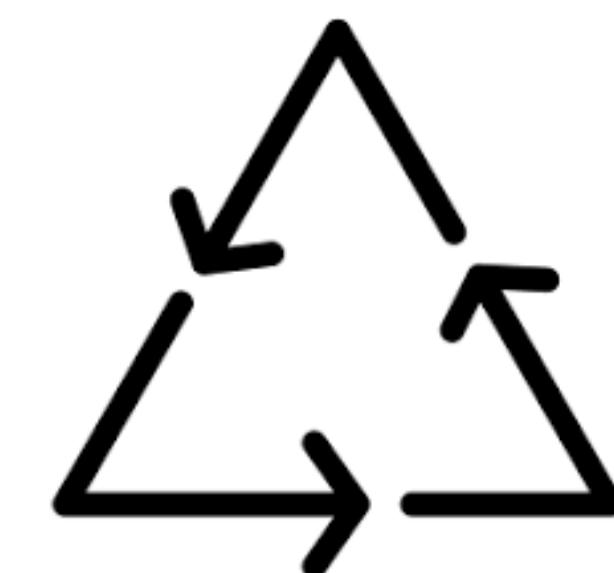




CHAPTER 2

파이썬 프로그래밍 환경

Python Programming Environment



박진수 교수
서울대학교·경영대학
jinsoo@snu.ac.kr



학습 목차

Lab : IDLE 둘러보기

Lab : 셀 명령어 기초

디버깅

명령어 셀 실행 환경

프로그램과 프로그램 개발 절차

IDLE 둘러보기

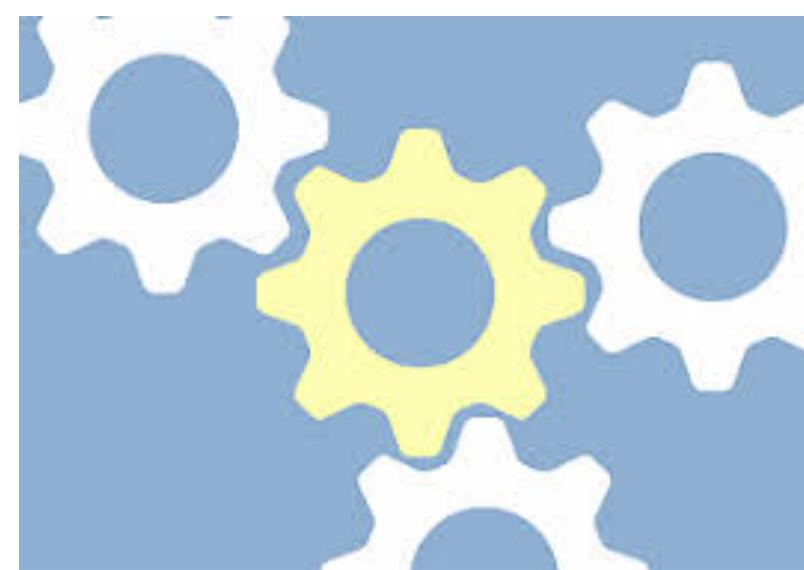
Lab Exercises





IDLE 기능 둘러보기

Configuring IDLE





IDLE 실행하기



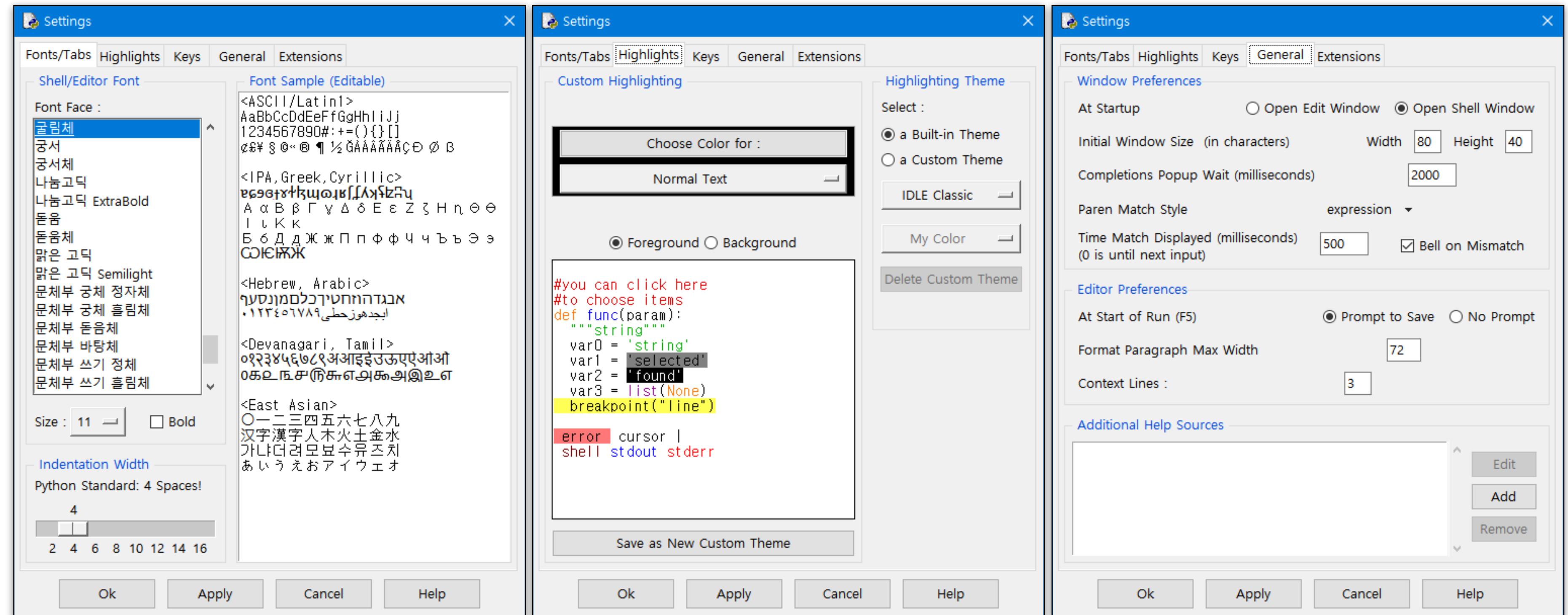
● IDLE 실행하기

- [Windows 검색]에서 ‘IDLE’ 입력 → [IDLE (Python 3.x 64-bit)] 항목 선택

● 옵션 메뉴에서 환경설정을 통한 IDLE 개인화

- [Options] → [Configure IDLE]

● [Settings] 창



● 도움말

- [Help] → [IDLE Help]



IDLE

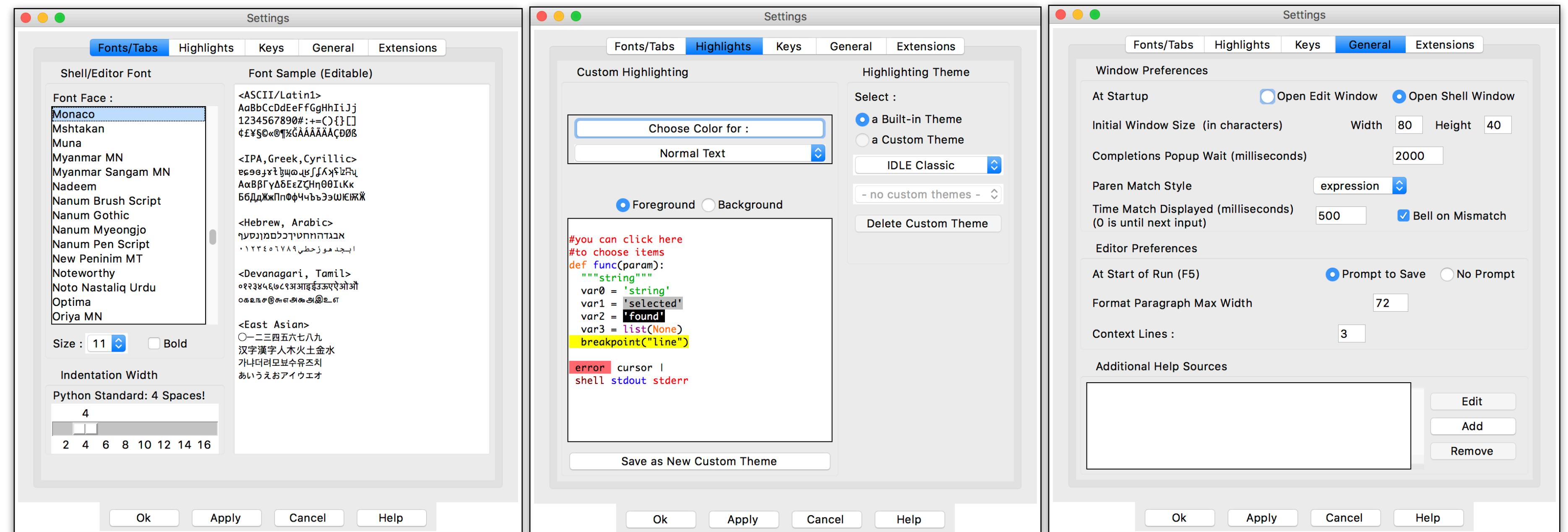


● IDLE 실행하기

- [command](또는 [control]) + [space bar]
- [Spotlight 검색] 창에서 ‘IDLE’ 입력 → [IDLE] 항목 더블 클릭

● 메뉴에서 환경설정을 통한 IDLE 개인화

- [IDLE] → [Preferences...]
- [Settings] 창



● 도움말

- [Help] → [IDLE Help]



IDLE 샘플 코드 맛보기



Tasting IDLE Turtle Demo





Windows

Turtle 데모 예시 파일 열기



- [Help] → [Turtle Demo] → [Examples] → [round_dance] → [START] 버튼 클릭 → 실행

round_dance - a Python turtle graphics example

Examples Fontsize Help

```
"""
    turtle-example-suite:
        tdemo_round_dance.py

(Needs version 1.1 of the turtle module that
comes with Python 3.1)

Dancing turtles have a compound shape
consisting of a series of triangles of
decreasing size.

Turtles march along a circle while rotating
pairwise in opposite direction, with one
exception. Does that breaking of symmetry
enhance the attractiveness of the example?

Press any key to stop the animation.

Technically: demonstrates use of compound
shapes, transformation of shapes as well as
cloning turtles. The animation is
controlled through update().
"""

from turtle import *
def stop():
    global running
    running = False
def main():
    global running
    clearscreen()
    bgcolor("gray10")
    tracer(False)
    shape("triangle")
    f = 0.793402
    phi = 9.064678
    s = 5
    c = 1
    # create compound shape
    sh = Shape("compound")
    for i in range(10):
        sh.addshape(sh.getshape())
        shapesize(s)
        p = get_shapopoly()
        s *= f
        c *= f
main()

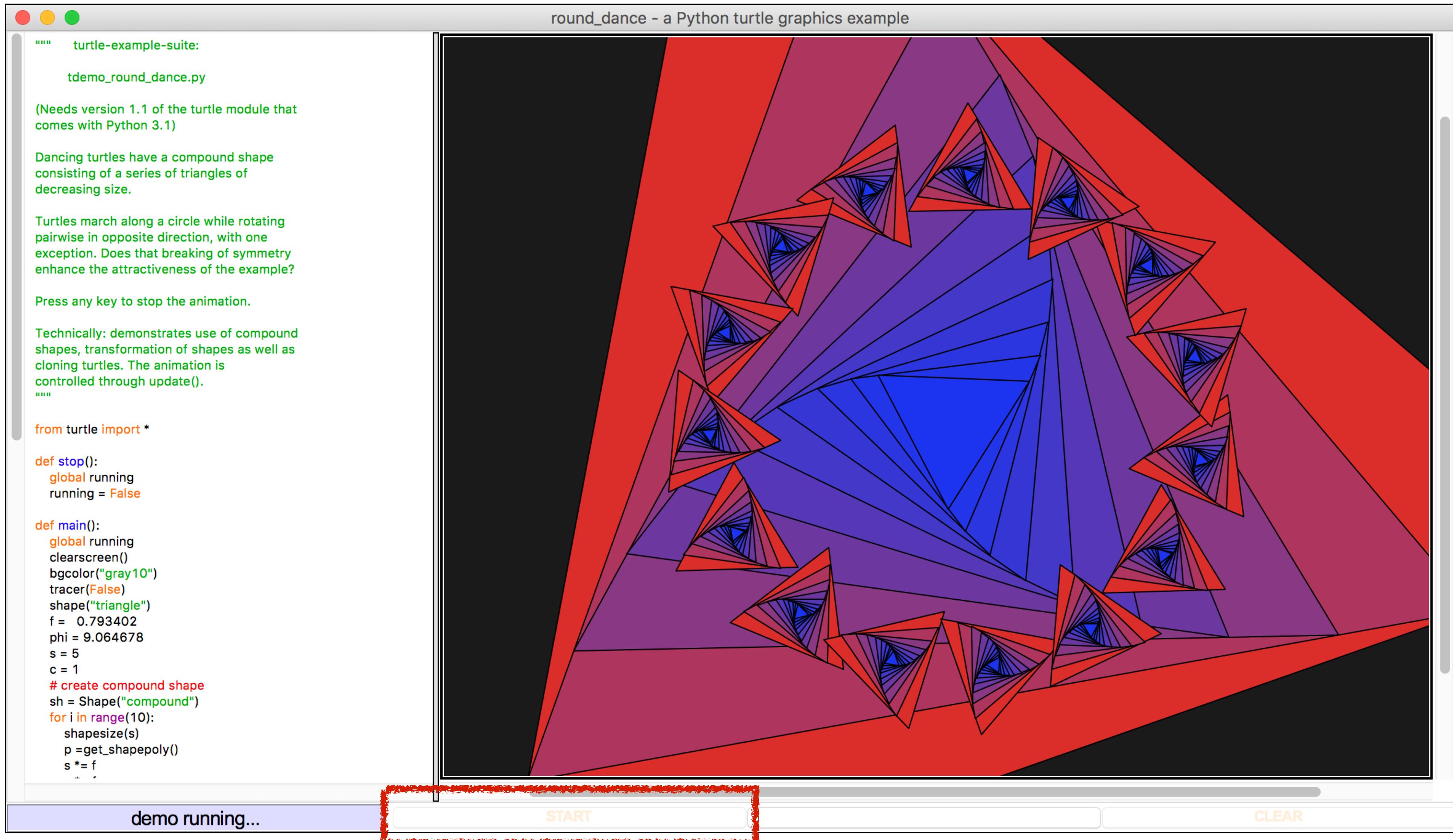
running = True
while running:
    update()
    if not running:
        break
done()
```



