2022 가을학기

Big Data Analysis & Visualization

Numpy

Day

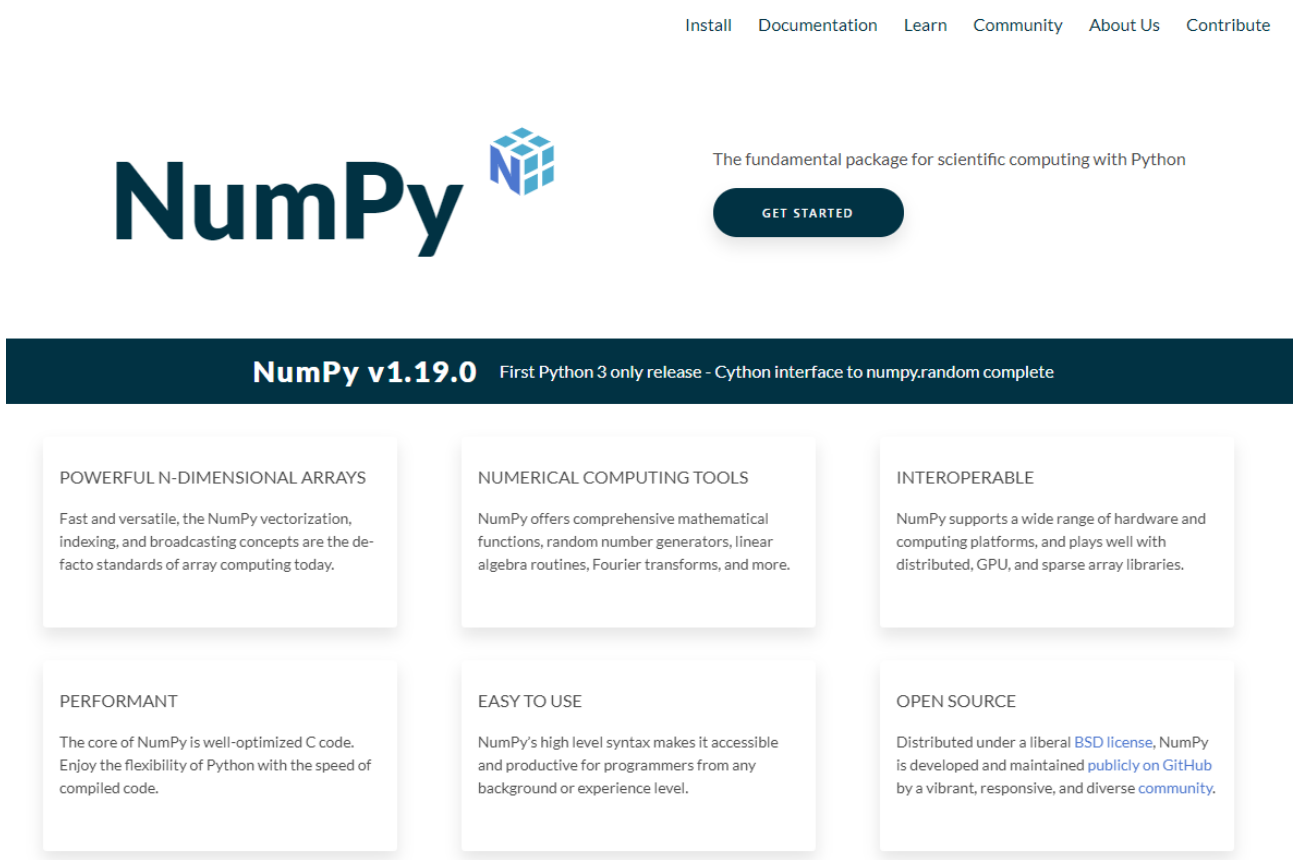
03 Numpy

CONTENTS

A Numpy

Numpy

- Numpy library
