

# 01\_Numpy 기초

# Numpy 기초

- Numpy 란?



*NumPy*

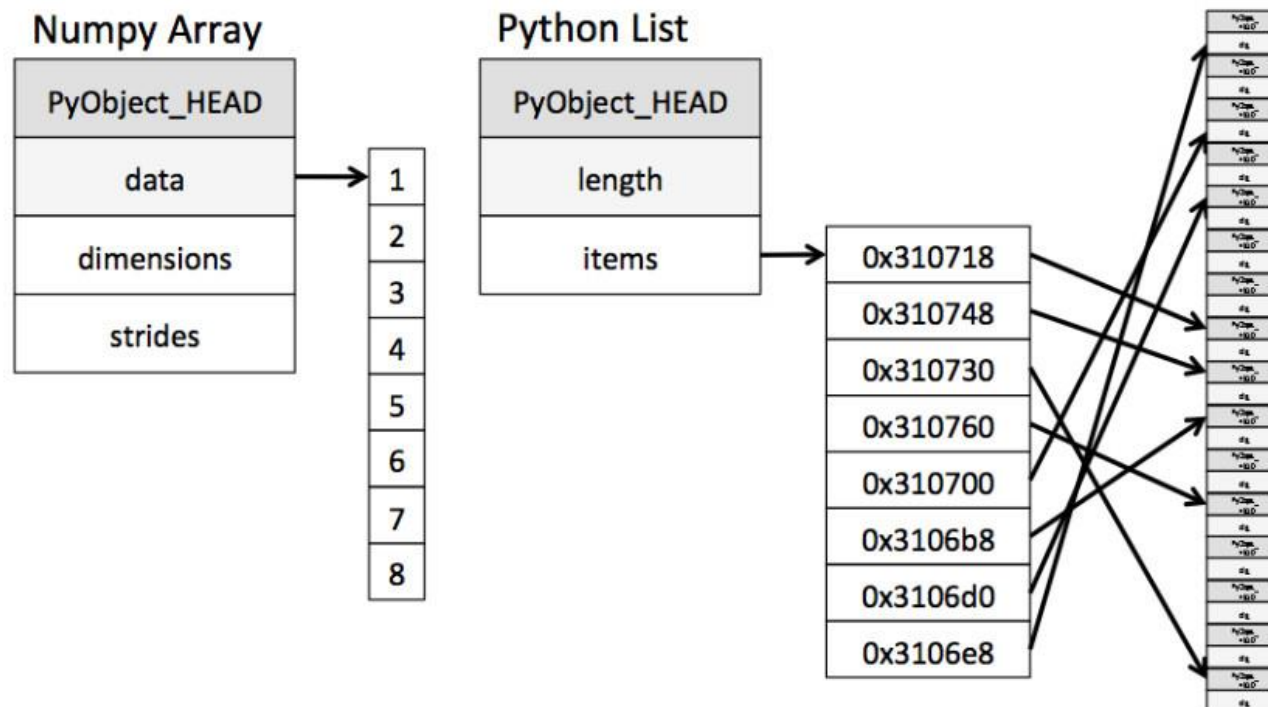
- Numerical Python
- 과학 계산 분야를 위한 라이브러리로서, 고성능의 다차원 배열 객체와 연산에 필요한 여러 유용한 기능을 제공
- 기본적으로 C언어의 Array(배열) 구조를 사용
- 별도의 Loop를 사용하지 않고 빠른 연산을 제공

