

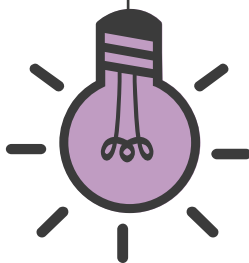
# Python

파이썬



01

## 파이썬 개요 및 설치



# 1. 파이썬이란?

- 1991년 귀도 반 로섬(Guido van Rossum)이 개발한 대화형 프로그래밍 언어이다.
- 파이썬이란 이름은 귀도가 좋아하는 BBC방송 코미디 “Monty Python’s Flying Circus”에서 따온 것이다.



## (1) 파이썬의 특징

- 생산성이 뛰어나다.
  - 간결하면서도 효율적인 프로그램을 빠르게 작성할 수 있다.
  - 다른 언어로 20줄짜리 프로그램을 파이썬으로는 단 몇 줄에 작성할 수 있다.
- 초보자한테 좋은 언어이다.
  - 파이썬은 한 줄의 문장을 입력하고 엔터를 치면 인터프리터(해석기)가 이것을 해석해서 바로 실행한다.
  - 파이썬은 실행 전에 컴파일 할 필요가 없다.
  - 자신이 작성한 문장의 결과를 즉시 볼 수 있어 초보 프로그래머에게 아주 바람직하다.
- 문법이 쉬워서 코드를 보면 직관적으로 알 수 있는 부분이 많다.

### 사용 예

```
if "사과" in ["딸기", "바나나", "포도", "사과"];  
    print("사과가 있습니다")
```

- 다양한 플랫폼에서 사용할 수 있고, 라이브러리가 풍부해서 여러 산업계에서 이용이 증가하고 있다.

## (2) 파이썬의 적용분야

- 시스템관리
- GUI
- 인터넷 프로그래밍
- DB프로그래밍
- 각종 텍스트 프로세싱
- 분산처리
- 수치연산, 그래픽스

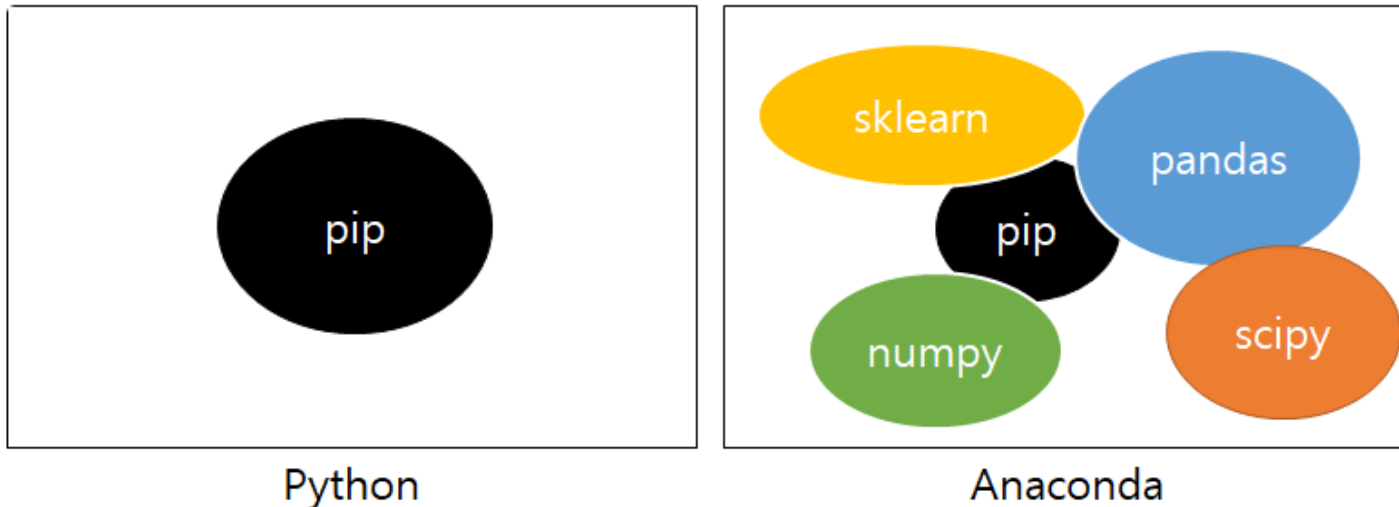
## 2. Python 프로그램 설치 방법 (2가지)

- Python 설치

- 파이썬 공식 홈페이지에서 받을 수 있으며, pip 툴만을 포함하고 있다.
- 필요한 패키지나 라이브러리 등을 설치할 때 모두 수동으로 해주어야 한다.

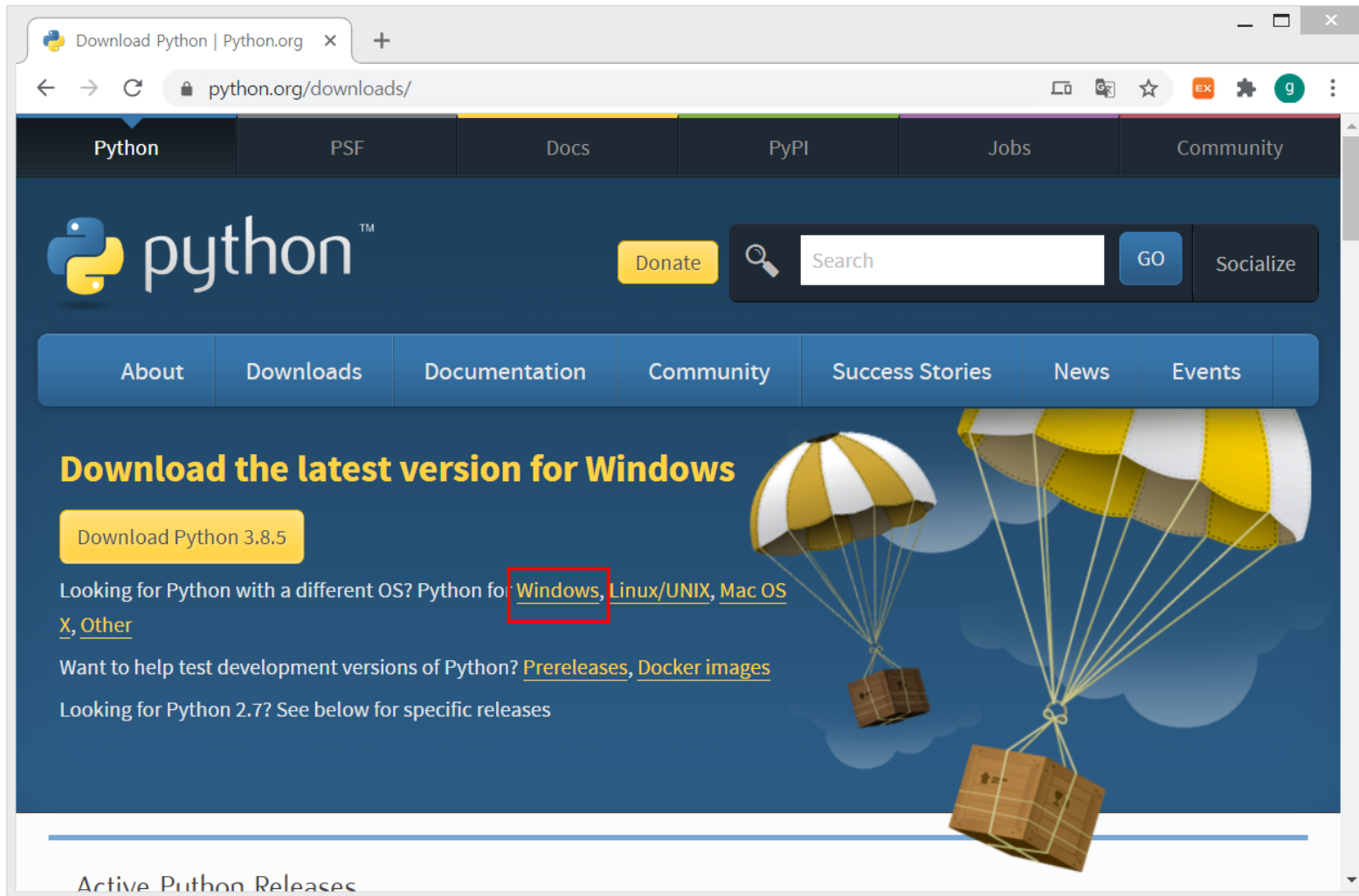
- Anaconda 설치

- Python 기본 패키지에 각종 수학/과학 라이브러리들을 같이 패키징해서 배포하는 버전이다.
- 요즘 유행하는 인공지능이나 빅데이터 관련 개발을 할 경우에는 결국 아나콘다에 포함된 라이브러리들을 설치할 가능성이 높기 때문에 처음부터 아나콘다를 설치하는 것이 더 유리하다.
- 일일이 라이브러리들을 설치하다 보면 의존성 문제 등이 발생할 수도 있기 때문이다.
- 아나콘다에 포함된 툴들로는 대표적으로 panda, numpy, scipy, sklearn, matplotlib, Jupyter Notebook 등이 있다.



## (1) Python을 이용한 설치

- 파이썬은 윈도우, 유닉스, 리눅스, 매킨토시에서 개발이 가능하다.
- 다운로드 → <https://www.python.org/downloads/>



Python Releases for Windows | x +

python.org/downloads/windows/

# Python Releases for Windows

- [Latest Python 3 Release - Python 3.8.5](#)
- [Latest Python 2 Release - Python 2.7.18](#)

## Stable Releases

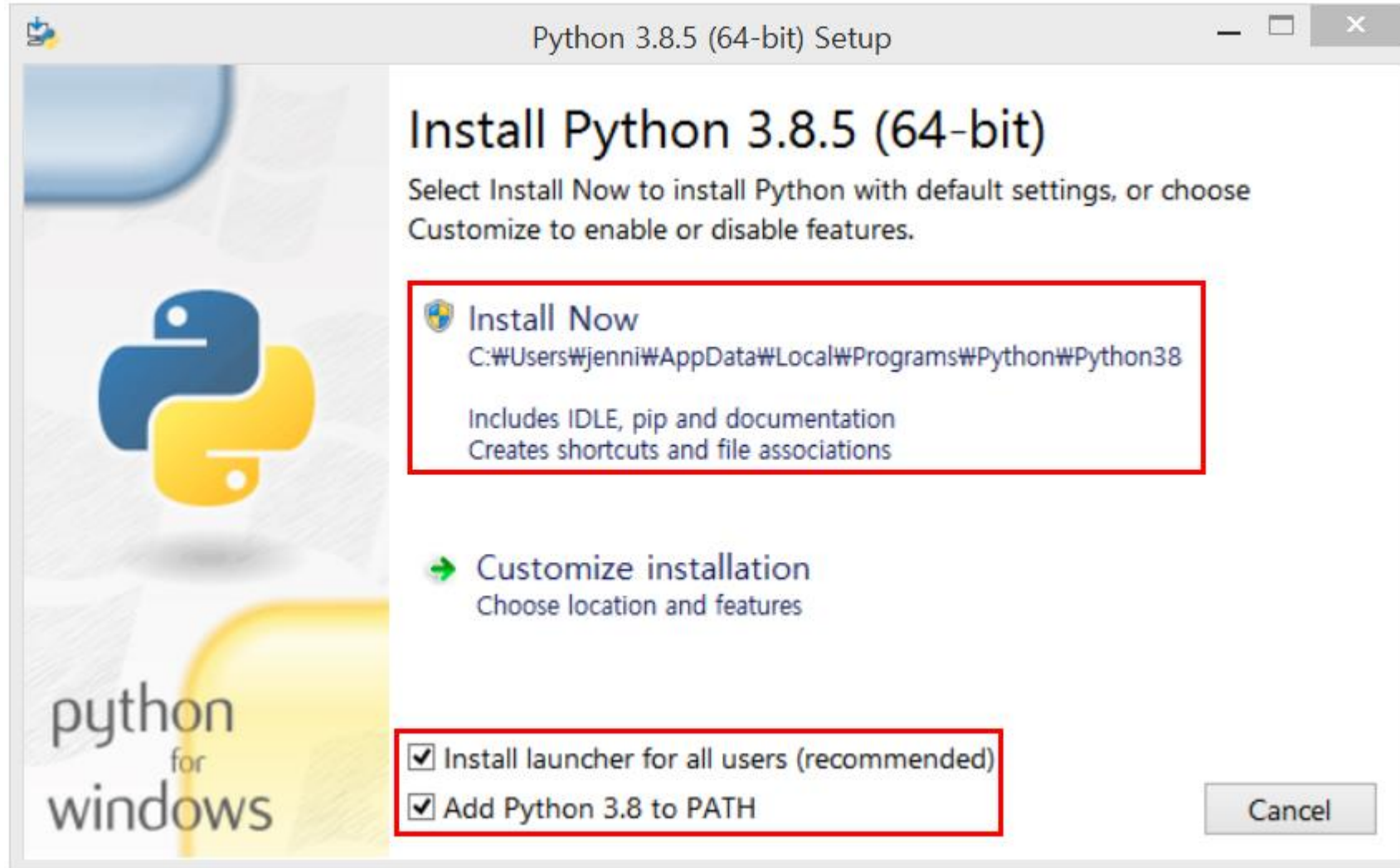
- [Python 3.8.5 - July 20, 2020](#)  
**Note that Python 3.8.5 cannot be used on Windows XP or earlier.**
  - Download [Windows help file](#)
  - Download [Windows x86-64 embeddable zip file](#)
  - **Download [Windows x86-64 executable installer](#)**
  - Download [Windows x86-64 web-based installer](#)
  - Download [Windows x86 embeddable zip file](#)
  - Download [Windows x86 executable installer](#)
  - Download [Windows x86 web-based installer](#)
- [Python 3.8.4 - July 13, 2020](#)  
**Note that Python 3.8.4 cannot be used on Windows XP or earlier.**
  - Download [Windows help file](#)

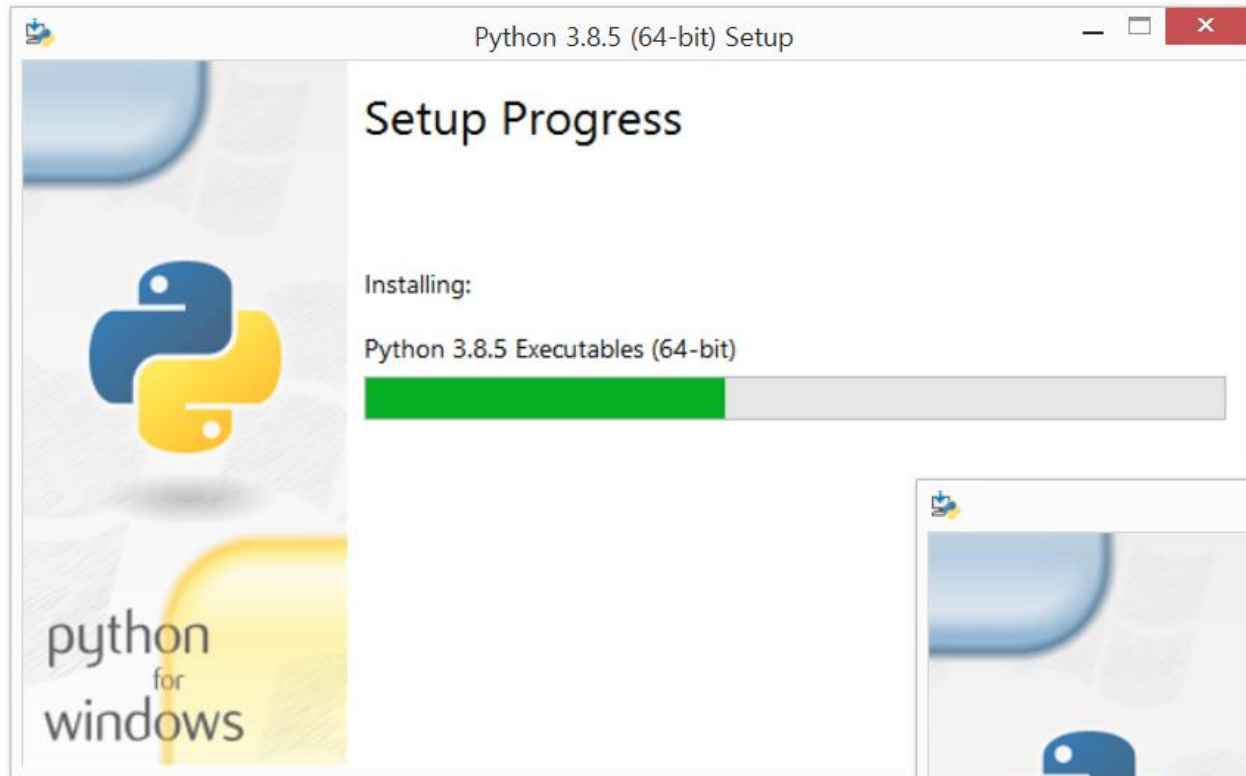
## Pre-releases

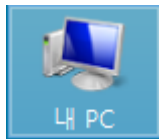
- [Python 3.9.0b5 - July 20, 2020](#)
  - Download [Windows help file](#)
  - Download [Windows x86-64 embeddable zip file](#)
  - Download [Windows x86-64 executable installer](#)
  - Download [Windows x86-64 web-based installer](#)
  - Download [Windows x86 embeddable zip file](#)
  - Download [Windows x86 executable installer](#)
  - Download [Windows x86 web-based installer](#)
- [Python 3.9.0b4 - July 3, 2020](#)
  - Download [Windows help file](#)
  - Download [Windows x86-64 embeddable zip file](#)
  - Download [Windows x86-64 executable installer](#)
  - Download [Windows x86-64 web-based installer](#)



