# 선형대수학

# 머신러닝에서 쓰이는 수학

1. 선형대수
   - 행렬

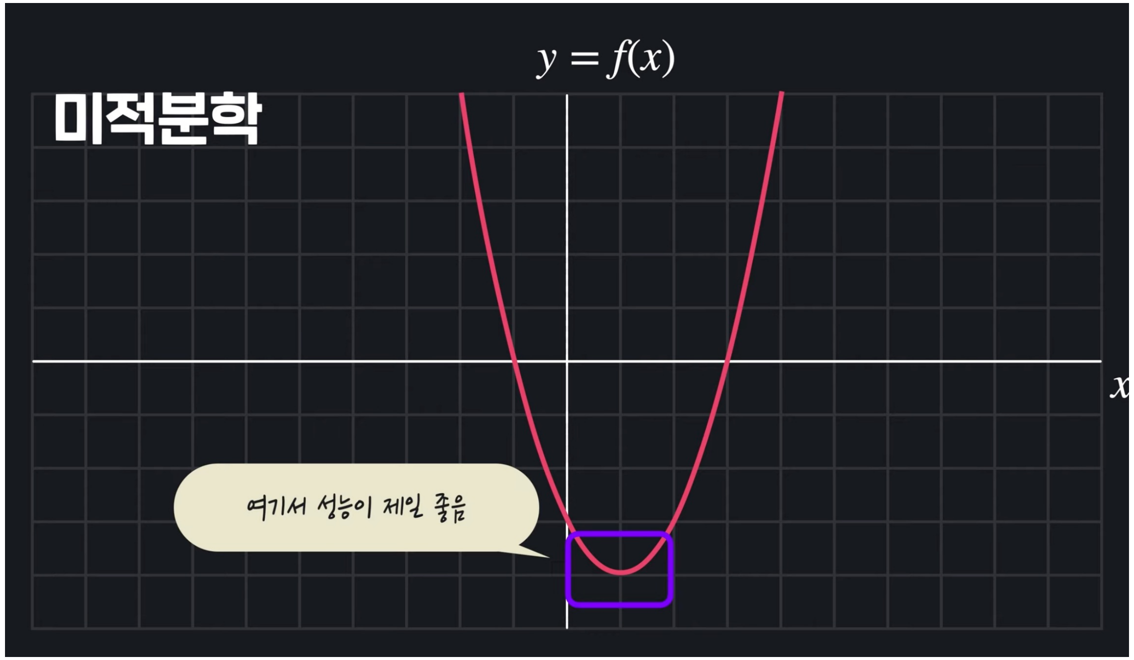
2. 미적분학
   - 최적화

3. 통계
   - 데이터 이해
   - 패턴

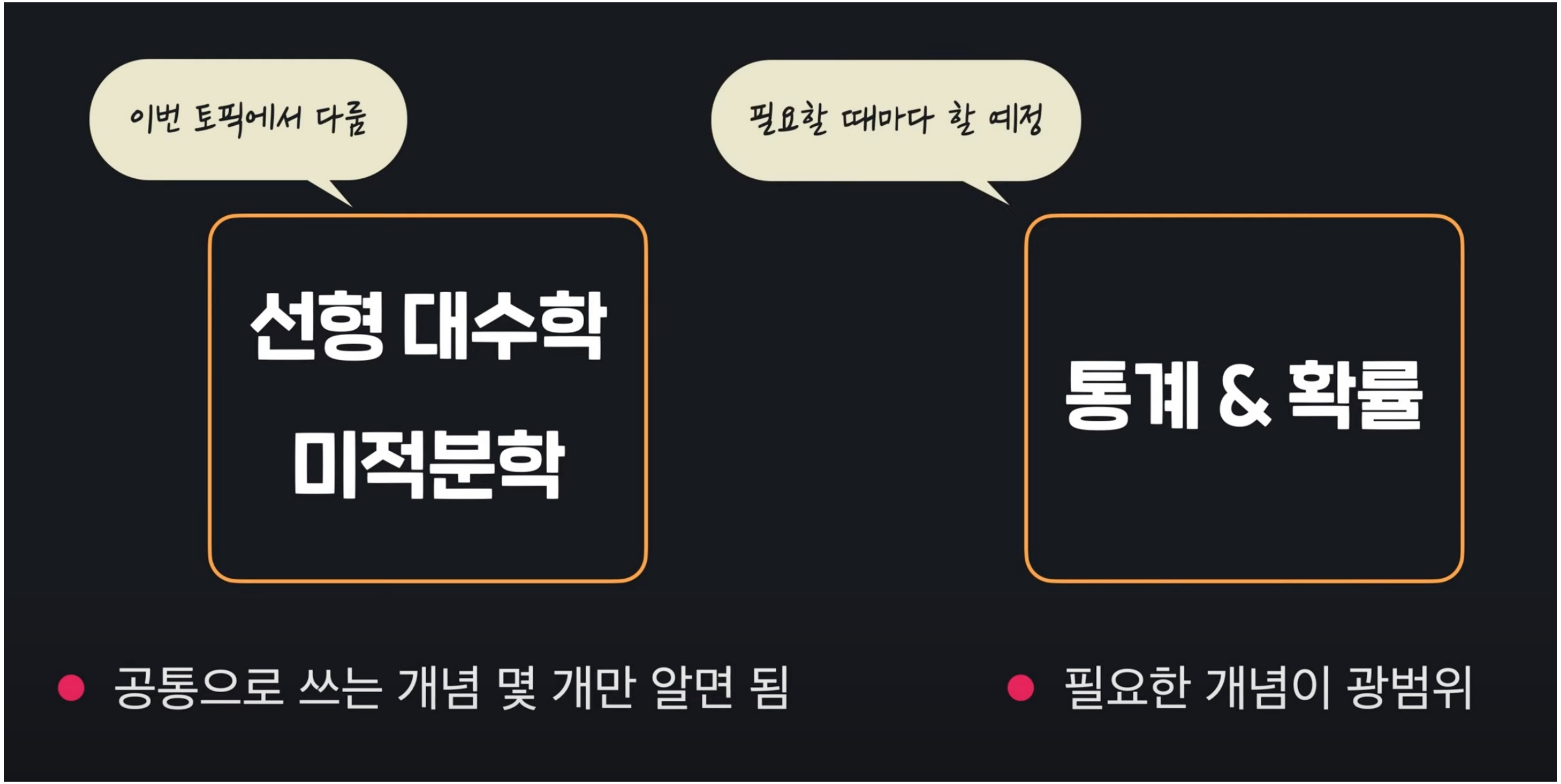
4. 확률
   - 예측/가능성

**행렬**

$$A = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 2 & 1 & 4 & 1 \\ 0 & 3 & 2 & 1 \end{bmatrix} \qquad a = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 4 \end{bmatrix}$$

