

NumPy는 "Numerical Python"의 약자로 대규모 다차원 배열과 행렬 연산에 필요한 다양한 함수를 제공

- Numerical Python을 의미하는 NumPy는 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 대표적인 패키지
- 많은 머신러닝 알고리즘이 넘파이 기반으로 작성돼 있으며 알고리즘의 입출력 데이터를 넘파이 배열 타입으로 사용함
- 넘파이의 기본 데이터 타입은 ndarray. ndarray를 이용해 넘파이에서 다차원 배열을 쉽게 생성하고 다양한 연산 수행

NumPy 특징

- 강력한 N 차원 배열 객체
- 정교한 브로드캐스팅(Broadcast) 기능
- C/C++ 및 포트란 코드 통합 도구
- 유용한 선형 대수학, 푸리에 변환 및 난수 기능
- 푸리에 변환(Fourier transform, FT)은 시간이나 공간에 대한 함수를 시간 또는 공간 주파수 성분으로 분해하는 변환
- 범용적 데이터 처리에 사용 가능한 다차원 컨테이너

Numpy Documentation

<https://numpy.org/doc/1.21/index.html>
(<https://numpy.org/doc/1.21/index.html>).

Numpy는 대용량 데이터 배열을 효율적으로 다룰 수 있도록 설계되었다.

- Numpy는 내부적으로 데이터를 다른 내장 파이썬 객체와 구분된 연속된 메모리 블록에 저장
- Numpy의 각종 알고리즘은 모두 C로 작성되어 타입 검사나 다른 오버헤드 없이 메모리를 직접 조작
- Numpy 배열은 또한 내장 파이썬의 연속된 자료형들보다 훨씬 더 적은 메모리를 사용
- Numpy 연산은 파이썬 반복문을 사용하지 않고 전체 배열에 대한 복잡한 계산을 수행

In [1]:

```
1 import numpy as np
2 my_arr = np.arange(1000000)
3 my_list = list(range(1000000))
4
5 %time for _ in range(10): my_array2 = my_arr * 2
```

Wall time: 14.3 ms

In [2]:

```
1 %time for _ in range(10): my_list2 = [x*2 for x in my_list]
```

Wall time: 806 ms

In [9]:

```
1 # 배열 연산
2 np.random.seed(0)
3 data = np.random.randn(2,3)
4 print(data, 'Wn')
5 print(data * 10, 'Wn')
6 print(data + data)
```

[[1.76405235 0.40015721 0.97873798]
 [2.2408932 1.86755799 -0.97727788]]

[[17.64052346 4.00157208 9.78737984]
 [22.40893199 18.6755799 -9.7727788]]

[[3.52810469 0.80031442 1.95747597]
 [4.4817864 3.73511598 -1.95455576]]

In [11]:

```
1 print(data.shape) # 크기
2 print(data.dtype) # 자료형
3 print(data.ndim)  # 차원
```

(2, 3)
float64
2

In [12]:

```
1 # 배열 생성
2 # 1차원 배열
3 data1 = [6,7,5,8,0,1]
4 arr1 = np.array(data1)
5 print(arr1, type(arr1))
```

[6 7 5 8 0 1] <class 'numpy.ndarray'>

In [13]:

```

1 # 2차원 배열
2 data2 = [[1,2,3,4],[5,6,7,8]]
3 arr2 = np.array(data2)
4 print(arr2,type(arr2),'\\n')
5 print(arr2.shape,'\\n')
6 print(arr2.ndim,'\\n')
7 print(arr2.dtype)

```

```

[[1 2 3 4]
 [5 6 7 8]] <class 'numpy.ndarray'>

```

```

(2, 4)

```

```

2

```

```

int32

```

In [25]:

```

1 # 3차원 배열
2 data3 = [[[1,2,3,4,5],[6,7,8,9,10]],
3           [[1,2,3,4,5],[6,7,8,9,10]],
4           [[1,2,3,4,5],[6,7,8,9,10]]          ]
5 array3 = np.array(data3)
6 print(array3,type(array3))
7 print(array3.shape)

```

```

[[[ 1  2  3  4  5]
   [ 6  7  8  9 10]]