## 1) python-3.8.5-amd64.exe 로 파이썬 설치



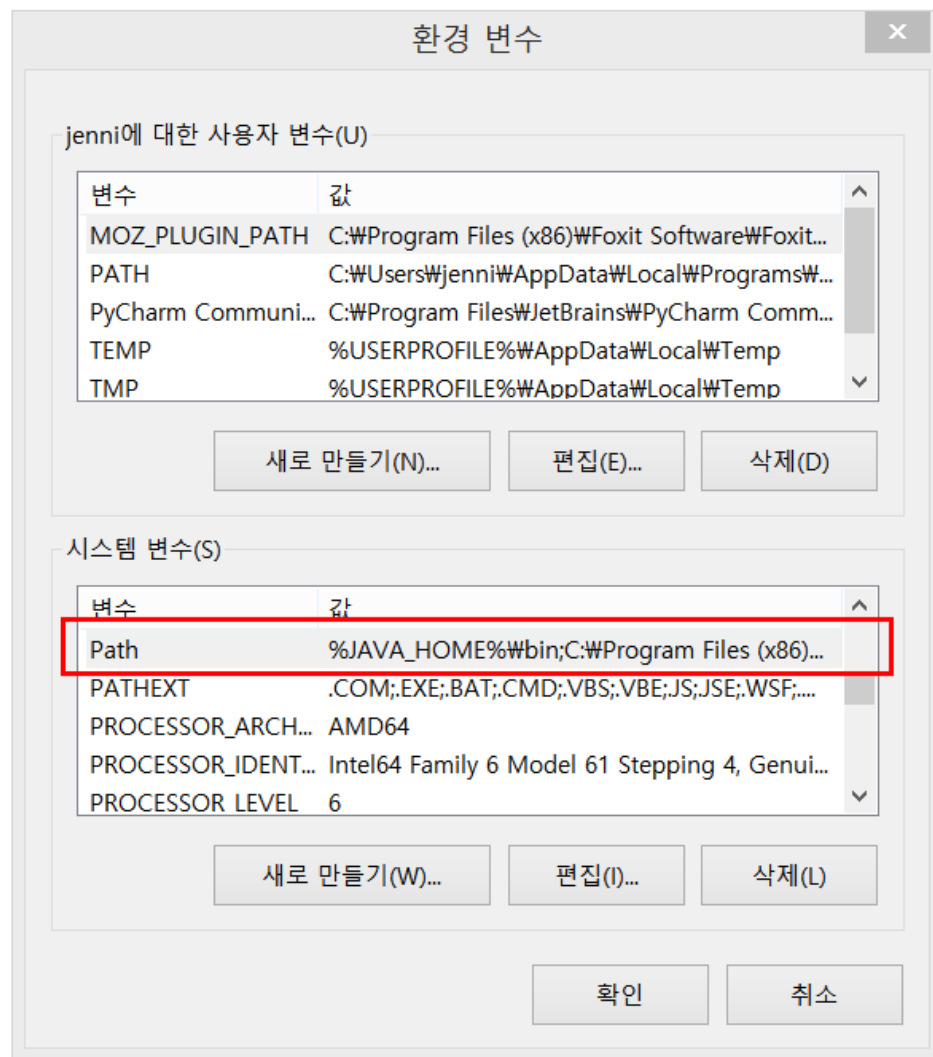
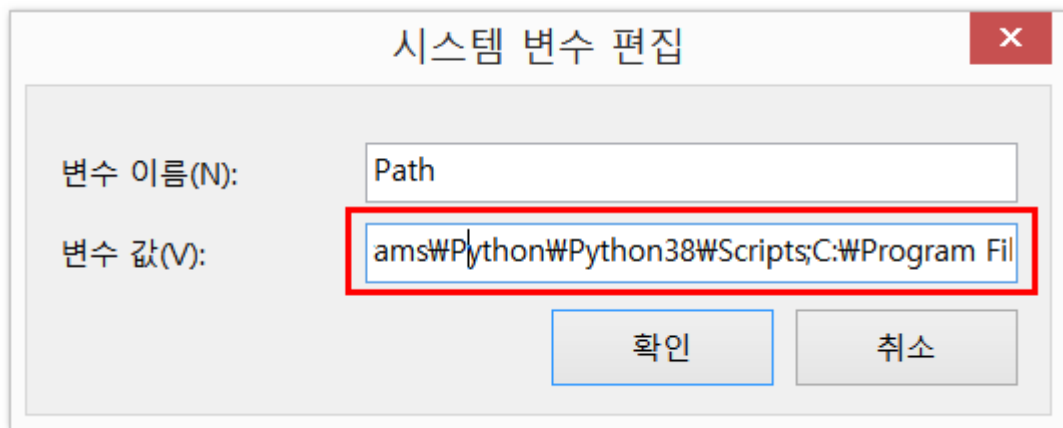




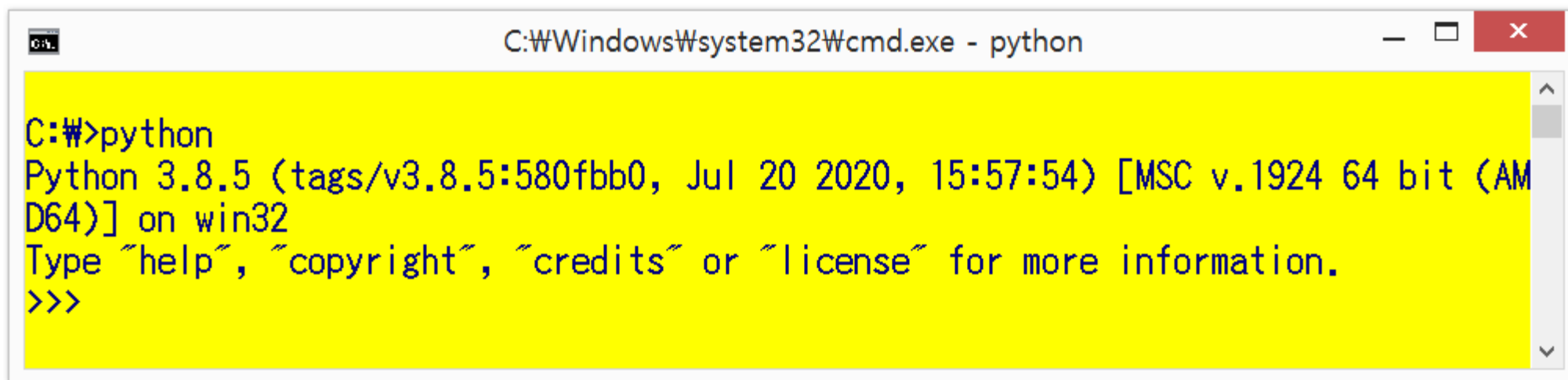
→ 속성 → 고급 시스템 설정 → 환경변수 → 시스템변수 에서 path를 확인한다.

설치 경로:

C:\Users\jenni\AppData\Local\Programs\Python\Python38;C:\Users\jenni\AppData\Local\Programs\Python\Python38\Scripts;



- 콘솔 창에서 간단한 테스트

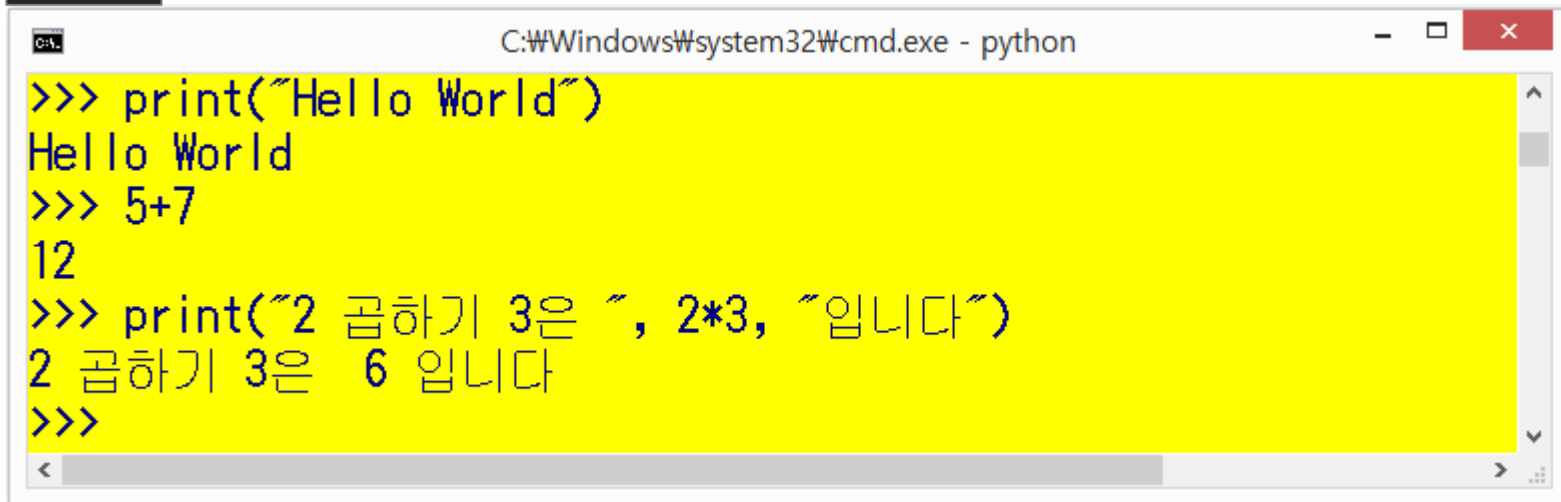


```
C:\Windows\system32\cmd.exe - python

C:\#>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- 파이썬 셸(python shell)이라고 하며 한번 에 하나의 명령어가 실행 되어 결과가 보여진다.

#### 사용 예

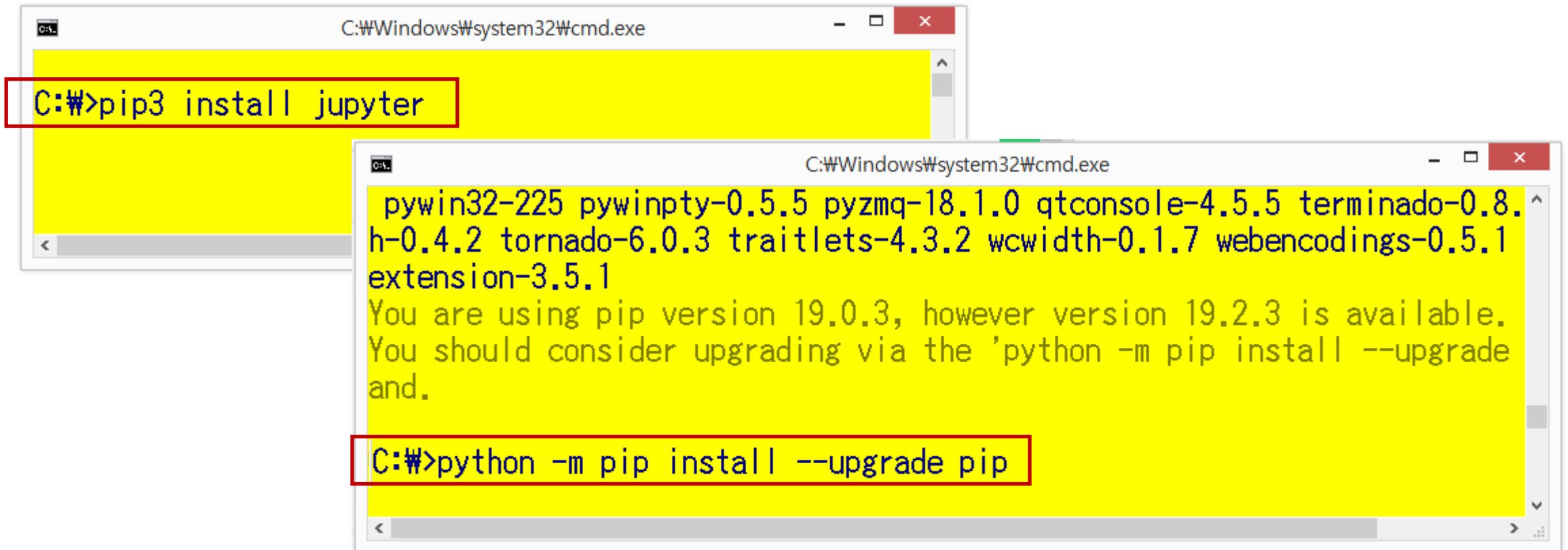


