

# Data Analysis with NumPy

NumPy 실습

Human-Computer Interaction Lab

Department of Computer Science and Engineering

Seoul National University

# 실습목표

- 간단한 실습 문제들을 풀면서 **Numpy**의 사용법에 익숙해져 봅시다.
- **Numpy**를 사용하면서 헛갈리기 쉬운 부분과 개발(**ML**, **DL**)을 진행하면서 자주 사용하는 기능을 위주로 실습을 진행해 봅시다.

# NumPy Exercise - 10 min

- NumPy의 `arange` 함수를 사용하여 3~11까지의 숫자를 담고 있는 1D array를 만들어 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np
```

```
nums = _____ # np.array를 사용하지 말고 내장함수를 사용해서 해결합니다.  
print(nums)
```

3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	----	----

# NumPy Exercise - 10 min

- NumPy의 `arange` 함수를 사용하여 3~11까지의 숫자를 담고 있는 1D array를 만들어 봅시다.

- 정답

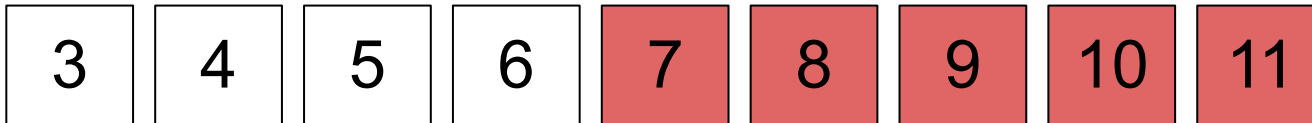
```
import numpy as np  
  
nums = np.arange(3,12,1)  
print(nums)
```

3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	----	----

# NumPy Exercise - 3 min

- Numpy에서는 `condition`을 사용하여 `array`에서 `element`를 선택할 수 있습니다.
- 아래의 코드를 실행해 봅시다.

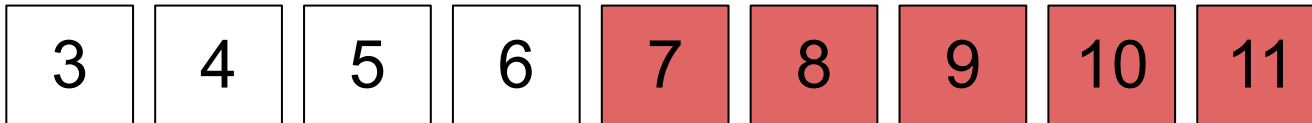
```
import numpy as np  
  
nums = np.arange(3,12,1)  
  
print(nums<7)
```



# NumPy Exercise

- Comparison Operator는 array의 모든 element에 적용되게 되고 원본 numpy array와 동일한 모양으로 boolean 결과값을 리턴하게 됩니다.

```
import numpy as np  
  
nums = np.arange(3,12,1)  
  
print(nums<7)  
# [True, True, True, True, False, False, False, False, False]
```



# NumPy Exercise

- 이러한 **boolean** 결과 **array**를 원본 **array**에 **pass**하게 되면 **array**는 해당 **condition**을 만족하는 **element**만을 리턴합니다.

```
import numpy as np

nums = np.arange(3,12,1)

print(nums<7)
# [True, True, True, True, False, False, False, False, False]

print(nums[nums<7])
# [3, 4, 5, 6]
```

3	4	5	6
---	---	---	---

# NumPy Exercise - 5 min

- 이번에는 이전에 만든 1D array에서 짝수값들을 추출해 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np  
  
nums = np.arange(3,12,1)  
  
print(_____)
```

4

6

8

10



# NumPy Exercise - 5 min

- 이번에는 이전에 만든 1D array에서 짝수값들을 추출해 봅시다.

- 정답

```
import numpy as np  
  
nums = np.arange(3,12,1)  
  
print(nums[nums%2 == 0])
```

4

6

8

10

# NumPy Exercise - 3 min

- 이번에는 이전에 만든 1D array의 짝수값을 음수로 만들어 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np  
  
nums = np.arange(3,12,1)  
nums[]  
  
print(nums)
```

3	-4	5	-6	7	-8	9	-10	11
---	----	---	----	---	----	---	-----	----

# NumPy Exercise - 3 min

- 이번에는 이전에 만든 1D array의 짝수값을 음수로 만들어 봅시다.

- 정답

```
import numpy as np

nums = np.arange(3,12,1)
nums[nums%2==0] *= -1

print(nums)
```

3	-4	5	-6	7	-8	9	-10	11
---	----	---	----	---	----	---	-----	----

# NumPy Exercise - 8 min

- **Conditional indexing**을 사용하여 **element**를 바꿀경우 원본 **array**의 값이 바뀌는 문제가 있습니다.
- 원본 **array**에 영향을 주지 않으면서 **Condition**을 통해 **element**를 선택하고 싶을때는 어떻게 해야 할까요?

