영상처리 프로그래밍

3주차

소개

- Instructor
  - 교수 : 김백섭 교수님
  - 연구실 : A1201
  - 이메일 : bskim@hallym.ac.kr

- TA
  - 조교 : 김형훈
  - 연구실 : A1409 (운영체제 연구실)
  - 이메일 : khh8996@naver.com
  - 전화번호 : 010-5316-7953

목차

- NumPy 라이브러리
- 실습
- 과제

NumPy 라이브러리

- NumPy
  - 행렬, 다차원 배열, 벡터 등을 쉽게 처리할 수 있도록 지원하는 Python 라이브러리
  - 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공함
  - ndarray는 NumPy의 핵심인 다차원 행렬 자료구조 클래스이며 모든 요소는 같은 자료형이어야 함

- pip install numpy을 이용하여 numpy 라이브러리를 설치할 수 있음
  - Jupyter Notebook에는 이미 설치되어 있는 것을 볼 수 있음

```
pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (1.18.2)
```

NumPy 라이브러리

- import을 이용하여 numpy 라이브러리를 불러와서 np라는 명칭으로 지정하여 사용함

```
import numpy as np
```

- 배열 생성
  - 배열 = np.array(리스트)
  - 배열 = np.array([], [], [])

- 생성한 배열의 형태와 차원을 확인할 때, shape와 ndim을 사용
  - 형태 : 배열.shape
  - 차원 : 배열.ndim

NumPy 라이브러리

- 예시 - 리스트를 이용하여 배열 생성

```
import numpy as np

# 리스트를 이용하여 배열 생성
print('-------------------------------------------------')
list1 = [0, 1, 2, 3, 4]
arr1 = np.array(list1)
print('arr1')
print('{}₩n'.format(arr1))
print('배열 자료형: {}'.format(type(arr1)))
print('배열 형태: {}, 배열 차원: {}'.format(arr1.shape, arr1.ndim))

print('-------------------------------------------------')
arr2 = np.array([5, 6, 7, 8, 9, 10])
print('arr2')
print('{}₩n'.format(arr2))
print('배열 형태: {}, 배열 차원: {}'.format(arr2.shape, arr2.ndim))

print('-------------------------------------------------')
arr3 = np.array([[1, 3, 5], [2, 4, 6], [7, 8, 9]])
print('arr3')
print('{}₩n'.format(arr3))
print('배열 형태: {}, 배열 차원: {}'.format(arr3.shape, arr3.ndim))
```

```
-------------------------------------------------
arr1
[0 1 2 3 4]

배열 자료형: <class 'numpy.ndarray'>
배열 형태: (5,), 배열 차원: 1
-------------------------------------------------
arr2
[ 5  6  7  8  9 10]

배열 형태: (6,), 배열 차원: 1
-------------------------------------------------
arr3
[[1 3 5]
 [2 4 6]
 [7 8 9]]

배열 형태: (3, 3), 배열 차원: 2
```

NumPy 라이브러리

- 배열 생성
  - zeros() : 모든 값이 0인 배열을 생성
  - ones() : 모든 값이 1인 배열을 생성
  - full() : 특정 값으로 배열을 생성
  - eye() : 항등행렬(대각선으로 1이고 나머지는 0인 2차원 배열) 생성
  - arange() : 특정 규칙에 따라 증가하는 배열을 생성

- 생성한 배열의 자료형을 확인할 때, dtype 사용
  - 배열 생성 시, dtype 인수를 사용하여 자료형 선언할 수 있음

NumPy 라이브러리

- 예시 - zeros

```
import numpy as np

print('zeros()를 이용한 배열 생성₩n')

# zeros
print('-------------------------------------------------')
arr_zeros_1 = np.zeros(5)
print('크기가 5인 1차원 배열')
print('{}₩n'.format(arr_zeros_1))
print('자료형: {}'.format(arr_zeros_1.dtype))

print('-------------------------------------------------')
arr_zeros_2 = np.zeros((2, 3))
print('2개의 행과 3개의 열로 이루어진 2차원 배열, (2 * 3)')
print('{}₩n'.format(arr_zeros_2))
print('자료형: {}'.format(arr_zeros_2.dtype))

print('-------------------------------------------------')
arr_zeros_3 = np.zeros((5, 2))
print('5개의 행과 2개의 열로 이루어진 2차원 배열, (5 * 2)')
print('{}₩n'.format(arr_zeros_3))
print('자료형: {}'.format(arr_zeros_3.dtype))
```

```
zeros()를 이용한 배열 생성

-------------------------------------------------
크기가 5인 1차원 배열
[0. 0. 0. 0. 0.]

자료형: float64
-------------------------------------------------
2개의 행과 3개의 열로 이루어진 2차원 배열, (2 * 3)
[[0. 0. 0.]
 [0. 0. 0.]]

자료형: float64
-------------------------------------------------
5개의 행과 2개의 열로 이루어진 2차원 배열, (5 * 2)
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]

자료형: float64
```

