

데이터분석을 위한 파이썬



PART I

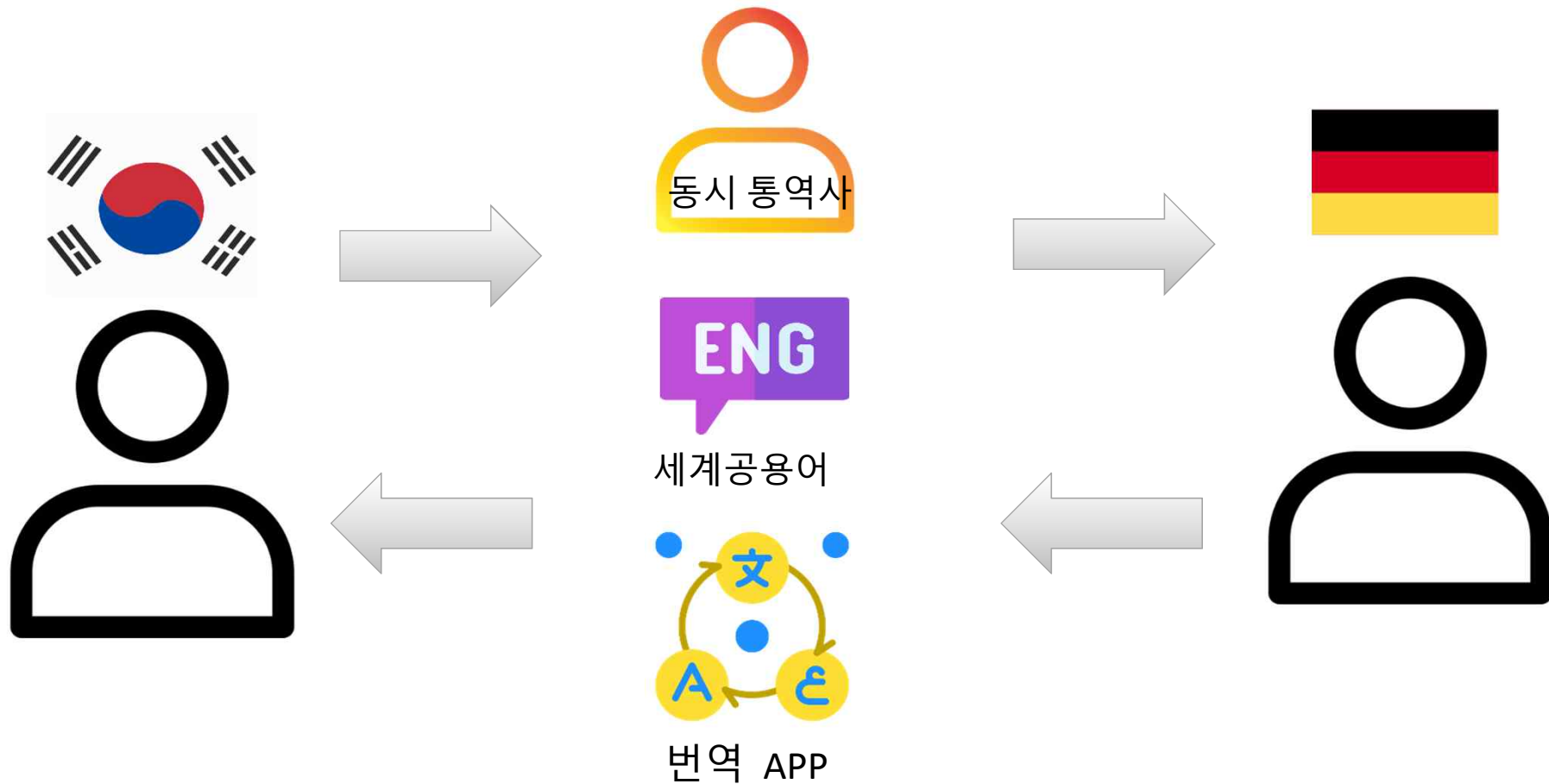
데이터 분석을 위한

PYTHON

CH01. 프로그래밍 살펴보기

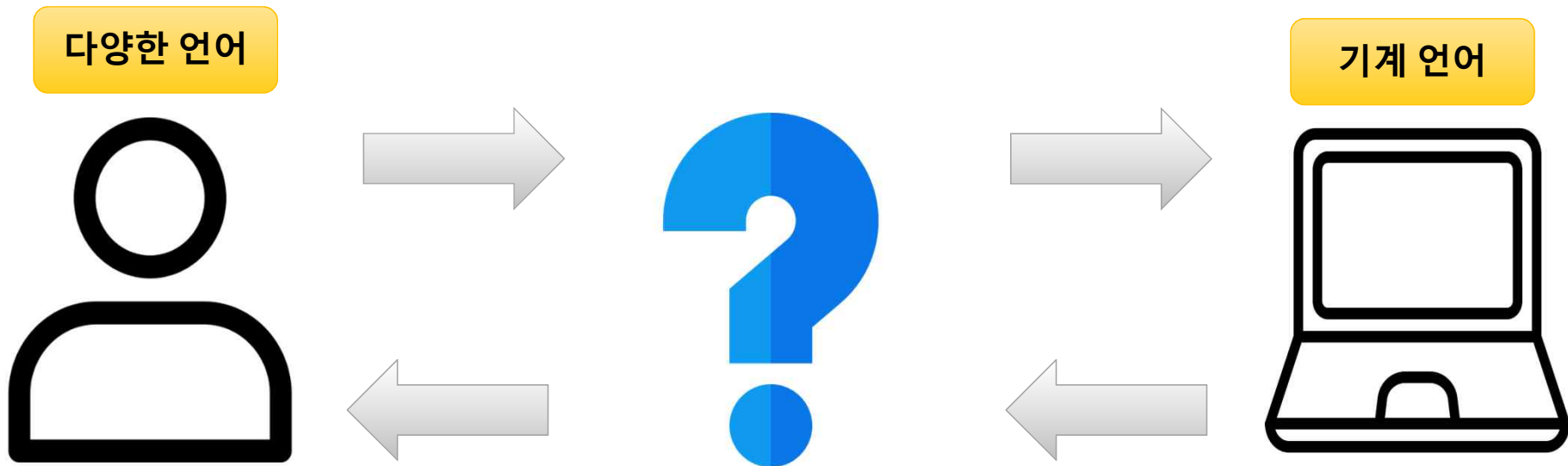
프로그래밍 살펴보기

◆ 사람과 사람 사이 언어



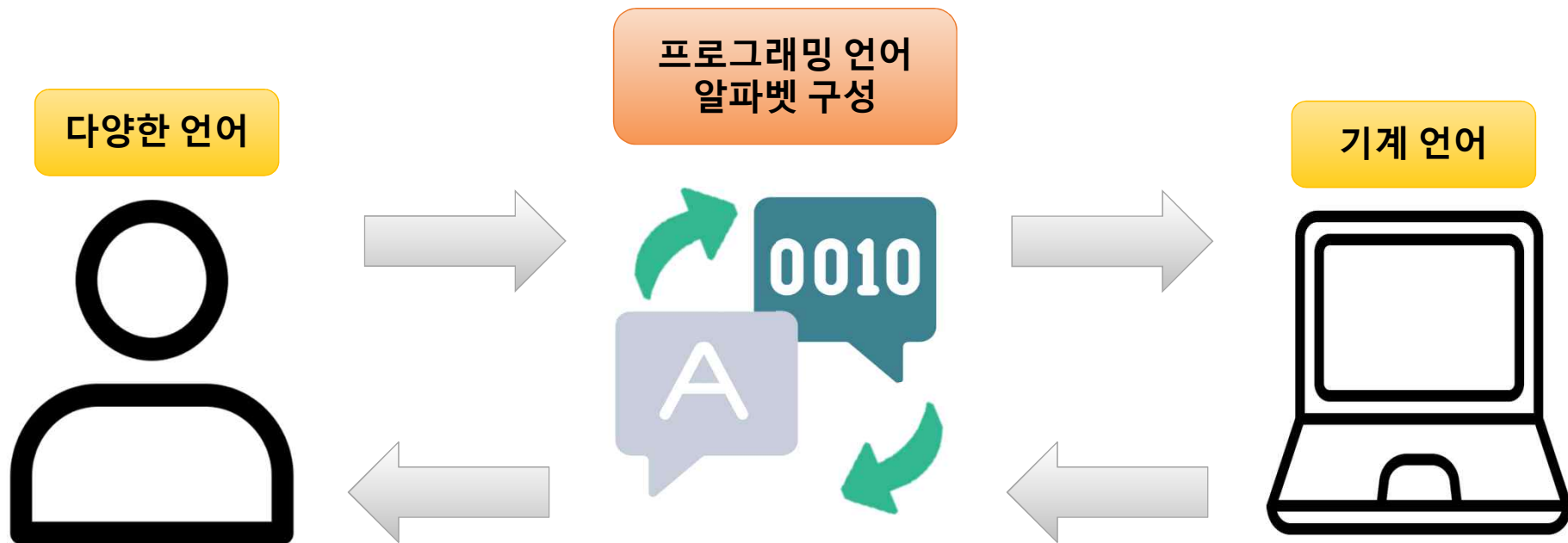
프로그래밍 살펴보기

◆ 사람과 컴퓨터 사이 언어



프로그래밍 살펴보기

◆ 사람과 컴퓨터 사이 언어



프로그래밍 살펴보기

◆ 프로그래밍 언어



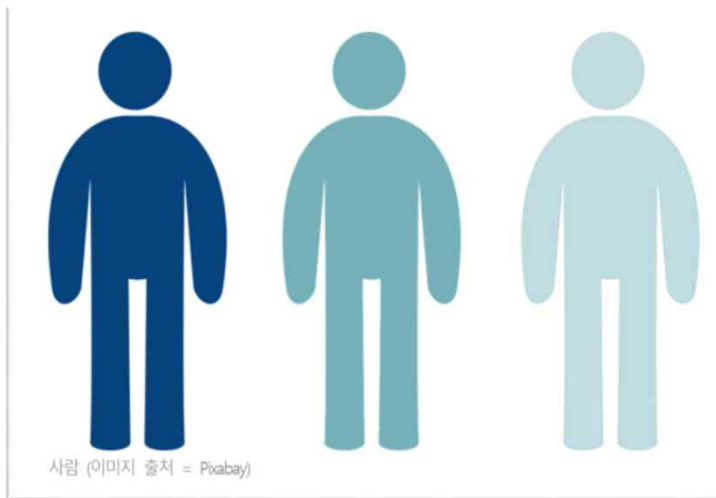
프로그래밍 살펴보기

◆ 프로그래밍(코딩) 이유

해결 해야할 일
1부터 100,000,000사이
곱하기

프로그램 / 프로그래밍

- 프로그래밍 언어
사용해서 컴퓨터에서
시킬일을 기록한 것



프로그래밍 살펴보기

◆ 프로그래밍(코딩) 이유

ㄱ ~ ㅎ
ㅏ ~ ㅣ
가 ~ 하
가방 엄마 아빠

문법+오늘은 학교에
갑니다.

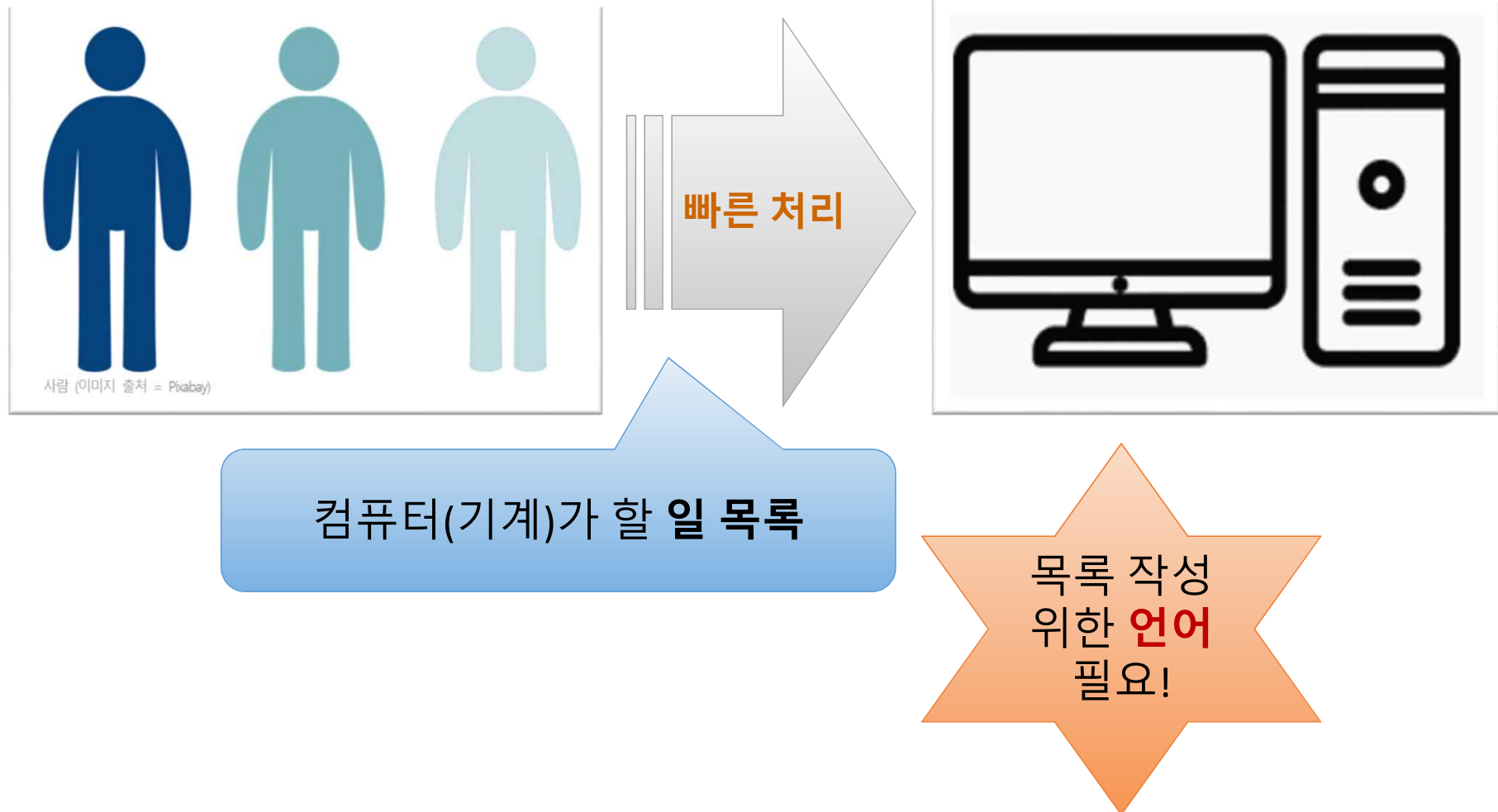
ㄱ ~ ㅎ
ㅏ ~ ㅣ
가 ~ 하

가방 엄마 아빠

문법 +오늘은 학교에
갑니다.

프로그래밍 살펴보기

◆ 프로그래밍 준비

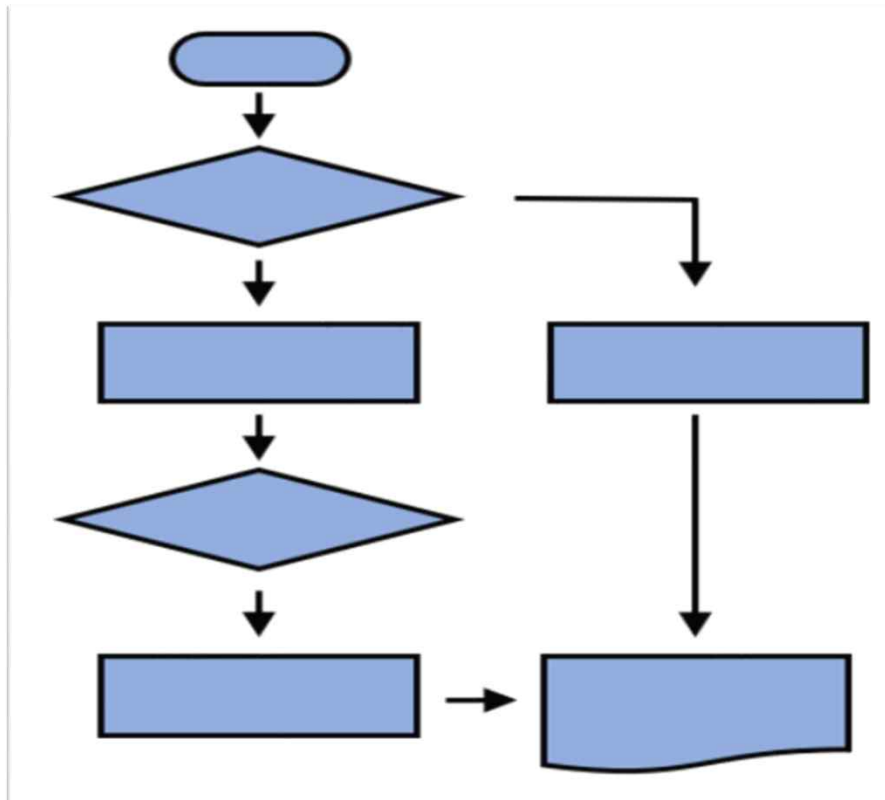


프로그래밍 살펴보기

◆ 프로그래밍 방법

절차적(구조적) 프로그래밍

- 작업 순서를 표현하는 컴퓨터 명령 집합
- 함수들의 집합으로 프로그램 작성



데이터

함수

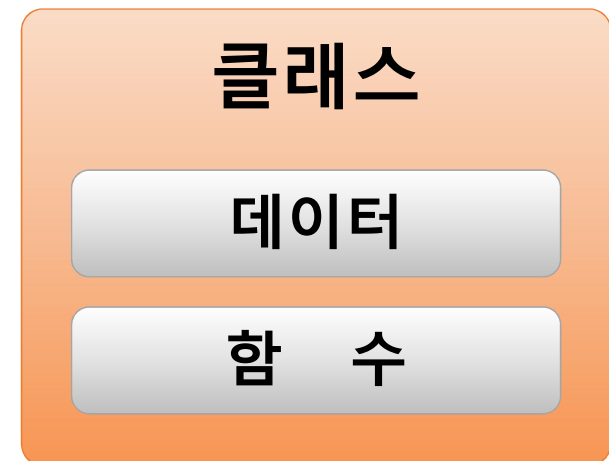
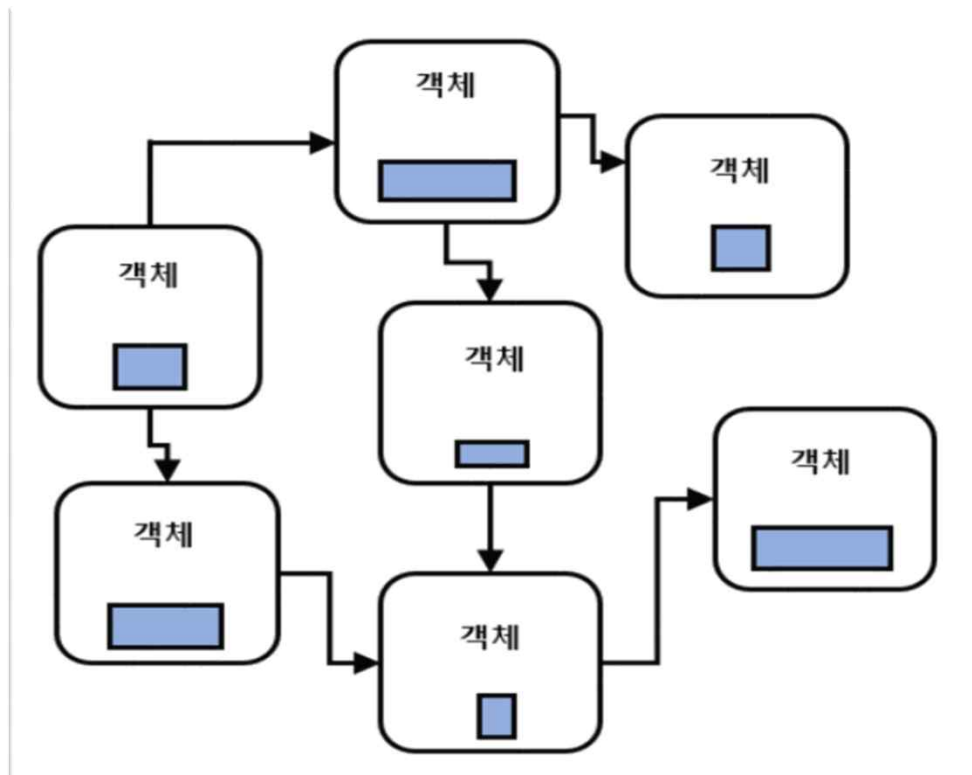
순서/선택/반복

프로그래밍 살펴보기

◆ 프로그래밍 방법

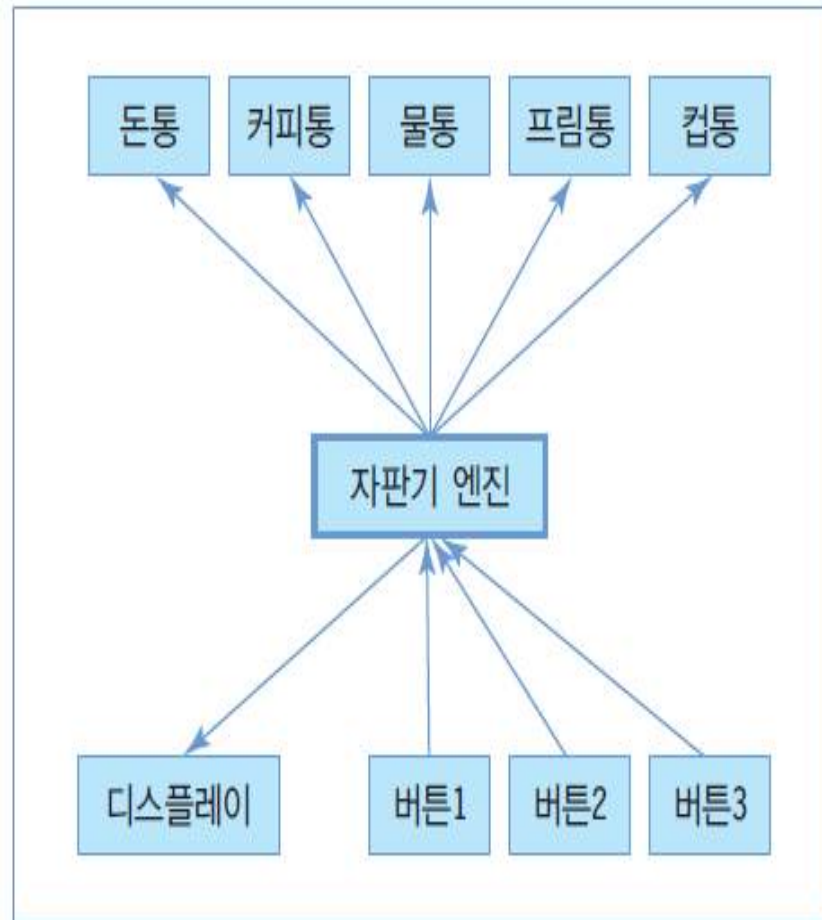
객체 지향 프로그래밍

- 프로그램을 실제 세상에 가깝게 모델링
- 컴퓨터가 수행하는 작업을 객체들간의 상호 작용으로 표현
- 클래스 혹은 객체들의 집합으로 프로그램 작성

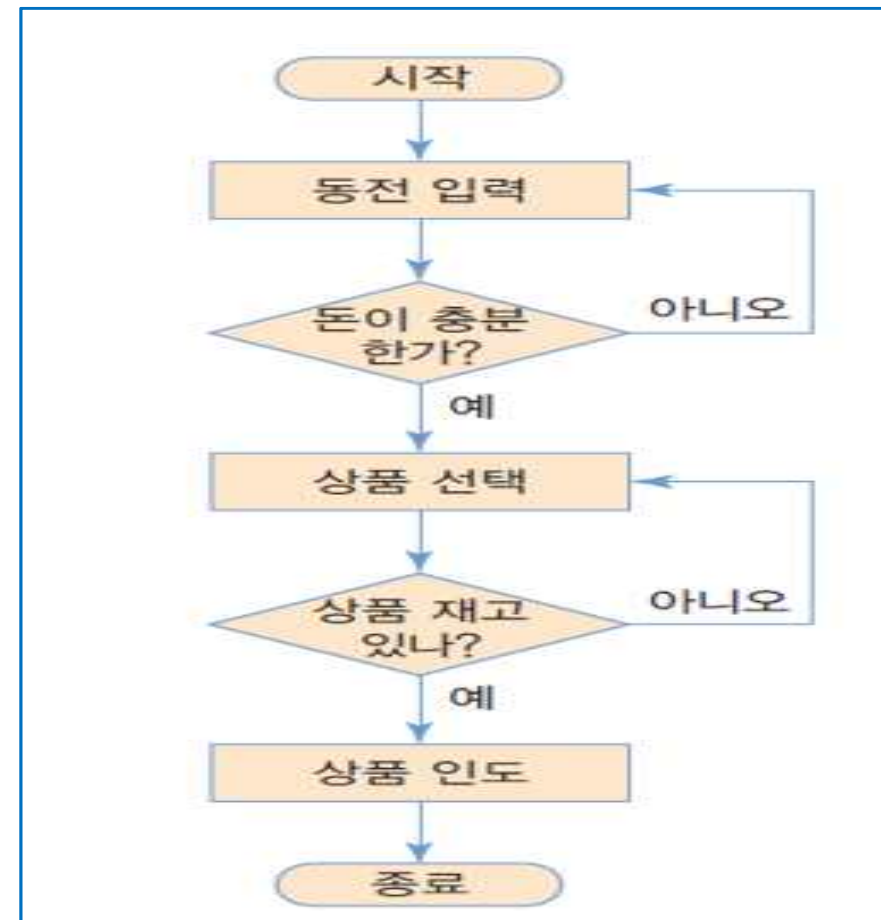


프로그래밍 살펴보기

◆ 프로그래밍 방법



객체들의 상호 관련성

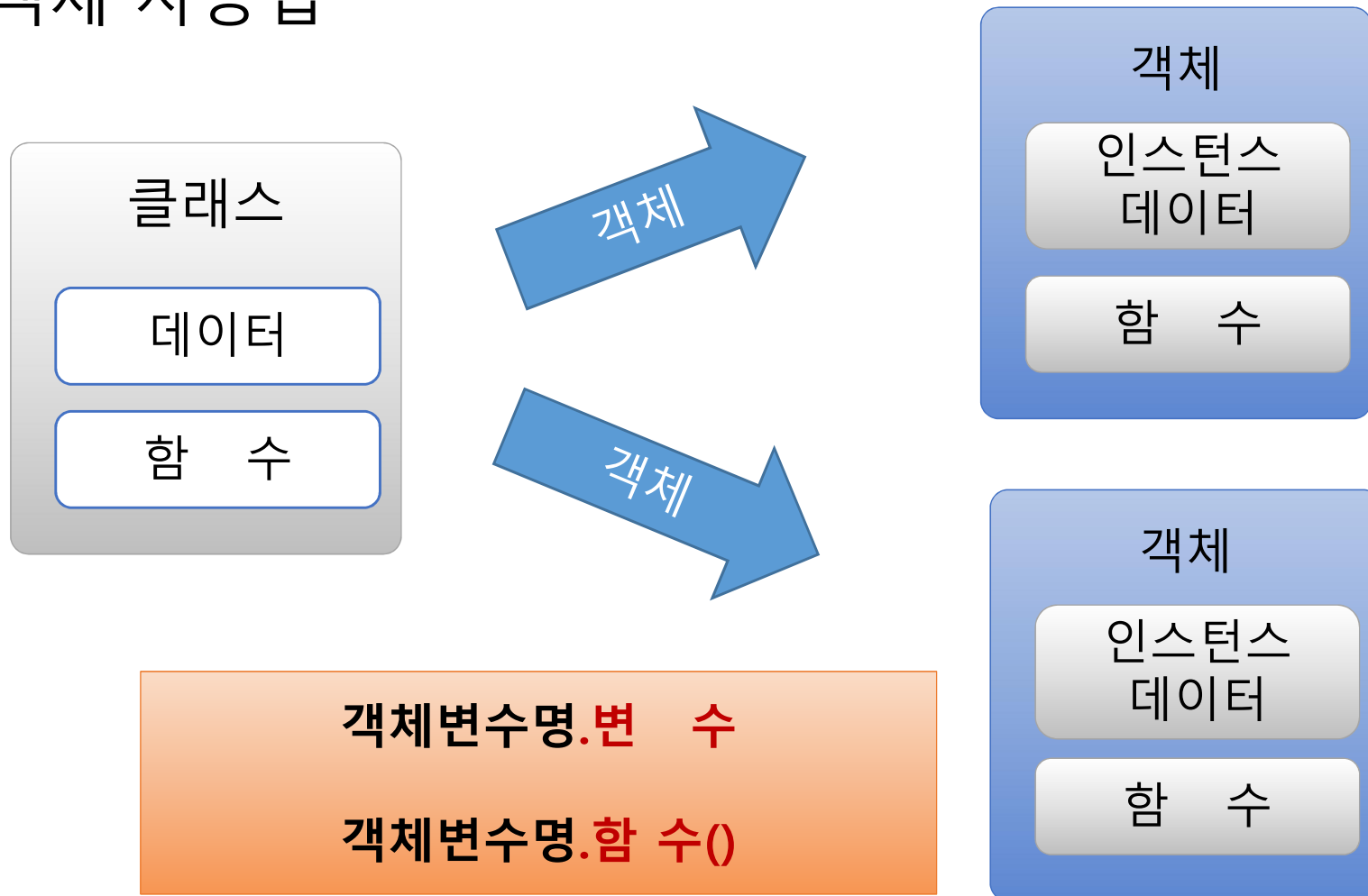


절차지향적 프로그래밍의 실행 절차

프로그래밍 살펴보기

◆ 프로그래밍 방법

➤ 객체 사용법



CH02. PYTHON 살펴보기

PYTHON 살펴보기

◆ Python 이란?

- 1991년 귀도 반 로섬(Guido Van Rossum)이 발표
- 플랫폼 독립적인 인터프리터 언어
- 객체 지향적, 동적 타이핑 언어
- 처음 C언어로 구현되었음



Guido Van Rossum

- 귀도 반 로섬은 1989년 크리스마스에 할 일이 딱히없어 파이썬을 개발하였다고 함
- “파이썬”이라는 이름은 코메디 프로그램 “Monty Python’s Flying Circus”에서 유래
- 파이썬의 원래 의미는 그리스 신화에 나오는 거대한 뱀
- 귀도는 구글에 근무했고 현재 DropBox에서 근무 중
- 귀도는 파이썬 개발자를 찾는 헤드헌터로 부터 취업 제안 메일을 받은 적이 있음

PYTHON 살펴보기

◆ Python 특징

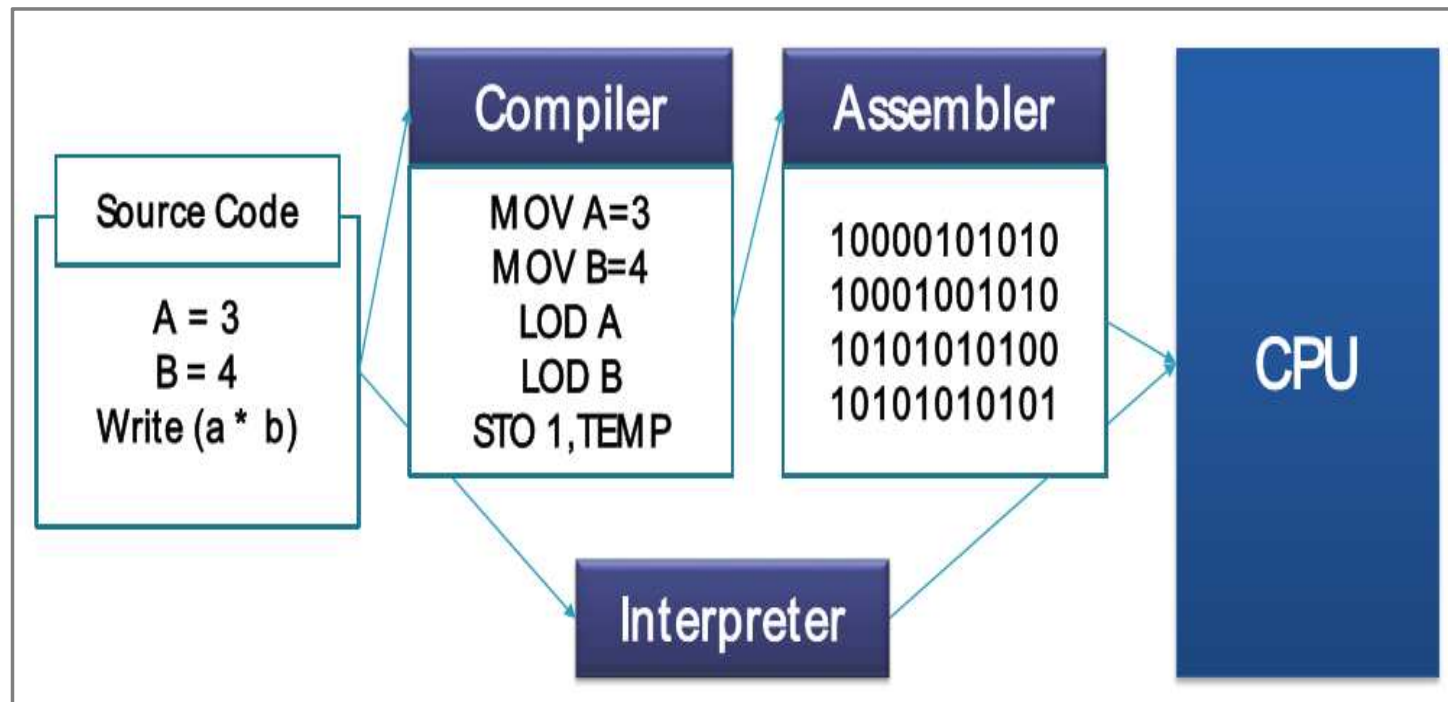
➤ 인터프리터(동시통역사) 언어란?

