# NumPY

- 대규모 다차원 배열을 쉽게 처리할 수 있는, 수학 및 과학 연산을 위한 파이썬 라이브러리
- ndarray를 데이터 관리를 위한 기본 단위로 사용
- 브로드캐스팅 연산

In [ ]:

```
import numpy as np
from pprint import pprint
import matplotlib.pyplot as plt
```

## Numpy Array 생성

- np.asarray(list):: list를 numpy ndarray로 변경
- np.arange(start, end, step):: 해당 범위의 numpy ndarray를 생성
- np.linspace(start, end, num_of_points):: start와 end사이에서 num_of_points개의 균일한 차이값을 가지는 ndarray 생성

In [ ]:

```
# list 생성
list_ = [1, 2, 3]

# python 내장함수 type()으로 list_의 타입 확인 :: list
print(type(list_))

# np.asarray() 함수로 list_의 타입을 list에서 numpy.ndarray로 변경
print(np.asarray(list_))      #  [1 2 3]

# python 내장함수 type()으로 list_의 타입 확인 :: ndarray
print(type(np.asarray(list_)))

# np.arange(start, end, step), python range와 비슷하게 작동하며
# start와 end사이에서 step 만큼 수를 건너뛰며 array를 생성
# range의 범위 start <= number < end
print(np.arange(1,10,2))   #  [1 3 5 7 9]

# np.arange()로 생성한 변수의 타입 확인 :: ndarray
arange_array = np.arange(1,10,2)
print(type(arange_array))

# np.linspace는 start와 end 사이에서 num_of_point개의 수를 가지는 array를 생성하며
# 범위는 start <= number <= end 이며, 각각의 수는 동일한 차이를 가진다.
linspace_array = np.linspace(0,10,5)
print(linspace_array) #  [ 0.   2.5  5.   7.5 10. ]

# np.linspace()로 생성한 변수의 타입 확인 :: ndarray
print(linspace_array)
```

# 문제

1. 1~100 사이의 짝수만 가지는 array 생성
2. np.linspace로 [0, 2.5, 5, 7.5, 10, 7.5, 5, 2.5, 0]을 요소로 가지는 array 생성, hint: np.append(arr1, arr2)

In [ ]:

```
# 문제1-1
# 1 ~ 100 사이의 짝수만 가지는 array를 생성하는 방법은 다양하지만
# 가장 간단한 방법은 np.arange()를 사용해 2~101까지의 범위에서
# 2씩 건너뛰며 생성하는 방법이다.
even_array = np.arange(2, 101, 2)
print(even_array)

# 문제1-2
# np.linspace()를 사용해서 2개의 array를 만들고 np.append()로 두개의
# array를 하나로 합치는게 가장 간단한 방법이다.
# 가장 쉬운 방법으로는 2가지 방법이 있으며
# 첫번째는 아래와 같이 slicing을 사용해 첫번째 수를 무시하고 두개의
# array를 합치는 방법이며
arr_1 = np.linspace(0, 10, 5)
arr_2 = np.linspace(10, 0, 5)[1:]
array_1 = np.append(arr_1, arr_2)
print(array_1)

# 두번째 방법은 아래와 같이 범위를 7.5~0까지고 설정하고 4개의 수만
# 사용하는 방법이다.
arr_1 = np.linspace(0, 10, 5)
arr_2 = np.linspace(7.5, 0, 4)
array_2 = np.append(arr_1, arr_2)
print(array_2)
```

## Numpy Array 생성

- np.ones(shape, type):: shape과 type을 맞추고 1로 초기화된 ndarray를 생성
- np.zeros(shape, type):: shape과 type을 맞추고 0로 초기화된 ndarray를 생성
- np.eye(rows, cols):: 대각행렬만 1이고 나머지는 0인 ndarray 생성
- np.diag(array):: array에서 대각행렬만 추출

