다.

파이썬에서 데이터베이스에 연결하기 위해 sqlite3 모듈을 사용합니다.

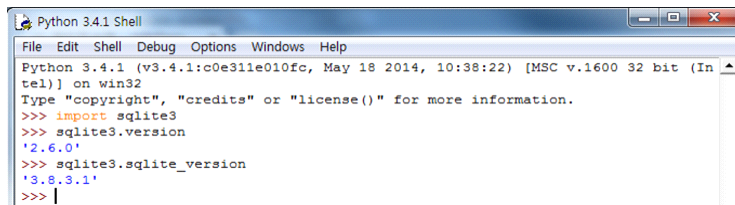


그림 1. sqlite3 버전 확인

### 1.2. 데이터베이스 연결

sqlite3 모듈의 connect 함수를 통해 데이터베이스에 연결합니다. 데이터베이스에 연결이 완료되면 Connection 객체가 반환됩니다.

```
import sqlite3
con = sqlite3.connect("kospi.db")

type(con)
<class 'sqlite3.Connection'>
```

### 1.3. 데이터베이스에 데이터 입력

CRUD는 Create, Read, Update, Delete의 약어이며 데이터베이스에 데이터를 입력, 조회, 수정, 삭제하는 것을 의미합니다.

데이터를 데이터베이스에 넣기 위해서는 다음 과정을 따라야 합니다.

1. sqlite3.connect 함수를 이용해서 DB 파일에 연결한 후 'Connection' 객체를 생성한다.
2. Connection객체를 통해 Cursor 객체를 생성한다.
3. 'Cursor' 객체의 execute 함수를 통해서 query를 실행한다.
4. 'Connection' 객체의 commit를 이용하여 변경된 내용을 commit한다.
5. DB와의 연결을 닫는다.

### 1) 데이터베이스에 연결하고 커서(Cursor) 객체 얻어오기

```
con = sqlite3.connect("contact.db")
cursor = con.cursor()
```

### 2) 데이터를 insert 전에 먼저 테이블을 생성

```
cursor.execute("CREATE TABLE contact(name text, age int, e_mail text)")
```

### 3) 데이터 insert

```
cursor.execute("INSERT INTO contact VALUES('kim', 30, 'kim@naver.com')")
```

### 4) 작업한 내용 DB에 반영 및 DB 연결 종료

```
con.commit()
con.close()
```

## 1.4. 데이터베이스에서 데이터 조회

데이터베이스에서 데이터 읽기 위해서는 다음 절차를 따라야 합니다.

1. sqlite3.connect 함수를 이용해서 DB 파일에 연결한 후 'Connection' 객체를 생성한다.
2. Connection객체를 통해 Cursor 객체를 생성한다.
3. 'Cursor' 객체의 execute 함수를 통해서 query를 실행한다.
4. 반복문을 이용하여 커서로부터 데이터 패치하기

### 1) 데이터베이스에 연결하고 커서(Cursor) 객체 얻어오기

```
con = sqlite3.connect("contact.db")
cursor = con.cursor()
```

### 2) 커서를 이용하여 select하는 구문을 실행

```
cursor.execute("SELECT * FROM contact")
```

### 3) 데이터를 반복문을 이용하여 처리

```
for row in cursor:
    print(row)
```

### 4) 행 단위로 데이터 조회

```
cursor.fetchone()
('kim', 30, 'kim@naver.com')      # 튜플 형태로 리턴됨
```

### 5) 데이터 한 번에 모두 가져오기

```
cursor.execute("SELECT * FROM contact")
mydata = cursor.fetchall()

mydata
[('kim', 30, 'kim@naver.com'), ('lee', 35, 'lee@daum.net'), ('park', 40, 'heo@coderby.com')]
```



## 2. 오라클 데이터베이스 연결

오라클 또는 MySQL 데이터베이스 등 사용 데이터베이스에 연결하기 위해서 import 하는 패키지 또는 모듈의 이름과 connect() 함수를 이용해 Connection 객체를 생성하는 것만 다르고 나머지는 앞에서 설명한 다릅니다.

### 2.1. cx\_Oracle 패키지

cx\_Oracle 패키지는 오라클 데이터베이스 연결할 수 있도록 합니다. cx\_Oracle 모듈을 이용해 데이터베이스를 연결하려면 makedsn 함수와 connect 함수를 이용해 데이터베이스 서버의 주소(호스트명), 포트번호, SID, 사용자이름, 비밀번호 등을 설정해야 합니다.

```
cx_Oracle.makedsn(host, port, sid=None)
```

```
cx_Oracle.connect(user=None, password=None, dsn=None)
```

- cx\_Oracle 패키지의 makedsn() 함수와 connect() 함수는 위에 보여진 구문보다 더 많은 파라미터를 가질 수 있습니다. 더 자세한 함수의 정의는 홈페이지<sup>53)</sup>를 참고하세요.