[컴파일러 언어]		[인터프리터 언어]
소스코드를 기계어로 먼저 번역하고 해당 플랫폼에 최적화되어 프로그램을 실행	작동방식	별도의 번역과정 없이 소스코드를 한줄 한줄 분석하여 컴퓨터가 처리할 수 있도록 함
실행속도가 빠름 한번의 많은 기억장소 필요	장점 단점	간단히 작성, 메모리가 적게 필요 실행속도가 느림
C, 자바, C++, C#	주요 언어	파이썬, 스칼라

PYTHON 살펴보기

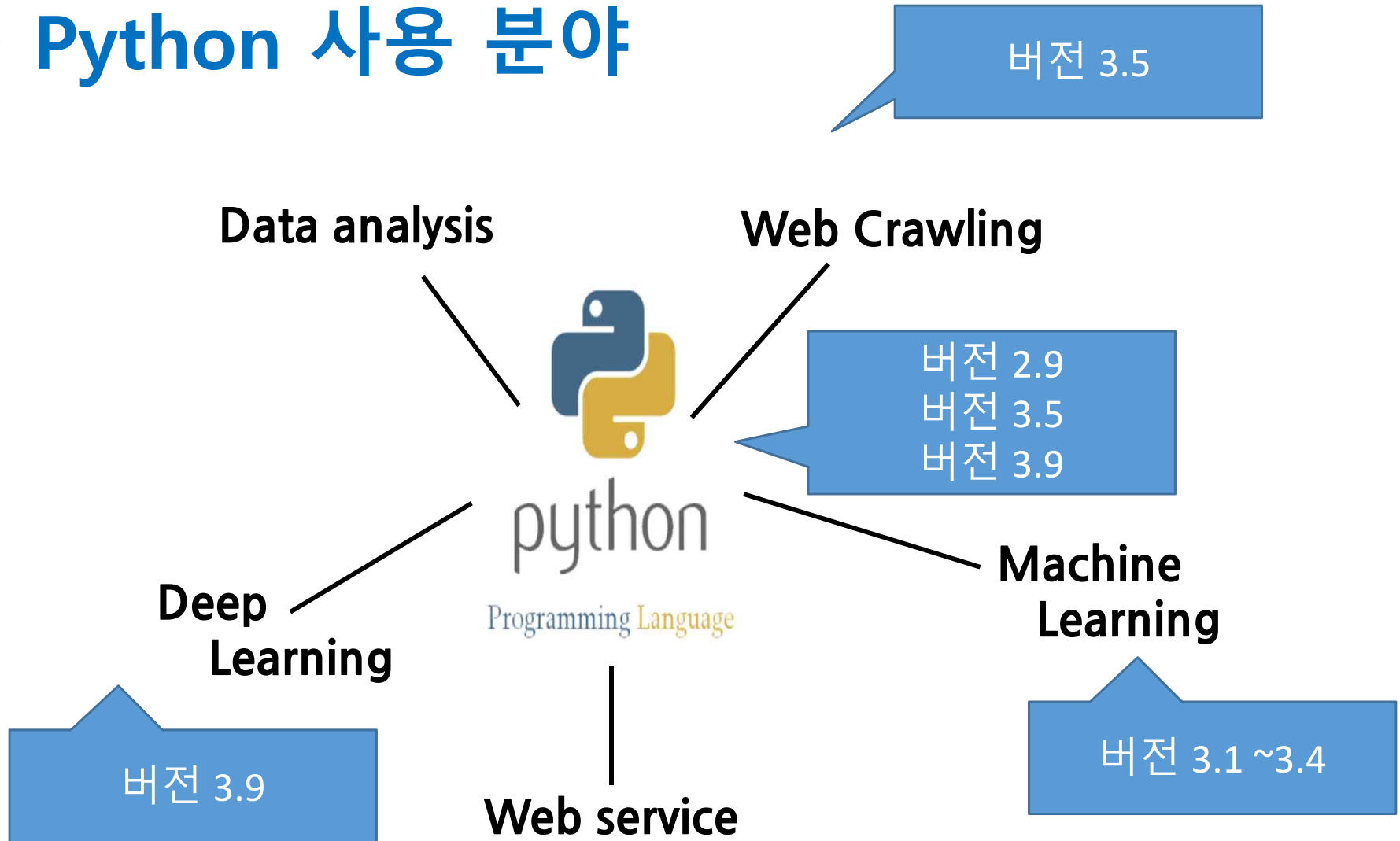
◆ Python 특징

➤ 인터프리터 언어 역할



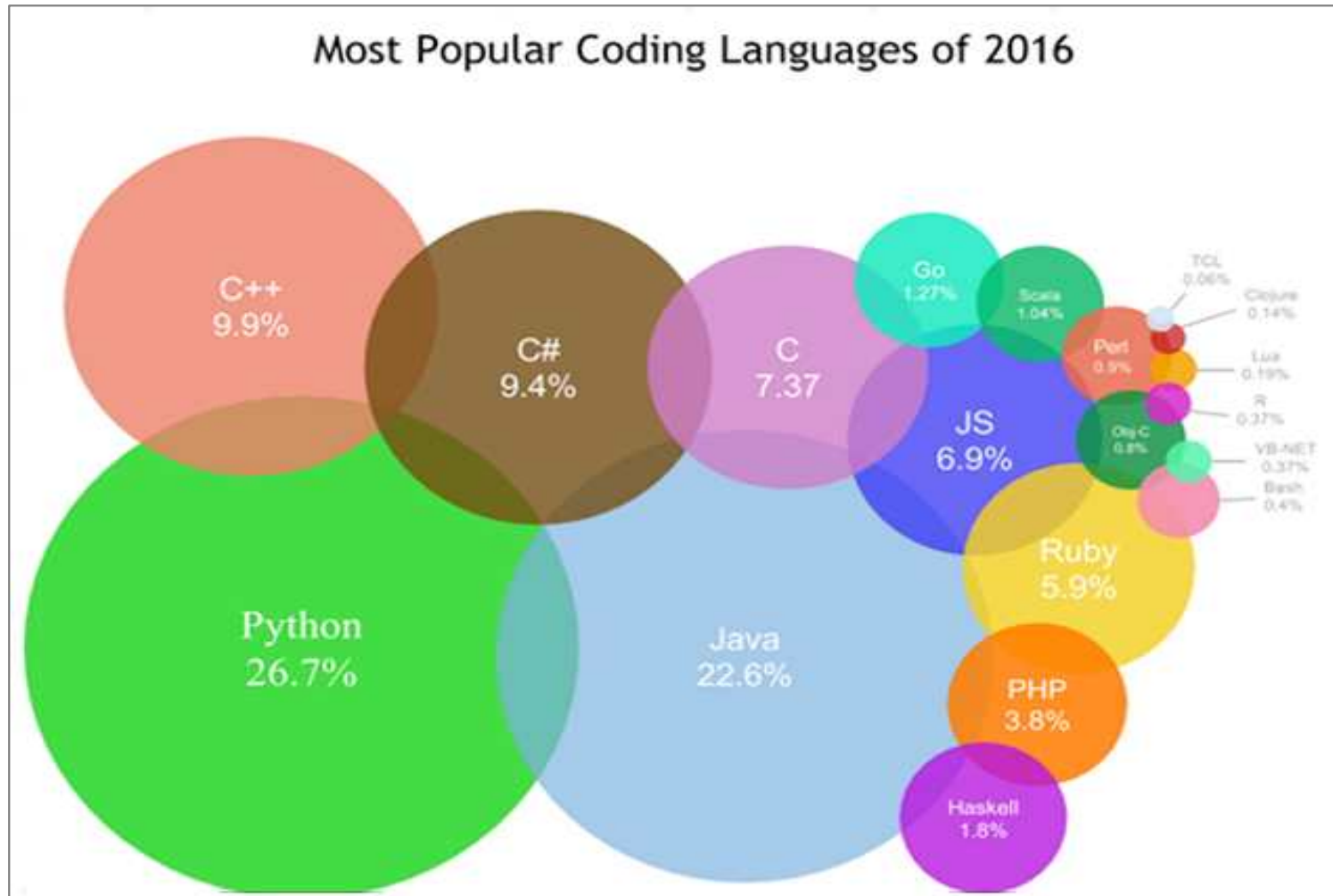
PYTHON 살펴보기

◆ Python 사용 분야



PYTHON 살펴보기

◆ Python 대중화



PYTHON 살펴보기

◆ Python 버전 비교

Python 2.X
<code>print x</code>
<code>print "%d%f%s"%(n,x,s)</code>
<code>print x ,</code>
<code>print >>ofile, x</code>
<code>string.split(s, sep)</code>
<code>string.join(lst, connector)</code>
<code>raw_input(prompt)</code>
<code>input(prompt)</code>
<code>rang(a, b, s)</code>
<code>map(...)</code>

호환 불가

Python 3.X
<code>print(x)</code>
<code>print("%d%f%s"%(n,x,s))</code>
<code>print(x , end=" ")</code>
<code>print(x, file=ofile)</code>
<code>s.split(sep)</code>
<code>connector.join(lst)</code>
<code>input(prompt)</code>
<code>eval(input(prompt))</code>
<code>list(rang(a, b, s))</code>
<code>list(map(...))</code>

CH03. 개발환경구축

PYTHON 개발환경구축

◆ Python 개발 프로그램

- REPL(Read Eval Print Loop) : 콘솔 화면에서 입력 & 결과 확인
- IDE(Integrated Development Environment) : 통합개발환경



PYTHON 개발환경구축

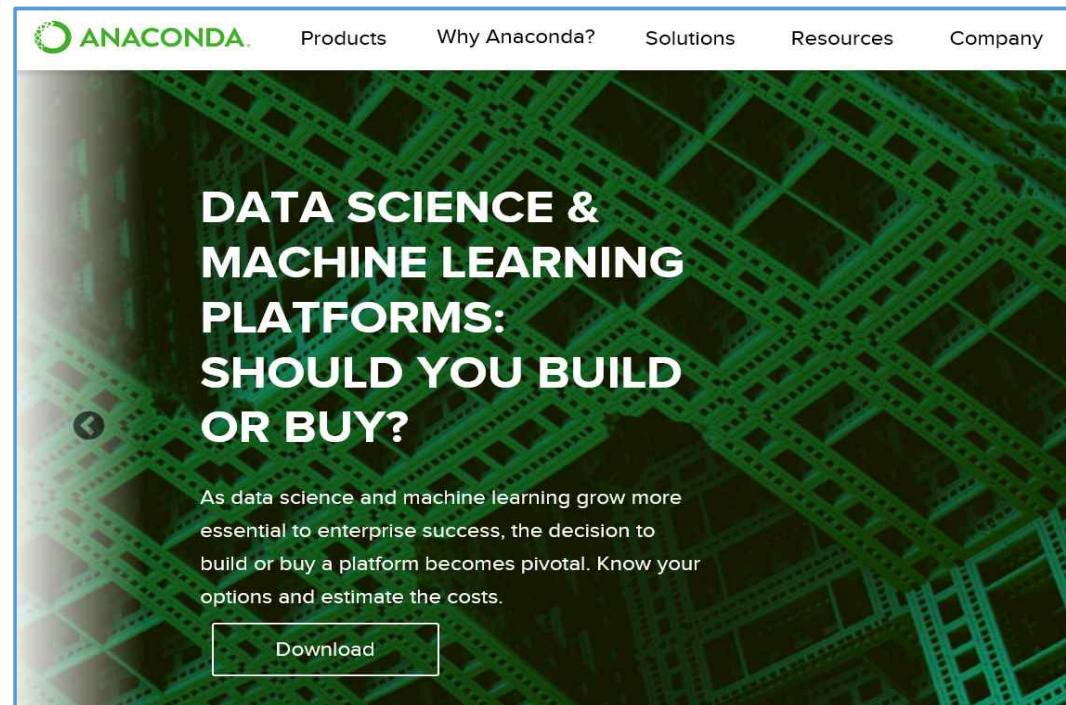
◆ 아나콘다(Anaconda)

- 수 백 개의 파이썬 패키지를 포함하고 있는 파이썬 배포판
- 패키지를 별도로 시간 들여 추가 설치하는 수고 감소
- 기능

- 패키지 설치 및 관리
- 가상환경 관리

- 관리 프로그램
 - conda 제공

www.anaconda.com



PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ conda 프로그램

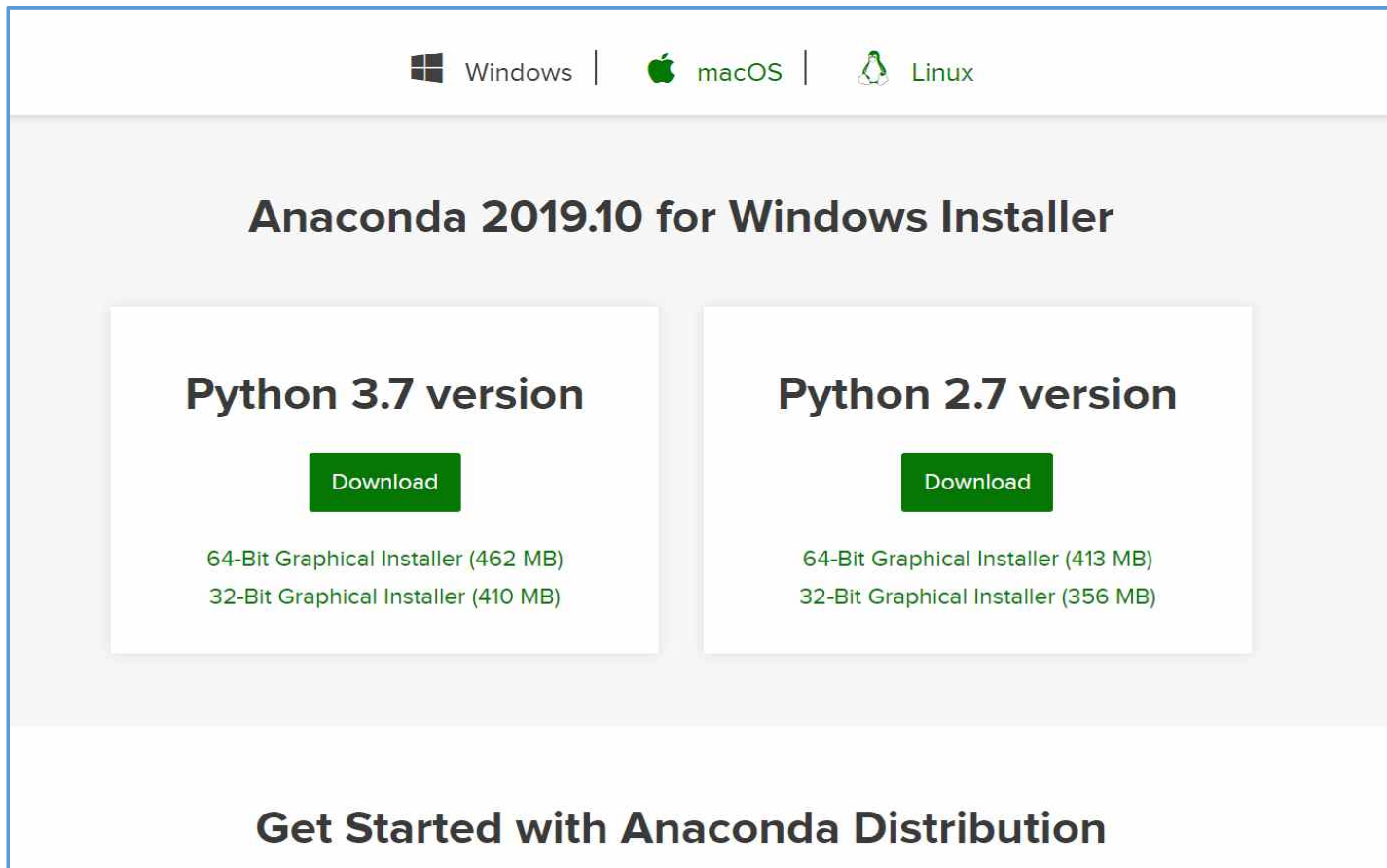
- 자동 의존성 체크 후 설치
- 모든 라이브러리 없으므로 pip와 병행 사용

기능	명령어	옵션	커맨드
패키지 설치	\$	conda install	numpy # numpy 설치
동시에 여러 패키지 설치	\$	conda install	numpy scipy pandas # numpy, scipy, pandas 동시 설치
특정 버전 설치	\$	conda install	numpy=1.10 # 특정 버전 설치
패키지 제거	\$	conda remove	package_name
패키지 업데이트	\$	conda update	package_name
모든 패키지 업데이트	\$	conda update	-all
설치된 목록 출력	\$	conda list	# 정보 확인
설치하려는 패키지 검색	\$	conda search	search_term

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ Anaconda3-2019.10-Windows-x86_64.exe 다운로드



The screenshot shows the Anaconda 2019.10 for Windows Installer download page. At the top, there are links for Windows, macOS, and Linux. The main heading is "Anaconda 2019.10 for Windows Installer". Below this, there are two columns for different Python versions. The left column is for "Python 3.7 version" and the right column is for "Python 2.7 version". Each column has a green "Download" button. Below the buttons, the file sizes for 64-bit and 32-bit graphical installers are listed. At the bottom, there is a link to "Get Started with Anaconda Distribution".

Python Version	64-Bit Graphical Installer	32-Bit Graphical Installer
Python 3.7 version	462 MB	410 MB
Python 2.7 version	413 MB	356 MB

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ Anaconda3-2019.10-Windows-x86_64.exe 다운로드

<https://repo.continuum.io/archive/>

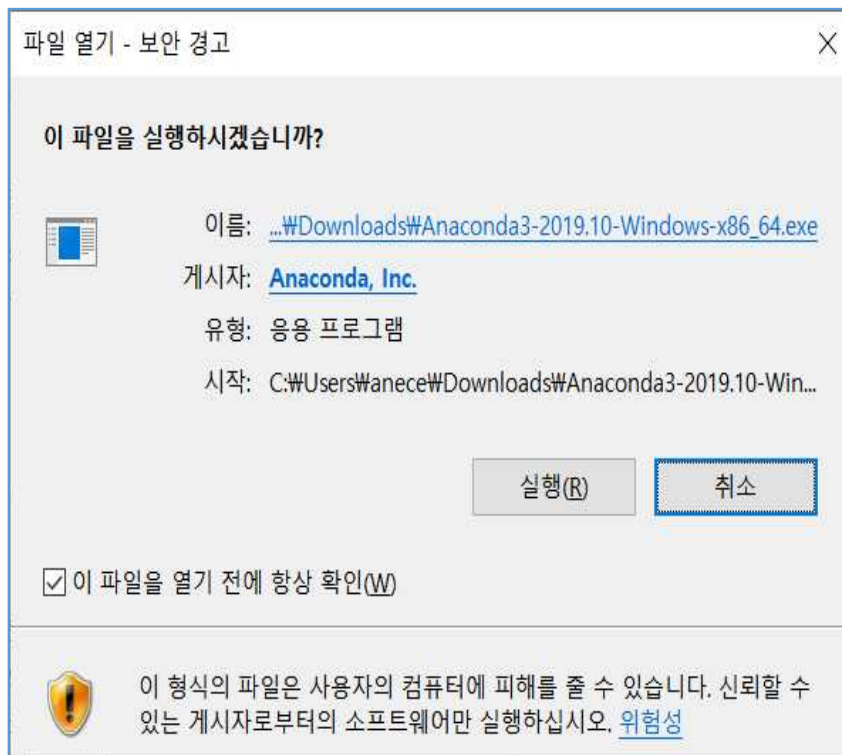
Anaconda installer archive

Filename	Size	Last Modified
Anaconda2-2019.10-Linux-ppc64le.sh	295.3M	2019-10-15 09:26:13
Anaconda2-2019.10-Linux-x86_64.sh	477.4M	2019-10-15 09:26:03
Anaconda2-2019.10-MacOSX-x86_64.pkg	635.7M	2019-10-15 09:27:30
Anaconda2-2019.10-MacOSX-x86_64.sh	408.8M	2019-10-15 09:27:31
Anaconda2-2019.10-Windows-x86.exe	355.6M	2019-10-15 09:26:15
Anaconda2-2019.10-Windows-x86_64.exe	412.8M	2019-10-15 09:26:08
Anaconda3-2019.10-Linux-ppc64le.sh	320.3M	2019-10-15 09:26:11
Anaconda3-2019.10-Linux-x86_64.sh	505.7M	2019-10-15 09:26:05
Anaconda3-2019.10-MacOSX-x86_64.pkg	653.5M	2019-10-15 09:27:33
Anaconda3-2019.10-MacOSX-x86_64.sh	424.2M	2019-10-15 09:27:31
Anaconda3-2019.10-Windows-x86.exe	409.6M	2019-10-15 09:26:10
Anaconda3-2019.10-Windows-x86_64.exe	461.5M	2019-10-15 09:27:17
Anaconda2-2019.07-Linux-ppc64le.sh	298.2M	2019-07-25 09:36:29

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

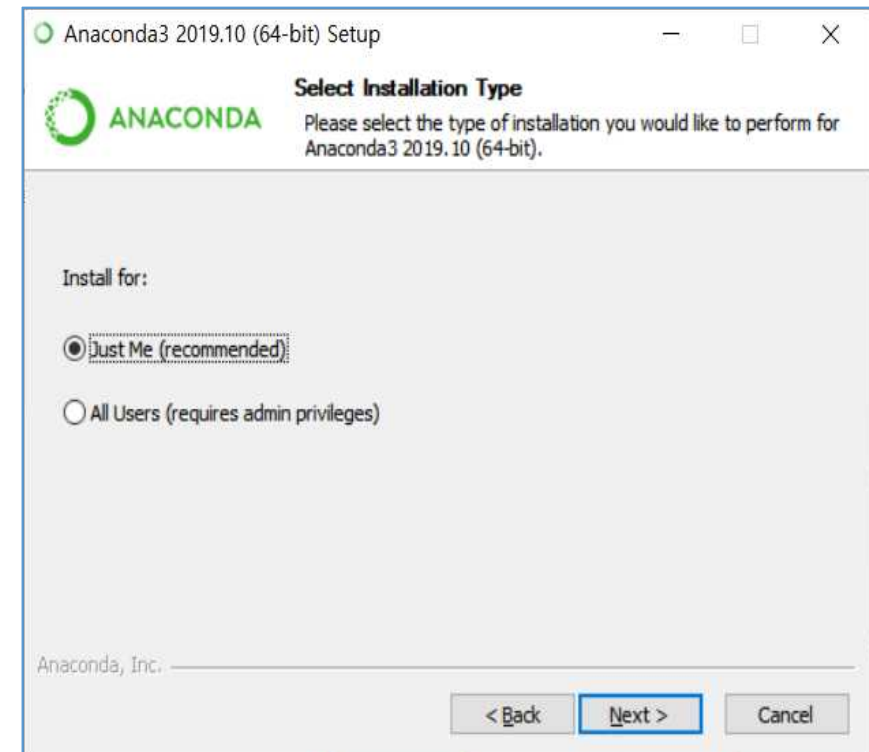
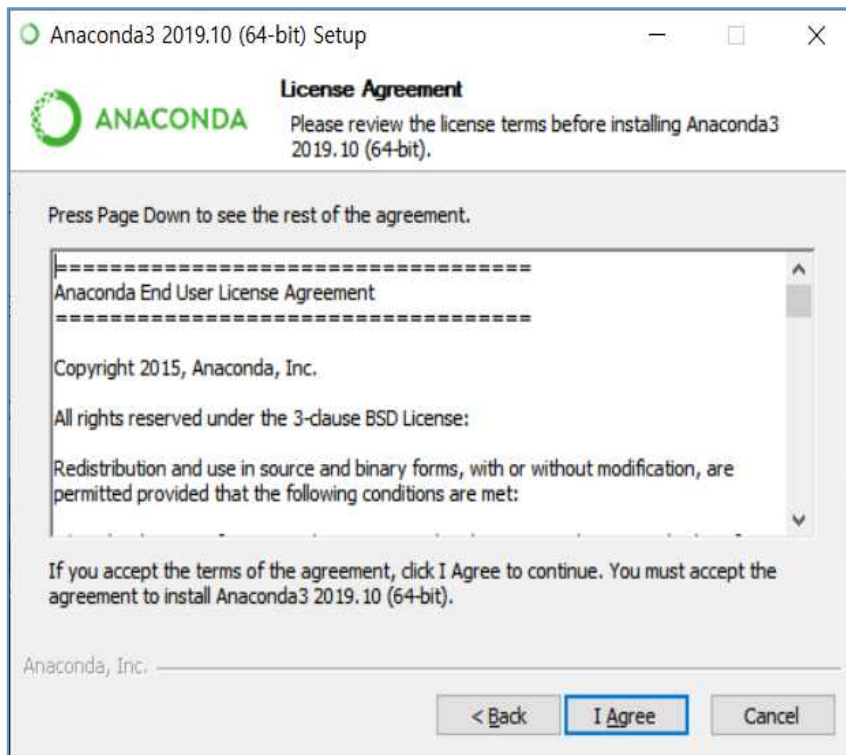
➤ Anaconda3-2019.10-Windows-x86_64.exe 실행



PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

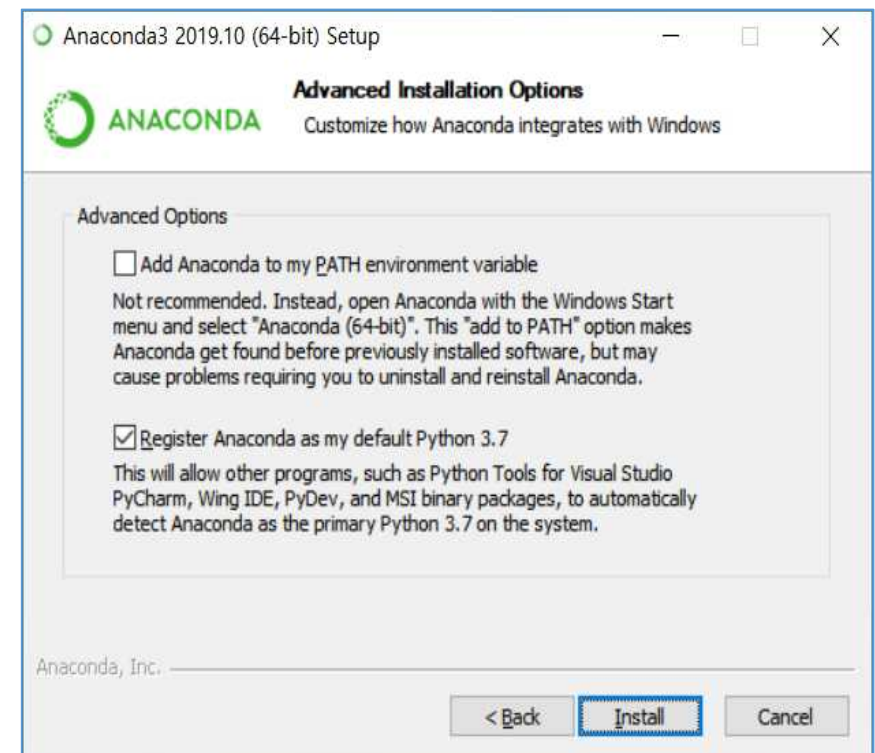
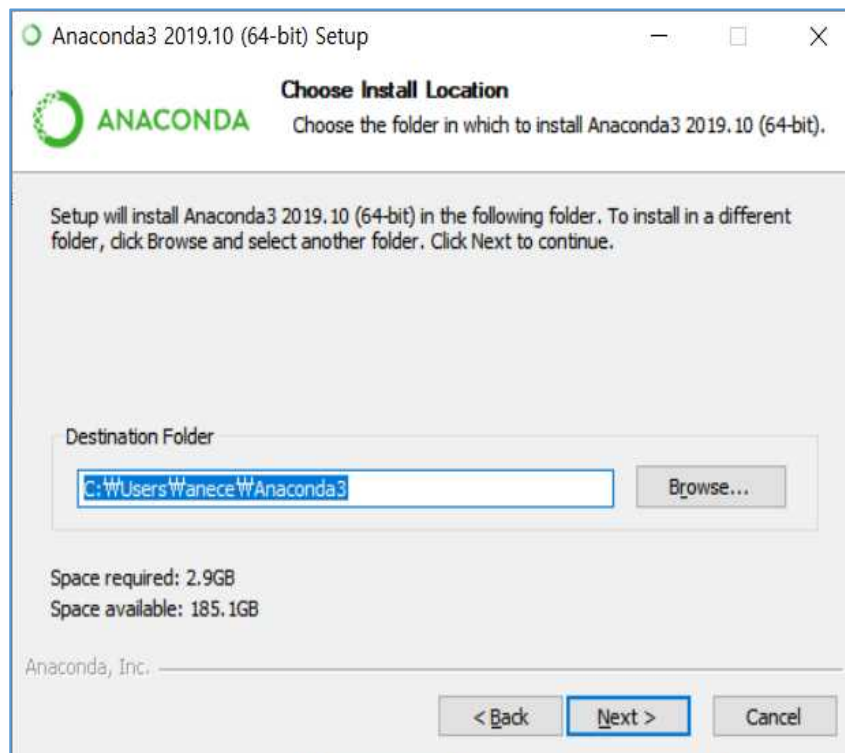
➤ Anaconda3-2019.10-Windows-x86_64.exe 실행



PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

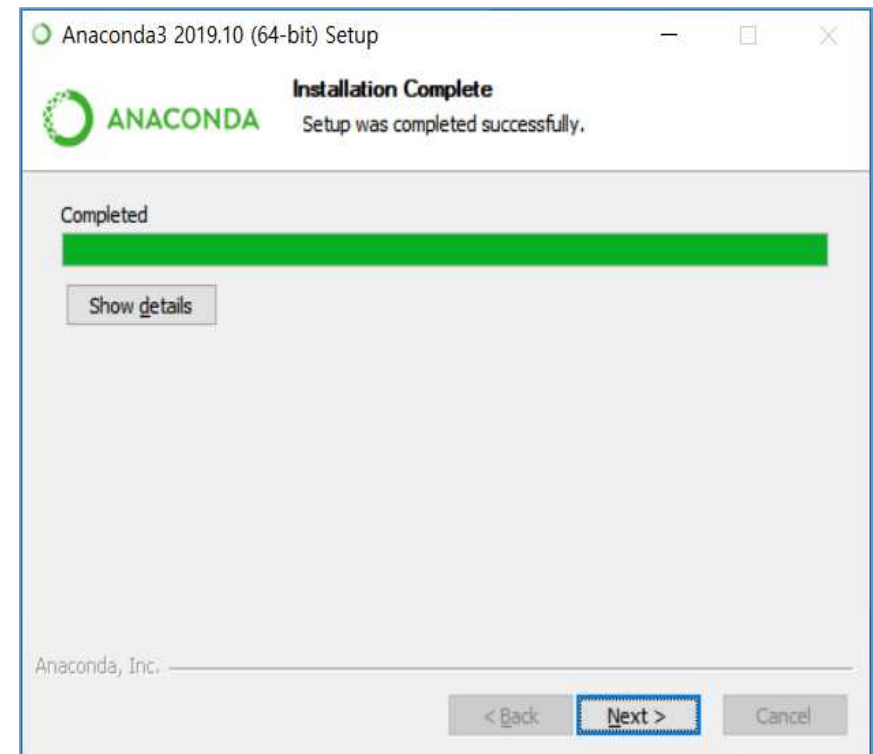
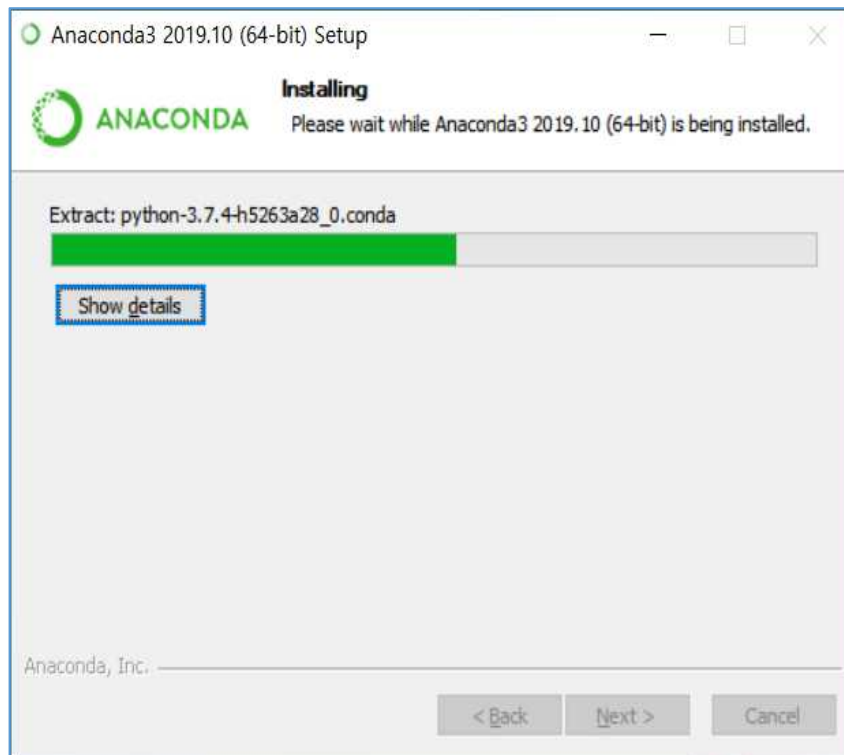
➤ Anaconda3-2019.10-Windows-x86_64.exe 실행



PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

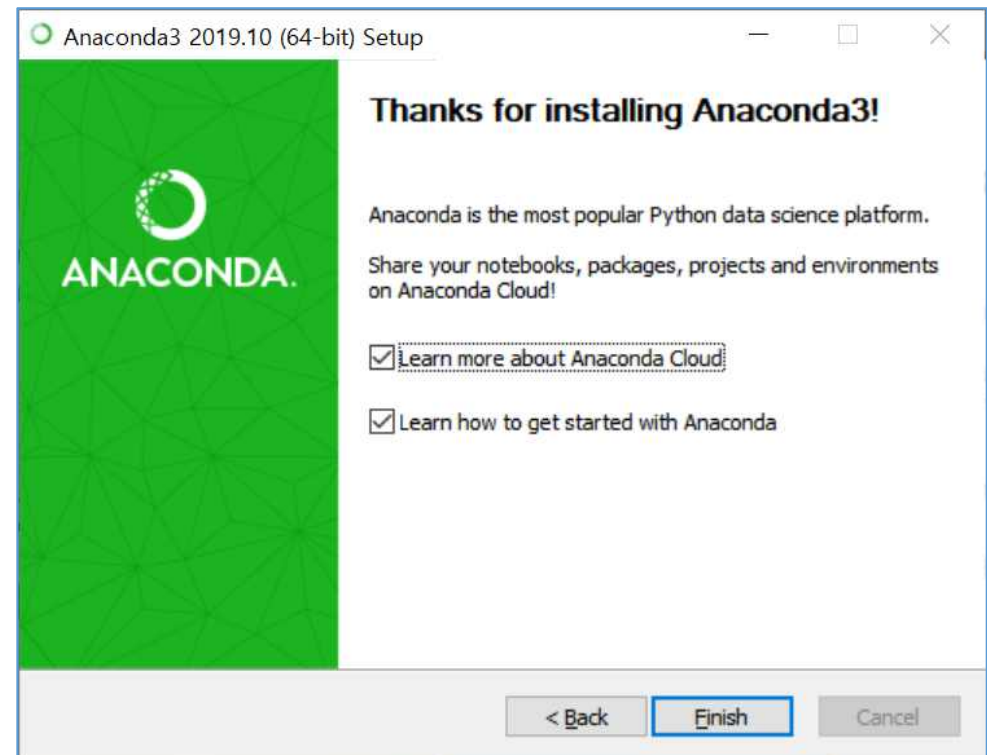
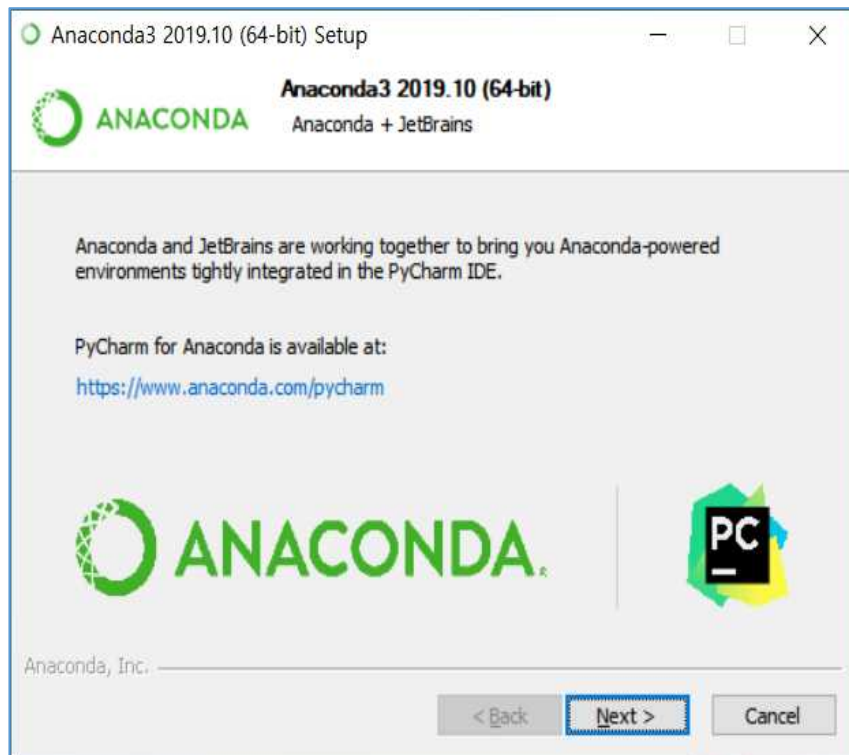
➤ Anaconda3-2019.10-Windows-x86_64.exe 실행



PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ Anaconda3-2019.10-Windows-x86_64.exe 실행

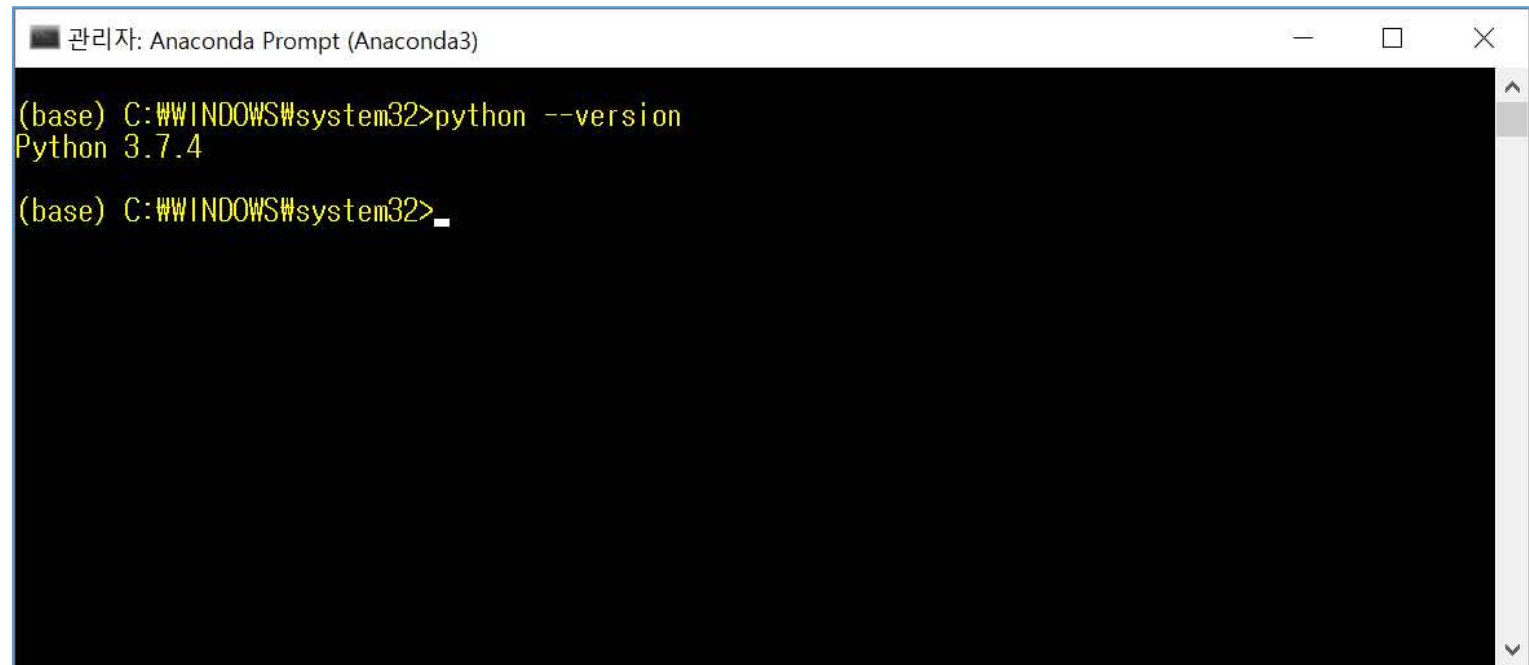


PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ Python 설치 버전 확인

- Anaconda Prompt 관리자 권한으로 실행
- `python --version` 입력 => 버전 확인



```
관리자: Anaconda Prompt (Anaconda3)

(base) C:\WINDOWS\system32>python --version
Python 3.7.4

(base) C:\WINDOWS\system32>_
```

