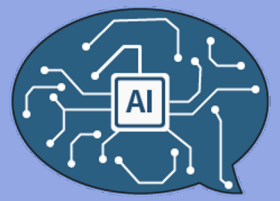
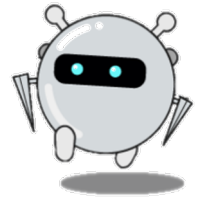
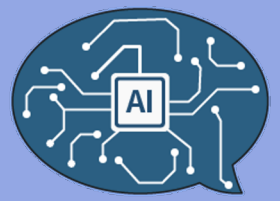
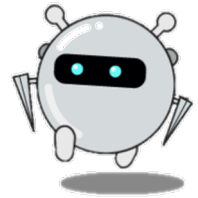


Numpy & Pandas





Numpy



Numpy 소개

- Numerical Python
- 대표적인 Python 기반 수치 해석 라이브러리

특징

- 배열 연산을 수행하는 다양한 함수 제공
- 다차원 배열을 효과적으로 처리할 수 있도록 함
- Python의 list보다 속도가 빠름

[List 사용]

```
list1 = list(range(1000000))
%time list2 = [x*2 for x in list1]
```

```
CPU times: user 56.7 ms, sys: 18 ms, total: 74.6 ms
Wall time: 74.9 ms
```

[Numpy 사용]

```
import numpy as np
arr1 = np.arange(1000000)
%time arr2 = arr1 * 2
```

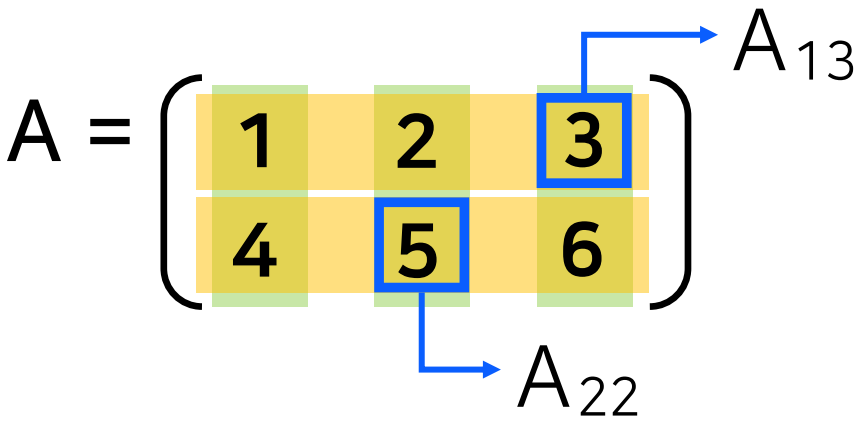
```
CPU times: user 1.64 ms, sys: 1.95 ms, total: 3.59 ms
Wall time: 3.61 ms
```

행렬

OUTTA

행렬 용어

- 행렬: 수를 직사각형 모양으로 배열한 것



- 행: 행렬의 가로줄
- 열: 행렬의 세로줄
- Shape: 2 X 3

- 성분, 원소: 행렬을 구성하는 요소

A_{ij}

행렬 용어

$A =$

1	2	3
4	5	6
7	8	9
10	11	12

$A^T =$

1	4	7	10
2	5	8	11
3	6	9	12

- Shape: 4 X 3
- 대각성분: 행과 열의 지표수가 같은 성분, A_{ij} 에서 i = j인 성분
- 주대각선: 대각성분을 지나는 선
- 전치(Transpose): 주대각선을 기준으로 행과 열을 바꾸는 것

행렬 연산

A =

1	2	3
4	5	6

- Shape: 2 X 3

B =

0	2	4
1	3	5

- Shape: 2 X 3

(1) 덧셈

A + B =

1+0	2+2	3+4
4+1	5+3	6+5

(2) 뺄셈

A - B =

1-0	2-2	3-4
4-1	5-3	6-5

⇨ 덧셈, 뺄셈 모두 두 행렬의 shape이 동일해야 함

행렬 연산

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- Shape: 2 X 3

(3) 상수배

$$A \times 2 = \begin{bmatrix} 1X2 & 2X2 & 3X2 \\ 4X2 & 5X2 & 6X2 \end{bmatrix}$$