```
import numpy as np

nums = np.arange(3,12,1)
nums[nums%2==0] *= -1

print(nums)
```

nums before	3	4	5	6	7	8	9	10	11
nums after	3	-4	5	-6	7	-8	9	-10	11

# NumPy Exercise

- `np.where(condition, Condition0| True일 경우, Condition0| False일 경우)`

`numpy.where(condition[, x, y])`

Return elements chosen from *x* or *y* depending on *condition*.

## Note

When only *condition* is provided, this function is a shorthand for `np.asarray(condition).nonzero()`. Using `nonzero` directly should be preferred, as it behaves correctly for subclasses. The rest of this documentation covers only the case where all three arguments are provided.

**Parameters:** `condition : array_like, bool`

Where True, yield *x*, otherwise yield *y*.

`x, y : array_like`

Values from which to choose. *x*, *y* and *condition* need to be broadcastable to some shape.

**Returns:** `out : ndarray`

An array with elements from *x* where *condition* is True, and elements from *y* elsewhere.

# NumPy Exercise

- **where** 함수를 사용하면 원본 **array**에 영향을 주지 않으면서 원하는 **element**만 선택, 변형할 수 있습니다.

- 정답

```
import numpy as np

nums = np.arange(3,12,1)
out = np.where(nums%2==0, nums*-1, nums)

print(nums)
print(out)
```

nums	3	4	5	6	7	8	9	10	11
out	3	-4	5	-6	7	-8	9	-10	11

# NumPy Exercise - 3 min

- 이번에는 이전에 만든 1D array를 3\*3 array로 변환해 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np  
  
nums = np.arange(3, 12, 1)  
print(nums._____)
```

3	4	5
6	7	8
9	10	11

# NumPy Exercise - 3 min

- 이전에 만든 1D array를 3\*3 array로 변환해 봅시다.

- 정답

```
import numpy as np  
  
nums = np.arange(3, 12, 1)  
print(nums.reshape(3,3))
```

3	4	5
6	7	8
9	10	11



# NumPy Exercise - 15 min

- 이번에는 **slicing**을 활용하여 이전에 만든 2d array 에서 아래의 범위를 선택해 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

nums = np.arange(3, 12, 1).reshape(3,3)
```

3	4	5
6	7	8
9	10	11

3	4	5
6	7	8
9	10	11

3	4	5
6	7	8
9	10	11

# NumPy Exercise - 15 min

- 이번에는 **slicing**과 **indexing**을 사용하여 1번째, 2번째 **column**과 **row**의 순서를 바꿔 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

nums = np.arange(3, 12, 1).reshape(3,3)
```

3	4	5
6	7	8
9	10	11

4	3	5
7	6	8
10	9	11

6	7	8
3	4	5
9	10	11

# NumPy Exercise - 15 min

- 이번에는 **slicing**과 **indexing**을 사용하여 1번째, 2번째 **column**과 **row**의 순서를 바꿔 봅시다.
- 정답

```
import numpy as np

nums = np.arange(3, 12, 1).reshape(3,3)

print(nums[:, [1,0,2]]) # Switching Columns
print(nums[[1,0,2], :]) # Switching Rows
```

3	4	5
6	7	8
9	10	11

4	3	5
7	6	8
10	9	11

6	7	8
3	4	5
9	10	11

# NumPy Exercise - 15 min

- `[::]`을 활용하여 아래와 같은 **array**를 만들어 봅시다.
- `[start:end:step]`으로 생각하시면 쉽습니다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

nums = np.zeros((5,5), dtype=int)
```

0	10	0	1	0
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	0	1	0

# NumPy Exercise - 15 min

- `[::]`을 활용하여 아래와 같은 **array**를 만들어 봅시다.
- `[start:end:step]`으로 생각하시면 쉽습니다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

nums = np.zeros((5,5), dtype=int)
```

0	00	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

# NumPy Exercise - 15 min

```
import numpy as np  
  
nums = np.zeros((5,5), dtype=int)  
nums[1::2,::2]=1
```

0	0	0	0	0
1	0	1	0	1
0	0	0	0	0
1	0	1	0	1
0	0	0	0	0

# NumPy Exercise - 15 min

```
import numpy as np

nums = np.zeros((5,5), dtype=int)
nums[1::2,::2]=1
nums[:,2,1::2]=1
```

0	1	0	1	0
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	0	1	0

# NumPy Exercise - 8 min

- 잠시 시간을 드릴테니 이번에는 아래와 같은 **array**를 만드려면 어떻게 해야 하는지 생각해봅시다.

```
import numpy as np

nums = np.zeros((5,5), dtype=int)
```