  - https://numpy.org/



Numpy

❖ 파이썬의 단점

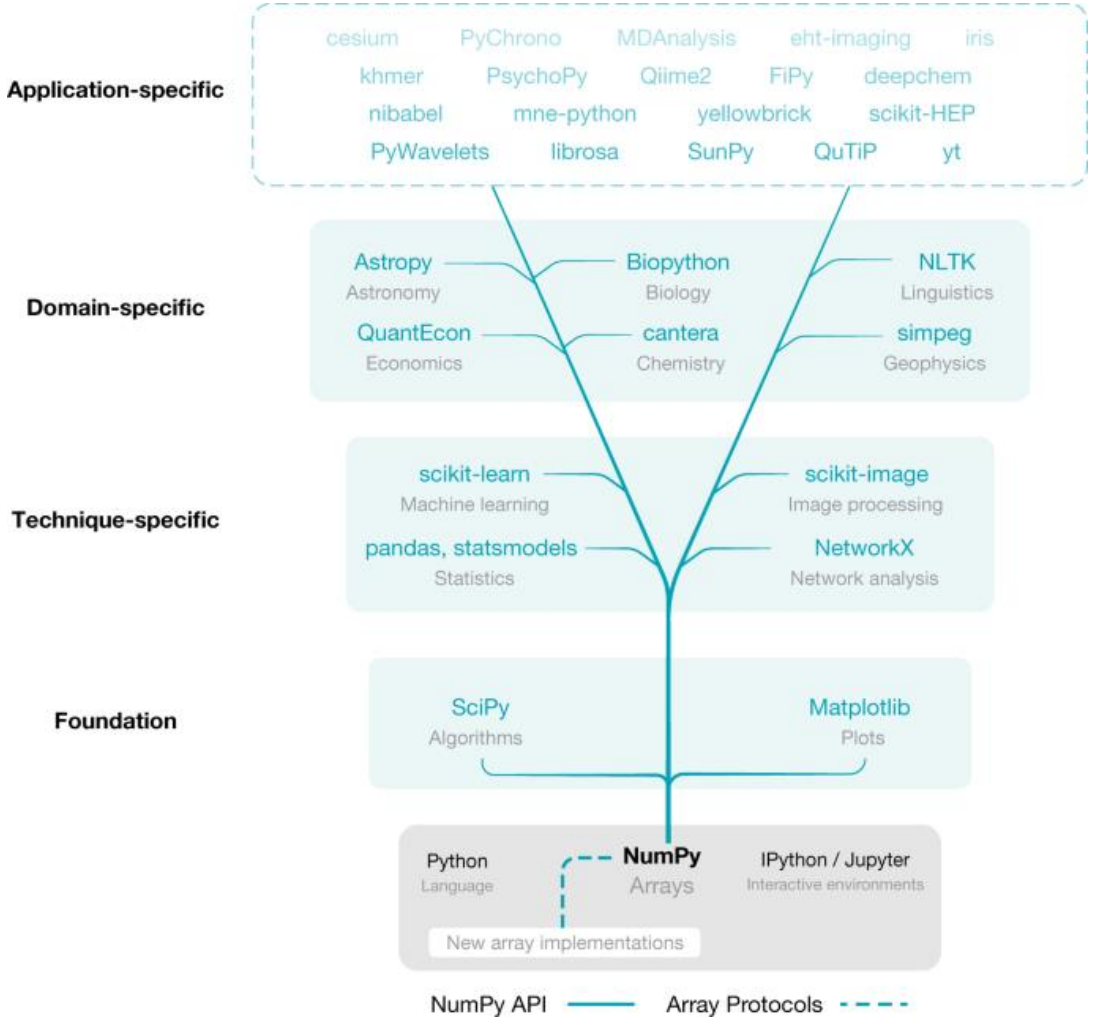
- glue language
- 속도가 매우 느림
- 특히, for, while loop를 사용하는 경우 속도가 상대적으로 매우 느림