```
C:\Windows\system32\cmd.exe - python
>>> print("Hello World")
Hello World
>>> 5+7
12
>>> print("2 곱하기 3은 ~, 2*3, ~입니다")
2 곱하기 3은 6 입니다
>>>
```

## 2) 파이썬 개발 툴 설치

- 개발 툴의 종류는 Visual Studio 2019, 파이 참, Jupyter Notebook 등이 있다
- Jupyter Notebook을 사용할 경우에는 다음과 같이 설정 한다.

 → cmd



The image shows two overlapping Windows Command Prompt windows. The top window has a yellow background and shows the command `C:\#>pip3 install jupyter` entered at the prompt. The bottom window also has a yellow background and shows the output of the command: a list of installed packages (pywin32-225, pywinpty-0.5.5, pyzmq-18.1.0, qtconsole-4.5.5, terminado-0.8.1, h-0.4.2, tornado-6.0.3, traitlets-4.3.2, wcwidth-0.1.7, webencodings-0.5.1, extension-3.5.1), a message about upgrading pip from 19.0.3 to 19.2.3, and the command `C:\#>python -m pip install --upgrade pip` entered at the prompt.

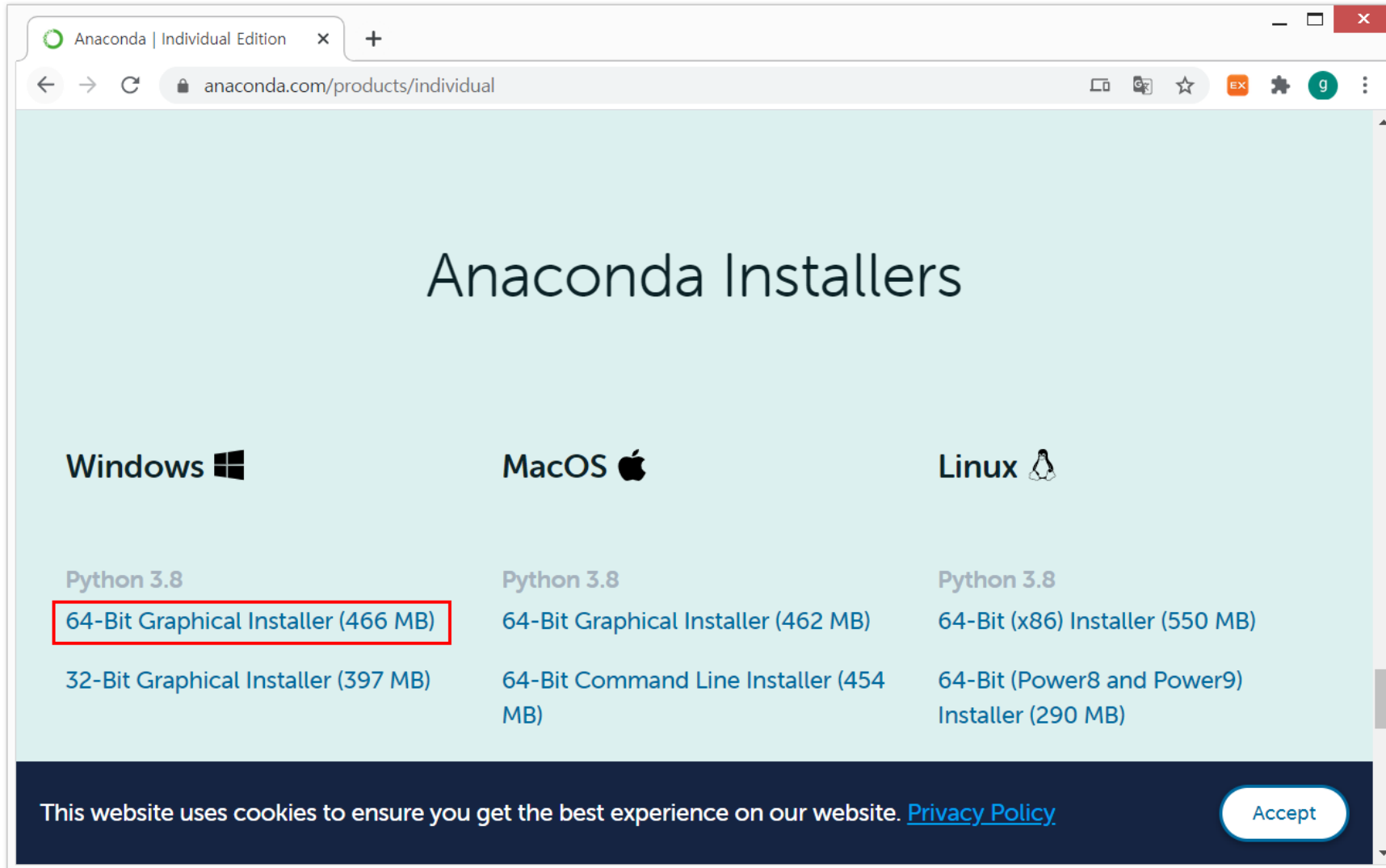
```
C:\#>pip3 install jupyter
```

