

데이터분석을 위한 PYTHON 2

2022. 06



PART Ⅱ

모듈 활용 프로그래밍

CH01. PYTHON 모듈 & 패키지

PYTHON 모듈 & 패키지

◆ 모듈(Module)

- 기능 수행 위해 변수, 함수, 클래스를 모아둔 파일
- python에 다양한 기능을 확장시켜 줄 수 있는 것
- 다른 Python 프로그램에서 불러서 사용가능하게 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

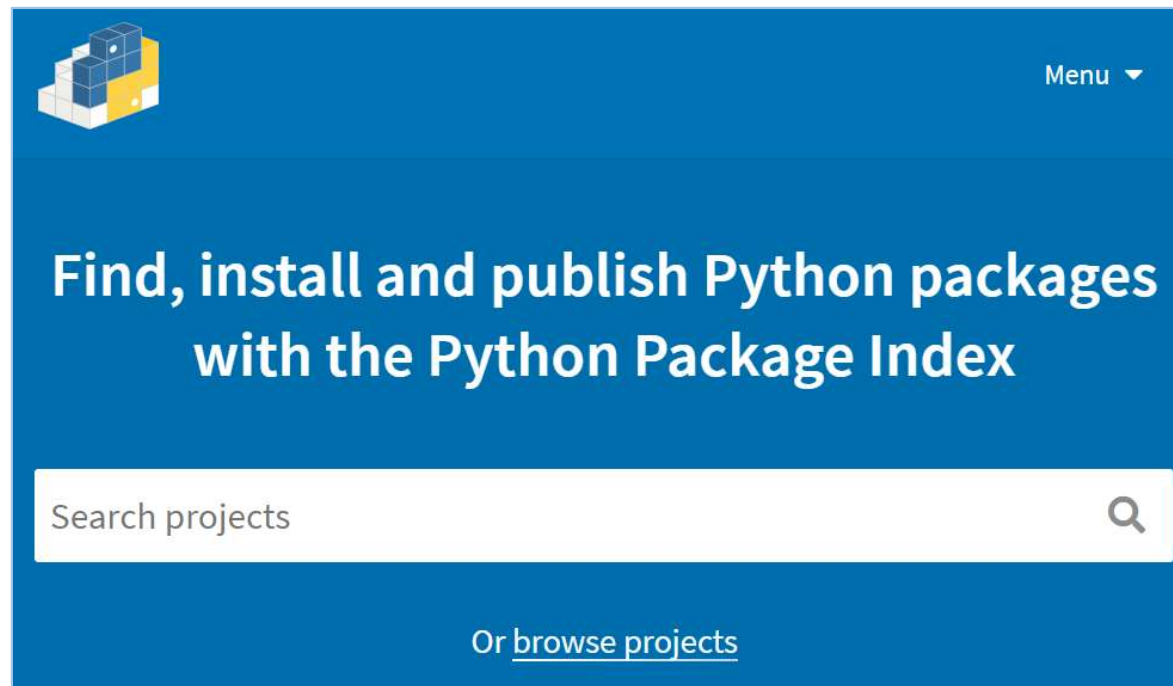
- 파이썬 패키지 인덱스(Python Package Index, PyPI)
- www.pypi.org

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 x +

(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 |

CH02. 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>

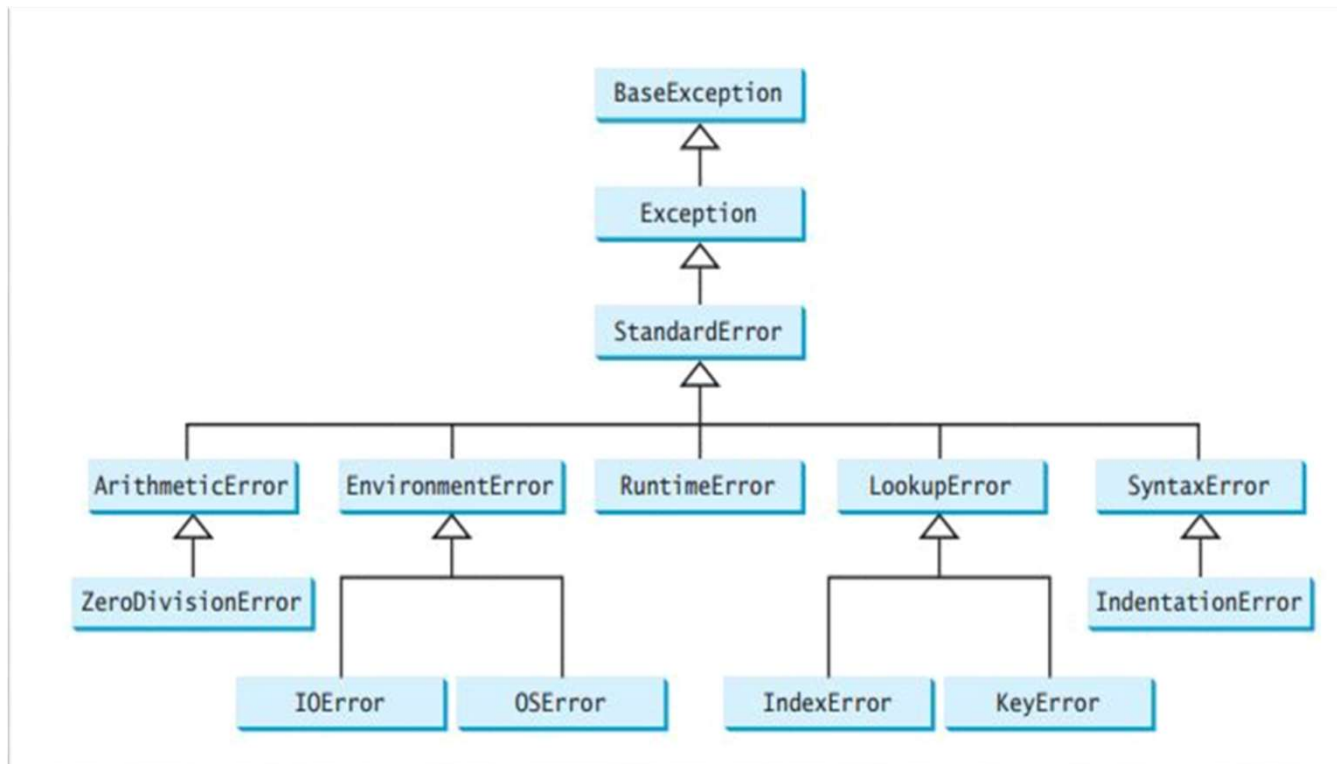
```
file = open("../Data/address.txt", 'r')
```