In [ ]:

```
# shape은 array의 형태를 말하며, 읽는 법은 뒤에서 부터 앞으로
# 읽으면 편하게 해석할 수 있다.
# 예를 들어 shape이 (2, 3, 2)이라면,
# 제일 뒤의 2는 [1, 2] 처럼 제일 안쪽의 array에 수가 2개 들어있다는 뜻이며
# 중간의 3은 [[1, 2], [1, 2], [1, 2]] 처럼 2개의 수가 들어있는 array가
# 총 3개가 있다는 뜻이다.
# 처음의 2는 [[[1, 2], [1, 2], [1, 2]], 처럼 2개의 수가 들어있는 array가
#             [[1, 2], [1, 2], [1, 2]] 3개가 있으며, 그것이 2개가 있다는 뜻이다.

# np.ones()는 설정한 shape에 맞게 array를 생성하며, 모든 값을 1로 초기화 한다.
print(np.ones((2,3,6)))

# np.zeros()는 설정한 shape에 맞게 array를 생성하며, 모든 값을 0으로 초기화 한다.
print(np.zeros((2,6)))     #  [[0. 0. 0. 0. 0. 0.]
                           #   [0. 0. 0. 0. 0. 0.]]

# np.eye()는 shape에 맞게 사각행렬을 생성하고, (0,0), (1,1)처럼
# 대각의 원소들을 1, 나머지는 0으로 초기화 한다.
print(np.eye(2, 2))        #  [[1., 0.]
                           #   [0., 1.]]

# np.diag()는 주어진 array에서 (0,0), (1,1) 처럼
# 대각의 원소들만 추출하는 함수이다.
print(np.diag(np.eye(2, 2))) #  [1. 1.]
```

# 문제

1. (5, 5)의 shape을 가지고 대각행렬이 3이고 나머지는 0 array 생성
2. (2, 5)의 shape을 가지고 arr[0]는 홀수번째 값이 1, arr[1]은 짝수번째 값이 1인 array 생성, (단, index는 1부터라 가정) hint: for문 사용

In [ ]:

```
# 문제2-1
# 대각행렬을 생성할것이 때문에 np.eye()를 사용해 주어진 shape으로 생성하면 되고,
# numpy는 브로드캐스팅을 지원하기 때문에 생성한 array에 바로 3을 곱해주면 된다.
array = np.eye(5, 5) * 3
print(array)

# 문제2-2
# 먼저 이 문제의 결과는 다음과 같이 출력된다.
# [[1, 0, 1, 0, 1]    ::  위
#  [0, 1, 0, 1, 0]]   ::  아래

# 먼저 np.zeros()를 통해 shape이 (2,5)인 array를 생성한다.
array = np.zeros((2, 5))

# 첫번째 for문은 array의 행의 수만큼 반복한다.
for i in range(len(array)):

# 두번째 for문은 array 안의 수의 개수만큼 반복한다.
    for j in range(len(array[i])):

# if문은 현재 확인하는 array가 위인지 아래인지 판단하기 위해 i%2를 사용하며
# array내부의 숫자가 홀수 번째인지 짝수 번째인지 판단하기 위해 j%2를 사용한다.
        if i % 2 == 0 and j % 2 == 0:
            array[i][j] = 1
        elif i % 2 == 1 and j % 2 == 1:
            array[i][j] = 1

print(array)
```

## Numpy Array 생성

- np.random.rand(shape):: shape을 맞추고 uniform distribution을 따르는 ndarray 생성
- np.random.randn(shape):: shape을 맞추고 normal distribution을 따르는 ndarray 생성
- np.random.normal(loc, scale, size):: loc(분포의 중심)을 기준으로 scale 표준편차 범위에서 정규 분포의 형상을 같는 난수를 size만큼 생성

In [ ]:

```
# np.random.rand() 함수와 np.random.randn() 함수는
# shape의 형태를 가지는 array를 생성하고, 값을 랜덤으로 초기화한다.
rand_1 = np.random.rand(2, 6)
print(rand_1.shape) # (2, 6)
print(rand_1)
rand_2 = np.random.randn(3, 2)
print(rand_2.shape) # (3, 2)
print(rand_2)

# np.random.normal(loc, scale, size) 함수는 2차원 그래프로 생각할때
# x = loc을 기준으로 scale 를 표준 편차로 하고 정규 분포의 형상을 같는
# 랜덤한 수를 size만큼 생성한다.
rand_norm = np.random.normal(0, 0.1, 10000)
plt.hist(rand_norm, 100)
plt.show()
```

# 문제

1. np.random.randn을 사용해 (10)의 랜덤 array를 만들고 0.5에 가장 가까운 수 출력하기, hint: np.abs(), 브로드캐스팅
   np.abs(arr) :: arr에 절대값을 취하는 함수