### 2.2. 패키지 설치 및 임포트

오라클 데이터베이스에 연결하려면 패키지를 설치해야 합니다. 다음 명령문은 아나콘다 프롬프트에서 cx\_Oracle 패키지를 설치합니다.

```
(base) C:\Users\COM>conda install cx_Oracle
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.4.10
  latest version: 4.5.4

Please update conda by running
```

53) 패키지 다운로드 : [https://pypi.org/project/cx\\_Oracle/](https://pypi.org/project/cx_Oracle/)  
 문서 : <http://cx-oracle.readthedocs.io/en/latest/>

```

$ conda update -n base conda

## Package Plan ##

environment location: C:\ProgramData\Anaconda3

added / updated specs:
- cx_oracle

The following packages will be downloaded:

package                        |          build
-----|-----
cx_oracle-6.2.1                |  py36hfa6e2cd_0      151 KB

The following NEW packages will be INSTALLED:

cx_oracle: 6.2.1-py36hfa6e2cd_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
cx_oracle 6.2.1: ##### |
100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Users\COM>

```

다음 구문은 패키지를 import 합니다.

```
import cx_Oracle
```

## 2.3. 데이터베이스 연결 객체

다음 코드는 데이터베이스의 연결정보를 입력하고 데이터베이스 연결 객체를 얻습니다.<sup>54)</sup>

54) 예제 코드를 실행하려면 오라클 데이터베이스가 설치되어 있고 사용자 설정이 되어 있어야 합니다.

```
oracle_dsn = cx_Oracle.makedsn(host="localhost", port=1521, sid="xe")
connection = cx_Oracle.connect(user="hr", password="hr", dsn=oracle_dsn)
```

## 2.4. 커서

쿼리문을 실행시키려면 커서(cursor) 객체가 있어야 합니다. cursor 객체의 execute() 함수를 이용해 쿼리문을 실행시킵니다.

```
cursor = connection.cursor()
```

```
sql = "select first_name, last_name from employees"
cursor.execute(sql)
```

```
<cx_Oracle.Cursor on <cx_Oracle.Connection to
hr@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNE
CT_DATA=(SID=xe)))>>
```

```
for name in cursor:
    print(name)
```

```
('Ellen', 'Abel')
('Sundar', 'Ande')
('Mozhe', 'Atkinson')
('David', 'Austin')
('Hermann', 'Baer')
('Shelli', 'Baida')
...생략
('Clara', 'Vishney')
('Shanta', 'Vollman')
('Alana', 'Walsh')
('Matthew', 'Weiss')
('Jennifer', 'Whalen')
('Eleni', 'Zlotkey')
```

```
cursor.execute("select * from employees")
```

```
<cx_Oracle.Cursor on <cx_Oracle.Connection to
```

employees 테이블은 hr 계정에 디폴트로 포함되어 있습니다.

```
hr@ (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNECT_DATA=(SID=x)))>>
```

```
for emp in cursor:
    print(emp)
```

```
(100, 'Steven', 'King', 'SKING', '515.123.4567', datetime.datetime(2003,
6, 17, 0, 0), 'AD_PRES', 30000.0, 0.0, None, 90)
(101, 'Neena', 'Kochhar', 'NKOCHHAR', '515.123.4568',
datetime.datetime(2005, 9, 21, 0, 0), 'AD_VP', 17000.0, None, 100, 90)
(102, 'Lex', 'De Haan', 'LDEHAAN', '515.123.4569', datetime.datetime(2001,
1, 13, 0, 0), 'AD_VP', 17000.0, None, 100, 90)
... 생략
(203, 'Susan', 'Mavris', 'SMAVRIS', '515.123.7777',
datetime.datetime(2002, 6, 7, 0, 0), 'HR_REP', 6500.0, None, 101, 40)
(204, 'Hermann', 'Baer', 'HBAER', '515.123.8888', datetime.datetime(2002,
6, 7, 0, 0), 'PR_REP', 10000.0, None, 101, 70)
(205, 'Shelley', 'Higgins', 'SHIGGINS', '515.123.8080',
datetime.datetime(2002, 6, 7, 0, 0), 'AC_MGR', 12008.0, None, 101, 110)
(206, 'William', 'Gietz', 'WGIETZ', '515.123.8181',
datetime.datetime(2002, 6, 7, 0, 0), 'AC_ACCOUNT', 8300.0, None, 205, 110)
```

