

# Python

---

Python programming

장민창

mcjang@hucloud.co.kr

# 0. Python Dev Env

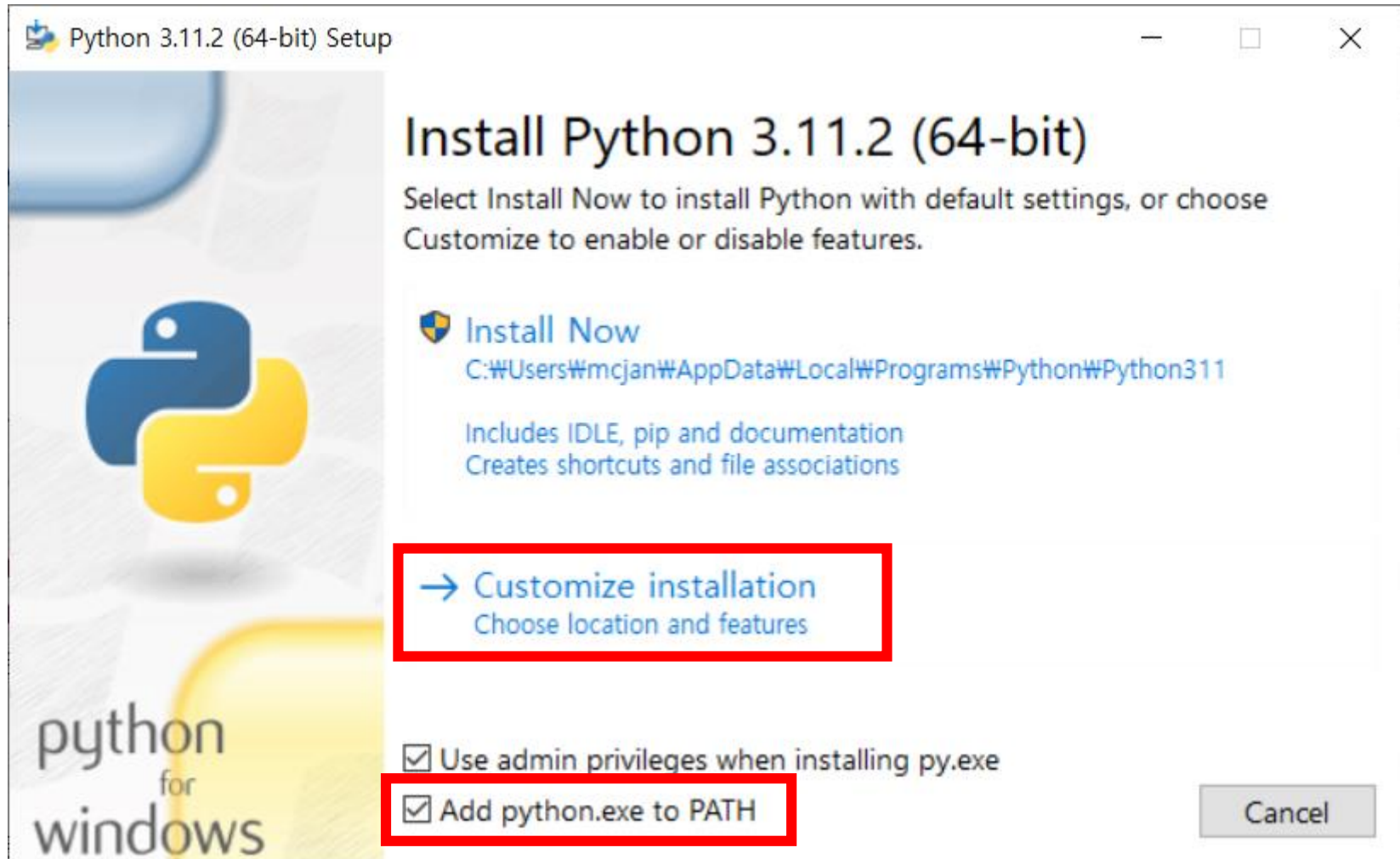
# 0. Python Dev Env

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



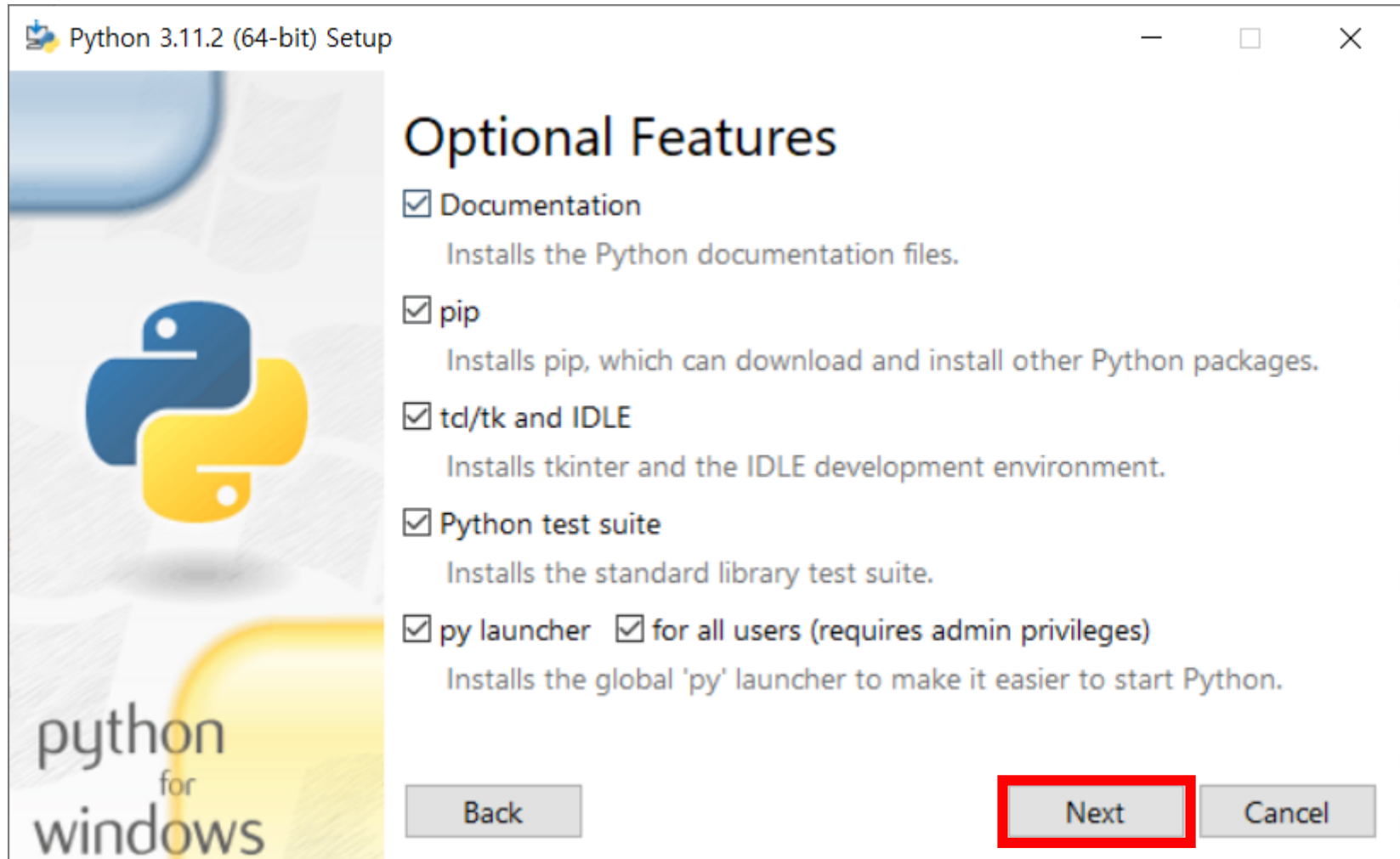
# 0. Python Dev Env

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



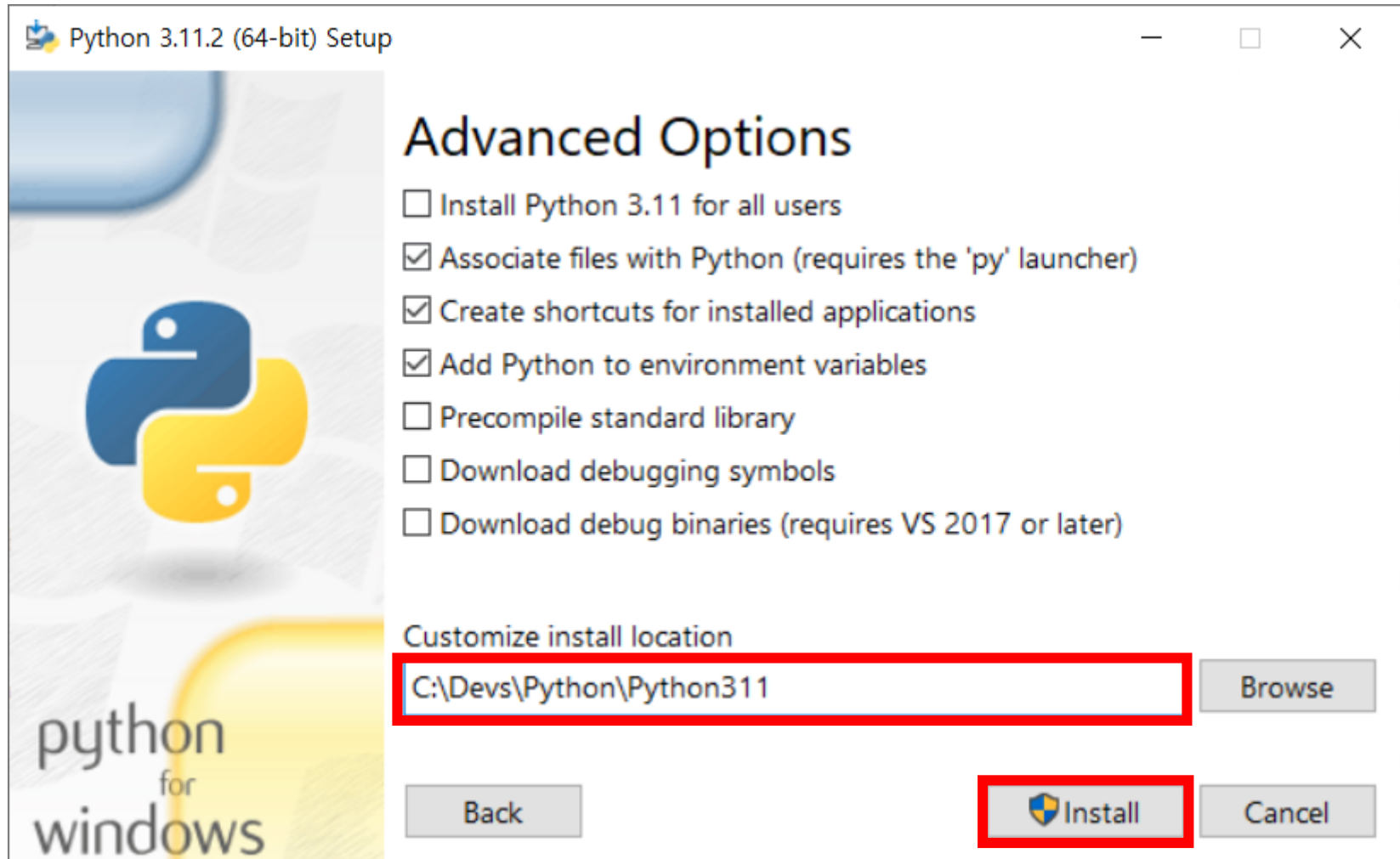
# 0. Python Dev Env

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



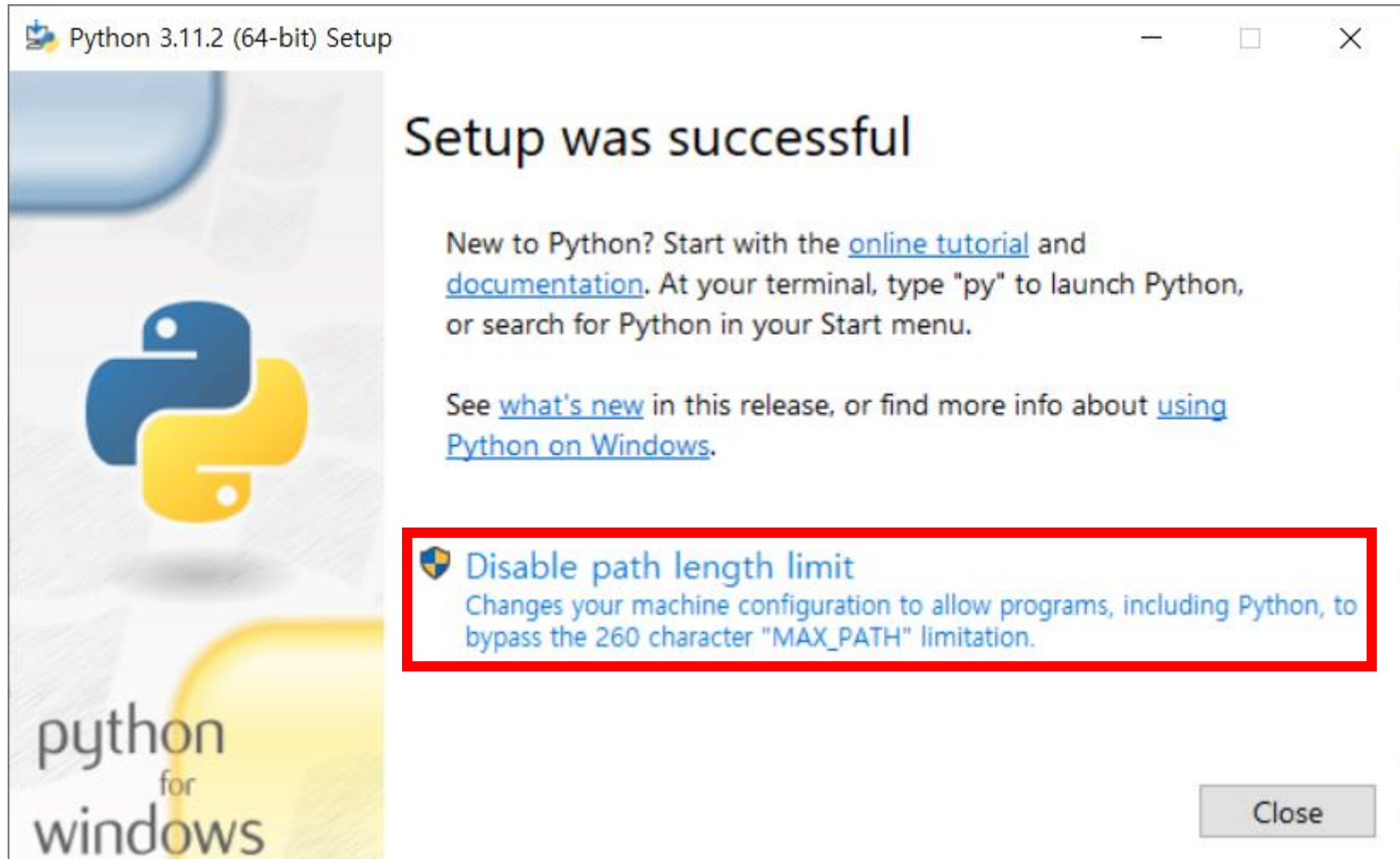
# 0. Python Dev Env

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



# 0. Python Dev Env

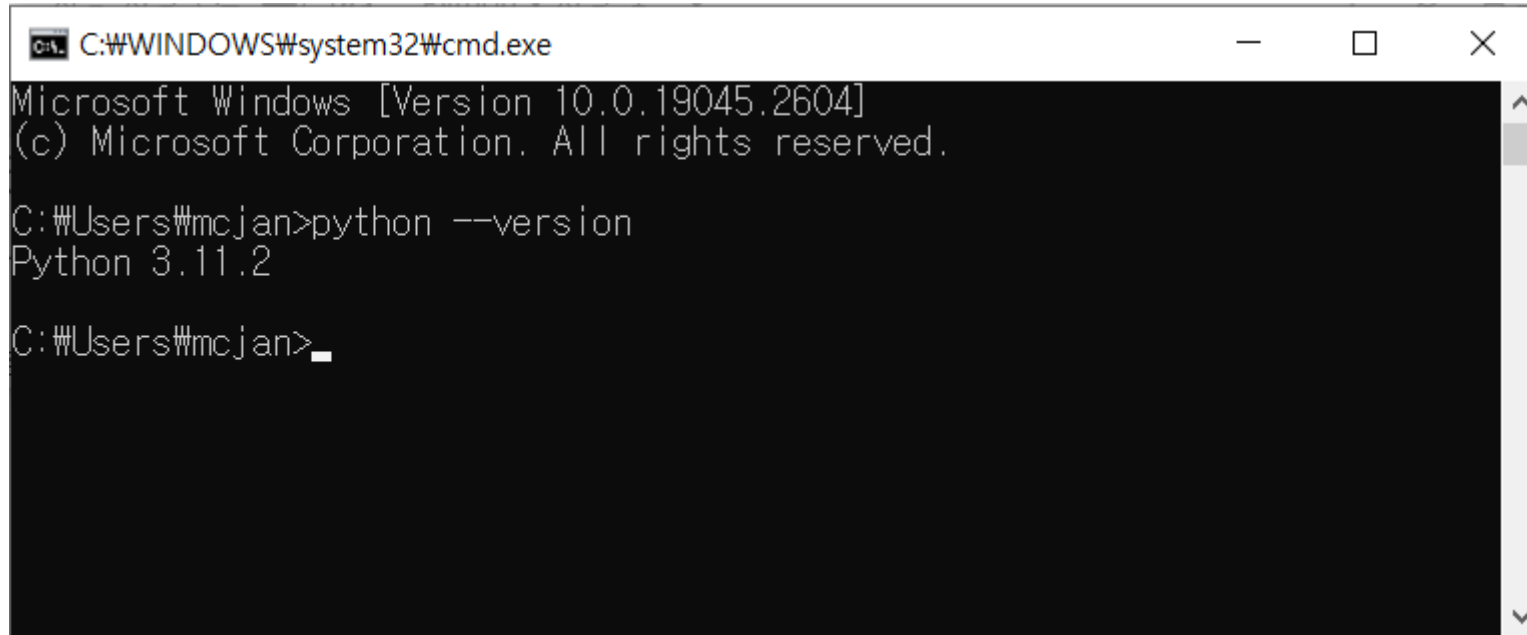
- <https://www.python.org/>



# 0. Python Dev Env

---

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



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mcjan>python --version
Python 3.11.2

C:\Users\mcjan>_
```



# 0. Python Dev Env

- vscode 플러그인 설치

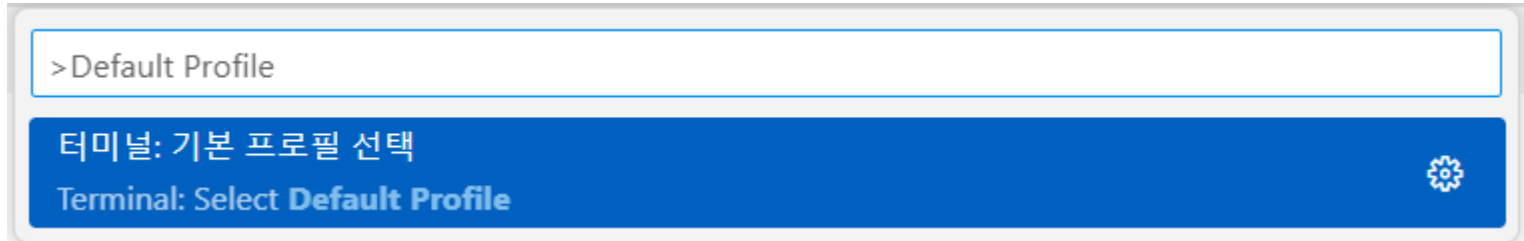
The screenshot displays the Visual Studio Code interface with the 'EXTENSIONS: MARKETPLACE' view open. The search bar contains 'python'. The left sidebar lists several Python-related extensions:

- Python** (Microsoft): IntelliSense (Pylance), Linting, Debugging (multi-threaded, r...  
80.7M stars, 4 stars, Install button
- Python Extension Pack** (Don Jayamanne): Popular Visual Studio Code extensions for Python  
4.9M stars, 4 stars, Install button
- Python for VSCode** (Thomas Haakon Townsend): Python language extension for vscode  
5M stars, 2 stars, Install button
- Python Indent** (Kevin Rose): Correct Python indentation  
4.4M stars, 4.5 stars, Install button
- autoDocstring - Python Docstring Generator**: 4.5M stars, 5 stars

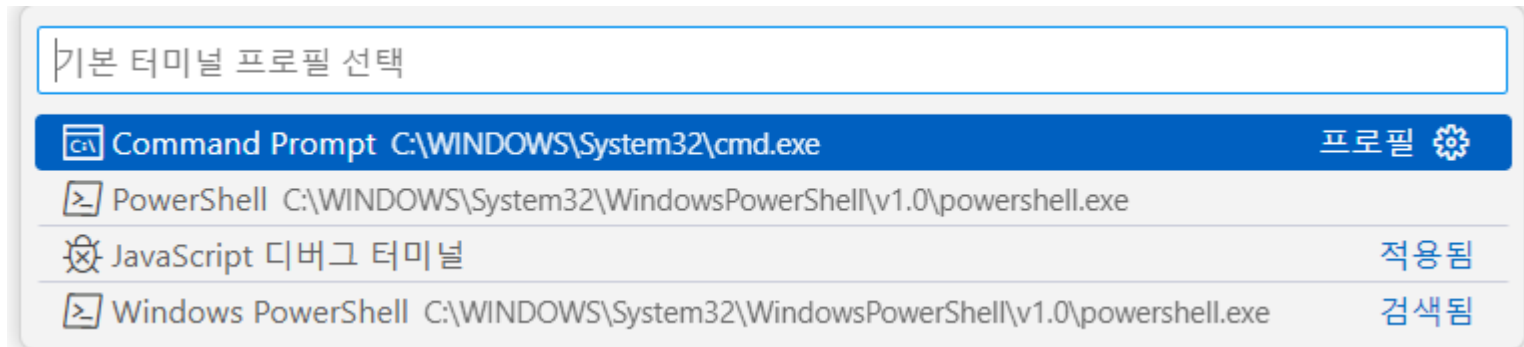
The right pane shows the details for the 'Python' extension (v2023.4.1) by Microsoft. It includes the Python logo, the extension name, version, publisher (Microsoft), and a description: 'IntelliSense (Pylance), Linting, Debugging...'. There is an 'Install' button and a settings gear icon. Below the main description, there are tabs for 'DETAILS', 'FEATURE CONTRIBUTIONS', 'CHANGELOG', and 'EXTENSION PACK'. The 'DETAILS' tab is active, showing the text 'Python extension for Visual Studio Code' and a blue circular button with an upward arrow. On the right side of the details pane, there is a 'Categories' section with a list of categories: 'Programming Languages', 'Linters', 'Debuggers', and 'Formatters'.

# 0. Python Dev Env

- vscode 기본 터미널 변경
  - Ctrl + Shift + P
  - Default Profile 검색 후 엔터

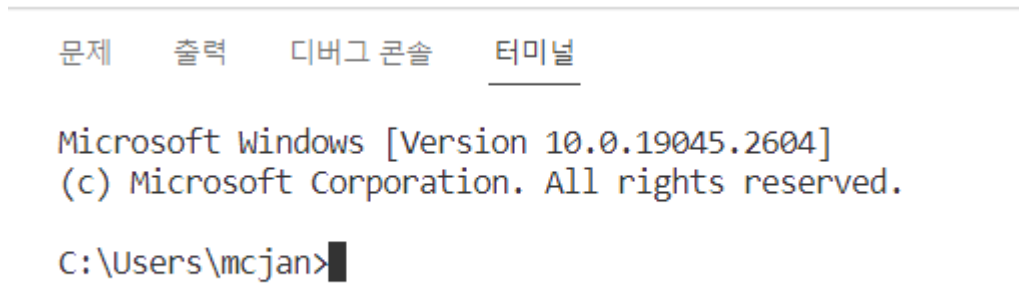


- Command Prompt 선택 후 엔터



# 0. Python Dev Env

- vscode 터미널 열기
  - 터미널(T) → 새 터미널 선택



# 0. Python Dev Env

---

- python 가상환경 (Virtual Environment: venv) 설정
  - 터미널에서 `cd c:\₩` 입력 후 엔터
  - `C:\₩` 에서 `mkdir python_projects` 입력 후 엔터 (폴더 만들기)

문제    출력    디버그 콘솔    터미널

```
c:\>mkdir python_projects█
```

- `cd python_projects` 입력 후 엔터 (폴더 이동)

문제    출력    디버그 콘솔    터미널