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 업데이트

- Conda 업데이트 : `conda update -n base conda`

```
관리자: Anaconda Prompt (Anaconda3) - conda update -n base conda

(base) C:\WINDOWS\system32>conda update -n base conda
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\anece\Anaconda3

  added / updated specs:
    - conda

The following packages will be downloaded:



| package                             | build  | py_0   |        |
|-------------------------------------|--------|--------|--------|
| backports.functools_lru_cache-1.6.1 |        |        | 11 KB  |
| conda-4.8.0                         | py37_1 | 2.8 MB |        |
| future-0.18.2                       | py37_0 | 656 KB |        |
| Total:                              |        |        | 3.5 MB |



The following packages will be UPDATED:

backports.functoo~      1.5-py_2 --> 1.6.1-py_0
conda                   4.7.12-py37_0 --> 4.8.0-py37_1
future                  0.17.1-py37_0 --> 0.18.2-py37_0

Proceed ([y]/n)? y_
```

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 업데이트

- Conda 패키지 업데이트 : **conda update --all**

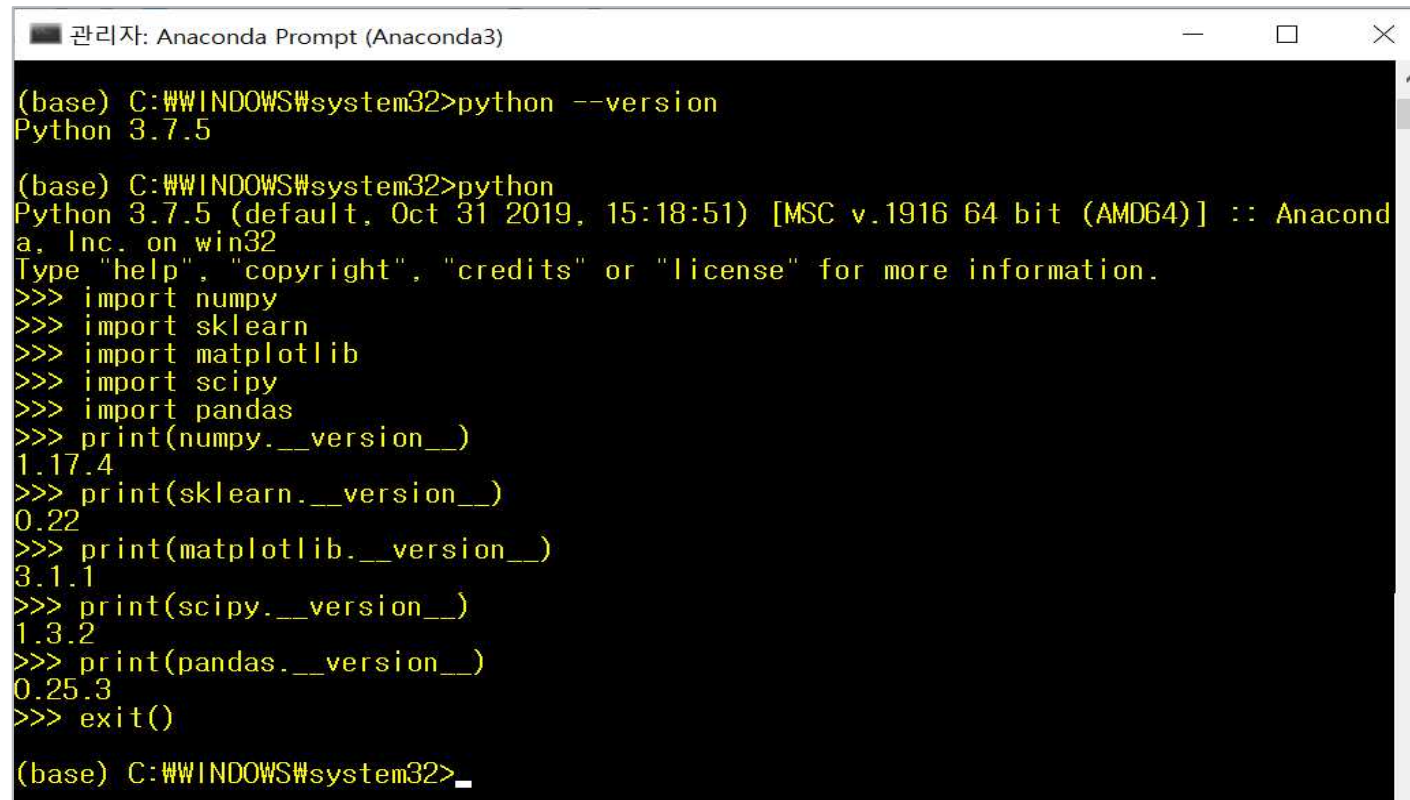
The first screenshot shows the command `conda update --all` being executed. It displays the progress of collecting metadata, solving the environment, and resolving package resolutions. A warning indicates that two packages will be updated: `jupyter_console` from 5.2.0 to 6.0.0 and `jupyter_console` from 6.0.0 to 6.0.0. The package plan shows the environment location as `C:\Users\Wanece\Anaconda3`. The second screenshot shows the list of packages to be downloaded, including `_anaconda_depends`, `anaconda-custom`, and `anaconda-project`. The third screenshot shows the list of packages to be downgraded, including `anaconda`, `jedi`, `jupyter_console`, and `pycosat`. The prompt asks to proceed with the update, and the user responds with 'y'.

```
선택 관리자: Anaconda Prompt (Anaconda3) - conda update --all
done
(base) C:\WINDOWS\system32>conda update --all
Collecting package metadata (current_repodata.json): done
Solving environment: /
Warning: 2 possible package resolutions (only showing differ
- defaults::jupyter_console-5.2.0-py37_1, defaults::prompt
- defaults::jupyter_console-6.0.0-py37_0, defaults::prompt
## Package Plan ##
environment location: C:\Users\Wanece\Anaconda3
The following packages will be downloaded:
package | build
-----|-----
_anaconda_depends-2019.03 | py37_0
anaconda-custom | py37_1
anaconda-project-0.8.4 | py_0
2
The following packages will be DOWNGRADED:
anaconda 2019.10-py37_0 --> custom-py37_1
jedi 0.15.1-py37_0 --> 0.14.1-py37_0
jupyter_console 6.0.0-py37_0 --> 5.2.0-py37_1
pycosat 0.6.3-py37hfa6e2cd_0 --> 0.6.3-py37he774522_0
Proceed ([y]/n)? y_
선택 관리자: Anaconda Prompt (Anaconda3)
'C:\Users\Wanece\Anaconda3\pythonw.exe', 'C:\Users\Wanece\Anaconda3\Scripts\spyder-script.py']
DEBUG menuinst win32:create(323): Shortcut cmd is C:\Users\Wanece\Anaconda3\python.exe
e, args are ['C:\Users\Wanece\Anaconda3\pythonw.exe', 'C:\Users\Wanece\Anaconda3\Scripts\spyder-script.py', '--reset']
done
(base) C:\WINDOWS\system32>
```

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 업데이트 후 설치 라이브러리 확인

A screenshot of the Anaconda Prompt window. The title bar reads '관리자: Anaconda Prompt (Anaconda3)'. The command prompt shows the user running 'python --version' which returns 'Python 3.7.5'. Then, the user runs 'python' which opens the Python interpreter. Inside the interpreter, several libraries are imported: numpy, sklearn, matplotlib, scipy, and pandas. Then, the version of each library is printed using 'print(library.__version__)'. The versions are: numpy 1.17.4, sklearn 0.22, matplotlib 3.1.1, scipy 1.3.2, and pandas 0.25.3. Finally, 'exit()' is called to close the interpreter, and the prompt returns to the command line.

```
(base) C:\WINDOWS\system32>python --version
Python 3.7.5

(base) C:\WINDOWS\system32>python
Python 3.7.5 (default, Oct 31 2019, 15:18:51) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> import sklearn
>>> import matplotlib
>>> import scipy
>>> import pandas
>>> print(numpy.__version__)
1.17.4
>>> print(sklearn.__version__)
0.22
>>> print(matplotlib.__version__)
3.1.1
>>> print(scipy.__version__)
1.3.2
>>> print(pandas.__version__)
0.25.3
>>> exit()

(base) C:\WINDOWS\system32>
```

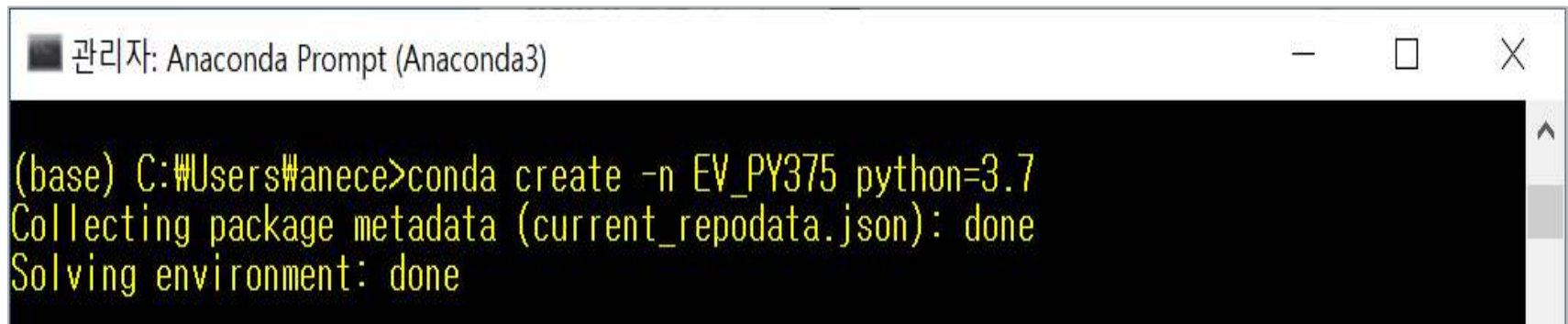
PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 가상환경 만들기

`conda create -n 가상환경이름 python=파이썬버전`

(base) C:\Users\사용자계정>`conda create -n EV_PY37 python=3.7`



```
관리자: Anaconda Prompt (Anaconda3)

(base) C:\Users\anece>conda create -n EV_PY375 python=3.7
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 가상환경 만들기

```
## Package Plan ##
environment location: C:\Users\anece\Anaconda3\envs\EV_PY375
added / updated specs:
- python=3.7

The following packages will be downloaded:

package                        | build                | 58 KB
-----|-----
wheel-0.33.6                   | py37_0               |
-----|-----
Total:                          |                      | 58 KB

The following NEW packages will be INSTALLED:

ca-certificates  pkgs/main/win-64::ca-certificates-2019.11.27-0
certifi          pkgs/main/win-64::certifi-2019.11.28-py37_0
openssl          pkgs/main/win-64::openssl-1.1.1d-he774522_3
pip              pkgs/main/win-64::pip-19.3.1-py37_0
python           pkgs/main/win-64::python-3.7.5-h8c8aaf0_0
setuptools       pkgs/main/win-64::setuptools-42.0.2-py37_0
sqlite           pkgs/main/win-64::sqlite-3.30.1-he774522_0
vc               pkgs/main/win-64::vc-14.1-h0510ff6_4
vs2015_runtime   pkgs/main/win-64::vs2015_runtime-14.16.27012-hf0eaf9b_1
wheel            pkgs/main/win-64::wheel-0.33.6-py37_0
wincertstore     pkgs/main/win-64::wincertstore-0.2-py37_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
wheel-0.33.6 | 58 KB | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate EV_PY375
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 가상환경 확인

```
conda env list
```

```
(base) C:\Users\anece>conda env list
# conda environments:
#
base                * C:\Users\anece\Anaconda3
EV_PY375            C:\Users\anece\Anaconda3\envs\EV_PY375

(base) C:\Users\anece>
```

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 가상환경 실행 & 종료

실행 : `conda activate` 가상환경이름

종료: `conda deactivate`

```
(base) C:\Users\#anece>conda activate EV_PY375
```

```
(EV_PY375) C:\Users\#anece>conda deactivate
```

```
(base) C:\Users\#anece>
```


PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 가상환경 설치 모듈(라이브러리) 체크

설치 모듈 정보 출력 : `conda list`

모듈 설치 명령어 : `conda install 라이브러리명`

```
관리자: Anaconda Prompt (Anaconda3)

(base) C:\Users\Wanece>conda activate EV_PY375

(EV_PY375) C:\Users\Wanece>conda list
# packages in environment at C:\Users\Wanece\Anaconda3\envs\EV_PY375:
#
# Name                                Version           Build    Channel
ca-certificates                      2019.11.27        0
certifi                              2019.11.28        py37_0
openssl                              1.1.1d            he774522_3
pip                                  19.3.1            py37_0
python                               3.7.5             h8c8aaf0_0
setuptools                           42.0.2            py37_0
sqlite                                3.30.1            he774522_0
vc                                    14.1              h0510ff6_4
vs2015_runtime                       14.16.27012       hf0eaf9b_1
wheel                                0.33.6            py37_0
wincertstore                         0.2               py37_0

(EV_PY375) C:\Users\Wanece>_
```

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 가상환경 설치 모듈(라이브러리) 설치

```
conda install numpy scipy matplotlib spyder  
conda install pandas seaborn scikit-learn  
conda install h5py statsmodels
```

```
관리자: Anaconda Prompt (Anaconda3) - conda install numpy scipy matplotlib ...  
(EV_PY375) C:\Users\wance>conda install numpy scipy matplotlib spyder pandas  
seaborn scikit-learn h5py statsmodels  
Collecting package metadata (current_repodata.json): done  
Solving environment: done  
  
## Package Plan ##  
  
environment location: C:\Users\wance\Anaconda3\envs\EV_PY375  
  
added / updated specs:  
- h5py  
- matplotlib  
- numpy  
- pandas  
- scikit-learn  
- scipy  
- seaborn  
- spyder  
- statsmodels  
  
The following packages will be downloaded:
```

PYTHON 개발환경구축

◆ 아나콘다(Anaconda)

➤ 가상환경 설치 모듈(라이브러리) 설치

```
관리자: Anaconda Prompt (Anaconda3) - conda install numpy scipy matplotlib ...
win_inet_pton      pkgs/main/win-64::win_inet_pton-1.1.0-py37_0
wrap               pkgs/main/win-64::wrap-1.11.2-py37he774522_0
yaml               pkgs/main/win-64::yaml-0.1.7-hc54c509_2
yapf               pkgs/main/noarch::yapf-0.28.0-py_0
zeromq             pkgs/main/win-64::zeromq-4.3.1-h33f27b4_3
zipp               pkgs/main/noarch::zipp-0.6.0-py_0
zlib               pkgs/main/win-64::zlib-1.2.11-h62dcd97_3

Proceed ([y]/n)? y_
```

```
관리자: Anaconda Prompt (Anaconda3)
.exe', 'C:\\Users\\anece\\Anaconda3\\envs\\EV_PY375\\Scripts\\spyder-script.py'
, '--reset']
done

(EV_PY375) C:\\Users\\anece>conda list
# packages in environment at C:\\Users\\anece\\Anaconda3\\envs\\EV_PY375:
#
# Name          Version      Build      Channel
alabaster       0.7.12       py37_0
argh            0.26.2       py37_0
asn1crypto      1.2.0        py37_0
```

CH04. PYTHON 프로그래밍

PYTHON 프로그래밍

◆ 코딩 컨벤션(coding conventions)

- 코딩 작성 가이드 라인
- 타인 코드 이해 및 분석 가능하도록 지정된 표준안
- 언어마다 코딩 컨벤션 즉 규칙 존재

PYTHON 프로그래밍

◆ 코딩 컨벤션(coding conventions)

- 들여쓰기(Indentation)는 공백 4칸을 권장
- 한 줄은 최대 79자까지
- 클래스 내의 메소드 정의는 1줄 씩 띄어쓰기
- 주석은 항상 갱신, 불필요한 주석은 삭제
- 소문자 l, 대문자 O, 대문자 I는 변수명으로 사용하지 말 것 (가독성)
- 함수명은 소문자로 구성, 필요하면 밑줄로 나눔

함수

```
print( 출력할 재료 )  
print( 123 )  
print( "apple" )
```

```
[ 전자기인이지 ]  
데우기 버튼  
해 동 버튼  
라 면 버튼
```

PYTHON 프로그래밍

◆ 기본 규칙

- **주석** : 코드에 대한 설명, 해석 및 실행 안 되는 부분
 - # : 한 줄 주석
 - """ ~ """ , ''' ~ ''' : 여러 줄 주석
- **객체** : Python의 모든 데이터
- **함수** : 기능의 코드 묶음 ➔ 함수명(재료) 예) print(123)
 - 내장 함수 : Python에서 제공하는 함수
 - **사용자 함수** : 개발자가 생성한 함수

PYTHON 프로그래밍

◆ 기본 규칙

컴퓨터에게 시킬 일 50개

1. 청소
2. 택배반송
3. 운동하기
4. :

코드
프로그램

컴퓨터 즉 기계

일을 처리 하는 부분 => CPU

일을 기억하는 부분 => 메모리

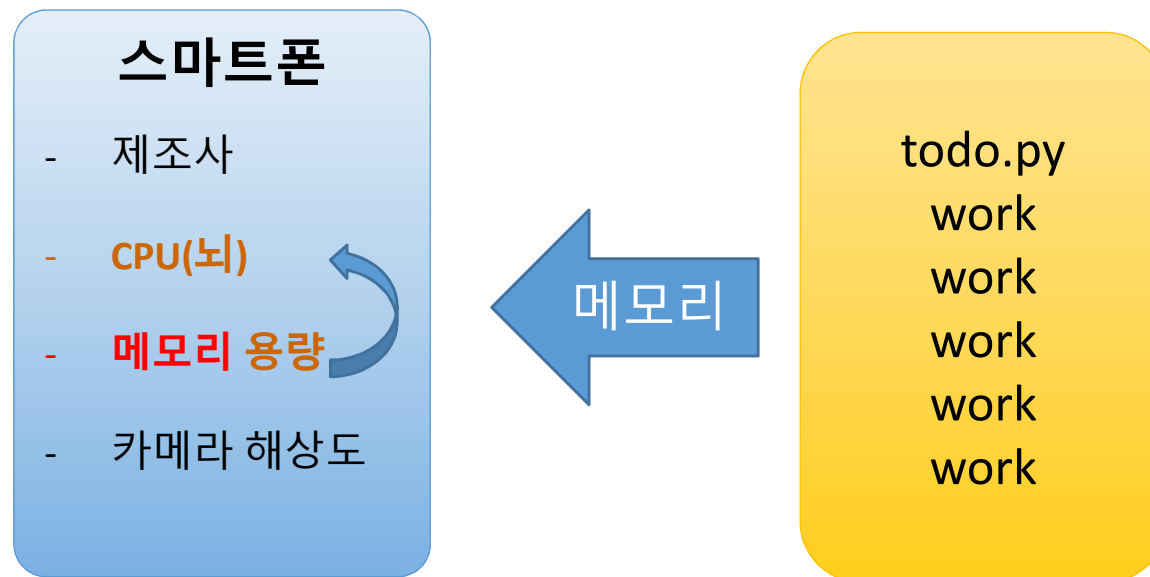
보조로 일을 기억하는 부분 =>

하드디스크, USB 메모리

PYTHON 프로그래밍

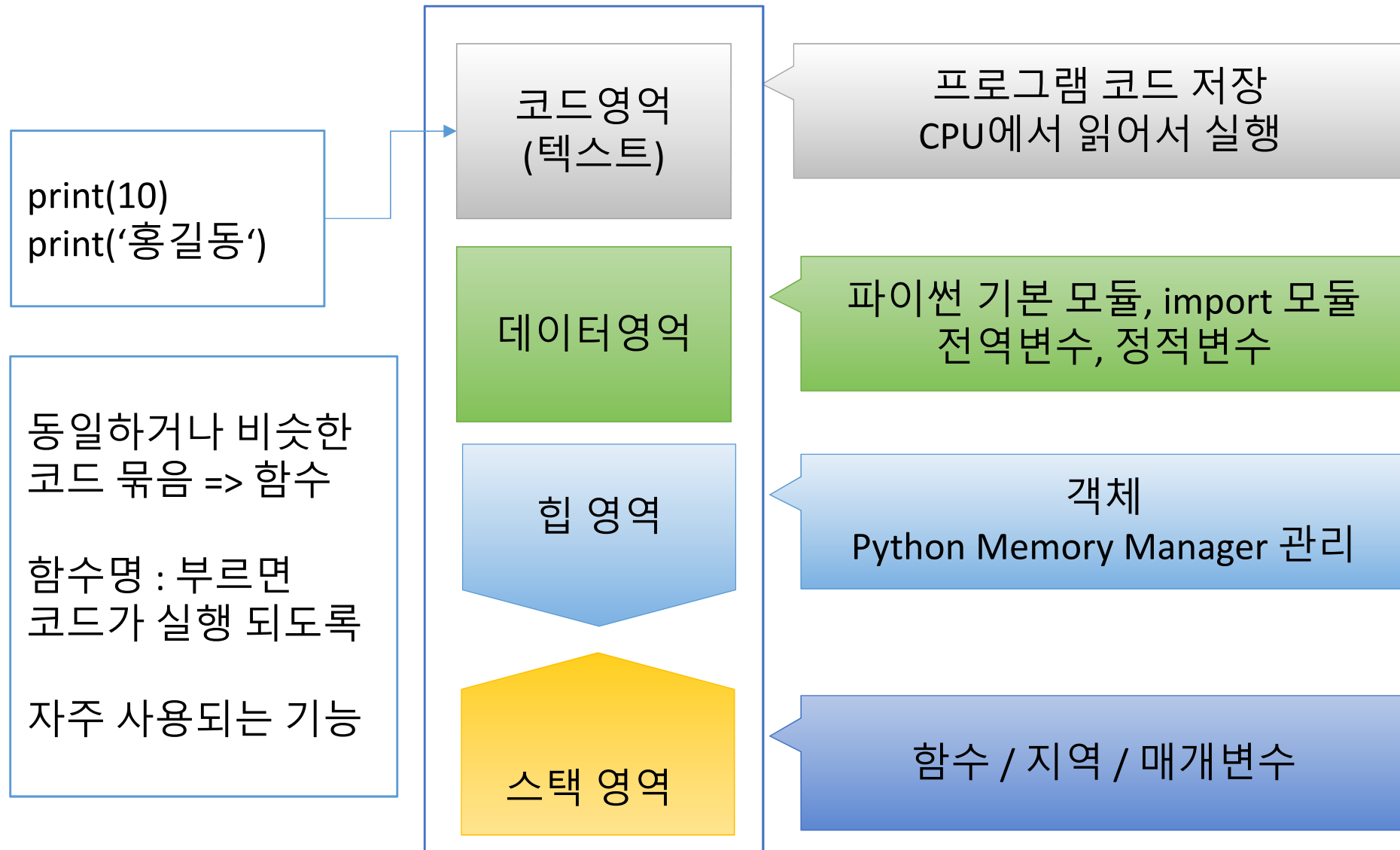
◆ 메모리와 프로그램

- 프로그램 실행시 코드가 저장되는 곳
- 효율적인 사용 필요
- 메모리 누수 또는 메모리 부족 문제 발생



PYTHON 프로그래밍

◆ 메모리와 프로그램



PYTHON 프로그래밍

◆ 메모리와 프로그램

- 정적 메모리 할당
 - 프로그램 컴파일시 메모리가 할당
 - 스택 영역 사용
- 동적 메모리 할당
 - 런타임에 메모리 할당
 - 힙 영역 사용

PYTHON 프로그래밍

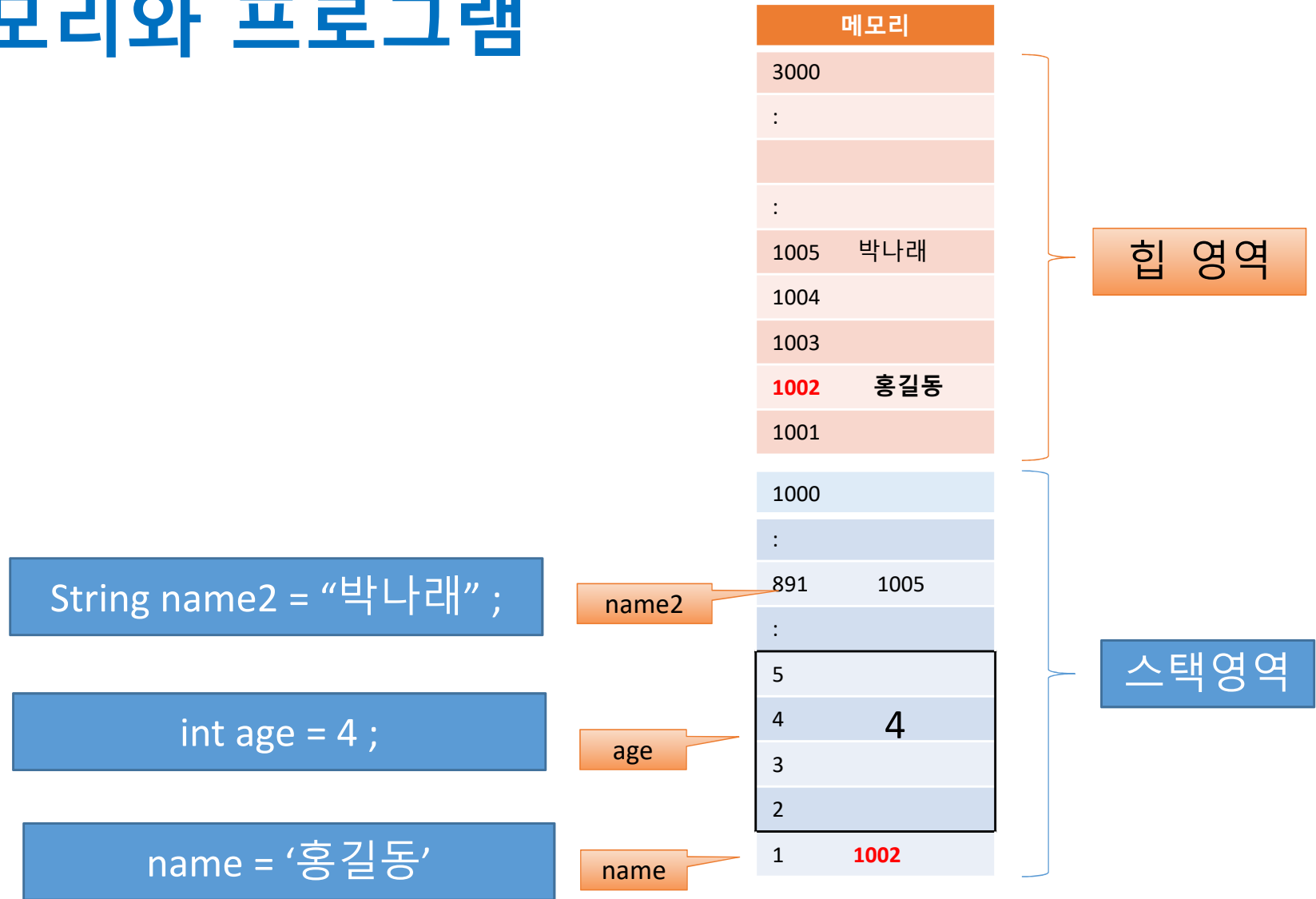
◆ 메모리와 프로그램

➤ Python Memory Manager

- 힙 영역 동적 메모리 관리
- 객체 생성 및 관리
- 사용되지 않는 메모리 처리
 - 가비지 컬렉션(Automatic Garbage Collection)
 - 레퍼런스 카운트(Reference Counts)

PYTHON 프로그래밍

◆ 메모리와 프로그램



PYTHON 프로그래밍

◆ 변수(Variable)

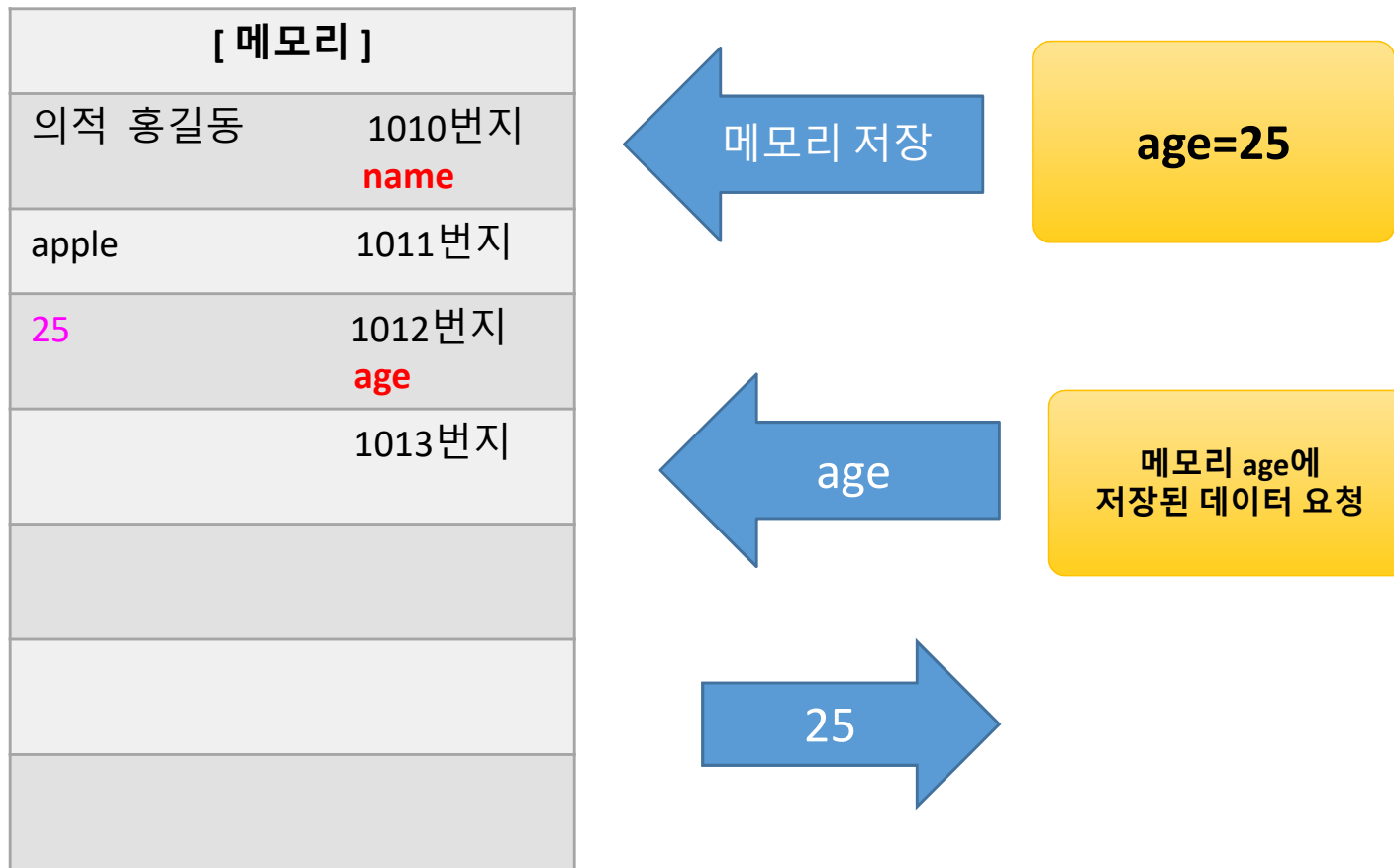
- 값(데이터) 기억해두기 위한 저장소
- 값(데이터)에 이름표를 붙이는 것
- 담은 값(데이터)은 언제나 변경 가능
- 형식 →

변수명 = 데이터



PYTHON 프로그래밍

◆ 변수(Variable)



PYTHON 프로그래밍

◆ 변수(Variable)

➤ 명명법

- 어떤 데이터가 저장되어 있는지 알 수 있도록 명명
- 문자, 숫자, 밑줄(_)만으로 변수 이름 구성
- 첫 문자로 숫자 사용 불가
- 대소문자 구분
- 예약어 사용 불가

PYTHON 프로그래밍

◆ 변수(Variable)

➤ 예약어 확인

- 파이썬에서 이미 사용하고 있는 단어
- 변수명 사용 불가

실행 결과

```
import keyword
```

```
print("Keyword \n", keyword.kwlist)
```

Keyword

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',  
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global',  
'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',  
'return', 'try', 'while', 'with', 'yield']
```

Process finished with exit code 0

PYTHON 프로그래밍

◆ 변수(Variable)

➤ 생성

다양한 변수 생성 -----

형식 ==> 변수명 = 데이터

age = 12

name = "Hong"

female = 'F'

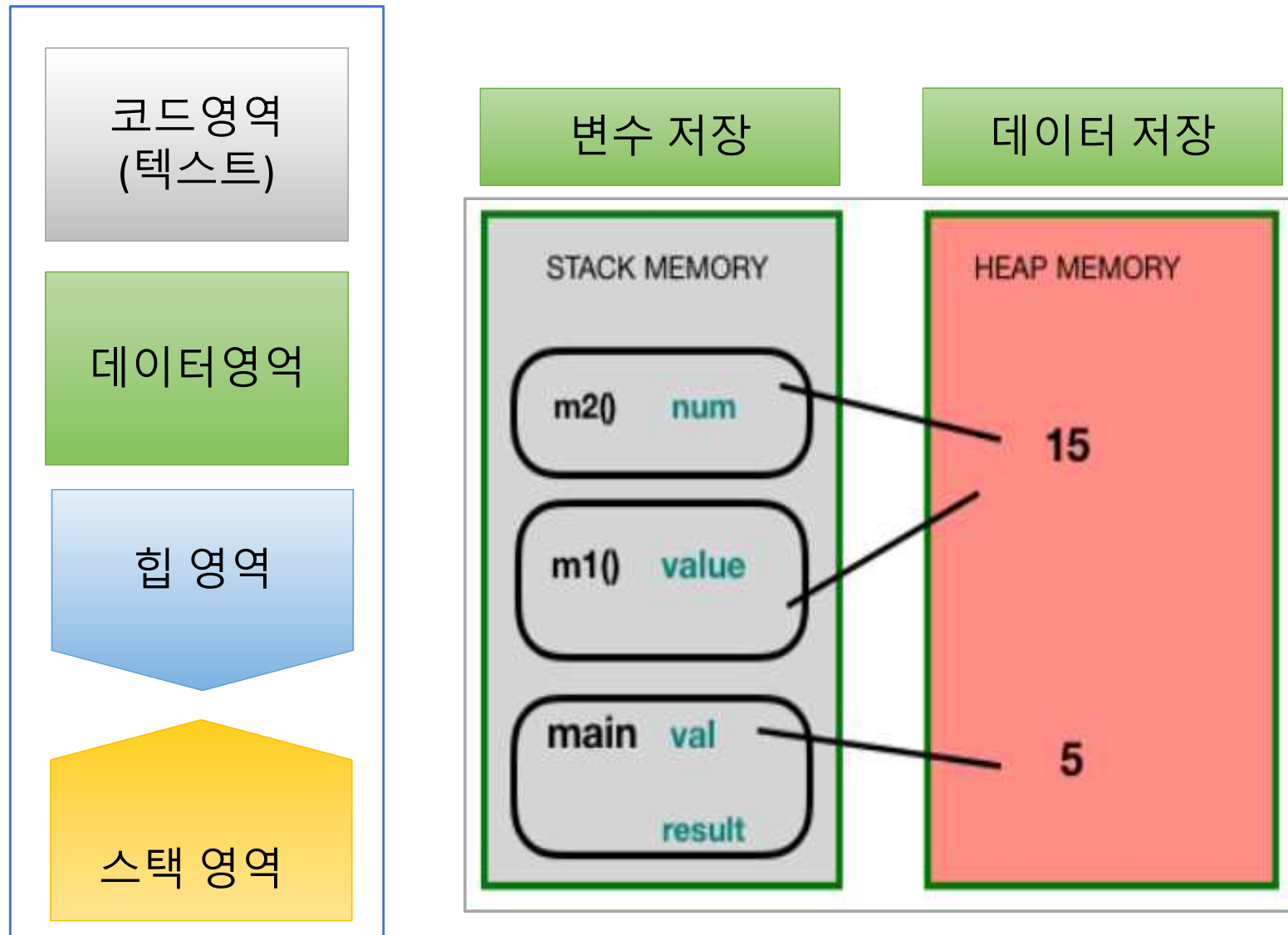
average = 87.9

ok = False

어떤 데이터가
저장되어 있는지
알수 있는 변수명

PYTHON 프로그래밍

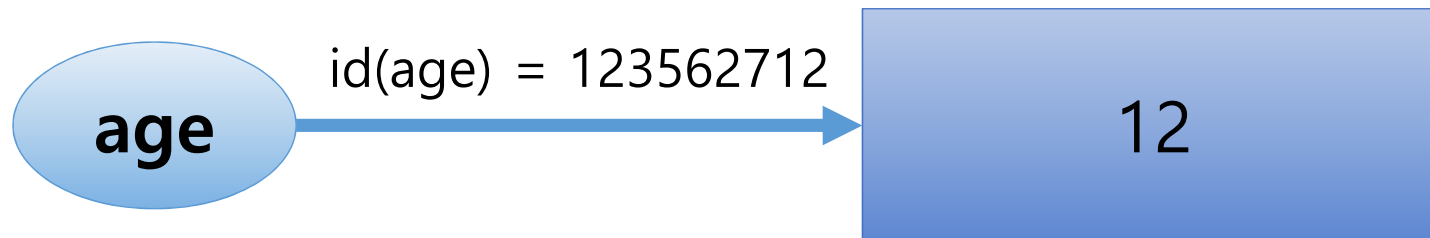
◆ 변수(Variable)



PYTHON 프로그래밍

◆ 변수(Variable)

- `id()` : 객체의 유니크(메모리 주소) 값 보여줌

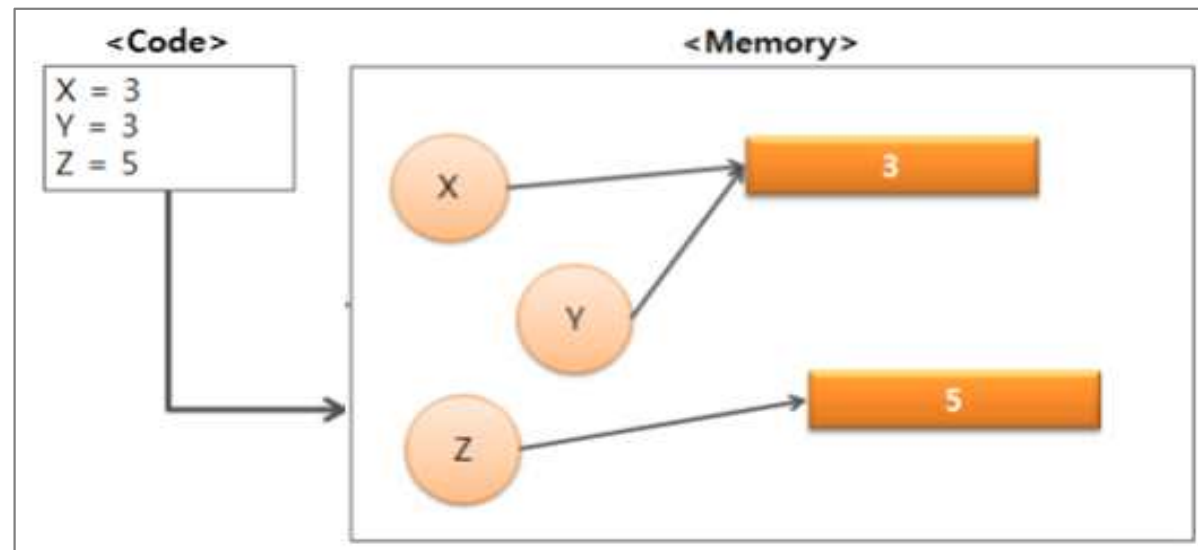


- -5 와 256 사이 정수 ➔ integer object 배열에 미리 저장

PYTHON 프로그래밍

◆ 변수(Variable)