NumPy 라이브러리

- 예시 - dtype 인수를 사용하여 자료형 선언

```
import numpy as np

print('zeros()를 이용한 배열 생성₩n')

# zeros
print('-------------------------------------------------')
arr_zeros_1 = np.zeros(5, dtype='i')
print('크기가 5인 1차원 배열')
print('{}₩n'.format(arr_zeros_1))
print('자료형: {}'.format(arr_zeros_1.dtype))

print('-------------------------------------------------')
arr_zeros_2 = np.zeros((2, 3), dtype='i')
print('2개의 행과 3개의 열로 이루어진 2차원 배열, (2 * 3)')
print('{}₩n'.format(arr_zeros_2))
print('자료형: {}'.format(arr_zeros_2.dtype))

print('-------------------------------------------------')
arr_zeros_3 = np.zeros((5, 2), dtype='i')
print('5개의 행과 2개의 열로 이루어진 2차원 배열, (5 * 2)')
print('{}₩n'.format(arr_zeros_3))
print('자료형: {}'.format(arr_zeros_3.dtype))
```

```
zeros()를 이용한 배열 생성

-------------------------------------------------
크기가 5인 1차원 배열
[0 0 0 0 0]

자료형: int32
-------------------------------------------------
2개의 행과 3개의 열로 이루어진 2차원 배열, (2 * 3)
[[0 0 0]
 [0 0 0]]

자료형: int32
-------------------------------------------------
5개의 행과 2개의 열로 이루어진 2차원 배열, (5 * 2)
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]]

자료형: int32
```

NumPy 라이브러리

- 예시 - ones

```
import numpy as np

print('ones()를 이용한 배열 생성₩n')

# ones
print('-------------------------------------------------')
arr_ones_1 = np.ones(5)
print('크기가 5인 1차원 배열')
print('{}₩n'.format(arr_ones_1))
print('자료형: {}'.format(arr_ones_1.dtype))

print('-------------------------------------------------')
arr_ones_2 = np.ones((2, 3))
print('2개의 행과 3개의 열로 이루어진 2차원 배열, (2 * 3)')
print('{}₩n'.format(arr_ones_2))
print('자료형: {}'.format(arr_ones_2.dtype))

print('-------------------------------------------------')
arr_ones_3 = np.ones((5, 2))
print('5개의 행과 2개의 열로 이루어진 2차원 배열, (5 * 2)')
print('{}₩n'.format(arr_ones_3))
print('자료형: {}'.format(arr_ones_3.dtype))
```

```
ones()를 이용한 배열 생성

-------------------------------------------------
크기가 5인 1차원 배열
[1. 1. 1. 1. 1.]

자료형: float64
-------------------------------------------------
2개의 행과 3개의 열로 이루어진 2차원 배열, (2 * 3)
[[1. 1. 1.]
 [1. 1. 1.]]

자료형: float64
-------------------------------------------------
5개의 행과 2개의 열로 이루어진 2차원 배열, (5 * 2)
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]

자료형: float64
```

