numpy

최필주

- ndarray(n-dimension array) class
  - numpy에서 지원되는 class
  - 배열을 이용한 벡터화 연산 지원
  - 데이터가 메모리에 연속적 저장됨
  - 동일한 자료형만 저장 가능

  - 강의자료에서 **ar, ar0, ar1, … 등**은 배열로 선언된 변수를 의미

◉ 데이터 직접 입력

```
In [ ] : import numpy as np
         ar0 = np.array([1, 2, 3])
         ar1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
         ar2 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
In [ ] : print([1, 2, 3])
         print(ar0)
```

- 문제 – 다음 만들어 보기

```
out [ ]  array([[0, 1, 2],
                [1, 2, 3],
                [2, 3, 4]])

out [ ]  array([[0, 1, 2],
                [1, 2, 3],
                [2, 3, 4]])

out [ ]  array(['Alice', 'Tom', 'Mike'], dtype='<U5')
```

```
out [ ]  array([[1, 0, 0],
                [0, 1, 0],
                [0, 0, 1]])
```

데이터 직접 입력 – 데이터 형태

In [ ]:
```
ar0 = np.array([1.5, 2.2, 3.9])
ar1 = np.array([1, 2, 3], dtype = 'int8')
```

In [ ]:
```
ar0.dtype
ar2 = ar0.astype('int')
ar2
```

In [ ]:
```
ar0 = np.array(range(10))
ar1 = ar0.astype('i1')
ar2 = ar0.astype('i2')
ar3 = ar0.astype('f')
ar4 = ar0.astype('f8')
```

In [ ]:
```
%whos
```

배열의 생성 – 데이터: 동일값

동일한 값으로 초기화

```
In [ ] : np.zeros(4)
In [ ] : np.ones((2, 3))
In [ ] : np.zeros(4, dtype = 'i')
In [ ] : np.full((2, 3), 3.2, dtype = 'i')   # 결과는 int? float?
```

- 문제
  - 3x2 크기의 1로 채워진 배열 만들기(각 데이터는 1바이트)
  - 4x3x2 크기의 0.0으로 채워진 배열 만들기
  - 2x7 크기의 'a'로 채워진 배열 만들기

단위 행렬 만들기

```
In [ ] : np.eye(4)
In [ ] : np.eye(7, dtype = int)
In [ ] : np.eye(5, dtype = np.float64)
```

like 함수

```
In [ ] : ar = np.array([[1, 2, 3], [4, 5, 6]])
In [ ] : np.ones_like(ar)
In [ ] : np.zeros_like(ar, dtype = 'f')
In [ ] : np.full_like(ar, 7.0)                    # int? float?
```

규칙적인 수로 채우기

```
In [ ] : np.linspace(0, 15, 4)
In [ ] : np.linspace(0, 1, 5)
In [ ] : np.arange(10)
In [ ] : np.arange(1.5, 5)
In [ ] : np.arange(2, 4.5, 0.1)
```

랜덤하게 채우기

```
In [ ] : np.random.random((2, 4))
In [ ] : np.random.rand(2, 4)
In [ ] : np.random.uniform(3.5, 4, (3, 3))
In [ ] : np.random.randint(0, 7, (2, 3, 4))
In [ ] : np.random.randint(-5, 5, 7)
In [ ] : np.random.normal(50, 3, 1000)
In [ ] : np.random.randn(10, 5)
In [ ] : np.random.seed(0)
         np.random.randint(0, 10, (3, 3))
In [ ] : np.random.seed(1)
         np.random.randint(0, 10, (3, 3))
In [ ] : np.random.seed(0)
         np.random.randint(0, 10, (3, 3))
```

랜덤하게 채우기

- 예시 – 시각화하여 비교하기

```
In [ ] : %matplotlib inline
         import matplotlib.pyplot as plt
In [ ] : plt.hist(np.random.normal(10, 3, 10000), bins = 10)
         plt.hist(np.random.randn(10000), bins = 10)
In [ ] : plt.hist(np.random.random(10000), bins = 10)
In [ ] : plt.hist(np.random.rand(10000), bins = 10)
In [ ] : plt.hist(np.random.uniform(-100, 100, 10000), bins = 10)
In [ ] : plt.hist(np.random.randint(-100, 100, 10000), bins = 10)
```