# Numpy 기초

## • Numpy Array와 Python List 차이

- Array 구조를 사용하여 연속된 메모리에 저장
- List는 각 객체의 주소 값을 저장하여 메모리에서 검색

03



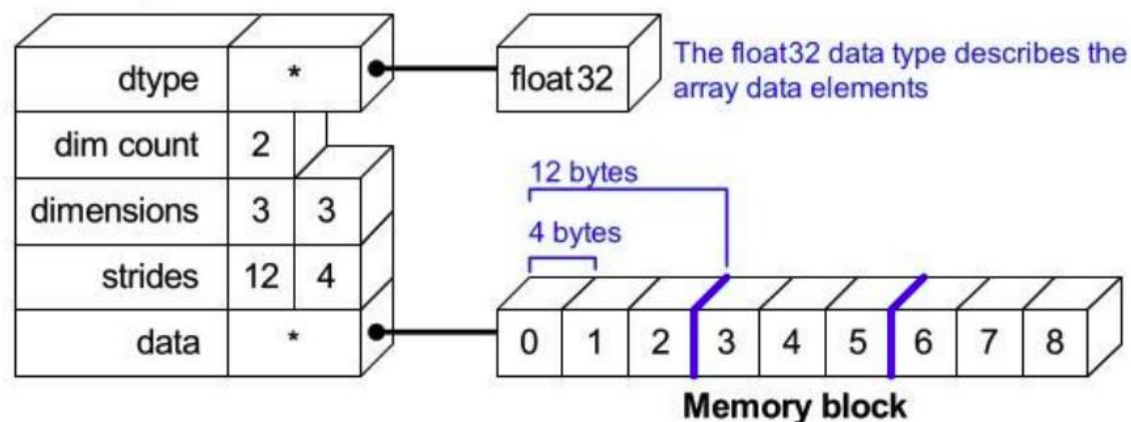
# Numpy 기초

## • Numpy Array와 Python List 차이

- 파이썬의 기본 List에 비해 빠르고, 메모리를 효율적으로 사용
- List와는 다르게 모든 원소가 같은 자료형이어야 한다.
- 원소의 개수를 바꿀 수 없다.

04

NDArray Data Structure



## NumPy dtypes

Basic Type	Available NumPy types	Comments
Boolean	<code>bool</code>	Elements are 1 byte in size.
Integer	<code>int8, int16, int32, int64, int128, int</code>	<code>int</code> defaults to the size of <code>int</code> in C for the platform.
Unsigned Integer	<code>uint8, uint16, uint32, uint64, uint128, uint</code>	<code>uint</code> defaults to the size of unsigned <code>int</code> in C for the platform.
Float	<code>float32, float64, float, longfloat,</code>	<code>float</code> is always a double precision floating point value (64 bits). <code>longfloat</code> represents large precision floats. Its size is platform dependent.
Complex	<code>complex64, complex128, complex, longcomplex</code>	The real and complex elements of a <code>complex64</code> are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	<code>str, unicode</code>	
Object	<code>Object</code>	Represent items in array as Python objects.
Records	<code>Void</code>	Used for arbitrary data structures.

# Numpy 기초

dtype 접두사	설명	사용 예
b	불리언	b (True 또는 False)
i	정수	i8 (64비트)
u	부호없는 정수	u8 (64비트)
f	부동 소수점	f8 (64비트)
c	복소 부동소수점	c16 (128비트)
O	객체	O (객체에 대한 포인터)
S	바이트 문자열	S24 (24 글자)
U	유니코드 문자열	U24 (24 유니코드 글자)

# Numpy 기초

- Narray(N-Dimensional Array)

- N차원의 배열
- Numpy에서 사용하는 배열 구조의 데이터 타입

1D array

1	3	5	7
---	---	---	---

2D array

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

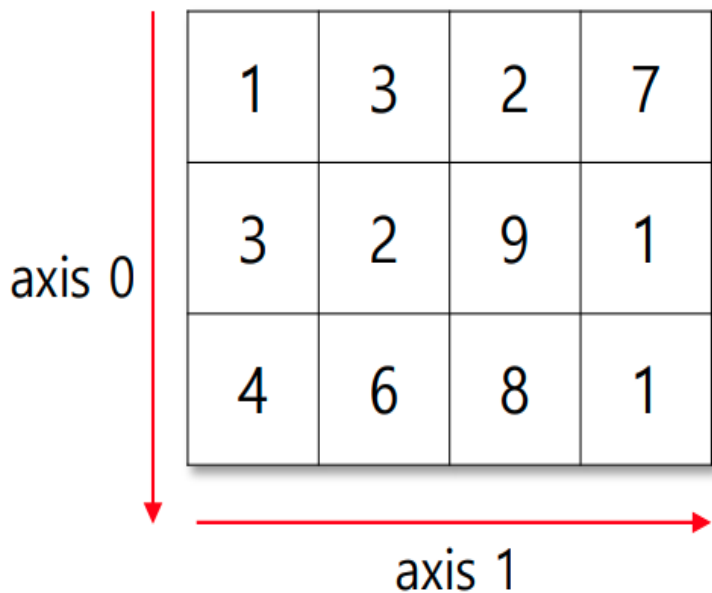
3D array

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

# Numpy 기초

## • 용어 정리

- Axis : 배열의 각 축(기본적으로 0은 수직, 1은 수평을 뜻함)
- Rank : 축의 개수(차원의 수)
- Shape : 축의 길이



1	3	2	7
3	2	9	1
4	6	8	1

<- [3X4] 형태의 배열인 경우

- Axis 0과 Axis 1을 가지는 Rank 2의 배열
- 행의 길이는 3, 열의 길이는 4로 Shape는 (3,4)