1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1



# NumPy Exercise - 15 min

- 이번에는 `[::-1]` 슬라이싱을 활용하여 2d array를 아래와 같이 뒤집어 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

nums = np.arange(3, 12, 1).reshape(3,3)
```

9	10	11
6	7	8
3	4	5

5	4	3
8	7	6
11	10	9


11	10	9
8	7	6
5	4	3

# NumPy Exercise - 15 min


- 이번에는 `[::-1]` 슬라이싱을 활용하여 2d array를 아래와 같이 뒤집어 봅시다.

3	4	5
6	7	8
9	10	11

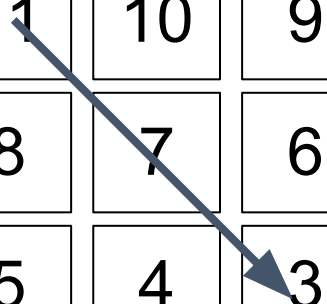
original array



9	10	11
6	7	8
3	4	5



5	4	3
8	7	6
11	10	9



11	10	9
8	7	6
5	4	3

# NumPy Exercise - 3 min

- 아래의 1D array의 element를 모두 더하려면 어떤 함수를 사용해야 할지 생각해봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np  
  
p = np.array([3,4,5])  
  
print(_____)
```



# NumPy Exercise - 3 min

- 정답

```
import numpy as np  
  
p = np.array([3,4,5])  
  
print(np.sum(p))
```

더 빨리 연산을 수행할 수 있는 방법은 없을까요?



# NumPy Exercise - 3 min

## numpy.ufunc.reduce

**ufunc.reduce**(*a*, *axis=0*, *dtype=None*, *out=None*, *keepdims=False*)

Reduces *a*'s dimension by one, by applying ufunc along one axis.

Let  $a.shape = (N_0, \dots, N_i, \dots, N_{M-1})$ . Then *ufunc.reduce*(*a*, each  $a[k_0, \dots, k_{i-1}, j, k_{i+1}, \dots, k_{M-1}]$ . For a one-dimensional array

```
r = op.identity # op = ufunc
for i in range(len(A)):
    r = op(r, A[i])
return r
```

reduce 함수는 ufunc(범용 함수)를 적용하여 array의 dimension을 하나 낮춥니다.

<b>add</b> (x1, x2, /[, out, where, casting, order, ...])	Add arguments element-wise.
<b>subtract</b> (x1, x2, /[, out, where, casting, ...])	Subtract arguments, element-wise.
<b>multiply</b> (x1, x2, /[, out, where, casting, ...])	Multiply arguments element-wise.
<b>matmul</b> (x1, x2, /[, out, casting, order, ...])	Matrix product of two arrays.
<b>divide</b> (x1, x2, /[, out, where, casting, ...])	Returns a true division of the inputs, element-wise.

# NumPy Exercise - 3 min

- 정답

```
import numpy as np  
  
p = np.array([3,4,5])  
  
print(np.add.reduce(p))
```

아래 코드의 실행결과에 대해서 한번 생각해 봅시다.

```
import numpy as np  
  
nums = np.arange(3, 12, 1).reshape(3,3)  
  
print(np.add.reduce(nums))  
print(np.add.reduce(nums,1))
```

# NumPy Exercise - 5 min

```
import numpy as np

nums = np.arange(3, 12, 1).reshape(3,3)

print(np.add.reduce(nums))
print(np.add.reduce(nums,1))
```