```
c:\>cd python_projects█
```

# 0. Python Dev Env

---

- python 가상환경 (Virtual Environment: venv) 설정
  - 가상환경 생성하기
    - `python -m venv python_exam`

문제    출력    디버그 콘솔    터미널

```
c:\python_projects>python -m venv python_exam
```

- python\_exam 폴더로 이동하기

문제    출력    디버그 콘솔    터미널

```
c:\python_projects>cd python_exam
```

# 0. Python Dev Env

---

- python 가상환경 (Virtual Environment: venv) 설정
  - 가상환경 실행하기
    - Scripts/activate.bat

문제    출력    디버그 콘솔    터미널

```
c:\python_projects\python_exam>Scripts\activate.bat
```

- python 가상환경으로 진입한 화면

문제    출력    디버그 콘솔    터미널

```
(python_exam) c:\python_projects\python_exam>
```

# 0. Python Dev Env

---

- python 가상환경 (Virtual Environment: venv) 설정
  - 가상환경 종료하기
    - Scripts/deactivate.bat

문제    출력    디버그 콘솔    터미널

```
(python_exam) c:\python_projects\python_exam>Scripts\deactivate.bat
```

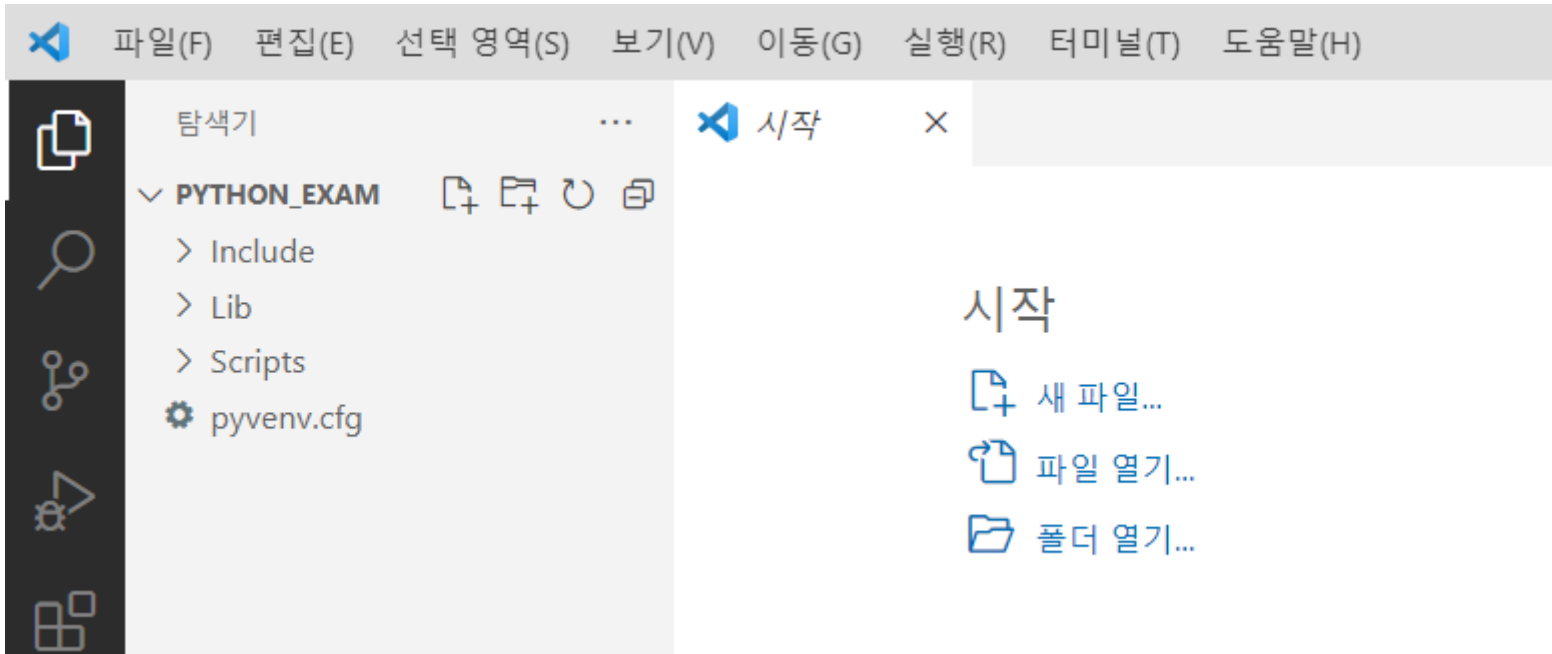
- 다시 가상환경으로 진입 후 아래 명령어 실행
  - code .

문제    출력    디버그 콘솔    터미널

```
(python_exam) C:\python_projects\python_exam>code .
```

# 0. Python Dev Env

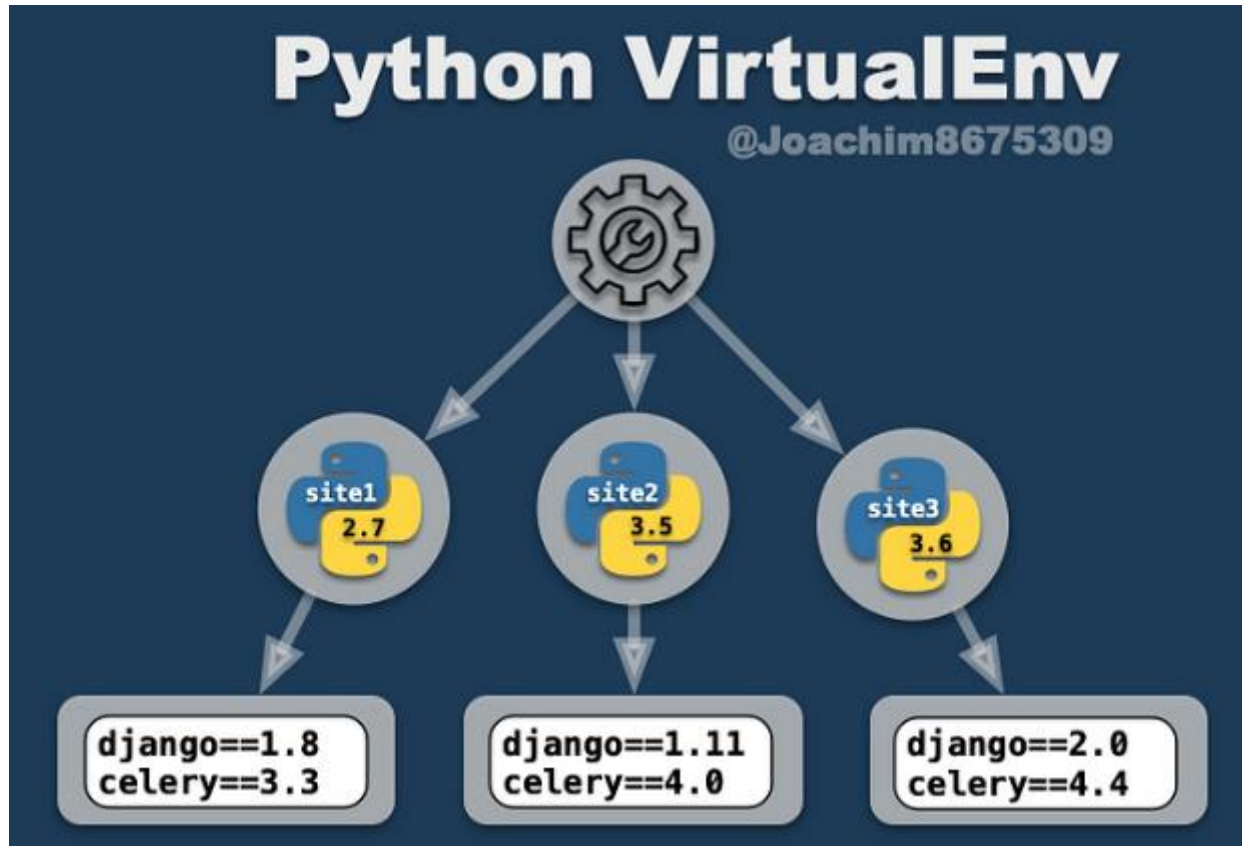
- python 가상환경 (Virtual Environment: venv) 설정
  - 새로운 vscode가 열리면서 폴더가 지정된다.





# 0. Python Dev Env

- python 가상 환경이란?
  - 독립된 환경을 제공하는 일종의 컨테이너이자 워크스페이스



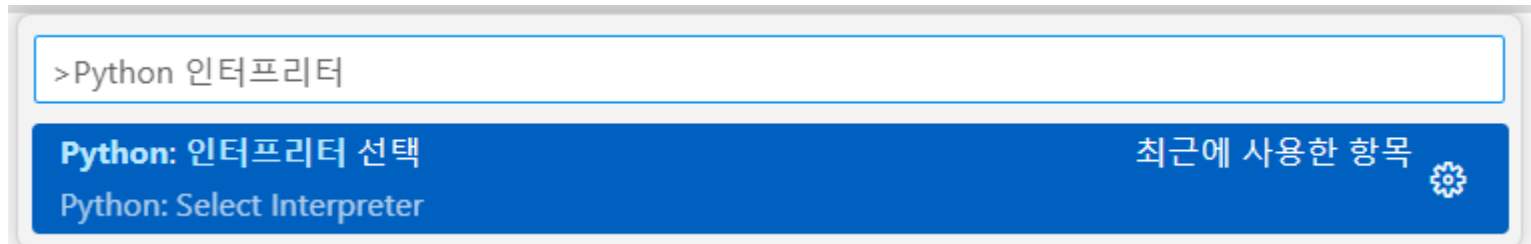
# 0. Python Dev Env

---

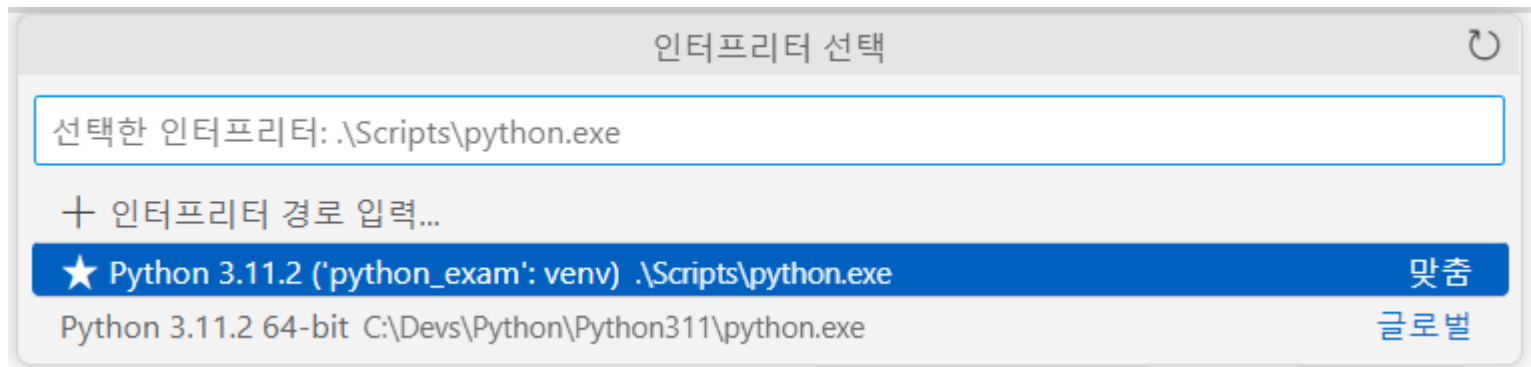
- python 가상 환경이란?
  - 독립된 환경을 제공하는 이유?
    - 1. python app 마다 사용하는 library가 다르다.
      - global 영역에 설치할 경우 충돌 문제 발생
    - 2. python app 마다 사용하는 python의 버전이 다르다.
      - 2020년에 개발하여 운영중인 파이썬의 버전 = 3.2
      - 2021년에 개발하여 운영중인 파이썬의 버전 = 3.4
      - 2022년에 개발하여 운영중인 파이썬의 버전 = 3.7

# 0. Python Dev Env

- python 인터프리터 설정
  - Ctrl + Shift + P
  - Python 인터프리터 검색 후 엔터



- 현재 가상환경의 인터프리터 선택 후 엔터



# 자료형\_1

---

1. 숫자형
2. 문자열
3. 리스트
4. 튜플
5. 딕셔너리

# 1. 숫자형

# 1. 숫자형

---

- 숫자 형태로 이루어진 자료형.
- 10진수 기반의 정수 및 실수와 8, 16진수 등

# 1. 숫자형 – 정수형 (Integer)

---

- 정수를 표현하는 자료형.
- 부호를 이용해 양/음수를 표현.

```
a = 123
b = -178
c = 0
print(a, type(a))
print(b, type(b))
print(c, type(c))
```

---

```
123 <class 'int'>
-178 <class 'int'>
0 <class 'int'>
```

# 1. 숫자형 - 실수형 (Float)

---

- 실수(부동소수점: Floating Point) = 소수점이 포함된 숫자.

```
a = 1.2  
b = 3.45  
print(a, type(a))  
print(b, type(b))
```

---

```
1.2 <class 'float'>  
3.45 <class 'float'>
```



# 1. 숫자형 – 사칙연산

- 숫자형에 사용할 수 있는 연산기호들

연산자	설명
+	숫자를 더함
-	숫자를 뺌
*	숫자를 곱함
/	숫자를 나눔 (소수점 형태로 나뉘어짐)
//	숫자를 나눔 (몫만 구함)
%	숫자를 나눔 (나머지만 구함)
**	숫자를 제곱함

# 1. 숫자형 - 사칙연산

---

- 파이썬 사칙연산 예제

```
a = 3
b = 4
c = a + b
d = a - b
e = a * b
f = a / b
g = a // b
h = a ** b
i = a % b
print(c) # c = a + b
print(d) # d = a - b
print(e) # e = a * b
print(f) # f = a / b
print(g) # g = a // b
print(h) # h = a ** b
print(i) # i = a % b
```

# 1. 숫자형 – 사칙연산

---

- 더해서/빼서/곱해서/나누어서 할당하기

```
a = 1
print(a) # 1
a = a + 1
print(a) # 2
```

- 변수 자신에게 값을 더해서 재할당 하는 경우 단축 표현이 가능

```
a = 3
print(a) # 3
a += 1
print(a) # 4
```

- 타 언어에서 사용하는 ++, -- 표현은 불가.

```
a = 5
a++ # SyntaxError: invalid syntax
a-- # SyntaxError: invalid syntax
++a # SyntaxError: invalid syntax
--a # SyntaxError: invalid syntax
```

# 1. 숫자형 - 사칙연산

연산자	단축형	문법 예제	결과예제 (a 값이 10일 경우)
+	+=	a += 1	11
-	-=	a -= 1	9
*	*=	a *= 2	20
/	/=	a /= 3	3.33333....5
//	//=	a //= 3	3
%	%=	a %= 3	1
**	**=	a **= 2	100

# 1. 숫자형 – 사칙연산 순서

- PEMDAS

우선순위 (낮을수록 높음)	이름	설명
1	Parenthese	괄호 ( )
2	Exponents	제곱 **
3	Multiplication	곱하기 *
3	Division	나누기 /, //, %
4	Addition	더하기 +
4	Subtraction	빼기 -
5	Left to right	그 외 왼쪽에서 오른쪽으로

## 2. 문자형

## 2. 문자형

---

- String
- 문자, 단어 등으로 구성된 문자들의 집합
- 따옴표 (Double quotation mark, Single quotation mark)로 표현
  - 따옴표 안에 있는 모든 것들은 모두 문자.
- e.g.
  - 'This is a pen'
  - "This is an apple"
  - "Apple Pen"
  - "123"
  - " "
  - "A"
  - """"This is multiple line string""""
  - """This is multiple line string too"""

## 2. 문자형 – Notation 활용

---

- 1. 문자열 안에 홑따옴표 (Single quote) 포함하기

```
str = "It's string syntax"  
print(str) # It's string syntax
```

- 문자열 내에 ' 를 표현하기 위해 쌍따옴표 (Double quote) 사용

- 2. 문자열 안에 쌍따옴표 사용

```
str = 'He said: "Python is very easy"'  
print(str) # He said: "Python is very easy"
```

- 문자열 내에 " 를 표현하기 위해 홑따옴표 사용



## 2. 문자형 – Notation 활용

- 3. 문자열 안에 쌍따옴표와 홑따옴표를 모두 사용한다면

```
str = "He said: \"Python's syntax is very very simple\""
print(str) # He said: "Python's syntax is very very simple"
```

- Escape Code를 사용한다.

Code	설명
\n	줄바꿈 표현
\t	탭(간격) 표현
\\	백슬래시 문자 \ 를 표현
\'	홑따옴표 표현
\"	쌍따옴표 표현
\r	커서를 가장 앞으로 이동

## 2. 문자형 – Notation 활용

---

- 4. 여러줄 문자열

```
str = """He said:  
"Python's syntax is very very simple"""  
print(str) # He said:\n"Python's syntax is very very simple
```

```
str = '''He said:  
"Python's syntax is very very simple"'''  
print(str) # He said:\n"Python's syntax is very very simple
```

- 여러줄 문자열에서 내부 쌍따옴표, 홑따옴표는 Escape code 없이 사용 가능.
- 단, \w, \t, \n 등은 Escape code가 반드시 필요.

## 2. 문자형 – 연산

---

- 1. 문자열 더하기
  - Python 문자열은 문자열 간의 더하기를 지원.
  - 문자열은 문자열만 더할 수 있다.
  - "a + b = " + 10 이런 코드는 불가능.

```
str_a = "Hello, "  
str_b = "Python world"  
str_c = str_a + str_b  
print(str_c) # Hello, Python world
```

```
str = "Hello, "  
str += "World!"  
print(str) # Hello, World!
```

## 2. 문자형 – 연산

---

- 2. 문자열 곱하기
  - 문자열을 곱할 경우 Repeat 된다.

```
str = "Python"  
new_str = str + " World" * 3  
print(new_str) # Python World World World
```

```
str = "Python"  
new_str = (str + " World") * 3  
print(new_str) # Python WorldPython WorldPython World
```

- 문자열 연산에도 PEMDAS Rule이 적용된다.

## 2. 문자형 – 길이구하기

---

- len() 함수를 이용해 문자열의 길이를 구할 수 있다.

```
str = "Welcome to Python world. Experience very easy python."  
str_len = len(str)  
# Welcome to Python world. Experience very easy python. 53  
print(str, str_len)
```

## 2. 문자형 – 인덱싱

---

- 문자열의 각 글자들은 인덱스(자리번호)를 가지고 있다.

```
str = "Welcome to Python world. Experience very easy python."  
# Welcome to Python world. Experience very easy python.  
# 0           1           2           3           4           5  
# 01234567890123456789012345678901234567890123456789012
```

- 인덱스 번호를 이용해 하나 이상의 글자를 받아올 수 있다.

```
chr = str[11]  
print(chr) # P
```

## 2. 문자형 – 인덱싱

---

- 파이썬은 역순의 인덱스도 함께 가지고 있다.

```
str = "Welcome to Python world. Experience very easy python."  
# Welcome to Python world. Experience very easy python.  
#      5         4         3         2         1         -  
# 32109876543210987654321098765432109876543210987654321-
```

- 역순 인덱스를 이용하면, 마지막부터 인덱싱이 가능하다.

```
chr = str[-1]  
print(chr) # .  
chr = str[-53]  
print(chr) # W
```

## 2. 문자형 – 인덱싱 조합

---

- 문자열의 첫 번째 단어 추출 해보기

```
word = str[0] + str[1] + str[2] + str[3] + str[4] + str[5] + str[6]  
print(word) # Welcome
```

- 0 ~ 6까지의 인덱스를 모두 탐색할 경우, 매우 번거로움



## 2. 문자형 – 슬라이싱

- 문자열 중 특정 인덱스 부터 지정 인덱스까지 잘라낼 수 있는 기능

```
# word = str[0] + str[1] + str[2] + str[3] + str[4] + str[5] + str[6]
# print(word) # Welcome
word = str[0:7] # 0 ~ 6까지 슬라이싱 함.
print(word)
```

- 문자열[시작인덱스:종료인덱스+1] 로 인덱스를 슬라이싱.

```
# Welcome to Python world. Experience very easy python.
# 0          1          2          3          4          5
# 01234567890123456789012345678901234567890123456789012
```

- Python world (11 ~ 22 인덱스) 추출 예제

```
word = str[11:23]
print(word) # Python world
```

## 2. 문자형 – 슬라이싱

---

- 25번 인덱스부터 끝까지 슬라이싱

```
# Welcome to Python world. Experience very easy python.  
# 0           1           2           3           4           5  
# 0123456789012345678901234567890123456789012
```

- 시작번호만 지정하고 마지막번호는 생략

```
word = str[25:]  
print(word) # Experience very easy  
python.
```

- 0 ~ 23번 슬라이싱

- 시작번호만 생략하고 마지막번호를 지정

```
word = str[:24]  
print(word) # Welcome to Python world.
```

## 2. 문자형 – 슬라이싱

---

- -19 ~ -28 까지 슬라이싱

```
str = "Welcome to Python world. Experience very easy python."  
# Welcome to Python world. Experience very easy python.  
#      5           4           3           2           1           -  
# 32109876543210987654321098765432109876543210987654321-
```

- 역순 슬라이싱 – 작은 숫자부터 큰 숫자로 슬라이싱

```
word = str[-28:-18]  
print(word) # Welcome to Python world.
```

## 2. 문자형 – 슬라이싱

---

- -47 ~ 0 까지 슬라이싱

```
str = "Welcome to Python world. Experience very easy python."  
# Welcome to Python world. Experience very easy python.  
#      5          4          3          2          1          -  
# 32109876543210987654321098765432109876543210987654321-  
# 0          1          2          3          4          5  
# 01234567890123456789012345678901234567890123456789012  
  
word = str[-47:0]  
print(word) # Welcome
```

## 2. 문자형 – 슬라이싱

---

- 실습예제

```
datetime = "20230331000135" # yyyyMMddHHmmss
```

```
year =
```

```
month =
```

```
day =
```

```
hour =
```

```
minute =
```

```
second =
```

```
# datetime 에서 연, 월, 일, 시, 분, 초를 각 변수에 넣고 출력
```

## 2. 문자형 – 함수

---

- 문자 개수 세기
  - 문자열에서 지정한 문자가 몇 개 있는지 조회

```
str = "Welcome to Python world. Experience very easy python."  
e_cnt = str.count("e");  
print(e_cnt)
```

- 대/소문자 변경하기

```
str = "Welcome to Python world. Experience very easy python."  
lower_str = str.lower()  
# welcome to python world. experience very easy python.  
print(lower_str)
```

```
upper_str = str.upper()  
# WELCOME TO PYTHON WORLD. EXPERIENCE VERY EASY PYTHON.  
print(upper_str)
```

## 2. 문자형 – 함수

---

- 인덱스 찾기

- 문자열에서 지정한 문자가 몇 번 인덱스에 있는지 조회

- 1. find

```
str = "Welcome to Python world. Experience very easy python."  
to_idx = str.find("to")  
print(to_idx) # 8
```

```
hard_idx = str.find("hard")  
print(hard_idx) # -1
```

- 2. index

```
str = "Welcome to Python world. Experience very easy python."  
to_idx = str.index("to")  
print(to_idx) # 8
```

```
hard_idx = str.index("hard")  
print(hard_idx) # ValueError: substring not found
```

## 2. 문자형 – 함수

---

- 인덱스 찾기

- 문자열에서 지정한 문자가 몇 번 인덱스에 있는지 조회
- 가장 마지막 인덱스를 찾는다.
- 3. rfind

```
str = "Welcome to Python world. Experience very easy python."  
to_idx = str.find("thon")  
print(to_idx) # 13
```

```
hard_idx = str.rfind("thon")  
print(hard_idx) # 48
```

- 4. rindex

```
str = "Welcome to Python world. Experience very easy python."  
to_idx = str.index("thon")  
print(to_idx) # 13
```

```
hard_idx = str.rindex("thon")  
print(hard_idx) # 48
```



## 2. 문자형 – 함수

---

- 문자열 연결하기
  - 글자들을 "," 로 연결하기

```
str = "abcd"  
join_str = ",".join(str)  
print(join_str) # a,b,c,d
```

- 문자열 수정하기
  - most difficult 를 easiest 로 수정하기

```
str = "Python is the most difficult computer language."  
str = str.replace("most difficult", "easiest")  
print(str) # Python is the easiest computer language.
```

## 2. 문자형 – 함수

---

- 공백 제거하기

