

# 파이썬

공현성

# 목차

1. 파이썬 개요
2. 프로그램 설치
3. 여러 명령들에 대한 설명



# 개요

파이썬은 프로그래밍 언어입니다.

사람이 쓰는 언어는 한국어 영어 다양하죠  
근데 컴퓨터는 이진수만 읽을 수 있습니다.

그렇기에 우리는 프로그래밍 언어를 이용해서 컴퓨터에게 명령을 내려야 해요.

대충 이런 식으로 작동됨(언어마다 아닌 것도 당연히 있음)

- 프로그래밍 언어 -> (모종의 방법으로 이진수로 변환됨) -> 컴퓨터가 읽음

파이썬은 인터프리트 언어(interprete language)라고 해서 소스코드를 바로 실행하는 구조로 되어있습니다.

인터프리트 말고 컴파일 언어(compile language)라는 것도 있는데 이진 소스코드를 한번 실행 파일(윈도우에서는 보통 exe)로 바꾸고 그걸 실행하는 방식을 취하고 있습니다.

파이썬은 인터프리트 언어니까 인터프리터라는 프로그램을 깔아서 내가 쓴 코드를 읽게 해야 합니다.

- 프로그래밍 언어는 종류가 무척이나 다양한데 파이썬이 특히 쓰기 엄청 편함



# 개요(언어비교)

파이썬

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

어셈블리

```
SECTION .data  
    msg db 'Hello World!', 0Ah
```

```
SECTION .text  
global _start
```

```
_start:  
    mov     edx, 13  
    mov     ecx, msg  
    mov     ebx, 1  
    mov     eax, 4  
    int     80h
```

참고로 어셈블리는 모르셔도 됩니다  
모르는게 오히려 좋아요



# 개요

이제 파이썬이 매우 편하고 아름다운 언어라는 것을 알았으니 써봐야겠네요



# 프로그램 설치

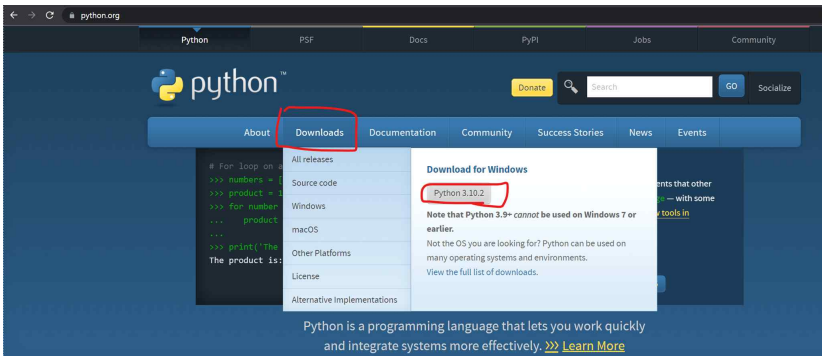
설치해야 되는 건 하나밖에 없습니다. 그런데 IDE(Integrated Development Environment)라고 개발 환경(저희가 앞으로 파이썬을 가지고 여러 프로그램들을 만들게 될건데 뭘 만들기 위해서 필요한 것들을 다 모아놓은 것을 개발환경이라 부릅니다)이 통합되어있고 뭐 그런게 있는데 쉽게 말해서 있으면 파이썬 프로그래밍할 때 삶의 질이 개선됩니다.

그래서 일단 설치를 합시다.



# 프로그램 설치 | Python(필수)

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



The screenshot shows the Python.org homepage. The 'Downloads' menu item is highlighted with a red box. A dropdown menu is open, showing options like 'All releases', 'Source code', 'Windows', 'macOS', 'Other Platforms', 'License', and 'Alternative Implementations'. The 'Windows' option is also highlighted with a red box. To the right, the 'Download for Windows' section is visible, featuring a button for 'Python 3.10.2' which is also highlighted with a red box. Below this, there is a note about Windows 7 compatibility and a link to 'View the full list of downloads.'.

python™

Donate Search GO Socialize

About Downloads Documentation Community Success Stories News Events

All releases  
Source code  
Windows  
macOS  
Other Platforms  
License  
Alternative Implementations

**Download for Windows**

Python 3.10.2

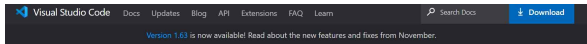
Note that Python 3.9+ cannot be used on Windows 7 or earlier.  
Not the OS you are looking for? Python can be used on many operating systems and environments.  
View the full list of downloads.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

# 프로그램 설치 IDE(선택1)

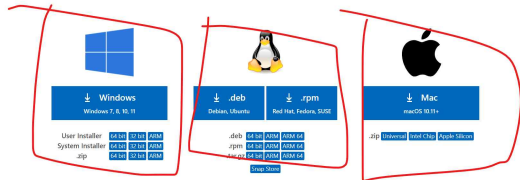
<https://code.visualstudio.com/download>

이거는 자기가 어떤 운영체제를 쓰느냐에 따라서 달라집니다.



## Download Visual Studio Code

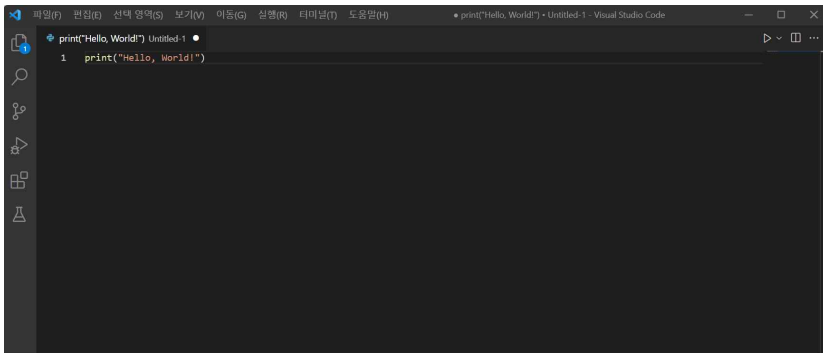
Free and built on open source. Integrated Git, debugging and extensions.





# 프로그램 설치 IDE(선택1)


얘는 설치하면 이렇게 생겼습니다.



# 프로그램 설치 IDE(선택2)

<https://www.jetbrains.com/ko-kr/pycharm/download/#section=windows>

이것도 만약 받는다면 운영체제 맞춰서 설치해야 됩니다. 당연히.



버전: 2021.3.2  
빌드: 213.6777.50  
2022년 1월 31일

시스템 요구 사항  
설치 안내  
기타 버전  
타사 소프트웨어


**PyCharm**  
2022.1 출시 예정 새로운 기능 기능 자세히 보기

## 다운로드 PyCharm

[Windows](#) [macOS](#) [Linux](#)

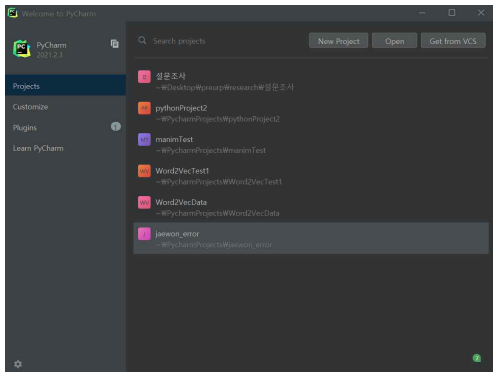
**Professional**  
과학 및 웹 Python 개발용. HTML, JS, SQL 지원.  
[다운로드](#)  
무료 평가판

**Community**  
순수 Python 개발용  
[다운로드](#)  
무료, 오픈 소스로 제공됩니다

 **Toolbox App**을 설치하여 **PyCharm** 및 향후 업데이트를 간편하게 다운로드하세요

# 프로그램 설치 IDE(선택2)

이건 설치하면 이렇게 생겼는데 개인적으로 이걸 추천하긴 합니다.

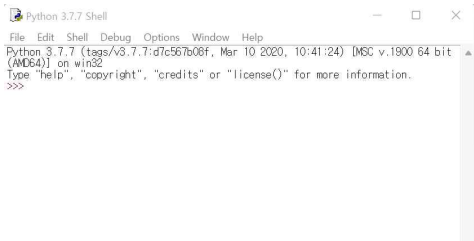


# 프로그램 설치 IDE(선택3)

이건 별도의 설치없이 Python을 깔면 바로 쓸수 있긴 한데 보기가 좀 불편합니다.

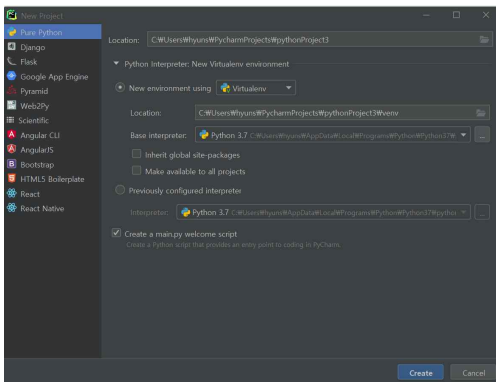
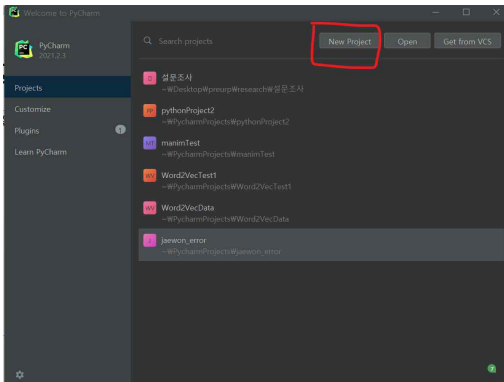
이제 IDE라는 것을 설치하는 목적 중 하나가 Code Highlighting이라고 코드 보기 화려하게 만들어주는게 있어요.

그게 있어야 코드가 난잡해보이지 않고 이뻐보입니다.