# Numpy 기초

- 모듈 호출

- Numpy 배열을 사용하려면 우선 numpy 라이브러리를 불러와야 합니다.
- numpy 라이브러리는 전세계적으로 np 별칭을 붙여 불러옵니다.

- > numpy 모듈 호출

```
import numpy as np
```

- > 정상적으로 호출이 됐는지 확인

```
a = np.array(1)  
a
```

Out : array(1)

# Numpy 기초

- 배열 생성

> 배열 생성

```
ndarr_1 = np.array([1, 2, 3])  
print(ndarr_1)
```

**Out :** [1 2 3]

```
ndarr_2 = np.array([1, 2.0, 3])  
print(ndarr_2)
```

**Out :** [1. 2. 3.]

```
ndarr_3 = np.array([1, 2.0, '3'])  
print(ndarr_3)
```

**Out :** [1. 2. 3.]

010

|

# Numpy 기초

- 배열의 정보 확인

- > 자료형 확인 속성

- `ndarray.dtype`

```
print(ndarr_1.dtype)
print(ndarr_2.dtype)
print(ndarr_3.dtype)
```

**Out :** int32  
float64  
<U32

```
print(np.array(1 + 2j).dtype)
print(np.array(print).dtype)
```

**Out :** complex128  
object

# Numpy 기초

- 자료형 지정

> dtype 속성을 사용하여 자료형 지정

```
dtype1 = np.array([1, 2, 3], dtype=np.int8)  
print(dtype1.dtype)
```

**Out :** int8

```
dtype2 = np.array([1, 2, 3], dtype=float)  
print(dtype2.dtype)
```

**Out :** float64

```
dtype3 = np.array([1, 2, 3], dtype=str)  
print(dtype3.dtype)
```

**Out :** <U1

# Numpy 기초

- 자료형 지정

- > 주의해야할 점

```
dtype4 = np.array([1, 2.1, '3'], dtype=int)
print(dtype4)
print(dtype4.dtype)
```

**Out :** [1 2 3]  
int32

```
dtype5 = np.array([1, 'data', print], dtype=float)
print(dtype5.dtype)
```

---

```
ValueError                                Traceback (most recent call last)
<ipython-input-344-441caa0e3f58> in <module>
----> 1 dtype5 = np.array([1, 'data', print], dtype=float)
      2 print(dtype5.dtype)

ValueError: could not convert string to float: 'data'
```

# Numpy 기초

- 배열의 정보 확인

- > 배열 크기 확인

- `ndarray.shape`

```
print(ndarr_1.shape)
```

**Out :** (3, )

- > 배열 차원 확인

- `ndarray.ndim`

```
print(ndarr_1.ndim)
```

**Out :** 1

# Numpy 기초

- N차원 배열

015

Scalar   Vector   Matrix   Tensor

1

$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$

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

$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$

- Scalar : 숫자
- Vector : scalar의 배열, 1차원 배열
- Matrix : vector의 배열, 2차원 배열
- Tensor : matrix의 배열, 3차원 배열이상

# Numpy 기초

- 배열 생성

- > 1차원 배열 생성

```
vector = np.array([1, 2, 3])  
print(vector)  
print(vector.shape)  
print(vector.ndim)  
print(vector.dtype)
```

**Out :** [1 2 3]

(3, )

1

int32



# Numpy 기초

- 배열 생성

- > 2차원 배열 생성

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])  
print(matrix)  
print(matrix.shape)  
print(matrix.ndim)  
print(matrix.dtype)
```

```
Out: [[1 2 3]  
      [4 5 6]]  
(2, 3)  
2  
int32
```

# Numpy 기초

- 배열 생성

- > 3차원 배열 생성

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

```
Out : [[[1 2 3]  
        [4 5 6]]  
  
        [[7 8 9]  
        [10 11 12]]]  
(2, 2, 3)  
3  
int32
```

# Numpy 기초

- 배열의 정보 확인

- > 배열 요소의 개수 확인

- `ndarray.size`

```
print(vector.size)
print(matrix.size)
print(tensor.size)
```

Out : 3

6

12

# Numpy 기초

## 연습문제

- 배열 생성

1	2	3
2	4	6
3	6	9

> [문제1] 위와 같은 배열을 생성하고 차원, 크기, 모양을 알아내시오.

# Numpy 기초

## 연습문제

- 배열 생성

1	2	3
2	4	6
3	6	9

> [문제1] 위와 같은 배열을 생성하고 차원, 크기, 모양을 알아내시오.