```
str = " hello? "  
print(str, len(str)) # hello? 8  
lstrip_str = str.lstrip() # 왼쪽공백 제거  
print(lstrip_str, len(lstrip_str)) # hello? 7  
  
rstrip_str = str.rstrip() # 오른쪽공백 제거  
print(rstrip_str, len(rstrip_str)) # hello? 7  
  
strip_str = str.strip() # 양쪽공백 제거  
print(strip_str, len(strip_str)) # hello? 6
```

## 2. 문자형 – 포메팅

---

- Python 문자열을 만들기 위한 강력한 수단.
- % 바인딩, format 바인딩, f-string, r-string 이 존재.

## 2. 문자형 - % 바인딩

- 바인딩 포맷에 따라 값을 지정.
- 정해진 문자열 코드를 따라야 한다.

코드	설명
%s	문자열 (String)
%c	문자 1개 (Character)
%d	정수 (Integer)
%f	부동소수(float)
%.nf	소수점을 n개 만큼 보여줌 (e.g. %.2f)
%l.nf	소수점을 n개 만큼 보여주고 총 문자열 길이를 l개로 지정 e.g. >>> print("%5.4f" % 3.112321341231235) # 3.1123 e.g. >>> print("%1.4f" % 3.112321341231235) # 3.1123 # l이 n보다 작을 경우 l값은 무시하고 n만 처리한다.
%o	8진수
%x	16진수
%%	% 문자

## 2. 문자형 - % 바인딩

---

- 바인딩 예제 (%s)

```
format_str = "%s 는 %s를 바인딩합니다." % "문자열"  
print(format_str) # %s 는 문자열을 바인딩합니다.
```

```
name = "홍길동"  
format_str = "내 이름은 %s 입니다." % name  
print(format_str) # 내 이름은 홍길동 입니다.
```

```
format_str = "제가 키우는 %s의 이름은 %s 입니다." % ("강아지", "해피")  
print(format_str) # 제가 키우는 강아지의 이름은 해피 입니다.
```

## 2. 문자형 - % 바인딩

---

- 바인딩 예제 (%d)

# 정수 대입

```
format_str = "성인나이 기준은 만 %d세 입니다." % 19
print(format_str) # 성인나이 기준은 만 19세 입니다.
```

# 계산식 대입\_1

```
format_str = "2 X 4 = %d" % 2 * 4
print(format_str) # 2 X 4 = 22 X 4 = 22 X 4 = 22 X 4 = 2
```

# 계산식 대입\_2

```
format_str = "2 X 4 = %d" % (2 * 4)
print(format_str) # 2 X 4 = 8
```

# 여러 개의 포맷 대입

```
format_str = "구구단: %d X %d = %d" % (2, 4, 2 * 4)
print(format_str) # 구구단: 2 X 4 = 8
```

## 2. 문자형 - % 바인딩

---

- 계산식 대입\_1 의 결과 분석

# 계산식 대입\_1

```
format_str = "2 X 4 = %d" % 2 * 4
```

```
print(format_str) # 2 X 4 = 22 X 4 = 22 X 4 = 22 X 4 = 2
```

# 1. "2 X 4 = %d" % 2 를 대입

# 2. "2 X 4 = 2"

# 3. "2 X 4 = 2" \* 4

# 4. "2 X 4 = 2" 가 4번 반복됨.

## 2. 문자형 - % 바인딩

- 바인딩 예제 (%f, %.nf)

```
format_str = "PI 는 %f 입니다." % 3.14159265359  
print(format_str) # PI 는 3.141593 입니다.
```

```
format_str = "PI 는 %.11f 입니다." % 3.14159265359  
print(format_str) # PI 는 3.14159265359 입니다.
```

```
format_str = "%f X %f = %f" % (10.5, 55.8, 10.5 * 55.8)  
print(format_str) # 10.500000 X 55.800000 = 585.900000
```

```
format_str = "%.2f X %.2f = %.2f" % (10.5, 55.8, 10.5 * 55.8)  
print(format_str) # 10.50 X 55.80 = 585.90
```

```
format_str = "%.0f X %.0f = %.0f" % (10.5, 55.8, 10.5 * 55.8)  
print(format_str) # 10 X 56 = 586
```

```
format_str = "%d X %d = %d" % (10.5, 55.8, 10.5 * 55.8)  
print(format_str) # 10 X 55 = 585
```



## 2. 문자형 - % 바인딩

---

- 바인딩 예제 (복합 바인딩)

```
format_str = "%s 의 값은 %f 입니다. 보통 %.2f 로 알고있습니다." % \
    ("원주율(pi)", 3.14159265359, 3.14159265359)
print(format_str)
# 원주율(pi) 의 값은 3.141593 입니다. 보통 3.14 로 알고있습니다.
```

## 2. 문자형 – format 바인딩

- 바인딩 예제 ( {} 바인딩 ) – 자릿수에 맞춰서 바인딩

```
f_str = "2 X {} = 16".format(8)
print(f_str) # 2 X 8 = 16
```

```
f_str = "{} X {} = {}".format(2, 8, 2*8)
print(f_str) # 2 X 8 = 16
```

```
f_str = "{0} X {1} = {2}".format(2, 8, 2 * 8)
print(f_str) # 2 X 8 = 16
```

```
f_str = "{1} X {0} = {2}".format(2, 8, 2 * 8)
print(f_str) # 8 X 2 = 16
```

```
f_str = "{2} X {0} = {1}".format(2, 8, 2 * 8)
print(f_str) # 16 X 2 = 8
```

```
f_str = "{0} X {0} = {0}".format(2, 8, 2 * 8)
print(f_str) # 2 X 2 = 2
```

## 2. 문자형 – format 바인딩

---

- 바인딩 예제 ( {} 바인딩 ) – 자릿수에 맞춰서 바인딩

```
f_str = "{0}는 {1:.4f} 입니다. 보통 {1:.2f} 로 알고있습니다."\  
        .format("파이", 3.14159265359)  
print(f_str) # 파이는 3.1416 입니다. 보통 3.14 로 알고있습니다.
```

## 2. 문자형 – format 바인딩

- 바인딩 예제 ( {} 바인딩 ) – 이름에 맞춰 바인딩

```
f_str = "{pi}는 {pi_num:.4f} 입니다. 보통 \  
{pi_num:.2f} 로 알고있습니다."\  
        .format(pi="파이", pi_num=3.14159265359)  
print(f_str) # 파이는 3.1416 입니다. 보통 3.14 로 알고있습니다.
```

# 순서가 상관없다.

```
f_str = "{pi}는 {pi_num:.4f} 입니다. 보통 \  
{pi_num:.2f} 로 알고있습니다."\  
        .format(pi_num=3.14159265359, pi="파이")  
print(f_str) # 파이는 3.1416 입니다. 보통 3.14 로 알고있습니다.
```

## 2. 문자형 – fstring

- 바인딩 예제 ( {변수명} ) – 변수명에 맞춰 바인딩

```
name = "파이"
pi = 3.14159265359
f_str = f"{name}는 {pi}입니다. 보통 {pi:.2f}로 알고있습니다."
print(f_str) # 파이는 3.14159265359입니다. 보통 3.14로 알고있습니다.
```

```
num_a = 10
num_b = 15
f_str = f"{num_a} + {num_b} = {num_a + num_b}"
print(f_str) # 10 + 15 = 25
```

```
age = 39
f_str = f"올해 {age}살 입니다. 내년엔 {age+1}살이 됩니다."
print(f_str) # 올해 39살 입니다. 내년엔 40살이 됩니다.
```

```
dic = {name: "홍길동", age: 30} # 딕셔너리
f_str = f"이름: {dic[name]}, 나이: {dic[age]}"
print(f_str) # 이름: 홍길동, 나이: 30
```

### 3. 리스트

### 3. 리스트

---

- 관련된 값들의 집합 = 배열과 유사
- e.g. OO고등학교 O학년 O반  
모든 학생의 점수의 합을 구하기 위해 점수 변수를 만든다.
  - `stud_1 = 100`
  - `stud_2 = 70`
  - `stud_3 = 60`
  - ...
  - `stud_10 = 70`
  - `sum = stud_1 + stud_2 + ... + stud_10`
  - 불필요하게 긴 코드 생성.

### 3. 리스트

---

- list로 풀어보기.

```
# 10명 학생의 점수를 list로 생성
```

```
scores = [100, 90, 70, 60, 50, 60, 77, 95, 30, 65]  
print(scores)
```

```
# 합계
```

```
sum_scores = sum(scores)  
print(sum_scores) # 697
```



### 3. 리스트

---

- list는 대괄호 [ ] 로 정의

```
scores = [100, 90, 70, 60, 50, 60, 77, 95, 30, 65]
```

- 대괄호 안에 데이터를 콤마 , 로 구분하여 넣는다.
- 타입과 상관없이 모든 값을 넣을 수 있다.

```
scores = ["백점", 90, 98.5, [10.5, 55.6], "빵점", "결석", True]
```

- 그러나 수월한 계산을 위해 동일한 타입으로 맞춘다.

```
scores = [100, 90, 55, 98, 0, 0]
```

# 3. 리스트

---

- 리스트는 시퀀스 타입
- 시퀀스 타입?
  - 1. 순서가 유지된다.
  - 2. 정수로 인덱싱한다.
  - 3. 길이가 있다.
  - 4. 순서가 유지되며, 정수로 인덱싱하고 길이가 있으니
    - 반복이 가능(Iterable, Repeat) 하다.
  - 5. 순서가 유지되고 정수로 인덱싱을 할 수 있으니
    - 슬라이싱이 가능하다.
  - 6. 시퀀스를 확장(+) 및 축소(sum, max, min 등)가 가능하다.
- 파이썬의 시퀀스 타입은 "문자열", "리스트", "튜플" 이 존재.

```
lst = list("Hello, world!") # 문자열을 리스트로 변경
print(lst)
```

# 3. 리스트

---

- 인덱싱 (문자열 인덱스와 동일)

```
lst = [1, 2, 3]
print(lst[0]) # 1
print(lst[1]) # 2
print(lst[2]) # 3
```

```
lst = [1, 2, 3]
print(lst[-1]) # 3
print(lst[-2]) # 2
print(lst[-3]) # 1
```

- 슬라이싱 (문자열 슬라이싱과 동일)

```
lst = [1, 2, 3, 4, 5, 6, 7]
sub_lst = lst[:3]
print(sub_lst) # [1, 2, 3]
```

```
sub_lst = lst[2:3]
print(sub_lst) # [3]
```

```
sub_lst = lst[-3:-1]
print(sub_lst) # [5, 6]
```

# 3. 리스트

---

- 길이 (문자열 길이구하기와 동일)

```
lst = [1, 2, 3, 4, 5, 6, 7]
lst_len = len(lst)
print(lst_len) # 7
```

```
lst = []
lst_len = len(lst)
print(lst_len) # 0
```

- 최대값, 최소값 구하기 (문자열 최대값, 최소값과 동일)

```
lst = [1, 2, 3, 4, 5, 6, 7]
```

```
max_val = max(lst)
min_val = min(lst)
sum_val = sum(lst)
```

```
print(max_val) # 7
print(min_val) # 1
print(sum_val) # 28
```

### 3. 리스트

---

- 리스트 반복 (문자열 반복과 동일)

```
lst = [1, 2, 3]
lst_2 = lst * 3
print(lst) # [1, 2, 3]
print(lst_2) # [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- 리스트 더하기 (문자열 더하기와 동일)

```
lst = [1, 2, 3]
lst_2 = [4, 5, 6] + lst
print(lst) # [1, 2, 3]
print(lst_2) # [4, 5, 6, 1, 2, 3]
```

# 3. 리스트

---

- 리스트 수정

```
lst = [1, 2, 3]
lst[1] = 4
print(lst) # [1, 4, 3]
```

```
lst = [1, 2, 3]
lst[3] = 4
print(lst) # IndexError
```

- 리스트 삭제

```
lst = [1, 2, 3]
del lst[0]
print(lst) # [2, 3]
```

```
lst = [1, 2, 3]
del lst[3]
print(lst) # IndexError
```

### 3. 리스트 – 리스트 내장함수

---

- 추가 list.append()

```
lst = [1, 2, 3]
lst.append(4)
print(lst) # [1, 2, 3, 4]
```

```
lst = [1, 2, 3]
lst.append("넷") # Warning (int list에 str을 넣으려함)
print(lst) # [1, 2, 3, '넷']
```

- 정렬 list.sort(), sorted(list)

```
lst = [5,6,1,8,4,2,7,9]
lst.sort() # 원본을 정렬
print(lst) # [1, 2, 4, 5, 6, 7, 8, 9]
```

```
lst = [5,6,1,8,4,2,7,9]
sorted_lst = sorted(lst) # 원본은 유지. 정렬된 리스트를 복제
print(lst) # [5, 6, 1, 8, 4, 2, 7, 9]
print(sorted_lst) #[1, 2, 4, 5, 6, 7, 8, 9]
```

### 3. 리스트 – 리스트 내장함수

- 역정렬 `list.reverse()`, `reversed(list)`

```
lst = [5,6,1,8,4,2,7,9]
lst.reverse() # 원본을 역정렬 (내림차순아님.)
print(lst) # [5, 6, 1, 8, 4, 2, 7, 9]
```

```
lst = [5,6,1,8,4,2,7,9]
sorted_lst = reversed(lst) # 원본은 유지
                        # list_reverseiterator 객체가 생성됨.
print(lst) # [5, 6, 1, 8, 4, 2, 7, 9]
print(sorted_lst) # <list_reverseiterator object at 0x...>
sorted_lst = list(sorted_lst) # list_reverseiterator를 list로 변환
print(sorted_lst) # [9, 7, 2, 4, 8, 1, 6, 5]
```

- 인덱스 찾기 `index()`

```
str_lst = list("hello, python list")
print(str_lst) # ['h', 'e', 'l', ... 'i', 's', 't']
p_index = str_lst.index("p")
print(p_index) # 7
```



### 3. 리스트 – 리스트 내장함수

---

- 삽입 insert()

```
lst = [2, 3, 4]
lst.insert(0, 1) # 0 인덱스에 1 추가
print(lst) # [1, 2, 3, 4]
```

- 제거 remove()

```
lst = [1, 2, 3, 4]
# lst.remove(0) # ValueError: list.remove(x): x not in list
lst.remove(1) # Value로 삭제. del 은 index로 삭제
print(lst) # [2, 3, 4]
```

```
lst = [1, 2, 3, 4, 1, 2, 3]
lst.remove(1) # 값이 중복될 경우, 첫 번째 1 만 삭제
print(lst) # [2, 3, 4, 1, 2, 3]
```

### 3. 리스트 – 리스트 내장함수

---

- 제거 후 반환 pop()

```
lst = [1, 2, 3, 4]
item = lst.pop(0) # 0번 인덱스값을 item에 할당 한 후
                  # 해당 요소를 리스트에서 삭제
print(item) # 1
print(lst) # [2, 3, 4]
```

```
lst = [1, 2, 3, 4]
item = lst.pop(2) # 2번 인덱스값을 item에 할당 한 후
                  # 해당 요소를 리스트에서 삭제
print(item) # 3
print(lst) # [1, 2, 4]
```

```
lst = [1, 2, 3, 4]
item = lst.pop(5) # IndexError: pop index out of range
```

### 3. 리스트 – 리스트 내장함수

---

- 중복요소 개수 세기 `count()`

```
str_lst = list("hello, python list")  
print(str_lst) # ['h', 'e', 'l', ... 'i', 's', 't']
```

```
l_count = str_lst.count("l")  
print(l_count) # 3
```

- 확장 `extend()` (+ 연산의 함수형)

```
lst = [1, 2, 3]  
lst.extend([4, 5, 6]) # list만 넣을 수 있다.  
print(lst) # [1, 2, 3, 4, 5, 6]
```

### 3. 리스트 - 시퀀스 특수 기능

---

- 시퀀스는 값을 분리 할당이 가능하다.

```
str1, str2 = "AB"  
print(str1) # A  
print(str2) # B
```

```
a, b = [1, 2]  
print(a) # 1  
print(b) # 2
```

- 여러 개의 값을 분리 할당하려면 \* 기호를 사용한다.

```
a, *b, c = [1, 2, 3, 4, 5, 6, 7]  
print(a) # 1  
print(b) # [2, 3, 4, 5, 6]  
print(c) # 7
```

## 4. 튜플

## 4. 튜플

---

- list와 매우 유사.
- 변경 불가능한 `list == tuple`
  - 값이 할당된 이후 값의 추가/수정/삭제가 불가능하다.
- 튜플은 시퀀스 타입
  - 순서 유지
  - 인덱싱 가능
  - 슬라이싱 가능
  - 반복 가능
  - 확장 및 축소 가능

## 4. 튜플

---

- 튜플은 ( ) 괄호로 표현.
- 값이 , 콤마로 구분되며 괄호는 생략가능하다.

```
tp = ()  
tp1 = (1)  
tp2 = (1, 2, 3)  
tp3 = 1, 2, 3  
tp4 = ("a", "b", "c", (1, 2))  
tp5 = tuple([1, 2, 3, 4, 5])
```

- 튜플은 생성되면 추가/수정/삭제가 불가능
- 상수처럼 사용하려면 tuple, 변경 가능성이 있다면 list 를 사용

## 4. 튜플

---

- 튜플 = 시퀀스 = 분리할당이 가능.

```
a, b = (1, 2)
print(a) # 1
print(b) # 2
```

```
a, b = 3, 4
print(a) # 3
print(b) # 4
```

```
a, *b, c = (1, 2, 3, 4, 5, 6, 7)
print(a) # 1
print(b) # [2, 3, 4, 5, 6]
print(c) # 7
```



## 4. 튜플

---

- 튜플은 변경이 불가능.

```
tp = (1, 2)
```

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

```
tp[0] = 1
```

```
# TypeError: 'tuple' object doesn't support item deletion
```

```
del tp[0]
```

## 4. 튜플

---

- 튜플 시퀀스 기능

- 인덱싱

```
tp1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
item = tp1[0]
print(item)
```

```
item = tp1[10] # IndexError: tuple index out of range
```

- 슬라이싱

```
tp1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
tp2 = tp1[1:5]
print(tp2) # (2, 3, 4, 5)
```

## 4. 튜플

---

- 확장

```
tp1 = (1, 2, 3, 4)
tp2 = (5, 6, 7)
tp3 = tp1 + tp2
print(tp3) # (1, 2, 3, 4, 5, 6, 7)
```

- 반복

```
tp1 = (1, 2, 3, 4)
tp2 = tp1 * 3
print(tp2) # (1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)
```

- 길이

```
tp1 = (1, 2, 3, 4)
print(len(tp1)) # 4
```

## 5. 덕셔너리

## 5. 딕셔너리

---

- Key: Value 쌍의 데이터 구조
- { } 중괄호로 데이터를 표현한다.
- { key: value, key1: value2, ... }
- key 혹은 value에는 어떠한 값이라도 들어갈 수 있다.
  - key는 참조하기 쉽도록 고유한 값으로 작성한다.
- 동일한 key는 작성할 수 없다.

## 5. 딕셔너리

- 아래와 같은 데이터를 dictionary로 표현하면.

Key	Value
name	"홍길동"
age	30
tall	180.5
address	한국
location	["동", "서", "남", "북"]
target	("탐관오리", "부자")

```
data = {  
    "name": "홍길동",  
    "age": 30,  
    "tall": 180.5,  
    "address": "한국",  
    "location": ["동", "서", "남", "북"],  
    "target": ("탐관오리", "부자")}
```

# 5. 딕셔너리

---

- 딕셔너리 k:v 추가하기

```
data = {"name": "홍길동",  
        "age": 30,  
        "tall": 180.5}
```

```
data["address"] = "한국"  
# {'name': '홍길동', 'age': 30, 'tall': 180.5, 'address': '한국'}  
print(data)
```

- 똑같은 key를 추가하면 데이터가 수정된다.

```
data = {"name": "홍길동",  
        "age": 30,  
        "tall": 180.5}
```

```
data["name"] = "한국"  
# {'name': '한국', 'age': 30, 'tall': 180.5}  
print(data)
```

## 5. 딕셔너리

---

- 딕셔너리에 딕셔너리 추가.
  - 딕셔너리의 value는 아무것이나 넣을 수 있으므로 딕셔너리를 추가할 수도 있다.

```
data = {"name": "홍길동",  
        "age": 30,  
        "tall": 180.5}
```

```
data["target"] = {"name": "둘리",  
                  "age": 1000000000}  
#{'name': '홍길동', 'age': 30, 'tall': 180.5, 'target':  
  {'name': '둘리', 'age': 1000000000}}  
print(data)
```



# 5. 딕셔너리

---

- 딕셔너리 k:v 삭제

```
data = {"name": "홍길동",  
        "age": 30,  
        "tall": 180.5}
```

```
del data["tall"]  
print(data) # {'name': '홍길동', 'age': 30}
```

- 딕셔너리 데이터 모두 삭제

```
data = {"name": "홍길동",  
        "age": 30}  
print(data) # {'name': '홍길동', 'age': 30}
```

```
data.clear() # list에도 clear()함수가 있다.  
print(data) # {}
```

# 5. 딕셔너리

---

- 딕셔너리 k:v 참조

```
member = {"id": "admin", "role": "administrator"}
member_id = member["id"]
member_role = member["role"]
print(member_id) # admin
print(member_role) # administrator
print(member["password"]) # KeyError: 'password'
```

- 리스트의 딕셔너리 참조 예제

```
members = [
    {"id": "admin", "role": "administrator"},
    {"id": "root", "role": "super administrator"},
    {"id": "user", "role": "user"}
]
member = members[0] # {"id": "admin", "role": "administrator"}
member_id = member["id"]
print(member_id) # admin
```

## 5. 딕셔너리 - 함수

---

- 모든 key 조회하기 keys()

```
member = {"id": "admin", "role": "administrator", "auth":  
          ["member", "system", "general"]}  
  
member_key = member.keys()  
member_key_lst = list(member_key) # 리스트로 변경 가능  
print(member_key) # dict_keys(['id', 'role', 'auth'])  
print(member_key_lst) # ['id', 'role', 'auth']
```

# 5. 딕셔너리 - 함수

---

- dict\_keys 알아보기

```
member_key = member.keys()
#[ '__and__', ... '__iter__', ... 'mapping' ]
print(dir(member_key))
```

- \_\_iter\_\_

- 반복을 가능하게 해주는 Special method

```
member_key = member.keys()
print( "id" in member_key ) # True
for k in member_key: # 반복문. 나중에 학습하는 내용.
    print(k, member[k])
```

- \_\_이름\_\_

- Special method, Magic method, dunder

## 5. 딕셔너리 - 함수

---

- 모든 value 조회하기 values()

```
member_value = member.values()  
member_value_lst = list(member_value)
```

```
# dict_values(['admin', 'administrator', ['member', 'system',  
'general']])  
print(member_value)  
# ['admin', 'administrator', ['member', 'system', 'general']]  
print(member_value_lst)
```

- dict\_values도 \_\_iter\_\_ 가 구현된 객체 == 반복이 가능하다.

```
for v in member_value:  
    print(v)
```

## 5. 딕셔너리 - 함수

---

- 모든 k:v 쌍 조회하기 items()

```
member = {"id": "admin", "role": "administrator", "auth":  
          ["member", "system", "general"]}  
member_item = member.items()  
member_item_lst = list(member_item)  
# dict_items([('id', 'admin'), ('role', 'administrator'), ('auth',  
['member', 'system', 'general'])])  
print(member_item)  
# [('id', 'admin'), ('role', 'administrator'), ('auth', ['member',  
'system', 'general'])]  
print(member_item_lst)
```

- items는 k:v 쌍을 tuple로 변환해 보여준다.
- dict\_items도 \_\_iter\_\_를 구현한 객체 == 반복이 가능하다.

## 5. 딕셔너리 - 함수

---

- 모든 k:v 쌍 제거하기 clear()

```
member = {"id": "admin", "role": "administrator"}  
print(member) # {'id': 'admin', 'role': 'administrator'}  
member.clear()  
print(member) # {}
```

## 5. 딕셔너리 - 함수

---

- key 로 value 찾기 get(key)

```
member = {"id": "admin", "role": "administrator"}  
item = member.get("id")  
print(item) # admin  
item = member.get("password")  
print(item) # None (Null)  
item = member.get("password", "없음")  
print(item) # 없음
```

- dict["없는key"] 로 찾을 때 에러가 발생.
- dict.get("없는Key") 로 찾을 때엔 None을 반환함.



