

2주차(2/3)

넴파이 튜토리얼 1

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

넘파이 튜토리얼 1

- 학습 목표
 - 기계학습에서 왜 넘파이를 사용하는지 이해한다.
 - 넘파이 개념과 기본적인 사용법을 익힌다.
- 학습 내용
 - 넘파이의 특징
 - 왜 넘파이인가?
 - 배열의 속성
 - 배열의 인덱싱과 슬라이싱
 - 배열/행렬의 연산

넘파이 튜토리얼 1

- 넘파이의 특징
 - NumPy – Numerical Python
 - 강력한 다차원 배열과 행렬 연산
 - 다양한 선형 대수학 함수와 난수
 - 간단한 코딩

넘파이 튜토리얼 1

- NumPy 라이브러리 사용법

1. `import numpy`
2. `import numpy as np`

```
In [1]: import numpy  
numpy.__version__
```

```
Out[1]: '1.14.0'
```

```
In [2]: import numpy as np  
np.__version__
```

```
Out[2]: '1.14.0'
```

왜 넘파이인가?

- 쉬운 다차원 행렬 연산
- 예제: 천만번 곱셈과 합
 1. 두 개의 **list**에 난수를 저장
 2. 두 개의 **numpy** 배열로 복사
 3. 각 원소별로 곱하고 합산
 - 한 번은 **for loop**로 실행
 - 한 번은 **numpy** 로 실행
 4. 두 계산 방법의 비교

```
1 import numpy as np
2 n = 10000000
3 w = [np.random.random() for _ in range(n)]
4 x = [np.random.random() for _ in range(n)]
5
6 # 리스트의 값을 np 배열로 복사
7 wnum = np.array(w) # ndarray type
8 xnum = np.array(x)
```

왜 넘파이인가?

- 쉬운 다차원 행렬 연산
- 예제: 천만번 곱셈과 합
 1. 두 개의 **list**에 난수를 저장
 2. 두 개의 **numpy** 배열로 복사
 3. 각 원소별로 곱하고 합산
 - 한 번은 **for loop**로 실행
 - 한 번은 **numpy** 로 실행
 4. 두 계산 방법의 비교

$$\frac{1.35}{0.00698} = 193.4$$

```
1 %%time
2 total = 0
3 for i in range(n):
4     total += w[i]*x[i]
5 print(total)
```

2500797.781292101
Wall time: 1.35 s

```
1 %%time
2 total = np.dot(wnum, xnum)
3 print(total)
```

2500797.7812923864
Wall time: 6.98 ms

왜 넘파이인가?

- 쉬운 다차원 행렬 연산
- 예제: 천만번 곱셈과 합
 1. 두 개의 **list**에 난수를 저장
 2. 두 개의 **numpy** 배열로 복사
 3. 각 원소별로 곱하고 합산
 - 한 번은 **for loop**로 실행
 - 한 번은 **numpy** 로 실행
 4. 두 계산 방법의 비교

속도 비교

$$\frac{1.35}{0.00698} = 193.4$$

코드 비교

```
total = 0
for i in range(n):
    total += w[i]*x[i]
```

```
total = np.dot(wnum, xnum)
```

NumPy배열의 속성

- **ndarray** – 넘파이 클래스 이름
- **ndarray** 속성
 - **ndim** – 차원, **axis** 개수, **rank**
 - **shape** – 형상, 각 차원의 배열의 크기, 예: (n,m)
 - **size** – 배열의 모든 원소의 개수
 - **dtype** – 원소의 자료 형식, 예: **float64**, **int32**...

1차원 배열

axis 0 →



Shape(3,)

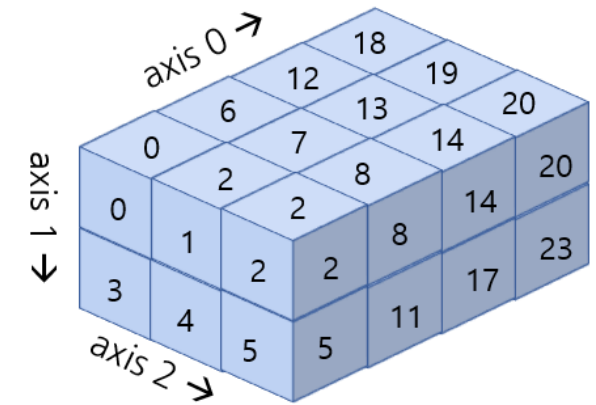
2차원 배열

axis 1 →



Shape(2, 3)