```
arr = np.array([[[1,2,3]],[[2,4,6]],[[3,6,9]]])  
print(arr.ndim)  
print(arr.size)  
print(arr.shape)
```

# Numpy 기초

- 자료형 변환(리스트)

> 리스트 -> ndarray 변환

```
list1 = [1,2,3]
arr1 = np.array(list1)
print(arr1)
```

Out : [1 2 3]

> ndarray -> 리스트 변환

```
arr2 = np.array([1,2,3])
list2 = list(arr2)
print(list2)
```

Out : [1, 2, 3]

# Numpy 기초

- 자료형 변환(리스트)

- > 2차원 리스트 변환

```
list3 = [[1,2,3], [4,5,6]]  
arr3 = np.array(list3)  
print(arr3)
```

**Out :**   
[[1 2 3]  
 [4 5 6]]

- > shape이 다른 리스트인 경우

```
list4 = [[1,2,3,4], [5,6,7]]  
arr4 = np.array(list4)  
print(arr4)
```

**Out :** [list([1, 2, 3, 4]) list([5, 6, 7])]

# Numpy 기초

- 자료형 변환(튜플) - 리스트와 동일

> 튜플 -> ndarray 변환

```
tuple1 = (1,2,3)
arr = np.array(tuple1)
print(arr)
```

Out : [1 2 3]

> ndarray -> 튜플 변환

```
arr2 = np.array([1,2,3])
tuple2 = tuple(arr2)
print(tuple2)
```

Out : (1, 2, 3)



# Numpy 기초

- 자료형 변환(세트, 딕셔너리)

> 세트 변환

```
set1 = {1,2,3}  
arr = np.array(set1)  
print(arr)
```

Out : {1, 2, 3}

```
print(arr.size)
```

Out : 1

# Numpy 기초

- 자료형 변환(세트, 딕셔너리)

> 딕셔너리 변환

```
dict1 = {'a':1,'b':2,'c':3}  
arr = np.array(dict1)  
print(arr)
```

**Out** : {'a': 1, 'b': 2, 'c': 3}

```
print(arr.size)
```

**Out** : 1

- 배열 인덱싱

- > 리스트 인덱싱과 유사

- ndarray[col]

- ndarray[row][col]

- ndarray[rnk][row][col]

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

```
# 1 가져오기
```

```
print(list_[0][0][0])
```

```
# 4 가져오기
```

```
print(list_[0][1][1])
```

```
# 7 가져오기
```

```
print(list_[1][1][0])
```

# Numpy 기초

- 배열 인덱싱

> 리스트 인덱싱과 유사

```
arr_ = np.array([ [ [1, 2], [3, 4] ], [ [5, 6], [7, 8] ] ])  
# 1 가져오기  
print(arr_[0][0][0])  
# 4 가져오기  
print(arr_[0][1][1])  
# 7 가져오기  
print(arr_[1][1][0])
```

# Numpy 기초

- 배열 인덱싱

- > ndarray의 인덱싱

- array[row, col]

- array[rank, row, col]

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

```
# 1 가져오기
```

```
print(arr_[0,0,0])
```

```
# 4 가져오기
```

```
print(arr_[0,1,1])
```

```
# 7 가져오기
```

```
print(arr_[1,1,0])
```

# Numpy 기초

- 배열 인덱싱

030

> ndarray의 인덱싱

```
arr_1 = np.array([[1, 2],      # 2 x 3 형태의 2차원 배열
                  [3, 4],
                  [5, 6]])

# 0 행 가져와 2차원 배열 만들기
print(arr_1[[0]])

# 0, 2 행 가져와 2차원 배열 만들기
print(arr_1[[0, 2]])

# 다음 결과는 ?
print(arr_1[[0, 1, 2], [0, 1, 0]])

# 같은 위치 데이터 여러 번 사용가능
print(arr_1[[0, 0], [1, 1]])
```

# Numpy 기초

## • 배열 인덱싱

031

> ndarray의 인덱싱

```
arr_2 = np.array([[ 1, 2, 3],  
                  [ 4, 5, 6],  
                  [ 7, 8, 9],  
                  [10, 11, 12]])
```

# 두 개의 1차원 배열 만들기

```
b = np.array([0, 1, 2, 3])
```

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

# 다음 코드의 실행 결과는?

