

파이썬 외부모듈 (matplotlib, numpy, pandas)

Seolyoung Jeong, Ph.D.

경북대학교 IT대학 컴퓨터학부

3rd Party 모듈 설치

3rd Party 모듈

- ◆ 표준 모듈 : 파이썬에서 기본적으로 제공하는 모듈
- ◆ 사용자 생성 모듈 : 개발자가 직접 작성한 모듈
- ◆ 서드 파티(3rd Party) 모듈 : 파이썬 재단이나 개발자가 아닌 다른 프로그래머 또는 업체에서 제공하는 모듈
- ◆ 모듈은 단독으로 사용될 수도 있지만, 하나의 모듈이 다른 모듈을 참조하여 사용할 수도 있음
- ◆ 하나의 모듈에서 다른 모듈을 참조할 시 의존성을 확인하여 관련되어 있는 모듈들을 모두 설치해 줘야 해당 모듈을 사용할 수 있음

※ 외부모듈 직접 설치

- ◆ **PIP : 파이썬으로 작성된 패키지(라이브러리) 관리 프로그램**
 - 3.4 버전 이전에는 PIP 모듈도 직접 설치해야 했었음
(3.4 버전 이후부터는 기본 포함되어 있음)
- ◆ **명령 프롬프트(실행창에서 'cmd') 관리자 모드 실행**
 - > pip install 모듈명

```
> pip install matplotlib
```

```
> pip install numpy
```


- ◆ **설치 모듈 확인 : > pip list**

```
> pip list
```

※ 다운로드 후 설치

◆ 웹에서 설치파일 다운로드

Home / Browse / Development / Software Development / matplotlib / Files









matplotlib

Brought to you by: [cjgohlke](#), [dsdale](#), [efiring](#), [heeres](#), and 8 others

Summary | **Files** | Reviews | Support | Wiki | Mailing Lists | Donate | Code

Looking for the latest version? [Download matplotlib-1.5.0-cp35-none-win_amd64.whl \(6.5 MB\)](#)

Home / matplotlib

Name ↕	Modified ↕	Size ↕	Downloads / Week ↕
↑ Parent folder			
matplotlib-1.5.2	2017-04-05		129 
matplotlib-1.5.3	2017-04-05		42 
matplotlib-2.0.0	2017-04-05		76 
matplotlib-1.5.1	2016-01-11		61 
matplotlib-1.5.0	2015-10-30		1,046 
/projects/matplotlib/f...	Click to enter matplotlib-1.5.0	0-22	10 

※ PyCharm에서 설치

- ◆ Settings → Project → Project Interpreter
- ◆ '+' 선택 → package 검색 → Install Package

The screenshot shows the PyCharm interface with the Settings window open to the Project Interpreter section. The Project Interpreter is set to Python 3.7 (matplotlib). A table lists installed packages:

Package	Version	Latest version
pip	19.0.3	19.1.1
setuptools	40.8.0	

Below the table, the Available Packages window is open, showing a search for 'matplotlib'. The package 'matplotlib' is selected in the list. The Description panel on the right shows details for matplotlib:

Description
Python plotting package
Version
3.1.0
Author
John D. Hunter, Michael Droettboom
matplotlib-users@python.org
<https://matplotlib.org>

At the bottom of the Available Packages window, there are checkboxes for 'Specify version' (set to 3.1.0) and 'Options', and buttons for 'Install Package' and 'Manage Repositories'.

Anaconda

- ◆ 300개 이상의 모듈을 포함하고 있는 종합 패키지
- ◆ Numpy, Matplotlib, Django, Pandas, Scikit-learn, Scipy 등 일반적으로 많이 사용되는 유용한 모듈들을 다수 포함하고 있음
- ◆ <https://www.anaconda.com/>

[Products](#)[Why Anaconda?](#)[Solutions](#)[Resources](#)[Company](#)[Contact Us](#)[Download](#)[Search](#)

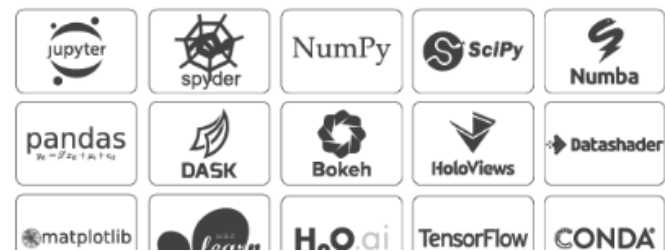
Anaconda Distribution

The World's Most Popular Python/R Data Science Platform

[Download](#)

The open-source [Anaconda Distribution](#) is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 15 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:

- Quickly download 1,500+ Python/R data science packages



Anaconda Download

①



Windows



macOS



Linux

Anaconda 2019.10 for Windows Installer

Python 3.7 version

②

Download

64-Bit Graphical Installer (462 MB)

32-Bit Graphical Installer (410 MB)

Python 2.7 version

Download

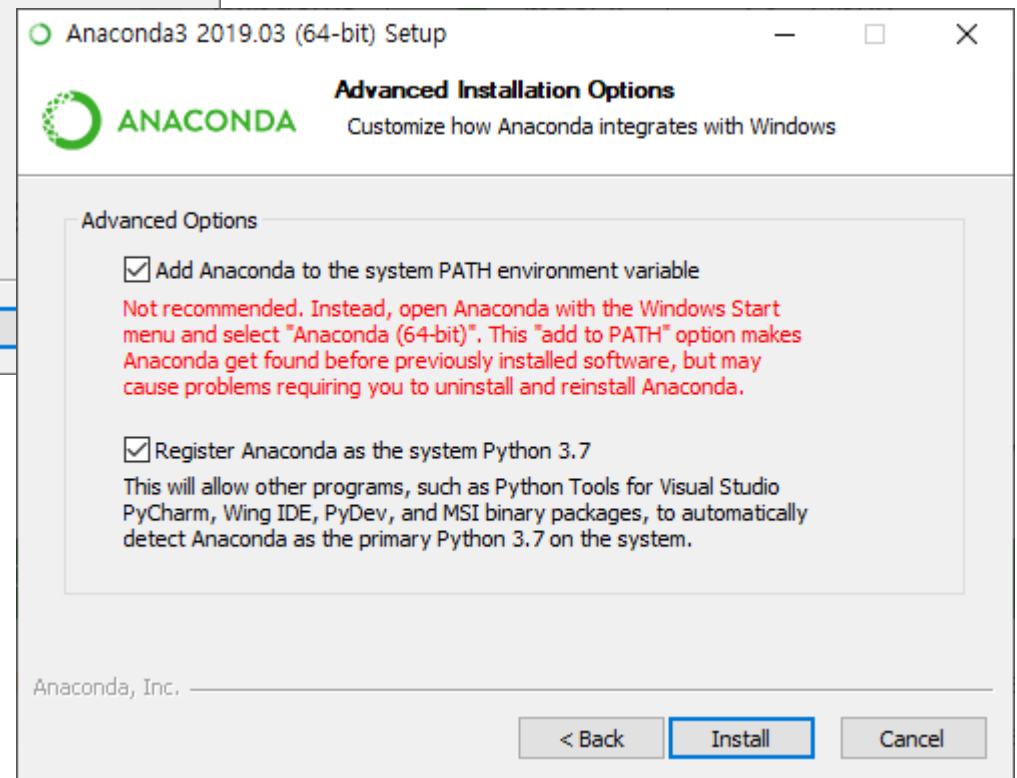
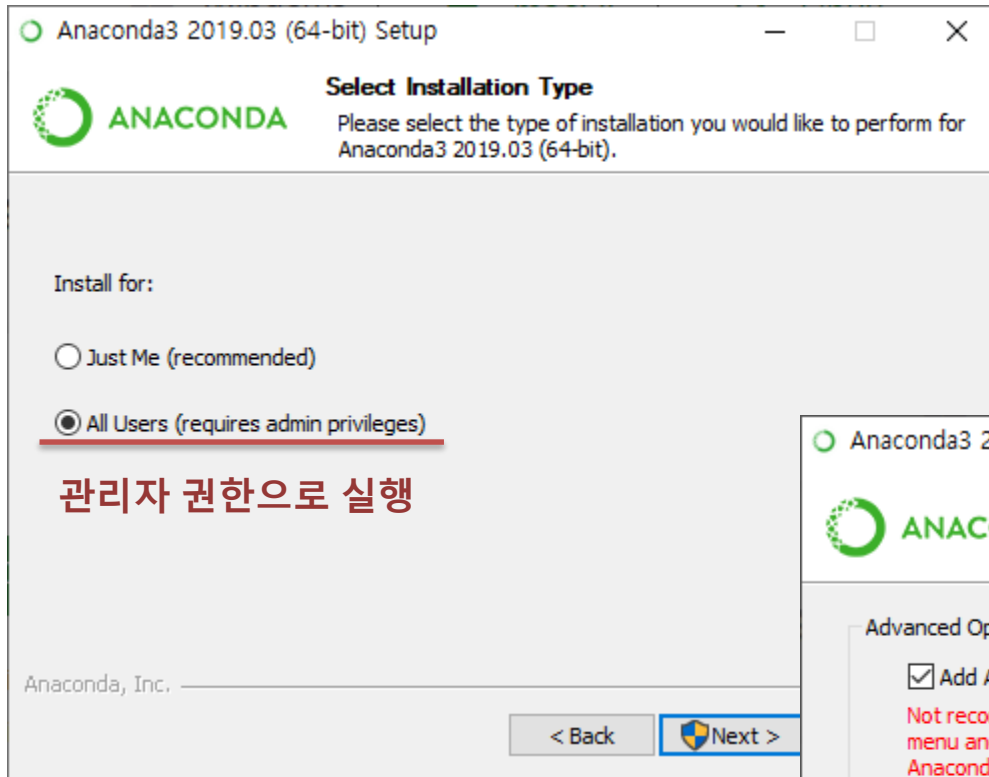
64-Bit Graphical Installer (413 MB)