3차원 배열



Shape(4, 2, 3)

배열의 속성 출력

- pprint() 함수

```
1 def pprint(arr):
2     print("type:{}, size:{}".format(type(arr), arr.size))
3     print("shape:{}, ndim/rank:{}, dtype:{}".
4           format(arr.shape, arr.ndim, arr.dtype))
5     print("Array's Data:")
6     print(arr)
```

배열의 생성

- 배열 생성
 - `np.array()` - 리스트, 튜플 이용

```
a = np.array([1,2,3,4]) # right - list  
a = np.array((1,2,3,4)) # right - tuple  
a = np.array(1,2,3,4)  # wrong
```

배열의 생성


- 배열 생성
 - `np.array()` - 리스트, 튜플 이용
 - 배열 생성 함수

```
1 import numpy as np
2 a = np.arange(12)
3 pprint(a)
```

```
type:<class 'numpy.ndarray'>, size:12
shape:(12,), ndim/rank:1, dtype:int32
Array's Data:
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

배열의 생성

- 배열 생성
 - `np.array()` - 리스트, 튜플 이용
 - 배열 생성 함수



```
1 a = np.arange(12, dtype=float).reshape(3, 4)
2 pprint(a)
```

```
type:<class 'numpy.ndarray'>, size:12
shape:(3, 4), ndim/rank:2, dtype:float64
Array's Data:
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]
```

배열의 생성

- 배열 생성 함수
 - **zeros** – 모든 원소 0
 - **ones** – 모든 원소 1
 - **full** – 사용자가 지정한 한 값
 - **empty** – 임의의 값
 - **eye** – 단위 행렬