```
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bi
on win32
Type "copyright", "credits" or "license()" for more information.
>>> X = 3
>>> Y = 3
>>> Z = 5
>>> print(id(X), id(Y), id(3), id(Z), id(5))
1617942352 1617942352 1617942352 1617942384 1617942384
```



PYTHON 프로그래밍

◆ 함수 (Function)

- 특정 기능을 하기 위한 코드 묶음
- 함수 이름을 보고 기능 파악
- 종류
 - ✓ 내장 함수 : 파이썬 제공 함수
 - ✓ 사용자 정의 함수 : 개발자가 만드는 함수

- 형태

```
함수명()  
함수명( 재료1 )  
함수명( 재료1, 재료2 )  
...  
함수명( 재료1, ... , 재료n)
```

PYTHON 프로그래밍

◆ 콘솔창 입출력 함수

➤ print()

- 콘솔창에 데이터 출력하는 함수
- 각각 값 사이에 공백과 마지막에 줄 바꿈 문자(Wn) 추가

```
>>> print( 'Hello~' )
```

```
Hello~
```

```
>>> print( "Good Luck~^^" )
```

```
Good Luck~^^
```

```
>>> print( 123 )
```

```
123
```

```
>>> print( "Hello~", 124 )
```

```
Hello~ 124
```

```
>>> x = 2017;
```

```
>>> print( "Good Luck~^^", x )
```

```
Good Luck~^^ 2017
```

PYTHON 프로그래밍

◆ 콘솔창 입출력 함수

➤ print() 서식지정자(Format String)

- 원하는 형태의 데이터 출력
- 형태 : %문자

[대표적인 서식지정자]

%d **d**ecimal fixed point
%f **f**loating point
%c **c**haracter
%s **s**tring

%d	부호가 있는 10진수, 정수
%i	부호가 있는 10진수, 정수
%c	단일 문자 (예: 알파벳, 정수 등)
%s	문자열
%f	부호가 있는 10진수, 실수
%u	부호가 없는 10진수, 실수
%o	부호가 없는 8진수, 정수
%x	소문자를 사용한 부호 없는 16진수, 정수
%X	대문자를 사용한 부호 없는 16진수, 정수
%e	e 표기법으로 표현한 실수
%E	E 표기법으로 표현한 실수
%g	%f와 %e중 적절한 하나를 값에 따라서 선택
%G	%F와 %E중 적절한 하나를 값에 따라서 선택
%%	%기호를 출력
%p	포인터의 주소 값 (void *)
%n	포인터의 주소 값 (int *)
%hd	short 자료형을 위한 서식 문자
%ld	long 자료형을 위한 서식 문자
%lf	double 자료형을 위한 서식 문자
%Lf	long double 자료형을 위한 서식 문자

PYTHON 프로그래밍

◆ 콘솔창 입출력 함수

➤ print() 서식지정자(Format String)

```
a=10  
b=12  
print("a+b= %d" %(a+b) )
```

```
a=10  
b=12  
print("a+b= %d, a*b= %d"%( (a+b), (a*b)) )
```

```
year=10  
name="Tome"  
print("Name = %s, Year= %d"%( name, year) )
```

PYTHON 프로그래밍

◆ 콘솔창 입출력 함수

➤ print() 포맷(Format) 문자열 리터럴

```
print( F"a+b= { var } + { var }")
```

```
print( f"a+b= { var } + { var }")
```

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

```
print( f"a+b= {a} + {b}")
```

```
print( F"a+b= {a} + {b}")
```

PYTHON 프로그래밍

◆ 콘솔창 입출력 함수

- `print(~, end= "")`
 - 마지막 줄 바꿈 문자(`\n`) 가 아닌 다른 값으로 변경
- `print(~, sep= "")`
 - 콤마(,)로 구분된 항목 간 출력 문자를 다른 값으로 변경

```
>>> print( 25, " 세" )  
25  세
```

```
>>> print( 25, " 세" , sep="")  
25세
```

PYTHON 프로그래밍

◆ 콘솔창 입출력 함수

- `print(~, file= F)`
 - 출력 내용이 파일 객체로 출력

```
>>> logFile = open('test.txt', 'w')  
  
>>> print( 1,2,3,4,5, file= logFile )  
  
>>> logFile.close()
```

PYTHON 프로그래밍

◆ 콘솔창 입출력 함수

➤ input() 입력함수

- **키보드로부터** 데이터 입력 받는 함수
- 엔터키를 누를 때까지 입력된 **문자열 반환**
- '=' 치환연산자 사용하여 문자열 변수에 저장 가능

```
>>> name = input( 'what's your name?' )  
What's your name? shk
```

```
>>> age = input( 'what's your age?' )  
What's your age? 20
```

```
>>> print( name, "(", age, "세)", sep=" "  
shk(20세)
```

PYTHON 프로그래밍

◆ Built-in 함수

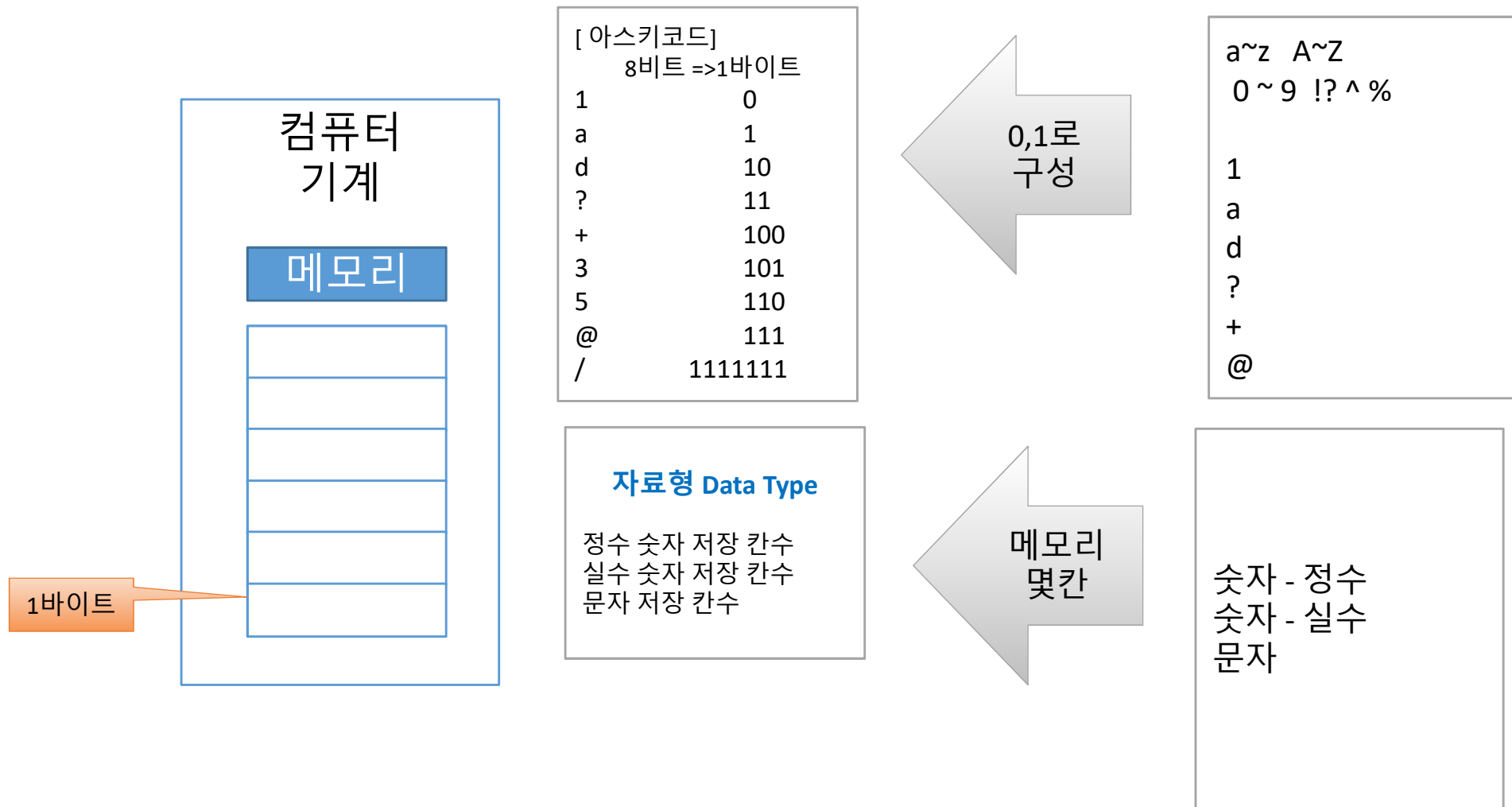
➤ Python 기본 제공되는 함수 www.python.org => DOC 탭

<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

CH05 PYTHON 자료형

PYTHON 자료형

◆ 자료형(Data Type)



PYTHON 자료형

◆ 자료형(Data Type)

➤ 데이터를 저장하기 위한 데이터 종류 → 클래스(class)

기본 자료형		
수치 자료형	정수 (Integer) 타입	양수(+), 0, 음수(-) 저장
	실수 (Float) 타입	실수 저장
	복소수 (Complex) 타입	복소수 저장
문자열 자료형	str 타입	문자열 저장 'aaa' "aaa"
논리 자료형	bool 타입	True, False 저장
bytes 자료형	bytes 타입	0 ~ 255 사이의 코드 저장

PYTHON 자료형

◆ 자료형(Data Type)

➤ 변환(Casting)

- 데이터 타입을 일시적으로 변환

내장함수	변환
int()	Int타입으로 변환
float()	Float타입으로 변환
str()	str타입으로 변환
complex()	complex타입으로 변환
bytes()	bytes타입으로 변환
list()	list타입으로 변환
tuple()	tuple타입으로 변환

PYTHON 자료형

◆ 자료형(Data Type)

➤ 타입 확인

- 데이터의 타입을 알려주는 **type()** 함수

- 형식 ➔

type(변수명)

type(데이터)

- (예) type(데이터) type(100) type('ABC')
- (예) type(변수명) type(age) type(name)

PYTHON 자료형

◆ 자료형(Data Type)

[아스키코드]
8비트 => 1바이트

1	0
a	1
d	10
?	11
+	100
3	101
5	110
@	111
/	1111111

a~z A~Z
0~9 !? ^ %

1
a
d
?
+
@

0,1로
구성

컴퓨터
기계

메모리

마우스
객체
(object)

1바이트

자료형 Data Type

정수 숫자 저장 칸수
실수 숫자 저장 칸수
문자 저장 칸수

메모리
몇칸

숫자 - 정수
숫자 - 실수
문자1개
문자열

문자열

마우스 클래스

마우스 Data Type

형태 문자열 -> 둥글, 사각, 타원
버튼 숫자 -> 2
휠 숫자 -> 1
색상 문자열 -> 핑크, 흰색, 검은색

왼쪽클릭 함수
오른쪽클릭 함수
휠 무빙 함수
드래그 함수

변수

마우스
- 형태
- 버튼개수
- 휠
- 색상
- 왼쪽클릭
- 오른쪽클릭
- 휠 무빙
- 드래그

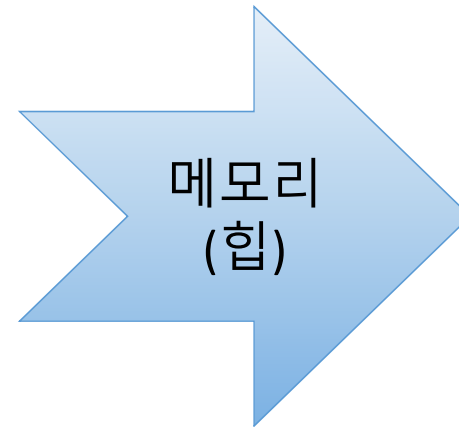
함수

PYTHON 자료형

◆ 자료형(Data Type)

➤ 클래스 (Class)

- 객체 만들기 위한 **틀, 설계도**
- 구성
 - 변 수 : 속성, 특징
 - 메 서드 : 기능, 동작



객체(Object)
인스턴스

- 파이썬 언어
 - 정수, 실수, 논리, 문자열, 바이트 <= 기본 데이터 타입
 - 힙 메모리 영역에 객체로 저장 됨!

클래스

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 숫자 형태로 이루어진 자료형

항목	사용 예
-----	-----
정수	123, -345, 0
실수	123.45, -1234.5, 3.4e10
8진수	0 34, 0 25
16진수	2A, FF
복소수	1 + 2j, -3j

```
>>> a = 123          # 양수 Integer
>>> a = -23          # 음수 Integer
>>> a = 1.2           # 실수 Float
>>> a = -3.12         # 실수 Float
>>> a = 0o127        # 8진수
>>> a = 0x8ff        # 16진수
>>> a = 1+3j         # 복소수
```

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 숫자 표현

2진수 → 0, 1, 10, 11, 100, 101..

8진수 → 0, 1, 2, 3, 4, 5, 6, 7, 10, 11,, 17, 20

10진수 → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,, 19, 20,..

16진수 → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11,, 1F, 20,...

2진수 → **0b**0

8진수 → **0o**0

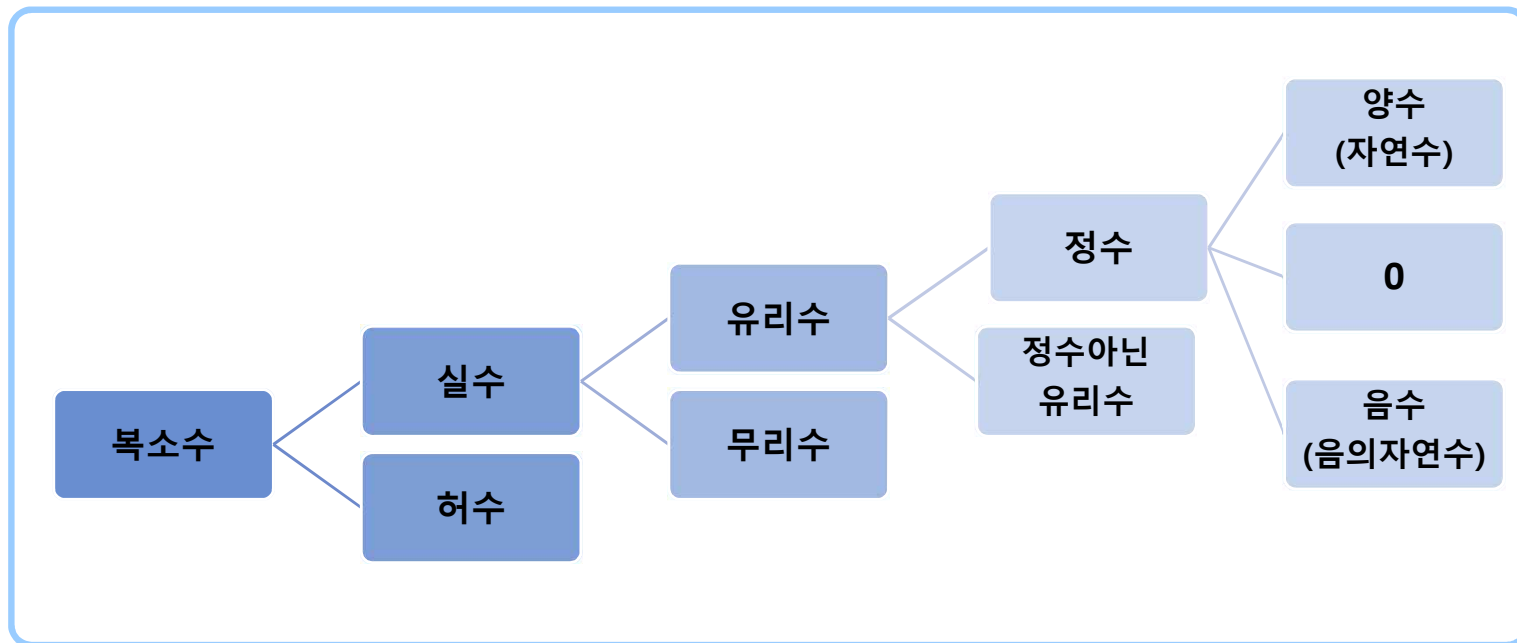
10진수 → 0

16진수 → **0x**0

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 숫자 형태로 이루어진 자료형

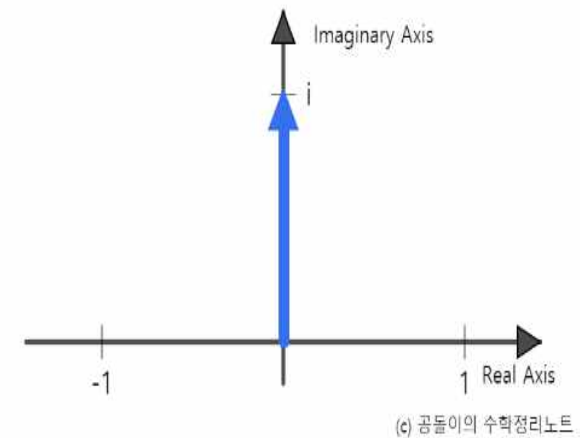


PYTHON 자료형

◆ 수치(Number) 자료형

➤ 숫자 형태로 이루어진 자료형

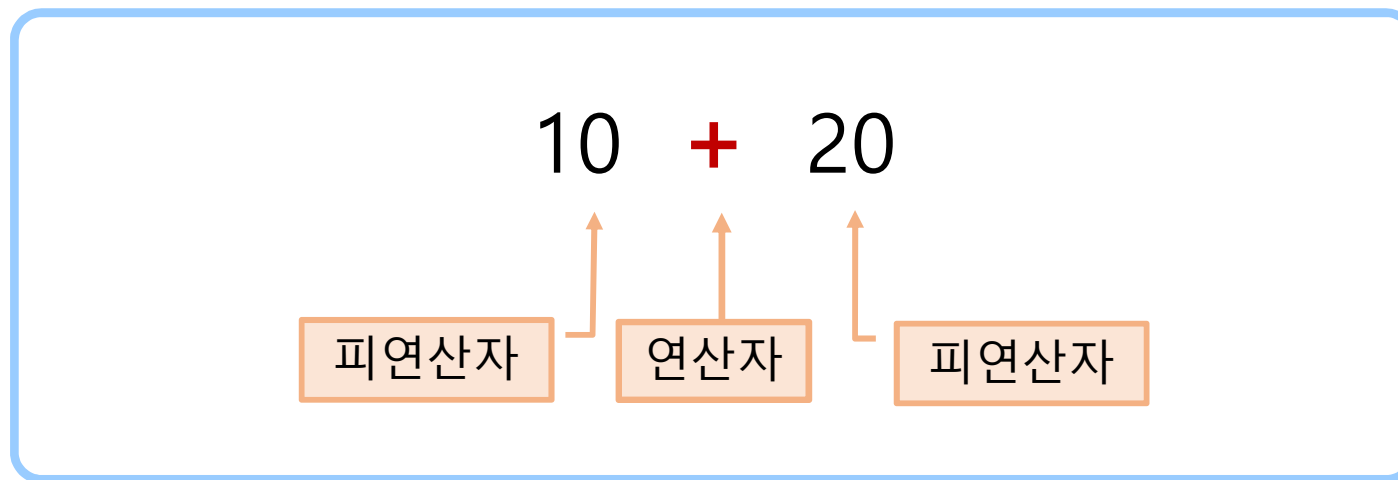
허수 : $x^2 = -1$ 이 되는 수
벡터 개념 등장



PYTHON 자료형

◆ 수치(Number) 자료형

➤ 연산자(Operator)



- 피연산자 개수에 따라 단항, 이항, 삼항연산자

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 연산자(Operator)

연산자		의미
할당연산자	L-value = R-value	변수 값 할당에 사용, 산술연산자와 함께 사용
산술연산자	+, -, *, /, //, %, **	사칙연산, 몫과 나머지, 제곱 연산 수행
비교연산자	>, <, >=, <=, ==, !=	값의 비교로 True, False 결과
논리연산자	and, or, not	왼쪽, 오른쪽의 참, 거짓 관련 체크, True, False 결과
멤버연산자	in, not in	왼쪽 값이 오른쪽에 속하지는 여부 체크, True, False 결과
비트연산자	&, , ^, ~, <<, >>	비트 단위 값 연산 수행
객체비교연산자	is, is not	왼쪽과 오른쪽의 객체 동일 여부 체크, True, False 결과

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 산술연산자

```
a=10
```

```
b=3
```

```
print(f"{a}+{b}={a+b}")
```

더하기

10+3=13

```
print(f"{a}-{b}={a-b}")
```

빼기

10-3=7

```
print(f"{a}*{b}={a*b}")
```

곱하기

10*3=30

```
print(f"{a}/{b}={a/b}")
```

나누기

10/3=3.333333

```
print(f"{a}://{b}={a//b}")
```

몫

10//3=3

```
print(f"{a}%{b}={a%b}")
```

나머지

10%3=1

```
print(f"{a}**{b}={a**b}")
```

제곱근

10**3=1000

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 비교연산자

```
a=10
b=3

print(f"{a}>{b}={a>b}")      # 크다
print(f"{a}<{b}={a<b}")      # 작다
print(f"{a}>={b}={a>=b}")    # 크거나 같다
print(f"{a}<={b}={a<=b}")    # 작거나 같다

print(f"{a}=={b}={a==b}")    # 같다
print(f"{a}!= {b}={a!=b}")   # 같지 않다
```

```
10>3=True
10<3=False
10>=3=True
10<=3=False
10==3=False
10!=3=True
```

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 논리연산자

```
a=10
b=3

print(f"{a}>5 and {b}>5 = {a>5 and b>5}")
print(f"{a}>5 and {b}<5 = {a>5 and b<5}")

print(f"{a}>5 or {b}>5 = {a>5 or b>5}")
print(f"{a}<5 or {b}>5 = {a<5 or b>5}")

print(f"not {a}>5 = {not a>5}")
print(f"not {a}<5 = {not a<5}")
```

10>5 and 3>5 = False

10>5 and 3<5 = True

10>5 or 3>5 = True

10<5 or 3>5 = False

not 10>5 = False

not 10<5 = True

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 비트연산자

- 비트 단위로 연산
- 2진수로 표기
- 형식 : **0b** 접두어

8 : **0b**1000

11 : **0b**1011

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 비트연산자

```
a = 8          # 0000 1000
b = 11         # 0000 1011
```

```
print(f"{a} & {b} = {a & b}")
```

```
print(f"{a} | {b} = {a | b}")
```

```
print(f"{a} ^ {b} = {a ^ b}")
```

```
print(f"2진수 표기 => {a} : {bin(a)}")
```

```
print(f"2진수 표기 => {b} : {bin(b)}")
```

& AND : 두 비트 모두 1
| OR : 두 비트 중 하나 이상 1
^ XOR : 두 비트 서로 다른 경우 1

8 & 11 = 8

8 | 11 = 11

8 ^ 11 = 3

2진수 표기 => 8 : 0b1000

2진수 표기 => 11 : 0b1011

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 비트연산자

0b1000	0b1000	0b1000
&		^
0b1011	0b1011	0b1011
0b1000	0b1011	0b0011

& AND : 두 비트 모두 1
| OR : 두 비트 중 하나 이상 1
^ XOR : 두 비트 서로 다른 경우 1

8 & 11 = 8

8 | 11 = 11

8 ^ 11 = 3

2진수 표기 => 8 : 0b1000

2진수 표기 => 11 : 0b1011

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 객체비교연산자

- 동일 객체 여부 결과 True, False 반환
- is → 동일 객체면 True
- is not → 동일 객체 아니면 True

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 객체비교연산자

```
a = 10
b = a
print(f"{a} is {b} => {a is b}")
print(f"{a} => {id(a)}")
print(f"{b} => {id(b)}")
```

a와 b가 같은 객체인지 확인
id() 존재 메모리 주소

```
b = 10.0
print(f"{a} is {b} => {a is b}")
print(f"{a} is not {b} => {a is not b}")
print(f"{a} == {b} => {a == b}")
```

a와 b가 같은 객체인지 확인
a와 b가 같은 객체 아는지 확인
a와 b의 값이 같은지 확인

```
print(f"{a} => {id(a)}")
print(f"{b} => {id(b)}")
```

```
10 is 10 => True
10 => 140735091155632
10 => 140735091155632
```

```
10 is 10.0 => False
10 is not 10.0 => True
10 == 10.0 => True
```

```
10 => 140735091155632
10.0 => 2381409780432
```

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 내장 함수

- `int(숫자)` : 정수 반환
- `int("문자열")` : 정수 반환, 단 숫자형태 문자열만("10") 가능
- `abs(a)` : a의 절대값 반환
- `pow(a, b)` : a^b (거듭제곱) 반환
- `max(a, b)` : a, b 중 큰 값 반환
- `min(a, b)` : a, b 중 작은 값 반환
- `round(a)` : a를 반올림 한 값 반환

PYTHON 자료형

◆ 수치(Number) 자료형

➤ 내장 함수 - 변환함수

- int(x)
- float(x)
- oct(x)
- hex(x)
- str(x)

```
>>> int(12.3)
12
>>> float(29)
29.0
>>> oct(12)
'0o14'
>>> hex(123)
'0x7b'
>>> str(111)
'111'
>>> type(111)
<class 'int'>
>>> type('111')
<class 'str'>
```

PYTHON 자료형

◆ 문자열(String) 자료형

- 한 개 이상의 문자로 구성된 자료형
- ' 또는 "로 문자열 감싸기

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

```
print('Python is Fun.')
```

```
print("₩'Life is too short~₩'")
```

```
print('₩"Life is too short~₩"')
```

```
print("Python's favorite food is perl")
```

Hello

Python is Fun.

'Life is too short~'

"Life is too short~"

Python's favorite food is perl

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 여러 줄 문자열

```
multiline="Life is too short\nYou need python\nPython is powerful language"  
print(multiline)
```

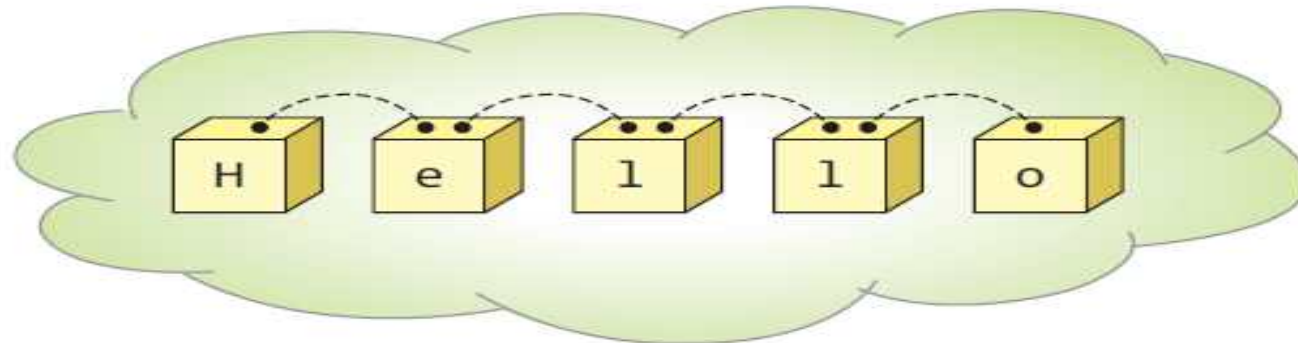
```
multiline="""  
Life is too short  
You need python  
Python is powerful language  
"""  
print(multiline)
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 메모리 공간 할당

a = "abcde"	a	0100 1001
	b	0100 1010
	c	0100 1011
	d	0100 1100
	e	0100 1101



PYTHON 자료형

◆ 문자열(String) 자료형

➤ 인덱싱(Indexing)

- 각 문자는 개별 주소를 가지며 주소로 값 가져오기

```
msg = 'Hello'
```

원소, 요소

Hello

0	1	2	3	4
-5	-4	-3	-2	-1

왼쪽부터 0~
오른쪽부터 -1~

문자열[번호]

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 인덱싱(Indexing)

- 문자열, 리스트, 튜플 등등 자료형 타입에서 데이터 요소 하나하나에 번호를 부여하는 것
- 데이터 덩어리를 **하나씩 또는 일부를 사용하기 위한 방법**

0	1	2	3	4	5	6	7	8	9	10	11	12	13
H	A	P	P	Y		N	E	W		Y	E	A	R
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 인덱싱(Indexing)

```
Life is too short, You need Python
```

```
0           1           2           3  
0123456789012345678901234567890123
```

```
>>> a[-0]
```

```
'L'
```

```
>>> a[-2]
```

```
'o'
```

```
>>> a[-5]
```

```
'y'
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 슬라이싱(Slicing)

- 각 문자의 **인덱스를 기반으로** 문자 **잘라내기**

시작문자 ≤ 문자열 < 끝문자

- 형식 →

문자열[시작 번호 : 끝 번호]

- **주의! 끝 번호 포함 안됨**

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 슬라이싱(Slicing)

- 형식 >> [시작번호 : 끝 번호]
- 주의 >> 끝 번호는 포함 안됨

[0 : 5] 0 ≤ ~ < 5

0 1 2 3 4 5 6 7 8 9 10 11 12 13

H A P P Y N E W Y E A R

-14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1

[-14 : -9]

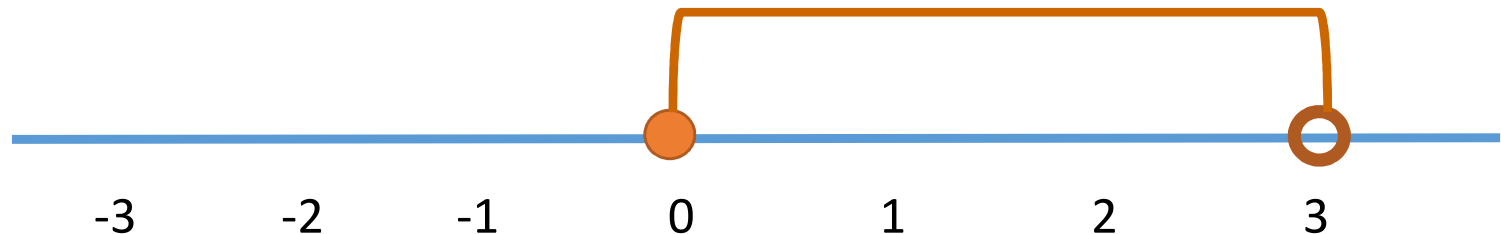
PYTHON 자료형

◆ 문자열(String) 자료형

➤ 슬라이싱(Slicing)

- 형식 >> [시작번호이상 : 끝 번호 미만]
- 주의 >> 끝 번호는 포함 안됨

0, 1, 2 => 0 ≤ 숫자 < 3
시작인덱스 이상 ≤ < 끝 인덱스



PYTHON 자료형

◆ 문자열(String) 자료형

➤ 슬라이싱(Slicing)

```
>>> a = "Life is too short, You need Python"
>>> a[0:4]
'Life'

>>> a[0:3]
'Lif'

>>> a[19:]
'You need Python'

>>> a[:17]
'Life is too short'

>>> a[:]
'Life is too short, You need Python'

>>> a[19:-7]
'You need'
```

PYTHON 자료형

◆ 문자열(String) 자료형

[문자 하나 하나씩 출력]

```
>>> Msg="Good-Luck"
```

```
>>> print( Msg[0], Msg[1], Msg[2])
```

```
>>> Msg[3]
```

```
>>> print(Msg)
```


PYTHON 자료형

◆ 문자열(String) 자료형

[일부 문자 출력]

```
>>> Msg="Good-Luck"
```

```
>>> len(Msg)           #문자열 길이
```

```
>>> Msg[0:3]
```

```
>>> Msg[3:]
```

```
>>> Msg[:-3]
```

```
>>> Msg[:]             #문자열 처음부터 끝까지
```

PYTHON 자료형

◆ 문자열(String) 자료형

[문자열 나누기]

```
>>> data="Happy New Year 2020"
```

```
>>> message=data[:14]
```

```
>>> message
```

```
>>> year=data[ len(message)+1 : ]
```

```
>>> year
```

PYTHON 자료형

◆ 문자열(String) 자료형

[특정 문자 바꾸기]

```
>>> data='Pithon'
```

```
>>> data[1] = 'y'    # Error
```

```
>>> data=data[:1]+'y'+data[2:]
```

```
>>> data
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 문자열 → 숫자 변환(캐스팅 Casting)

```
t = input("정수를 입력하세요: ")
```

```
x = int( t )
```

```
t = input("정수를 입력하세요: ")
```

```
y = int( t )
```

```
print(x+y)
```

<class 'str'> type

<class 'int'> type

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 숫자 → 문자열 변환

```
>>> print("나는 현재 " + str(21) + "살이다.")
```

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

```
print('나는 현재 ' + 21 + '살이다.')
```

TypeError: Can't convert 'int' object to str implicitly

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 숫자 → 문자열 변환

```
>>> print("나는 현재 " + str(21) + "살이다.")
```

나는 현재 21살이다.

```
>>> print("원주율은 " + str(3.14) + "입니다.")
```

원주율은 3.14입니다.

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 이스케이프 문자

- 특수 기능 수행 문자
- 형식 ₩문자

이스케이프 문자	설 명
\n, ₩n	개행(Newline, 줄바꿈)
\t, ₩t	탭(Tab)
\0, ₩0	NULL 문자
\\, ₩₩	문자 '\'
\', ₩'	단일 인용부호(')
\", ₩"	이중 인용부호(")

가
비
미
X

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 이스케이프 문자

```
print("오늘은 ₩'수요일₩' 입니다.")
```

```
print('오늘은 ₩'수요일₩' 입니다.')
```

```
print("오늘은 ₩n날씨가 출근요.₩n")
```

```
print('₩n오늘은 ₩n날씨가 출근요.₩n')
```

```
print("오늘은₩t날씨가₩t출근요")
```

```
print("오늘은 ₩"크리스마스₩" 입니다.")
```


PYTHON 자료형

◆ 문자열(String) 자료형

➤ Raw String

- 문자열 내 이스케이프문자 무시
- 형식 ==> `r"문자열"`
- 경로, URL 문자열 처리에 자주 사용됨

```
print( "오늘은 날씨가 너무\n 좋군요" )
```

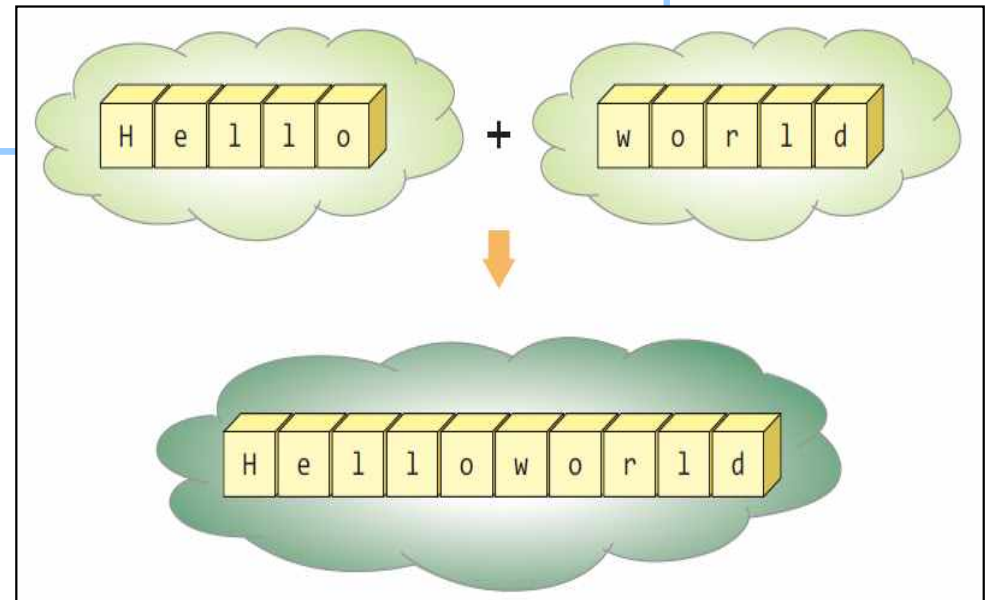
```
print( r"오늘은 날씨가 너무\n 좋군요" )
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 합치기 (Concatenation) ➔ + 연산자

```
>>> head = "Hello"  
>>> tail = "World"  
>>> print(head + tail)
```



PYTHON 자료형

◆ 문자열(String) 자료형

➤ 합치기 (Concatenation) ➔ + 연산자

```
msg="Good Luck"
name="Hong"
total=msg+name
print("id(msg) =>", id(msg))
print("id(name) =>", id(name))
print("id(total) =>", id(total))

# int 타입 + str 타입
print(str(2021)+"Happy New Year")
print("Happy New Year"+str(2021))
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 곱하기 → * 연산자

- '*'의미는 숫자 곱하기의 의미와는 다르게 사용
- '*'의미는 문자열의 **반복** 의미

```
>>> print("="* 5)
```

```
>>> print("My Program ")
```

```
>>> print("="* 5)
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 변수값 포함 → % 연산자

```
>>> price=10000
```

```
>>> print("상품 가격은 %s원입니다." %price)
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ is , not is 연산자

```
# a in B : a가 B안에 존재하면 True
```

```
print( "P" in "Price is 1000")
```

```
print( "p" in "Price is 1000")
```

```
print( "is" in "Price is 1000")
```

```
# a not in B : a가 B안에 존재하지 않으면 True
```

```
print( "is" not in "Price is 1000")
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 포매팅

- 원하는 형식의 문자열 생성하는 format()메서드

index=value 형식 → "문자열 { index }". format(value , ...)