## 5. 딕셔너리 - 함수

---

- 딕셔너리에 Key가 이미 있는지 확인하기 in

```
dct = {"id": "admin", "role": "administrator"}  
print( "id" in dct ) # True  
print( "password" in dct ) # False
```

- 리스트나 튜플에도 동일한 연산이 있다.

```
lst = [1, 2, 3, 4, 5]  
print( 3 in lst ) # True  
print( 6 in lst ) # False
```

```
tp = (1, 2, 3, 4, 5)  
print( 3 in tp ) # True  
print( 6 in tp ) # False
```

# 자료형\_2

---

- 6. 집합 (set)
- 7. 불린
- 8. 변수
- 9. 연습문제

## 6. 집합 (set)

## 6. 집합 (set)

---

- 집합과 관련된 데이터자료형
- `set( )` 을 이용해 만들거나 `{ }` 를 사용한다.

```
set1 = set([1,2,3,4,5])  
print(set1) # {1, 2, 3, 4, 5}
```

```
set2 = {1,2,3,4,5}  
print(set2) # {1, 2, 3, 4, 5}
```

## 6. 집합 (set)

---

- Set 타입의 특징

- 중복 엘리먼트의 제거

- 순서가 없다.

```
set1 = set("hello")  
print(set1) # {'e', 'o', 'l', 'h'}
```

```
set2 = set("HeLlO")  
print(set2) # {'O', 'l', 'e', 'L', 'H'}
```

- 인덱싱을 할 수 없다 == 슬라이싱이 불가능하다.

```
set1 = {1, 2, 3}  
print(set1)  
print(set1[0]) # TypeError: 'set' object is not subscriptable
```

- 길이는 구할 수 있다.

```
set1 = {1, 2, 3}  
print(len(set1)) # 3
```

## 6. 집합 (set)

---

- Set 타입의 특징
  - 순서가 없고, 정수형 인덱싱이 없으므로
  - Set 타입은 시퀀스가 아니다!
- 오로지 집합 처리만을 위한 데이터 타입.
  - 교집합, 합집합, 차집합을 처리할 수 있다.

## 6. 집합 (set)

---

- 교집합

```
set1 = {1, 2, 3, 4, 5, 6}
set2 = {4, 5, 6, 7, 8, 9}
set3 = set1 & set2
print(set3) # {4, 5, 6}
```

```
set3 = set1.intersection(set2) # & 연산의 함수형
print(set3) # {4, 5, 6}
```

- 합집합

```
set1 = {1, 2, 3, 4, 5, 6}
set2 = {4, 5, 6, 7, 8, 9}
set3 = set1 | set2
print(set3) # {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
set3 = set1.union(set2) # | 연산의 함수형
print(set3) # {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

## 6. 집합 (set)

---

- 차집합

```
set1 = {1, 2, 3, 4, 5, 6}
set2 = {4, 5, 6, 7, 8, 9}
set3 = set1 - set2 # s1 에서 s2와 같은 값을 제거
print(set3) # {1, 2, 3}
```

```
set3 = set1.difference(set2) # - 연산의 함수형
print(set3) # {1, 2, 3}
```

```
set1 = {1, 2, 3, 4, 5, 6}
set2 = {4, 5, 6, 7, 8, 9}
set3 = set2 - set1 # s2 에서 s1와 같은 값을 제거
print(set3) # {8, 9, 7}
```

```
set3 = set2.difference(set1) # - 연산의 함수형
print(set3) # {8, 9, 7}
```



## 6. 집합 (set) - 함수

---

- 1개 값 추가 (add())

```
set1 = {1, 2, 3}
set1.add(4)
print(set1) # {1, 2, 3, 4}
```

- 여러 값 추가 (update())

```
set1 = {1, 2, 3}
set1.update({4, 5})
print(set1) # {1, 2, 3, 4, 5}
```

- 특정 값 제거 (remove())

```
set1 = {1, 2, 3}
# set1.remove(4) # KeyError: 4
set1.remove(2)
print(set1) # {1, 3}
```

## 7. 불린

# 7. 불린

---

- 참(True), 거짓(False) 를 표현하는 자료형
  - True / False 값만 가질 수 있음.

```
a_true = True
b_false = False
print(a_true, type(a_true)) # True <class 'bool'>
print(b_false, type(b_false)) # False <class 'bool'>
```

- 비교연산의 결과는 불린.

```
is_true = 1 == 1
print(is_true, type(is_true)) # True <class 'bool'>

is_false = "안녕하세요" == "안녕?"
print(is_false, type(is_false)) # False <class 'bool'>
```

# 7. 불린

---

- 다양한 데이터는 bool() 함수를 통해 불린으로 캐스팅(형변환) 가능

```
print(bool(0)) # False
print(bool("")) # False
print(bool([])) # False
print(bool({})) # False
print(bool(())) # False
print(bool(1)) # True
print(bool("1")) # True
print(bool([1])) # True
print(bool({2})) # True
print(bool({"a": "b"})) # True
print(bool((3))) # True
```

- 데이터가 있으면 True, 없으면 False로 변환.

## 8. 변수

## 8. 변수

---

- 값이 저장(할당)된 메모리 주소를 개발자가 알기 쉽게 별칭을 부여한 것
- 메모리?
  - 컴퓨터가 프로그램에서 사용하는 데이터를 기억하는 공간

## 8. 변수

---

- 메모리 구조의 예

	0	1	2	3	4
0	0, 0	0, 1	0, 2	0, 3	0, 4
1	1, 0	1, 1	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	2, 3	2, 4
3	3, 0	3, 1	3, 2	3, 3	3, 4
4	4, 0	4, 1	4, 2	4, 3	4, 4

## 8. 변수

- 메모리 공간 중 비어있는 곳을 임의로 선택하여 데이터를 저장.

	0	1	2	3	4
0	0, 0	0, 1	0, 2	0, 3	<b>True</b>
1	1, 0	"이름"	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	2, 3	2, 4
3	3, 0	3, 1	3, 2	<b>30</b>	3, 4
4	4, 0	4, 1	4, 2	4, 3	4, 4

- True 데이터의 주소 = 0, 4
- "이름" 데이터의 주소 = 1, 1
- 30 데이터의 주소 = 3, 3



## 8. 변수

	0	1	2	3	4
0	0, 0	0, 1	0, 2	0, 3	<b>True</b>
1	1, 0	<b>"이름"</b>	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	2, 3	2, 4
3	3, 0	3, 1	3, 2	<b>30</b>	3, 4
4	4, 0	4, 1	4, 2	4, 3	4, 4

- 0, 4 주소의 별칭을 is\_true로 지정.

```
is_true = True # 0, 4
```

- 1, 1 주소의 별칭을 name으로 지정

```
name = "이름" # 1, 1
```

- 3, 3 주소의 별칭을 age로 지정

```
age = 30 # 3, 3
```

## 8. 변수

---

- 결국 변수의 정의와 할당은 다음과 같은 의미.

# "이름" 데이터가 1, 1 주소에 할당되었다.

```
name = "이름"
```

# True 데이터가 0, 4 주소에 할당되었다.

```
is_true = True
```

# 30 데이터가 3, 3 주소에 할당되었다.

```
age = 30
```

- 메모리 주소의 확인

```
is_true = True
```

# is\_true 변수의 메모리 주소

```
print(id(is_true)) # 140703465581416
```

## 8. 변수 참조

- 아래와 같은 코드는 다음과 같은 의미가 된다.

```
num_1 = 30  
num_2 = 50  
num_3 = num_1 + num_2
```

- 이렇게 데이터가 저장되었다고 가정.

	0	1	2	3	4
0	0, 0	0, 1	0, 2	0, 3	0, 4
1	1, 0	<b>30</b>	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	<b>50</b>	2, 4
3	3, 0	3, 1	<b>80</b>	3, 3	3, 4
4	4, 0	4, 1	4, 2	4, 3	4, 4

## 8. 변수 참조

- 아래와 같은 코드는 다음과 같은 의미가 된다.

```
num_3 = num_1 + num_2
```

	0	1	2	3	4
0	0, 0	0, 1	0, 2	0, 3	0, 4
1	1, 0	<b>30</b>	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	<b>50</b>	2, 4
3	3, 0	3, 1	<b>80</b>	3, 3	3, 4
4	4, 0	4, 1	4, 2	4, 3	4, 4

- num\_1 (1, 1)의 값 30과 num\_2 (2, 3)의 값 50을 더해
- num\_3 (3, 2)에 80을 할당한다.

## 8. 변수 참조

---

- 메모리 주소에 별칭을 부여해 사용하는 것을
- “변수 참조” 라고 한다.

- 아래 코드 결과 생각해보기

```
lst_1 = [1, 2, 3]
lst_2 = [lst_1, 4, 5]
print(lst_1) # [1, 2, 3]
print(lst_2) # [[1, 2, 3], 4, 5]

lst_1[0] = 999
print(lst_1) # [999, 2, 3]
print(lst_2) # 어떤일이 벌어졌을까?
```

## 8. 변수 참조

- 코드의 참조 구조

	0	1	2	3	4
0	0, 0	0, 1	0, 2	0, 3	0, 4
1	1, 0	<b>[1, 2, 3]</b>	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	2, 3	2, 4
3	3, 0	3, 1	3, 2	<b>[[1, 1메모리 값], 4, 5]</b>	3, 4
4	4, 0	4, 1	4, 2	4, 3	4, 4

- 결과는

메모리 주소를 출력해보자.

[[999, 2, 3], 4, 5]

```
print(id(lst_1)) # 1612088101632
print(id(lst_2)) # 1612089094656
print(id(lst_2[0])) # 1612088101632
```

## 8. 변수 참조 끊기 - 복사

---

- 변수를 사용해 개발할 경우 메모리 참조에 의해 예상치 못한 버그가 발생할 경우가 많음.
- 이런 버그를 예방하기 위해
- 변수A의 값 만 복사해 변수B에 할당해 사용해야 함.
- 값 복사는 두 가지 방법이 존재함.
  - 1. 얕은 복사 (Shallow Copy)
  - 2. 깊은 복사 (Deep Copy)

## 8. 변수 참조 끊기 – 얇은 복사

---

- 얇은 복사: 메모리 주소를 복사하지 않고 값을 복사
- 변수를 다른 변수에 직접 할당 할 경우, 메모리가 복사됨.

# 메모리 주소 복사

```
lst_1 = [1, 2]
```

```
lst_2 = lst_1
```

```
print(id(lst_1)) # 2030593723136
```

```
print(id(lst_2)) # 2030593723136
```



## 8. 변수 참조 끊기 – 얇은 복사

---

- 데이터를 복사하려면?

```
lst_1 = [1, 2]
```

```
# 시퀀스타입의 경우 슬라이싱을 이용
```

```
lst_2 = lst_1[:]
```

```
lst_1[0] = 99 # 값 변경 시도
```

```
print(lst_1, id(lst_1)) # [99, 2] 1946935446272
```

```
print(lst_2, id(lst_2)) # [1, 2] 1946936439296 참조가 끊어진 것을 확인.
```

## 8. 변수 참조 끊기 – 얕은 복사

---

- 슬라이싱이 불가능한 변수의 경우

```
from copy import copy
```

```
data_1 = {"a": "b"}
```

```
data_2 = copy(data_1)
```

```
data_1["a"] = "CC"
```

```
print(data_1, id(data_1)) # {'a': 'CC'} 1877758077824
```

```
print(data_2, id(data_2)) # {'a': 'b'} 1877756757760
```

## 8. 변수 참조 끊기 – 얇은 복사의 한계

- 얇은 복사의 한계
  - 얇은 복사는 메모리에 들어있는 값만 복사함.
  - 한 메모리 내에서 다른 메모리를 참조하고 있다면
  - 메모리 주소를 복사해온다.

```
from copy import copy
```

```
data_1 = {"a": "b", "b": [1, 2, 3]}  
data_2 = copy(data_1)
```

```
data_1["a"] = "CC"  
data_2["b"].append(4)
```

```
# {'a': 'CC', 'b': [1, 2, 3, 4]} 2736911182720  
print(data_1, id(data_1))  
# {'a': 'b', 'b': [1, 2, 3, 4]} 2736910190336  
print(data_2, id(data_2))
```

## 8. 변수 참조 끊기 – 얇은 복사의 한계

- 얇은 복사의 한계
  - 얇은 복사는 메모리에 들어있는 값만 복사함.
  - 한 메모리 내에서 다른 메모리를 참조하고 있다면
  - 메모리 주소를 복사해온다.

	0	1	2	3	4
0	0, 0	0, 1	[1, 2, 3]	0, 3	0, 4
1	1, 0	{"a": "b", "b": 0,2메모리주소}	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	2, 3	2, 4
3	3, 0	3, 1	3, 2	{"a": "b", "b": 0,2메모리주소}	3, 4
4	4, 0	4, 1	4, 2	4, 3	4, 4

## 8. 변수 참조 끊기 – 얇은 복사의 한계

- 얇은 복사의 한계

	0	1	2	3	4
0	0, 0	0, 1	[1, 2, 3]	0, 3	0, 4
1	1, 0	{"a": "b", "b": 0,2메모리주소}	1, 2	1, 3	1, 4
2	2, 0	2, 1	2, 2	2, 3	2, 4
3	3, 0	3, 1	3, 2	{"a": "b", "b": 0,2메모리주소}	3, 4
4	4, 0	4, 1	4, 2	4, 3	4, 4

```
from copy import copy
```

```
data_1 = {"a": "b", "b": [1, 2, 3]}  
data_2 = copy(data_1)
```

```
print(id(data_1)) # 1571029871488  
print(id(data_1["b"])) # 1571030138496  
print(id(data_2)) # 1571028223744  
print(id(data_2["b"])) # 1571030138496
```

## 8. 변수 참조 끊기 – 깊은 복사

---

- 얕은 복사의 한계 – 깊은 복사(deepcopy 이용)

```
from copy import deepcopy
```

```
data_1 = {"a": "b", "b": [1, 2, 3]}
```

```
data_2 = deepcopy(data_1)
```

```
print(id(data_1)) # 2364549262208
```

```
print(id(data_1["b"])) # 2364549512768 >> 주소가 분리됨
```

```
print(id(data_2)) # 2364549511808
```

```
print(id(data_2["b"])) # 2364549397056 >> 주소가 분리됨
```

## 8. 변수 참조 끊기 – 깊은 복사

---

- 깊은 복사(deepcopy 이용)
  - 메모리 참조가 완전히 끊어진다.

```
from copy import deepcopy
```

```
# 메모리 주소 복사
```

```
data_1 = {"a": "b", "b": [1, 2, 3]}
```

```
data_2 = deepcopy(data_1)
```

```
data_1["a"] = "CC"
```

```
data_1["b"].append(4)
```

```
print(data_1) # {'a': 'CC', 'b': [1, 2, 3, 4]}
```

```
print(data_2) # {'a': 'b', 'b': [1, 2, 3]}
```

## 9. 연습문제



## 9. 연습문제

---

- 1. 자동차 모델별 가격이 아래와 같다.  
제시된 모델의 가격을 변수에 할당하고 가격의 합을 구해 출력
  - 테슬라 5000만원
  - 아이오닉5 4000만원
  - 아반떼 2400만원
- 2. 자연수 35가 홀수인지 짝수인지 출력
  - num = 35
- 3. 주민등록번호 앞 7자리를 아래와 같이 입력받았다.  
birthday = 198104012  
"연도(4자리)-월(2자리)-일(2자리) 성별" 로 출력
  - 예> 1981-04-01 여
  - 성별: 1, 3 = 남, 2, 4 = 여,

## 9. 연습문제

---

- 4. 다음 문장에 오타가 발견되어 수정하려한다.  
틀린 문장을 수정하는 코드를 작성. (replace)
  - 안녕하세요. 반갑습니다. => "녕" 을 "녕" 으로 변경
- 5. [9, 8, 7, 6, 5, 4, 3, 2, 1] 를 순정렬하여 출력
  - 결과: [1, 2, 3, 4, 5, 6, 7, 8, 9]
- 6. [9, 9, 7, 4, 3, 2, 2, 8] 데이터를 중복을 제거하고 역정렬하여 출력
  - 결과: [9, 8, 7, 4, 3, 2]
- 7. "안녕하세요" 를 리스트로 변경해 출력
  - 결과: ["안", "녕", "하", "세", "요"]

## 9. 연습문제

---

- 8. ["안", "녕", "하", "세", "요"] 를 "안-녕-하-세-요" 로 출력
- 9. 딕셔너리 a에서 'B'에 해당하는 값을 추출해 보자.
  - a = {'A':90, 'B':80, 'C':70}
  - 결과: 80
- 10. 아래 변수를 이용해 "안녕하세요, 제 이름은 "홍길동" 입니다."를 출력. ( % 바인딩 이용 )
  - name = "홍길동"
- 11. 아래 변수를 이용해 "안녕하세요, 제 이름은 "홍길동" 입니다."를 출력. ( {} 바인딩 이용 )
  - name = "홍길동"

## 9. 연습문제

---

- 12. 아래 변수를 이용해 "안녕하세요, 제 이름은 "홍길동" 입니다."를 출력. ( f-string 이용 )
  - name = "홍길동"
- 13. "abcdefghij" 문자열에서 "g" 문자의 위치(index)를 찾아 출력
- 14. " ab c " 문자열의 앞 뒤 공백을 모두 제거하고 아래와 같이 출력
  - 결과: 공백을 제거한 결과는 "ab c" 입니다.
- 15. "Hello, Python World" 문자열에서 소문자L 의 마지막 위치를 찾아 출력
- 16. "Hello, Python World" 문자열을 ["Hello,", "Python", "World"]로 변경해 출력

# 제어문

---

10. if

11. while

12. for

13. 연습문제

**10. if**

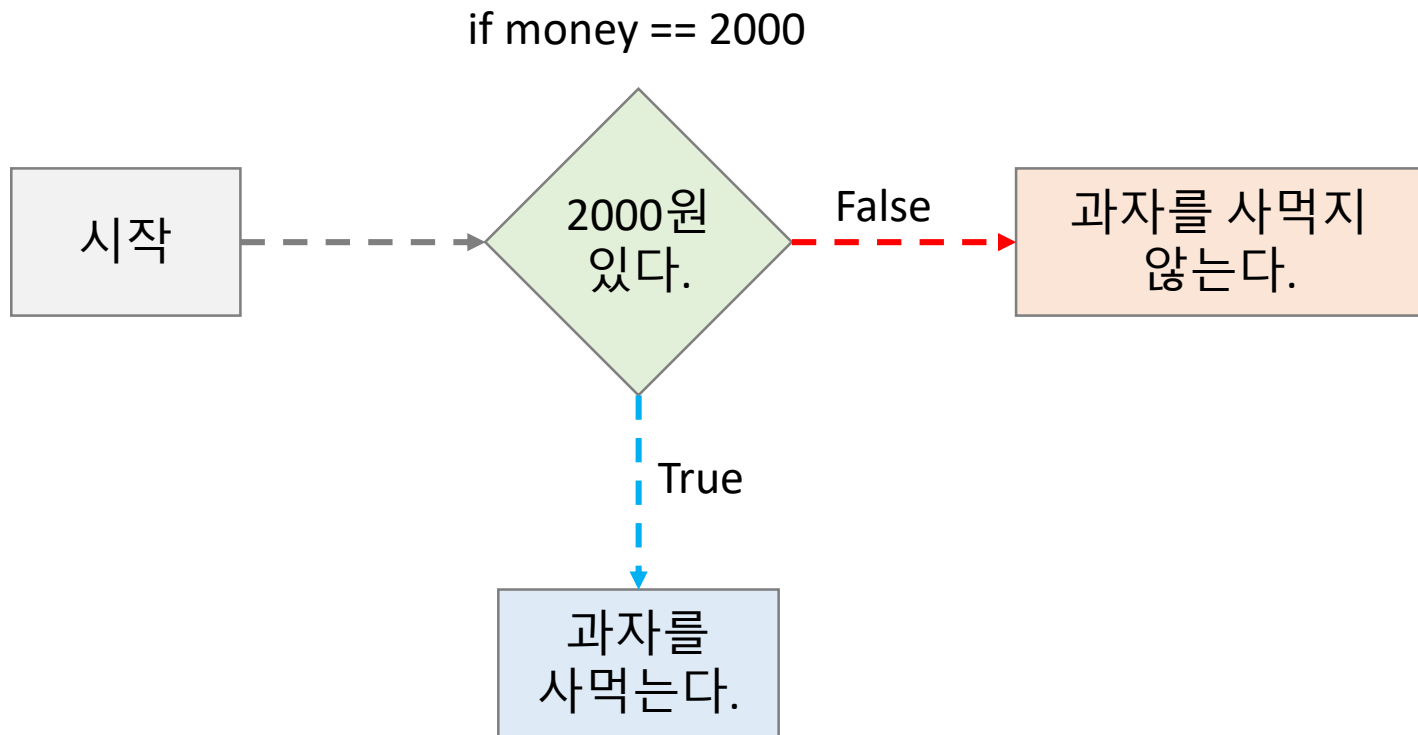
# 10. if

---

- 프로세스의 흐름을 제어하는 분기문.
  - 필요지식
    - 1. 변수
    - 2. 불린
    - 3. 비교연산자와 논리연산자

# 10. if

- 2000원이 있으면 과자를 사먹고 없으면 사먹지 않는다.





# 10. if

---

- 2000원이 있으면 과자를 사먹고 없으면 사먹지 않는다.

```
money = 2000
```

```
if money == 2000:  
    print("과자를 사먹는다.") # 실행  
else:  
    print("과자를 사먹지 않는다.") # 실행하지 않음.
```

# 10. if

---

- if 는 여러 조건 중 결과가 True 인 첫 번째 문장만 실행한다.
  - 다른 문장들은 무시해버린다.

```
money = 2000
```

```
if money == 2000:  
    print("과자를 사먹는다.") # 실행  
else:  
    print("과자를 사먹지 않는다.") # 실행하지 않음.  
  
print("맛있게 먹었다.") # 실행
```

# 10. if

---

- if 작성 방법

`if` 비교연산 및 논리연산:

[들여쓰기]비교연산 결과가 `True`일때 실행할 문장.

`else`:

[들여쓰기]비교연산 결과가 `False`일때 실행할 문장.

- 들여쓰기는 보통 Space 4칸으로 한다.
- 들여쓰기가 맞지 않을 경우 에러 발생함.

```
money = 2000
```

```
if money == 2000:
```

```
    print("과자를 사먹는다.") # 실행
```

```
    print("냠냠냠") # IndentationError: unexpected indent
```

```
else:
```

```
    print("과자를 사먹지 않는다.") # 실행하지 않음.
```

# 10. if

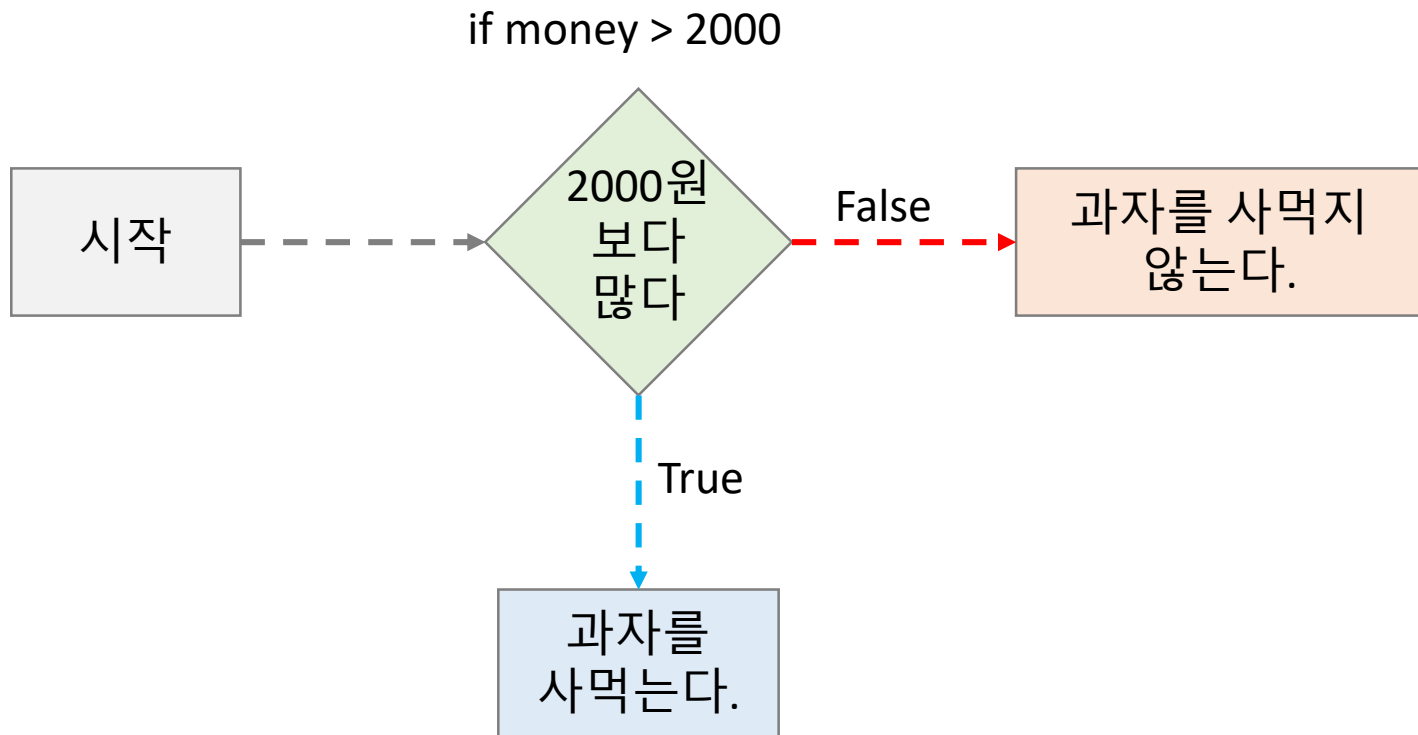
- 비교연산자

연산자	코드	설명	결과 (x = 10, y = 20)
<	x < y	x가 y 보다 작다	True
>	x > y	x가 y 보다 작다	False
==	x == y	x가 y 와 같다	False
!=	x != y	x가 y 와 같지 않다 (다르다)	True
>=	x >= y	x가 y 보다 크거나 같다	False
<=	x <= y	x가 y 보다 작거나 같다	True

- 코드를 작성해 결과를 직접 확인해보자.

# 10. if

- 돈이 2000원 보다 많으면 과자를 사먹고 그렇지 않으면 사먹지 않는다.



# 10. if

---

- 돈이 2000원 보다 많으면 과자를 사먹고 그렇지 않으면 사먹지 않는다.