In [ ]:

```
# 문제 3-1

# 먼저 np.random.randn() 함수로 (10)의 shape을 가지는 랜덤 array를 생성한다.
arr = np.random.randn(10)

# 다음으로 array에서 0.5를 뺀 뒤 절대값 함수를 취한 새로운 array를 생성한다.
# 새로 생성된 array에서 0에 가까운 수일수록 기존의 array에서는 0.5에 가깝다.
# 따라서 새로 생성된 array에서 최소값을 찾고, 그 값의 index를 기존의 array에
# 입력하여 값을 출력하면 0.5에 가장 가까운 수가 출력된다.
temp_array = np.abs(arr-0.5)
#print(temp_array)

# minimum값을 찾는 방법은 다음과 같다.
mini = 10000
mini_index = 0

# for문을 사용해 temp_array에 포함된 수의 개수만큼 반복한다.
for i in range(len(temp_array)):
    # mini 의 값이 array에 있는 값보다 크다면 mini의 값이 최소값이 아니기
    # 때문에 값을 더 작은 수로 변경해주며, 그때의 index를 저장한다.
    if mini > temp_array[i]:
        mini = temp_array[i]
        mini_index = i

# 기존의 array를 출력하고, 기존 array의 mini_index 번째의 값을 출력해
# 0.5에 가까운 수인지 확인한다.
print(arr)
print(arr[mini_index])
```

## Array 확인 및 변환

- array.shape:: array의 shape을 가지고있는 속성
- array.reshape(shape): array의 기존 shape을 입력된 shape으로 변경후 반환
- array.T: array의 행과 열을 교환한 전치행렬을 반환

In [ ]:

```
# shape은 상기 설명한 것과 같으며,
# array.shape 은 array의 shape을 확인하는 방법이다.
array = np.random.randn(1, 3, 5)
print(array)
print(array.shape)

# reshape은 array의 형태를 바꾸는 것이다.
# reshape을 할때는 shape의 모든 값을 곱한 값이 동일해야 한다.
# 예를 들어 (2, 10)은 (1, 20), (4, 5), (5, 4) 처럼
# 곱한 값이 20인 shape으로만 reshape이 가능하다.
array_2 = array.reshape(3, 1, 5)
print(array_2.shape)

# array.T는 array의 행과 열을 교환한 전치행렬을 반환하는 함수이다.
# 예를 들어 [[1, 2]  라는 array가 있다면, 이 array의 전치행렬은 [[1, 3] 가 된다.
#           [3, 4]]                                              [2, 4]]
array_3 = np.arange(15).reshape(3, 5)
print(array_3)
print(array_3.T)
```

## Numpy Indexing & Slicing

- arr[x] for 1D array
- arr[x, y] for 2D array
- arr[x, y, z] for 3D array
- arr[::] for slicing
- arr**[x_start : x_end : x_step, y_start : y_end : y_step]**
- ! Slicing 은 복제가 아닌 동일 array의 일부
- Boolean mask 사용 가능

In [ ]:

```
arr1 = np.arange(5)

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

arr3 = np.asarray([[[1,2,3],
                    [4,5,6]],
                   [[7,8,9],
                    [10,11,12]]])

print(arr1.shape) # (5,)
print(arr2.shape) # (2, 3)
print(arr3.shape) # (2, 2, 3)


# numpy의 ndarray에서 indexing과 slicing은 간단하면서도 중요합니다.
# 다양한 shape의 array를 생성하신 다음, 양수와 음수 indexing을 확인해 보시고,
# 다양한 방법의 slicing을 사용해 출력값과 array를 비교해 보시면서
# 익숙해지는 방법 외에는 설명할 방법이 없습니다...
pprint(arr1) # array([0, 1, 2, 3, 4])
print(arr1[2]) # 2
print(arr1[2:4]) # [2 3]

pprint(arr2) # array([[1, 2, 3],
             #        [4, 5, 6]])
print(arr2[0, 1]) # 2
print(arr2[0, -2:]) # [2 3]
print(arr2[:, 0]) # [1 4]

pprint(arr3) # array([[[ 1,  2,  3],
             #         [ 4,  5,  6]],
             #        [[ 7,  8,  9],
             #         [10, 11, 12]]])
print(arr3[0]) # [[1 2 3]
               # [4 5 6]]
print(arr3[1, 0]) # [7 8 9]
print(arr3[0, 0, 0]) # 1
print(arr3[:, :, 0]) # [[ 1  4]
                     #  [ 7 10]]
print(arr3[:, 0, 1]) # [2 8]


temp_arr_1 = arr1[2:]
print(np.may_share_memory(arr1, temp_arr_1)) # True
print(temp_arr_1) # [2 3 4]
temp_arr_1[-1] = 999
print(arr1) # [  0   1   2   3 999]

temp_arr_2 = arr1[2:].copy()
print(np.may_share_memory(arr1, temp_arr_2)) # False
```