```
np.ones((3, 4))
```

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

```
np.full((2, 3), 7)
```

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

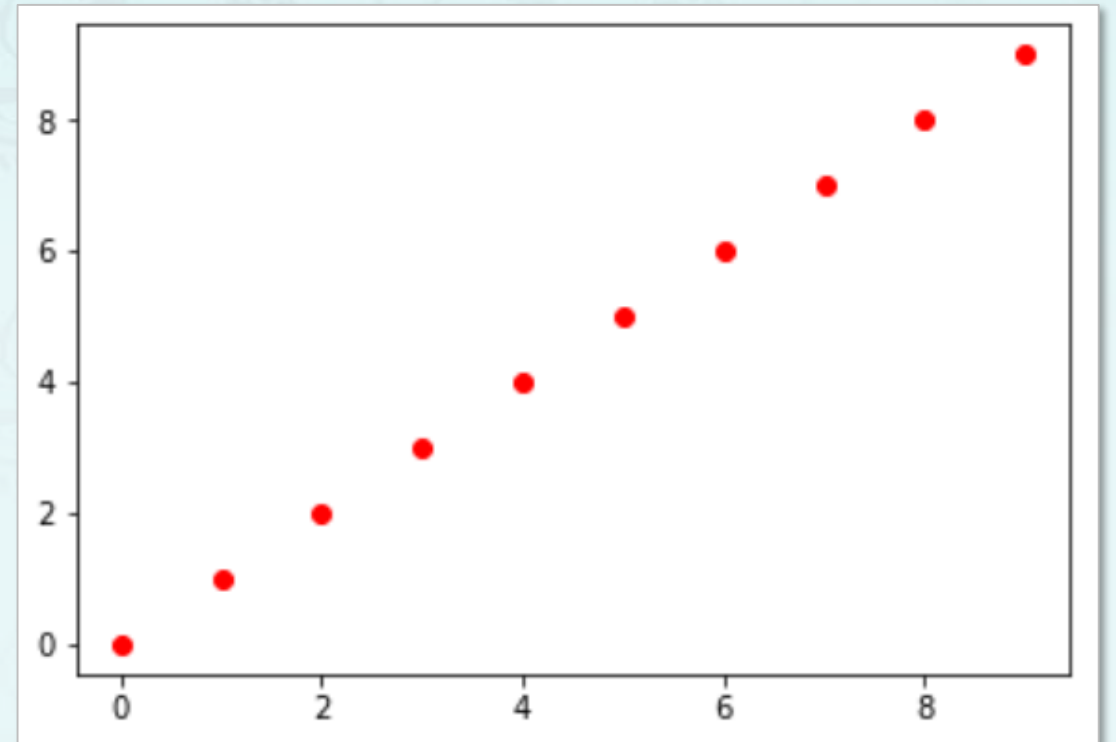
배열의 생성

- 데이터 생성 함수
 - **arange([start,] stop[, step,], dtype=None)**
start에서 **stop**미만까지, **step**간격으로 데이터 생성
 - **linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)**
start부터 **stop**의 범위에서 **num** 개의 데이터를 균일한 간격으로 생성
 - **logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)**
start부터 **stop**의 범위에서 로그 스케일로 **num** 개의 데이터를 생성

배열의 생성

- 데이터 생성 함수
 - `arange([start,] stop[, step,], dtype=None)`
`start`에서 `stop`미만까지, `step`간격으로 데이터 생성

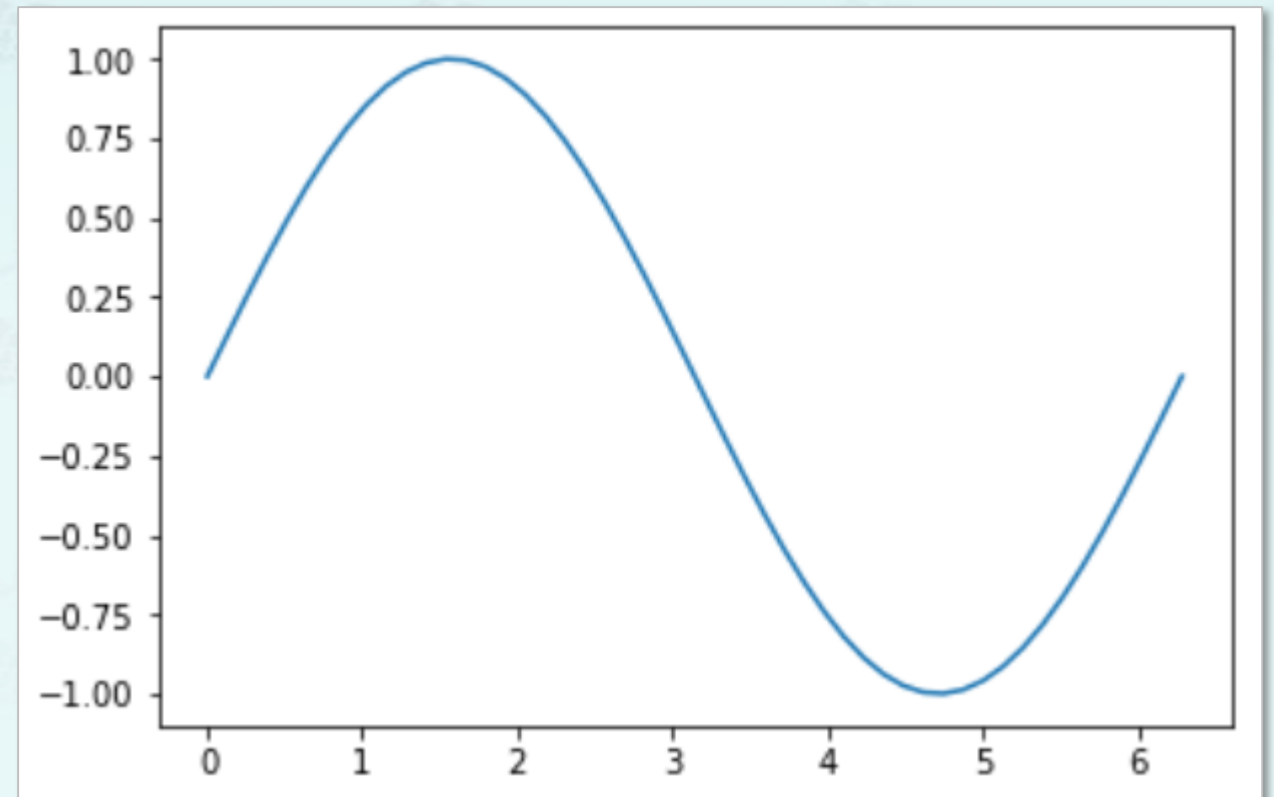
```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3
4 a = np.arange(10)
5 plt.plot(a, 'or')
6 plt.show()
```



배열의 생성

- 데이터 생성 함수
 - `linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)`
`start`부터 `stop`의 범위에서 `num` 개의 데이터를 균일한 간격으로 생성