```
money = 2000
```

```
if money > 2000:  
    print("과자를 사먹는다.") # 실행하지 않음.  
else:  
    print("과자를 사먹지 않는다.") # 실행.
```

# 10. if

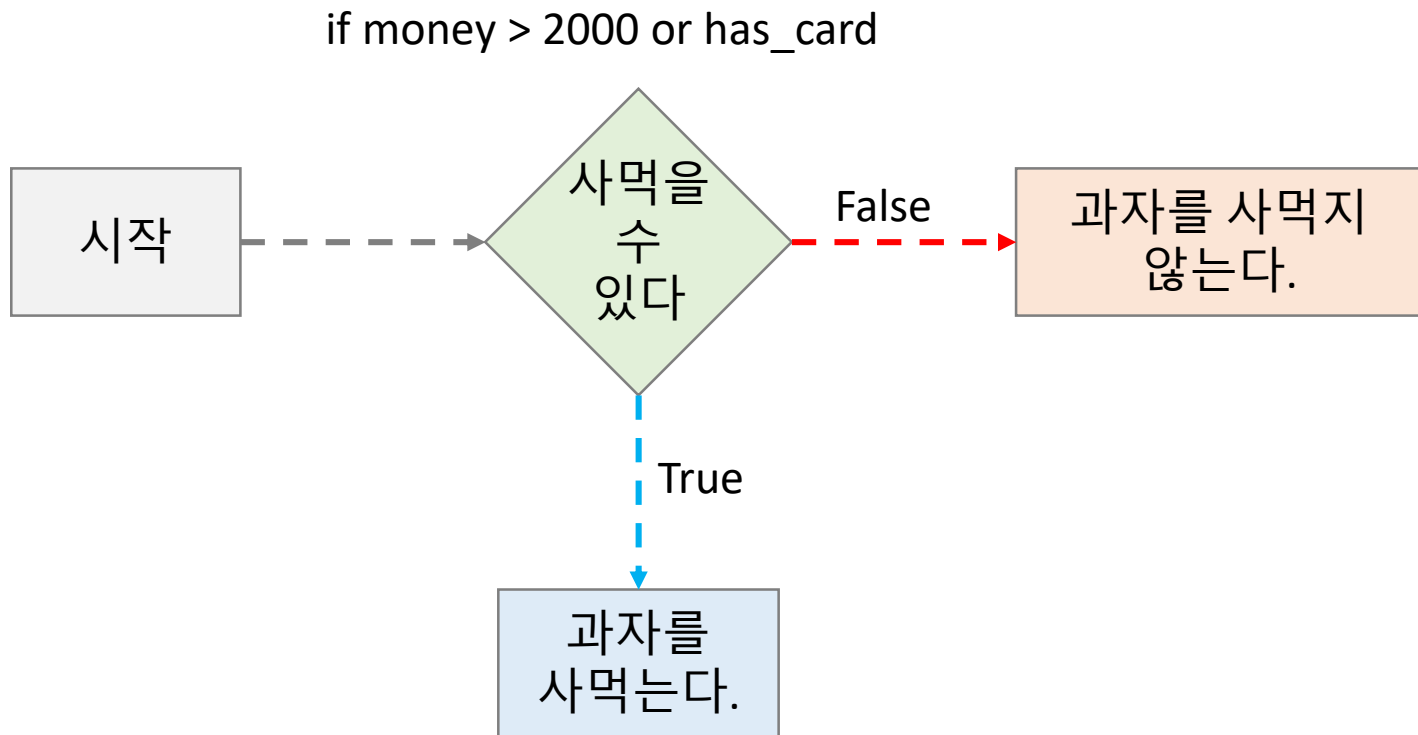
- 논리연산자

연산자	코드	설명	결과 (x = True, y = False)
	x   y	x 와 y 중 하나 이상이 True이다.	True
&	x & y	x 와 y 모두 True이다.	False
or	x or y	x 와 y 중 하나 이상이 True이다.	True
and	x and y	x 와 y 모두 True이다.	False
not	not x	x가 False면 True다. x가 True면 False다.	False
not	not (x   y)	x   y 가 True면 False다 x   y 가 False면 True다	False
not	not (x & y)	x & y 가 True면 False다 x & y 가 False면 True다	True

- 코드를 작성해 결과를 직접 확인해보자.

# 10. if

- 돈이 2000원 보다 많거나 카드를 가지고 있다면 과자를 사먹고 그렇지 않으면 사먹지 않는다.





# 10. if

- 돈이 2000원 보다 많으면 과자를 사먹고 그렇지 않으면 사먹지 않는다.

```
money = 900  
has_card = True
```

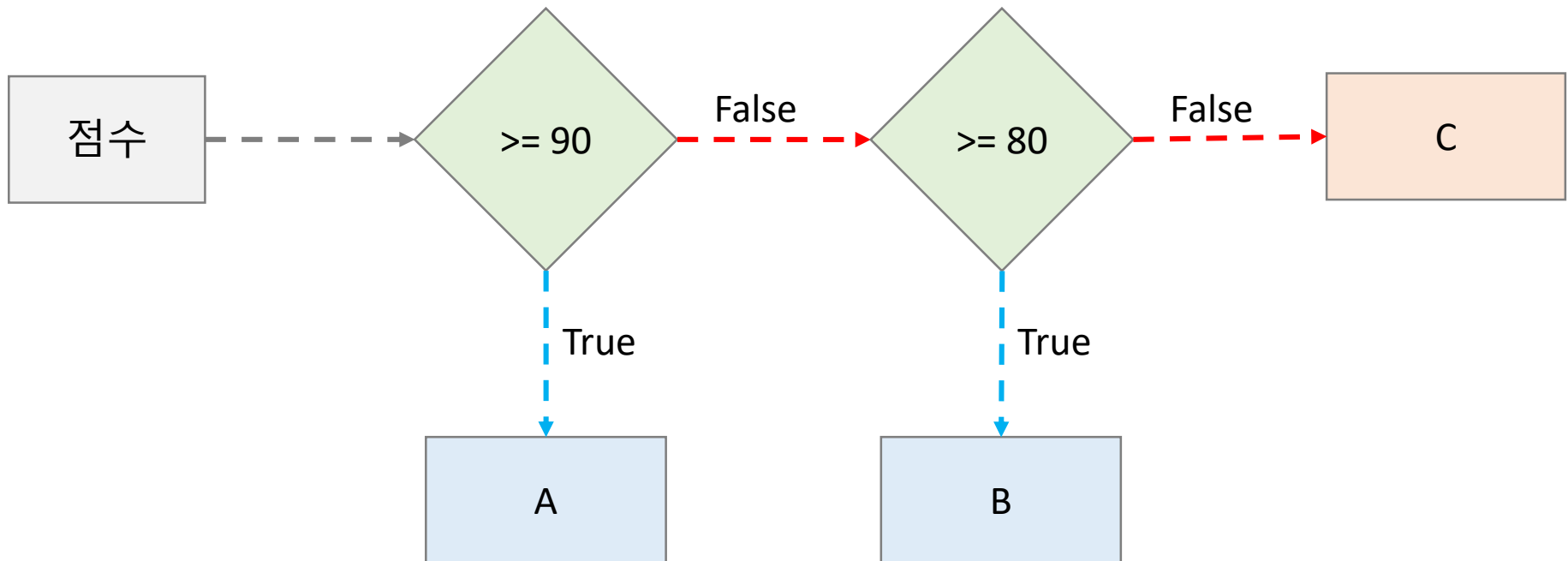
```
if money > 2000 or has_card == True:  
    print("과자를 사먹는다.") # 실행.  
else:  
    print("과자를 사먹지 않는다.") # 실행하지 않음.
```

- has\_card 변수에 이미 True가 할당되어 있으므로 == True는 생략가능

```
if money > 2000 or has_card:  
    print("과자를 사먹는다.") # 실행.  
else:  
    print("과자를 사먹지 않는다.") # 실행하지 않음.
```

# 10. if – elif - else

- 점수가 90점보다 크거나 같다면 A 등급
- 점수가 90점보다 크진 않지만 80점보다 크거나 같다면 B 등급
- 점수가 80점보다 작다면 C 등급을 출력.



# 10. if – elif - else

- 점수가 90점보다 크거나 같다면 A 등급
- 점수가 90점보다 크진 않지만 80점보다 크거나 같다면 B 등급
- 점수가 80점보다 작다면 C 등급을 출력.

score = 79

```
if score >= 90:
    print("A") # 실행 안함
else: # 실행
    if score >= 80:
        print("B") # 실행 안함
    else:
        print("C") # 실행
```

복잡하고 한번에 이해하기 힘들



score = 79

```
if score >= 90:
    print("A") # 실행 안함
elif score >= 80:
    print("B") # 실행 안함
else:
    print("C") # 실행
```

단순하고 한번에 이해하기 좋음

# 10. if – elif - else

---

- elif는 개수의 제한없이 사용 가능하다.

```
score = 79
```

```
if score >= 90:  
    print("A")  
elif score >= 80:  
    print("B")  
elif score >= 70:  
    print("C")  
elif score >= 60:  
    print("D")  
else:  
    print("F")
```

# 10. if – elif - else

---

- 분기의 결과를 변수에 할당하기

```
score = 79  
grade = None # 아무 값도 할당하지 않음
```

```
if score >= 90:  
    grade = "A"  
elif score >= 80:  
    grade = "B"  
elif score >= 70:  
    grade = "C"  
elif score >= 60:  
    grade = "D"  
else:  
    grade = "F"  
  
print(grade) # C
```

# 10. if – elif - else

---

- 분기문이 단순할 경우 조건부표현식으로 변경 가능.

```
score = 59  
grade = "F"  
pass_result = None
```

```
if grade != "F":  
    pass_result = "success"  
else:  
    pass_result = "failure"
```



```
# Condition Expression (조건부 표현식)  
# grade가 "F"가 아니면 "success", 그 외엔 "failure" 를 할당  
pass_result = "success" if grade != "F" else "failure"  
print(pass_result)
```

# 11. while

# 11. while

---

- 반복이 필요할 경우 while을 사용
- 반복문은 while, for 두 가지 종류가 있음.
- while:
  - 반복의 길이가 모호하거나 알 수 없을 때 사용
  - e.g. > 무한반복, 입/출력스트림(키보드 입력 등)
- for:
  - 반복의 길이가 명확할 때 사용.
  - e.g. > list, dict 등의 반복가능한(\_\_iter\_\_) 타입들.



# 11. while

---

- 1 부터 10 까지 반복해보기.

```
index = 1
while index <= 10:
    print(index)
    index += 1

print("반복이 종료되었습니다.")
```

- **index <= 10 의 결과가 True 인 동안 반복한다.**
- index 값이 11이 되는 순간 while 문은 종료되며 다음 명령문을 수행.

# 11. while

- 1 부터 10 까지 반복해보기.

index	조건	조건결과	수행문장	반복지속여부
1	1 <= 10	True	print(index) index += 1	지속
2	2 <= 10	True	print(index) index += 1	지속
3	3 <= 10	True	print(index) index += 1	지속
4	4 <= 10	True	print(index) index += 1	지속
5	5 <= 10	True	print(index) index += 1	지속
6	6 <= 10	True	print(index) index += 1	지속
7	7 <= 10	True	print(index) index += 1	지속
8	8 <= 10	True	print(index) index += 1	지속
9	9 <= 10	True	print(index) index += 1	지속
10	10 <= 10	True	print(index) index += 1	지속
11	11 <= 10	False		종료

# 11. while

---

- 상대방의 HP가 0이 될 때 까지 공격을 하는 무한 반복문 작성.
- 상대방의 HP는 반복이 시작되기 전에 사용자로부터 입력받음.
- 매 공격력은 0 ~ 10 중 랜덤하게 선택된다.
- 상대방의 HP가 0이 되면 무한반복은 종료되며
- “승리하셨습니다!” 가 출력되며 프로그램이 종료된다.

# 11. while

```
from random import randint

hp = int(input("상대 체력을 입력하세요:"))
while hp > 0:
    damage = randint(0, 10)
    print("공격력: %d" % damage)
    hp -= damage
    print("상대 체력: %d" % hp)

print("승리하셨습니다!")
```

상대 체력을 입력하세요:30  
공격력: 7  
상대 체력: 23  
공격력: 5  
상대 체력: 18  
공격력: 7  
상대 체력: 11  
공격력: 8  
상대 체력: 3  
공격력: 2  
상대 체력: 1  
공격력: 1  
상대 체력: 0  
승리하셨습니다!

# 11. while

```
from random import randint

hp = int(input("상대 체력을 입력하세요:"))
while True:
    damage = randint(0, 10)
    print("공격력: %d" % damage)
    hp -= damage
    print("상대 체력: %d" % hp)

    if hp <= 0:
        break #반복문 강제종료

print("승리하셨습니다!")
```

상대 체력을 입력하세요:40  
공격력: 8  
상대 체력: 32  
공격력: 2  
상대 체력: 30  
공격력: 2  
상대 체력: 28  
공격력: 5  
상대 체력: 23  
공격력: 10  
상대 체력: 13  
공격력: 6  
상대 체력: 7  
공격력: 2  
상대 체력: 5  
공격력: 1  
상대 체력: 4  
공격력: 10  
상대 체력: -6  
승리하셨습니다!

# 11. while

---

- 로또 번호 6개, 보너스번호 1개를 list에 담는다.
- list에는 중복된 번호는 넣을 수 없다.
- list 아이템의 개수가 7개가 되면 반복은 종료되고
- 로또번호가 출력된다.

# 11. while

---

```
from random import randint

lotto = []
while len(lotto) < 7:
    lotto_num = randint(1, 45)
    if (lotto_num not in lotto):
        lotto.append(lotto_num)

print(lotto)
```

```
[34, 36, 19, 38, 45, 23, 1]
```

**12. for**



# 12. for

---

- List – for
  - 리스트 아이템을 하나씩 반복한다.

```
lst = ["One", "Two", "Three"]
```

```
for item in lst:  
    print(item)
```

- Tuple – For
  - 튜플 아이템을 하나씩 반복한다.

```
tp = ("One", "Two", "Three")
```

```
for item in tp:  
    print(item)
```

# 12. for

---

- List-Tuple – for
  - 리스트 아이템을 하나씩 반복한다.

```
lst = [("One", "Two"), ("Three", "Four")]
```

```
for f, s in lst:  
    print(f, s)
```

- String – for

```
msg = "안녕하세요?"
```

```
for s in msg:  
    print(s)
```

# 12. for

---

- 모든 값 더해보기
  - `scores = [100, 90, 50, 20, 40]`
- 최대값 구해보기
  - `numbers = [2, 3, 1, 7, 5, 9, 8, 0]`
- 최소값 구해보기
  - `numbers = [2, 3, 1, 7, 5, 9, 8, 0]`

## 12. for – range

---

- for 반복문을 사용할 때 함께 많이 쓰이는 range 함수.
  - 지정한 범위만큼을 만들어 준다.

```
rng = range(1, 11) # 1 부터 10까지의 범위를 만듦.
```

```
# ['__bool__', ... '__iter__', ... 'stop']  
print(dir(rng))
```

```
for n in rng:  
    print(n)
```

- 줄여서.

```
for n in range(1, 11):  
    print(n)
```

## 12. for – continue

---

- 1 부터 30 중 홀수만 출력 – continue 이용.

```
for n in range(1, 31):  
    if n % 2 != 1:  
        continue  
    print(n)
```

- 1 부터 30 중 홀수만 출력 – range 이용.

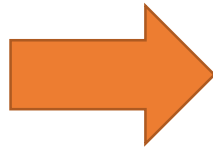
```
for n in range(1, 31, 2):  
    print(n)
```

# 12. for (list comprehension)

---

- List Comprehension
  - 리스트 안에 for를 포함시키면 좀 더 편리하게 리스트를 생성할 수 있음.

```
lst1 = []  
for i in range(1, 11):  
    lst1.append(i)  
print(lst1)
```



```
lst2 = [i for i in range(1, 11)]  
print(lst2)
```

# 12. for (list comprehension)

---

- List Comprehension

# 구구단 2단 결과를 List에 넣기

```
lst = [f"2 X {i} = {i * 2}" for i in range(1, 10)]  
print(lst)
```

# 구구단 2, 3단 결과를 List에 넣기

```
lst = [f"{i} X {j} = {i * j}" for i in range(2, 4)  
                                     for j in range(1, 10)]  
print(lst)
```

# 1 부터 30 중 짝수만 List에 넣기

```
lst = [i for i in range(1, 31) if i % 2 == 0]  
print(lst)
```

# 구구단 2, 3단 결과가 5의 배수인 것만 List에 넣기

```
lst = [f"{i} X {j} = {i * j}" for i in range(2, 4)  
                                     for j in range(1, 10)  
                                     if (i * j) % 5 == 0]  
print(lst)
```

# 12. for (list comprehension)

- list and for 예제
  - 패스워드 생성기 만들어보기

```
lower_alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',  
'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']  
# ord(문자) => ascii 코드  
# chr(ascii코드) => 문자  
upper_alphabets = []  
for small_letter in lower_alphabets:  
    upper_alphabets.append(small_letter.upper())  
alphabets = lower_alphabets + upper_alphabets  
numbers = [str(i) for i in range(0, 11)] # list comprehension  
symbols = ['!', '@', '#', '$', '%', '^', '&', '*', '(', ')', '_', '-', '=',  
'+']
```



# 12. for (list comprehension)

---

- 패스워드 생성기 만들어보기

```
import random # 난수를 가져오기 위해 다른 모듈 import

# Password 생성 규칙
num_of_alphabets = 9
num_of_numbers = 3
num_of_symbols = 2

strong_password = []
# alphabet 9개 추출
for _ in range(0, num_of_alphabets):
    # random.randint(0, 26) => 0 부터 26 사이의 난수 아무거나 하나를 추출
    strong_password.append(alphabets[random.randint(0, len(alphabets) - 1)])

# 숫자 3개 추출
for _ in range(0, num_of_numbers):
    strong_password.append(numbers[random.randint(0, len(numbers) - 1)])

# 특수기호 2개 추출
for _ in range(0, num_of_symbols):
    strong_password.append(symbols[random.randint(0, len(symbols) - 1)])
```

# 12. for (list comprehension)

---

- 패스워드 생성기 만들어보기

```
print(strong_password)
# random.shffle(list 항목을 무작위로 뒤섞음)
random.shuffle(strong_password)
print(strong_password)
new_password = "".join(strong_password)
print(new_password)
```

# 12. for (list comprehension)

---

- 패스워드 생성기 코드 줄여보기

```
from random import shuffle
import re
```

```
lower_alphabets = [chr(a) for a in range(97, 123)]
upper_alphabets = [chr(a) for a in range(65, 91)]
numbers = [str(i) for i in range(0, 10)] # list comprehension
symbols = re.findall("\\W", "".join([chr(i) for i in range(33, 126)]))
```

```
alphabets = lower_alphabets + upper_alphabets
shuffle(alphabets)
shuffle(numbers)
shuffle(symbols)
```

```
alpha_num, num_num, sym_num = (9, 3, 2)
```

```
password = alphabets[:alpha_num] + numbers[:num_num] + symbols[:sym_num]
shuffle(password)
```

```
strong_password = "".join(password)
print(strong_password)
```

## 13. 연습문제

# 13. 연습문제

---

- 1. 아래 변수와 for loop를 이용해 다음과 같이 출력.

```
str = "Long live rock 'N' roll"
```

- 출력

```
Long  
live  
rock  
'N'  
roll
```

- 2. 아래 변수와 for loop, dictionary를 이용해 다음과 같이  
글자별 개수를 출력

```
str = "Long live rock 'N' roll"
```

- 출력

```
{'L': 1, 'o': 3, 'n': 1, 'g': 1, ' ': 4, 'l': 3, 'i': 1, 'v': 1, 'e': 1, 'r': 2, 'c': 1, 'k': 1, "'": 2, 'N': 1}
```

# 13. 연습문제

---

- 3. 아래 코드를 List comprehension으로 변경

```
lst = []
```

```
for i in range(1, 50):  
    if (i % 3 == 0):  
        lst.append(i)
```

```
print(lst)
```

# 함수

---

14. 함수

15. 파일 입/출력

16. 연습문제

## 14. 함수



# 14. 함수

---

- 관련된 코드를 분리해 하나의 기능으로 분리시킨 것.
  - 파라미터
  - 리턴

# 함수 정의 방법

```
def 함수명(파라미터) -> 반환타입:
```

```
    # ... code ...
```

```
    return 반환 값
```

- 파라미터와 반환 값은 필요에 따라 작성하지 않을 수 있다.
- -> 반환타입 은 Type Hint
  - 함수가 반환할 타입을 작성

# 14. 함수

---

- 파라미터도 없고 반환도 없는 함수

```
def print_hello() -> None: # 아무것도 반환하지 않음
    print("Hello, Python!") # return 키워드가 없음
```

```
print(print_hello, type(print_hello))
print_hello() # function call or function invoke
```

- -> None 구문은 생략가능 (return hint)

```
def print_hello(): # 아무것도 반환하지 않음
    print("Hello, Python!") # return 키워드가 없음
```

```
print(print_hello, type(print_hello))
print_hello() # function call or function invoke
```

# 14. 함수

---

- 파라미터가 있는 함수

```
def print_hello(name):  
    print(f"Hello, {name}!")
```

```
print_hello("Python") # positional arguments  
print_hello(name="python") # keyword arguments
```

- keyword arguments 를 더 권장한다.
  - 파라미터 순서가 변경되더라도 영향을 받지 않는다.

```
def print_hello(name="What is your name"): # default parameter  
    print(f"Hello, {name}!")
```

```
print_hello(); # default parameter 사용  
print_hello("Python") # positional arguments  
print_hello(name="python") # keyword arguments
```

# 14. 함수

---

- 반환을 하는 함수

```
def get_welcome_message() -> str:  
    return "안녕하세요!"
```

```
welcome_message = get_welcome_message()  
print(welcome_message)
```

# 14. 함수

---

- 패스워드 생성기 다시보기

```
from random import shuffle
import re
```

```
lower_alphabets = [chr(a) for a in range(97, 123)]
upper_alphabets = [chr(a) for a in range(65, 91)]
numbers = [str(i) for i in range(0, 10)] # list comprehension
symbols = re.findall("\\W", "".join([chr(i) for i in range(33, 126)]))
```

```
alphabets = lower_alphabets + upper_alphabets
shuffle(alphabets)
shuffle(numbers)
shuffle(symbols)
```

```
alpha_num, num_num, sym_num = (9, 3, 2)
```

```
password = alphabets[:alpha_num] + numbers[:num_num] + symbols[:sym_num]
shuffle(password)
```

```
strong_password = "".join(password)
print(strong_password)
```

# 14. 함수

---

- 패스워드 생성기 함수로 만들기 – 여러번 호출 가능.

```
from random import shuffle
import re
```

```
def generate_strong_password(alpha_num, num_num, sym_num):
    lower_alphabets = [chr(a) for a in range(97, 123)]
    upper_alphabets = [chr(a) for a in range(65, 91)]
    numbers = [str(i) for i in range(0, 10)] # list comprehension
    symbols = re.findall("\\W", "".join([chr(i) for i in range(33, 126)]))

    alphabets = lower_alphabets + upper_alphabets
    shuffle(alphabets)
    shuffle(numbers)
    shuffle(symbols)

    password = alphabets[:alpha_num] + numbers[:num_num] + symbols[:sym_num]
    shuffle(password)
    strong_password = "".join(password)
    return strong_password

print(generate_strong_password(9, 3, 2))
```

# 14. 함수 – 가변 파라미터

---

- 가변 파라미터 (Variable positional parameter)
  - 입력값의 수가 정해지지 않았을 때 사용할 수 있는 파라미터

```
def add_all(*args):  
    # *args 는 전달된 파라미터를 Tuple로 만들어준다.  
    print(type(args)) # <class 'tuple'>  
    print(args) # 1 2 3 4 5 6 7 8 9 10  
    result = 0  
    for i in args:  
        result += i  
    return result  
  
print(add_all(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)) # 55
```

# 14. 함수 – 가변 파라미터

---

- 가변파라미터와 일반 파라미터를 함께 사용가능.

