파이썬 외부모듈 (matplotlib, numpy)

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(콘다의 경우 terminal 창)에서 실행**
  - > pip install 모듈명

```
> pip install numpy
```

```
> pip install pandas
```

```
> pip install matplotlib
```

- **설치 모듈 확인 : > pip list**

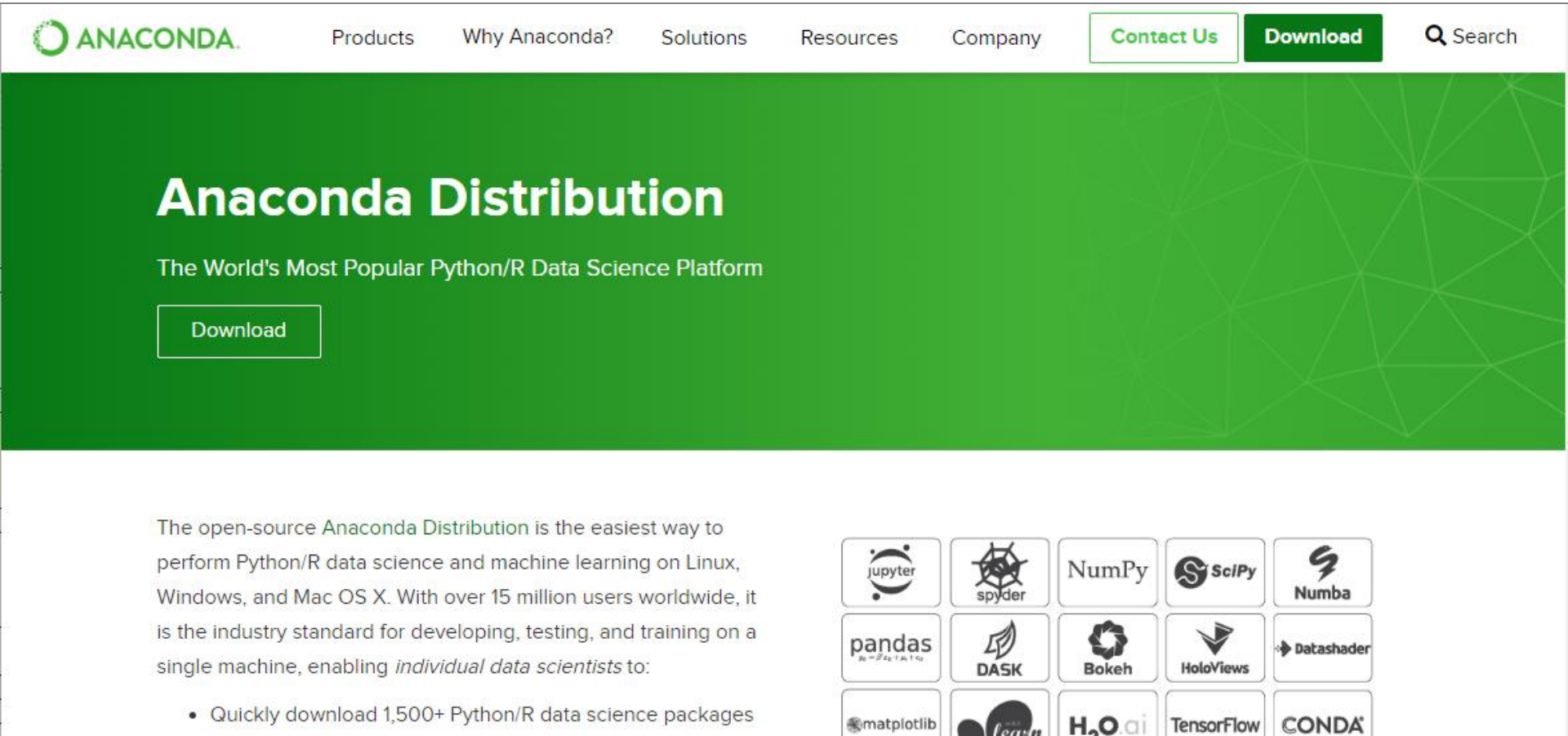
```
> pip show numpy
```

```
> pip show pandas
```

```
> pip show matplotplib
```