❖ Numpy

- Numerical Python
- C언어로 구현된 저수준 고성능 라이브러리
- 빠르고 메모리 효율적으로 다차원 배열 ndarray 연산 지원
- 선형대수, 난수 발생기, 푸리에 변환 등 다양한 연산 기능 지원

Numpy

❖ Numpy 중요성



Numpy

- Jupyter Notebook을 실행합니다
- 명령 프롬프트(CMD)를 실행하고 다음 명령어를 사용합니다
  - python pip install numpy
- numpy 라이브러리 버전을 확인합니다

```
import numpy as np                    //Importing numpy library
print(np.__version__)                 //Printing numpy library version
```

```
1.16.3
```

B

Creating Numpy Array

Creating Numpy Array

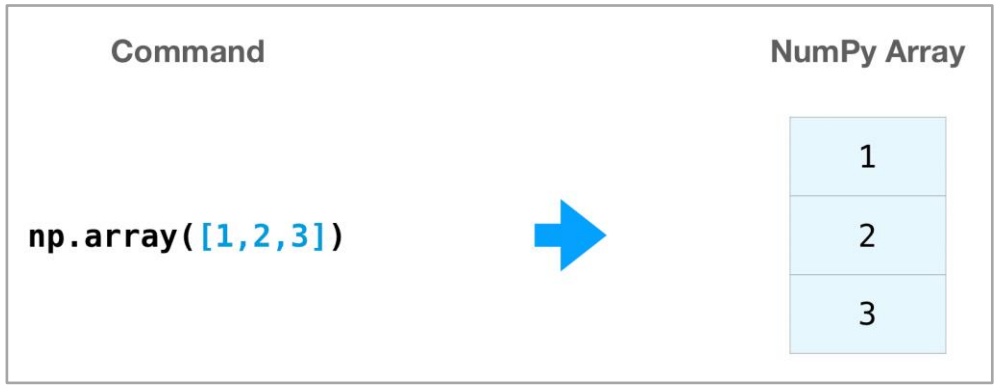
❖ Creating a basic array

- To create a NumPy array, you can use the function np.array()

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

```
[1 2 3]
```

- You can visualize your array this way:



Creating Numpy Array

Array types

- 0-D Arrays

```
arr = np.array(42)

print(arr)
```

```
42
```

- 1-D Arrays (Also called uni-dimensional array)

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

print(arr)
```

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

Creating Numpy Array

- **Array types**
  - 2-D Arrays

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

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

  - 3-D Arrays

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

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

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

Creating Numpy Array

- **Initial Placeholders**
  - An array filled with 0's

```
np.zeros(2)
```

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

  - An array filled with 1's

```
np.ones(2)
```

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

Creating Numpy Array

Initial Placeholders

- An empty array

```
np.empty(3)
```

```
array([4.24399158e-314, 8.48798317e-314, 1.27319747e-313])
```

- A full array with a specific number

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

```
np.full(shape = 3, fill_value = 7)
```

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

Creating Numpy Array

Initial Placeholders

- Create a NxN ‘identity’ matrix

```
np.eye(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

- Create an array with random values

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

```
array([[0.90253491, 0.86274535],
       [0.77733784, 0.71906303]])
```

Creating Numpy Array

- **Initial Placeholders**
  - An array with a range of elements

```
np.arange(4)
```

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

  - An array that contains a range of evenly spaced intervals
    - first number, last number, and step size

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

```
array([2, 4, 6, 8])
```

  - An array with values that are spaced linearly in interval
    - np.linspace()

```
np.linspace(0, 10, num=5)
```

```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

C

Inspecting Numpy Array

Inspecting Numpy Array

Data types in numpy

- dtype returns the data type of the array

```
arr = np.array([0, 1, 2, 3])
print(arr.dtype)
```

```
int32
```

- Function np.array() can take an optional argument: dtype
  - Allows us to define the expected data type of the array