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')
```

CH03. 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

map (함수명, 반복가능객체)

반복가능객체 요소에 함수를 적용한 결과 반환 함수

map 객체 반환으로 list나 tuple로 캐스팅하여 사용

```
numList=['1','2','3','4','5' ]
```

```
numList=list(map(int, numList))
```

filter (함수명, 반복가능객체)

반복가능객체 요소에 함수 적용한 후 True인 결과만 반환 함수

filter 객체 반환으로 list나 tuple로 캐스팅하여 사용

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

```
numList=list(filter(int, numList))
```

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)
```

CH04. 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`

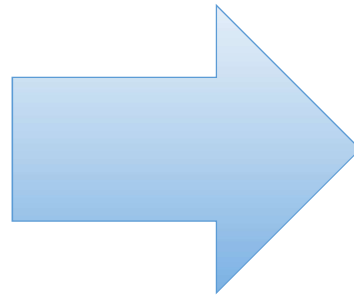
파일모드 반환 → `f.mode`

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

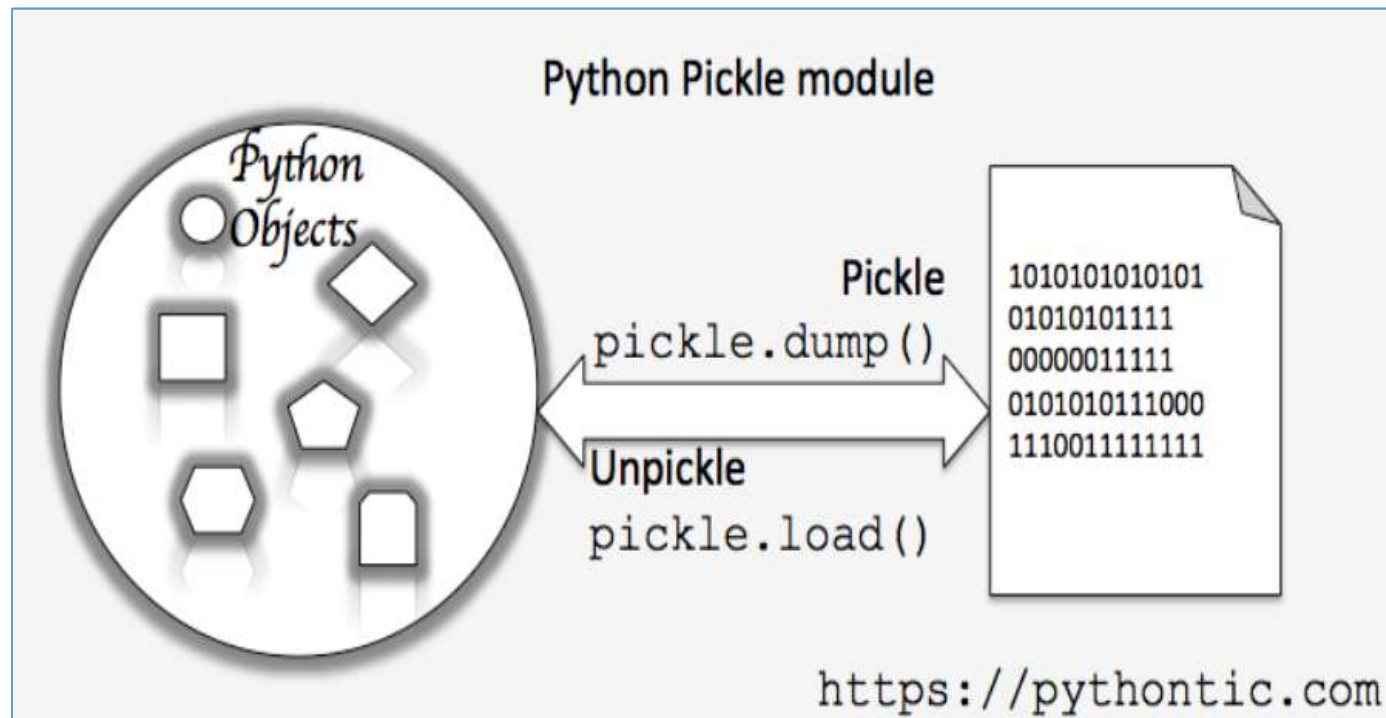
쓰기 → `pickle.dump(data, file)`

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!!!