```

```

 [[ 1  2  3  4  5]
  [ 6  7  8  9 10]]

```

```

[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]] <class 'numpy.ndarray'>
(3, 2, 5)

```

배열 생성 및 초기화

- Numpy는 원하는 shape로 배열을 설정하고 각 요소를 특정 값으로 초기화하는 `zeros`, `ones`, `full`, `eye` 함수 제공
- 파라미터로 입력한 배열과 같은 shape의 배열을 만드는 `zeros_like`, `ones_like`, `full_like` 함수도 제공

In [20]:

```
1 print(np.zeros(10), 'Wn')
2 print(np.zeros((3,5)), 'Wn')
3 print(np.zeros((2,3,2)))
```

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

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

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

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

In [21]:

```
1 print(np.ones(10), 'Wn')
2 print(np.ones((3,5)), 'Wn')
3 print(np.ones((2,3,2)))
```

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

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

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

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

[과제] zeros_like, ones_like, full_like 함수 사용 예를 작성하세요.

In [2]:

```
1 import numpy as np
2 a = np.arange(10).reshape(2,5)
3 print(a, '\n')
4 print(np.zeros_like(a))
5 print(np.ones_like(a))
6 print(np.full_like(a,5))
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]]
[[1 1 1 1 1]
 [1 1 1 1 1]]
[[5 5 5 5 5]
 [5 5 5 5 5]]
```

In [22]:

```
1 a = np.arange(10).reshape(2,5)
2 a
```

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

In [27]:

```
1 # arange 함수 : 파이썬의 range 함수의 배열 버전
2 ar = np.arange(15)
3 ar
```

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

In [32]:

```
1 # reshape 함수
2 ar.reshape(3,5)
```

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

In [33]:

```
1 # Q. array1에 reshape 함수를 이용 (5,2) 배열을 생성하고 형태
2 array1 = np.arange(10)
3 array1.reshape(5,2)
```

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

In [34]:

```
1 # Q. ar에 reshpa() 함수 이용, 1차원, 2차원, 3차원 배열 생성
2 ar = np.arange(12)
3 print(ar.reshape(12), 'Wn')
4 print(ar.reshape(3,4), 'Wn')
5 print(ar.reshape(2,3,2))
```

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

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

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

```
[[ 6  7]
 [ 8  9]
 [10 11]]]
```

In [38]:

```
1 # 차원 변경
2 ar1 = np.arange(30)
3 print('1 -> 2,3차원')
4 ar12 = ar1.reshape(2,-1)
5 ar13 = ar1.reshape(-1,2,5)
6 print(ar12, 'Wn')
7 print(ar13)
```

1 -> 2,3차원

```
[[ 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]]
```

```
[[[ 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]]]
```

In [39]:

```
1 print('2 -> 1,3차원')
2 print(ar12, 'Wn')
3 ar21 = ar12.reshape(-1,)
4 ar23 = ar12.reshape(-1,2,5)
5 print(ar21, 'Wn')
6 print(ar23)
```

2 -> 1,3차원

```
[[ 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]]
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 1
 6 17 18 19 20 21 22 23
24 25 26 27 28 29]
```

```
[[[ 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]]]
```

In [40]:

```
1 print('3 -> 1,2차원')
2 ar31 = ar13.reshape(-1,)
3 ar32 = ar13.reshape(3,-1)
4 print(ar31, 'Wn')
5 print(ar32)
```

3 -> 1,2차원

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 1
 6 17 18 19 20 21 22 23
24 25 26 27 28 29]
```

```
[[ 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]]
```



```
In [42]:
1 ar3 = np.arange(30).reshape(3,2,5)
2 print(ar3,type(ar3),'\\n')
3 l3 = ar3.tolist()
4 print(l3,type(l3))

[[[ 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]]] <class 'numpy.ndarray'>

