

Python & Data Engineering

인공지능 직무전환자 과정

- Day 1
Python Basic



Sanghyun Seo

0. Getting Started

Instructor introduction

- 서상현 (Sanghyun Seo)

Course Descriptions

- Python & Data Engineering

Day 1 – Python Basic	Day 2 – Advanced Python	Day 3 – Numpy, Pandas	Day 4 – Visualization, Data Analysis
<ul style="list-style-type: none">• Getting started• Introduction to Python• Data Type & Variable• Flow control• Function• Python programming practice	<ul style="list-style-type: none">• Review• File• Class• Exception Handling• Advanced python	<ul style="list-style-type: none">• Review• Module, Package• Numpy Basic• Advanced Numpy• Pandas Basic• Advanced Pandas	<ul style="list-style-type: none">• Review• Introduction to machine learning• Data preprocessing• Visualization• Course Summerization

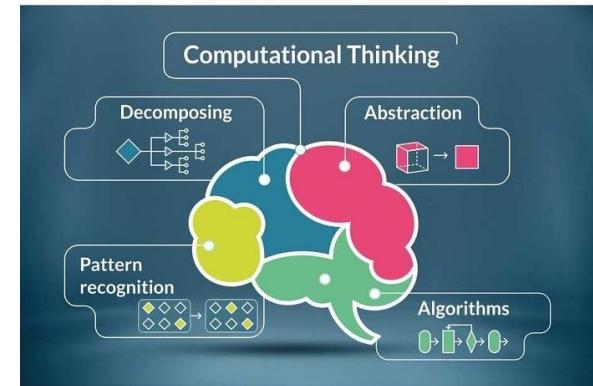
Contents

1. Introduction to Python
2. Data Type & Variable
3. Flow Control
4. Function
5. Python Programming Practice

1. Introduction to Python

Computational Thinking

- **분해(decomposition)**
 - 문제를 잘게 나눠서, 문제를 작고 다룰 수 있는 부분으로 나누기
 - 일을 할 때 어떻게 나눠서 할지 생각하는 것
- **패턴 인식(pattern recognition)**
 - 데이터 안에 있는 패턴과 규칙을 찾기
 - 부분과 부분의 유사점과 차이점을 찾는 것
- **패턴 만들기(추상화) (abstraction)**
 - 패턴을 만드는 일반적인 규칙을 발견하는 것
- **알고리즘 (algorithm)**
 - 문제를 풀기 위한 명령을 순서대로 만드는 것

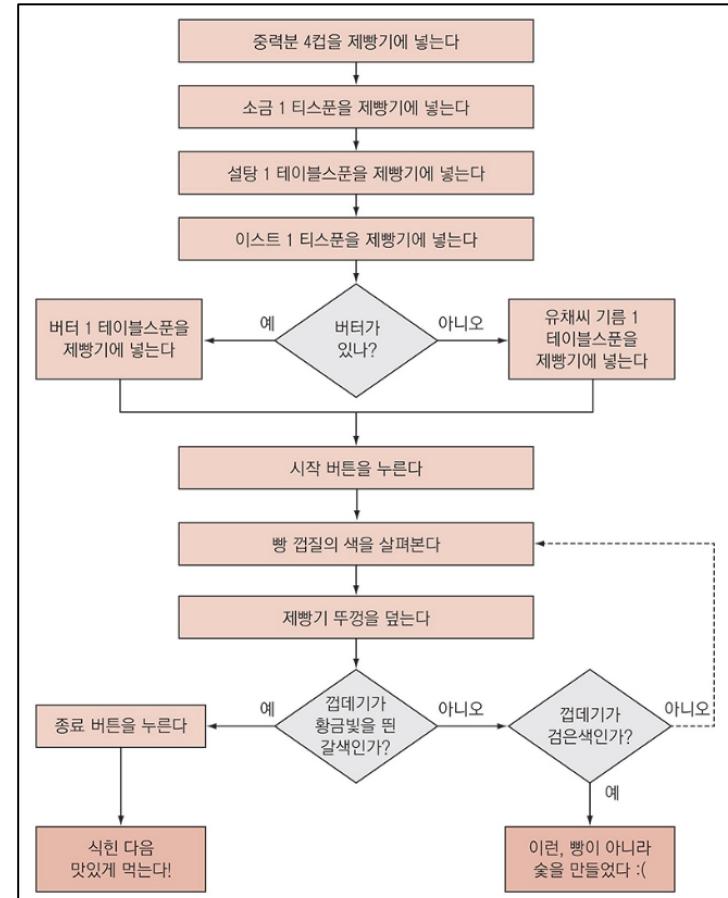


Computational Thinking

- 레시피 → 작업흐름도 → 프로그래밍

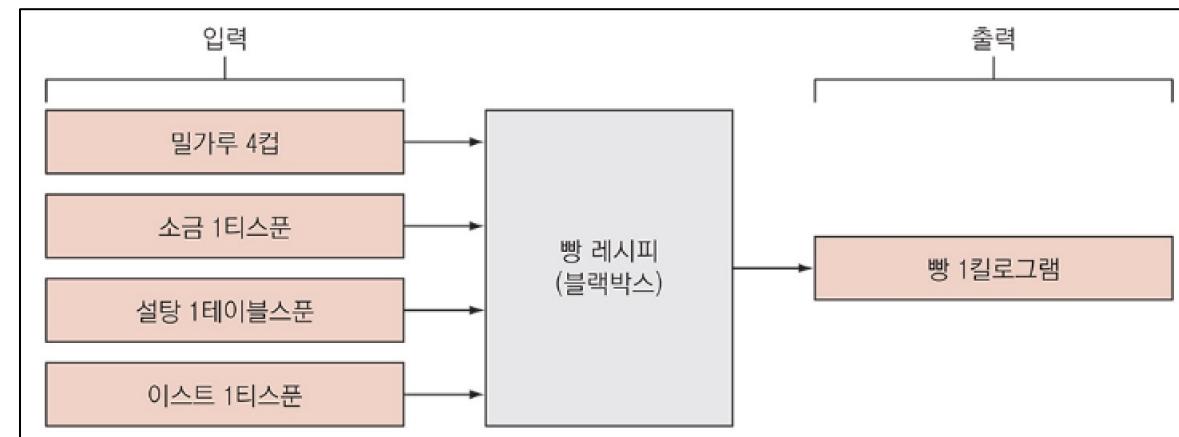
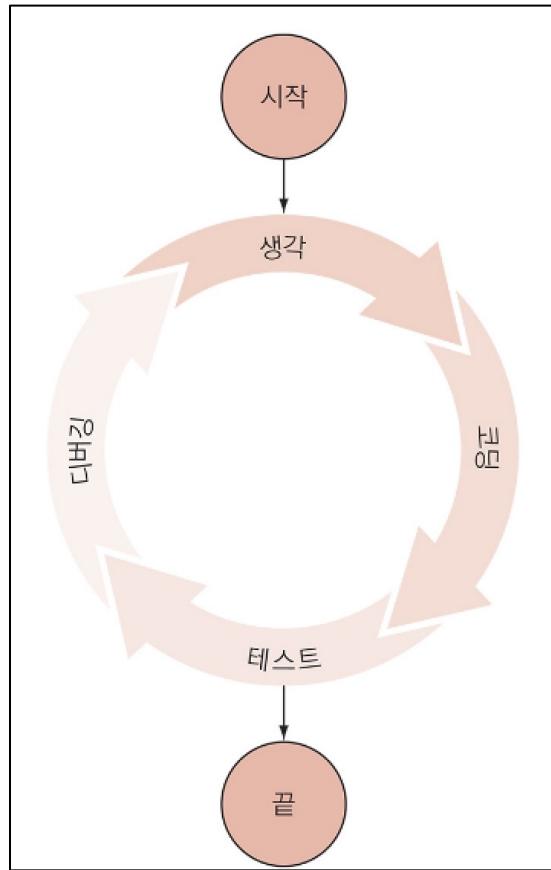
양	재료
1/4 온스	활성 드라이이스트
1 테이블스푼	소금
2 테이블스푼	버터(유채씨 기름으로 대체 가능)
3 테이블스푼	설탕
2-1/4 컵	따뜻한 물
6-1/2 컵	중력분

1. 큰 그릇에 따뜻한 물을 붓고 이스트를 푼다.
설탕, 소금, 버터(기름), 중력분 3컵을 넣고 부드러워질 때까지 젓는다.
2. 반죽이 부드러워질 때까지 남은 밀가루를 휘젓는다.
3. 밀가루를 뿐린 반죽판에 반죽을 놓는다.
4. 겉이 부드러워지고 반죽에 탄력이 생길 때까지 치댄다.
5. 기름을 바른 그릇에 담고 반죽을 한번 돌려서 전체에 기름을 묻힌다.
6. 반죽이 든 그릇을 덮고 따뜻한 곳에 두어 부피가 2배 될 때까지
30분~1시간 숙성시킨다.
7. 반죽을 세게 내려쳐 이스트가 부풀면서 생긴 기포를 없애고,
밀가루를 살짝 뿐린 판에 반죽을 놓는다.
8. 반죽을 두 덩어리로 나눈다. 각각 모양을 잡는다.
기름을 바른 가로 25cm 세로 13cm 베이킹 판 두 개에 각각 반죽을 넣는다.
9. 반죽을 덮고 부피가 2배 될 때까지 30~45분간 반죽을 재운다.
10. 190도에서 30분간(또는 껍질이 황금빛을 띤 갈색이 될 때까지) 굽는다.
11. 베이킹 판에서 빵을 꺼내서 철사 선반에 올려두고 식힌다.
12. 잘라서 맛있게 먹는다!



Computational Thinking

- 레시피 → 작업흐름도 → 프로그래밍



Computational Thinking

- **프로그래밍의 기본 개념**

- 변수, 식, 문장 사용하기
- 조건에 따라 결정하기
- 특정 조건이 성립하는 동안 작업을 반복 수행하기
- 보다 더 효율적인 프로그램 만들기
- 코드를 더 읽기 좋게 만드는 방법과 큰 작업을 작은 작업으로 나눠 유지보수(maintenance) 하기 쉽게 만들기
- 특정 상황에 대해 어떤 데이터 구조가 적합한지 알기

Programming

- **프로그래밍(programming)**
 - 하나 이상의 관련된 추상 알고리즘을 특정한 프로그래밍 언어를 이용해 구체적인 컴퓨터 프로그램으로 구현하는 기술
 - 특정한 프로그래밍 언어로 쓰인 프로그램은 기계어로 번역되어 컴퓨터에 의해 실행됨
- **프로그래밍 언어(programming language)**
 - 프로그래밍 언어는 컴퓨터 시스템을 구동시키는 소프트웨어를 작성하기 위한 형식언어

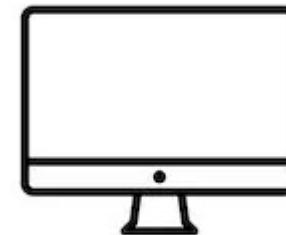
Natural Language



Programming Language

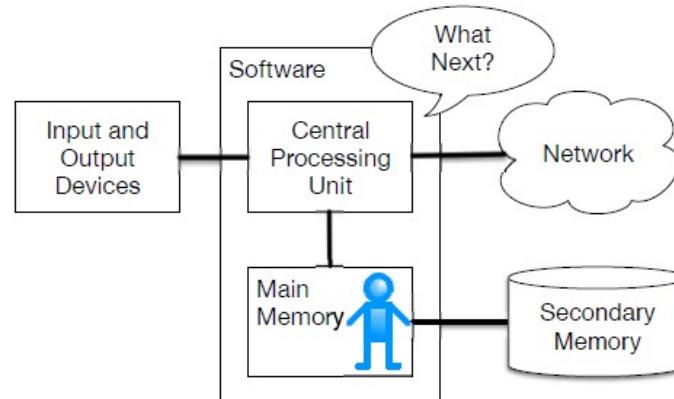


Machine Language



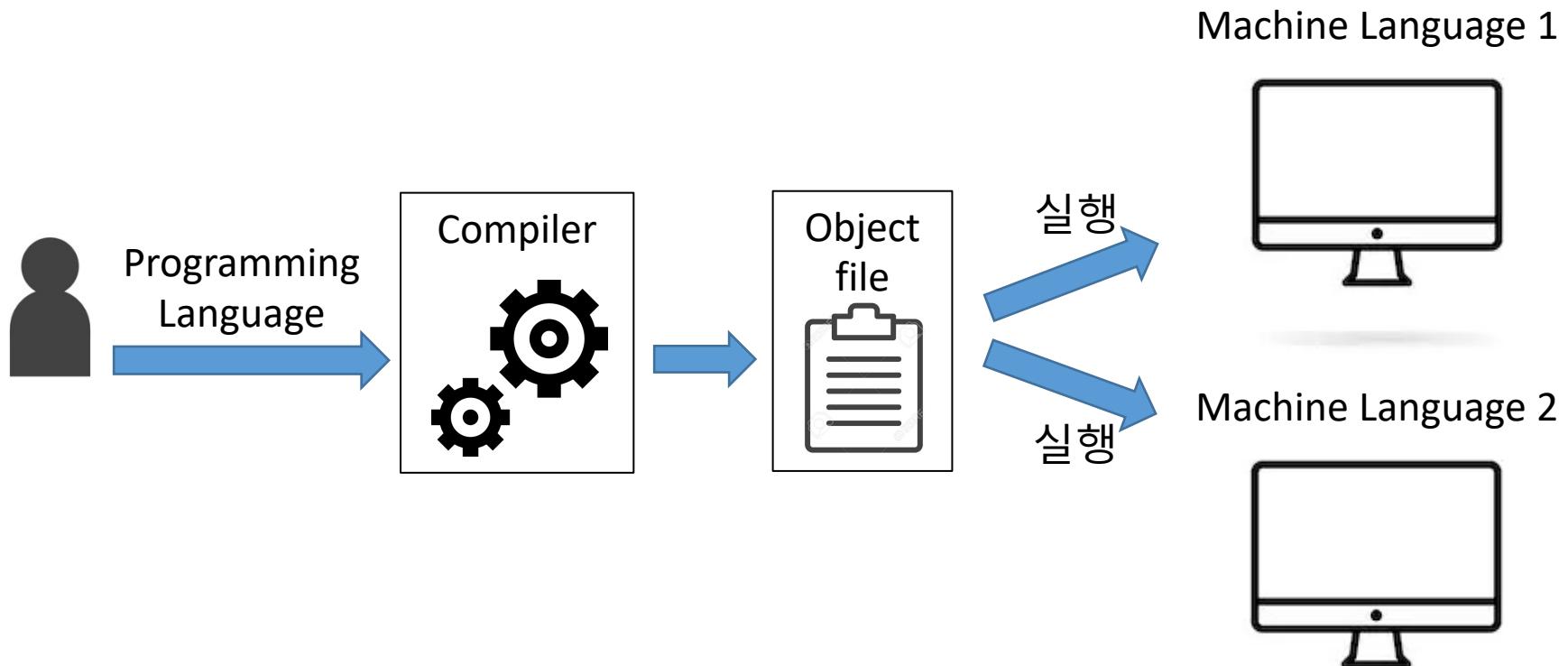
Hardware Architecture

- 중앙처리장치(central processing unit, CPU)
 - 주어진 정보에 대해서 입력된 명령어에 따라서 정보처리
- 주기억장치(main memory)
 - CPU가 명령어를 처리하는데 필요한 정보를 저장
 - 정보처리 속도가 빠르지만 전원이 꺼지면 정보를 상실
- 보조 기억장치(secondary memory)
 - 정보처리 속도가 느리지만 전원을 잃어도 지속적으로 정보 보관



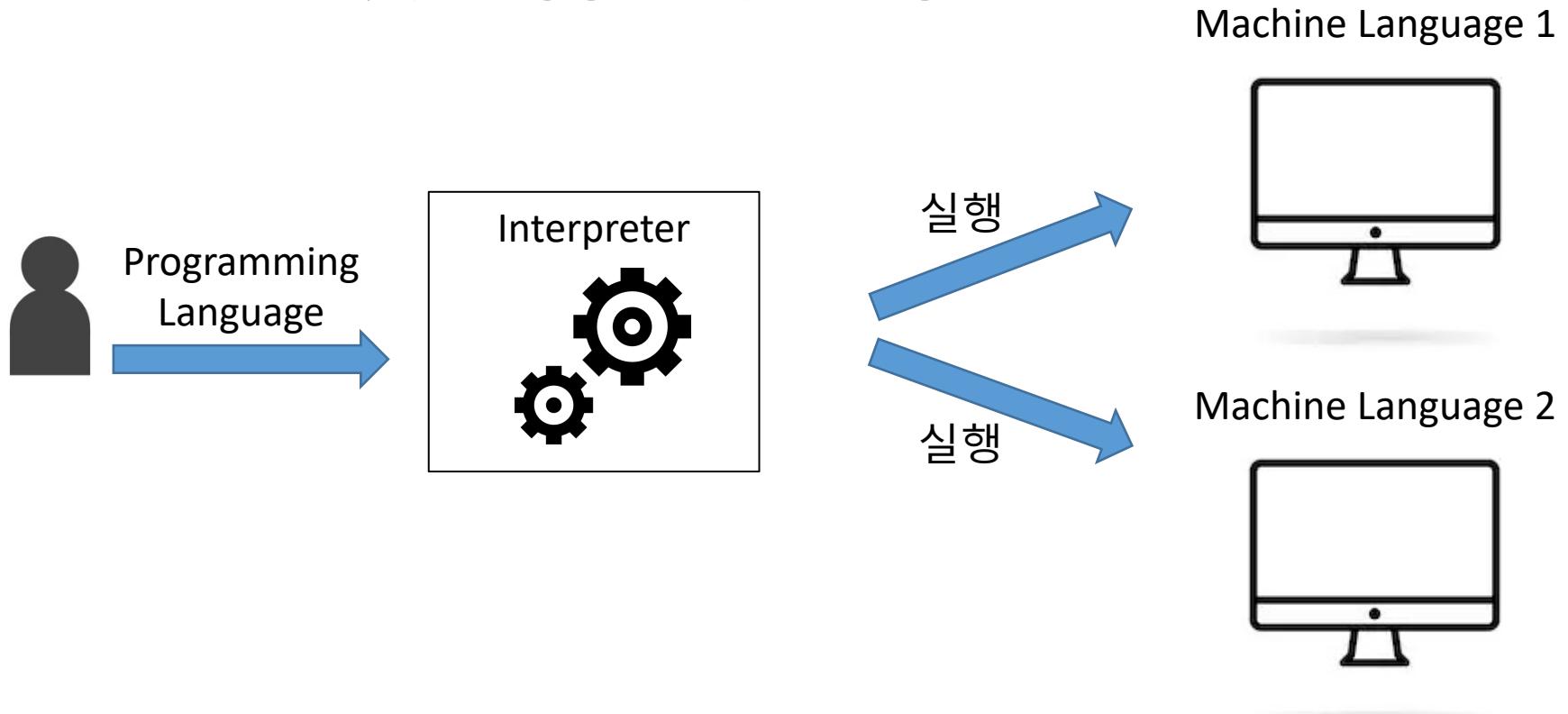
Programming languages

- 컴파일러
 - 프로그래머가 코드를 작성할 때 소스 코드를 이용하여 목적 코드(파일)을 컴파일하고, 필요시 이를 실행하는 방식으로 프로그램 실행



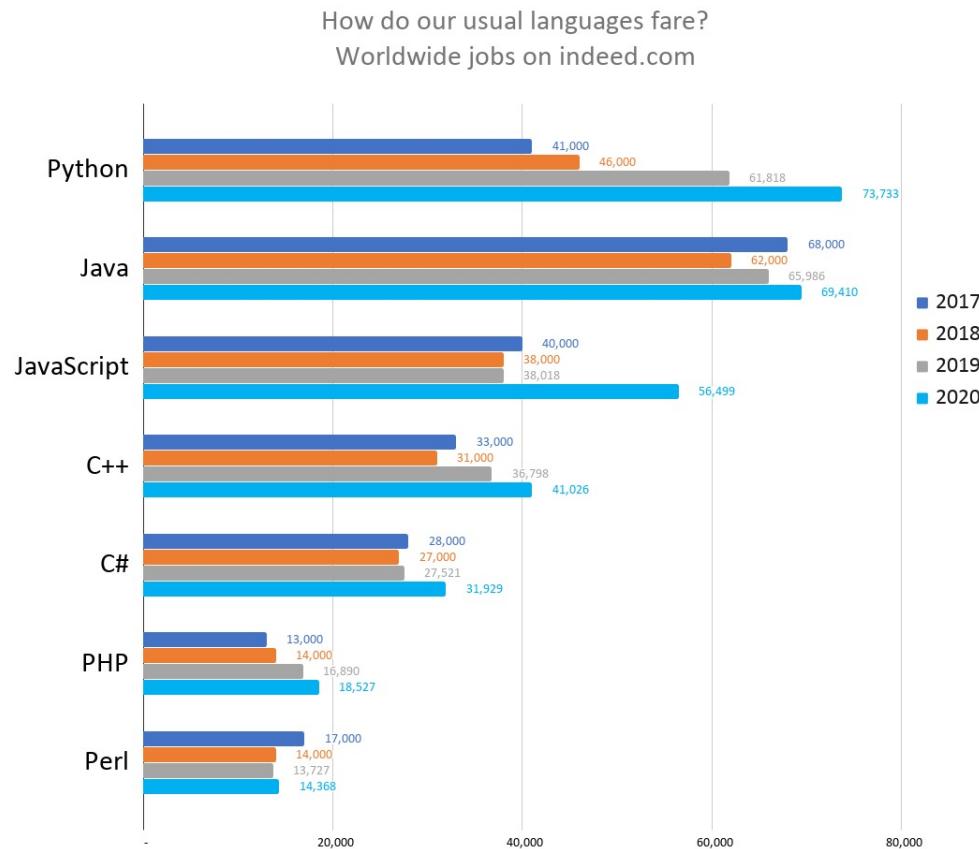
Programming languages

- 인터프리터
 - 프로그래머가 코드를 작성할 때 소스 코드를 읽고, 소스코드를 파싱하고, 즉석에서 명령을 해석하여 실행



Programming languages

- 프로그래밍 언어 점유율 비교



Python

- Python

- 파이썬(Python)은 1990년 암스테르담의 귀도 반 로섬(Guido Van Rossum)이 개발한 **인터프리터(interpreter)** 언어



- 귀도는 파이썬이라는 이름을 자신이 좋아하는 코미디 쇼인 "Monty Python's Flying Circus"에서 따옴
- 파이썬의 사전적 의미는 고대 신화에 나오는 파르나소스 산의 동굴에 살던 큰 뱀을 뜻하며, 아폴로 신이 델파이에서 파이썬을 퇴치했다는 이야기가 전해지고 있음
- 대부분의 파이썬 책 표지와 아이콘이 뱀 모양으로 그려져 있는 이유가 여기에 있음

Python

특징	활용분야
High-level Language Easy grammar Open source Concise Fast development speed	시스템 유틸리티 제작 GUI 프로그래밍 웹 프로그래밍 수치 연산 (with C/C++) numpy 데이터베이스 프로그래밍 데이터 분석 pandas, matplotlib

Python 설치 - overview

- Python 설치
 - “Hello, world!” 출력
- Vscode 설치
 - Python interpreter 선택
- Anaconda 설치
 - Conda:Python 가상환경 설치
- Jupyter Notebook 설치
 - Conda: jupyter notebook 설치 및 실행
- Colab 실행
 - Google colab 실행 및 활용

Python 설치 - python

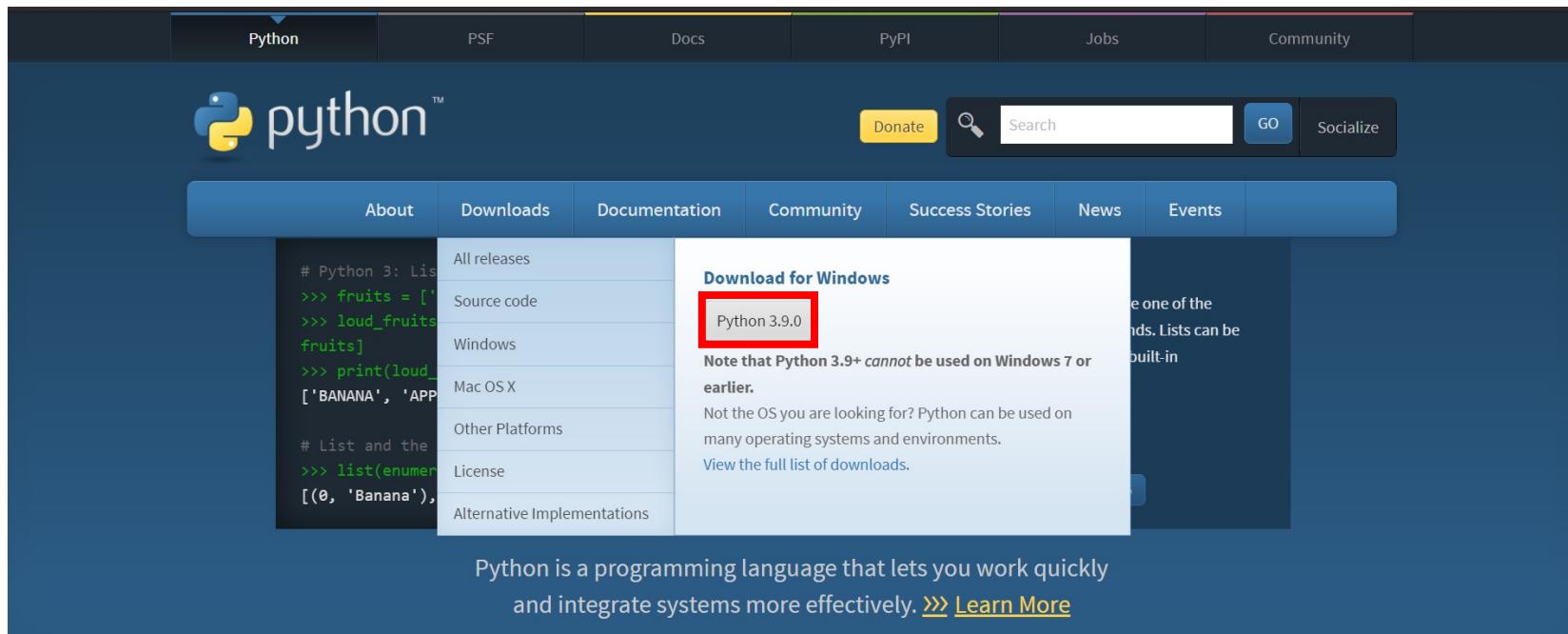
- Python official page
 - <https://www.python.org/>

The screenshot shows the Python official website's homepage. The top navigation bar includes links for Python, PSF, Docs (highlighted in yellow), PyPI, Jobs, and Community. Below the navigation is the Python logo and a search bar with a 'GO' button. A secondary navigation bar below the logo includes links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features two columns. The left column contains code snippets demonstrating Python list comprehensions and the enumerate function:# Python 3: List comprehensions
>>> fruits = ['Banana', 'Apple', 'Lime']
>>> loud_fruits = [fruit.upper() for fruit in
fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']

List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]The right column has a section titled "Compound Data Types" with a brief description of lists and a link to "More about lists in Python 3". At the bottom of the page, there is a footer with a "Learn More" button.