행렬 연산

$A =$

1	2	3
4	5	6

- Shape: 2 X 3

$B =$

1	2
3	4
5	6

- Shape: 3 X 2

(4) 곱셈 ⇨ 앞 행렬의 열 개수 = 뒤 행렬의 행 개수

$A \times B =$

1X1 + 2X3 + 3X5	1X2 + 2X4 + 3X6
4X1 + 5X3 + 6X5	4X2 + 5X4 + 6X6

- Shape: 2 X 2

Numpy 기능

Numpy 배열 : ndarray (N-dimensional array)

- 하나의 배열 속 값들은 모두 동일한 자료형

1차원 배열

1.3	2.2	3.1

axis = 0

- dtype = float
- shape = (3,)

2차원 배열

T	F	T
F	T	F

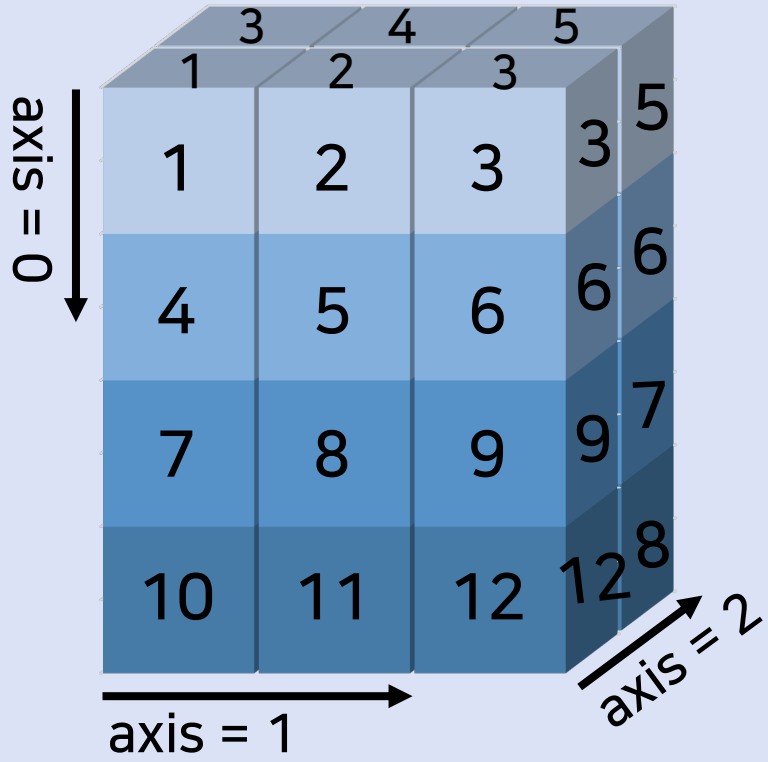
axis = 0

axis = 1

- dtype = bool
- shape = (2, 3)

3차원 배열

- dtype = int
- shape = (4, 3, 2)



OUTTA

Numpy 배열 : Ndarray (N-dimensional array)

- 하나의 배열 속 값들은 모두 동일한 자료형

1차원 배열

```
[1.3  2.2  3.1]
```

- shape = (3,)

2차원 배열

Tip) 괄호의 개수 = 차원

```
[[ True False  True]
 [False  True False]]
```

- shape = (2, 3)

3차원 배열

Tip) 2D 배열 개수 파악 → 2D 배열 형태 파악

```
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]

 [[ 3  4]
  [ 5  4]
  [ 5  6]]

 [[ 5  6]
  [ 7  6]
  [ 7  8]]]
```

- shape = (4, 3, 2)

배열 속성 확인

- ndarray.dtype: 데이터의 자료형 확인
- ndarray.ndim: 데이터의 차원 확인
- ndarray.shape: 데이터의 축(axis)별 크기 확인
- ndarray.size: 데이터의 전체 요소 개수 확인

구분		Type	Type Code
숫자형 (numeric)	bool형 (booleans)	bool	?
	정수형 (integers)	int8 int16 int32 int64	i1 i2 i4 i8
	부호없는 (양수) 정수형 (unsigned integers)	uint8 uint16 uint32 uint64	u1 u2 u4 u8
	부동소수형 (floating points)	float16 float32 float64	f2 f4 f8
	복수수형(실수 + 허수) (complex)	complex64 complex128	c8 c16
문자형 (character)	문자형 (string)	string_	S