```
arr = np.array([1, 2, 3, 4], dtype='f')
print(arr)
print(arr.dtype)
```

```
[1. 2. 3. 4.]
float32
```

Inspecting Numpy Array

Data types in numpy

- Below is a list of all data types in NumPy and the characters used to represent them
  - i - integer
  - b - byte
  - u - unsigned integer
  - f - float
  - c - complex float
  - m - timedelta
  - M - datetime
  - O - object
  - S - string
  - U - unicode string
  - V - fixed chunk of memory for other type (void)

Inspecting Numpy Array

- **Data types in numpy**
  - A non integer string like 'a' can not be converted to integer
    - Will raise an error

```
arr = np.array(['a', '2', '3'], dtype='i')

print(arr)
print(arr.dtype)
```

```
ValueError: invalid literal for int() with base 10: 'a'
```

Inspecting Numpy Array

- **Data types in numpy**
  - Convert data type of existing array by copying of the array with the astype() method

```
arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```

```
[1 2 3]
int32
```

Inspecting Numpy Array

- **Find number of dimensions of the array**
  - ndarray.ndim will tell you the number of axes, or dimensions, of the array

```
array_example = np.array([[[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                          [[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                          [[0 ,1 ,2, 3],
                           [4, 5, 6, 7]]])

print(array_example.ndim)
```

```
3
```

Inspecting Numpy Array

- **Find the total number of elements in the array**
  - ndarray.size will tell you the total number of elements of the array

```
array_example = np.array([[[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                          [[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                          [[0 ,1 ,2, 3],
                           [4, 5, 6, 7]]])

print(array_example.size)
```

```
24
```

Inspecting Numpy Array

- Find the length or the number of elements in the array

```
array_example = np.array([[[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                          [[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                          [[0 ,1 ,2, 3],
                           [4, 5, 6, 7]]])

print(len(array_example))
```

```
3
```

Inspecting Numpy Array

- **Find the shape of your array**
  - ndarray.shape will display a tuple of integers that indicate the number of elements stored along each dimension of the array

```
array_example = np.array([[[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                          [[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                          [[0 ,1 ,2, 3],
                           [4, 5, 6, 7]]])

print(array_example.shape)
```

```
(3, 2, 4)
```

Inspecting Numpy Array

Array indexing

- Accessing index of 3-D array

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

print(arr[0, 1, 2])
```

- Result?

```
6
```

D

Numpy Array Manipulation

Inspecting Numpy Array

- **Reshaping an array**
  - Using arr.reshape() will give a new shape to an array without changing the data

```
a = np.arange(6)

b = a.reshape(3, 2)

print(b)
```

```
[[0 1]
 [2 3]
 [4 5]]
```

Inspecting Numpy Array

- **Flattening the arrays**
  - Flattening array means converting a multidimensional array into a 1D array

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

newarr = arr.reshape(-1)

print(newarr)
```