# 시작

IDE를 설치하면(저는 PyCharm 기준으로 설명하겠습니다. 이게 제일 기능이 많아요) 이제 처음 열고 프로젝트를 만들어야 됩니다.



# 시작

저 사진같은 화면이 나오면 Location에 C:\Users\hyuns\PycharmProjects\[이름부분] 이런 식으로 되어있는데 이름 부분을  
알아서 기입하시면 됩니다.

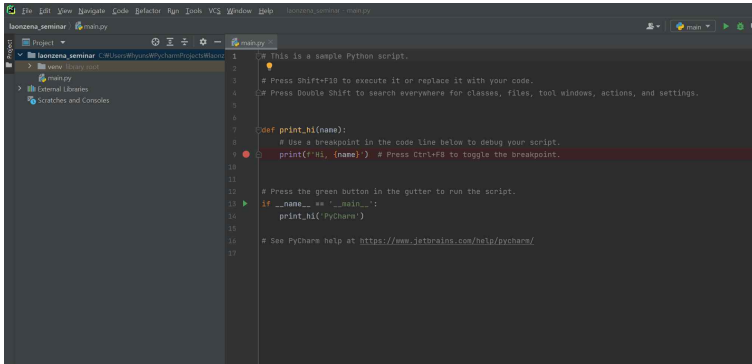
딱히 건들 건 없고 그냥 Create 누르시면 새로운 프로젝트가 생성됩니다.



# 시작

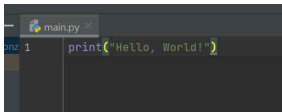
프로젝트 생성하고 조금만 기다리시면 알아서 프로그램이 로딩창 띄우고 계속 로딩하다가 기본 파일을 하나 만들어서 띄울겁니다.

main.py라는 이름을 가지고 있는데 .py가 파이썬 코드 파일 확장자입니다.



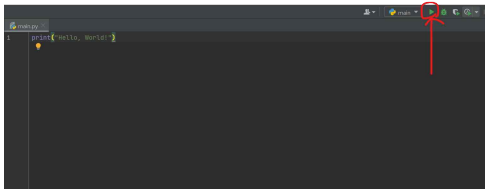
# 내용

안에 들어있는 내용은 썩다 날리시면 됩니다. 그 후에 정말 간단하게 `print("Hello, World!")` 치시고



```
main.py x
1 print("Hello, World!")
```

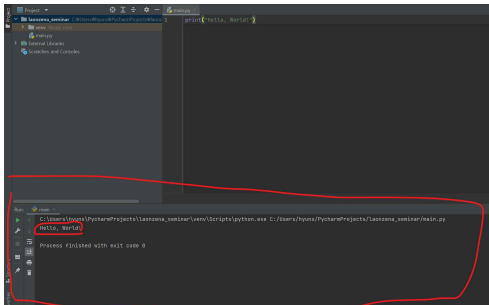
우측 상단에 이 버튼을 눌러봅시다.





# 내용

창이 하나 열리면서 Hello, World!가 출력이 되었습니다. 앞으로 코드를 작성하고 그걸 실행하려면 앞서 본 버튼을 누르면 됩니다.



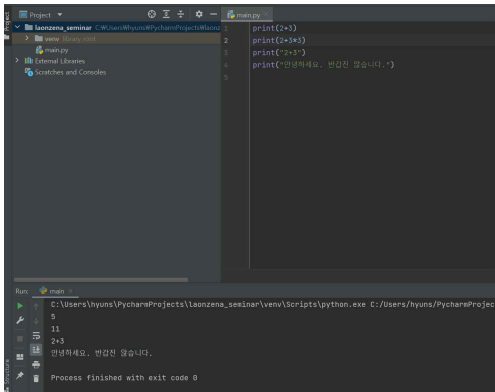
`print()`와 같은 저런 구문을 함수라고 부르는데 함수는 기본적으로 내장되어있는 함수가 있고 저희가 필요에 의해서 새로 만들수도 있습니다. 나중에 할겁니다. `print` 함수는 저희가 딱히 만들진 않았는데 그냥 사용할 수 있으니까 내장 함수입니다.

함수는 이름(내용) 으로 이루어져 있는데 저 내용이라고 적은 부분은 매개변수 부분입니다. 함수에 필요한 값들을 넣어주면 알아서 함수가 값을 반환(여기서는 출력)하게 됩니다.



# 내용

여러 값들을 print를 이용해서 한번 출력해봤습니다.  $2+3$ ,  $2+3*3$ (+\*/가 컴퓨터에서 각각 덧셈, 뺄셈, 곱셈, 나눗셈)처럼 저런 계산은 한번에 한 뒤에 출력을 해주고 “로 감싼 내용은 그냥 출력해주는 것을 알 수 있습니다.



The screenshot shows a Python IDE with a project named 'laonzena\_seminar'. The file 'main.py' is open and contains the following code:

```
1 print(2+3)
2 print(2+3*3)
3 print("2+3")
4 print("안녕하세요. 반갑진 않습니다.")
5
```

The 'Run' window at the bottom shows the output of the script:

```
C:\Users\hyuns\PycharmProjects\laonzena_seminar\venv\Scripts\python.exe C:/Users/hyuns/PycharmProjec
5
11
2+3
안녕하세요. 반갑진 않습니다.
Process finished with exit code 0
```

# 내용

엄청 기본적인 것들을 해보았는데 아직 할 것이 많이 남았습니다.

- 변수(데이터 타입을 포함해서)
- 연산자
- 반복문
- 조건문
- 함수
- import 활용하는 법
- 예외처리
- 객체지향

...

사실 외울게 좀 많습니다.

어쨌든 배우면 R&E, 창의개인연구, 동아리 활동 등 쓸모가 많고 언젠가는 배워야 하니까 아직은 좀 여유로운 학기초에 배워두는 것이 좋습니다.



# 내용 - 변수

그럼 변수부터 다시 시작할게요.

변수가 뭘까요?

말 그대로 변하는 수인데 쉽게 말해서 그냥 그릇같은 겁니다. 그릇에는 여러 물건들을 담을 수 있죠.

변수에는 숫자, 문자 등의 여러 자료(데이터)들을 담을 수 있습니다.

예를 들어서 제가 샌즈의 뼈를 하나 들고 있다고 가정하겠습니다.



애가 샌즈예요. 엄청 유명하니까 다들 아시겠죠? 참고로 모르는 사람을 위해서 말씀드리자면 해골입니다. 몸의 부속품으로 뼈다귀가 있겠죠.

# 내용 - 변수

이 뼈 하나의 무게가 얼마나 될까요?

부위마다 다르겠죠.

제가 만약 갈비뼈 하나를 들고 있다고 합시다. 정확히 얼마나 무게가 나가는지는 모르겠지만 대충 하나당 200g 정도 나간다고 하면

`wasans = 200`

이렇게 하면 됩니다.

지금 뭘한거냐고요?

변수를 하나 선언했(만들었)습니다.

파이썬은 다른 언어들과는 좀 다르게

**변수이름 = 어떤 값**

이렇게만 하면 바로 변수를 만들수 있습니다.



# 내용 - 변수

제가 들고 있는 뼈가 만약 손가락 뼈면 좀 더 가볍겠지요?  
한 25g이라고 하면

```
wasans = 25
```

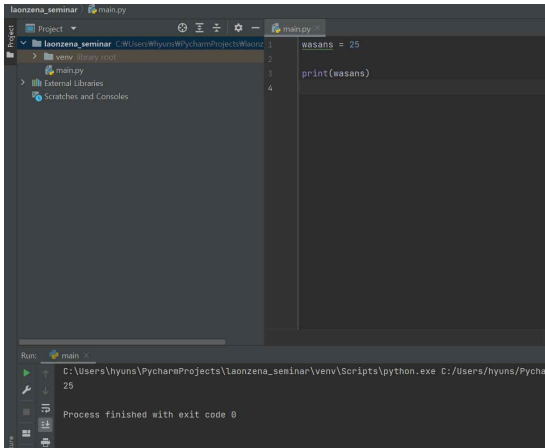
이렇게 하면 됩니다.

값이 하나가 저장되었어요.

이제 print(wasans)를 치면???

잘 나오네요.

wasans 밑에 있는 저 초록색 줄은 무시합니다.  
원래 변수 이름을 짓는데 규칙이 있는데 그 규칙을 무시해서  
저런 줄이 그어지는 겁니다.



The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for running, debugging, and other development tools. The left sidebar displays the project structure for 'laonzena\_seminar', showing a 'venv' directory and a 'main.py' file. The main editor window displays the code in 'main.py':

```
1 wasans = 25
2
3 print(wasans)
4
```

Below the editor, the 'Run' console shows the execution of the script. The command executed is 'C:\Users\hyuns\PycharmProjects\laonzena\_seminar\venv\Scripts\python.exe C:/Users/hyuns/Pyche'. The output is '25', and the status message at the bottom indicates 'Process finished with exit code 0'.

# 내용 - 변수

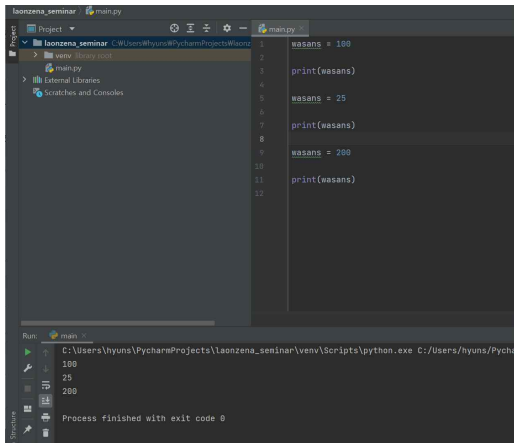
변수는 말그대로 변하는 수니까 그 값을 바꿀 수 있겠죠?  
바로 한번 해봅시다.

이렇게 내용을 바꾸면서 출력해보니까  
순서대로 100, 25, 200이 잘 나오는 것을 알 수 있습니다.