Anaconda

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

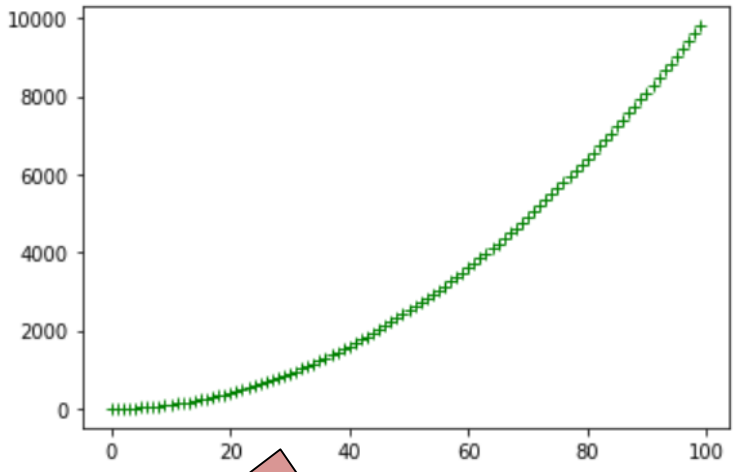


matplotlib

간단한 그래프

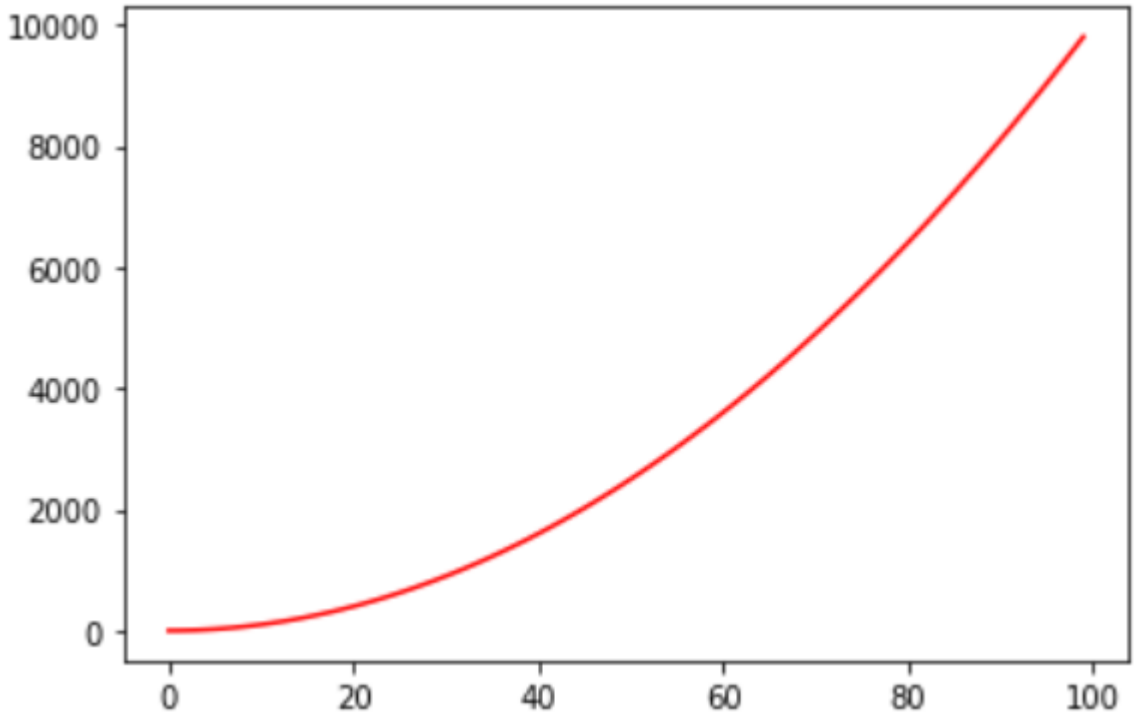
- **matplotlib.pyplot 모듈**
- **plot() 함수 : 값을 서로 연결하여 라인 형태의 그래프 그림**
  - 기본 포맷 : 파란색 선
  - ‘r’ : 빨간색 선, plt.plot(x, y, ‘r’)
  - ‘ro’ : 빨간색 원 마커 모양, plt.plot(x, y, ‘ro’)
  - ‘r--‘ : 빨간색 대쉬라인, plt.plot(x, y, ‘r--‘)
  - ‘bs’ : 파란색 사각형, plt.plot(x, y, ‘bs’)
  - ‘g+’ : 녹색 십자모양, plt.plot(x, y, ‘g+’)

```
1  import matplotlib.pyplot as plt
2  %matplotlib inline
3
4  x = [i for i in range(100)]
5  y = [i**2 for i in x]
6
7  plt.plot(x, y, 'g+')
```

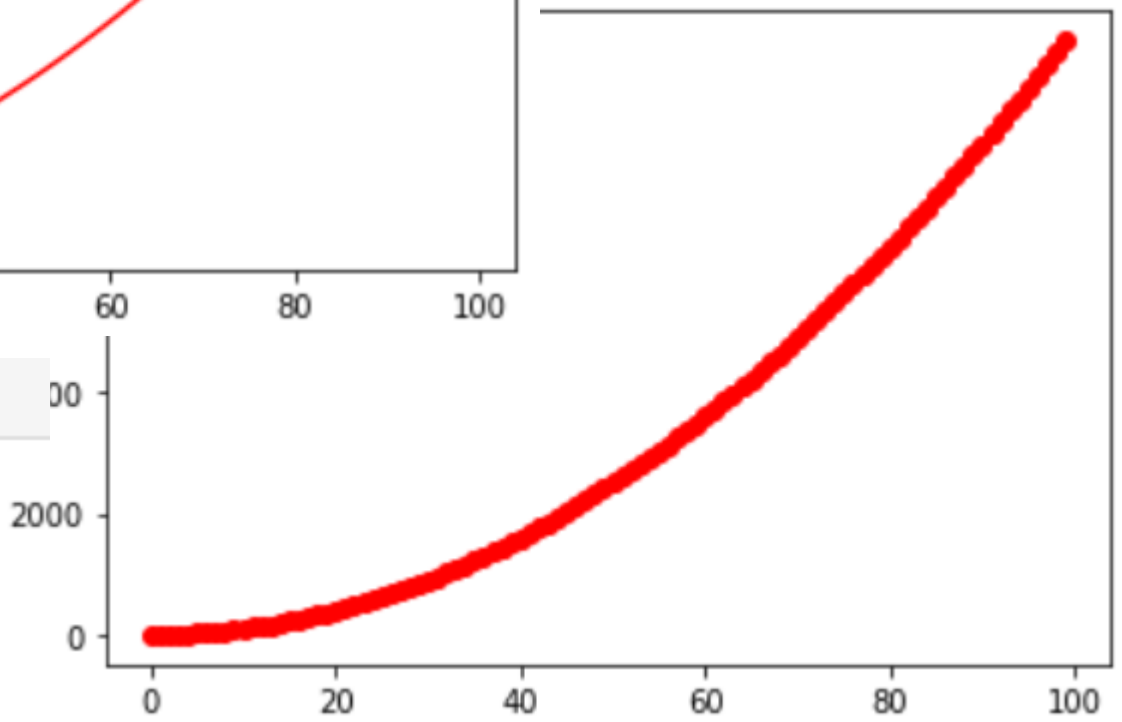


간단한 그래프

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



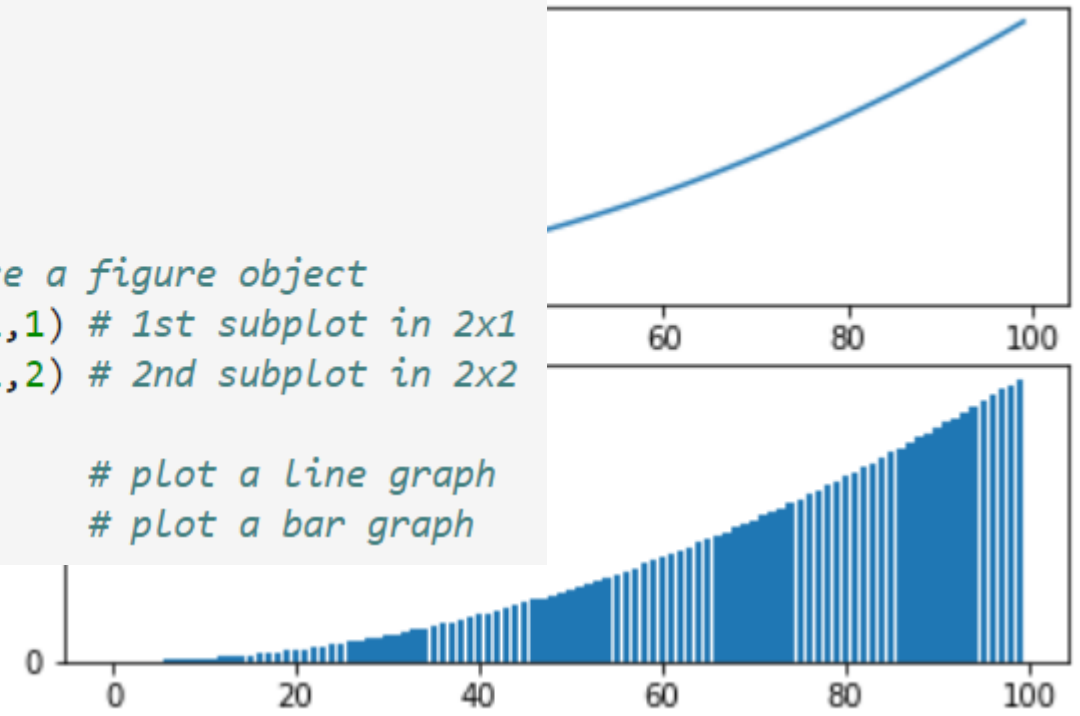

```
1 plt.plot(x, y, 'ro')
```



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**

```
1  import matplotlib.pyplot as plt
2  %matplotlib inline
3
4  x = [i for i in range(100)]
5  y = [i**2 for i in x]
6
7  fig = plt.figure() # instance a figure object
8  axe_01 = fig.add_subplot(2,1,1) # 1st subplot in 2x1
9  axe_02 = fig.add_subplot(2,1,2) # 2nd subplot in 2x2
10
11 axe_01.plot(x, y)              # plot a line graph
12 axe_02.bar(x, y)               # plot a bar graph
```