3	4	5
6	7	8
9	10	11

original array

18	21	24
----	----	----

reduce with axis = 0

12	21	30
----	----	----

reduce with axis = 1

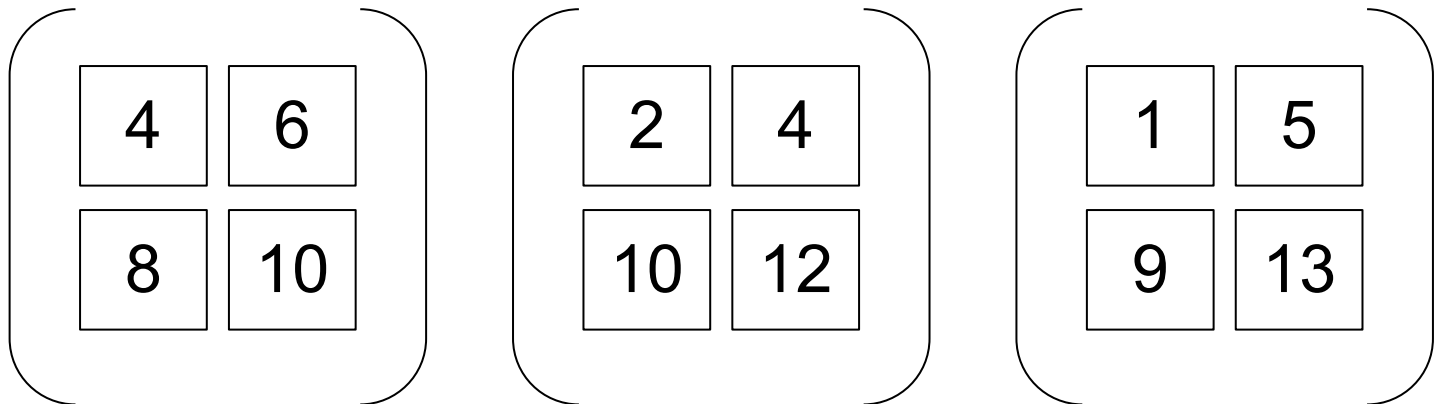
# NumPy Exercise - 15 min

- 이번에는 3D array의 각 axis에 대해 reduce를 적용하여 아래의 결과를 도출해 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

p = np.arange(8).reshape(2,2,2)