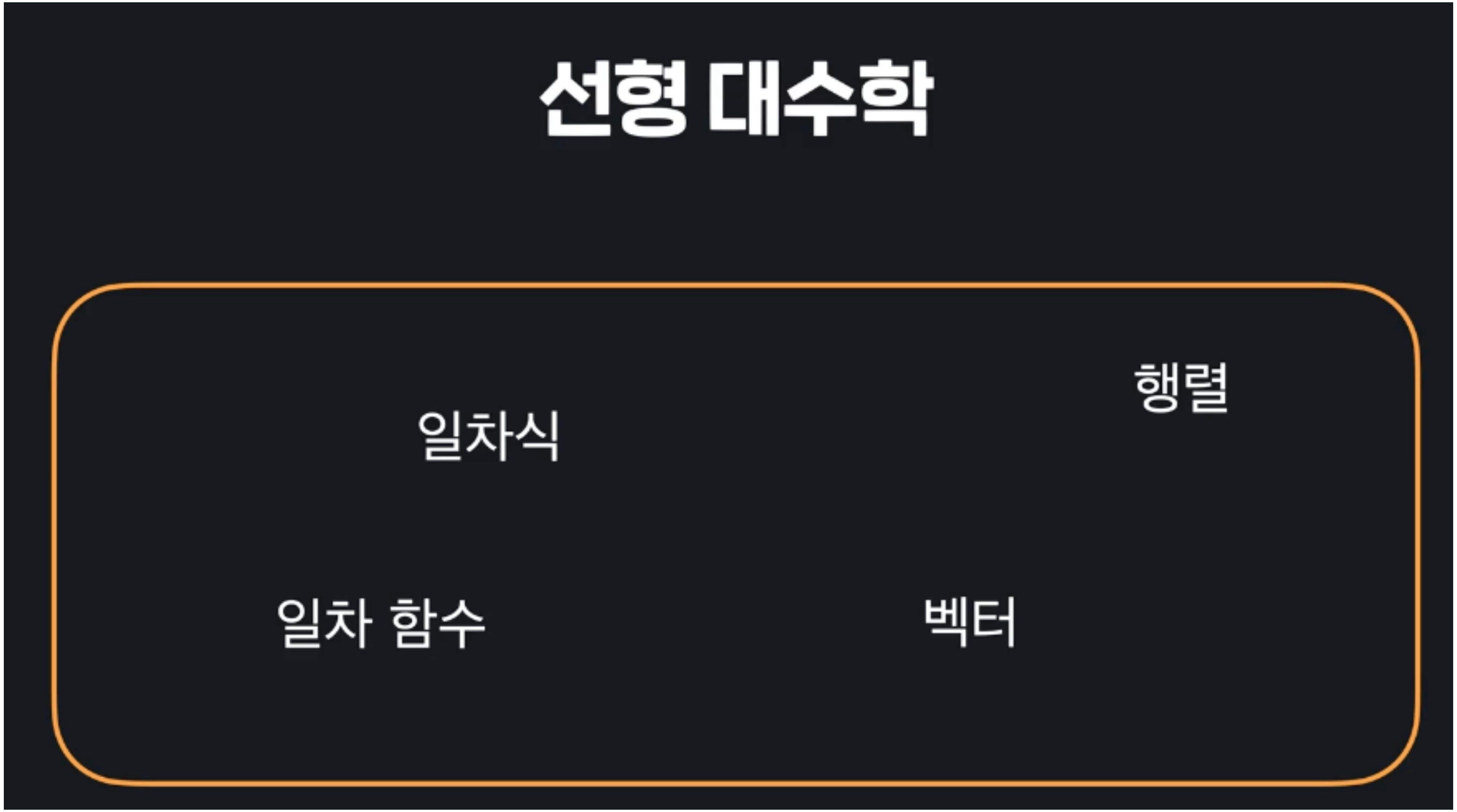



# 머신러닝에서 쓰이는 수학



# 선형대수학을 배우는 이유

- 선형대수학 : 일차식이나 일차 함수를 공부하는 학문



# 선형대수학을 배우는 이유

- 복잡한 선형 시스템을 행렬과 벡터로 쉽게 표현 가능

선형 시스템

$$2x_0 - 4x_1 + x_2 = 3$$
$$3x_0 + x_1 - 6x_2 = 10$$
$$x_0 + x_1 + x_2 = 5$$

$$\boxed{Ax = y}$$

$$\begin{bmatrix} 2 & -4 & 1 \\ 3 & 1 & -6 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 5 \end{bmatrix}$$