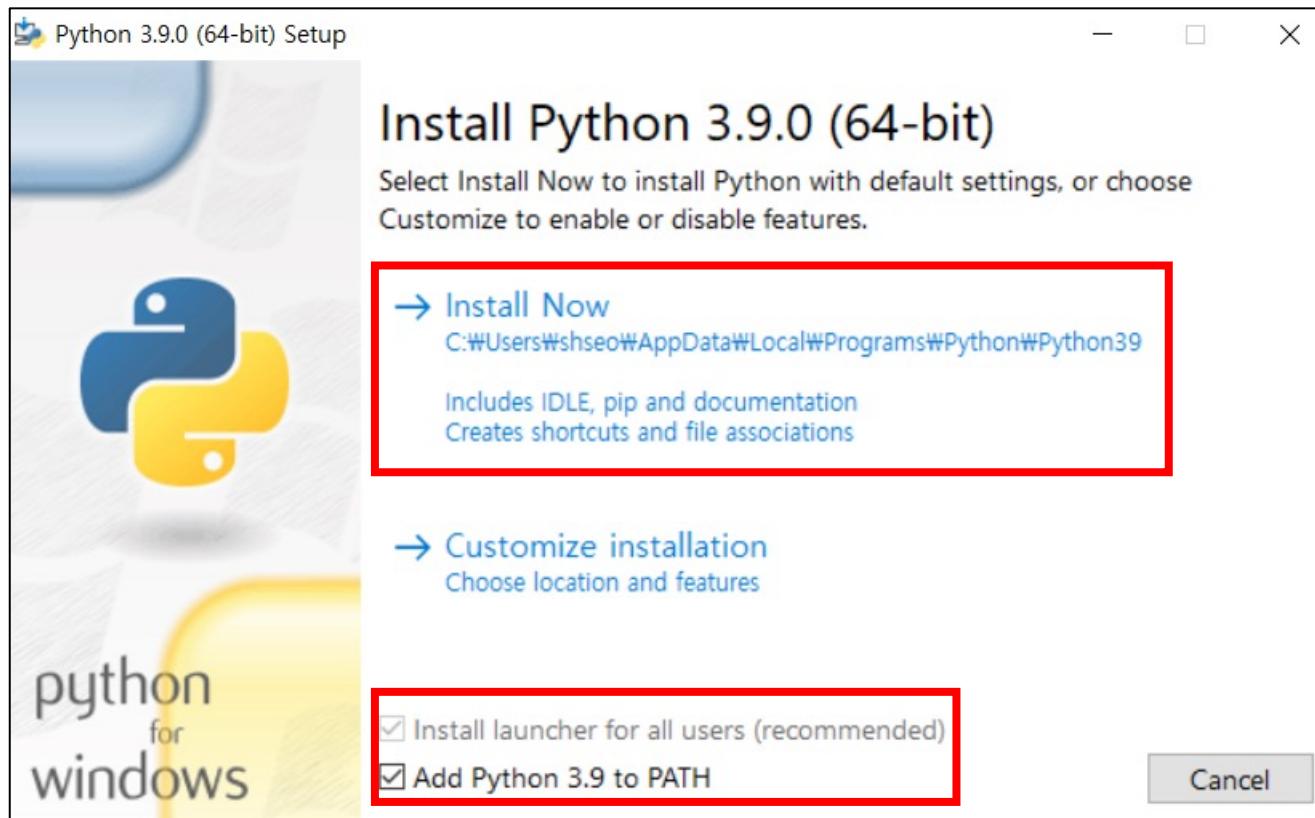
Python 설치 - python

- Python official page
 - Download → python 3.9



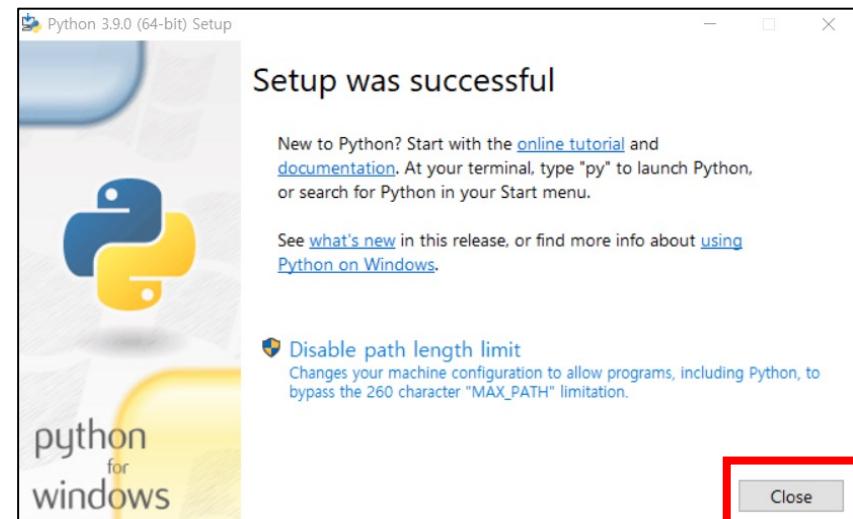
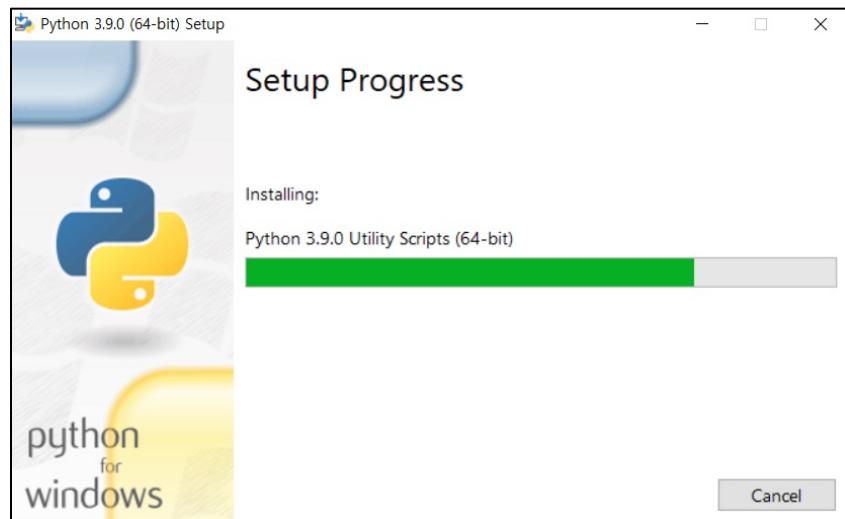
Python 설치 - python

- Python installer 실행



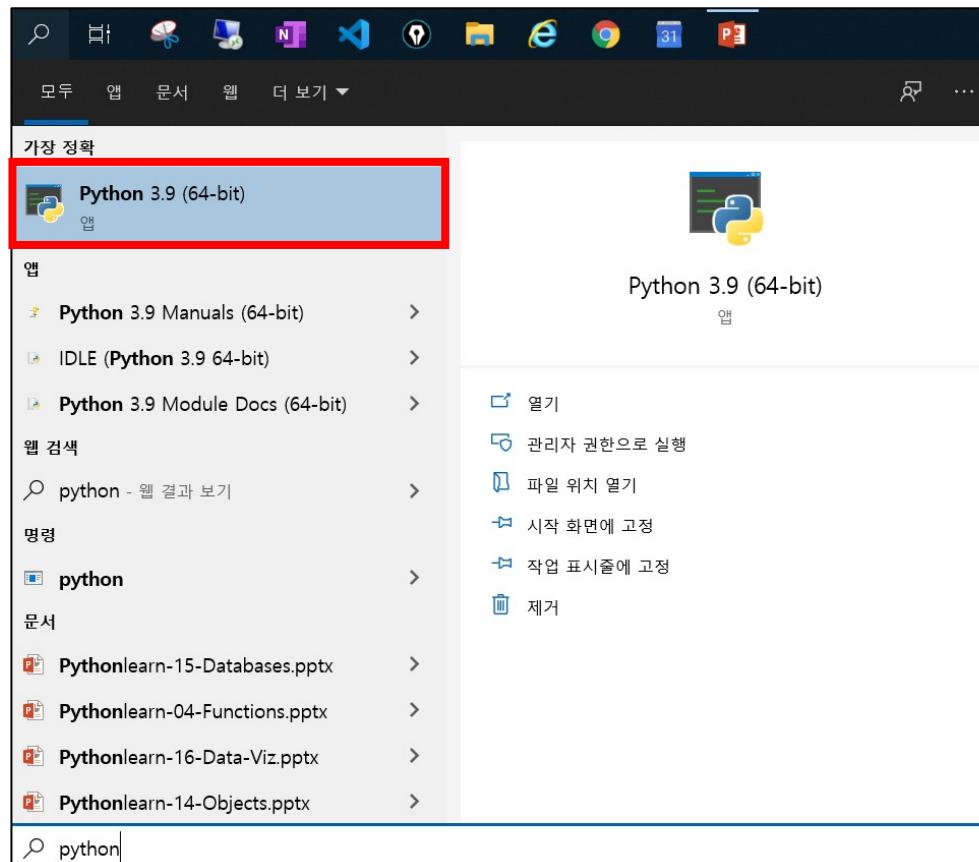
Python 설치 - python

- Python install 진행 및 종료



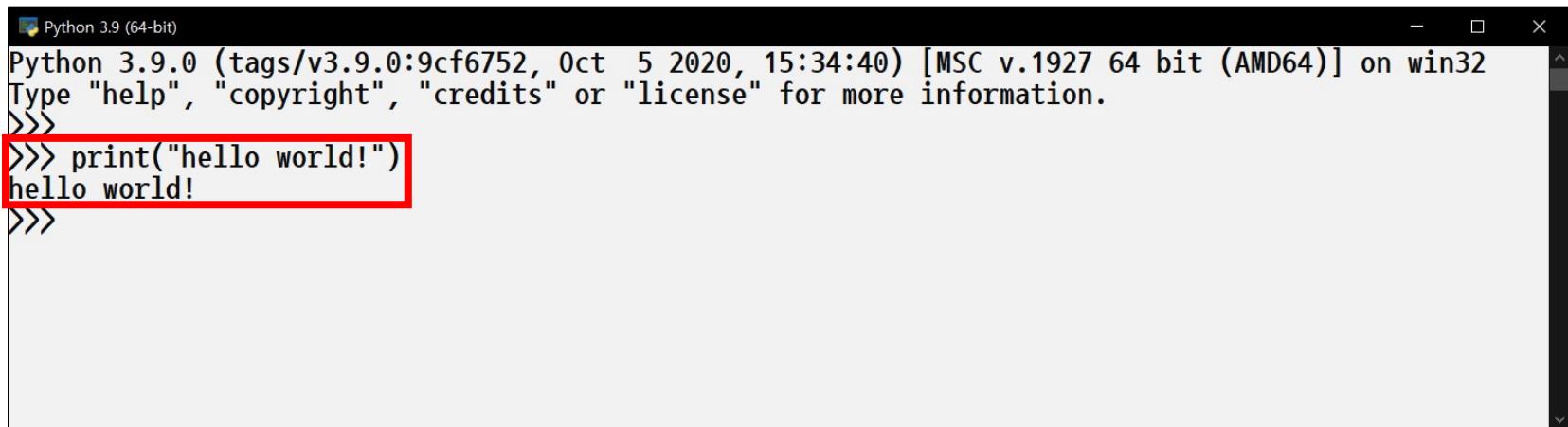
Python 실행 – interactive mode

- Python app 실행
 - Windows 버튼 → ‘python’ 입력 → python 명령 프롬포트 실행



Python 설치 - interactive mode

- “hello_world” 출력 명령
 - print(“hello world”) 입력

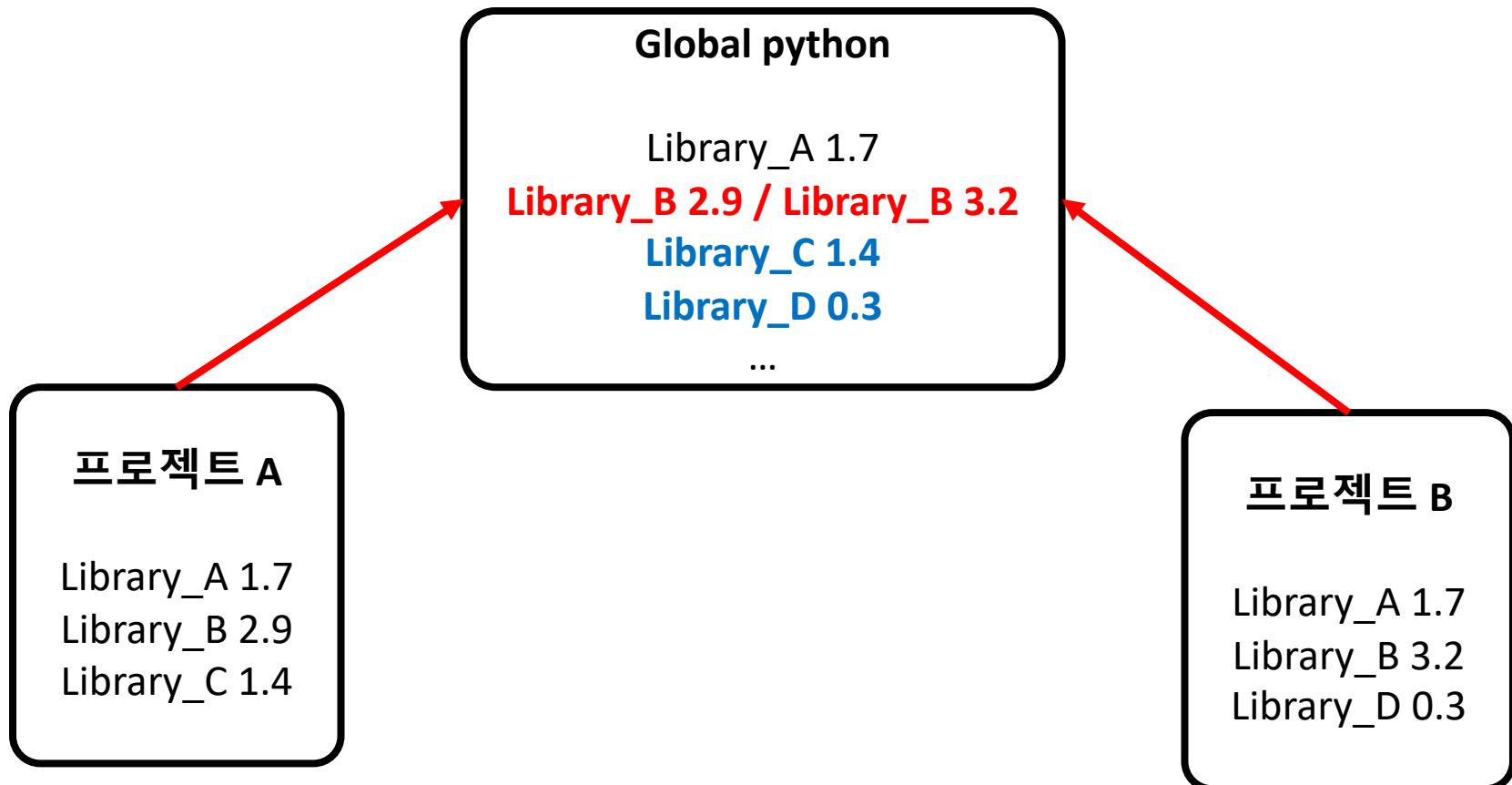


```
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> >>> print("hello world!")
hello world!
>>>
```

The screenshot shows a Windows terminal window titled "Python 3.9 (64-bit)". It displays the Python 3.9.0 startup message and a command-line interface. A red box highlights the command "print("hello world!")" which has been entered by the user. The terminal also shows the resulting output "hello world!".

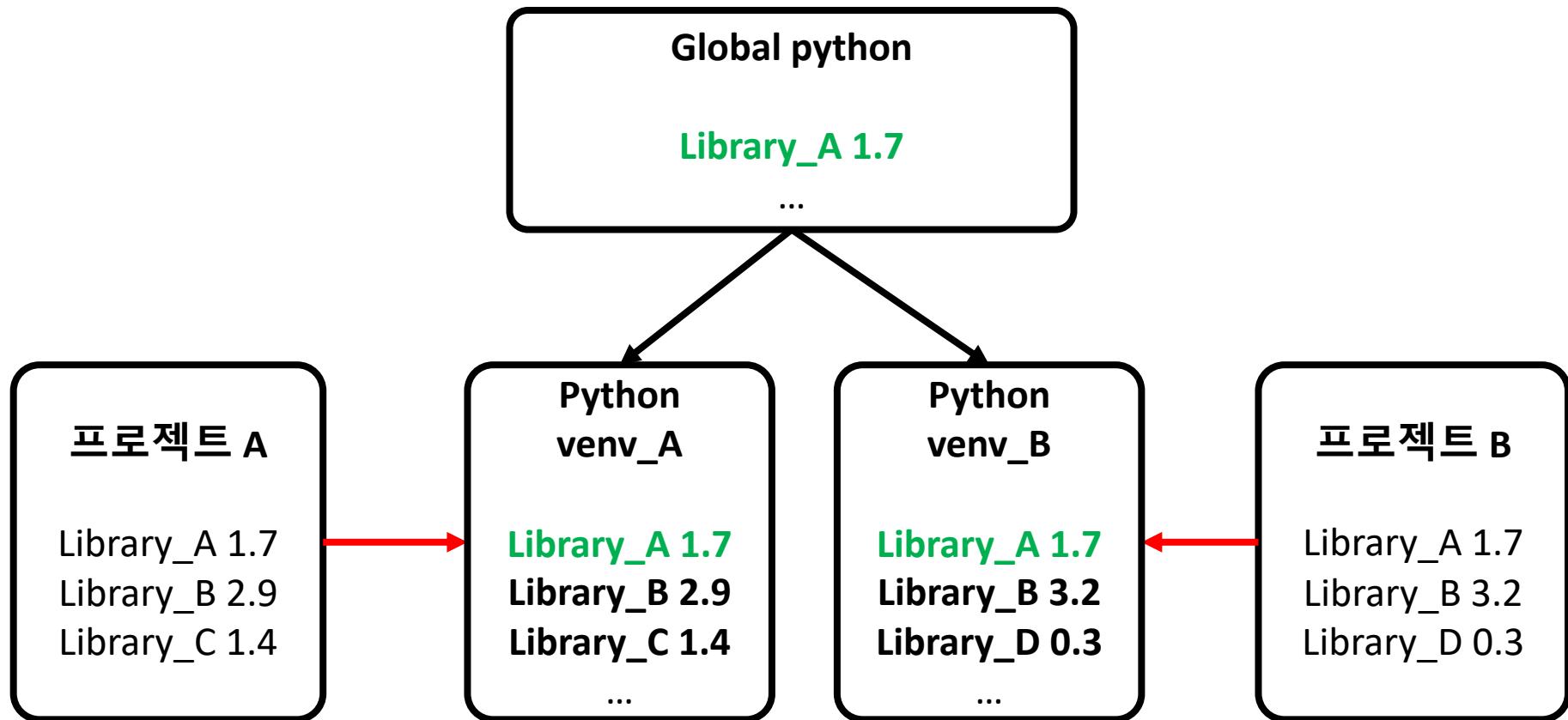
Python 설치 - 가상환경

- 파이썬 가상환경(virtual environment) 필요성



Python 설치 - 가상환경

- 파이썬 가상환경(virtual environment) 필요성



Python 설치 - 가상환경

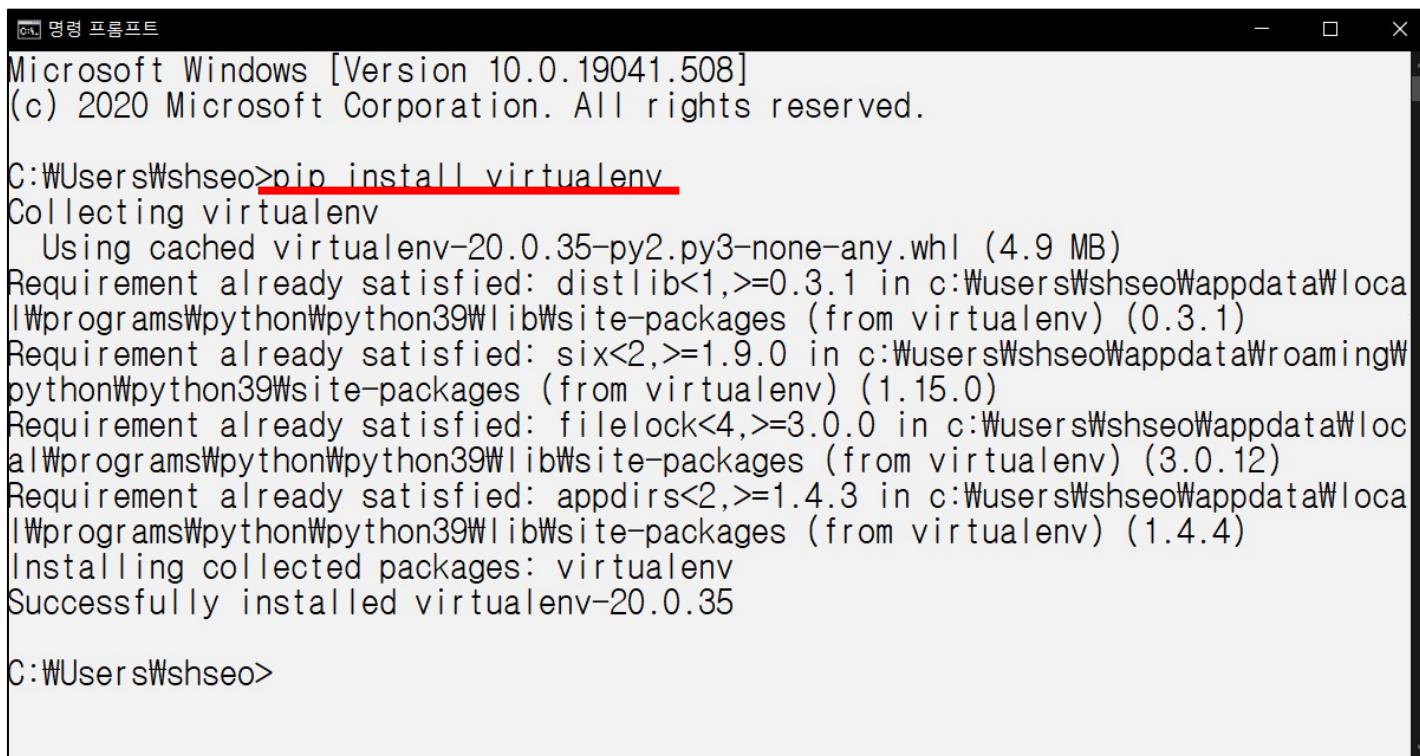
- PyPI (python package index)
 - pip은 파이썬으로 작성된 패키지 소프트웨어를 설치 · 관리하는 패키지 관리 시스템