32-Bit Graphical Installer (356 MB)

Get Started with Anaconda Distribution



Anaconda Install



PyCharm Settings

The image shows the PyCharm Settings window for the 'matplot' project, specifically the 'Project Interpreter' tab. The left sidebar lists various settings categories, with 'Project Interpreter' selected. The main area displays a list of installed packages and their versions. A file explorer window is overlaid on the bottom right, showing the contents of the 'C:\ProgramData\Anaconda3' directory, with 'python.exe' highlighted.

Project: matplot > Project Interpreter For current project

Project Interpreter: Python 3.7 (2) C:\ProgramData\Anaconda3\python.exe

Available interpreters:

- <No interpreter>
- Python 3.7 C:\Python37-32\python.exe
- Python 3.7 (2) C:\ProgramData\Anaconda3\python.exe (Selected)
- Show All...

Installed Packages:

Package Name	Version	Latest Version
alabaster		
anaconda	2019.10	2019.10
anaconda-client	1.7.2	1.7.2
anaconda-navigator	1.9.7	1.9.7
anaconda-project	0.8.3	▲ 0.8.4
asn1crypto	1.0.1	▲ 1.2.0
astroid	2.3.1	▲ 2.3.2
astropy	3.2.1	▲ 3.2.3
atomicwrites	1.3.0	
attrs	19.2.0	
babel	2.7.0	
backcall	0.1.0	
backports	1.0	
backports.functools_lru_cache	1.5	
backports.os	0.1.1	
backports.shutil_get_terminal_size	1.0.0	
backports.tempfile	1.0	
backports weakref	1.0.post1	
beautifulsoup4	4.8.0	
bitarray	1.0.1	
bkcharts	0.2	

File Explorer (C:\ProgramData\Anaconda3):

File Name	Size
api-ms-win-crt-stdio-l1-1-0.dll	25KB
api-ms-win-crt-string-l1-1-0.dll	25KB
api-ms-win-crt-time-l1-1-0.dll	21KB
api-ms-win-crt-utility-l1-1-0.dll	19KB
concr140.dll	325KB
cwp.py	2KB
LICENSE_PYTHON.txt	13KB
msvc140.dll	614KB
msvc140_1.dll	31KB
msvc140_2.dll	202KB
python.exe	93KB
python.pdb	428KB
python3.dll	51KB
python37.dll	3,661KB
python37.pdb	9,316KB
pythonw.exe	92KB
pythonw.pdb	428KB
qt.conf	1KB

Jupyter Notebook

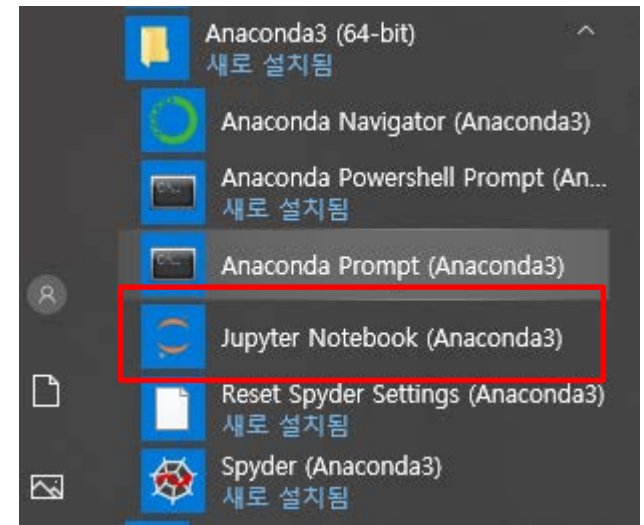
- ◆ 코드, 수식, 시각화 및 설명 텍스트가 포함된 문서를 작성하고 공유 가능
- ◆ 오픈소스 웹 애플리케이션 (웹 브라우저 기반)
- ◆ 데이터 정리, 변환, 수치 시뮬레이션, 통계 모델링, 데이터 시각화, 머신러닝 등에 사용

- ◆ **장점**

- 데이터 시각화 용이
- 코드 공유 용이
- 실시간 실행 (대화형) 가능
- 코드 샘플 기록

- ◆ **단점**

- IDE 기능 없음
- 디버깅, 모듈관리 등을 지원하는 본격적인 파이썬 개발 환경은 아님
- 세션 상태 저장 안됨 : 노트북을 불러올 때마다 코드를 다시 실행해야만 상태 복원 가능



Jupyter Notebook

The screenshot displays the Jupyter Notebook web interface in a browser window. The address bar shows the URL `localhost:8888/tree/Documents/jupyter_notebook`. The interface includes a top navigation bar with the Jupyter logo and 'Quit' and 'Logout' buttons. Below this, there are tabs for 'Files', 'Running', and 'Clusters'. The 'Files' tab is active, showing a file browser with a list of folders: `2019F_Python`, `BigData2019`, `handson_ML`, and `TTT_2019`. A red box highlights the 'New' button in the top right corner of the file browser. A dropdown menu is open, showing options for creating new files or folders. The 'Notebook:' section is highlighted with a red box, and the 'Python 3' option is selected. Other options visible include 'Text File', 'Folder', and 'Terminal'. The browser's address bar and various extension icons are also visible at the top.

Documents/jupyter_notebook/ x +

localhost:8888/tree/Documents/jupyter_notebook

jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them.

0 / Documents / jupyter_notebook

2019F_Python

BigData2019

handson_ML

TTT_2019

Upload New

Notebook:

Python 3

Other:

Text File

Folder

Terminal

창 캡처(W)

Jupyter Notebook

The screenshot shows a web browser window with two tabs: "Documents/jupyter_notebook/2" and "Untitled - Jupyter Notebook". The address bar shows the URL: `localhost:8888/notebooks/Documents/jupyter_notebook/2019F_Python/Untitled.ipynb?kernel_name=python3`. The browser's bookmark bar includes links to Google Translate, Google Scholar, Google Maps, Google Docs, Google Drive, EC2 Management, GitHub, and various bookmarks.