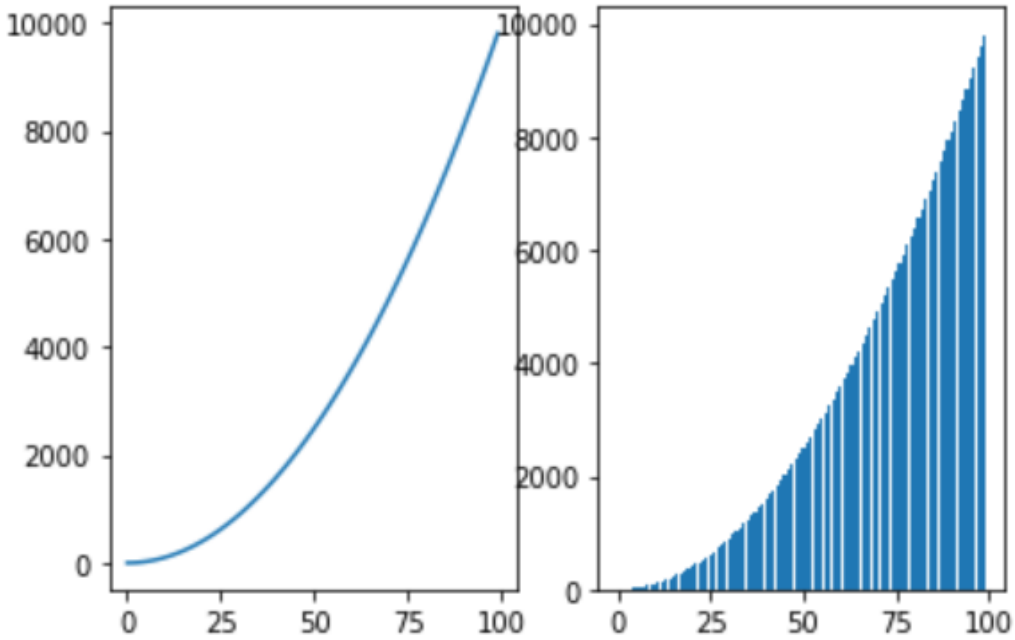


한 화면에 여러 개 그래프

- **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 두 번째

```
1  fig = plt.figure() # instance a figure object
2  axe_01 = fig.add_subplot(1,2,1) # 1st subplot in 2x1
3  axe_02 = fig.add_subplot(1,2,2) # 2nd subplot in 2x2
4
5  axe_01.plot(x, y)                # plot a line graph
6  axe_02.bar(x, y)                 # plot a bar graph
```

```
<BarContainer object of 100 artists>
```

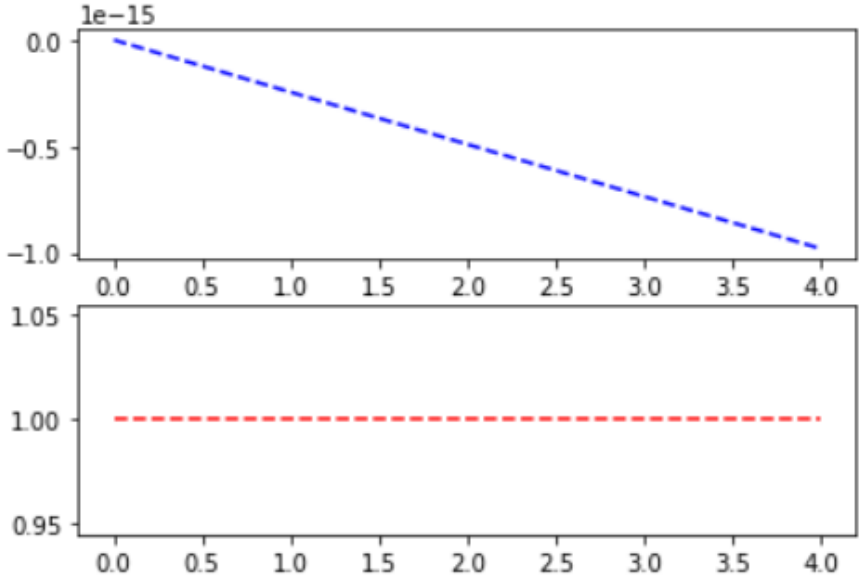


sine, cosine 그래프

- **파이썬의 기본 math 모듈 사용**

```
1  import math
2
3  x = [i for i in range(5)]
4  sin_y = [math.sin(2*math.pi*i) for i in x]
5  cos_y = [math.cos(2*math.pi*i) for i in x]
6
7  fig = plt.figure() # instance a figure object
8  axe_01 = fig.add_subplot(2,1,1) # 1st subplot in 2x1
9  axe_02 = fig.add_subplot(2,1,2) # 2nd subplot in 2x2
10
11 axe_01.plot(x, sin_y, 'b--')    # plot a sin graph
12 axe_02.plot(x, cos_y, 'r--')    # plot a cos graph
```

```
[<matplotlib.lines.Line2D at 0x1531b92aa88>]
```