- pip install --upgrade pip
- <example>
 - pip install [패키지 이름] / pip uninstall [패키지 이름]
 - pip install [패키지 이름] == 1.0.0
 - pip install --upgrade [패키지 이름]
 - pip list

Python 설치 - 가상환경

- virtualenv 패키지 설치
 - pip install virtualenv



```
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

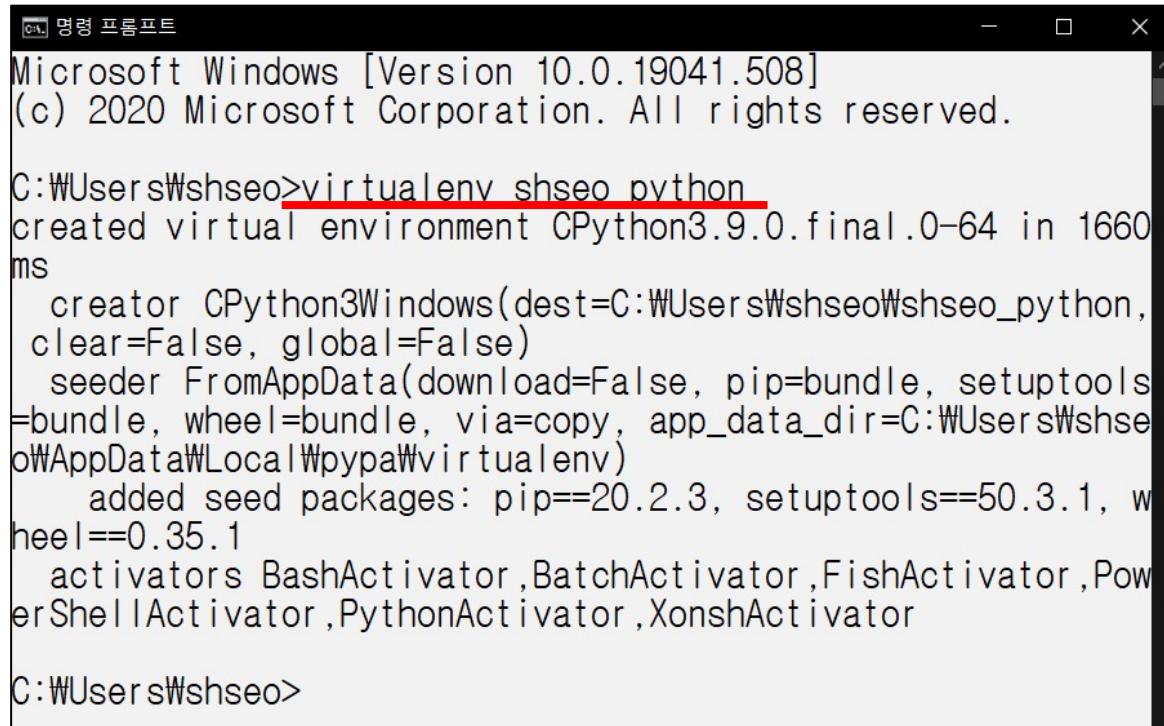
C:\Users\shseo>pip install virtualenv
Collecting virtualenv
  Using cached virtualenv-20.0.35-py2.py3-none-any.whl (4.9 MB)
Requirement already satisfied: distlib<1,>=0.3.1 in c:\Users\shseo\AppData\local\Programs\python\python39\lib\site-packages (from virtualenv) (0.3.1)
Requirement already satisfied: six<2,>=1.9.0 in c:\Users\shseo\AppData\Roaming\python\python39\site-packages (from virtualenv) (1.15.0)
Requirement already satisfied: filelock<4,>=3.0.0 in c:\Users\shseo\AppData\local\Programs\python\python39\lib\site-packages (from virtualenv) (3.0.12)
Requirement already satisfied: appdirs<2,>=1.4.3 in c:\Users\shseo\AppData\local\Programs\python\python39\lib\site-packages (from virtualenv) (1.4.4)
Installing collected packages: virtualenv
Successfully installed virtualenv-20.0.35

C:\Users\shseo>
```

Python 설치 - 가상환경

- 가상환경 생성

- virtualenv [env_name]
- 현재 위치인 “C:\Users\shseo”에 “shseo_python” 생성



```
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\shseo>virtualenv shseo_python
created virtual environment CPython3.9.0.final.0-64 in 1660
ms
    creator CPython3Windows(dest=C:\Users\shseo\shseo_python,
    clear=False, global=False)
    seeder FromAppData(download=False, pip=bundle, setuptools=
bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\shse
o\AppData\Local\pypa\virtualenv)
    added seed packages: pip==20.2.3, setuptools==50.3.1, w
heel==0.35.1
    activators BashActivator,BatchActivator,FishActivator,Pow
erShellActivator,PythonActivator,XonshActivator

C:\Users\shseo>
```

Python 설치 - 가상환경

- 가상환경 생성
 - .\[env_name]\Scripts\ctivate
 - .\shseo_python\Scripts\activate

The screenshot shows a Windows command prompt window titled "명령 프롬프트 - python". The window displays the following text:

```
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\shseo>.\shseo_python\Scripts\activate
(shseo_python) C:\Users\shseo>python -V
Python 3.9.0

(shseo_python) C:\Users\shseo>python
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct  5 2020, 15:34:40) [
MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

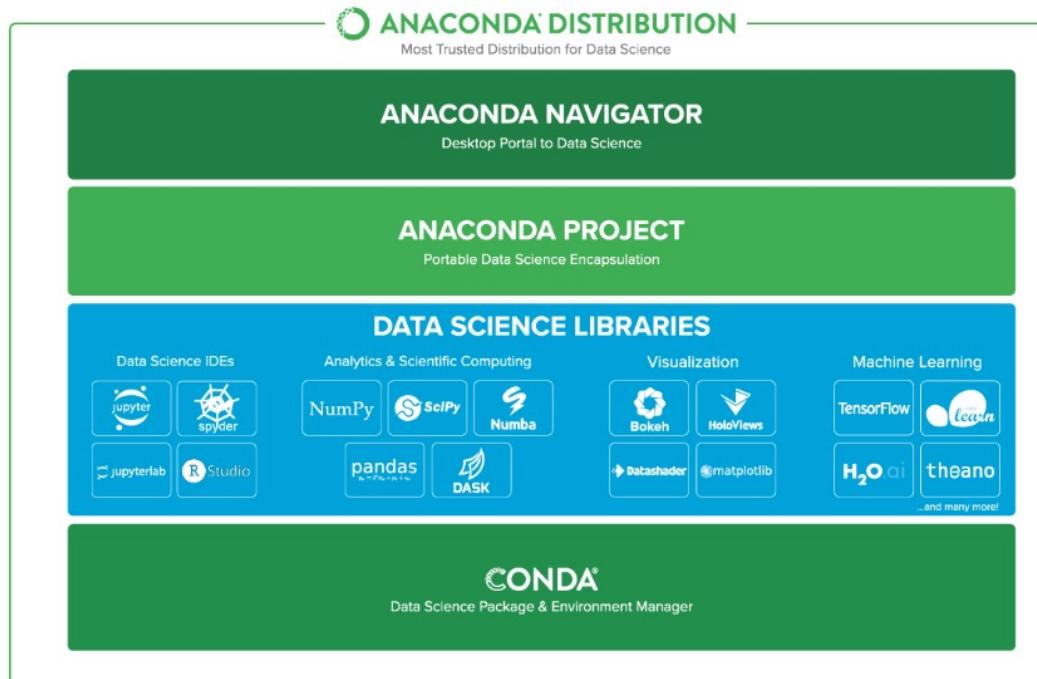
>>> print("hello, world!")
hello, world!
>>>
```

The command `.\shseo_python\Scripts\activate` is highlighted in red. The command `print("hello, world!")` is also highlighted in red.

Python 설치 - anaconda

- 아나콘다

- 아나콘다(Anaconda)는 패키지 관리와 디플로이를 단순화 할 목적으로 과학 계산(데이터 과학, 기계 학습 애플리케이션, 대규모 데이터 처리, 예측 분석 등)을 위해 파이썬과 R 프로그래밍 언어의 자유-오픈 소스 배포판임



Python 설치 - anaconda

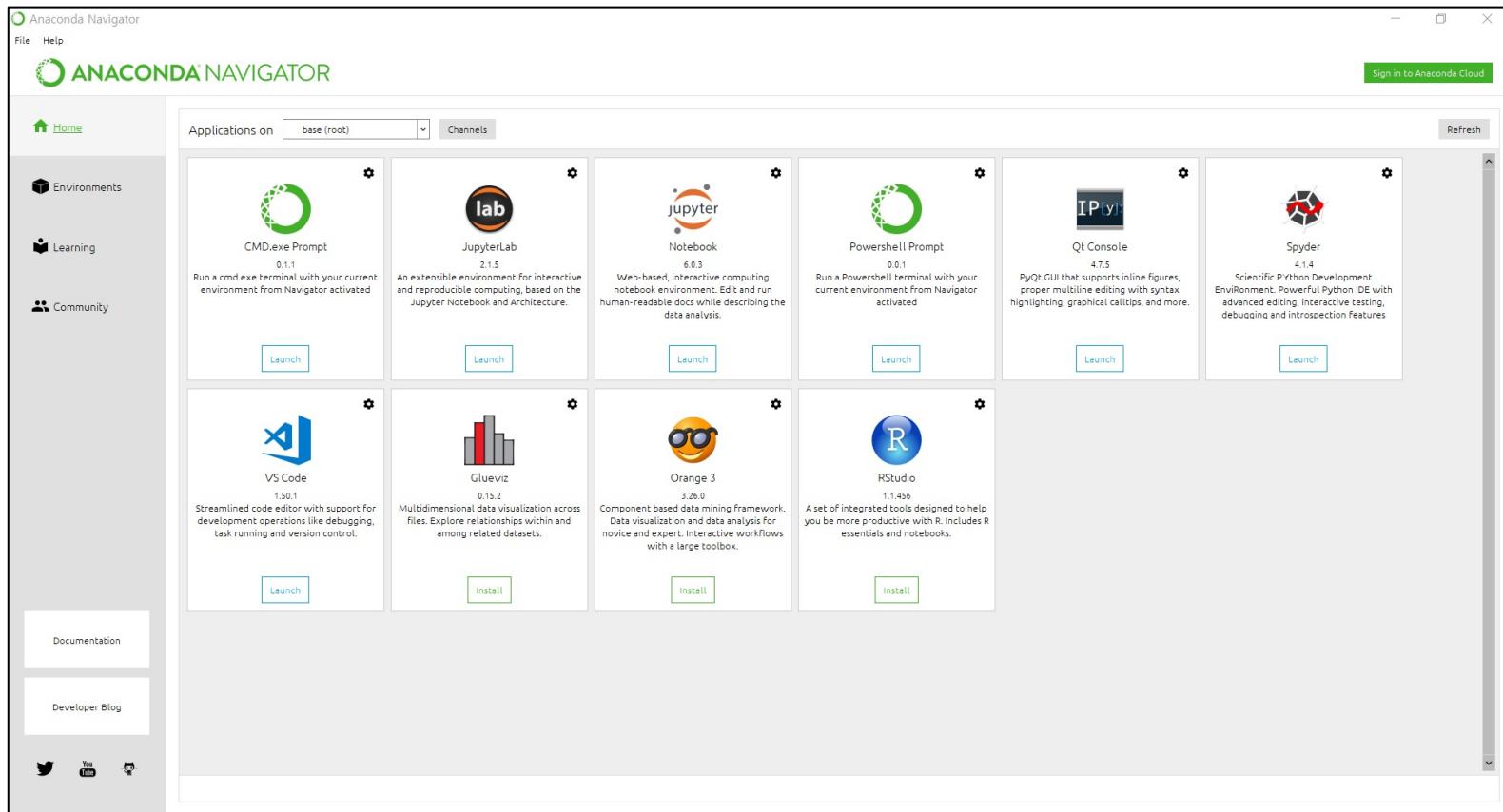
- 아나콘다 설치파일 다운로드
 - <https://www.anaconda.com/products/individual>

Anaconda Installers

Windows	MacOS	Linux
Python 3.8 64-Bit Graphical Installer (466 MB) 32-Bit Graphical Installer (397 MB)	Python 3.8 64-Bit Graphical Installer (462 MB) 64-Bit Command Line Installer (454 MB)	Python 3.8 64-Bit (x86) Installer (550 MB) 64-Bit (Power8 and Power9) Installer (290 MB)

Python 실행 - anaconda

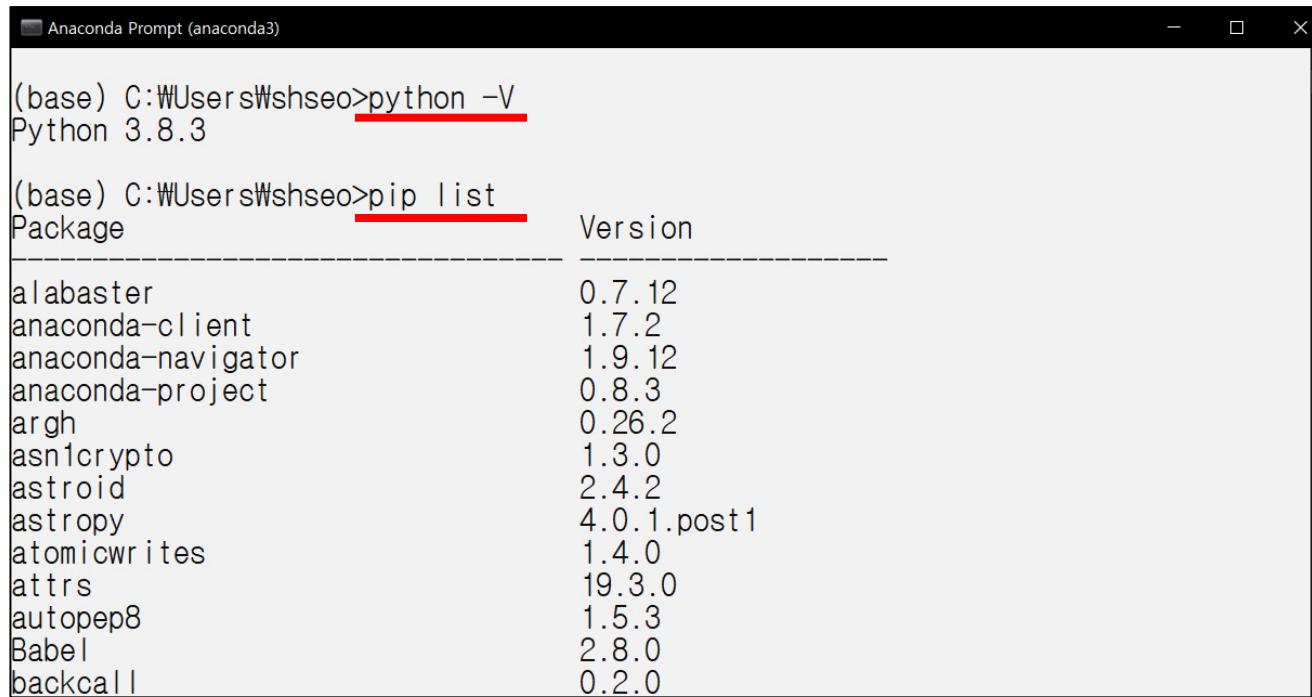
- Anaconda navigator
 - Windows key → anaconda navigator 실행



Python 실행 - anaconda

- Anaconda prompt

- Windows key → anaconda prompt 실행
- python -V
- pip list



The screenshot shows a Windows-style window titled "Anaconda Prompt (anaconda3)". Inside, the terminal output is displayed:

```
(base) C:\Users\shseo>python -V
Python 3.8.3

(base) C:\Users\shseo>pip list
Package           Version
alabaster        0.7.12
anaconda-client  1.7.2
anaconda-navigator 1.9.12
anaconda-project 0.8.3
argh              0.26.2
asn1crypto        1.3.0
astroid            2.4.2
astropy            4.0.1.post1
atomicwrites      1.4.0
attrs              19.3.0
autopep8          1.5.3
Babel              2.8.0
backcall           0.2.0
```

The command "python -V" is highlighted in red. The command "pip list" is also highlighted in red. The table output shows the installed packages and their versions.

Python IDE 설치 - jupyter

- Jupyter notebook

- 프로젝트 주피터는 "오픈 소스 소프트웨어, 개방형 표준, 그리고 여러 개의 프로그래밍 언어에 걸쳐 인터랙티브 컴퓨팅을 위한 서비스 개발"을 위해 설립된 비영리 단체로서 다양한 프로그래밍 언어에 걸쳐 인터랙티브 데이터 과학과 사이언티픽 컴퓨팅을 지원



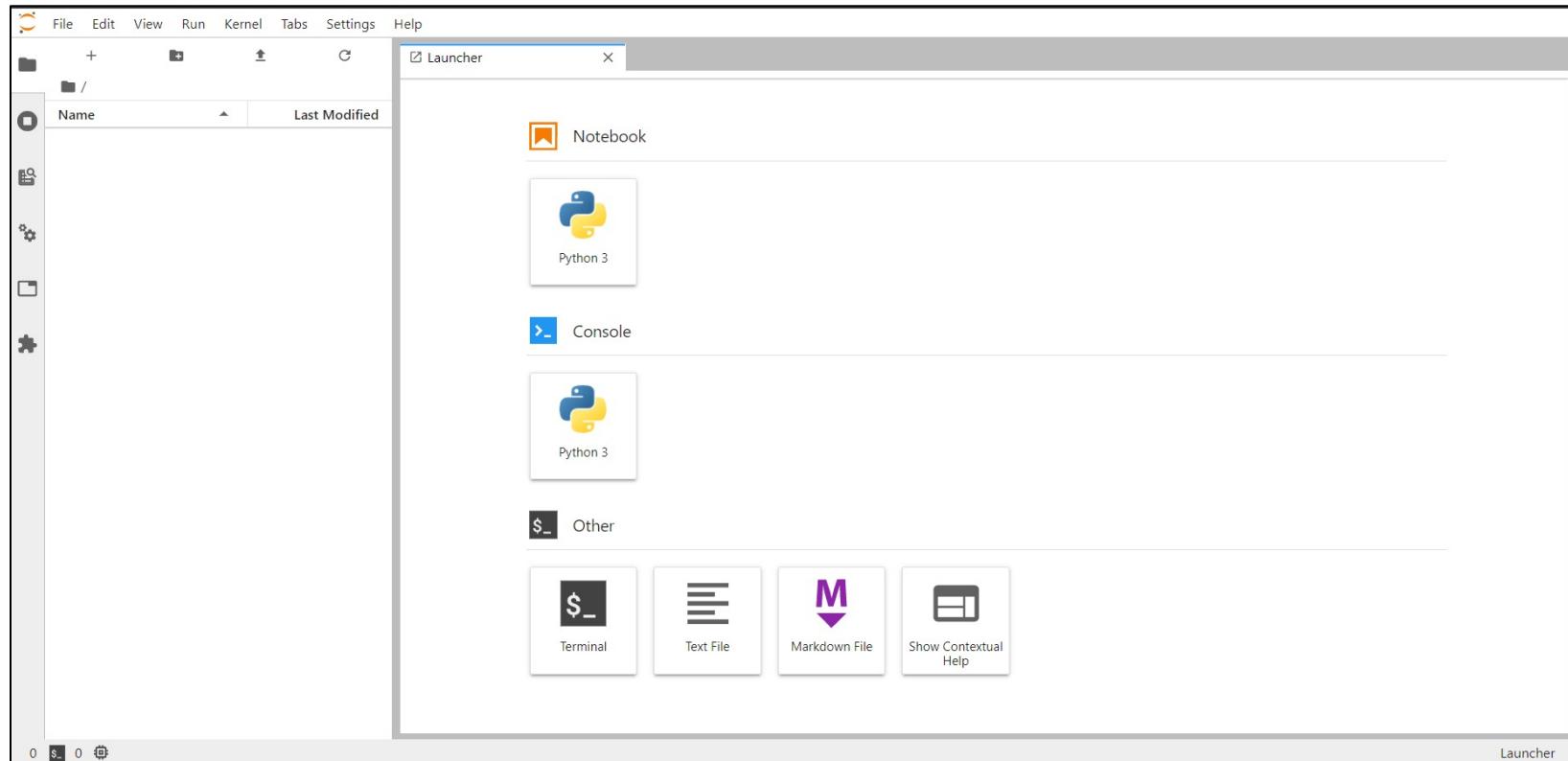
Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

Python IDE 설치 - jupyter

- jupyter lab 실행
 - LG_Electroins 폴더생성 → mkdir LG_Electronics
 - LG_Electroins 폴더로 이동 → cd LG_Electronics
 - jupyter lab 실행 → jupyter lab

Python IDE 실행 - jupyter

- jupyter lab 실행
 - Browser를 통한 간편한 개발환경 사용 가능



Python IDE 실행 - jupyter

- jupyter lab 실행

- File → new file → hello_world.ipynb
- print("hello, world!")

The screenshot shows the Jupyter Lab interface. On the left, there's a sidebar with tabs for Files, Running, Commands, Cell Tools, Tabs, and Cell Tools. The Files tab is active, showing a list of files: '1-1_hello_world.ipynb' (which is currently selected and highlighted in blue) and another unnamed file. The Running tab shows no other processes. The main area is a large code editor window titled '1-1_hello_world'. It contains a single code cell with the Python command `print("hello, world!")`. When run, the cell outputs the text `hello, world!`. The top right corner of the interface indicates the kernel is set to 'Python 3'.

Python IDE 실행 - jupyter

- jupyter lab 단축키
 - 셀 추가, 셀 삭제, 복사/잘라내기/붙여넣기

```
In [ ]: # Edit mode : Enter  
# Command mode : ESC
```

```
In [ ]: # 셀 추가 (Command mode에서)  
# 현재 셀 위로 : a  
# 현재 셀 아래로 : b
```

```
In [ ]: # 셀 삭제 (Command mode에서) : dd
```

```
In [ ]: # 선택한 셀을  
# 복사하기 : c  
# 잘라내기 : x
```

```
In [ ]: # 셀 붙여넣기  
# 선택한 셀 위로 : Shift + v  
# 선택한 셀 아래로 : v
```

```
In [ ]: # 셀에 라인 추가하기 (Command mode에서) : l
```

Python IDE 실행 - jupyter

- jupyter lab 단축키
 - 셀 변경, 실행, 실행 후 아래 셀 선택, 코드 자동완성

```
In [ ]: # 셀 변경하기 (Command mode에서)
# code : y
# markdown : m

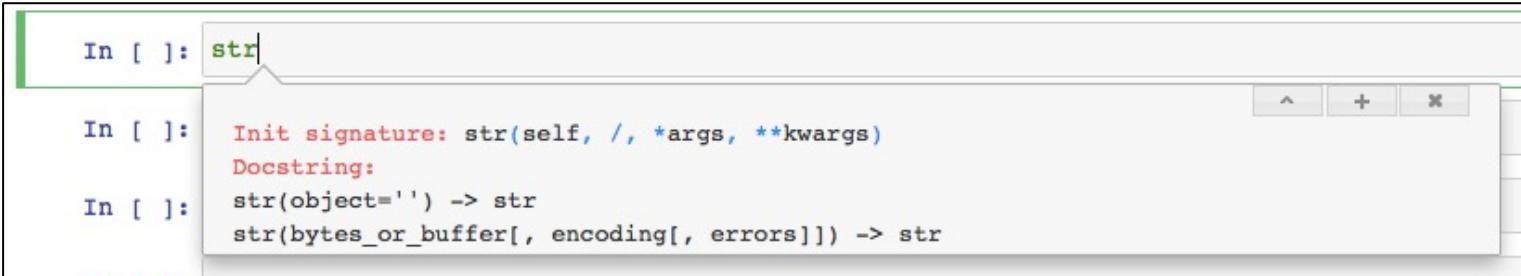
In [ ]: # 해당 셀을 실행하기 : Ctrl + Enter

In [ ]: # 해당 셀을 실행하고 아래에 셀 선택 : Shift + Enter

In [ ]: # 코드 자동 완성 : Tab
str.| capitalize
In [1]: str| casefold
center
count
encode
endswith
expandtabs
find
format
format_map
```

Python IDE 실행 - jupyter

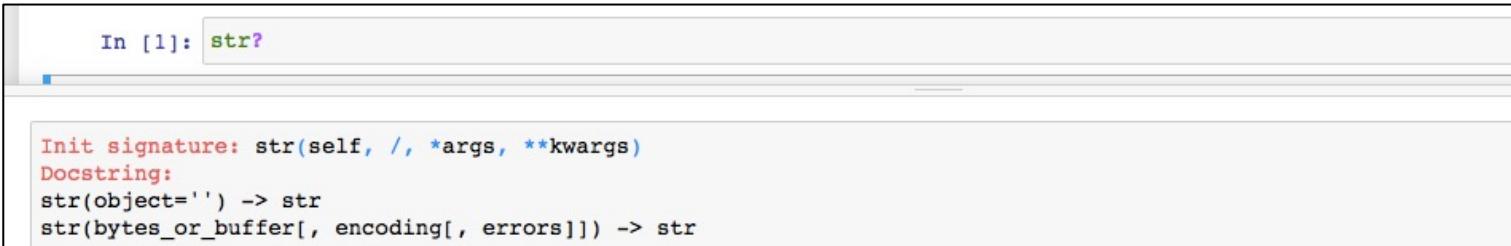
- jupyter lab 단축키
 - 함수 및 변수 설명: shift + tab



In []: str

In []: Init signature: str(self, /, *args, **kwargs)
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

- 함수 및 변수 설명: ?