파라미터를 이용해 쿼리문에 값을 전달할 수 있습니다. 파라미터의 변수는 쿼리문 안에서 : 을 붙여 사용합니다. 다음 구문은 50번 부서의 사원 정보만 출력합니다.

```
dept_id = 50
cursor.execute("select * from employees where department_id = :dept_id",
               dept_id=50)
```

```
<cx_Oracle.Cursor on <cx_Oracle.Connection to
hr@ (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNECT_DATA=(SID=x)))>>
```

```
for emp in cursor:
    print(emp)
```

```
(120, 'Matthew', 'Weiss', 'MWEISS', '650.123.1234',
datetime.datetime(2004, 7, 18, 0, 0), 'ST_MAN', 8000.0, None, 100, 50)
(121, 'Adam', 'Fripp', 'AFRIPP', '650.123.2234', datetime.datetime(2005,
4, 10, 0, 0), 'ST_MAN', 8200.0, None, 100, 50)
(122, 'Payam', 'Kaufling', 'PKAUFLIN', '650.123.3234',
datetime.datetime(2003, 5, 1, 0, 0), 'ST_MAN', 7900.0, None, 100, 50)
... 생략
(196, 'Alana', 'Walsh', 'AWALSH', '650.507.9811', datetime.datetime(2006,
```

```
4, 24, 0, 0), 'SH_CLERK', 3100.0, None, 124, 50)
(197, 'Kevin', 'Feeney', 'KFEENEY', '650.507.9822',
datetime.datetime(2006, 5, 23, 0, 0), 'SH_CLERK', 3000.0, None, 124, 50)
(198, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833',
datetime.datetime(2007, 6, 21, 0, 0), 'SH_CLERK', 2600.0, None, 124, 50)
(199, 'Douglas', 'Grant', 'DGRANT', '650.507.9844',
datetime.datetime(2008, 1, 13, 0, 0), 'SH_CLERK', 2600.0, None, 124, 50)
```

쿼리문에 전달할 파라미터가 딕셔너리 형식으로 정의되어 있다면 딕셔너리 인수 언패킹을 이용해 쿼리문에 딕셔너리 객체의 데이터를 전달할 수 있습니다.

```
params = {"dept_id":50, "mgr_id":124}
cursor.execute("select * from employees where department_id=:dept_id and
manager_id=:mgr_id",**params)
```

```
<cx_Oracle.Cursor on <cx_Oracle.Connection to
hr@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNE
CT_DATA=(SID=xex)))>>
```

```
for emp in cursor:
    print(emp)
```

```
(141, 'Trenna', 'Rajs', 'TRAJS', '650.121.8009', datetime.datetime(2003,
10, 17, 0, 0), 'ST_CLERK', 3500.0, None, 124, 50)
(142, 'Curtis', 'Davies', 'CDAVIES', '650.121.2994',
datetime.datetime(2005, 1, 29, 0, 0), 'ST_CLERK', 3100.0, None, 124, 50)
(143, 'Randall', 'Matos', 'RMATOS', '650.121.2874',
datetime.datetime(2006, 3, 15, 0, 0), 'ST_CLERK', 2600.0, None, 124, 50)
(144, 'Peter', 'Vargas', 'PVARGAS', '650.121.2004',
datetime.datetime(2006, 7, 9, 0, 0), 'ST_CLERK', 2500.0, None, 124, 50)
(196, 'Alana', 'Walsh', 'AWALSH', '650.507.9811', datetime.datetime(2006,
4, 24, 0, 0), 'SH_CLERK', 3100.0, None, 124, 50)
(197, 'Kevin', 'Feeney', 'KFEENEY', '650.507.9822',
datetime.datetime(2006, 5, 23, 0, 0), 'SH_CLERK', 3000.0, None, 124, 50)
(198, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833',
datetime.datetime(2007, 6, 21, 0, 0), 'SH_CLERK', 2600.0, None, 124, 50)
(199, 'Douglas', 'Grant', 'DGRANT', '650.507.9844',
datetime.datetime(2008, 1, 13, 0, 0), 'SH_CLERK', 2600.0, None, 124, 50)
```

커서를 통해 조회한 데이터베이스 정보를 데이터프레임 형식으로 변환하면 데이터를 분석하기 더 쉽습니다. pandas 패키지의 DataFrame 패키지를 이용하면 커서로부터 조회한 데이터를 데이터프레임 객체로 변환할 수 있습니다.

```
cursor.execute("select * from employees")
```

```
<cx_Oracle.Cursor on <cx_Oracle.Connection to  
hr@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNE  
CT_DATA=(SID=x)))>>>
```

```
from pandas import DataFrame
```

```
emp = DataFrame(cursor.fetchall())  
cursor.rowcount
```

```
107
```

```
cursor.description[0]
```

```
('EMPLOYEE_ID', cx_Oracle.NUMBER, 7, None, 6, 0, 0)
```

```
col_names = [i[0] for i in cursor.description]  
emp.columns = col_names  
DataFrame.head(emp)
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	\
0	100	Steven	King	SKING	515.123.4567	2003-06-17	
1	101	Neena	Kochhar	NKOCHHAR	515.123.4568	2005-09-21	
2	102	Lex	De Haan	LDEHAAN	515.123.4569	2001-01-13	
3	103	Alexander	Hunold	AHUNOLD	590.423.4567	2006-01-03	
4	104	Bruce	Ernst	BERNST	590.423.4568	2007-05-21	

	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
0	AD_PRES	30000.0	0.0	NaN	90.0
1	AD_VP	17000.0	NaN	100.0	90.0
2	AD_VP	17000.0	NaN	100.0	90.0
3	IT_PROG	9000.0	NaN	102.0	60.0
4	IT_PROG	6000.0	NaN	103.0	60.0

### 3. MySQL 데이터베이스 연결

MySQL 데이터베이스 연결은 PyMySQL 패키지를 이용합니다. connection 객체를 생성하고 커서를 통해 데이터를 조회하는 방법은 오라클의 cx\_Oracle을 사용할 때와 같습니다. 자세한 내용은 공식사이트<sup>55)</sup>를 참고하세요.

MariaDB 데이터베이스를 설치하고 데이터베이스에 연결해 보겠습니다.

#### 1) 데이터베이스 다운로드 및 설치

MariaDB 데이터베이스는 <https://downloads.mariadb.org/>에서 설치파일을 다운로드 합니다. 윈도우의 경우 msi파일(mariadb-10.2.12-winx64.msi<sup>56)</sup>)을 다운로드 받으면 설치파일을 더블클릭하면 쉽게 설치할 수 있습니다. 설치 시 root 사용자 비밀번호만 설정하면 됩니다.

#### 2) 사용자 생성

MariaDB 설치 후 사용할 계정을 생성합니다. 데이터베이스 설치 후 시작 프로그램에서 [Command Prompt]를 실행시키세요.