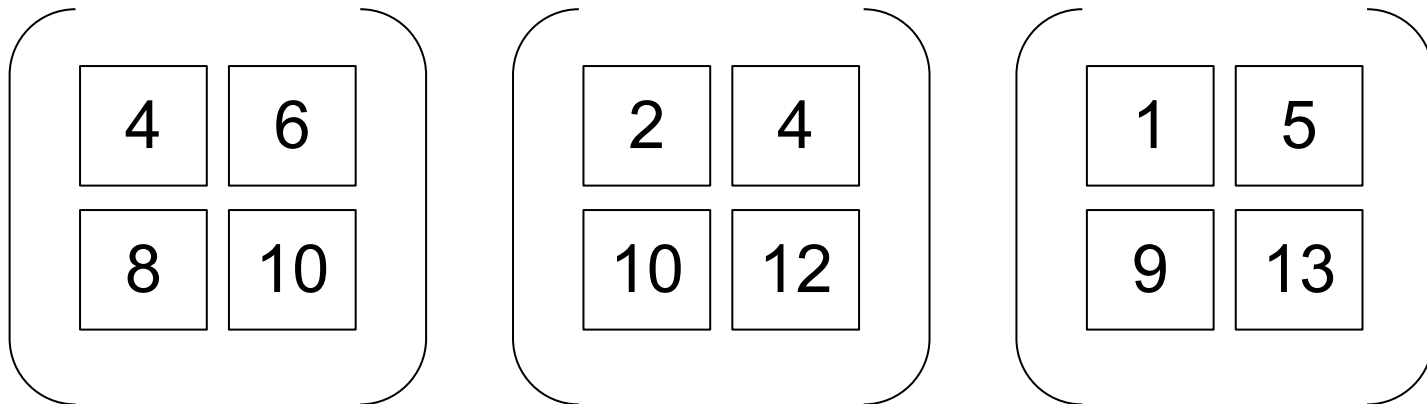
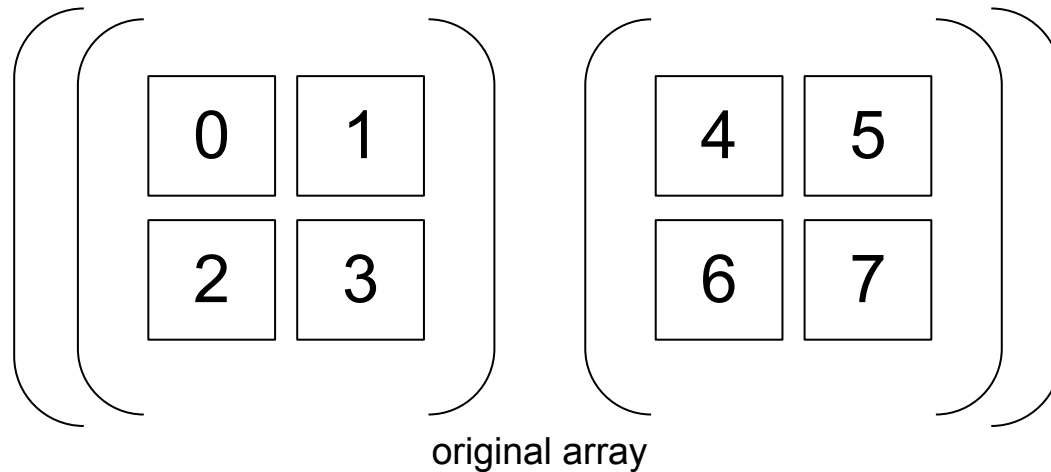
print(p)
```





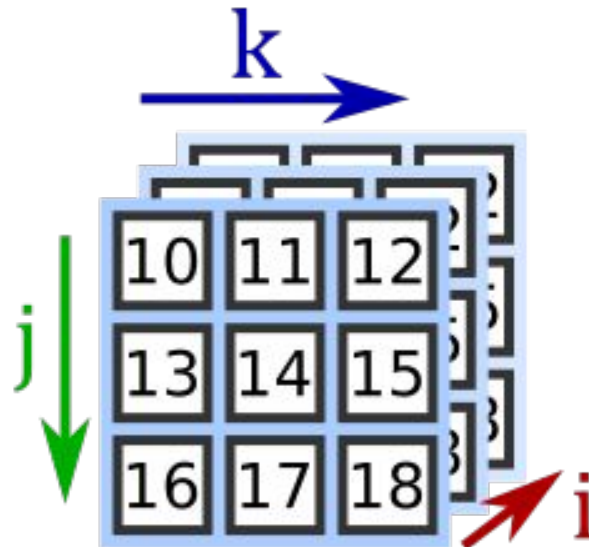
# NumPy Exercise - 15 min

- 이번에는 3D array의 각 axis에 대해 reduce를 적용하여 아래의 결과를 도출해 봅시다.



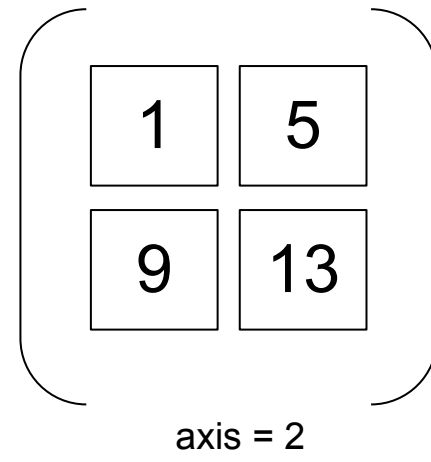
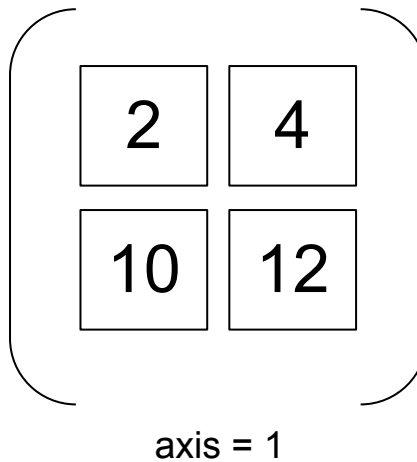
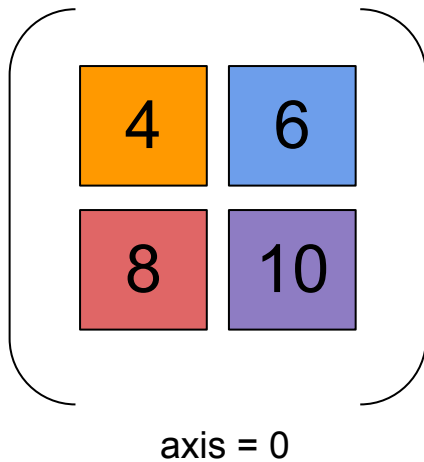
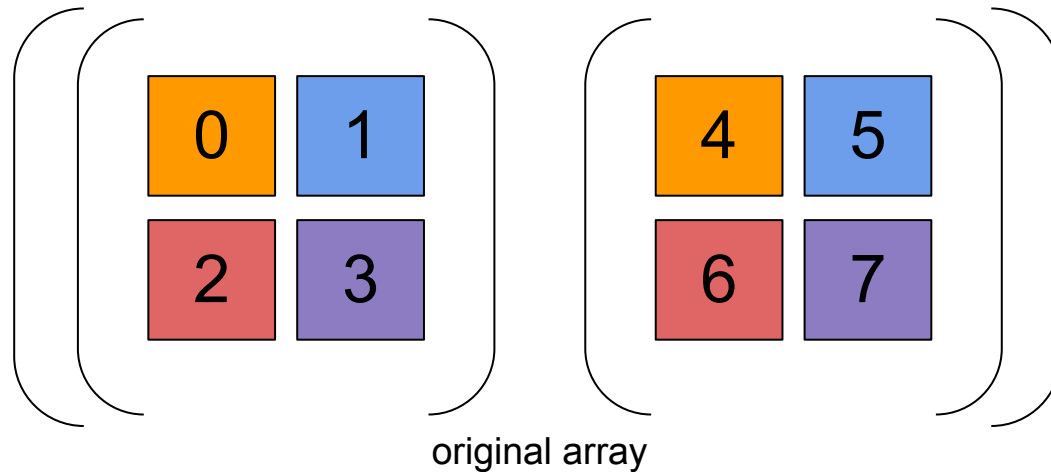
# NumPy Exercise - 15 min

- index  $i$  (axis = 0): selects the matrix
- index  $j$  (axis = 1): selects the row
- index  $k$  (axis = 2): selects the column



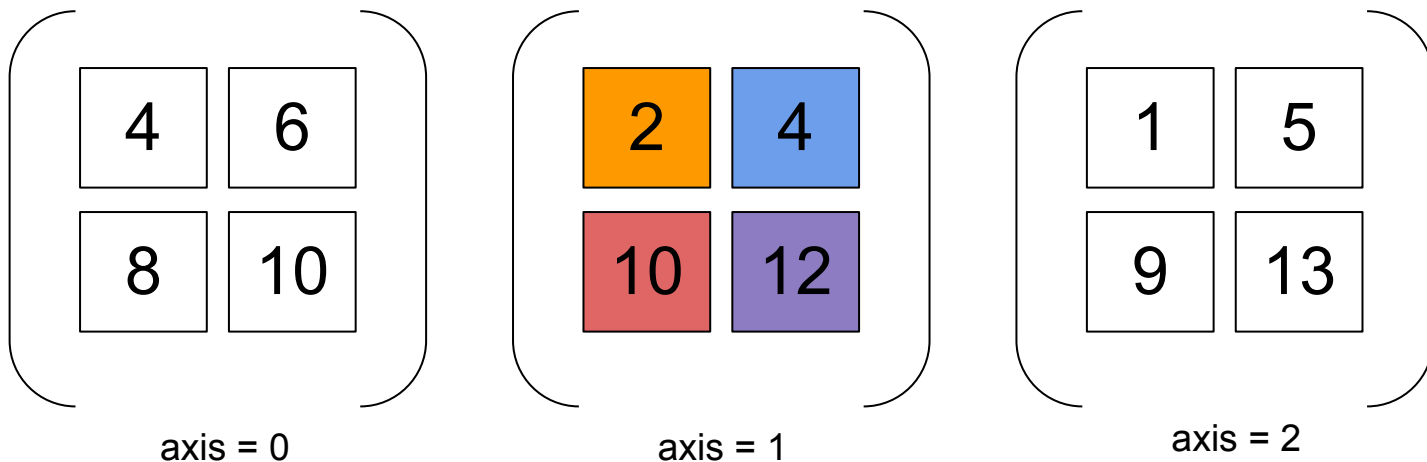
