파이썬 라이브러리 시작하기

학습 목표

- Numpy 패키지에 대해 이해하고 실습해 보기
- 시각화 라이브러리에 대해 이해하고 실습해 본다.
- 데이터 처리 라이브러리에 대해 이해하고 실습해 본다.
- 웹 데이터 수집 라이브러리에 대해 이해하고 실습해 본다.

1-1-1 Numpy 라이브러리 시작하기

> NumPy (Numerical Python)은 과학 계산을 위해 파이썬에서 가장 널리 사용되는 라이브러리 중 하나입니다. NumPy는 대규모 다차원 배열과 행렬을 처리할 수 있는 강력한 기능을 제공하며, 수학 함수 라이브러리를 포함하여 이러한 배열과 행렬을 효율적으로 조작할 수 있습니다.

- 머신러닝 기본 알고리즘은 선형대수와 통계 등에 기반
  - 파이썬에서 선형대수 기반의 연산 및 기본 수학함수 등을 지원
  - 다차원 배열을 쉽게 생성하고 다양한 연산 가능
- 매우 빠른 연산 속도
- Pandas, Tensorflow의 내부 연산에서도 이를 활용하여 속도을 개선함.

numpy 불러오기

In [2]:
```
import numpy as np
```

1차, 2차, 3차 배열 만들기

In [3]:
```
# 1차 배열
dim1 = np.array( [1,2,3] )

# 2차 배열
dim2 = np.array( [ [1,2,3],
                   [4,5,6] ])

print("dim1의 크기 :", dim1.shape)
print("dim2의 크기 :", dim2.shape)
```

```
dim1의 크기 : (3,)
dim2의 크기 : (2, 3)
```

In [4]:
```
# 3차 배열
dim3 = np.array( [
        [ [1,2,3], [4,5,6], [7,8,9] ],
        [ [11,12,13], [14,15,16], [17,18,19] ],
        [ [21,22,23], [24,25,26], [27,28,29] ]
]
)
```

shape를 활용한 크기 확인

In [5]:
```
print("dim3의 크기 :", dim3.shape)
```

```
dim3의 크기 : (3, 3, 3)
```

ndim - (attribute) 을 활용한 차원 확인

In [6]:
```
print("dim1의 크기 :", dim1.ndim)
print("dim2의 크기 :", dim2.ndim)
print("dim3의 크기 :", dim3.ndim)
```

```
dim1의 크기 : 1
dim2의 크기 : 2
dim3의 크기 : 3
```

ndarray link 확인해 보기

- https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html

[실습1] numpy를 이용하여 배열을 만들어보자.

- 3 x 3 열의 0의 값을 갖는 배열

```
[0 0 0]
[0 0 0]
[0 0 0]
```

- 3 x 3 열의 1의 값을 갖는 배열

```
[1 1 1]
[1 1 1]
[1 1 1]
```

numpy의 zeros() 함수를 사용하여 0의 값으로 채우기

In [7]:
```
# 전부 0으로 채운다.
zero_array = np.zeros( (3,3) , dtype="int64" )  # dtype은 기본값으로 float64
print(zero_array)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

In [8]:
```
# 전부 1으로 채운다.
one_array = np.ones( (3,3) )  # dtype은 기본값으로 float64
print(one_array)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

- 2 x 4 열의 1씩 증가하는 값을 갖는 배열

```
[1 2 3 4]
[5 6 7 8]
```

- 4 x 2 열의 1의 값을 갖는 배열

```
[1 2]
[3 4]
[5 6]
[7 8]
```

np.arange() 함수를 사용한 일정한 간격을 갖는 배열 만들기

In [9]:
```
# 값을 하나씩 증가시킨다.
# 아래 세 가지 방법은 전부 동일 값을 출력한다.
val1 = np.arange(start=0, stop=8, step=1)
print( val1 )

val2 = np.arange(0, 8, 1)
print( val2 )

val3 = np.arange(8)
print( val3 )
```

```
[0 1 2 3 4 5 6 7]
[0 1 2 3 4 5 6 7]
[0 1 2 3 4 5 6 7]
```

In [10]:
```
# 1~8의 값
val = np.arange(1,9)
print( val )

# 2 x 4 열의 2차원 배열
val.reshape(2,4)
```

```
[1 2 3 4 5 6 7 8]
```

Out[10]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

실습 2

- 4 x 2열을 만들어보기

다른 방법 열만 정하고, 행은 알아서 하도로 하기

In [12]:
```
# 2 x 4 열의 2차원 배열
val.reshape(-1,2)
```

Out[12]:
```
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

행의 값 하나에 접근하기

In [13]:
```
a = [1,2,3,4]
print(a[0])
print(a[3])
print(a[-2])
```

```
1
4
3
```

행 여러개의 값에 접근하기

In [14]:
```
a = [1,2,3,4]
print(a[1:3])  # 위치(1,2) 첫번째, 두번째 값 가져오기
print(a[1:])   # 두번째(1)부터 끝까지
print(a[:])    # 전체
print(a[-2:])  # 맨마지막 값에서 2번째부터 끝까지
```

```
[2, 3]
[2, 3, 4]
[1, 2, 3, 4]
[3, 4]
```

In [15]:
```
val1 = np.arange(start=0, stop=8, step=1)
print( val1 )

# 2 x 4 열의 2차원 배열
val2 = val1.reshape(-1,2)
val2
```

```
[0 1 2 3 4 5 6 7]
```

Out[15]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
```

In [16]:
```
print(val2[1])     # 2행에 접근
print(val2[1][1])  # 2행2열 값에 접근
print(val2[3][0])  # 4행1열 값에 접근
print(val2[0:2])   # 2행, 3행에 접근
```

```
[2 3]
3
6
[[0 1]
 [2 3]]
```

- last update : 24/06/10 내용 추가 및 일부 코드 변경
- @ by DJ, Lim