```
pywin32-225 pywinpty-0.5.5 pyzmq-18.1.0 qtconsole-4.5.5 terminado-0.8.1
h-0.4.2 tornado-6.0.3 traitlets-4.3.2 wcwidth-0.1.7 webencodings-0.5.1
extension-3.5.1
You are using pip version 19.0.3, however version 19.2.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade
and.
C:\#>python -m pip install --upgrade pip
```



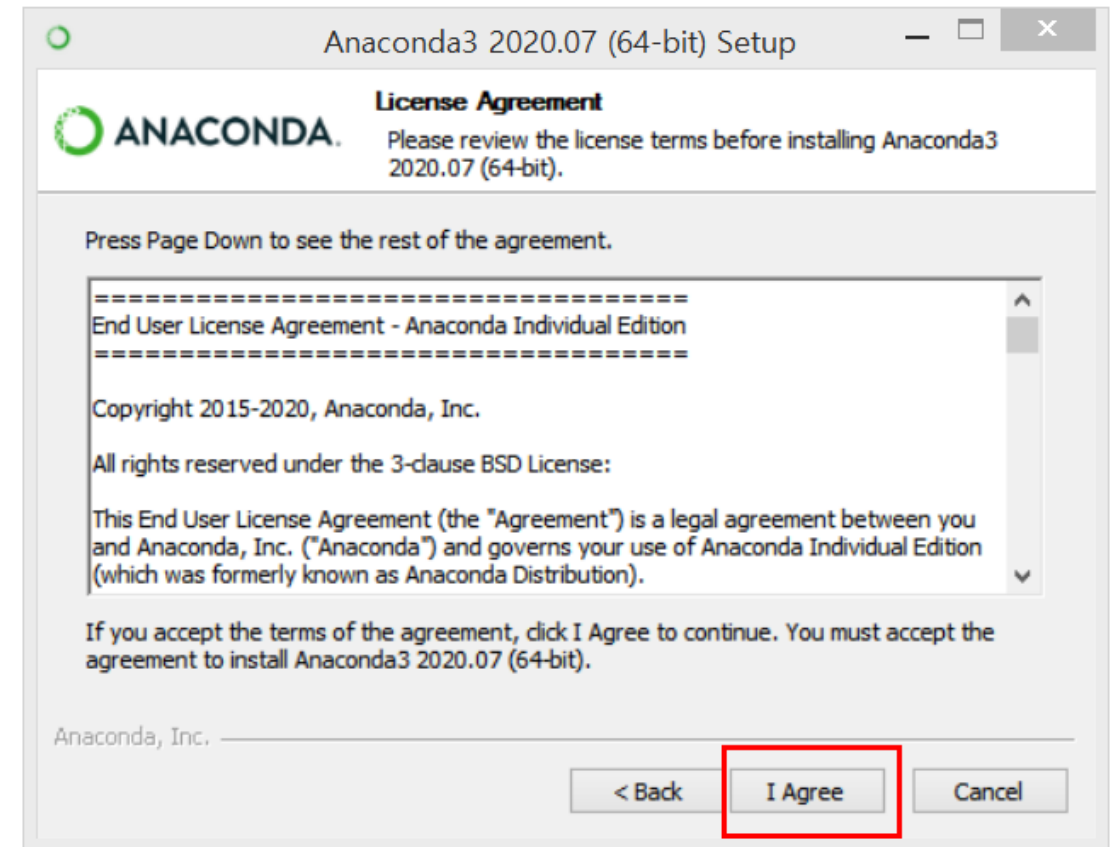
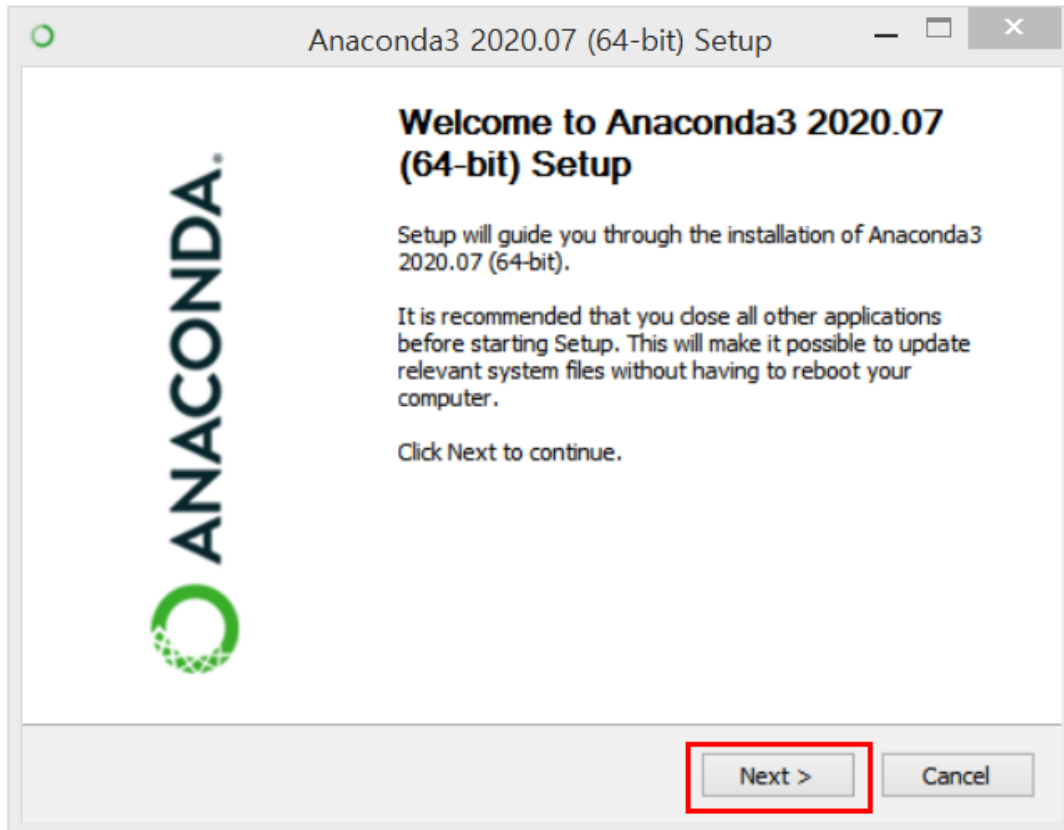
## (2) Anaconda를 이용한 설치

- 다운로드 → <https://www.anaconda.com/download/>

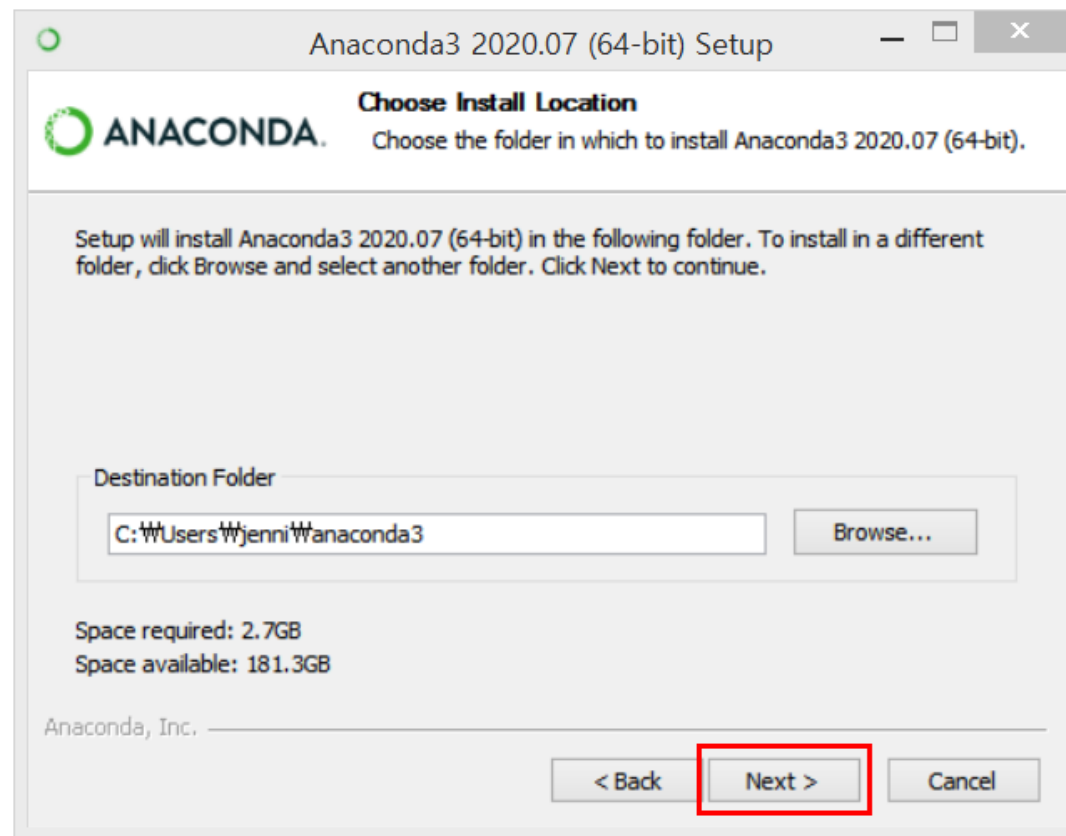
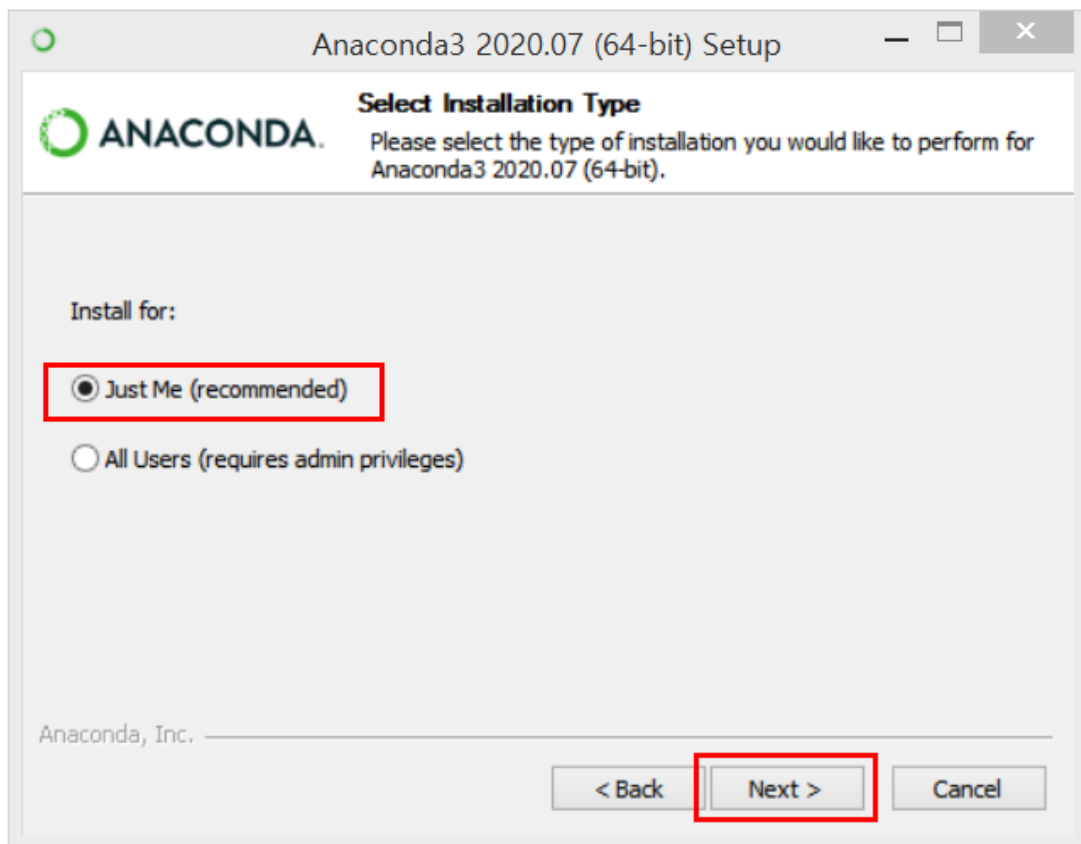




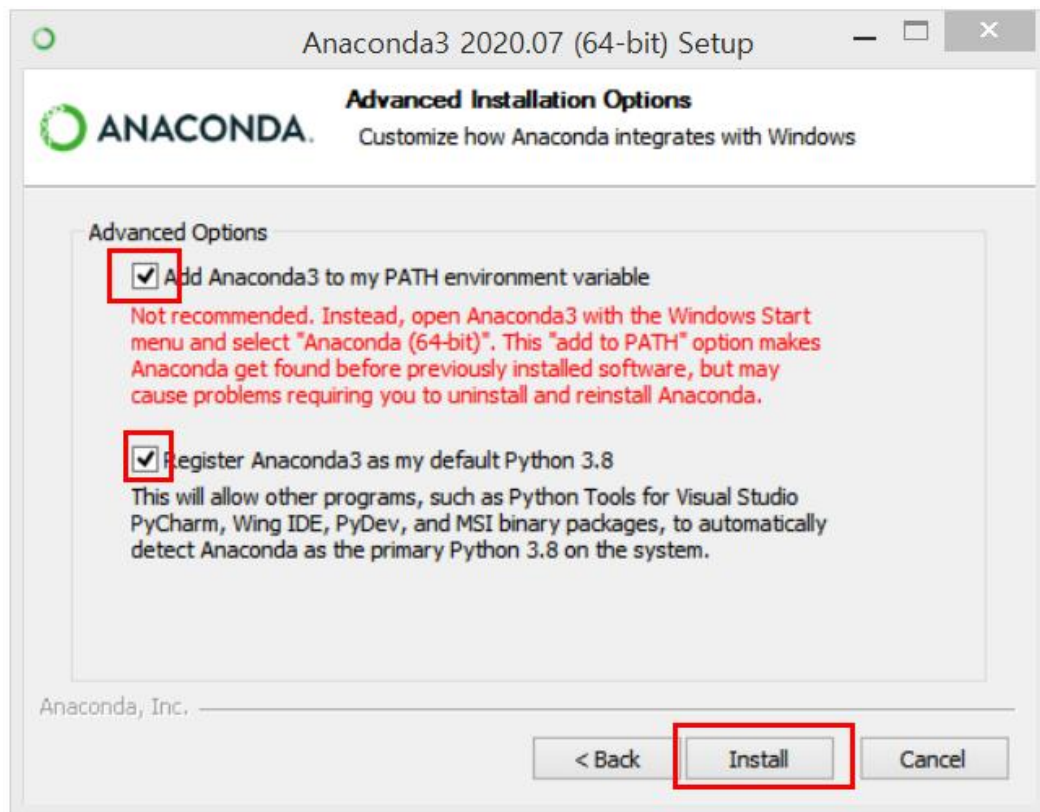
- Anaconda3-2020.07-Windows-x86\_64.exe 실행



- All Users를 선택할 경우 Just ME와 설치되는 경로가 다르며 패키지를 설치 또는 삭제 업그레이드 시 CMD창을 관리자 권한으로 열어 실행해야 하는 경우가 있다.



- 2가지 Advanced Options(고급옵션)



- Adding Anaconda to my PATH environment variable(기본값은 체크 해제)

➤ 아나콘다 측 설명을 보면 "PATH 환경 변수에 Anaconda를 추가할지 여부를 선택한다. PATH 환경 변수에 다른 소프트웨어를 방해 할 수 있으므로 Anaconda 를 추가하지 않는 것이 좋다 . 대신 시작 메뉴에서 Anaconda Navigator 또는 Anaconda Prompt를 열어 Anaconda 소프트웨어를 사용한다."

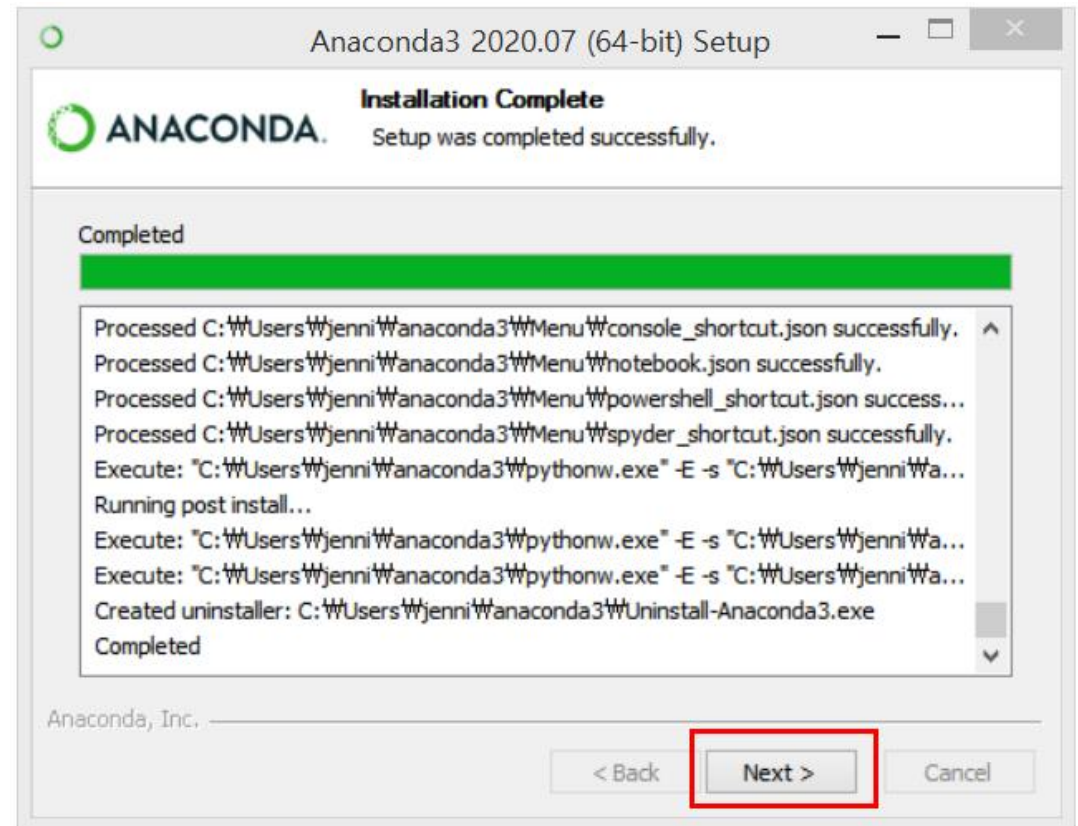