```
C:\windows\system32>mysql -u root -p
Enter password: ***** <--설치 시 입력한 관리자 비밀번호 입력
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 11
Server version: 10.2.12-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]> create database testdb;
Query OK, 1 row affected (0.00 sec)
```

55) 패키지 다운로드 : <https://pypi.org/project/PyMySQL/>

도큐먼트 : <https://pymysql.readthedocs.io/en/latest/>

56) <https://downloads.mariadb.org/mariadb/>에서 다운로드 버튼을 클릭하세요. 버전은 이책과 다를 수 있습니다.

```

MariaDB [(none)]> create user 'user01'@'%' identified by 'user123';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> grant all privileges on testdb.* to 'user01'@'%';
Query OK, 0 rows affected (0.00 sec)

MariaDB [testdb]> exit;
Bye

```

### 3) 예제 테이블 및 데이터 추가

새로 생성한 사용자 계정으로 접속하여 예제 테이블 및 데이터를 생성해 줍니다.

```

C:\windows\system32>mysql -u user01 -p
Enter password: ***** # user123
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 11
Server version: 10.2.12-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]> use testdb
Database changed

```

공유폴더에서 sql 파일(MySQL\_HR\_DDL.sql<sup>57)</sup>, MySQL\_HR\_DML.sql<sup>58)</sup>) 다운로드해서 C:\W에 복사해 주세요.

그리고 MariaDB 프롬프트에서 아래 내용 입력해서 sql 파일을 실행시키세요. 테이블이 생성되는 도중에 경고가 있어도 무시하세요. MySQL\_HR\_DDL.sql 파일은 오라클 데이터베이스에 포함되어 있는 HR 스키마의 테이블들을 생성하는 구문입니다.

MySQL\_HR\_DML.sql 파일은 샘플 데이터를 insert 하는 구문입니다.

```

MariaDB [testdb]> source C:/MySQL_HR_DDL.sql
Query OK, 0 rows affected, 1 warning (0.01 sec)

```

57) <http://jasvaspecialist.co.kr/pds/340>

58) <http://jasvaspecialist.co.kr/pds/341>



```

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0

... 생략...

Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

MariaDB [testdb]> source C:/MySQL_HR_DML.sql
Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

... 생략...

Query OK, 1 row affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

```

예제 테이블이 생성되고 데이터가 입력되면 commit; 문으로 변경사항을 저장하고 select 문으로 employees 테이블에 저장되어 있는 행의 수를 조회해 보세요. 결과가 107이 출력되어야 합니다.

```

MariaDB [testdb]> commit;

MariaDB [testdb]> select count(*) from employees;
+-----+
| count(*) |
+-----+
|      107 |

```

```

+-----+
1 row in set (0.00 sec)
MariaDB [testdb]> exit;
Bye

```

### 3.2. MariaDB 데이터베이스 정보 조화

MySQL 데이터베이스에 연결하는 예입니다.

```
import pymysql
```

```

connection = pymysql.connect(host='localhost',
                             user='user',
                             password='passwd',
                             db='db',
                             charset='utf8')

```

```

try:
    with connection.cursor() as cursor:
        sql = "INSERT INTO `users` (`email`, `password`) VALUES (%s, %s)"
        cursor.execute(sql, ('webmaster@python.org', 'very-secret'))
        connection.commit()

    with connection.cursor() as cursor:
        sql = "SELECT `id`, `password` FROM `users` WHERE `email`=%s"
        cursor.execute(sql, ('webmaster@python.org',))
        result = cursor.fetchone()
        print(result)
finally:
    connection.close()

```