CH05. PYTHON 다양한 파일 입출력

PYTHON 다양한 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 다양한 파일 입출력

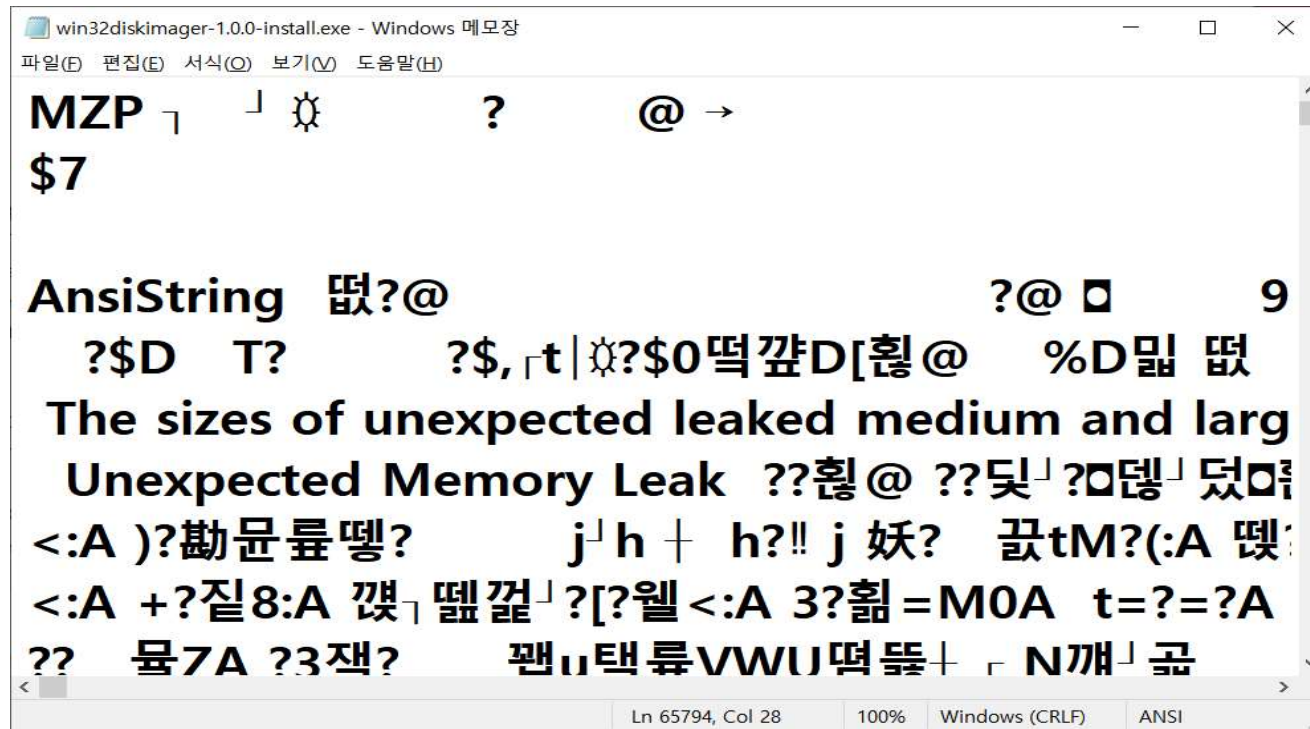
◆ 바이너리 데이터 타입

bytes 데이터 타입

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

PYTHON 다양한 파일 입출력

◆ 바이너리 파일



```
win32diskimager-1.0.0-install.exe - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
MZIP
$7
AnsiString
The sizes of unexpected leaked medium and larg
Unexpected Memory Leak
<:A )?勘문릍땡? j h + h?!! j 妖? 끝tM?(:A 땡:
<:A +?질8:A 갯 땡겔?[*웰<:A 3?휼=M0A t=?=?A
?? 뭉7A ?3책? 팬,택릍VWU땡똥+ N개 공
Ln 65794, Col 28 100% Windows (CRLF) ANSI
```

PYTHON 다양한 파일 입출력

◆ 바이너리 데이터 타입

bytes 객체 생성

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

PYTHON 다양한 파일 입출력

◆ 바이너리 데이터 타입

bytes 객체 생성

```
# -----  
# 바이너리 타입 => bytes  
# 1바이트 데이터를 연속적으로 저장하는 Sequence 자료형  
# 수정 불가  
# -----  
a=bytes(10)  
print(f'a=>{a}, type => {type(a)}')  
  
a=bytes([11,12])  
print(f'a=>{a}, type => {type(a)}')  
  
for aa in a:    print(aa)  
print(f'a[0]=>{a[0]} a[1]=>{a[1]}')
```


PYTHON 다양한 파일 입출력

◆ 바이너리 데이터 타입

bytes 객체 생성

```
# str => byte
b=bytes('Hello'.encode())

# byte => str
print(f'b=>{b}, type => {type(b)}, b.decode()=> {b.decode()}')

b2=bytes('한글'.encode('utf-8'))

# byte => str
print(f'b2=>{b2}, type => {type(b2)}, b2.decode()=> {b2.decode()}')
```

PYTHON 다양한 파일 입출력

◆ 바이너리 데이터 타입

bytearray 객체 생성

```
# -----  
# 바이너리 타입 => bytearray  
# 1바이트 데이터를 연속적으로 저장하는 Sequence 자료형  
# 수정 가능  
# -----  
a=bytearray(10)  
print(f'a=>{a}, type => {type(a)}')  
  
a=bytearray([11,12])  
print(f'a=>{a}, type => {type(a)}')  
  
a[0]=254
```

PYTHON 다양한 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)

PYTHON 다양한 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 다양한 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 다양한 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 다양한 파일 입출력