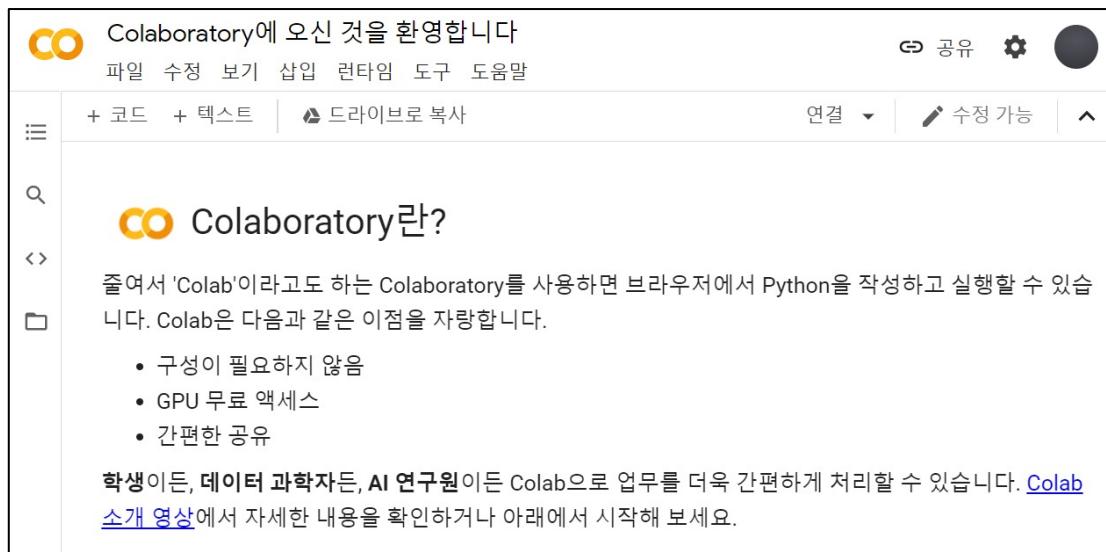


In [1]: str?

Init signature: str(self, /, *args, **kwargs)
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

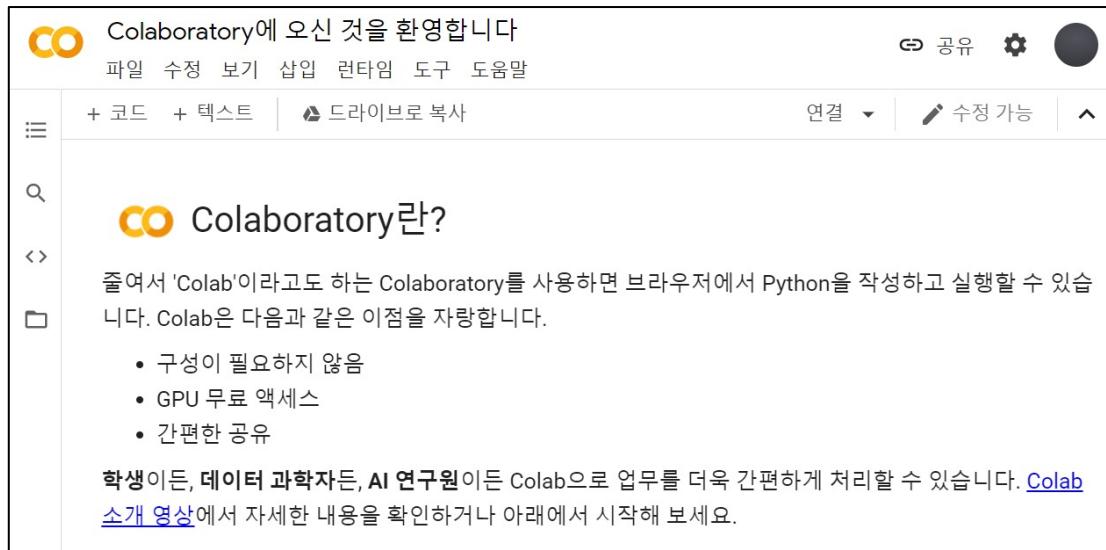
Google Colab

- 코랩(colab): colab.research.google.com
 - 데이터 과학자, AI 연구원 및 학생 등을 위해서 공개된 클라우드 기반 **Jupyter Notebook 개발환경**
 - 대표적인 딥러닝 프레임워크인 **Tensorflow**, Pytorch 설치 및 GPU 활용이 간소하여 간단한 딥러닝 실험을 수행하기에 적합
 - 데이터 및 노트북 파일은 **구글 드라이브와 연동**되어 사용되기 때문에 구글 계정이 있는 누구나 활용가능한 장점이 있음



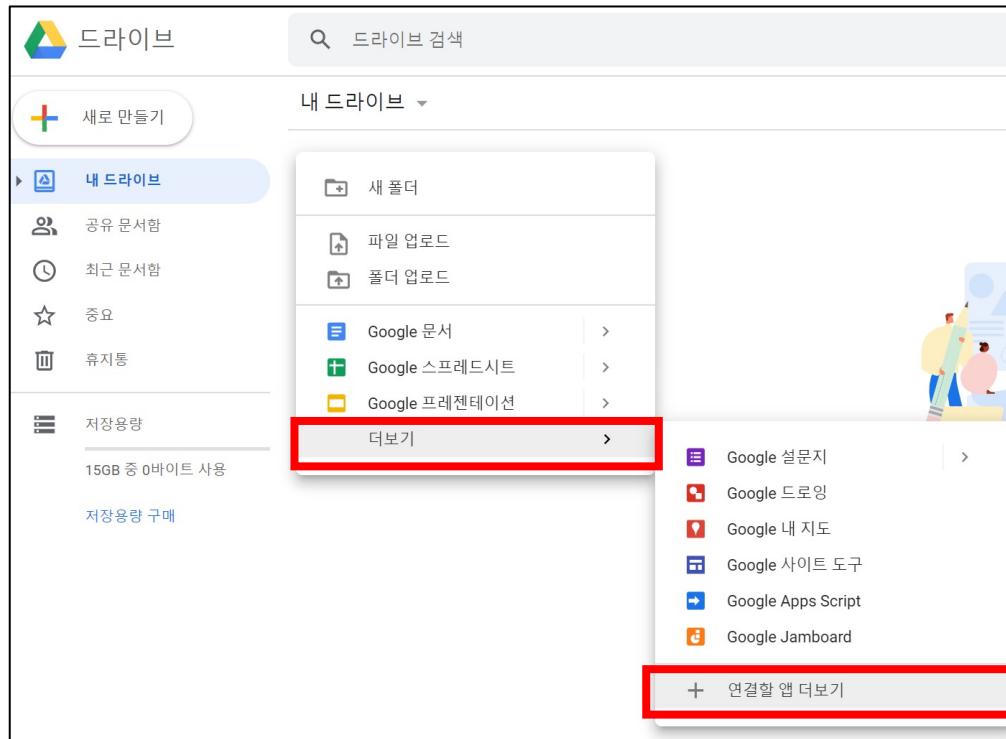
Google Colab

- 코랩(colab): colab.research.google.com
 - 데이터 과학자, AI 연구원 및 학생 등을 위해서 공개된 클라우드 기반 **Jupyter Notebook 개발환경**
 - 대표적인 딥러닝 프레임워크인 **Tensorflow**, Pytorch 설치 및 GPU 활용이 간소하여 간단한 딥러닝 실험을 수행하기에 적합
 - 데이터 및 노트북 파일은 **구글 드라이브와 연동**되어 사용되기 때문에 구글 계정이 있는 누구나 활용가능한 장점이 있음



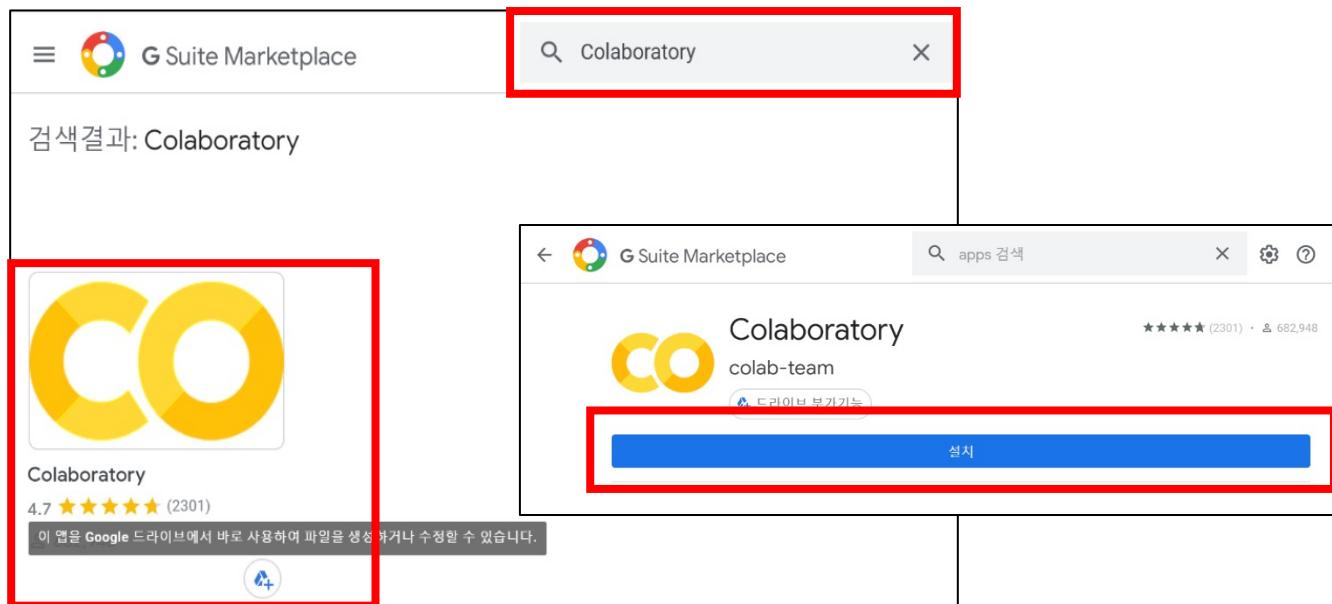
Google Colab

- 코랩(colab): colab.research.google.com
 - 구글 드라이브(drive.google.com)와 연동
 - 드라이브 화면 위의 임의의 폴더에서 [더보기] → [연결할 앱 더보기] 클릭



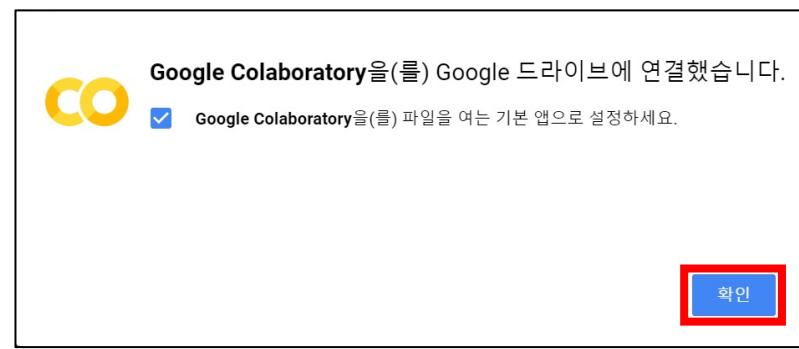
Google Colab

- 코랩(colab): colab.research.google.com
 - 구글 드라이브(drive.google.com)와 연동
 - [Colaboratory] 검색 후 검색된 [Colaboratory] 선택
→ [설치] 클릭



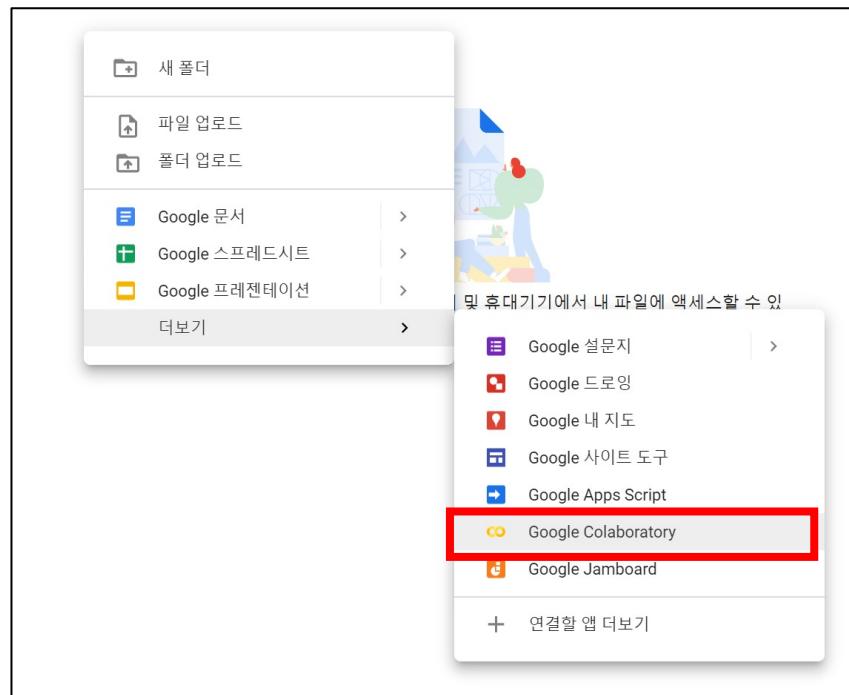
Google Colab

- 코랩(colab): colab.research.google.com
 - 구글 드라이브(drive.google.com)와 연동
 - [계속] 클릭
 - [체크] 후 확인을 클릭하여 Google Colaboratory를 기본 앱으로 설정



Google Colab

- 코랩(colab): colab.research.google.com
 - 구글 드라이브(drive.google.com)와 연동
 - 구글 드라이브에서 마우스 우클릭 → [더보기] → [Google Colaboratory]를 통해 원하는 구글 드라이브 경로에서 새 노트를 생성 가능



Google Colab

- 코랩(colab): colab.research.google.com
 - 구글 드라이브(drive.google.com)와 연동
 - 자신의 구글 드라이브에 실습 폴더 생성
 - ipynb 파일을 생성하고 hello world 출력



The screenshot shows the Google Colab interface. At the top, there's a header with the CO logo, the file name "hello_world.ipynb", a star icon, and various navigation links like 파일, 수정, 보기, 삽입, 런타임, 도구, and 도움말. To the right of the header are buttons for 댓글 (Comments), 공유 (Share), settings, and profile. Below the header, there are two tabs: "+ 코드" (Code) and "+ 텍스트" (Text). The main area shows a code cell with the command `[1] 1 print("hello, world!")`. The output cell below it displays the result `hello, world!`. On the far left, there are icons for search, refresh, and other navigation functions.

Google Colab

- 코랩(colab): colab.research.google.com

- 주요 단축키 및 명령어

- 운영체제 확인: !cat /etc/issue.net
- 파이썬 버전체크: !python --version
- CPU 사양: !head /proc/cpuinfo
- GPU 정보: !nvidia-smi
- 메모리 사양: !head -n 3 /proc/meminfo
- 디스크 사양: !df -h
- Command Pallete – Ctrl+Shift+P
- Show keyboard shortcuts – Ctrl+M+H

Actions	Colab	Jupyter
show keyboard shortcuts	Ctrl/Cmd M H	H
Insert code cell above	Ctrl/Cmd M A	A
Insert code cell below	Ctrl/Cmd M B	B
Delete cell/selection	Ctrl/Cmd M D	DD
Interrupt execution	Ctrl/Cmd M I	II
Convert to code cell	Ctrl/Cmd M Y	Y
Convert to text cell	Ctrl/Cmd M M	M
Split at cursor	Ctrl/Cmd M -	Ctrl Shift -

2. Data Type & Variable

파이썬 자료형

- 숫자자료형
 - int, float, complex
 - 사칙연산(operations)
 - 변수(variable)
- 불자료형
 - 관계연산
 - bool (True or False)
- 군집자료형
 - str, list, tuple, dictionary, set

클래스	설명	불변 객체
bool	부울	o
int	정수	o
float	실수	o
list	리스트	x
tuple	리스트와 튜플의 차이는 불변 여부이며 이외에는 거의 동일하다. 튜플은 불변이므로 생성할 때 설정한 값은 변경할 수 없다.	o
str	문자	o
set	중복된 값을 갖지 않는 집합 자료형	x
dict	딕셔너리	x

숫자 자료형

- 정수(**integer**)
 - 양의 정수, 음의 정수, 숫자 0을 포함하는 자료형

```
print(123) #양의 정수  
print(-321) #음의 정수  
print(0) #0
```

```
123  
-321  
0
```

숫자 자료형

- 정수의 표현방식

- 10진수 123의 표현 예시

- 16진수 : 7B
- 10진수 : 123
- 8진수 : 173
- 2진수 : 1111011

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
영	일	이	삼	사	오	육	칠	팔	구	십	십일	십이	십삼	십사	십오

- 16진수 12A →

숫자 자료형

- 실수(**float**)
 - 실수형(Floating-point)은 소수점이 포함된 숫자를 의미

```
print(3.14)
print(314e-2) # 314 * 10-2
```

3.14

3.14

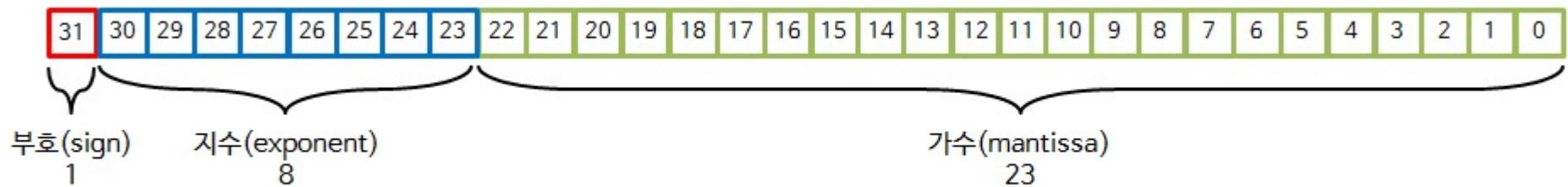
- 실수표현 방식

- 정수부.소수부 → 263.3 → 263 | 0.3
→ 100000111 | 0.01001100110011... (무한반복)

정수										소수				
x2	x2	x2	x2	÷2	÷2	÷2	÷2							
16	8	4	2	1	0.5	0.25	0.125	0.0625						
2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}						

숫자 자료형

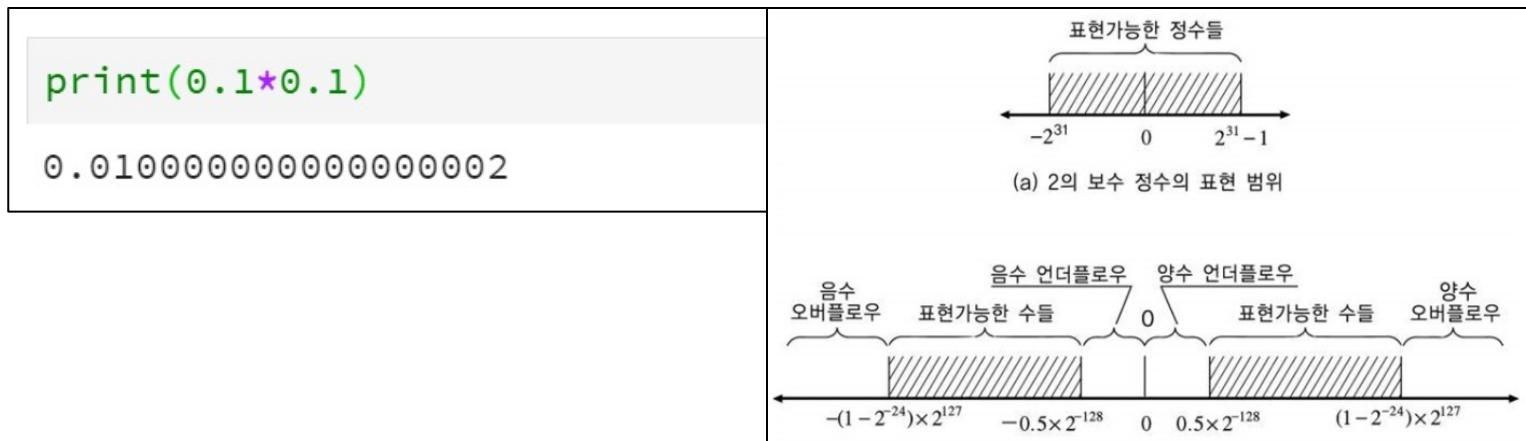
- 부동 소수점(floating point)
 - Single precision
 - 32 bit (4byte)
 - 부호(1bit) + 지수(8bit) + 가수(23bit)
 - Double precision
 - 64 bit (8byte)
 - 부호(1bit) + 지수(11bit) + 가수(52bit)



숫자 자료형

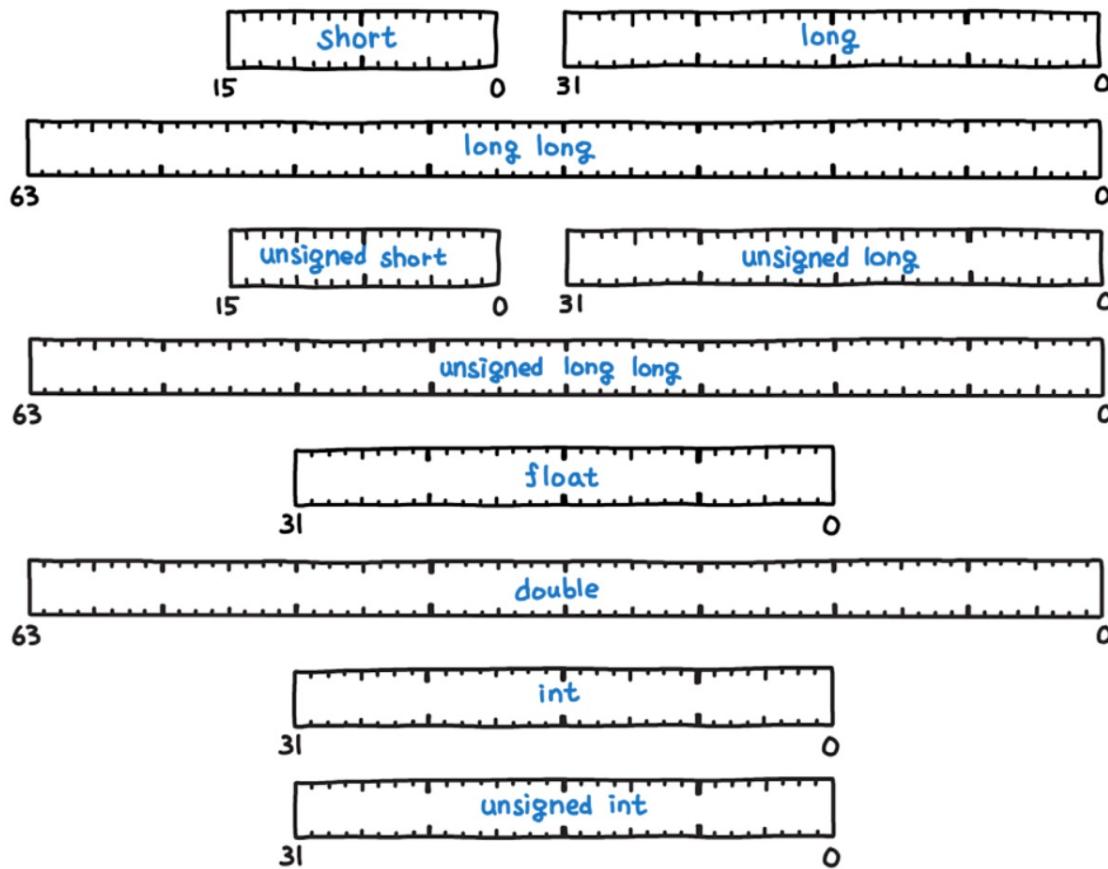
- 부동 소수점(floating point)

