

# 10. 넘파이 어레이

# 개요

- 넘파이 어레이 소개
- 어레이 기초 연산

## 10.1. 넘파이란?

- 넘파이 numpy: NUMerical PYthon

In [1]:

```
import numpy as np
```

- 파이썬 데이터 과학에서 가장 중요한 도구를 제공하는 라이브러리
  - 다차원 어레이(배열)
  - 메모리 효율적인 빠른 어레이 연산
- 판다스 pandas 라이브러리 이해에 절대적

## 10.2. 다차원 어레이

# 1차원 어레이

- 리스트 활용

In [2]:

```
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
```

Out[2]:

```
array([6. , 7.5, 8. , 0. , 1. ])
```

- 튜플 활용

In [3]:

```
data1 = (6, 7.5, 8, 0, 1)
arr1 = np.array(data1)
arr1
```

Out[3]:

```
array([6. , 7.5, 8. , 0. , 1. ])
```

## ndarray 자료형

In [4]:

```
type(arr1)
```

Out[4]:

```
numpy.ndarray
```

## 2차원 어레이

- 중첩된 리스트를 2차원 어레이로 변환 가능
- 단, 항목으로 사용된 리스트의 길이가 모두 동일해야 함

In [5]:

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]  
arr2 = np.array(data2)  
arr2
```

Out[5]:

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

## shape 속성: 어레이 모양

- 2차원 어레이의 모양은 길이가 2인 튜플

In [6]:

```
arr2.shape
```

Out[6]:

```
(2, 4)
```

- 1차원 어레이의 모양: 길이가 1인 튜플

In [7]:

```
arr1.shape
```

Out[7]:

```
(5, )
```

## dtype 속성: 어레이 항목의 자료형

어레이의 모든 항목은 동일한 자료형을 가져야 함

In [8]:

```
arr2.dtype
```

Out[8]:

```
dtype('int32')
```

## `ndim` 속성: 어레이의 차원

- `shape`에 저장된 튜플의 길이와 동일.

In [9]:

```
arr2.ndim
```

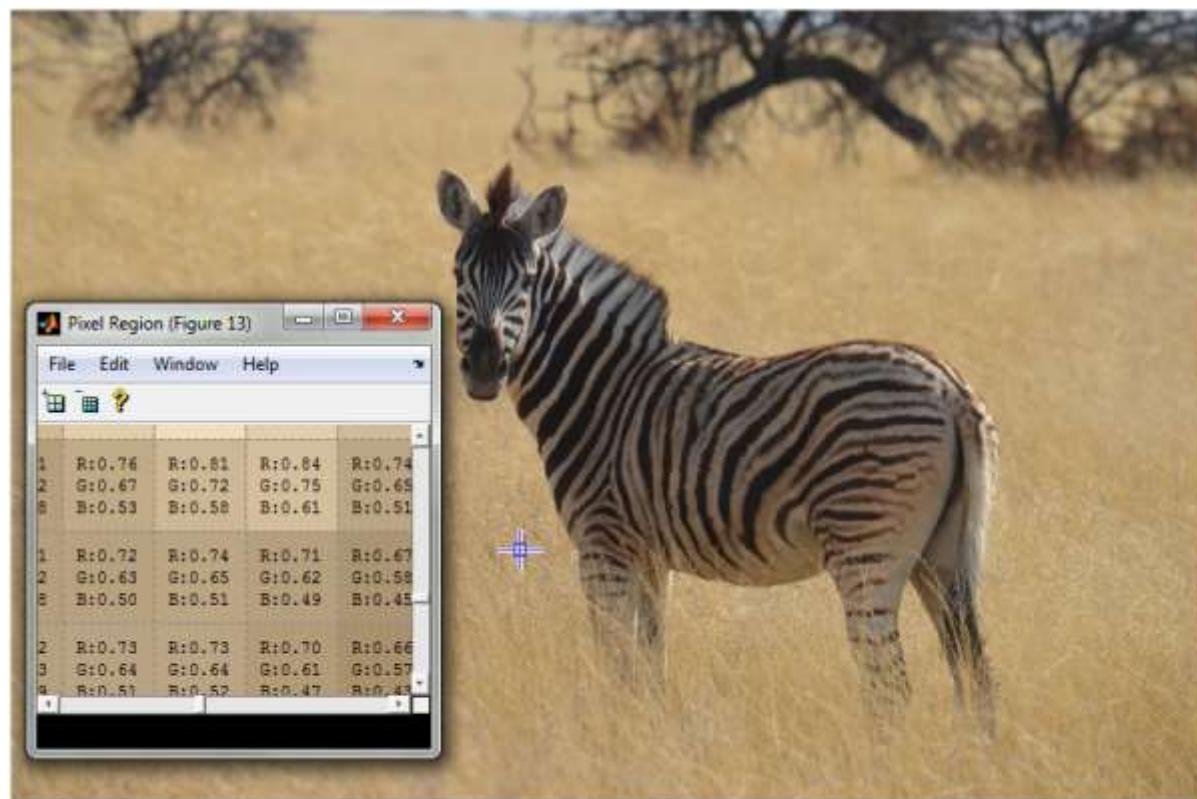
Out[9]:

```
2
```

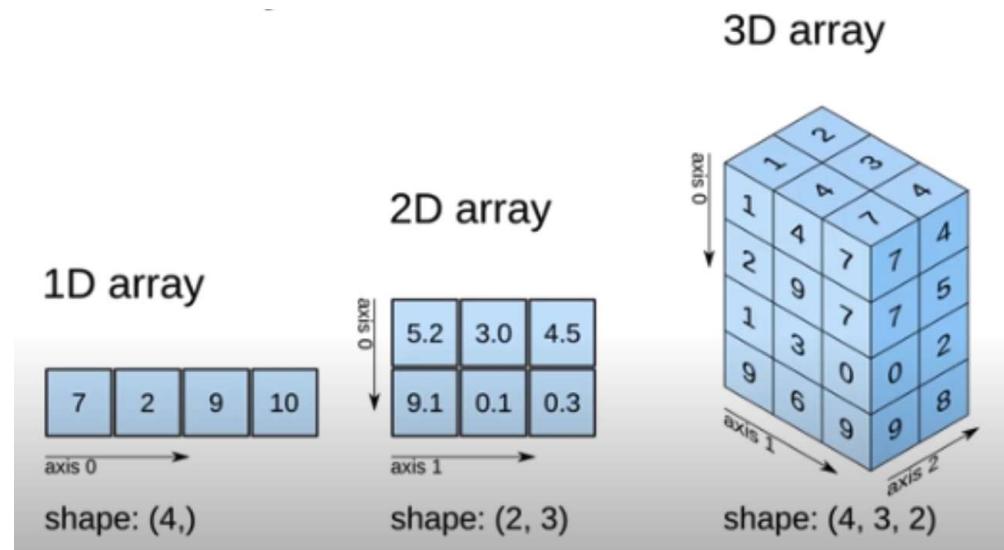
## 3차원 어레이

- ( $n, m, p$ ) 모양의 3차원 어레이를 이해하는 게 중요

- 방법 1: 바둑판을  $(n, m)$  크기의 격자로 나누고 각각의 칸에 길이가  $p$  인 1차원 어레이가 위치하는 것으로 이해



- 방법 2: ( $m, p$ ) 모양의 2차원 어레이  $n$  개를 항목으로 갖는 1차원 어레이로 이해



<그림 출처: [NumPy Arrays and Data Analysis](#)>

In [10]:

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

Out[10]:

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

어레이 객체 생성 함수

## zeros() 함수

- 1차원

In [11]:

```
np.zeros(10)
```

Out[11]:

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

- 2차원부터는 정수들의 튜플로 모양을 지정한다.

In [12]:

```
np.zeros((3, 6))
```

Out[12]:

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

In [13]:

```
np.zeros((4, 3, 2))
```

Out[13]:

```
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.]])
```

## arange() 함수

- range() 함수와 유사하게 작동하며 부동소수점 스텝도 지원

In [14]:

```
np.arange(15)
```

Out[14]:

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

In [15]:

```
np.arange(0, 1, 0.1)
```

Out[15]:

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

### 10.3. dtype의 종류

## 기본 dtype

자료형	자료형 코드	설명
int8 / uint8	i1 / u1	signed / unsigned 8 비트 정수
int16 / uint16	i2 / u2	signed / unsigned 16 비트 정수
int32 / uint32	i4 / u4	signed / unsigned 32 비트 정수
int64 / uint64	i8 / u8	signed / unsigned 64 비트 정수
float16	f2	16비트(반 정밀도) 부동소수점
float32	f4 또는 f	32비트(단 정밀도) 부동소수점
float64	f8 또는 d	64비트(배 정밀도) 부동소수점
float128	f16 또는 g	64비트(배 정밀도) 부동소수점
bool	?	부울 값
object	O	임의의 파이썬 객체
string_	S	고정 길이 아스키 문자열 / 예) S8 , S10
unicode_	U	고정 길이 유니코드 문자열 / 예) U8 , U10

## 넘파이 어레이 기본 자료형: float64 자료형

In [16]:

```
arr1 = np.array([1, 2, 3], dtype=np.float64)  
arr1.dtype
```

Out[16]:

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

In [17]:

```
arr1 = np.array([1, 2, 3], dtype='f8')  
arr1.dtype
```

Out[17]:

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

## 넘파이 어레이 기본 자료형: int32 자료형

In [18]:

```
arr2 = np.array([1, 2, 3], dtype=np.int32)  
arr2.dtype
```

Out[18]:

```
dtype('int32')
```

In [19]:

```
arr2 = np.array([1, 2, 3], dtype='i4')  
arr2.dtype
```

Out[19]:

```
dtype('int32')
```

## 형변환: `astype()` 메서드

- `int` 자료형을 `float` 자료형으로 형변환하기

In [20]:

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

Out[20]:

```
dtype('int32')
```

In [21]:

```
float_arr = arr.astype(np.float64)
float_arr.dtype
```

Out[21]:

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

- float 자료형을 int 자료형으로 형변환하기

In [22]:

```
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
arr
```

Out[22]:

```
array([ 3.7, -1.2, -2.6,  0.5, 12.9, 10.1])
```

In [23]:

```
arr.astype(np.int32)
```

Out[23]:

```
array([ 3, -1, -2,  0, 12, 10])
```

## 넘파이 어레이 기본 자료형: S

- 문자열 자료형의 크기는 자동 결정됨

In [24]:

```
numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
numeric_strings.dtype
```

Out[24]:

```
dtype('S4')
```

In [25]:

```
numeric_strings.astype(float)
```

Out[25]:

```
array([ 1.25, -9.6 , 42. ])
```

In [26]:

```
numeric_strings2 = np.array(['1.25345', '-9.673811345', '42'], dtype=np.string_)
numeric_strings2.dtype
```

Out[26]:

```
dtype('S12')
```

- `astype()`: 타 객체의 `dtype` 정보 이용 가능

In [27]:

```
int_array = np.arange(10)
int_array.dtype
```

Out[27]:

```
dtype('int32')
```

In [28]:

```
calibers = np.array([.22, .270, .357, .380, .44, .50], dtype=np.float64)
```

In [29]:

```
int_array.astype(calibers.dtype)
```

Out[29]:

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

- 자료형 코드 활용

In [30]:

```
empty_uint32 = np.empty(8, dtype='u4')
empty_uint32.dtype
```

Out[30]:

```
dtype('uint32')
```

## 10.4. 어레이 연산

# 사칙연산

- 넘파이 어레이 연산은 기본적으로 항목별로 실행

In [31]:

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])  
arr2 = np.array([[3., 2., 1.], [4., 2., 12.]])
```

In [32]:

```
arr + arr2
```

Out[32]:

```
array([[ 4.,  4.,  4.],  
       [ 8.,  7., 18.]])
```

In [33]:

```
arr / arr2
```

Out[33]:

```
array([[0.33333333, 1.           , 3.           ],  
       [1.           , 2.5          , 0.5          ]])
```

- 숫자와의 연산은 모든 항목에 동일한 값 사용

In [34]:

```
arr - 2.4
```

Out[34]:

```
array([[-1.4, -0.4,  0.6],  
       [ 1.6,  2.6,  3.6]])
```

In [35]:

```
1 / arr
```

Out[35]:

```
array([[1.          , 0.5        , 0.33333333],  
       [0.25        , 0.2        , 0.16666667]])
```

# 거듭제곱(지수승)

In [36]:

```
arr ** arr2
```

Out[36]:

```
array([[1.0000000e+00, 4.0000000e+00, 3.0000000e+00],
       [2.5600000e+02, 2.5000000e+01, 2.17678234e+09]])
```

In [37]:

```
2 ** arr
```

Out[37]:

```
array([[ 2.,  4.,  8.],
       [16., 32., 64.]])
```

In [38]:

```
arr ** 0.5
```

Out[38]:

```
array([[1.          , 1.41421356, 1.73205081],
       [2.          , 2.23606798, 2.44948974]])
```

## 비교 연산

In [39]:

```
arr2 > arr
```

Out[39]:

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

In [40]:

```
arr2 <= arr
```

Out[40]:

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

In [41]:

```
arr == arr2
```

Out[41]:

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

In [42]:

```
arr != arr2
```

Out[42]:

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

# 논리 연산

In [43]:

```
~(arr == arr)
```

Out[43]:

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

In [44]:

```
(arr == arr) & (arr2 == arr2)
```

Out[44]:

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

In [45]:

```
~(arr == arr) | (arr2 != arr)
```

Out[45]:

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