```
1 x = np.linspace(0, 2*np.pi)
2 y = np.sin(x)
3 plt.plot(x, y)
4 plt.show()
```

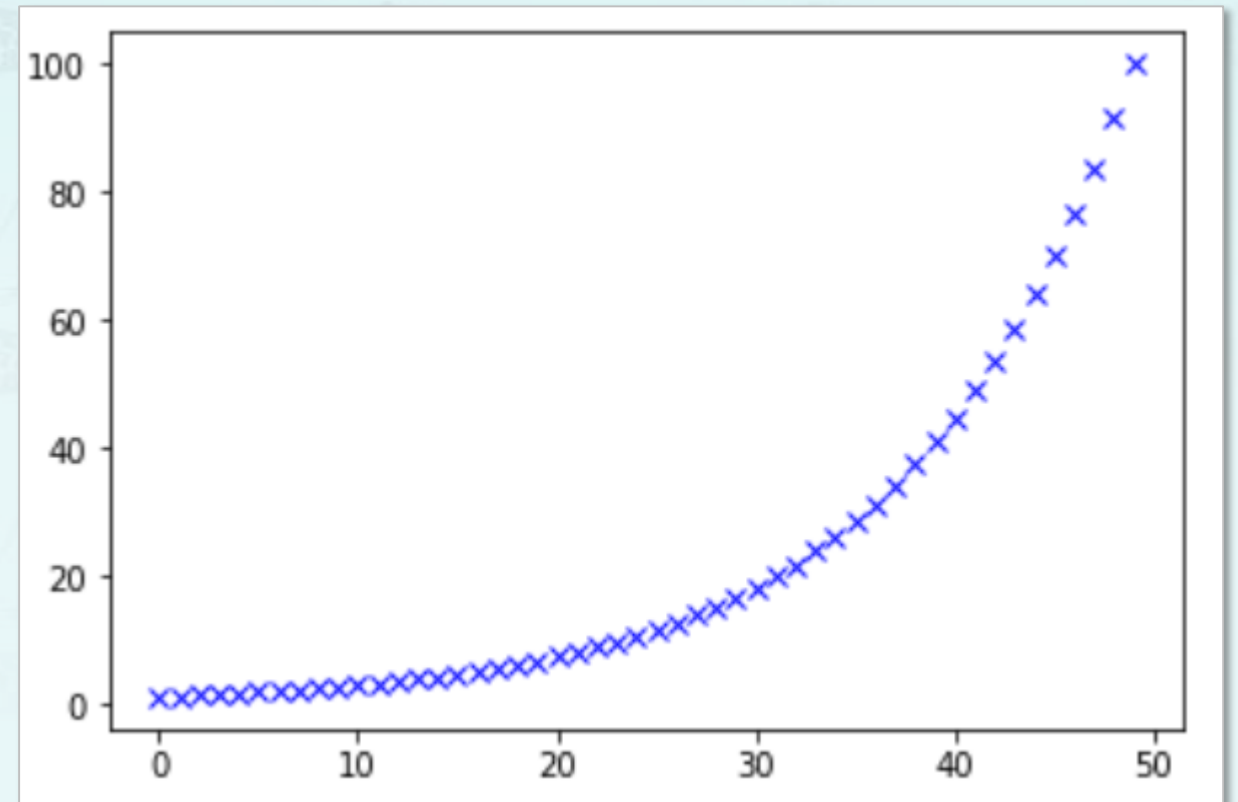


배열의 생성

- 데이터 생성 함수
 - `logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)`
`start`부터 `stop`의 범위에서 로그 스케일로 `num` 개의 데이터를 생성
 - 출력 범위는 $base^{start} \sim base^{end}$

출력 범위 $10^{0.1} \sim 10^2$