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

Numpy Array Manipulation

- **Slicing**
  - Similar to slicing Python lists

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

print(data[1])
print(data[0:2])
print(data[1:])
print(data[-2:])
```

```
2
[1 2]
[2 3]
[2 3]
```

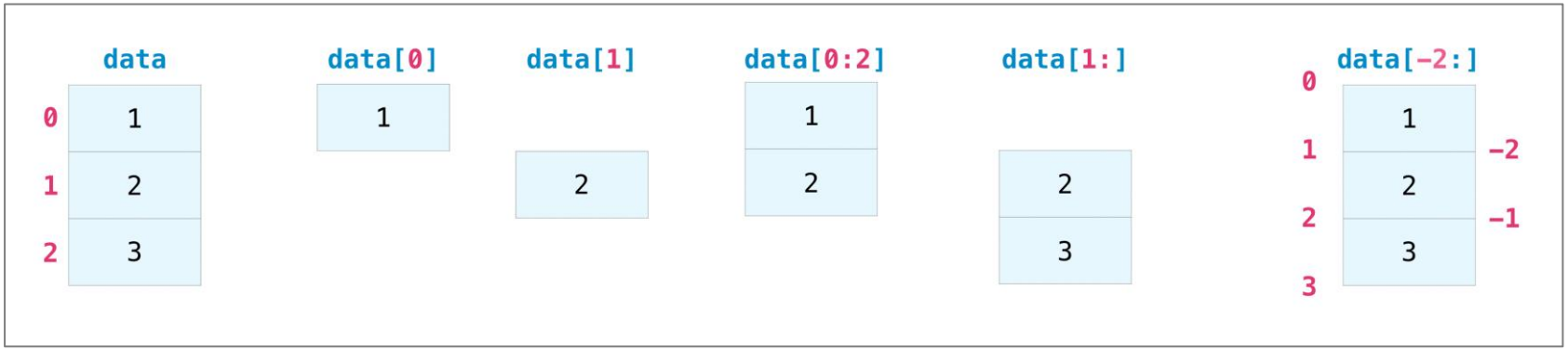
Numpy Array Manipulation

❖ Slicing

- Similar to slicing Python lists

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

print(data[1])
print(data[0:2])
print(data[1:])
print(data[-2:])
```



Numpy Array Manipulation

Slicing

- Negative slicing example

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

print(arr[-3:-1])
```

```
[5 6]
```

- Slicing on 2-D array example

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

print(arr[0:2, 2])
```

```
[3 8]
```

Numpy Array Manipulation

Joining arrays

- Use concatenate() function to join two arrays

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

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

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

Numpy Array Manipulation

- **Joining arrays using stack() function**
  - Stacking is done along a new axis

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

arr2 = np.array([4, 5, 6])

arr = np.stack((arr1, arr2), axis=1)

print(arr)
```

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

Numpy Array Manipulation

- **Split numpy arrays**
  - Use array_split() for splitting arrays

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

newarr = np.array_split(arr, 3)

print(newarr)
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

Numpy Array Manipulation

- **Copy numpy array**
  - Use copy() function

```
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

Numpy Array Manipulation

Search in numpy array

- Use where() function to find the elements

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

x = np.where(arr == 4)

print(x)
```

```
(array([3, 5, 6], dtype=int32),)
```

Numpy Array Manipulation

- **Search in numpy array**
  - Use where() function to find the elements

```
a = np.arange(15)

index = np.where((a >= 5) & (a <= 10))

print(a[index])
```

```
[ 5  6  7  8  9 10]
```

Numpy Array Manipulation

- **Intersection of two numpy arrays**
  - For 1-D array, you can use intersect1d() function

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

np.intersect1d(a,b)
```

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

Numpy Array Manipulation

- **Difference between two numpy arrays**
  - For 1-D array, you can use setdiff1d() function

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

# From 'a' remove all of 'b'
np.setdiff1d(a,b)
```

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

Numpy Array Manipulation

- **Sorting numpy array**
  - Use sort() function

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

print(np.sort(arr))
```

```
[0 1 2 3]
```

Numpy Array Manipulation

- **Getting unique numbers from numpy array**
  - Use np.unique to print the unique values in your array

```
a = np.array([11, 11, 12, 13, 14, 15, 16, 17, 12, 13, 11, 14, 18, 19, 20])

unique_values = np.unique(a)

print(unique_values)
```

```
[11 12 13 14 15 16 17 18 19 20]
```