아무리 복잡한 선형 시스템도
행렬과 벡터로 쉽게 표현할 수 있음!

$$\begin{bmatrix} 2x_0 - 4x_1 + x_2 \\ 3x_0 + x_1 - 6x_2 \\ x_0 + x_1 + x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 5 \end{bmatrix}$$

# 일차 식과 일차 함수

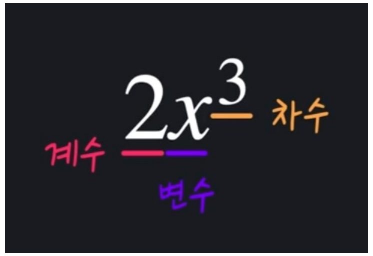
# (n차)다항식

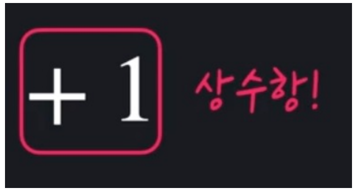
- 일차식 : 가장 높은 차수가 1인 다항식

- n차식 : 가장 높은 차수가 n인 다항식

  e.g. 3차 다항식

$$2x^3 - y^2 + 4y + 1$$



# (n차)함수

- 일차 함수

y는 x에 대한 함수 $y = 3x + 6$

함수 f는 x에 대한 함수 $f(x) = 3x + 6$

- 3차 함수

$$f(x, y) = 2x^3 - y^2 + 4y + 1$$

# 일차식 표기법

# 변수 개수에 따른 함수 표현

- 변수 x, y에 대한 함수(예시)

$$f(x, y) = 3x + 6y + 4$$

- 변수 여러 개에 대한 함수 표현

$$f(x, y, z, a, b, c) = 4x + y - 2z + 16a + 7b + 3c$$

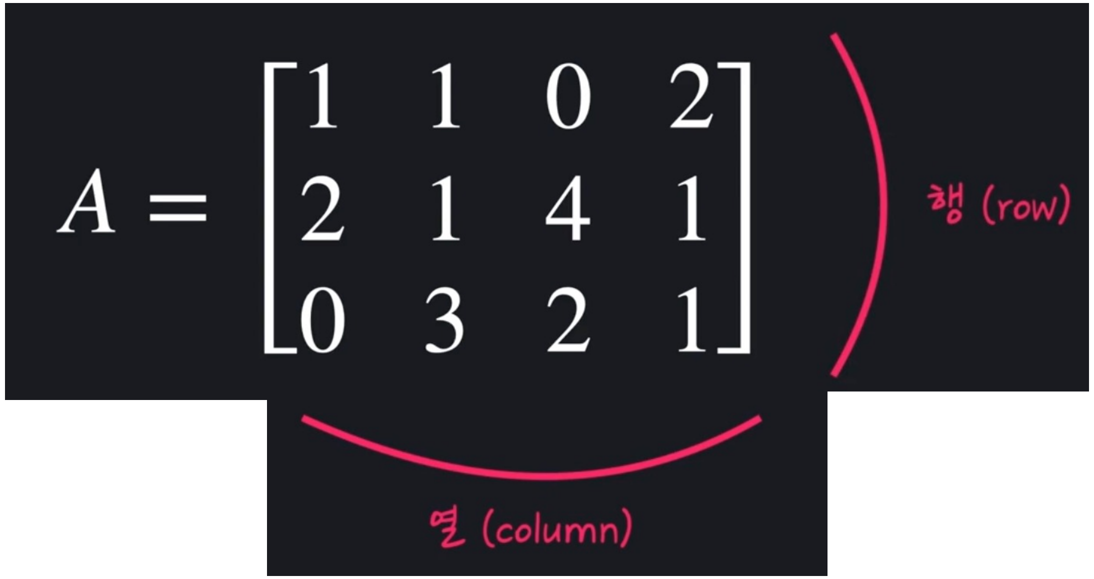
$$\underbrace{f(x_0, x_1, \ldots, x_n)}_{\text{변수}} = \underbrace{a_0}_{\text{계수}} x_0 + a_1 x_1 + \cdots + a_n x_n + \underbrace{b}_{\text{상수항}}$$