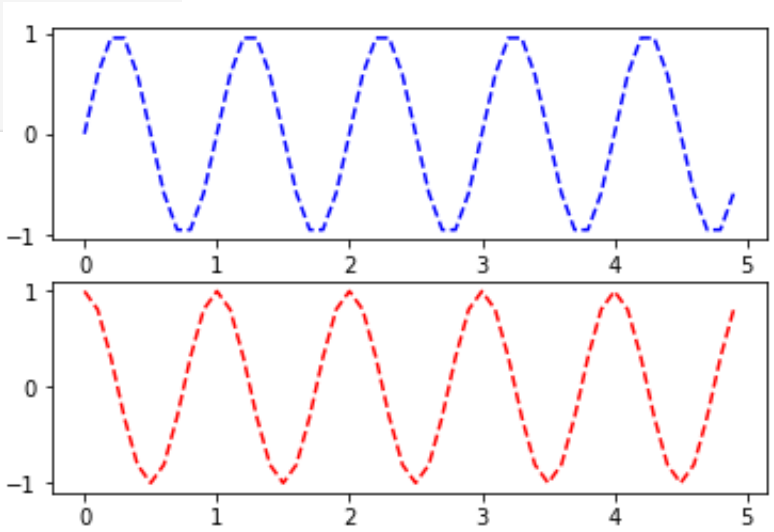


정수 0~5 사이 범위의 값에서
range(0,5)로는 시간의 값을
촘촘하게 만들 수 없음

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**

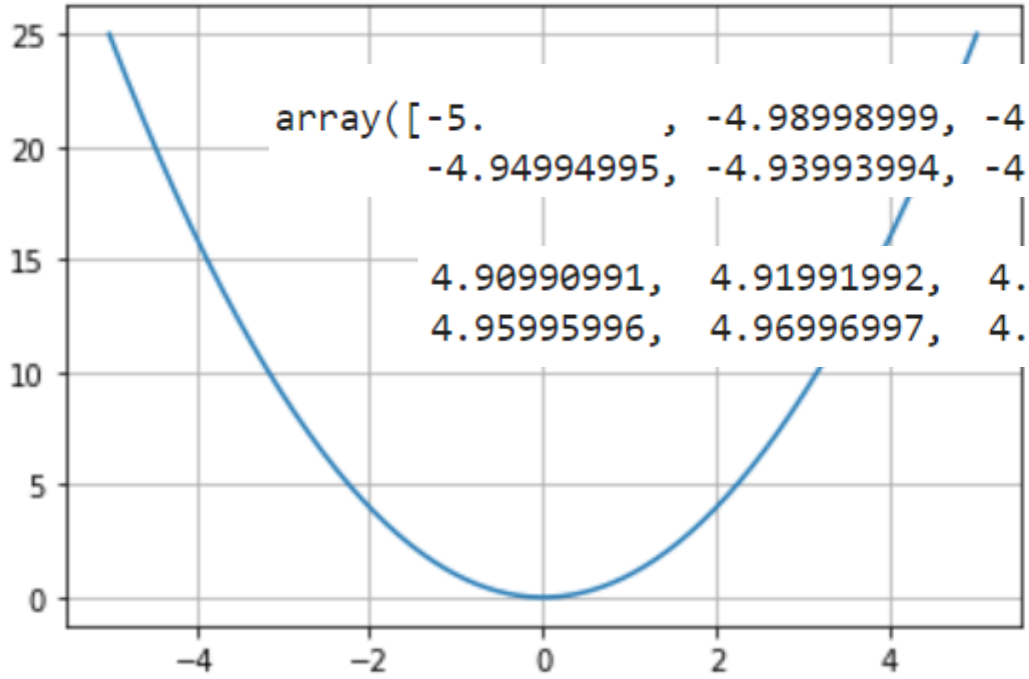
```
1  import numpy as np
2
3  x = np.arange(0., 5., 0.1)
4  sin_y = [np.sin(2*np.pi*i) for i in x]
5  cos_y = [np.cos(2*np.pi*i) for i in x]
6
7  fig = plt.figure() # instance a figure object
8  axe_01 = fig.add_subplot(2,1,1) # 1st subplot in 2x1
9  axe_02 = fig.add_subplot(2,1,2) # 2nd subplot in 2x2
10
11 axe_01.plot(x, sin_y, 'b--')    # plot a sin
12 axe_02.plot(x, cos_y, 'r--')    # plot a cos
```



numpy 기반 값 생성

- **numpy.linspace(start, end, num-points) : 시작점과 끝점을 균일 간격으로 나눈 값 생성**
- **numpy.power(x, y) : x의 y제곱 값**

```
1  x = np.linspace(-5, 5, 1_000) # ndarray
2  y = np.power(x, 2)            # ndarray
3
4  plt.plot(x, y)  # plot a line with x, y
5  plt.grid()      # show a grid
6  plt.show()      # show a whole graph
```



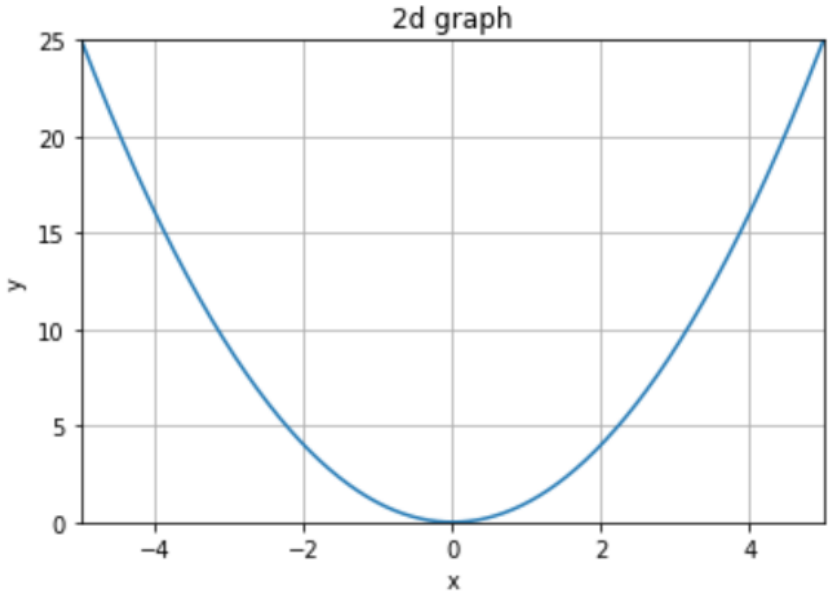

```
array([-5.        , -4.98998999, -4.97997998, -4.96996997, -4.95995996,
       -4.94994995, -4.93993994, -4.92992993, -4.91991992, -4.90990991,

       4.90990991,  4.91991992,  4.92992993,  4.93993994,  4.94994995,
       4.95995996,  4.96996997,  4.97997998,  4.98998999,  5.        ])
```

그래프 라벨 및 범례 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

```
1  x = np.linspace(-5, 5, 1_000) # ndarray
2  y = np.power(x, 2)            # ndarray
3
4  plt.plot(x, y)   # plot a line with x, y
5  plt.title("2d graph")
6  plt.xlabel("x")  # x axe label
7  plt.ylabel('y')  # y axe label
8  plt.xlim((-5,5)) # range tuple
9  plt.ylim((0,25)) # range tuple
10 plt.grid()       # show a grid
11 plt.show()       # show a whole graph
```