다음 배열의 속성은?

```
print(arr)
```

```
[[[ 1.23097977  0.81635839  1.4948242 ]
  [-0.7771737  -1.93447111  0.06874766]
  [ 0.17122791  1.89280083 -0.7387371 ]
  [-0.31382569 -1.75302997  0.34374879]]

 [[ 0.35175738 -0.71792605  0.56032363]
  [-0.39320025  0.45868636 -0.90386499]
  [ 0.5319692  -0.17565067 -0.55597465]
  [-0.31341657  1.80514092 -0.48895059]]]
```

- arr.dtype: **float64**
- arr.ndim: **3**
- arr.shape: **(2, 4, 3)**
- arr.size: **24**

OUTTA

배열 생성

유형1

- np.array(list or tuple): Python의 list나 tuple을 numpy 배열로 만듦

유형2 # 모두 같은 원소

- np.zeros(shape): 모든 원소가 0인 배열 생성
- np.ones(shape): 모든 원소가 1인 배열 생성
- np.full(shape, n): 모든 원소가 n인 배열 생성

유형3 # 연속적인 원소

- np.arange(start, stop, step): start 이상 stop 미만 간격이 step인 배열 생성
- np.linspace(start, stop, n등분): start 이상 stop 이하 n등분하여 배열 생성

배열 생성

유형4 # 주어진 배열 존재

- np.random.choice(data, shape): 주어진 배열에서 임의로 원소 선택하여 배열 생성
- np.random.shuffle(data): 주어진 배열의 순서 임의로 변경

유형5 # 주어진 배열 존재 X

- np.random.rand(shape): [0, 1)에서 각 구간의 난수 수가 균등분포 따르도록 샘플링
- np.random.randn(shape): 난수가 표준정규분포를 따르도록 샘플링
- np.random.randint(start, stop, shape): start 이상 stop 미만인 정수 샘플링

배열 생성

유형1

- np.array(list or tuple): Python의 list나 tuple을 numpy 배열로 만듦

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

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

배열 생성

유형2 # 모두 같은 원소

- np.zeros(shape, dtype): 모든 원소가 0인 배열 생성
- np.ones(shape, dtype): 모든 원소가 1인 배열 생성
- np.full(shape, n): 모든 원소가 n인 배열 생성

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

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

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

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

```
np.full((2, 3), 5)
```

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

배열 생성

유형3 # 연속적인 원소

- np.arange(start, stop, step): start 이상 stop 미만 간격이 step인 배열 생성
- np.linspace(start, stop, n등분): start 이상 stop 이하 n등분하여 배열 생성

```
np.arange(1, 9, 2)
```

```
array([1, 3, 5, 7])
```

```
np.linspace(1, 9, 5)
```

```
array([1., 3., 5., 7., 9.])
```

배열 생성

유형4 # 주어진 배열 존재

- np.random.choice(data, shape): 주어진 배열에서 임의로 원소 선택하여 배열 생성
- np.random.shuffle(data): 주어진 배열의 순서 임의로 변경

유형5 # 주어진 배열 존재 X

- np.random.rand(shape): [0, 1)에서 각 구간의 난수 수가 균등분포 따르도록 샘플링
- np.random.randn(shape): 난수가 표준정규분포를 따르도록 샘플링
- np.random.randint(start, stop, shape): start 이상 stop 미만인 정수 샘플링

배열 생성

유형4 # 주어진 배열 존재

- np.random.choice(data, shape): 주어진 배열에서 임의로 원소 선택하여 배열 생성
- np.random.shuffle(data): 주어진 배열의 순서 임의로 변경

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

```
np.random.choice(data, (2, 3))
```

```
array([[5, 5, 2],
       [2, 4, 2]])
```

```
np.random.shuffle(data)
data
```