- 참고  
이런 식으로 쓰면 300이 출력됩니다.

```
wasans = 100  
wasans2 = 200
```

```
print(wasans + wasans2)
```



The screenshot shows the PyCharm IDE interface. The top pane displays a Python file named `main.py` with the following code:

```
1 wasans = 100  
2  
3 print(wasans)  
4  
5 wasans = 25  
6  
7 print(wasans)  
8  
9 wasans = 200  
10  
11 print(wasans)  
12
```

The bottom pane shows the Run console output:

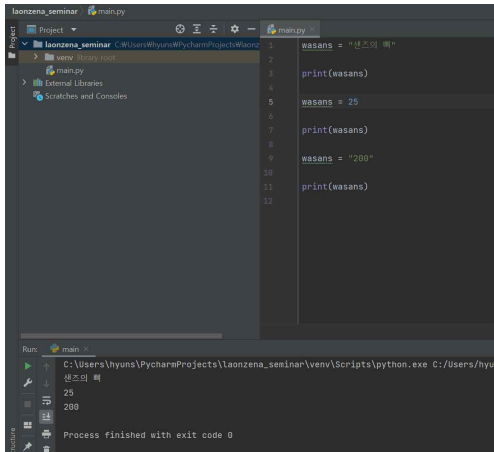
```
Run: main  
C:\Users\hyuns\PycharmProjects\laonzena_seminar\venv\Scripts\python.exe C:/Users/hyuns/Pyche  
100  
25  
200  
Process finished with exit code 0
```

# 내용 - 변수

이렇게 문자를 넣을 수도 있습니다.

사실 문자와 숫자를 같은 변수 내에 넣는 걸 권장하지는 않아요.

(사실 문자랑 숫자를 같은 변수 내에 못넣는다고 알려주려고 코드를 작성했는데 잘 작동해서 좀 놀랐어요. 다른 언어에서는 택도 없는데 파이썬이 역시 매우 너그럽네요)



The screenshot shows the PyCharm IDE interface. The top pane displays a Python file named `main.py` with the following code:

```
1 wasans = "샌즈의 해"  
2  
3 print(wasans)  
4  
5 wasans = 25  
6  
7 print(wasans)  
8  
9 wasans = "200"  
10  
11 print(wasans)  
12
```

The bottom pane shows the Run console output:

```
C:\Users\hyuns\PycharmProjects\laonzena_seminar\venv\Scripts\python.exe C:/Users/hyuns/PycharmProjects/laonzena_seminar/main.py  
샌즈의 해  
25  
200  
Process finished with exit code 0
```



# 내용 - 자료형

이 문자, 숫자 얘기를 왜 갑자기 꺼냈나면  
이제 자료형이라는 걸 설명하려는 자연스러운 흐름의 일부인데

자료형(=데이터형=데이터타입)은 데이터를 여러 종류로 분류해놓은 것인데 저희가 알아야 할 것은

형태	이름	기능
문자	char	말 그대로 문자("안녕", "2+3", "Hello" 등)
숫자	int	정수(-3, 2, 5, 11100 등)
	float(double)	실수(1.2, -3.3, 15251.15215 등)
논리	boolean	참, 거짓(True, False)

정도가 되겠습니다.

원래 다른 언어는 저 자료형 이름마다 변수를 만드는 방법이 다른데 파이썬은 그냥 감사하게도 **변수이름 = 값** 으로 그냥 만들면 됩니다.

이런 식으로

```
char_value = "안녕"
```

```
int_value = 17
```

```
float_value = 342.21422
```

```
boolean_value = True
```



# 내용 - 반복문

갑자기 넘어가는 감이 없지않아 있는데 사실 딱히 변수에서 알려드릴진 더 없습니다.

반복문은 말 그대로 어떤 구문을 반복하게 하는데 노가다를 획기적으로 줄일 수 있습니다.

예를 들어서 갑자기 괴한이 집에 침입해서 총을 머리에 겨누고 1분내로 파이썬으로 wasans를 1000번 출력하는 프로그램을 만들 어내라고 협박할 경우 Ctrl C+V를 사용할 수 도 있겠지만 정확히 1000번인지는 모르고 또 시간도 많이 걸리잖아요?

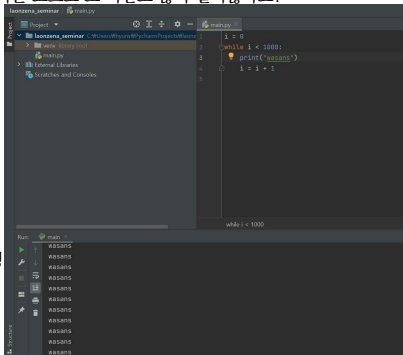
그때는 그냥 이렇게 하면 긴박한 상황에서 목숨을 건질 수 있습니다.

```
i = 0
while i < 1000:
    print("wasans")
    i = i + 1
```

i라는 변수를 하나 만들고 0을 넣습니다. 정확히 1000번 실행하기 위해서 실행한 횟수를 카운트 해주는 변수로 사용할 예정입니다.

while <- 이게 반복문 을 이용하는데

**while 조건:** 이렇게 적으면 주어진 조건을 만족할 경우에 아래의 명령을 실행 합니다. 이때 아래의 명령의 기준은 while 명령보다 띄어쓰기가 Tab 하나만 큼 더 되어있는 것들입니다.



# 내용 - 반복문

```
i = 0  
while i < 1000:  
    print("wasans")  
    i = i + 1
```

이제 print로 wasans를 출력하고 그 다음줄로 넘어가서 i에 i+1을 대입(즉 i 값을 하나 증가)시킵니다. 이렇게 i를 증가해서 더 이상 i < 1000 이라는 조건을 만족하지 않을때까지 명령을 계속해서 수행합니다. 즉, 이 프로그램을 실행하면 i가 0부터 999까지 총 1000번 wasans가 출력되게 됩니다.



# 내용 - 반복문

한번 별을 출력하는 프로그램을 만들어볼까요?

이때 별은

\*

\*\*

\*\*\*

\*\*\*\*

...

과 같이 출력이 되어야 합니다. 한 20정도까지만 할까요?

쉽게 생각해서

i, j = 1, 0 <- 이렇게 하면 여러 변수들을 한번에 값을 대입해줄 수 있습니다.

```
while i <= 20:
```

```
    while j < i:
```

```
        print("*")
```

```
        j = j + 1
```

```
    i = i + 1
```

```
    j = 0 <- 이걸 안하면 j 값이 유지되서 반복문이 원하는 방향으로 출력되지 않습니다.
```

이렇게 하면?



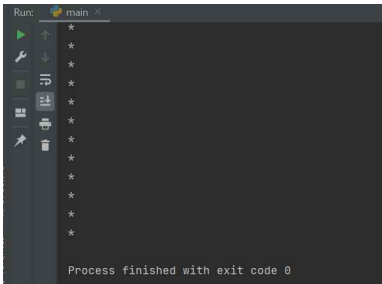
# 내용 - 파이썬 문서

???????

이렇게 나옵니다.  
왜 그럴까요?

이건 print라는 함수에 비밀이 숨겨져 있는데  
지금껏 별 생각없이 보셨을 수도 있겠지만 print는  
사실 안의 내용물을 출력하면 바로 개행(줄바꿈)을  
해버립니다.

이걸 제가 어떻게 알았을까요??  
print는 비교적 간단하니까 경험상 알았을 수도 있지만 뭐든 확실한게 좋잖아요?



The screenshot shows a 'Run' window in a Python IDE. The window title is 'Run: main'. On the left, there is a vertical toolbar with icons for running, stepping through, and other debugging actions. The main area of the window displays the output of a program: ten asterisks (\*) arranged vertically, one per line. At the bottom of the window, a status bar indicates 'Process finished with exit code 0'.

# 내용 - 파이썬 문서

<https://docs.python.org/3.10/library/functions.html#print>

파이썬 공식 홈페이지에서 제공해주는 문서에 들어가보면

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

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *objects* are given, `print()` will just write *end*.

The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead.

Whether the output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

*Changed in version 3.3:* Added the *flush* keyword argument.

이런 식으로 뭐가 많이 써져 있는데 결국 if no objects are given: 만약 매개변수가 주어지지 않는다면, `print()` will just write end: `print`함수는 `end`를 작성할 것이다.

`end`는 위에 `\n` 으로 나와있습니다. 저 “한국 돈의 단위(원)”표시가 영어 자판에는 “\” 로 나와있습니다. 동일한 역할이에요.



# 내용 - 특수 문자들

저 end의 \n이 뭘까요?  
바로 개행문자라는 겁니다.

파이썬을 비롯한 여러 언어에는 저런 특수한 역할을 하는 문자들을 정의해놨는데  
보통 **\[어떤 알파벳]** 이런 식으로 정의되어 있습니다. 종류가 몇개 있는데 사용하게 된다면 나중에 알려드리겠습니다.



# 내용 - print

그래서 다시 돌아와서 결국 end라는 내용(매개변수)를 입력해줄 수 있는데 저희가 안해줘서 이 print라는 친구가 미리 end에 입력되어있었던 \n를 입력한겁니다.

그러면 그걸 바꿔버리면 되겠죠?

간단하게 `print("*", end="")`

이렇게 하면 end가 비워지니까 줄이 바뀌는 문제가 사라집니다.

i, j = 1, 0

while i <= 20:

while j < i:

print("\*", end='')

j = j + 1

print("") <- 이제 줄이 바뀌는 걸 따로 추가해줬어요. 그냥 원래 print를 이용하면 자동으로 줄바꿈이 되니까 그렇게 넣어놨어요.

i = i + 1

j = 0





## 내용 - 별을 줄력

이제 잘 나오네요.