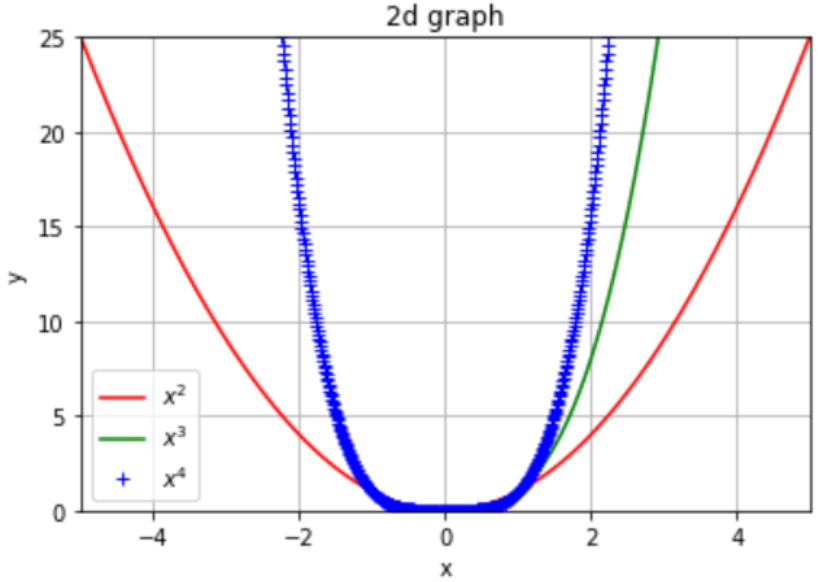


multiple lines

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

```
1  x = np.linspace(-5, 5, 1_000) # ndarray
2
3  plt.plot(x, np.power(x, 2), "r-", label="$x^2$")
4  plt.plot(x, np.power(x, 3), "g-", label="$x^3$")
5  plt.plot(x, np.power(x, 4), "b+", label="$x^4$")
6
7  plt.title("2d graph")
8  plt.xlabel("x")  # x axe label
9  plt.ylabel('y')  # y axe label
10 plt.xlim((-5,5)) # range tuple
11 plt.ylim((0,25)) # range tuple
12 plt.grid()       # show a grid
13 plt.legend(loc="best")
14 plt.show()       # show a whole graph
```

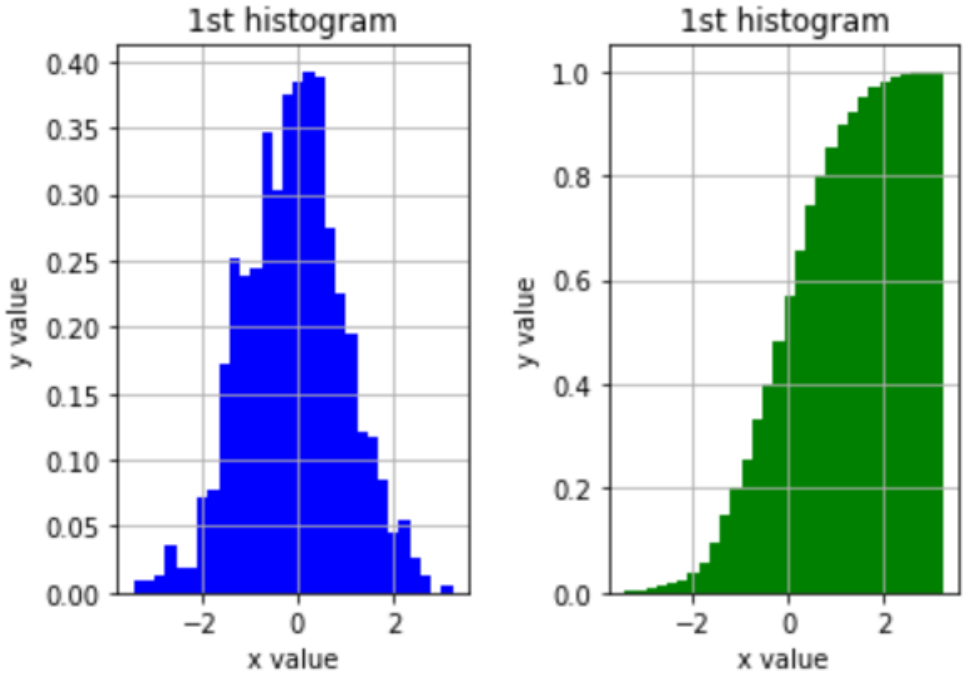
Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10



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**

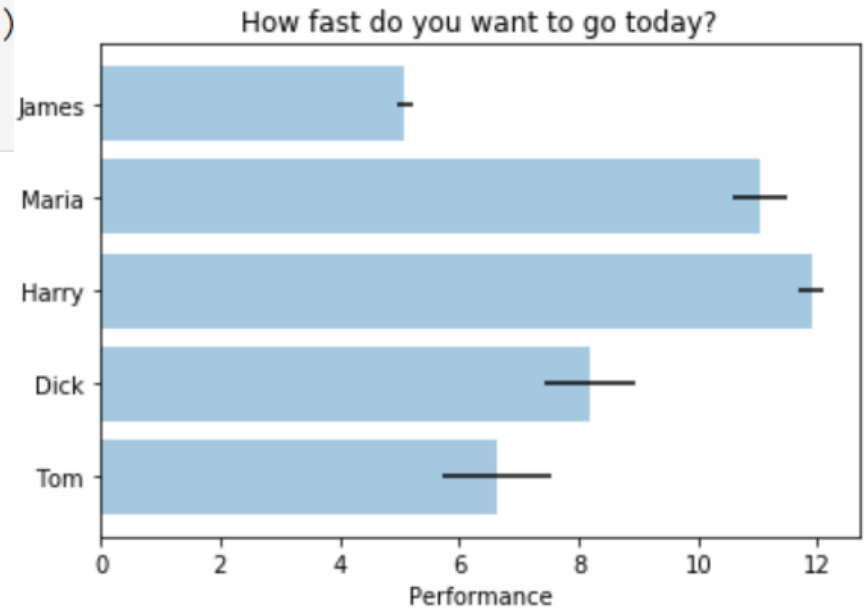
```
1  data = np.random.randn(1_000)
2
3  fig    = plt.figure()           # instance a figure
4  axe_01 = fig.add_subplot(121) # 1st subplot in 2x1
5  axe_02 = fig.add_subplot(122) # 2nd subplot in 2x2
6
7  axe_01.set_title("1st histogram")
8  axe_01.set_xlabel("x value")
9  axe_01.set_ylabel("y value")
10 axe_01.grid(True)
11 axe_01.hist(data, bins=30, density=True, color='b')
12
13 axe_02.set_title("1st histogram")
14 axe_02.set_xlabel("x value")
15 axe_02.set_ylabel("y value")
16 axe_02.grid(True)
17 axe_02.hist(data, bins=30, density=True, color='g', cumulative=True)
18
19 plt.subplots_adjust(wspace=.4)
20 plt.show()
```