The Jupyter Notebook interface is displayed below the browser window. It features the Jupyter logo and the text "Untitled Last Checkpoint: 몇 초 전 (unsaved changes)". A "Logout" button is visible in the top right corner. The main menu bar includes "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, creating new cells, deleting cells, undo, redo, and running code. The "Run" button is highlighted. The code editor area shows a single code cell with the prompt `In []:` and a text input field.

matplotlib

간단한 그래프

◆ matplotlib.pyplot 모듈

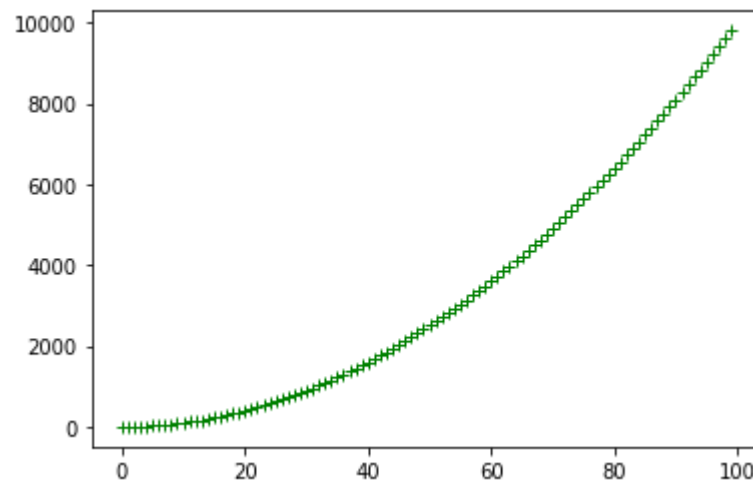
◆ plot() 함수 : 값을 서로 연결하여 라인 형태의 그래프 그림

- 기본 포맷 : 파란색 선
- 'r' : 빨간색 선, plt.plot(x, y, 'r')
- 'ro' : 빨간색 원 마커 모양, plt.plot(x, y, 'ro')
- 'r--' : 빨간색 대쉬라인, plt.plot(x, y, 'r--')
- 'bs' : 파란색 사각형, plt.plot(x, y, 'bs')
- 'g+' : 녹색 십자모양, plt.plot(x, y, 'g+')

```
In [1]: %matplotlib inline
import matplotlib.pyplot as plt
```

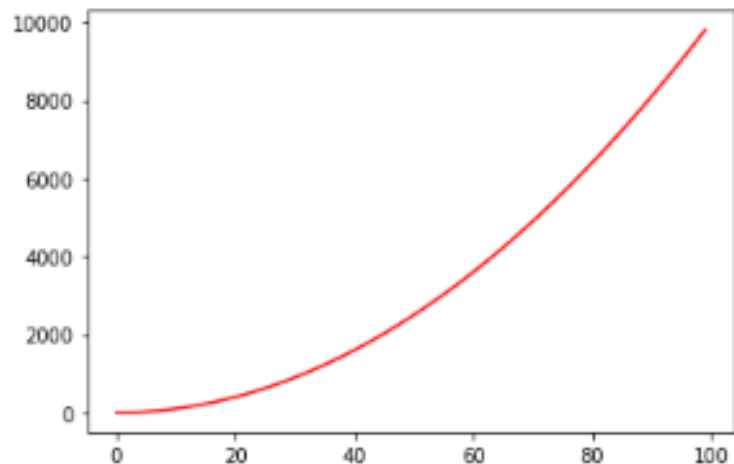
```
x = range(100)          # x = [0, 1, 2, 3, 4, ..., 98, 99]
y = [i*i for i in x]    # y = [0, 1, 4, 9, 16, ..., 9804, 9801]

plt.plot(x, y, 'g+')    # x axis, y axis
plt.show()              # drawing figure
```

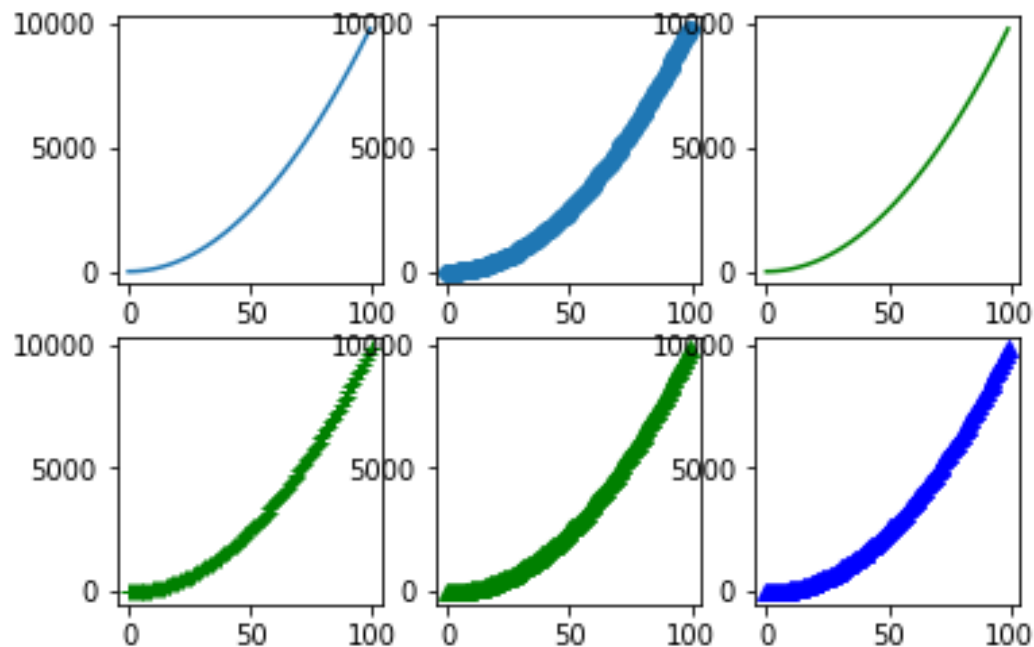
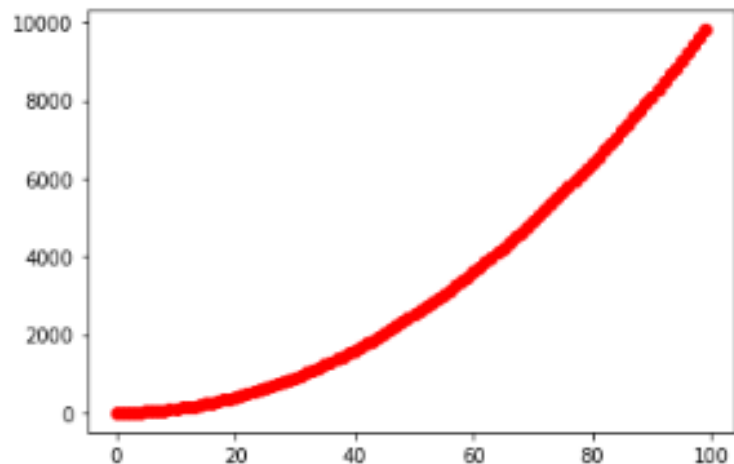


간단한 그래프

```
plt.plot(x, y, 'r')  
plt.show()
```



```
plt.plot(x, y, 'ro')  
plt.show()
```



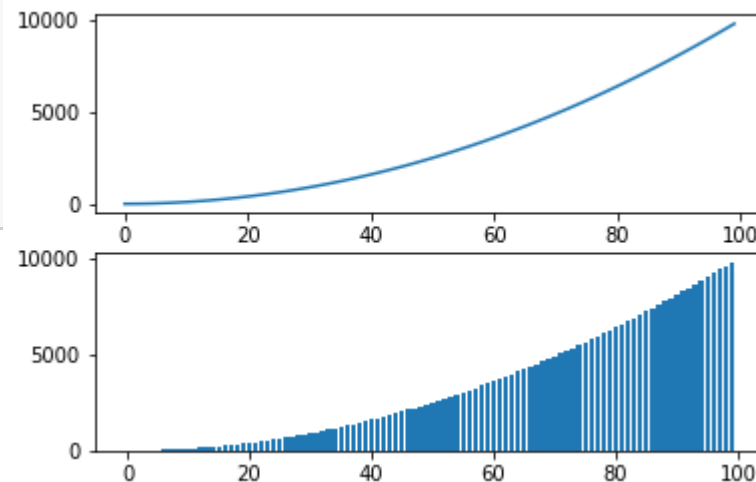
한 화면에 여러 개 그래프

- ◆ **figure()** : figure 객체 생성
- ◆ **add_subplot(위치 및 개수)** : 생성된 figure 객체에 subplot 추가
 - `add_subplot(2, 1, 1)` : 2x1 형태의 subplot, 두 개의 subplot 중 첫 번째
 - `add_subplot(2, 1, 2)` : 2x1 형태의 subplot, 두 개의 subplot 중 두 번째
- ◆ **add_subplot()** 함수 호출 시 **AxesSubplot** 객체 생성됨
- ◆ 생성된 subplot 객체를 그래프 drawing 함수와 연결

```
fig = plt.figure()           # figure object
ax1 = fig.add_subplot(2, 1, 1) # 1st subplot in 2x1
ax2 = fig.add_subplot(2, 1, 2) # 2nd subplot in 2x1
```

```
ax1.plot(x, y)               # line graph
ax2.bar(x, y)                 # bar graph

plt.show()                   # drawing figure
```