- plt.hist(데이터, bins = n): n개의 구간 내 데이터들의 빈도수를 그래프화

배열 모양 정보 얻기

In [ ] :
```
ar0 = np.array([1, 2, 3])
ar1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
ar2 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

In [ ] :
```
print('차원:', ar0.ndim, '₩t크기', ar0.shape)
print('차원:', ar1.ndim, '₩t크기', ar1.shape)
print('차원:', ar2.ndim, '₩t크기', ar2.shape)
```

In [ ] :
```
print(ar0.size, ar1.size, ar2.size)
```

전치 행렬 구하기

In [ ] :
```
ar = np.arange(8).reshape((4, 2))
ar
```

In [ ] :
```
ar.T
```

In [ ] :
```
np.transpose(ar)
```

In [ ] :
```
np.swapaxes(ar, 0, 1)
```

배열 모양 변경하기 – 요소의 개수 동일

```
In [ ] : ar0 = np.arange(12)
         ar0
In [ ] : ar0.reshape(3, 4)
In [ ] : np.reshape(ar0, (4, 3))
In [ ] : ar1 = ar0.reshape(2, 2, 3)
         ar1
In [ ] : ar1.ravel()
```

배열 모양 변경하기 – 요소의 개수 변경

```
In [ ] : ar0 = np.arange(12)
         ar0
In [ ] : ar0.resize(3, 3)
In [ ] : np.reshape(ar0, (4, 4))
```

◉ 배열 요소 추가/삽입/삭제

In [ ] :
```
ar = np.arange(20).reshape(4, 5)
ar
```

In [ ] :
```
np.append(ar, 20)
```

In [ ] :
```
np.append(ar, np.arange(20, 25).reshape(1,5), axis = 0)
```

In [ ] :
```
np.insert(ar, 5, 20)
```

In [ ] :
```
np.insert(ar, 3, 7, axis = 1)
```

In [ ] :
```
np.delete(ar, 3)
```

In [ ] :
```
np.delete(ar, 3, axis = 0)
```

In [ ] :
```
np.delete(ar, 3, axis = 1)
```

indexing

```
In [ ] : ar = np.arange(12).reshape(2, 2, 3)
         ar
In [ ] : ar[0]
In [ ] : ar[0][1]          # ar[0, 1]과 같을까?
In [ ] : ar[0, 1, 2]
```

slicing

```
In [ ] : ar = np.arange(20).reshape(4, 5)
         ar
In [ ] : ar[2:]
In [ ] : ar[2:, :-2]     #ar[2:][:-2]와 같을까?
In [ ] : ar[::-1, ::-1]
```

indexing + slicing

In [ ] :
```
ar = np.arange(20).reshape(4, 5)
ar
```

out [ ]
```
array([[ 0, 1, 2, 3, 4],
       [ 5, 6, 7, 8, 9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

In [ ] :
```
ar[[0, 1]]
```

In [ ] :
```
ar[2:, [1, 3, 4]]
```

In [ ] :
```
ar[[0, 2], [2, 4]]
```

- Shallow copy vs. deep copy
  - ar.copy()의 사용

In [ ] :
```
ar0 = np.array([1, 2, 3, 4, 5])
ar1 = ar0
ar2 = ar0.copy()
```

In [ ] :
```
ar0[0] = 0
ar0
```

In [ ] :
```
ar1
```

In [ ] :
```
ar2
```

- Shallow copy vs. deep copy
  - indexing이나 slicing 된 경우

```
In [ ] : ar0 = np.arange(12).reshape((3, 4))
         ar0
In [ ] : ar1 = ar0[0]
         ar1
In [ ] : ar2 = ar1[2:]
In [ ] : ar2[0] = 77
In [ ] : ar1
In [ ] : ar0
```

Vectorization operation

```
In [ ] : a = [1, 2, 3, 4, 5]
         a * 2
In [ ] : np.array(a) * 2
```

Vectorization operation - 산술, 비교 연산

```
In [ ] : ar0 = np.random.randint(-10, 10, 5)
         ar1 = np.random.randint(-10, 10, 5)
In [ ] : ar0
In [ ] : ar1
In [ ] : ar0 * ar1
In [ ] : ar0 + ar1 * 2
In [ ] : ar0 ** 2
In [ ] : ar0 > 5
```

Vectorization operation - 산술, 비교 연산

- for문과 비교하기

In [ ] :
```
ar0 = np.random.randint(-10, 10, 100000)
ar1 = np.random.randint(-10, 10, 100000)
```