```
1 a = np.logspace(0.1, 2)  
2 plt.plot(a, 'xb')  
3 plt.show()
```



배열 인덱싱과 슬라이싱

- 인덱싱(indexing)
 - 0 부터 시작
 - -1 배열의 끝(음수 indexing)
 - : 범위 지정(start:end)
- 예:
 1. [2:5] 원소 2, 3, 4
 2. [:5] 원소 0, 1, 2, 3, 4
 3. [5:] 원소 5부터 끝까지
 4. [:] 모든 원소

배열 인덱싱과 슬라이싱

■ 2차원 배열 인덱싱 연습



1. 원소 **7**:
2. 원소 **12**:
3. **A**의 마지막 원소:
4. **A**의 마지막 행:
5. **A**의 첫째 열:
6. **A**의 마지막 열:
7. **A**의 위, 왼쪽의 **2x2**:
8. **A**의 아래, 오른쪽 **2x2**:
9. **A**의 첫 **2** 열:
10. **A** 전체:

A	2차원 배열: Shape(3,4)			
	1	2	3	4
	5	6	7	8
	9	10	11	12

배열 인덱싱과 슬라이싱

■ 2차원 배열 인덱싱 연습

1. 원소 7: **A[1,2]**



2. 원소 12:

3. A의 마지막 원소:

4. A의 마지막 행:

5. A의 첫째 열:

6. A의 마지막 열:

7. A의 위, 왼쪽의 2x2:

8. A의 아래, 오른쪽 2x2:

9. A의 첫 2 열:

10. A 전체:

A	2차원 배열: Shape(3,4)			
	1	2	3	4
	5	6	7	8
	9	10	11	12

배열 인덱싱과 슬라이싱

■ 2차원 배열 인덱싱 연습

1. 원소 7: $A[1,2]$

2. 원소 12: $A[2, 3]$



3. A 의 마지막 원소:

4. A 의 마지막 행:

5. A 의 첫째 열:

6. A 의 마지막 열:

7. A 의 위, 왼쪽의 2×2 :

8. A 의 아래, 오른쪽 2×2 :

9. A 의 첫 2 열:

10. A 전체:

A	2차원 배열: Shape(3,4)			
	1	2	3	4
	5	6	7	8
	9	10	11	12

배열 인덱싱과 슬라이싱

■ 2차원 배열 인덱싱 연습

1. 원소 7: $A[1,2]$

2. 원소 12: $A[2, 3]$

➡ 3. A 의 마지막 원소: $A[-1, -1]$

4. A 의 마지막 행:

5. A 의 첫째 열:

6. A 의 마지막 열:

7. A 의 위, 왼쪽의 2×2 :

8. A 의 아래, 오른쪽 2×2 :

9. A 의 첫 2 열:

10. A 전체:

A	2차원 배열: Shape(3,4)			
	1	2	3	4
	5	6	7	8
	9	10	11	12

배열 인덱싱과 슬라이싱

■ 2차원 배열 인덱싱 연습

1. 원소 7: $A[1,2]$
2. 원소 12: $A[2, 3]$
3. A 의 마지막 원소: $A[-1, -1]$
- ➔ 4. A 의 마지막 행:
5. A 의 첫째 열:
6. A 의 마지막 열:
7. A 의 위, 왼쪽의 2×2 :
8. A 의 아래, 오른쪽 2×2 :
9. A 의 첫 2 열:
10. A 전체:

A	2차원 배열: Shape(3,4)			
	1	2	3	4
	5	6	7	8
	9	10	11	12

배열 인덱싱과 슬라이싱

■ 2차원 배열 인덱싱 연습

1. 원소 7: $A[1,2]$
2. 원소 12: $A[2, 3]$
3. A 의 마지막 원소: $A[-1, -1]$
4. A 의 마지막 행: $A[-1]$
- 5. A 의 첫째 열:
6. A 의 마지막 열:
7. A 의 위, 왼쪽의 2×2 :
8. A 의 아래, 오른쪽 2×2 :
9. A 의 첫 2 열:
10. A 전체:

A	2차원 배열: Shape(3,4)			
	1	2	3	4
	5	6	7	8
	9	10	11	12

배열 인덱싱과 슬라이싱

■ 2차원 배열 인덱싱 연습

1. 원소 7: $A[1,2]$
2. 원소 12: $A[2, 3]$
3. A 의 마지막 원소: $A[-1, -1]$
4. A 의 마지막 행: $A[-1]$
- ➔ 5. A 의 첫째 열: $A[:,0]$
6. A 의 마지막 열:
7. A 의 위, 왼쪽의 2×2 :
8. A 의 아래, 오른쪽 2×2 :
9. A 의 첫 2 열:
10. A 전체:

A	2차원 배열: Shape(3,4)			
	1	2	3	4
	5	6	7	8
	9	10	11	12

배열 인덱싱과 슬라이싱

■ 2차원 배열 인덱싱 연습

1. 원소 7: $A[1,2]$
2. 원소 12: $A[2, 3]$
3. A 의 마지막 원소: $A[-1, -1]$
4. A 의 마지막 행: $A[-1]$
- 5. A 의 첫째 열: $A[:,0]$
6. A 의 마지막 열: $A[:, -1]$
7. A 의 위, 왼쪽의 2×2 : $A[:2, :2]$
8. A 의 아래, 오른쪽 2×2 : $A[-2:, -2:]$
9. A 의 첫 2 열: $A[:, :2]$
10. A 전체: $A[:, :]$

A	2차원 배열: Shape(3,4)			
	1	2	3	4
	5	6	7	8
	9	10	11	12

배열 인덱싱과 슬라이싱

- 2차원 배열 인덱싱 연습