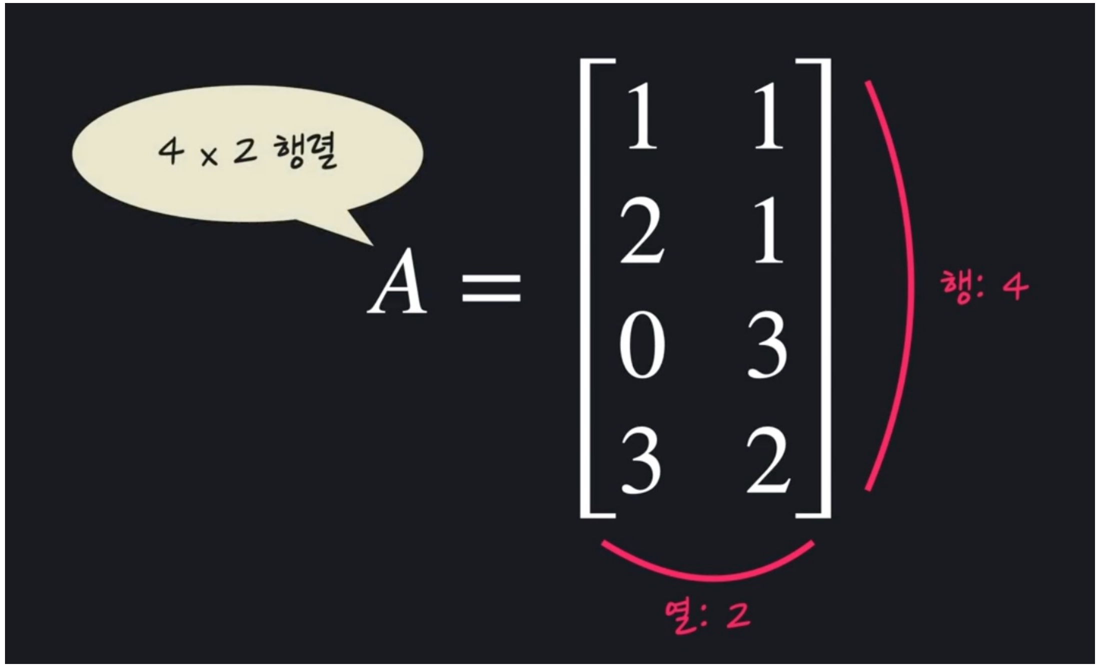
-> 앞으로 많이 다루게 될 형태

# 행렬

# 행렬(Matrix)

- 행렬 A의 원소 12개
  - 3행(row), 4열(column)




-> 3x4 행렬

-> 4x2 행렬

# 행렬의 원소 표현

- 행렬 A의 i행, j열에 위치한 원소를 표현하려면 : **$A_{ij}$**

$A_{ij}$ i 번째 행, j 번째 열에 있는 원소

$$A = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 2 & 1 & 4 & 1 \\ 0 & 3 & 2 & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 2 \\ \boxed{2} & 1 & 4 & 1 \\ 0 & 3 & 2 & 1 \end{bmatrix}$$

$A_{21}$

$$A = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 2 & 1 & \boxed{4} & 1 \\ 0 & 3 & 2 & 1 \end{bmatrix}$$

$A_{23}$

# 실습 - 넘파이로 행렬 사용하기

1. 4 x 3 행렬 $A = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 2 & 3 \\ 2 & 1 & 0 \\ -1 & 2 & -4 \end{bmatrix}$ 를 numpy array로 정의해보세요

2. 3 x 2 행렬 $B = \begin{bmatrix} 0 & 2 \\ 1 & 1 \\ -1 & -2 \end{bmatrix}$ 를 numpy array로 정의해보세요

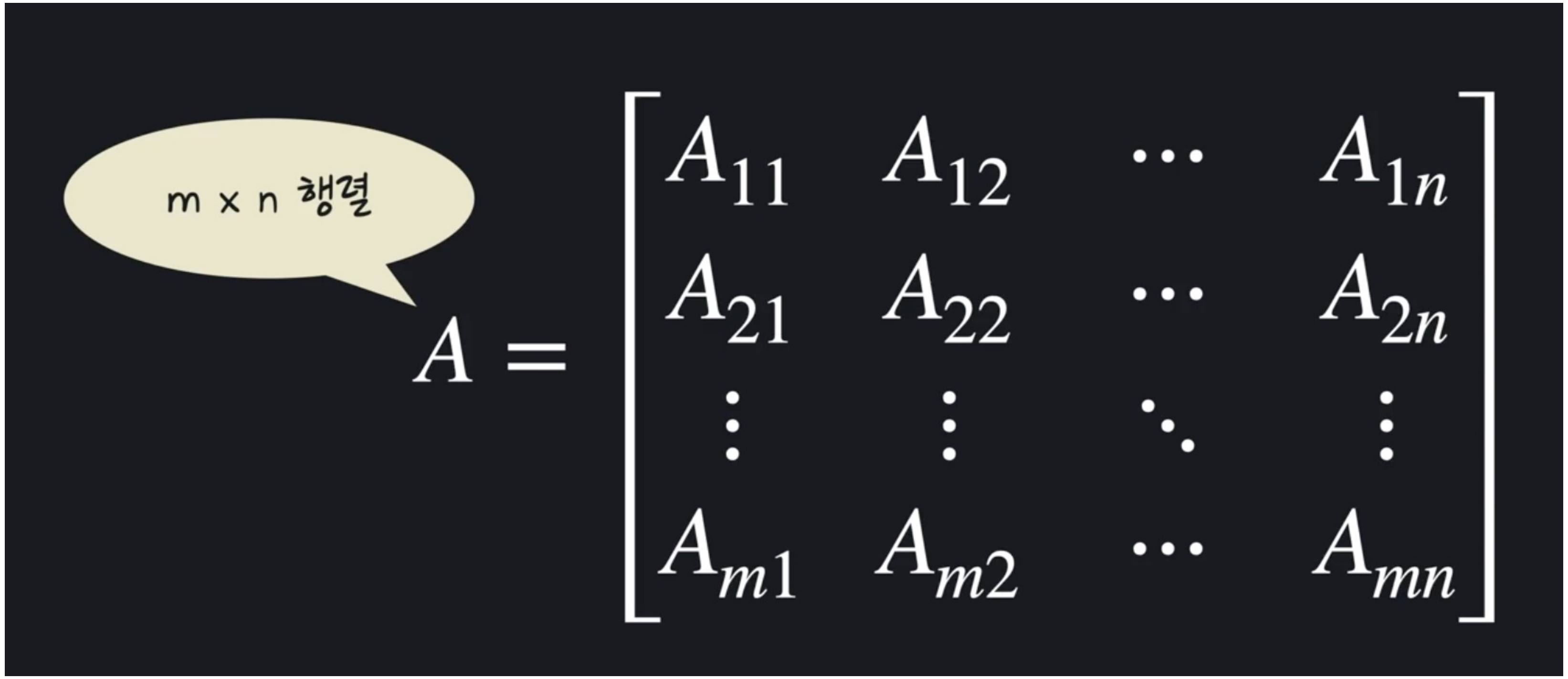
3. 정의한 행렬 $A$ 의 2 행 2 열 원소에 접근해보세요.