In [ ] :
```
%%time
ar2 = np.zeros_like(ar0)
for i in range(len(ar2)):
    ar2[i] = ar0[i] + ar1[i]
```

In [ ] :
```
%%time
ar3 = ar0 + ar1
```

  - %%time: 셀의 실행시간을 알려줌

Vectorization operation – broadcasting

In [ ] :
```
ar0 = np.arange(12).reshape(4, 3)
ar1 = np.arange(3)
ar2 = np.arange(4).reshape(4, 1)
print(ar0, '₩n₩n', ar1, '₩n₩n', ar2)
```

In [ ] :
```
ar0 + ar1
```

In [ ] :
```
ar0 + ar2
```

Vectorization operation – numpy의 함수 사용

In [ ] :
```
ar = np.random.randint(0, 10, 5)
ar
```

In [ ] :
```
np.exp(ar) # 자연 상수 e를 밑으로 지수 연산 수행
```

In [ ] :
```
np.log(ar[ar>0]) # 자연 상수 e를 밑으로 log 연산 수행
```

In [ ] :
```
np.log10(ar[ar>0]) # 10을 밑으로 log 연산 수행
```

In [ ] :
```
np.sin(ar)
```

In [ ] :
```
np.cos(ar)
```

In [ ] :
```
np.abs(ar)
```

◉ 행렬곱(내적)

In [ ] :
```
ar0 = np.arange(2, 22, 2).reshape(2, 5)
ar1 = np.arange(1, 11).reshape(5, 2)
print(ar0, '₩n₩n', ar1)
```

In [ ] :
```
ar0 @ ar1
```

In [ ] :
```
np.dot(ar1, ar0)
```

요소별 비교

```
In [ ]: ar0 = np.arange(12).reshape(4, 3)
        ar0 % 2 == 1
In [ ]: ar0[ar0 % 2 == 1]
In [ ]: np.dot(ar1, ar0)
In [ ]: ar0 > 5
In [ ]: ar0[ar0 > 5]
In [ ]: ar0 % 2 == 1
In [ ]: np.where(ar0 % 2 == 1)
```

배열 전체의 비교

```
In [ ]: ar0 = np.random.randint(0, 5, 10)
        ar0
In [ ]: ar1 = np.random.randint(0, 5, 10)
        ar1
In [ ]: ar0 == ar1
In [ ]: np.array_equal(ar0, ar1)
```

◉ 최소, 최대, 합계, 평균, 중앙값, 분산, 표준편차 구하기

In [ ] :
```
ar = np.random.randint(0, 10, (3, 4))
ar
```

In [ ] :
```
print(ar.min(), np.min(ar))
```

In [ ] :
```
print(ar.max(), np.max(ar))
```

In [ ] :
```
print(ar.argmin(), np.argmin(ar))
```

In [ ] :
```
print(ar.argmax(), np.argmax(ar))
```

In [ ] :
```
print(ar.sum(), np.sum(ar))
```

In [ ] :
```
print(ar.mean(), np.mean(ar))
```

In [ ] :
```
print(np.median(ar))
```

In [ ] :
```
print(ar.var(), np.var(ar))
```

In [ ] :
```
print(ar.std(), np.std(ar))
```

Axis 지정하기(2차원 배열 기준)

```
In [ ] : ar = np.random.randint(0, 10, (3, 4))
         ar
In [ ] : ar.sum()
In [ ] : ar.sum(axis = 0)
In [ ] : ar.sum(axis = 1)
```

정렬하기

```
In [ ] : ar = np.random.randint(-10, 10, (2, 5))
         ar
In [ ] : np.sort(ar)
In [ ] : ar
In [ ] : ar.sort(axis = 0)
```

차원/중복 무시하고 값들 정렬하기

```
In [ ] : ar = np.random.randint(-10, 10, (2, 5))
         ar
In [ ] : np.unique(ar)
```

배열을 수직/수평 방향으로 결합

In [ ] :
```
ar0 = np.arange(6).reshape(2, 3)
ar1 = np.arange(6, 12).reshape(2, 3)
ar2 = np.arange(12, 18).reshape(2, 3)
```

In [ ] :
```
np.concatenate([ar0, ar1, ar2])
```

In [ ] :
```
np.concatenate([ar0, ar1, ar2], axis = 1)
```

In [ ] :
```
np.vstack([ar0, ar1, ar2])
```

In [ ] :
```
np.hstack([ar0, ar1, ar2])
```