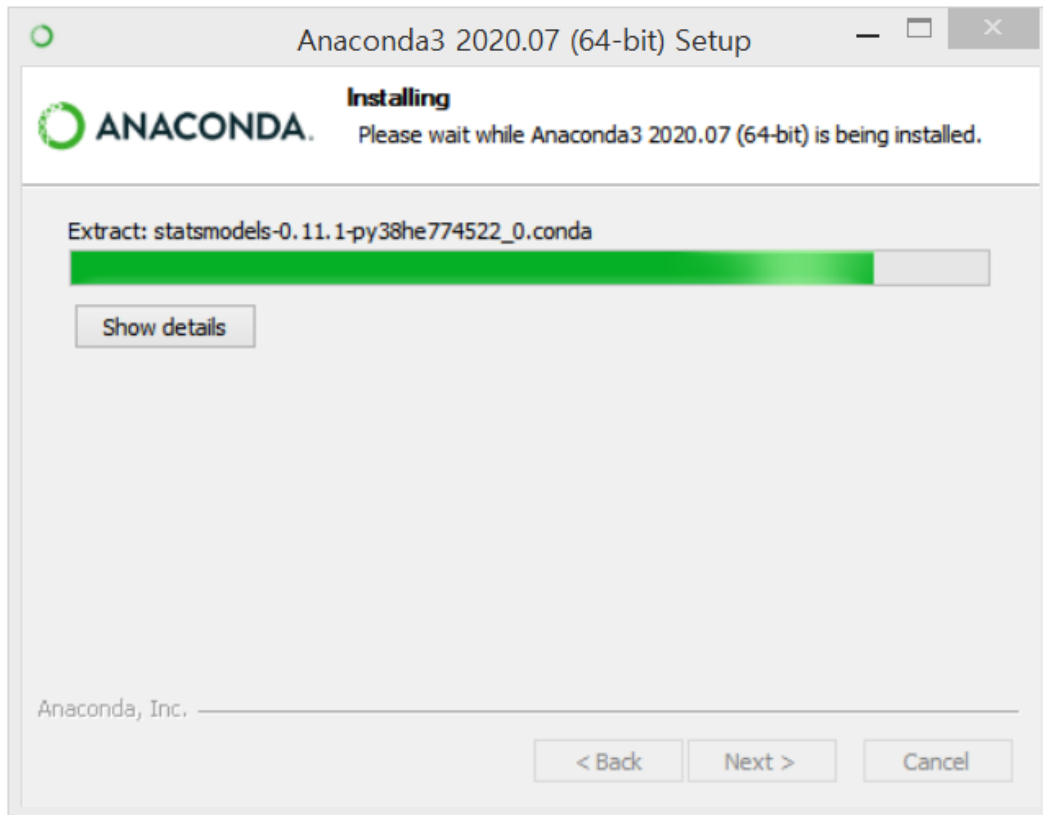
➤ 환경변수에 추가할지를 선택 아나콘다 외에 다른 파이썬 인터프리터를 환경변수에 등록해서 사용 한다면 체크 해제 하고 아나콘다만을 사용하는 경우 또는 아나콘다가 주력일 경우로 윈도우 CMD창에서 파이썬을 실행할 경우 선택한다.

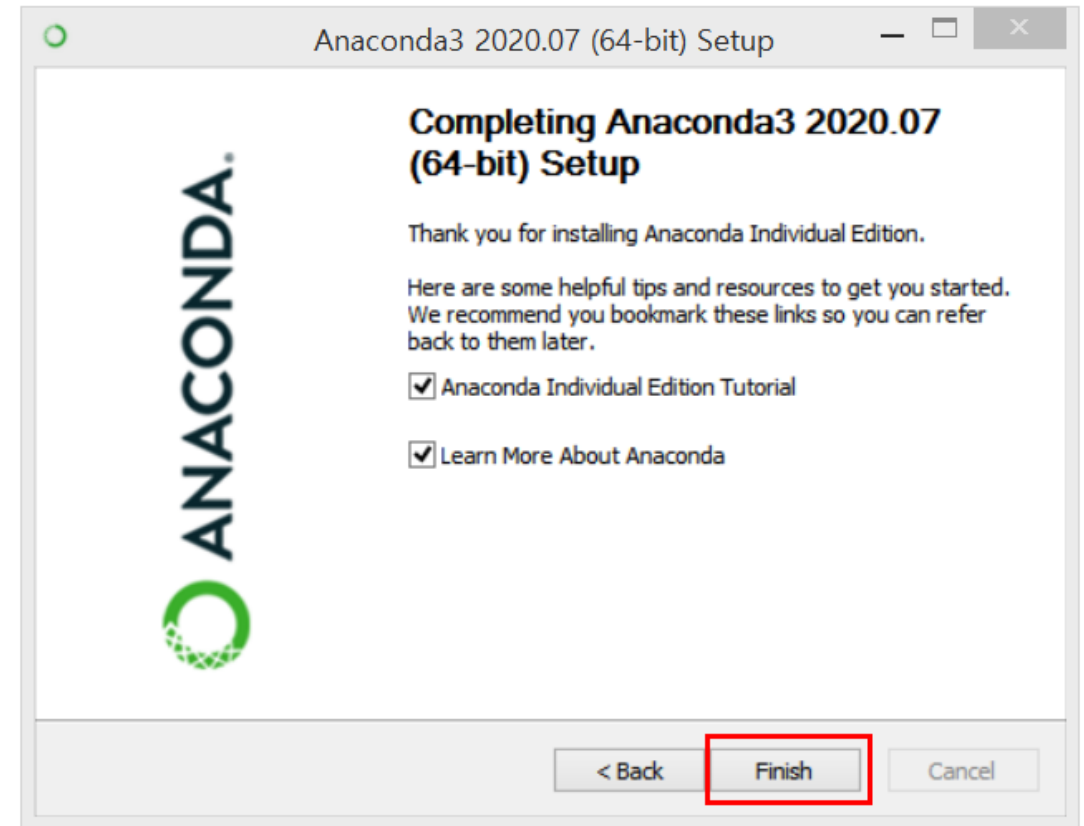
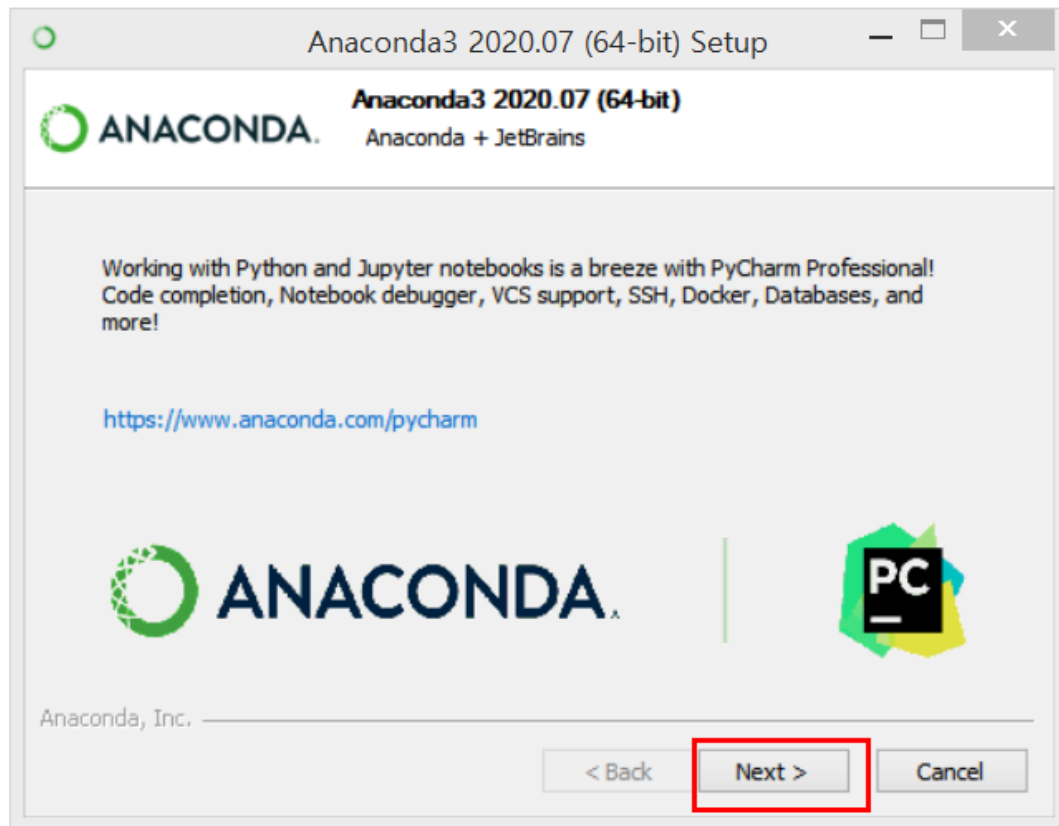
➤ 선택할 경우 윈도우 CMD창 경로와 상관없이 아나콘다를 파이썬으로 인식한다.

- Register Anaconda as my default Python 3.8(기본값 선택)

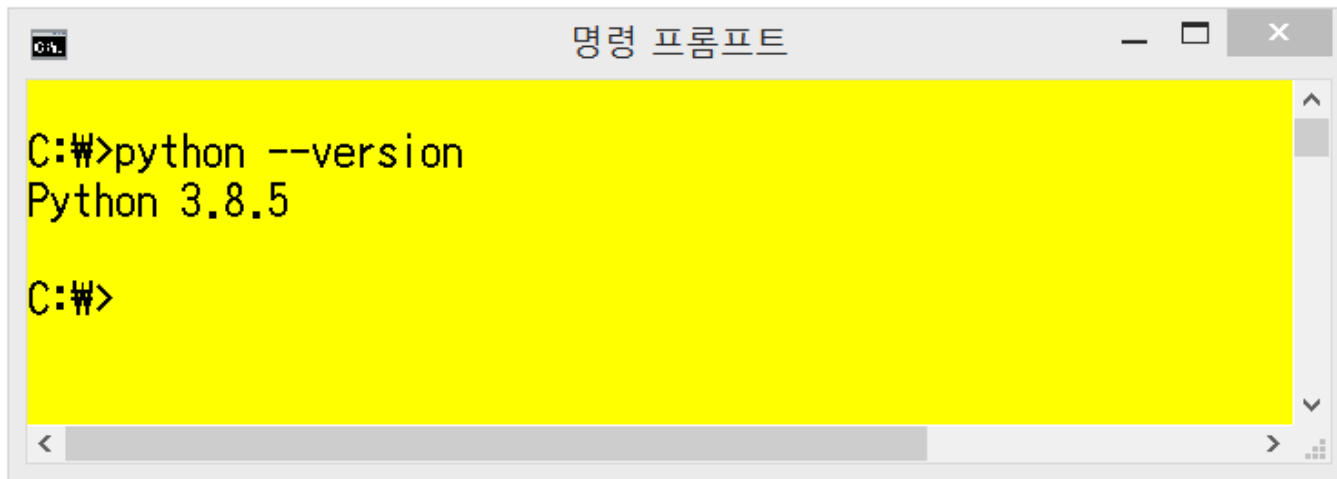
➤ 아나콘다를 기본 파이썬으로 등록할지 여부를 선택한다.

➤ 개발 도구나 에디터에서 아나콘다를 파이썬으로 인식한다.





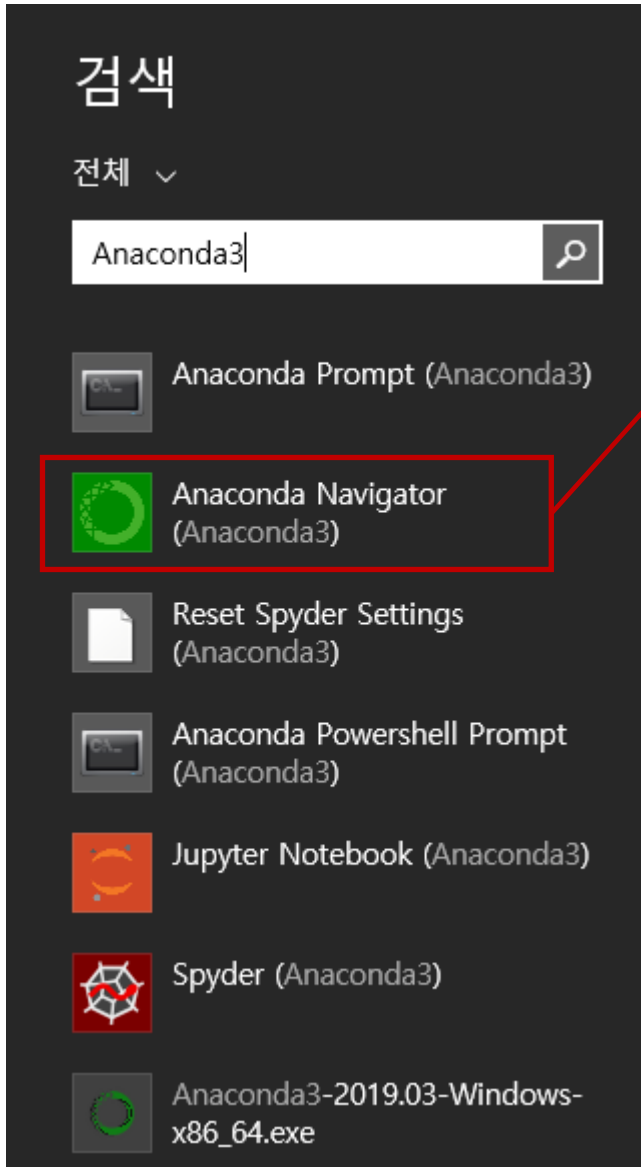
- 실행 → cmd 에서 버전 확인



A screenshot of a Windows Command Prompt window. The title bar at the top reads '명령 프롬프트' (Command Prompt) and includes standard window controls (minimize, maximize, close). The main area has a yellow background and displays the command 'C:\W>python --version' followed by the output 'Python 3.8.5'. Below this, the prompt 'C:\W>' is shown again. A vertical scrollbar is on the right side of the text area, and a horizontal scrollbar is at the bottom.

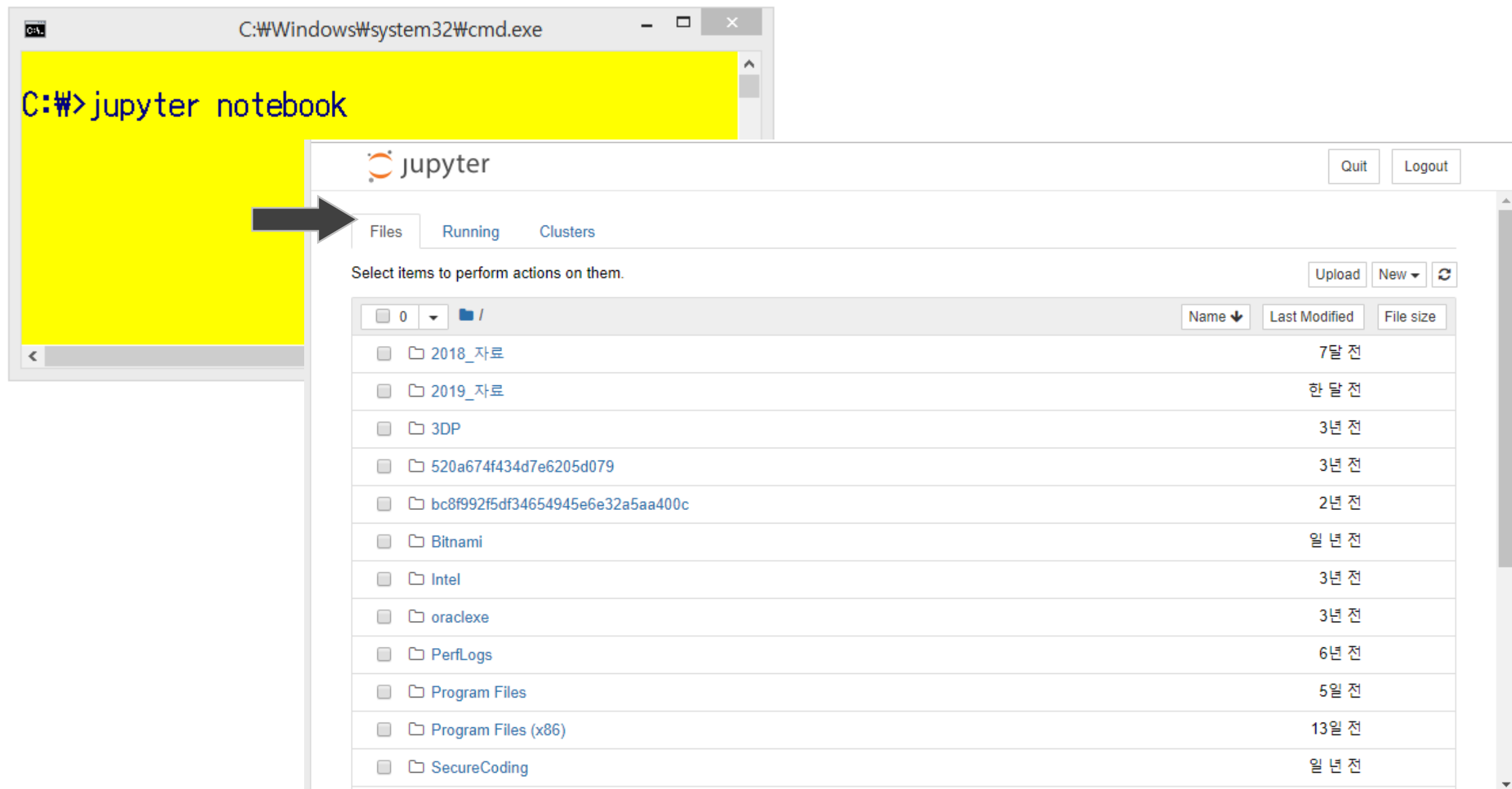
```
C:\W>python --version
Python 3.8.5

