Big Data

NumPy

Python

모듈(라이브러리)

- 확장자가 .py인 파일
- 함수, 클래스 혹은 변수들을 모아놓은 파일.
- 파이썬 프로그램에서 불러와 사용할 수 있게끔 만들어진 파이썬 파일이라고도 한다.

Python

분석에 특화된 모듈(라이브러리)

- NumPy
  - 고성능 과학계산을 위한 데이터 분석 라이브러리
- Pandas
  - 행과 열로 구성된 표 형식의 데이터를 지원하는 라이브러리
- Matplotlib
  - 2D 그래프로 시각화가 가능한 라이브러리

NumPy의 주요 기능

- 빠르고 효율적인 벡터 산술연산을 제공하는 다차원배열 제공 (ndarray 클래스)
- 반복문 없이 전체 데이터 배열 연산이 가능한 표준 수학 함수 (sum(), sqrt(), mean())
- 선형대수, 난수 생성, 푸리에 변환

Numpy Basic

모듈(라이브러리) 사용하기

```
1 import numpy as np
```

- Numpy 모듈(라이브러리)를 import하고 앞으로 np라는 이름으로 부른다.

numpy.ndarray 클래스

- 동일한 자료형을 가지는 값들이 배열 형태로 존재함
- N 차원 형태로 구성이 가능하다
- 각 값들은 양의 정수로 색인(index)이 부여되어 있다
- ndarray를 줄여서 array로 표현한다

array생성 하기 : 1차원

```
1 list = [1,2,3,4,5]
2 list
```

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

```
1 arr = np.array(list)
2 arr
```

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

or

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

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

array생성 하기 : 2차원

```
1 arr2 = np.array([[1,2,3],[4,5,6]])
2 arr2
```

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

배열의 크기 확인하기

```
1 arr
```

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

```
1 arr.shape
```

```
(5,)
```

```
1 arr2
```

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

```
1 arr2.shape
```

```
(2, 3)
```

배열의 전체 요소 개수 확인하기

```
1 print(arr)
2 print(arr.size)
```

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

```
1 print(arr2)
2 print(arr2.size)
```

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

배열의 타입 확인

```
1 print(arr)
2 print(arr.dtype)
```

```
[1 2 3 4 5]
int32
```

```
1 print(arr2)
2 print(arr2.dtype)
```

```
[[1 2 3]
 [4 5 6]]
int32
```

배열의 차원(Dimension) 확인

```
1 print(arr)
2 print(arr.ndim)
```

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

```
1 print(arr2)
2 print(arr2.ndim)
```

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

특정한 값으로 배열 생성하기

0으로 초기화

```
1 arr_zeros = np.zeros((3,4))
2 arr_zeros
```

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

1로 초기화

```
1 arr_ones = np.ones((3,4))
2 arr_ones
```

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

특정한 값으로 배열 생성하기

```
1 arr_full = np.full((5,5),7)
2 arr_full
```

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

1,2,3,…,50이 담긴 배열 생성하기

```
1 arr = np.arange(1,51)
2 arr
```

```
array([ 1,  2,  3, ..., 48, 49, 50])
```

```
1 arr = np.arange(1,51,10)
2 arr
```

```
array([ 1, 11, 21, 31, 41])
```

타입 지정하여 배열 생성하기

```
1 arr_type = np.array([1.2,2.3,3.4], dtype=np.int64)
2 arr_type
```

```
array([1, 2, 3], dtype=int64)
```

타입변경하기

```
1 arr_type = arr_type.astype("float64")
2 arr_type
```

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

```
1 arr_type.dtype
```

```
dtype('float64')
```

랜덤 값 배열 생성하기

```
1 arr = np.random.rand(2,3)
2 arr
```

```
array([[0.49813944, 0.13250847, 0.90819973],
       [0.43032648, 0.228296  , 0.12117948]])
```

랜덤 값 배열 생성하기

```
1 arr = np.random.randint(2,10)
2 arr
```

```
2
```

```
1 arr = np.random.randint(2,10, size=(2,3))
2 arr
```

```
array([[3, 5, 9],
       [8, 2, 2]])
```

Array Operation

array 연산 (요소별 연산)

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

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

```
1 arr+arr
```

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

```
1 arr*arr
```

```
array([1, 4, 9])
```

Indexing & Slicing

array 인덱싱

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

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

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

```
[1 2 3]
```

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

```
1
```

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

```
1
```

1차원 array 슬라이싱

```
arr1 = np.arange(10)
arr1
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
arr1[3:8]
```

```
array([3, 4, 5, 6, 7])
```

```
arr1[3:8] = 12
```

```
arr1
```

```
array([ 0,  1,  2, 12, 12, 12, 12, 12,  8,  9])
```

2차원 array 생성

```
arr2 = np.arange(50).reshape(5,10)
arr2
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49]])
```