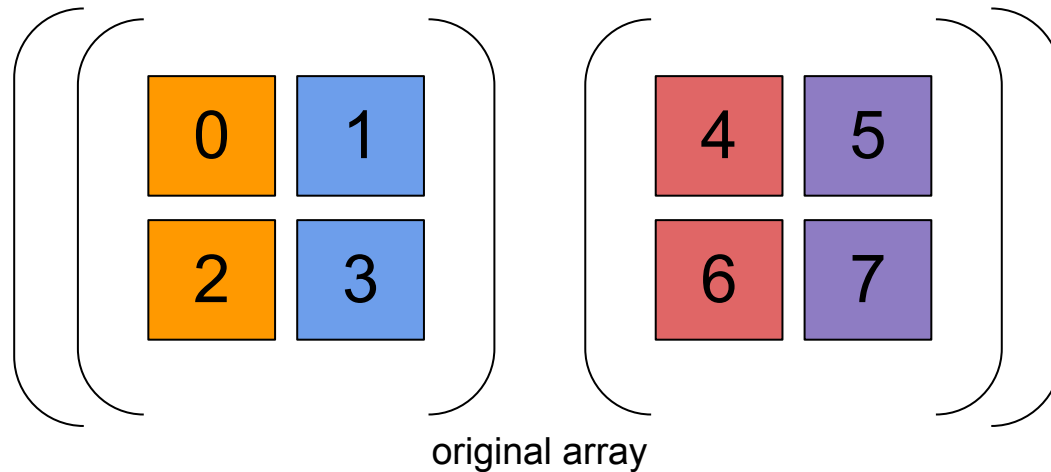
# NumPy Exercise - 15 min

- reduce with axis = 0



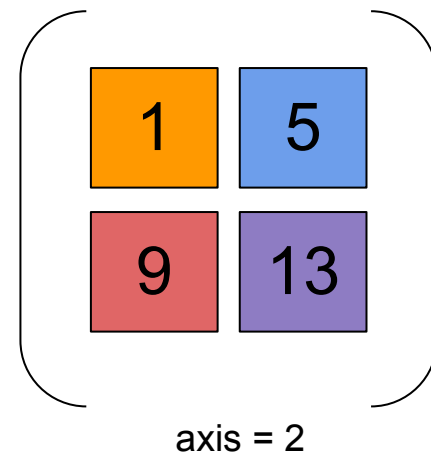
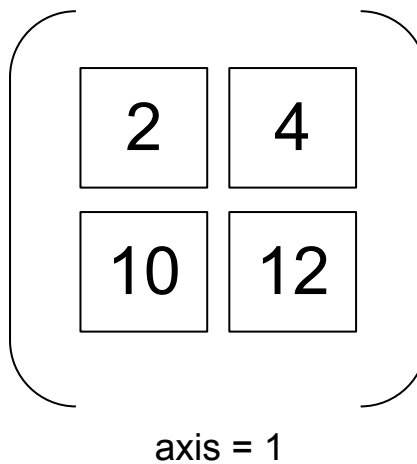
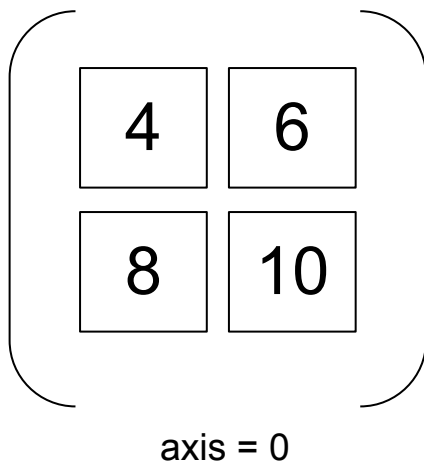
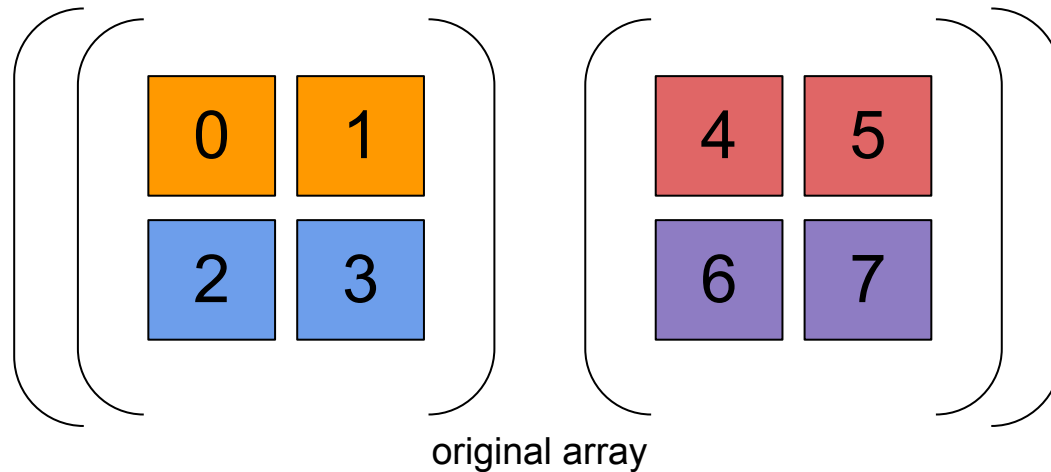
# NumPy Exercise - 15 min

- reduce with axis = 1



# NumPy Exercise - 15 min

- reduce with axis = 2



# NumPy Exercise - 8 min

- numpy의 `zeros_like` 함수를 사용하여 아래와 같은 `array`를 만들어 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np  
  
p = np.arange(3, 12, 1).reshape(3,3)  
q = np.array([10, 11, 12])  
  
print(_____)
```