```
def add_or_mul(oper, *args):  
    result = 0  
    if oper == "add":  
        for i in args:  
            result += i  
    elif oper == "mul":  
        result = 1  
        for i in args:  
            result *= i  
    return result
```

```
print(add_or_mul("add", 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)) # 55  
print(add_or_mul("mul", 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)) # 3628800
```



# 14. 함수 – 가변 파라미터

- \* 파라미터 뒤에 일반 파라미터가 있을 경우

```
def add_or_mul(*args, oper):  
    ...  
    return result
```

```
print(add_or_mul(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, "add"))  
print(add_or_mul(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, "mul"))
```

- 위 처럼 작성하면 에러가 발생한다.

TypeError: add\_or\_mul() missing 1 required keyword-only argument: 'oper'

- 가변인자 뒤에 일반 파라미터를 작성할 경우
- 반드시 키워드파라미터로 전달해야 한다.

```
print(add_or_mul(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, oper="add"))  
print(add_or_mul(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, oper="mul"))
```

# 14. 함수 – 키워드 가변 파라미터

---

- 파라미터에 \*를 붙이면 가변 파라미터(Tuple)
- \*\* 를 붙이면 키워드 가변 파라미터가(Dictionary) 된다
- 아래와 같은 함수를 정의하고

```
def add_or_mul(oper, num_1, num_2):  
    if oper == "add":  
        return num_1 + num_2  
    elif oper == "mul":  
        return num_1 * num_2  
    return 0
```

- 아래와 같이 호출한다면 키워드 가변 파라미터를 고려해볼만하다.

```
print(add_or_mul(num_1 = 10, num_2 = 20, oper = "add")) # 30  
print(add_or_mul(oper = "mul", num_2 = 10, num_1 = 20)) # 200
```

# 14. 함수 – 키워드 가변 파라미터

- 키워드 가변 파라미터로 변경해보기

```
def add_or_mul(**kwargs):  
    print(kwargs)  
    print(type(kwargs))  
    if kwargs.get("oper", "add") == "add":  
        return kwargs.get("num_1", 0) + kwargs.get("num_2", 0)  
    elif kwargs.get("oper", "add") == "mul":  
        return kwargs.get("num_1", 1) * kwargs.get("num_2", 1)  
    return 0
```

```
print(add_or_mul(num_1 = 10, num_2 = 20, oper = "add")) # 30  
print(add_or_mul(oper = "mul", num_2 = 10, num_1 = 20)) # 200
```

- 주의할 점
  - 키워드 가변 파라미터로 전달 할 경우,
  - 함수 내부에서 사용하는 파라미터에 대해 조사가 되어야 한다.

# 14. 함수 – 파라미터를 다른 함수로 전달

---

- 가변 파라미터를 다른 함수로 전달하려면?

```
def add_or_mul_1(oper, *args):  
    if oper == "add":  
        return add(*args) # add 함수로 가변인자를 전달.  
    elif oper == "mul":  
        return mul(*args) # mul 함수로 가변인자를 전달.  
    return 0
```

```
def add(*args):  
    result = 0  
    for i in args:  
        result += i  
    return result
```

```
def mul(*args):  
    result = 1  
    for i in args:  
        result *= i  
    return result
```

# 14. 함수 – 파라미터를 다른 함수로 전달

---

- 키워드 파라미터를 다른 함수로 전달하려면?

```
def add_or_mul_2(**kwargs):  
    if kwargs.get("oper", "add") == "add":  
        return add(**kwargs) # add 함수로 키워드 파라미터 전달  
    elif kwargs.get("oper", "add") == "mul":  
        return mul(**kwargs) # mul 함수로 키워드 파라미터 전달  
    return 0
```

```
def add(**kwargs):  
    return kwargs.get("num_1", 0) + kwargs.get("num_2", 0)
```

```
def mul(**kwargs):  
    return kwargs.get("num_1", 1) * kwargs.get("num_2", 1)
```

# 14. 함수 – 주의점

---

- Default parameter 뒤에 일반 파라미터가 있을 경우
- 아래와 같은 에러가 발생.

```
# non-default argument follows default argument
```

```
def add_number(type="add", num_1, num_2):  
    print(type)  
    print(num_1)  
    print(num_2)
```

```
add_number("add", 1, 2)
```

- Default parameter는 항상 뒤에 위치해야 한다.

```
def add_number(num_1, num_2, type="add"):  
    print(type)  
    print(num_1)  
    print(num_2)
```

# 14. 함수 – 변수의 범위

---

- Python의 함수는 별도의 분리된 namespace라는 범위를 가짐.

```
age = 10 # global namespace
```

```
def add_age(a):  
    # add_age namespace 시작  
    age = a  
    print("in add_age", age) # 1  
    # add_age namespace 끝
```

```
print("1. age", age) # 10
```

```
add_age(1)
```

```
print("2. age", age) # 10
```

- add\_age 안의 age 변수와 바깥의 age 변수는 다른 변수로 작동.

# 14. 함수 – 변수의 범위

---

- add\_age 함수에서 global namespace의 age를 변경하려면
- global 키워드 사용.

```
age = 10 # global namespace
```

```
def add_age(a):  
    # add_age namespace 시작  
    global age # global namespace의 age 사용 선언  
    age = a  
    print("in add_age", age) # 1  
    # add_age namespace 끝
```

```
print("1. age", age) # 10  
add_age(1)  
print("2. age", age) # 1
```



# 14. 함수 – nested function 변수의 범위

---

- python은 함수 안에 다른 함수를 정의할 수 있음.
- 함수 안의 함수 역시 별도의 namespace를 가짐.

```
def a():  
    x = 10  
    def b():  
        x = 20  
    b()  
    print(x) # 10  
a()
```

# 14. 함수 – nested function 변수의 범위

---

- 함수 안의 함수들의 변수 영역은
- enclosed scope, enclosing scope 로 구분
  - enclosed scope:
    - 내부 함수의 변수가 상위 함수 변수를 참조하는 범위
  - enclosing scope
    - 상위 함수의 변수가 내부 함수의 변수로 전달되는 범위
- 이런 범위를 사용하려면
- nonlocal 키워드가 필요하다.

# 14. 함수 – nested function 변수의 범위

---

- nonlocal

```
def a():  
    x = 10 # enclosing scope  
    def b():  
        # a function의 x 변수(가장 가까이 있는)를 사용하도록 변경  
        nonlocal x  
        x = 20 # enclosed scope  
    b() # 20  
    print(x)  
a()
```

# 14. 함수 – LEGB Rule

---

- Python이 변수를 찾아가는 규칙

```
# (LEGB rule)
# local
# enclosing
# global
# build-in >>> print(dir(__builtins__))
```

- 별도의 namespace에서 참조만 할 경우 LEGB에 의해 변수를 참조

```
def a():
    x = 10
    def b():
        # x = 20 # LEGB Rule 동작 안함
        print(x) # LEGB Rule 동작 함.
        c = 10 * x # LEGB Rule 동작 함.
        print(c)
    b()
    print(x)
a()
```

# 14. 함수 – lambda

---

- 함수를 파라미터로 전달하기 위한 방법
- lambda == lambda function
- lambda를 이용하면 function을 간편하게 정의 가능.

```
add = lambda a, b: a + b  
result = add(1, 2)  
print(add)
```

# 14. 함수 – lambda

---

- lambda를 이용한 함수 파라미터 전달

```
add = lambda a, b: a + b
```

```
def print_add(add_fn):  
    result = add_fn(10, 50)  
    print(result)
```

```
print_add(add)
```

- 혹은

```
def print_mul(add_fn):  
    result = add_fn(10, 50)  
    print(result)
```

```
print_add(lambda a, b: a * b)
```

# 14. 함수 – lambda

---

- 일반 함수도 파라미터 전달이 가능.

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

```
def print_mul(mul_fn):  
    result = mul_fn(10, 50)  
    print(result)
```

```
print_mul(mul)
```

- 일반 함수를 파라미터로 전달하기 위해서는 함수의 정의가 필요.
- 간단한 함수라면 lambda로 전달하는 것이 유리.

# 15. 파일 입/출력



# 15. 파일 입/출력

---

- python 에서 파일을 읽고 쓰는 함수.
- 파일을 제어하는 여러 Mode가 존재함.
- <https://docs.python.org/ko/3/library/functions.html#open>

문자	의미
'r'	읽기용으로 엽니다 (기본값)
'w'	쓰기용으로 엽니다, 파일을 먼저 자릅니다.
'x'	독점적인 파일 만들기용으로 엽니다, 이미 존재하는 경우에는 실패합니다.
'a'	open for writing, appending to the end of file if it exists
'b'	바이너리 모드
't'	텍스트 모드 (기본값)
'+'	갱신(읽기 및 쓰기)용으로 엽니다

# 15. 파일 입/출력

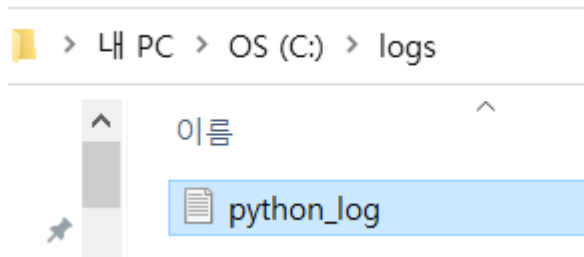
- 파일을 생성하고 쓰기. 실행할 때마다 파일을 새로 쓴다.

```
from os import makedirs
```

```
# c:/logs 폴더 생성 (os.makedirs 사용)  
# exist_ok=True: 폴더가 존재한다면 아무일도 하지않음.  
makedirs("C:/logs", exist_ok=True)
```

```
# 파일 쓰기
```

```
text_file = open("C:/logs/python_log.txt", "w")  
text_file.write("write file (line 1)\n")  
text_file.writelines(["apple\n", "banana\n", "pineapple\n"]) # list  
text_file.writelines(("blueberry\n", "strawberry\n", "grape\n")) # tuple  
text_file.close()
```



write file (line 1)

apple  
banana  
pineapple  
blueberry  
strawberry  
grape

# 15. 파일 입/출력

---

- 파일 내용을 추가하기.

# 파일 내용 추가하기

```
text_file = open("C:/logs/python_log.txt", "a")
text_file.write("write file (line 1)\n")
text_file.writelines(["apple\n", "banana\n", "pineapple\n"]) # list
text_file.writelines(("blueberry\n", "strawberry\n", "grape\n")) # tuple
text_file.close()
```

```
write file (line 1)
apple
banana
pineapple
blueberry
strawberry
grape
write file (line 1)
apple
banana
pineapple
blueberry
strawberry
grape
```

# 15. 파일 입/출력

---

- 파일 내용 읽어오기

# 파일 내용 읽기

```
text_file = open("C:/logs/python_log.txt", "rt")
print(text_file.read()) # 파일의 전체 내용 읽어오기
text_file.close()
```

# 파일 내용 읽기

```
text_file = open("C:/logs/python_log.txt", "rt")
print(text_file.read(1)) # 한 글자만 가져오기
print(text_file.read(1)) # 그 다음 한 글자만 가져오기
text_file.close()
```

# 한글자씩 읽어오기

```
text_file = open("C:/logs/python_log.txt", "rt")
for c in text_file.read():
    print(c, end="") # print 함수의 개행 무시
text_file.close()
```

# 15. 파일 입/출력

---

- 파일 내용 읽어오기

# 한 줄씩 읽어오기

```
text_file = open("C:/logs/python_log.txt", "rt")
print(text_file.readline())
text_file.close()
```

# 한 줄씩 모두 읽어오기

```
text_file = open("C:/logs/python_log.txt", "rt")
for line in text_file:
    print(line)
text_file.close()
```

# 텍스트 파일을 list로 읽어오기

```
text_file = open("C:/logs/python_log.txt", "rt")
print(text_file.readlines())
text_file.close()
```

# 15. 파일 입/출력

---

- with as alias

```
# with를 이용해 쓰기  
# 자동으로 close 된다.
```

```
with open("C:/logs/python_log.txt", "a") as f:  
    f.write("자동 Close\n")
```

```
# with를 이용해 열기  
# 자동으로 close 된다.
```

```
with open("C:/logs/python_log.txt", "rt") as f:  
    print(f.read())
```

# 15. 파일 입/출력 – CSV

---

- Comma Seperate Value (.csv)
- csv 모듈 사용

```
import csv
```

```
# CSV 파일 읽기
```

```
csv_data_list = []
```

```
with open("c:/logs/scores.csv", "r", encoding="utf-8") as f:
```

```
    csv_reader = csv.reader(f)
```

```
    next(csv_reader) # header 부분 생략
```

```
    for row in csv_reader:
```

```
        csv_data_list.append(row)
```

```
print(csv_data_list)
```

# 15. 파일 입/출력 – CSV

---

- Comma Seperate Value (.csv)
- csv 모듈 사용

```
import csv
```

```
# CSV 파일 읽기
```

```
csv_data_list = []
```

```
with open("c:/logs/scores.csv", "r", encoding="utf-8") as f:
```

```
    csv_reader = csv.reader(f)
```

```
    next(csv_reader) # header 부분 생략
```

```
    for row in csv_reader:
```

```
        csv_data_list.append(row)
```

```
print(csv_data_list)
```

```
# CSV 파일 쓰기
```

```
with open("c:/logs/new_scores.csv", "a", encoding="utf-8", newline="") as f:
```

```
    csv_writer = csv.writer(f)
```

```
    for data in csv_data_list:
```

```
        csv_writer.writerow(data)
```



# 15. 파일 입/출력 – Excel

---

- Python 데이터 분석 모듈 Pandas 로 엑셀 파일 핸들링
  - CSV, JSON, XML, HTML 등등 핸들링 가능.
- [https://pandas.pydata.org/docs/user\\_guide/io.html#excel-files](https://pandas.pydata.org/docs/user_guide/io.html#excel-files)
- 가상 환경에 아래 모듈 설치
  - xlrd, openpyxl, pandas
  - e.g. pip install xlrd openpyxl pandas

# 15. 파일 입/출력 – Excel

---

- xlrd
  - .xls 파일 읽고 쓰기
- openpyxl
  - .xlsx 파일 읽고 쓰기
- pandas
  - 데이터 분석 모듈

# 15. 파일 입/출력 – Excel

- Step 1
  - Excel 파일 준비

	A	B	C	D	E
1	순번	이름	주소	이메일	연락처
2	1	김언정	서울 구로구	abc1@gmail.com	010-1234-1111
3	2	차차	서울 동작구	abc2@gmail.com	010-1234-1112
4	3	안치현	서울 영등포구	abc3@gmail.com	010-1234-1113
5	4	유난희	서울 강남구	abc4@gmail.com	010-1234-1114
6	5	구은미	서울 강북구	abc5@gmail.com	010-1234-1115
7	6	최재연	서울 노원구	abc6@gmail.com	010-1234-1116
8	7	하종준	서울 용산구	abc7@gmail.com	010-1234-1117
9	8	오수진	서울 마포구	abc8@gmail.com	010-1234-1118
10	9	한희수	서울 송파구	abc9@gmail.com	010-1234-1119
11	10	신혜영	서울 서대문구	abc10@gmail.com	010-1234-1120

# 15. 파일 입/출력 – Excel

- Step 2
  - Excel 파일 읽기
  - pandas 모듈의 ExcelFile로 엑셀파일 로딩
  - read\_excel 함수로 엑셀 내용 읽어서 DataFrame으로 생성

```
import pandas as pd
```

```
with pd.ExcelFile(r"C:\Users\mcjan\Documents\name.xlsx") as xlsx:  
    df1 = pd.read_excel(xlsx, "Sheet1", index_col=None)  
    df2 = pd.read_excel(xlsx, "Sheet1", index_col=1)  
    df3 = pd.read_excel(xlsx, "Sheet1", index_col=2)  
    # na_values에 지정된 값들은 모두 NaN으로 지정  
    df4 = pd.read_excel(xlsx, "Sheet1", na_values=[""], usecols="A,C:E")  
    df5 = pd.read_excel(xlsx, "Sheet1", usecols=["이름", "순번"])
```

```
print(df1)  
print(df2)  
print(df3)  
print(df4)  
print(df5)
```

# 15. 파일 입/출력 – Excel

---

- Step 3
  - Excel 파일 쓰기
  - DataFrame의 to\_excel 함수로 쓰기
  - pandas의 ExcelWriter 모듈로 쓰기

```
df1.to_excel(r"C:\Users\mcjan\Documents\name_df1.xlsx")
df2.to_excel(r"C:\Users\mcjan\Documents\name_df2.xlsx", sheet_name="SheetDf2")
with pd.ExcelWriter(r"C:\Users\mcjan\Documents\name_df345.xlsx") as writer:
    # index=False (DataFrame의 Index는 엑셀에 쓰지 않는다.)
    df3.to_excel(writer, sheet_name="SheetDf3", index=False);
    df4.to_excel(writer, sheet_name="SheetDf4", index=False);
    df5.to_excel(writer, sheet_name="SheetDf5", index=True);
```

# 16. 연습문제

---

- 1. 주어진 자연수가 홀수인지 짝수인지 구분해 True 혹은 False를 반환해주는 `is_odd` 함수를 작성
- 2. 주어진 파라미터들의 평균값을 계산해 반환하는 `avg()` 함수를 작성 (단, 파라미터의 개수는 정해져있지 않다.)

# 객체지향

---

- 17. 클래스
- 18. Special method
- 19. 클래스 상속
- 20. 클래스 변수
- 21. 클래스 메소드
- 22. 모듈 / 패키지
- 23. 연습문제

# 17. 클래스



# 17. 클래스

---

- Class
  - 객체에 대한 **속성** 및 **기능**을 정의해 놓은 추상적 설계도.
- Instance
  - Class를 사용할 수 있도록 객체화 시킨 것
  - Class는 인스턴스화 시키지 않으면 사용할 수 없다.
- 하나의 클래스로 여러 개의 서로 다른 인스턴스를 생성할 수 있음.

# 클래스 생성

```
class Car:  
    pass
```

# 17. 클래스 - 메소드

---

- 메소드?
  - 함수지향, 절차지향 언어에서 특별한 기능을 하는 코드 모음을
    - 함수 라고 한다.
  - 객체지향 키워드인 클래스에서는 함수를 “메소드” 라고 부른다.

# 17. 클래스 - 메소드

---

- 계산기 예제

# 계산기 클래스

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

# 인스턴스화

```
calc = Calc()
```

# 메소드 호출

```
print( calc.add(10, 40) ) # 50
```

# 17. 클래스 - 메소드

---

- 의문 투성이.
  - self ?
  - Calc() ?
  - calc.add() 에 self는 왜 없나? Default Parameter도 아닌데?
  - calc 와 add 사이의 점(.) 은 뭔가?

# 계산기 클래스

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

# 인스턴스화

```
calc = Calc()
```

# 메소드 호출

```
print( calc.add(10, 40) ) # 50
```

# 17. 클래스 – 메소드 (생성자)

---

- 클래스를 사용하려면 반드시 인스턴스화를 해야만 한다.
- 클래스명() 이라고 실행하면 인스턴스가 만들어진다.

# 계산기 클래스

```
class Calc:
```

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

# 인스턴스화

```
calc = Calc() # 생성자가 호출되면 인스턴스가 만들어진다.
```

```
print(Calc) # <class '__main__.Calc'>
```

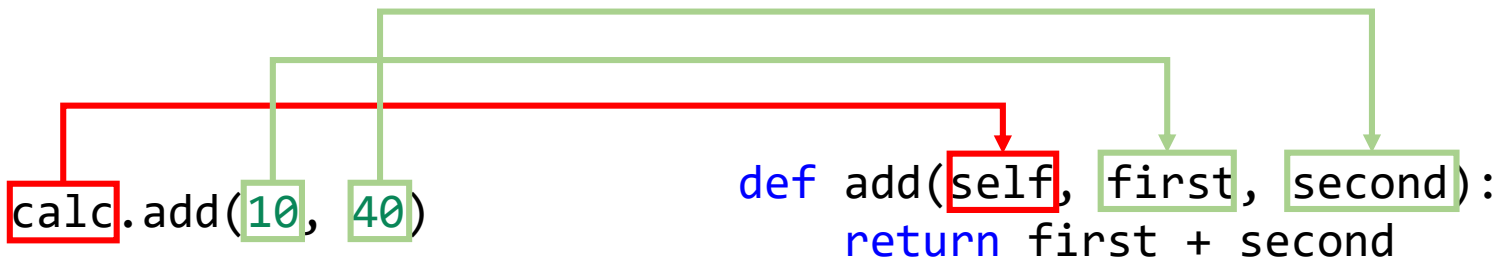
```
print(calc) # <__main__.Calc object at 0x0000013BE4877F10>
```

```
print(type(calc)) # <class '__main__.Calc'>
```

```
# Calc는 클래스, calc는 객체임에 주목하자.
```

# 17. 클래스 – 메소드 (self)

- python 클래스의 큰 특징
  - 모든 메소드는 self를 필수 파라미터로 지정한다.
- 메소드에는 self 파라미터가 가장 첫 번째 위치에 있어야 한다.
  - self 파라미터에는 메소드를 호출 한 인스턴스가 전달된다.



- self 는 메소드를 호출한 인스턴스(객체)를 의미한다.
- self 는 예약어이기에 다른 이름으로 변경시 구문 경고가 발생한다.

# 17. 클래스 – 메소드 ( . )

---

- calc.add
- 메소드를 호출하려면 인스턴스.메소드명 처럼 점(.) 으로 연결해야 한다.
- “calc 인스턴스의 add 메소드를 호출”이라는 의미로 받아들이면 됨.
- 존재하지 않는 메소드는 호출할 수 없다.

# 계산기 클래스

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

# 인스턴스화

```
calc = Calc()
```

# AttributeError: 'Calc' object has no attribute 'add\_all'

```
result = calc.add_all(10, 40)
```

# 17. 클래스 – 메소드 ( . )

---

- 인스턴스화 하지 않고 호출할 수 없다.

# 계산기 클래스

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

# 인스턴스화

```
calc = Calc()
```

```
result = Calc.add(50, 25) # TypeError
print(result)
```



# 17. 클래스

---

- 클래스에 데이터를 할당해보자.

# 계산기 클래스

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

# 인스턴스화

```
calc = Calc()
```

```
result = Calc.add(50, 25) # TypeError
print(result)
```

- 지금까지의 클래스는 숫자를 주면 더하는 단순한 계산기.

# 17. 클래스

---

- 더할 숫자를 클래스에 할당해 놓고 add()만 호출하도록 변경.

# 계산기 클래스

```
class Calc:
```

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

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

# 인스턴스화

```
calc = Calc()
```

```
calc.set_datas(40, 76) # 데이터 저장  
result = calc.add() # 76  
print(result)
```

# 17. 클래스

---

- set\_datas 메소드가 하고 있는 일.

```
# 인스턴스화  
calc = Calc()
```

```
print(vars(calc)) # {}  
calc.set_datas(40, 76) # 데이터 저장  
print(vars(calc)) # {'first': 40, 'second': 76}
```

- calc 인스턴스 자체에 변수 만들어 값을 할당하는 중.
- set\_datas를 호출하기 전/후의 결과가 다르다.

# 17. 클래스

---

- 현재까지의 문제점.
- set\_datas를 호출하지 않을 경우 에러가 발생한다.

# 계산기 클래스

```
class Calc:
```

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

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

# 인스턴스화

```
calc = Calc()
```

```
# AttributeError: 'Calc' object has no attribute 'first'  
result = calc.add()  
print(result)
```

# 17. 클래스 - \_\_init\_\_

- 인스턴스를 만들 때 부터 값을 할당하자.
- 생성자가 호출되면, 클래스내의 \_\_init\_\_ 메소드가 실행된다.
- \_\_init\_\_ 메소드는 클래스를 작성하면 자동으로 만들어지는 메소드.

# 계산기 클래스

```
class Calc:
    def set_datas(self, first, second):
        self.first = first
        self.second = second
    def add(self):
        return self.first + self.second
```

```
print(dir(Calc))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattr__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'add',
'set_datas']
```

# 17. 클래스 - \_\_init\_\_

---

- \_\_init\_\_을 만들어 생성자를 호출 해보기

# 계산기 클래스

```
class Calc:
    def __init__(self):
        print("생성자가 호출되나요?")

    def set_datas(self, first, second):
        self.first = first
        self.second = second

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

```
calc = Calc() # 생성자가 호출되나요?
```

# 17. 클래스 - \_\_init\_\_

---

- set\_datas 메소드 제거 후 아래와 같이 작성
- 생성자에 first, second 파라미터를 보내주지 않으면 에러 발생!