Turtle 데모 예시 파일 열기



- [Help] → [Turtle Demo] → [Examples] → [round_dance] → [START] 버튼 클릭 → 실행





IDLE로 파이썬 실행하기



Running Python with IDLE



IDLE 대화형 모드

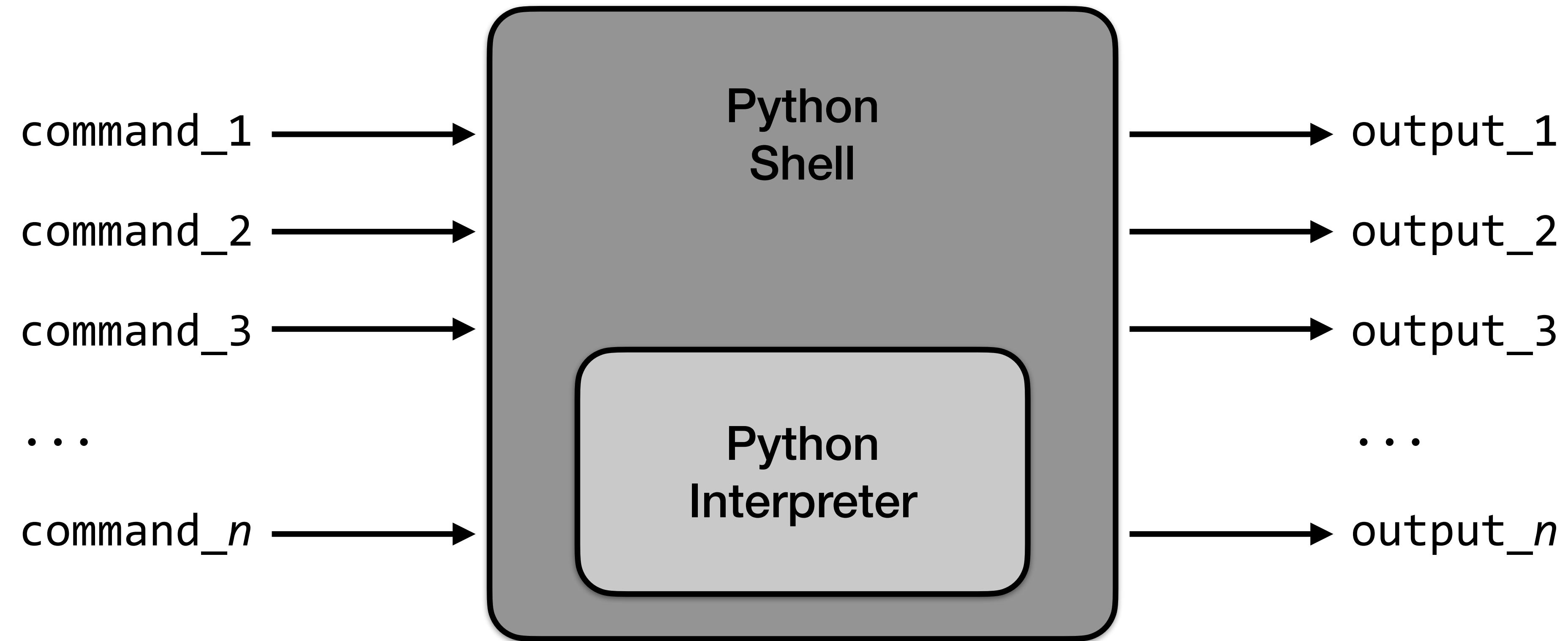
IDLE Interactive Mode





● Interactive Mode

- 파이썬 셸(shell) 실행 후, 명령어(코드)를 하나씩 실행
- 명령어 별 결과를 바로 확인 가능





- 아래 코드 3줄을 입력

```
>>> for i in range(1, 10):  
    for j in range(1, 10):  
        print(f'{i} x {j} = {i * j}')
```

```
1 x 1 = 1  
1 x 2 = 2  
1 x 3 = 3  
1 x 4 = 4  
1 x 5 = 5
```

- IDLE의 [파이썬 셀] 대화형 모드에서 첫 줄 작성 후 [Enter] 또는 [return] 키를 누르면 자동으로 다음 줄로 넘어간 후 4칸을 들여쓰기를 해줌
- 커서가 이동한 위치에서 둘째 줄의 코드를 입력하고 [Enter] 또는 [return] 키를 누르면 자동으로 다음 줄로 넘어간 후 4칸을 들여쓰기를 해줌
- 커서가 이동한 위치에서 셋째 줄의 코드를 입력하고 [Enter] 또는 [return] 키를 연속 두 번 누르면 코드가 실행됨



● Auto-complete>Show completions

- [파이썬 셀] 창과 [파이썬 프로그램] 창 모두 적용됨

● 'pr'만 입력 → [tab] 키 → [print] 선택

- 만약 'pri'까지 입력하고 [tab] 키를 누르면 드롭다운 메뉴를 보여주지 않고 바로 'print'가 완성됨

```
>>> pr|  
print  
property  
quit  
range  
repr  
reversed  
round  
set  
setattr  
slice
```

● print 다음에 '('를 입력

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

IDLE 인터프리터 모드

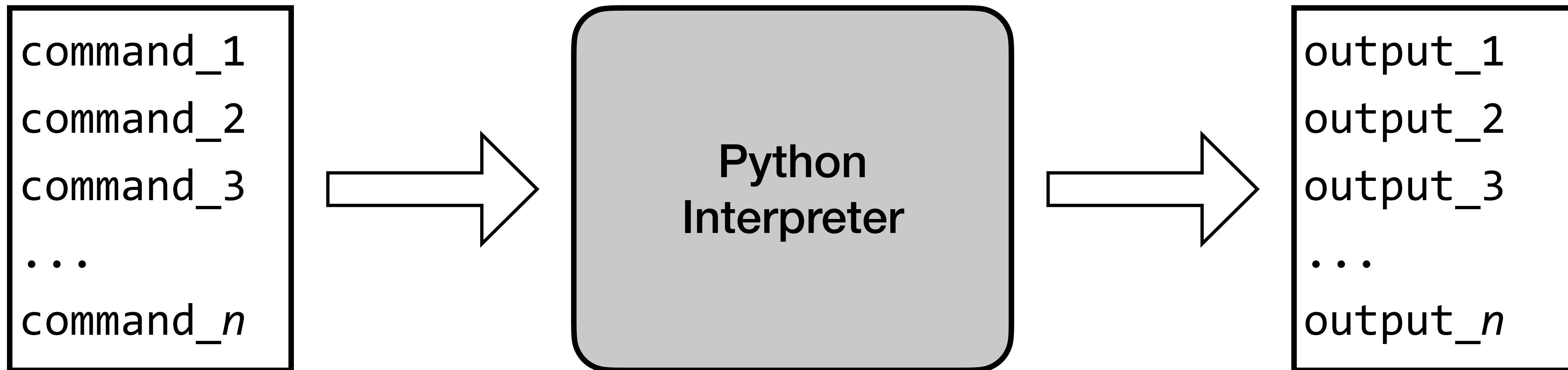
IDLE Interpreter Mode





● Interpreter Mode

- 텍스트 편집기를 사용하여 프로그램을 작성한 후 한번에 실행
- 인터프리터가 프로그램 전체의 문법 오류 확인 후, 명령어 하나 씩 실행



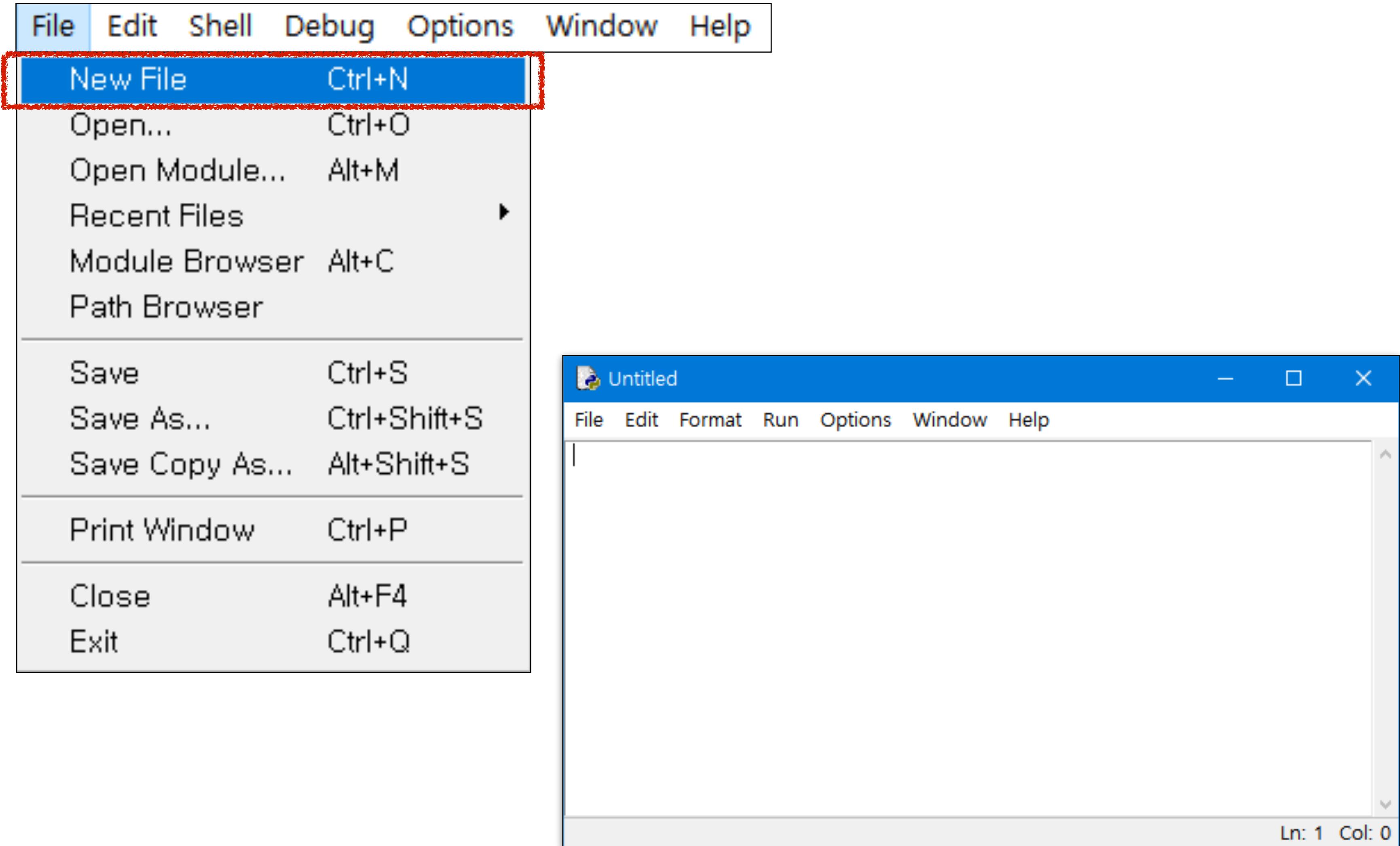


Windows

프로그램 작성을 위해 새 파일 창 열기



- [File] → [New File]



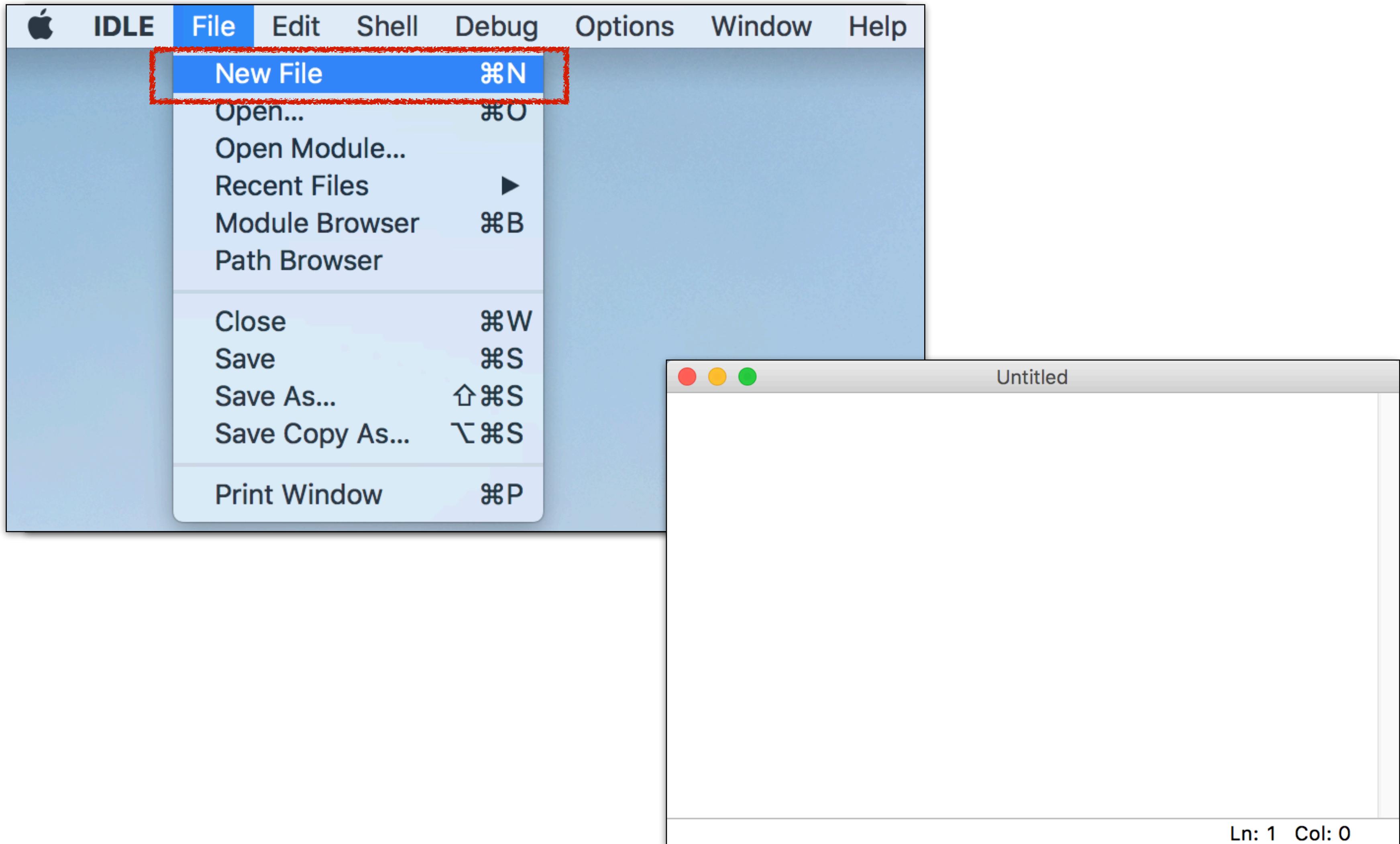


macOS®

프로그램 작성을 위해 새 파일 창 열기



- [File] → [New File]





- Copy & Paste 또는 입력
 - 명령문만 복사

The screenshot shows a code editor window with the title bar "*untitled*". The code in the editor is:

```
for i in range(1, 10):
    for j in range(1, 10):
        print(f'{i} x {j} = {i * j}')
```

In the bottom right corner of the editor window, the text "Ln: 3 Col: 8" is displayed.