C:\W>
```



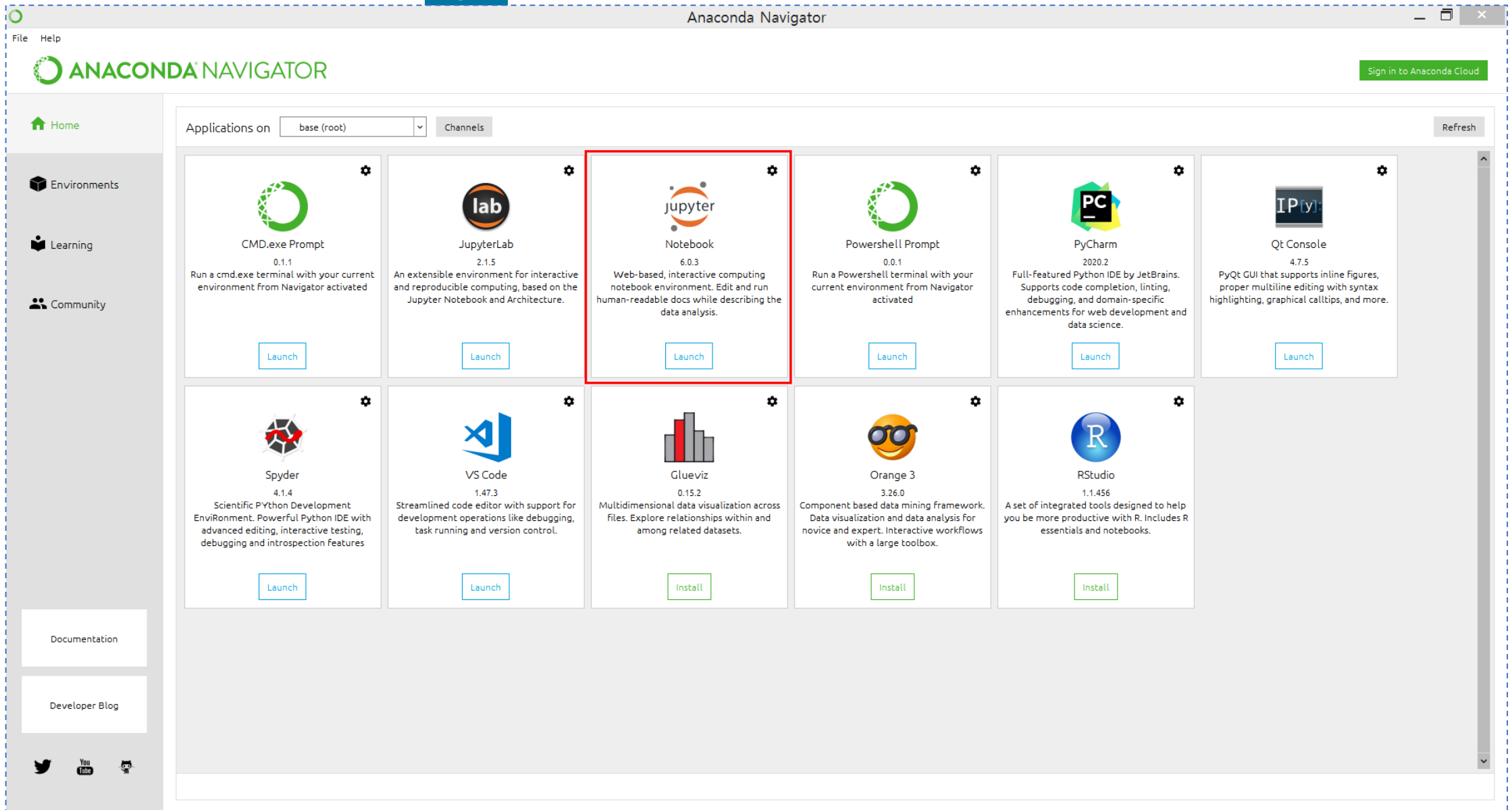
바탕화면에 단축 아이콘 만들기

- 실행 방법1) cmd → jupyter notebook

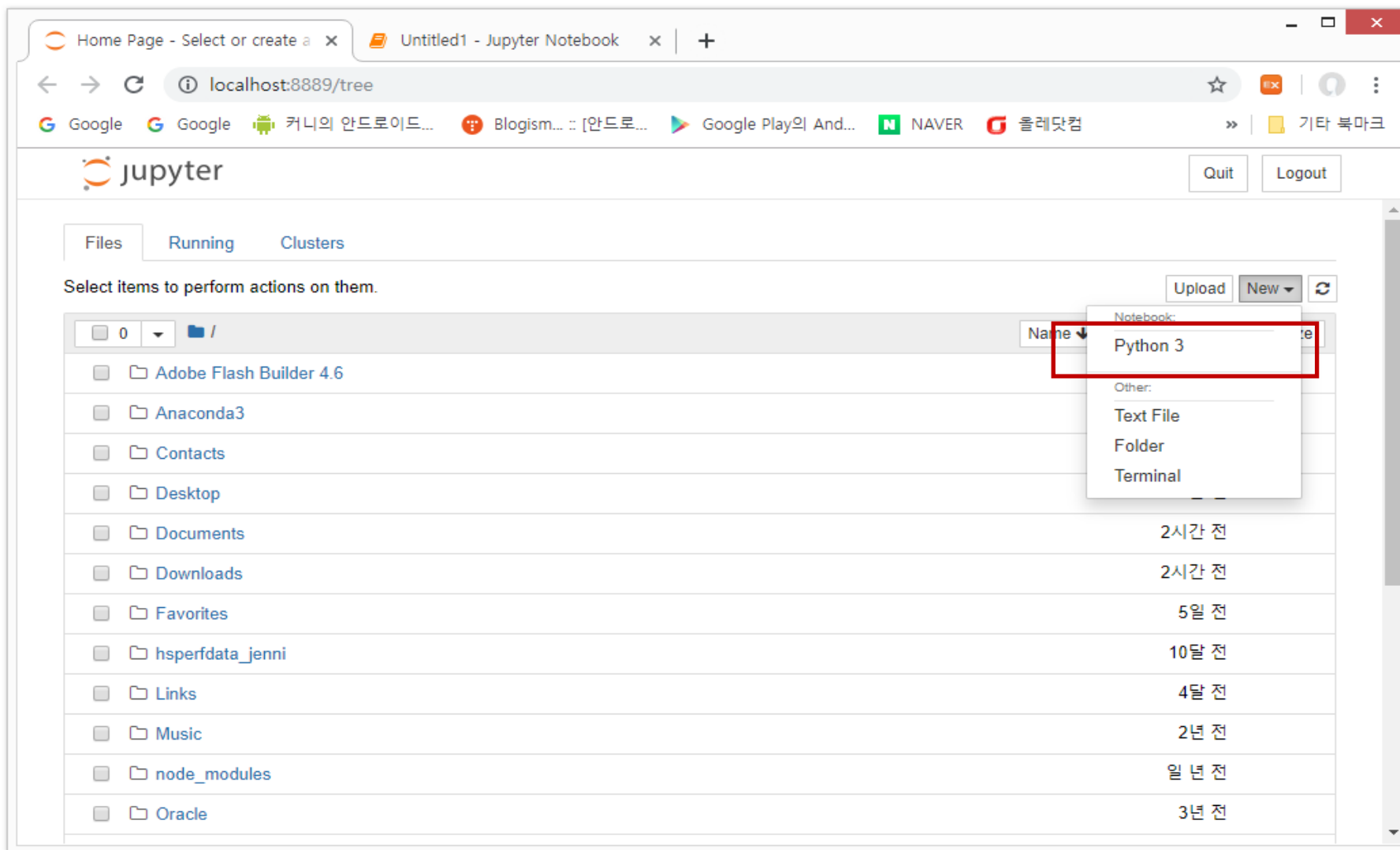




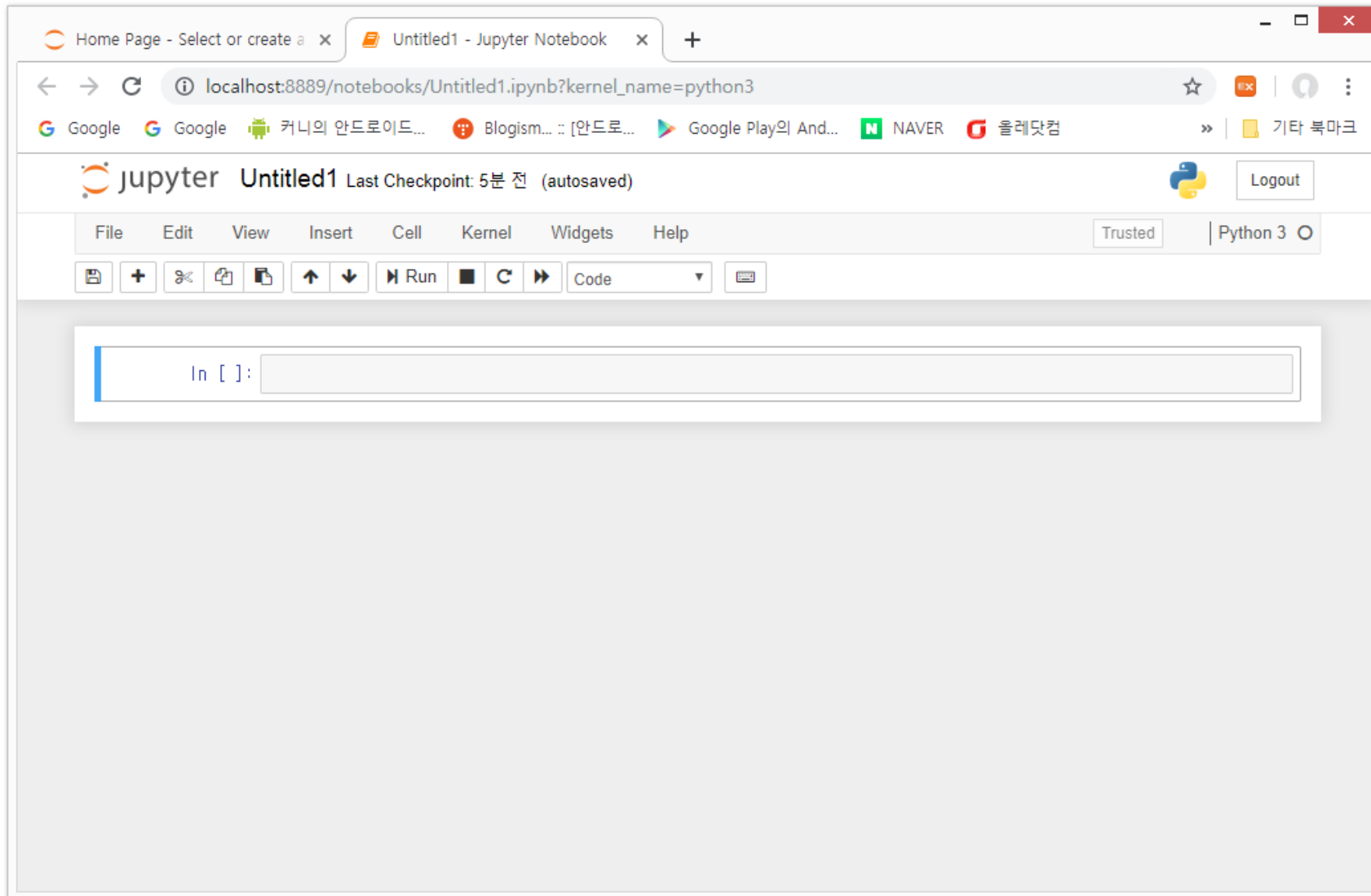
- 실행 방법2) 바탕화면 →  → 실행



- New → Python3(선택)



- jupyter notebook 이 실행 되었다.



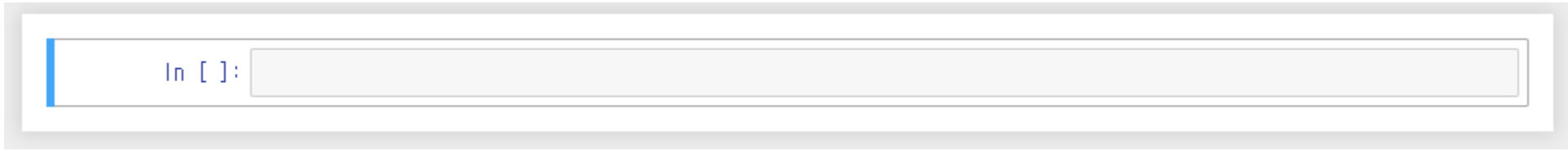
### (3) Jupyter notebook 사용 방법

- jupyter notebook 단축키
  - jupyter notebook의 코드 편집 모드는 2가지가 있다.
  - 셀 편집 모드는 셀을 편집할 때 사용한다. 코드 편집 모드는 셀 안의 내용을 편집 할 때 사용한다.

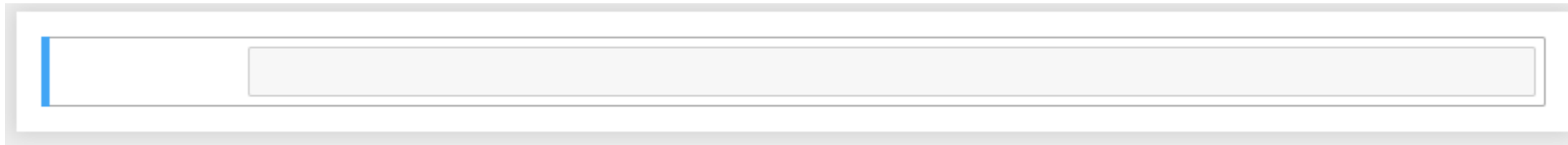
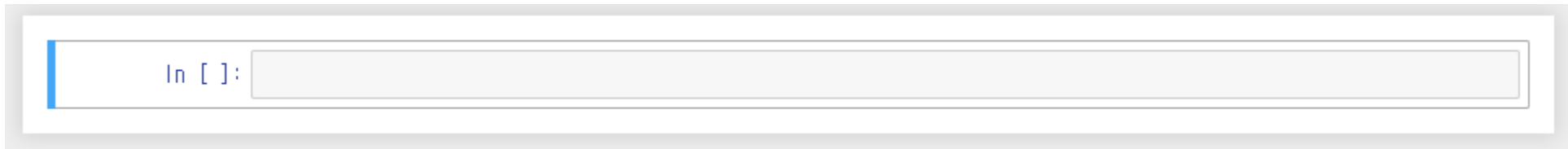
모드	내용	단축키
셀 선택 모드 (Command Mode)	셀 추가	위에 셀 추가: [ a ]    아래 셀 추가: [ b ]
	셀 삭제	[ dd ]
	복사 / 잘라내기	잘라내기: [ x ]    복사하기: [ c ]    붙여넣기: [ p ]
	아래 셀과 합치기	[Shift] + [ m ]
	셀 타입 변경	마크 다운: [ m ]    코드: [ y ]
	파일 저장	[ctrl] + [ s ] 또는 [ s ]
	코드 편집 모드	[enter]

모드	내용	단축키
코드 편집 모드 (Edit Mode)	실행	셀 실행: [ctrl] + [enter] 실행 후 다음 셀로 이동(없으면 새로 추가): [Shift] + [enter] 실행 취소: [ctrl] + [ z ] 셀 다시 실행: [ctrl] + [ y ]
	커서위치에서 셀 둘로 나누기	[Shift] + [ctrl] + [ - ]
	셀 선택 모드 가기	[Esc] 또는 [ctrl + m]
	주석처리	[ctrl] + [ / ]

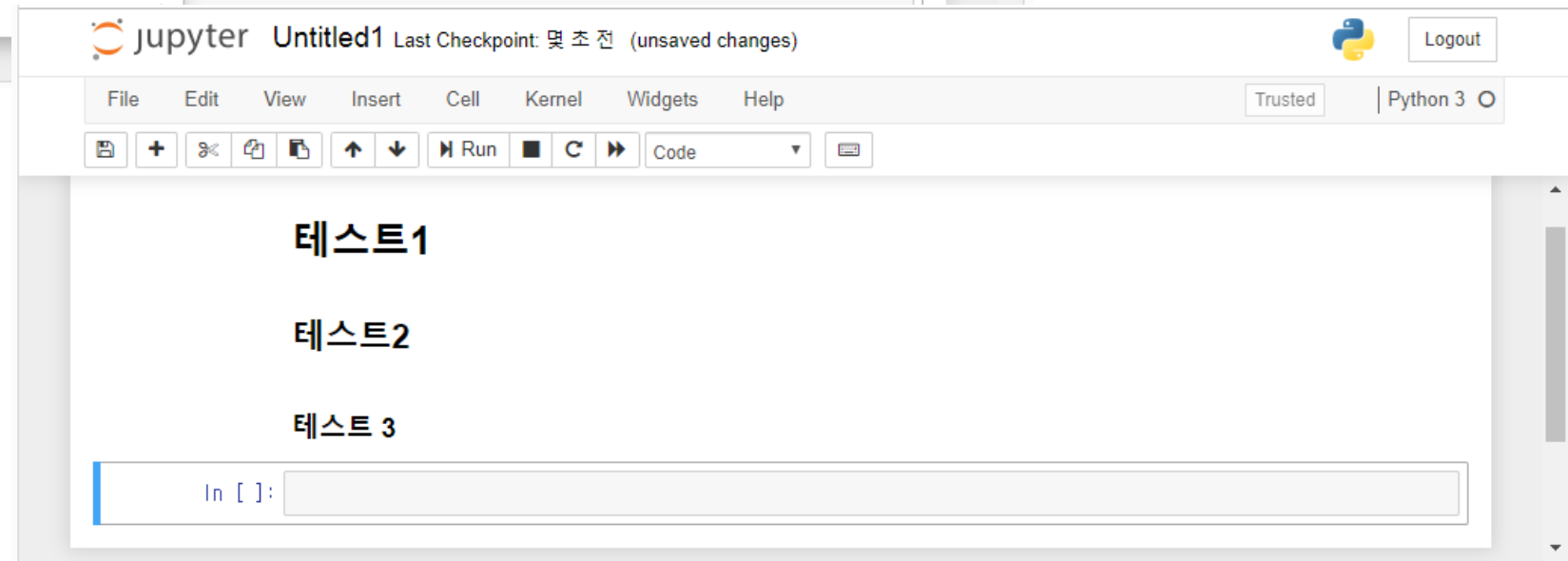
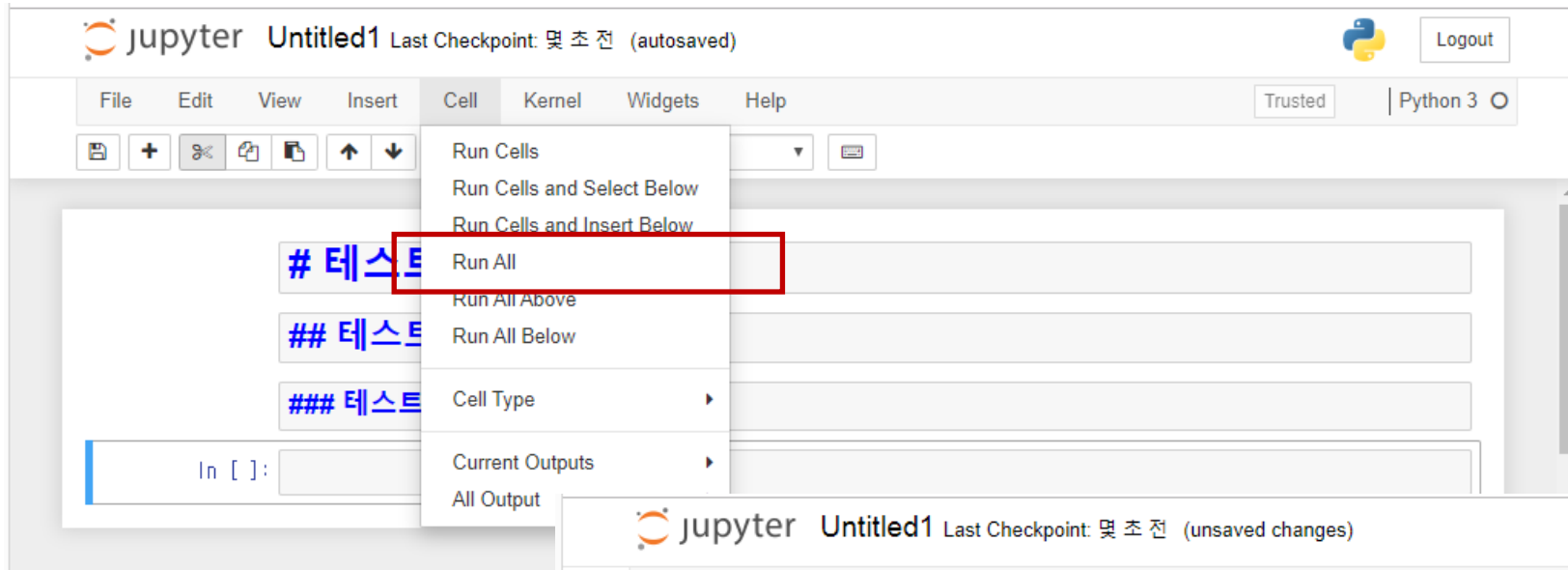
- 하늘색 모드 → m 입력 → Markdown으로 변경됨

A screenshot of a Jupyter Notebook input field. The text "In [ ]:" is visible in blue on the left, followed by a large, empty, light gray rectangular box for text input. The entire input area is enclosed in a thin gray border.

- 하늘색 모드 → y 입력 → Code로 변경됨

A screenshot of a Jupyter Notebook input field. The text "In [ ]:" is visible in blue on the left, followed by a large, empty, light gray rectangular box for text input. The entire input area is enclosed in a thin gray border.A screenshot of a Jupyter Notebook input field. The text "In [ ]:" is visible in blue on the left, followed by a large, empty, light gray rectangular box for text input. The entire input area is enclosed in a thin gray border.

- 타이틀 작성

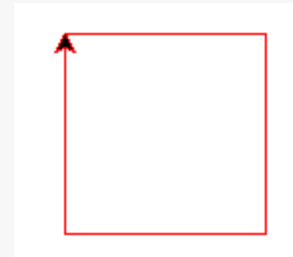


- Jupyter notebook 프로그램 테스트