```
{'password': 'very-secret', 'id': 1}
```

## 4. 연습문제

---

### 1) 주소록 애플리케이션

- 파이썬으로 고객관리 애플리케이션을 작성하세요.
- 고객 정보는 이름, 전화번호, 나이, 고객 등급(1~5), 이메일, 특징입니다.
- 고객 클래스를 이용하여 정보를 저장하고 리스트를 이용하여 관리해야 합니다.
- 고객 정보들은 파일에 저장하고 불러올 수 있어야 합니다.(피클 사용)
- 리스트 데이터는 CSV 파일로 내보내기 기능이 있어야 합니다.
- 고객의 정보는 입력, 고객정보 전체 조회, 검색(이름으로 검색), 삭제(이름으로) 할 수 있어야 합니다.

정답 및 풀이

1)

```
>>> class Customer:
...     def __init__(self, name, phone, email, age, grade, etc):
...         self.name = name
...         self.phone = phone
...         self.email = email
...         self.age = age
...         self.grade = grade
...         self.etc = etc
...
...     def print_info(self):
...         print('{:>5} {:<3} {:<15} {:<15} {:>3}'.format('*'*self.grade,
...             self.name, self.phone, self.email, self.age), self.etc,
...             sep=' ', end='\n')
...
... 
```

```
>>> def print_customer(cust_list):
...     print("=====")
...     print("                고객 정보 리스트")
...     print("-----")
...     print('{:<5} {:<6} {:<15} {:<15} {:>3} {:'
...         .format("GRADE", "NAME ", "PHONE", "EMAIL", "AGE", "ETC"))
...     print("=====")
...     for cust in cust_list:
...         cust.print_info()
...     print("=====")
... 
```

```
>>> def search_customer(cust_list, name):
...     search_list = []
...     for i, cust in enumerate(cust_list):
...         if cust.name == name:
...             search_list.append(cust_list[i])
...     return search_list;
... 
```

```
>>> def delete_customer(cust_list, name):
...     for i, cust in enumerate(cust_list):
...         if cust.name == name:
...             del cust_list[i]
...             break
... 
```

```
>>> def insert_customer_info():
...     name = input("이름: ")
... 
```

```
...     phone = input("전화번호: ")
...     email = input("이메일: ")
...     age = input("나이: ")
...     grade = input("고객등급: ")
...     etc = input("기타정보: ")
...     cust = Customer(name, phone, email, age, int(grade), etc)
...     return cust
... 
```

```
>>> def save_customer(cust_list):
...     try:
...         f = open("customer_db.data", "wb")
...         import pickle
...         pickle.dump(cust_list, f)
...     except Exception as e:
...         print("데이터 저장 예외 :", e)
...     finally:
...         f.close()
... 
```

```
>>> def load_customer():
...     cust_list = []
...     try:
...         f = open("customer_db.data", "rb")
...         import pickle
...         cust_list = pickle.load(f)
...     except Exception as e:
...         print("데이터 로드 예외 :", e)
...     finally:
...         try:
...             f.close()
...         except Exception:
...             pass
...     return cust_list
... 
```

```
>>> def print_menu():
...     print("1.입력", "2.전체조회", "3.삭제", "4.찾기", "5.내보내기(CSV)", "9. 종료",
...           sep=" | ", end="")
...     menu = input("메뉴선택: ")
...     return int(menu)
... 
```

```
>>> def save_csv(cust_list):
...     try:
...         file_name = input("저장할 파일 이름을 입력하세요.")
... 
```

```

...     f = open(file_name, "wt")
...     for cust in cust_list:
...         print(cust.name, cust.phone, cust.email, cust.age, cust.grade,
...             cust.etc,
...                 sep=',', end="\n", file=f)
...     except Exception as e:
...         print("예외발생 :", e)
...     else:
...         print("CSV 파일이 저장되었습니다. 파일명:", file_name)
...     finally:
...         f.close()
...

```

```

>>> def main():
...     cust_list = []
...
...     try:
...         cust_list = load_customer()
...     except OSError as e:
...         print(e)
...     else:
...         print("데이터파일이 로드되었습니다.")
...
...     while 1:
...         menu = print_menu()
...         if menu == 1:
...             cust = insert_customer_info()
...             cust_list.append(cust)
...         elif menu == 2:
...             print_customer(cust_list)
...         elif menu == 3:
...             name = input("삭제할 이름을 입력하세요.")
...             delete_customer(cust_list, name)
...         elif menu == 4:
...             name = input("찾을 고객 이름을 입력하세요.")
...             print_customer(search_customer(cust_list, name))
...         elif menu == 5:
...             save_csv(cust_list)
...         elif menu == 9:
...             save_customer(cust_list)
...         break
...

```

