5. 넘파이 (Numpy)

자료 구조란?

- 문제 해결을 위해 많은 데이터를 효율적으로 저장하고, 처리하고, 관리할 수 있도록 자료들 사이의 관계를 나타내는 것.

- 많은 양의 데이터를 효율적으로 저장 및 처리 하려면 그에 적합한 자료 구조 설계 및 활용 능력이 필요함.

파이썬에서 제공하는 기본 자료 구조

- 리스트(List)
  - 여러 가지 원소를 하나의 묶음으로 표현할 수 있는 자료형
- 튜플(Tuple)
  - 리스트와 비슷하나, 저장된 원소를 변경 또는 삭제할 수 없음
- 집합(Set)
  - 수학에서의 집합 개념의 자료형
  - 원소의 중복이 허용되지 않으며, 원소들 간의 순서가 없음
- 사전(Dictionary)
  - 키와 값의 쌍으로 구성된 원소를 표현할 수 있는 자료형

튜플 (tuple)

- 튜플은 리스트와 비슷하지만 다음과 같은 다른 점이 있다.
  - 리스트는 [ ]으로 둘러싸지만 튜플은 ( )으로 둘러싼다.
  - 리스트는 그 값의 생성, 삭제, 수정이 가능하지만 튜플은 그 값을 바꿀 수 없다.

=> 프로그램이 실행되는 동안 값이 변하지 않기를 바라거나

값이 바뀔까 걱정하고 싶지 않다면 튜플을 사용

튜플의 사용 예시

```
>>> t1 = (1, 2, 3)
>>> print(t1)
(1, 2, 3)
>>> type(t1)
<class 'tuple'>
>>> t2 = (2022001, '신사임당', (100, 90, 95))
>>> print(t2)
(2022001, '신사임당', (100, 90, 95))
```

```
>>> print(t1[0], t1[1], t1[2])
1 2 3
>>> print(t2[0], t2[1], t2[2])
2022001 신사임당 (100, 90, 95)
>>> t1[0] = 100
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in <module>
    t1[0] = 100
TypeError: 'tuple' object does not support item assignment
```

집합 (set)

- 집합은 집합에 관련된 것을 쉽게 처리하기 위한 자료형
  - 중복을 허용하지 않는다.
  - 순서가 없다(Unordered).

=> 리스트나 튜플은 순서가 있기(ordered) 때문에 인덱싱을 통해 자료형의 값을 얻을 수 있지만 집합과 사전 자료형은 순서가 없기(unordered) 때문에 인덱싱으로 값을 얻을 수 없다.

집합의 사용 예시 (1/2)

```
>>> s1 = set([3,2,1,2,3])
>>> print(s1)
{1, 2, 3}
>>> type(s1)
<class 'set'>
>>> s2 = set("Hello")
>>> print(s2)
{'o', 'H', 'e', 'l'}
```

```
>>> print(s1[0])          # 인덱스 접근 불가
Traceback (most recent call last):
  File "<pyshell#66>", line 1, in <module>
    print(s1[0])
TypeError: 'set' object does not support indexing
>>> lst1 = list(s1)       # 리스트로 자료형 변환
>>> print(lst1)
[1, 2, 3]
>>> print(lst1[0])
1
```

집합의 사용 예시 (2/2)

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])
>>> s1.intersection(s2)          # 교집합
{4, 5, 6}
>>> s1.union(s2)                 # 합집합
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> s1.difference(s2)            # 차집합
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)                    #값 1개 추가
>>> print(s1)
{1, 2, 3, 4}
>>> s1.update([3, 4, 5, 6])      #값 여러 개 추가
>>> print(s1)
{1, 2, 3, 4, 5, 6}
>>> s1.remove(2)                 #특정 값 제거
>>> print(s1)
 {1, 3, 4, 5, 6}
```

Numpy (넘파이)

- Numpy 는 계산과학분야 프로그램을 개발할 때 핵심 역할을 하는 라이브러리
  - Numerical Python 의 줄임말
  - Python에서 벡터, 행렬 등 수치 연산을 지원하고 빠른 속도로 수행됨
- **설치 방법** : pip install numpy
- **라이브러리 사용법** : import numpy as np

배열 (Array)

- 넘파이에서 제공하는 자료형 : 배열
  - 같은 자료형 값들을 저장하는 자료구조
  - 리스트는 각 요소가 다른 자료형 이 될 수 있음
  - 배열은 다수의 요소를 연산하고 저장하기에 효율적
  - 리스트에서 사용할 수 없는 (브로드캐스트) 연산 방식 지원
  - 추후 머신러닝 등에서 사용되는 주요 자료구조 임

넘파이 (1차원) 배열 생성 예시