```
In [5]: print("I Love Python")  
print("5 곱하기 3 은 ", 5*3, "입니다")
```

I Love Python  
5 곱하기 3 은 15 입니다

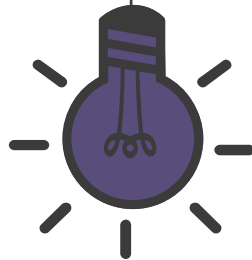
```
In [2]: import turtle  
t=turtle.Pen()  
  
t.pencolor("red")  
t.forward(100)  
t.right(90)  
t.forward(100)  
t.right(90)  
t.forward(100)  
t.right(90)  
t.forward(100)
```





02

**Numpy**



## ● 데이터 마이닝

- 데이터에서 의미 있는 정보를 추출하는 기술
- 고급 통계 분석과 모델링 기법을 적용하여 데이터 안의 패턴과 관계를 찾아 내는 과정

## ● 텍스트 마이닝

- 텍스트 문서에서 의미 있는 정보를 추출하는 기술
- 비정형 텍스트 데이터를 정형화 및 특징을 추출하는 과정이 요구됨
- 컴퓨터가 인식해 처리하는 자연어 처리(NLP) 기술에 기반을 두고 데이터를 가공하는 기술

# 1. Numpy란 ?

- C, C++, 포트란 으로 쓰인 코드를 통합하는 도구
- 다차원 배열(ndarray)을 편리하게 처리할 수 있도록 지원하는 파이썬 라이브러리다.
  - 다차원 배열(N-dimensional Array)을 지원한다
  - 서로 다른 타입의 데이터를 담을 수 없다.
  - 내부 반복문을 사용해서 속도가 빠르다.
  - 배열 간에 산술 연산이 가능하다.
- 선형대수, 난수 발생기가 있다
- 반복문을 작성할 필요 없어 전체 데이터에 대해 빠른 연산을 제공하는 표준 수학 함수이다

## ※ ndarray vs list

- ndarray : 적은 메모리로 데이터를 빠르게 처리하고, 모든 원소는 같은 자료형을 가진다.
- list : 속도가 매우 느리고, 원소가 각각 다른 자료형을 가질 수 있다.

## 2. 데이터 타입 종류

(1) int(8bit, 16bit, 32bit, 64bit) i1, i2, i4, i8

- 부호가 있다.
- 비트수 만큼 크기를 가지는 정수형이다.
- 저장할 수 있는 값의 범위 :  $-2^{n-1} \sim 2^{n-1}-1$

(2) uint(8bit, 16bit, 32bit, 64bit) u1, u2, u4, u8

- 부호가 없다.
- 비트수 만큼 크기를 가지는 정수형이다.
- 저장할 수 있는 값의 범위 :  $0 \sim 2^n-1$

(3) float(16bit, 32bit, 64bit, 128bit) f2, f4, f8, f16

- 부호가 있다.
- 비트수 만큼 크기를 가지는 실수형이다.

## 2. 데이터 타입 종류

### (4) 복소수형

- complex64 : 두개의 32비트 부동 소수점으로 표시되는 복소수 c8
- complex128 : 두개의 64비트 부동소수점으로 표시되는 복소수 c16

### (5) unicode

- 고정 길이 문자열 Unicode

### (6) bool

- True, False

### (7) 데이터 타입 확인 및 변경

- dtype : 자료형 확인
- astype : 자료형 변경

In [1]: `import numpy as np`

In [2]: `data = [[1,2,3],[4,5,6],[7,8,9]]  
a=np.array(data)  
a`

Out[2]: `array([[1, 2, 3],  
 [4, 5, 6],  
 [7, 8, 9]])`

In [3]: `a.dtype`

Out[3]: `dtype('int32')`

In [4]: `type(a)`

Out[4]: `numpy.ndarray`

```
In [5]: a=a.astype(np.float32) # a = a.astype('float32')  
a
```

```
Out[5]: array([[1., 2., 3.],  
              [4., 5., 6.],  
              [7., 8., 9.]], dtype=float32)
```

```
In [6]: a[0][1]
```

```
Out[6]: 2.0
```

```
In [7]: a[0]
```

```
Out[7]: array([1., 2., 3.], dtype=float32)
```

```
In [8]: np.arange(1,10,2)
```

```
Out[8]: array([1, 3, 5, 7, 9])
```

In [9]:

```
np.arange(1,10).reshape(3,3)
```

Out[9]:

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

In [10]:

```
np.arange(1,13).reshape(3,2,2)
```

Out[10]:

```
array([[[ 1,  2],  
        [ 3,  4]],  
       [[ 5,  6],  
        [ 7,  8]],  
       [[ 9, 10],  
        [11, 12]])
```

In [11]:

```
np.nan * 10
```

Out[11]:

```
nan
```



```
In [12]: a[0][1] = np.nan  
a
```

```
Out[12]: array([[ 1., nan,  3.],  
               [ 4.,  5.,  6.],  
               [ 7.,  8.,  9.]], dtype=float32)
```

```
In [13]: a=np.arange(1,10).reshape(3,3)  
a
```

```
Out[13]: array([[1, 2, 3],  
               [4, 5, 6],  
               [7, 8, 9]])
```

```
In [14]: a[0][1]=np.nan # nan은 int에서는 사용할수 없다. float에서만 사용할수 있다
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-14-058aef2a21af> in <module>  
----> 1 a[0][1]=np.nan # nan은 int에서는 사용할수 없다. float에서만 사용할수 있다
```

```
ValueError: cannot convert float NaN to integer
```

```
In [15]: a=np.linspace(1,10,20)  
a
```

```
Out[15]: array([ 1.          ,  1.47368421,  1.94736842,  2.42105263,  2.89473684,  
                3.36842105,  3.84210526,  4.31578947,  4.78947368,  5.26315789,  
                5.73684211,  6.21052632,  6.68421053,  7.15789474,  7.63157895,  
                8.10526316,  8.57894737,  9.05263158,  9.52631579, 10.          ])
```

### 3. 연산

- 반복문을 사용하지 않아도, 배열의 모든 원소는 반복연산이 사용된다.
- 선형 대수 식을 사용하여 연산 가능하다.
  - np.dot, @ : 행렬 곱 구하는 연산

```
In [16]: data=np.arange(1,10).reshape(3,3)
data
```

```
Out[16]: array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```

```
In [17]: data + data
```

```
Out[17]: array([[ 2,  4,  6],
               [ 8, 10, 12],
               [14, 16, 18]])
```

```
In [18]: [1,2,3] + [4,5,6]  # 파이썬에서는 연산이라 아니라 연결
```

```
Out[18]: [1, 2, 3, 4, 5, 6]
```

In [19]:

```
data - data
```

Out[19]:

```
array([[0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0]])
```

In [20]:

```
data * data
```

Out[20]:

```
array([[ 1,  4,  9],  
       [16, 25, 36],  
       [49, 64, 81]])
```

In [21]:

```
data / data
```

Out[21]:

```
array([[1., 1., 1.],  
       [1., 1., 1.],  
       [1., 1., 1.]])
```

```
In [22]: np.dot(data,data)    #행, 열의 곱
```

```
Out[22]: array([[ 30,  36,  42],  
               [ 66,  81,  96],  
               [102, 126, 150]])
```

```
In [23]: data @ data
```

```
Out[23]: array([[ 30,  36,  42],  
               [ 66,  81,  96],  
               [102, 126, 150]])
```

## 4. 차원

- 0차원 : Scalar (하나의 데이터 값으로만 존재하는 것)
- 1차원 : Vector (숫자들의 배열 (1D array))
- 2차원 : Matrix (숫자들의 2D array (rows: 행, columns: 열))
- 3차원 이상 : Tensor (숫자들의 다차원 배열)

In [24]:

```
# 0차원: 스칼라  
a=np.array(1)  
print(a)  
print(a.shape)  
print(a.ndim)    # 몇차원인지?
```

```
1  
()  
0
```

In [25]:

```
# 1차원: 벡터  
a=np.array([1])  
print(a)  
print(a.shape)  
print(a.ndim)
```

```
[1]  
(1,)  
1
```

In [26]:

```
# 1차원: 벡터
a=np.array([1,2,3,4,5])
print(a)
print(a.shape)  # 5개의 형태
print(a.ndim)
```

```
[1 2 3 4 5]
(5,)
1
```

In [27]:

```
# 2차원: 매트릭스
a=np.array([[1,2,3],[4,5,6]])
print(a)
print(a.shape)  # 2행 3열
print(a.ndim)
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
2
```

```
In [28]: # 2차원: 매트릭스
a=np.array([[1]])
print(a)
print(a.shape)
print(a.ndim)
```

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

```
In [29]: # 3차원: 3차원 이상의 다차원의 행렬을 Tensor
# TensorFlow : 다차원 행렬을 가지고 연산을 해서 어떤 결과를 도출해 내는것
a=np.array([[[1,2],[3,4],[5,6]], [[7,8],[9,10],[11,12]]])

print(a)
print(a.shape)
print(a.ndim)
```

```
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
(2, 3, 2)
3
```



## 5. matrix 유형

- zeros : 0으로 초기화된 배열 생성
- ones : 1로 초기화 된 배열 생성
- eye : 주 대각선의 원소가 모두 1이고 나머지 원소는 0인 정사각행렬 (단위행렬)
- empty : 초기화 하지 않고 배열만 생성, 기존에 메모리에 저장되어 있는 값으로 나타남
- full : 지정한 숫자로 초기화
- linspace(start, end(포함), number, endpoint = ) : 선형 구간을 지정한 개수만큼 분할한다.
  - endpoint = True or False : 마지막 값을 포함시킬지 시키지 않을지 선택

```
In [30]: a=np.arange(12).reshape(2,3,2)
a
```

```
Out[30]: array([[[ 0,  1],
                 [ 2,  3],
                 [ 4,  5]],

                [[ 6,  7],
                 [ 8,  9],
                 [10, 11]]])
```

```
In [31]: b=np.ones(12)
b
```

```
Out[31]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
In [32]: b=np.ones(12).reshape(2,3,2)
b
```

```
Out[32]: array([[[1., 1.],
                 [1., 1.],
                 [1., 1.]],

                [[1., 1.],
                 [1., 1.],
                 [1., 1.]])
```

```
In [33]: c=np.zeros(12).reshape(2,3,2)
c
```

```
Out[33]: array([[0., 0.],
               [0., 0.],
               [0., 0.]],

            [[0., 0.],
             [0., 0.],
             [0., 0.]])
```

```
In [34]: d=np.eye(3)    # 단위 행렬
d
```

```
Out[34]: array([[1., 0., 0.],
               [0., 1., 0.],
               [0., 0., 1.]])
```

```
In [35]: e=np.zeros([3,4])
e
```

```
Out[35]: array([[0., 0., 0., 0.],
               [0., 0., 0., 0.],
               [0., 0., 0., 0.]])
```

```
In [36]: e=np.zeros([3,4,2])  
e
```

```
Out[36]: array([[[0., 0.],  
                [0., 0.],  
                [0., 0.],  
                [0., 0.]],  
               [[0., 0.],  
                [0., 0.],  
                [0., 0.],  
                [0., 0.]],  
               [[0., 0.],  
                [0., 0.],  
                [0., 0.],  
                [0., 0.]])
```

```
In [37]: f=np.empty([2,3])  # 0에 가까운 수  
f
```

```
Out[37]: array([[4.24399158e-314, 8.48798317e-314, 1.27319747e-313],  
               [1.69759663e-313, 2.12199579e-313, 2.54639495e-313]])
```

```
In [38]: g=np.full((3,4),1000)
g
```

```
Out[38]: array([[1000, 1000, 1000, 1000],
               [1000, 1000, 1000, 1000],
               [1000, 1000, 1000, 1000]])
```

```
In [39]: h=np.linspace(2,10,6)
h
```

```
Out[39]: array([ 2. ,  3.6,  5.2,  6.8,  8.4, 10. ])
```

## 6. 집계함수

- mean : 평균을 구하는 함수
- median : 데이터를 크기로 정렬하고 그 중 가운데 수 출력하는 함수  
※ 만약 데이터 개수가 짝수일 경우 가장 가운데의 두 수의 평균을 구한다.
- var : 분산 구하는 함수
- std : 표준편차 구하는 함수
- sum : 합계 구하는 함수
- max : 가장 큰 수를 구하는 함수
- min : 가장 작은 수를 구하는 함수

```
In [40]: a = np.arange(10).reshape(2,5)
a
```

```
Out[40]: array([[0, 1, 2, 3, 4],
               [5, 6, 7, 8, 9]])
```

```
In [41]: a[0][0]=100
a
```

```
Out[41]: array([[100,  1,  2,  3,  4],
               [  5,  6,  7,  8,  9]])
```

```
In [42]: a[0][1]=1000
a
```

```
Out[42]: array([[ 100, 1000,  2,  3,  4],
               [  5,   6,  7,  8,  9]])
```

```
In [43]: np.mean(a)  #평균
```

```
Out[43]: 114.4
```

```
In [44]: np.median(a) #중앙값
```

```
Out[44]: 6.5
```

```
In [45]: np.std(a) #표준편차
```

```
Out[45]: 296.5485457728633
```

```
In [46]: np.var(a) #분산
```

```
Out[46]: 87941.04
```

```
In [47]: np.sum(a)
```

```
Out[47]: 1144
```

```
In [48]: sum(a)
```

```
Out[48]: array([ 105, 1006,    9,   11,   13])
```



```
In [49]:
```

```
a
```

```
Out[49]: array([[ 100, 1000,   2,   3,   4],  
               [   5,   6,   7,   8,   9]])
```

```
In [50]: np.sum(a, axis=0)
```

```
Out[50]: array([ 105, 1006,   9,  11,  13])
```

```
In [51]: np.sum(a, axis=1)
```

```
Out[51]: array([1109,   35])
```

```
In [52]: np.max(a)
```

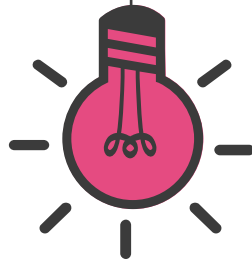
```
Out[52]: 1000
```

```
In [53]: np.min(a)
```

```
Out[53]: 2
```

03

**Pandas**



# 1. Pandas 란 ?

- 데이터를 분석할 때 가장 많이 쓰이는 라이브러리다.
- 아나콘다를 설치하면 Pandas가 자동으로 포함되어 설치 한다
- 행과 열을 쉽게 처리할 수 있는 함수를 제공하는 도구이다.
  - ※ 각 열은 단일 데이터 형식만 저장
- Numpy 보다 유연하게 수치 연산 가능하다.

## ※ pandas 라이브러리

- 결측치 처리
- 열(필드)의 삽입과 제거
- 데이터 정렬/병합
- 타 자료구조를 pandas 데이터 프레임 형태로 의 변환
- 색인,인덱싱,부분집합
- 데이터 세트 조인
- 데이터 세트 차원 변환
- 축의 계층적 레이블링
- 다양한 포맷으로의 입출력 기능
- 다양한 시계열 함수

## 2. Series

- index와 values로 이루어진 1차원 배열이다.
- 모든 유형의 데이터를 보유할 수 있다.
- 인덱스를 지정해 줄 수 있다.
- 인덱스 길이는 데이터의 길이와 같아야 한다.
- 명시적 인덱스와 암묵적 인덱스를 가진다.
  - loc : 인덱스값을 기반으로 행 데이터를 읽는다.
  - iloc : 정수 인덱스를 기반으로 행 데이터를 읽는다.

