



## 04. 라이브러리의 활용

# 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용

## NumPy

- ▶ 메모리 효율적인 벡터 산술연산 기능 제공
- ▶ 반복문 없이 전체 데이터 배열 일괄 연산 기능 제공
- ▶ 선형대수, 난수 발생, 푸리에 변환 등 다양한 연산 기능 제공

라이브러리 호출

```
1 import numpy as np
```



```
1 from numpy import *
```



```
1 import numpy as np
2 np.array()
```

### Arguments

```
array(object, dtype=None, copy=True, order='K',
      subok=False, ndmin=0)
```

# 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용


## $f(x)$ Numpy의 배열 생성 함수

- ▶ 메모리 효율적인 벡터 산술연산 기능 제공
- ▶ 반복문 없이 전체 데이터 배열 일괄 연산 기능 제공
- ▶ 선형대수, 난수 발생, 푸리에 변환 등 다양한 연산 기능 제공

함수	설명
array	입력 데이터를 ndarray로 변환
asarray	입력 데이터를 ndarray로 변환하지만 입력 데이터가 이미 ndarray일 경우 복사가 되지 않음
ones, ones_like	1로 채워진 배열
zeros, zeros_like	0으로 채워진 배열
empty, empty_like	비어있는 배열
eye, identity	단위 행렬

## 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용

 like 함수의 이해

```
In [2]: np.zeros([5, 4])  
Out[2]:  
array([[0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 0., 0.],  
       [0., 0., 0., 0.]])
```

```
In [3]: np.zeros_like([2, 4, 5, 6])  
Out[3]: array([0, 0, 0, 0])
```



## 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용

</> 배열의 생성

```
In [2]: np.array([1, 2, 3])  
Out[2]: array([1, 2, 3])
```

```
In [3]: np.array([[1, 2], [3, 4]])  
Out[3]:  
array([[1, 2],  
       [3, 4]])
```

# 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용

## </> 배열의 확인

- ▶ 배열 생성 후 객체의 상세한 속성을 확인 가능
- ▶ 대표적인 attribute로는 `ndim`, `shape`, `dtype`가 있다.

```
1 import numpy as np
2 data = np.array([2, 4, 7, 9.1])
3 data.ndim
4 data.shape
5 data.dtype
```

```
In [3]: data.ndim
Out[3]: 1
```

```
In [4]: data.shape
Out[4]: (4,)
```

```
In [5]: data.dtype
Out[5]: dtype('float64')
```

## 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용

</> 색인 - index

● 데이터 준비

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

● 단일 원소

```
In [3]: arr[0][0]  
Out[3]: 1
```

```
In [4]: arr[0, 0]  
Out[4]: 1
```

## 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용

</> 색인 - index

● 다중 원소 - 연속

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

```
In [5]: arr[0, 1:]  
Out[5]: array([2, 3])
```

```
In [6]: arr[0, :1]  
Out[6]: array([1])
```

```
In [7]: arr[0, :2]  
Out[7]: array([1, 2])
```


```
In [8]: arr[0, 1:2]  
Out[8]: array([2])
```

```
In [9]: arr[0, 0:2]  
Out[9]: array([1, 2])
```



## 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용

 색인 - index


 View의 이해

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

```
In [3]: arr_sub = arr[0, 0:2]  
...: arr_sub  
Out[3]: array([1, 2])
```

```
In [4]: arr[0, 0:2] = [99, 88]  
...: arr  
Out[4]:  
array([[99, 88, 3],  
       [ 4,  5,  6]])
```

```
In [5]: arr_sub  
Out[5]: array([99, 88])
```



# 01 수치연산 라이브러리

[Chapter 04] 라이브러리의 활용

</> 색인 - index

View의 이해

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

```
In [3]: arr_sub = arr[0, 0:2].copy()  
...: arr_sub  
Out[3]: array([1, 2])
```

```
In [4]: arr[0, 0:2] = [99, 88]  
...: arr  
Out[4]:  
array([[99, 88, 3],  
       [ 4,  5,  6]])
```

```
In [5]: arr_sub  
Out[5]: array([1, 2])
```

## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용



### Pandas

- ▶ NumPy 기반 라이브러리
- ▶ 행과 열로 이루어진 객체를 다루기 용이(DataFrame)
- ▶ 시계열과 비시계열 데이터를 같이 다룰 수 있는 데이터 구조 제공

● 라이브러리 호출

```
1 import pandas as pd
```

```
1 import pandas as pd
```

```
2 pd.DataFrame()
```

```
3
```

Arguments

DataFrame(rows and columns)

## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용

### </> Series의 생성과 확인

- ▶ NumPy 자료형을 담을 수 있는 1차원 배열
- ▶ value와 index로 구성되어 있음
- ▶ Series() 함수와 대괄호를 사용하여 생성

#### ● 기본 Series 생성

```
In [2]: ser = pd.Series([2, 4, 6, 8])
...: ser
Out[2]:
0      2
1      4
2      6
3      8
dtype: int64
```

```
1 ser.values
2 ser.index
```



## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용

### </> Series의 생성과 확인

● 색인 지정 Series 생성

```
In [2]: ser2 = pd.Series([2, 4, 6, 8], index = ["a", "b", "c", "d"])
...: ser2
Out[2]:
a      2
b      4
c      6
d      8
dtype: int64
```

```
In [3]: ser2[1]
Out[3]: 4
```

```
In [4]: ser2["a"]
Out[4]: 2
```

```
In [5]: ser2[2:]
Out[5]:
c      6
d      8
dtype: int64
```

## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용

### Series의 조작

● 치환

```
In [6]: ser2[1] = 1

In [7]: ser2[1] = "asdf"
Traceback (most recent call last):

  File "<ipython-input-7-bdc2e5371b24>", line 1, in <module>
    ser2[1] = "asdf"

  File "C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py", line 939, in __setitem__
    setitem(key, value)

  File "C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py", line 896, in setitem
    values[key] = value

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

## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용

### </> Series의 조작

● 치환

```
In [8]: ser2["a"] = "asdf"
```

```
In [9]: ser2[:2]
```

```
Out[9]:
```

```
a      asdf
```

```
b          1
```

```
dtype: object
```

## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용

### DataFrame의 생성과 확인

- ▶ 표 같은 스프레드시트 형식의 2차원 자료 구조
- ▶ 각 열(column)에는 숫자, 문자 등 서로 다른 속성의 자료를 넣을 수 있다.
- ▶ DataFrame() 함수, 중괄호, 콜론, 대괄호를 사용하여 생성

#### 기본 DataFrame 생성

```
In [2]: values = {"aa": [1, 2, 3],  
...:              "bb": ["a", "b", "c"]}  
...: df = pd.DataFrame(values)  
...: df
```

Out[2]:

	aa	bb
0	1	a
1	2	b
2	3	c



## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용

### DataFrame의 생성과 확인

● 딕셔너리를 활용한 DataFrame 생성

```
In [2]: values = {"aa": {1: 1, 2: 2, 3: 3},  
...:              "bb": {1: "a", 2: "b", 3: "c"}}  
...: df = pd.DataFrame(values)  
...: df
```

Out[2]:

	aa	bb
1	1	a
2	2	b
3	3	c

## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용

### </> DataFrame의 생성과 확인

● arange와 reshape를 활용한 DataFrame 생성

```
1 import numpy as np
2 import pandas as pd
3
4 df = pd.DataFrame(np.arange(6).reshape(3, 2),
5                   index = ["a", "b", "c"],
6                   columns = ["aa", "bb"])
```

In [3]: df

Out[3]:

	aa	bb
a	0	1
b	2	3
c	4	5

## 02 데이터분석 라이브러리

[Chapter 04] 라이브러리의 활용

### DataFrame의 생성과 확인

#### DataFrame 색인

```
1 df.aa  
2 df["aa"]  
3 df["aa"][1]  
4 df["aa"][:2]
```

```
1 df.columns  
2 df.columns[1]  
3 df.columns = ["xx", "yy"]  
4 df.columns
```

```
1 df = df.rename(columns = {"xx": "obs"})  
2 df.rename(columns = {"xx": "obs"}, inplace = True)
```

```
In [4]: df  
Out[4]:  
      aa bb  
1      1  a  
2      2  b  
3      3  c
```