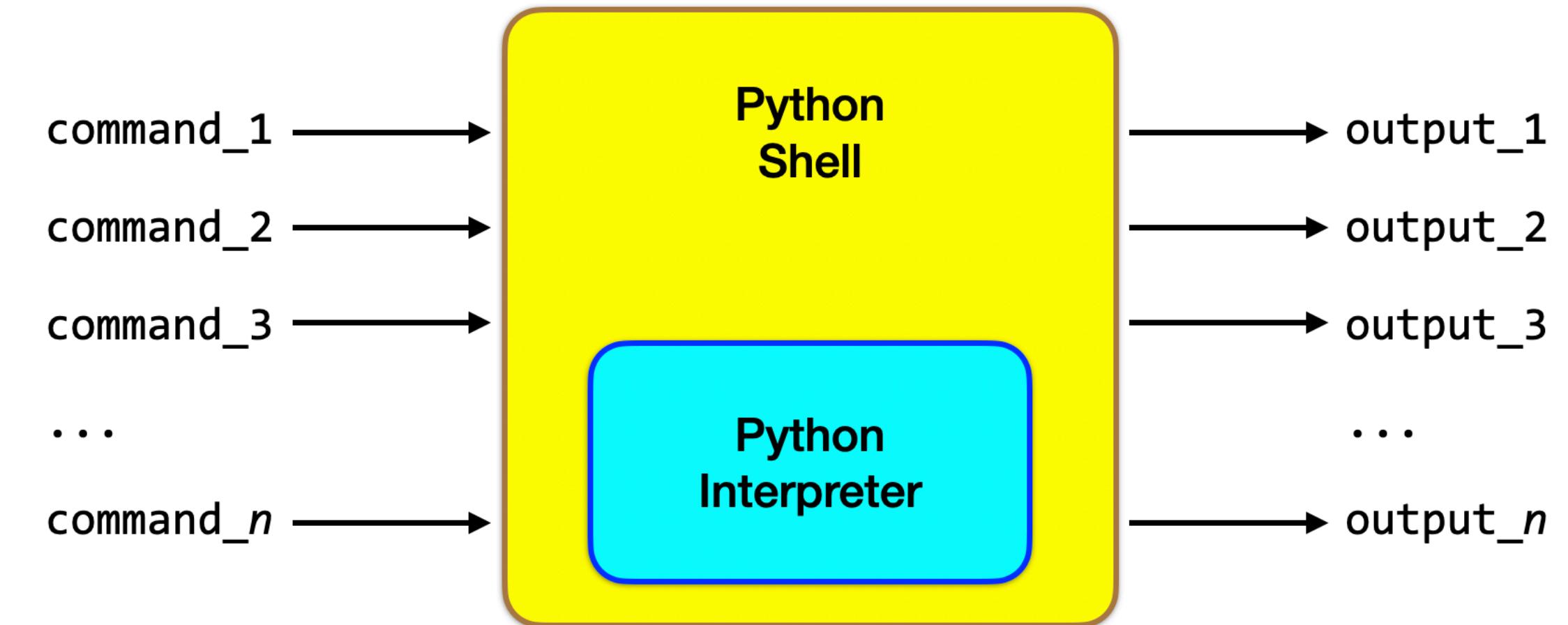
- [File] → [Save]
- [Run] → [Run Module] 또는 단축 키 [F5]
- 실행 결과 확인



정리 : 두 가지 모드에서 파이썬 프로그램 실행

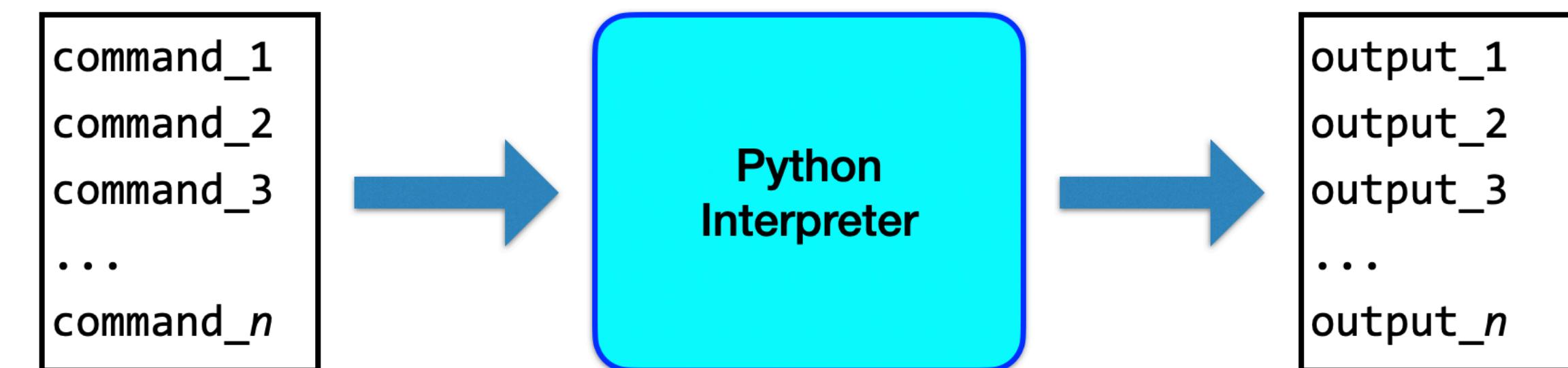
● 대화형 모드(interactive mode)

- 대화 창(파이썬 셀)에서 실시간으로 명령어를 입력
- 대화형 모드를 실행하면 프롬프트('>>>' 표시)가 나타남
 - 프롬프트는 사용자로부터 명령어를 입력받을 준비가 되었다는 뜻
 - 명령어가 처리되면 다음 명령어를 기다리는 프롬프트가 다시 나타남
 - 단, 잘못된 명령어를 입력해서 실행할 경우 오류 메시지가 출력됨
- 파이썬의 다양한 기능을 테스트해 볼 때 편리함



● 인터프리터 모드(interpreter mode)

- 파이썬 명령어들을 문서 형태로 저장(즉, 코드 파일을 생성한다는 뜻)
 - 프로그램 창에서 작성한 코드를 파일 형태로 저장
 - 참고 : 대화형 모드에서는 입력한 명령어들이 저장되지 않음
 - 반드시 파일 확장자를 .py로 설정
- 프로그램 파일을 실행하기 위해 메뉴에서
 - [Run] → [Run Module]
 - 또는
 - [F5]



디버깅

Debugging





- 디버깅(debugging)

- 컴퓨터 프로그램의 문법적 오류(오타 등)나 논리적 오류인 버그(bug)를 찾아내기 위해 테스트하고 수정하는 과정

- [파이썬 셸] 창 선택 → 아래 내용 입력 → [Enter] 또는 [return] 키

```
>>> print('안녕 파이썬')
```

- 오류 발생!!!

```
>>> print('안녕 파이썬')
```

```
SyntaxError: EOL while scanning string literal
```

```
>>>
```

- 오류 수정

```
>>> print('안녕 파이썬')
```

- 실행

```
>>> print('안녕 파이썬')
```

```
안녕 파이썬
```

```
>>>
```



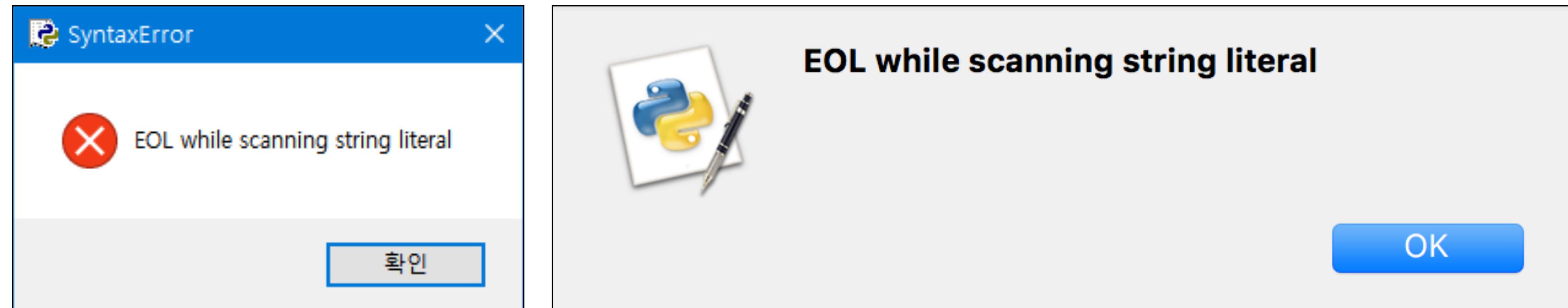
IDLE 인터프리터 모드



- [File] → [New File] → 아래 내용 입력

```
print('안녕 파이썬')
```

- [File] → [Save] → [F5] → 오류 발생!!!



- 디버깅 → 저장 → 실행



● 네이버나 구글 검색창에 오류 메시지 Copy & Paste

NAVER SyntaxError: EOL while scanning string literal

통합검색 블로그 카페 지식iN 이미지 동영상 어학사전 뉴스 더보기 ▾

정렬 ▾ 기간 ▾ 영역 ▾ 옵션유지 깨짐 켜짐 | 상세검색 ▾

블로그

Error: SyntaxError: EOL while scanning string literal 2015.05.21.
Return to main page Error: SyntaxError: EOL while scanning string literal <error_note>"EOL" stands for "end of line". An EOL error means that Python...
엠지님의 블로그~! blog.naver.com/swwwwaa?Redirect=Log&logNo... | 블로그 내 검색

SyntaxError 2017.11.26.
SyntaxError SyntaxError: EOL while scanning string literal 문자열의 끝에서 줄마음표 찍지 않았을 경우에 발생
시간을 들여서 노... blog.naver.com/h2odra?Redirect=Log&lo...

파이썬 문자열 선언 - Python String 2010.01.21.
hello = "test ^ SyntaxError: EOL while scanning string literal """(쌍따옴표 3개)를 사용한 녀석은 아래와 같이 """ 가 나올 때 까지 계속 입력을 받고...
구차니의 잡동사니 ... minimonk.net/1250 | 블로그 내 검색

[블로그 더보기 >](#)

카페

파이썬 기초 코딩 2017.09.04.
에러>>> "반갑습니다" "반갑습니다">>> '안녕' SyntaxError: EOL while scanning string literal>>> print("문자를 입력할때는 ", "로 짹을 맞춰야 함")문자를 입력할때는...
양주종의 코딩스쿨 ► C언어 · C++· 파... cafe.naver.com/funcc/4374... | 카페 내 검색

파이썬(Python) 기초 – 파이썬입출력,파이썬자료형 2017.06.14.
함>>> 'korea"SyntaxError: EOL while scanning string literal>>> "korea'SyntaxError: EOL while scanning string literal>>> "korea" "korea'>>> 'korea"korea'>>> print(30,50,60,70)30 50 60 70>>> a=100...
양주종의 코딩스쿨 ► C언어 · C++· 파... cafe.naver.com/funcc/4238... | 카페 내 검색

Google SyntaxError: EOL while scanning string literal

All Images Videos News More Settings Tools

About 12,900 results (0.38 seconds)

[python: SyntaxError: EOL while scanning string literal - Stack ...](#)
<https://stackoverflow.com/questions/3561691/python-syntaxerror-eol-while-scanning-string-literal>
I have the above mentioned error in s1="some very long string....." Anyone know what i am doing wrong?

8 answers

Best Answer 102 votes
You are not putting a " before the end of the line. Use "" if you want to do this: "" a very long stringthat can span multiple lines ""

Answer 2 of 8 49 votes
I had this problem - I eventually worked out that the reason was that I'd included \ characters in the string. If you have any of these, "escape" them with \ a...

Answer 3 of 8 12 votes
(Assuming you don't h... line breaks in your str... long is this string really suspect there is a limit long a line read from a

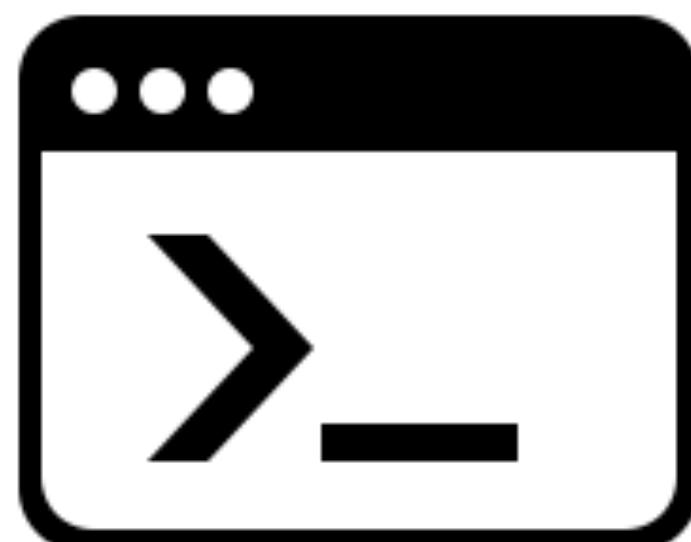
[What's an EOL error and how do I fix it? | Codecademy](#)
https://www.codecademy.com/en/forum_questions/52c0b0977c82ca009d006037 ▾
Jan 23, 2014 - 2 posts - 1 author
Hello. I'm trying to finish Multiline Comments (lesson 9/16) and have run into a problem. When I try to submit my code I get an error pointing to the end of line one "SyntaxError: EOL while scanning string literal". What does that mean? What should be at the end of line one?

5/9 SyntaxError: EOL while scanning string literal 2 posts 13 Jan 2016
SyntaxError: EOF while scanning triple-quoted string literal ... 6 posts 19 Jan 2014
Error when writing sentence over 2 lines 3 posts 14 Jan 2013
More results from www.codecademy.com

[Getting "EOL while scanning string literal" error, and what happened ...](#)
<https://teamtreehouse.com/.../getting-eol-while-scanning-string-literal-error-and-what-...> ▾
May 21, 2015 - Hi: I am going over projects and code challenges and adding them to my portfolio of work. When I did Shopping List Take Three, I noticed that I got a "EOL while scanning string literal" error. I decided to try and make the input statement all one line. One problem: when I did, the other functions acted odd.

명령어 셸 실행 환경

Command Shell Environment





● 셸(shell)이란?

- ⦿ 커널(kernel)이라는 운영체제의 내부 핵심과 사용자 사이의 인터페이스
- ⦿ 사용자의 명령을 해석해서 운영체제에 전달하고 그 처리 결과를 사용자에게 보여주는 시스템 프로그램
- ⦿ 사용자는 다양한 셸(shell) 환경에서 프로그램을 실행할 수 있음

● 셸의 종류

⦿ 그래픽 셸(graphic shell)

- ⦿ 그래픽 사용자 인터페이스(GUI, graphic user interface)
- ⦿ 윈도우(Windows)
- ⦿ 맥 OS(macOS)
- ⦿ 유닉스(UNIX)와 리눅스(LINUX)의 X 윈도우 시스템(X Window System, X11)