4. 정의한 행렬 $A$ 의 4 행 1열 원소에 접근해보세요.

**(프로그래밍할 때는 인덱스를 0부터 세는 걸 잊지마세요!!)**

# 행렬의 원소 표현

- m x n 행렬의 원소 표현

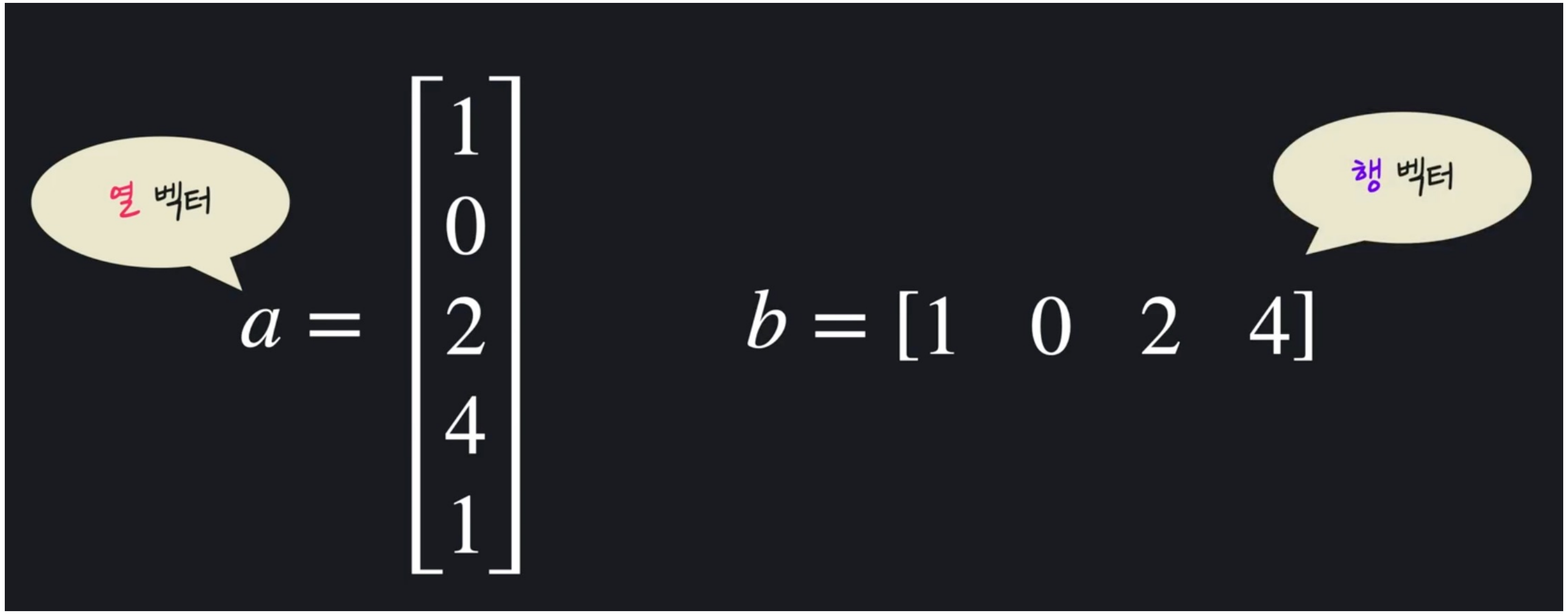


# 벡터

# 벡터(Vector)

● 벡터 : 행이 하나거나 열이 하나인 행렬

- 보통 벡터라 하면 열벡터를 의미함




● 차원

- a는 5차원 열 벡터, b는 4차원 행 벡터

# 벡터(Vector)

- m차원 벡터

m 차원 벡터

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}$$

- 보통 행렬은 대문자, 벡터는 소문자로 표현

$$\underline{A} = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 2 & 1 & 4 & 1 \\ 0 & 3 & 2 & 1 \end{bmatrix}$$

행렬은 대문자

$$\underline{a} = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 4 \\ 1 \end{bmatrix}$$

벡터는 소문자

# 행렬 덧셈과 곱셈

# 행렬의 덧셈

- 같은 위치에 있는 원소들을 더해준다.

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$

두 행렬은 차원이 같아야 한다!

$$\underline{A + B} = \begin{bmatrix} 4 & 2 \\ 4 & 7 \end{bmatrix}$$

# 행렬의 스칼라곱

- 스칼라곱

i는 스칼라

$$i = 5 \qquad A = \begin{bmatrix} 3 & 1 \\ 2 & 3 \end{bmatrix}$$

$$iA = \begin{bmatrix} 5 \times 3 & 5 \times 1 \\ 5 \times 2 & 5 \times 3 \end{bmatrix} = \begin{bmatrix} 15 & 5 \\ 10 & 15 \end{bmatrix}$$

