데이터 분석을 위한 라이브러리

02 데이터 핸들링을 위한 라이브러리 NumPy

/* elice */

01

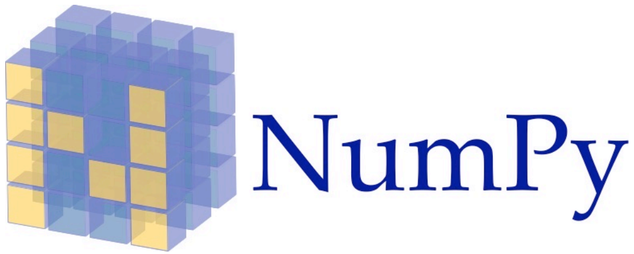
Numpy란?

Numpy?

: Numerical Python

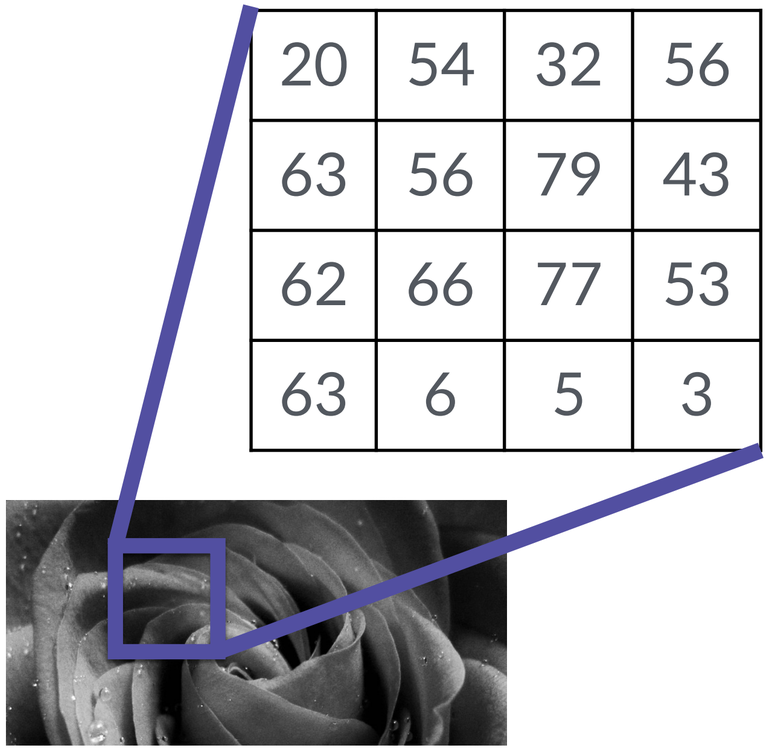
Python에서 **대규모 다차원 배열**을

다룰 수 있게 도와주는 라이브러리

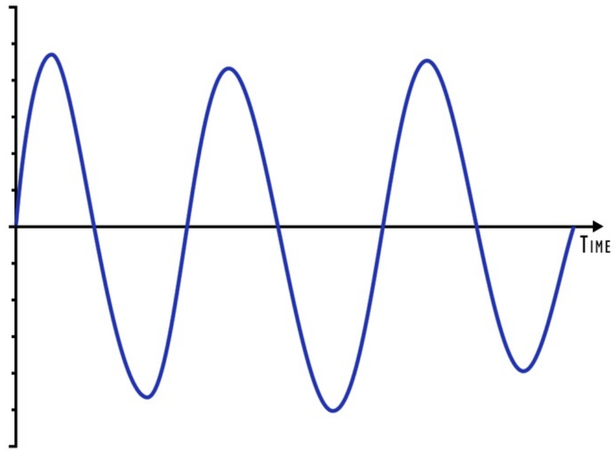


왜 Numpy를 사용하는가?

데이터의 대부분은 숫자 배열로 볼 수 있다



23	44	52	56	42	38



왜 Numpy를 사용하는가?

반복문 없이 배열 처리 가능!

파이썬 리스트에 비해,

빠른 연산을 지원하고 **메모리를 효율적**으로 사용

Numpy 사용하기

list 배열 생성 및 출력 형태 확인

```
list_arr = list(range(5))
print(list_arr)          # [0, 1, 2, 3, 4] -> 콤마(,)로 구분
print(type(list_arr))    # <class 'list'>
```

Numpy 사용하기

import(불러오다) 키워드를 이용해서 numpy 불러오기

```
import numpy as np
# numpy 모듈 불러와서 'np' 별칭 부여
```

배열 생성하기

numpy 배열 생성 및 출력 형태 확인

ndarray = n차원의 배열(n-dimensional array)

```
import numpy as np

np_arr = np.array(range(5))

print(np_arr)              # [0 1 2 3 4] -> 공백으로 구분

print(type(np_arr))        # <class 'numpy.ndarray'>
```