[[[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]]] <class 'list'>
```

행렬의 종류

<https://math-development-geometry.tistory.com/52> (<https://math-development-geometry.tistory.com/52>)

- 정방행렬은 행과 열의 수가 같은 행렬
- 대각행렬은 주대각선 원소를 제외한 모든 원소들이 0인 정방행렬
- 삼각행렬은 주대각선 원소를 기준으로 위 또는 아래에 있는 성분이 모두 0인 정방행렬
- 항등행렬은 행렬 곱셈 연산에 항등원으로 작용하는 행렬
- 영행렬은 모든 원소가 0인 행렬로 곱셈 연산에서 영원으로 작용하는 행렬
- 전치행렬은 주대각선 원소를 기준으로 행과 열을 바꿔주는 행렬
- 직교행렬은 행렬 A의 역행렬이 A의 전치행렬이고 A의 전치행렬과 A 행렬을 곱하였을때 항등행렬이 나오는 행렬

```
In [46]:
1 # 정방행렬
2 a = np.full((2,2),7)
3 a

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

In [47]:

```
1 # 항등행렬, 단위행렬
2 ar = np.eye(3)
3 ar
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [49]:

```
1 # 대각행렬
2 diag_mat = np.diag([1,2,3,4,5])
3 diag_mat
```

```
array([[1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0],
       [0, 0, 0, 0, 5]])
```

In [51]:

```
1 # 삼각행렬
2 upper_tri_mat = np.triu([[1,2,3],[4,5,6],[7,8,9]])
3 print(upper_tri_mat)
```

```
[[1 2 3]
 [0 5 6]
 [0 0 9]]
```

[과제] 하삼각행렬을 생성하세요.

In [3]:

```
1 lower_tri_mat=np.tril([[1,2,3],[4,5,6],[7,8,9]])
2 print(lower_tri_mat)
```

```
[[1 0 0]
 [4 5 0]
 [7 8 9]]
```

In [53]:

```

1 # 전치행렬 : 원래의 행렬에서 행과 열을 바꾼 행렬을 의미
2 mat = np.array([[1, 2], [3, 4], [5, 6]])
3 print(mat, '\n')
4 transpose_mat = mat.T
5 print(transpose_mat)

```

```

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

```

```

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

```

직교행렬

- 행과 열이 서로 직교하는 정방행렬.
- 모든 열벡터와 행벡터가 서로 직교하고 크기가 1인 단위벡터로 이루어짐
- np.linalg.qr() 함수를 사용하여 QR 분해 수행하여 직교행렬을 추출
- q를 추출하여 직교행렬 orth_mat을 만든다
- np.dot(orth_mat, orth_mat.T)를 계산하여 직교성을 검증
- np.allclose() 함수로 두 행렬이 동일한지 검사. True를 반환하면 두 행렬은 동일

In [57]:

```

1 # A의 전치행렬과 A 행렬을 곱하였을때 항등행렬이 나오는 행렬
2 mat = np.random.randn(3,3)
3 print(mat)
4
5 # qr 분해
6 q,r = np.linalg.qr(mat)
7
8 # 직교행렬 추출
9 orth_mat = q
10 print(orth_mat)
11
12 # 직교성 검증
13 print(np.allclose(np.dot(orth_mat, orth_mat.T), np.eye(3)))
14

```

```

[[-1.98079647 -0.34791215  0.15634897]
 [ 1.23029068  1.20237985 -0.38732682]
 [-0.30230275 -1.04855297 -1.42001794]]
[[-0.84243028 -0.43595686  0.3166273 ]
 [ 0.5232411  -0.52171175  0.67382164]
 [-0.12856898  0.73332016  0.66761632]]

```

True

In [58]:

```
1 # 인덱싱, 슬라이싱
2 ar2 = np.arange(1,10).reshape(3,3)
3 ar2
```

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

In [63]:

```
1 print(ar2[2])
2 print(ar2[0][2])
3 print(ar2[0,2])
```

```
[7 8 9]
3
3
```

In [64]:

```
1 ar = np.arange(20).reshape(5,4)
2 ar
```

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

In [65]:

```
1 ar[:,2,1:]
```

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

In [4]:

```
1 # Q. ar에서 슬라이싱을 사용해서 아래와 같이 출력하세요.
2 ar = np.arange(1,10).reshape(3,3)
3 ar
```

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

In [74]:

```
1 print(ar[:2,:2], 'Wn')
2 print(ar[1:], 'Wn')
3 print(ar[1:,:], 'Wn')
4 print(ar, 'Wn')
5 print(ar[:2,1:], 'Wn')
6 print(ar[:2,0])
```

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

```
[[4 5 6]
 [7 8 9]]
```

```
[[4 5 6]
 [7 8 9]]
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
[[2 3]
 [5 6]]
```

```
[1 4]
```

```
In [ ]:
1  [[1 2]
2  [4 5]]
3
4  (2가지 방법)
5  [[4 5 6]
6  [7 8 9]]
7
8  [[1 2 3]
9  [4 5 6]
10 [7 8 9]]
11
12 [[2 3]
13 [5 6]]
14
15 [1 4]
```

[과제] ar에서 인덱스를 이용해서 값을 선택하고 리스트로 아래와 같이 출력하세요.

```
In [ ]:
1  [3, 6]
2  [[1, 2], [4, 5]]
3  [[1, 2, 3], [4, 5, 6]]
```

```
In [6]:
1  ar2 = np.arange(1,10).reshape(3,3)
2  print(ar2)

[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
In [5]:
1  # 리스트로 출력
2  print(ar2[:2,2].tolist())      #[3, 6]
3  print(ar2[:2,:2].tolist())    #[[1, 2], [4, 5]]
4  print(ar2[:2,:].tolist())     #[[1, 2, 3], [4, 5, 6]]

[[1 2 3]
 [4 5 6]
 [7 8 9]]

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

In [68]:

```
1 # Boolean indexing
2 ar = np.arange(1,10)
3 ar
```

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

In [69]:

```
1 ar[ar > 5]
```

```
array([6, 7, 8, 9])
```

In []:

```
1 # Q. 1 ~ 14까지 ndarray를 만들어 array_e로 저장하고 (array_e
```

In [75]:

```
1 array_e = np.arange(1,15)
2 array_e[(array_e/2) > 5]
```

```
array([11, 12, 13, 14])
```

In [82]:

```
1 # arr에서 0.5보다 큰 수를 출력하세요.
2 np.random.seed(0)
3 arr = np.random.randn(30)
4 arr[arr>0.5]
```

```
array([1.76405235, 0.97873798, 2.2408932 , 1.86755
799, 0.95008842,
        1.45427351, 0.76103773, 1.49407907, 0.65361
86 , 0.8644362 ,
        2.26975462, 1.53277921, 1.46935877])
```

In [84]:

```
1 # Q. data에서 3의 배수인 것 수만 출력하세요.  
2 np.random.seed(0)  
3 data = np.arange(30)  
4 data[data%3==0]
```

```
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27])
```

[과제] [1,2,0,0,4,0]에서 zero가 아닌 인덱스를 배열 형태로 출력하세요.

np.where(condition)은

- condition 배열의 요소가 True인 인덱스를 반환하는 함수입니다.
- condition 배열은 bool 타입이어야 하며, 반환값은 tuple 타입으로 (array of row indices, array of column indices)의 형태로 반환

In [29]:

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

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

In [30]:

```
1 array = np.where(arr > 5)  
2 array
```

```
(array([1, 2, 2, 2], dtype=int64), array([2, 0, 1,  
2], dtype=int64))
```

In [31]:

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

```
[1 2 2 2] [2 0 1 2]
```


In [9]:

```
1 a=np.array([1,2,0,0,4,0])
2 np.where(a!=0, 1, 0)
```

```
array([1, 1, 0, 0, 1, 0])
```

In [11]:

```
1 a=np.array([1,2,0,0,4,0])
2
3 index=np.where(a!=0)      #조건을 만족하는 인덱스 반환
4 print(index)
5 print(index[0])
```

```
(array([0, 1, 4], dtype=int64),)
[0 1 4]
```

In [14]:

```
1 a=np.array([1,2,0,0,4,0])
2 nz = np.nonzero(a)
3 print(nz[0])
```

```
[0 1 4]
```

