

## 2025-04-08(Tue.) : 파이썬 라이브러리와 활용 3일차 - Numpy 2일차, Matplotlib 연습 1일차,

- [Python] Python 코드 실행시간 측정 4가지 방법 (feat. Jupyter Notebook)
  - 코드 실행시간 측정
    1. time 라이브러리 이용 (초 단위) ★ time.time()의 의미
    2. datetime 라이브러리 이용 (시:분:초 단위)
    3. %%timeit
      - 주피터 노트북에서 지원하는 매직 커맨드
      - 셀 맨 위에 %%timeit를 작성하면 해당 셀에 있는 코드를 반복 실행해서 더 정확하게 실행 시간을 측정
    4. %%time
      - 주피터 노트북에서 지원하는 매직 커맨드
      - %%timeit과 달리 한 번만 수행하지만 CPU 수행 시간과 실제 시간을 같이 출력
- Gemma 3
  - Gemma 3 모델 개요
- Matplotlib: Visualization with Python
  - ★ Pyplot tutorial
- pandas documentation
  - ★ API reference
- NumPy
  - ★ NumPy fundamentals
  - NumPy documentation
  - NumPy reference
    - [Sorting, searching, and counting](#)
      - [numpy.sort](#)
        - Return a sorted copy of an array.
      - [numpy.argsort](#)
        - Returns the indices that would sort an array.
      - [numpy.argmax](#)
        - Returns the indices of the maximum values along an axis.
      - [numpy.where](#)
        - Return elements chosen from x or y depending on condition.
      -
    - [Mathematical functions](#)
      - [numpy.exp](#)
        - Calculate the exponential of all elements in the input array.
      - [numpy.sqrt](#)
        - Return the non-negative square-root of an array, element-wise.
      - [numpy.sin](#)
        - Trigonometric sine, element-wise.
    - [Linear algebra \(numpy.linalg\)](#)
      - [numpy.dot](#)
    - [Statistics](#)
      - [numpy.std](#)
        - Compute the standard deviation along the specified axis.

- `numpy.mean`

- Compute the arithmetic mean along the specified axis.

- 점프 투 파이썬 - 라이브러리 예제 편
- 점프 투 파이썬
  - [pahkey/jump2python](#)
- 파이썬 - 기본을 알고 닦자!
- Python Snippets - 파이썬 조각 코드 모음집
- 왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문법과 실습
- Kaggle에서 파이썬으로 데이터 분석 시작하기
  - 02-데이터분석 라이브러리
    - 01-Pandas
    - 02-NumPy
    - 03-scikit-learn
    - 04-matplotlib
- ★★★★★ [Python 완전정복 시리즈] 2편 : Pandas DataFrame 완전정복
  - 01. 객체 간 연산
  - 02. 객체 내 연산
  - 03. 함수 적용
  - 04. 인덱싱
  - 05. 비교 & 필터링
  - 06. 결측제어
  - 07. 정렬
  - 08. 결합
  - 09. 가공
  - 10. 정보
  - 11. 데이터타입
  - 12. 확인
  - 13. 축 및 레이블
  - 14. 통계 (기초)
  - 15. 통계 (심화)
  - 16. 시간
  - 17. 멀티인덱스 (Multi Index)
  - 18. 반복
  - 19. 형식 변환
  - 20. 플로팅 (plot)
- ★★★★★ 공학자를 위한 Python
  - NumPy
    - 3.1 ndarray
    - 3.2 인덱싱과 합치기
    - 3.3 연산
    - 3.4 브로드캐스팅
    - 3.5 복사
    - 3.6 데이터 읽고 쓰기
    - 3.7 타입과 structured array
    - 3.8 기타
  - 4. Matplotlib