```
1 a = np.arange(1, 13).reshape(3,4)
2 print(a[1,2])
3 print(a[2, 3])
4 print(a[-1, -1])
5 print(a[-1])
6 print(a[:,0])
7 print(a[:, -1])
8 print(a[:2, :2])
9 print(a[-2:, -2:])
10 print(a[:, :2])
11 print(a[:, :])
```


A 2차원 배열: Shape(3,4)

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

배열 인덱싱과 슬라이싱

- 슬라이싱 – 서브배열(subarray)

```
1 a = np.arange(1, 13).reshape(3,4)
2 b = a[:2, :2]
3 print(b)
4 b[0, 0] = 99
5 print(a)
```



A 2차원 배열: Shape(3,4)

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

배열의 복사

- 슬라이싱 - 서브배열(subarray)

```
1 a = np.arange(1, 13).reshape(3,4)
2 b = a[:2, :2]
3 print(b)
4 b[0, 0] = 99
5 print(a)
```

```
[[1 2]
 [5 6]]
[[99 2 3 4]
 [ 5 6 7 8]
 [ 9 10 11 12]]
```

A 2차원 배열: Shape(3,4)

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

배열의 복사


```
1 a = np.arange(1, 13).reshape(3,4)
2 aa = a
3 aa[0, 0] = 99
4 print(a)
```

```
[[99  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

배열의 복사

- 배열을 복사하는 방법은?

```
1 a = np.arange(1, 13).reshape(3,4)
2 aa = a.copy()
3 aa[0, 0] = 99
4 print(a)
```




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

불린 배열 인덱싱


- 불린(**Boolean**)값으로 된 배열

```
1 a = np.random.random(7)
2 results = a > 0.6
3 print(results)
```



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

```
print(np.sum(results))
print(np.argwhere(results))
```



```
3
[[3]
 [4]
 [5]]
```


배열/행렬의 곱

- 두 열 벡터 $\mathbf{x}, \mathbf{y} \in R^m$
 - 내적: $x^T y \rightarrow$ 스칼라
 - 외적: $xy^T \rightarrow \mathbf{m} \times \mathbf{m}$ 행렬

배열/행렬의 곱

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

- 내적: $x^T y \rightarrow$ 스칼라

$$\begin{aligned} \mathbf{x}^T \cdot \mathbf{y} &= (x_1 \ x_2 \ \cdots \ x_m) \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \\ &= \sum_{i=1}^m x_i y_i \\ &= x_1 y_1 + x_2 y_2 + \cdots + x_m y_m \end{aligned}$$

- 외적: $x y^T \rightarrow m \times m$ 행렬

$$\begin{aligned} \mathbf{x} \cdot \mathbf{y}^T &= \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} (y_1 \ y_2 \ \cdots \ y_m) \\ &= \begin{pmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_m \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_m \\ \cdots & \cdots & \cdots & \cdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_m \end{pmatrix} \end{aligned}$$

넘파이 튜토리얼 1

- 학습 정리
 - 넘파이 라이브러리 소개
 - 넘파이의 특성 학습
 - **ndim, axis, size, shape, dtype**
 - 인덱싱과 슬라이싱 원리 학습
 - **subarray** (서브 배열)를 다룰 때 조심해야 할 점
 - 행렬의 연산 방법 학습