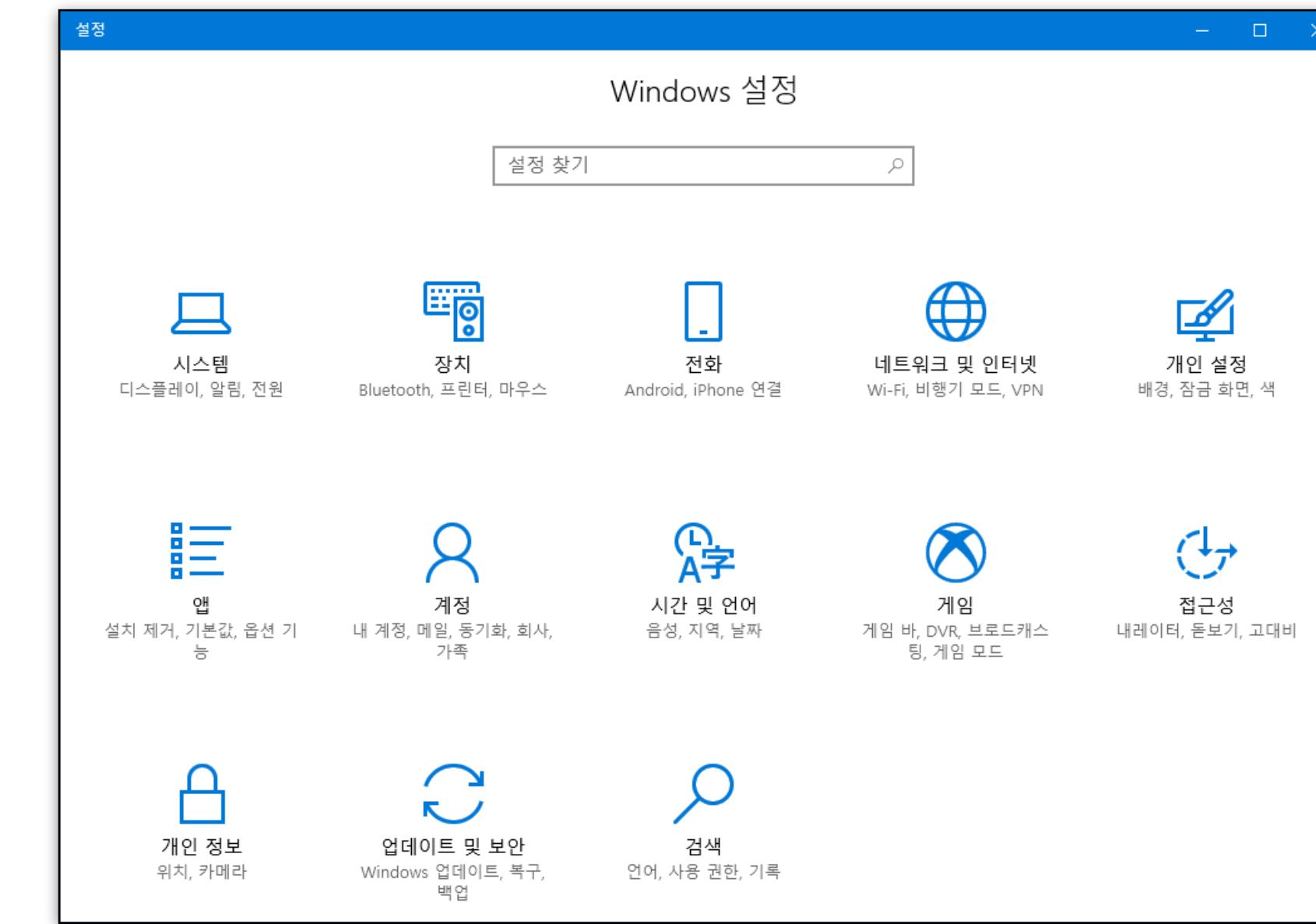
⦿ 명령어 셸(command/character shell)

- ⦿ 명령어 인터페이스 또는 명령 줄 인터페이스(CLI, command line interface)
 - 사용자가 컴퓨터 키보드 등을 통해 문자열 형태로 입력을 하며, 컴퓨터로부터의 출력 또한 문자열로 이루어짐
- ⦿ 윈도우(Windows) 운영체제 : 도스(DOS) 셸 환경의 명령 프롬프트
- ⦿ 맥 OS(macOS) 운영체제 : bash 셸 환경의 터미널



예시 : 그래픽 셀

● 윈도우(Windows)



● 맥 OS(macOS)



- 원도우(Windows) : 도스(DOS) 셸 환경의 명령 프롬프트

```
C:\Users\montol>dir
C 드라이브의 볼륨: ontology
볼륨 일련 번호: 3EOA-201C

C:\Users\montol 디렉터리

2017-12-28 오전 11:53 <DIR>
2017-12-28 오전 11:53 <DIR>
2017-12-15 오후 08:36 <DIR>
2017-12-18 오후 08:44 <DIR>
2017-12-18 오후 08:44 <DIR>
2017-12-18 오후 08:44 <DIR>
2017-12-18 오후 08:44 <DIR>
2017-12-11 오후 06:28 <DIR>
2018-01-04 오후 07:00 <DIR>
2017-12-18 오후 08:48 <DIR>
2017-12-28 오전 11:53 <DIR>
2017-12-18 오후 08:44 <DIR>
2017-12-18 오후 08:44 <DIR>
2018-01-02 오후 12:04 <DIR>
2017-12-11 오후 06:28 <DIR>
2017-05-11 오전 11:27 <DIR>
2017-12-18 오후 08:44 <DIR>
2017-12-18 오후 08:44 <DIR>
2017-12-18 오후 08:44 <DIR>

. . .
:idlerc
3D Objects
Contacts
Desktop
Documents
Downloads
Favorites
Intel
Links
Music
OneDrive
Pictures
Roaming
Saved Games
Searches
Videos

0개 파일
0 바이트
18개 디렉터리 156,936,695,808 바이트 남음
```

- 맥 OS(macOS) : bash 셸 환경의 터미널

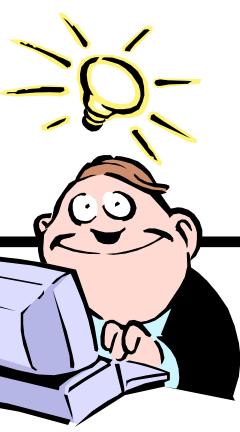
```
[> ls -l
total 8
-rw-r--r--  1 root  staff   558 12  3 22:36 AT.postflight.1381
drwx-----@ 3 jsp   staff    96 12  1 22:44 Applications
drwx-----+ 4 jsp   staff   128 12  3 21:48 Desktop
drwx-----+ 3 jsp   staff    96 12  1 22:27 Documents
drwx-----+ 4 jsp   staff   128  1  5 00:24 Downloads
drwx-----@ 12 jsp  staff   384 12  1 23:06 Dropbox
drwx-----@ 14 jsp  staff   448  1  6 22:10 Google 드 라 이 브
drwx-----@ 66 jsp  staff  2112  1  2 23:07 Library
drwx-----+ 3 jsp   staff    96 12  1 22:27 Movies
drwx-----+ 4 jsp   staff   128  1  2 22:40 Music
drwx-----+ 3 jsp   staff    96  1  4 23:54 Pictures
drwxr-xr-x+ 5 jsp   staff   160 12  1 22:27 Public
drwxr-xr-x  10 jsp  staff   320  1  4 23:47 software-downloads
> ]
```



● 명령어 셸에서 파이썬 셸 실행

- 실행 파일 이름 : **python** 또는 **python3**
- 파이썬 셸이 시작 메시지인 파이썬 버전 및 저작권 안내를 출력한 후 프롬프트가 나타난다
 - 대화형 모드 기본 프롬프트 : '**>>>**'
 - 한 명령문 내에서 줄 바꿈이 있을 경우 '**...**' 프롬프트가 표시된다

```
> python
Python 3.x.x ...
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 1
2
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>>
```



- 방법 1

```
>>> quit()
```

- 방법 2

```
>>> import sys  
>>> sys.exit()
```

- 방법 3

- 창 닫기 아이콘 클릭

- 방법 4 : macOS 경우 단축 키 사용

- [control] + Z
 - [control] + D



만약 같은 파이썬 코드를 여러 번 반복해서 실행해야 한다면?



- 프로그램 파일은 파이썬 인터프리터를 통해 실행
- 주로 명령어 셸(CLI)에서 이루어진다
- 윈도우(Windows) 명령 프롬프트에서 실행

파이썬 환경설정에서 ‘PATH’에 파이썬 설치경로가
지정되지 않았을 경우 여기서 에러가 발생

C:\Users\montol>python hello.py
안녕 파이썬

C:\Users\montol>_

A screenshot of a Windows Command Prompt window titled "명령 프롬프트". The command "python hello.py" is entered, but the file "hello.py" is underlined in red, indicating a syntax error. The output "안녕 파이썬" is displayed below the command.

- 맥 OS(macOS)의 터미널이나 리눅스(Linux)에서 실행

course-examples — -bash — 49x5

[\$ python3 hello.py]
안녕 파이썬

\$

A screenshot of a macOS Terminal window. The title bar shows "course-examples — -bash — 49x5". The command "\$ python3 hello.py" is entered, with "python3" underlined in red. The output "안녕 파이썬" is shown below the command. A cursor is visible at the end of the line.



셸 명령어 기초



Lab Exercises

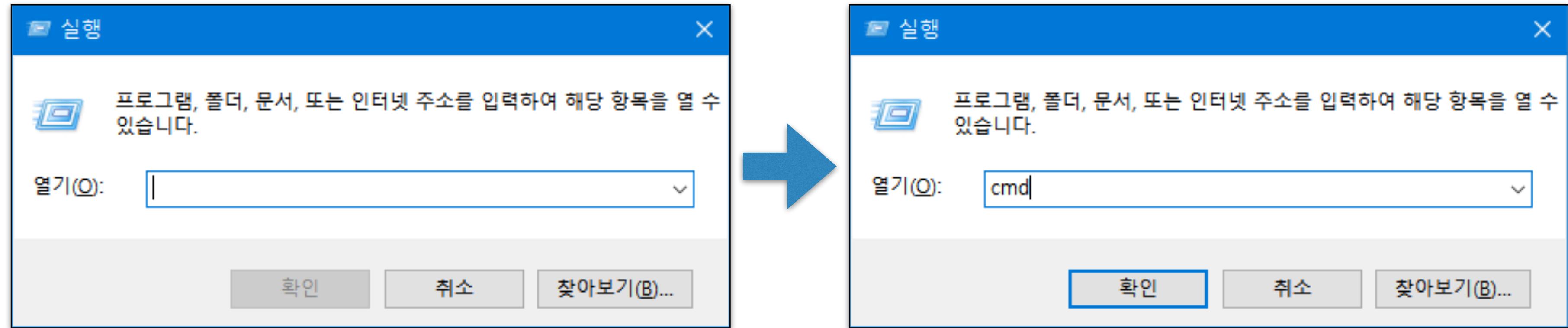




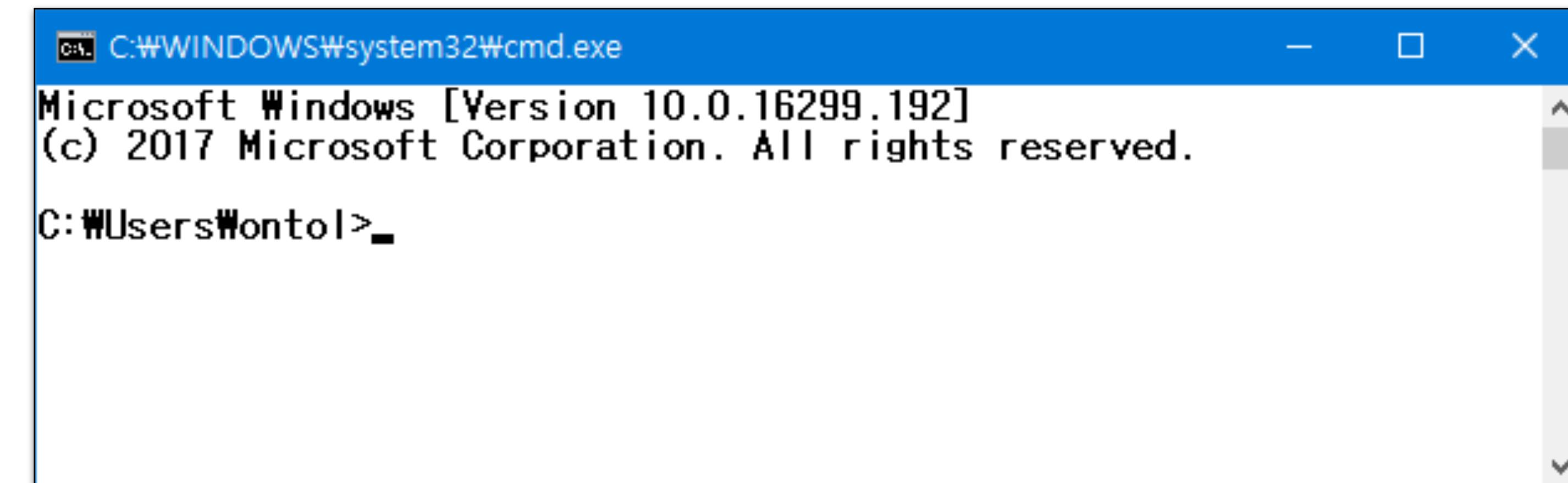
명령 프롬프트 열기



- 키보드에서 [윈도우 시작버튼(창문 아이콘)] + [R]을 누름
- 실행 창이 나타나면 ‘cmd’라고 입력한 후 [Enter] 키를 누름 → cmd.exe 파일이 실행됨



- 그러면 아래와 같이 ‘명령 프롬프트’ 창이 열림



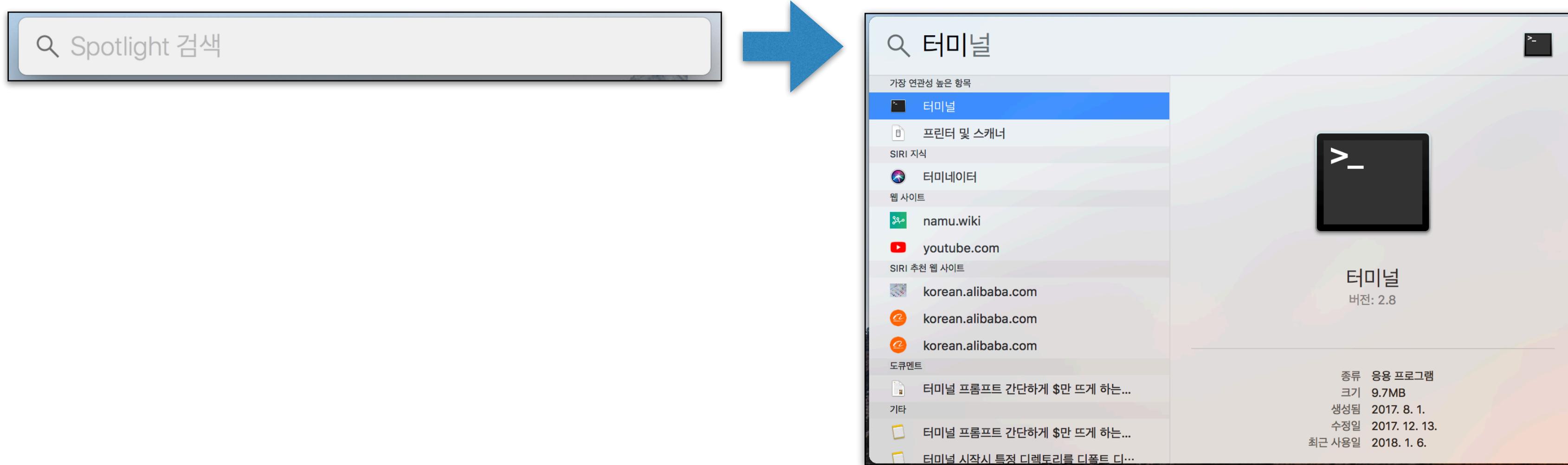


macOS®

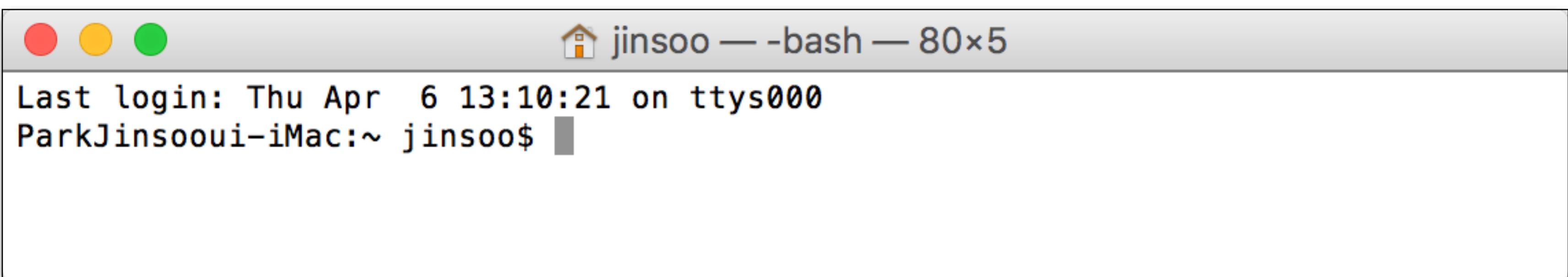
터미널 열기



- 키보드에서 [command](혹은 [control]) + [스페이스 바]를 누름
- [스포트라이트(Spotlight) 검색] 창이 나타나면 ‘터미널’이라고 입력한 후 [터미널] 항목을 더블 클릭