```

>>> if __name__ == '__main__':
...     main()

```

## 11장. 웹 데이터 수집

## 1. 웹 데이터 수집

### 1.1. BeautifulSoup

뷰티풀수프는 웹페이지 데이터를 수집하는 것 같은 스크린 스크래핑(screen-scraping) 프로젝트를 위해 설계된 파이썬 라이브러리입니다. 다음은 BeautifulSoup의 3가지 특징입니다.

1. BeautifulSoup은 구문 분석, 트리 탐색, 검색 및 수정을 위한 몇 가지 간단한 방법과 Pythonic 관용구를 제공합니다. 문서를 분석하고 필요한 것을 추출하는 툴킷입니다. 응용 프로그램을 작성하는 데 많은 코드가 필요하지 않습니다.
2. BeautifulSoup은 들어오는 문서를 유니코드로 보내고 문서를 UTF-8로 자동 변환합니다. 문서에서 인코딩을 지정하지 않고 BeautifulSoup에서 인코딩을 감지 할 수 없다면 인코딩에 대해 생각할 필요가 없습니다. 그런 다음 원래 인코딩을 지정하면 됩니다.
3. BeautifulSoup은 lxml 및 html5lib 과 같은 인기 있는 파이썬 파서 위에 위치하여 유연성을 위해 다양한 파싱 전략이나 거래 속도를 시도 할 수 있습니다.

현재 뷰티풀수프의 버전은 4입니다. 뷰티풀수프 3은 더 이상 개발되지 않고 모든 새 프로젝트에 뷰티풀수프 4가 권장됩니다. 뷰티풀수프 3과 뷰티풀수프 4의 차이점에 대해 알아보려면 BS4로 코드를 포팅(porting)하는 문서<sup>59)</sup>를 참조하세요.

뷰티풀수프 버전 4의 패키지 이름은 BeautifulSoup에서 bs4로 변경되었습니다. 그리고 그 외에 많은 메서드들의 이름이 바뀌었습니다. 뷰티풀수프 4는 파이썬 2.7 이상과 파이썬 3 버전에서 동작합니다.

다음은 뷰티풀수프 공식 사이트와 도큐먼트 페이지입니다.

- 공식 사이트 : <https://www.crummy.com/software/BeautifulSoup/>
- Documentation : <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

뷰티풀수프를 이용하려면 패키지를 설치해야 합니다.

```
pip install beautifulsoup4
```

\* 여러분이 아나콘다를 설치해 사용하고 있다면 이 패키지는 다시 설치할 필요 없습니다.

59) <https://www.crummy.com/software/BeautifulSoup/bs4/doc/#porting-code-to-bs4>



## 1.2. 파서 라이브러리

다음 표는 뷰티풀소프에서 사용하는 파서들의 장/단점을 설명합니다.

표 1. 파서 라이브러리

파서	사용법	장점	단점
Python's html.parser	BeautifulSoup(markup, "html.parser")	<ul style="list-style-type: none"> <li>Batteries included</li> <li>Decent speed</li> <li>Lenient (as of Python 2.7.3 and 3.2.)</li> </ul>	<ul style="list-style-type: none"> <li>Not very lenient (before Python 2.7.3 or 3.2.2)</li> </ul>
lxml's HTML parser	BeautifulSoup(markup, "lxml")	<ul style="list-style-type: none"> <li>Very fast</li> <li>Lenient</li> </ul>	<ul style="list-style-type: none"> <li>External C dependency</li> </ul>
lxml's XML parser	BeautifulSoup(markup, "lxml-xml") BeautifulSoup(markup, "xml")	<ul style="list-style-type: none"> <li>Very fast</li> <li>The only currently supported XML parser</li> </ul>	<ul style="list-style-type: none"> <li>External C dependency</li> </ul>
html5lib	BeautifulSoup(markup, "html5lib")	<ul style="list-style-type: none"> <li>Extremely lenient</li> <li>Parses pages the same way a web browser does</li> <li>Creates valid HTML5</li> </ul>	<ul style="list-style-type: none"> <li>Very slow</li> <li>External Python dependency</li> </ul>

## 1.3. Selector API

BeautifulSoup은 가장 일반적으로 사용되는 CSS 선택자를 지원합니다. 문자열을 태그 객체의 `.select()` 메서드 또는 BeautifulSoup 객체 자체에 전달하기 만하면 됩니다.

BeautifulSoup의 Selector API는 `select()`와 `select_one()` 이 있습니다.

- `soup.select("CSS 선택자")` : CSS 선택자에 해당하는 모든 요소를 반환합니다.
- `soup.select_one("CSS 선택자")` : CSS 선택자에 맞는 오직 첫 번째 태그 요소만 반환합니다.

BeautifulSoup에서 가장 중요한 메서드는 `select()`와 `select_one()`입니다. 그러나 그 외에도 돋트리(DOM Tree)의 구조를 변경시킬 수 있는 많은 메서드들이 있으므로 [도큐먼트<sup>60\)</sup>](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)를 참고하시기 바랍니다.

60) <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

## 1.4. 환율 정보 가져오기

다음 코드는 네이버 환율 정보 사이트에서 usd/krw 환율 정보를 출력하는 예입니다.

```
import requests
from bs4 import BeautifulSoup
```

환율 정보를 제공하는 URL을 GET 방식으로 요청합니다.