```
year=2020  
month=11  
day=6
```

```
msg="오늘은 {0}년 {1}월 {2}일 입니다.".format(year, month, day)  
print("msg =>", msg)
```

```
msg="오늘은 {}년 {}월 {}일 입니다.".format(year, month, day)  
print("msg =>", msg)
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 포매팅

- 원하는 형식의 문자열 생성하는 format()메서드

name=value 형식 ➔

“문자열 { name }”.format(name=value , ...)

```
msg="{y}년 {m}월 {d}일 입니다.".format( d=25, y=2020, m=12)
```

```
print("msg =>", msg)
```


PYTHON 자료형

◆ 문자열(String) 자료형

➤ 포매팅

분류	형태	설명
정렬	:<n	총자릿수n,<왼쪽정렬
	:>n	총자릿수n,>오른쪽정렬
	:^n	총자릿수n,^가운데정렬
공백채우기	:문자^n	총자릿수n,공백대신채울문자
소수점	:n.xf	총자릿수n,소수점표현자릿수x
'{'','}' 표현	{{}}	중괄호({ }) 그대로 표현

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 포매팅

```
print(" {0:<10} 님! 좋은 하루되세요." .format("Happy") )  
print(" {0:>10} 님! 좋은 하루 되세요." .format("Happy") )  
print(" {0:^10} 님! 좋은 하루 되세요." .format("Happy") )  
print(" {0:=^10} 님! 좋은 하루 되세요." .format("Happy") )  
print(" {:>10.5f} 님! 좋은 하루 되세요." .format(12.12345678) )  
print("그림_{:0>3}.jpg, 그림_{:0>3}.jpg 저장!".format(1, 123) )
```

```
안녕하세요. Happy 님! 좋은 하루되세요.  
안녕하세요. Happy 님! 좋은 하루 되세요.  
안녕하세요. Happy 님! 좋은 하루 되세요.  
안녕하세요. ==Happy== 님! 좋은 하루 되세요.  
안녕하세요. --12.12346 님! 좋은 하루 되세요.
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 메서드

함수	기능
count(문자)	문자열내문자개수세기
find(문자)	문자가처음나오는위치알려주기-1:존재하지않은문자
index(문자)/index(문자열)	문자또는문자열의위치알려줌,없을경우error발생
join(문자열)	문자열합치기
lstrip()	왼쪽공백지우기
rstrip()	오른쪽공백지우기
strip()	양쪽공백지우기
replace(이전문자열,바꿀문자열)	문자열내특정문자열치환
split(구분자)	문자열을구분자기준으로문자열나누어리스트에담기

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 메서드

```
# replace() => 원본 문자열을 새로운 문자열로 변환
# replace(old, new, [count])

msg = 'AB-C-D-EFGH'
print("msg =>", msg)
print("msg.replace('-', '') =>", msg.replace('-', ''))
print("msg.replace('-', '', 2) =>", msg.replace('-', '', 2))
print("msg =>", msg)
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 메서드

```
# strip() => 양쪽 공백 모두 제거  
# lstrip() => 왼쪽 공백만 제거  
# rstrip() => 오른쪽 공백만 제거  
msg = '    1234 56    '  
print("msg =>", msg)  
print("msg.strip()=>", msg.strip())  
print("msg.lstrip()=>", msg.lstrip())  
print("msg.rstrip()=>", msg.rstrip())  
print("msg =>", msg)
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 메서드

```
# split() => 구분자를 기준으로 문자열 나누기
msg = '1/2/3/4/5/6'
data=msg.split('/')    # 나누어진 문자열 리스트에 담아 반환
print("msg =>", msg)
print("data =>", data, type(data))

#join() => 구분자 통해 각각의 리스트 요소들 순서대로 연결
sep = "-"
print(sep.join(data))
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 메서드

```
msg="15 kim Daegu 010-222-1111"
```

```
data=msg.split(' ')
```

```
print("data =>", data, type(data))
```

```
sep = ","
```

```
print(sep.join(data))
```

PYTHON 자료형

◆ 문자열(String) 자료형

➤ 메서드

```
# find() => 문자열 내에서 문자열 찾아서 인덱스 반환  
#          없으면 -1
```

```
msg = 'It`s always such a pleasure'
```

```
print("#nmsg =>", msg)
```

```
print("msg.find('way') =>", msg.find('way'))
```

```
print("msg.find('such') =>", msg.find('such'))
```

```
print("msg.find('such', 13) =>", msg.find('such', 13))
```


PYTHON 자료형

◆ 문자열(String) 자료형

➤ 메서드

```
# index() => 문자열 내에서 문자열 찾아서 인덱스 반환  
# #          없으면 Error 발생  
  
print("msg.index('way') =>", msg.index('way'))  
print("msg.index('such', 13) =>", msg.index('such', 13))
```

PYTHON 자료형

◆ 순서 관련 자료형

불변시퀀스	str타입	class 'str'	문자열 저장
	tuple타입	class 'tuple'	다양한 데이터타입 저장
	bytes타입	class 'bytes'	바이트 데이터저장
변경시퀀스	list타입	class 'list'	다양한 데이터타입 저장
	bytearrays타입	class 'bytearray'	바이트 데이터 저장

PYTHON 자료형

◆ 순서 관련 자료형

- 여러 개의 자료들을 모아서 하나로 묶음으로 저장
 - 요소들이 연속적으로 이루어진 자료형
- 인덱스([인덱스]) 사용하여 요소 접근
 - 슬라이싱으로 부분 요소 추출
- in, not in 연산자로 요소(element) 존재 여부
 - + 연산자 이용한 데이터 연결
 - * 연산자 이용한 데이터 반복

PYTHON 자료형

◆ 순서 관련 자료형

➤ 내장 메서드

함수	기능
del 요소 또는 del(요소) del 변수명	요소 삭제 삭제
len(변수명)	길이
sum(변수명)	요소 합계 (추치 데이터만 가능)
max(변수명) , min(변수명)	최대 요소, 최소 요소
sorted(변수명)	요소 정렬

PYTHON 자료형

◆ 순서 관련 자료형

➤ 내장 메서드 range()

기 능 : 범위 지정 데이터 range객체 생성

문 법 : range(숫자) : 0 ~ (숫자-1) 연속된 데이터

range(시작, 숫자) : 시작 ~ (숫자-1) 연속된 데이터

range(시작, 숫자, 간격) : 시작 ~(숫자-1) , 간격 연속 데이터

반 환 : range 객체 반환

PYTHON 자료형

◆ List 자료형

- 여러 개의 자료들을 모아서 하나로 묶음으로 저장하는 것
- 요소들이 연속적으로 이루어진 자료형
- 대괄호[]로 감싸고 쉼표(,)로 구분
- 여러 종류 타입의 데이터 저장 가능

```
변수명=[ 값1, 값2, 값3 ]
```

```
변수명=[]     ← 공백 리스트 생성
```

PYTHON 자료형

◆ List 자료형

➤ 생성

<code>slist=[]</code>	<i>#빈 리스트</i>
<code>number=[1,2,3,4]</code>	<i>#정수 리스트</i>
<code>names=["Jane", "Tom", "Lee"]</code>	<i># 문자열 리스트</i>
<code>exist=[False, False, True, False]</code>	<i># 논리형 리스트</i>
<code>data=["Kim", 30, 23.23]</code>	<i># 다양한 리스트</i>

PYTHON 자료형

◆ List 자료형

➤ 요소 인덱싱

```
data2=[1, 2, 3, ['a', 'b', 'c']]
```

```
data2[0]
```

```
data2[3]
```

```
data2[3][0]
```

```
data3=[1, 2, 3, ['a', 'b', 'c', [11, 22,33]]]
```

```
data3
```


PYTHON 자료형

◆ List 자료형

➤ List 메서드

함수	기능
append(x)	리스트의 맨 마지막에 x 추가
sort(key=None, reverse=False) sort(reverse=True)	오름차순으로 리스트 정렬 내림차순으로 리스트 정렬
reverse()	현재 리스트 요소 순서를 역순으로 뒤집기, 정렬이 되지는 않음
index(x)	리스트에 x 값 존재할 경우 위치값 리턴 / 없으면 오류
insert(a, b)	리스트의 a번째 위치에 b 삽입
remove(x)	리스트에서 처음 나오는 x 삭제

PYTHON 자료형

◆ List 자료형

➤ List 메서드

함수	기능
pop(), pop(x)	리스트 맨 마지막 요소 꺼내기 리스트의 x번째 요소 꺼내기
count(x)	리스트에 포함된 요소 x 개수 세기
extend(x)	리스트에 x리스트를 더하기
clear()	리스트의 모든 요소 삭제

PYTHON 자료형

◆ Tuple 자료형 ← Read Only List

- 요소들이 연속적으로 이루어진 자료형
- 소괄호()로 감싸고 쉼표(,)로 구분
- 한 개의 요소를 가질 경우 반드시 요소 뒤에 콤마(,)
- 소괄호() 생략 가능
- 여러 종류 타입의 데이터 저장 가능

변수명=(값1, 값2, 값3)

변수명= 값1, 값2, 값3

변수명= (값1,)

변수명= 값1,

PYTHON 자료형

◆ Tuple 자료형

➤ 생성

stuple=()	#빈 튜플
number=(1,2,3,4)	#정수 튜플
names=("Jane", "Tom", "Lee")	# 문자열 튜플
exist=(False, False, True, False)	# 논리형 튜플
data=("Kim", 30, 23.23)	# 다양한 튜플
alone='A',	# 문자열 튜플

PYTHON 자료형

◆ Tuple 자료형

➤ 강제 요소 변환

```
data1 = (1, 2, 3, 4)
print("data1 => ", data1)
```

```
tlist = list(data1)                                # list 타입으로 변환
print(tlist, type(tlist))
```

```
tlist[1] = 222                                     # 값 변경
```

```
data1 = tuple(tlist)                                # tuple로 변환
print("data1 => ", data1)
```

PYTHON 자료형

◆ 맵핑 자료형

Dictionary	dic타입	class 'dic'	다양한 데이터
------------	-------	-------------	---------

PYTHON 자료형

◆ Dict 자료형

- Key와 Value를 쌍으로 구성
- Key를 통해서 Value을 얻는 방식
- 연관 배열 또는 해시(Hash)라고 함
- 중괄호{ }로 묶고 key:value를 콤마(,)로 구분
- 순서가 의미 없음
- 주의!! Key 중복, 중복 시 하나의 키 제외한 나머지 무시

```
변수명={ 'key1':'value1', 'key2':'value2', 'key3':'value3' }
```

PYTHON 자료형

◆ Dict 자료형

➤ 생성

```
dic = {}
```

```
dic1={"name":"Tom", "age":10, "birth":"1224"}
```

```
dic2={1:"Hi", 2:"Apple", "Num":1}
```

```
dic3={"a":[1,2,3], "b":123}
```

```
dic4={1:("A","B",123), 2:"Happy"}
```


PYTHON 자료형

◆ Dict 자료형

➤ 값 읽기 → 객체변수명[Key]

```
print("\n==== Dic객체 요소 값 읽기 =====")
```

```
print(f"dic1[1]    => {dic1['name']}")
```

```
print(f"dic5[1]    => {dic5[1]}")
```

```
print(f"dic6[(1,2)] => {dic6[(1,2)]}")
```

PYTHON 자료형

◆ Dict 자료형

➤ 값 변경 및 요소 추가 → 객체변수명[Key] = Value

```
print("\n==== Dic객체 요소 값 변경 & 추가 =====")
```

```
print(f"[전] dic1  => {dic1}")
```

```
dic1['name']="KKK"
```

존재 키일 경우 '변경'

```
print(f"[후] dic1  => {dic1}")
```

```
dic1['location']="DAEGU"
```

미존재 키일 경우 '추가'

```
print(f"[후] dic1  => {dic1}")
```

PYTHON 자료형

◆ Dict 자료형

➤ 요소 삭제

```
print("\n===== Dic객체 요소 삭제 =====")  
del dic1['name']  
print(f"[후] dic1  => {dic1}")  
  
del(dic1['location'])  
print(f"[후] dic1  => {dic1}")
```

PYTHON 자료형

◆ Dict 자료형

함수	기능
keys()	딕셔너리의 Key 리스트 리턴 => 2.7까지 딕셔너리의 dict_keys 객체 리턴 => 3.0 이후
values()	딕셔너리의 Value 리스트 리턴 => 2.7까지 딕셔너리의 dict_values 객체 리턴 => 3.0 이후
items()	딕셔너리의 Key-Value 쌍 튜플로 묶은 dict_items 객체 리턴
clear()	모든 쌍 지우기 즉, 딕셔너리안의 모든 요소 삭제
get(key)	Key로 Value 값 얻기 Key에 해당 값이 없으면 None 리턴
get(key , default)	Key에 해당 값이 없으면 기본값 리턴
key in 변수명	Key가 딕셔너리 안에 존재하는 지 체크 True / False 리턴

PYTHON 자료형

◆ Dict 자료형

➤ 메서드

```
mydic={"name":"Tom", "age":10, "birth":"1224"}  
print("\n===== Dic객체 메서드 =====")  
values=mydic.values()  
print(f"values=> {type(values)}, {values}")  
  
keys=mydic.keys()  
print(f"keys=> {type(keys)}, {keys}")  
  
items=mydic.items()  
print(f"items=> {type(items)}, {items}")
```

PYTHON 자료형

◆ Dict 자료형

➤ 메서드

```
print(f"mydic.get('name')=> {mydic.get('name')}")
print(f"mydic.get('location', '---')=> {mydic.get('location', '---')}")

print(f"[전] mydic=> {type(mydic)}, {mydic}")
mydic.update({'name':'Blue Sky', 'age':99, 'gender':'F'})
print(f"[후] mydic=> {type(mydic)}, {mydic}")

mydic.clear()
print(f"[후] mydic=> {type(mydic)}, {mydic}")
```

PYTHON 자료형

◆ 집합 자료형

변경집합	set타입	class 'set'	다양한 데이터
불변집합	frozenset타입	class 'frozenset'	다양한 데이터

PYTHON 자료형

◆ Set 자료형

- 파이썬 2.3부터 지원된 자료형
- 집합 데이터 관리
- 중복 불허, 순서 없음

변수명=set(데이터)

PYTHON 자료형

◆ Set 자료형

➤ 생성

```
numSet1 = { 3, 5, 7, 2, 4, 5, 7 }
```

```
print(f"numSet1 => {type(numSet1)}, {numSet1}")
```

```
numSet1 = {"1", 3, 5, (1,3)}
```

```
print(f"numSet1 => {type(numSet1)}, {numSet1}")
```

```
#numSet1 = {"1", 3, 5, [1,3]} # 수정가능 타입은 사용 불가
```

```
print(f"numSet1 => {type(numSet1)}, {numSet1}")
```

```
numSet1 = {3, 5, 7, 3, 1, 2, 19, 21, 3, 5}
```

```
print(f"numSet1 => {type(numSet1)}, {numSet1}")
```

PYTHON 자료형

◆ Set 자료형

➤ 메서드 → set객체변수명.메서드()

함수	기능
intersection(집합변수)	교집합 $S1 \ \& \ S2$ <code>S1.intersection(S2)</code>
union(집합변수)	합집합 $S1 \ \ S2$ <code>S1.union(S2)</code>
difference(집합변수)	차집합 $S1 - S2$ <code>S1.difference(S2)</code>
add(값)	1개 값 추가 <code>s1.add(4)</code>
update(여러 개 값)	여러 개 값 한꺼번에 추가 <code>s1.update([1,2,3])</code>
remove(값)	특정 값 제어 <code>s1.remove(값)</code>

CH05. PYTHON 조건문

PYTHON 조건문

◆ 조건문

- 프로그램 실행이 **특정 조건에 따라 실행 방향 변경**
- 조건이 **참인 경우와 거짓인 경우**에 따라서 실행
- 주의 !!!
 - 조건문에 속하는 모든 문장은 **들여쓰기**

PYTHON 조건문

◆ 조건문

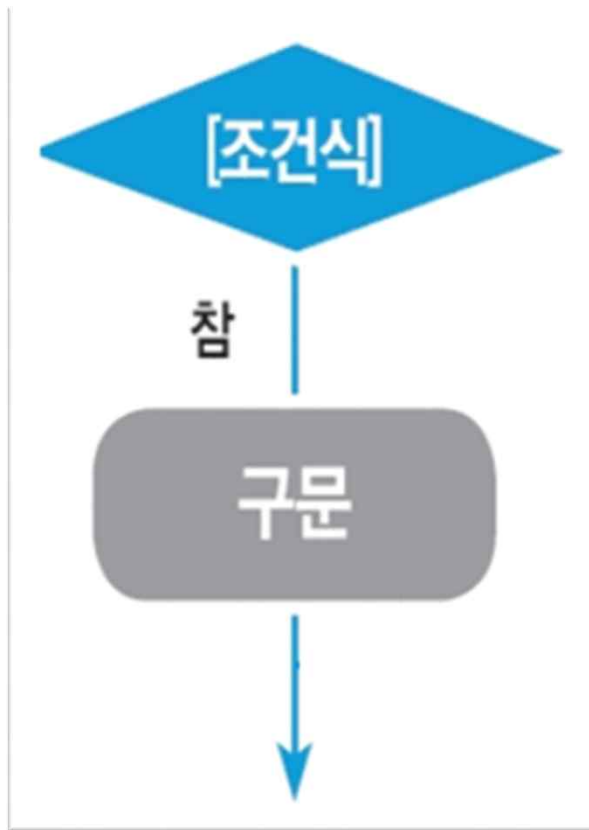
■ 자료형에 따른 조건 결과

자료형	참	거짓
숫자형 타입	0이 아닌 모든 숫자	0
문자형 타입	"abc"	""
리스트 타입	[1,2,3,4,5]	[]
튜플 타입	(1, 2, 3, 4, 5)	()
딕셔너리 타입	{ "A" : 90 }	{ }

PYTHON 조건문

◆ 조건문

➤ 단순 조건문



if 조건식:

구문

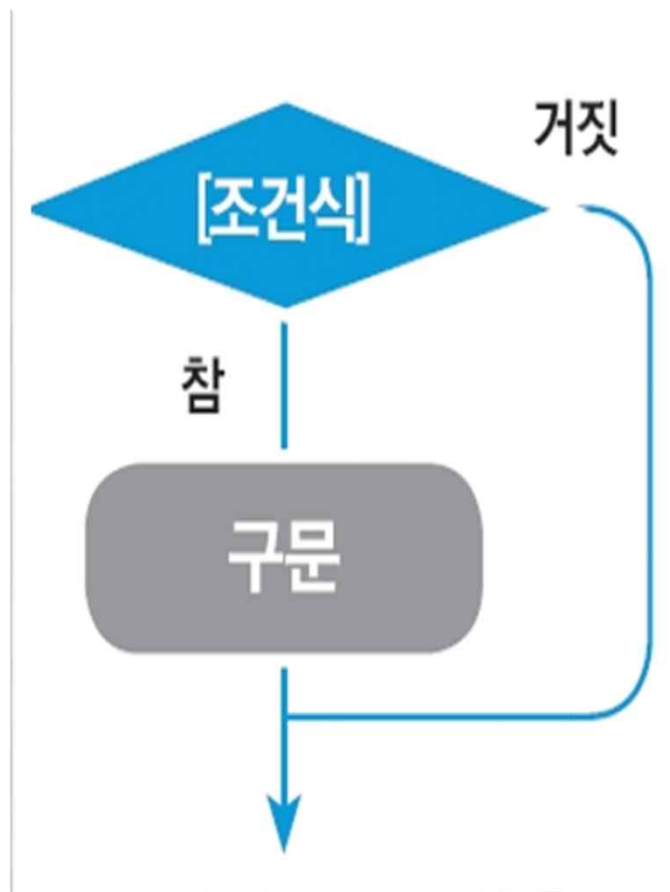
구문

구문

PYTHON 조건문

◆ 조건문

➤ 단순 조건문



```
if 조건식:  
    구문  
    구문  
else :  
    구문  
    구문
```

PYTHON 조건문

◆ 조건문

➤ 예제

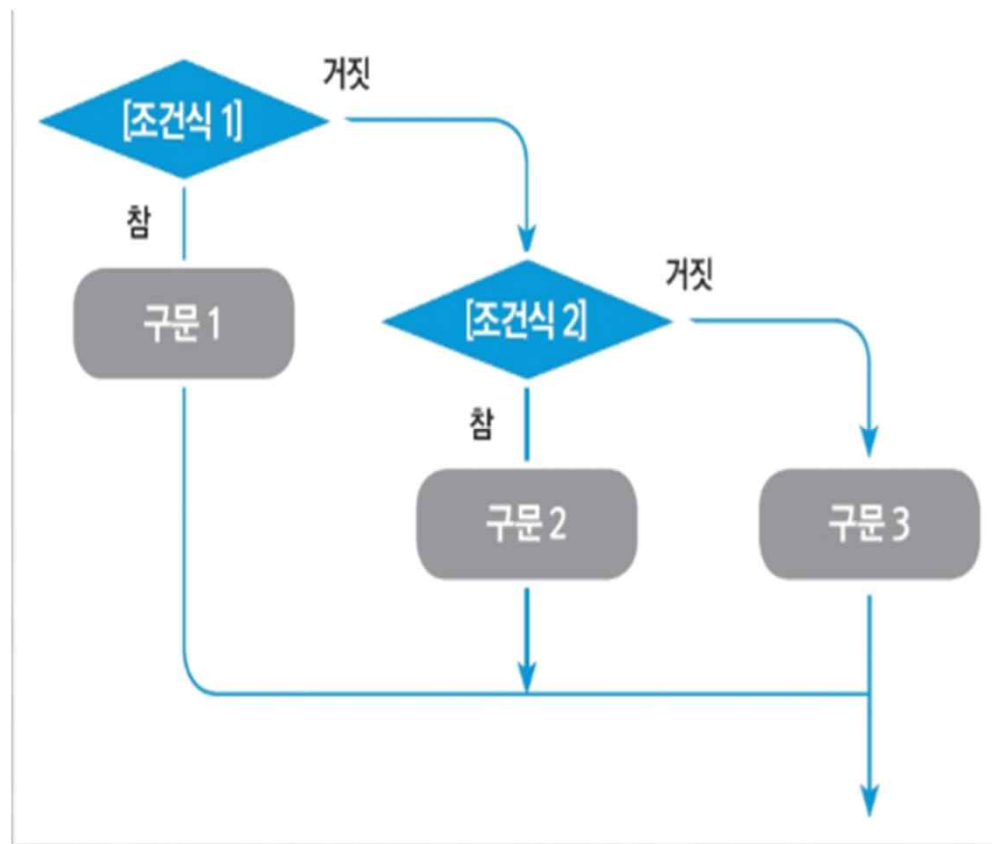
```
if score >= 60 :  
    print("합격입니다.")  
else:  
    print("불합격입니다.")
```

```
if num % 2 == 0 :  
    print("짝수입니다.")  
else:  
    print("홀수입니다.")
```


PYTHON 조건문

◆ 조건문

➤ 다중 조건문



```
if 조건식1:  
    구문  
    구문  
elif 조건식2:  
    구문  
    구문  
else:  
    구문
```

PYTHON 조건문

◆ 조건문

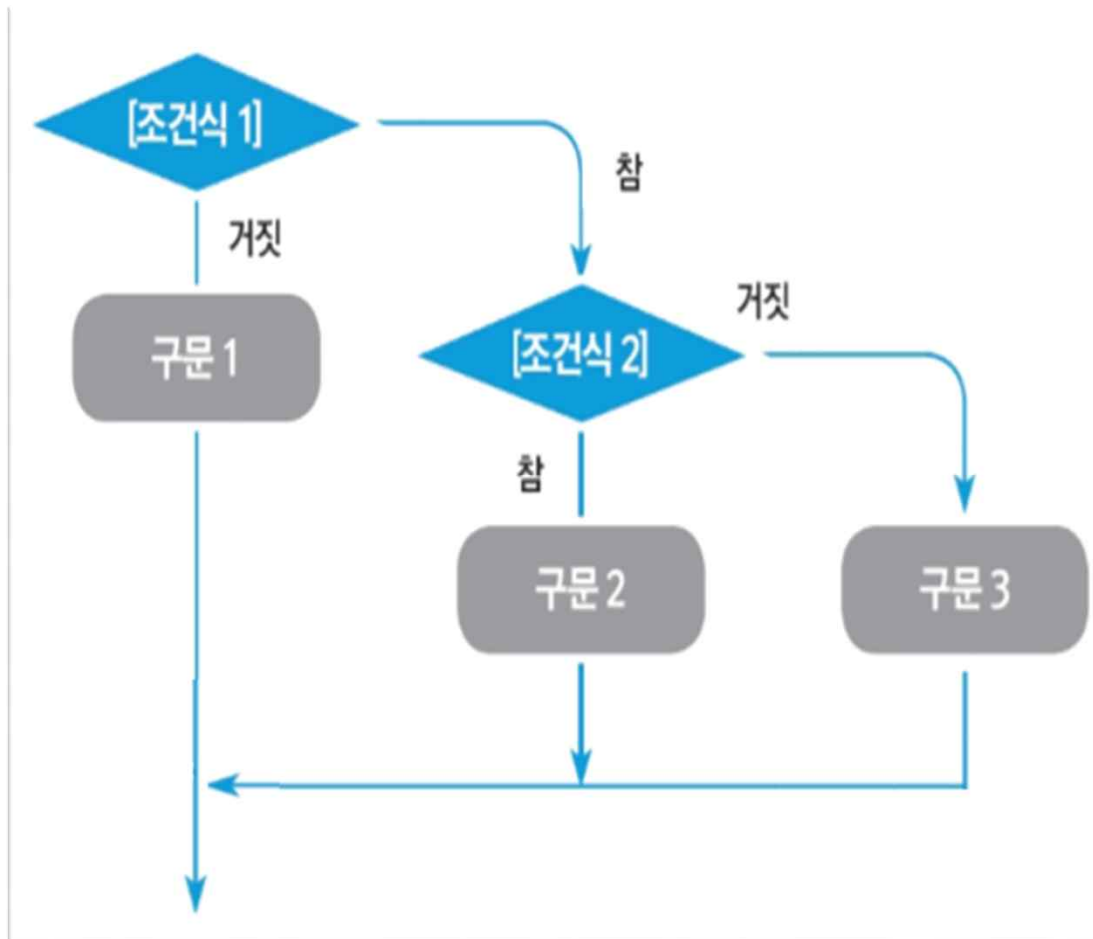
➤ 다중 조건문

```
if value>0:  
    print("양수")  
elif value <0:  
    print("음수")  
else:  
    print("0")
```

PYTHON 조건문

◆ 조건문

➤ 중첩 조건문



```
if 조건식1:
    if 조건식2:
        구문2-1
    else:
        구문2-2
else:
    구문1
```

PYTHON 조건문

◆ 조건문

➤ 중첩 조건문

```
if value > 0:
    if value % 2:
        print("홀수")
    else:
        print("짝수")
else:
    print("음수")
```

PYTHON 조건문

◆ 조건문

➤ 특이한 조건문

in	not in
X in 리스트	X not in 리스트
X in 튜플	X not in 튜플
X in 문자열	X not in 문자열

```
>>> 1 in [1, 2, 3]           True
>>> 1 not in [1, 2, 3]       False
```

```
>>> 'a' in ['a', 'b', 'c']   True
>>> 'i' not in 'python'      True
```

PYTHON 조건문

◆ 조건문

➤ 조건부 표현식

참인 경우 **if** 조건문 **else** 거짓인 경우

```
print("odd number") if value%2 else print("even number")
```

```
kind="Odd Number" if value%2 else "Even Number"
```

PYTHON 조건문

◆ 조건문

➤ 조건부 표현식

```
print("홀수") if value%2 else print("짝수")
```

```
print("짝수") if value%2==0 else print("홀수")
```

```
msg="홀수" if value%2 else "짝수"
```

```
msg="짝수" if value%2==0 else "홀수"
```

CH06 PYTHON 반복문

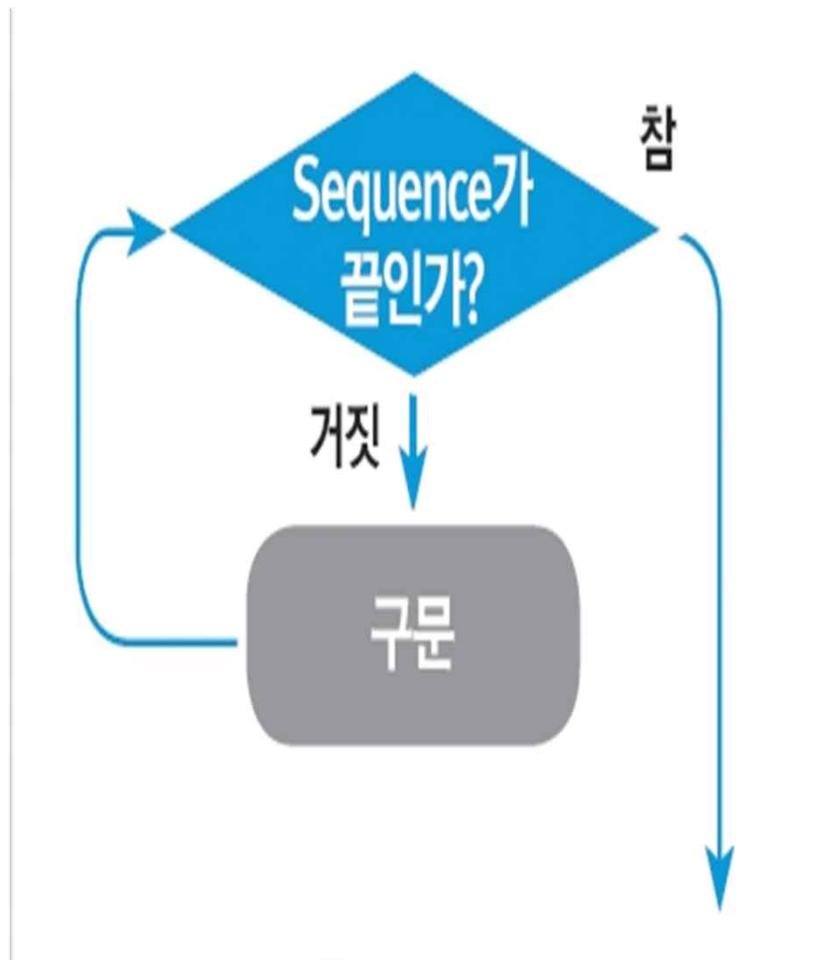
PYTHON 반복문

◆ 반복문

- 동일한 코드 반복해야 할 경우 사용되는 제어문
- 종류 : for문, while문

PYTHON 반복문

◆ 반복문 - for



for 요소 **in** 시퀀스 객체 :
반복문

시퀀스객체에서
하나씩 원소(요소) 가져오기

PYTHON 반복문

◆ 반복문 - for

```
data=[10, 3.25, "A"]  
for item in data:  
    print( item )
```

```
data=[11,22,33,44,55]  
for item in data:  
    print( type(item), item )
```

```
data="Good Luck"  
for item in data:  
    print( item, end='\\\\t' )
```

PYTHON 반복문

◆ 반복문 - for

```
data={'name':'Hong', 'age':10, 3:10.23}
```

```
for (k, v) in data.items():
```

```
    print("Item => {},{}".format(k, v))
```

```
a=[(1,2), (3,4),(5,6)]
```

```
for (first, last) in a:
```

```
    print( "{0} + {1} = {2}".format(first, last, first + last) )
```

PYTHON 반복문

◆ 반복문 - for

```
for 요소 in range(종료 값) :
```

반복문

```
for 요소 in range(시작, 종료, 증가) :
```

반복문

PYTHON 반복문

◆ 반복문 - for

```
for i in range(10):  
    print("Item => {}".format(i), end='\\t')  
  
print("")
```

```
for i in range(2, 11, 2) :  
    print("Item => {}".format(i), end='\\t')  
  
print("")
```

PYTHON 반복문

◆ 반복문 - for

➤ 중첩 for

for 요소 **in** 시퀀스 객체 :

for 요소 **in** 시퀀스 객체 :

반복문

PYTHON 반복문

◆ 반복문 - for

➤ 중첩 for

```
for dan in range(2,10):  
    print(f"====={dan}단=====")  
  
for num in range(1,10):  
    print(f"{dan}*{num}={dan*num}")
```


PYTHON 반복문

◆ 반복문 - for

■ Comperhesion

- 반복 표현식을 포함하여 변형된 다른 Sequence 생성

```
list_objet = [ 표현식 for 요소 in 반복가능 객체 if 조건 ]  
             [ 표현식 if 조건 else 표현식 for 요소 in 반복가능 객체 ]
```

```
set_objet = { 표현식 for 요소 in 반복가능 객체 if 조건 }  
            { 표현식 if 조건 else 표현식 for 요소 in 반복가능 객체 }
```

```
dict_objet = { key:value for 요소 in 반복가능 객체 if 조건 }  
            { key:value if 조건 else 표현식 for 요소 in 반복가능 객체 }
```

PYTHON 반복문

◆ 반복문 - for

```
aa=[1,2,3]
result=[]
for num in aa:
    result.append(num*3)
```

List Comprehension

```
result=[ num*3 for num in aa ]
```

```
result=[ num*3 for num in aa if num==2 ]
```

```
result=[ num*3 if num==2 else num for num in aa ]
```

PYTHON 반복문

◆ 반복문 - enumerate 객체

- 시퀀스 데이터의 요소 꺼낼때 **인덱스 부여 반환** 객체
- 반환 : enumerate 객체
- 문법 : range(data) : 인덱스 0부터 데이터 구성
range(data,start=number) : 인덱스 number부터 데이터 구성

PYTHON 반복문

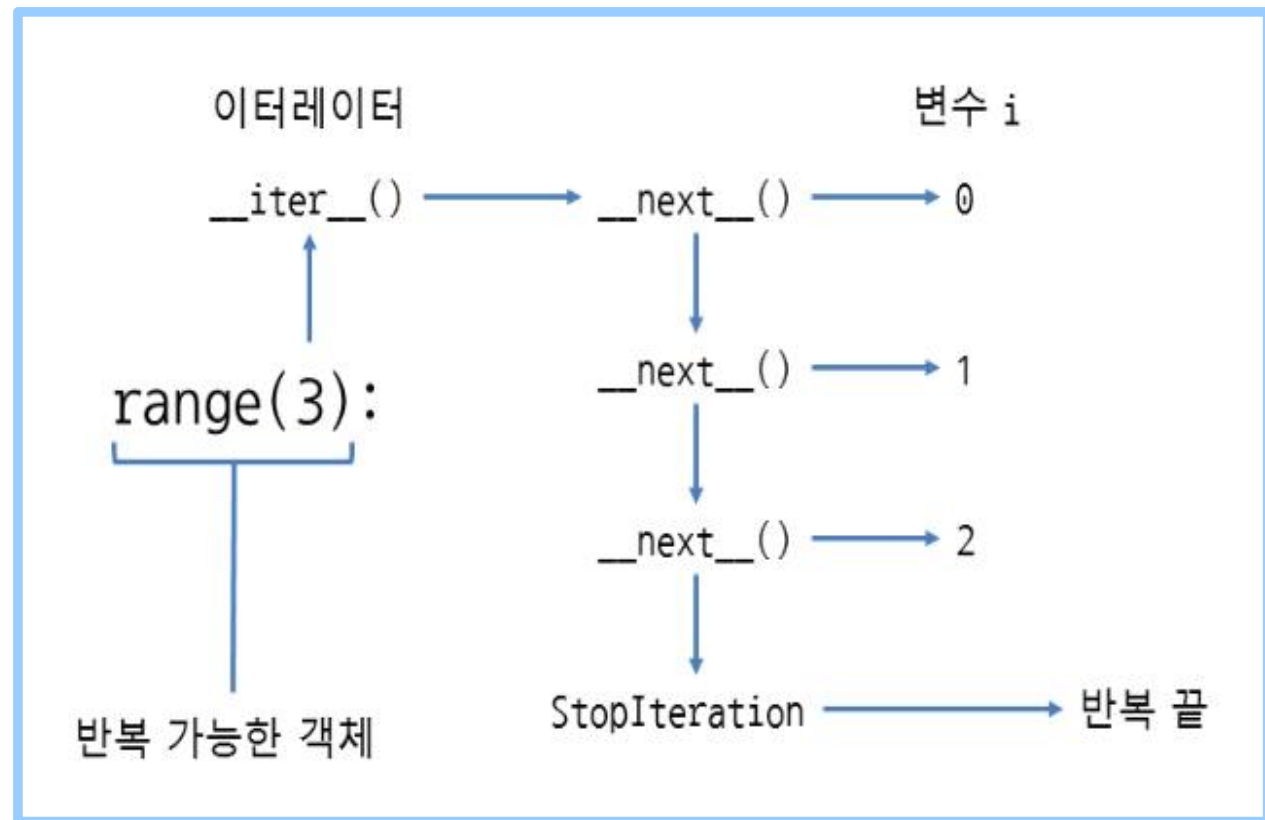
◆ 반복문 - iterable 객체

- **반복이 가능한 객체 의미**
- 타입 : 순서형 및 컬렉션 데이터 타입
 - str, list, tuple, dict, set, bytes, range
- 사용 : for ~ in 반복문
- 특징 : iterator 객체 생성 메서드 `__iter__()` 존재

PYTHON 반복문

◆ 반복문 - iterable 객체

- 동작원리



PYTHON 반복문

◆ 반복문 - Iterator 객체

- 데이터를 차례대로 하나씩 꺼낼 수 있는 객체
- iterable한 객체로 생성된 객체
- 반복자 패턴이라고 함
- 한 번 데이터 사용 후 폐기
- 구성
 - 반복을 위한 첫 번째 요소 가리키는 메서드
 - 다음 요소 가리키는 메서드
 - 마지막 요소 가리키는 메서드
 - 현재 요소 반환하는 메서드

PYTHON 반복문

◆ 반복문 - Iterator 객체

- 생성 메서드 → `iter(iterable)`
- 데이터 꺼내는 메서드 → `next(iterator)`

```
numList = [1, 2, 3]
numListiter = iter(numList)

print(type(numListiter), numListiter)

try:
    # 내장함수 사용
    print(next(numListiter))
    print(next(numListiter))
    print(next(numListiter))
    print(next(numListiter))
except StopIteration:
    print("End of Iterator")
finally:
    print("END")
```

PYTHON 반복문

◆ 반복문 - Iterator 객체

- 생성 메서드 → `seqobj.__iter__()`
- 데이터 꺼내는 메서드 → `seqobj.__next__()`

```
numList = [1, 2, 3]
numListiter = iter(numList)

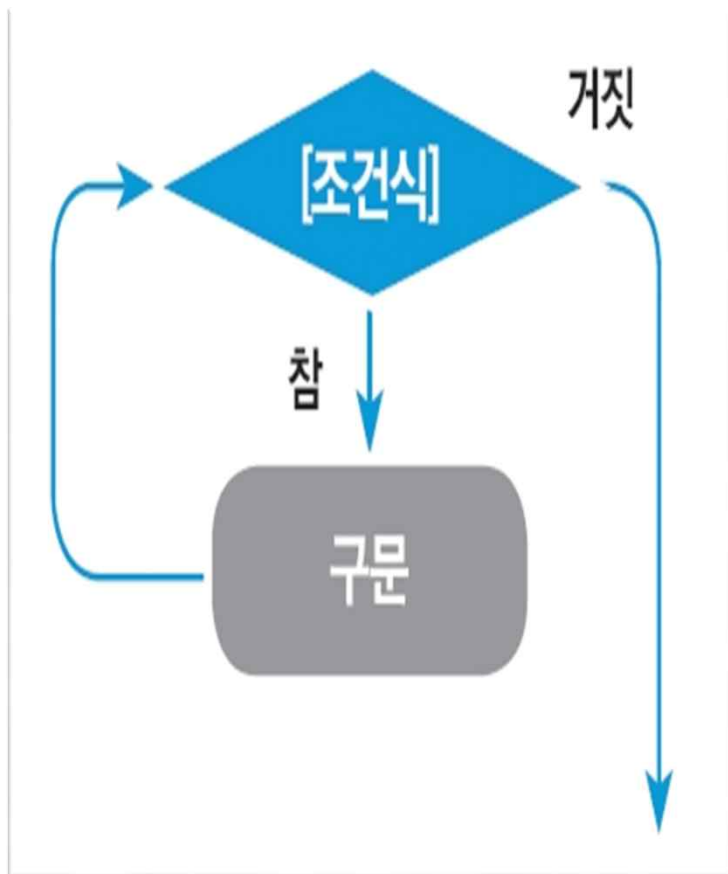
print(type(numListiter), numListiter)

try:
    # 메서드 사용
    print(numListiter.__next__())
    print(numListiter.__next__())
    print(numListiter.__next__())
    print(numListiter.__next__())
except StopIteration:
    print("End of Iterator")
finally:
    print("END")
```


PYTHON 반복문

◆ 반복문 - while

- 반복횟수가 명확하지 않는 경우 사용



초기값

while 조건식 :

실행문

초기값 갱신

반복 중단
위해 필수!