- 4.1 기본 사용법
- 4.2 IPython
- 4.3 Matplotlib의 이해
- 4.4 코드 조각
- 4.5 3차원 그래픽
- 4.4.1 Line and scatter plot
- 4.4.2 Surface plot
- 4.4.3 bar3d plot
- ☆☆☆ 빠르게 따라하는 Python
  - 2. 파이썬 기초
  - 5. 파이썬 응용
    - 5.1. 파이썬 라이브러리
    - 5.2. Numpy
    - 5.3. 판다스
    - 5.4. 플라스크
- Github 블로그 만들기 - 1. 시작하기
- The Python Tutorial (v.3.13)
- 파이썬 자습서 (v.3.13)
- yeonsookim-wt/lgtm
- 점프 투 파이썬
- ☆ 파이썬 독학하기 좋은 위키독스 모음집
- ☆ Jupyter notebook 단축키
- [Jupyter 주피터] 주요 단축키 모음
- ☆ 주피터 노트북(Jupyter Notebook) 사용법 - 기본 설치/실행, 단축키, 매직 명령어, Markdown, 테마스킨, nbextensions
- 기초탄탄 파이썬\_데이터 시각화
  - 02장 판다스 pandas
- 파이썬 마스터하기 : Pandas
- 파이썬으로 배우는 알고리즘 트레이딩 (개정판-2쇄)
  - 13. Pandas를 이용한 데이터 분석 기초 (revision)
- 파이썬을 이용한 비트코인 자동매매 (개정판)
- Git 설치 및 환경설정(mac os)
- pandas apply 함수와 lambda 설명
- 03-01. 축 기준 (apply)
- Jupyter Notebook에서 Magic Command 사용하기
- 01-IPython 매직커맨드
- pandas.DataFrame.transform
- 판다스 pandas IQR 활용해서 이상점(outlier) 찾고 삭제하기

## ◆ matplotlib

- Matplotlib: Visualization with Python
  - Pyplot tutorial

```
# 사용자함수 정의
def pprint(arr: np.ndarray):
```

```

print(f"type: {type(arr)}")
print(f"shape: {arr.shape}")
print(f"ndim: {arr.ndim}")
print(f"dtype: {arr.dtype}")

a = np.linspace(0,1,5)
pprint(a)

# matplotlib – 파이썬의 생태계에서 가장 근본적인 차트 라이브러리
# from matplotlib import pyplot as plt
import matplotlib.pyplot as plt

# numpy 데이터
# 그
plt.plot(a, 'o')
plt.show()

```

### ◆ matplotlib.pyplot.plot

```
matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None,
**kwargs)
```

- Plot y versus x as lines and/or markers.
- Call signatures:

```

plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

```

- The coordinates of the points or line nodes are given by x, y.
- The optional parameter fmt is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the Notes section below.

```

plot(x, y)          # plot x and y using default line style and
color
plot(x, y, 'bo')    # plot x and y using blue circle markers
plot(y)             # plot y using x as index array 0..N-1
plot(y, 'r+')       # ditto, but with red plusses

```

### ◆ matplotlib.pyplot.show

```
matplotlib.pyplot.show(*, block=None)
```

- Display all open figures.

## np.logspace

```
numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)
```

- ◆ [numpy.logspace](#)
  - Return numbers spaced evenly on a log scale.

```
a = np.logspace(0.1, 1, 20, endpoint=True)
pprint(a)

from matplotlib import pyplot as plt
plt.plot(a, 'o')
plt.show()
```

## matplotlib.pyplot.hist

```
matplotlib.pyplot.hist(x, bins=None, *, range=None, density=False,
weights=None, cumulative=False, bottom=None, histtype='bar', align='mid',
orientation='vertical', rwidth=None, log=False, color=None, label=None,
stacked=False, data=None, **kwargs)[source]
```

- Compute and plot a histogram.

## 약속된 난수

### ◆ [numpy.random.seed](#)

```
random.seed(seed=None)
```

- Reseed the singleton RandomState instance.

## NumPy 입출력

### [numpy.save](#)

```
numpy.save(file, arr, allow_pickle=True, fix_imports=<no value>)
```

- Save an array to a binary file in NumPy .npy format.