In [85]:

```
1 bools = np.array([False,False, True, True])
2 bools
```

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

In [86]:

```
1 # any 메서드 : 하나 이상의 값이 True인지 검사
2 bools.any()
```

```
True
```

In [87]:

```
1 # all 메서드 : 모든 원소가 True인지 검사
2 bools.all()
```

False

In [88]:

```
1 # Q. arr에서 0보다 크면 2, 아니면 -2로 변경하세요.
2 arr = np.random.randn(4,4)
3 arr
```

```
array([[ 1.76405235,  0.40015721,  0.97873798,  2.
 2408932 ],
       [ 1.86755799, -0.97727788,  0.95008842, -0.
 15135721],
       [-0.10321885,  0.4105985 ,  0.14404357,  1.
 45427351],
       [ 0.76103773,  0.12167502,  0.44386323,  0.
 33367433]])
```

In [89]:

```
1 np.where(arr > 0, 2, -2)
```

```
array([[ 2,  2,  2,  2],
       [ 2, -2,  2, -2],
       [-2,  2,  2,  2],
       [ 2,  2,  2,  2]])
```

In [90]:

```
1 # Q. arr의 모든 양수를 2로 바꾸세요.  
2 np.where(arr > 0 , 2, arr)
```

```
array([[ 2.          ,  2.          ,  2.          ,  2.  
],  
       [ 2.          , -0.97727788,  2.          , -0.  
15135721],  
       [-0.10321885,  2.          ,  2.          ,  2.  
],  
       [ 2.          ,  2.          ,  2.          ,  2.  
]])
```

In [92]:

```
1 # np.sort() : 복사본을 반환. 원본 미반영  
2 np.random.seed(0)  
3 arr = np.random.randint(1,100,size=10)  
4 arr
```

```
array([45, 48, 65, 68, 68, 10, 84, 22, 37, 88])
```

In [94]:

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

```
[10 22 37 45 48 65 68 68 84 88]  
[45 48 65 68 68 10 84 22 37 88]
```

In [95]:

```
1 sorted = np.sort(arr)  
2 print(sorted)
```

```
[10 22 37 45 48 65 68 68 84 88]
```

In [99]:

```
1 # 행렬이 2차원 이상일 경우 axis 축 값 설정을 통해 로우 방향,  
2 ar2 = np.array([[8,12],  
3                [7,1]])  
4 print(ar2)  
5 print(np.sort(ar2,axis=0),'\n')  
6 print(np.sort(ar2,axis=1))
```

```
[[ 8 12]  
 [ 7  1]]  
[[ 7  1]  
 [ 8 12]]
```

```
[[ 8 12]  
 [ 1  7]]
```

In [104]:

```
1 # ndarray.sort() : 원본 반영  
2 np.random.seed(0)  
3 arr = np.random.randint(10,size=10)  
4 print(arr)  
5 arr.sort()  
6 arr
```

```
[5 0 3 3 7 9 3 5 2 4]
```

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

In [105]:

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

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

In [108]:

```
1 # 다차원 배열의 정렬은 sort 메서드에 넘긴 축에 따라 1차원 부
2 arr.sort(1)
3 # arr.sort(0) # 행방향
4 arr.sort(1) # 열방향. Default
5 arr
```

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

In [109]:

```
1 # 정렬된 행렬의 인덱스 반환 : 기존 원본 행렬의 원소에 대한 인
2 org_array = np.array([3,1,9,5])
3 sort_indices = np.argsort(org_array)
4 print(org_array)
5 sort_indices
```

```
[3 1 9 5]
```

```
array([1, 0, 3, 2], dtype=int64)
```

In [110]:

```
1 # 배열 데이터의 입출력
2 # np.save, np.load는 바이너리 형식. npy 파일로 저장
3 arr= np.arange(10)
4 np.save('some_array',arr)
5
6 np.load('some_array.npy')
```

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

In [112]:

```
1 # np.savez : 여러개의 배열을 압축된 형식으로 저장
2 np.savez('array_archive.npz', a=arr, b=arr)
3 arch = np.load('array_archive.npz')
4 arch['a']
```

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

In [116]:

```
1 # Q. 인덱싱을 사용하여 [1 0 1 0 1 0 1 0 1 0]를 출력하세요
2
3 a = np.ones(10,dtype=int)
4 a[[1,3,5,7,9]] = 0
5 print(a)
```

```
[1 0 1 0 1 0 1 0 1 0]
```