한 화면에 여러 개 그래프

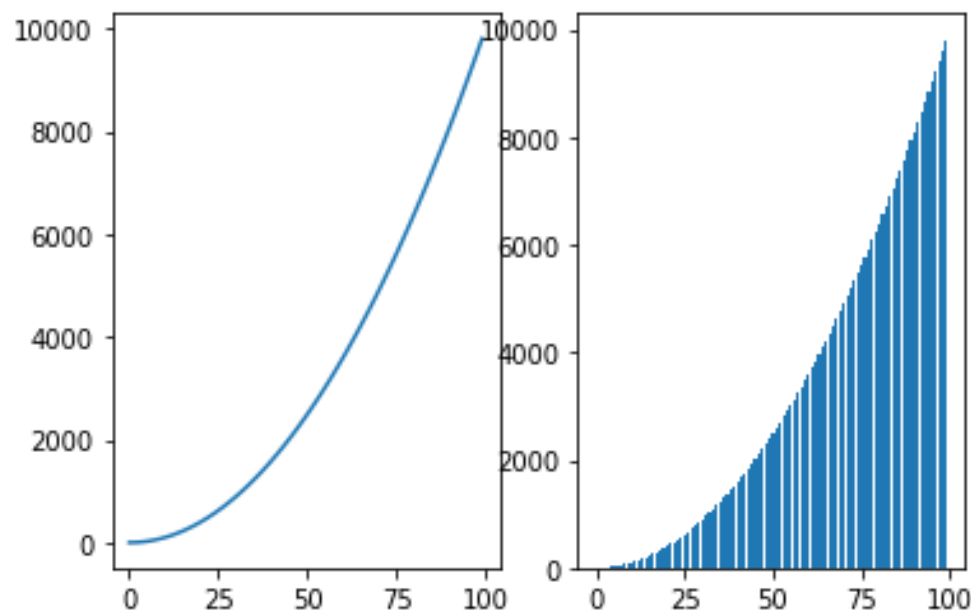
◆ 1x2 형태 그래프들

- `add_subplot(1, 2, 1)` : 1x2 형태의 subplot, 두 개의 subplot 중 첫 번째
- `add_subplot(1, 2, 2)` : 1x2 형태의 subplot, 두 개의 subplot 중 첫 번째

```
fig = plt.figure()           # figure object
ax1 = fig.add_subplot(1, 2, 1) # 1st subplot in 1x2
ax2 = fig.add_subplot(1, 2, 2) # 2nd subplot in 1x2

ax1.plot(x, y)               # line graph
ax2.bar(x, y)                 # bar graph

plt.show()                   # drawing figure
```



sine, cosine 그래프

◆ 파이썬의 기본 math 모듈 사용

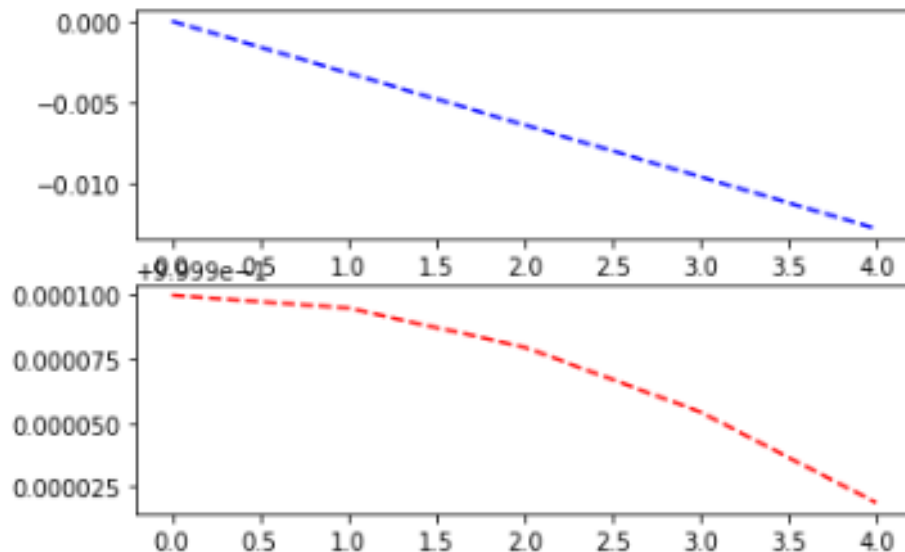
```
import math

x = range(5)
sin_y = [math.sin(2*3.14*i) for i in x]
cos_y = [math.cos(2*3.14*i) for i in x]

fig = plt.figure()           # figure object
ax1 = fig.add_subplot(2, 1, 1) # 1st subplot in 2x1
ax2 = fig.add_subplot(2, 1, 2) # 2nd subplot in 2x1

ax1.plot(x, sin_y, 'b--')     # sine graph
ax2.plot(x, cos_y, 'r--')     # cosine graph

plt.show()                   # drawing figure
```



정수 0~5 사이 범위의 값에서
range(0,5)로는 시간의 값을
촘촘하게 만들 수 없음

Numpy 모듈

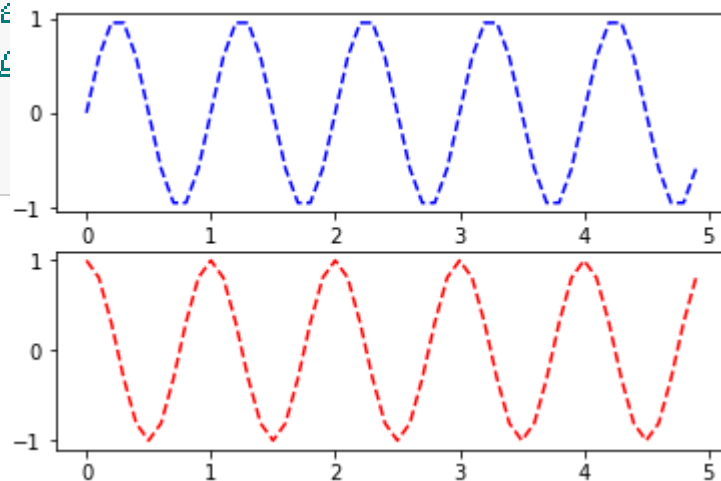
- ◆ Numpy : Numerical Python의 약자. 행렬 연산이나 수치 계산에 자주 사용
- ◆ `numpy.arange()` 함수 : 실수 단위로도 step 값 가능
 - `arange(0.0, 5.0, 0.1)` : 0.0 ~ 5.0 사이에서 0.1 간격으로 값 생성
- ◆ `numpy.pi` : π 값

```
import numpy as np

fig = plt.figure()           # figure object
ax1 = fig.add_subplot(2, 1, 1) # 1st subplot in 2x1
ax2 = fig.add_subplot(2, 1, 2) # 2nd subplot in 2x1

t = np.arange(0.0, 5.0, 0.1)
ax1.plot(t, np.sin(2*np.pi*t), 'b--') # sine graph
ax2.plot(t, np.cos(2*np.pi*t), 'r--') # cosine graph

plt.show()                   # drawing figure
```

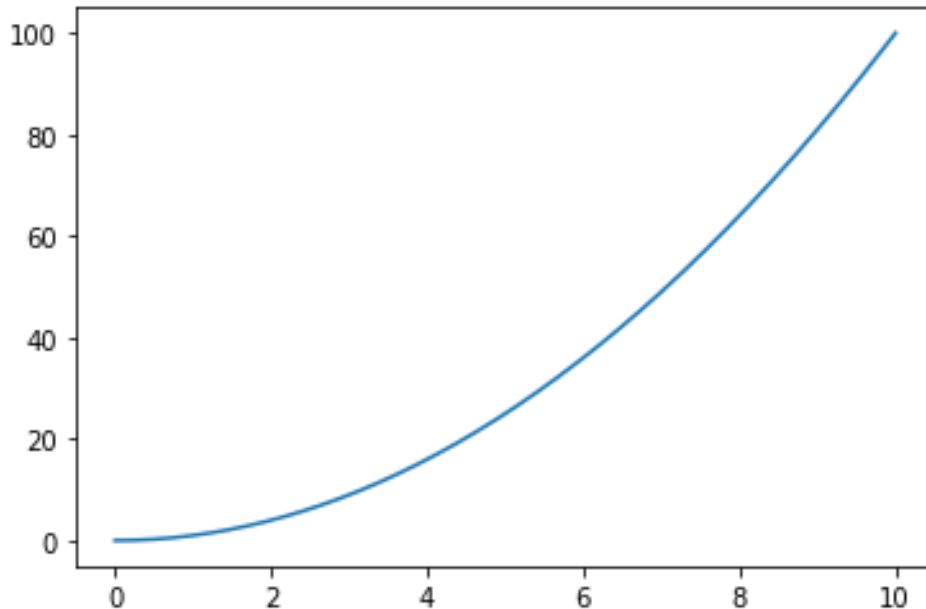


Numpy 기반 값 생성

- ◆ `numpy.linspace(start, end, num-points)` : 시작점과 끝점을 균일 간격으로 나눈 값 생성
- ◆ `numpy.power(x, y)` : x 의 y 제곱 값

```
x = np.linspace(0, 10, 1000)
y = np.power(x, 2)

plt.plot(x, y)
plt.show()
```