- 지수(exponent)는 정밀도 결정
- 가수(mantissa)는 표현 가능한 수의 범위를 결정
 - NaN(Not a Number): 0으로 나누거나 음수에 대한 제곱근 등
 - overflow/underflow: 주어진 정밀도 형식으로 표현할 수 있는 범위를 넘어선 경우

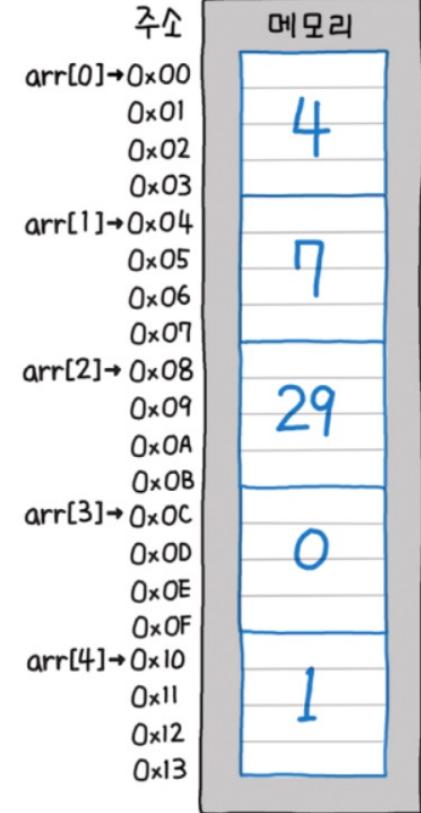


숫자 자료형

- C언어 원시타입(Primitive Type) 예시



int arr[5] = { 4, 7, 29, 0, 1 }



숫자 자료형

- 복소수(**complex**)
 - 허수(imaginary number)의 약자로는 j를 사용하여 표현
 - 실수부: real
 - 허수부: imag
 - 캔леж복소수: conjugate()

```
print(1+2j)
print((1+2j).real)
print((1+2j).imag)
print((1+2j).conjugate())
```

```
(1+2j)
1.0
2.0
(1-2j)
```

숫자 자료형

- 숫자 자료형 확인
 - type()

```
print(type(3))
print(type(3.14))
print(type(1+2j))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

- Python의 모든 자료형은 class로 구성되어 있음

사칙연산

- 산술연산자

- $x + y$	덧셈		$x - y$	뺄셈
- $x * y$	곱셈		x / y	나눗셈
- $x ** y$	제곱			
- $x // y$	몫(Floor 나눗셈)		$x \% y$	나머지(모듈러)

```
print(12+3)
print(12-3)
print(12*3)
print(12/3)
print(12**3)
print(12%3)
print(12%3)
```

```
15
9
36
4.0
1728
0
0
```

사칙연산

- 연산의 우선순위

- 사칙연산의 연산자 우선수위와 같음
- 괄호()가 있다면 가상 우선 처리

```
print(12+3*4)
print((12+3)*4)
```

24
60

- Mini Quiz!

- $-(2^{**}4) \rightarrow$
- $(-2)^{**}4 \rightarrow$
- $-2^{**}4 \rightarrow$

Practice 1

- Q1
 - 10의 제곱 출력
- Q2
 - 2^*5+3 & $2^*(5+3)$ 출력

1번

10의 제곱을 출력해보자

정답을 적어주세요

100

정답을 적어주세요

100

2번

$2 \times 5 + 3$ 과 $2 \times (5 + 3)$ 을 각각 화면에 출력해보자

정답을 적어주세요

13

정답을 적어주세요

16

변수 (variable)

- **변수**
 - 데이터를 담는 메모리 공간으로서 숫자, 문자, 리스트 등의 다양한 데이터를 저장 가능
 - 적절한 변수명을 사용함으로써 보다 직관적으로 이해할 수 있는 코드 작성 가능
- **할당 연산자(assignment): =**
 - 변수 = 연산/조건/함수
 - 할당 연산자의 우측 부분에 대한 연산을 수행한 뒤, 그 결과를 좌측의 변수에 할당

변수 (variable)

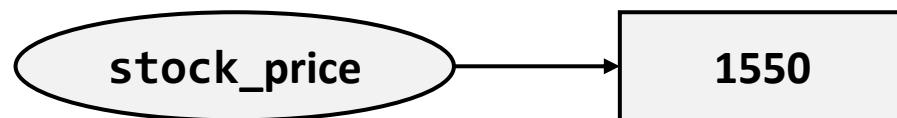
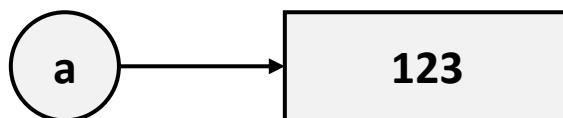
- **변수**

- 변수는 메모리 공간에 있는 특정 값(value)을 활용하는 방식으로 동작

```
a=123
print(a)
print(id(a))

stock_price = 1550
print(stock_price)
print(id(stock_price))
```

```
123
1409186688
1550
1290549633168
```



변수 (variable)

- **변수**

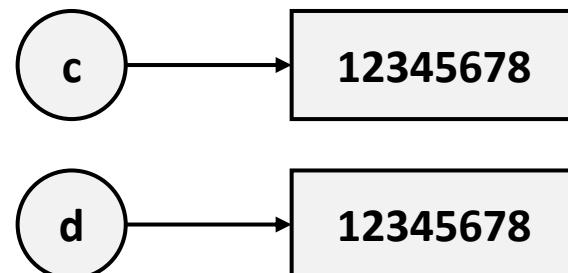
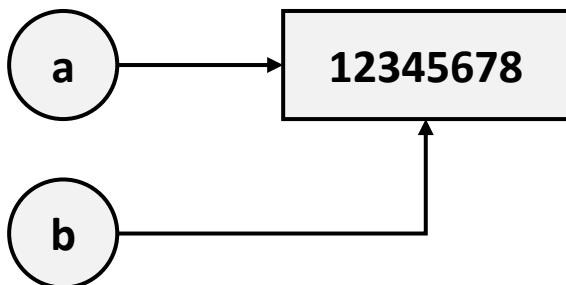
- 서로 다른 변수명을 갖더라도 같은 메모리 공간을 가리킬 수도 있고, 같은 값을 가질지라도 서로 다른 메모리 공간을 가리키는 경우가 있음

```
a=b=12345678  
print(a)  
print(id(a))  
print(b)  
print(id(b))
```

```
12345678  
1290548636432  
12345678  
1290548636432
```

```
c=12345678  
d=12345678  
print(c)  
print(id(c))  
print(d)  
print(id(d))
```

```
12345678  
1290548637008  
12345678  
1290548637264
```



변수 (variable)

- **변수**

- 산술 연산자와 할당 연산자를 함께 사용 가능
→ 코드의 간결성 향상

```
variable = 2
variable
```

2

```
variable += 1
variable
```

3

```
variable = variable + 1
variable
```

4

```
variable -= 5
variable
```

-1

```
variable *= -1
variable
```

1

```
variable /= 5
variable
```

0.2

변수 (variable)

- **변수**

- 숫자 간 연산 뿐 아니라 변수 간 연산 가능

```
a = 5  
b = 7  
c = a + b  
print(c)
```

12

- 대소문자 구분 유의

```
c = A + B
```

```
-----  
-----  
NameError
```

```
t call last)
```

```
<ipython-input-23-b698ae2737d5> in <module>()
```

```
----> 1 c = A + B
```

```
Traceback (most recen
```

```
NameError: name 'A' is not defined
```

변수 (variable)

- 파이썬 예약어

- 예약어는 python에서 미리 정의한 의미있는 단어로서 상수, 또는 변수 등 별도의 식별자로 사용할 수 없음
- <주요 예약어>

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Practice 2

- Q1(변수 할당 및 산술 연산)
 - 파일 수 카운팅
- Q2 (변수 할당 및 산술 연산)
 - 평균 점수 계산

1번

사과가 5개 오렌지가 3개 있을 때 총 과일의 갯수를 구해보자
사과는 apple이라는 변수, 오렌지는 orange라는 변수에 할당 한 후, 총 과일의
갯수를 total이라는 변수에 저장해보자

```
# 정답을 적어주세요
```

```
# 아래 주석을 풀고 실행시켜서 원하는 값이 나왔는지 확인하세요  
# total
```

2번

국어는 100점, 영어는 88점, 수학은 94점 일 때, 평균을 구하려고 한다.
각각의 점수를 kor, eng, math라는 변수에 저장한 후 평균을 구해 avg라는
변수에 할당해보자.

```
# 정답을 적어주세요
```

```
# 아래 주석을 풀고 실행시켜서 원하는 값이 나왔는지 확인하세요  
# avg
```

불리안(Boolean)

- **불(bool)**

- 참과 거짓을 나타내는 자료형
 - 참: True
 - 거짓: False

```
True
True
type(False)
bool
```

- 비교 연산자와 조건문의 결과는 불자료형이 사용됨

불리안(Boolean)

- 비교연산자

- $x == y$ 값이 동일하다 | $x != y$ 값이 동일하지 않다
- $x > y$ x가 y보다 크다 | $x < y$ x가 y보다 작다
- $x >= y$ x가 y보다 크거나 동일하다
- $x <= y$ x가 y보다 작거나 동일하다

```
x=10
y=20

print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

```
False
True
False
True
False
True
```

불리안(Boolean)

- 논리연산자

- x and y AND(논리곱), 양쪽 모두 참일 때 참
- X or y OR(논리합), 양쪽 중 한쪽만 참이라도 참
- not x NOT(논리부정), 참가 거짓을 뒤집음

```
x=y=True  
print(x and y)  
print(x or y)  
print(not x)
```

True
True
False

```
x=False  
y=True  
print(x and y)  
print(x or y)  
print(not x)
```

False
True
True

문자열 (string)

- 문자열 (**string**)

- ‘ ’(작은 따옴표) 또는 “ ”(큰 따옴표)를 이용하여 문자 자료형임을 표현

```
string = 'string example'  
print(string)  
type(string)
```

```
string example  
str
```

문자열 (string)

- 문자열 (**string**)

- ‘ ’(작은 따옴표) 또는 “ ”(큰 따옴표)를 이용하여 문자 자료형임을 표현

```
apostrophe = 'apostrophe example: "example" .'  
print(apostrophe)
```

```
apostrophe example: "example" .
```

```
quotation = "quotation example: 'example' ."  
print(quotation)
```

```
quotation example: 'example' .
```

```
escape = "escape \" \\ \"(back slash, won) ."  
print(escape)
```

```
escape " \ "(back slash, won) .
```

문자열 (string)

- 형변환(casting)
 - 데이터를 다른 자료형으로 변환 가능

```
print(float(3))      #실수형으로 바꿈
print(int(3.0))      #정수형으로 바꿈
print(str(3))        #문자열로 바꿈
print(hex(12))       #16진수로 바꿈
print(oct(10))       #8진수로 바꿈
print(bin(10))       #2진수로 바꿈
```

```
3.0
3
3
0xc
0o12
0b1010
```

문자열 (string)

- 형변환(casting)

```
g = 3
h = '5'
print(type(g))
print(type(h))
```

```
i = str(g)
print(i)
print(type(i))
j = int(h)
print(j)
print(type(j))
```

```
<class 'int'>
<class 'str'>
3
<class 'str'>
5
<class 'int'>
```

문자열 (string)

- String 연산
 - 덧셈: 두 문자열의 연결

```
str_1 = 'string'  
space = ' '  
str_2 = 'operation'  
result = str_1 + space + str_2  
print(result)
```

```
string operation
```

```
str_3 = ': plus'  
result += str_3  
print(result)
```

```
string operation: plus
```

문자열 (string)

- String 연산
 - 곱셈: 문자열의 반복

```
str_4 = 'multiplication '
print(str_4 * 5)
```

```
multiplication multiplication multiplication multiplication mul
tiplication
```

```
str_5 = 'repeat! '
print((str_4 + str_5)*2)
```

```
multiplication repeat! multiplication repeat!
```

문자열 (string)

- String 연산
 - 스트링 간 연산만 가능

```
int_1 = 8
str_6 = 'string'
error = str_6 + int_1
```

```
-----
-----  
TypeError
t call last)
<ipython-input-14-f940df6a826c> in <module>()
    1 int_1 = 8
    2 str_6 = 'string'
----> 3 error = str_6 + int_1
```

Traceback (most recent)

```
TypeError: must be str, not int
```

문자열 (string)

- 인덱싱(indexing)

- 순서가 있는 변수의 특정 위치에 접근하는 방식
- 데이터[위치] # 인덱스는 0부터 시작
- len() 길이를 측정하는 함수

```
str_5 = 'indexing example string'  
len(str_5)
```

23

```
str_5[2]
```

'd'

문자열 (string)

- **슬라이싱(slicing)**

- 순서가 있는 변수의 일정 부분에 접근하는 방식
- **데이터[시작위치:끝위치:간격]** # 간격이 1이면 생략 가능
- 슬라이싱은 시작위치부터 끝위치-1 까지의 데이터를 접근

```
str_6 = 'slicing example string'  
str_6[8:14]
```

```
'exampl'
```

```
str_6[:3] #시작위치가 없으면 처음부터
```

```
'sli'
```

```
str_6[5:] # 끝 위치가 없으면 끝까지
```

```
'ng example string'
```

```
str_6[:] # 시작위치, 끝위치가 없으면 전체
```

```
'slicing example string'
```

문자열 (string)

- 인덱싱(indexing)

- 마이너스 인덱싱 → 처음위치를 0으로 두고 역으로 접근

```
print(str_6[-1])
```

g

```
print(str_6[-4:])
```

ring

```
print(str_6[:-2])
```

slicing example stri

```
print(str_6[::-1])
```

gnirts elpmaxe gnicils

문자열 (string)

- 메소드(method)
 - 파이썬의 객체가 가지고 있는 고유의 함수
 - 온점 .을 이용하여 사용
 - 대소문자 변환: lower, upper
 - 문자 개수 카운팅: count
 - 문자의 위치 찾기: find, index
 - 문자열 바꾸기: replace
 - 문자열 나누기: split
 - 문자열 삽입하기: join
 - 문자열 활용하기: split
 - 기타 메소드

문자열 (string)

- 대소문자 변환: lower, upper

```
str_7 = 'Alphabet'  
print(str_7.lower())
```

alphabet

```
print(str_7.upper())
```

ALPHABET

문자열 (string)

- 문자 개수 카운팅: count

```
str_7 = 'Alphabet'  
str_7.count('a')
```

1

- 대소문자 구분없이 문자의 개수를 세고자 한다면?

문자열 (string)

- 문자의 위치 찾기: `find`, `index`
 - 해당하는 문자열의 위치(인덱스)를 반환

```
str_7 = 'Alphabet'  
str_7.find('t')
```

7

```
str_7.index('t')
```

7

문자열 (string)

- 문자의 위치 찾기: find, index
 - find(): query가 문자열에 없으면 -1을 반환
 - index(): query가 문자열에 없으면 error 발생

```
str_7 = 'Alphabet'  
str_7.find('z')
```

```
-1
```

```
str_7.index('z')
```

```
-----  
-----  
ValueError
```

```
    at call last)
```

```
<ipython-input-34-66403266caad> in <module>()
```

```
----> 1 str_7.index('z')
```

```
Traceback (most recent
```

```
ValueError: substring not found
```

문자열 (string)

- 문자열 바꾸기: `replace`
 - 특정 문자열을 새로운 문자열로 대체

```
str_8 = 'Life is C between B and D'  
str_8.replace('C', 'choice')
```

```
'Life is choice between B and D'
```

- 문자열 나누기: `split`
 - 특정 문자를 기준으로 문자열을 분리 → 리스트로 반환
 - 생략시 기준은 띄어쓰기

```
str_8.split(' ')
```

```
['Life', 'is', 'C', 'between', 'B', 'and', 'D']
```

문자열 (string)

- 문자열 삽입하기: join
 - 각 문자 사이에 특정 문자열 삽입

```
str_9 = 'abcd'  
'.'.join(str_9)
```

```
'a,b,c,d'
```

- 문자열 분할 후 합치기: split + join

```
str_10 = str_8.split()  
str_10
```

```
['Life', 'is', 'C', 'between', 'B', 'and', 'D']
```

```
' '.join(str_10)
```

```
'Life is C between B and D'
```

문자열 (string)

- 기타 string 메소드
 - dir(str)

```
print(dir(str_9))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
 '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
 '__init_subclass__', '__iter__', '__le__', '__len__',
 '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__', 'capitalize',
 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs',
 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha',
 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
 'isprintable', 'isspace', 'istitle', 'isupper', 'join',
 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace',
 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']
```

리스트(list)

- 리스트(list): []
 - 대괄호 []을 이용하여 리스트 선언

```
list_1 = [1,2,3]
list_2 = list()

print(type(list_1))
print(type(list_2))
```

```
<class 'list'>
<class 'list'>
```

리스트(list)

- **리스트(list)의 특징**

- 리스트는 모든 자료형을 담을 수 있음
- 리스트는 삽입, 수정, 삭제 등이 자유로움
- 리스트는 순서가 있는 자료형 → 인덱싱 슬라이싱 가능

```
list_3 = [1, 2, 'character', ['two_dimensional_array', 'available'], ('tuple', 'in_list')]
```

```
list_3[3]
```

```
['two_dimensional_array', 'available']
```

```
list_3[4][0]
```

```
'tuple'
```

리스트(list)

- 리스트(list)의 인덱싱

- 가장 바깥 괄호부터 접근
- 첫번째 위치부터 시작하여 인덱스 계산

```
list_4 = [[1,2,3,4],  
          [2,4,6,8],  
          [3,6,9,12],  
          [4,8,12,16],  
          [5,10,15,20]]
```

```
list_4
```

```
[[1, 2, 3, 4], [2, 4, 6, 8], [3, 6, 9, 12], [4, 8, 12, 16], [5,  
10, 15, 20]]
```

```
list_4[1]
```

```
[2, 4, 6, 8]
```

```
list_4[1][3]
```

```
8
```

리스트(list)

- 리스트(list)의 인덱싱
 - 가장 바깥 괄호부터 접근
 - 첫번째 위치부터 시작하여 인덱스 계산

```
list_5 = [[[1,2,3],[4],[5,[6,7,8,9]]],  
          [10,11,[12,13,[14]]]]
```

```
list_5[1][2][2]
```

```
[14]
```

리스트(list)

- 리스트(list)의 인덱싱
 - 복잡한 리스트 인덱싱, 슬라이싱 예시

```
list_4[1][:3]
```

```
[2, 4, 6]
```

```
list_4[:2][1]
```

```
[2, 4, 6, 8]
```

```
list_5[:2][0]
```

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

```
list_5[0][:2]
```

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

리스트(list)

- 리스트(list)의 연산

- 덧셈: 두 리스트를 연결하여 하나의 리스트로 만듦
- 곱셈: 리스트를 반복하여 연결하여 하나의 리스트로 만듦

```
list_6 = ['a', 'b', 'c']
```

```
list_7 = ['가', '나', '다']
```

```
list_6 + list_7
```

```
['a', 'b', 'c', '가', '나', '다']
```

```
list_6 * 3
```

```
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']
```

리스트(list)

- 리스트(list)의 메소드
 - 리스트는 삽입, 수정, 삭제 등이 자유로움
 - 삽입: append, insert
 - 수정(덮어쓰기): 인덱싱이나 슬라이싱 사용
 - 삭제: remove
 - 값 꺼내기: pop
 - 정렬: sort
 - 뒤집기: reverse

리스트(list)

- 리스트(list)의 메소드
 - 추가: append
 - 리스트의 가장 뒤에 요소 추가

```
list_8 = ['파이썬', 'C', 'java']
```

```
list_8.append('R')  
list_8
```

```
['파이썬', 'C', 'java', 'R']
```

리스트(list)

- 리스트(list)의 메소드
 - 삽입: insert
 - 원하는 위치에 요소 추가

```
list_8.insert(2, 'C++')
list_8
```

```
['파이썬', 'C', 'C++', 'java', 'R']
```

```
list_8.insert(5, 'C')
list_8
```

```
['파이썬', 'C', 'C++', 'java', 'R', 'C']
```

리스트(list)

- 리스트(list)의 메소드
 - 수정: Indexing

```
list_8[0] = 'Python'  
list_8
```

```
['Python', 'C', 'C++', 'java', 'R', 'C']
```

- 삭제: remove → 지우고자 하는 요소 중 가장 앞에 있는 요소 삭제

```
list_8
```

```
['Python', 'C', 'C++', 'java', 'R', 'C']
```

```
list_8.remove('C')  
list_8
```

```
['Python', 'C++', 'java', 'R', 'C']
```

리스트(list)

- 리스트(list)의 메소드

- 값 반환: pop

- Pop은 괄호() 안 인덱스에 해당하는 요소를 반환
 - 인덱스를 생략하면 가장 마지막 요소를 반환
 - 반환한 요소는 리스트에서 사라짐

```
list_8.pop()
```

```
'C'
```

```
list_8
```

```
['Python', 'C++', 'java', 'R']
```

리스트(list)

- 리스트(list)의 메소드
 - 정렬: sort
 - 리스트의 요소를 크기 순으로 정렬

```
list_9 = [1, 5, 7, 2, 6]
```

```
list_9.sort()  
list_9
```

```
[1, 2, 5, 6, 7]
```

list.sort(key=None, reverse=False)

- key : 정렬해주고 싶은 기준, 기본적으로 < 비교에 의해 정렬(오름차순)
- reverse : True로 설정하게 되면 > 비교에 의해 정렬(내림차순)

```
list_10 = [1, 47, 12, 6]  
list_10.sort(reverse=True)  
list_10
```

```
[47, 12, 6, 1]
```

리스트(list)

- 리스트(list)의 메소드
 - 역행: reverse
 - 리스트의 요소 순서를 뒤집음

```
list_9
```

```
[1, 2, 5, 6, 7]
```

```
list_9.reverse()  
list_9
```

```
[7, 6, 5, 2, 1]
```

```
list_10.reverse()  
list_10
```

```
[1, 6, 12, 47]
```

Practice 3

- Q1
 - 변수출력

a = '아메리카노를'
b = '좋아한다'
c = '많이'

정답을 적어주세요

'아메리카노를많이많이많이좋아한다'

Practice 3

- Q2
 - 리스트 생성

다음의 도시 다섯 곳을 cities라는 리스트에 담아보자

```
# 정답을 적어주세요
```

```
# 아래의 주석을 풀고 실행시켜 보세요  
# cities
```

```
['seoul', 'LA', 'beijing', 'paris', 'rome']
```

Practice 3

- Q3
 - 리스트 삽입

리스트 cities의 가장 앞에 busan이, 가장 뒤에는 incheon이라는 단어가 오도록 수정해보자

```
# 정답을 적어주세요
```

```
# 아래의 주석을 풀고 실행시켜 보세요  
# cities
```

```
['busan', 'seoul', 'LA', 'beijing', 'paris', 'rome', 'incheon']
```

Practice 3

- Q4
 - pop을 이용한 요소 반환

pop을 이용하여 busan을 삭제해보자

```
# 정답을 적어주세요
```

```
'busan'
```

```
# 아래의 주석을 풀고 실행시켜 보세요  
# cities
```

```
['seoul', 'LA', 'beijing', 'paris', 'rome', 'incheon']
```

Practice 3

- Q5
 - remove 사용한 요소 삭제

remove를 이용하여 incheon을 삭제해보자

```
# 정답을 적어주세요
```

```
# 아래의 주석을 풀고 실행시켜 보세요
```

```
# cities
```

```
['seoul', 'LA', 'beijing', 'paris', 'rome']
```

- Q6
 - pop()와 remove()의 차이는 무엇인가?