10	11	12
10	11	12
10	11	12

# NumPy Exercise - 8 min

```
import numpy as np

p = np.arange(3, 12, 1).reshape(3,3)
q = np.array([10, 11, 12])
y = np.zeros_like(p)

print(y)
```

0	0	0
0	0	0
0	0	0

# NumPy Exercise - 8 min

```
import numpy as np

p = np.arange(3, 12, 1).reshape(3,3)
q = np.array([10, 11, 12])
y= np.zeros_like(p)

for i in range(3):
    y[i,:] += q

print(y)
```

10	11	12
10	11	12
10	11	12



# NumPy Exercise - 8 min

```
import numpy as np

p = np.arange(3, 12, 1).reshape(3,3)
q = np.array([10, 11, 12])

print(np.tile(q,(3,1)))
```

10	11	12
10	11	12
10	11	12

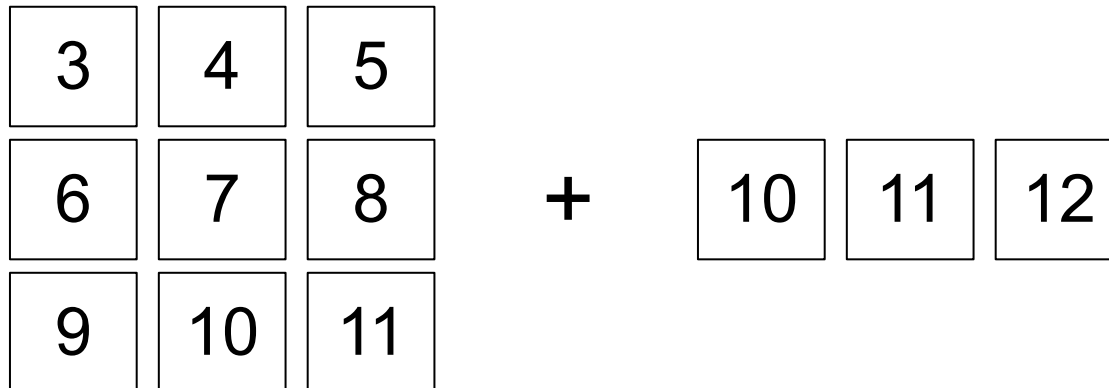
# NumPy Exercise - 5 min

- numpy의 **zeros\_like** 함수 또는 **tile** 함수를 사용하여 아래의 두 array를 더해봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