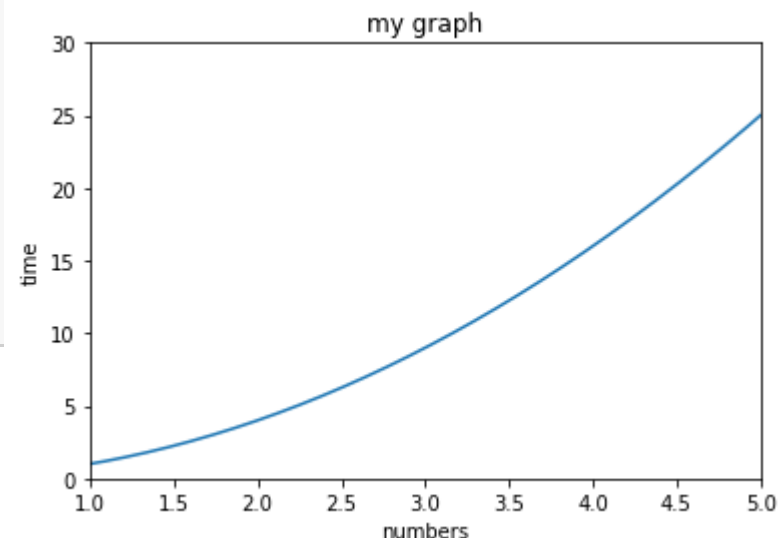


그래프 라벨 및 범위 표시

- ◆ `title(문자열)` : 그래프 제목
- ◆ `xlabel(문자열)` : x축 라벨
- ◆ `ylabel(문자열)` : y축 라벨
- ◆ `xlim((시작값, 끝값))` : x축 값 범위
- ◆ `ylim((시작값, 끝값))` : y축 값 범위

```
plt.title("my graph")      # title
plt.xlabel("numbers")      # x label
plt.ylabel("time")         # y label
plt.xlim((1, 5))          # x value range
plt.ylim((0, 30))         # y value range

plt.plot(x,y)
plt.show()
```



multiple lines

- ◆ **legend()** : 범례 추가, loc 파라미터 = 표시 위치

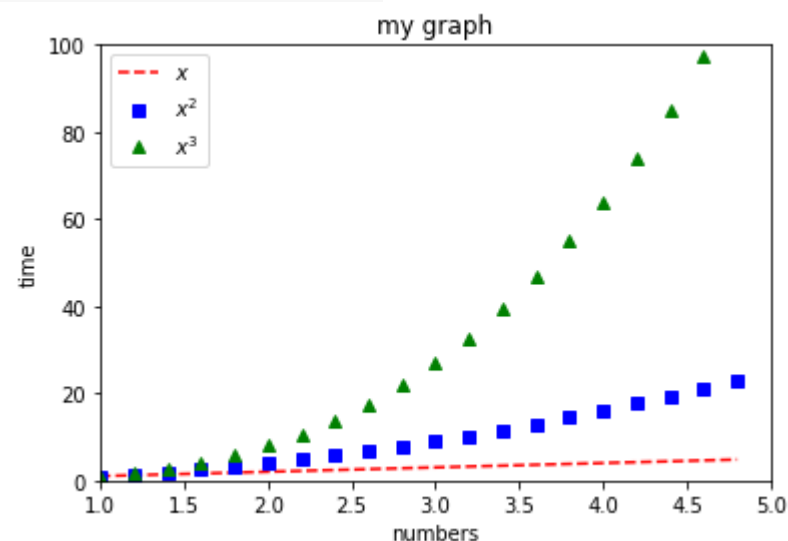
```
plt.title("my graph")      # title
plt.xlabel("numbers")      # x label
plt.ylabel("time")         # y label
plt.xlim((1, 5))          # x value range
plt.ylim((0, 100))        # y value range

x = np.arange(0.0, 5.0, 0.2)

plt.plot(x, x, 'r--', label="$x$")
plt.plot(x, np.power(x, 2), 'bs', label="$x^2$")
plt.plot(x, np.power(x, 3), 'g^', label="$x^3$")

plt.legend(loc="upper left")
plt.show()
```

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10



histogram

- ◆ **numpy.random** 모듈 **randn()** 함수 : 임의의 표준정규분포 데이터 생성
- ◆ **matplotlib** 모듈 **hist()** 함수 : 히스토그램 그리기
 - bin=나누는 구간
 - cumulative=누적 옵션
- ◆ **grid(True)** : 그리드 배경
- ◆ **savefig("파일명")** 파일로 저장

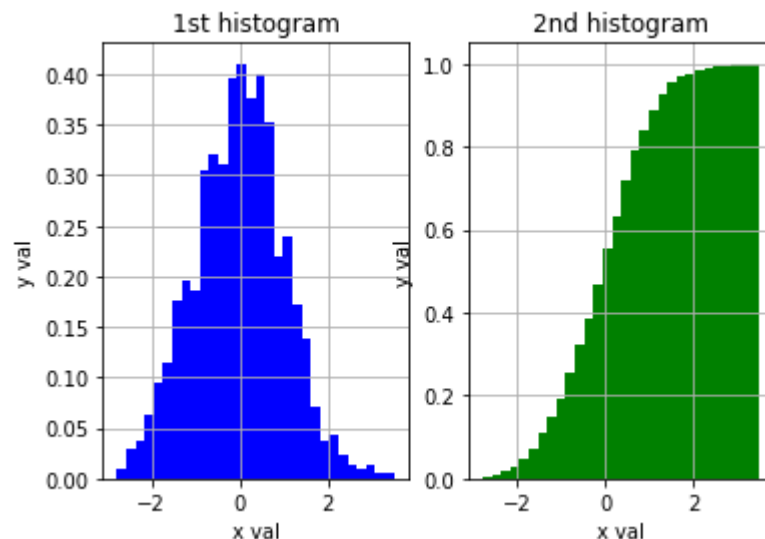
```
data = np.random.randn(1000)      # generate random number

fig = plt.figure()                # figure object
ax1 = fig.add_subplot(1, 2, 1)    # 1st subplot in 2x1
ax2 = fig.add_subplot(1, 2, 2)    # 2nd subplot in 2x1

ax1.set_title("1st histogram")    # 1st graph's title
ax1.set_xlabel("x val")           # x label
ax1.set_ylabel("y val")           # y label
ax1.hist(data, bins=30, density=True, color='b')
ax1.grid(True)

ax2.set_title("2nd histogram")    # 1st graph's title
ax2.set_xlabel("x val")           # x label
ax2.set_ylabel("y val")           # y label
ax2.hist(data, bins=30, density=True, color='g', cumulative=True)
ax2.grid(True)

plt.show()
plt.savefig("histogram.png")
```



Horizontal bar

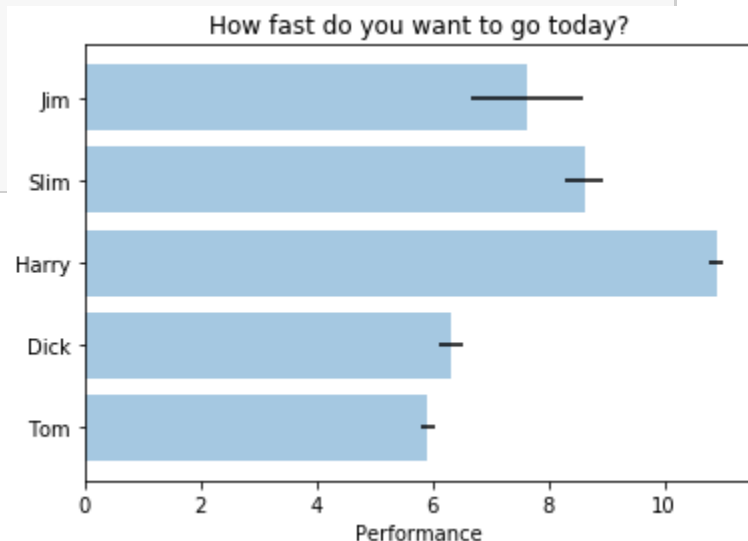
- ◆ `barh()` 함수 : 수평 차트 그리기
- ◆ `yticks()` 함수 : `ticker` 위치별 각 위치에서의 라벨 설정

```
# Example data
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim')

y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
error = np.random.rand(len(people))

plt.barh(y_pos, performance, xerr=error, align='center', alpha=0.4)
plt.yticks(y_pos, people)
plt.xlabel('Performance')
plt.title('How fast do you want to go today?')

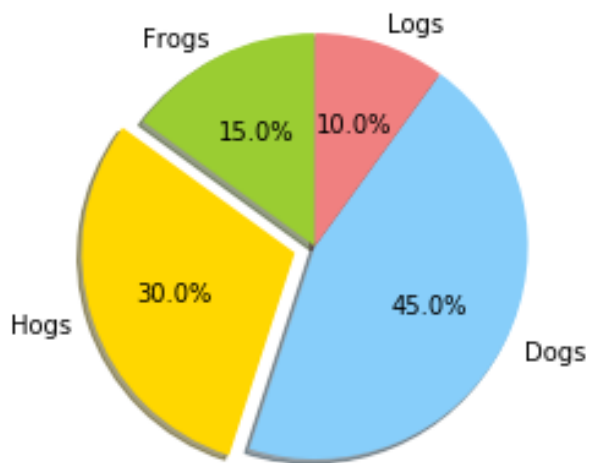
plt.show()
```