Practice 3

- Q7

- 리스트 길이 측정

```
len() 함수를 이용하여 리스트의 길이를 확인해보자
```

```
# 정답을 적어주세요
```

```
5
```

- Q8

- 리스트의 요소 길이 측정

```
len() 함수를 이용하여 리스트 세번째 요소의 길이를 확인해보자
```

```
# cities
```

```
['seoul', 'LA', 'beijing', 'paris', 'rome']
```

```
# 정답을 적어주세요
```

```
7
```

튜플 (tuple)

- 튜플(tuple): ()

- 괄호 ()을 이용하여 튜플 선언
- 한 개의 요소만 사용하여 튜플을 선언할 때에는 반드시 콤마(,)를 사용해야 함
- 괄호 없이 튜플 선언 가능

```
tuple_1 = ()  
tuple_2 = tuple()
```

```
tuple_3 = (1, 2)  
tuple_4 = (3,)  
tuple_5 = (4, 5, (6, 7))  
tuple_6 = 8, 9, 10
```

튜플 (tuple)

- 튜플(tuple)의 특징
 - 삽입, 삭제, 수정 등이 불가능

```
tuple_3 = (1, 2)
tuple_3[0] = 3
```

```
-----  
-----  
TypeError
```

```
t call last)
```

```
<ipython-input-87-c8372825bae8> in <module>()
    1 tuple_3 = (1, 2)
----> 2 tuple_3[0] = 3
```

```
Traceback (most recent
```

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

튜플 (tuple)

- 튜플(tuple)의 연산

- 덧셈: 두 튜플을 연결하여 하나의 튜플로 만듦
- 곱셈: 튜플을 반복하여 연결하여 하나의 튜플로 만듦

```
tuple_3 = (1, 2)
tuple_4 = (3, )
tuple_3 + tuple_4
```

(1, 2, 3)

```
tuple_4 * 2
```

(3, 3)

- 튜플의 필요성?
 - 불변성(immutable)

딕셔너리(dictionary)

- 딕셔너리(dictionary): {}

- 중괄호 {}을 이용하여 딕셔너리 선언
- 딕셔너리는 **key**와 **value**로 구성됨
- 순서가 있는 자료형이 아니며, key를 통해 value에 접근
- key는 고유한 값으로 중복될 수 없음
- value는 중복가능

```
dict_1 = {'key' : 'value'}  
dict_2 = dict()
```

딕셔너리(dictionary)

- 딕셔너리(dictionary)의 활용
 - 요소 추가 → 새로운 key에 value를 할당

```
dict_3 = {'amazon' : 100, 'apple' : 200}
```

```
dict_3['google'] = 300  
dict_3
```

```
{'amazon': 100, 'apple': 200, 'google': 300}
```

- 요소 수정 → key를 이용하여 value에 접근하여 덮어쓰기

```
dict_3['amazon'] = 1  
dict_3
```

```
{'amazon': 1, 'apple': 200, 'google': 300}
```

딕셔너리(dictionary)

- 딕셔너리(dictionary)의 활용
 - 요소 삭제 → key 사용

```
del dict_3['google']
dict_3

{'amazon': 1, 'apple': 200}
```

- del: 파일 자체 명령어로서 메모리 자체에서 값을 삭제

딕셔너리(dictionary)

- 딕셔너리(dictionary)의 메소드
 - 딕셔너리 key에 접근 → keys()

```
t_3 = {'amazon' : 100, 'apple' : 200, 'google' : 150, 'facebook'  
nt(dict_3.keys())  
nt(type(dict_3.keys())))  
  
dict_keys(['amazon', 'apple', 'google', 'facebook'])  
<class 'dict_keys'>
```

- 딕셔너리 value에 접근 → values()

```
print(dict_3.values())  
print(type(dict_3.values()))  
  
dict_values([100, 200, 150, 250])  
<class 'dict_values'>
```

딕셔너리(dictionary)

- 딕셔너리(dictionary)의 메소드
 - key와 value에 함께 접근 → items()

```
print(dict_3.items())
print(type(dict_3.items()))

dict_items([('amazon', 100), ('apple', 200), ('google', 150),
('facebook', 250)])
<class 'dict_items'>

list(dict_3.keys())[0]

'amazon'
```

딕셔너리(dictionary)

- 딕셔너리(dictionary)의 메소드

- key를 이용하여 value에 접근
 - 인덱싱처럼 키 값을 이용하여 접근 가능
 - 메소드 get() 사용 가능

```
dict_3
```

```
{'amazon': 100, 'apple': 200, 'google': 150, 'facebook': 250}
```

```
print(dict_3['facebook'])  
print(dict_3.get('facebook'))
```

```
250
```

```
250
```

딕셔너리(dictionary)

- 딕셔너리(dictionary)의 특징

- key를 이용하여 value에 접근

- key를 이용한 접근에서 해당 key가 없으면 오류 발생
 - 메소드 get()을 이용한 접근에서 해당 key가 없으면 None 반환

```
1 print(dict_3['lg'])

-----
KeyError Traceback (most recent call last)
<ipython-input-121-8eea37fb86d6> in <module>()
----> 1 print(dict_3['lg'])

KeyError: 'lg'

SEARCH STACK OVERFLOW

1 print(dict_3.get('lg'))
```

None

셋(set)

- 셋(set): {}

- 중괄호 {}을 이용하여 셋 선언
- 중복을 허용하지 않음
- 순서가 없음

```
set_1 = set()  
set_2 = set('Hello')  
set_3 = set([1, 2, 3])  
set_4 = {3,5, 'hi'}
```

```
set_2
```

```
{'H', 'e', 'l', 'o'}
```

```
set_3
```

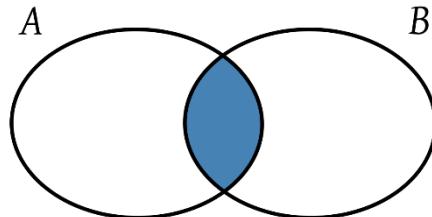
```
{1, 2, 3}
```

```
set_4
```

```
{3, 5, 'hi'}
```

셋(set)

- 셋(set)의 연산
 - 교집합 → intersection()



```
set_5 = {1,2,3,4}  
set_6 = {4,5,6,7}
```

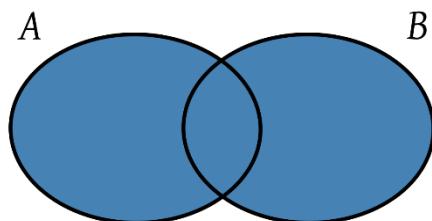
```
set_5 & set_6
```

```
{4}
```

```
set_5.intersection(set_6)
```

```
{4}
```

셋(set)



- 셋(set)의 연산
 - 합집합 → union()

```
set_5 | set_6
```

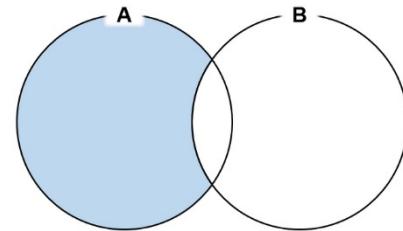
```
{1, 2, 3, 4, 5, 6, 7}
```

```
set_5.union(set_6)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

셋(set)

- 셋(set)의 연산
 - 차집합 → difference()



```
set_5 - set_6
```

```
{1, 2, 3}
```

```
set_5.difference(set_6)
```

```
{1, 2, 3}
```

셋(set)

- 셋(set)의 메소드
 - 셋의 요소 추가 → add()

```
set_6 = {4, 5, 6, 7}  
set_6.add(8)  
set_6
```

```
{4, 5, 6, 7, 8}
```

- 셋의 요소 수정 → update()

```
set_6.update([1,2,3])  
set_6
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

셋(set)

- 셋(set)의 메소드
 - 셋의 요소 삭제 → remove()

```
set_6.remove(3)  
set_6  
  
{1, 2, 4, 5, 6, 7, 8}
```

Practice 4

- Q1

- 딕셔너리 생성

국어, 영어, 수학 점수가 키로 하는 딕셔너리를 만들어보자.

각각의 점수는 다음과 같다.

국어 : 87

영어 : 88

수학 : 92

정답을 적어주세요

```
{'Korean': 87, 'English': 88, 'Math': 92}
```

Practice 4

- Q2

- 연산 결과 예측

```
# (1)
```

```
3 <= 1
```

```
# (2)
```

```
6 % 3 == 0
```

```
s1 = {'a', 'b', 'c'}
```

```
s2 = {'b', 'e', 'f', 'g'}
```

```
# (3)
```

```
s1 & s2 == 'b'
```

```
# (4)
```

```
s1 & s2 == {'b'}
```

Quiz 1

- 숫자자료형
 - int, float, complex
 - 사칙연산(operations)
 - 변수(variable)
- 불자료형
 - 관계연산
 - bool (True or False)
- 군집자료형
 - str, list, tuple, dictionary, set

3. Flow Control condition & loop

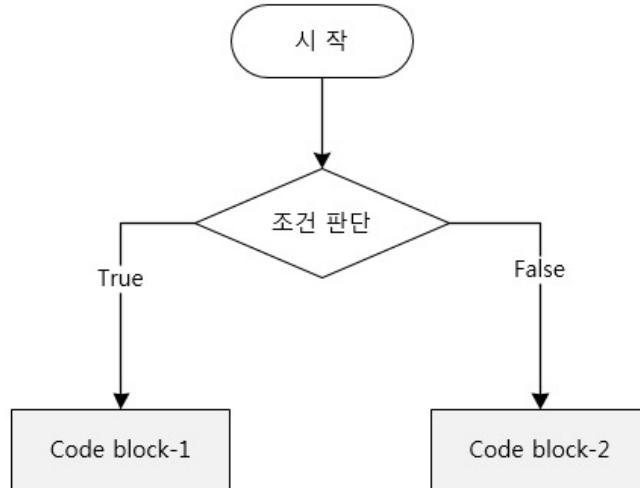
흐름 제어(Flow Control)

- **흐름제어**
 - 프로그래밍은 입력, 처리, 출력 등의 과정을 순차적으로 수행하는 프로그램을 작성하는 과정을 의미
 - 이때, 특정한 조건에 맞는 경우에만 처리를 수행하거나, 반복적인 작업이 필요할 경우 프로그램의 흐름을 제어할 수 있음
- **조건문**
 - if / elif / else
- **반복문**
 - for
 - while

조건문 (conditional statement)

- 조건문

- 들여쓰기(indentation)
 - 파이썬은 들여쓰기를 사용하여 영역을 지정
→ 조건문, 반복문, 함수, 클래스 등
 - 일반적으로 탭, 4칸 들여쓰기 방식 사용
- 조건(true/false)
 - 조건문은 조건의 참/거짓을 이용하여 작업 흐름을 제어



조건문 (conditional statement)

- 조건(True/False)

- 비교 연산자

```
a = 3  
b = 5
```

```
a < b
```

True

```
c = 3  
d = 3
```

```
c <= d
```

True

```
c != d
```

False

조건문 (conditional statement)

- 조건(True/False)
 - 논리 연산자: and / or / not

and	or	not
True and True	True or True	not True
True	True	False
True and False	True or False	not False
False	True	True
	False or False	
	False	

조건문 (conditional statement)

- 조건(True/False)

- 논리 연산자: in

```
in
```

- 해당 자료형의 요소인지 파악하기 : in / not in

```
e = [1, 3, 5, 7]
```

```
0 in e
```

```
False
```

```
1 in e
```

```
True
```

```
2 not in e
```

```
True
```

```
3 not in e
```

```
False
```

조건문 (conditional statement)

- 조건문

- If / else 조건문

- 조건문을 테스트해서 참이면 if문 다로 다음 문장(if 블록)들을 수행하고, 조건문이 거짓이면 else문 다음 문장(else 블록)들을 수행
 - else문은 if문 없이 독립적으로 사용 불가능

```
if 조건 :  
    수행 할 문장1  
    수행 할 문장2  
else :  
    수행 할 문장3  
    수행 할 문장4
```

```
money = True  
  
if money :  
    print("여행을 간다")  
else :  
    print("집에서 쉰다")
```

여행을 간다

조건문 (conditional statement)

- If / elif / else

```
if 조건1 :  
    수행 할 문장1  
elif 조건2 :  
    수행 할 문장2  
else :  
    수행 할 문장3
```

```
money = 20000  
  
if money < 5000 :  
    print('라면을 먹는다')  
elif 5000 < money < 25000 :  
    print('치킨을 먹는다')  
elif 25000 < money < 50000 :  
    print('삼겹살을 먹는다')  
else :  
    print('소고기를 먹는다')
```

치킨을 먹는다

조건문 (conditional statement)

- pass

- 조건을 충족해도 특정 작업이 일어나지 않게 함 (에러 방지)
- 프로그램 초기 설계시 사용 용이

```
money = 20000
card = True

if card :
    if money < 30000 :
        print("삼겹살을 먹는다")
    else :
        print("소고기를 먹는다")
else :
    if money <= 1000 :
        pass
    else :
        print("라면을 먹는다")
```

삼겹살을 먹는다

조건문 (conditional statement)

- 조건문의 중첩

- 조건문 속에 또다른 조건문을 작성 가능

```
money = 1000
card = False

if card :
    if money < 30000 :
        print("삼겹살을 먹는다")
    else :
        print("소고기를 먹는다")
else :
    if money <= 1000 :
        pass
    else :
        print("라면을 먹는다")
```

Practice 5

- Q1
 - If-else문 작성
- Q2
 - If-elif-else문 작성

a와 b가 주어졌을 때, a 곱하기 b가 30 초과면 실패, 30이하면 성공이 출력되도록 만들어보자.

```
a = 6  
b = 3
```

정답을 작성해 주세요.

성공

b로 a를 나눈 나머지가 3 초과면 실패, 3이면 무승부, 3 미만이면 성공이 출력되도록 만들어 보자

```
a = 3  
b = 4
```

정답을 작성해 주세요.

정답을 작성해 주세요.

무승부

Practice 6

- Q1

- 비교연산자, 논리연산자 활용
- 조건문 활용

```
3 <= 6

15 % 8 == 3

298 % 100 < 5

tmp = []
if tmp :
    print('Not empty!')
else :
    print("empty!")

tmp2 = [1,5,8,'e','n','.']

if 3 not in tmp2 :
    print("X")
else :
    print("O")
```

Practice 6

- Q2
 - 조건문 활용

정부에서는 여름철 전력수급에 차질이 없도록 하기 위해서 실내온도가 26도 이상일때만 에어컨을 사용할 수 있도록 권유하고 있다. 이에 맞춰 회사에서는 중앙냉난방을 제어하려고한다.

실내온도가 26도 미만일 경우 '선풍기', 26도 이상일 경우 '에어컨'을 알려주도록 코딩을 해보자. (화면에 각 단어를 출력하기)
실제로 잘 작동하는지 온도를 바꾸어가면서 확인해봅시다.

```
temp = 29
```

```
# 정답을 작성해 주세요.
```

```
에어컨
```

Practice 6

- Q3
 - 조건문 활용

숫자의 자릿수를 판별하는 기계를 만들려고 한다. 한자리수일 경우 1, 두자리수 일 경우 2, 세자리 수 일 경우 3을 출력하는 조건문을 만들어보자.(주어지는 숫자는 세자리수 이하이다.)

```
num = 78
```

```
# 정답을 작성해 주세요.
```

2

```
# 정답을 작성해 주세요.
```

2

반복문 (loop)

- **반복문**

- 특정 조건이 거짓일 될 때까지 반복하거나(while), 특정 범위의 크기 만큼을 반복(for)하여 작업 수행
- 반복문은 종료 조건이 필수적으로 필요 → 종료 조건이 없을 시 무한 루프 발생

- **While문**

- 특정 조건을 기준으로 조건이 거짓이 될 때까지 반복작업 수행
- 반복 종료의 기준은 조건문의 참/거짓 판별

- **For문**

- 특정 범위를 기준으로 범위의 끝까지 반복작업 수행
- 반복 종료의 기준은 주어진 자료의 범위

반복문 (loop)

- **while** 반복문
 - 조건문이 참인 경우 반복해서 문장을 수행

```
while 조건 :  
    수행할 문장1  
    수행할 문장2
```

```
# 100/2의 짝수 프린트하기  
i = 1  
  
while i <= 10 :  
    if i % 2 == 0 :  
        print(i)  
    i += 1
```

```
2  
4  
6  
8  
10
```

반복문 (loop)

- **while** 반복문

- 조건문과 더불어 break 명령어를 사용하여 반복작업 종료 가능
 - 0은 False / 이외의 숫자는 True

```
# 100번째 방문자 찾기
i = 90

while i :
    i += 1
    if i == 100 :
        print("축하합니다. %d번째 방문자입니다." % i)
        break
print("감사합니다. 이벤트가 종료되었습니다.")
```

```
축하합니다. 100번째 방문자입니다.
감사합니다. 이벤트가 종료되었습니다.
```

반복문 (loop)

- **while** 반복문

- 조건문과 더불어 continue 명령어를 사용하여 반복문 내 작업의 처음으로 이동 가능

```
i = 0

while i < 11 :
    i += 1
    if i == 6 :
        continue
    if i % 2 == 0 :
        print(i)
```

```
2
4
8
10
```

반복문 (loop)

- **for** 반복문

- 변수의 범위로부터 반복적으로 하나의 요소들을 가져오며 작업수행

```
for 변수 in range(변수가 속한 자료형, 혹은 변수의 범위) :  
    수행해야할 문장1  
    수행해야할 문장2
```

- **range(시작값, 끝값, 간격)**: 변수의 범위 지정, 간격 생략시 1

```
for x in range(0,5) :  
    print(x)
```

```
0  
1  
2  
3  
4
```

반복문 (loop)

- **for** 반복문
 - 자료형으로 범위제공

```
word = 'Hello!'

for w in word:
    print(w)
```

```
H  
e  
l  
l  
o  
!
```

```
for a, b in [(2,1), (2,2), (2,3), (2,4)] :
    print(a*b)
```

```
2  
4  
6  
8
```

Practice 7

- Q1
 - 반복문을 통해 카운트 다운하기

```
for count in range(5, -1, -1) :  
    print(count)
```

```
5  
4  
3  
2  
1  
0
```

Practice 8

- Q1
 - 구구단 프로그램 작성 (반복문의 중첩)

구구단 2단과 3단을 출력하기

정답을 작성해주세요.

==== 2 단 ===

2

4

6

8

10

12

14

16

18

==== 3 단 ===

3

6

9

12

15

18

21

24

27

Quiz 2

▼ 문제 1

20이하의 자연수 중 3으로 나눴을 때 나머지가 1인 숫자를 출력하기

[] 1 # 정답을 작성해주세요.

1
4
7
10
13
16
19

▼ 문제 2

다음과 같이 출력하기

*

**

[] 1 # 정답을 작성해주세요.

*

**

▼ 문제 3

고객의 개인정보보호를 위하여 이름을 비식별화 하여 출력하기

예시) 홍길동 → 홍*동

[] 1 names = ['홍길동', '홍계월', '김철수', '이영희', '박첨지']

[] 1 # 정답을 작성해주세요.

홍*동
홍*월
김*수
이*희
박*지

4. Function

함수 (Function)

- **함수**

- 함수는 프로그램을 개발할 때, 어떤 문제를 해결할 때 하나의 기능을 따로 떼어내서 구현할 수 있게 하여 효율을 향상시킴

```
def 함수이름( 매개변수 ) :  
    함수의 내용  
    return 반환값
```

- **함수이름**: 사용자가 정의하는 함수 이름, 기존에 사용되는 함수나 예약어들을 제외하고 사용
- **매개변수**: 함수 안에서 사용 할 변수들 (생략 가능)
- **return**: 함수 안에서 모든 연산을 마친 후 반환할 값(생략 가능)

```
def 함수이름(함수인자) :  
    수행할 문장1  
    수행할 문장2  
    return 반환값
```



함수 (Function)

- **함수 정의 (Quiz!)**

- 두 입력을 받아 덧셈 연산을 수행한 뒤, 그 결과값을 반환하는 함수 작성
- 함수 정의 → 함수 호출

```
# add 함수를 짜봅시다
```

```
8
```

```
c = add(3, 5)  
c
```

```
8
```

함수 (Function)

- return이 없는 함수

```
def sub(a, b) :  
    print('뺄셈의 결과는 %d입니다.' %(a-b))  
    return
```

```
d = sub(1, 2)
```

뺄셈의 결과는 -1입니다.

```
print(d)
```

None

```
def mul(a, b) :  
    print('%d 와 %d의 곱은 %d입니다.' %(a, b, a*b))
```

```
mul(3, 2)
```

3 와 2의 곱은 6입니다.

함수 (Function)

- 매개변수와 return이 모두 없는 함수

```
def start():
    print("Hello World!")
```

```
start()
```

```
Hello World!
```

Practice 9

- Q1

- 함수 작성

함수를 만들어 보자.

- 이 함수는 두 개의 숫자를 input으로 받으면 앞의 숫자를 뒤의 숫자로 나누는 함수이다.
- 나눗셈을 한 후에는 몫과 나머지 순으로 된 튜플 값을 반환한다.

```
# 정답을 작성해주세요.
```

```
div(10, 3)
```

```
(3, 1)
```

```
div(3, 5)
```

```
(0, 3)
```

```
div(1, 10)
```

```
(0, 1)
```

Practice 9

- Q2
 - 함수 작성

함수를 만들어보자.

- 이 함수는 두 개의 숫자를 input으로 받으면 작은 수로 큰 수를 나눈 몫과 나머지를 반환하는 함수이다.
- 반환 값은 튜플로 되어 있으며 몫, 나머지 순으로 되어 있다.
- 단, 0으로 나누는 것은 불가하기 때문에 두 수 중에 작은 수가 0이라면 화면에 '0은 사용할 수 없습니다.'를 출력하고 종료되어야 한다.

```
# 정답을 작성해주세요.
```

아래의 코드를 실행하여 결과를 확인해보세요.

```
div2(3, 5)
```

```
(1, 2)
```

```
div2(0, 5)
```

0은 사용할 수 없습니다.

```
div2(10, 1)
```

```
(10, 0)
```

함수 (Function)

- 함수의 매개변수 초기값 설정
 - 초기값 설정시 매개변수 순서에 주의
→ 초기값 없는 변수, 초기값 있는 변수 순으로 배치
 - positional argument, keyword argument

```
def function(a, n=2) :  
    print("%d의 제곱은 %d 입니다." %(a, a**n))
```

```
function(4)
```

4의 제곱은 16 입니다.

함수 (Function)

- 함수의 매개변수가 몇 개가 필요한지 모를 때
 - *args (non-keyword argument)

```
def test(*args):  
    print(type(args))  
    print(args)
```

```
test(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
<class 'tuple'>  
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
test([1,2],3)
```

```
<class 'tuple'>  
([1, 2], 3)
```

함수 (Function)

- 함수의 매개변수가 몇 개가 필요한지 모를 때
 - **kwargs (keyword argument)

```
def test(**kwargs):
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))
```

```
test(first ='1', mid ='2', last='3')
```

```
first == 1
mid == 2
last == 3
```

Practice 10

- Q1
 - 함수 정의
- Q2
 - 함수 정의

어떠한 string을 받으면 일정한 단위로 끊어서 화면에 출력하는 함수를 짜보자.

