

0. 노트북 단축키

- **anaconda** 설치

- 경로를 터미널로 연 다음 jupyter notebook 실행

- **jupyter notebook** 설치

- python 3 파일 만들기

- 명령어 사용 (명령모드)(선택모드)(파란색)

- shift + m : 셀 합치기
 - a : 위로 셀 생성
 - b : 아래로 셀 생성
 - dd : 셀 지우기
 - x / c / v : 잘라내기 / 복사 / 붙여넣기
 - m : markdown
 - y : code

- 명령어 사용 (입력모드)(초록색)

- command + enter, shift + enter, option + enter
 - option + shift + - : 셀 나누기
 - ctrl + a : 선택 셀의 전체 코드 선택
 - ctrl + z : 선택 셀 내 실행 취소
 - ctrl + y : 선택 셀 내 다시 실행
 - ctrl + / : 커서 위치 주석 처리
 - ctrl + enter : 선택 셀 코드 실행
 - shift+enter : 선택 셀 코드 실행 후 다음 cell로 이동(없으면 새로추가)

1. Numpy 모듈 선언

- 새로운 노트북을 실행한 경우 다시 선언해야 함

```
In [1]: import numpy as np  
np.__version__  
Out[1]: '1.20.3'
```

2. Array 정의 및 사용

- 시퀀스 데이터(리스트, 튜플등)로부터 배열 생성
- <http://numpy.org/doc/stable/reference/arrays.html>

a. array 생성

```
In [10]: data1 = [1,2,3]
```

```
In [11]: data1
```

```
Out[11]: [1, 2, 3]
```

```
In [12]: data2 = [1,2,3,3.5,5]
```

```
data2
```

```
Out[12]: [1, 2, 3, 3.5, 5]
```

Command

NumPy Array

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



1
2
3

```
In [13]: #리스트 객체를 이용하여 array 생성  
arr1 = np.array(data1)
```

```
Out[13]: array([1, 2, 3])
```

b. shape : array 크기 확인

```
In [14]: # array 크기 확인  
arr1.shape #1차원 자료형
```

```
Out[14]: (3,)
```

```
In [15]: # 리스트를 직접 입력하여 array 생성  
arr2 = np.array([1,2,3,4,5])
```

```
Out[15]: array([1, 2, 3, 4, 5])
```

```
In [16]: arr2.shape
```

```
Out[16]: (5,)
```

c. dtype : array 자료형 확인

```
In [17]: # array의 자료형 확인
arr2.dtype

Out[17]: dtype('int64')

In [18]: arr3 = np.array([1,2,3,3.5,4])
arr3

Out[18]: array([1. , 2. , 3. , 3.5, 4. ])

In [19]: arr3.dtype

Out[19]: dtype('float64')

In [20]: arr3.shape

Out[20]: (5,)

In [21]: arr4 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])

Out[21]: array([[ 1,  2,  3],
   [ 4,  5,  6],
   [ 7,  8,  9],
   [10, 11, 12]])

In [22]: arr4.shape

Out[22]: (4, 3)

In [23]: arr4.dtype

Out[23]: dtype('int64')
```

d. numpy 자료형

- 부호가 있는 정수 **int(8,16,32,64)**

```
arr3 = np.array([1,2,3,4,5])
arr3,arr3.dtype,arr3.shape

(array([1, 2, 3, 4, 5]), dtype('int64'), (5,))

arr4 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
arr4,arr4.dtype,arr4.shape

(array([[ 1,  2,  3],
   [ 4,  5,  6],
   [ 7,  8,  9],
   [10, 11, 12]]),
 dtype('int64'),
 (4, 3))
```

- 부호가 없는 정수 **uint(8,16,32,64)**
- 실수 **float(16,32,64,128)**
- 복소수 **complex(64,128,256)**

```
arr5 = np.array([5+2j])
arr5,arr5.dtype,arr5.shape

(array([5.+2.j]), dtype('complex128'), (1,))
```

- 불리언 **bool**

```
arr6 = np.array([True])
arr6,arr6.dtype,arr6.shape

(array([ True]), dtype('bool'), (1,))
```

- 문자열 **string_**

- 파이썬 오브젝트 object

```
arr8 = np.array([1, 0.1, 'one!'], dtype=object)
arr8,arr8.dtype,arr8.shape
(array([1, 0.1, 'one!'], dtype=object), dtype('O'), (3,))

arr8[0]
1

arr8[1],arr8[2]
(0.1, 'one!')

arr8 *2
array([2, 0.2, 'one!one!'], dtype=object)
```