NumPy 라이브러리

- 예시 - full

```
import numpy as np

print('full()를 이용한 배열 생성₩n')

# full
print('-------------------------------------------------')
arr_full_1 = np.full(5, 7)
print('크기가 5인 1차원 배열')
print('{}₩n'.format(arr_full_1))
print('자료형: {}'.format(arr_full_1.dtype))

print('-------------------------------------------------')
arr_full_2 = np.full((2, 3), 8)
print('2개의 행과 3개의 열로 이루어진 2차원 배열, (2 * 3)')
print('{}₩n'.format(arr_full_2))
print('자료형: {}'.format(arr_full_2.dtype))

print('-------------------------------------------------')
arr_full_3 = np.full((5, 2), 15)
print('5개의 행과 2개의 열로 이루어진 2차원 배열, (5 * 2)')
print('{}₩n'.format(arr_full_3))
print('자료형: {}'.format(arr_full_3.dtype))
```

```
full()를 이용한 배열 생성

-------------------------------------------------
크기가 5인 1차원 배열
[7 7 7 7 7]

자료형: int64
-------------------------------------------------
2개의 행과 3개의 열로 이루어진 2차원 배열, (2 * 3)
[[8 8 8]
 [8 8 8]]

자료형: int64
-------------------------------------------------
5개의 행과 2개의 열로 이루어진 2차원 배열, (5 * 2)
[[15 15]
 [15 15]
 [15 15]
 [15 15]
 [15 15]]

자료형: int64
```

NumPy 라이브러리

- 예시 - eye

```
import numpy as np

print('eye()를 이용한 배열 생성₩n')

# eye
print('-------------------------------------------------')
arr_eye_1 = np.eye(2)
print('(행과 열이 2인 2차원) 배열, (2 * 2)')
print('{}₩n'.format(arr_eye_1))
print('자료형: {}'.format(arr_eye_1.dtype))

print('-------------------------------------------------')
arr_eye_2 = np.eye(5)
print('행과 열이 5인 2차원 배열, (5 * 5)')
print('{}₩n'.format(arr_eye_2))
print('자료형: {}'.format(arr_eye_2.dtype))
```

```
eye()를 이용한 배열 생성

-------------------------------------------------
(행과 열이 2인 2차원) 배열, (2 * 2)
[[1. 0.]
 [0. 1.]]

자료형: float64
-------------------------------------------------
행과 열이 5인 2차원 배열, (5 * 5)
[[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.]]

자료형: float64
```

NumPy 라이브러리

- 예시 - arange

```
import numpy as np

print('arange()를 이용한 배열 생성₩n')

# arange
print('-------------------------------------------------')
arr_arange_1 = np.arange(5)
print('크기가 5인 1차원 배열')
print('{}₩n'.format(arr_arange_1))
print('자료형: {}'.format(arr_arange_1.dtype))

print('-------------------------------------------------')
arr_arange_2 = np.arange(10)
print('크기가 10인 1차원 배열')
print('{}₩n'.format(arr_arange_2))
print('자료형: {}'.format(arr_arange_2.dtype))
```

```
arange()를 이용한 배열 생성

-------------------------------------------------
크기가 5인 1차원 배열
[0 1 2 3 4]

자료형: int64
-------------------------------------------------
크기가 10인 1차원 배열
[0 1 2 3 4 5 6 7 8 9]

자료형: int64
```

NumPy 라이브러리

- 사칙연산
  - 배열의 형태가 동일할 때, 사칙연산이 가능함
  - 행렬의 사칙연산과 동일함

- 행렬 곱(Dot Product)
  - dot 함수를 사용하여 행렬 곱을 계산함
  - 좌측 행렬의 열(Column) 수와 우측 행렬의 행(Row) 수가 서로 같아야 함

- 브로드캐스팅(Broadcasting)
  - 형태가 다른 배열 간에 연산을 자동으로 지원해 줌

NumPy 라이브러리

- 예시 - 사칙연산

```
import numpy as np

# 배열 생성
print('-------------------------------------------------')
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
arr2 = np.array([[7, 8, 9], [10, 11, 12]])

print('arr1 자료형: {}'.format(arr1.dtype))
print('{}₩n'.format(arr1))
print('arr2 자료형: {}'.format(arr2.dtype))
print('{}'.format(arr2))
```

```
-------------------------------------------------
arr1 자료형: int64
[[1 2 3]
 [4 5 6]]

arr2 자료형: int64
[[ 7  8  9]
 [10 11 12]]
```