끊는 단위는 따로 정하지 않으면 2로 설정해보자.

```
# 정답을 작성해주세요.
```

```
func('테스트를 위한 문장입니다.')
```

```
테스  
트를  
위  
한  
문장  
입니  
다.
```

```
func('테스트를 위한 문장입니다.', 4)
```

```
테스트를  
위한  
문장입니  
다.
```

add_all 함수를 짜봅시다

```
# 정답을 작성해주세요.
```

```
add_all(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
55
```

함수 (Function)

- 전역변수(global), 지역변수(local)

- 함수 안에 선언한 변수를 함수 밖에서도 사용이 가능한가?
- 함수 안의 블록은 함수 고유의 영역이기 때문에 변수를 공유하지 않음
- 즉, 함수 안에서 선언한 변수는 함수 밖에서 사용이 불가능

```
def myfunction() :  
    in_val = '함수 안의 변수'  
  
myfunction()  
  
in_val  
  
-----  
NameError  
t call last)  
<ipython-input-30-a23adeea90d3> in <module>  
----> 1 in_val  
  
NameError: name 'in_val' is not defined
```

함수 (Function)

- 전역변수(global), 지역변수(local)
 - 함수 실행시 출력값은? → a의 값은?

```
a = 1

def myfunction2(b) :
    b += 1
    print(b)
```

```
myfunction2(a)
```

```
a
```

함수 (Function)

- 전역변수(global), 지역변수(local)
 - 함수 안에 있는 변수를 밖에서도 사용하고 싶다면?
 - return 사용

```
def myfunction3(a) :  
    a += 1  
    print(a)  
    return a
```

```
b = myfunction3(a)
```

```
2
```

```
b
```

```
2
```

```
a = 1
```

함수 (Function)

- 전역변수(global), 지역변수(local)
 - 함수 안에 있는 변수를 밖에서도 사용하고 싶다면?
 - global 사용

```
def myfunction4():
    global a
    a += 1
    print(a)
```

```
myfunction4()
```

```
2
```

```
a
```

```
2
```

함수 (Function)

- 전역변수(global), 지역변수(local)
 - 함수 안에 있는 변수를 밖에서도 사용하고 싶다면?
 - global 사용시 유의사항

```
for _ in range(5) :  
    c = myfunction3(a)  
    print('함수의 결과 : %d' %c )  
    print('a의 값 : %d' %a)  
  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2
```

```
for _ in range(5) :  
    myfunction4()  
    print('a의 값 : %d' %a)  
  
3  
a의 값 : 3  
4  
a의 값 : 4  
5  
a의 값 : 5  
6  
a의 값 : 6  
7  
a의 값 : 7
```

함수 (Function)

- 전역변수(global), 지역변수(local)
 - 함수 안에 있는 변수를 밖에서도 사용하고 싶다면?
 - global 사용시 유의사항

```
for _ in range(5) :  
    c = myfunction3(a)  
    print('함수의 결과 : %d' %c )  
    print('a의 값 : %d' %a)
```

```
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2  
3  
함수의 결과 : 3  
a의 값 : 2
```

```
for _ in range(5) :  
    myfunction4()  
    print('a의 값 : %d' %a)
```

```
3  
a의 값 : 3  
4  
a의 값 : 4  
5  
a의 값 : 5  
6  
a의 값 : 6  
7  
a의 값 : 7
```

함수 (Function)

- 함수 호출 추적 예제
 - 다음 예제의 실행 시 어떠한 절차로 함수가 활용되는가?

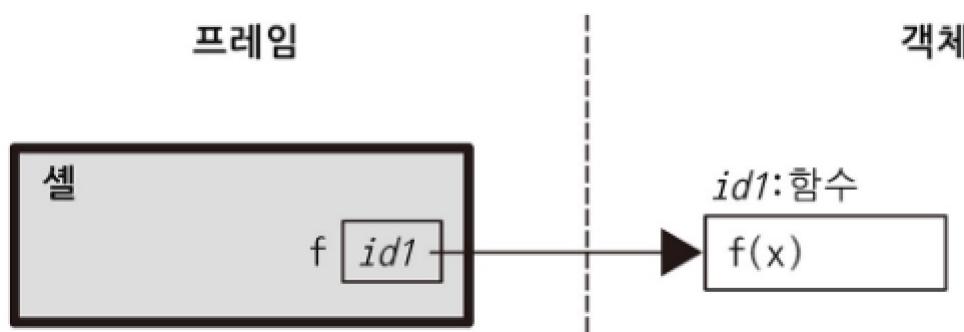
```
[47] 1 def f(x):
2     x = 2*x
3     return x
4
5 x=1
6 x=f(x+1) + f(x+2)
7 x
```



10

함수 (Function)

- 함수 호출 추적 예제
 - 변수 f 생성
 - 함수객체 생성

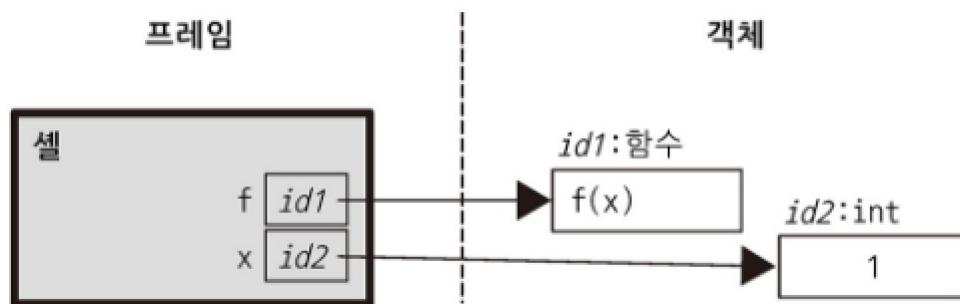


```
[47] 1 def f(x):
      2     x = 2*x
      3     return x
      4
      5 x=1
      6 x=f(x+1) + f(x+2)
      7 x
      8
      9
      10
```

함수 (Function)

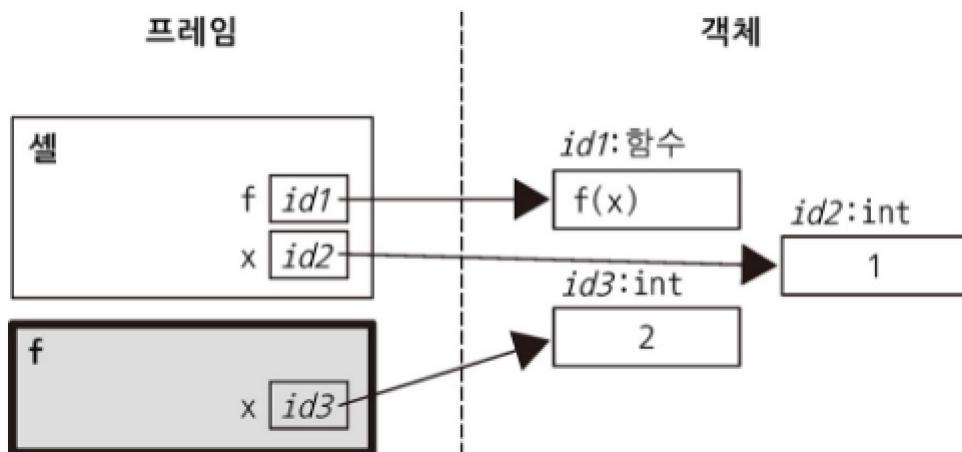
- 함수 호출 추적 예제
 - 셀 안의 전역변수 x 할당

```
[47] 1 def f(x):  
2     x = 2*x  
3     return x  
4  
5 x=1  
6 x=f(x+1) + f(x+2)  
7 x
```



함수 (Function)

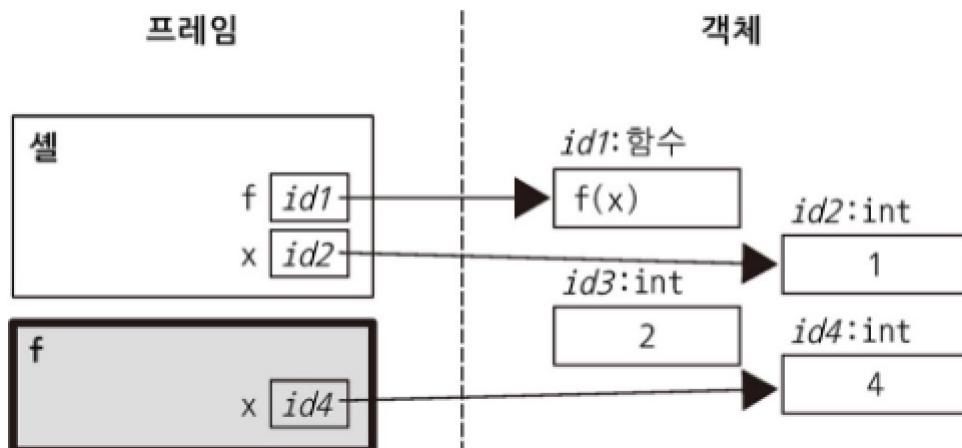
- 함수 호출 추적 예제
 - 표현식에서 $f(x+1)$ 실행
 - $x+1$ 을 평가
 - 평가결과를 함수 안의 지역변수 x 가 참조하게 함



```
[47] 1 def f(x):  
2     x = 2*x  
3     return x  
4  
5 x=1  
6 x=f(x+1) + f(x+2)  
7 x
```

함수 (Function)

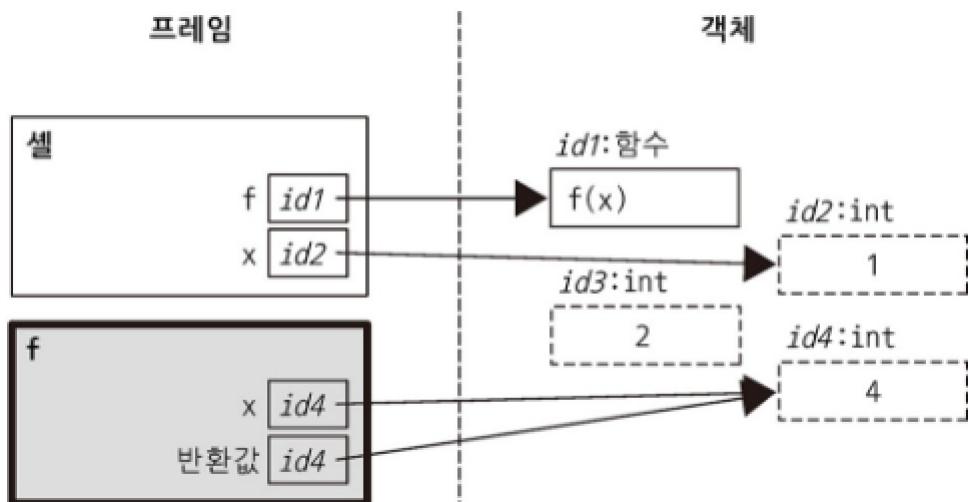
- 함수 호출 추적 예제
 - 지역변수 x에 대한 연산 후 4를 참조



```
[47] 1 def f(x):
[47] 2     x = 2*x
[47] 3     return x
[47] 4
[47] 5 x=1
[47] 6 x=f(x+1) + f(x+2)
[47] 7 x
```

함수 (Function)

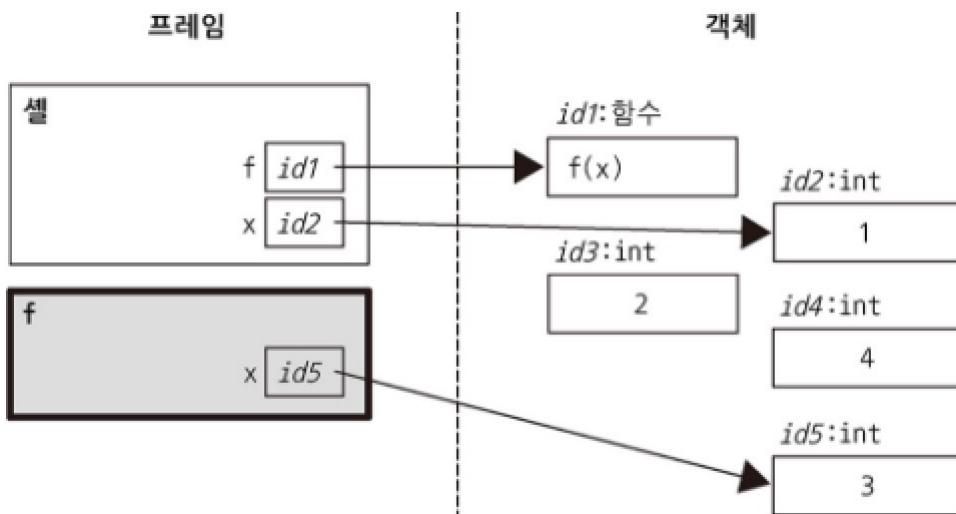
- 함수 호출 추적 예제
 - 함수의 반환 (표현식 평가)



```
[47] 1 def f(x):  
     2     x = 2*x  
     3     return x  
     4  
     5 x=1  
     6 x=f(x+1) + f(x+2)  
     7 x
```

함수 (Function)

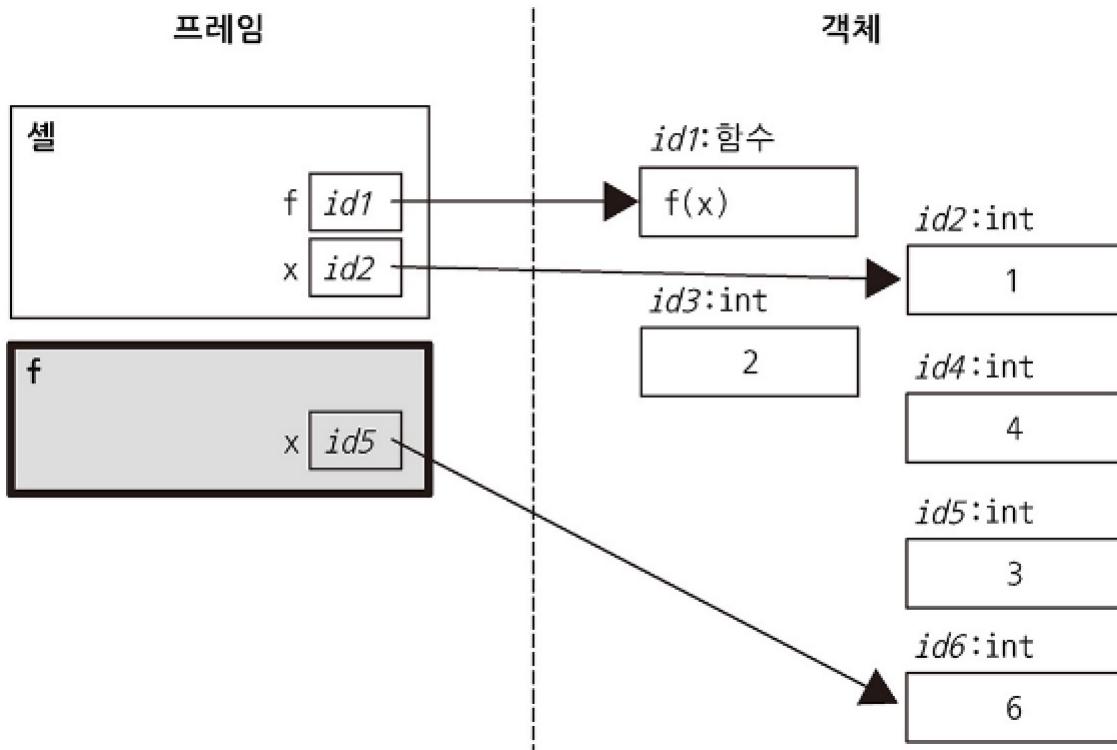
- 함수 호출 추적 예제
 - 표현식에서 $f(x+2)$ 실행
 - $x+2$ 평가
 - 평가결과를 지역변수 x 가 참조하도록 함



```
[47] 1 def f(x):  
2     x = 2*x  
3     return x  
4  
5 x=1  
6 x=f(x+1) + f(x+2)  
7 x
```

함수 (Function)

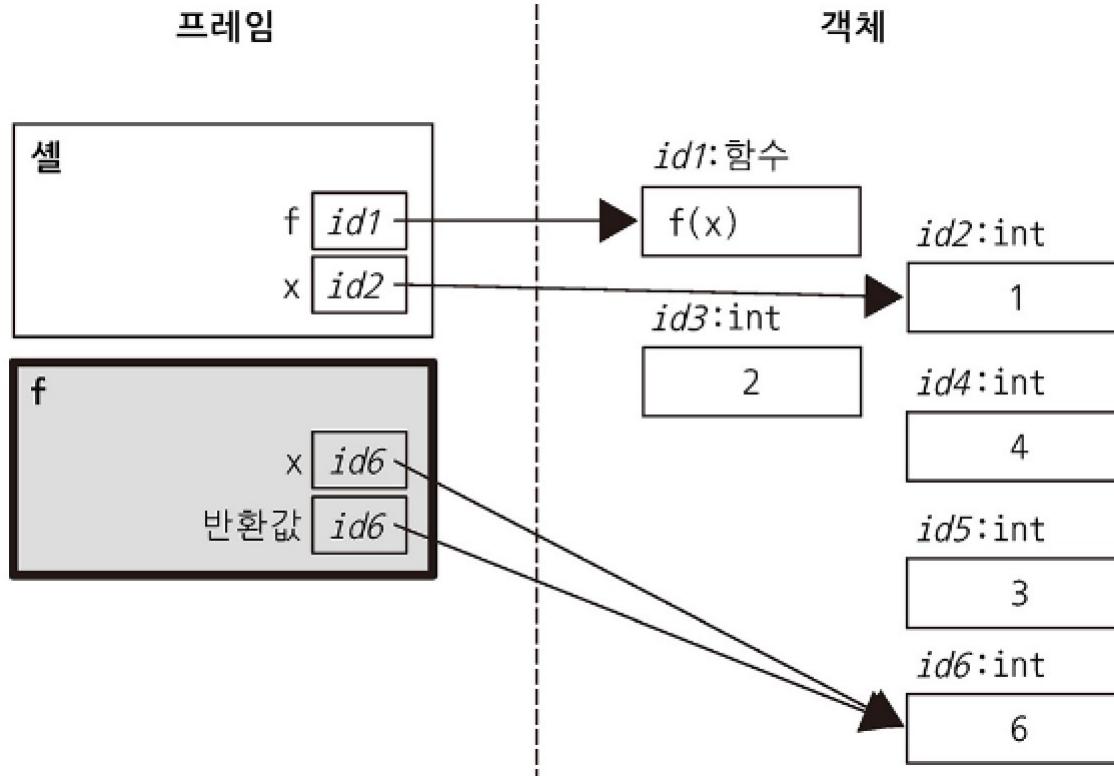
- 함수 호출 추적 예제
 - 지역변수 x에 대한 연산 후 6을 참조



```
[47] 1 def f(x):
      2     x = 2*x
      3     return x
      4
      5 x=1
      6 x=f(x+1) + f(x+2)
      7 x
      8
      9
      10
```

함수 (Function)

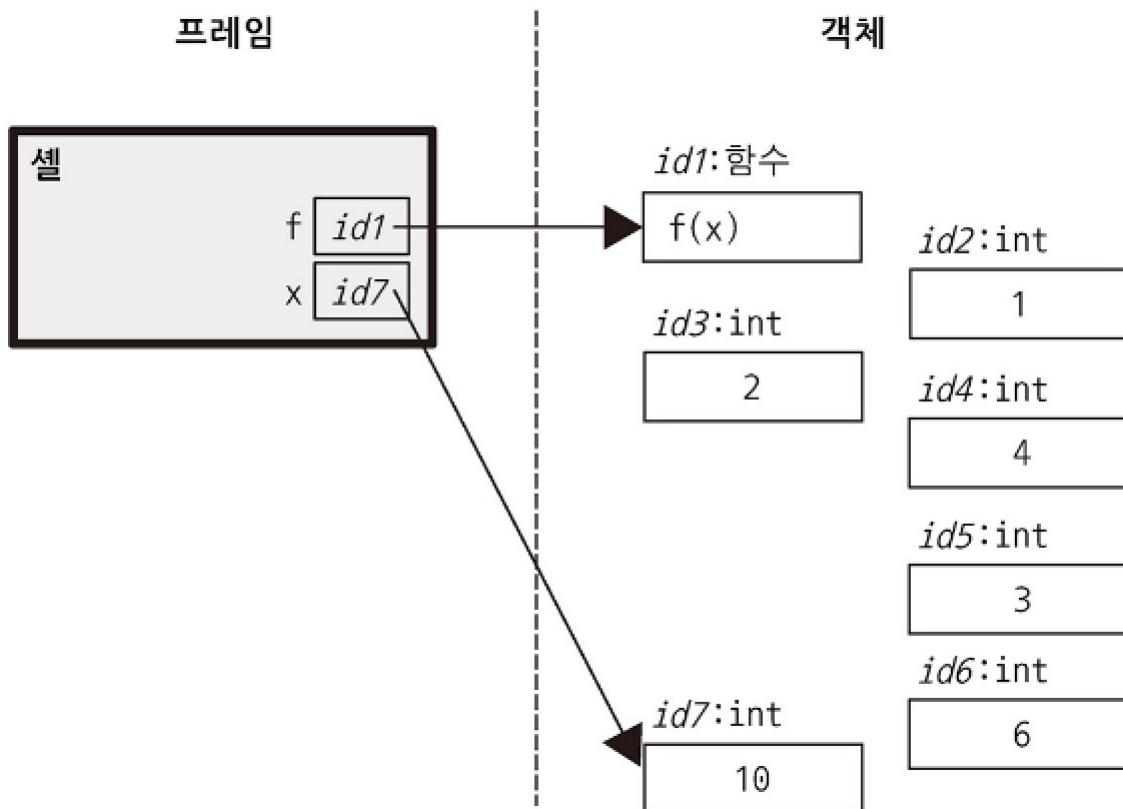
- 함수 호출 추적 예제
 - 함수의 반환 (표현식 평가)



```
[47] 1 def f(x):  
2     x = 2*x  
3     return x  
4  
5 x=1  
6 x=f(x+1) + f(x+2)  
7 x
```

함수 (Function)

- 함수 호출 추적 예제
 - 표현식 평가
 - 평가결과를 전역변수 x가 참조



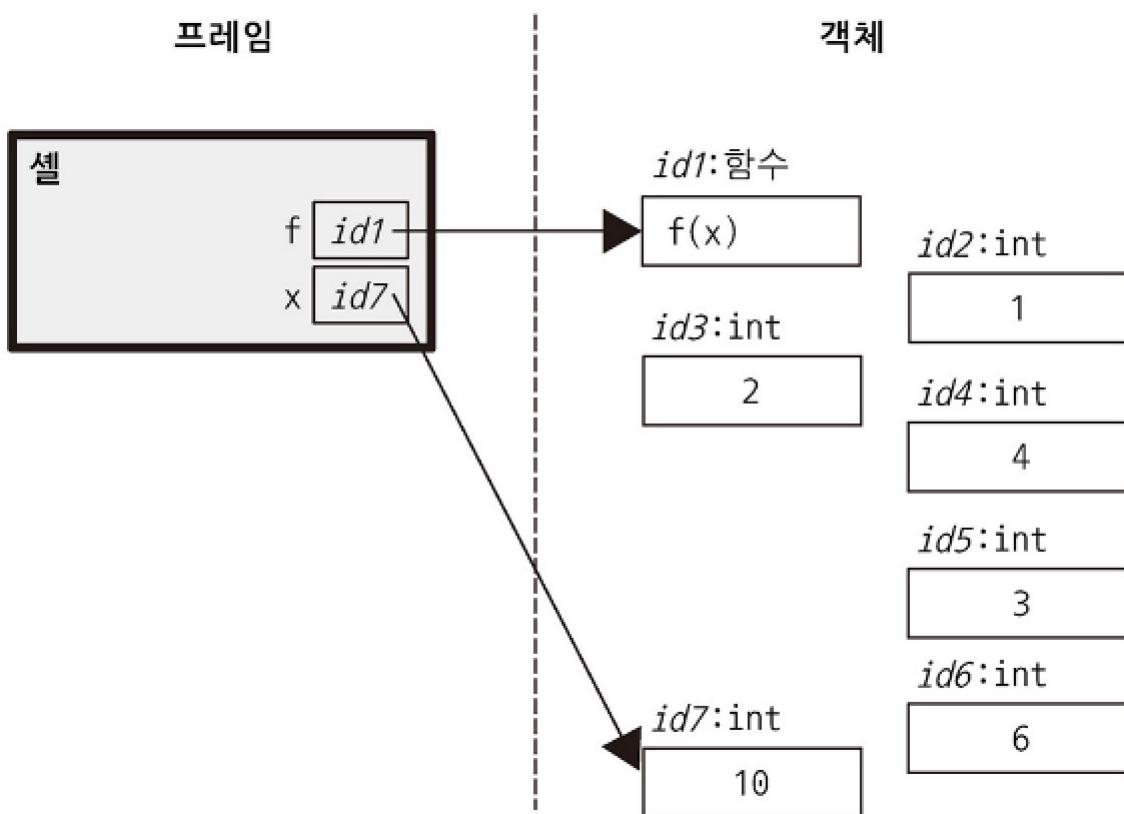
```
[47] 1 def f(x):  
2     x = 2*x  
3     return x  
4  
5 x=1  
6 x=f(x+1) + f(x+2)  
7 x
```

함수 (Function)

- 함수 호출 추적 예제
 - 표현식 평가
 - 평가결과를 전역변수 x가 참조

프레임

객체



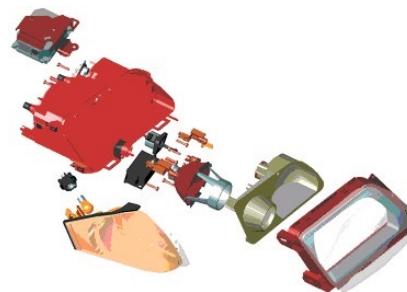
```
1 number = 0
2
3 def id_print(name):
4     global number
5     print(f"{number}. id: {id(name)} -> ", name)
6     number += 1
7
8 def f(x):
9     id_print(x)
10    x = 2*x
11    id_print(x)
12    return x
13 id_print(f)
14
15 x=1
16 id_print(x)
17
18 x=f(x+1) + f(x+2)
19 id_print(x)

0. id: 140259411253872 -> <function f at 0x7f90b0625a70>
1. id: 94831765727744 -> 1
2. id: 94831765727776 -> 2
3. id: 94831765727840 -> 4
4. id: 94831765727808 -> 3
5. id: 94831765727904 -> 6
6. id: 94831765728032 -> 10
```

5. Python Programming Practice

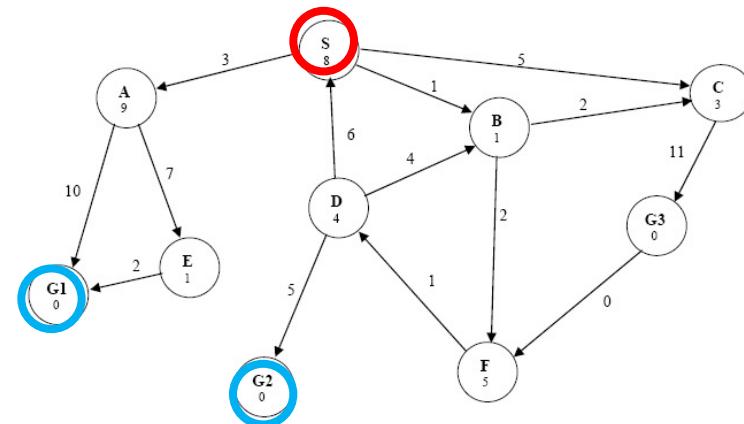
Search

- Problem solving
 - Searching among alternative choices to **find a sequence of actions** by which a goal can be achieved
 - Example
 - Route finding : finding a sequence of road
 - Assembly planning : finding a sequence of part assembly
 - Playing game : finding a sequence of move
 - Inference in FOPC : finding a sequence of applying M.P.