2차원 array 슬라이싱

```
arr2[:2,:]
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]])
```

```
arr2[:,0]
```

```
array([ 0, 10, 20, 30, 40])
```

2차원 array 슬라이싱

```
arr2[:4,:5]
```

```
array([[ 0,  1,  2,  3,  4],
       [10, 11, 12, 13, 14],
       [20, 21, 22, 23, 24],
       [30, 31, 32, 33, 34]])
```

```
arr2[2,1]
```

```
21
```

```
arr2[[2,3],[2,3]]
```

```
array([22, 33])
```

Boolean 색인

```
1 name_score = np.array(np.random.randint(50,100, size=8))
2 name_score
```

```
array([68, 57, 81, 52, 76, 88, 71, 88])
```

```
1 name_score>=80
```

```
array([False, False,  True, False, False,  True, False,  True])
```

```
1 name_score[name_score>=80]
```

```
array([81, 88, 88])
```

Universally Function

sum 함수

```
1 arr = np.random.randint(1,10,size=(4,7))
2 arr
```

```
array([[7, 9, 4, 6, 4, 6, 8],
       [7, 2, 6, 8, 9, 6, 8],
       [5, 5, 3, 7, 6, 2, 7],
       [8, 7, 1, 8, 6, 5, 5]])
```

```
1 print(arr.sum())
2 print(np.sum(arr))
```

```
15
15
```

mean 함수

```
1 arr = np.random.randint(1,10,size=(4,7))
2 arr
```

```
array([[7, 9, 4, 6, 4, 6, 8],
       [7, 2, 6, 8, 9, 6, 8],
       [5, 5, 3, 7, 6, 2, 7],
       [8, 7, 1, 8, 6, 5, 5]])
```

```
1 print(arr.mean())
2 print(np.mean(arr))
```

```
3.0
3.0
```

abs 함수

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

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

```
1 np.abs(arr)
```

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

Universally 함수

단일 배열에 사용하는 함수

함수	설명
abs, fabs	각 원소의 절대값을 구한다. 복소수가 아닌 경우에는 fabs로 빠르게 연산가능
sqrt	제곱근을 계산 arr ** 0.5와 동일
square	제곱을 계산 arr ** 2와 동일
Exp	각 원소에 지수 e^x를 계산
Log, log10, log2, logp	각각 자연로그, 로그10, 로그2, 로그(1+x)
sign	각 원소의 부호를 계산
ceil	각 원소의 소수자리 올림
floor	각 원소의 소수자리 버림
rint	각 원소의 소수자리 반올림. dtype 유지
modf	원소의 몫과 나머지를 각각 배열로 반환
isnan	각 원소가 숫자인지 아닌지 NaN 나타내는 불리언 배열
isfinite, isinf	배열의 각 원소가 유한한지 무한한지 나타내는 불리언 배열
cos, cosh, sin, sinh, tan, tanh	일반 삼각함수와 쌍곡삼각 함수
logical_not	각 원소의 논리 부정(not) 값 계산. -arr와 동일

Universally 함수

서로 다른 배열 간에 사용하는 함수

함수	설명
add	두 배열에서 같은 위치의 원소끼리 덧셈
subtract	첫번째 배열 원소 – 두번째 배열 원소
multiply	배열의 원소끼리 곱셈
divide	첫번째 배열의 원소에서 두번째 배열의 원소를 나눗셈
power	첫번째 배열의 원소에 두번째 배열의 원소만큼 제곱
maximum, fmax	두 원소 중 큰 값을 반환. fmax는 NaN 무시
minimum, fmin	두 원소 중 작은 값 반환. fmin는 NaN 무시
mod	첫번째 배열의 원소에 두번째 배열의 원소를 나눈 나머지
greater, greater_equal, less, less_equal, equal, not_equal	두 원소 간의 〉, 〉=, 〈, 〈=, ==, != 비교연산 결과를 불리언 배열로 반환
logical_and, logical_or, logical_xor	각각 두 원소 간의 논리연산, &, |, ^ 결과를 반환

10명에 대한 키와 몸무게가 들어있는 파일
'height_weight.txt'을 읽어 각 사람별 BMI 지수를 구하시오.

```
1 data = np.loadtxt("height_weight.txt", delimiter=",")
2 data
```

```
array([[175.2, 180.3, 175. , ..., 178.2, 177. , 179. ],
       [ 65.6,  88. ,  79.2, ...,  68.9,  74. ,  82. ]])
```

$$\text{BMI 지수} = \frac{\text{몸무게(kg)}}{\text{키(m)} \times \text{키(m)}}$$

결과 값

```
array([21.37153104, 27.07018468, 25.86122449, 24.20652885, 16.03543423,
       20.14486193, 23.14392095, 21.69720651, 23.62028791, 25.59220998])
```

GiGA WiFi

"Big Data"를 해제하였습니다.

다음시간에 배울내용

Pandas