```
import numpy as np
arr1 = np.array( [0, 2, 5, 7] )
print( type(arr1) )
print(arr1)
print()
print( arr1.ndim )   # 차원수
print( arr1.shape )  # 각 차원의 크기
print( arr1.dtype )  # 요소 자료형
print( arr1.size )    # 전체 요소 개수
```

```
<class 'numpy.ndarray'>
[0  2  5  7]

1
(4,)
int32
4
```

- ndarray.ndim : **차원의 수**(Dimension)
- ndarray.shape : **배열의 각 차원의 크기**
- ndarray.dtype : **각 요소**(Element)**의 타입**
- ndarray.size : **전체 요소**(Element)**의 개수**

넘파이 (2차원) 배열 생성 예시

```
import numpy as np
arr2 = np.array( [ [1, 2, 3,], [4, 5, 6] ] )
print( type(arr2) )
print(arr2)
print()
print( arr2.ndim )   # 차원수
print( arr2.shape )  # 각 차원의 크기
print( arr2.dtype )  # 요소 자료형
print( arr2.size )     # 전체 요소 개수
```

```
<class 'numpy.ndarray'>
[[1 2 3]
 [4 5 6]]

2
(2, 3)
int32
6
```

- ndarray.ndim : **차원의 수**(Dimension)
- ndarray.shape : **배열의 각 차원의 크기**
- ndarray.dtype : **각 요소**(Element)**의 타입**
- ndarray.size : **전체 요소**(Element)**의 개수**

넘파이 배열 생성 함수 : numpy.zeros()

```
import numpy as np
arr = np.zeros(5)
print(type(arr))
print(arr)
print( arr.ndim )
print( arr.shape )
print( arr.dtype )
print( arr.size )
```

```
<class 'numpy.ndarray'>
[0. 0. 0. 0. 0.]
1
(5,)
float64
5
```

```
import numpy as np
arr = np.zeros( (4,2) )
print(type(arr))
print(arr)
print( arr.ndim )
print( arr.shape )
print( arr.dtype )
print( arr.size )
```

```
<class 'numpy.ndarray'>
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
2
(4, 2)
float64
8
```

넘파이 배열 생성 함수 : numpy.ones()

```
import numpy as np
arr = np.ones(5)
print(type(arr))
print(arr)
print( arr.ndim )
print( arr.shape )
print( arr.dtype )
print( arr.size )
```

```
<class 'numpy.ndarray'>
[1. 1. 1. 1. 1.]
1
(5,)
float64
5
```

```
import numpy as np
arr = np.ones( (4,2) )
print(type(arr))
print(arr)
print( arr.ndim )
print( arr.shape )
print( arr.dtype )
print( arr.size )
```

```
<class 'numpy.ndarray'>
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
2
(4, 2)
float64
8
```

넘파이 배열 생성 시 요소 자료형 지정

```
import numpy as np
arr = np.ones(5, dtype=int)
print(type(arr))
print(arr)
print( arr.ndim )
print( arr.shape )
print( arr.dtype )
print( arr.size )
```

```
<class 'numpy.ndarray'>
[1 1 1 1 1]
1
(5,)
int32
5
```

넘파이 배열 생성 함수 : numpy.arange()

```
import numpy as np
arr = np.arange(5)
print(type(arr))
print(arr)
print( arr.ndim )
print( arr.shape )
print( arr.dtype )
print( arr.size )
```

```
<class 'numpy.ndarray'>
[0 1 2 3 4]
1
(5,)
int32
5
```

```
import numpy as np
arr = np.arange(3, 10, 2)
print(type(arr))
print(arr)
print( arr.ndim )
print( arr.shape )
print( arr.dtype )
print( arr.size )
```

```
<class 'numpy.ndarray'>
[3 5 7 9]
1
(4,)
int32
4
```

넘파이 차원 변경하기 : numpy.reshape()

```
import numpy as np
arr = np.arange(12)
print(arr)
print(arr.ndim)
print(arr.shape)
print()
arr2 = arr.reshape(3, 4)
print(arr2)
print(arr2.ndim)
print(arr2.shape)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
1
(12,)

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
2
(3, 4)
```

1차원 배열을 2차원 으로 변형

넘파이 차원 변경하기 : numpy.reshape()

```
import numpy as np
arr = np.arange(12) .reshape(3, 4)
print(arr)
print(arr.ndim)
print(arr.shape)
print()
arr2 = arr.reshape(-1)
print(arr2)
print(arr2.ndim)
print(arr2.shape)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
2
(3, 4)