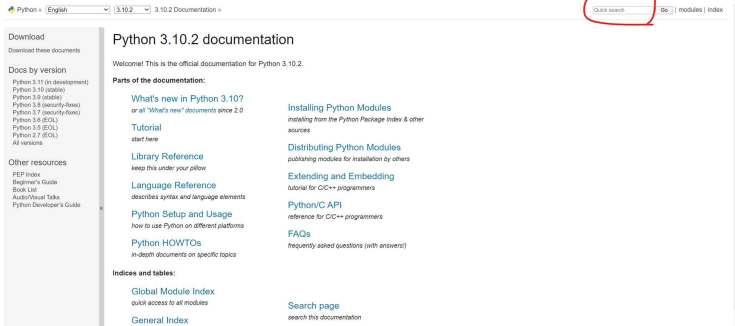
[illegible]

# 내용 - 여담

이렇게 파이썬에서 공식으로 제공하는 문서를 들춰보면 여러 기능들을 알 수 있습니다.  
매우 친절하게 설명이 되어있어요.

<https://docs.python.org/3.10/>

빠른 탐색으로 원하는 것만 바로 검색해서 찾아볼 수 있습니다.



Python » English » 3.10.2 » 3.10.2 Documentation »

Quick search  Go | modules | index

## Download

Download these documents

### Docs by version

- Python 3.11 (in development)
- Python 3.10 (stable)
- Python 3.9 (stable)
- Python 3.8 (security-fixes)
- Python 3.7 (security-fixes)
- Python 3.6 (EOL)
- Python 3.5 (EOL)
- Python 2.7 (EOL)
- All versions

### Other resources

- PEP Index
- Beginner's Guide
- Book List
- Audio/Visual Talks
- Python Developer's Guide

## Python 3.10.2 documentation

Welcome! This is the official documentation for Python 3.10.2.

### Parts of the documentation:

- [What's new in Python 3.10?](#)  
or all "What's new" documents since 2.0
- [Tutorial](#)  
start here
- [Library Reference](#)  
keep this under your pillow
- [Language Reference](#)  
describes syntax and language elements
- [Python Setup and Usage](#)  
how to use Python on different platforms
- [Python HOWTOs](#)  
in-depth documents on specific topics
- [Installing Python Modules](#)  
installing from the Python Package Index & other sources
- [Distributing Python Modules](#)  
publishing modules for installation by others
- [Extending and Embedding](#)  
tutorial for C/C++ programmers
- [Python/C API](#)  
reference for C/C++ programmers
- [FAQs](#)  
frequently asked questions (with answers!)

### Indices and tables:

- [Global Module Index](#)  
quick access to all modules
- [General Index](#)
- [Search page](#)  
search this documentation



# 내용 - 여담

```
i = 1
```

```
while i < 20:  
    print("*" * i)  
    i = i + 1
```

사실 이렇게 하면 훨씬 간단하게 할 수 있습니다.

파이썬은 문자열을 이렇게 곱해서 여러번 출력하게 할 수 있기 때문에 이런 식으로 할 수도 있어요.

한번 스스로 여러번 연습해보는 활동을 해봅시다.

구구단 출력하기 이런거 한번 해보세요.



# 내용 - 반복문을 계속쓰면?

반복문은 한번만 써도 여러번 반복이 됩니다.

그러면 그걸 엄청 여러번 쓰면??

렉이 걸리겠죠..

```
while 조건:  
    while 조건:  
        while 조건:  
            ...
```

이런 식으로 계속 쓰면 시간이 매우 오래 걸리는 비효율적인 코드가 완성이 됩니다.  
사실 이런 일이 많이 생기진 않겠지만 그래도 효율적인 알고리즘을 만드는게 생각보다 중요합니다.



# 내용 - 시간복잡도

여기서 시간복잡도라는 개념이 나오는데 반복문을 어떻게 쓰느냐에 따라서 얼마나 시간이 걸리냐를 따져보는 겁니다.

$$n^2 + 2n + 12211221$$

이런 수식이 있습니다.

$n=1, 2, 3 \dots$  이렇게 점점 커집니다.

처음에는 12211221의 존재감이 너무 크겠죠

근데  $n$ 이 10000정도만 돼도  $n$  제곱이 벌써 12211221 과 비슷하고

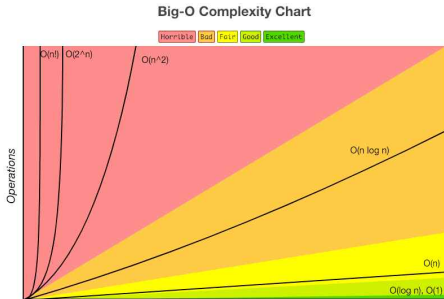
$n$ 이 커지면 커질수록 12211221은 그냥 아무것도 아닌게 되겠죠.

$2n$ 도 있지만  $n^2$ 의 존재감이  $2n$ 보다 훨씬 더 큼니다.

이렇게 반복문이 얼마나 오래걸리느냐를 이런 수식 비스무리 한 것으로 나타내서 최고차항만을 계수 포함해서 표기하게 되는데( $n$ 이 엄청 커지면 다른 항들은 그냥 개미정도로 작아집니다) 이런 표기법을 **빅오(Big-O) 표기** 이라고 합니다.

$O(n)$ ,  $O(2n)$ ,  $O(n^2)$  등으로 표현하는데 위의 수식에서는  $O(n^2)$ 으로 표현하겠죠.

빅오 표기법으로 대략적인 그래프를 그려서 어떤 반복문이 얼마나 오래 걸리느냐를 나타낸 차트가 있는데 당연히 빨간색이 오래 걸리는거고 안좋은 알고리즘입니다.



# 내용 - 조건문

시간 복잡도는 이쯤해서 넘어가고 반복문도 넘어갑시다. 사실 for라는 반복문이 하나 더 있긴 한데 나중에 알려드릴게요. 안알려드리는 건 아닙니다.

이제 배워야될게 조건문입니다.

if 조건:

여러 명령들

else:

또 다른 여러 명령들

if 하고 조건이 적혀있는데 주어진 조건이 만족한다면 여러 명령들을 실행하고  
만족하지 않는다면?(else) 또 다른 여러 명령들을 실행합니다.



# 내용 - 조건문

뭔가를 배웠으니까 간단하게 실습을 해봐야겠죠.

```
a = 1212 * 2
```

```
b = 54/12 * 21 + 1234
```

```
if a < b:
```

```
    print("a가 b보다 큼니다")
```

```
else:
```

```
    print("b가 a보다 큼니다")
```

이렇게 하고 실행하면 b가 a보다 큼니다.가 출력됩니다.

근데 만약 a와 b가 우연히 같으면 어떻게 될까요?

그래도 b가 a보다 큼니다. 가 출력이 될겁니다. 왜냐하면 if a<b: 에서 a<b라는 조건이 만족되지 않았으니까요...

이거 문제가 있어보이네요?



# 내용 - 조건문

그래서 새로운 조건을 더 추가할 겁니다

**elif 조건:** 을 이용하면 if 말고도 여러 조건을 더 추가할 수 있습니다. 참고로 else if의 줄임말입니다.

```
a = 1212 * 2
```

```
b = 606 * 4
```

```
if a < b:
```

```
    print("a가 b보다 큼니다")
```

```
elif a == b:
```

```
    print("놀랍게도 a와 b가 같습니다")
```

```
else:
```

```
    print("b가 a보다 큼니다")
```

놀랍게도 a와 b가 같네요.

근데 a와 b가 같은지 확인하는 조건에 ==가 들어가 있습니다. 이게 뭘까요??





# 내용 - 기본 연산자

==, <, > 같은 것들은 기본 연산자라고 부릅니다.

파이썬에는

산술 연산자 (Arithmetic Operators)

비교 연산자 (Comparison Operators)

할당 연산자 (Assignment Operators)

논리 연산자 (Logical Operators)

비트 연산자 (Bitwise Operators)

멤버 연산자 (Membership Operators)

식별 연산자 (Identity Operators)

이렇게 무려 7개나 존재하는데요?

산술, 비교, 할당, 논리 연산자만 우선 배우고 나머지는 나중에 필요할 때 더 설명드리겠습니다.



# 내용 - 기본 연산자

산술 연산자 (Arithmetic Operators)

너무나도 당연한 것들이라서 설명할 부분이 없습니다. 숨 쉬는 것 다음으로 자연스럽게 할 수 있도록 하면 됩니다.

연산자	설명
+	더하기
-	빼기
*	곱하기
**	거듭제곱(3 ** 4는 3 <sup>4</sup> 이다)
/	나누기
//	나누기 후 소수점 부분을 버림(즉, 몫만 구함, 5//2는 2)
%	나누기 후 나머지만 구함(5%2는 1)

# 내용 - 기본 연산자

## 비교 연산자 (Comparison Operators)

여기도 설명할 것이 크게 없으나 왜 같다가 =가 아니라 ==인지 설명하자면 =는 이미 변수에서 wasans = 100 이런 식으로 값을 넣는 데에 사용하고 있어서 겹치니까 ==로 대체한 겁니다.

연산자	설명
==	같다(a==b는 a와 b가 같다는 뜻)
!=	다르다
>	왼쪽이 더 크다
<	오른쪽이 더 크다
>=	왼쪽이 더 크거나 같다
<=	오른쪽이 더 크거나 같다

# 내용 - 기본 연산자

할당 연산자 (Assignment Operators)

연산자	설명
=	대입(wasans = 100)
+=	왼쪽 변수에 오른쪽을 더함( $i+=1$ , $i$ 에 1을 더한다)
-=	왼쪽 변수에 오른쪽을 뺌( $i-=1$ , $i$ 에 1을 뺀다)
*=	왼쪽 변수를 오른쪽으로 곱한 결과를 넣음
/=	왼쪽 변수를 오른쪽으로 나눈 결과를 넣음
%=	왼쪽 변수를 오른쪽으로 나눈 결과의 나머지를 넣음
//=	왼쪽 변수를 오른쪽으로 나눈 결과의 몫을 넣음
**=	왼쪽 변수의 오른쪽 변수만큼의 제곱을 한 결과를 넣음

# 내용 - 기본 연산자

논리 연산자 (Logical Operators)

논리 연산자는 참, 거짓의 논리에 대한 연산자입니다.