# 계산기 클래스

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

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

```
# TypeError: Calc.__init__() missing 2
# required positional arguments: 'first' and 'second'
calc = Calc()
```

# 17. 클래스 - \_\_init\_\_

---

- 파라미터를 전달하고 실행.
- 이전 예제보다 더 자연스럽다.

# 계산기 클래스

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

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

```
calc = Calc(4, 5)
result = calc.add()
print(result) # 9
```



# 17. 클래스 - \_\_init\_\_

---

- 클래스는 한번 정의해 두면, 횟수에 상관없이 여러번 생성해서 사용할 수 있다.

# 계산기 클래스

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

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

```
calc1 = Calc(4, 5)
print(id(calc1)) # 3063257268432
```

```
calc2 = Calc(4, 5)
print(id(calc2)) # 3063257268560
```

```
calc3 = Calc(4, 5)
print(id(calc3)) # 3063257268496
```

# 18. Special Methods (Magic Method / dunder)

<https://docs.python.org/ko/3/reference/datamodel.html#special-method-names>

# 18. Special method

---

- 언더스코어( \_ ) 두개로 이루어진 특별한 메소드
  - `__init__`
- 문자열 관련
  - `__str__`
  - `__repr__`
- 연산자 오버로딩 가능
  - `__lt__`
  - `__le__`
  - `__eq__`
  - `__add__`
  - `__sub__`
  - `__mul__`
  - 등등..

# 18. Special method

---

```
class Student:
    def __init__(self, name:str, score:int):
        self.name = name
        self.score = score

    def __str__(self):
        # https://docs.python.org/ko/3/reference/datamodel.html#object.\_\_repr\_\_
        # User friendly format
        return f"__str__ name: {self.name}, score: {self.score}"

    def __repr__(self):
        # https://docs.python.org/ko/3/reference/datamodel.html#object.\_\_str\_\_
        # Developer friendly format
        return f"__repr__ name: {self.name}, score: {self.score}"
```

# 18. Special method

---

```
class Student:
    def __le__(self, other):
        print("__le__")
        return self.score <= other.score

    def __add__(self, other):
        print("__add__")
        return self.score + other.score

    def __sub__(self, other):
        print("__sub__")
        return self.score - other.score

    def __mul__(self, num:float):
        print("__mul__")
        return self.score * num
```

# 18. Special method

---

```
student1 = Student("Lee", 90) # __init__  
student2 = Student("Kim", 91) # __init__
```

```
print(student1) # __str__  
print(student2) # __str__
```

```
print(str(student1)) # __str__  
print(repr(student2)) # __repr__
```

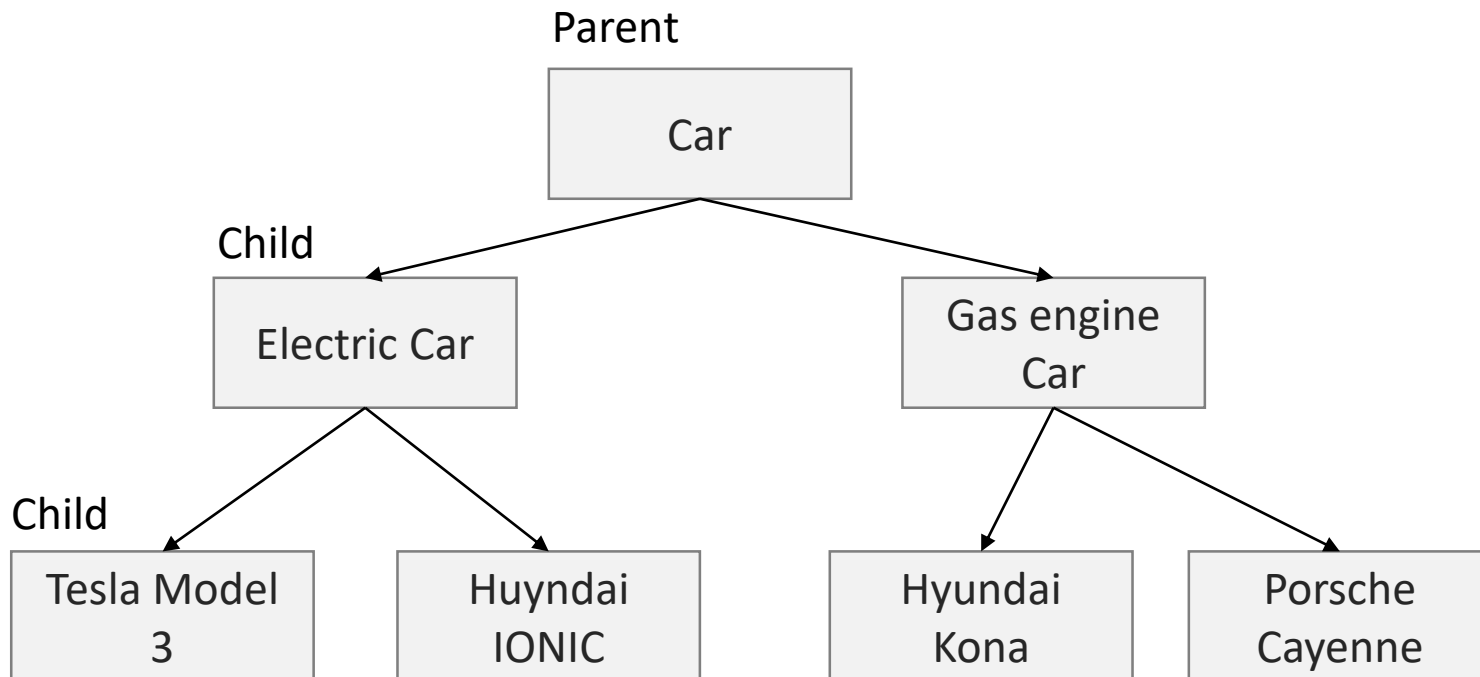
```
print(student1 <= student2) # __le__
```

```
print(student1 + student2) # __add__  
print(student1 - student2) # __sub__  
print(student1 * 5) # __mul__  
print(student2 * 2.5) # __mul__
```

## 19. 클래스 상속

# 19. 클래스 상속

- 클래스 상속
  - A 클래스를 확장해 새로운 B 클래스를 만드는 것
    - A 클래스 = Parent(Super) Class
    - B 클래스 = Child(Sub) Class
  - **Subclass is a Superclass**





# 19. 클래스 상속

---

- Car

```
class Car:
    def __init__(self, brand:str, name:str):
        self.brand = brand
        self.name = name

    def start(self):
        print("시동을 걸었습니다.")

    def drive(self):
        print("주행중입니다.")
```

# 19. 클래스 상속

---

- ElectricCar (Car 상속)

```
class ElectricCar(Car):  
    def __init__(self, brand:str, name:str):  
        super().__init__(brand = brand, name = name)  
  
    def start(self):  
        super().start()  
        print("소리 없음")
```

- GasEngineCar (Car 상속)

```
class GasEngineCar(Car):  
    def __init__(self, brand:str, name:str):  
        super().__init__(brand = brand, name = name)  
  
    def start(self):  
        super().start()  
        print("부르릉!")
```

# 19. 클래스 상속

---

- 테스트

```
ev = ElectricCar(brand="테슬라", name="Model3")  
ev.start()  
ev.drive()
```

```
gv = GasEngineCar(brand="현대", name="KONA")  
gv.start()  
gv.drive()
```

```
print(isinstance(ev, Car)) # True  
print(isinstance(ev, ElectricCar)) # True  
print(isinstance(ev, GasEngineCar)) # False
```

```
print(isinstance(gv, Car)) # True  
print(isinstance(gv, ElectricCar)) # False  
print(isinstance(gv, GasEngineCar)) # True
```

# 19. 클래스 상속

---

- 상속관계 테스트
- 인스턴스별로 알맞은 결과가 출력되는지 확인.

```
def go_and_stop(car: Car):  
    car.start()  
    car.drive()
```

```
ev = ElectricCar(brand="테슬라", name="Model3")  
gv = GasEngineCar(brand="현대", name="KONA")
```

```
go_and_stop(ev)  
go_and_stop(gv)
```

## 20. 클래스 변수

## 20. 클래스 변수

---

- 인스턴스 변수 = 인스턴스에서 접근 가능한 변수.
- `self.변수명` 으로 접근
- 클래스 변수 = 인스턴스와 클래스에서 접근 가능한 변수.
- `인스턴스명.변수명` 혹은 `Class명.변수명` 으로 접근
- 모든 클래스 객체에서 공유가능한 변수 = 클래스 변수

## 20. 클래스 변수

---

- 클래스 변수 테스트

```
class Car:
    product_count = 0

    def __init__(self):
        Car.product_count += 1
```

```
class ElectricCar(Car):
    def __init__(self):
        Car.product_count += 1
```

```
car1 = Car()
car2 = Car()
car3 = Car()
car4 = ElectricCar()
print(Car.product_count) # 4
```

```
print(vars(Car)) # vars 함수로 클래스 변수 확인 가능
```

## 21. 클래스 메소드



# 21. 클래스 메소드

---

- 인스턴스 메소드 = 인스턴스에서 실행 가능한 메소드
- 인스턴스명.메소드명 으로 실행
- 클래스 메소드 = 인스턴스와 클래스에서 접근 실행 가능한 메소드.
- 인스턴스명.메소드명 혹은 Class명.메소드명 으로 실행

# 21. 클래스 메소드

---

- 클래스 메소드 테스트

```
class Car:
    product_count = 0

    def __init__(self):
        Car.product_count += 1

    # 클래스 메소드는 반드시 이 Decorator 선언 필요.
    @classmethod
    def show_product_count(cls):
        # 클래스 메소드의 파라미터는 cls를 받는다.
        # cls는 현재 실행중인 인스턴스의 클래스
        print(cls)
        print(cls.product_count)

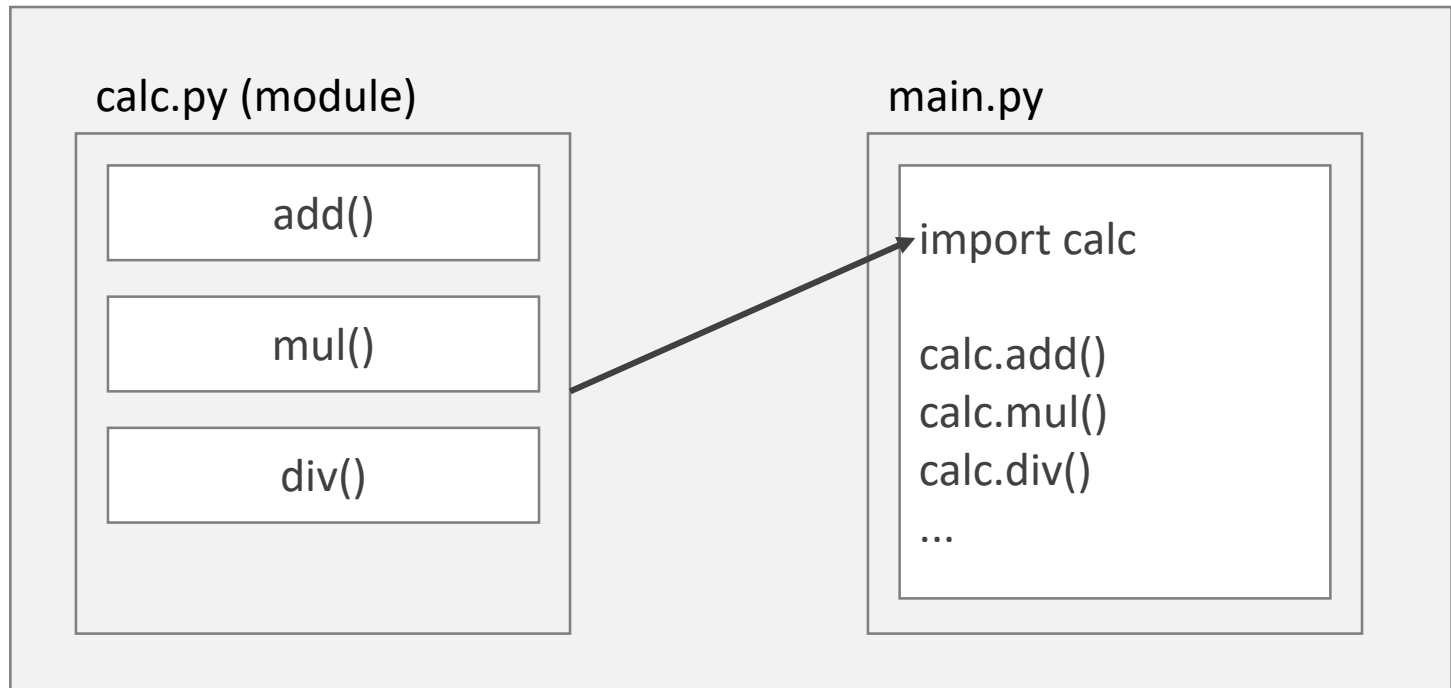
car1 = Car()
car2 = Car()
car1.show_product_count() # 2
Car.show_product_count() # 2
```

## 22. 모듈 / 패키지

## 22. 모듈 / 패키지 - 모듈

- 모듈
  - 함수, 변수, 클래스를 모아 놓은 .py 파일
  - .py 파일은 모두 모듈이다.

venv



## 22. 모듈 / 패키지 - 모듈

---

- calc.py 모듈

```
"""calc.py"""
```

```
def add(num1, num2):  
    return num1 + num2
```

```
def mul(num1, num2):  
    return num1 * num2
```

```
def div(num1, num2):  
    return num1 / num2
```

```
"""main.py"""
```

```
import calc
```

```
print(calc.add(10, 5)) # 15  
print(calc.mul(10, 5)) # 50  
print(calc.div(10, 5)) # 2.0
```

## 22. 모듈 / 패키지 - 모듈

- 모듈 개발시 주의점.
- calc.py 모듈을 개발하면서 테스트 코드를 작성한 경우.

```
"""calc.py"""
```

```
def add(num1, num2):  
    return num1 + num2
```

```
def mul(num1, num2):  
    return num1 * num2
```

```
def div(num1, num2):  
    return num1 / num2
```

```
print(add(1, 2)) # 3 == 정상  
print(mul(1, 2)) # 2 == 정상  
print(div(1, 2)) # 0.5 == 정상
```

```
"""main.py"""
```

```
import calc
```

```
# print(calc.add(10, 5)) # 15  
# print(calc.mul(10, 5)) # 50  
# print(calc.div(10, 5)) # 2.0
```

- calc.py를 import하는 순간 실행이 되어버림.

```
3  
2  
0.5
```

## 22. 모듈 / 패키지 - 모듈

- 모듈 개발시, 이런 문제를 방지하기 위해 실행 thread의 이름을 체크해야 한다.
- 파이썬은 현재 코드를 실행중인 thread의 이름을 확인할 수 있다.
- calc.py, main.py 에 아래 코드를 작성하고 실행.

```
"""calc.py"""
```

```
def add(num1, num2):  
    return num1 + num2  
... 생략 ...  
  
print(__name__) # __main__  
  
print(add(1, 2)) # 3 == 정상  
...
```

```
"""main.py"""
```

```
import calc  
  
print(__name__) # __main__  
  
calc  
3  
2  
0.5  
__main__
```

## 22. 모듈 / 패키지 - 모듈

- `__name__` :
  - 코드가 사용자에게 의해 실행되었을 경우 `"__main__"`이 출력
- 만약 다른 py파일에 의해 실행되었을 경우
  - 실행을 시킨 py파일의 이름이 나온다. (calc)

```
"""calc.py"""
```

```
def add(num1, num2):  
    return num1 + num2  
... 생략 ...  
  
print(__name__) # __main__  
  
print(add(1, 2)) # 3 == 정상  
...
```

```
"""main.py"""
```

```
import calc  
  
print(__name__) # __main__  
  
calc  
3  
2  
0.5  
__main__
```



## 22. 모듈 / 패키지 - 모듈

---

- calc.py의 print 코드를 `__main__` 일 때만 실행하도록 수정.

```
"""calc.py"""
```

```
def add(num1, num2):  
    return num1 + num2
```

```
def mul(num1, num2):  
    return num1 * num2
```

```
def div(num1, num2):  
    return num1 / num2
```

```
if __name__ == "__main__":  
    print(__name__) # __main__  
    print(add(1, 2)) # 3 == 정상  
    print(mul(1, 2)) # 2 == 정상  
    print(div(1, 2)) # 0.5 == 정상
```

## 22. 모듈 / 패키지 - 모듈

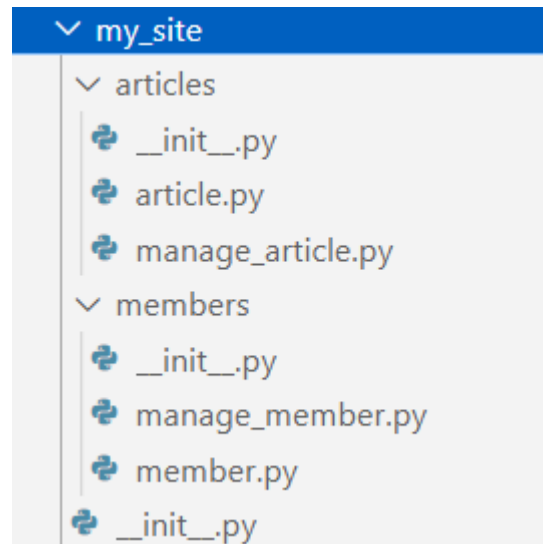
---

- main.py를 실행하면 더이상 calc의 테스트 코드는 실행되지 않는다.

## 22. 모듈 / 패키지 - 패키지

- 관련된 py 파일을 하나의 디렉토리로 관리하는 것.
- 프로젝트에서 아래처럼 패키지를 만들어 본다.

```
my_site/  
├── __init__.py  
├── members/  
│   ├── __init__.py  
│   ├── member.py  
│   └── manage_member.py  
└── articles/  
    ├── __init__.py  
    ├── article.py  
    └── manage_article.py
```



## 22. 모듈 / 패키지 - 패키지

---

- `__init__.py`
  - 이 파일이 있는 디렉터리가 패키지의 일부임을 알려주는 역할.
  - python 3.3 부터 이 파일이 없어도 패키지로 인식을 한다.
    - 하위호환을 위해 파일을 생성하는 것이 안전.

## 22. 모듈 / 패키지 - 패키지

---

- 패키지 코딩

```
"""manage_member.py"""
```

```
from .member import Member
```

```
members = []
```

```
def add_member(member: Member):  
    if member not in members:  
        members.append(member)
```

```
def get_member(id: str):  
    if id in members:  
        return members[members.index(id)]  
    return None
```

## 22. 모듈 / 패키지 - 패키지

---

- 패키지 코딩

```
"""member.py"""
```

```
class Member:
    def __init__(self, id: str, name: str):
        self.id = id
        self.name = name

    def __eq__(self, other):
        if isinstance(other, Member):
            return self.id == other.id
        elif isinstance(other, str):
            return self.id == other
        else:
            return False

    def __str__(self):
        return f"ID: {self.id}, NAME:{self.name}"
```

## 22. 모듈 / 패키지 - 패키지

---

- 패키지 코딩

```
"""manage_article.py"""
```

```
from .article import Article
```

```
articles = []
```

```
def add_article(article: Article):  
    if article not in articles:  
        articles.append(article)
```

```
def get_article(no: int):  
    if no in articles:  
        return articles[articles.index(no)]  
    return None
```

## 22. 모듈 / 패키지 - 패키지

---

- 패키지 코딩

```
"""my_site.main.py"""
```

```
from my_site.articles.manage_article import add_article, get_article
from my_site.articles.article import Article
from my_site.members.manage_member import add_member, get_member
from my_site.members.member import Member
```

```
add_member(Member(id="adm", name="관리자1"))
```

```
add_article(Article(member=get_member("adm"), no=1, name="테스트!"))
```

```
print(get_article(1))
```

```
print(get_article(2))
```



## 23. 연습문제

## 23. 연습문제

---

- 1. 다음 클래스는 학생 1명의 평균 점수를 구하는 코드다. 완성되지 않은 Student 클래스를 완성시켜야 한다.

```
class Student:
```

```
    def get_average(self):  
        sum_score = sum(self.scores)  
        avg = sum_score / len(self.scores)  
        return avg
```

```
student1 = Student(name="홍길동", scores=[100, 100, 90, 70, 60])  
print(f"{student1.get_name()} 님의 평균 점수는\  
{student1.get_average()}점 입니다.")
```

## 23. 연습문제

---

- 2. 다음 클래스는 평균의 학점을 구하는 모듈이다.  
아무 폴더에 main.py를 만들고 grade.py를 import 하여 실행하는 코드를 작성한다.

```
"""score/grade.py"""
```

```
def grade(*scores):
```

```
    def average(*scores):  
        sum_scores = sum(scores)  
        return sum_scores / len(scores)
```

```
    avg = average(*scores)  
    if (avg >= 90): return "A"  
    elif (avg >= 80): return "B"  
    elif (avg >= 70): return "C"  
    elif (avg >= 60): return "D"  
    else: return "F"
```

## 23. 연습문제

---

- 3. 인스턴스에서 멤버변수를 직접 참조하는 것은 올바른 방법이 아니기 때문에, 항상 getter와 setter를 이용해야 한다.  
아래 코드에서 멤버변수를 직접 참조하지 않도록 변경하는 코드를 작성한다.

```
class Calc:
    def __init__(self, a, b):
        self.a = a
        self.b = b
```

```
calc = Calc(10, 20)
a = calc.a
b = calc.b
```

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

## 23. 연습문제

---

- 4. 아래에 Member 클래스가 있다. Member클래스를 상속한 VipMember를 생성하고, welcome() 메소드가 실행되면 "환영합니다 VIP회원님!" 가 출력되도록 코드를 작성한다.

```
class Member:
    def welcome(self):
        print("어서오세요~")
```

## 23. 연습문제

---

- 5. 다음은 시장에 방문한 손님을 표현한 클래스다.  
물건을 하나 살 때엔 + 연산을 이용하여 과일의 개수를 늘리고  
- 연산을 이용하여 돈을 빼는 Special Method를 작성한다.

```
class Customer:
    def __init__(self, fruit_count:int, money:int):
        self.fruit_count = fruit_count
        self.money = money

    def __str__(self):
        return f"과일개수: {self.fruit_count}, 남은 돈: {self.money}"

cust = Customer(0, 10000)
cust += 10 # 과일을 10개 산다.
cust -= (10 * 500) # 5000원을 뺀다.
print(cust)
```

# 예외처리

---

24. 오류가 발생하는 원인

25. 예외처리 방법

26. 예외 회피

27. 사용자 예외 만들고 발생시키기

## 24. 오류가 발생하는 원인



## 24. 오류가 발생하는 원인

---

- 오류가 발생하는 원인은 매우 다양하다.
- e.g.
  - 존재하지 않는 파일을 읽으려 할 때
  - 숫자를 0 으로 나누려 할 때
  - 숫자로 변경할 수 없는 문자를 숫자로 변경하려 할 때
  - 시퀀스의 인덱스 범위가 벗어 났을 때 등

## 24. 오류가 발생하는 원인

---

- 존재하지 않는 파일을 읽으려 할 때

```
f = open("abcd~.txt", "r") # open 함수에서 에러가 발생  
# FileNotFoundError: [Errno 2] No such file or directory: 'abcd~.txt'
```

- 숫자를 0 으로 나누려 할 때

```
print( 10 / 0 ) # 나누기에서 에러 발생  
# ZeroDivisionError: division by zero
```

## 24. 오류가 발생하는 원인

---

- 숫자로 변경할 수 없는 문자를 숫자로 변경하려 할 때

```
a:int = int("abc") # int 함수에서 에러 발생
```

```
# ValueError: invalid literal for int() with base 10: 'abc'
```

- 시퀀스의 인덱스 범위가 벗어 났을 때

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

```
print(lst[4]) # 인덱싱에서 에러 발생
```

```
# IndexError: list index out of range
```

## 25. 예외처리 방법

## 25. 예외처리 방법

---

- 프로그램 실행 중 예외가 발생하면  
예외 내용을 콘솔에 출력하고 프로그램이 강제종료되어버린다.
- 때문에, 예외가 발생할 가능성이 높은 코드는 예외를 따로 처리해 주어야한다.

## 25. 예외처리 방법

---

- 예외처리는 try – except – else – finally 문법을 사용한다.
- 경우에 따라
  - try – except
  - try – finally
  - try – except – else
  - try – except – finally
  - try – except – else – finally
  - 등으로 조합해서 사용하며 try가 없는 문법은 허용되지 않는다.

## 25. 예외처리 방법

---

- try – except

try:

# 예외 발생가능성이 높은 코드 수행

except [발생오류 [as 오류변수]]:

# 예외를 처리할 코드

- except의 [발생오류 [as 오류변수]] 는 생략가능하며 생략될 경우에는 모든 예외를 처리한다.