Horizontal bar

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

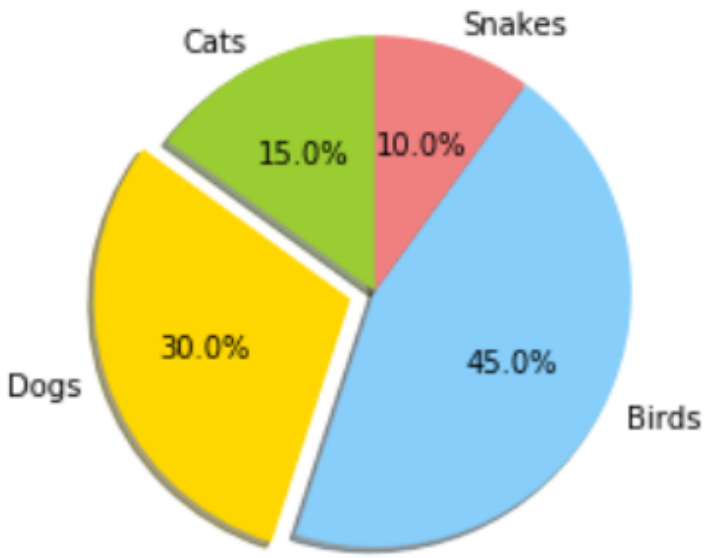
```
1  people = ["Tom", "Dick", "Harry", "Maria", "James"]
2
3  y_position = np.arange(len(people))
4  performance = 3 + 10 * np.random.rand(len(people))
5  error = np.random.rand(len(people))
6
7  plt.barh(y_position, performance,
8           xerr=error, align='center', alpha=0.4)
9  plt.yticks(y_position, people)
10 plt.xlabel("Performance")
11 plt.title("How fast do you want to go today?")
12
13 plt.show()
```



원 그래프

- **pie () 함수 : 원 그래프 그리기**

```
1  # THe slces will be ordered and plotted counter-clockwise.
2  labels  = ["Cats", "Dogs", "Birds", "Snakes"]
3  sizes   = [15, 30, 45, 10]
4  colors  = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
5  explode = [0, 0.1, 0, 0] # only explode : 2nd slice(i.e. Dogs)
6
7  plt.pie(sizes, explode=explode, labels=labels, colors=colors,
8          autopct="%3.1f%%", shadow=True, startangle=90)
9
10 plt.show()
```



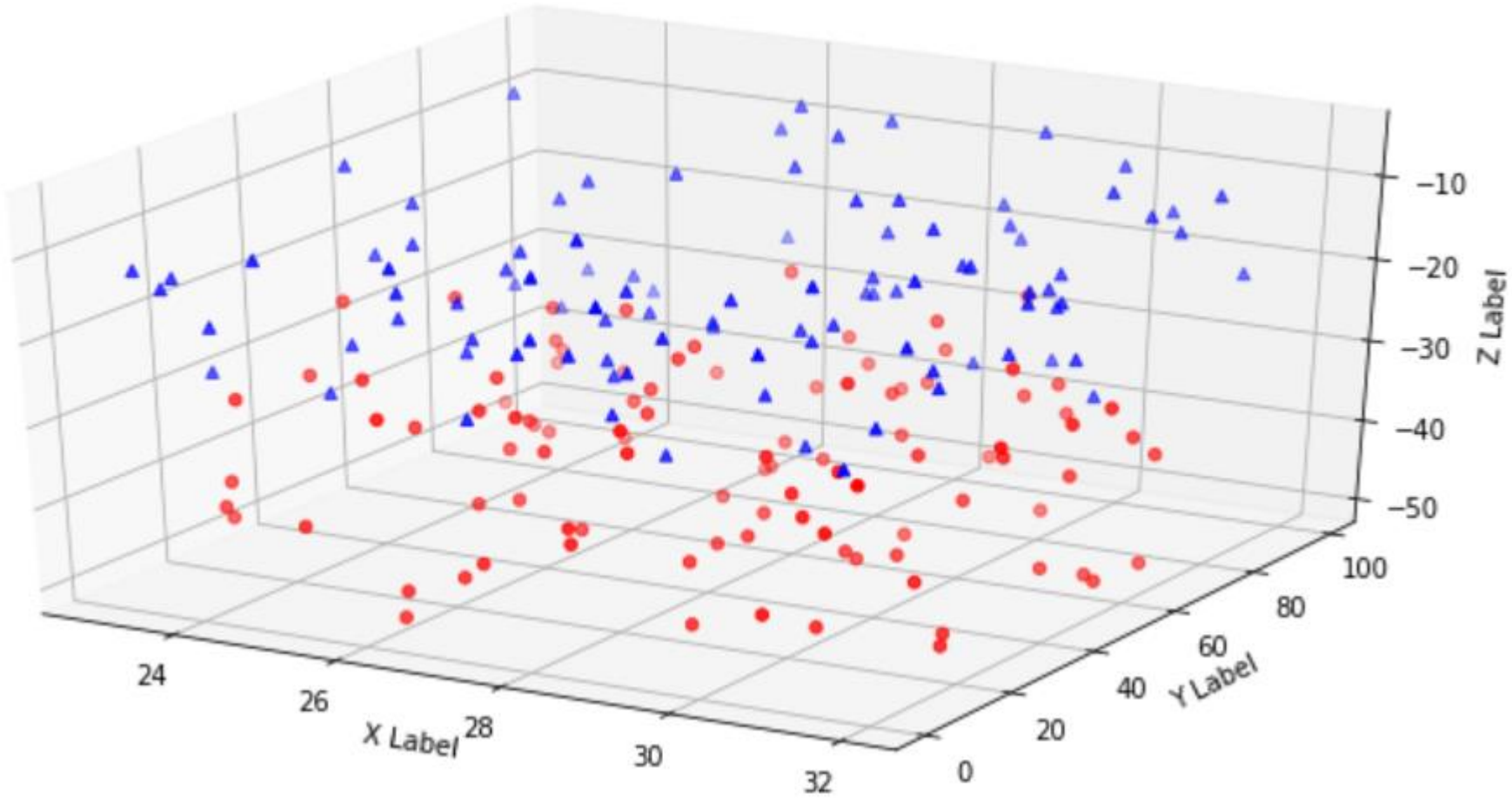
3D 그래프(2/1)

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

```
1  from mpl_toolkits.mplot3d import Axes3D
2
3  def randrange(n, min, max):
4      return (max-min)*np.random.rand(n)+min
5
6  n = 100
7  fig = plt.figure()
8  axe = fig.add_subplot(111, projection='3d')
9
10 for c, m, zl, zh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
11     xs = randrange(n, 23, 32)
12     ys = randrange(n, 0, 100)
13     zs = randrange(n, zl, zh)
14     axe.scatter(xs, ys, zs, c=c, marker=m)
15
16 axe.set_xlabel('X Label')
17 axe.set_ylabel('Y Label')
18 axe.set_zlabel('Z Label')
19
20 plt.show()
```

3D 그래프(2/2)