원 그래프

◆ pie () 함수 : 원 그래프 그리기

```
# The slices will be ordered and plotted counter-clockwise.  
labels = ('Frogs', 'Hogs', 'Dogs', 'Logs')  
sizes = [15, 30, 45, 10]  
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']  
explode = (0, 0.1, 0, 0)    # only "explode" the 2nd slice (i.e. 'Hogs')  
  
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=90)  
plt.show()
```



3D 그래프

- ◆ `mpl_toolkits.mplot3d` 모듈 추가
- ◆ `scatter()` : 점 그래프 그리기

```
from mpl_toolkits.mplot3d import Axes3D

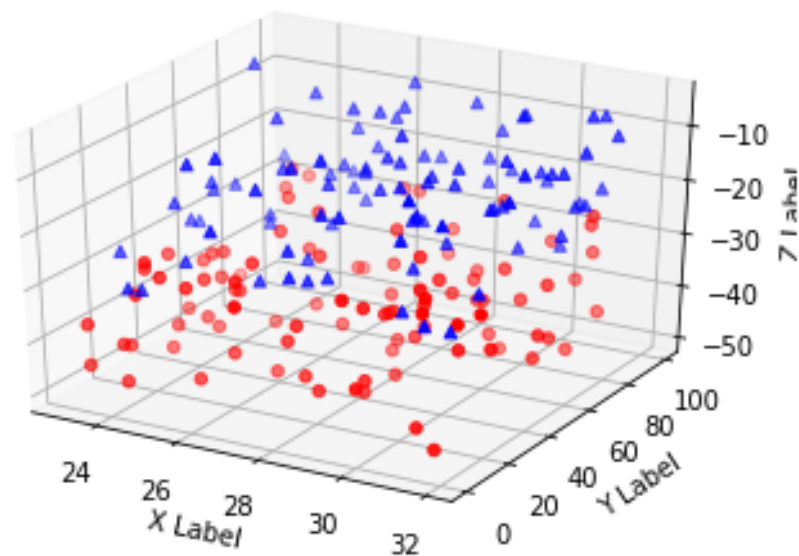
def randrange(n, min, max):
    return (max - min) * np.random.rand(n) + min

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection='3d')
n = 100

for c, m, zl, zh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
    xs = randrange(n, 23, 32)
    ys = randrange(n, 0, 100)
    zs = randrange(n, zl, zh)
    ax.scatter(xs, ys, zs, c=c, marker=m)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```



NumPy

◆ 대용량 데이터 배열을 효율적으로 다룰 수 있도록 설계됨

- NumPy는 데이터를 다른 내장 파이썬 객체와 구분되는 연속된 메모리 블록에 저장
- NumPy의 각종 알고리즘은 모두 C로 작성되어 타입 검사나 다른 오버헤드 없이 메모리를 직접 조작할 수 있음
- NumPy 배열은 내장 파이썬의 연속된 자료형들보다 훨씬 더 적은 메모리를 사용함
- NumPy 연산은 파이썬 반복문을 사용하지 않고 전체 배열에 대한 복잡한 계산을 수행함

NumPy 배열 vs. Python 리스트

◆ 성능 비교

```
import numpy as np
my_arr = np.arange(1000000)    # numpy array
my_list = list(range(1000000)) # python list
```

```
%time for _ in range(10): my_arr2 = my_arr * 2
```

Wall time: 27 ms

```
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]
```

Wall time: 797 ms

NumPy ndarray : 다차원 배열 객체

◆ ndarray

- 같은 종류의 데이터를 담을 수 있는 포괄적인 다차원 배열
- 모든 원소는 같은 자료형이어야 함

```
# Generate some random data  
data = np.random.randn(2, 3)  
data
```

```
array([[ 0.29548381, -0.0796997 ,  0.29164707],  
       [-1.51937054, -0.5403895 ,  0.41363654]])
```

```
data * 10
```

```
array([[ 2.9548381 , -0.79699699,  2.91647073],  
       [-15.19370545, -5.40389499,  4.13636544]])
```

```
data + data
```

```
array([[ 0.59096762, -0.1593994 ,  0.58329415],  
       [-3.03874109, -1.080779 ,  0.82727309]])
```

NumPy ndarray : 다차원 배열 객체

◆ shape

- 각 차원의 크기를 알려줌

◆ dtype

- 튜플과 배열에 저장된 자료형을 알려줌

```
data.shape
```

```
(2, 3)
```

```
data.dtype
```

```
dtype('float64')
```


ndarray 생성하기

◆ array 함수

- 배열 혹은 순차적인 객체를 전달 받아 새로운 NumPy 배열 생성

```
data1 = [6, 7.5, 8, 0, 1]  
arr1 = np.array(data1)  
arr1
```

```
array([6. , 7.5, 8. , 0. , 1. ])
```

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]  
arr2 = np.array(data2)  
arr2
```

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

ndarray 생성하기

◆ ndim

- 차원 확인

```
arr2.ndim
```

```
2
```

```
arr2.shape
```

```
(2, 4)
```

```
arr1.dtype
```

```
dtype('float64')
```

```
arr2.dtype
```

```
dtype('int32')
```

ndarray 생성하기

◆ zeros, ones 함수

- 주어진 길이나 모양에 각각 0과 1이 들어 있는 배열 생성

◆ empty 함수

- 초기화되지 않은 배열 생성

◆ 생성 시 원하는 형태의 튜플을 함수로 전달

```
np.zeros(10)
```

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

```
np.zeros((3, 6))
```

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

```
np.empty((2, 3, 2))
```

```
array([[[1.29702558e-311, 3.16202013e-322],  
        [0.00000000e+000, 0.00000000e+000],  
        [0.00000000e+000, 7.71625812e-043]],  
       [[2.00376266e-052, 4.75263091e+174],  
        [8.38127398e+165, 1.09409880e-042],  
        [3.50821702e-033, 7.60371476e-042]]])
```

ndarray 생성하기

◆ arange 함수

- 파이썬 range 함수의 배열 버전

```
np.arange(15)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

ndarray의 dtype

◆ dtype

- ndarray가 메모리에 있는 특정 데이터를 해석하기 위해 필요한 정보(또는 메타데이터)를 담고 있는 특수한 객체
- C 언어 등으로 작성된 코드와 쉽게 연동 가능

```
arr1 = np.array([1, 2, 3], dtype=np.float64)
arr2 = np.array([1, 2, 3], dtype=np.int32)
arr1.dtype
```

```
dtype('float64')
```

```
arr2.dtype
```

```
dtype('int32')
```

Type	Type Code	Description
int8, uint8	i1, u1	Signed and unsigned 8-bit (1 byte) integer types
int16, uint16	i2, u2	Signed and unsigned 16-bit integer types
int32, uint32	i4, u4	Signed and unsigned 32-bit integer types
int64, uint64	i8, u8	Signed and unsigned 32-bit integer types
float16	f2	Half-precision floating point
float32	f4 or f	Standard single-precision floating point. Compatible with C float
float64, float128	f8 or d	Standard double-precision floating point. Compatible with C double and Python floatobject
float128	f16 or g	Extended-precision floating point
complex64, complex128, complex256	c8, c16, c32	Complex numbers represented by two 32, 64, or 128 floats, respectively
bool	?	Boolean type storing True and False values
object	O	Python object type
string_	S	Fixed-length string type (1 byte per character). For example, to create a string dtype with length 10, use 'S10'.
unicode_	U	Fixed-length unicode type (number of bytes platform specific). Same specification semantics as string_ (e.g. 'U10').

◆ NumPy data type

ndarray의 dtype

◆ astype 메소드

- 명시적 타입 변환 (캐스팅)
- 정수형 → 부동 소수점형