PYTHON 반복문

◆ 반복문 - while

```
num=20;                                # 초기값

while num>0:
    print(f"num is {num}")             # 반복
    num=num-1                          # 초기값 갱신
```

```
sum=0                                   # 초기값

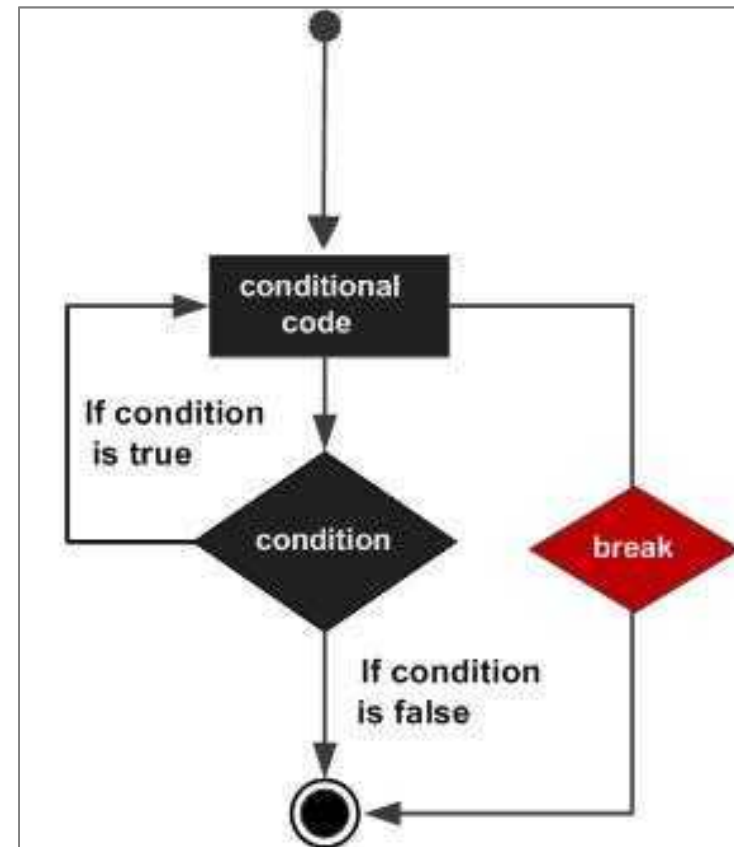
while sum<50:
    print(f"num is {sum}")
    sum=sum+5                           # 초기값 갱신
```

PYTHON 반복문

◆ 반복 중단

➤ break 문

- for와 while에서 반복 중단을 위해 사용



PYTHON 반복문

◆ 반복 중단

➤ break 문

```
for item in range(50):  
    if item > 17:  
        break  
    print(f"Item : {item}")
```

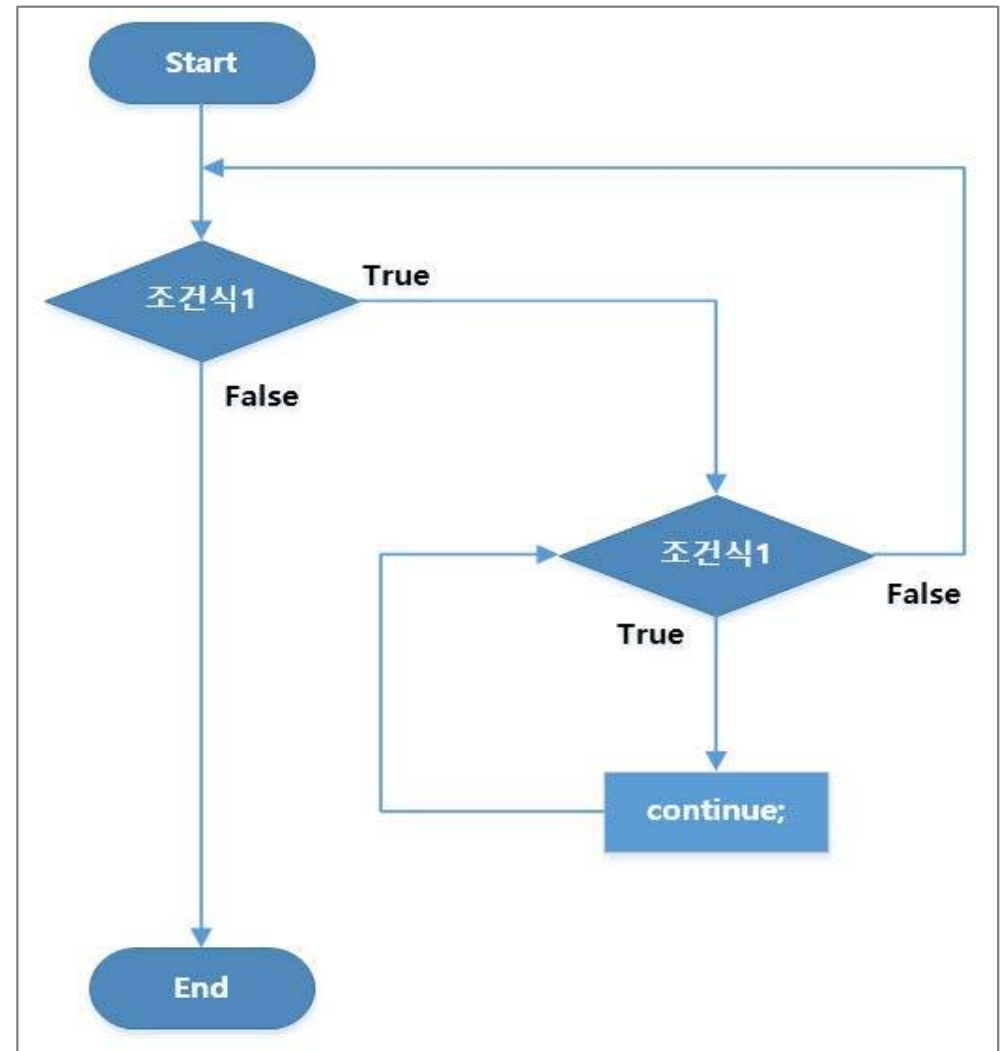
```
num=sum=0  
print(num, sum)  
while True:  
    if sum >= 33:  
        print(" === ")  
        break  
    sum = sum + num  
    num = num + 1  
print(" END WHILE")
```

PYTHON 반복문

◆ 반복 스킵

➤ continue 문

- 반복문에서 일부코드
- 실행하지 않고
- 다음으로 진행



PYTHON 반복문

◆ 반복 스킵

➤ continue 문

```
for value in range(50):  
  
    if value%2==0:  
        continue  
  
    print(f"value : {value}")
```

PYTHON 반복문

◆ 조건 무시

➤ pass 문

```
i = 0
while i < 10:
    i += 1

    if i % 2 == 0:
        continue

    print(i)
```

```
i = 0
while i < 10:
    i += 1

    if i % 2 == 0:
        pass

    print(i)
```

CH07. PYTHON 함수

PYTHON 함수

◆ 함수(Function)란?

반복되는 기능 코드를 하나로 묶어 이름을 붙인 것을 **함수**라 함
해당 기능 필요 시 **함수 이름을 불러서 사용** 가능

함수 호출

함수 호출이 되지 **않으면**

함수 코드는 **실행(동작)하지 않음**

PYTHON 함수

◆ 함수(Function) 선언

def 함수명(매개변수1, 매개변수2,..., 매개변수N):

→ 실행문

→ 실행문

→ **return** 반환값 ← 선택!

함 수 명 → 코드의 기능을 알 수 있도록 작성
make juice => make**J**uice**A**nd**S**ndwich

매개 변수 → 기능 구현에 필요한 **재료**로 **파라미터**라 함

반 환 값 → 기능 구현 후 **결과**물, 존재하지 않을 수도 있음

PYTHON 함수

◆ 함수(Function) 사용 → 함수 호출

함수명(매개변수1, 매개변수2)

Parameter

결과 저장 변수 = 함수명(인자1, 인자2)

Argument

함수 선언

```
def add(a, b):  
    value=a+b  
    return value
```

함수 호출(사용)

```
result=add(10, 20)  
print('add(10, 20) =', result)
```

PYTHON 함수

◆ return 반환

- 함수 종료 및 호출한 곳으로 돌아감
- 어떤 종류의 객체도 반환 가능
- return 사용하지 않거나 return만 적어도 함수 종료
- 함수 중간에 빠져나가기
- 여러 개 값 반환 가능

PYTHON 함수

◆ 함수 형태

- 인자와 반환값 존재하는 경우

```
def sum(x, y):  
    return x+y;  
  
print(30,20)  
  
print("Hello", "Tom")
```

PYTHON 함수

◆ 함수 형태

- 파라미터/매개변수만 존재하는 경우

```
def print_address(name):
```

```
    print("서울 특별시 종로구 1번지")
```

```
    print("파이썬 빌딩 7층")
```

```
    print(name)
```

```
print_address("홍길동")
```

PYTHON 함수

◆ 함수 형태

- 인자, 반환값 없는 경우

```
def print_address( ):
    print("서울 특별시 종로구 1번지")
    print("파이썬 빌딩 7층")
    print("홍길동")
```

```
print_address()    # 함수 호출
```

PYTHON 함수

◆ 함수 형태

- 인자, 반환값 모두 존재 하는 경우

```
def calculate_area (radius):
```

```
    area = 3.14 * radius**2
```

```
    return area
```

```
print( calculate_area (5.0) )
```


PYTHON 함수

◆ 함수 형태

➤ 인자 개수 미정인 경우

```
def 함수명( *매개변수 ):
```

```
    실행문
```

```
    실행문
```

```
    return 반환값
```

가변인자

PYTHON 함수

◆ 함수 형태

➤ 인자 개수 미정인 경우

```
def getSum(*args):  
    sum=0  
    for i in args:  
        sum+=i  
    return sum
```

가변인자
마지막에 위치

```
result=getSum(1,2,3,4,5)  
print("result %d" %result)
```

```
result=getSum(11,22,33,44,55,66,77,88,99)  
print("result %d" %result)
```

PYTHON 함수

◆ 함수 형태

➤ 키워드 파라미터 형태

0개~N개
키=값

```
def 함수명( **매개변수 ):
```

```
    실행문
```

```
    실행문
```

```
    return 반환값
```

key=value 형태
매개변수

딕셔너리
형태 반환

PYTHON 함수

◆ 함수 형태

➤ 키워드 파라미터 형태

```
def print_kwargs(**kwargs):  
    print(kwargs)
```

```
# 함수 호출
```

```
print_kwargs(name='kim', age=30, gender='F')
```

PYTHON 함수

◆ 함수 형태

➤ 매개변수 초기값 설정

```
def 함수명( 매개변수1, 매개변수2, 매개변수3=초기값):  
    실행문  
    실행문  
    return 반환값
```

맨마직 위치!!

PYTHON 함수

◆ 함수 형태

➤ 매개변수 초기값 설정

```
def set_info(name, old, man=True):  
    print("Name is %s" %name)  
    print("Old is %d" %old)  
  
    if man:  
        print("Man")  
    else:  
        print("Woman")  
  
    return  
  
set_info('Kim', 10)  
set_info('Tom Lee', 15, False)
```

PYTHON 함수

◆ 함수 형태

➤ 반환값이 여러 개 인 경우

```
def get_sum_mul(a, b):  
    return a+b, a*b
```

튜플 타입 리턴
언팩킹

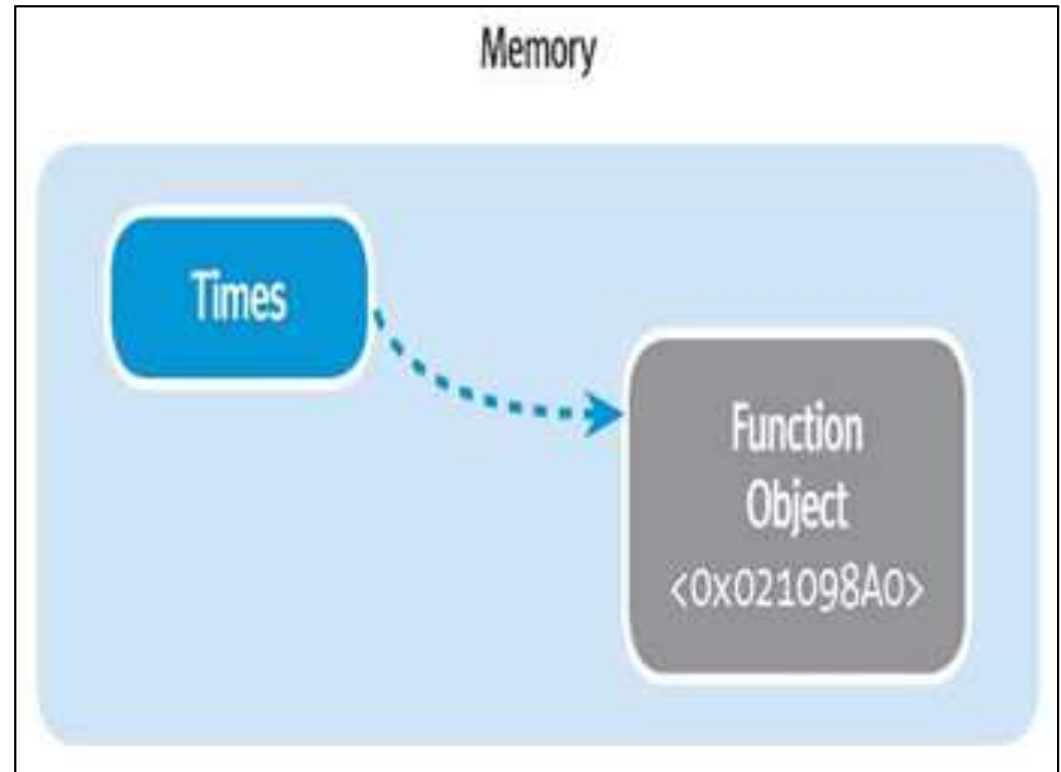
```
result=get_sum_mul(2, 10)  
print("get_sum_mul() result : " , result)
```

```
sum, mul =get_sum_mul(2, 10)  
print("get_sum_mul() result - sum %d, mul %d " %(sum, mul))
```

PYTHON 함수

◆ 변수에 함수 담기

- 메모리에 객체 생성
- 레퍼런스 생성



PYTHON 함수

◆ 변수에 함수 담기

- 함수도 class 'function' 타입으로 변수에 대입 가능

```
def something(a):  
    print(a)  
  
p=something  
print("type(p)=", type(p))  
  
p(123)  
p("Good Happy")
```

```
type(p)= <class 'function'>  
123  
Good Happy
```

PYTHON 함수

◆ 변수에 함수 담기

```
def plus(a,b):  
    return a*b
```

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

```
first=[plus, minus]  
print("type(first)", type(first))  
  
print("first[0](1,2)", first[0](1,2))  
print("first[1](1,2)", first[1](1,2))
```

PYTHON 함수

◆ 함수와 변수 관계

➤ 변수 위치에 따른 분류

지역변수 (Local Variable)

- 함수 안에 존재하는 함수
- 함수에서만 사용 가능
- 함수 실행 후 메모리에서 사라짐

전역변수 (Global Variable)

- 함수 밖에 존재하는 함수
- 프로그램 어디서나 사용 가능
- 프로그램 종료 시 메모리에서 사라짐

PYTHON 함수

◆ 함수와 변수 관계

test.py

```
a=10
```

전역변수

```
def test(a):
```

```
    a += 1
```

지역변수

```
    print('a =>', a)
```

```
# 함수 호출(사용)
```

```
test(20)
```

```
print("a is %d" %a)
```

PYTHON 함수

◆ 함수와 변수 관계

test.py

```
a=10                                # 전역변수

def test( ):
    # global a                      #전역변수 사용알림
    a += 1                          # 전역변수
    a = a + 1
# 함수 호출(사용)
test(a)
print("a is %d" %a)
```

PYTHON 함수

◆ 함수 안에 함수

```
# 함수 선언 -----  
  
def print_hello():  
    hello = 'Hello, world!'  
    def print_message():  
        print(hello)  
    print_message()  
  
# 함수 호출 -----  
  
print_hello()
```

PYTHON 함수

◆ Lambda(람다) 표현식

➤ 이름 없는 한 줄 함수 즉, 익명함수, 한줄 함수

변수명 = **lambda** 인수1, 인수2, ... : 실행코드 (리턴 값)

```
add=lambda x, y : x+y  
print( add(1,2) )
```

```
cal=[ lambda x,y:x+y, lambda x,y: x-y, lambda x,y:x/y ]  
print( cal[0](1,2) )  
print( cal[1](1,2) )  
print( cal[2](1,2) )
```



PART II

모듈 활용 프로그래밍

CH07. PYTHON 클래스

PYTHON 자료형

◆ 문자열 자료형

```
name = "Hello"  
<class 'str'>
```

```
name.upper()  
  .lower()  
  .islower()  
  .isNumeric()  
  :  
  :
```

str 전용
함수들
메소드

코드 영역

내장함수
클래스

힙 영역
데이터

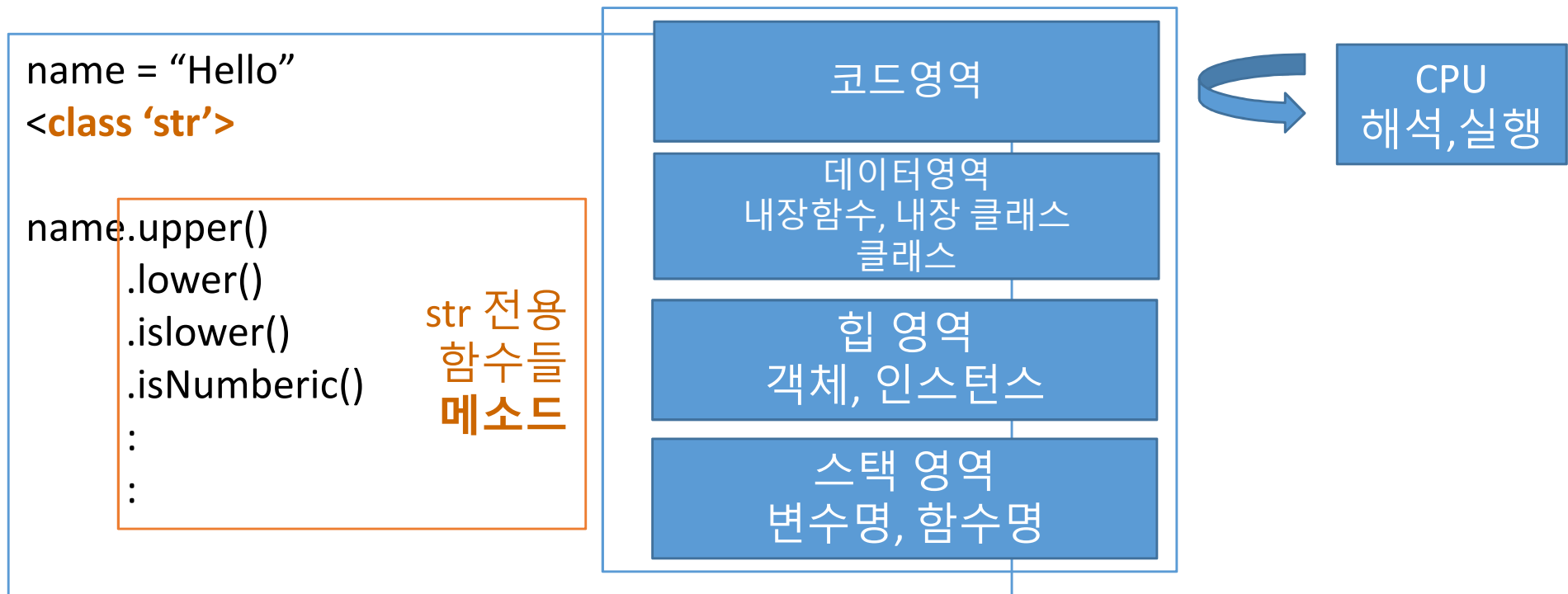
스택 영역
변수명,
함수명



CPU
해석, 실행

PYTHON 자료형

◆ 클래스(Class)



PYTHON 자료형

◆ 클래스(Class)

[자동차 중고회사]

자동차 관리 프로그램

- 자동차 데이터

- 차번호	12가1234
- 색상	흰색
- 종류	SUV
- 제조사	현대
- 문 수	4
- 연료	휘발류

- 전진	함수
- 후진	함수
- 좌회전	함수
- 우회전	함수
- 자율주행	함수

class car

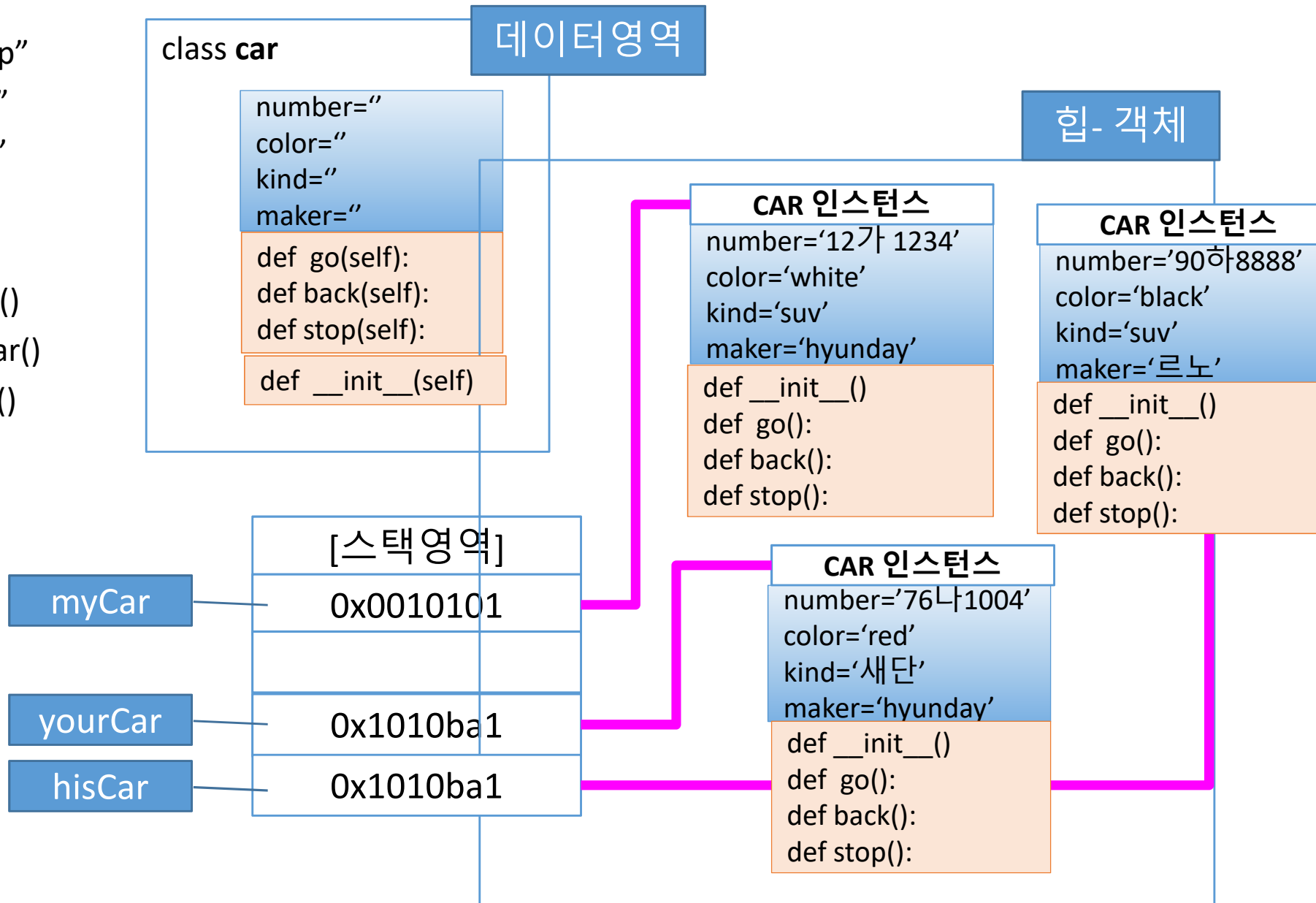
```
number=""  
color=""  
kind=""  
maker=""  
door = 0
```

```
def go():  
def back():  
def stop():
```

속성
Attribute,
property

메서드
Method

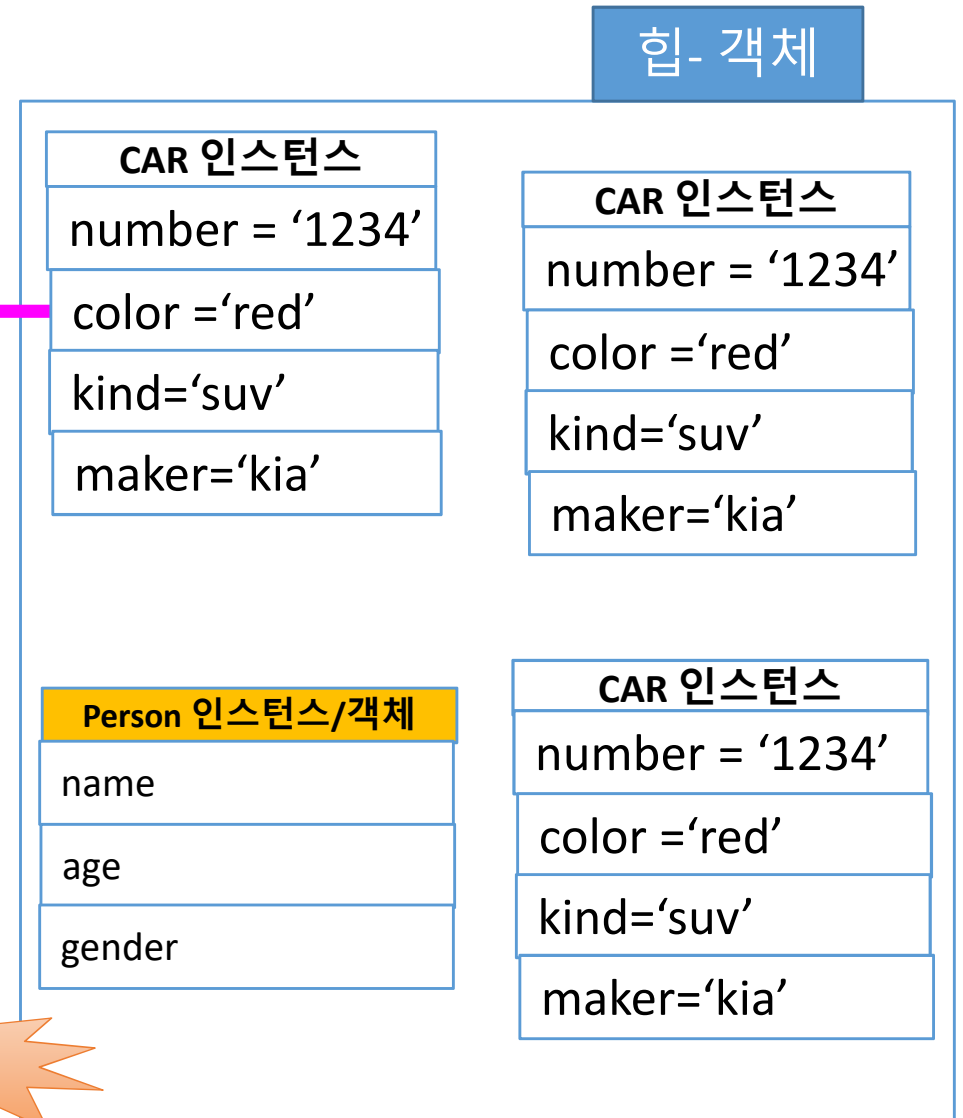
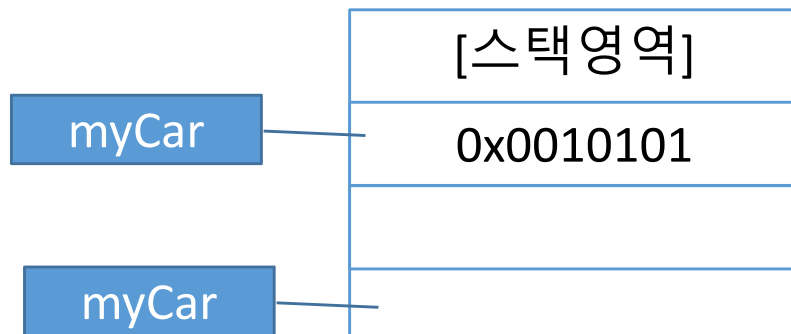
- name="Happ"
- msg="Good"
- year="2020"
- myCar = Car()
- yourCar = Car()
- hisCar = Car()
- myCar.go()
- yourCar.go()



- name="Happ"
- msg="Good"
- year="2020"



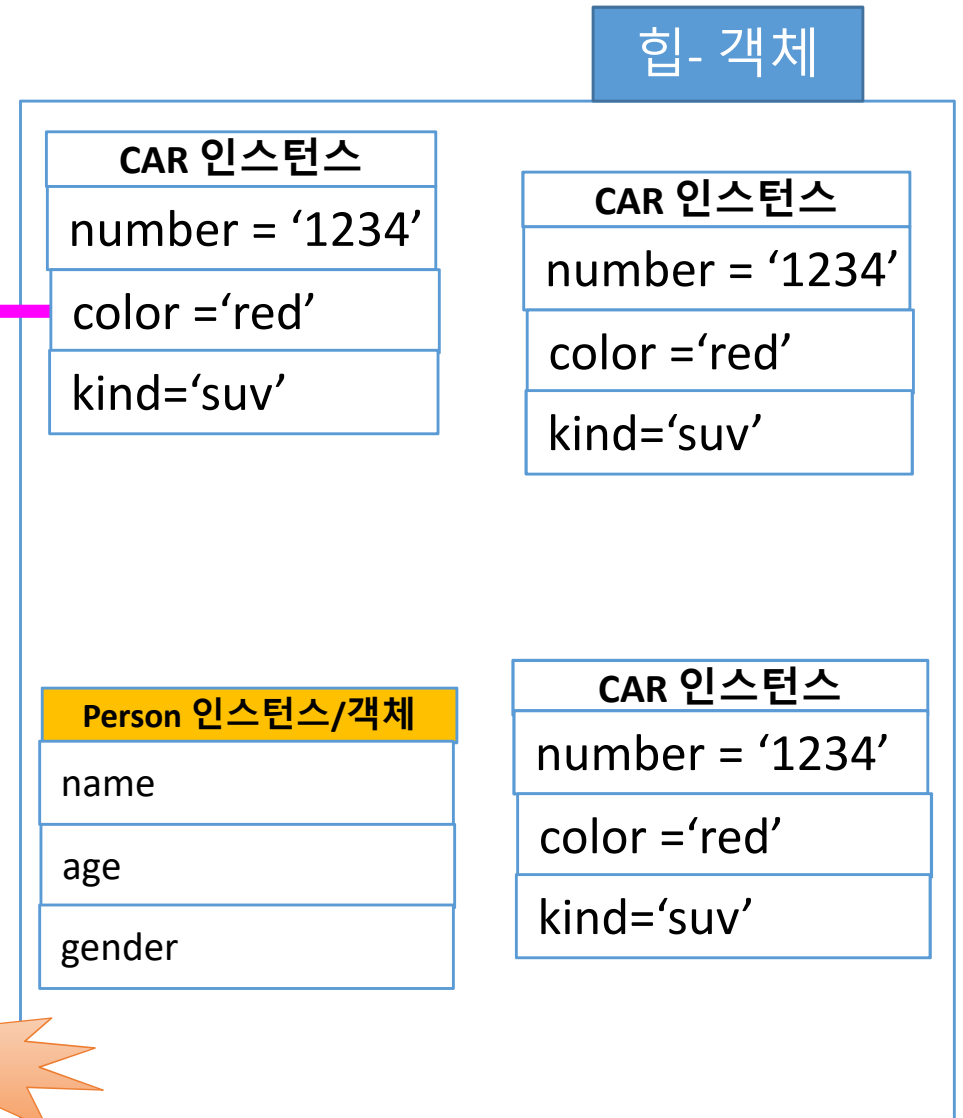
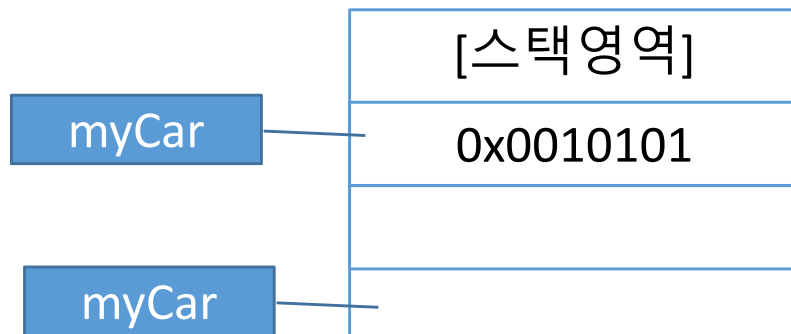
- myCar = Car('1234', 'red', 'suv', 'kia')



- name="Happ"
- msg="Good"
- year="2020"



- myCar = Car('1234', 'red', 'suv', 'kia')



class **Person**

name

age

gender

def eat(self):

def sleep (self):

def work(self):

def showInfo(self)

def **__init__**(self,

name, age,

gender)

- jenny = Person('jenny', 16, '여')

1

self

0x0010101

Person 인스턴스
0x0010101

List 인스턴스
0x001010ab

__hash__

2

List 인스턴스
0x0010195Fa

__hash__

Person 인스턴스
0x0010101

name = '박제니'

age = 12

gender = '여'

jenny

0x0010101

nums

0x001010ab

namesList

0x0010195Fa

객체 생성 후 속성 값 변경
jenny.age=12
jenny.name="박제니"

[가]

심리 감지 프로그램

- 사랑 데이터

- 예뻐진다
- 행복해진다
- 즐거워진다.