◆ EXCEL파일

>> MS사의 Excel

>> xlsx 확장자지원

>> 설치

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

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

PYTHON 다양한 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 다양한 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 다양한 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
```

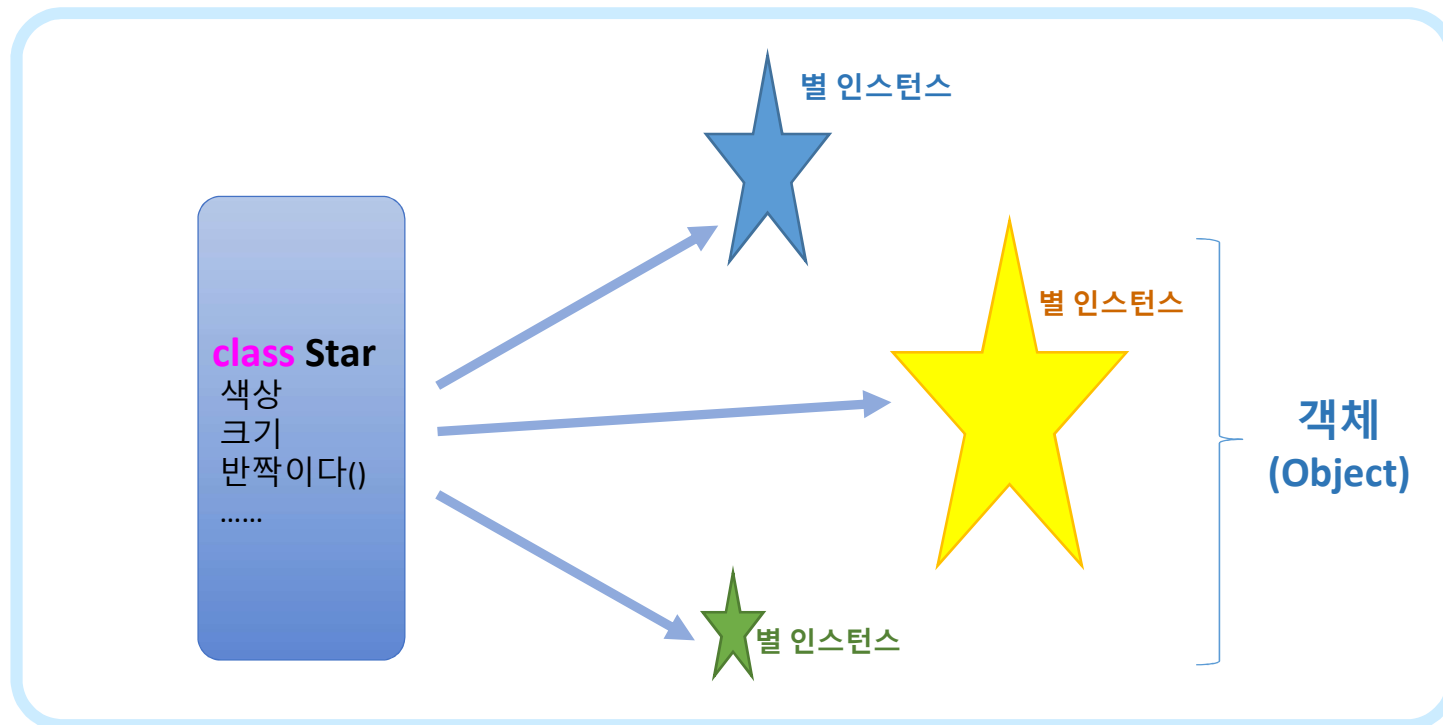
```
    json_data = json.load(f)
```

CH06. PYTHON 클래스

PYTHON 클래스

◆ 클래스(CLASS)

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

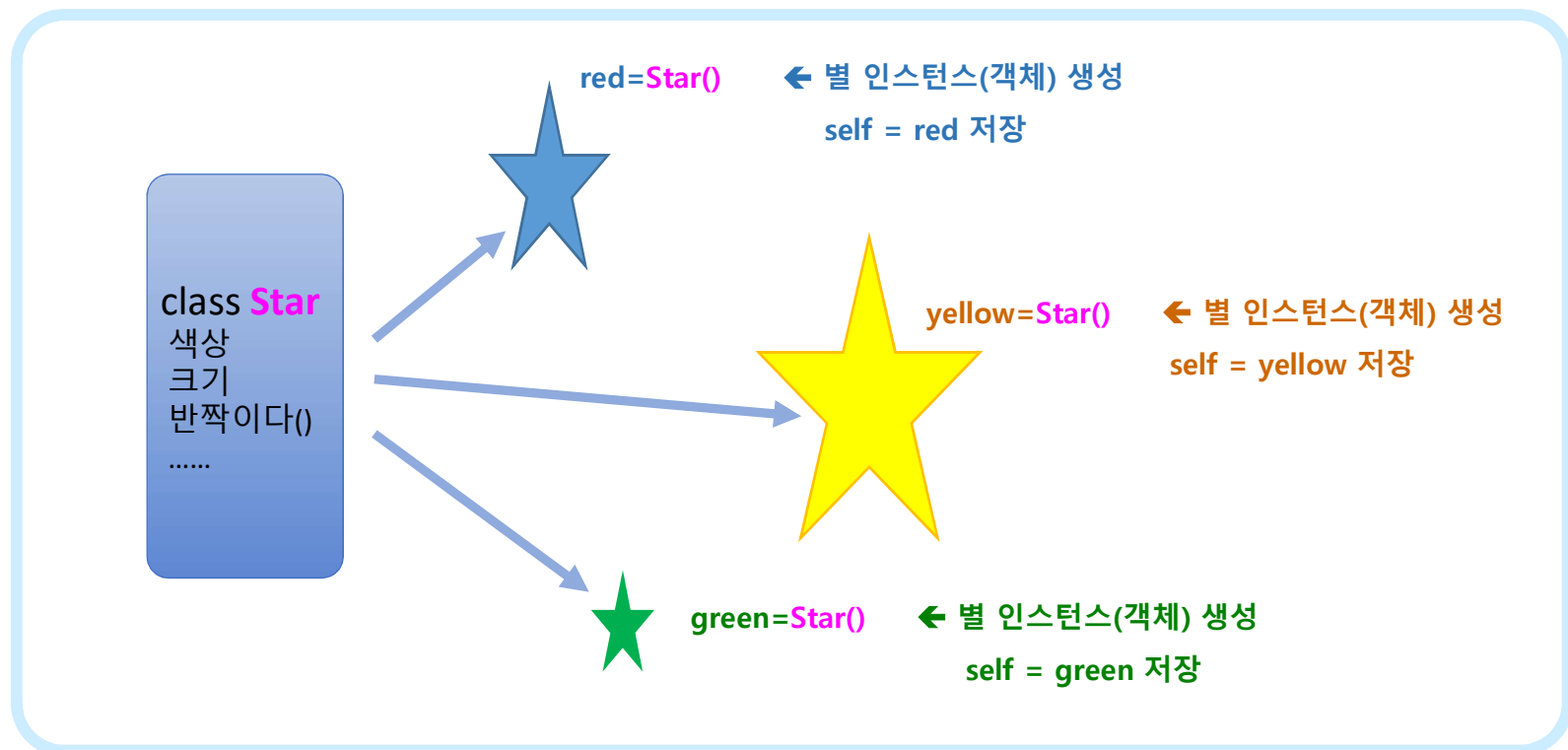


PYTHON 클래스

◆ 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")
```