```
print(arr_2[b, c])
```

# 4x3 형태의 2차원 배열

배열 a

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


배열 b

0	1	2	3
---	---	---	---

배열 c

0	2	0	1
---	---	---	---

배열 a



11	2	3
4	5	16
17	8	9
10	21	12

# Numpy 기초

- 배열 인덱싱

032

> Boolean 인덱싱

```
# 기본 데이터
```

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

```
# True, False
```

```
print(arr[[True, False, True, True, False]])
```

**Out :** [1 3 4]

```
# 조건
```

```
print(arr[arr >= 3])
```

**Out :** [3 4 5]



# Numpy 기초

## 연습문제

- 배열 인덱싱

> [문제2]

```
score = np.array([[78, 91, 84, 89, 93, 65],  
                  [82, 87, 96, 79, 91, 73]])
```

위와 같은 2x6 형태의 2차원 배열을 생성하고  
각각 0행과 1행에서 90이상인 숫자를 추출해봅시다.

# Numpy 기초

## 연습문제

- 배열 인덱싱

> [문제2]

```
score = np.array([[78, 91, 84, 89, 93, 65],  
                  [82, 87, 96, 79, 91, 73]])
```

위와 같은 2x6 형태의 2차원 배열을 생성하고  
각각 0행과 1행에서 90이상인 숫자를 추출해봅시다.

```
score = np.array([[78, 91, 84, 89, 93, 65],  
                  [82, 87, 96, 79, 91, 73]])  
  
# 0행에서 90이상인 숫자 가져오기  
print(score[0][score[0] >= 90])  
print(score[1][score[1] >= 90])
```

- 배열 슬라이싱

- > ndarray의 슬라이싱

- 행과 열 부분을 나눠서 슬라이싱 가능
    - 다차원 배열의 부분 데이터를 추출할 때 유용

```
# 기본 데이터 4x5 형태의 2차원 배열
matrix = np.array([ [ 1, 2, 3, 4, 5 ],
                    [ 6, 7, 8, 9, 10],
                    [11, 12, 13, 14, 15],
                    [16, 17, 18, 19, 20] ])
```

# Numpy 기초

- 배열 슬라이싱

036

> ndarray의 슬라이싱

# 0열 전체 슬라이싱

```
print(matrix[:,0])
```

Out: [ 1 6 11 16]

[	1	2	3	4	5]
[	6	7	8	9	10]
[	11	12	13	14	15]
[	16	17	18	19	20]

# 0열 전체 슬라이싱 (2차원 유지)

```
print(matrix[:, :1])
```

Out: [[ 1]

[ 6]

[11]

[16]]

# Numpy 기초

- 배열 슬라이싱

037

> ndarray의 슬라이싱

# 1행 전체 슬라이싱

print(matrix[1,:])

Out : [ 6 7 8 9 10]

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
```

# 1행 전체 슬라이싱

print(matrix[1]) # rank > row > col 순으로 :(컬론) 생략 가능

Out : [ 6 7 8 9 10]

# 1행 전체 슬라이싱 (2차원 유지)

print(matrix[1:2,:]) # or print(matrix[[1]])

Out : [ 6 7 8 9 10]

# Numpy 기초

- 배열 슬라이싱

038

> ndarray의 슬라이싱

# 3~4열 전체 슬라이싱

```
print(matrix[:,3:])
```

Out :   
[[ 4 5]  
[ 9 10]  
[14 15]  
[19 20]]

[[	1	2	3	4	5]
[	6	7	8	9	10]
[11	12	13	14	15]	
[16	17	18	19	20]]	

# 0행, 1~3열 슬라이싱

```
print(matrix[0,1:-1])
```

Out : [2 3 4]

[[	1	2	3	4	5]
[	6	7	8	9	10]
[11	12	13	14	15]	
[16	17	18	19	20]]	