p = np.arange(3, 12, 1).reshape(3,3)
q = np.array([10, 11, 12])

print(_____)
```



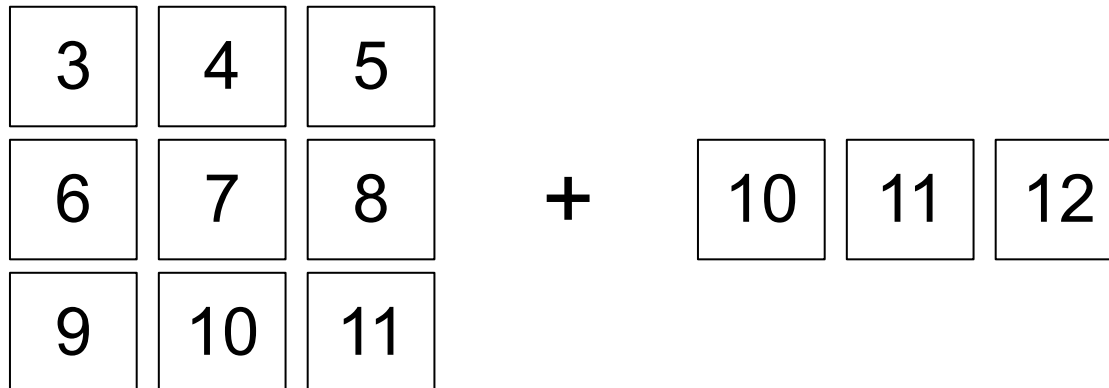
# NumPy Exercise - 5 min

- 이번에는 이전과 동일한 작업을 **broadcasting**을 사용하여 수행해 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

p = np.arange(3, 12, 1).reshape(3,3)
q = np.array([10, 11, 12])

print(_____)
```



# NumPy Exercise - 5 min

- 이번에는 이전과 동일한 작업을 **broadcasting**을 사용하여 수행해 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

p = np.arange(3, 12, 1).reshape(3,3)
q = np.array([10, 11, 12])

print(p+q)
```

3	4	5		10	11	12
6	7	8	+	10	11	12
9	10	11		10	11	12

# Broadcasting rule (recap)

- 두 Array에 대해 operation을 적용할 때, numpy는 두 array의 shape를 element-wise로 비교합니다.
- 각 array는 최소한 한 차원(dimension)을 가지고 있다.
- 두 array의 차원을 뒤쪽부터 시작해서 비교하였을 때, 크기가 동일하거나, 둘 중 하나가 1이거나, 둘 중 하나가 존재하지 않으면 broadcastable하다.

```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):    7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5
```

# Broadcasting rule (recap)

- Broadcasting이 적용되는 경우

```
A      (2d array): 5 x 4
B      (1d array): 1
Result (2d array): 5 x 4
```

```
A      (2d array): 5 x 4
B      (1d array): 4
Result (2d array): 5 x 4
```

```
A      (3d array): 15 x 3 x 5
B      (3d array): 15 x 1 x 5
Result (3d array): 15 x 3 x 5
```

```
A      (3d array): 15 x 3 x 5
B      (2d array): 3 x 5
Result (3d array): 15 x 3 x 5
```

```
A      (3d array): 15 x 3 x 5
B      (2d array): 3 x 1
Result (3d array): 15 x 3 x 5
```

# Broadcasting rule (recap)

- Broadcasting이 적용되지 않는 경우

```
A      (1d array):  3
B      (1d array):  4 # trailing dimensions do not match

A      (2d array):    2 x 1
B      (3d array):  8 x 4 x 3 # second from last dimensions mismatched
```

# NumPy Exercise - 5 min

- 아래의 두 **array**를 **broadcasting**을 사용하여 더하면 어떤 결과가 나올지 생각해 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

p = np.array([[3], [6], [9]])
q = np.array([10, 11, 12])

print(p+q)
```

$$\begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix} + \begin{matrix} 10 & 11 & 12 \end{matrix}$$



# NumPy Exercise - 5 min

- 아래의 두 **array**를 **broadcasting**을 사용하여 더하면 어떤 결과가 나올지 생각해 봅시다.
- 아래의 코드에서 시작해 봅시다.

```
import numpy as np

p = np.array([[3], [6], [9]])
q = np.array([10, 11, 12])

print(p+q)
```