연산자	설명
and	양쪽이 둘다 참이면 참
or	양쪽 중 한쪽만 참이면 참
not	논리 상태를 반전시킴

조건1 and 조건2 는 조건1과 조건2가 모두 참일 때 True(참)이고  
조건1 or 조건2는 조건1또는 2중 하나만 참이어도 True(참)입니다.  
not 조건1은 조건이 참이면 거짓으로 거짓이면 참으로 바꿔줍니다.

다음은 예제입니다. 실행해보면 저 3문장이 모두 출력됩니다.

a, b = 3, 2

if a > 2 and b > 1:

print("a가 2를 넘고 b가 1을 넘습니다")

if a > 2 or b > 1:

print("a가 2를 넘거나 b가 1을 넘습니다")

if not a < 2:

print("a가 2보다 작지 않습니다")



# 내용 - 조건 + 반복

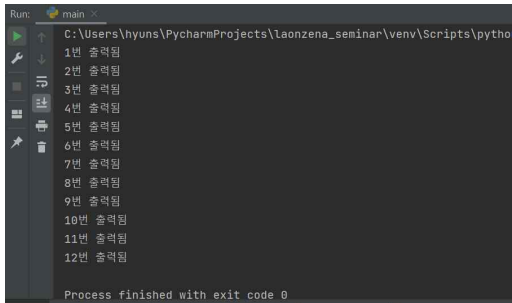
조건문과 반복문을 둘다 알았으니 이 둘을 한번 합쳐봅시다.

```
i = 1
while True:
    print(str(i) + "번 출력됨")
    if i == 10:
        break
    i += 1
```

이렇게 쓰고 실행을 해보면?

옆처럼 나오게 됩니다.

전에 보지 못한 명령들이 좀 많으니까  
한번 보도록 하죠.



```
Run: main x
C:\Users\hyuns\PycharmProjects\laonzena_seminar\venv\Scripts\python.exe
1번 출력됨
2번 출력됨
3번 출력됨
4번 출력됨
5번 출력됨
6번 출력됨
7번 출력됨
8번 출력됨
9번 출력됨
10번 출력됨
11번 출력됨
12번 출력됨
Process finished with exit code 0
```

# 내용 - 조건 + 반복

```
i = 1
while True:
    print(str(i) + "번 출력됨")
    if i == 10:
        break
    i += 1
```

우선 위의 명령에서 True는 참을 의미하는데 while 조건: 에서 조건이 참이면 반복문이 계속 돌아간다고 했었습니다. 근데 조건에 그냥 참(True)이 들어가 있으니까 이 반복문은 별다른 일이 없으면 영원히 실행될 겁니다.

근데 중간에 멈춘걸 보니까 반복문을 중지시키는 요인이 있다는 것을 알 수 있는데 그게 바로 break 구문입니다. 반복문 안에 break라는 명령이 들어가 있으면 가장 가까운 반복문이 한개가 정지됩니다. 여기서도 조건문 안에 break가 들어있으니까 조건을 만족하면? break랑 가장 가까운 while True: 이게 하나가 정지되는 겁니다.

```
while True:
    while True:
        while True: <-
            break
```

이런 식으로 써놓으면 화살표로 가리켜져 있는 반복문 하나만 정지되겠죠?



# 내용 - 조건 + 반복

```
i = 1
while True:
    print(str(i) + "번 출력됨")
    if i == 10:
        break
    i += 1
```

계속해서 print 함수 안에 보면 str이라는 구문이 있는데 이진 함수입니다.  
이름(구문)과 같은 것들은 보통 함수입니다.

파이썬에서는 문자를(사실 문자가 한개가 아니라 여러개가 있으니까 문자열이라고 해야됩니다) +를 이용해서 합칠 수가 있는데  
숫자 + 문자를 하면 당연히 에러가 나옵니다. 에러에 관한건 뒤에 부록에 넣어놔으니까 한번 보세요.

어쨌든 str은 내부의 숫자를 문자로 바꿔주고 문자 + 문자는 가능하니까 (어떤 숫자)번 출력됨 이 10번 나오는 코드가 완성이 되는 겁니다.





# 내용 - 함수

지금까지는 함수를 쓰기만 했습니다.

print와 같은 것들이 함수입니다. 아직 몇개 안쓰긴 했는데 파이썬에는 수많은 함수가 있습니다.

그리고 나중에 배우겠지만 import를 이용하면 세상에 사는 다른 사람이 만들어놓은 함수도 사용할 수 있습니다.

그러면 저희도 한번 만들어봐야겠죠?

함수는 여러 반복되는 명령들을 묶어놓은 놈 정도로 생각하시면 됩니다.



# 내용 - 함수

```
def print_hello(a): # 이걸 print 함수입니다.  
    print("hello\nhello")  
    print("hello" + a)
```

또 새로운게 막 등장했습니다.

우선 함수는 **def 함수이름(함수의인자, 매개변수):** 로 구성이 됩니다.

지금은 print\_hello라는 이름의 함수를 새로 만들었고 a라는 인자를 받게 되어있습니다.

즉, 저 함수를 사용하려면

print\_hello("안녕") 이런 식으로 괄호 안에 뭔가를 넣어야 합니다.

저기있는 #은 뭘까요?

저건 주석이라는 건데 어떤 줄에 #이 있으면 그 문자 뒤로는 전부 인터프리터가 나중에 코드를 읽을 때 무시하는 부분이 됩니다.

코드의 설명을 달아놓을 때 주로 사용합니다.

print 안에 hello\nhello라고 적어놨는데 \n은 앞서도 말했듯이 줄을 바꿔주는 특수문자로 저걸 출력하면

hello

hello

이렇게 출력이 됩니다.



# 내용 - 함수

이렇게 쓰면 이런 식으로 출력됩니다.

```
main.py x
1
2 def print_hello(a): # 이걸 print 함수입니다.
3     print("hello\nhello")
4     print("hello" + a)
5
6
7 print_hello("안녕")
8
9
```

```
C:\Users\hyuns\PycharmProjects\laor
hello
hello
hello안녕

Process finished with exit code 0
```



# 내용 - 함수

한가지 유의할 점은 모든 건 항상 정의 후에 사용해야 한다는 점입니다.

오른쪽처럼 print\_hello라는 것을 정의(def)하기 전에 먼저 호출(사용)한다면 저렇게 빨간줄이 그어지게 됩니다.

```
main.py x
1
2 def print_hello(a): # 이걸 print 함수입니다.
3     print("hello\nhello")
4     print("hello" + a)
5
6
7 print_hello("안녕")
8
9
```

```
1
2 print_hello("안녕")
3
4 def print_hello(a): # 이걸 print 함수입니다.
5     print("hello\nhello")
6     print("hello" + a)
7
8
9
```

# 내용 - 함수

근데 여기에 숫자를 넣으면 어떻게 될까요?

```
1
2 def print_hello(a):
3     print("hello\nhello")
4     print("hello" + a)
5
6     print_hello(111)
```

당연히 에러가 납니다

```
C:\Users\hyuns\PycharmProjects\laonzena_seminar\venv\Scripts\python.exe C:/Users/hyuns/Pychar
hello
hello
Traceback (most recent call last):
  File "C:/Users/hyuns/PycharmProjects/laonzena_seminar/main.py", line 6, in <module>
    print_hello(111)
  File "C:/Users/hyuns/PycharmProjects/laonzena_seminar/main.py", line 4, in print_hello
    print("hello" + a)
TypeError: can only concatenate str (not "int") to str

Process finished with exit code 1
```



# 내용 - 함수

함수의 마지막 줄에 "hello" + a에서 a가 숫자이기에 숫자 + 문자 라는 연산을 할 수가 없어서 생기는 문제인데

```
C:\Users\hyuns\PycharmProjects\laonzena_seminar\venv\Scripts\python.exe C:/Users/hyuns/Pych
hello
hello
Traceback (most recent call last):
  File "C:/Users/hyuns/PycharmProjects/laonzena_seminar/main.py", line 6, in <module>
    print_hello(111)
  File "C:/Users/hyuns/PycharmProjects/laonzena_seminar/main.py", line 4, in print_hello
    print("hello" + a)
TypeError: can only concatenate str (not "int") to str

Process finished with exit code 1
```

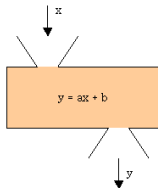
보통 그래서 "hello" + str(a)로 한번 처리를 한 뒤에 사용하거나

함수를 사용하는 사람들에게 있어서 한번 알려주는 목적으로

def print\_hello(a: str): 과 같이 표기하여 a에는 str(문자열)이 들어와야 한다고 알려줄 수 있습니다.



# 내용 - 함수



이런 그림이 있죠

원래 함수는 뭘 넣으면 뭐가 나오는 형태로 되어있는데  
파이썬에서의 함수에는 그런게 없는 것처럼 보입니다.

하지만 그렇지 않습니다.

# 내용 - 함수

return 구문을 이용하면 무언가를 반환(내뱉는)하는 함수를 만들 수 있습니다.

```
def add_num(a: int, b: int):  
    return a + b
```

```
print(add_num(2, 3))
```

한번 이렇게 입력해봅시다. 그리고 실행하면?

5  
가 출력됩니다.

add\_num이라는 함수를 만들고 그걸 사용했더니 사용한 위치에 a와 b를 더한 값이 들어갔는데 이러한 역할을 하는 것이 return 구문입니다.





# 내용 - 함수

지역변수, 전역변수

우리가 원래 변수라는 것을 잘 만들고 살았는데 만약 이런 일이 있다면 어떻게 될까요?

```
def hello():  
    num = 20  
    print(num)
```

```
hello()
```

```
num += 10
```

```
hello()
```

실행이 잘 되다가 저 빨간 줄에서 오류가 발생합니다.