```
url = 'https://finance.naver.com/marketindex/' # 환율 정보
market_index = requests.get(url)
```

요청 후 응답(response)이 바르게 되었는지 확인합니다. 응답 코드 [200]은 서버의 응답이 올바르게 되었다는 것을 의미합니다. [404]가 나오면 URL이 잘 못된 것입니다.

```
market_index
```

```
<Response [200]>
```

HTML 파서를 이용해 응답 콘텐츠를 파싱합니다.

```
soup = BeautifulSoup(market_index.content, "html.parser")
```

파싱한 soup 객체에서 CSS 선택자를 이용해 엘리먼트를 찾습니다.

```
price = soup.select_one("div.head_info > span.value")
print(price)
```

```
<span class="value">1,135.50</span>
```

엘리먼트의 내용은 text 속성 또는 string 속성을 통해 알 수 있습니다.

```
print("usd/krw=", price.text)
```

```
usd/krw= 1,132.50
```

## 1.5. 네이버 영화 랭킹 출력하기

이번에는 네이버에서 제공하는 영화 랭킹을 출력해 보겠습니다.

```
import requests
from bs4 import BeautifulSoup
```

영화 랭킹 정보를 제공하는 URL을 get 방식으로 요청합니다.

```
url = 'https://movie.naver.com/movie/sdb/rank/rmovie.nhn'
movie_ranking = requests.get(url)
movie_ranking
```

```
<Response [200]>
```

HTML 파서를 이용해 응답 내용을 파싱해 soup 객체로 만듭니다.

```
soup = BeautifulSoup(movie_ranking.content, 'html.parser')
```

```
title_list = soup.select('div.tit3 > a')
```

```
title_list[0]
```

```
<a href="/movie/bi/mi/basic.nhn?code=168058" title="퍼스트맨">퍼스트맨</a>
```

영화 랭킹은 한 개가 아니므로 반복문을 이용해 처리해야 합니다.

```
for i, title in enumerate(title_list):
    print('{}위: {}'.format(i+1, title.text))
```

```
1위: 퍼스트맨
2위: 미쓰백
3위: 암수살인
4위: 베놈
5위: 스타 이즈 본
6위: 박화영
7위: 명당
8위: 안시성
9위: 협상
10위: 창궐
```

```

11위: 배반의 장미
12위: 액슬
13위: 서치
14위: 물괴
15위: 곰돌이 푸 다시 만나 행복해
16위: 원더풀 고스트
17위: 여곡성
18위: 82년생 김지영
19위: 완벽한 타인
20위: 펭귄 하이웨이
21위: 그랜드 부다페스트 호텔
22위: 마라
23위: 핫 썸머 나이트
24위: 국가부도의 날
25위: 호밀밭의 반항아
26위: 킨: 더 비기닝
27위: 나를 차버린 스파이
28위: 노크: 초대받지 않은 손님
29위: 배드 사마리안
30위: 신비한 동물들과 그린델왈드의 범죄
31위: 상류사회
32위: 성난황소
33위: 다이노 어드벤처2: 육해공 공룡 대백과
34위: 해리 포터와 마법사의 돌
35위: 보헤미안 랩소디
36위: 에브리데이
37위: 할로윈
38위: 그놈이다
39위: 인피니티 포스 : 독수리오형제 최후의 심판
40위: 동네사람들
41위: 리즈와 파랑새
42위: 맥퀸
43위: 크레이지 리치 아시안
44위: 무뢰한
45위: 적인길 3: 사대천왕
46위: 타샤 튜더
47위: 너의 결혼식
48위: 업그레이드
49위: 블랙 47
50위: 군산: 거위를 노래하다

```

엘리먼트의 속성을 읽으려면 `attrs['속성명']`을 이용하세요. 위의 for 반복문은 다음처럼 작성할 수 있습니다.