메서드 만 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

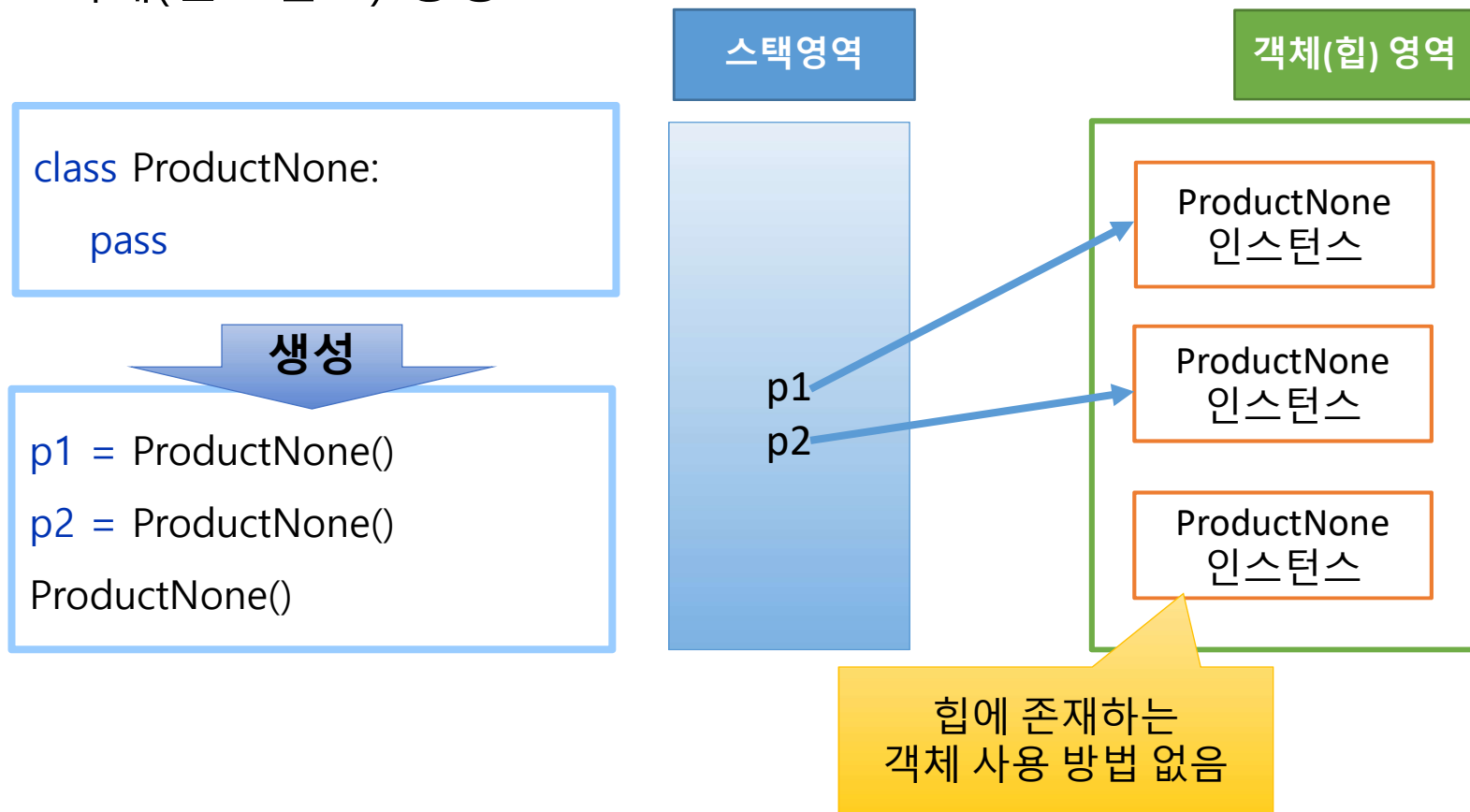
➤ 객체(인스턴스) 속성 & 메서드 사용

```
# -----  
# 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
```