E

Numpy Array Math Operators

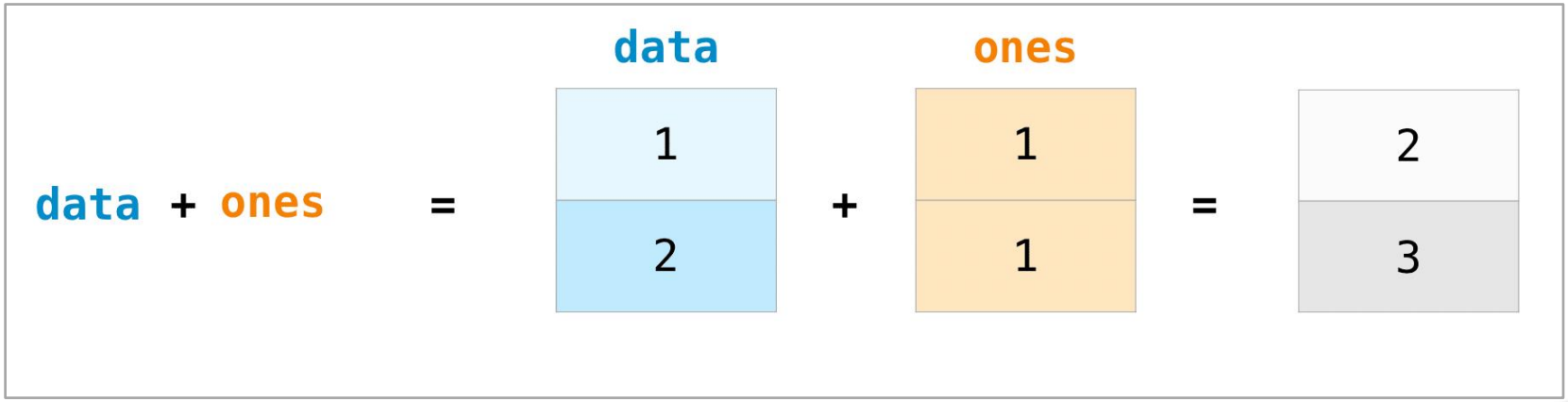
Numpy Array Math Operators

- **Basic arithmetic operations**
  - Addition

```
data = np.array([1, 2])
ones = np.ones(2, dtype=int)

print(data + ones)
```

```
[2 3]
```

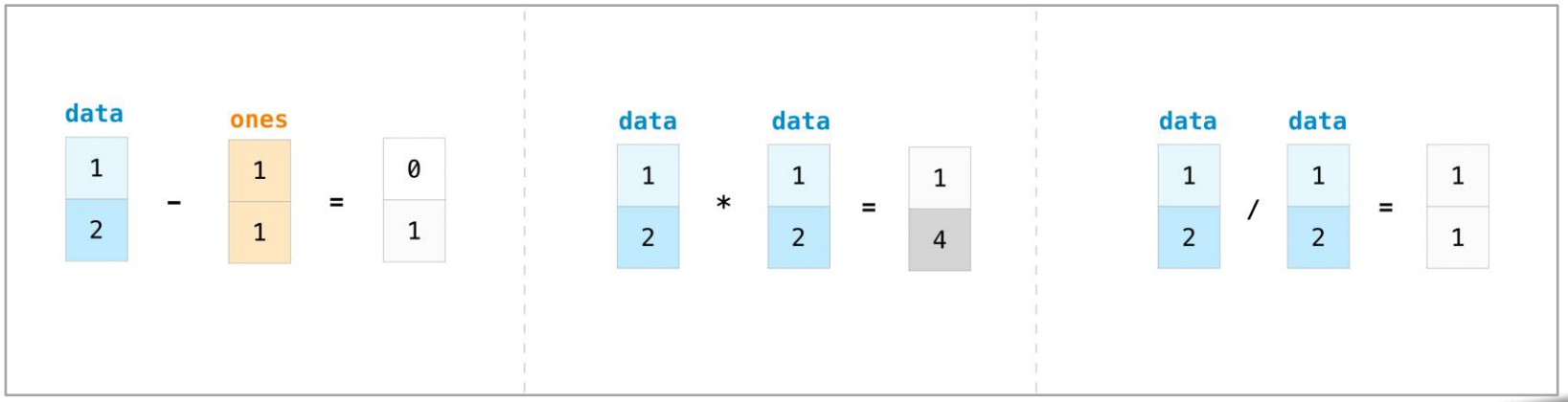


Numpy Array Math Operators

- **Basic arithmetic operations**
  - Subscription, multiplication, division

```
print(data - ones)
print(data * data)
print(data / data)
```

```
[0 1]
[1 4]
[1. 1.]
```