[파이썬]

```
class int
class float
class str
class bool
class list
class tuple
class dict
class set
```

class **love**

```
def go():
def back():
def leftGo():
def rightGo():
def autoDrive():
```

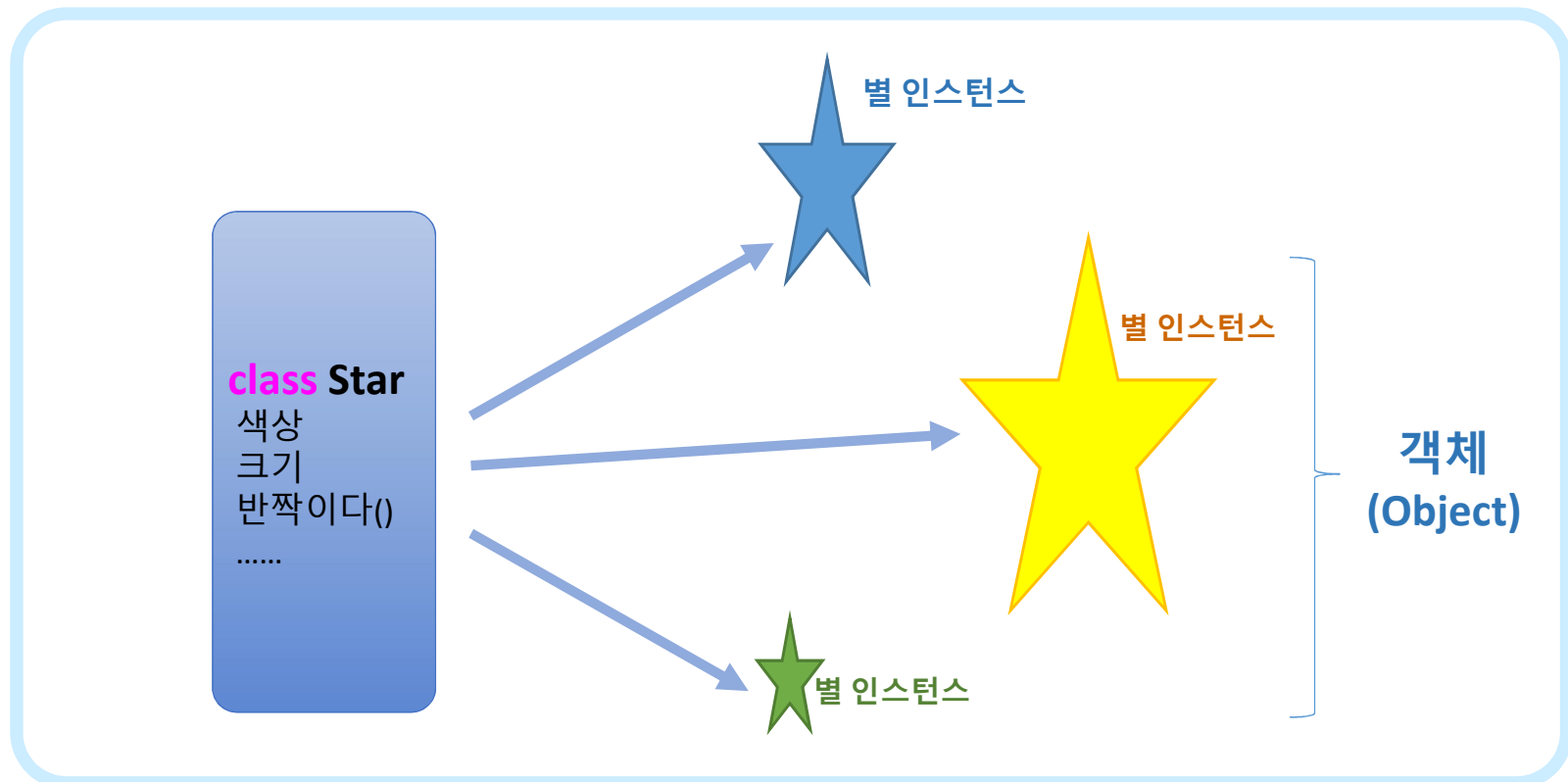
속성
Attribute,
property

메서드
Method

PYTHON 클래스

◆ 클래스(CLASS)

- 특정 기능을 하기 위한 **변수와 함수를 하나로 묶어 둔 Type**
- 제품의 **설계도**에 비유 / **틀**
- 설계도에 근거해서 만들어진 제품이 바로 **객체**
- 제품이 생성된 상세 클래스 정보를 **인스턴스**

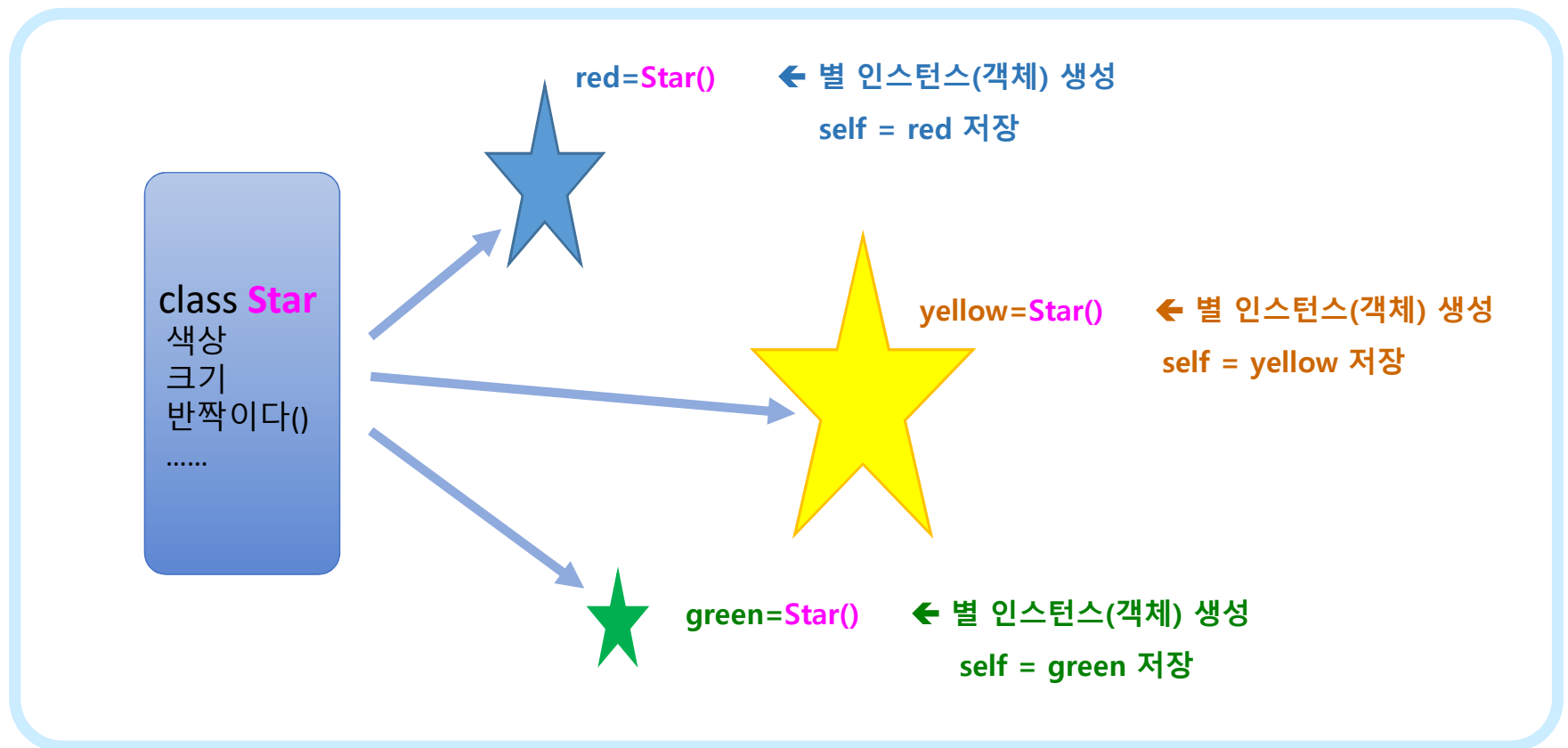


PYTHON 클래스

◆ 클래스(CLASS)

■ self 키워드

- 클래스로 객체를 생성한 경우 **인스턴스가 저장된 변수**



PYTHON 클래스

◆ 클래스(CLASS) 구성

class 클래스이름:

클래스 변수 1
:
클래스 변수 N

클래스 속성, 특징, 성질
객체에 **공유되는 변수**

객체 초기화 함수 생성자(Constructor)
def `_init_`(**self** [, 인수1, 인수2,...]):

인스턴스 생성 시 초기화 값
인스턴스 **마다 각자의 값** 저장

클래스 함수/메서드
def 함수명(**self** [, 인수1, 인수2,...]):

기능, 동작, 행동

PYTHON 클래스

◆ 클래스(CLASS) 구성

➤ 다양한 클래스 생성

```
class ProductNone:  
    pass
```

속성도 메서드도 없는 클래스

PYTHON 클래스

◆ 클래스(CLASS) 구성

➤ 다양한 클래스 생성

```
class Product:
```

```
    def displayInfo(self, name, price):  
        print(f"PNAME = {name}")  
        print(f"PRICE = {price}\n")
```

메서드만 존재하는 클래스

PYTHON 클래스

◆ 클래스(CLASS) 구성

➤ 다양한 클래스 생성

```
class Product:
```

필드(속성) 생성

```
def __init__(self, pname, price):
```

```
    self.pname=pname
```

```
    self.price=price
```

```
def displayInfo(self, name, price):
```

```
    print(f"PNAME = {self.pname}")
```

```
    print(f"PRICE = {self.price}\n")
```

필드, 메서드
모두 존재하는 클래스

특pecially 생성자 메서드 라 함

PYTHON 클래스

◆ 클래스(CLASS) 구성

➤ 객체(인스턴스) 생성

객체변수명 = 클래스명()

```
class ProductNone:  
    pass
```

생성

```
p1 = ProductNone()  
p2 = ProductNone()
```


PYTHON 클래스

◆ 클래스(CLASS) 구성

➤ 객체(인스턴스) 속성 & 메서드 사용

속성값 변경 : 객체변수명.속성명 = 값

속성값 읽기 : 객체변수명.속성명

메서드 호출 : 객체변수명.메서드명()

PYTHON 클래스

◆ 클래스(CLASS) 구성

➤ 객체(인스턴스) 속성 & 메서드 사용

```
# -----  
# 객체(인스턴스) 생성  
# 방법(규칙) : 변수명 = 클래스명()  
# -----  
  
p1=Product('Cake', 'Home', 10000)  
p2=Product('Bag', 'PARK', 5000)  
  
# -----  
# 객체(인스턴스)의 메서드 또는 속성(필드) 사용  
# 객체(인스턴스)변수명.메서드()  
# -----  
  
p1.displayInfo()           # 객체( Product 인스턴스 ) 메서드 사용
```

PYTHON 클래스

◆ 클래스(CLASS) 구성

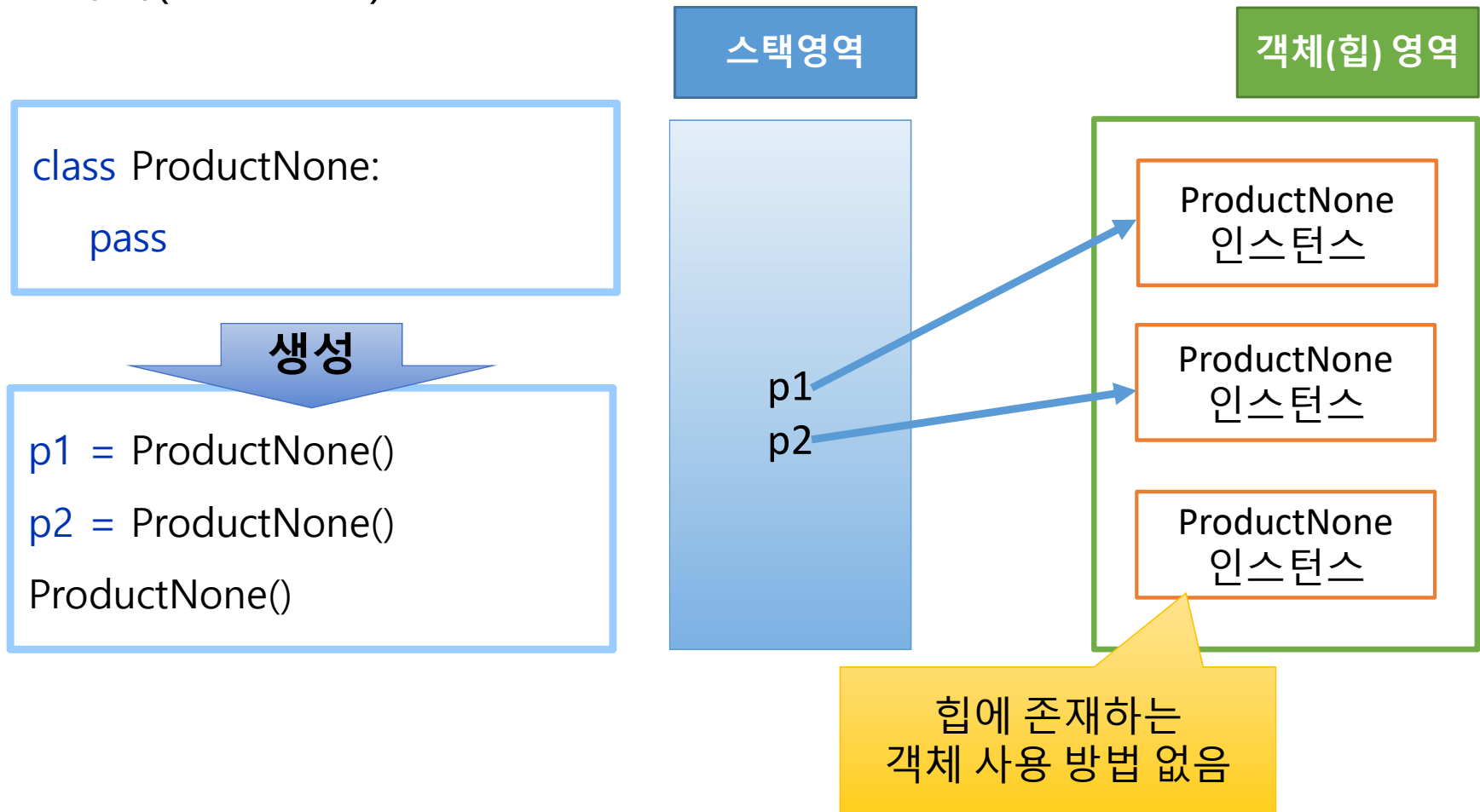
➤ 객체(인스턴스) 속성 & 메서드 사용

```
# -----  
# 객체(인스턴스)의 메서드 또는 속성(필드) 사용  
# 객체(인스턴스)변수명.속성명  
# -----  
  
print(f"p1.pmaker =>{p1.pmaker}")    # 객체( Product 인스턴스 ) 필드 값 사용  
print(f"p1.price =>{p1.price}")      # 객체( Product 인스턴스 ) 필드 값 사용  
  
p1.price=25000                        # 객체( Product 인스턴스 ) 필드 값 변경  
p1.pmaker='Pari'                      # 객체( Product 인스턴스 ) 필드 값 변경  
  
print(f"p1.pmaker =>{p1.pmaker}")    # 객체( Product 인스턴스 ) 필드 값 사용  
print(f"p1.price =>{p1.price}")      # 객체( Product 인스턴스 ) 필드 값 사용
```

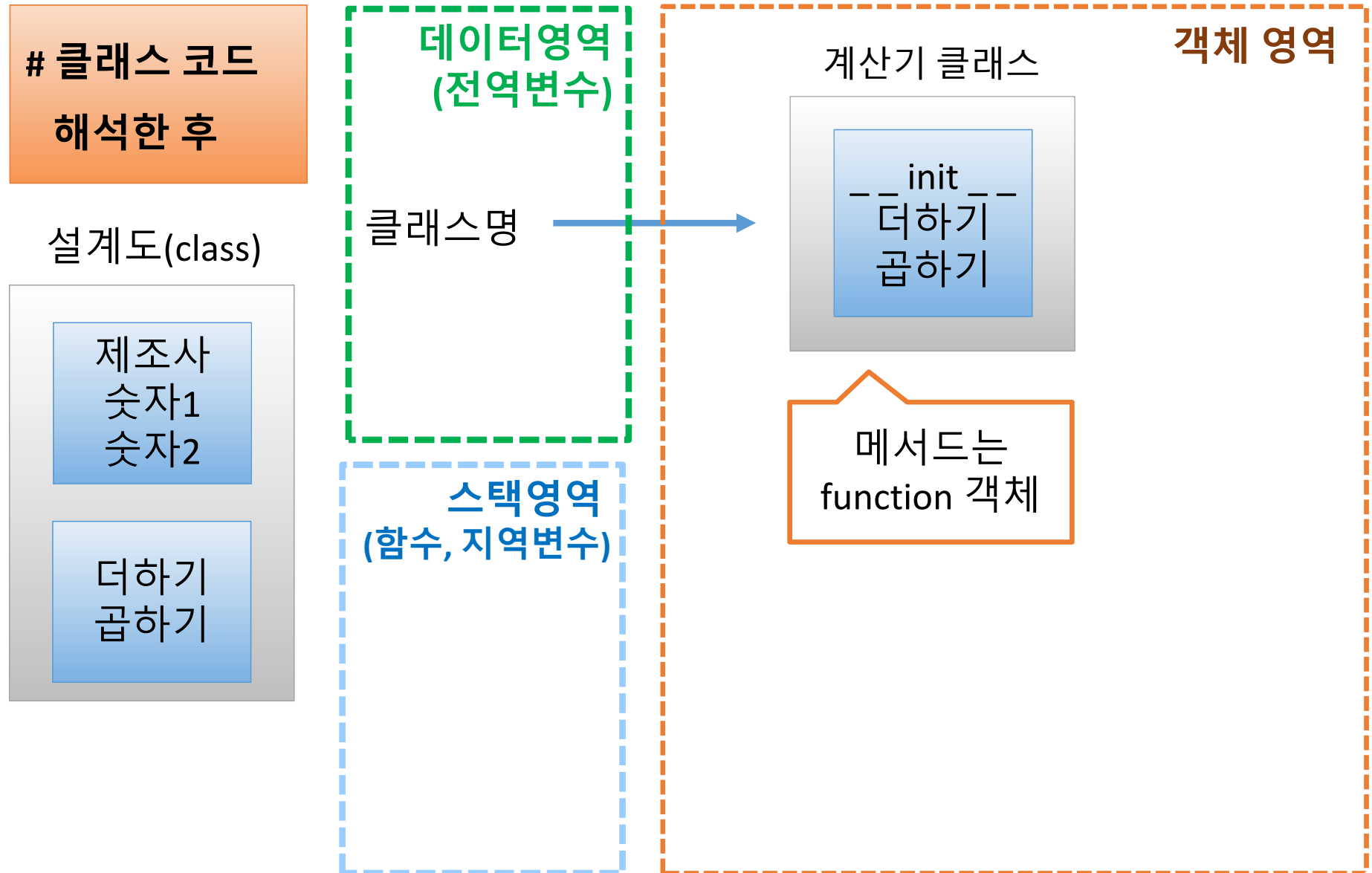
PYTHON 클래스

◆ 클래스(CLASS) 구성

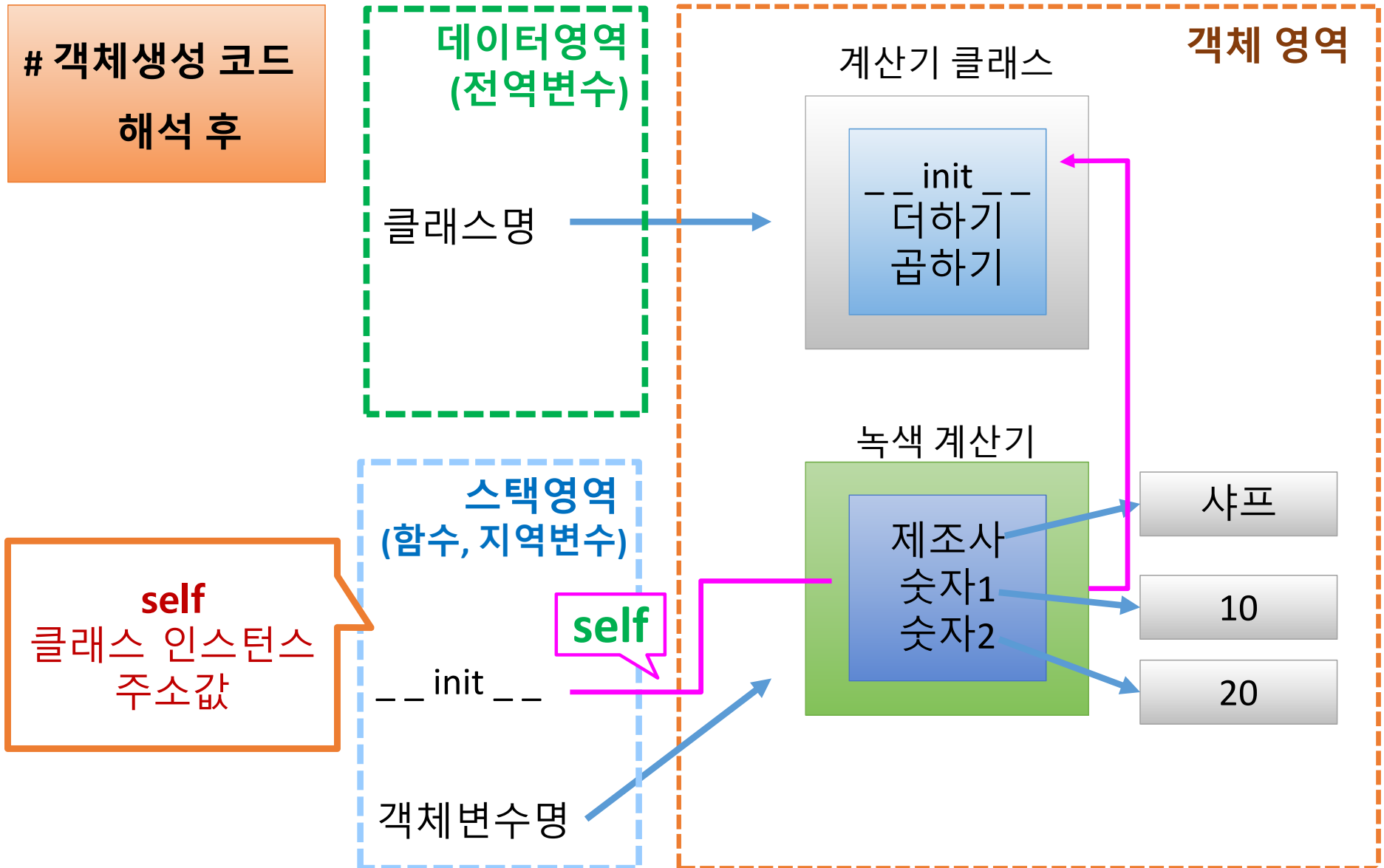
➤ 객체(인스턴스) 생성



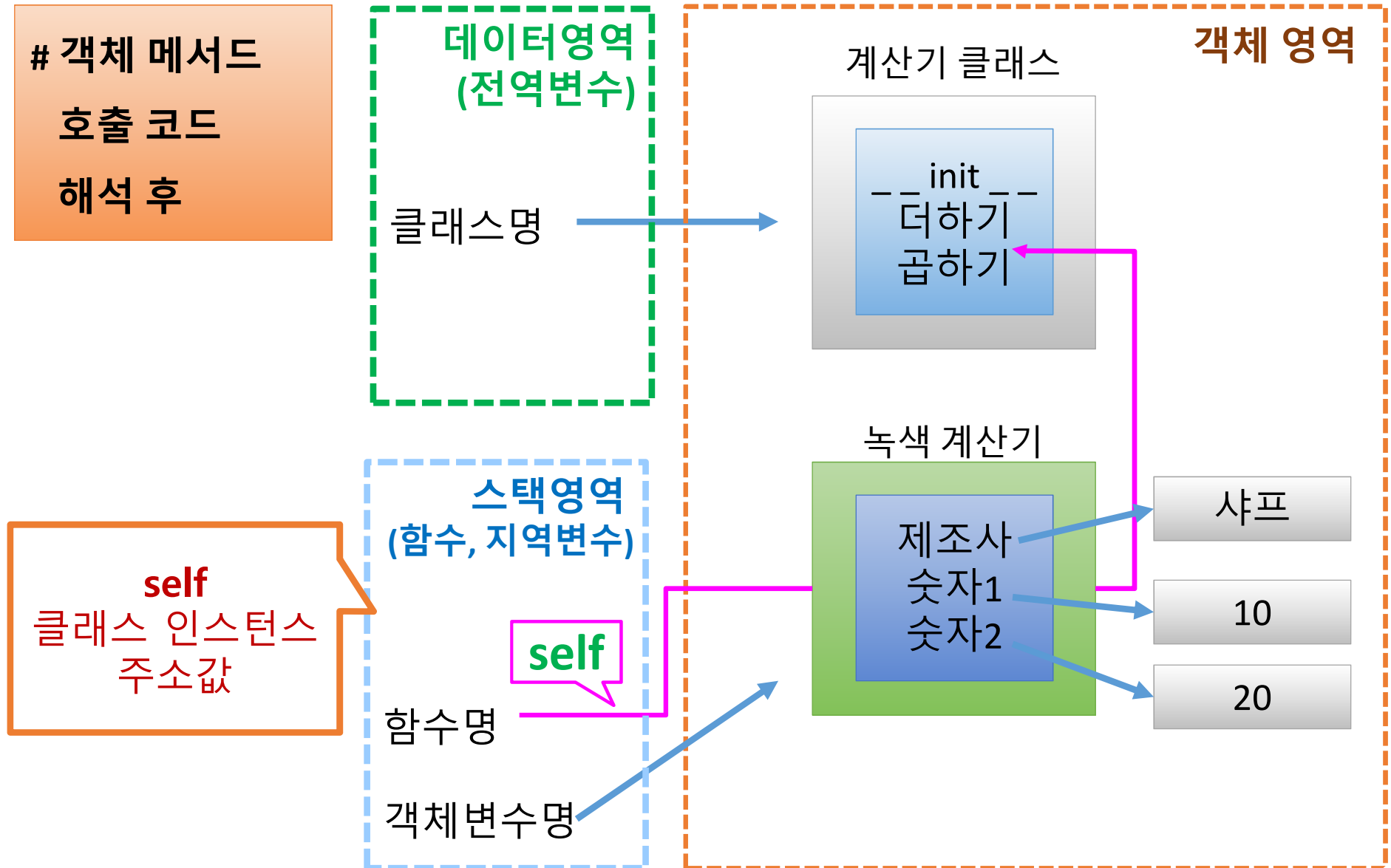
PYTHON 클래스



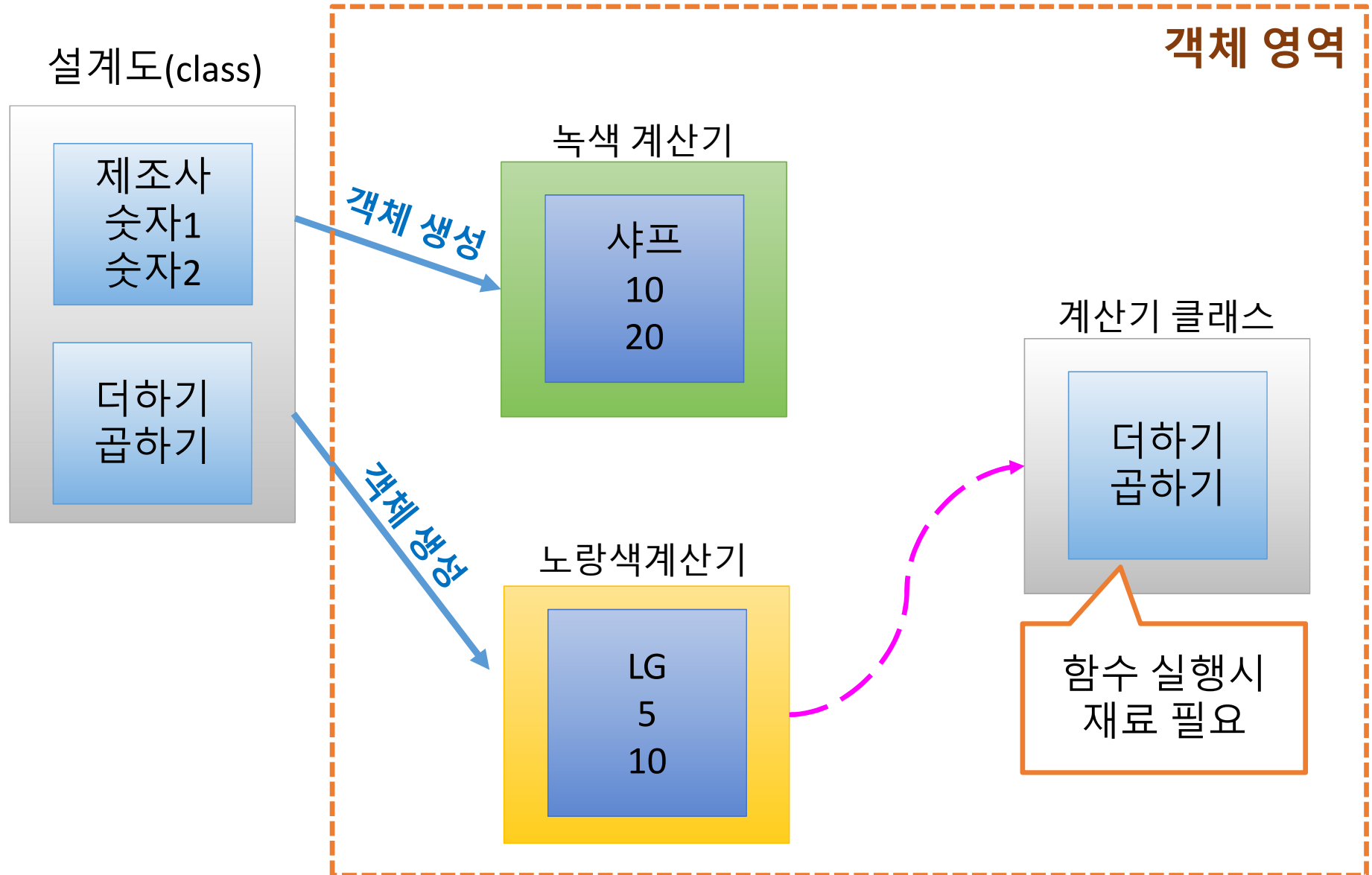
PYTHON 클래스



PYTHON 클래스



PYTHON 클래스



PYTHON 클래스

◆ 클래스(CLASS) 구성

➤ 속성 & 메서드 확인

인스턴스명.__dict__

클래스명.__dict__

print(f'p1.__dict__ : {p1.__dict__}')

print(f'Product.__dict__ : {Product.__dict__}')

p1.__dict__ : {'pname': 'Cake', 'pmaker': 'Home', 'price': 10000}

Product.__dict__ : {'__module__': '__main__', '__init__': <function Product.__init__ at 0x000001F3C58B2DC8>, 'displayInfo': <function Product.displayInfo at 0x000001F3C58B2D38>, '__dict__': <attribute '__dict__' of 'Product' objects>, '__weakref__': <attribute '__weakref__' of 'Product' objects>, '__doc__': None}

PYTHON 클래스

◆ 클래스(CLASS) 구성

생성자(Constructor)

- 객체(인스턴스) 생성 시 호출
- 인스턴스 변수 초기화
- `def __init__(self)`

소멸자(Destructor)

- 객체(인스턴스) 생성 시 호출
- 인스턴스 변수 초기화
- `def __del__(self)`

PYTHON 클래스

◆ 클래스(CLASS) 구성

➤ 스페셜 메서드 / 매직 메서드

파이썬 시스템에서 **자동 호출되는 메서드**

형태 : `__메서드명__()`

PYTHON 클래스

◆ 클래스(CLASS) 구성

```
class CC:  
    name="계산기"
```

```
    def add(self, first, second):  
        return first + second
```

```
    def sub(self, first, second):  
        return first - second
```

```
class CCC:  
    name = "계산기"
```

```
    def __init__(self, first, second):  
        self.first=first  
        self.second=second
```

```
    def add(self):  
        return self.first + self.second
```

```
    def sub(self):  
        return self.first - self.second
```

```
c=CC()  
print( "{0} + {1} = {2}".format(10, 20, c.add(10, 20)))  
print( "{0} - {1} = {2}".format(10, 20, c.sub(10, 20)))
```

```
ccc=CCC(1000, 3000)  
print( "{0} + {1} = {2}".format(ccc.first, ccc.second, ccc.add()))  
print( "{0} - {1} = {2}".format(ccc.first, ccc.second, ccc.sub()))
```

PYTHON 클래스

◆ 클래스(CLASS)

➤ 변수 종류

인스턴스 변수

- 인스턴스 마다 존재하는 변수
- 객체변수명으로 읽기 & 변경

비공개 변수 : **__변수명**

- 객체변수명으로 보이지 않음
- 클래스 내에서만 사용 가능

```
class CCC:
```

```
def __init__(self, datas, age):
```

```
    self.data=datas
```

```
    self.__age=age
```

```
def getsum(self):
```

```
    sum=0
```

```
    for i in range(0,len(self.data)):
```

```
        sum += self.data[i]
```

```
    return sum
```

PYTHON 클래스

◆ 클래스(CLASS)

➤ 변수 종류

클래스 변수

- 인스턴스 공유하는 변수
- 접근 : 클래스명.변수명

비공개 변수 : **__변수명**

- 클래스명으로 보이지 않음
- 클래스 내부에서만 사용 가능

```
class CCC:
```

```
    __share = 1000
```

```
    def __init__(self, datas, age):
```

```
        self.data=datas
```

```
        self.__age=age
```

```
    def getsum(self):
```

```
        sum=0
```

```
        for i in range(0,len(self.data)):
```

```
            sum += self.data[i]
```

```
        return sum
```

PYTHON 클래스

◆ 클래스(CLASS)

➤ 오버로딩(overloading)

- 함수이름 동일
- 매개변수 개수, 타입, 순서가 다른 함수 정의

PYTHON 클래스

◆ 클래스(CLASS)

➤ 연산자 오버로딩(overloading)

- 객체에서 연산자를 클래스 목적에 맞게 기능 부여 사용
- 함수이름 앞뒤에 언더스코어(_) 두개 연속으로 붙은 함수

형태 : `def __함수이름__()`:

매직함수명	연산자
<code>def __add__(self, other)</code>	<code>+</code>
<code>def __sub__(self, other)</code>	<code>-</code>
<code>def __mul__(self, other)</code>	<code>*</code>
<code>def __truediv__(self, other)</code>	<code>/</code>
<code>def __floordiv__(self, other)</code>	<code>//</code>
<code>def __mod__(self, other)</code>	<code>%</code>
<code>def __pow__(self, other)</code>	<code>**</code>

PYTHON 클래스

◆ 클래스(CLASS)

➤ 연산자 오버로딩

```
# 클래스 생성 -----  
class A:  
    def __init__(self, num):  
        self.num=num
```

```
# 인스턴스 생성 -----  
a=A(10)  
b=A(20)
```

```
# + 연산 및 출력 -----  
print(a+b)
```

Traceback (most recent call last):

File "C:/PyChamProject/DAY03/src/EX_Class.py", line 103, in <module>
 print(a+b)

TypeError: unsupported operand type(s) for +: 'A' and 'A'

PYTHON 클래스

◆ 클래스(CLASS)

➤ 연산자 오버로딩

```
# 클래스 생성 -----  
class A:  
    def __init__(self, num):  
        self.num=num  
  
    def __add__(self, other):  
        return self.num+other.num  
  
# 인스턴스 생성 -----  
a=A(10)  
b=A(20)  
  
# + 연산 및 출력 -----  
print(a+b)
```

PYTHON 클래스

◆ 클래스(CLASS)

➤ Setter & Getter 메서드

- 변수의 값 설정 ➔ setter method
- 변수의 값 읽기 ➔ getter method

PYTHON 클래스

◆ 상속(inheritance)

- 클래스 확대 및 **기존 클래스 재사용 & 기능 확장**
- 부모 클래스가 가진 것 모두 자식 클래스에서 사용
- 부모 클래스로부터 상속받은 함수를 **재정의** 가능
 - **오버라이딩(Overriding)**

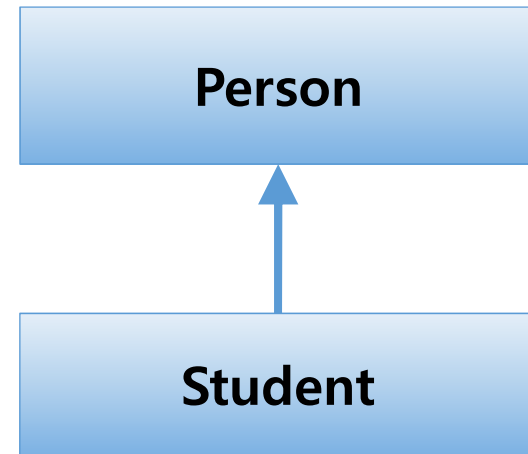
형식 → **class** 자식클래스명 (부모클래스명)

PYTHON 클래스

◆ 상속(inheritance)

```
class Person:  
    pass
```

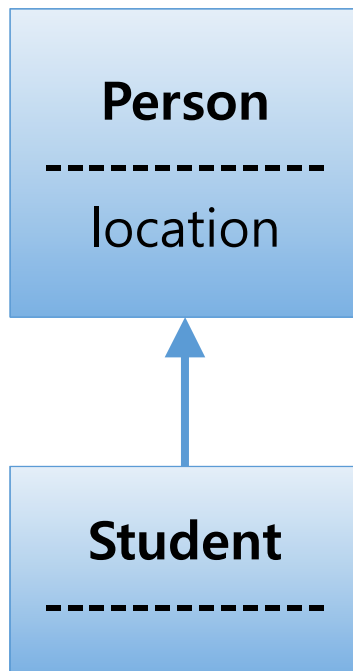
```
class Student(Person):  
    pass
```



```
# 상속 관계 확인하기 -----  
print(f"Student는 Person의 자식? {issubclass(Student, Person)}")
```