02

배열의 기초

배열의 데이터 타입 dtype

파이썬 리스트와 달리 **같은 데이터 타입**만 저장 가능!

```
arr = np.array([0, 1, 2, 3, 4], dtype=float)
print(arr)                    # [0. 1. 2. 3. 4.]
print(arr.dtype)              # ‘float64’
print(arr.astype(int))        # [0 1 2 3 4]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

배열의 데이터 타입 dtype

dtype	설명	다양한 표현
int	정수형 타입	i, int_, int32, int64, i8
float	실수형 타입	f, float_, float32, float64, f8
str	문자열 타입	str, U, U32
bool	부울 타입	?, bool_

배열의 속성

ndarray 의 차원 관련 속성 : ndim & shape

1차원 배열.py

```
list = [0, 1, 2, 3]
arr = np.array(list)
print(arr.ndim)          # 1
print(arr.shape)         # (4,)
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

2차원 배열.py

```
list = [[0,1,2],[3,4,5]]
arr = np.array(list)
print(arr.ndim)          # 2
print(arr.shape)         # (2,3)
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

배열의 속성

ndarray 의 크기 속성과 shape 조절

크기속성.py

```
arr = np.array([0, 1, 2, 3, 4, 5])
print("arr.shape : {}".format(arr.shape))        # arr.shape : (6,)
print("배열 요소의 수 : {}".format(arr.size))    # 배열 요소의 수 : 6
print("배열의 길이 : {}".format(len(arr)))       # 배열의 길이 : 6
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

배열의 속성

ndarray 의 크기 속성과 shape 조절

크기속성.py

```
…
arr.shape = 3, 2
print("arr.shape : {}".format(arr.shape))        # arr.shape : (3,2)
print("배열 요소의 수 : {}".format(arr.size))    # 배열 요소의 수 : 6
print("배열의 길이 : {}".format(len(arr)))       # 배열의 길이 : 3
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

03

Indexing & Slicing

찾고 잘라내기

indexing : 인덱스로 값을 찾아냄

```
x = np.arange(7)
print(x)      # [0 1 2 3 4 5 6]
print(x[3])   # 3
print(x[7])
# IndexError: index 7 is out of bounds
x[0] = 10
print(x)      # [10  1  2  3  4  5  6]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

```
x = np.arange(1, 13 ,1)
x.shape = 3, 4
print(x)          # [[ 1  2  3  4]
                  #  [ 5  6  7  8]
                  #  [ 9 10 11 12]]

print(x[2, 3])    # 12
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

찾고 잘라내기

slicing : 인덱스의 값으로 배열의 일부분을 가져옴

```
x = np.arange(7)
print(x)          # [0 1 2 3 4 5 6]

print(x[1:4])     # [1 2 3]
print(x[1:])      # [1 2 3 4 5 6]
print(x[:4])      # [0 1 2 3]
print(x[::2])     # [0 2 4 6])
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

```
x = np.arange(1, 13 ,1)
x.shape = 3, 4
print(x)              # [[ 1  2  3  4]
                      #  [ 5  6  7  8]
                      #  [ 9 10 11 12]]
print(x[1:2,:2:3]) # [[5]]
print(x[1:,:2])     # [[ 5  6]
                      #  [ 9 10]]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

Boolean indexing

Boolean indexing : 배열의 각 요소의 선택 여부를 **Boolean mask**를 이용하여 지정하는 방식

```
x = np.arange(7)


# Boolean mask? True, False로 구성된 mask array
print(x)          # [0 1 2 3 4 5 6]
print(x < 3)      # [True True True False False False False]
print(x > 7)      # [False False False False False False False]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

Boolean indexing

Boolean indexing : 배열의 각 요소의 선택 여부를 **Boolean mask**를 이용하여 지정하는 방식

```
x = np.arange(7)


# Boolean mask의 True 요소에 해당하는 index만을 조회
print(x[x < 3])         # [0 1 2]
print(x[x % 2 == 0])    # [0 2 4 6]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

Fancy indexing

Fancy indexing : 배열의 각 요소 선택을
Index 배열을 전달하여 지정하는 방식

```
x = np.arange(7)
print(x)                    # [0 1 2 3 4 5 6]


print(x[[1, 3, 5]])     # [1 3 5]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

```
x = np.arange(1, 13 ,1).reshape(3, 4)
print(x)                # [[ 1  2  3  4]
                        #  [ 5  6  7  8]
                        #  [ 9 10 11 12]]
print(x[[0, 2]]) # [[ 1  2  3  4]
                        #  [ 9 10 11 12]]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

indexing x slicing

원하는 요소를 지정하기 위해

Indexing과 **Slicing**을 적절히 조합하여 사용 가능

```
x = np.arange(1, 13 ,1).reshape(3, 4)
print(x)              # [[ 1  2  3  4]
                      #  [ 5  6  7  8]
                      #  [ 9 10 11 12]]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

```
print(x[1:2, 2])      #[7]
print(x[[0,2], 2])    #[ 3 11]
print(x[[0,2], :2])   #[[ 1  2]
                      #  [ 9 10]]
```

*numpy 라이브러리는 이미 import 해둔 것으로 가정

크레딧

/* elice */

코스 매니저
하주희

콘텐츠 제작자
임원균, 하주희

강사
황지영

감수자
장석준

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io