Numpy Array Math Operators

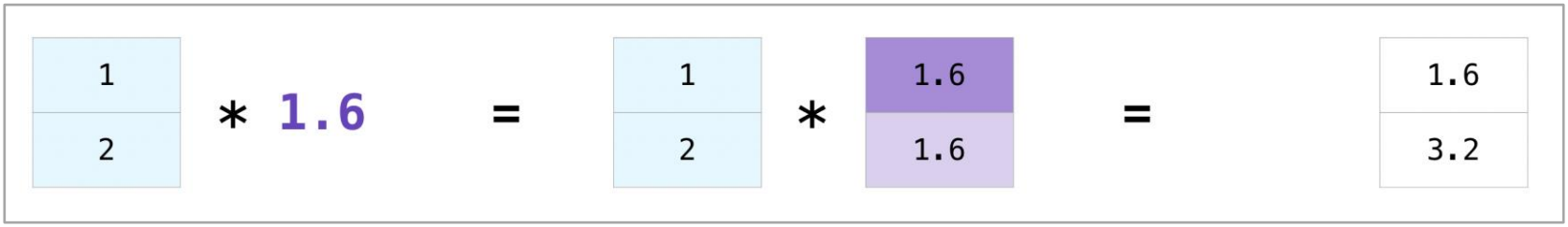
❖ Broadcasting

- An operation between an array and a single number

```
data = np.array([1.0, 2.0])

print(data * 1.6)
```

```
[1.6 3.2]
```

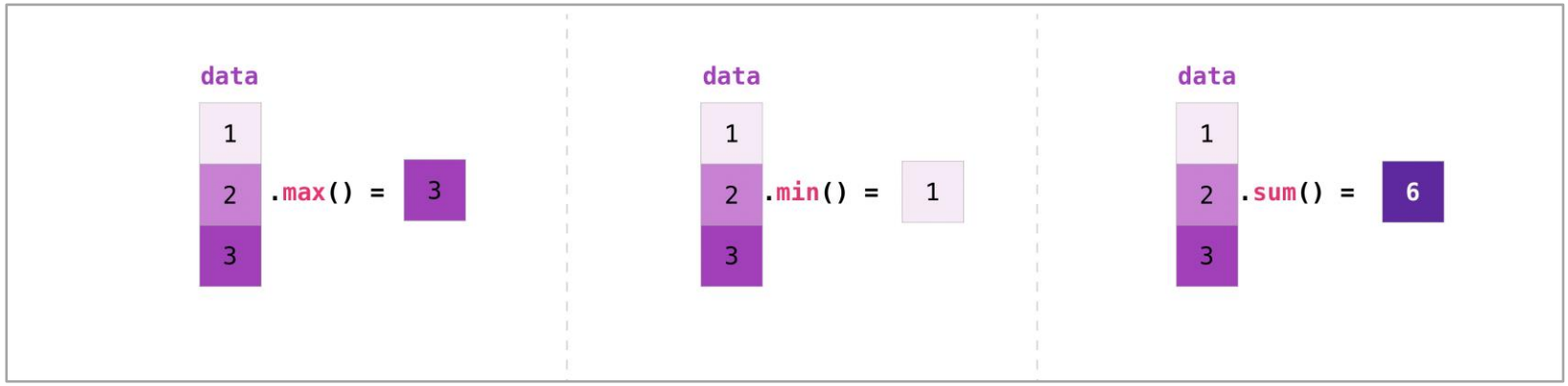


Numpy Array Math Operators

- **Other useful operations**
  - Max, min and sum

```
print(data.max())
print(data.min())
print(data.sum())
```

```
2.0
1.0
3.0
```



감사합니다!