

■ 배열 생성 함수

● 간단히 배열을 생성 할 수 있는 함수

reshape	arange	zeros	ones
empty	zeros_like	ones_like	empty_like
identity	eye	diag	random.uniform random.normal random.randn

■ 배열 크기 변형 (reshape)

- 만들어진 배열 요소의 데이터와 개수를 유지한 채로 형태를 변형
- 변형하기 전 원본의 size와 변형된 결과의 size가 반드시 같아야 됨

– 2 x 3 배열

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

1	2	3
4	5	6

– 3 x 2 배열로 변형

```
np.array(matrix).reshape(3, 2)
```

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

1	2
3	4
5	6

■ 배열 크기 변형 (reshape)

- 만들어진 배열 요소의 데이터와 개수를 유지한 채로 형태를 변형
- 변형하기 전 원본의 size와 변형된 결과의 size가 반드시 같아야 됨

– 6 x 1 배열로 변형

```
np.array(matrix).reshape(6, 1)
```

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

1
2
3
4
5
6

– 1 x 6 배열로 변형

```
np.array(matrix).reshape(1, 6)
```

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

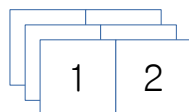
1	2	3	4	5	6
---	---	---	---	---	---

■ 배열 크기 변형 (reshape)

- 만들어진 배열 요소의 데이터와 개수를 유지한 채로 형태를 변형
- 변형하기 전 원본의 size와 변형된 결과의 size가 반드시 같아야 됨
 - 3 x 1 x 2 배열로 변형 (개수에 맞추어서 동적 변형)

```
np.array(matrix).reshape(3, 1, 2)
```

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



1	2
---	---

- ? x 2 배열로 변형 (개수에 맞추어서 동적 변형)

```
np.array(matrix).reshape(-1, 2)
```

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

1	2
3	4
5	6

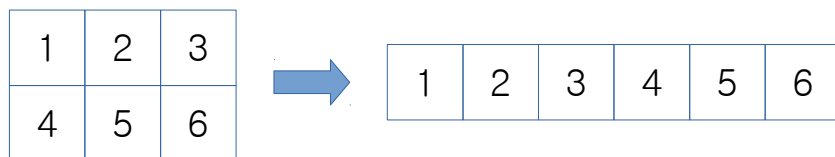
■ 배열 크기 변형 (flatten)

● 다차원 배열을 1차원 배열로 변형

– shape (2, 3) → shape (6,)

```
np.array(matrix).flatten()
```

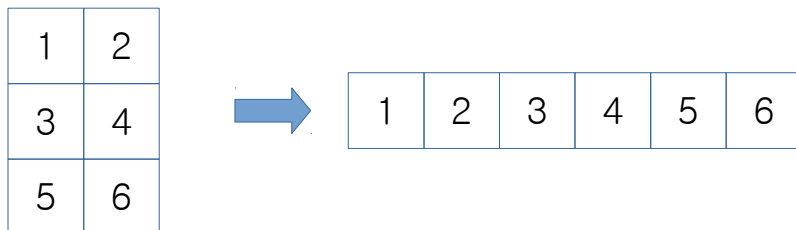
```
array([1, 2, 3, 4, 5, 6])
```



– shape (3, 2) → shape (6,)

```
np.array(matrix).reshape(-1, 2).flatten()
```

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



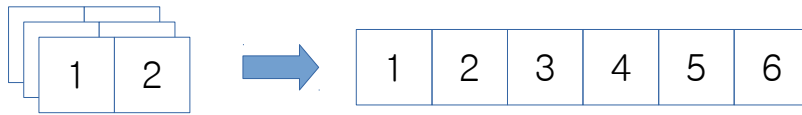
■ 배열 크기 변형 (flatten)

● 다차원 배열을 1차원 배열로 변형

– shape (3, 1, 2) → shape (6,)

```
np.array(matrix).reshape(3, 1, 2).flatten()
```

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



■ 배열 생성 (arange)