- 그러면 아래와 같이 ‘터미널’ 창이 열림





- **help**

- ‘명령 프롬프트’에서 제공하는 명령어 목록을 볼 수 있음

- **help [명령어]** 또는 **[명령어] /?**

- 특정 명령어에 대한 자세한 내용 도움 요청

- **cls**

- 화면에 나타난 모든 출력을 지움

- **exit**

- ‘명령 프롬프트’ 종료

```
C:\Users\montol>help cls
화면을 지웁니다.

CLS

C:\Users\montol>cls /?
화면을 지웁니다.

CLS

C:\Users\montol>exit
```



macOS®

도우미 명령어



● man [명령어]

- 특정 명령어에 대한 자세한 내용 도움 요청

● clear

- 화면에 나타난 모든 출력을 지움

● exit

- ‘터미널’ 프로세스를 종료

The screenshot shows two terminal windows. The top window is titled 'jinsoo — bash — 80x5' and displays the command 'ParkJinsooui-iMac:~ jinsoo\$ man clear'. The bottom window is titled 'jinsoo — less ▶ man clear — 80x22' and displays the man page for 'clear(1)'. The man page includes sections for NAME, SYNOPSIS, DESCRIPTION, and SEE ALSO, along with a note about the version.

```
jinsoo — bash — 80x5
ParkJinsooui-iMac:~ jinsoo$ man clear

jinsoo — less ▶ man clear — 80x22
clear(1)                                         clear(1)

NAME
    clear — clear the terminal screen

SYNOPSIS
    clear

DESCRIPTION
    clear clears your screen if this is possible. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen.

    clear ignores any command-line parameters that may be present.

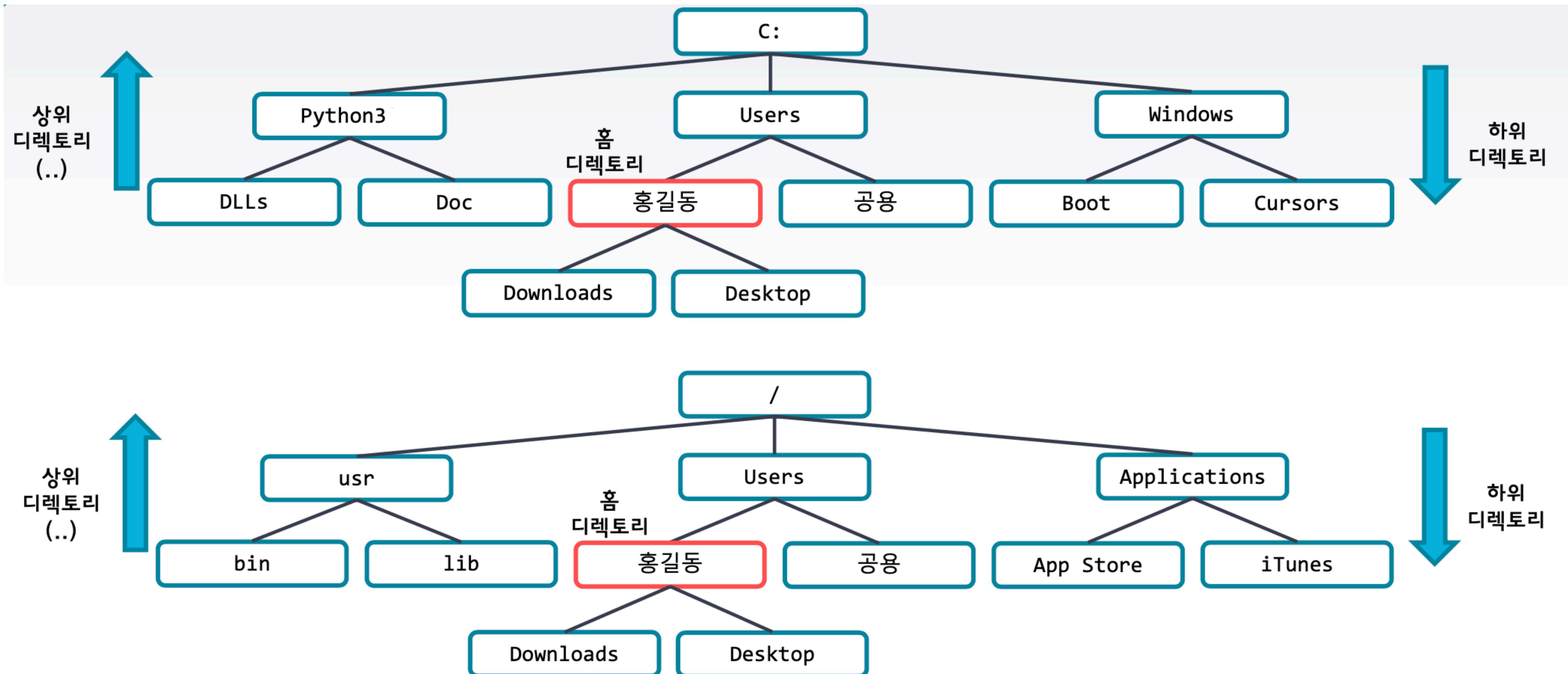
SEE ALSO
    tput(1), terminfo(5)

This describes ncurses version 5.7 (patch 20081102).

(END)
```



디렉토리 계층 구조



[출처: 이일주(2021)]



현재 작업 디렉토리 위치 출력 명령어

- 현재 작업이 진행 중인 디렉토리의 위치를 알고 싶다면 아래와 같이 입력

- ◉ Windows의 경우

```
> cd
```

또는

```
> chdir
```

- ◉ macOS의 경우

```
$ pwd
```

- 디렉토리(directory)
 - ▶ 운영체제 파일 시스템의 한 부분으로 파일과 다른 디렉토리를 가지고 있으며 폴더(folder)와 같은 뜻으로 사용
- 경로(path)
 - 파일 시스템에서 특정한 위치에 있는 파일이나 디렉토리를 구분자(' / ', '\ ', '₩' 등)로 분리해서 문자열로 표현한 디렉토리 트리 계층(tree hierarchy) 형식
 - e.g., Windows의 경우 'C:\Users\홍길동', macOS의 경우 '/Users/홍길동'



- 실행 예시

- Windows의 경우

```
> cd  
C:\Users\ontology
```

- macOS의 경우

```
$ pwd  
/Users/ontology
```



새로운 디렉토리 생성 명령어

- 현재 경로 아래에 새로운 디렉토리를 만들기 위해서는 아래 명령어를 입력

- 현재 작업 디렉토리에서 새로운 디렉토리를 만들면 이를 ‘하위 디렉토리’라고 함

- Windows의 경우

```
> mkdir [디렉토리 이름]
```

- macOS의 경우

```
$ mkdir [디렉토리 이름]
```

새로운 디렉토리 생성 예시



● 실행 예시

앞으로 작업할 파이썬 프로그램을 저장하기 위해 사용자 홈 디렉토리에 ‘`pyprg`’라는 이름을 가진 디렉토리를 생성

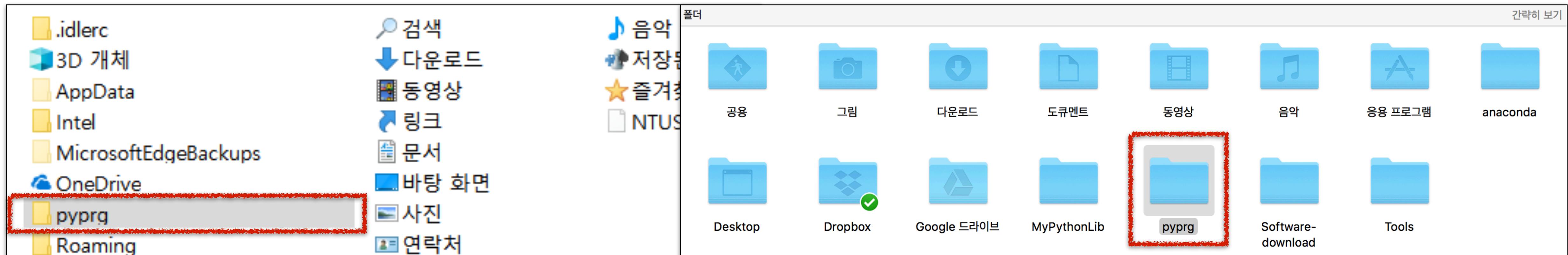
Windows의 경우

```
> mkdir pyprg
```

macOS의 경우

```
$ mkdir pyprg
```

● 사용자 홈 디렉토리 아래에 새로운 디렉토리(폴더)가 생성된 것을 확인할 수 있음





- 현재 경로에서 다른 디렉토리로 작업 디렉토리를 변경하고 싶으면 아래 명령어를 사용

- Windows의 경우

```
> cd [경로]
```

- macOS의 경우

```
$ cd [경로]
```

- 경로

- 절대경로(absolute path)

- 현재 작업 디렉토리와 상관없이 최상위(루트) 디렉토리부터 시작하는 고유한 경로
 - Windows 예 : C:\Users\ontology\pyprg\homework
 - macOS 예 : /Users/ontology/pyprg/homework

- 상대경로(relative path)

- 현재 작업 디렉토리를 기준으로 상대적인 위치에 있는 경로
 - Windows 예 : ./pyprg\homework
 - macOS 예 : ./pyprg/homework



하위 디렉토리로 경로 변경

- 하위 디렉토리로 이동하고 싶으면 아래 명령어를 사용

- Windows의 경우

```
> cd [디렉토리 이름]
```

- macOS의 경우

```
$ cd [디렉토리 이름]
```



- 실행 예시

- Windows의 경우

```
> cd pyprg  
> cd  
C:\Users\ontology\pyprg
```

- macOS의 경우

```
$ cd pyprg  
$ pwd  
/Users/ontology/pyprg
```



- 상위 디렉토리로 이동하고 싶으면 아래 명령어를 사용

- Windows의 경우

```
> cd ..
```

- macOS의 경우

```
$ cd ..
```



- 실행 예시

- Windows의 경우

```
> cd  
C:\Users\ontology\pyprg  
> cd ..  
> cd  
C:\Users\ontology
```

- macOS의 경우

```
$ pwd  
/Users/ontology/pyprg  
$ cd ..  
$ pwd  
/Users/ontology
```



기타 다른 디렉토리로 경로 변경 예시



- 실행 예시 : 사용자 홈 디렉토리에서 시작

- Windows의 경우 : ‘Windows’ 디렉토리 아래 있는 ‘System’ 디렉토리로 이동

```
> cd C:\Windows\System  
> cd  
C:\Windows\System
```

절대경로 사용

```
> cd ..\..\Windows\System  
> cd  
C:\Windows\System
```

상대경로 사용

- macOS의 경우 : ‘usr’ 디렉토리 아래 있는 ‘local’ 디렉토리의 하위 디렉토리인 ‘bin’으로 이동

```
$ cd /usr/local/bin  
$ pwd  
/usr/local/bin
```

절대경로 사용

```
$ cd ../../usr/local/bin  
$ pwd  
/usr/local/bin
```

상대경로 사용



홈 디렉토리로 복귀

- 현재 경로에서 사용자 홈 디렉토리로 돌아가기 위해서는 아래 명령어를 입력

- Windows의 경우

```
> cd %homepath% (또는 %HOMEPATH%)
```

- macOS의 경우

```
> cd ~
```

- **홈 디렉토리(home directory)**
 - ▶ 시스템에 계정을 가진 사용자가 로그인하면 그 사용자에게 할당된 개인 작업 영역의 최상위 디렉토리



- 실행 예시

- Windows의 경우

```
> cd %homepath%
> cd
C:\Users\ontology
```

- macOS의 경우

```
> cd ~
> pwd
/Users/ontology
```



디렉토리 열람 명령어

- 현재 디렉토리에 있는 파일과 하위 디렉토리 목록을 열람하기 위해서는 아래 명령어를 사용

- Windows의 경우

```
> dir
```

- macOS의 경우

```
$ ls
```



● 실행 예시

⦿ Windows의 경우

```
> dir  
C 드라이브의 볼륨: ontology  
볼륨 일련 번호: 3E0A-201C  
  
C:\Users\ontology 디렉터리  
  
2018-01-08 오후 08:51 <DIR> .  
2018-01-08 오후 08:51 <DIR> ..  
2018-01-08 오후 12:18 <DIR> Contacts  
2018-01-08 오후 12:18 <DIR> Desktop  
2017-12-11 오후 06:28 <DIR> Documents  
2018-01-08 오후 12:18 <DIR> Downloads  
2018-01-10 오전 11:47 <DIR> Favorites
```

⦿ macOS의 경우

```
$ ls  
Applications Documents Library Music Public pyprg  
Desktop Downloads Movies Pictures Tools
```



파일 복사 명령어

- 파일을 복사하기 위해서는 아래 명령어를 사용

- Windows의 경우

```
> copy [원본 (경로)파일 이름] [대상 (경로)파일 이름]
```

- macOS의 경우

```
$ cp [원본 (경로)파일 이름] [대상 (경로)파일 이름]
```

- 파일을 복사할 때 다른 이름으로 복사 가능

복사하려는 파일이 경로에 존재하기 않으면 에러 메시지가 나타남

같은 파일 이름이 대상 디렉토리에 있을 경우 기존 파일을 덮어씀



● 실행 예시

- ‘test.txt’라는 파일이 현재 작업 디렉토리에 있다고 가정

Windows의 경우

```
> copy test.txt test2.txt  
> dir /w  
...      test.txt      test2.txt  
> copy test.txt C:\Temp\test.txt  
> dir /w C:\Temp  
...      test.txt
```

macOS의 경우

```
$ cp test.txt test2.txt  
$ ls  
...      test.txt      test2.txt  
$ cp test.txt /tmp/test.txt  
$ ls /tmp  
...      test.txt
```



파이썬 프로그램 실행 명령어 : 인터프리터 모드

- 파이썬 파일을 실행하려면 아래 명령어를 사용

- Windows의 경우

```
> python [(경로)파일 이름].py
```

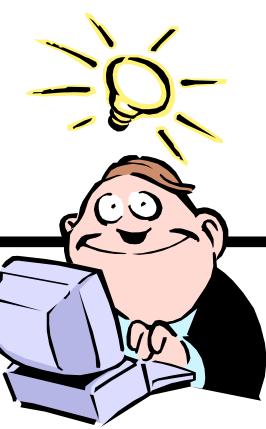
- macOS의 경우

```
$ python3 [(경로)파일 이름].py
```



macOS의 경우 환경 설정에 따라 **python**은 파이썬 2이고 **python3**이 파이썬 3일 수 있음

- 본 수업에서는 **python**이 파이썬 3이라 가정
- 만약 사용자 환경에서 **python3**이 파이썬 3일 경우는 **python3**을 사용해야 함



● 실행 예시

- ▣ 우측과 같은 ‘hello.py’ 파일이 현재 작업 디렉토리에 있다고 가정

▣ Windows의 경우

```
> python hello.py  
안녕 파이썬
```

```
hello.py - C:\Users\Wontol\Wp... File Edit Format Run Options Window Help print('안녕 파이썬') Ln: 1 Col: 0
```