NumPy 라이브러리

- 예시 - 사칙연산

```
# 사칙연산 (덧셈)
print('-------------------------------------------------')
print('덧셈 결과 (arr1 + arr2)')
print(arr1 + arr2)

# 사칙연산 (뺄셈)
print('-------------------------------------------------')
print('뺄셈 결과 (arr1 - arr2)')
print(arr1 - arr2)

# 사칙연산 (곱셈)
print('-------------------------------------------------')
print('곱셈 결과 (arr1 * arr2)')
print(arr1 * arr2)

# 사칙연산 (나눗셈)
print('-------------------------------------------------')
print('나눗셈 결과 (arr1 / arr2)')
print(arr1 / arr2)
```

```
-------------------------------------------------
덧셈 결과 (arr1 + arr2)
[[ 8 10 12]
 [14 16 18]]
-------------------------------------------------
뺄셈 결과 (arr1 - arr2)
[[-6 -6 -6]
 [-6 -6 -6]]
-------------------------------------------------
곱셈 결과 (arr1 * arr2)
[[ 7 16 27]
 [40 55 72]]
-------------------------------------------------
나눗셈 결과 (arr1 / arr2)
[[0.14285714 0.25       0.33333333]
 [0.4        0.45454545 0.5       ]]
```

NumPy 라이브러리

- 예시 - 행렬 곱

```python
import numpy as np

# 배열 생성
print('-------------------------------------------------')
arr1 = np.array([1, 2])

print('arr1 자료형: {}'.format(arr1.dtype))
print('{}'.format(arr1))

# 행렬 곱
print('-------------------------------------------------')
print('arr1와 5의 행렬 곱 결과')
print(np.dot(arr1, 5))
```

```
-------------------------------------------------
arr1 자료형: int64
[1 2]
-------------------------------------------------
arr1와 5의 행렬 곱 결과
[ 5 10]
```

NumPy 라이브러리

- 예시 - 행렬 곱

```
# 배열 생성
print('-------------------------------------------------')
arr2 = np.array([1, 2, 3])
arr3 = np.array([[4, 5, 6], [7, 8, 9], [10, 11, 12]])

print('arr2 자료형: {}'.format(arr2.dtype))
print('{}₩n'.format(arr2))
print('arr3 자료형: {}'.format(arr3.dtype))
print('{}'.format(arr3))

# 행렬 곱
print('-------------------------------------------------')
print('arr2와 arr3의 행렬 곱 결과')
print(np.dot(arr2, arr3))
```

```
-------------------------------------------------
arr2 자료형: int64
[1 2 3]

arr3 자료형: int64
[[ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
-------------------------------------------------
arr2와 arr3의 행렬 곱 결과
[48 54 60]
```

NumPy 라이브러리

- 예시 - 행렬 곱

```
# 배열 생성
print('-------------------------------------------------')
arr4 = np.array([[1, 2, 3], [4, 5, 6]])
arr5 = np.array([[7, 8], [9, 10], [11, 12]])

print('arr4 자료형: {}'.format(arr4.dtype))
print('{}₩n'.format(arr4))
print('arr5 자료형: {}'.format(arr5.dtype))
print('{}'.format(arr5))

# 행렬 곱
print('-------------------------------------------------')
print('arr4와 arr5의 행렬 곱 결과')
print(np.dot(arr4, arr5))
```

```
-------------------------------------------------
arr4 자료형: int64
[[1 2 3]
 [4 5 6]]

arr5 자료형: int64
[[ 7  8]
 [ 9 10]
 [11 12]]
-------------------------------------------------
arr4와 arr5의 행렬 곱 결과
[[ 58  64]
 [139 154]]
```

NumPy 라이브러리

- 예시 - 브로드캐스팅

```
import numpy as np

# 배열 생성
print('-----------------------------------------------')
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
scalar = 3

print('arr1 자료형: {}'.format(arr1.dtype))
print('{}₩n'.format(arr1))
print('scalar: {}'.format(scalar))
```

```
-----------------------------------------------
arr1 자료형: int64
[[1 2 3]
 [4 5 6]]

scalar: 3
```

NumPy 라이브러리

- 예시 - 브로드캐스팅

```
# 브로드캐스팅 (덧셈)
print('-----------------------------------------------')
print('덧셈 결과 (arr1 + scalar)')
print(arr1 + scalar)

# 브로드캐스팅 (뺄셈)
print('-----------------------------------------------')
print('뺄셈 결과 (arr1 - scalar)')
print(arr1 - scalar)

# 브로드캐스팅 (곱셈)
print('-----------------------------------------------')
print('곱셈 결과 (arr1 * scalar)')
print(arr1 * scalar)

# 브로드캐스팅 (나눗셈)
print('-----------------------------------------------')
print('나눗셈 결과 (arr1 / scalar)')
print(arr1 / scalar)
```

```
-----------------------------------------------
덧셈 결과 (arr1 + scalar)
[[4 5 6]
 [7 8 9]]
-----------------------------------------------
뺄셈 결과 (arr1 - scalar)
[[-2 -1  0]
 [ 1  2  3]]
-----------------------------------------------
곱셈 결과 (arr1 * scalar)
[[ 3  6  9]
 [12 15 18]]
-----------------------------------------------
나눗셈 결과 (arr1 / scalar)
[[0.33333333 0.66666667 1.        ]
 [1.33333333 1.66666667 2.        ]]
```