지역 변수라는 것은 함수 내부에서 만들어진 변수인데 이런 변수들은 함수가 종료되면 사라지게 됩니다.



# 내용 - 함수

지역변수, 전역변수

그렇다면 전역변수는?

함수 밖에서 만들어진 변수들입니다.

이 친구들은 프로그램이 종료되기 전까진 사라지지 않고 함수 내부에서도 잘 이용할 수 있습니다.

```
x = 50
```

```
def hello():  
    x -= 20
```

```
print(x)  
hello()  
print(x)
```

이렇게 하면?

50

30이 순서대로 잘 출력이 됩니다.



# 내용 - 함수

그렇다면 함수 내에서 만든 변수는 모두 지역변수일까요?

그렇지 않습니다

global이라는 구문을 이용하면 됩니다

```
def hello():
```

```
    global a
```

```
    a = 10
```

```
    print(a)
```

```
hello()
```

```
print(a)
```

이렇게 하면 함수 내에서 만들어졌더라도 a는 전역변수가 됩니다.

참고로 global a = 10과 같이 대입과 global은 한번에 사용할 순 없습니다.



# 내용 - 리스트

아까 변수하면서 자료형 설명할 때 말하지 않았던게 있습니다.

바로 리스트라는 건데 그냥 데이터 여러개를 묶은 겁니다.

`list = ['wa', 'sans', 'ashi', 'nun', 'guna']` # 참고로 문자열은 “” 말고 “”로 묶어도 됩니다.

리스트는 이렇게 대괄호 안에 콤마로 각 데이터들을 구분해서 넣어놓는데  
저렇게 만들어진 리스트를 어떻게 쓰냐면

`list[0]` 이런 식으로 이름[숫자]로 각 항목을 불러오거나 수정하는 등의 활동을 수행할 수 있습니다.

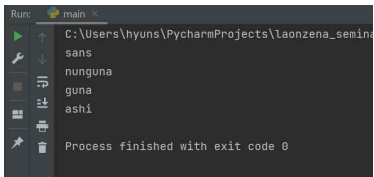
예제로 보면 좀 더 쉽겠죠?

# 내용 - 리스트

```
list = ["wa", "sans", "ashi", "nun", "guna"]
```

```
print(list[1])  
print(list[3] + list[4])  
print(list[-1])  
print(list[-3])
```

리스트를 만들고 실행하면?

A screenshot of a Python IDE's console window. The window title is 'Run: main'. On the left, there is a vertical toolbar with icons for running, stepping through, and other debugging actions. The main area of the console shows the output of the program: 'C:\Users\hyuns\PycharmProjects\laonzena\_seminar' (on the first line), 'sans' (on the second), 'nunguna' (on the third), 'guna' (on the fourth), and 'ashi' (on the fifth). At the bottom, it says 'Process finished with exit code 0'.

```
Run: main ×  
C:\Users\hyuns\PycharmProjects\laonzena_seminar  
sans  
nunguna  
guna  
ashi  
Process finished with exit code 0
```

이렇게 나옵니다.  
차근차근 살펴봅시다.



# 내용 - 리스트

```
list = ["wa", "sans", "ashi", "nun", "guna"]
```

```
print(list[1])  
print(list[3] + list[4])  
print(list[-1])  
print(list[-3])
```

우선 list[1]을 출력(print)하니까 wa가 아니라 sans가 나왔습니다.

컴퓨터에서는 숫자를 1부터가 아니라 0부터 세기 때문에 wa를 출력하려면 list[0]을 적어야 합니다.

list[3] + list[4]를 하니까 nunguna가 출력이 잘 되었습니다.

list[-1]은 뭘까요?

바로 오른쪽에서부터 세는 겁니다. 이걸 -1이 가장 오른쪽인데 가장 오른쪽을 0으로 하면 가장 왼쪽과 겹치니까 구분하려고 이렇게 만든 겁니다.

list[-3]은?

오른쪽에서 3번째니까 ashi가 출력됩니다.

# 내용 - 리스트

리스트와 관련된 여러 명령들이 있습니다.

```
list = ["wa", "sans", "ashi", "nun", "guna"]  
list2 = ["cham", "goro", "gupna"]  
print(len(list))
```

를 입력하면 리스트의 길이인 5가 출력됩니다.

list.remove("wa") 를 입력하면 wa가 리스트에서 제거됩니다.

list.append("wawa") 를 입력하면 wawa가 리스트 맨 뒤쪽에 추가됩니다.

list.index(숫자, "wawa") 를 입력하면 숫자 자리에 입력한 위치에 wawa가 추가됩니다.

list.expend(list2) 를 입력하면 list뒤에 list2의 모든 원소(리스트 내의 각 데이터들을 이르는 말)가 붙습니다.



# 내용 - 리스트 + 튜플, 딕셔너리, 세트

리스트 외에도 Tuple, Dictionary라는 리스트 비슷한 무언가도 존재합니다.

튜플은 간단히 말해서 내용을 수정할 수 없는 리스트라고 생각하시면 됩니다.

```
tuple = (1, 2, "3")
```

정의는 이렇게 합니다. 리스트도 그렇고 전부 해당되는 내용인데 각각의 자료형 묶음 안에는 숫자, 문자 등이 섞여 들어가도 됩니다.

딕셔너리는 키값과 내용값으로 구분되는데

키를 통해서 내용을 검색할 수가 있습니다. 사전처럼요.

```
dictionary = {"ddd":"www", 2:3, 15:'2332'} # 키값: 내용값 으로 내용을 구분
```

```
print(dictionary[15])  
print(dictionary["ddd"])
```

이렇게 입력하면 각각 2332, www가 출력됩니다.

정말 당연하겠지만 키는 중복해서 적을 수 없습니다.



# 내용 - 리스트 + 튜플, 딕셔너리, 세트

Set는 영어로 집합인데 말 그대로 수학에서의 집합을 나타냅니다.

```
set = {12, 13, 4242}
```

와 같이 딕셔너리처럼 중괄호를 사용하지만 키가 없습니다.

Set는 다른 데이터 집합과는 좀 다르게 데이터 순서에 의미가 없습니다. 또한 중복되는 원소를 가질 수 없습니다. 리스트나 튜플, 딕셔너리는 원소가 중복될 수 있습니다.

```
set.add("망고")
```

갑자기 망고를 추가하면 그냥 set라는 집합안에 잘 스며들어갑니다.

set1, set2가 있다고 가정합시다.

set1 | set2 : 합집합  
set1 & set2 : 교집합  
set1 - set2 : 차집합

과 같이 집합의 연산도 할 수 있습니다.

set = set()라고 입력하면 공집합이 만들어집니다.

그 뒤에 set = {"여러", "값들"}을 넣어서 새로이 설정할 수 있습니다. 공집합을 {"여러", "값들"}로 대체하는 겁니다.



# 내용 - 리스트

모든 문자열은 기본적으로 리스트입니다.

```
text = "apple"
```

이라는 문자열이 있으면

```
print(text[2])
```

이런 식으로 문자를 고를 수 있고

```
print(text[2:4])
```

이런 식으로 범위를 선택할 수 있습니다.

한번 실행해보시고 판단하시길 바랍니다.



# 내용 - 리스트

문자열 말고 리스트도 범위를 지정할 수 있습니다.  
문자열이 리스트니까 당연한 내용이지요.

```
list = [2, 3, "221", "2323", 55]
```

```
print(list[1:3])
```

이름[범위1:범위2]  
로 범위를 지정하면  
범위1 <= 그 사이 숫자들 < 범위2 의 내용이 전부 출력됩니다.

한쪽의 범위를 지정하지 않으면 그 범위 방향으로 끝까지 출력됩니다.

```
list[1:]
```

```
list[:3]
```

```
list[:-1]
```

이런 식으로 쓸 수 있고 한번 직접 넣어보면서 확인해보시면 될 것 같습니다.



# 내용 - 모듈

뭔가 많은 것을 했는데 머릿속에 잘 들어갔을지는 잘 모르겠네요.

사실 이제 더 많은 것을 해야 합니다.  
많이 연습해보셔야 합니다.

이제 모듈에 대해 알려드리겠습니다.

모듈은 파이썬에서 여러 기능들을 함수로 만든 뒤에 그것들을 묶어서 모아둔 것인데

파이썬에서 기본적으로 제공하는 내장모듈이 있고 사람들이 스스로 만들어서 배포하는 모듈도 존재합니다.  
파이썬은 결국 이 모듈에 대한 시스템이 굉장히 잘 되어있기에 다른 언어들보다 쓰기 편합니다.

모듈을 하나 불러와서 그냥 그 안의 기능을 쓰기만 하면 됩니다.



# 내용 - 모듈

```
import 모듈이름  
from 모듈이름 import 이름
```

의 2가지 방법을 이용할 수 있습니다.

첫번째 방법은 모듈의 전체를 불러오는 방법이고 두번째 방법은 필요한 것만 가져오는 방법입니다

```
import random # 랜덤한 수를 출력하기 위한 모듈입니다.
```

```
print(random.random())
```

위의 코드를 실행하면 임의의 랜덤한 수가 하나 출력됩니다.

보통 사용할 때에는 모듈이름.이름() 이런 식으로 사용하는데 모듈 내에 어떤 기능이 있는지는 직접 찾아보셔야 합니다.



# 내용 - 모듈

다른 사람들이 만든 새로운 모듈을 불러오고 싶다면

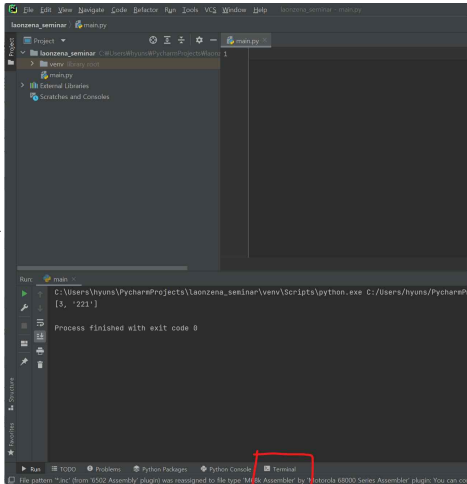
터미널을 이용해 새로운 모듈을 설치해야 합니다.