# 두 행렬의 곱 – 내적곱, 외적곱

- 우리는 내적곱만 배울 예정

내적곱

외적곱

# 두 행렬의 내적곱

3x2 2x3

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

Q. 행렬 A, B의 내적곱을 해보세요

# 두 행렬의 내적곱

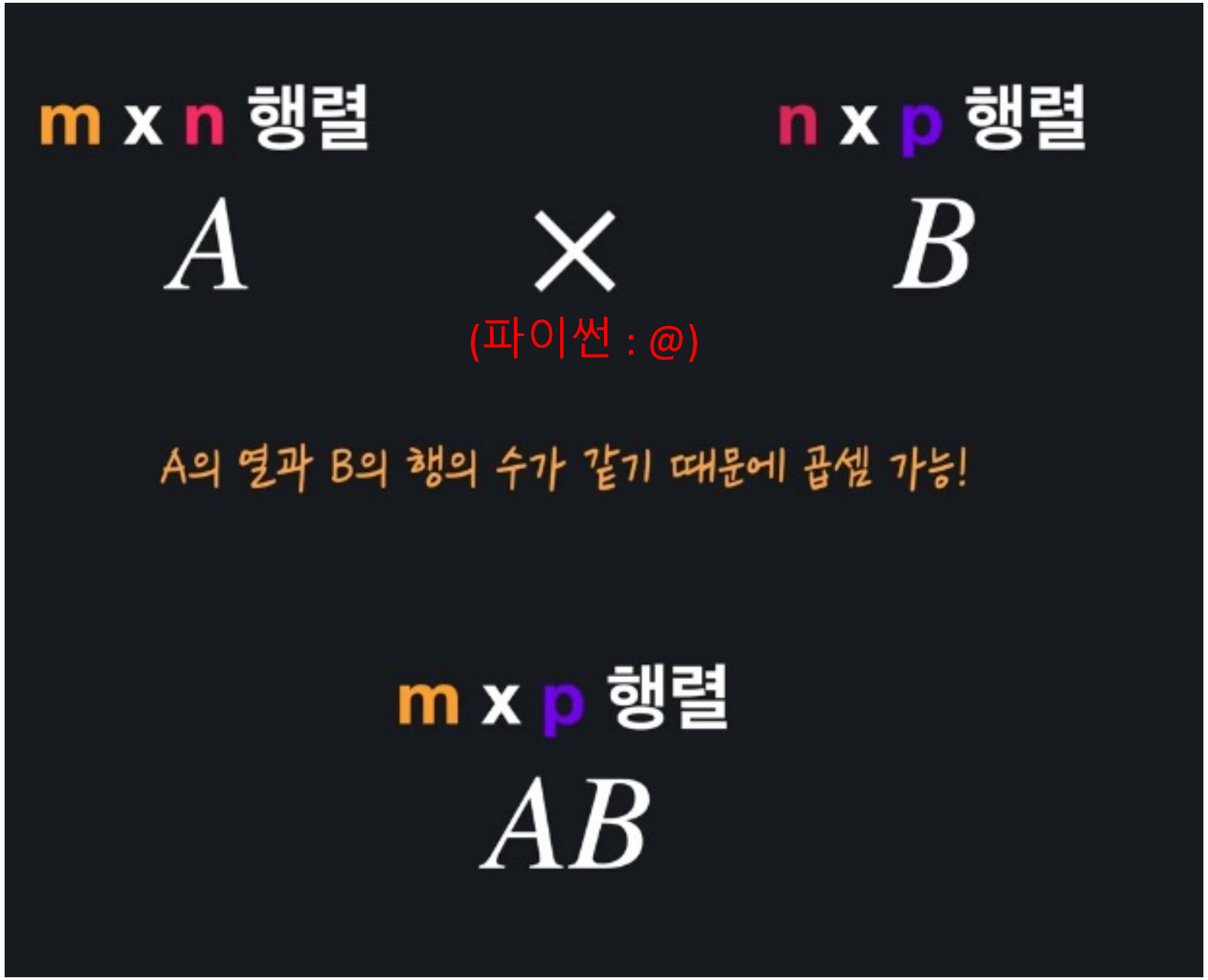
3x2　　　　2x3

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

2 x 2 행렬

$$AB = \begin{bmatrix} 20 & 13 \\ 21 & 17 \end{bmatrix}$$

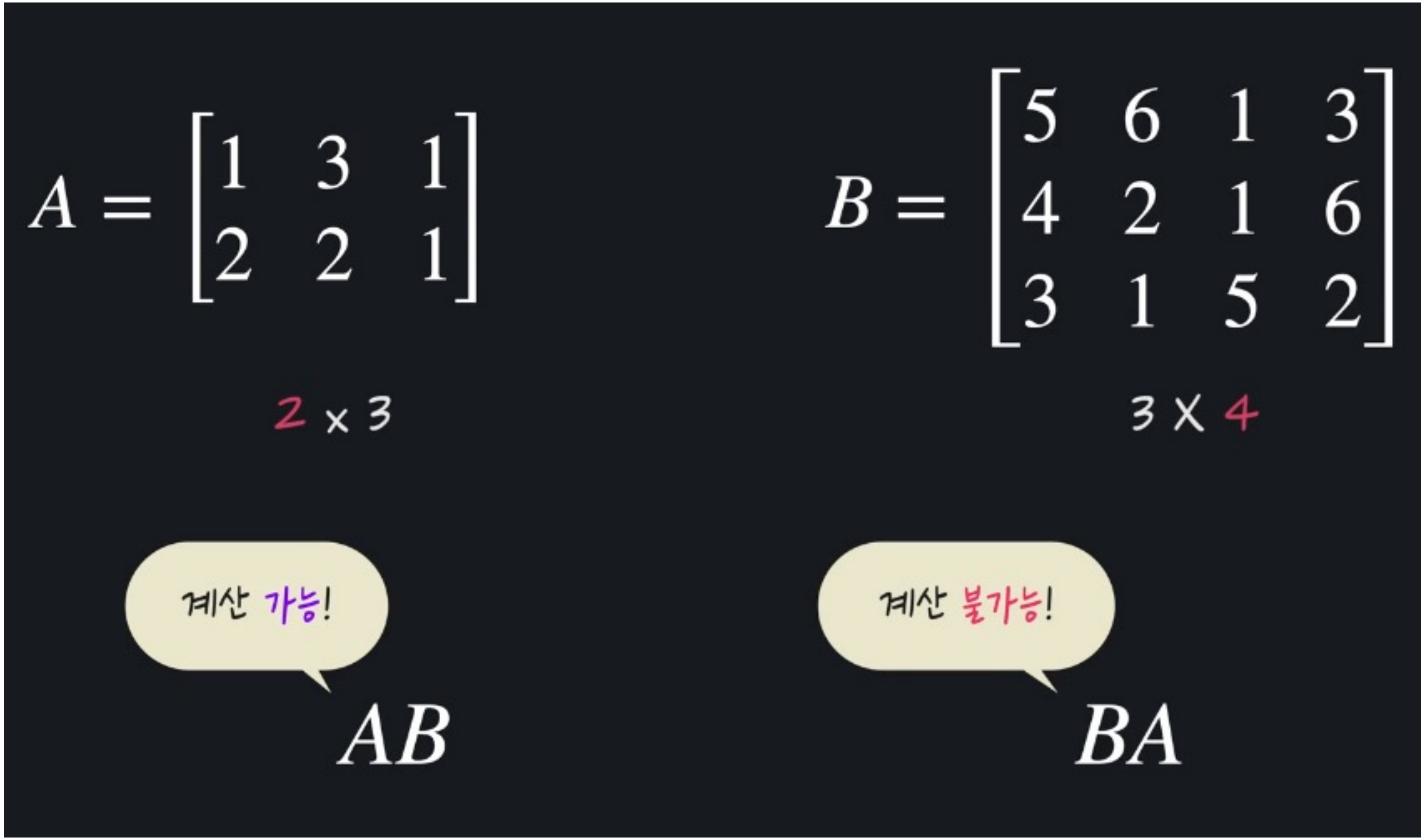
# 두 행렬의 내적곱


m x n 행렬
A
×
n x p 행렬
B
A의 열과 B의 행의 수가 같기 때문에 곱셈 가능!
m x p 행렬
AB

(파이썬 : @)


$AB \neq BA$
순서와 상관 있음

# 두 행렬의 내적곱 - 예시


$A = \begin{bmatrix} 1 & 3 & 1 \\ 2 & 2 & 1 \end{bmatrix}$
$B = \begin{bmatrix} 5 & 6 & 1 & 3 \\ 4 & 2 & 1 & 6 \\ 3 & 1 & 5 & 2 \end{bmatrix}$
2 x 3
3 X 4
계산 가능!
AB
계산 불가능!
BA

# 실습 - 넘파이로 행렬 사용하기

이번 과제에서는 numpy를 사용해서 행렬 연산을 해보겠습니다. 지금 템플릿에는 행렬 4개, 각각 $A, B, C, D$ 의 행렬이 정의되어 있는데요. 이 행렬들을 이용해서 다음과 아래 행렬 연산을 해보세요.

$$2A \times -B \times (3C + D)$$

행렬곱(=내적) : @

```python
import numpy as np

A = np.array([
    [1, -1, 2],
    [3, 2, 2]
])

B = np.array([
    [0, 1],
    [-1, 1],
    [5, 2]
])

C = np.array([
    [2, -1],
    [-3, 3]
])

D = np.array([
    [-5, 1],
    [2, 0]
])
```

# 행렬 연산 퀴즈

질문 1

$$\begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & -3 \end{bmatrix} + \begin{bmatrix} 3 & 2 & -2 \\ -1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$ 다음 식에서 $a_{13}$ 에 들어갈 값을 쓰세요.