- (1) Series

```
In [1]: import pandas as pd  
import numpy as np
```

```
In [2]: obj = pd.Series([4, 7, -5, 3])  
obj
```

```
Out[2]: 0    4  
1    7  
2   -5  
3    3  
dtype: int64
```

```
In [3]: obj.values
```

```
Out[3]: array([ 4,  7, -5,  3], dtype=int64)
```

```
In [4]: obj.index
```

```
Out[4]: RangeIndex(start=0, stop=4, step=1)
```

```
In [5]: obj2=pd.Series([1,3,-4,0], index=['a','b','k','z'])  
obj2
```

```
Out[5]: a      1  
       b      3  
       k     -4  
       z      0  
       dtype: int64
```

```
In [6]: obj2['k']
```

```
Out[6]: -4
```

```
In [7]: obj2['a']=11  
obj2[['k','a','b']]
```

```
Out[7]: k      -4  
       a      11  
       b       3  
       dtype: int64
```

```
In [8]: obj2.isnull()
```

```
Out[8]: a    False  
       b    False  
       k    False  
       z    False  
       dtype: bool
```

```
In [9]: data=np.arange(0,50,10)  
       data
```

```
Out[9]: array([ 0, 10, 20, 30, 40])
```

```
In [10]: a=pd.Series(data, index=['a','b','c','d','e'])  
       a
```

```
Out[10]: a     0  
       b    10  
       c    20  
       d    30  
       e    40  
       dtype: int32
```

```
In [11]: b=pd.Series(data)
b
```

```
Out[11]: 0    0
         1   10
         2   20
         3   30
         4   40
         dtype: int32
```

```
In [12]: a['b']
```

```
Out[12]: 10
```

```
In [13]: a.loc['b'] # 명시적인 index
```

```
Out[13]: 10
```

```
In [14]: a.iloc[1] # 순서
```

```
Out[14]: 10
```



- (2) 산술연산

In [15]:

```
a
```

Out[15]:

```
a      0
b     10
c     20
d     30
e     40
dtype: int32
```

In [16]:

```
a + 10
a - 10
a * 10
a ** 2
a / 5
a // 5
a % 3
```

Out[16]:

```
a      0
b      1
c      2
d      0
e      1
dtype: int32
```

In [17]:

```
a
```

Out[17]:

```
a      0
b     10
c     20
d     30
e     40
dtype: int32
```

In [18]:

```
a > 15
```

Out[18]:

```
a      False
b      False
c       True
d       True
e       True
dtype: bool
```

In [19]:

```
a[a > 15]
```

Out[19]:

```
c     20
d     30
e     40
dtype: int32
```

## (1) 집계함수

- add : 더하기 함수
- sub : 빼기 함수
- mul : 곱하기 함수
- div : 나누기 함수
- mod : 나머지 구하는 함수
- min : 최소값 구하는 함수
- max : 최대값 구하는 함수
- mean : 평균 구하는 함수
- median : 중앙값 구하는 함수
- std : 표준편차 구하는 함수
- var : 분산 구하는 함수

```
In [20]: a.add(100)
```

```
Out[20]: a    100  
         b    110  
         c    120  
         d    130  
         e    140  
         dtype: int32
```

```
In [21]: a.sub(100)
```

```
Out[21]: a   -100  
         b    -90  
         c    -80  
         d    -70  
         e    -60  
         dtype: int32
```

```
In [22]: a.mul(100)
```

```
Out[22]: a      0  
         b   1000  
         c   2000  
         d   3000  
         e   4000  
         dtype: int32
```

```
In [23]: a.div(100)
```

```
Out[23]: a    0.0  
         b    0.1  
         c    0.2  
         d    0.3  
         e    0.4  
         dtype: float64
```

```
In [24]: a.mod(3)
```

```
Out[24]: a    0  
         b    1  
         c    2  
         d    0  
         e    1  
         dtype: int32
```

```
In [25]: a.min()
```

```
Out[25]: 0
```

```
In [26]: a.max()
```

```
Out[26]: 40
```

```
In [27]: a.sum()
```

```
Out[27]: 100
```

```
In [28]: a.mean() #평균
```

```
Out[28]: 20.0
```

```
In [29]: a.median() #중앙값
```

```
Out[29]: 20.0
```

```
In [30]: a.std() #표준편차
```

```
Out[30]: 15.811388300841896
```

```
In [31]: a.var() #분산
```

```
Out[31]: 250.0
```

### 3. DataFrame

- 가장 기본적인 데이터 구조이다.
- 행과 열로 구성된 2차원 데이터 구조(Data Structure)
  - 2차원 배열에 행과 열에 인덱스를 붙인 것이다.
- RDB의 테이블 또는 엑셀(스프레드 시트)와 유사

```
In [32]: rawData = np.random.randint(50,100, size=(4,3))  
rawData
```

```
Out[32]: array([[61, 64, 75],  
               [91, 79, 95],  
               [74, 98, 61],  
               [82, 58, 94]])
```

```
In [33]: df=pd.DataFrame(rawData, index=['1반', '2반', '1반', '2반'],
                        columns=['국', '영', '수'])
df
```

Out[33]:

	국	영	수
1반	61	64	75
2반	91	79	95
1반	74	98	61
2반	82	58	94

```
In [34]: #df[0]    --> error
df['국']
```

Out[34]: 1반 61  
2반 91  
1반 74  
2반 82  
Name: 국, dtype: int32



In [35]:

```
df
```

Out[35]:

	국	영	수
1반	61	64	75
2반	91	79	95
1반	74	98	61
2반	82	58	94

In [36]:

```
df['평균']=round((df['국'] + df['영'] + df['수'])/3,2)  
df
```

Out[36]:

	국	영	수	평균
1반	61	64	75	66.67
2반	91	79	95	88.33
1반	74	98	61	77.67
2반	82	58	94	78.00

```
In [37]: df['na']=np.nan  
df
```

Out[37]:

	국	영	수	평균	na
1반	61	64	75	66.67	NaN
2반	91	79	95	88.33	NaN
1반	74	98	61	77.67	NaN
2반	82	58	94	78.00	NaN

```
In [38]: del df['na']  
df
```

Out[38]:

	국	영	수	평균
1반	61	64	75	66.67
2반	91	79	95	88.33
1반	74	98	61	77.67
2반	82	58	94	78.00

```
In [39]: df[df.평균 > 75]
```

Out[39]:

	국	영	수	평균
2반	91	79	95	88.33
1반	74	98	61	77.67
2반	82	58	94	78.00

```
In [40]: df=df.drop(['평균'], axis='columns')
df
```

Out[40]:

	국	영	수
1반	61	64	75
2반	91	79	95
1반	74	98	61
2반	82	58	94

## (2) 결측값 처리

### 1. NaN

- 자료형이 Float형이다.
- 배열에서 연산할 경우 오류가 발생하지 않지만 결과값이 NaN이 된다.

### 2. None

- 자료형이 None이다.
- 배열 연산을 할 경우 오류가 발생한다.

### 3. 처리방법

- isnull() : 결측값 확인 (결측 이면 True , 결측이 아니면 False )
- notnull() : 결측값 확인 (결측 이면 False , 결측이 아니면 True )
- dropna() : 결측값 삭제
  - inplace = True : drop후 원본에 반영
- fillna(Num) : 결측값을 Num으로 채워 넣는다.

```
In [41]: df=df.astype('float64')
df
```

Out[41]:

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
1반	74.0	98.0	61.0
2반	82.0	58.0	94.0

```
In [42]: df['수'][2]=np.nan
df
```

Out[42]:

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
1반	74.0	98.0	NaN
2반	82.0	58.0	94.0

```
In [43]: df.dropna(axis=0) #원본을 수정하지 않는다
#df.dropna(axis=0, inplace=True) # inplace는 원본을 수정한다
```

Out[43]:

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
2반	82.0	58.0	94.0

```
In [44]: df
```

Out[44]:

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
1반	74.0	98.0	NaN
2반	82.0	58.0	94.0

```
In [45]: df.dropna(axis=1)
```

Out[45]:

	국	영
1반	61.0	64.0
2반	91.0	79.0
1반	74.0	98.0
2반	82.0	58.0

```
In [46]: df
```

Out[46]:

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
1반	74.0	98.0	NaN
2반	82.0	58.0	94.0

```
In [47]: df.fillna('hello')
```

Out[47]:

	국	영	수
1반	61.0	64.0	75
2반	91.0	79.0	95
1반	74.0	98.0	hello
2반	82.0	58.0	94

```
In [48]: df.fillna(0)
```

Out[48]:

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
1반	74.0	98.0	0.0
2반	82.0	58.0	94.0



```
In [49]: df.fillna(df.mean())
```

Out[49]:

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
1반	74.0	98.0	88.0
2반	82.0	58.0	94.0

In [50]:

```
df
```

Out[50]:

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
1반	74.0	98.0	NaN
2반	82.0	58.0	94.0

In [51]: `df.T` *#행과 열을 바꿈*

Out[51]:

	1반	2반	1반	2반
국	61.0	91.0	74.0	82.0
영	64.0	79.0	98.0	58.0
수	75.0	95.0	NaN	94.0

```
In [52]: df
```

```
Out[52]:
```

	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0
1반	74.0	98.0	NaN
2반	82.0	58.0	94.0

```
In [53]: df.index=[['1학년', '1학년', '2학년', '2학년'],  
                    ['1반', '2반', '1반', '2반']]  
df
```

```
Out[53]:
```

		국	영	수
1학년	1반	61.0	64.0	75.0
	2반	91.0	79.0	95.0
2학년	1반	74.0	98.0	NaN
	2반	82.0	58.0	94.0

```
In [54]: df.columns=[['언어', '언어', '수리'], ['국', '영', '수']]
df
```

Out[54]:

		언어		수리
		국	영	수
1학년	1반	61.0	64.0	75.0
	2반	91.0	79.0	95.0
2학년	1반	74.0	98.0	NaN
	2반	82.0	58.0	94.0

```
In [55]: df['언어']['국']
```

Out[55]: 1학년 1반 61.0  
          2반 91.0  
2학년 1반 74.0  
          2반 82.0  
Name: 국, dtype: float64

```
In [56]: df.iloc[0]
```

```
Out[56]: 언어  국    61.0  
        영    64.0  
        수리 수    75.0  
        Name: (1학년, 1반), dtype: float64
```

```
In [57]: df.loc['1학년']
```

```
Out[57]:
```

	언어	수리	
	국	영	수
1반	61.0	64.0	75.0
2반	91.0	79.0	95.0

```
In [58]: df.loc['1학년'].loc['1반']
```

```
Out[58]: 언어  국    61.0  
        영    64.0  
        수리 수    75.0  
        Name: 1반, dtype: float64
```

### (3) 데이터 사전 분석

- `info()` : DataFrame을 구성하는 행과 열에 대한 정보를 나타내 주는 함수
- `head(n)` : DataFrame의 처음부터 n줄의 행을 출력
- `tail(n)` : DataFrame의 마지막 n줄의 행을 출력
- `describe()` : Series, DataFrame의 각 열에 대한 요약 통계
- `dtypes` : 데이터 자료형 확인

```
In [59]: df
```

```
Out[59]:
```

		언어		수리
		국	영	수
1학년	1반	61.0	64.0	75.0
	2반	91.0	79.0	95.0
2학년	1반	74.0	98.0	NaN
	2반	82.0	58.0	94.0

```
In [60]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 4 entries, ('1학년', '1반') to ('2학년', '2반')
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   (언어, 국)   4 non-null     float64
1   (언어, 영)   4 non-null     float64
2   (수리, 수)   3 non-null     float64
dtypes: float64(3)
memory usage: 248.0+ bytes
```

```
In [61]: df.head() # 5개의 행까지 보여줄
```

Out[61]:

		언어		수리
		국	영	수
1학년	1반	61.0	64.0	75.0
	2반	91.0	79.0	95.0
2학년	1반	74.0	98.0	NaN
	2반	82.0	58.0	94.0

```
In [62]: df.tail()
```

Out[62]:

		언어		수리
		국	영	수
1학년	1반	61.0	64.0	75.0
	2반	91.0	79.0	95.0
2학년	1반	74.0	98.0	NaN
	2반	82.0	58.0	94.0



```
In [63]: df.dtypes
```

```
Out[63]: 언어  국      float64  
        영      float64  
        수리 수      float64  
        dtype: object
```

```
In [64]: df.describe()
```

```
Out[64]:
```

	언어		수리
	국	영	수
<b>count</b>	4.000000	4.000000	3.000000
<b>mean</b>	77.000000	74.750000	88.000000
<b>std</b>	12.727922	17.839563	11.269428
<b>min</b>	61.000000	58.000000	75.000000
<b>25%</b>	70.750000	62.500000	84.500000
<b>50%</b>	78.000000	71.500000	94.000000
<b>75%</b>	84.250000	83.750000	94.500000
<b>max</b>	91.000000	98.000000	95.000000

```
In [65]: #결측치 여부  
df.isnull()
```

Out[65]:

		언어		수리
		국	영	수
1학년	1반	False	False	False
	2반	False	False	False
2학년	1반	False	False	True
	2반	False	False	False

```
In [66]: df.isnull().sum() #결측치가 몇개인지?
```

Out[66]: 언어 국 0  
영 0  
수리 수 1  
dtype: int64

## (4) 값의 연결

- concat : DataFrame끼리 결합
  - axis=0 or 1 : 아래로 데이터 연결 / 옆으로 데이터 연결
- append : 마지막 행에 데이터를 추가
- ※ concatenate : 배열끼리 결합

```
In [67]: a=pd.DataFrame(np.arange(1,10).reshape(3,3))  
a
```

Out[67]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

```
In [68]: b=pd.Series(np.arange(10,40,10))  
b
```

```
Out[68]: 0    10  
1     20  
2     30  
dtype: int32
```

```
In [69]: pd.concat([a,b], axis=1)
```

```
Out[69]:
```

	0	1	2	0
0	1	2	3	10
1	4	5	6	20
2	7	8	9	30

```
In [70]: pd.concat([a,b], axis=1, ignore_index=True)
```

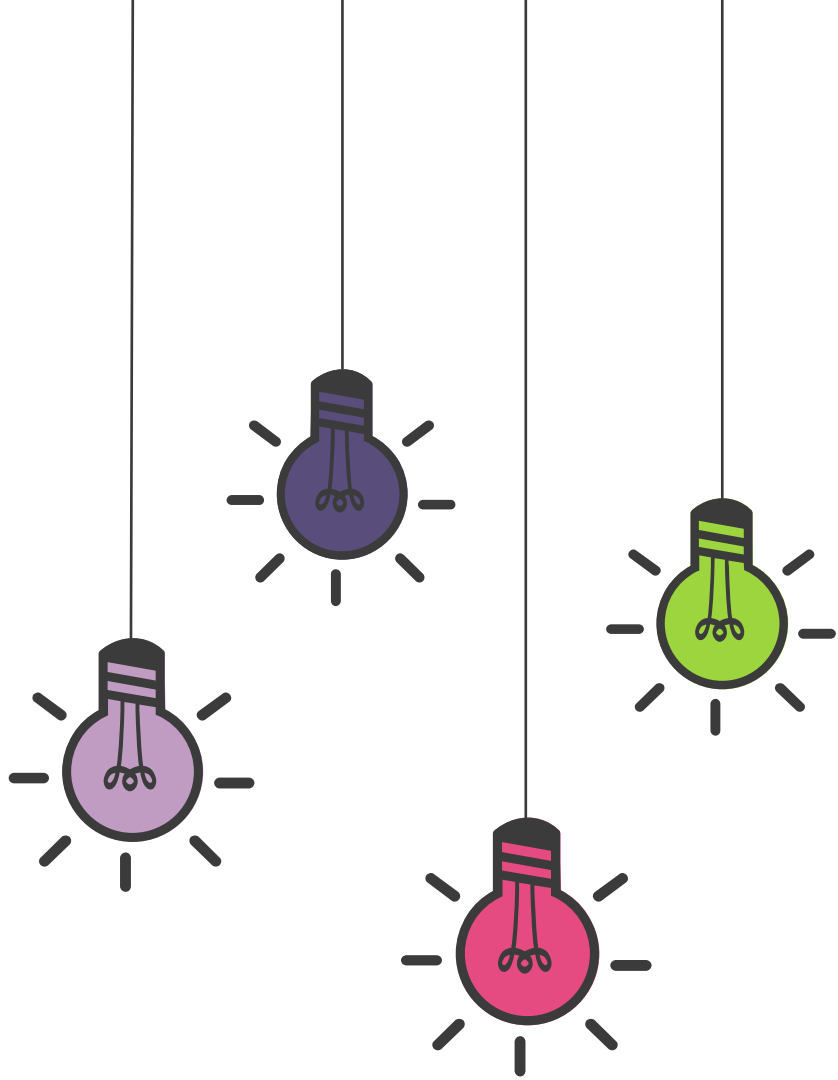
Out[70]:

	0	1	2	3
0	1	2	3	10
1	4	5	6	20
2	7	8	9	30

```
In [71]: a.append(b, ignore_index=True)
```

Out[71]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	20	30



# 감사합니다

THANK YOU FOR WATCHING



김 계 희  
jenni7@naver.com