**pip install 모듈이름** 을 입력하면?

모듈이름이 있다는 가정하에 원하는 모듈을 온라인으로 받을 수 있습니다.

모듈이름은 인터넷에 파이썬을 치고 필요한 기능을 입력하면 보통 모듈이름도 함께 나옵니다.

보통 모듈마다 신경써야 할 것이 다르기에 그런 것은 블로그나 여러 매체들을 통해 스스로 습득하셔야 합니다.



## 내용 - 모듈

matplotlib라는 모듈을 한번 이용해보겠습니다.

```
Terminal: Local x + v
새로운 기능 및 개선 사항에 대한 최신 PowerShell을 설치하세요! https://aka.ms/PSWindows

PS C:\Users\hyuns\PcharmProjects\laonzena_seminar> pip install matplotlib
Collecting matplotlib
Using cached matplotlib-3.5.1-cp37-cp37m-win_amd64.whl (7.2 MB)
Collecting pyparsing>=2.2.1
Using cached pyparsing-3.0.7-py3-none-any.whl (98 kB)
Collecting pillow>=6.2.0
Downloading Pillow-9.0.1-cp37-cp37m-win_amd64.whl (3.2 MB)
| ████████████████████████████████████████ | 3.2 MB 6.4 MB/s
Collecting kiwisolver>=1.0.1
Using cached kiwisolver-1.3.2-cp37-cp37m-win_amd64.whl (51 kB)
Collecting cycler>=0.10
Using cached cycler-0.11.0-py3-none-any.whl (6.4 kB)
```

이 모듈은 수학과 관련된 모듈인데 워낙 유명해서 자료도 많고 공식 웹사이트에서 튜토리얼도 볼 수 있습니다.



# 내용 - 모듈

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# example data
x = np.arange(0.1, 4, 0.1)
y1 = np.exp(-1.0 * x)
y2 = np.exp(-0.5 * x)
```

```
# example variable error bar values
y1err = 0.1 + 0.1 * np.sqrt(x)
y2err = 0.1 + 0.1 * np.sqrt(x/2)
```

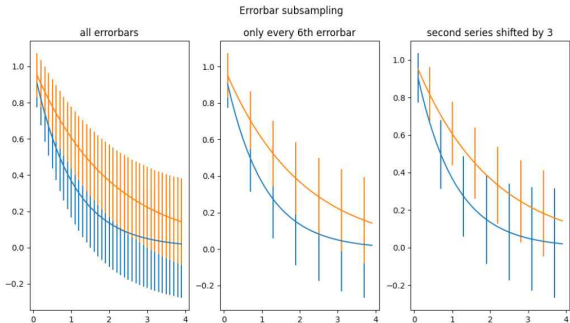
```
fig, (ax0, ax1, ax2) = plt.subplots(nrows=1, ncols=3, sharex=True,
                                     figsize=(12, 6))
```

```
ax0.set_title('all errorbars')
ax0.errorbar(x, y1, yerr=y1err)
ax0.errorbar(x, y2, yerr=y2err)
```

```
ax1.set_title('only every 6th errorbar')
ax1.errorbar(x, y1, yerr=y1err, errorevery=6)
ax1.errorbar(x, y2, yerr=y2err, errorevery=6)
```

```
ax2.set_title('second series shifted by 3')
ax2.errorbar(x, y1, yerr=y1err, errorevery=(0, 6))
ax2.errorbar(x, y2, yerr=y2err, errorevery=(3, 6))
```

```
fig.suptitle('Errorbar subsampling')
plt.show()
```





# 내용 - 모듈

뭔가 많죠?

제가 짠 코드는 아니고 matplotlib 사이트에서 예시로 보여주는 것들중 하나입니다.

[https://matplotlib.org/stable/gallery/lines\\_bars\\_and\\_markers/errorbar\\_subsample.html#sphx-glr-gallery-lines-bars-and-markers-errorbar-subsample-py](https://matplotlib.org/stable/gallery/lines_bars_and_markers/errorbar_subsample.html#sphx-glr-gallery-lines-bars-and-markers-errorbar-subsample-py)

만약 이 모듈에 대해 더 알고싶으시다면

[https://matplotlib.org/stable/users/getting\\_started/index.html](https://matplotlib.org/stable/users/getting_started/index.html)

여기 들어가셔서 차근차근 보시면 됩니다.

# 내용 - Anaconda, R언어

번외로 Anaconda라는 과학 계산용으로 1400여개의 모듈(패키지라고도 합니다)들을 모아놓은 게 있는데 이것 설치하면 여러 모듈들을 따로 다운받을 수고를 덜 수 있습니다.

아나콘다(Anaconda)는 파이썬, R언어 이렇게 2개의 언어에서 사용이 가능한데 R언어는 파이썬보다 더 과학의 분석에 초점이 맞추어져 있는 언어로 저희 동아리 선생님이신 박현종 선생님이 나중에 알려드릴겁니다.



# 내용 - try, except 구문

오류는 예상하지 못한 부분에서 발생하게 됩니다. 예상했으면 벌써 고쳤겠죠.

원래 프로그래밍 언어란 것이 오류가 나면 프로그램이 정지하게 됩니다.

try, except 는 오류가 나면 프로그램이 종료되지 않고 계속 실행할 수 있게 해주는 구문이라 생각하시면 됩니다.

명령을 시도(try)했는데 오류가 나면 예외(exception) 처리를 해주는 겁니다.

```
try:  
    print(4/0)  
except:  
    print("와 어디서 오류가 났는데 어딘지 모르겠다")
```

```
print("hello")
```

위의 예시처럼 정말 알기 어려운 오류가 났을 때나 별거 아닌 오류라서 그냥 넘겨도 될때

오류가 날 수 있는 여러 부분들을 try, except로 묶어놓으면 됩니다.



# 내용 - try, except 구문

이외에도 finally라는 구문도 있는데

```
try:  
    print("안녕")  
except:  
    print("정말 중요한 문장인데 오류가 만나서 출력이 안됨 ㅋㅋ")
```

이런 경우가 있을 때 finally라는 구문을 추가해두면 오류가 나든 안나든 실행을 합니다.

```
f = open('foo.txt', 'w')  
try:  
    # 무언가를 수행한다.  
finally:  
    f.close()
```

이런식으로 보통은 파일을 열고(리소스를 사용) 사용한 뒤의 리소스를 닫을 때 사용합니다. try가 되든 말든 알아서 finally에 있는 놈들은 실행됩니다.



# 내용 - try, except 구문

```
try:
    ...
except 발생 오류1:
    ...
except 발생 오류2:
    ...
```

와 같이 오류의 원인에 따라 except를 나눠놓을 수 있습니다.

```
try:
    a = [1,2]
    print(a[3])
4/0
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except IndexError:
    print("인덱싱 할 수 없습니다.")
```

이런 예제가 있는데 ZeroDivisionError와 같은 예러가 무엇인지 궁금하다면 부록을 만들어놓았으니 참고해주세요.



# 내용 - try, except 구문

```
try:  
    ...  
except ZeroDivisionError as e:  
    print(e)
```

이렇게 하면 에러를 출력할 수 있습니다.

참고하셔서 활용하시면 좋을 것 같습니다.

```
try:  
    f = open("나없는파일", 'r')  
except FileNotFoundError:  
    pass
```

이렇게 하면 에러를 그냥 건너뛰(pass) 수 있습니다.



# 내용 - try, except 구문

```
def hello():  
    raise NotImplementedError
```

와 같이 raise 구문을 이용하면 오류를 스스로 창조할 수도 있는데 이런 짓을 왜하냐면 hello라는 함수를 나중에 만들 때 실수로 까먹고 안만들었을 경우 시기적절하게 오류를 띄워주기 위함입니다.

NotImplementedError는 파이썬 내장 오류로, 꼭 작성해야 하는 부분이 구현되지 않았을 경우 일부러 오류를 일으키기 위해 사용됩니다.



# 내용 - 객체지향

사실 엄청 중요한건데 파이썬을 사용하는데에 초점이 맞춰져 있어서 많이 설명하지는 않을겁니다. 궁금하면 더 찾아보시면 됩니다.

객체지향이라는 것은 프로그래밍을 하는 하나의 패러다임으로 쉽게 말해서 그냥 방식이라고 생각하시면 될 것 같습니다. 또 다른 방식으로는 절차지향이 있습니다.

원래 명령은 순서대로 하나하나 출력이 되는 것이 기본인데(절차지향)

파이썬과 같은 객체지향 언어는 객체라는 것들을 만들고 객체에 속성을 부여한 뒤 객체 간의 상호작용으로 프로그램이 만들어지고 작동됩니다.

class라는 구문을 이용합니다.

자세한 내용은 <https://wikidocs.net/84> 여기를 참고하시면 될 것 같습니다.





# 부록 - 에러 처리하는 법

정말 중요한 파트인데 부록으로 빼서 설명합니다.

사실 에러라는 건 별게 아닌데 컴퓨터가 잘못된건 당연히 없고 사람이 뭔가를 이상하게 해서 생깁니다.

PyCharm에서는 사람이 실수를 하면 정말 친절하게도 어디가 잘못되었고 뭐가 잘못되었는지 알려줍니다. 사실 이렇게 PyCharm(인터프리터)가 잡아주는 오류는 별게 아니고 그냥 아무 문제가 없어 보이는데 잘 작동하지 않는게 가장 짜증나긴 합니다.

에러 창을 보니까 Line 1, 첫번째 줄에 `print(12 + "안녕?")` 이라는 구문이 `TypeError`라는 에러를 발생시켰다고 합니다.




The screenshot shows the Run window in PyCharm. The title bar says 'Run: main'. The console output shows the command: `C:\Users\hyuns\PycharmProjects\laonzena_seminar\venv\Scripts\python.exe C:/Users/hyuns/PycharmPr`. Below that, it says 'Traceback (most recent call last):'. Then, it shows the file path: `File "C:/Users/hyuns/PycharmProjects/laonzena_seminar/main.py", line 1 in <module>`. The line `print(12 + "안녕?")` is highlighted in red. Below that, the error message is: `TypeError: unsupported operand type(s) for +: 'int' and 'str'`. At the bottom, it says 'Process finished with exit code 1'. A red circle highlights 'line 1' in the traceback.