NumPy 라이브러리

- 예시 - 브로드캐스팅

```
import numpy as np

# 배열 생성
print('-------------------------------------------------')
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
arr2 = np.arange(10, 40, 10)

print('arr1 자료형: {}'.format(arr1.dtype))
print('{}₩n'.format(arr1))
print('arr2 자료형: {}'.format(arr2.dtype))
print('{}'.format(arr2))
```

```
-------------------------------------------------
arr1 자료형: int64
[[1 2 3]
 [4 5 6]]

arr2 자료형: int64
[10 20 30]
```

NumPy 라이브러리

- 예시 - 브로드캐스팅

```
# 브로드캐스팅 (덧셈)
print('-----------------------------------------------')
print('덧셈 결과 (arr1 + arr2)')
print(arr1 + arr2)

# 브로드캐스팅 (뺄셈)
print('-----------------------------------------------')
print('뺄셈 결과 (arr1 - arr2)')
print(arr1 - arr2)

# 브로드캐스팅 (곱셈)
print('-----------------------------------------------')
print('곱셈 결과 (arr1 * arr2)')
print(arr1 * arr2)

# 브로드캐스팅 (나눗셈)
print('-----------------------------------------------')
print('나눗셈 결과 (arr1 / arr2)')
print(arr1 / arr2)
```

```
-----------------------------------------------
덧셈 결과 (arr1 + arr2)
[[11 22 33]
 [14 25 36]]
-----------------------------------------------
뺄셈 결과 (arr1 - arr2)
[[ -9 -18 -27]
 [ -6 -15 -24]]
-----------------------------------------------
곱셈 결과 (arr1 * arr2)
[[ 10  40  90]
 [ 40 100 180]]
-----------------------------------------------
나눗셈 결과 (arr1 / arr2)
[[0.1  0.1  0.1 ]
 [0.4  0.25 0.2 ]]
```

NumPy 라이브러리

- 배열 요소 접근
  - 배열도 리스트와 마찬가지로 슬라이스(Slice) 기능을 지원함
  - 배열 각 요소를 Boolean(True, False)으로 표현이 가능함
- 배열 요소의 합과 배열 요소 중 최댓값과 최솟값의 색인 위치 또는 조건에 맞는 색인 위치를 찾아주는 함수를 제공
  - 합 : np.sum(배열)
  - 최댓값 : np.max(배열)
  - 최댓값 색인 위치 : np.argmax(배열)
  - 최솟값 : np.min(배열)
  - 최솟값 색인 위치 : np.argmin(배열)
  - 조건에 맞는 색인 위치 : np.where(조건식)
  - axis=0 : 배열의 요소에서 열(Column)을 기준으로 함
  - axis=1 : 배열의 요소에서 행(Row)을 기준으로 함

NumPy 라이브러리

- 예시 - 배열 요소 접근

```
import numpy as np

# 리스트를 이용하여 배열 생성
print('-------------------------------------------------')
arr1 = np.array([5, 6, 7, 8, 9, 10])
print('arr1')
print('{}'.format(arr1))

print('-------------------------------------------------')
print('arr1[0]: {}'.format(arr1[0]))
print('arr1[3]: {}'.format(arr1[3]))
print('arr1[-1]: {}'.format(arr1[-1]))

print('-------------------------------------------------')
print('arr1[1:3]: {}'.format(arr1[1:3]))
print('arr1[:-1]: {}'.format(arr1[:-1]))
```

```
-------------------------------------------------
arr1
[ 5  6  7  8  9 10]
-------------------------------------------------
arr1[0]: 5
arr1[3]: 8
arr1[-1]: 10
-------------------------------------------------
arr1[1:3]: [6 7]
arr1[:-1]: [5 6 7 8 9]
```