▣ macOS의 경우

```
$ python hello.py  
안녕 파이썬
```

```
hello.py - /Users/jinso... print('안녕 파이썬') Ln: 1 Col: 0
```

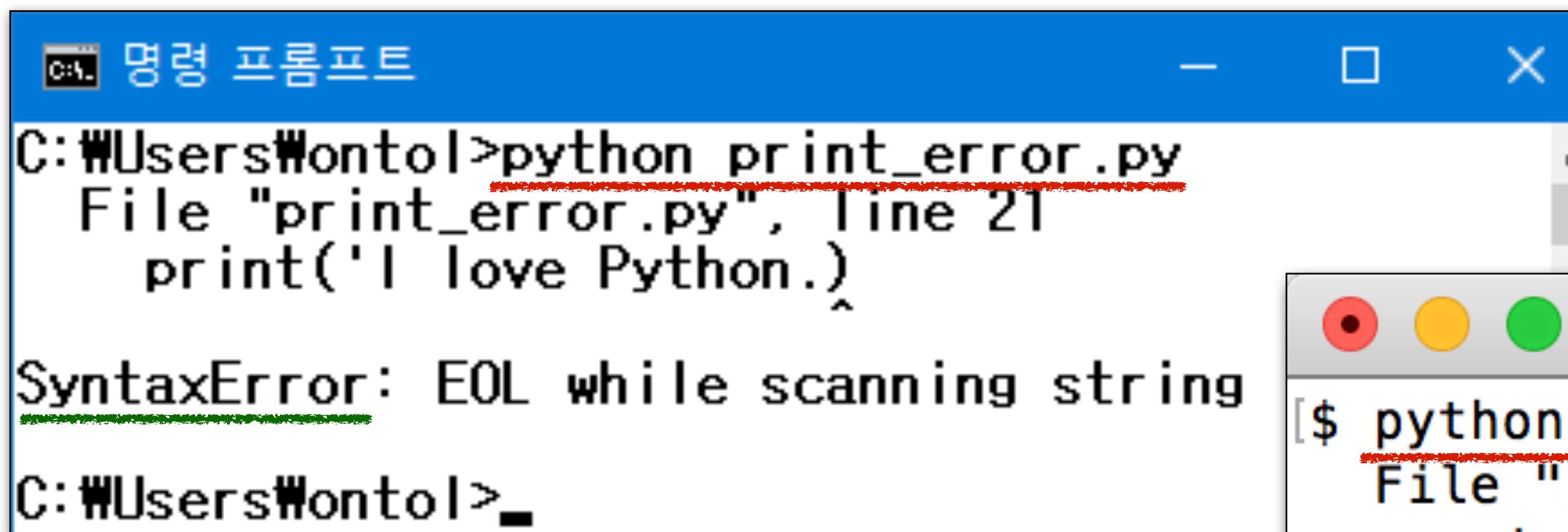


● 코드 작성 및 실행

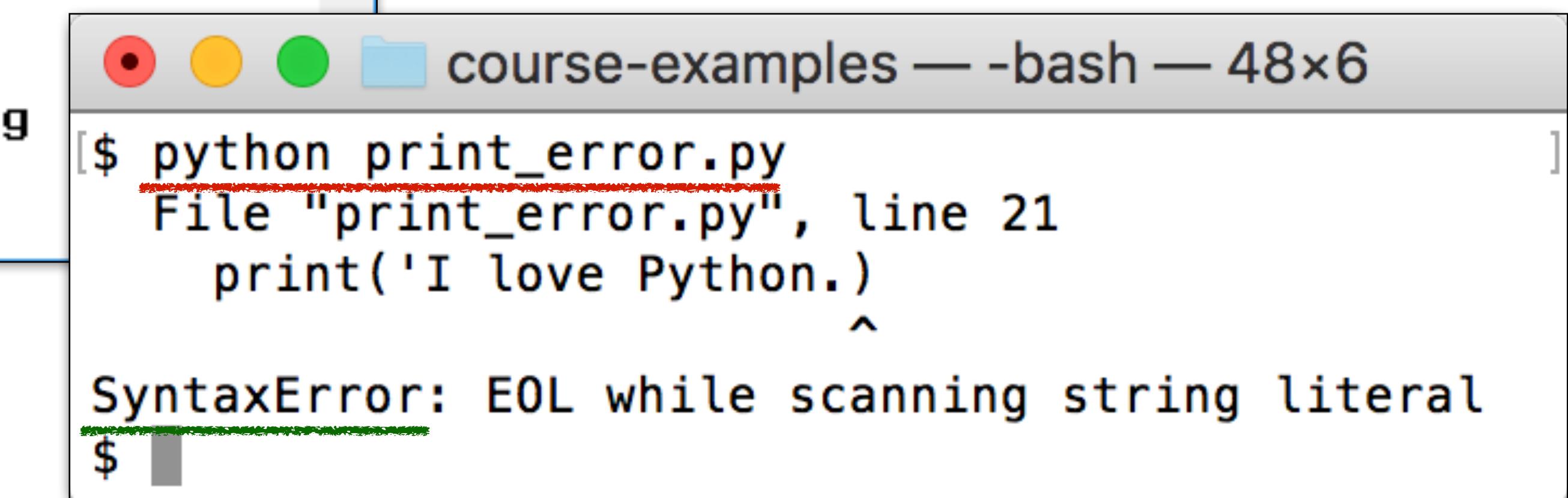
- 아래 코드를 작성한 후 'print_error.py'라는 파일명으로 저장

```
print("Hello", "Python~~~!")
print('Hello Python~~~!')
print('I love Python.')
print("So I'll master Python.")
```

- 저장한 파일 실행 예시 → 오류 발생



```
명령 프롬프트
C:\Users\montol>python print_error.py
File "print_error.py", line 21
    print('I love Python.')
SyntaxError: EOL while scanning string
C:\Users\montol>
```



```
course-examples — -bash — 48x6
$ python print_error.py
File "print_error.py", line 21
    print('I love Python.')
^
SyntaxError: EOL while scanning string literal
$
```



Lab : 명령어 셸에서 파이썬 코드 실행하기



- 디렉토리 만들기

- 사용자 홈 디렉토리 아래 ‘`pyprg`’ 디렉토리를 만들고 그 아래 ‘`lab`’ 디렉토리를 생성

- 파이썬 프로그램 작성하기

- ‘`lab`’ 디렉토리에 아래 ‘`helloworld.py`’ 파일을 만들어 아래 내용을 입력한 후 저장

```
x = 'Hello '
y = 'World!'
z = x + y
print(z)
```

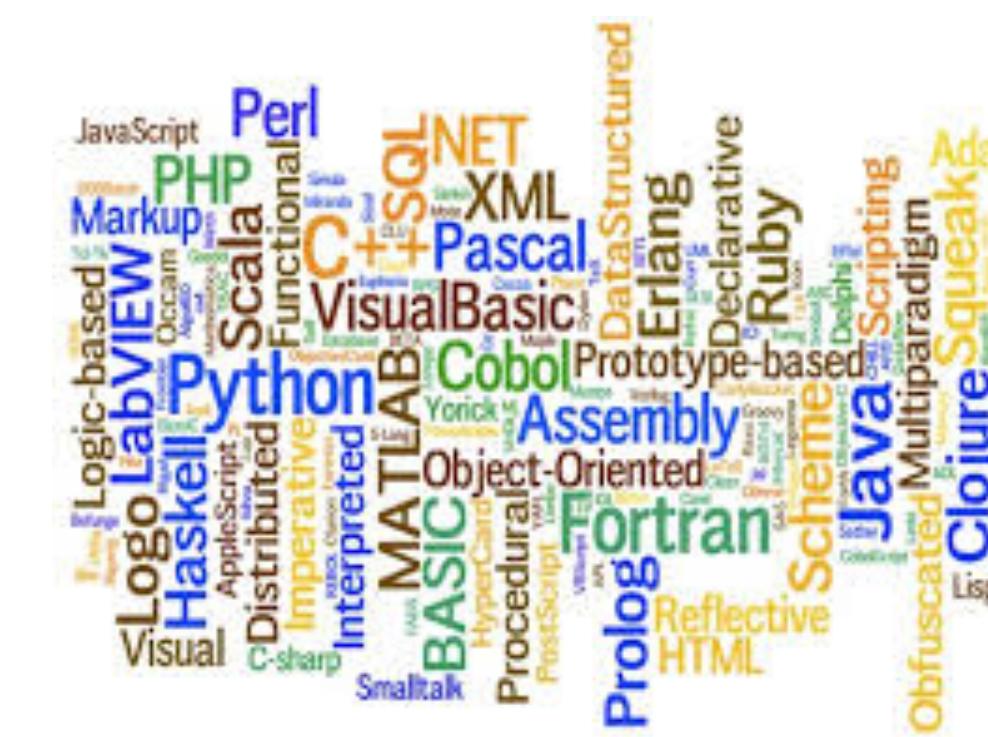
- 파이썬 프로그램 실행하기

- ‘`helloworld.py`’를 실행하면 아래와 같은 결과가 나와야 함

```
> python helloworld.py
Hello World!
```

프로그래밍과 프로그램 개발 절차

Program & Program Development Process





● 프로그램(program)이란?

- 특정한 작업을 어떻게 수행해야 하는지 그 순서를 일련의 명령어로 나열한 것

● 프로그램 언어의 공통 기본기능

- 파이썬(Python), 자바(Java), C 언어를 포함한 모든 프로그래밍 언어는 공통적으로 다음과 같은 기본 기능들을 제공

● **입력**

- 키보드나 파일 또는 별도의 장치로부터 데이터를 입력

● **출력**

- 데이터를 컴퓨터 화면에 보여주거나 파일 또는 별도의 장치로 전송

● **처리**

- **연산 처리** : 더하기, 빼기, 곱하기, 나누기 같은 기본적인 수학적 연산을 수행

- **순차 처리** : 순서대로 작업을 수행

- **선택(조건) 처리** : 특정한 조건에 따라 그에 맞는 적절한 코드를 선택해서 실행

- **반복 처리** : 주어진 횟수나 조건에 따라 특정한 작업을 (주로 약간의 변화를 주면서) 반복적으로 수행



예시 : 1학년 학생들의 평균 점수를 계산하여 화면에 출력하는 프로그램

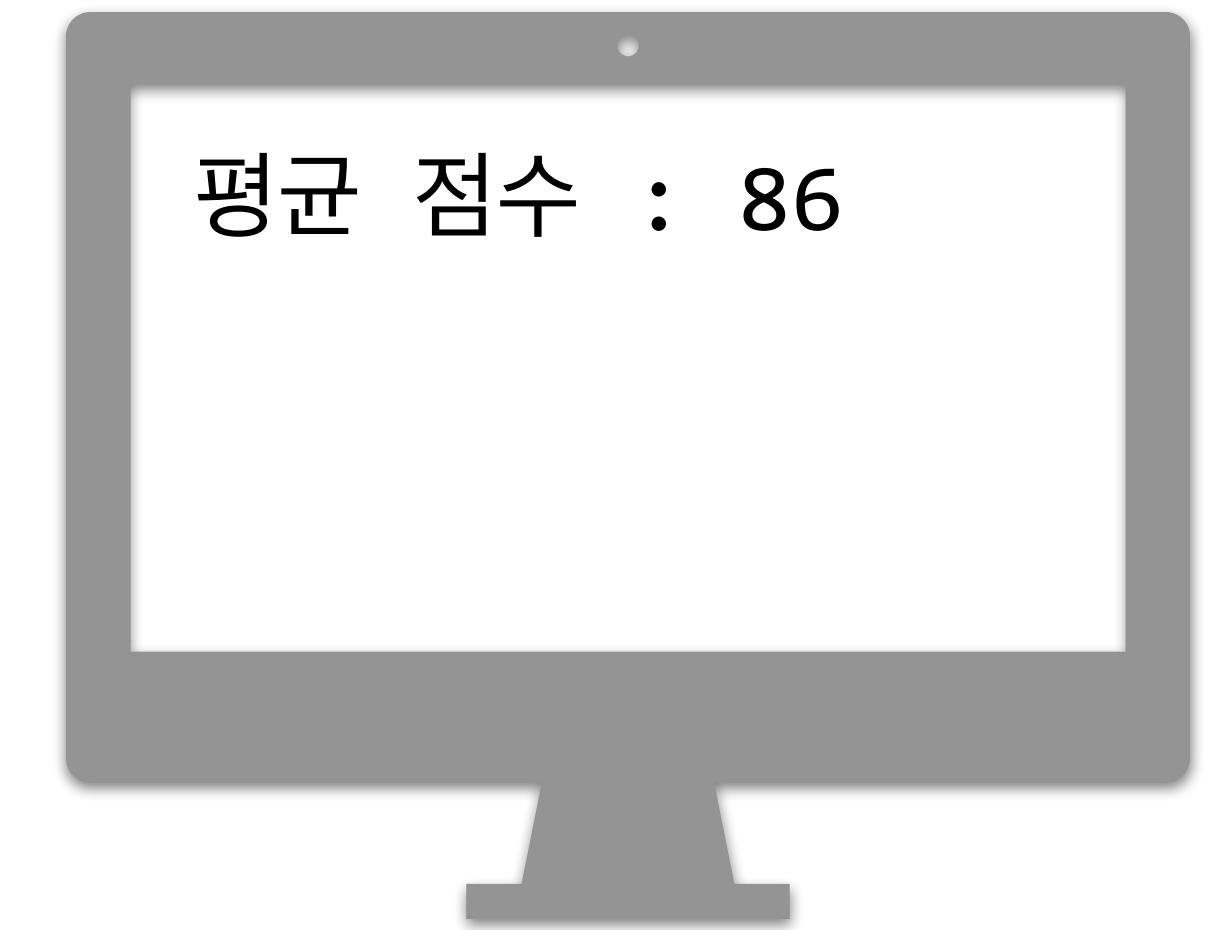
이름	학년	점수
Hong	2	83
Kim	1	90
Park	2	95
Lee	1	80
Hahn	1	88

grade.csv

목표: grade.csv의 표에서 1학년 학생들의 평균 점수를 계산한다.

- 1) 'grade.csv' 파일을 읽는다.
- 2) 표의 데이터를 한 줄씩 읽으면서,
- 3) 학년이 1학년이라면,
- 4) 해당 점수를 총 점수 값에 더하고, 학생 수를 1 증가시킨다.
- 5) 평균 점수를 계산(총 점수 ÷ 학생 수)하여 화면에 출력한다.