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

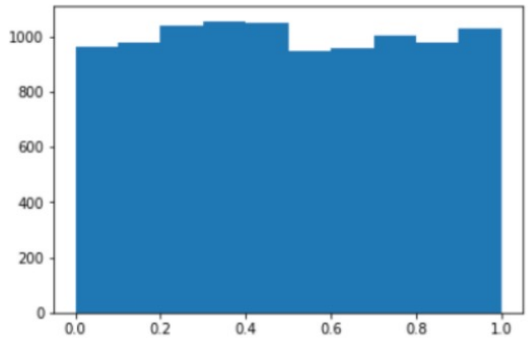
배열 생성

유형5 # 주어진 배열 존재 X

- np.random.rand(shape): [0, 1)에서 각 구간의 난수 수가 균등분포 따르도록 샘플링
- np.random.randn(shape): 난수가 표준정규분포를 따르도록 샘플링
- np.random.randint(start, stop, shape): start 이상 stop 미만인 정수 샘플링

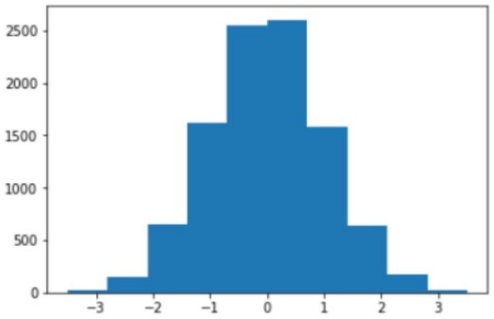
```
np.random.rand(2, 3)
```

```
array([[0.77297586, 0.06617773, 0.10479327],
       [0.19086849, 0.99001807, 0.88904369]])
```




```
np.random.randn(2, 3)
```

```
array([[ 1.19905106,  0.59890451, -0.08861081],
       [ 0.55784835,  1.32297991,  0.03322923]])
```



OUTTA

배열 생성

유형5 # 주어진 배열 존재 X

- np.random.rand(shape): [0, 1)에서 각 구간의 난수 수가 균등분포 따르도록 샘플링
- np.random.randn(shape): 난수가 표준정규분포를 따르도록 샘플링
- np.random.randint(start, stop, shape): start 이상 stop 미만인 정수 샘플링

```
np.random.randint(1, 10, (2, 3))
```

```
array([[9, 1, 7],
       [2, 7, 2]])
```

배열 형태 변환

- .reshape(shape): 지정한 shape으로 형태 변환
- .ravel(): 1차원으로 형태 변환
- .T: 전치(Transpose) 변환

np.arange(0, 6)

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

np.arange(6).reshape(3, 2)

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

np.arange(6).reshape(3, 2).T

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

배열 연산

[List 사용]

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
print(list1 + list2)
```

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

[Numpy 사용]

```
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])
print(arr1 + arr2)
```

```
[ 6  8 10 12]
```

배열 연산

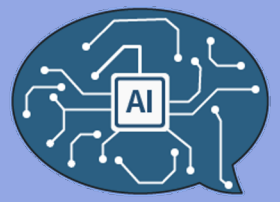
- Numpy는 기본적으로 같은 위치에 있는 원소끼리 연산 수행
  - array1 + array2: 같은 위치에 있는 원소끼리 덧셈 수행
  - array1 - array2: 같은 위치에 있는 원소끼리 뺄셈 수행
  - array1 * array2: 같은 위치에 있는 원소끼리 곱셈 수행
  - array1 / array2: 같은 위치에 있는 원소끼리 나눗셈 수행

- 행렬의 곱 (앞 행렬의 열 개수 = 뒤 행렬의 행 개수)
  - array1 @ array2

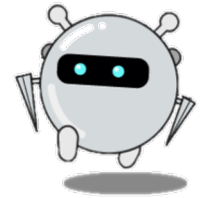
배열 연산

- Numpy 집계함수

함수	설명
np.**sum**(array)	전체 원소의 합을 계산
np.**mean**(array)	전체 원소의 평균을 계산
np.**std**(array), np.**var**(array)	전체 원소의 표준편차, 분산을 계산
np.**min**(array), np.**max**(array)	전체 원소의 최솟값, 최댓값 반환
np.**argmin**(array), np.**argmax**(array)	전체 원소의 최솟값, 최댓값이 위치한 인덱스 반환
np.**cumsum**(array)	첫 원소부터 끝 원소까지의 누적합 반환
np.**cumprod**(array)	첫 원소부터 끝 원소까지의 누적곱 반환



감사합니다



OUTTA