# Numpy 기초

- 배열 슬라이싱

039

> ndarray의 슬라이싱

# 2~3행, 2~4열 슬라이싱

```
print(matrix[-2:,-3:])
```

Out: `[[13 14 15]`

`[18 19 20]]`

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
```

# 1~2행, 1~2열 슬라이싱

```
print(matrix[1:-1,1:3])
```

Out: `[[ 7 8]`

`[12 13]]`

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
```

# Numpy 기초

- 배열 인덱싱, 슬라이싱 복합

040

> ndarray의 슬라이싱

# 1, 3행, 2~4열 슬라이싱

```
print(matrix[[1,3],2:])
```

Out :   
[[ 8 9 10]  
[18 19 20]]

[[	1	2	3	4	5]
[	6	7	8	9	10]
[11	12	13	14	15]	
[16	17	18	19	20]]	

# 1~2행, 1, 3열 슬라이싱

```
print(matrix[1:-1,[1,3]])
```

Out :   
[[ 7 9]  
[12 14]]

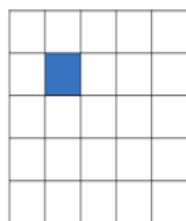
[[	1	2	3	4	5]
[	6	7	8	9	10]
[11	12	13	14	15]	
[16	17	18	19	20]]	



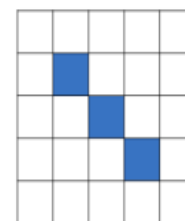
# Numpy 기초

- 배열 인덱싱, 슬라이싱 정리

041



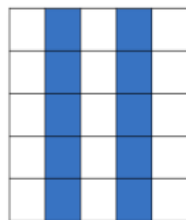
배열[행, 열] →



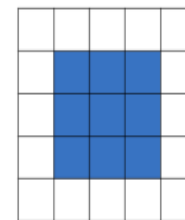
배열[[행1, 행2, ...], [열1, 열2, ...]] →



배열[[행1, 행2, ...], :] 또는 배열[[행1, 행2, ...]] →



배열[:, [열1, 열2, ...]] →

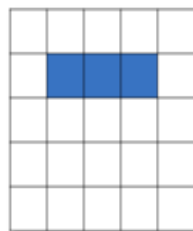



배열[행1:행N, 열1:열N] →

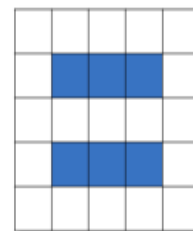
# Numpy 기초

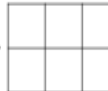
- 배열 인덱싱, 슬라이싱 정리

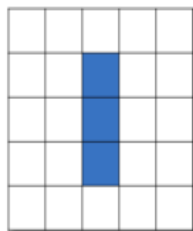
042




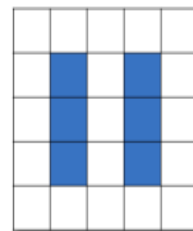
배열[행, 열1:열N] → 




배열[[행1, 행2,...], 열1:열N] → 



배열[행1:행N, 열] → 



배열[행1:행N, [열1, 열2,...]] → 

# Numpy 기초

## 연습문제

- 배열 부분 추출 예제

043

> [문제3]

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

위와 같은 2x2x3 형태의 3차원 배열을 생성하고 인덱싱을 사용해  
[10 2 6] 배열을 추출해봅시다.

> [문제4]

문제3의 tensor 배열에서 인덱싱, 슬라이싱을 사용해

```
[ [1 2]
```

```
[4 5] ] 배열과 [9 12] 배열을 추출해봅시다.
```

# Numpy 기초

## 연습문제

- 배열 부분 추출 예제

> [문제3]

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

위와 같은 2x2x3 형태의 3차원 배열을 생성하고 인덱싱을 사용해  
[10 2 6] 배열을 추출해봅시다.

```
tensor = np.array([ [ [1, 2, 3],  
                      [4, 5, 6]],  
                  [ [7, 8, 9],  
                    [10, 11, 12] ] ])  
  
print(tensor[[1,0,0],[1,0,1],[0,1,2]])
```

# Numpy 기초

## 연습문제

- 배열 부분 추출 예제

> [문제4]

문제3의 tensor 배열에서

[ [1 2]

[4 5] ] 배열과 [9 12] 배열을 추출해봅시다.

```
# [[1 2] [4 5]] 추출  
print(tensor[0, :, 2])
```

```
# [9 12] 추출  
print(tensor[1, :, 2])
```