질문 2

$$\begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & -3 \end{bmatrix} \times \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix}$$ 다음 식에서 $a_{11}$ 에 들어갈 값을 고르세요.

질문 3

$$\begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & -3 \end{bmatrix} \times \begin{bmatrix} 3 & 2 \\ 1 & 1 \\ 2 & -1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$ 다음 식에서 $a_{12}$ 에 들어갈 값을 쓰세요.

질문 4

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & -3 \\ 3 & 3 & 1 \end{bmatrix}, B = \begin{bmatrix} 3 & 2 \\ 1 & 1 \\ 2 & -1 \end{bmatrix}$$ 이 있을 때, 행렬 연산 $AB$의 결과 행렬의 차원을 고르세요.

# 두 행렬 요소 별 곱하기 : o

바로 요소별 곱하기, 영어로는 Element-wise Multiplication이라고 부르는 연산인데요. 행렬 덧셈 연산과 거의 똑같은 성질을 갖는 연산입니다. 바로 예시를 볼게요.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$$

이렇게 두 행렬이 있다고 할게요. 이걸 요소별 곱하기를 한다는 표기는 동그라미(○)를 써서 이렇게 표현합니다.

$$A \circ B$$

요소별 곱하기는 이름 그대로 같은 행과 열에 있는 요소들끼리 곱해서 새로운 행렬을 만드는 연산입니다. $A$의 행과 열에 있는 요소들과 $B$의 행과 열에 있는 요소들을 서로 곱해서 새로운 행렬을 만드는 거죠. 실제로 계산을 하면 이렇게 되겠죠?

$$A \circ B = \begin{bmatrix} -1 & 4 \\ 9 & 4 \end{bmatrix}$$

# 두 행렬 요소 별 곱하기 : o – 파이썬 넘파이로 구현

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

B = np.array([
    [0, 1, 2],
    [2, 0, 1],
    [1, 2, 0]
])
```

두 행렬 사이에 `*` 연산자를 쓰면 됩니다.