- range 함수와 비슷한 기능
- 값의 범위를 지정하여 값이 채워져 있는 배열을 생성
- 특정한 규칙에 따라 증가하는 값을 넣는 것도 가능

– 기본 : arange(size)

```
np.arange(30)
```

```
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])
```

– 간격 지정 : arange(start, end, step)

```
np.arange(0, 10, 2) # 간격 지정 생성
```

```
array([0, 2, 4, 6, 8])
```

```
np.arange(0, 2, 0.2) # 간격 지정 생성
```

```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8])
```

■ 배열 생성 (arange)

- range 함수와 비슷한 기능
- 값의 범위를 지정하여 값이 채워져 있는 배열을 생성
- 특정한 규칙에 따라 증가하는 값을 넣는 것도 가능

– 다차원 배열 생성 : arange + reshape

```
np.arange(1, 26).reshape(5, 5) # 2차원 배열로 변형
```

```
array([[ 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]])
```

```
np.arange(0, 30).reshape(3, 2, 5) # 3차원 배열로 변형
```

```
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]]])
```


■ 배열 생성 (zeros)

- 0으로 채워진 배열 생성
- arange 와는 다르게 shape 지정이 가능
 - 사용법 : zeros(shape, dtype, order)

```
np.zeros(shape=(5,), dtype=np.int8, order='C')
```

```
array([0, 0, 0, 0, 0], dtype=int8)
```

```
np.zeros((5,)) # dtype=float64
```

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

```
np.zeros((3, 2)) # 3 x 2
```

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

■ 배열 생성 (ones)

- 1로 채워진 배열 생성
- arange 와는 다르게 shape 지정이 가능
 - 사용법 : ones(shape, dtype, order)

```
np.ones(shape=(5,), dtype=np.int8, order='C')
```

```
array([1, 1, 1, 1, 1], dtype=int8)
```

```
np.ones((5,)) # dtype=float64
```

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

```
np.ones((3, 2)) # 3 x 2
```

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

■ 배열 생성 (empty)

- shape만 주어지고 비어있는 배열 생성
- 배열을 생성만 하고 값을 초기화 하지 않아서

메모리에 저장되어 있던 기존 값이 저장 될 수 있음 (초기값 알 수 없음)

```
np.empty(shape=(10,), dtype=np.int32)
```

```
array([ 0, 0, 1556801328, 0, 2, 0, -1, -1, 536870912, 1981820462])
```

```
np.empty(shape=(5, 6), dtype=np.int8)
```

```
array([[ 80,  1, -10,  41, 121,  1],
       [  0,  0,  64,  58, -86,  52],
       [121,  1,  0,  0,  0,  16],
       [  0,  0,  0,  0,  0,  0],
       [-1, -1, -1, -1, -1, -1]], dtype=int8)
```

■ 배열 생성 (zeros_like / ones_like / empty_like)

- shape을 명시하여 배열을 만들지 않고 다른 배열과 같은 shape으로 생성

```
matrix = np.arange(24).reshape(4, 6)  
np.zeros_like(matrix)
```

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

```
matrix = np.arange(18).reshape(3, 6)  
np.ones_like(matrix)
```

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

```
matrix = np.arange(30).reshape(5, 6)  
np.empty_like(matrix, dtype=np.int8)
```

```
array([[ 0,  0,  0,  0,  0,  0],  
       [ 0,  0, 48, -25, -54, 92],  
       [ 0,  0,  0,  0,  2,  0],  
       [ 0,  0,  0,  0,  0,  0],  
       [-1, -1, -1, -1, -1, -1]], dtype=int8)
```

■ 배열 생성 (identity)

● 단위 행렬 생성

- 주대각선의 모든 값이 1이고, 나머지는 0인 정사각 행렬

```
np.identity(5)
```

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

```
np.identity(7, dtype=np.int8)
```

```
array([[1, 0, 0, 0, 0, 0, 0],  
       [0, 1, 0, 0, 0, 0, 0],  
       [0, 0, 1, 0, 0, 0, 0],  
       [0, 0, 0, 1, 0, 0, 0],  
       [0, 0, 0, 0, 1, 0, 0],  
       [0, 0, 0, 0, 0, 1, 0],  
       [0, 0, 0, 0, 0, 0, 1]], dtype=int8)
```