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



numpy

numpy

- **대용량 데이터 배열을 효율적으로 다룰 수 있도록 설계됨**
  - numpy는 데이터를 다른 내장 파이썬 객체와 구분되는 연속된 메모리 블록에 저장
  - numpy 의 각종 알고리즘은 모두 C로 작성되어 타입 검사나 다른 오버헤드 없이 메모리를 직접 조작할 수 있음
  - numpy 배열은 내장 파이선의 연속된 자료형들보다 훨씬 더 적은 메모리를 사용함
  - numpy 연산은 파이선 반복문을 사용하지 않고 전체 배열에 대한 복잡한 계산을 수행함

넘파이

- 파이썬 리스트 자료형의 대안으로 선형대수, 행렬, 이미지 연산 등 지원

```
mass  = [10, 20, 30, 40, 50]
accel = [1, 3, 7, 9, 11]

force = mass * accel
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-9ca3a37b620b> in <module>
      2 accel = [1, 3, 7, 9, 11]
      3
----> 4 force = mass * accel

TypeError: can't multiply sequence by non-int of type 'list'
```

```
force = []
for i in range(len(mass)):
    force.append(mass[i] * accel[i])

force
```

```
[10, 60, 210, 360, 550]
```

```
import numpy as np

mass = np.array([10, 20, 30, 40, 50])
accel = np.array([1, 3, 7, 9, 11])

force = mass * accel
force
```

```
array([ 10,  60, 210, 360, 550])
```

numpy 배열 vs. Python 리스트

- 성능 비교

```
1  import numpy as np
2
3  a_array = np.arange(1_000_000)  # numpy ndarray
4  a_list = list(range(1_000_000)) # python list
```

```
1  %time for _ in range(10): b_array = a_array * 2
```

```
Wall time: 21 ms
```

```
1  %time for _ in range(10): b_list = [x*2 for x in a_list]
```

```
Wall time: 899 ms
```

```
1  round(899/21, 1) # 넘파이가 약 43배 빠름
```

```
42.8
```

NumPy ndarray : 다차원 배열 객체

- **ndarray**
  - 같은 종류의 데이터를 담을 수 있는 포괄적인 다차원 배열
  - 단, 리스트와 달리 모든 원소는 같은 자료형 이어야 함

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

```
1  data
```

```
array([[-0.14039762,  0.41357354, -0.54189105],
       [-0.1666982 ,  0.0870287 ,  0.73809299]])
```

```
1  data * 10
```

```
array([[-1.40397618,  4.13573537, -5.41891054],
       [-1.66698199,  0.87028696,  7.38092994]])
```

```
1  data + data
```

```
array([[-0.28079524,  0.82714707, -1.08378211],
       [-0.3333964 ,  0.17405739,  1.47618599]])
```

numpy ndarray : 다차원 배열 객체

- **shape**
  - 각 차원의 크기를 알려줌

- **dtype**
  - 튜플과 배열에 저장된 자료형을 알려줌

```
1 data.shape
```

```
(2, 3)
```

```
1 data.dtype
```

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

ndarray 생성하기

- **array(배열) 함수**
  - 배열 혹은 순차적인 객체를 전달 받아 새로운 numpy 배열 생성

```
1  data_01 = [5, 6, 7.5, 8, 9.9, 10]
2  array_01 = np.array(data_01)
3  array_01  # 1차원 배열
```

```
array([ 5. ,  6. ,  7.5,  8. ,  9.9, 10. ])
```

```
1  data_02 = [[1,2,3,4,5], [6,7,8,9,10]]
2  array_02 = np.array(data_02)
3  array_02  # 2차원 배열
```

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

ndarray 생성하기

- **ndim**
  - 차원 확인

```
1 array_01.ndim
```

```
1
```

```
1 array_01.shape  # tuple
```

```
(6,)
```

```
1 array_01.dtype
```

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

```
1 array_02.ndim
```

```
2
```

```
1 array_02.shape # tuple
```

```
(2, 5)
```

```
1 array_02.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**

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

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

```
1  np.ones((2, 10))
```

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

```
1  np.empty((2, 5, 3)) # 5 x 3 배열 2개인 3차원 배열
```

```
array([[[2.29175545e-312, 2.10077583e-312, 2.07955588e-312],
        [2.05833592e-312, 2.27053550e-312, 2.29175545e-312],
        [2.10077583e-312, 2.07955588e-312, 2.05833592e-312],
        [2.27053550e-312, 2.29175545e-312, 2.10077583e-312],
        [2.07955588e-312, 2.05833592e-312, 2.27053550e-312]],

       [[2.29175545e-312, 2.10077583e-312, 2.07955588e-312],
        [2.05833592e-312, 2.27053550e-312, 2.29175545e-312],
        [2.10077583e-312, 2.07955588e-312, 2.05833592e-312],
        [2.27053550e-312, 2.29175545e-312, 2.10077583e-312],
        [2.07955588e-312, 2.05833592e-312, 2.27053550e-312]]])
```

ndarray 생성하기

- **arange 함수**
  - 파이썬 range 함수의 배열 버전

```
1  np.arange(15)  # np.arange(0, 15, 1)
```

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

```
1  np.linspace(1, 5, 10) # 1부터 10까지 수에서 10개로 쪼개는 range
```

```
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

```
1
```

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

```
1  array_01 = np.array([1,2,3,4,5], dtype=np.float64)
2  array_02 = np.array([1,2,3,4,5], dtype=np.int32)
```

```
1  array_01.dtype
```

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

```
1  array_02.dtype
```

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

- **numpy data type ➔**

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 wi th 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 fl oats, 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 exa mple, to create a string dtype with length 10, use 'S10' .
unicode_	U	Fixed-length unicode type (number of bytes platform s pecific). Same specification semantics as string_ (e.g. ' U10').

ndarray의 dtype

- **astype 메소드**
  - 명시적 타입 변환 (캐스팅)
  - 정수형 → 부동 소수점형

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

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

```
1 float_array = int_array.astype(np.float64)
2 float_array.dtype
```

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

  - 부동소수점형 → 정수형 (소수점 아래 자리 버짐)

```
1 float_array = np.array([1.2, 2.5, 3,4, 5,6, 7,8, 9.0])
2 float_array
```

```
array([1.2, 2.5, 3. , 4. , 5. , 6. , 7. , 8. , 9. ])
```

```
1 float_array.astype(np.int32)
```

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

ndarray의 dtype

- 숫자 형식 문자열 → 숫자

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

```
1 numeric_string = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
2 numeric_string.astype(np.float64) # 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)

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

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

```
1 calibers = np.array([.22, .270, .356, .380, .44, .50], dtype=np.float64)
2 int_array.astype(calibers.dtype)
```

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

- 축약 코드 사용 가능 uint32