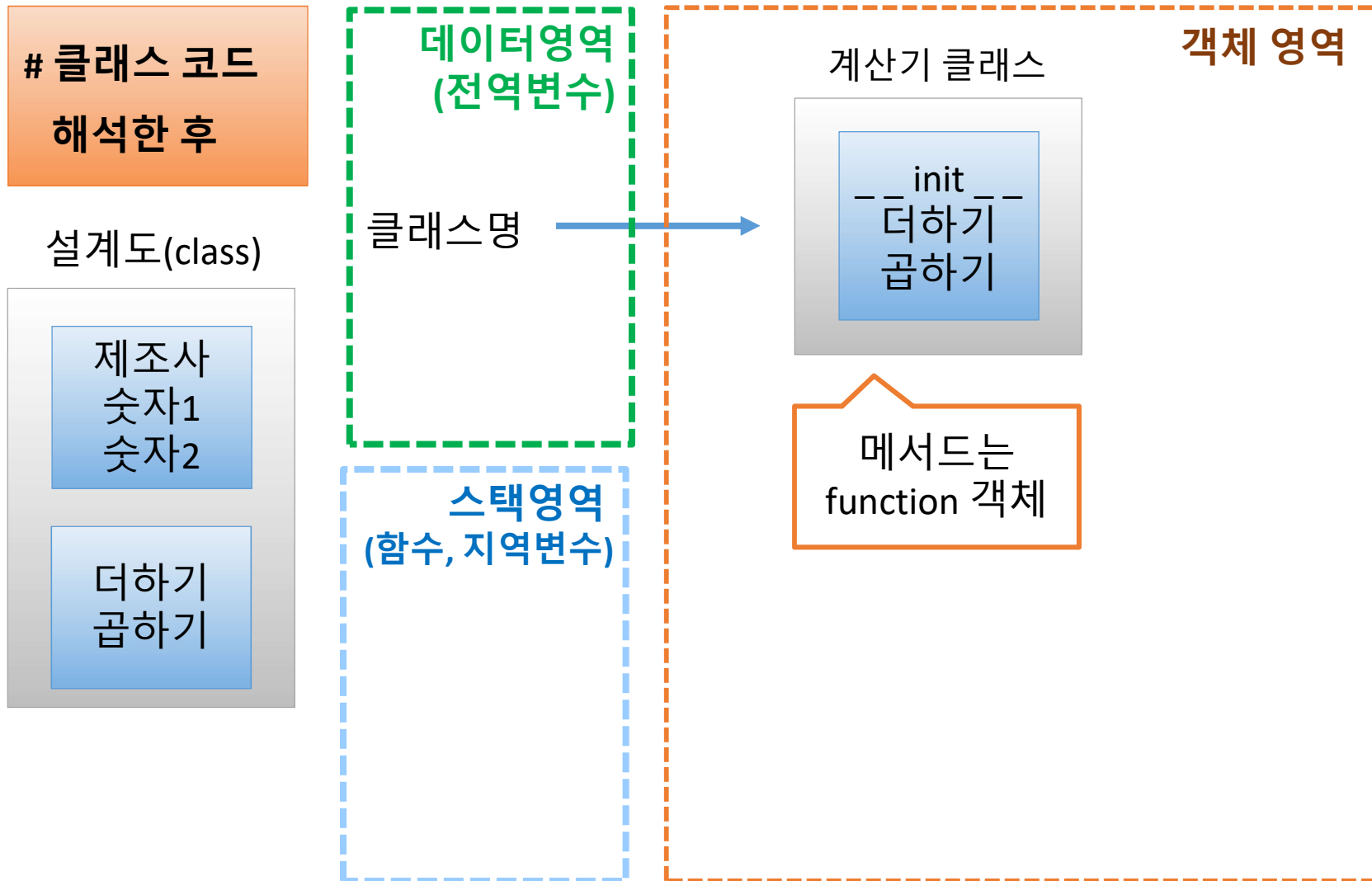
PYTHON 클래스

◆ 클래스(CLASS) 구성

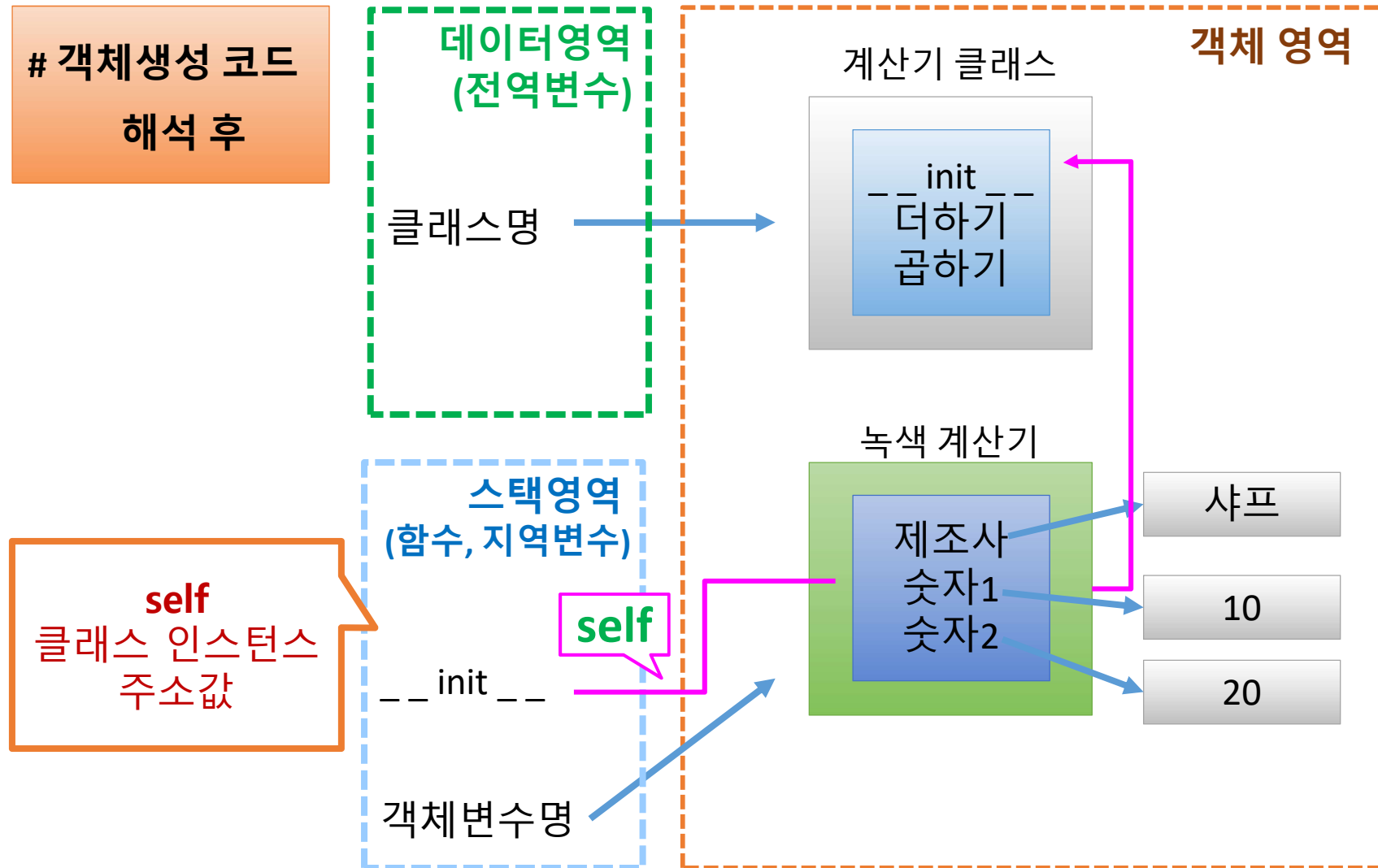
➤ 객체(인스턴스) 생성



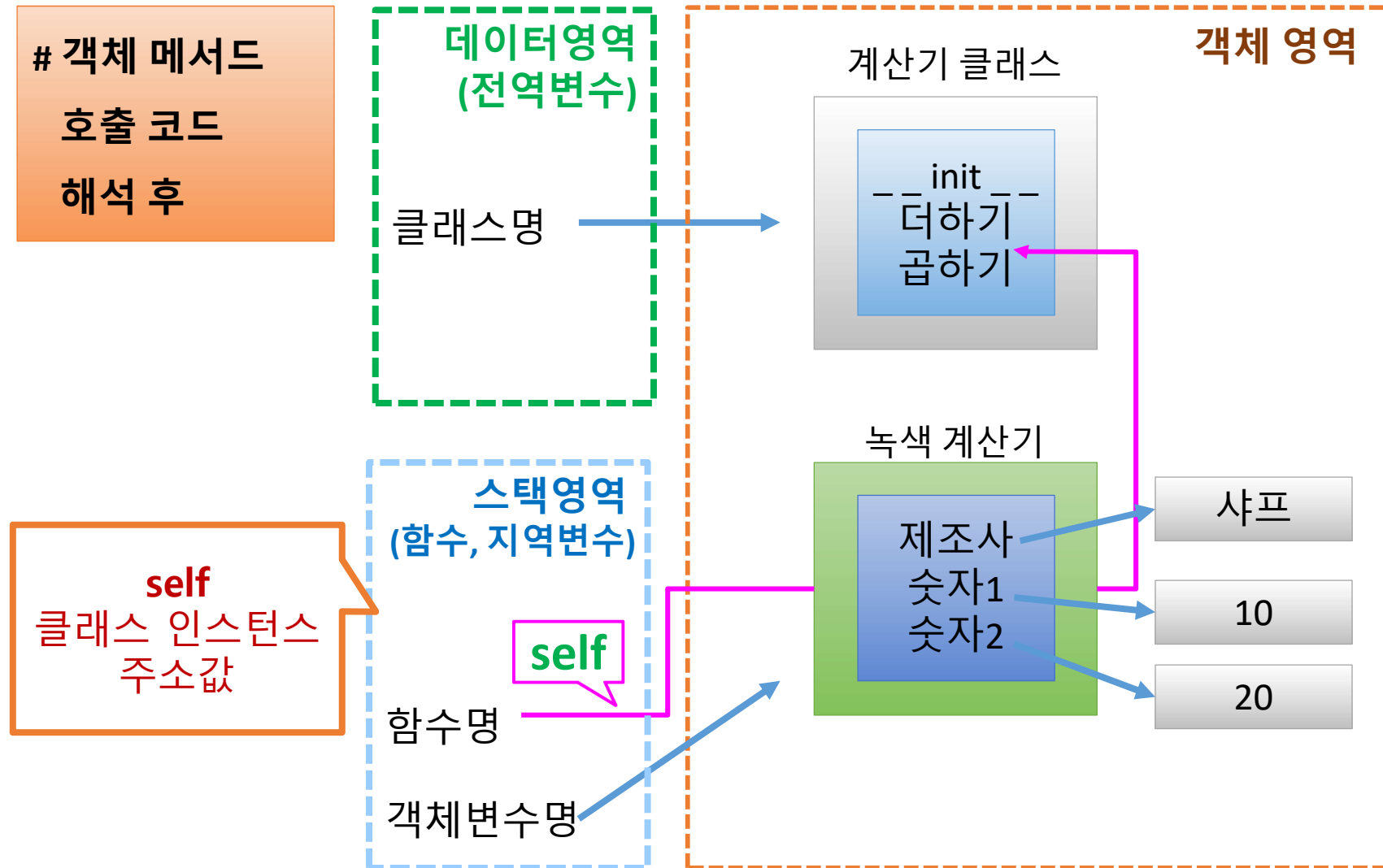
PYTHON 클래스