### [numpy.savez](#)

```
numpy.savez(file, *args, allow_pickle=True, **kwargs)
```

- Save several arrays into a single file in uncompressed .npz format.

```
>>> import numpy as np
>>> from tempfile import TemporaryFile
>>> outfile = TemporaryFile()
>>> x = np.arange(10)
>>> y = np.sin(x)

>>> np.savez(outfile, x, y)
>>> _ = outfile.seek(0) # Only needed to simulate closing & reopening
file
>>> npzfile = np.load(outfile)
>>> npzfile.files
['arr_0', 'arr_1']
npzfile['arr_0']
>>> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

## numpy.load

```
numpy.load(file, mmap_mode=None, allow_pickle=False, fix_imports=True,
encoding='ASCII', *, max_header_size=10000)
```

- Load arrays or pickled objects from .npy, .npz or pickled files.

## numpy.savetxt

```
numpy.savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n',
header='', footer='', comments='# ', encoding=None)[source]
```

- Save an array to a text file.

```
import numpy as np
x = y = z = np.arange(0.0,5.0,1.0)
np.savetxt('test.out', x, delimiter=',') # X is an array
np.savetxt('test.out', (x,y,z)) # x,y,z equal sized 1D arrays
np.savetxt('test.out', x, fmt='%1.4e') # use exponential notation
```

## numpy.loadtxt

```
numpy.loadtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None,
converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0,
encoding=None, max_rows=None, *, quotechar=None, like=None)
```

- Load data from a text file.

---

```
%timeit a = np.random.randn(2,4)
pprint(a)

np.save("./a.npy", a)

a = np.zeros((2,4))
pprint(a)

%timeit b = np.load("./a.npy")
# 122 µs ± 3.28 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops
each)
pprint(b)

%timeit aa = np.random.randn(2,4)
pprint(aa)

np.savetxt("./aa.txt", aa)

aa = np.zeros((2,4))
pprint(aa)

%timeit bb = np.loadtxt("./aa.txt")
# 148 µs ± 2.57 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops
each)
pprint(bb)
```

---

## numpy.reshape

```
numpy.reshape(a, /, shape=None, order='C', *, newshape=None, copy=None)
```

- Gives a new shape to an array without changing its data.

```
>>> a = np.arange(6).reshape((3, 2))
>>> a
array([[0, 1],
       [2, 3],
       [4, 5]])
```

```
>>> import numpy as np
>>> a = np.array([[1,2,3], [4,5,6]])
>>> np.reshape(a, 6)
array([1, 2, 3, 4, 5, 6])
>>> np.reshape(a, 6, order='F')
array([1, 4, 2, 5, 3, 6])

>>> np.reshape(a, (3,-1))      # the unspecified value is
inferred to be 2
array([[1, 2],
       [3, 4],
       [5, 6]])
```

## numpy.exp

```
numpy.exp(x, /, out=None, *, where=True, casting='same_kind',
order='K', dtype=None, subok=True[, signature]) = <ufunc 'exp'>
```

- Calculate the exponential of all elements in the input array.

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

>>> x = np.linspace(-2*np.pi, 2*np.pi, 100)
>>> xx = x + 1j * x[:, np.newaxis] # a + ib over complex plane
>>> out = np.exp(xx)

>>> plt.subplot(121)
>>> plt.imshow(np.abs(out),
...           extent=[-2*np.pi, 2*np.pi, -2*np.pi, 2*np.pi], cmap='gray')
>>> plt.title('Magnitude of exp(x)')

>>> plt.subplot(122)
>>> plt.imshow(np.angle(out),
...           extent=[-2*np.pi, 2*np.pi, -2*np.pi, 2*np.pi], cmap='hsv')
>>> plt.title('Phase (angle) of exp(x)')
>>> plt.show()
```