```
Run: main ×
C:\Users\hyuns\PycharmProjects\laonzena_seminar\venv\Scripts\python.exe C:/Users/hyuns/PycharmPr
Traceback (most recent call last):
  File "C:/Users/hyuns/PycharmProjects/laonzena_seminar/main.py", line 1 in <module>
    print(12 + "안녕?")
TypeError: unsupported operand type(s) for +: 'int' and 'str'
Process finished with exit code 1
```

# 부록 - 에러 처리하는 법

TypeError: unsupported operand type(s) for +  
라는 에러는 결국 int(정수)와 str(문자열)을 덧셈 기호로 합치는 것이 불가능하기 때문에 이를 시도해서 발생한 문제인데

해결책을 알면 그냥 그 줄에 가서 해결을 하시면 되고 모르시면 Google에 치시면 그냥 다 나옵니다.



[검색](#) [인쇄](#) [뉴스](#) [동영상](#) [소셜](#) [이미지](#) [도구](#)

검색결과 약 114,000개 (0.41초)

[https://www.inflearn.com/questions/unsupported-operand-type\(s\)-for-+-int-and-str-질문드립니다...](https://www.inflearn.com/questions/unsupported-operand-type(s)-for-+-int-and-str-질문드립니다...)  
unsupported operand type(s) for +: 'int' and 'str' 질문드립니다... 아맞나, - 2020.11.13.  
user\_input = input("계산식을 입력하세요").

<https://hashcode.co.kr/questions/문자열에-정수를-붙이는데-안돼요-Hashcode>  
python - 문자열에 정수를 붙이는데 안돼요 | Hashcode  
2016. 2. 16. --- 문자열에 정수를 붙이는데 안돼요 그런 어색함이 지나고 777? 해커 내용.  
TypeError: unsupported operand type(s) for +: 'int' and 'str' 소스코드.

<https://daewonyoon.tistory.com/1...>  
[Python] TypeError: unsupported operand type(s) for -: 일괄블록  
2021. 5. 3. --- (most recent call last): File "opyshell.py", line 1, in <module> 1 + "3"  
TypeError: unsupported operand type(s) for +: 'int' and 'str'.

<https://poppy-leni.tistory.com/entry/Python-TypeErr...>  
unsupported operand type(s) for +: 'int' and 'str' - Poppy\_Leni ...  
2017. 8. 18. --- [Python] TypeError: unsupported operand type(s) for +: 'int' and 'str' ... - 1.  
int형을 문자열형으로 강제종다. 우리는 str () 내용합수를 이용 ...

<https://stackoverflow.com/questions/11400000/unsupported-op...>  
Unsupported operand type(s) for +: 'int' and 'str' - Stack Overflow  
2013. 12. 7. --- Closed 5 years ago. I am currently learning Python so I have no idea what is  
going on. num1 = int(input)...  
답변 2개 · 인기 답변: You're trying to concatenate a string and an integer, which is incorrect. C...  
TypeError: unsupported operand type(s) for +: 'str' and 'int'    답변 2개    2010년 3월 4일  
TypeError: unsupported operand type(s) for +: 'int' and 'str' ...    답변 1개    2020년 1월 29일  
TypeError: unsupported operand type(s) for +: 'str' and 'int'    답변 4개    2021년 4월 6일  
unsupported operand type(s) for +: 'str' and 'int' error in python    답변 2개    2020년 11월 17일  
[stackoverflow.com 검색결과 더보기](#)

# 부록 - 에러 처리하는 법

보통은 Stack Overflow라는 프로그래밍 커뮤니티 사이트를 참고합니다.

자주 발생하는 오류는 다음과 같습니다.

1. ValueError
2. IndexError
3. SyntaxError
4. NameError
5. ZeroDivisionError
6. FileNotFoundError
7. TypeError
8. AttributeError
9. KeyError
10. OverflowError



# 부록 - 에러 유형 1

## 1. ValueError

함수가 부적절한 값을 받았을 때 발생하는 에러입니다.

예를 들어 `int("와센즈")`라는 구문을 입력했다고 가정합니다.

`int()` 함수는 안에 "1212" 와 같이 문자로 된 숫자가 들어올 때 그것을 숫자 자료형으로 바꿔주는 역할을 합니다.  
근데 와센즈는 숫자가 아니죠.

그래서 에러가 발생하게 됩니다.



# 부록 - 에러 유형 2

## 2. IndexError

보통 리스트에서 생기는 오류인데 리스트와 같이 묶음으로 된 자료형에서 없는 값을 찾으려 하거나 범위를 벗어나는 값을 찾으려 하면 발생합니다.

wasans = ['1', '2', '3'] 라는 리스트가 있다고 합시다.

리스트이름.index()를 치면 리스트 내의 값을 찾아서 그 위치를 표시해줍니다.

근데 wasans.index('W') 이렇게 치면???

W는 당연히 wasans라는 리스트 안에 없기 때문에 에러가 발생하게 됩니다.

# 부록 - 에러 유형 3

## 3. SyntaxError

파이썬 문법(= 구문 = 명령)을 잘못 사용하면 발생하는 오류입니다.

```
while True  
    print("와센즈")
```

이런 코드를 쳤다고 합시다.

근데 while 조건: 에서 이 콜론(:)이 빠졌네요.

그러면 이런 오류가 발생합니다. 대부분은 IDE(PyCharm) 이 미리미리 잡아줍니다.



# 부록 - 에러 유형 4

## 4. NameError

선언하지 않은 변수 이름을 불러오려고 하면 발생하는 에러입니다.

```
ll = 12  
print(ll)
```

ll(LL) 이라는 변수를 만들었는데 print에는 ll(LI)를 출력하라고 하니까 만든 적이 없는 변수는 에러가 뜨겠죠?



# 부록 - 에러 유형 5

## 5. ZeroDivisionError

수를 0으로 나누려고 하면 발생합니다

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

이라고 치면 생깁니다. 간단하죠?





# 부록 - 에러 유형 6

## 6. FileNotFoundError

컴퓨터에 없는 파일을 불러오려고 하면 발생합니다. 이름이 직관적이어서 좋네요.

컴퓨터에는 test.xlsx라는 파일이 있는데 불러올 때

```
f = open("text.xlsx", "r")
```

를 입력하면 없는 파일을 불러오려 했으니까 오류가 출력됩니다.



# 부록 - 에러 유형 7

## 7. TypeError

잘못된 데이터 타입을 사용하였을 때 생기는 에러입니다

```
a = 1 + "asdf"
```

라고 입력하면 숫자와 문자를 더하는게 될리가 없으니까 Type(자료형)에 관한 에러가 나오겠네요.



# 부록 - 에러 유형 8

## 8. AttributeError

Attribute는 속성인데 객체.메서드 <- 이 부분이 속성입니다. 없는 속성을 불러오려 하면 생기는 에러입니다.

```
import math
```

```
a = math.ceil(1.2)
```

```
b = math.cell(1.2)
```

math.ceil은 소수점을 올림해주는 명령인데 math.cell이라는 구문은 없습니다.

그래서 에러가 발생하게 됩니다.



# 부록 - 에러 유형 9

## 9. KeyError

Dictionary에 접근할 때 해당하는 키가 없으면 발생합니다.

```
d = {"a": 12, "b": 33}
```

```
result = d["z"]
```

이렇게 치면 당연히 z라는 키가 d 내부에 없으니까 오류가 뜹니다.



# 부록 - 에러 유형 10

## 10. OverflowError

연산의 결과가 너무 큰 경우 발생하는 에러입니다.

파이썬은 정수 자료형은 이 오류가 발생하지 않습니다. Arbitrary-precision arithmetic 라는 방식을 이용하는데 그냥 이 방식을 쓰면 엄청 큰 연산의 결과도 받아들일 수 있다고만 아시면 됩니다.

어쨌든 실수 연산을 할 때

`sys.float_info.max`를 치면 나오는  $1.7976931348623157 \times 10^{308}$  이상의 수는 에러가 납니다. 사실 이렇게 큰 수를 다룰 일이 있을진 잘 모르겠어요.



# 부록 - 에러 유형 10 여담

```
infinity = float("inf")
```

```
print(infinity)  
print(infinity / 10000)
```

실수에서 가장 큰 수는 무한대라고 합니다...

위의 코드를 실행하면 둘다 inf라는 값이 나와요.

```
class float([x])
```

Return a floating point number constructed from a number or string x.

If the argument is a string, it should contain a decimal number, optionally preceded by a sign, and optionally embedded in whitespace. The optional sign may be '+' or '-'; a '+' sign has no effect on the value produced. The argument may also be a string representing a NaN (not-a-number), or positive or negative infinity. More precisely, the input must conform to the following grammar after leading and trailing whitespace characters are removed:

```
sign      ::= "+" | "-"  
infinity  ::= "Infinity" | "inf"  
nan       ::= "nan"  
numeric_value ::= floatnumber | infinity | nan  
numeric_string ::= [sign] numeric_value
```

# 기타

소중한 방학 시간을 할애해서 만든 자료입니다.

이게 부족하다 생각하면

<https://wikidocs.net/book/1>

<https://wikidocs.net/book/2>

여기 가시면 됩니다... 좋은 내용들이 많아요.

별로 길지는 않으니까 금방 해내실 수 있으실 겁니다.

오타, 오류 지적 매우 환영합니다.