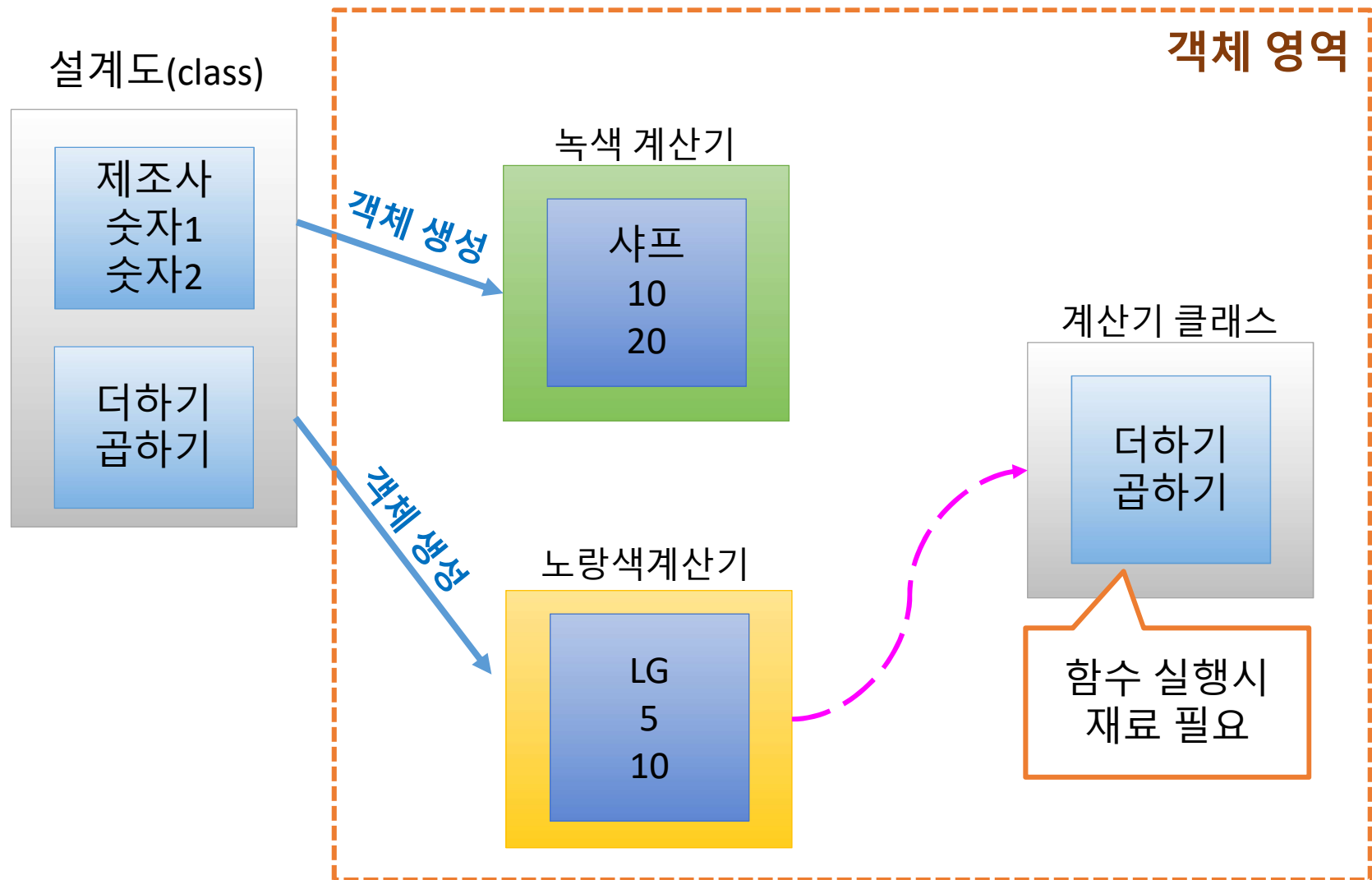
PYTHON 클래스



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)**

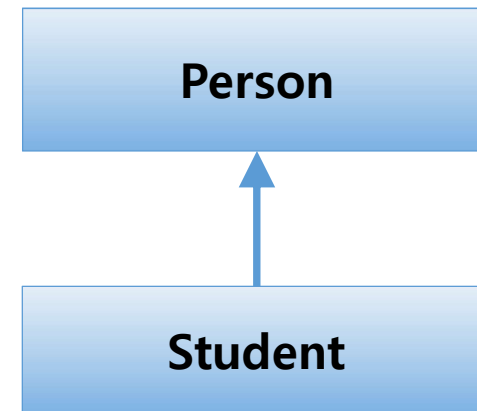
형식 → `class` 자식클래스명 (부모클래스명)