평균 점수 : 86





예시 : 1학년 학생들의 평균 점수를 계산하여 화면에 출력하는 프로그램

이름	학년	점수
Hong	2	83
Kim	1	90
Park	2	95
Lee	1	80
Hahn	1	88

grade.csv

순차 처리

```
total = 0  
freshmen = 0  
  
data = open('grade.csv') 입력  
  
for row in data: 반복 처리  
    if row['학년'] == 1: 선택 처리  
        total += row['점수'] 연산 처리  
        freshmen += 1 연산 처리  
  
    else:  
        avg = total / freshmen 연산 처리  
  
print('평균 점수 : ', avg) 출력
```

평균 점수 : 86



- 프로그램 개발 과정

- 프로그램 개발은 다음과 같은 과정을 거친다

1	프로그램 논리 설계 및 개발
2	프로그램 코드화(coding)
3	기계어(machine language)로 변환(컴파일)
4	프로그램 실행과 검증



Step 1 : 프로그램 논리 설계 및 개발

핵심 단계

- 프로그래밍 개발 단계 중 가장 중요한 핵심 단계
- 알고리즘(algorithm) 개발이라고도 함
- 어떤 작업 절차를 어떤 순서로 실행할지 결정

논리 설계에 사용되는 툴(tool)

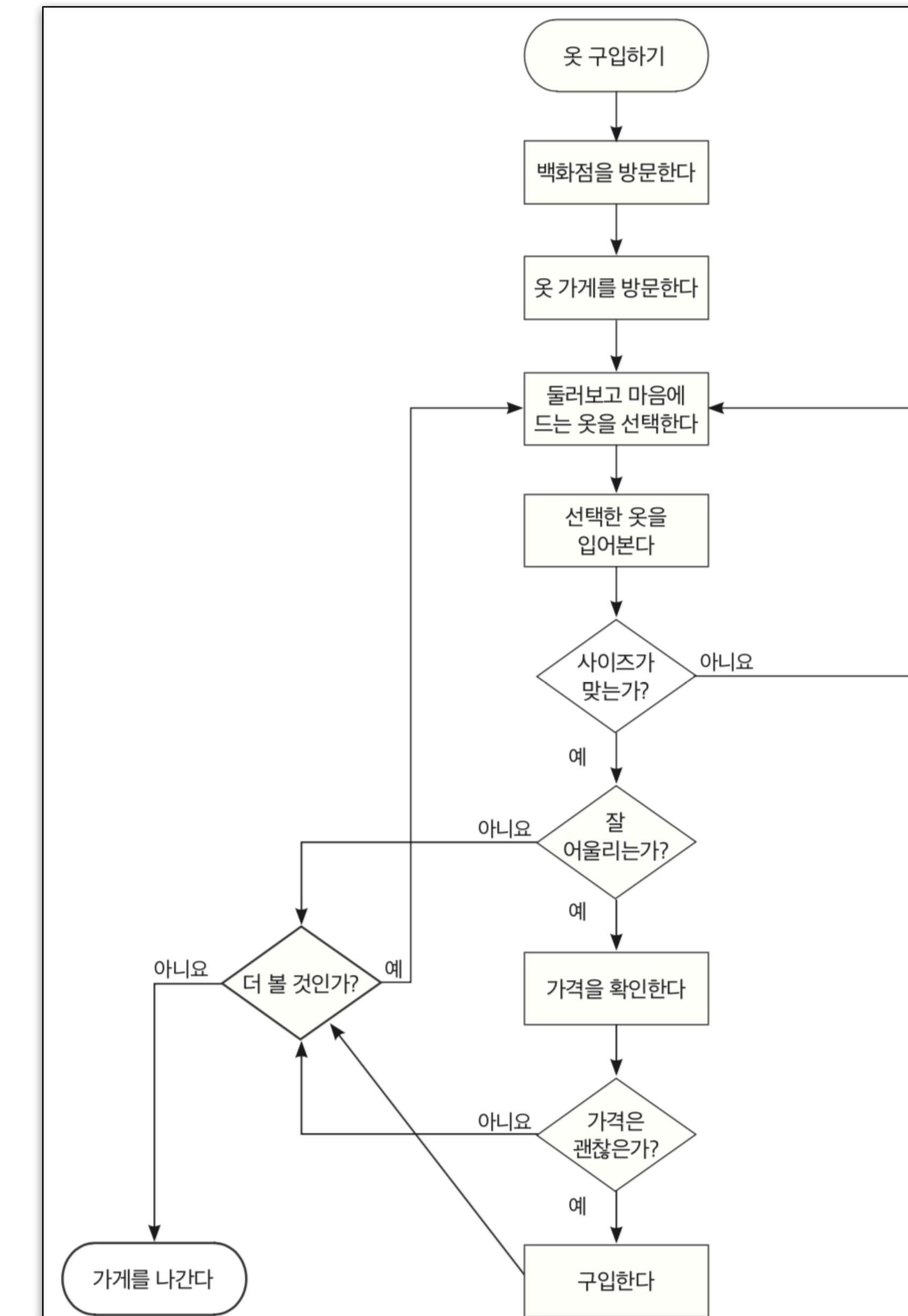
- 흐름도(flowcharts) / 순서도
- 의사(擬似)코드(슈도코드, pseudocode)
- 일상어(자연어)

탁상 검사(desk-checking)

- 눈으로 프로그램의 논리를 검사하는 것
- 언어적 문법은 이 단계에서 고려하지 않음

논리적 오류(logic(al) errors)

- 프로그램이 정상적으로 실행되는 것 같지만 잘못된 결과가 나오는 경우





프로그램 논리 설계 : 컴퓨터처럼 생각하는 방법

● 흐름도(flowchart, 순서도)

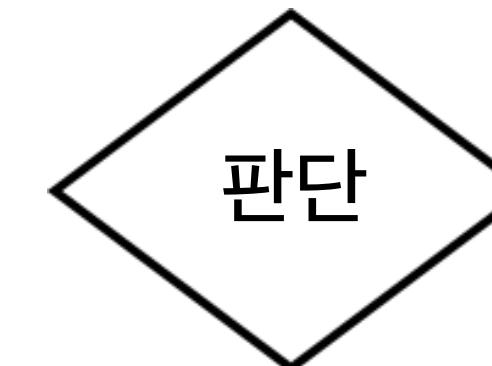
- 프로그래밍이 처리하는 과정을 시각적으로 나타낸 도표

● 단말기호(terminal symbol)

- 프로그램의 시작과 끝을 나타냄



● 비교/판단기호(decision symbol)



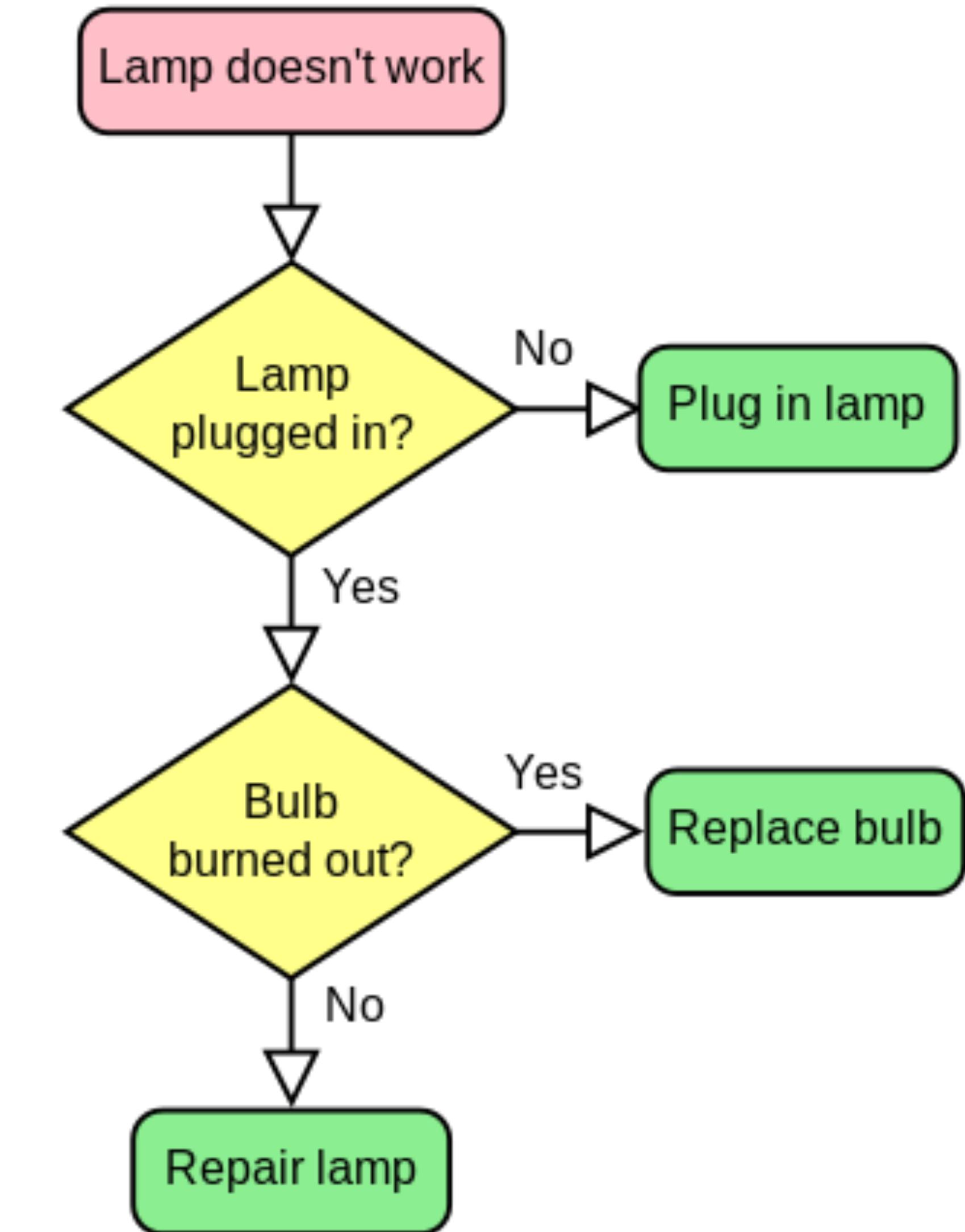
● 입출력기호(input and output symbol)



● 처리기호(processing symbol)



- 각 기호들은 화살표로 연결되어 프로그램의 처리 과정을 나타냄



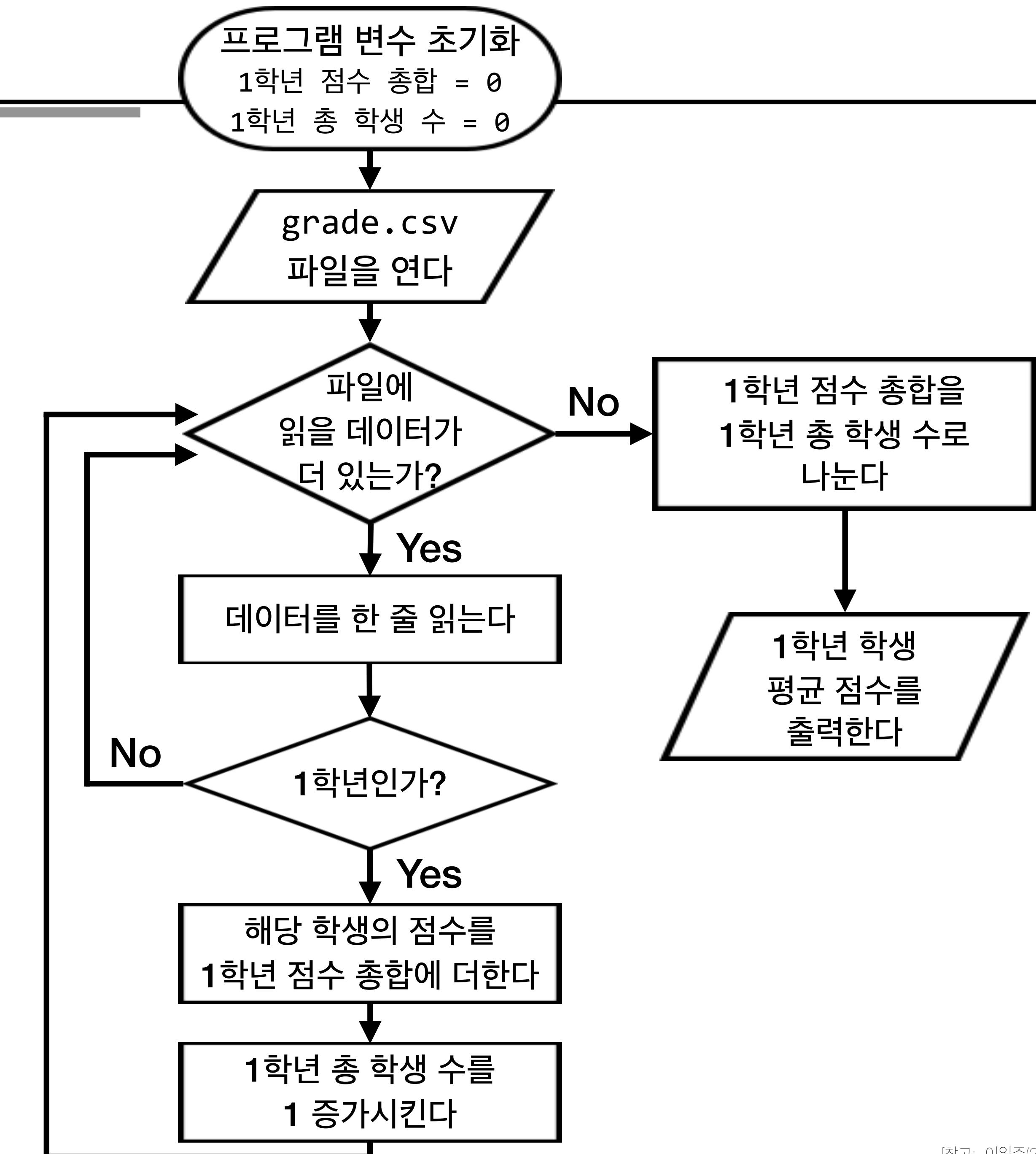
[출처: <https://en.wikipedia.org/wiki/Flowchart>]



예시 : 흐름도

목표: grade.csv의 표에서 1학년 학생들의 평균 점수를 계산한다.

- 1) 'grade.csv' 파일을 읽는다.
- 2) 표의 데이터를 한 줄씩 읽으면서,
- 3) 학년이 1학년이라면,
- 4) 해당 점수를 총 점수 값에 더하고, 학생 수를 1 증가시킨다.
- 5) 평균 점수를 계산(총 점수 ÷ 학생 수)하여 화면에 출력한다.



[참고: 이일주(2021)]



프로그램 논리 설계 : 컴퓨터처럼 생각하는 방법

● 특정한 작업을 수행하기 위한 필요한 과정을 순서대로 나열해 봄

- 반드시 거쳐야 하는 과정들을 논리적 순서대로 정리해서 **알고리즘**을 생성함

☞ **알고리즘(algorithm)** : 특정한 작업을 수행하기 위해 필요한 과정들을 논리적으로 나열한 것

● 의사(擬似)코드(슈도코드, pseudocode)

- 실제 프로그래밍 언어가 아닌 유사한 형태의 언어나 일상어 형태로 작성한 코드

- 프로그램 설계 시에만 사용하므로 문법(구문) 오류를 신경 쓸 필요가 없음

☞ 컴파일 또는 실행에 사용하는 코드가 아님(가짜 코드)

- 의사코드는 어떤 프로그램 언어로든지 실제 코드로 옮겨 적는 것이 가능

- e.g., 녹차 만들기 (일상어 형태)

☞ 주전자에 물을 넣는다;
☞ 주전자의 물을 끓인다;
☞ 주전자의 물이 끓으면 불을 끈다;
☞ 컵에 티백을 넣는다;
☞ 컵에 물을 붓는다;
☞ 약간 기다린 뒤 티백을 꺼낸다;
☞ 마신다;

1학년 학생들의 평균 성적 구하기 의사코드

- 'grade.csv' 파일을 읽는다.
- 표의 데이터를 한 줄씩 읽으면서,
- 학년이 1학년이라면,
- 해당 점수를 총 점수 값에 더하고, 학생 수를 1 증가시킨다.
- 평균 점수를 계산(총 점수 ÷ 학생 수)하여 화면에 출력한다.