```
1 empty_unit32 = np.empty(10, dtype='u4')
2 empty_unit32
```

```
array([ 375165360,        339,    3801156,    7143516,    7209057,
          6750305,    3014757,    7929968, 2058878976, 2059041465],
      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(broadcasting)

```
a_array = np.array([[1., 2., 3.], [4., 5., 6.]])
a_array   # 2 by 3 matrix
```

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

```
a_array * a_array
```

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

```
a_array - a_array
```

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

```
1/ a_array
```

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

```
a_array ** 0.5
```

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

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

```
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)

```
1 array_2d = np.array([[1, 2, 3],[4, 5, 6], [7, 8, 9]])
2 array_2d   # 3 by 3 matrix
```

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

```
1 array_2d[2] # 3rd index
```

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

```
1 array_2d[0][2] # 0th row 2nd col
```

```
3
```

```
1 array_2d[0, 2] # 0th row 2nd col
```

```
3
```

```
1 array_2d[ :2] # from 0th to 1st
```

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

```
1 array_2d[ :2, 1: ] # 0,1 row, 1st, 2nd col
```

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

```
1 array_2d[1, :2] # 1st row, 0,1 col
```

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

```
1 array_2d[ :2, 2] # 0,1 row, 2nd col
```

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

```
1 array_2d[:, :1] # all row, 0th col
```

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

```
1 array_2d[:2, 1:] # 0,1 row, 1, 2 col
```

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

```
1 array_2d[:2, 1:] = 0
2 array_2d
```

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

Boolean Indexing

- **Boolean 배열을 배열의 색인으로 사용**

```
1 name_list = ['Bob', 'Joe', "Will", 'Bob', "Will", "joe", "Joe"]
2 names = np.array(name_list)
3 data = np.random.randn(7,4)
```

```
1 names  # 7 elements
```

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

joe is not Joe

```
1 data   # 7 by 4 matrix
```

```
array([[ 0.24225625,  2.28339039,  0.83670996,  0.12247309],
       [ 0.80877744, -0.72551652,  1.65148444, -1.24723941],
       [-0.08816675,  0.4060429 ,  1.27812113, -0.01765826],
       [ 0.71457754, -0.50468375,  0.60981897, -0.22058279],
       [-0.08531094,  0.22222095,  0.65656547,  0.12164069],
       [ 2.39122459,  0.35761168,  2.25078093,  0.52452568],
       [-1.25806627, -1.57186624, -0.57212614,  0.73847539]])
```

```
1 names == 'Bob' # 2 matches in the names array
```

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

```
1 data[names == 'Bob'] # 2 rows extract(0th and 3rd row)
```

```
array([[ 0.24225625,  2.28339039,  0.83670996,  0.12247309],
       [ 0.71457754, -0.50468375,  0.60981897, -0.22058279]])
```

```
1 data[names == 'Bob', 2:] # 0,3 rows and from 2 to last col
```

```
array([[ 0.83670996,  0.12247309],
       [ 0.60981897, -0.22058279]])
```

Boolean Indexing

```
1 data[names == 'Bob', 3] # 0,3 rows and 3rd col only
```

```
array([ 0.12247309, -0.22058279])
```

```
1 data[~(names == 'Bob')] # 0,3 row 제외한 모든 rows
```

```
array([[ 0.80877744, -0.72551652,  1.65148444, -1.24723941],
       [-0.08816675,  0.4060429 ,  1.27812113, -0.01765826],
       [-0.08531094,  0.22222095,  0.65656547,  0.12164069],
       [ 2.39122459,  0.35761168,  2.25078093,  0.52452568],
       [-1.25806627, -1.57186624, -0.57212614,  0.73847539]])
```

```
1 data[data < 0] = 0 # if data < 0, then data = 0
2 data
```

```
array([[0.24225625, 2.28339039, 0.83670996, 0.12247309],
       [0.80877744, 0.        , 1.65148444, 0.        ],
       [0.        , 0.4060429 , 1.27812113, 0.        ],
       [0.71457754, 0.        , 0.60981897, 0.        ],
       [0.        , 0.22222095, 0.65656547, 0.12164069],
       [2.39122459, 0.35761168, 2.25078093, 0.52452568],
       [0.        , 0.        , 0.        , 0.73847539]])
```

```
1 data[names != 'Joe'] = 7 # if data != 'Joe', then data = 7
2 data
```

```
array([[7.        , 7.        , 7.        , 7.        ],
       [0.80877744, 0.        , 1.65148444, 0.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [0.        , 0.        , 0.        , 0.73847539]])
```

Fancy Indexing

- 정수 배열을 사용한 indexing

```
1 array_x = np.empty((8, 4)) # 8 by 4 matrix
2 for x in range(8): array_x[x] = x
3 array_x
```

```
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.]])
```

```
1 array_x[[4, 3, 0, 6]] # 4,3,0,6 rows extract
```

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

```
1 array_x[[-4, -5, -8, -2]] # 4,3,0,6 rows extract
```

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

Fancy Indexing

- 다차원 배열의 fancy indexing 결과는 항상 1차원

```
1 array_f = np.arange(32).reshape(8, 4) # 8 by 4 matrix
2 array_f
```

```
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]])
```

```
1 array_f[[1, 3, 5, 7], [0, 1, 2, 3]] # row, col diagonostic
```

```
array([ 4, 13, 22, 31])
```

```
1 array_f[[1, 3, 5, 7]] [:, [0, 1, 2]] # 1,3,5,7 rows, 0,1,2 cols
```

```
array([[ 4,  5,  6],
       [12, 13, 14],
       [20, 21, 22],
       [28, 29, 30]])
```

```
1 array_f[[1, 3, 5, 7]] # 1,3,5,6 rows, all cols extract
```

```
array([[ 4,  5,  6,  7],
       [12, 13, 14, 15],
       [20, 21, 22, 23],
       [28, 29, 30, 31]])
```

- 행렬의 low와 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

```
1 array_f[[1, 3, 5, 7]] # 1,3,5,6 rows, all cols extract
```

```
array([[ 4,  5,  6,  7],
       [12, 13, 14, 15],
       [20, 21, 22, 23],
       [28, 29, 30, 31]])
```

```
1 array_r = np.arange(15).reshape(3, 5)
2 array_r
```

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

```
1 array_r.reshape(5, -1) # -1 means 알아서 계산하라
```

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

```
1 array_r.reshape(4, -1) # 4 by x에 맞는 정수가 없음
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-223-bd70aa29438e> in <module>
----> 1 array_r.reshape(4, -1) # 4 by x에 맞는 정수가 없음

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

배열 전치와 축 바꾸기

- T : 축을 뒤바꾸는 간단한 전치

```
1  array_r   # 3 by 4 matrix
```

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

```
1  array_r.T # 4 by 3 transpose
```

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

NumPy Universal 함수

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

NumPy Universal 함수

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

- **dir(np) : numpy의 속성(변수)과 함수 알아보기**

Any Questions...
Just Ask!