PYTHON 클래스

◆ 상속(inheritance)

```
class Person:  
    pass
```

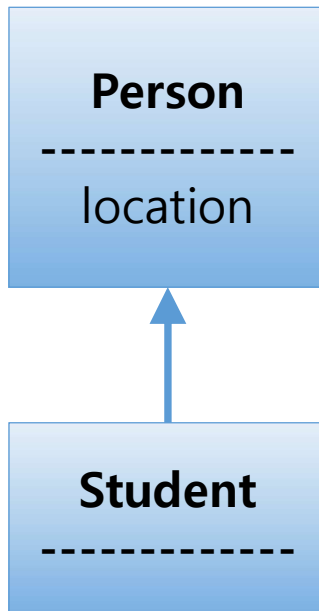
```
class Student(Person):  
    pass
```



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

PYTHON 클래스

◆ 상속(inheritance)



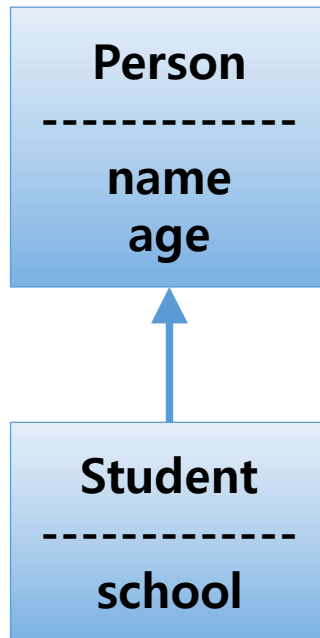
```
class Person:
    def __init__(self):
        self.location='Korea'

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}wt school : {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. 정규식

PYTHON 정규식

◆ 정규식(Regular Expression)

- 특정한 규칙 가진 문자열 집합을 표현하는 형식영어
- Regex 또는 Regexp라고도 함
- 텍스트 편집기와 프로그래밍 언어에서 문자열의 검색과 치환 위해 지원
- 텍스트가 준수해야 하는 “패턴” 표현하는 특정한 표준의 텍스트문법
- 구성 ➔ 메타문자 & 수량자

PYTHON 정규식

◆ 정규식(Regular Expression)

➤ 메타문자(Meta Character)

메타문자	형식	예시	의미
.	.	a.b	줄바꿈 문자(\n) 제외한 임의의 모든 문자
^	^abc	abc1234 abcTEST	시작 문자열
\$	abc\$	124abc testabc	종료 문자열
*	*	ca*t	*앞의 문자 0번 이상 반복
+	+	ca+t	+앞의 문자 1번 이상 반복
?	?	ab?c	?앞의 문자가 0번 또는 1번 발생



PYTHON 정규식

◆ 정규식(Regular Expression)

➤ 메타문자(Meta Character)

메타문자	형식	예시	의미
{ }	{ n,m }	cat{1,3}t	앞문자가 n개 이상 m개 이하
	{ n }	ca{2}t	앞문자가 n개
	{ n, }	cat{3,}t	앞문자가 n개 이상
[]	[문자]	[a-zA-Z] [0-9]	대괄호 사이 문자와 일치
[^]	[^문자]	[^a-zA-Z] [^0-9]	대괄호 사이 문자 제외
()	(문자)	"a(b d)c"	소괄호 안의 문자를 하나로 인식, 그룹핑
	a b	"abc adc"	a 또는 b 여야 함 (or)



PYTHON 정규식

◆ 정규식(Regular Expression)

➤ 메타문자(Meta Character) - 확장문자

메타문자	의미
\w	확장문자의 시작
\wb	단어의 경계
\WB	단어가 아닌 것의 경계
\WA	입력의 시작 부분
\WG	이전 매치의 끝
\WZ	입력의 끝이지만 종결자가 있는 경우
\Wz	입력의 끝



PYTHON 정규식

◆ 정규식(Regular Expression)

➤ 메타문자(Meta Character) - 확장문자

메타문자	의미
\s	공백문자
\S	공백문자가 아닌 나머지 문자
\w	알파벳이나 숫자
\W	알파벳이나 숫자 제외한 문자
\d	[0-9]와 동일
\D	숫자를 제외한 모든 문자



PYTHON 정규식

◆ 정규식(Regular Expression)

➤ 메타문자(Meta Character) - 확장문자

<code>^[0-9]*\$</code>	숫자
<code>^[a-zA-Z]*\$</code>	영문자
<code>^[가-힣]*\$</code>	한글
<code>\w+@\w+\.\w+(\.\w+)?</code>	이메일 주소
<code>^\d{2,3}-\d{3,4}-\d{4}\$</code>	전화번호
<code>^01(?:0 1 [6-9])-(?:\d{3} \d{4})-\d{4}\$</code>	핸드폰 번호
<code>\d{6} \- [1-4]\d{6}</code>	주민등록 번호
<code>^\d{3}-\d{2}\$</code>	우편번호



PYTHON 정규식

◆ re 모듈

➤ import re

함수명	기능
compiler(패턴)	패턴 컴파일 / 패턴 객체 리턴
match(패턴, 확인 문자열)	확인 문자열이 전부 패턴에 일치하는 지 여부, match 객체 리턴
search(패턴, 확인 문자열)	확인 문자열에 패턴이 존재하는지 여부, match 객체 리턴
findall(확인 문자열)	패턴과 일치되는 문자열 리스트 리턴
finditer(확인 문자열)	정규식과 매치되는 모든 문자열을 반복 가능한 객체로 돌려준다.



PYTHON 정규식

◆ re 모듈

➤ Match 객체 함수

함수명	기능
<code>group()</code>	매치된 첫번째 문자열 반환
<code>start()</code>	매치된 문자열 처음 위치 반환
<code>end()</code>	매치된 문자열 끝 위치 반환
<code>span()</code>	문자열 문자열의 시작과 끝을 튜플로 반환