```

for i, title in enumerate(title_list):
    print("{}위: {}".format(i+1, title.attrs['title']))

```

## 2. 연습문제

- 1) yes24의 베스트셀러 정보를 제공하는 사이트에서 베스트셀러 정보를 수집해서 파일에 저장하세요.

베스트셀러 주소 : <http://www.yes24.com/24/category/bestseller>

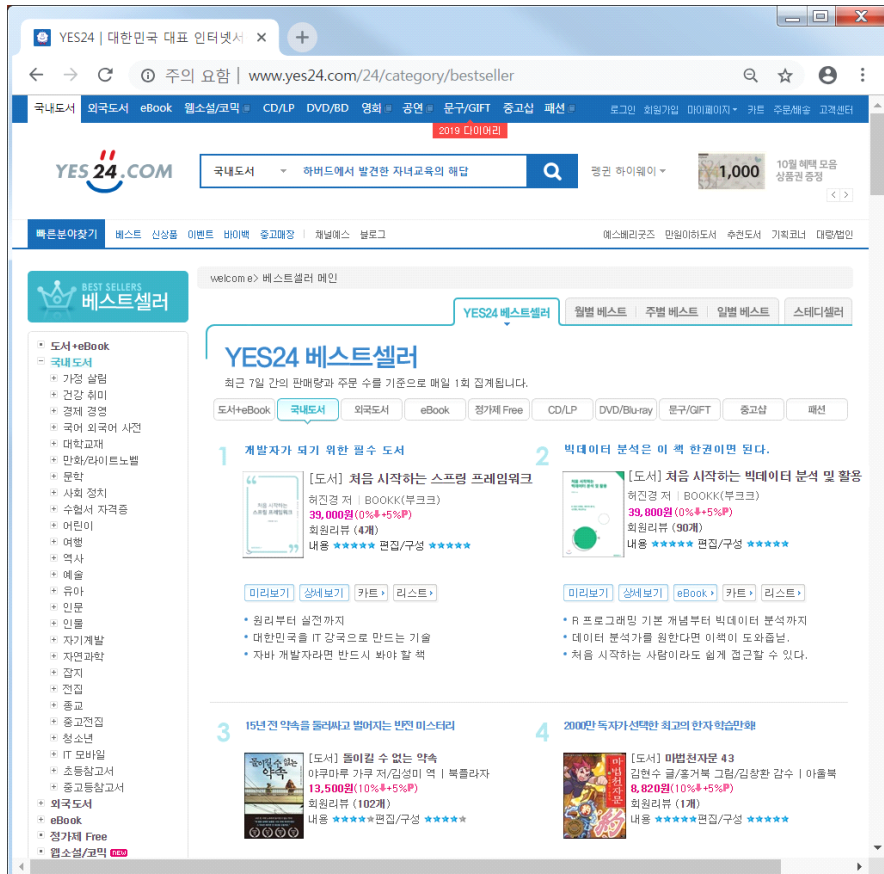


그림 1. YES24 베스트셀러

정답

```
import requests
import codecs
from bs4 import BeautifulSoup
```

```
bestseller = requests.get('http://www.yes24.com/24/category/bestseller')
soup = BeautifulSoup(bestseller.content, "html.parser")
```

```
###도서 목록을 받아올 리스트를 선언
book_titles=[]
book_authors=[]
book_prices=[]
```

```
###각 리스트에 항목을 찾아서 순차적으로 append
for i in range(1,11):
    book_titles.append(soup.find("ol", class_="")
                        .find("li", class_="num"+str(i))
                        .find_all("p")[2].text)
```

```
for i in range(1,11):
    book_authors.append(soup.find("ol", class_="")
                        .find("li", class_="num"+str(i))
                        .find("p", class_="au pu").text)
```

```
for i in range(1,11):
    book_prices.append(soup.find("ol", class_="")
                       .find("li", class_="num"+str(i))
                       .find("p", class_="price").text)
```

```
#받아온 값 콘솔에 찍어보기
for i in range(0,10):
    print("<Best Rank "+str(i+1)+">")
    print(book_titles[i])
    print(book_authors[i])
    print(book_prices[i]+"\n")
```

```
#파일 열기
f=codecs.open("yes24_best10.txt",encoding="utf-8",mode="w")
```

```
#헤더 입력
f.write("순위,제목,저자&출판사,가격\r\n") #윈도우 줄바꿈은 2바이트 \r\n
```

```
#항목입력
```

```
for i in range(0,10):
    f.write("{} , {} , {}\n".format(str(i+1), book_titles[i],
    book_authors[i], book_prices[i]))
f.close()
```

<Best Rank 1>

[도서] 처음 시작하는 스프링 프레임워크  
허진경 저 | B00KK(부크크)  
39,000원(0%)

<Best Rank 2>

[도서] 처음 시작하는 빅데이터 분석 및 활용  
허진경 저 | B00KK(부크크)  
39,800원(0%)

<Best Rank 3>

[도서] 돌이킬 수 없는 약속  
야쿠마루 가쿠 저/김성미 역 | 북플라자  
13,500원(10%+5%)

<Best Rank 4>

[도서] 마법천자문 43  
김현수 글/홍거북 그림/김창환 감수 | 아울북  
8,820원(10%+5%)

<Best Rank 5>

[도서] 골든아워 1  
이국종 저 | 흐름출판  
14,220원(10%+5%)

<Best Rank 6>

[도서] 마흔에게  
기시미 이치로 저/전경아 역 | 다산초당  
12,600원(10%+5%)

<Best Rank 7>

[도서] 열두 발자국  
정재승 저 | 어크로스  
15,120원(10%+5%)

<Best Rank 8>

[도서] [예약판매] 백일의 낭군님 포토에세이  
백일의 낭군님 제작팀 저 | artePOP(아르테팝)  
19,800원(10%+5%)

<Best Rank 9>

[도서] 움츠러들지 않고 용기있게 딸 성교육 하는 법  
손경이 저 | 다산에듀  
13,500원(10%+5%)

<Best Rank 10>  
[도서] 골든아워 2  
이국종 저 | 흐름출판  
14,220원(10%+5%)







---

출판사: (주)KG아이티뱅크

---

금 액: 20,000원

---