● 제어문 의사(擬似)코드(슈도코드, pseudocode) 작성법

- 정해진 규칙은 없지만
- 최소한 프로그래밍 언어의 일반적으로 사용하는 키워드를 사용하여 표현하고
- 블록은 들여쓰기를 할 것을 권장

```
if (condition)
    statements...
else-if (condition)
    statements...
else
    statements...
end if
```

```
while (condition)
    statements...
break if (condition)
    statements...
end while
```

```
for (condition)
    statements...
break if (condition)
    statements...
end for
```

```
function name(input1, input2...)
    statements...
return (output1, output2...)
```

```
class name(input1, input2...)
    statements...
return (output1, output2...)
```

```
output1, output2 = call name(input1, input2...)
```



Step 2 : 프로그램 코드화

● 프로그래밍(programming)

- 파이썬(Python), 자바(Java), R, 루비(Ruby), C, C++, 비주얼 베이직(Visual Basic), SQL 등의 해당 프로그래밍 언어 문법에 맞추어 코드를 작성
- 인간 언어처럼 프로그래밍 언어도 각기 다른 표현과 문법을 가지고 있음
- 일반적으로 프로그램 논리개발보다 코드화 작업이 더 쉬움

```
inputs = Input(shape=(28, 28))
flatten = Flatten()(inputs)
hidden = Dense(256, activation=relu)(flatten)
hidden = Dense(64, activation=relu)(hidden)
hidden = Dense(32, activation=relu)(hidden)
outputs = Dense(10, activation=softmax)(hidden)
model = Model(inputs, outputs)

model.compile(optimizer=Adam(), loss=CategoricalCrossentropy(), metrics=CategoricalAccuracy())
model.fit(X_train, y_train, validation_split=0.2, epochs=100)
```



코드화의 기본 작업

- **선언(declaration)** : 프로그램에서 사용될 자료형(data type)를 컴퓨터에게 알려주는 단계

- 데이터는 사용하기 전에 반드시 미리 선언 (일반적으로 코드화 작업 앞부분에서 선언)
 - 자료형을 선언함으로써 메모리에 차지할 저장 공간과 쓰임새를 컴퓨터에게 미리 알려줌

- **입력(input)** : 컴퓨터에 데이터를 입력하는 것

- 사람으로부터 : 키보드, 마우스, 펜 등
 - 파일로부터 : 텍스트 파일, 데이터베이스, 웹 페이지 등
 - 센서로부터 : 빛, 동작 감지, 생체인식 등

- **처리(processing)** : 프로그램에 의해 수행되는 작업

- e.g., 급여 계산, 체스 게임에서 말의 이동

- **출력(output)** : 결과 또는 답

- 화면에 표시
 - 파일로 저장
 - 종이에 출력

```
total = 0  
freshmen = 0  
  
data = open('grade.csv')  
  
for row in data:  
    if row['학년'] == 1:  
        total += row['점수']  
        freshmen += 1  
    else:  
        avg = total / freshmen  
  
print('평균 점수 : ', avg)
```

선언

입력

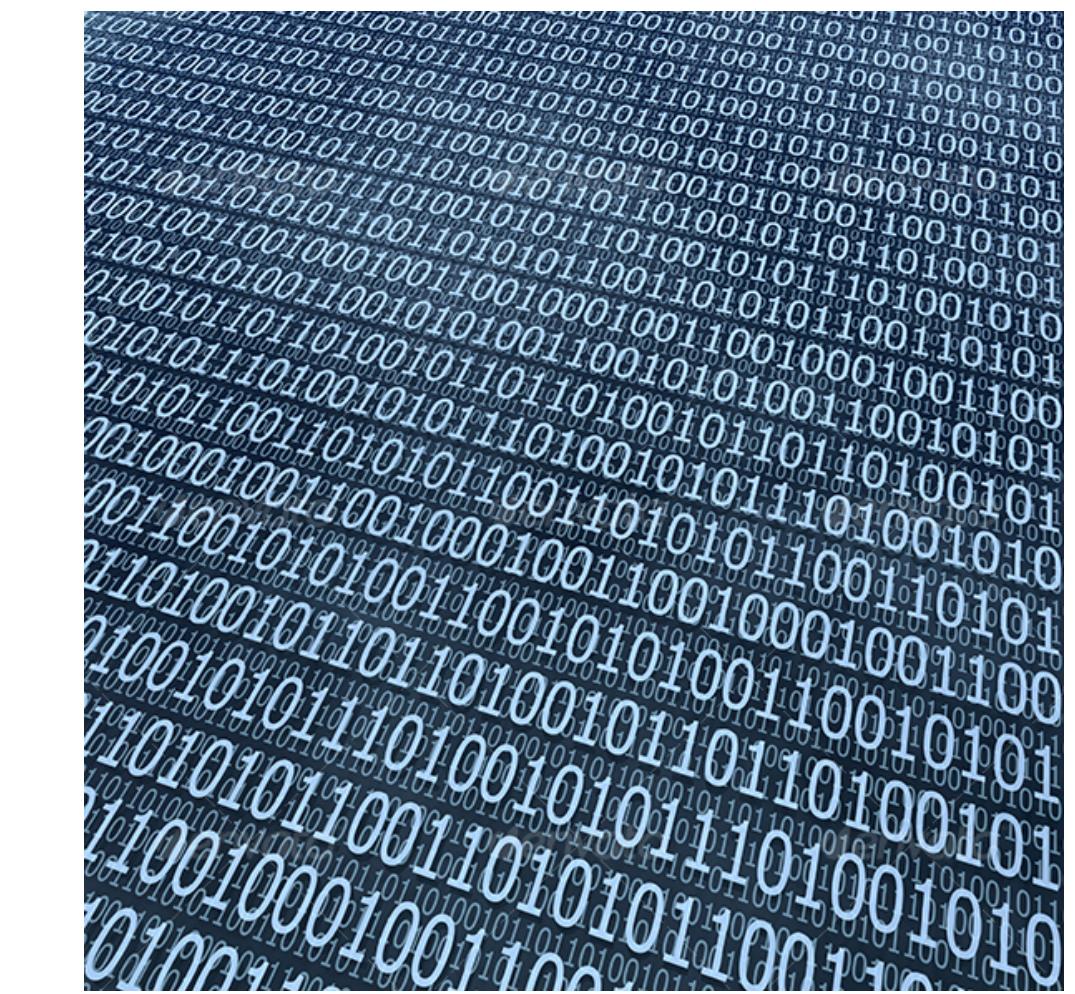
처리

출력



Step 3 : 기계어(machine language)로 변환(컴파일)

- 컴퓨터는 1 또는 0 밖에 모른다
 - 소프트웨어를 통해 프로그램을 컴퓨터가 이해할 수 있는 기계어(0과 1)로 변환
 - 다양한 프로그래밍 언어가 있지만 컴퓨터는 오로지 1 또는 0만 이해할 수 있기 때문에 어떤 언어든지 기계어로 변환이 필요
- 컴파일러(compiler)와 인터프리터(interpreter)
 - 일상어와 유사한 형태를 지닌 고수준 프로그래밍 언어를 저수준 기계어로 변환해주는 방식
- 문법 오류(syntax error)
 - 컴파일러가 코드를 변환할 수 없을 때 문법 오류가 발생
 - 원인
 - 오타
 - 잘못된 문법
 - 코드에서 잘못된 부분을 수정한 뒤 다시 변환(recompile)해야 함



A large grid of binary code (0s and 1s) representing machine language. The grid consists of approximately 20 columns and 30 rows of binary digits, forming a dense pattern of black and white squares.

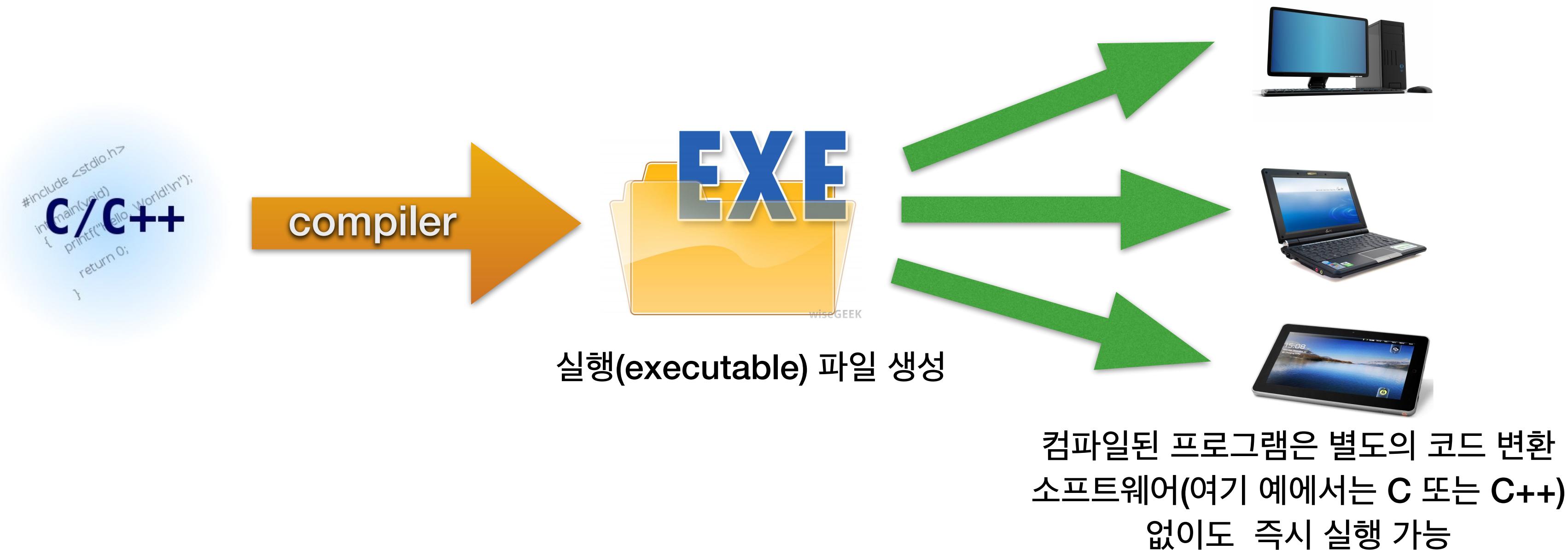


컴파일러

- 고수준 언어로 작성된 프로그램을 컴퓨터가 ‘이해’할 수 있는 별도의 기계어 프로그램(1 또는 0으로 구성된)으로 변환해주는 프로그램
- 고수준 언어 프로그램은 실행하기 전에 기계어로 변환하는 과정(compile)이 필요한 반면, 기계어(원시코드, native code) 프로그램은 바로 실행이 가능

컴파일러의 장점

- 미리 기계어로 변환이 되었기 때문에 **빠른 실행**이 가능
- 기계어로 변환된 프로그램은 역설계(reverse-engineer)을 통해 원본 코드(source code)가 유출되는 것을 어렵게 함



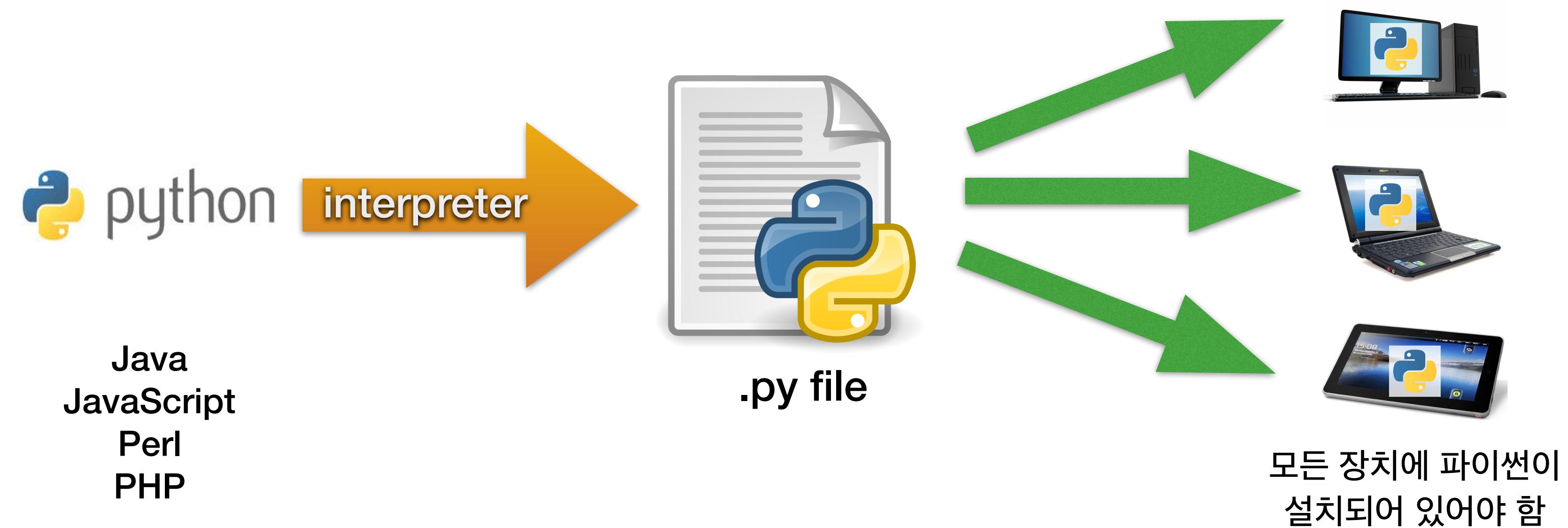


인터프리터

- 고수준 언어 프로그램의 명령어들을 변환하고 실행하는 프로그램
- 한번에 한 줄씩 명령문을 해석함(코드가 실시간으로 한 줄씩 실행됨)
- 컴파일러와 달리 기계어로 변환된 별도의 실행(executable) 파일이 생성되지 않음(즉, 프로그램이 원시코드(native code)로 변환되지는 않음)
- 따라서 프로그램을 실행하려는 장치에 반드시 인터프리터가 설치되어 있어야 함

인터프리터의 장점

- 인터프리터만 설치되어 있으면 어떤 장치에서도 실행이 가능(platform independence)
- 인터프리터로 작성된 프로그램은 상대적으로 파일 크기가 작음





Step 4 : 프로그램 실행과 검증

● 프로그램 오류의 종류와 디버깅(debugging)

● 문법 오류(syntax error)

- ➊ 오류 수정이 쉬움(즉, 버그 잡기가 쉬움)

● 런타임 오류(runtime error)

- ➋ 예외(exception) 처리를 해야 함

● 논리 오류(logical error)

- ➌ 오류 없이 프로그램이 실행되지만 예상한 결과가 나오지 않는 경우

- ➍ $1 / 2 * 3$

- ➎ 문법 오류는 컴파일 단계에서, 런타임 오류와 논리 오류는 실행 단계에서 발생



● 실행 단계에서의 오류

- ➏ 문법 오류가 없는 프로그램이라도 실행단계에서 오류가 발생하지 않는다고 말할 수 없음
- ➐ 실행 단계에서 오류를 피하기 위해서는 샘플 데이터를 넣어 테스트 해보는 것이 좋음
- ➑ 샘플 데이터는 신중히 선택하고 오류 발생 가능한 다양한 경우의 수를 고려 —> be a creative pessimist!
- ➒ 발견된 오류로 인해 프로그램 논리를 변경해야 할 수도 있음



프로그램 논리 설계

Lab Exercises





Lab : 의사코드를 파이썬 코드로 변환



- 의사코드 예시

- 사용자가 입력한 값이 짝수인지 홀수인지 구하기

- 파이썬 코드로 변환 한 예시