```
arr = np.array([1, 2, 3, 4, 5])  
arr.dtype
```

```
dtype('int32')
```

```
float_arr = arr.astype(np.float64)  
float_arr.dtype
```

```
dtype('float64')
```

- 부동소수점형 → 정수형 (소수점 아래 자리 버려짐)

```
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])  
arr
```

```
array([ 3.7, -1.2, -2.6,  0.5, 12.9, 10.1])
```

```
arr.astype(np.int32)
```

```
array([ 3, -1, -2,  0, 12, 10])
```

ndarray의 dtype

- 숫자 형식 문자열 → 숫자

※ 주의: NumPy에서 문자열 데이터는 고정 크기를 가지며 별다른 경고를 출력하지 않고 입력을 임의로 잘라낼 수 있음

```
numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
numeric_strings.astype(float)

array([ 1.25, -9.6 , 42.  ])
```

- 다른 배열의 dtype 속성 이용

※ 주의: astype을 호출하면 새로운 dtype이 이전 dtype과 동일해도 항상 새로운 배열을 생성(데이터를 복사)

```
int_array = np.arange(10)
int_array

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

```
calibers = np.array([.22, .270, .357, .380, .44, .50], dtype=np.float64)
int_array.astype(calibers.dtype)

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

- 축약 코드 사용 가능 `uint32`

```
empty_uint32 = np.empty(8, dtype='u4')
empty_uint32

array([          0, 1075314688,           0, 1075707904,           0,
        1075838976,           0, 1072693248], dtype=uint32)
```

NumPy 배열의 산술 연산

- ◆ 벡터화 : for 문을 사용하지 않고 데이터를 일괄 처리 가능
 - 같은 크기의 배열 간의 산술 연산은 배열의 각 원소 단위로 적용
 - 크기가 다른 배열 간의 연산 : 브로드캐스팅

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])  
arr * arr
```

```
array([[ 1.,  4.,  9.],  
       [16., 25., 36.]])
```

```
arr - arr
```

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

```
1 / arr
```

```
array([[1.      , 0.5      , 0.33333333],  
       [0.25     , 0.2      , 0.16666667]])
```

```
arr ** 0.5
```

```
array([[1.      , 1.41421356, 1.73205081],  
       [2.      , 2.23606798, 2.44948974]])
```

```
arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])  
arr2 > arr
```

```
array([[False,  True, False],  
       [ True, False,  True]])
```


NumPy 색인 및 슬라이싱

- ◆ 조작 시 데이터 복사가 아니라 원본 배열에 그대로 반영됨
- ◆ 대용량 데이터 처리 염두 (복사 시에는 `copy` 함수 사용)

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
arr2d[2]
```

```
array([7, 8, 9])
```

```
arr2d[0][2]
```

```
3
```

```
arr2d[0, 2]
```

```
3
```

```
arr2d[:2]
```

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

```
arr2d[:2, 1:]
```

```
array([[2, 3],  
       [5, 6]])
```

```
arr2d[1, :2]
```

```
array([4, 5])
```

```
arr2d[2, 2]
```

```
array([3, 6])
```

```
arr2d[:, :1]
```

```
array([[1],  
       [4],  
       [7]])
```

```
arr2d[:2, 1:] = 0  
arr2d
```

```
array([[1, 0, 0],  
       [4, 0, 0],  
       [7, 8, 9]])
```

Boolean Indexing

◆ Boolean 배열을 배열의 색인으로 사용

```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])  
data = np.random.randn(7, 4)
```

```
names
```

```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')
```

```
data
```

```
array([[ 0.24710767, -2.49375296,  1.1874322 ,  0.14683878],  
       [-0.57630713,  0.86108617, -1.58105064, -0.01155967],  
       [-0.60570971,  1.23361679, -0.02048547,  1.38753089],  
       [ 0.21440945,  1.82681938,  2.41173904, -0.92674394],  
       [-0.07049914, -0.69409118,  1.90912179, -0.5777343 ],  
       [-1.34994767, -0.72320016, -0.60883865,  0.73959355],  
       [ 1.7435593 , -0.34954507, -1.79965788, -1.22392568]])
```

```
names == 'Bob'
```

```
array([ True, False, False,  True, False, False, False])
```

```
data[names == 'Bob']
```

```
array([[ 0.24710767, -2.49375296,  1.1874322 ,  0.14683878],  
       [ 0.21440945,  1.82681938,  2.41173904, -0.92674394]])
```

```
data[names == 'Bob', 2:]
```

```
array([[ 1.1874322 ,  0.14683878],  
       [ 2.41173904, -0.92674394]])
```

Boolean Index

```
data[names == 'Bob', 3]
```

```
array([ 0.14683878, -0.92674394])
```

```
data[~(names == 'Bob')]
```

```
array([[ -0.57630713,  0.86108617, -1.58105064, -0.01155967],  
       [ -0.60570971,  1.23361679, -0.02048547,  1.38753089],  
       [ -0.07049914, -0.69409118,  1.90912179, -0.5777343 ],  
       [ -1.34994767, -0.72320016, -0.60883865,  0.73959355],  
       [ 1.7435593 , -0.34954507, -1.79965788, -1.22392568]])
```

```
mask = (names == 'Bob') | (names == 'Will')  
data[mask]
```

```
array([[ 0.24710767, -2.49375296,  1.1874322 ,  0.14683878],  
       [ -0.60570971,  1.23361679, -0.02048547,  1.38753089],  
       [ 0.21440945,  1.82681938,  2.41173904, -0.92674394],  
       [ -0.07049914, -0.69409118,  1.90912179, -0.5777343 ]])
```

```
data[data < 0] = 0  
data
```

```
array([[0.24710767, 0.          , 1.1874322 , 0.14683878],  
       [0.          , 0.86108617, 0.          , 0.          ],  
       [0.          , 1.23361679, 0.          , 1.38753089],  
       [0.21440945, 1.82681938, 2.41173904, 0.          ],  
       [0.          , 0.          , 1.90912179, 0.          ],  
       [0.          , 0.          , 0.          , 0.73959355],  
       [1.7435593 , 0.          , 0.          , 0.          ]])
```

```
data[names != 'Joe'] = 7  
data
```

```
array([[7.          , 7.          , 7.          , 7.          ],  
       [0.          , 0.86108617, 0.          , 0.          ],  
       [7.          , 7.          , 7.          , 7.          ],  
       [7.          , 7.          , 7.          , 7.          ],  
       [7.          , 7.          , 7.          , 7.          ],  
       [0.          , 0.          , 0.          , 0.73959355],  
       [1.7435593 , 0.          , 0.          , 0.          ]])
```

Fancy Indexing

◆ 정수 배열을 사용한 indexing

```
arr = np.empty((8, 4))  
for i in range(8):  
    arr[i] = i  
arr
```

```
array([[0., 0., 0., 0.],  
       [1., 1., 1., 1.],  
       [2., 2., 2., 2.],  
       [3., 3., 3., 3.],  
       [4., 4., 4., 4.],  
       [5., 5., 5., 5.],  
       [6., 6., 6., 6.],  
       [7., 7., 7., 7.]])
```

```
arr[[4, 3, 0, 6]]
```

```
array([[4., 4., 4., 4.],  
       [3., 3., 3., 3.],  
       [0., 0., 0., 0.],  
       [6., 6., 6., 6.]])
```

```
arr[[-3, -5, -7]]
```

```
array([[5., 5., 5., 5.],  
       [3., 3., 3., 3.],  
       [1., 1., 1., 1.]])
```

Fancy Indexing

- ◆ 다차원 배열의 fancy indexing 결과는 항상 1차원

```
arr = np.arange(32).reshape((8, 4))  
arr
```

```
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11],  
       [12, 13, 14, 15],  
       [16, 17, 18, 19],  
       [20, 21, 22, 23],  
       [24, 25, 26, 27],  
       [28, 29, 30, 31]])
```