■ 배열 생성 (eye)

- 대각선이 1로 채워지는 행렬
- 대각선의 시작 위치 지정 가능

```
np.eye(5)
```

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

```
np.eye(5, k=1)
```

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

```
np.eye(N=4, M=6, dtype=np.int8, k=2)
```

```
array([[0, 0, 1, 0, 0, 0],  
       [0, 0, 0, 1, 0, 0],  
       [0, 0, 0, 0, 1, 0],  
       [0, 0, 0, 0, 0, 1]], dtype=int8)
```

■ 배열 생성 (full)

● 입력 값으로 채워지는 행렬

– full((shape), value)

```
np.full((2, 3), 5)
```

```
array([[5, 5, 5],  
       [5, 5, 5]])
```

```
np.array([5] * 6).reshape(2, 3)
```

```
array([[5, 5, 5],  
       [5, 5, 5]])
```

```
np.full([5, 4], 'a')
```

```
array([[ 'a', 'a', 'a', 'a'],  
       [ 'a', 'a', 'a', 'a'],  
       [ 'a', 'a', 'a', 'a'],  
       [ 'a', 'a', 'a', 'a'],  
       [ 'a', 'a', 'a', 'a']], dtype='<U1')
```

■ 배열 생성 (diag - diagonal matrix)

- 모든 비대각 요소가 0인 행렬 (대각 행렬)
- 대각선의 시작 위치 지정 가능
- 반드시 정방 행렬일 필요는 없음
- 1차원 배열을 입력 시 해당 값을 이용하여 대각 행렬 생성

```
np.diag(list(range(3)))
```

```
array([[0, 0, 0],  
       [0, 1, 0],  
       [0, 0, 2]])
```

```
np.diag([2, 7, 11, 15])
```

```
array([[ 2,  0,  0,  0],  
       [ 0,  7,  0,  0],  
       [ 0,  0, 11,  0],  
       [ 0,  0,  0, 15]])
```


■ 배열 생성 (diag - diagonal matrix)

● 2차원 배열을 입력 시 대각선의 값을 추출

```
matrix = np.arange(25).reshape(5, 5)  
matrix
```

```
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]])
```

```
np.diag(matrix)
```

```
array([ 0,  6, 12, 18, 24])
```

```
np.diag(matrix, k=2)
```

```
array([ 2,  8, 14])
```

■ 배열 생성 (random.uniform / random.normal / random.randn)

● 균등 분포에 따른 배열 생성

```
np.random.uniform(1, 2, 10)
```

```
array([1.33951356, 1.39533543, 1.56751171, 1.91104563, 1.32673962,  
       1.30626941, 1.15482684, 1.69006847, 1.43678418, 1.1270499 ])
```

```
np.random.uniform(1, 7, 6)
```

```
array([5.98948422, 1.53805996, 6.81655757, 1.52411261, 5.39652364,  
       5.42927572])
```

● 정규 분포에 따른 배열 생성

```
np.random.normal(1, 2, 10)
```

```
array([ 0.43202233,  4.34198193,  0.83000795,  0.0992409 ,  0.56559026,  
        2.27134868,  2.99616998, -0.09524534,  1.55612464,  0.754127  ])
```

```
np.random.normal(1, 2, 10)
```

```
array([ 0.65637218, -0.35142033, -1.60175579,  1.53361009, -1.3562361 ,  
       -1.77995504,  0.49843321,  1.43932665,  3.35621282,  0.65851473])
```

■ 배열 생성 (random.uniform / random.normal / random.randn)

● 지정된 개수에 맞게 랜덤 값을 가지는 배열 생성

```
np.random.randn(4)
```

```
array([-1.25679016,  0.30774647, -0.01102206,  0.38871575])
```

```
np.random.randn(2, 3)
```

```
array([[ 1.49690419,  0.72655239,  0.81915058],  
       [ 0.22165529, -0.26523271, -0.94316217]])
```