- 유니코드 unicode_

3. 범위 지정해 배열 생성

a. np.arange() 함수

- 형식 : arr_obj = np.arange([start,]stop[,step])

```
np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.arange(1,5)
array([1, 2, 3, 4])

np.arange(1,10,2)
array([1, 3, 5, 7, 9])

np.arange(12).reshape(3,4) #shape을 재정의
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

b1 = np.arange(12).reshape(3,4)
b1.shape
(3, 4)
```

reshape(m,n)의 m*n의 개수와 arange()로 생성되는 원소의 개수와 일치해야 함

```
b3 = b1.reshape(4,3)
b3
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

b. linspace() 함수 (실수)

- 형식 : arr_obj = np.linspace([start,]stop[,num=50])

```

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

np.linspace(0,np.pi,20)
array([0.          , 0.16534698, 0.33069396, 0.49604095, 0.66138793,
       0.82673491, 0.99208189, 1.15742887, 1.32277585, 1.48812284,
       1.65346982, 1.8188168 , 1.98416378, 2.14951076, 2.31485774,
       2.48020473, 2.64555171, 2.81089869, 2.97624567, 3.14159265])

np.linspace(1,10) # 갯수 지정하지 않으면 50개
array([ 1.          , 1.18367347, 1.36734694, 1.55102041, 1.73469388,
       1.91836735, 2.10204082, 2.28571429, 2.46938776, 2.65306122,
       2.83673469, 3.02040816, 3.20408163, 3.3877551 , 3.57142857,
       3.75510204, 3.93877551, 4.12244898, 4.30612245, 4.48979592,
       4.67346939, 4.85714286, 5.04081633, 5.2244898 , 5.40816327,
       5.59183673, 5.7755102 , 5.95918367, 6.14285714, 6.32653061,
       6.51020408, 6.69387755, 6.87755102, 7.06122449, 7.24489796,
       7.42857143, 7.6122449 , 7.79591837, 7.97959184, 8.16326531,
       8.34693878, 8.53061224, 8.71428571, 8.89795918, 9.08163265,
       9.26530612, 9.44897959, 9.63265306, 9.81632653, 10.        ])

```

c. 특별한 형태의 배열 생성

1) np.zeros(shape, dtype=float,...)

: 모든 요소가 0인 배열 생성

```

np.zeros(10,dtype=int)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

np.zeros((3,5))
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]))

np.zeros((3,5)).reshape(5,3)
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])

```

2) np.ones(shape, dtype=float,...)

: 모든 요소가 1인 배열 생성

```

np.ones(10)
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])

np.ones((3,5),dtype=int)
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])

```

3) np.eye(n,m,k=K,dtype=float)

: 대각 요소가 1인 배열 생성

```
np.eye(3) #3행 3열의 대각요소만 1인 행렬  
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])  
  
np.eye(3,4)  
array([[1., 0., 0., 0.],  
       [0., 1., 0., 0.],  
       [0., 0., 1., 0.]])  
  
np.eye(3,4,k=1,dtype=int) #k : 시작 열주소(+1)  
array([[0, 1, 0, 0],  
       [0, 0, 1, 0],  
       [0, 0, 0, 1]])  
  
np.eye(3,4,k=2)  
array([[0., 0., 1., 0.],  
       [0., 0., 0., 1.],  
       [0., 0., 0., 0.]])  
  
np.eye(3,4,k=-1) # k = -n : 원쪽으로 이동  
array([[0., 0., 0., 0.],  
       [1., 0., 0., 0.],  
       [0., 1., 0., 0.]])
```

4) np.identity(n,dtype=float)

: 대각 요소가 1인 배열 생성 (2)

```
# n * n 크기의 단위 행렬 생성  
np.identity(5,dtype=int)  
array([[1, 0, 0, 0, 0],  
       [0, 1, 0, 0, 0],  
       [0, 0, 1, 0, 0],  
       [0, 0, 0, 1, 0],  
       [0, 0, 0, 0, 1]])
```

5) np.empty(shape,dtype=float)

: 초기화되지 않은 배열 생성

```
np.empty(4)  
array([0., 0., 0., 0.])
```

배열 생성 함수

함수	내용
np.array	입력 데이터를 ndarray로 변환 dtype을 명시하면 자료형을 설정할 수 있음
np.asarray	입력 데이터를 ndarray로 변환하거나 ndarray일 경우 새로 메모리에 ndarray가 생성되지 않음
np.arange	range함수와 유사하나 ndarray를 반환 기본 자료형은 float64
np.ones	전달인자로 전달한 dtype과 모양(행,렬)으로 배열을 생성하고 모든 내용을 1로 초기화하여 ndarray를 반환
np.zeros	ones와 같으나 초기값이 0
np.eye	대각요소를 1로 하고 나머지는 0으로 초기화
np.empty	ones, zeros와 비슷하나 값을 초기화하지 않음

5. 배열의 데이터 타입 변환

[astype() 함수 형식]