```
arr[[1, 5, 7, 2], [0, 3, 1, 2]]  
array([ 4, 23, 29, 10])
```

```
arr[[1, 5, 7, 2]][:, [0, 3, 1, 2]]  
array([[ 4,  7,  5,  6],  
       [20, 23, 21, 22],  
       [28, 31, 29, 30],  
       [ 8, 11,  9, 10]])
```

- ◆ 행렬의 row와 column에 대응하는 값 선택되길 기대
- ◆ fancy indexing은 slicing과 달리 선택된 데이터를 새로운 배열로 복사

배열 전치와 축 바꾸기

◆ reshape : 내부 데이터 보존한 채 형태 변경

- 숫자는 -1 사용 가능. 계산되어 사용

```
arr = np.arange(15).reshape((3, 5))  
arr
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
arr.reshape(5, -1)
```

```
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11],  
       [12, 13, 14]])
```

```
arr.reshape(4, -1)
```

```
-----  
-  
ValueError                                Traceback (most recent call last)  
<ipython-input-79-4e31163886f5> in <module>  
----> 1 arr.reshape(4, -1)
```

ValueError: cannot reshape array of size 15 into shape (4,newaxis)

배열 전치와 축 바꾸기

◆ T : 축을 뒤바꾸는 간단한 전치

```
arr
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
arr.T
```

```
array([[ 0,  5, 10],  
       [ 1,  6, 11],  
       [ 2,  7, 12],  
       [ 3,  8, 13],  
       [ 4,  9, 14]])
```

배열 전치와 축 바꾸기

- ◆ **transpose** : 축 번호 받아서 치환
- ◆ **swapaxes** : 두 개의 축 번호 받아서 배열 뒤바꿈

```
arr = np.arange(16).reshape((2, 2, 4))  
arr
```

```
array([[[ 0,  1,  2,  3],  
        [ 4,  5,  6,  7]],  
       [[ 8,  9, 10, 11],  
        [12, 13, 14, 15]]])
```

```
arr.transpose((1, 0, 2))
```

```
array([[[ 0,  1,  2,  3],  
        [ 8,  9, 10, 11]],  
       [[ 4,  5,  6,  7],  
        [12, 13, 14, 15]]])
```

```
arr.swapaxes(1, 2)
```

```
array([[[ 0,  4],  
        [ 1,  5],  
        [ 2,  6],  
        [ 3,  7]],  
       [[ 8, 12],  
        [ 9, 13],  
        [10, 14],  
        [11, 15]]])
```


Numpy 조건절 표현

◆ where 함수

- 'x if 조건 else y'와 같은 삼항식의 벡터화된 버전
- 다른 배열에 기반한 새로운 배열 생성

```
xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])  
yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])  
cond = np.array([True, False, True, True, False])
```

```
result = [(x if c else y)  
          for x, y, c in zip(xarr, yarr, cond)]  
result
```

```
[1.1, 2.2, 1.3, 1.4, 2.5]
```

```
result = np.where(cond, xarr, yarr)  
result
```

```
array([1.1, 2.2, 1.3, 1.4, 2.5])
```

NumPy Universal 함수

Function	Description
<code>abs</code> , <code>fabs</code>	Compute the absolute value element-wise for integer, floating-point, or complex values
<code>sqrt</code>	Compute the square root of each element (equivalent to <code>arr ** 0.5</code>)
<code>square</code>	Compute the square of each element (equivalent to <code>arr ** 2</code>)
<code>exp</code>	Compute the exponent e^x of each element
<code>log</code> , <code>log10</code> , <code>log2</code> , <code>log1p</code>	Natural logarithm (base e), log base 10, log base 2, and $\log(1 + x)$, respectively
<code>sign</code>	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
<code>ceil</code>	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
<code>floor</code>	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
<code>rint</code>	Round elements to the nearest integer, preserving the <code>dtype</code>
<code>modf</code>	Return fractional and integral parts of array as a separate array
<code>isnan</code>	Return boolean array indicating whether each value is NaN (Not a Number)
<code>isfinite</code> , <code>isinf</code>	Return boolean array indicating whether each element is finite (non- <code>inf</code> , non-NaN) or infinite, respectively
<code>cos</code> , <code>cosh</code> , <code>sin</code> , <code>sinh</code> , <code>tan</code> , <code>tanh</code>	Regular and hyperbolic trigonometric functions
<code>arccos</code> , <code>arccosh</code> , <code>arcsin</code> , <code>arcsinh</code> , <code>arctan</code> , <code>arctanh</code>	Inverse trigonometric functions
<code>logical_not</code>	Compute truth value of <code>not x</code> element-wise (equivalent to <code>~arr</code>).

NumPy Universal 함수

Function	Description
<code>add</code>	Add corresponding elements in arrays
<code>subtract</code>	Subtract elements in second array from first array
<code>multiply</code>	Multiply array elements
<code>divide, floor_divide</code>	Divide or floor divide (truncating the remainder)
<code>power</code>	Raise elements in first array to powers indicated in second array
<code>maximum, fmax</code>	Element-wise maximum; <code>fmax</code> ignores NaN
<code>minimum, fmin</code>	Element-wise minimum; <code>fmin</code> ignores NaN
<code>mod</code>	Element-wise modulus (remainder of division)
<code>copysign</code>	Copy sign of values in second argument to values in first argument

pandas

- ◆ 표 형식의 데이터나 다양한 형태의 데이터를 다루는 데 초점을 맞춰 설계된 라이브러리
- ◆ NumPy 배열 기반 계산 스타일을 많이 차용
 - NumPy는 단일 산술 배열 데이터를 다루는 데 특화됨
 - Pandas는 표 형식 데이터를 다루는데 특화됨
- ◆ 자료구조
 - Series
 - 일련의 객체를 담을 수 있는 1차원 배열 같은 자료구조
 - 어떤 NumPy 자료형이라도 담을 수 있음
 - 색인 – 배열의 연관 구조
 - DataFrame
 - 표 같은 스프레드시트 형식의 자료구조
 - R의 데이터프레임에서 유래
 - 행과 열에 대한 색인을 가짐
 - 색인은 같은 Series 객체를 담고 있는 파이선 사전으로 간주

Series

◆ Series 생성

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

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

```
obj.values
```

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

```
obj.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])  
obj2
```

```
d    4  
b    7  
a   -5  
c    3  
dtype: int64
```

◆ 색인 지정 가능

Series

◆ 파이썬 딕셔너리 객체로부터 생성

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}  
obj3 = pd.Series(sdata)  
obj3
```

```
Ohio      35000  
Texas     71000  
Oregon    16000  
Utah       5000  
dtype: int64
```

```
states = ['California', 'Ohio', 'Oregon', 'Texas']  
obj4 = pd.Series(sdata, index=states)  
obj4
```

```
California    NaN  
Ohio          35000.0  
Oregon        16000.0  
Texas         71000.0  
dtype: float64
```

Series

◆ 값 찾기

```
pd.isnull(obj4)
```

```
California    True  
Ohio         False  
Oregon       False  
Texas        False  
dtype: bool
```

```
pd.notnull(obj4)
```

```
California    False  
Ohio         True  
Oregon       True  
Texas        True  
dtype: bool
```

```
obj4.isnull()
```

```
California    True  
Ohio         False  
Oregon       False  
Texas        False  
dtype: bool
```


Series

◆ 연산 색인, 라벨 자동 정렬

```
obj3
```

```
Ohio      35000  
Texas     71000  
Oregon    16000  
Utah       5000  
dtype: int64
```

```
obj4
```

```
California    NaN  
Ohio         35000.0  
Oregon        16000.0  
Texas         71000.0  
dtype: float64
```

```
obj3 + obj4
```

```
California    NaN  
Ohio         70000.0  
Oregon        32000.0  
Texas        142000.0  
Utah          NaN  
dtype: float64
```

Series

◆ 할당문으로 색인 변경

```
obj
```

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

```
obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
```

```
obj
```

```
Bob      4  
Steve    7  
Jeff    -5  
Ryan     3  
dtype: int64
```

DataFrame

◆ 딕셔너리를 이용한 DataFrame 생성

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],  
        'year': [2000, 2001, 2002, 2001, 2002, 2003],  
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}  
frame = pd.DataFrame(data)
```

frame

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

DataFrame

◆ 열 순서 지정하여 DataFrame 생성

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

◆ 열 추가

```
frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],  
                       index=['one', 'two', 'three', 'four',  
                              'five', 'six'])  
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

DataFrame

◆ 열 확인

```
frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

◆ 열 접근

```
frame2['state']
```

```
one      Ohio
two      Ohio
three    Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object
```

◆ 행 접근 (라벨로 색인)

```
frame2.loc['three']
```

```
year      2002
state     Ohio
pop        3.6
debt      NaN
Name: three, dtype: object
```

Any Questions...
Just Ask!