# 25. 예외처리 방법

---

- try – except 예제 1

```
try:
    print(10 / 0) # ZeroDivisionError
    print("나누기를 완료했습니다.") # 실행되지 않음
except:
    print("에러가 발생했습니다.") # 실행
```

- try – except 예제 2

```
try:
    print(10 / 0) # ZeroDivisionError
    print("나누기를 완료했습니다.") # 실행되지 않음
except ZeroDivisionError:
    print("에러가 발생했습니다.") # 실행
except SyntaxError:
    print("문법이 틀렸습니다.") # 실행되지 않음
```



## 25. 예외처리 방법

---

- try – except 예제 3

```
try:
    print(10 / 0) # ZeroDivisionError
    print("나누기를 완료했습니다.") # 실행되지 않음
except ZeroDivisionError as zde:
    print("에러가 발생했습니다.", zde) # 실행
```

## 25. 예외처리 방법

---

- try – finally

```
try:
    print(10 / 0) # ZeroDivisionError
    print("나누기를 완료했습니다.") # 실행되지 않음
finally:
    print("실행이 완료됐습니다.") # 실행
```

- finally 구문은 예외의 발생여부와 관계없이 항상 마지막에 실행된다.

```
try:
    print(10 / 1)
    print("나누기를 완료했습니다.") # 실행
finally:
    print("실행이 완료됐습니다.") # 실행
```

# 25. 예외처리 방법

- try – except – finally

```
try:
    print(10 / 0) # ZeroDivisionError
    print("나누기를 완료했습니다.") # 실행되지 않음
except ZeroDivisionError as zde:
    print("에러가 발생했습니다.", zde) # 실행
else:
    print("에러가 발생하지 않았습니다.") # 실행되지 않음
finally:
    print("항상 실행됩니다.") # 실행
```

```
try:
    print(10 / 1)
    print("나누기를 완료했습니다.") # 실행
except ZeroDivisionError as zde:
    print("에러가 발생했습니다.", zde) # 실행되지 않음
else:
    print("에러가 발생하지 않았습니다.") # 실행
finally:
    print("항상 실행됩니다.") # 실행
```

# 25. 예외처리 방법

---

- try – except – else

```
try:
    print(10 / 0) # ZeroDivisionError
    print("나누기를 완료했습니다.") # 실행되지 않음
except ZeroDivisionError as zde:
    print("에러가 발생했습니다.", zde) # 실행
else:
    print("에러가 발생하지 않았습니다.") # 실행되지 않음
```

- else 구문은 try에서 에러가 없을 때에만 실행된다.

```
try:
    print(10 / 1)
    print("나누기를 완료했습니다.") # 실행
except ZeroDivisionError as zde:
    print("에러가 발생했습니다.", zde) # 실행되지 않음
else:
    print("에러가 발생하지 않았습니다.") # 실행
```

## 26. 예외 회피

## 26. 예외 회피

- 예외를 회피하는 가장 좋은 방법은 예외를 발생시키지 않는 것.

```
a = None
b = 1
# TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
c = a + b
```

```
a = None
b = 2
if a is None:
    a = 1
c = a + b
print(c)
```

```
a = None
b = 2
if a is not None:
    a = 1
    c = a + b
    print(c)
```

## 26. 예외 회피

---

- 예외를 회피하는 가장 좋은 방법은 예외를 발생시키지 않는 것.

```
a = 10
b = 0
# ZeroDivisionError: division by zero
c = a / b
```

```
a = 10
b = 0
if a == 0:
    a = 1
if b == 0:
    b = 0
c = a / b
```

## 26. 예외 회피

---

- 예외를 회피하는 가장 좋은 방법은 예외를 발생시키지 않는 것.

```
lst = []  
# IndexError: list index out of range  
print(lst[2])
```

```
lst = []  
if len(lst) >= 2:  
    print(lst[2])  
else:  
    print("인덱스가 없습니다.")
```



## 26. 예외 회피

- 예외를 회피하는 가장 좋은 방법은 예외를 발생시키지 않는 것.

```
# FileNotFoundError: [Errno 2]  
# No such file or directory: 'C:\\python_test\\test_file.log'  
f = open(r"C:\python_test\test_file.log")  
f.close()
```

```
from os.path import isfile, exists  
  
file_path = r"C:\python_test\test_file.log"  
  
print("exists(file_path)", exists(file_path))  
print("isfile(file_path)", isfile(file_path))  
if exists(file_path) & isfile(file_path):  
    print("파일이 있다.")  
    f = open(file_path, "r")  
    f.close()
```

## 26. 예외 회피

---

- 부득이하게 try – except를 써야할 때  
예외를 처리해도 마땅히 할 것이 없을 때

```
try:  
    print(10 / 0)  
except ZeroDivisionError:  
    pass
```

- pass 처리한다.
- 권장하지 않는다.

## **27. 사용자 예외 만들고 발생시키기**

## 27. 사용자 예외 만들고 발생시키기

- 파이썬에 내장되어있는 예외 이외의 개발자가 예외를 만들어 사용할 수 있다.
- 이를 커스텀예외 혹은 사용자예외라고 부른다.

```
class AlreadyUserIdError(Exception):
    def __init__(self, user_id):
        self.user_id = user_id
    def __str__(self):
        return f"{self.user_id}는 이미 사용중인 아이디입니다."

def regist_user_id(user_id):
    if (user_id == "admin"):
        raise AlreadyUserIdError(user_id)
    else:
        print(f"{user_id} 가입이 완료되었습니다.")

regist_user_id("userId") # userId가입이 완료되었습니다.
# AlreadyUserIdError: admin는 이미 사용중인 아이디입니다.
regist_user_id("admin")
```

# 알아두면 좋은 내장 모듈

---

28.json

29.datetime

30.os

## 28. json

# 28. json

---

- sample.json 파일 준비

```
{
  "id": "0001",
  "type": "donut",
  "name": "Cake",
  "image": {
    "url": "images/0001.jpg",
    "width": 200,
    "height": 200
  },
  "thumbnail": {
    "url": "images/thumbnails/0001.jpg",
    "width": 32,
    "height": 32
  }
}
```

# 28. json

---

- 파일 읽고 쓰기

```
import json
import os.path as p

# 현재 실행중인 py 파일의 경로 얻어오기
path = p.dirname(p.abspath(__file__))

# json 파일 읽기
with open(path + "/sample.json", mode="r") as f:
    data = json.loads(f.read()) # 파일을 읽어서 json으로 로드
    print(type(data)) # <class 'dict'>
    print(data)
    print(data["id"]) # 0001
    print(data["image"]["url"]) # images/0001.jpg

    data["type"] = "drink"
# json 파일 쓰기
with open(path + "/sample1.json", mode="w") as w:
    w.write(json.dumps(data))
```



# 28. json

---

- sample1.json 확인

```
{  
  "id": "0001",  
  "type": "drink",  
  "name": "Cake",  
  "image": {  
    "url":  
      "images/0001.jpg",  
    "width": 200,  
    "height": 200  
  },  
  "thumbnail": {  
    "url": "images/thumbnails/0001.jpg",  
    "width": 32,  
    "height": 32  
  }  
}
```

## 29. datetime

# 29. datetime

---

- 시간과 관련된 모듈.

```
import datetime
```

```
# 현재 날짜와 시간 조회
```

```
current = datetime.datetime.now()
```

```
print(current) # 2023-04-04 22:22:46.595210
```

```
# 현재 연, 월, 일 조회
```

```
print(current.year) # 2023 (int)
```

```
print(current.month) # 4 (int)
```

```
print(current.day) # 4 (int)
```

```
# 현재 요일
```

```
# 0 ~ 6 (월 ~ 일)
```

```
day_of_week = current.weekday()
```

```
print(day_of_week) # 1 (화요일)
```

# 29. datetime

---

- 시간과 관련된 모듈.

# 날짜 지정

```
custom_date = datetime.datetime(  
    year=2023,  
    month=1,  
    day=1  
)  
print(custom_date) # 2023-01-01 00:00:00  
print(type(custom_date)) # <class 'datetime.datetime'>
```

# 지정 날짜 포매팅 지정

```
datetime_object = datetime.datetime.\  
    strptime("2023-01-01 01:00:00", "%Y-%m-%d %H:%M:%S")  
print(datetime_object) # 2023-01-01 01:00:00  
print(type(datetime_object)) # <class 'datetime.datetime'>
```

# 29. datetime

---

- 시간과 관련된 모듈.

# 날짜를 str로 변경

```
datetime_str = datetime_object.strftime("%Y-%m-%d %H:%M:%S")  
print(datetime_str) # 2023-01-01 01:00:00  
print(type(datetime_str)) # <class 'str'>
```

# 현재날짜/시간을 str로 변경

```
now_str = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
print(now_str) # 2023-04-04 22:33:18  
print(type(now_str)) # <class 'str'>
```

# 일, 시, 분, 초 더하고 빼기

```
print(current)  
print(current + datetime.timedelta(days=1))  
print(current + datetime.timedelta(days=-1))
```

# 29. datetime

---

- 시간과 관련된 모듈.

```
from dateutil.relativedelta import relativedelta
```

```
# 현재 날짜에 6개월 더하기
```

```
print(current + relativedelta(months=6))
```

```
# 현재 날짜에 1년 더하기
```

```
print(current + relativedelta(years=1))
```

**30. os**

# 30. os

---

- 환경변수, 디렉터리, 파일 등 OS의 자원을 제어함.

```
import os
```

```
# OS의 환경변수 조회
```

```
print("environ", os.environ)
```

```
print("=" * 50)
```

```
print("environ items", os.environ.items())
```

```
print("=" * 50)
```

```
print("environ keys", os.environ.keys())
```

```
print("=" * 50)
```

```
print("environ PATH", os.environ.get("PATH"))
```

```
print("=" * 50)
```

```
print(type(os.environ)) # <class 'os._Environ'>
```

```
print(dir(os.environ)) # <class 'os._Environ'>
```



# 30. os

---

- 환경변수, 디렉터리, 파일 등 OS의 자원을 제어함.

```
import os
```

```
# OS 이름 조회
```

```
print(os.name) # nt
```

```
# 현재 디렉토리 경로 조회
```

```
print(os.getcwd()) # C:\python_projects\python_exam
```

```
# 디렉토리 경로 변경
```

```
os.chdir("C:\\ProgramData")
```

```
print(os.getcwd()) # C:\ProgramData
```

```
# 실행중인 파일의 경로 조회
```

```
# c:\python_projects\python_exam\pytest
```

```
print(os.path.dirname(os.path.abspath(__file__)))
```

# 30. os

- 환경변수, 디렉터리, 파일 등 OS의 자원을 제어함.

```
import os
```

```
print(os.system("dir"))
```

```
"""
```

```
C 드라이브의 볼륨: OS  
볼륨 일련 번호: 98ED-C0F6
```

```
C:\python_projects\python_exam 디렉터리
```

```
2023-04-02   오후 11:03   <DIR>           .  
2023-04-02   오후 11:03   <DIR>           ..  
2023-03-24   오후 11:50   <DIR>           Include  
2023-03-24   오후 11:50   <DIR>           Lib  
2023-03-24   오후 11:50           225 pyvenv.cfg  
2023-03-29   오후 10:04   <DIR>           Scripts  
                1개 파일                225 바이트  
                8개 디렉터리  132,255,924,224 바이트 남음
```

```
"""
```

# 30. os

---

- 환경변수, 디렉터리, 파일 등 OS의 자원을 제어함.

```
import os
```

```
# 시스템 명령어의 결과를 파일 객체로 리턴
```

```
f = os.popen("dir")
```

```
# 필요에 따라 파일로 쓸 수도 있다.
```

```
now_dir = os.path.dirname(os.path.abspath(__file__))
```

```
with open(now_dir + "/print.txt", mode="w", encoding="utf-8") as p:  
    p.write(f.read())
```

# 30. os

---

- 환경변수, 디렉터리, 파일 등 OS의 자원을 제어함.

```
import os

print(os.getlogin()) # OS 계정
try:
    os.mkdir("./new_folder") # 폴더 1개만 생성
except FileExistsError:
    # 폴더가 이미 존재함.
    pass

try:
    # 여러개 폴더 동시 생성
    os.makedirs("./new_folder3/test/pic")
except FileExistsError:
    # 폴더가 이미 존재함.
    pass
```

# 30. os

---

- 환경변수, 디렉터리, 파일 등 OS의 자원을 제어함.

```
import os

# 폴더 삭제
# 완전히 비어있는 폴더만 삭제 가능
if os.path.isdir("./new_folder"):
    if len(os.listdir("./new_folder")) == 0:
        os.rmdir("./new_folder3")
    else:
        print("디렉토리가 비어있지 않습니다.")

# 파일 삭제
if os.path.isfile("./pytest/sample1.json"):
    os.remove("./pytest/sample1.json")
```

# 30. os

---

- 환경변수, 디렉터리, 파일 등 OS의 자원을 제어함.

```
import shutil
```

```
# 폴더 삭제
```

```
# 폴더가 비어있지 않더라도 삭제
```

```
shutil.rmtree("./new_folder")
```

# 30. os

---

- 환경변수, 디렉터리, 파일 등 OS의 자원을 제어함.

```
import os
```

```
# 파일시스템 재귀 탐색
```

```
def recursive_path(path, depth):
```

```
    try:
```

```
        dirlist = os.listdir(path)
```

```
    except PermissionError:
```

```
        return
```

```
    for f in os.listdir(path):
```

```
        print(" " * depth, f)
```

```
        if os.path.isdir(path + "/" + f):
```

```
            recursive_path(path + "/" + f, depth + 3)
```

```
recursive_path("C:/", 0)
```

# 파이썬 고급 기능

---

31. 클로저 / 데코레이터

32. 이터레이터 / 제너레이터



## 31. 클로저 / 데코레이터

# 31. 클로저 / 데코레이터 - 클로저

---

- 클로저
  - 정의된 함수에서 내부 함수를 리턴하는 함수.
  - 리턴된 함수는 내부의 변수나 함수를 참조할 수 있다.
- 함수마다 공통된 기능을 처리하고 싶을 때 클로저 혹은 데코레이터를 사용할 수 있다.

# 31. 클로저 / 데코레이터 - 클로저

---

- 클로저 예제 - 카운터 만들기

```
def counter(start_num=0):  
    i = start_num  
    def next():  
        nonlocal i  
        i += 1  
        return i  
    return next
```

```
stepper = counter()
```

```
print(stepper()) # 1  
print(stepper()) # 2  
print(stepper()) # 3  
print(stepper()) # 4  
print(stepper()) # 5
```

# 31. 클로저 / 데코레이터 - 클로저

- 클로저 예제 - 카운터 만들기

```
def counter(start_num=0):  
    i = start_num  
    def next():  
        nonlocal i  
        i += 1  
        return i  
    return next
```

counter namespace { } next namespace

- next 함수는 counter namespace에 포함되어 있으므로 counter 내의 변수에 접근이 가능하다.

# 31. 클로저 / 데코레이터 - 클로저

---

- 클로저 예제 - 카운터 만들기

```
def counter(start_num=0):  
    i = start_num  
    def next():  
        nonlocal i  
        i += 1  
        return i  
    return next  
  
stepper = counter()  
# <function counter.<locals>.next at 0x000001FDE797CB80>  
print(stepper)
```

- counter()를 호출하여 반환받은 stepper는 next() 함수.
- stepper() 로 실행할 때마다 counter namespace의 i 값이 증가한다.

# 31. 클로저 / 데코레이터 - 클로저

---

- 클로저에 함수를 파라미터로 전달하면?

```
def computer(func):  
    def print_result(num1, num2):  
        print("계산을 시작합니다.")  
        result = func(num1, num2)  
        print(f"결과는 {result} 입니다.")  
        return result  
    return print_result
```

```
def calc_add(num1, num2):  
    return num1 + num2
```

```
calc = computer(calc_add)  
print(calc(10, 25)) # 35
```

- calc\_add 함수를 꾸며주는 역할을 수행한다.
- 이런 패턴을 "데코레이터" 라고 한다.

# 31. 클로저 / 데코레이터 - 데코레이터

---

- computer 함수를 python의 데코레이터로 사용하면?

```
def computer(func):  
    def print_result(num1, num2):  
        print("계산을 시작합니다.")  
        result = func(num1, num2)  
        print(f"결과는 {result} 입니다.")  
        return result  
    return print_result
```

**@computer**

```
def calc_add(num1, num2):  
    return num1 + num2
```

```
print(calc_add(10, 25)) # 35
```

- computer 호출 없이도 calc\_add를 직접 사용할 수 있다.

# 31. 클로저 / 데코레이터 - 데코레이터

- 데코레이터는 언제쓸까?
  - 다른 함수를 래핑해 원래코드를 변경하지 않고 기능을 확장하려 할때.

```
def computer(func):  
    def print_result(num1, num2):  
        print("계산을 시작합니다.")  
        result = func(num1, num2)  
        print(f"결과는 {result} 입니다.")  
        return result  
    return print_result
```

확장된 코드

**@computer**

```
def calc_add(num1, num2):  
    return num1 + num2
```

원래코드

```
print(calc_add(10, 25)) # 35
```



## 32. 이터레이터 / 제너레이터

## 32. 이터레이터 / 제너레이터 - 이터레이터

---

- `__iter__` 메소드와 `__next__` 메소드가 구현된 객체.
  - `__iter__` 반복가능 객체 선언하는 special 메소드
  - `__next__ next()` 함수 사용가능 객체 선언하는 special 메소드
- 이터레이터는 `next()` 함수를 호출할 때마다 그 다음 값을 반환한다.

## 32. 이터레이터 / 제너레이터 - 이터레이터

- list 타입은 이터레이터일까?

```
lst = [1, 2, 3]
print(dir(lst)) # ['__add__', '__class__', '__class_getitem__',
'__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

- `__iter__`는 있지만 `__next__`가 없다.
- 따라서 list 타입은 이터레이터가 아니다.

```
lst = [1, 2, 3]
print(next(lst)) # TypeError: 'list' object is not an iterator
```

## 32. 이터레이터 / 제너레이터 - 이터레이터

---

- list를 이터레이터로 변경가능하다.
- next() 를 호출할 때마다 다음 값이 나오며
- 모두 반환 했을 경우 StopIteration 예외가 발생.

```
lst = [1, 2, 3]
i_lst = iter(lst) # list를 iterator로 변경
print(type(i_lst)) # <class 'list_iterator'>
print(next(i_lst)) # 1
print(next(i_lst)) # 2
print(next(i_lst)) # 3
print(next(i_lst)) # StopIteration
```

## 32. 이터레이터 / 제너레이터 - 이터레이터

---

- for 는 이터레이터를 반복해준다.

```
lst = [1, 2, 3]
i_lst = iter(lst) # list를 iterator로 변경
for x in i_lst:
    print(x)
```

- StopIteration 예외가 발생하지 않는다.

## 32. 이터레이터 / 제너레이터 - 이터레이터

- 이터레이터 객체 만들어보기 (`__iter__`, `__next__`)

```
class MyIterator:
    def __init__(self, data):
        self.data = data
        self.position = 0
    def __iter__(self):
        return self # __next__와 함께 선언될 경우, self를 반환
    def __next__(self):
        if self.position >= len(self.data):
            raise StopIteration # 범위를 벗어남.
        result = self.data[self.position]
        self.position += 1 # 인덱스 증가
        return result
```

```
i = MyIterator(list(range(1, 3)))
print(next(i)) # 1
print(next(i)) # 2
# print(next(i)) # StopIteration
```

```
i = MyIterator(list(range(1, 3)))
for x in i:
    print(x)
```

## 32. 이터레이터 / 제너레이터 - 이터레이터

- 반복 가능 객체 만들어보기 (\_\_iter\_\_)

```
class MyIterable:
    def __init__(self, data):
        self.data = data
    def __iter__(self):
        return iter(self.data) # 이터러블 객체로 선언
```

```
i = MyIterable(list(range(1, 3)))
for x in i:
    print(x)
```

- \_\_next\_\_ 없이 \_\_iter\_\_를 선언할 경우 반환할 때,
- iter()함수를 사용해야한다.

## 32. 이터레이터 / 제너레이터 - 제너레이터

---

- 제너레이터는 lazy iterator를 반환하는 특별한 종류의 함수.
- list 처럼 반복할 수 있는 객체
- 제너레이터는 리스트와 달리 내용을 메모리에 저장하지 않고 최소한의 메모리만 사용.
- 이터레이터와 유사함.
  - 반복을 일시중지, 재개 한다는 점에서 다르다.
- 프로그래밍에서 게으른(Lazy)의 의미
  - 미리 실행해두지 않고 무엇인가 필요로할 때에만 실행.
- 즉, 제너레이터는 필요할 때에만 반복을 한다.
  - 반복 후 멈춤. 필요할 때 다시 반복 후 멈춤.



## 32. 이터레이터 / 제너레이터 - 제너레이터

- 제너레이터 예제

```
def get_next_value(n):  
    for x in range(n):  
        # yield 값을 반환하고 반복을 잠시 멈춤.  
        # next()가 호출되면 다시 반복을 재생  
        yield x  
  
seq = get_next_value(3)  
# <generator object get_next_value at 0x000002481DCEF1D0>  
print(seq)  
print(next(seq)) # 0  
print(next(seq)) # 1  
print(next(seq)) # 2  
print(next(seq)) # StopIteration
```

## 32. 이터레이터 / 제너레이터 - 제너레이터

- 제너레이터 예제 – for loop는 이터레이터의 반복을 구현한 반복자.

```
def get_next_value(n):  
    for x in range(n):  
        # yield 값을 반환하고 반복을 잠시 멈춤.  
        # next()가 호출되면 다시 반복을 재생  
        yield x
```

```
seq = get_next_value(3)
```

```
for x in seq:  
    print(x) # StopIteration이 발생하지 않음
```

# 32. 이터레이터 / 제너레이터 - 제너레이터

- 일반for와 제너레이터 비교 예제.
  - 1부터 1000000000까지의 값을 출력해보기

- list에 담기

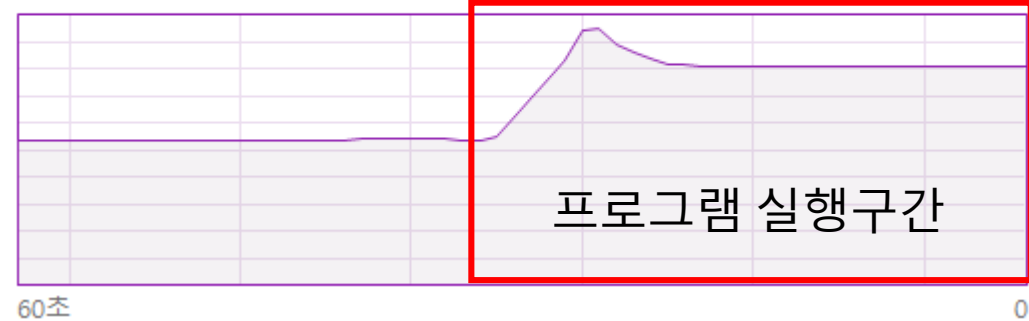
```
def get_range_value(s, e):  
    lst = []  
    for x in range(s, e + 1):  
        lst.append(x)  
    return lst
```

```
for x in get_range_value(1, 1000000000):  
    print(x)
```

- 제너레이터로 해보기

## 메모리

메모리 사용



8.0GB

7.9GB

60초

0

프로그램 실행구간

# 32. 이터레이터 / 제너레이터 - 제너레이터

- 일반for와 제너레이터 비교 예제.
  - 1부터 1000000000까지의 값을 출력해보기

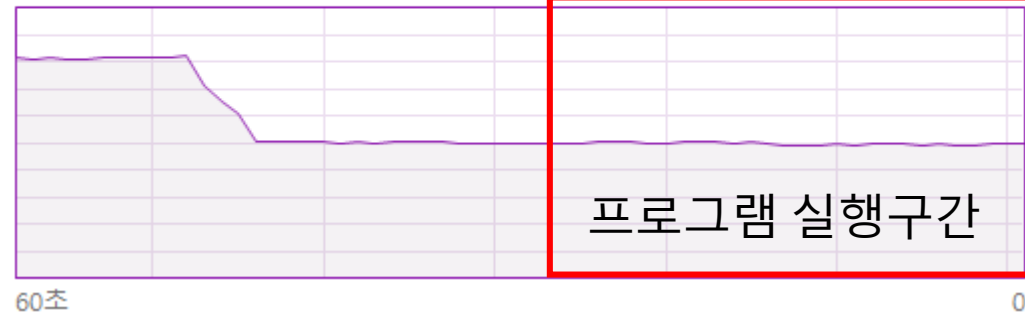
- 제너레이터로 해보기

```
def get_range_value2(s, e):  
    for x in range(s, e + 1):  
        yield x
```

```
for x in get_range_value2(1,1000000000):  
    print(x)
```

## 메모리

메모리 사용



# 감사합니다.

---

Python

장민창

mcjang@hucloud.co.kr