[ 0  1  2  3  4  5  6  7  8  9 10 11]
1
(12,)
```

2차원 배열을 1차원 으로 변형

넘파이 차원 변경하기 : numpy.reshape()

```
import numpy as np
arr = np.arange(12)
print(arr)
print(arr.ndim)
print(arr.shape)
print()
arr2 = arr.reshape(-1, 2)
print(arr2)
print(arr2.ndim)
print(arr2.shape)
```

열의 크기만 지정, 행의 크기는 자율배정

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
1
(12,)

[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]]
2
(6, 2)
```

넘파이 차원 변경하기 : numpy.reshape()

```
import numpy as np
arr = np.arange(12)
print(arr)
print(arr.ndim)
print(arr.shape)
print()
arr2 = arr.reshape(4, -1)
print(arr2)
print(arr2.ndim)
print(arr2.shape)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
1
(12,)

[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
2
(4, 3)
```

행의 크기만 지정, 열의 크기는 자율배정

넘파이 요소 값 접근 및 변경

```
import numpy as np
arr = np.arange(5)
print( arr )
print( arr[0],  arr[1] )   # index 로 접근
print()
arr[0] = 777
print( arr )
arr[1] = "hi"
```

```
[0 1 2 3 4]
0 1
[777 1 2 3 4]
Traceback (most recent call last):
  File "C:\...\my_first.py", line 8, in <module>
    arr[1] = "hi"
ValueError: invalid literal for int() with base 10: 'hi'
```

넘파이 배열 요소는 자료형이 같아야 함

넘파이 요소 값 접근 및 변경

```
import numpy as np
arr = np.arange(5)
print(arr)
print(arr[0], arr[1])
arr[0] = 777
print(arr)
arr[1] = 3.14
print(arr)
```

```
[0 1 2 3 4]
0 1
[777   1   2   3   4]
[777   3   2   3   4]
```

넘파이 배열 요소는 자료형이 같아야 함

다차원 인덱싱 & 슬라이싱

```
import numpy as np
arr = np.arange(20).reshape(4, 5)
print( arr )
print()
print( arr[0] )              # 행 인덱싱
print( arr[0][1:3] )         # 특정 행에 대한 슬라이싱
print( arr[2][1] )           # 개별 요소 인덱싱
print()
print( arr[2, 1] )           # 개별 요소 인덱싱
print( arr[ [0, 2], [1, 3] ] )   # 여러 요소 인덱싱
# (0,1) 과  (2,3) 을 접근함
```