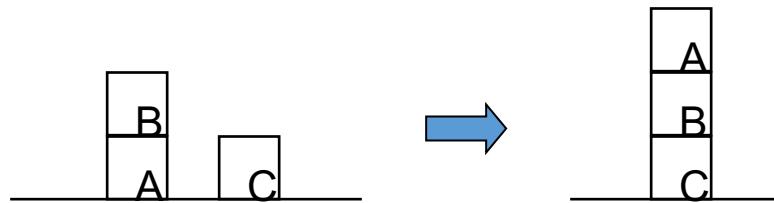
State Space

- State space representation
 - Represent problem space as a *graph*
 - Nodes: states
 - Arcs: Operations to other states
- State space
 - [N, A, S, G]
 - N – nodes(states)
 - A – arcs(operations)
 - S – start state
 - G – goal state



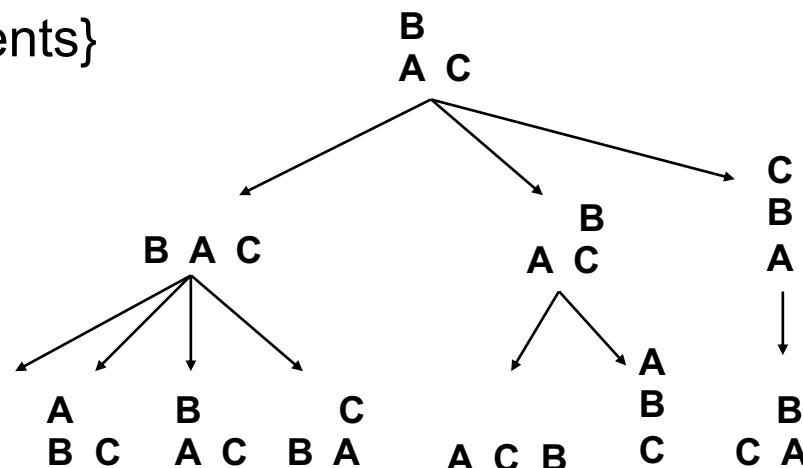
State Space

- Planning in a blocks world



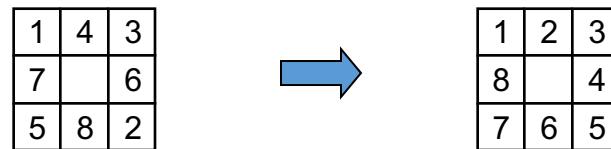
$N = \{\text{different arrangements}\}$

$A = \{\text{moving a block}\}$



State Space

- 8-Puzzle



N = {different arrangements}

A = {moving the blank}

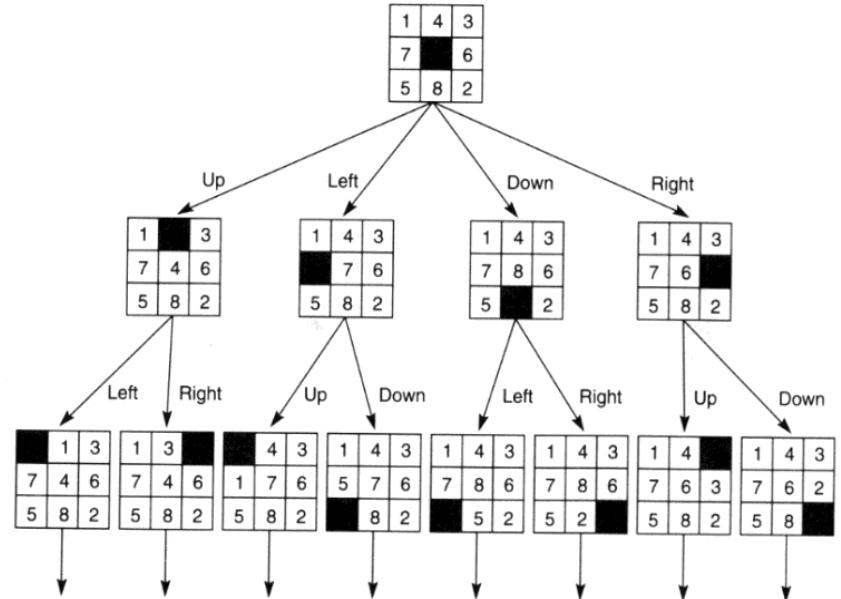


Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

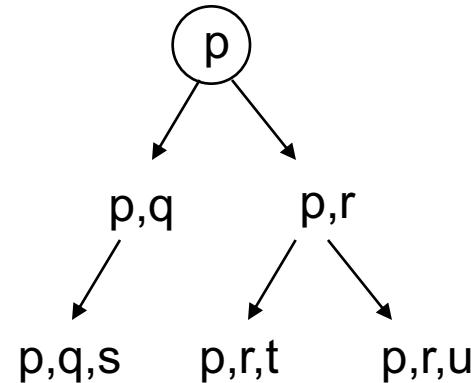
State Space

- Inference

$p \Rightarrow q, p \Rightarrow r, q \Rightarrow s,$
 $r \Rightarrow t, r \Rightarrow u, p$  t ?

N = {true sentences}

A = {applying M.P.}



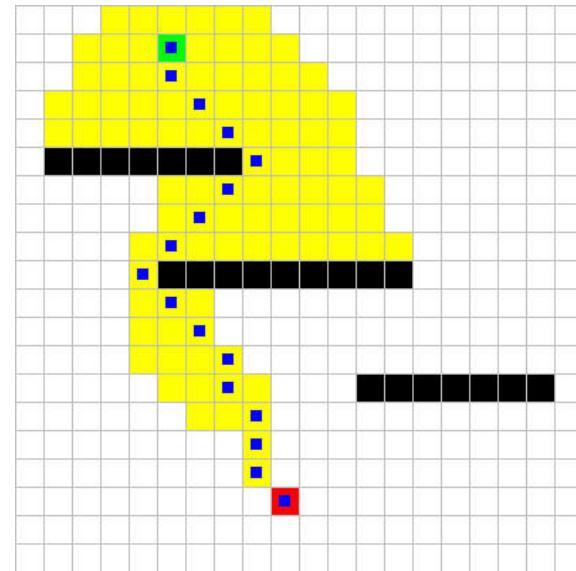
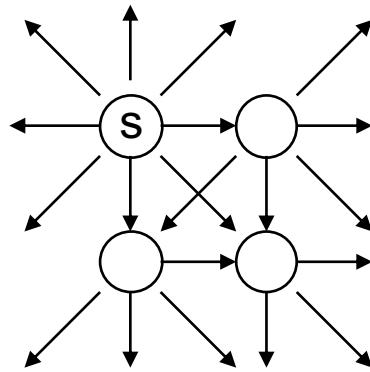
State Space

- Route finding

start location  goal location

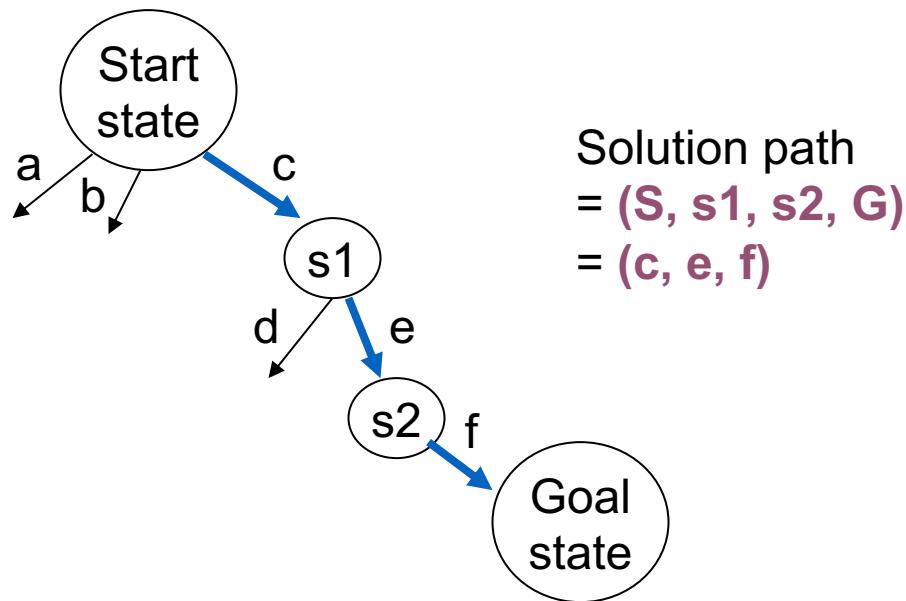
N = {different locations}

A = {move to the adjacent location}



State Space Search

- Search
 - Finding a solution path from S to G



Depth-First Search

- Goes deeper whenever possible

open = {START}, **closed** = \emptyset

while (**open** $\neq \emptyset$)

 remove leftmost state from **open** $\rightarrow X$

 if (X is GOAL) return (success)

 else

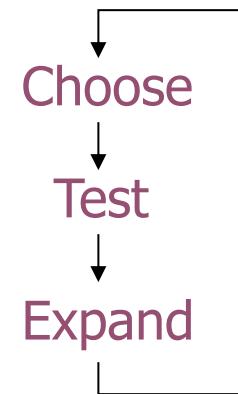
 generate children of X

 put X into **closed**

 eliminate child if it is in open or close

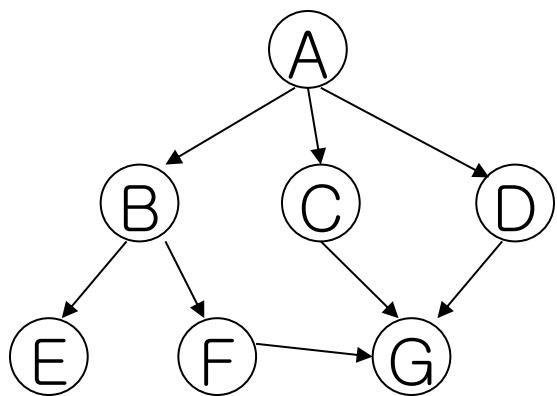
 put remaining children into left of **open**(stack)

 return (fail)



Depth-First Search

- Example



1. open = {}, closed = {}
2. X = , open = {}, closed = {}
3. X = , open = {}, closed = {}
4. X = , open = {}, closed = {}
5. X = , open = {}, closed = {}
6.

Depth-First Search

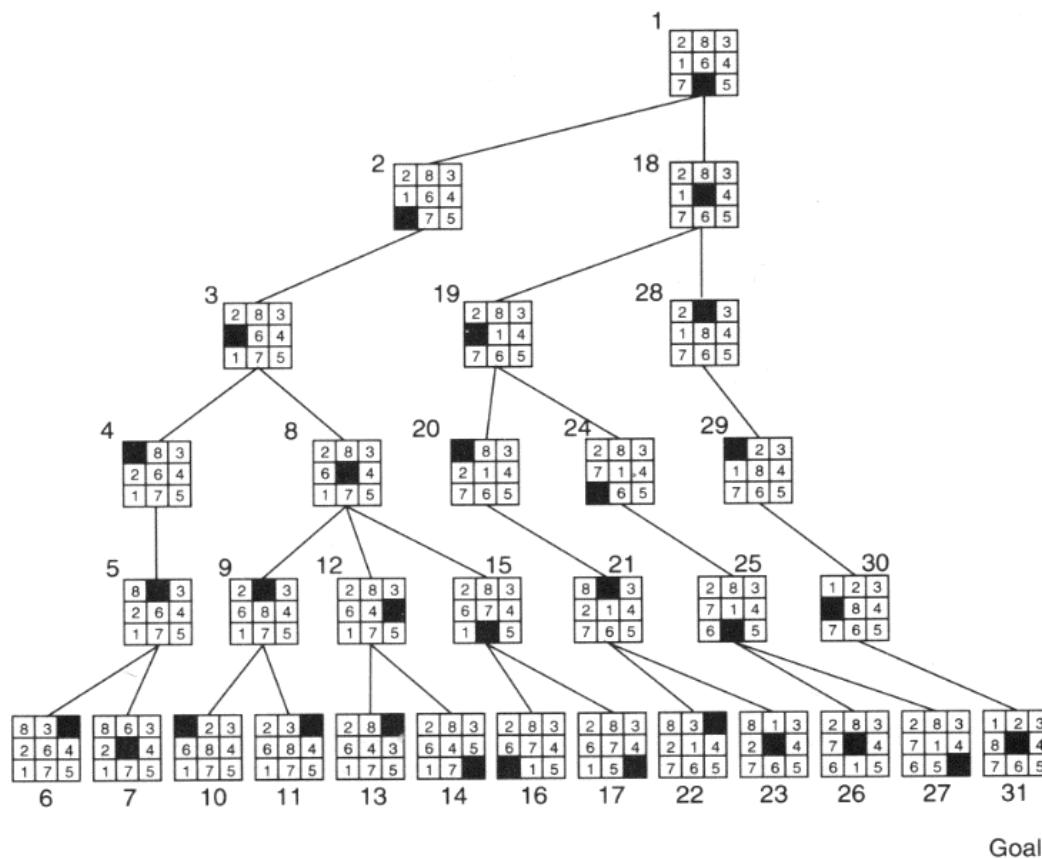


Figure 3.17 Depth-first search of the 8-puzzle with a depth bound of 5.

Breadth-First Search

- Explore search space level by level

open = {START}, **closed** = \emptyset

while (**open** $\neq \emptyset$)

 remove leftmost state from **open** $\rightarrow X$

 if (X is GOAL) return (success)

 else

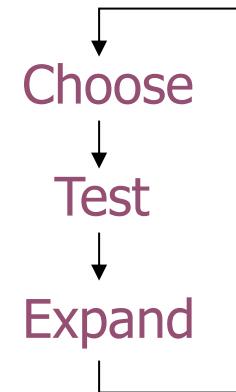
 generate children of X

 put X into **closed**

 eliminate child if it is in open or close

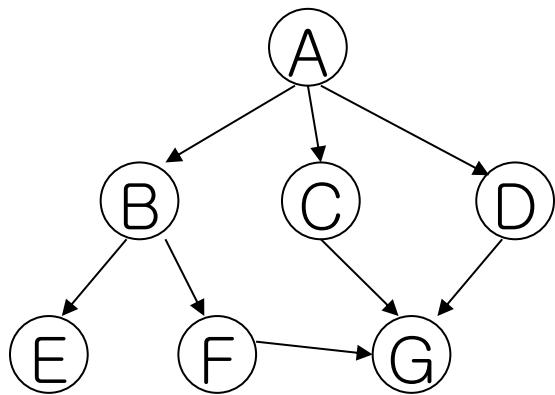
 put remaining children into right of **open**(queue)

 return (fail)



Breadth-First Search

- Example



1. open = {}, closed = {}
2. X = , open = {}, closed = {}
3. X = , open = {}, closed = {}
4. X = , open = {}, closed = {}
5. X = , open = {}, closed = {}
6. X = , open = {}, closed = {}
7. X = , open = {}, closed = {}
8.

Breadth-First Search

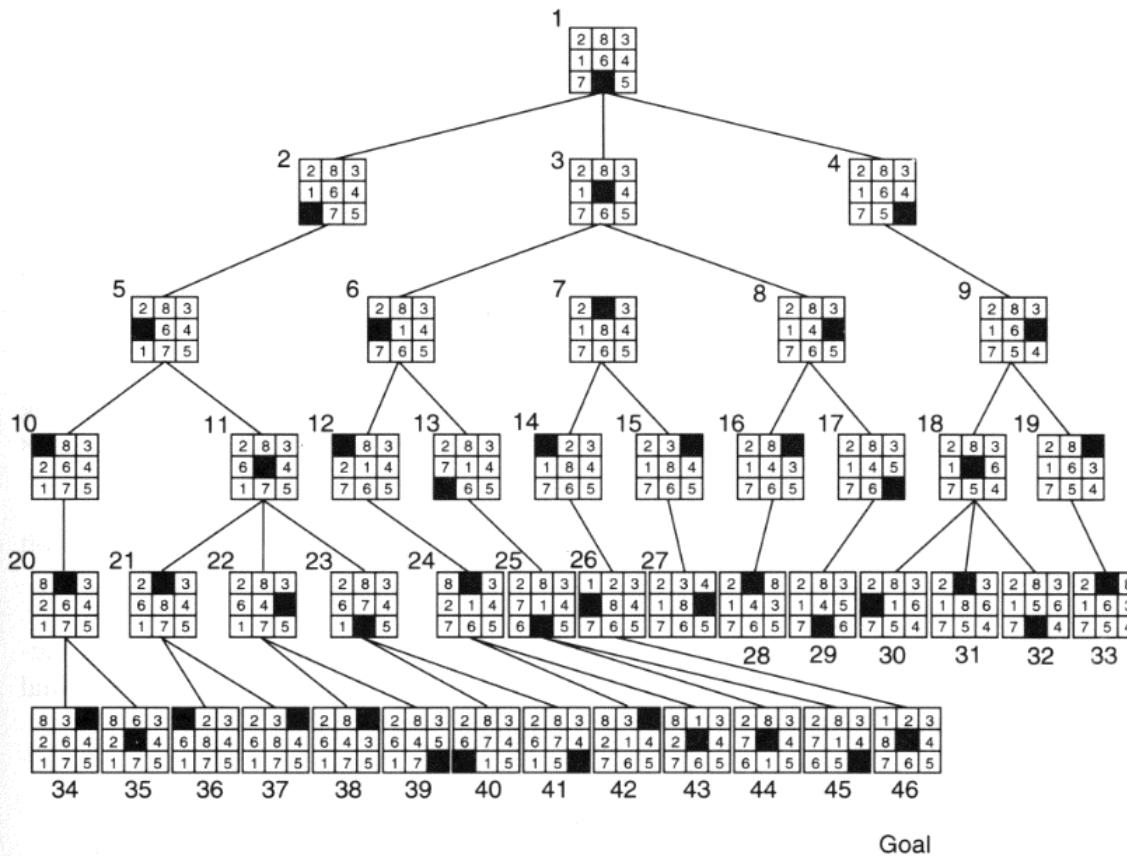


Figure 3.15 Breadth-first search of the 8-puzzle, showing order in which states were removed from open.

Solution Path

- Store parent information with the states
- Example
 - closed = { (A,0), (B,A), (C,A), (D,A), (E,B), (F,B) }
 - Goal = (G, C)

G ← C ← A

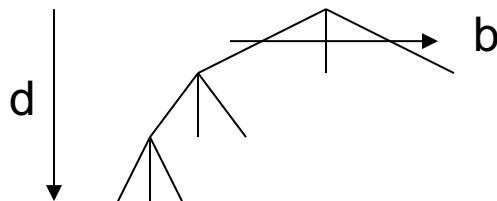


DFS vs. BFS

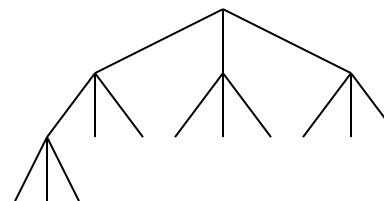
- Let

- b: Average branching factor
- d: Search depth

	OPEN	Time	Space	Optimal
DFS	stack	$O(b^d)$	$O(b \cdot d)$	No
BFS	queue	$O(b^d)$	$O(b^d)$	Yes



- DFS: $b + b + \dots + b$



- BFS: $b + b^2 + \dots + b^{d-1}$

Practice 11

- 두 수의 합(Two Sum)
 - 덧셈하여 타겟을 만들 수 있는 배열의 두 인덱스를 리턴하시오.

```
1 def two_sum(nums, target):  
2     """ Write your code """  
3  
4     nums = [2, 7, 11, 15, 3, 6, 9, 5]  
5     target = 14  
6  
7     two_sum(nums, target)
```

```
[2, 4]
```

Practice 12

- 깊이 우선 탐색(Depth First Search)

```
1 def DFS(graph, start, goal):
2     open_list = []
3     closed_list = []
4
5     #open = {START}, closed = Ø
6     open_list.extend(start)
7     print_list(None, open_list, closed_list)
8
9     while(open_list):
10        #remove leftmost state from open -> X
11        (1)
12
13        #if (X is GOAL) return (success)
14        if (2):
15            print_list(X, open_list, closed_list)
16            return "*** Success ***"
17
18        else:
19            #generate children of X
20            (3)
21
22            #put X into closed
23            (4)
24
25            # eliminate child if it is in open or close
26            (5)
27
28            #put remaining children into left of open(stack)
29            (6)
30
31            #print list
32            print_list(X, open_list, closed_list)
33
34    return "*** Fail ***"
```

```
1 start_state = input("Start State: ")
2 goal_state = input("Goal State: ")
3 print(DFS(deepcopy(graph), start_state, goal_state))

Start State: A
Goal State: G
_____
X = None
open : ['A']
closed : []
_____
X = A
open : ['B', 'C', 'D']
closed : ['A']
_____
X = B
open : ['E', 'F', 'C', 'D']
closed : ['A', 'B']
_____
X = E
open : ['F', 'C', 'D']
closed : ['A', 'B', 'E']
_____
X = F
open : ['G', 'C', 'D']
closed : ['A', 'B', 'E', 'F']
_____
X = G
open : ['C', 'D']
closed : ['A', 'B', 'E', 'F']
*** Success ***
```

Practice 12

• 너비 우선 탐색(Breath First Search)

```
1 def BFS(graph, start, goal):
2     open_list = []
3     closed_list = []
4
5     #open = {START}, closed = Ø
6     open_list.extend(start)
7     print_list(None, open_list, closed_list)
8
9     while(open_list):
10        #remove leftmost state from open -> X
11        (1)
12
13        if (2):
14            print_list(X, open_list, closed_list)
15            return "*** Success ***"
16        else:
17            #generate children of X
18            (3)
19
20            #put X into closed
21            (4)
22
23            # eliminate child if is in open or close
24            (5)
25
26            # put remaining children into right of open(queue)
27            (6)
28
29            #print list
30            print_list(X, open_list, closed_list)
31
32    return "*** Fail ***"
```

```
1 start_state = input("Start State: ")
2 goal_state = input("Goal State: ")
3 print(BFS(deepcopy(graph), start_state, goal_state))
_____
Start State: A
Goal State: G
_____
X = None
open : ['A']
closed : []
_____
X = A
open : ['B', 'C', 'D']
closed : ['A']
_____
X = B
open : ['C', 'D', 'E', 'F']
closed : ['A', 'B']
_____
X = C
open : ['D', 'E', 'F', 'G']
closed : ['A', 'B', 'C']
_____
X = D
open : ['E', 'F', 'G']
closed : ['A', 'B', 'C', 'D']
_____
X = E
open : ['F', 'G']
closed : ['A', 'B', 'C', 'D', 'E']
_____
X = F
open : ['G']
closed : ['A', 'B', 'C', 'D', 'E', 'F']
_____
X = G
open : []
closed : ['A', 'B', 'C', 'D', 'E', 'F']
*** Success ***
```

Thank you!
Q&A