## numpy.dot

```
numpy.dot(a, b, out=None)
```

- Dot product of two arrays. Specifically,



```
>>> >>> import numpy as np
>>> np.dot(3, 4)
12

# Neither argument is complex-conjugated:
>>> np.dot([2j, 3j], [2j, 3j])
(-13+0j)

# For 2-D arrays it is the matrix product:
>>> a = [[1, 0], [0, 1]]
>>> b = [[4, 1], [2, 2]]
>>> np.dot(a, b)
array([[4, 1],
       [2, 2]])

>>> a = np.arange(3*4*5*6).reshape((3,4,5,6))
>>> b = np.arange(3*4*5*6)[::-1].reshape((5,4,6,3))
>>> np.dot(a, b)[2,3,2,1,2,2]
499128
>>> sum(a[2,3,2,:] * b[1,2,:,2])
499128
```

## numpy.cumsum

```
numpy.cumsum(a, axis=None, dtype=None, out=None)[source]
```

- Return the cumulative sum of the elements along a given axis.
  - axis : int, optional
    - Axis along which the cumulative sum is computed. The default (None) is to compute the cumsum over the flattened array.

```
>>> import numpy as np
>>> a = np.array([[1,2,3], [4,5,6]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> np.cumsum(a)
array([ 1,  3,  6, 10, 15, 21])
>>> np.cumsum(a, dtype=float)      # specifies type of output value(s)
array([ 1.,  3.,  6., 10., 15., 21.])

>>> np.cumsum(a,axis=0)            # sum over rows for each of the 3 columns
array([[1, 2, 3],
       [5, 7, 9]])
>>> np.cumsum(a,axis=1)            # sum over columns for each of the 2 rows
array([[ 1,  3,  6],
       [ 4,  9, 15]])
```

```
# cumsum(b)[-1] may not be equal to sum(b)
>>> b = np.array([1, 2e-9, 3e-9] * 1000000)
>>> b.cumsum()[-1]
1000000.0050045159
>>> b.sum()
1000000.0050000029
```

## 8. 배열 복사

깊은복사    덤프    원본자체    새로운    원본


얕은복사    샬로우카피    포인터    메모리    주소만    가지고    연결되어있음

### numpy.copy

```
numpy.copy(a, order='K', subok=False)
```

- Return an array copy of the given object.

```
>>> np.array(a, copy=True)
```

-  The copy made of the data is shallow, i.e., for arrays with object dtype, the new array will point to the same objects. See Examples from ndarray.copy.

[Example]

```
>>> import numpy as np
```

```
# Create an array x, with a reference y and a copy z:
```

```
>>> x = np.array([1, 2, 3])
```

```
>>> y = x
```

```
>>> z = np.copy(x)
```

```
# Note that, when we modify x, y changes, but not z:
```

```
>>> x[0] = 10
```

```
>>> x[0] == y[0]
```

```
True
```

```
>>> x[0] == z[0]
```

```
False
```

```
# Note that, np.copy clears previously set WRITEABLE=False flag.
```

```
>>> a = np.array([1, 2, 3])
```

```
>>> a.flags["WRITEABLE"] = False
```

```
>>> b = np.copy(a)
```

```
>>> b.flags["WRITEABLE"]
True
>>> b[0] = 3
>>> b
array([3, 2, 3])
```

## 데모 배열 생성

```
a1 = np.arange(1, 25).reshape((4,6)) # 2차원 배열
print(f"◆ 원본 a1 array:\n{a1}")

# 22를 특정해서 220 으로 업데이트
a1 = np.where(a1 == 22,220,a1)
print(f"\n ◆ 22를 특정해서 220 으로 업데이트 a1 array:\n{a1}")

# 21, 22, 23 특정해서 210, 220, 230 업데이트
for x in range(21, 24):
    a1 = np.where(a1 == x, a1*10,a1)

print(f"\n ◆ 1, 22, 23 특정해서 210, 220, 230 업데이트 array:\n{a1}")
----
◆ 원본 a1 array:
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]

◆ 22를 특정해서 220 으로 업데이트 a1 array:
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 220 23 24]]

◆ 1, 22, 23 특정해서 210, 220, 230 업데이트 array:
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 210 220 230 24]]
```