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

[0 1 2 3 4]
[1 2]
11

11
[ 1 13 ]
```

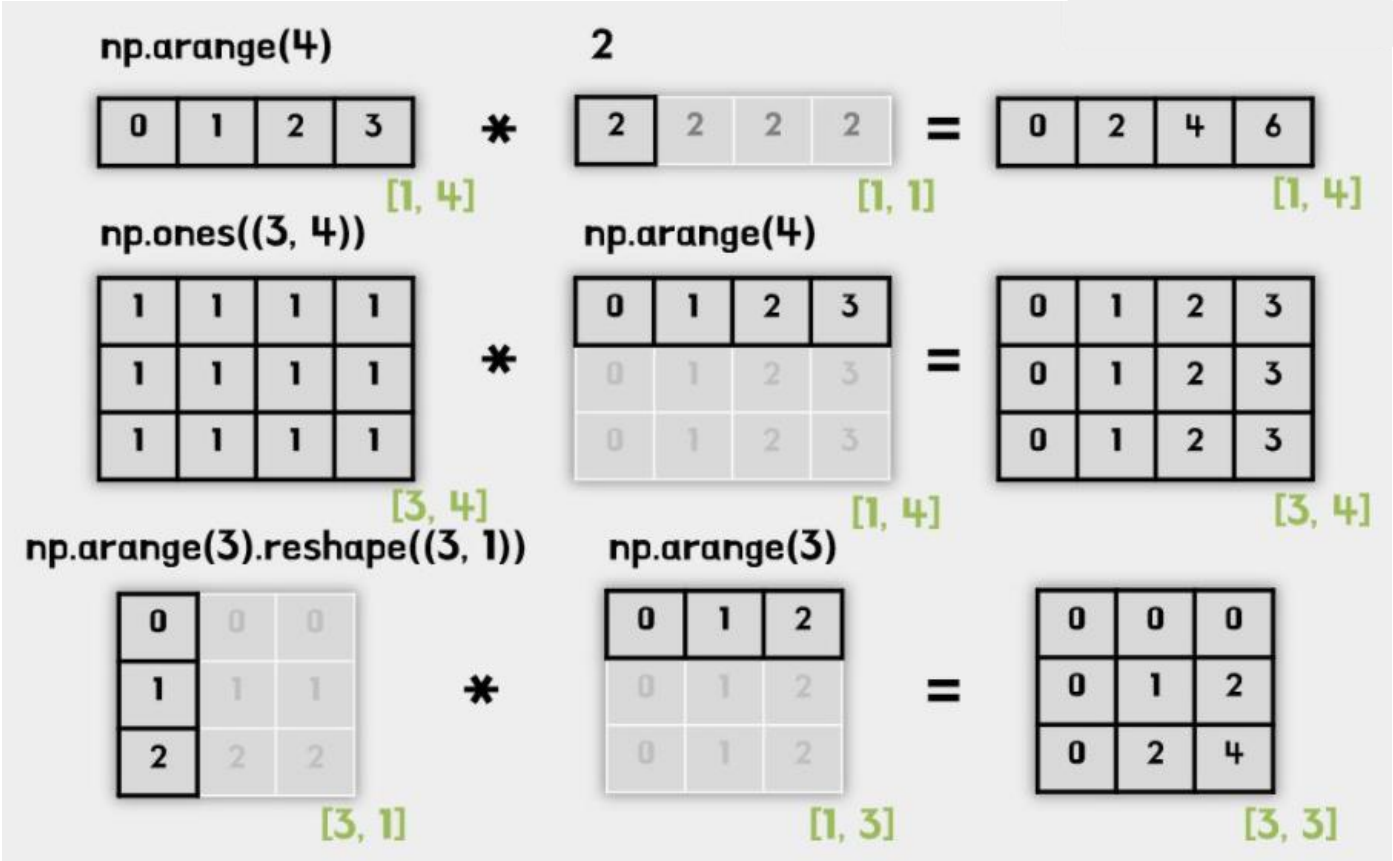
다차원 인덱싱 & 슬라이싱

```
import numpy as np
arr = np.arange(20).reshape(4, 5)
print( arr )
print()
print( arr[1:3] )       # 행 슬라이싱
print()
print( arr[1:3, 1:4] ) # 행과 열 슬라이싱
print()
print( arr[:, 1:3] )    # 열만 슬라이싱
```

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

[[ 5  6  7  8  9]
 [10 11 12 13 14]]

[[ 6  7  8]
 [11 12 13]]

[[ 1  2]
 [ 6  7]
 [11 12]
 [16 17]]
```

넘파이 브로드캐스팅

- Shape 이 다른 넘파이 배열 끼리 연산을 수행



(브로드캐스팅) 리스트와 넘파이 차이

```
import numpy as np
arr = np.arange(5)
print( arr )
print()
arr2 = arr * 2
print( arr2)
```

```
[0 1 2 3 4]

[0 2 4 6 8]
```

넘파이 배열 각 요소에 산술적 곱셈이 적용

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

```
lst = [0, 1, 2, 3, 4]
print( lst )
print()
lst2 = lst * 2
print( lst2 )
```

```
[0 1 2 3 4]

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

리스트 전체가 반복 복사됨

브로드캐스팅 예시

- 학생들의 [국어, 영어, 수학] 점수가 주어졌을 때 수학 시험에 오류가 있어 모두 +5 점을 주고 싶다면?

```
scores = np.array( [ [83, 65, 57], [90, 71, 64], [84, 83, 59], [83, 72, 44], [78, 66, 67] ] )
print(scores)
print()
print(scores[:, 2])    # 세 번째 열 (수학) 선택하기
scores[:, 2] = scores[:, 2] + 5
print()
print(scores)
```

```
[[83 65 57]
 [90 71 64]
 [84 83 59]
 [83 72 44]
 [78 66 67]]

[57 64 59 44 67]

[[83 65 62]
 [90 71 69]
 [84 83 64]
 [83 72 49]
 [78 66 72]]
```

넘파이 사용 예시

- 학생들의 [국어, 영어, 수학] 점수가 주어졌을 때 학생 별 평균 구하기

```
scores = np.array( [ [83, 65, 57], [90, 71, 64], [84, 83, 59], [83, 72, 44], [78, 66, 67] ] )
print(scores)
print()
num_students = scores.shape[0]      # 배열 행의 크기
for i in range(num_students):
    print( np.mean(scores[i]) )          # np.mean( )  numpy 제공 함수
```

```
[[83 65 57]
 [90 71 64]
 [84 83 59]
 [83 72 44]
 [78 66 67]]

68.33333333333333
75.0
75.33333333333333
66.33333333333333
70.33333333333333
```

* 그렇다면 과목별 평균은 어떻게 구할까요?

https://numpy.org/doc/stable/reference/routines.statistics.html