In [113]:

```
1 arr = np.array([1,2,3,4,5,6,7,8,9,10])
2 arr[:] = arr%2
3 arr
```

```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0])
```

[과제] 1 ~ 100을 아래와 같이 출력하세요

In []:

```
1 [[ 1  2  3  4  5  6  7  8  9 10]
2  [20 19 18 17 16 15 14 13 12 11]
3  [21 22 23 24 25 26 27 28 29 30]
4  [40 39 38 37 36 35 34 33 32 31]
5  [41 42 43 44 45 46 47 48 49 50]
6  [60 59 58 57 56 55 54 53 52 51]
7  [61 62 63 64 65 66 67 68 69 70]
8  [80 79 78 77 76 75 74 73 72 71]
9  [81 82 83 84 85 86 87 88 89 90]
10 [100 99 98 97 96 95 94 93 92 91]] <class 'numpy.nd
```

In [18]:

```

1 arr = np.arange(1,101).reshape(10,10)
2 arr[1::2] = arr[1::2, ::-1]
3 arr

```

```

array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [20, 19, 18, 17, 16, 15, 14, 13, 12, 11],
       [21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
       [40, 39, 38, 37, 36, 35, 34, 33, 32, 31],
       [41, 42, 43, 44, 45, 46, 47, 48, 49, 50],
       [60, 59, 58, 57, 56, 55, 54, 53, 52, 51],
       [61, 62, 63, 64, 65, 66, 67, 68, 69, 70],
       [80, 79, 78, 77, 76, 75, 74, 73, 72, 71],
       [81, 82, 83, 84, 85, 86, 87, 88, 89, 90],
       [100, 99, 98, 97, 96, 95, 94, 93, 92, 91]])

```

[과제] np.ones((10,10))을 아래와 같이 출력하세요

In []:

```

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

```


In [21]:

```
1 arr = np.ones((10,10))
2 arr
3 arr[1:9,1:9] = 0
4 arr
```

```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

[과제] np.ones((5,5))을 아래와 같이 출력하세요

In []:

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

In [23]:

```

1 arr = np.zeros((10,10))
2 arr[2:8,2:8] = 1
3 arr

```

```

array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 1., 1., 1., 1., 1., 0., 0.],
       [0., 0., 1., 1., 1., 1., 1., 1., 0., 0.],
       [0., 0., 1., 1., 1., 1., 1., 1., 0., 0.],
       [0., 0., 1., 1., 1., 1., 1., 1., 0., 0.],
       [0., 0., 1., 1., 1., 1., 1., 1., 0., 0.],
       [0., 0., 1., 1., 1., 1., 1., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])

```

[과제] np.zeros((8,8))을 이용해서 아래와 같이 출력하세요(두가지 방식: 인덱싱, tile 함수)

In []:

```

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

```

In [27]:

```

1 z = np.zeros((8,8),dtype=int)
2 z[1::2,::2] = 1
3 z[:,1::2] = 1
4 z

```

```

array([[0, 1, 0, 1, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 1, 0],
       [0, 1, 0, 1, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 1, 0],
       [0, 1, 0, 1, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 1, 0],
       [0, 1, 0, 1, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 1, 0]])

```

[과제] np.arange(336).reshape(6,7,8)에서 100번째 요소의 인덱스를 구하세요

In [28]:

```
1 # unravel_index() 함수는 1차원 배열에서 인덱스를 다차원 인덱
2 arr = np.arange(336).reshape(6,7,8)
3 np.unravel_index(100, arr.shape)
```

(1, 5, 4)

In [117]:

```
1 np.arange(336).reshape(6,7,8)
```



```
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, 50, 51, 52, 53, 54, 55]],
      dtype=int64)
```

```
[[ 56,  57,  58,  59,  60,  61,  62,  63],
 [ 64,  65,  66,  67,  68,  69,  70,  71],
 [ 72,  73,  74,  75,  76,  77,  78,  79],
 [ 80,  81,  82,  83,  84,  85,  86,  87],
 [ 88,  89,  90,  91,  92,  93,  94,  95],
 [ 96,  97,  98,  99, 100, 101, 102, 103]]
```