# 문제

1. np.arange()로 0~300을 요소로 가지는 array를 생성하고, 3의 배수만 추출하여 (25, 4)으로 shape을 변경한 array 생성 (0 제외), hint: bool operation, slicing
2. 1에서 생성한 array에서 모든 배열에서 처음과 마지막 값을 추출한 array 생성, hint:slicing

In [ ]:

```
# 문제 4-1
# np.arange()함수로 0~300을 가지는 array 생성
arr = np.arange(0, 301)

# 생성한 array에서 3으로 나눈 나머지가 0인것만 남기며,
# arr[0]에는 0이 있기때문에 slicing을 사용해 넘어갑니다.
arr_temp = arr[arr%3 == 0][1:]
print(arr_temp)

# 그리고 reshape을 통해 (25,4)로 shape을 변경합니다.
arr = arr_temp.reshape(25, 4)
print(arr)

# 문제 4-2
# 위에서 생성한 array에서 처음과 마지막 값을 추출하는 방법입니다.
# 왜 이렇게 되는지 실험해보시고 생각하보시면 좋을것 같습니다.
arr = arr[:,::3]
print(arr)
```

## Numpy with Operations

- List와는 다르게 수학적 연산을 BroadCasting으로 실행
- + x: arr의 모든 값에 x를 더함
- - x: arr의 모든 값에 x를 뺌
- * x: arr의 모든 값에 x를 곱합
- ** x: arr의 모든 값에 pos(x)를 취함
- == x: arr의 모든 값과 비교해 True/False array 반환
- > x: arr의 모든 값과 비교해 True/False array 반환
- < x: arr의 모든 값과 비교해 True/False array 반환

In [ ]:

```
# 이부분은 한번 확인만 해보시고 넘어가셔도 됩니다.
arr = np.arange(1, 11)
arr1 = np.arange(3, 8)
arr2 = np.arange(1, 11)

print(arr) # [ 1  2  3  4  5  6  7  8  9 10]

print(arr+5) #  [ 6  7  8  9 10 11 12 13 14 15]
print(arr-5) #  [-4 -3 -2 -1  0  1  2  3  4  5]
print(arr*5) #  [ 5 10 15 20 25 30 35 40 45 50]
print(arr**2) #  [  1   4   9  16  25  36  49  64  81 100]


print(arr1) # [3 4 5 6 7]
print(arr1==5) #  [False False  True False False]

print(arr1<5) #  [ True  True False False False]
print(arr1[arr1<5]) #  [3 4]

print(arr1>5) #  [False False False  True  True]
print(arr1[arr1>5]) #  [6 7]
```

## Numpy Simple Statistics

- np.sum(arr, axis): axis에 맞춰 array의 값을 모두 더함
- np.min(arr, axis): axis에 맞춰 array의 최소값을 반환
- np.argmin(arr, axis): axis에 맞춰 array의 최소값의 index를 반환
- np.max(arr, axis): axis에 맞춰 array의 최대값을 반환
- np.argmax(arr, axis): axis에 맞춰 array의 최대값의 index를 반환
- np.mean(arr, axis): axis에 맞춰 array의 평균값을 반환

In [ ]:

```
# 이부분은 한번 확인만 해보시고 넘어가셔도 됩니다.
arr = np.asarray([[4, 5, 6],
                  [3, 2, 1],
                  [9, 8, 7],
                  [-1, 10, 0]])
print(arr.shape) # (4, 3)

print(np.sum(arr, axis=0)) # [15 25 14]
print(np.sum(arr, axis=1)) # [15  6 24  9]

print(np.min(arr, axis=0)) # [-1  2  0]
print(np.min(arr, axis=1)) # [ 4  1  7 -1]

print(np.argmin(arr, axis=0)) # [3 1 3]
print(arr[3, 0], arr[1, 1], arr[3, 2])
print(np.argmin(arr, axis=1)) # [0 2 2 0]
print(arr[0, 0], arr[1, 2], arr[2, 2], arr[3, 0]) # 4 1 7 -1

print(np.max(arr, axis=0)) # [ 9 10  7]
print(np.max(arr, axis=1)) # [ 6  3  9 10]

print(np.mean(arr, axis=0)) # [3.75 6.25 3.5 ]
print(np.mean(arr, axis=1)) # [5. 2. 8. 3.]
```