```
A * B
```

결과를 확인해보면 이렇게 나오는데요.

```
arrray([0, 2, 6]
       [8, 0, 6]
       [7, 16, 0])
```

# 실습 – 넘파이로 행렬 연산하기

jupyter numpy로 행렬 연산하기 Last Checkpoint: 2 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

```
B = np.random.rand(3, 3)

In [3]: A

Out[3]: array([[ 1, -1,  2],
               [ 3,  2,  2],
               [ 4,  1,  2]])

In [4]: B

Out[4]: array([[0.81036836, 0.48035945, 0.03060491],
               [0.6674185 , 0.76455226, 0.51256986],
               [0.30019928, 0.68420637, 0.4390987 ]])

In [5]: A + B

Out[5]: array([[ 1.81036836, -0.51964055,  2.03060491],
               [ 3.6674185 ,  2.76455226,  2.51256986],
               [ 4.30019928,  1.68420637,  2.4390987 ]])
```

```
In [6]: 5 * A

Out[6]: array([[ 5, -5, 10],
               [15, 10, 10],
               [20,  5, 10]])

In [7]: np.dot(A, B)

Out[7]: array([[0.74334842, 1.08421993, 0.39623245],
               [4.36634064, 4.33859559, 1.99515187],
               [4.5092905 , 4.05440278, 1.51318692]])

In [8]: A @ B

Out[8]: array([[0.74334842, 1.08421993, 0.39623245],
               [4.36634064, 4.33859559, 1.99515187],
               [4.5092905 , 4.05440278, 1.51318692]])
```

# 특수 행렬 :
# 전치 행렬, 단위 행렬, 역행렬

# 전치 행렬(Transposed Matrix)

- 행과 열을 서로 바꾸어주는 것

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 2 & 2 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 3 \\ 2 & 2 \\ 1 & 2 \end{bmatrix}$$

- 행렬 연산 시 행 수와 열 수를 맞춰주기 위해 사용

# 단위 행렬(identity matrix)

- 대각 행렬 값이 모두 1인 행렬

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3x3 단위 행렬

- 어떤 행렬 A에 단위 행렬을 곱하면 그대로 A가 된다.

$$A = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} \qquad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$AI = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}$$

# 역행렬(Inverse Matrix)

- 행렬 자기 자신과 곱했을 때 단위 행렬이 나오는 행렬

$$A = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 1 & -2 \\ -\frac{1}{2} & \frac{3}{2} \end{bmatrix}$$

# 실습1 – 넘파이로 전치, 단위, 역행렬 사용하기

jupyter numpy로 전치, 단위, 역행렬 사용하기 Last Checkpoint: 2 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run Code

```
In [1]: import numpy as np
```

```
In [2]: A = np.array([
            [1, -1, 2],
            [3, 2, 2],
            [4, 1, 2]
        ])
```

```
In [3]: A

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

```
In [4]: A_tranpose = A.T
        A_tranpose

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

# 선형대수학을 사용하게 되면

- 복잡한 선형 시스템을 행렬과 벡터로 쉽게 표현 가능

선형 시스템

$$2x_0 - 4x_1 + x_2 = 3$$

$$3x_0 + x_1 - 6x_2 = 10$$

$$x_0 + x_1 + x_2 = 5$$

$$\boxed{Ax = y}$$

$$\begin{bmatrix} 2 & -4 & 1 \\ 3 & 1 & -6 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 5 \end{bmatrix}$$

아무리 복잡한 선형 시스템도
행렬과 벡터로 쉽게 표현할 수 있음!

$$\begin{bmatrix} 2x_0 - 4x_1 + x_2 \\ 3x_0 + x_1 - 6x_2 \\ x_0 + x_1 + x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 5 \end{bmatrix}$$

# 선형대수학이 머신러닝에 필요한 이유

# 예시 - 아파트 가격 예측

- 복잡한 수식을 행렬과 벡터로 요약해서 표현할 수 있다.

집 값 = 크기 $\times a_1 +$ 지하철 역 거리 $\times a_2 +$ 층수 $\times a_3$

일차식

첫 번째 아파트 가격: $110 \times a_1 + 400 \times a_2 + 20 \times a_3$

두 번째 아파트 가격: $100 \times a_1 + 1000 \times a_2 + 5 \times a_3$

세 번째 아파트 가격: $180 \times a_1 + 10 \times a_2 + 30 \times a_3$

네 번째 아파트 가격: $50 \times a_1 + 300 \times a_2 + 5 \times a_3$

$\vdots \qquad \vdots$

$$X = \begin{bmatrix} 110 & 400 & 20 \\ 100 & 1000 & 5 \\ 180 & 10 & 30 \\ 50 & 300 & 5 \\ & \vdots & \end{bmatrix} \qquad a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

깔끔하게 요약

모든 집 값 $= Xa$

# 선형대수학과 머신러닝 요약

- 선형대수학은 일차식, 일차 함수, 행렬, 벡터를 다루는 학문이다.
- 머신러닝을 할 때 데이터를 일차식으로 표현하는 경우가 많다.
- 행렬과 벡터를 이용하면 정돈된 형태로 효율적 계산이 가능하다.