PYTHON 클래스

◆ 상속(inheritance)



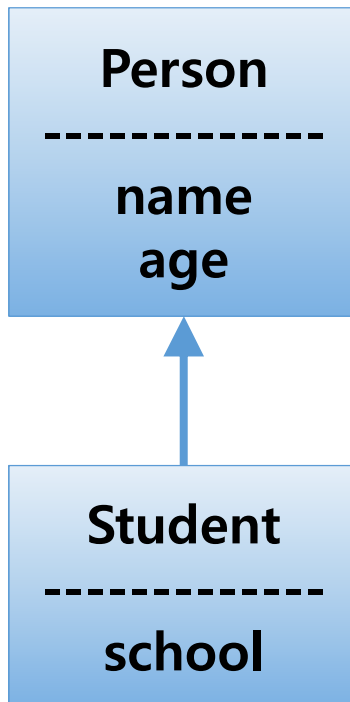
```
class Person:
    def __init__(self):
        self.location='Korea'

class Student(Person):

    def showInfo(self):
        print(f"Location : {self.location}")
```

PYTHON 클래스

◆ 상속(inheritance)



```
class Person:
    def __init__(self, name, age):
        print("Person __init__() ")
        self.name=name
        self.age=age

class Student(Person):
    def __init__(self, name, age, school):
        print("Student __init__() ")
        #self.name = name
        #self.age = age
        #Person.name=name
        #Person.age=age
        #Person.__init__(self,name,age)
        super().__init__(name, age)
        self.school=school

    def showInfo(self):
        print(f"name : { self.name}Wtschool : {self.school}")
```

PYTHON 클래스

◆ 상속(inheritance)

➤ 오버라이딩(Overring)

- 함수 구현 부분만 다시 **재정의**
- 상속관계에서 **부모에서 상속 받은 메소드**에 한정

PYTHON 클래스

◆ 상속(inheritance)

```
# 클래스 생성 -----  
class Point:  
    def __init__(self, x, y):  
        self.x=x  
        self.y=y  
  
    def show_point(self):  
        return "{0}, {1}".format(self.x, self.y)
```

```
# 클래스 생성 -----  
class DPoint(Point):  
    def __init__(self, x, y, z):  
        #self.x=x  
        #self.y=y  
        super().__init__(x, y):  
        self.z=z  
  
    def show_point(self):  
        return "{0}, {1}, {2}".format(self.x, self.y, self.z)
```

오버라이딩

PYTHON 클래스

◆ Object 클래스

- 모든 클래스의 부모 클래스
- 자동으로 상속받게 되는 클래스

<code>__class__</code>	<code>object</code>
<code>__str__(self)</code>	<code>object</code>
<code>__annotations__</code>	<code>object</code>
<code>__delattr__(self, name)</code>	<code>object</code>
<code>__dir__(self)</code>	<code>object</code>
<code>__eq__(self, o)</code>	<code>object</code>
<code>__format__(self, format_spec)</code>	<code>object</code>
<code>__getattr__(self, name)</code>	<code>object</code>
<code>__hash__(self)</code>	<code>object</code>
<code>__init_subclass__(cls)</code>	<code>object</code>
<code>__ne__(self, o)</code>	<code>object</code>
<code>__new__(cls)</code>	<code>object</code>

CH07. PYTHON 모듈 & 패키지

PYTHON 모듈 & 패키지

◆ 모듈(Module)

- 기능 수행 위해 **변수, 함수, 클래스를 모아둔 파일**
- python에 **다양한 기능을 확장**시켜 줄 수 있는 것
- **다른 Python 프로그램에서 불러서 사용**가능하게 만든 것

분류	설명
표준 모듈	파이썬에서 기본 제공 모듈
사용자 생성 모듈	개발자가 만든 모듈
써드 파티션 모듈	다른 사람들이 만들어서 공개하는 모듈

PYTHON 모듈 & 패키지

◆ 모듈(Module) 사용

➤ 전체 사용

import 모듈명

<= 모듈 내의 변수, 함수 호출 사용

import 모듈명 **as** 별칭

<= 모듈 내의 변수, 함수 호출 사용

```
import math
```

```
print(" 원주율 = ", math.pi)  
print(" 2의 3승 = ", math.pow(2,3) )  
print(" 3! = ", math.factorial(3) )
```

```
import math as m
```

```
print(" 원주율 = ", m.pi)  
print(" 2의 3승 = ", m.pow(2,3) )  
print(" 3! = ", m.factorial(3) )
```

PYTHON 모듈 & 패키지

◆ 모듈(Module) 사용

➤ 모듈 일부만 사용

from 모듈명 **import** 함수명1, 함수명2 <= 특정 함수만 사용

```
from math import pow
print("제곱 = ", pow(2,3))
```

```
from math import pow as p
print(" 원주율 = ", p(2,3))
```

```
from math import pow, pi
print(" 제곱 = ", pow(2,3))
print(" 원주율 = ", pi )
```

```
from math import *
print(" 제곱 = ", pow(2,3))
print(" 원주율 = ", pi )
```

PYTHON 모듈 & 패키지

◆ 모듈(Module) & 스크립트(Script)

➤ 2가지 동작 모드 제어

`__name__` <= 현재 실행 모듈 이름 저장하고 있는 내장 변수

- 현재 실행 중일 경우 변수 저장 값 : `__main__`
- 다른 파일에 포함된 경우 변수 저장 값 : **파일명**

`__name__`.py <= 패키지의 경우 동일 효과

PYTHON 모듈 & 패키지

◆ 모듈(Module) & 스크립트(Script)

➤ 2가지 동작 모드 제어

```
print( __name__ )

if __name__ == "__main__":
    print("나는 현재 실행 중입니다.")
else:
    print("나의 이름은 {0}입니다.".format(__name__))
```


PYTHON 모듈 & 패키지

◆ 모듈(Module) & 스크립트(Script)

➤ 2가지 동작 모드 제어

```
def get_sum(a, b):  
    return a+b
```

```
def main():  
    data_list=[[1,1], [2,2], [3,3], [4,4]]  
    sum =0  
    for i in range(0,len(data_list)):  
        sum += get_sum(data_list[i][0], data_list[i][1])  
  
    print("sum = %d" %sum)
```

PYTHON 모듈 & 패키지

◆ 모듈(Module) & 스크립트(Script)

➤ 2가지 동작 모드 제어

```
if __name__ == "__main__":  
    print("나는 현재 실행 중입니다.")  
    print("aaa.py 시작합니다.")  
    main()  
  
else:  
    print("나는 {0}입니다.".format(__name__))
```

PYTHON 모듈 & 패키지

◆ 패키지(Package)

- 특정 기능과 관련된 여러 모듈을 묶은 것
- python에 다양한 기능을 확장시켜 줄 수 있는 것
- 다른 Python 프로그램에서 불러서 사용가능하게 만든 것
- 설치 작업이 추가로 필요한 경우도 있음

- 파이썬 패키지 인덱스(Python Package Index, PyPI)
- www.pypi.org

PYTHON 모듈 & 패키지

◆ 패키지(Package)

- 파이썬 패키지 인덱스(Python Package Index, PyPI)

www.pypi.org



PYTHON 모듈

◆ 패키지(Package)

➤ 전체 사용

```
import 패키지명.모듈명
```

```
import 패키지명.모듈명1, 모듈명2
```

```
import 패키지명.모듈명 as 별칭
```

```
import urllib.request
```

```
import urllib.request as r
```

PYTHON 모듈

◆ 패키지(Package)

➤ 전체 사용

```
import urllib                # urllib 전체 패키지

req = urllib.request.Request('http://www.google.co.kr')
response = urllib.request.urlopen(req)
```

```
import urllib.request as request    # urllib패키지 request 모듈

req = request.Request('http://www.google.co.kr')
response = request.urlopen(req)
```

PYTHON 모듈

◆ 패키지(Package)

➤ 일부만 사용

from 패키지명.모듈명 **import** 변수명

from 패키지명.모듈명 **import** 함수명

from 패키지명.모듈명 **import** 클래스명

클래스, 함수만 사용

from urllib.request **import** Request, urlopen

PYTHON 모듈

◆ 패키지(Package)

➤ 일부만 사용

```
from urllib.request import Request, urlopen      # urlopen 함수, Request 클래스 가져옴  
  
req = Request('http://www.google.co.kr')        # Request 클래스를 사용하여 req 생성  
response = urlopen(req)                         # urlopen 함수 사용
```

```
from urllib.request import *                    # urllib의 request 모듈의 모든 변수, 함수, 클래스  
  
req = Request('http://www.google.co.kr')  
response = urlopen(req)
```


PYTHON 모듈

◆ 패키지(Package)

➤ 설치 명령어

`pip install 패키지명`

➤ 버전 2가지 존재

`pip --version`

Terminal: Local × +

(c) 2020 Microsoft Corporation. All rights reserved.

(EV_PY37) D:\BEGINNER_AI_2ND\EXAM_PY>pip --version

pip 20.2.2 from C:\Users\anece\anaconda3\envs\EV_PY37\lib\site-packages\pip (python 3.7)

(EV_PY37) D:\BEGINNER_AI_2ND\EXAM_PY>

PYTHON 모듈

◆ 패키지(Package)

➤ 명령어 사용법 확인

`pip --help`

```
Terminal: Local x +  
  
(EV_PY37) D:\BEGINNER_AI_2ND\EXAM_PY>pip --help  
  
Usage:  
  pip <command> [options]  
  
Commands:  
  install          Install packages.  
  download         Download packages.  
  uninstall        Uninstall packages.  
  freeze           Output installed packages in requirements  
  list             List installed packages.  
  show            Show information about installed packages.  
  check           Verify installed packages have compatible
```

PYTHON 모듈

◆ 패키지(Package)

➤ pip 명령어 옵션

- | | |
|------------------------------|--|
| • pip search 패키지 | 패키지 검색 |
| • pip install 패키지==버전 | 특정 버전 패키지 설치
(예: pip install requests==2.9.0) |
| • pip list | 패키지 목록 출력 |
| • pip freeze | 패키지 목록 출력 |
| • pip uninstall 패키지 | 패키지 삭제 |

CH08. PYTHON 예외처리

PYTHON 예외처리

◆ 예외처리란?

- 프로그램 오류 발생 시 종단을 막기 위해 처리해주는 방법

```
num1=10  
num2=0
```

```
print(f'{num1}/{num2} = {num1/num2}')
```

ex_exception_01 ×

```
C:\Users\anece\anaconda3\envs\EV_PY37\python
```

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

```
File "D:/BEGINNER_AI_2ND/EXAM_PY/EXAM_EXCE
```

```
    print(f'{num1}/{num2} = {num1/num2}')
```

```
ZeroDivisionError: division by zero
```

PYTHON 예외처리

◆ 예외처리

- 예시

```
file = open("../Data/address.txt", 'r')  
print(file.read())  
file.close()
```

Traceback (most recent call last):

File "C:/Users/RNU/PycharmProjects/PY_BASIC/FILE_IO/ex_read.py", line 48, in <module>

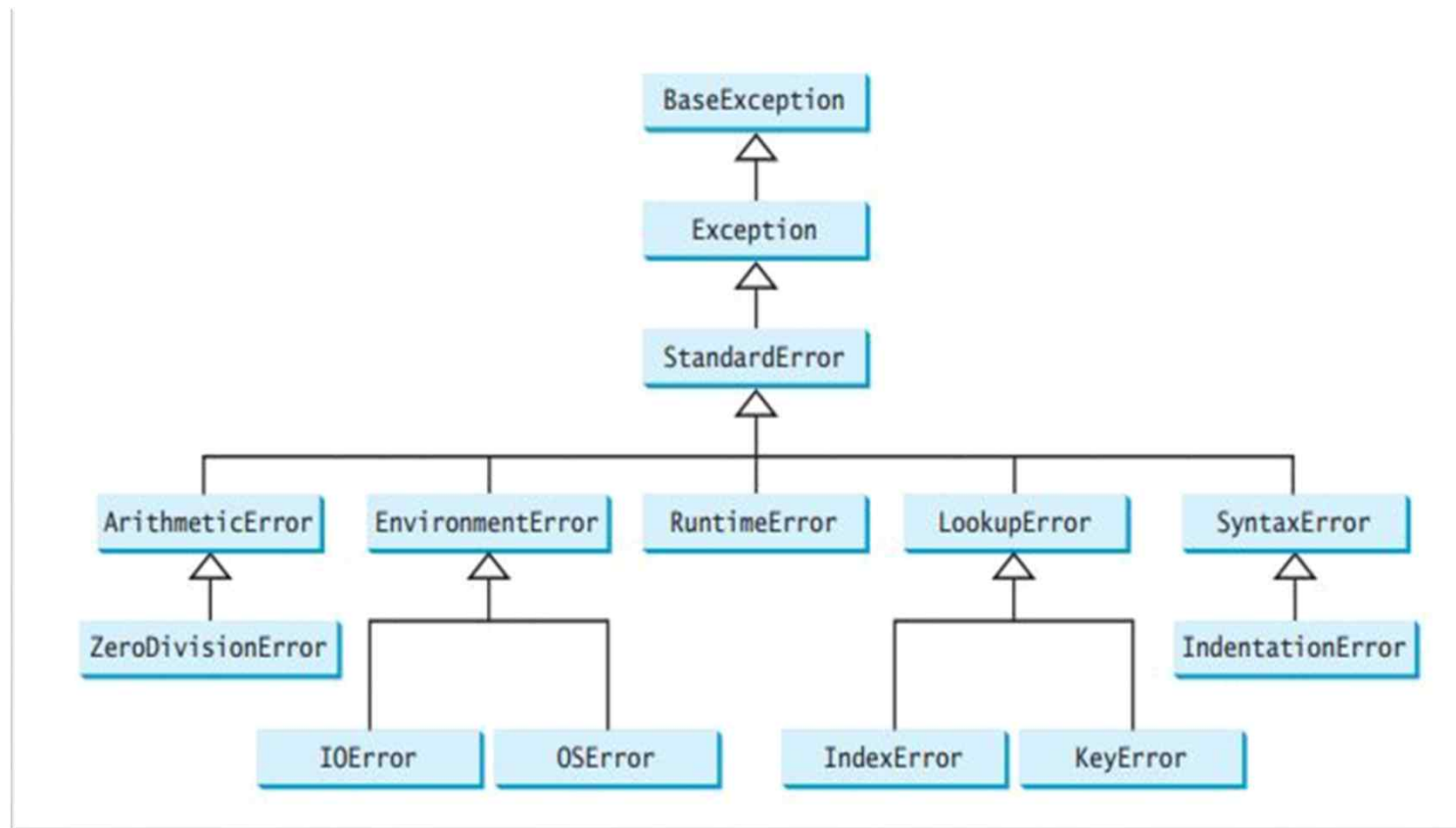
```
file = open("../Data/address.txt", 'r')
```

FileNotFoundError: [Errno 2] No such file or directory: '../Data/address.txt'

PYTHON 예외처리

◆ 예외처리 분류

- BaseException 클래스 하위 클래스로 존재



<https://thepythonguru.com/python-exception-handling/>

PYTHON 예외처리

◆ 예외처리 구문

■ try ~ except 문

try:

예외발생가능 코드

except 오류:

예외발생 처리코드

try:

예외발생가능 코드

except 오류:

예외발생 처리코드

else:

예외 발생하지 않은 경우

try:

예외발생가능 코드

except 오류:

예외발생 처리코드

else:

예외 발생하지 않은 경우

finally:

무조건실행코드

PYTHON 예외처리

◆ 예외처리 예

```
try:
```

```
    x = int( input('나눌 숫자를 입력하세요: ') )
```

```
    y = 10 / x
```

```
except ZeroDivisionError: # 0으로 나눠서 에러 발생 때 실행
```

```
    print('숫자를 0으로 나눌 수 없습니다.')
```

```
else:
```

```
    print( y )
```

```
finally:
```

```
    print(" - 끝 -- ")
```

PYTHON 예외처리

◆ 예외처리 예

```
try:
```

```
    x = int( input('나눌 숫자를 입력하세요: ') )
```

```
    y = 10 / x
```

```
except ZeroDivisionError as ze :          # 0으로 나뉘서 에러 발생 때 실행
```

```
    print('숫자를 0으로 나눌 수 없습니다.' , ze )
```

```
else:
```

```
    print( y )
```

```
finally:
```

```
    print(" - 끝 -- ")
```

PYTHON 예외처리

◆ 예외처리 예

```
y = [10, 20, 30]
```

```
try:
```

```
    index, x = map(int, input('인덱스와 나눌 숫자를 입력하세요: ').split())
```

```
    print(y[index] / x)
```

```
except ZeroDivisionError as ze:
```

```
    print('숫자를 0으로 나눌 수 없습니다.', ze)
```

```
except IndexError as ie:
```

```
    print('잘못된 인덱스입니다.', ie)
```

```
finally:
```

```
    print(" 무조건 끝")
```

여러 개 처리

PYTHON 예외처리

◆ 예외 회피/무시 구문

- try ~ except 문

try:

예외발생가능 코드

except 오류:

pass

try:

예외발생가능 코드

except 오류:

pass

else:

예외가없을경우처리

try:

예외발생가능 코드

except 오류:

pass

else:

예외없을경우처리코드

finally:

무조건실행코드

PYTHON 예외처리

◆ 강제 오류 발생

- 프로그램 실행 중 강제성 가진 작업 실행을 위한 방법
- 미 실행 시 강제 오류 발생

raise 에러이름

PYTHON 예외처리

◆ 강제 오류 발생

```
try:

    x = int(input('3의 배수를 입력하세요: '))

    if x % 3 != 0:                                # 3의 배수 아니면
        raise Exception('3의 배수가 아닙니다.') # 예외 발생시킴
    print(x)

except Exception as e:                           # 예외 발생 시 실행

    print('예외가 발생했습니다.', e)
```

PYTHON 예외처리

◆ 강제 오류 발생

```
def three_number():  
    x = int(input('3의 배수를 입력하세요: '))  
    if x % 3 != 0:                                # 3의 배수 아닌경우  
        raise Exception('3의 배수가 아닙니다.') # 예외 발생  
    print(x)  
  
try:  
    three_number()  
except Exception as e:  
    print(" 예외 발생 : ", e)
```

호출한 곳으로 예외 넘김

- 함수 내에서 예외발생
- 처리 : 함수 호출한 쪽

PYTHON 예외처리

◆ 강제 오류 발생

```
def three_number():  
    x = int(input('3의 배수를 입력하세요: '))  
    if x % 3 != 0:                                # 3의 배수 아닌경우  
        raise Exception('3의 배수가 아닙니다.') # 예외 발생  
    print(x)  
  
try:  
    three_number()  
except Exception as e:  
    print(" 예외 발생 : ", e)
```

호출한 곳으로 예외 넘김

- 함수 내에서 예외발생
- 처리 : 함수 호출한 쪽

PYTHON 예외처리

◆ 예외 생성

- 사용자 정의 예외 만들기

```
class UserException(Exception):  
  
    def __init__(self):  
        super().__init__('error message')
```

CH09. 다양한 BUILTIN FUNC

PYTHON 다양한 BUILTIN FUNC

◆ Func

all(반복 가능한(iterable) 자료형)

- 요소가 모두 참이면 True, 거짓이 하나라도 있으면 False

any(반복 가능한(iterable) 자료형)

- 요소 중 하나라도 참이면 True, 모든 요소가 거짓이면 False

PYTHON 다양한 BUILTIN FUNC

◆ Func

chr(code_value)

아스키(ASCII) 코드 값(0~127)을 입력받아 해당하는 문자 출력

```
print(f'chr(97)={chr(97)}, chr(65)={chr(65)}')
```

ord(문자)

문자의 아스키(ASCII) 코드 값(0~127) 반환 출력

```
print(f'ord("a")={ord("a")}, ord("Z")={ord("Z")}')
```

PYTHON 다양한 BUILTIN FUNC

◆ Func

divmod(a, b)

a를 b로 나눈 몫과 나머지를 튜플 형태로 돌려주는 함수

```
print(f'divmod(11,2) => {divmod(11,2)}')
```

eval(expression)

실행 가능 문자열(1+2, 'hi' + 'a' 같은 것)입력 받아 실행 결과 반환

```
print(f"eval('1+2')={eval('1+2')}")
```

```
print(f"eval('divmod(17,2)')={eval('divmod(17,2)')}")
```

PYTHON 다양한 BUILTIN FUNC

◆ Func

```
# zip( 반복 가능한(iterable) 자료형 )  
# 동일한 개수로 이루어진 자료형을 묶어 주는 역할  
  
datas=zip([1, 2, 3], [4, 5, 6])  
print(f'datas ={datas}')  
for x, y in datas:  
    print(x,y)  
  
datas=list(zip("abc", "def"))  
print(f'datas ={datas}')  
for x, y in datas:  
    print(x,y)
```

CH10. PYTHON 파일 입출력

PYTHON 파일 입출력

◆ 바이너리 & 텍스트

[데이터 분류]

데이터 타입	장점	단점
텍스트	<ul style="list-style-type: none">- 텍스트 편집기로 편집 가능- 데이터 설명 추가 가능	<ul style="list-style-type: none">- 바이너리에 비해 크기가 큼- 문자 인코딩 주의 (대부분 UTF-8)
바이너리	<ul style="list-style-type: none">- 텍스트 데이터에 비해 크기 작음- WEB에서 사용되는 데이터	<ul style="list-style-type: none">- 텍스트 편집기로 편집 불가- 데이터 설명 추가 불가

[텍스트 데이터 파일]

파일	특징
XML 파일	<ul style="list-style-type: none">- 범용적인 형식, 웹 API 활용 형식
JSON 파일	<ul style="list-style-type: none">- 자바스크립트 객체 표기 방법 기반 형식 파일- 데이터 교환에 활용
YAML	<ul style="list-style-type: none">- JSON 대용으로 사용, 어플리케이션 설정 파일에 많이 사용되는 파일
CSV/TSV	<ul style="list-style-type: none">- WEB상에서 많이 사용되는 파일

PYTHON 파일 입출력

◆ 인코딩 & 디코딩

[사람 중심]

UNICODE

가을입니다.

인코딩(Encoding)
코드화/암호화

디코딩(Decoding)
역코드화/복호화

[기계 중심]

UTF-8, ASCII 등등 형식의 BYTE

```
\xea\x00\x80\xec\x9d\x84\xec\x9e\x85\xeb\x8b\x88\xeb\x8b\xa4
```

```
0b100000000b11101100
0b100111010b10000100
0b111011000b10011110
0b100001010b11101011
0b100010110b10001000
0b111010110b10001011
0b101001000b10111000
```

PYTHON 파일 입출력

◆ 파일 읽기& 쓰기

쓰기 → file_Object = `open(file , mode='w', encoding=None)`

읽기 → file_Object = `open(file , mode='r', encoding=None)`

racter	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)

PYTHON 파일 입출력

◆ 파일 문자열 데이터 쓰기

파일 전체 데이터 쓰기 ➔ `alldata=f.write(str)`

파일 줄 단위 데이터 쓰기 ➔ `line = f.writeline(list)`

**** 자동 개행 안됨**

PYTHON 파일 입출력

◆ 파일 문자열 데이터 읽기

파일 전체 데이터

→ `alldata=f.read()`

파일 `n`만큼 데이터

→ `alldata=f.read(n)`

파일 줄 단위 데이터

→ `line = f.readline()`

파일 줄 단위 전체 리스트 반환

→ `lines = f.readlines()`

PYTHON 파일 입출력

◆ 파일 포인터 위치

파일 위치 설정/이동 ➔ `f.seek(offset)` # `f.seek(0)` 파일 처음

파일 위치 읽기 ➔ `f.tell()`

파일 닫힘 여부 반환 ➔ `f.closed`

파일모드 반환 ➔ `f.mode`

파일이름 반환 ➔ `f.name`

PYTHON 파일 입출력

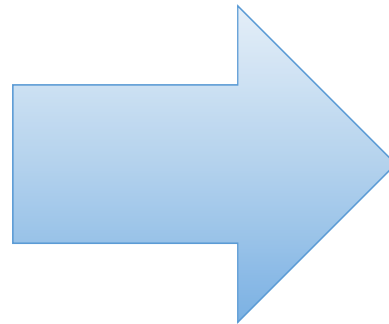
◆ 파일 문자열 데이터 읽기

```
alldata=f.read( )
```

```
alldata=f.read( n )
```

```
line = f.readline( )
```

```
lines = f.readlines( )
```



1바이트 크기
배열 bytes 객체

PYTHON 파일 입출력

◆ 파일 객체 데이터 쓰기 & 읽기

■ Pickle 모듈

■ 객체 직렬화

- 일반적인 텍스트 파일 이외에 자료형 데이터 저장
- 자료형의 변경없이 파일로 저장하여 그대로 로드
- 바이트 형식으로 읽기 & 쓰기
- 모든 파이썬 데이터 객체 저장 & 읽기

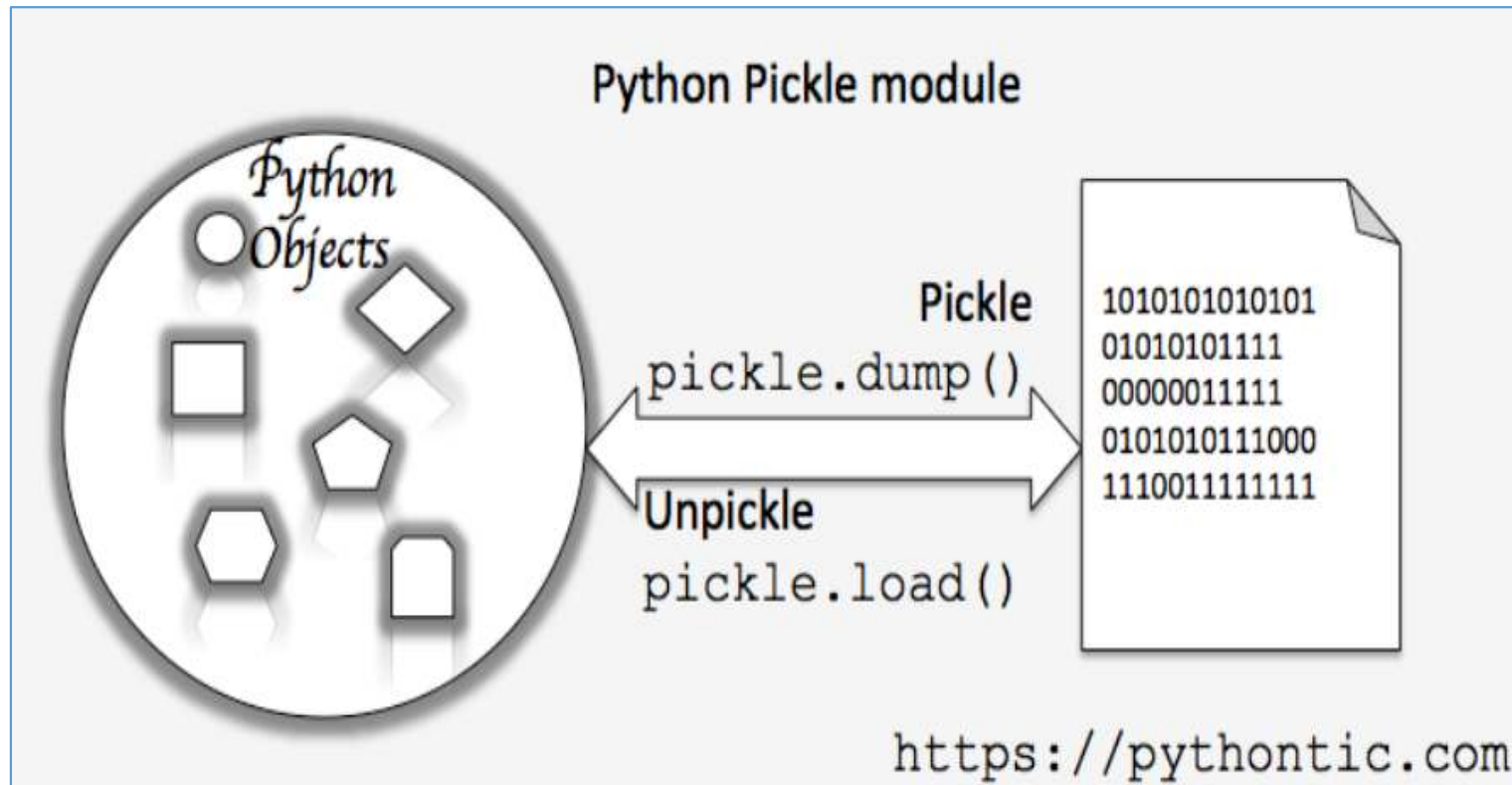
쓰기 → `pickle.dump(data, file)`

읽기 → `pickle.load(file)`

PYTHON 파일 입출력

◆ 파일 객체 데이터 쓰기 & 읽기

■ Pickle 모듈



PYTHON 파일 입출력

◆ with 파일 as 구문

```
with open(file , mode='w', encoding=None) as file_obj:  
    file_obj.write( ' data ' )
```

file_obj.close() 생략 가능

파일의 close() 자동으로 처리됨!!!

다양한 FILE I/O

PYTHON 파일 입출력

◆ CSV 파일

- >> 데이터 값을 쉼표(,)로 구분하는 파일
- >> Comma Separated Values 약자
- >> 쉼표(,)로 열 구분, 줄바꿈으로 행(row) 구분
- >> TSV(Tab), SSV(Space) 데이터 파일도 존재

PYTHON 파일 입출력

◆ CSV 파일

형태 → 데이터,데이터,데이터

1,13.2,9.1,blue

1,98,2.8,gray

2,10.5,81.3,red

1,8.8,5.21,yellow

1;13.2;9.1;blue

1;98;2.8;gray

2;10.5;81.3;red

1;8.8;5.21;yellow

PYTHON 파일 입출력

◆ CSV 파일

import csv

- <https://docs.python.org/ko/3.7/library/csv.html?highlight=csv>
- csv 파일 처리 표준 라이브러리

csv.reader(file) ➔ 읽은 csv 데이터 문자열 리스트 반환

csv.writer(file) ➔ csv 파일 데이터 쓰기 위한 객체 반환

writer_obj.writerow(row) ➔ 한 줄 쓰기

writer_obj.writerows([row, row, ..]) ➔ 여러 줄 쓰기

PYTHON 파일 입출력

◆ EXCEL파일

>> MS사의 Excel

>> xlsx 확장자지원

>> 설치

```
pip install openpyxl == 3.0.1
```

<https://pypi.org/project/openpyxl/3.0.1/>

PYTHON 파일 입출력

◆ JSON 파일

- >> JavaScript Object Notation 약자
- >> 자바스크립트에서 사용하는 객체 표기 방법
- >> 다양한 프로그래밍 언어에서 데이터 교환에 사용
- >> 인코딩/디코딩 표준으로도 사용
- >> <https://docs.python.org/ko/3.7/library/json.html?highlight=json#module-json>

PYTHON 파일 입출력

◆ JSON 파일

형태 : [{ 키:값, 키:값, 키:{ 키:값, 키:값 } }]

```
[ { 'id':1,  
    'name':'jane',  
    'data' : { 'major':'science',  
                'grade': 1 }  
  }, { 'id':2,  
        'name':'Tom',  
        'data' : { 'major':'math',  
                    'grade': 2 }  
  } ]
```


PYTHON 파일 입출력

◆ JSON 파일

```
import json
```

```
with open('../Data/test.json', 'w') as f:  
    # json파일 쓰기  
    json.dump(json_data , f)
```

```
with open('../Data/test.json', 'r') as f:  
    # json파일 읽기  
    json_data = json.load(f)
```

바이너리 FILE I/O

PYTHON 파일 입출력

◆ 바이너리 파일

- 이진 파일 또는 바이너리 파일(binary file)
- 컴퓨터 저장과 처리 목적 위해 이진 형식으로 인코딩된 데이터 파일
- 문서 편집기로 열었을 경우 알아볼수 없는 문자들

[바이너리 데이터 파일]

분류	파일 종류
이미지 파일	- jpg, png,
오디오 파일	- mp3, mp4, ...
실행 파일	- exe, bin,

PYTHON 파일 입출력

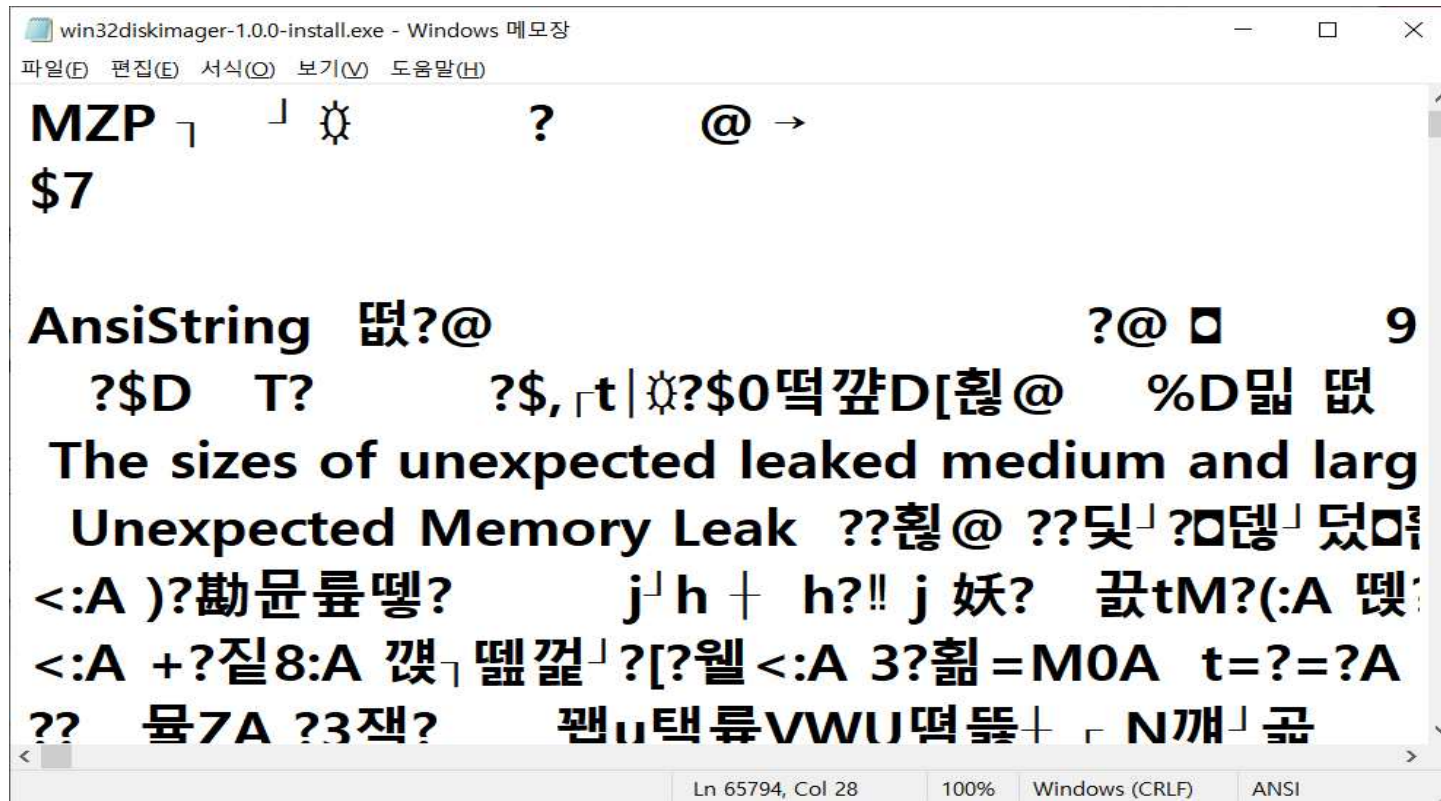
◆ 바이너리 데이터 타입

bytes 데이터 타입

- 1바이트 단위 값을 연속적으로 저장하는 시퀀스 자료형
- 1바이트 => 8비트
- 0~255(0x00~0xFF)까지 정수 사용

PYTHON 파일 입출력

◆ 바이너리 파일



win32diskimager-1.0.0-install.exe - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

MZIP ǀ ǂ ǃ ? @ →

\$7

AnsiString 덫?@ ?@ 9

?\$D T? ?\$, ǂt|ǃ? \$0떡깟D[훤@ %D밧 덫

The sizes of unexpected leaked medium and larg

Unexpected Memory Leak ??훤@ ??덫 ǂ ǃ덫 덫

<:A)?勘문릏덫? j ǂh + h?!! j 妖? ǃtM?(:A 덫

<:A +?질8:A ǃ 덫 ǃ ǂ[?웰<:A 3?훤=M0A t=?=?A

?? ㄴ7A ?3잭? ǃ ǂ ǃ ǃ VWU떡뽏+ ǂ N개 ǃ ǃ

Ln 65794, Col 28 100% Windows (CRLF) ANSI

PYTHON 파일 입출력

◆ 바이너리 데이터 타입

bytes 객체 생성

bytes(숫자):	숫자만큼 0으로 채워진 바이트 객체 생성
bytes(반복가능한객체)	반복 가능한 객체로 바이트 객체 생성
bytes(b'바이트객체')	바이트 객체로 바이트 객체 생성

PYTHON 파일 입출력

◆ 바이너리 모듈

`import struct`

- C언어의 구조체를 구현한 모듈
- 파일이나 네트워크 연결에 사용하는 이진 데이터 다루는 모듈

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1), (2)
B	unsigned char	integer	1	(2)
?	_Bool	bool	1	(1)
h	short	integer	2	(2)
H	unsigned short	integer	2	(2)
i	int	integer	4	(2)
I	unsigned int	integer	4	(2)
l	long	integer	4	(2)
L	unsigned long	integer	4	(2)
q	long long	integer	8	(2)