```
num_arr = str_arr.astype(dtype) (dtype : int ,float, str)
```

numpy 데이터 형식 : dtype

- 'i' : 부호가 있는 정수 int(8, 16, 32, 64)
- 'u' : 부호가 없는 정수 uint(8 ,16, 32, 54)
- 'f' : 실수 float(16, 32, 64, 128)
- 'c' : 복소수 complex(64, 128, 256)
- 'b' : 불리언 bool
- 'S' 혹은 'a' : 문자열 string_
- 'O' : 파이썬 오브젝트 object
- 'U' : 유니코드 unicode_
- 'M' : 날짜, datetime

a. 문자열 배열 -> 숫자형 배열

```
str_a1 = np.array(['1.5','0.62','2','3.14','3.141592'])  
# num_a1 = str_a1.astype(float32) # 자료형 비트 지정 가능  
num_a1 = str_a1.astype(float)  
num_a1
```

```
array([1.5      , 0.62     , 2.        , 3.14      , 3.141592])
```

```
num_a1.dtype
```

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

```
str_a1.dtype
```

```
dtype('<U8')
```

```
str_a2 = np.array(['1','3','5','7','9'])
num_a2 = str_a2.astype(int)
num_a2
array([1, 3, 5, 7, 9])
```

b. 실수형 배열 -> 정수형 배열

```
num_f1 = np.array([10,21,0.549,4.75,5.98])
num_f1
array([10., 21., 0.549, 4.75, 5.98])

num_f1.dtype
dtype('float64')

num_i1 = num_f1.astype(int)
num_i1
array([10, 21, 0, 4, 5])

num_i1.dtype
dtype('int64')
```

c. 숫자형 배열 -> 문자열 배열

```
num_f1.astype('U')
array(['10.0', '21.0', '0.549', '4.75', '5.98'], dtype='|U32')

num_f1.astype('S')
array([b'10.0', b'21.0', b'0.549', b'4.75', b'5.98'], dtype='|S32')
```

6. 난수 배열 생성

random.rand() , random.randint() 함수

a. random.rand([d0, d1, ..., dn])

- [0과 1)사이 실수 난수를 갖는 numpy 배열을 생성 #0포함, 1은 포함하거나 아니거나
- rand(d0, d1, ..., dn)을 실행하면 (d0, d1, ..., dn)의 형태를 보이는 실수 난수 배열 생성

```

np.random.rand(10)
array([ 0.59828309,  0.16485581,  0.65499111,  0.54653506,  0.605247 ,
       0.86846909,  0.00506944,  0.54786077,  0.21347468,  0.79120127])

np.random.rand(2,5)
array([[ 0.53972373,  0.79548011,  0.24350184,  0.38038225,  0.15789019],
       [ 0.32179613,  0.77290325,  0.10645029,  0.99675838,  0.72742246]])

np.random.rand()
0.9739645344043508

np.random.rand(2,3,4) #3행 4열의 행렬을 2개 만들어줌
array([[[ 0.29340694,  0.37987105,  0.96272548,  0.19498745],
       [ 0.96248981,  0.09564651,  0.84973437,  0.3985183 ],
       [ 0.68502764,  0.22651633,  0.06460557,  0.35439572]],
      [[ 0.92579369,  0.1870831 ,  0.39385043,  0.87363082],
       [ 0.22301397,  0.09546072,  0.13015511,  0.8622173 ],
       [ 0.8774342 ,  0.71309841,  0.42704078,  0.45618589]]])

```

b.random.randint([low,] high [,size])

- [low,high) 사이의 정수 난수를 갖는 numpy 배열 생성
- size : (d0, d1, ..., dn) 형식으로 입력

```

np.random.randint(10) #1에서 10까지, 사이즈는 1개
4

np.random.randint(10,20) #10에서 20까지, 사이즈는 1개
16

np.random.randint(10,20,size=(3,5))
array([[10, 14, 14, 17, 10],
       [15, 16, 16, 14, 16],
       [15, 17, 15, 17, 12]])

np.random.randint(10,20,size=3)
array([16, 17, 10])

```

7. Array 연산

1. 기본 연산 (합, 차, 곱, 나눗셈 등)

: 기본적으로 동일한 크기의 array간 연산 수행

```

arr1 = np.array([[1,3,4],[4,3,6]])
arr1
array([[1, 3, 4],
       [4, 3, 6]])

# arr2 = np.array([[10,11,12],[13,14,15]]) #방법 1
arr2 = np.arange(10,16).reshape(2,3) # 방법 2
arr2
array([[10, 11, 12],
       [13, 14, 15]])

print(arr1.shape,arr2.shape)
(2, 3) (2, 3)

```

- 배열의 합

- 배열의 차
- 배열의 곱
- 배열의 나눗셈
- 배열의 스칼라 곱
- 배열의 비교 연산

8. 배열의 Broadcasting