NumPy 라이브러리

- 예시 - 배열 요소 접근

```
print('-------------------------------------------------')
arr2 = np.array([[1, 3, 5], [2, 4, 6], [7, 8, 9]])
print('arr2')
print('{}'.format(arr2))

print('-------------------------------------------------')
print('arr2[0]: {}'.format(arr2[0]))
print('arr2[1]: {}'.format(arr2[1]))
print('arr2[-1]: {}'.format(arr2[-1]))

print('-------------------------------------------------')
print('arr2[1:]₩n{}₩n'.format(arr2[1:]))
print('arr2[:1]: {}'.format(arr2[:1]))

print('-------------------------------------------------')
print('arr2[0][0]: {}'.format(arr2[0][0]))
print('arr2[1][-1]: {}'.format(arr2[1][-1]))
print('arr2[-1][1]: {}'.format(arr2[-1][1]))

print('-------------------------------------------------')
print('arr2[0][:-1]: {}'.format(arr2[0][:-1]))
print('arr1[1][:1]: {}'.format(arr2[1][:1]))
```

```
-------------------------------------------------
arr2
[[1 3 5]
 [2 4 6]
 [7 8 9]]
-------------------------------------------------
arr2[0]: [1 3 5]
arr2[1]: [2 4 6]
arr2[-1]: [7 8 9]
-------------------------------------------------
arr2[1:]
[[2 4 6]
 [7 8 9]]

arr2[:1]: [[1 3 5]]
-------------------------------------------------
arr2[0][0]: 1
arr2[1][-1]: 6
arr2[-1][1]: 8
-------------------------------------------------
arr2[0][:-1]: [1 3]
arr1[1][:1]: [2]
```

NumPy 라이브러리

- 예시 - 배열 요소 Boolean(True, False)

```python
import numpy as np

# 배열 생성
print('-----------------------------------------------------')
arr = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
print('arr₩n{}'.format(arr))

# boolean indexing 생성
print('-----------------------------------------------------')
bool_index_arr = np.array([[True, False, True],
                           [False, True, False],
                           [True, True, False]])
print('bool_index_arr₩n{}'.format(bool_index_arr))

# 배열 요소 중 True인 요소를 반환
print('-----------------------------------------------------')
result = arr[bool_index_arr]
print('결과: {}'.format(result))
```

```
-----------------------------------------------------
arr
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-----------------------------------------------------
bool_index_arr
[[ True False  True]
 [False  True False]
 [ True  True False]]
-----------------------------------------------------
결과: [1 3 5 7 8]
```

NumPy 라이브러리

- 예시 - 배열 요소의 합

```
import numpy as np

# 리스트를 이용하여 배열 생성
print('-------------------------------------------------')
arr1 = np.array([5, 6, 7, 8, 9, 10])
print('arr1')
print('{}'.format(arr1))

# 배열 요소의 합
print('-------------------------------------------------')
print('arr1 요소의 합: {}'.format(np.sum(arr1)))
```

```
-------------------------------------------------
arr1
[ 5  6  7  8  9 10]
-------------------------------------------------
arr1 요소의 합: 45
```

NumPy 라이브러리

- 예시 - 배열 요소의 합

```
print('-----------------------------------------------')
arr2 = np.array([[1, 3, 5], [2, 4, 6]])
print('arr2')
print('{}'.format(arr2))

# 배열 요소의 합
print('-----------------------------------------------')
print('arr2 요소의 합: {}'.format(np.sum(arr2)))

# 열(Column)을 기준으로 배열 요소의 합
print('-----------------------------------------------')
print('열(Column) 기준₩narr2 요소의 합: {}'.format(np.sum(arr2, axis=0)))

# 행(row)을 기준으로 배열 요소의 합
print('-----------------------------------------------')
print('행(row) 기준₩narr2 요소의 합: {}'.format(np.sum(arr2, axis=1)))
```

```
-----------------------------------------------
arr2
[[1 3 5]
 [2 4 6]]
-----------------------------------------------
arr2 요소의 합: 21
-----------------------------------------------
열(Column) 기준
arr2 요소의 합: [ 3  7 11]
-----------------------------------------------
행(row) 기준
arr2 요소의 합: [ 9 12]
```

NumPy 라이브러리

- 예시 - 배열 요소 중 최댓값과 최솟값의 색인 위치

```python
import numpy as np

# 배열 생성
print('-------------------------------------------------')
arr1 = np.array([22, 3, 90, 53, 77])

print('arr1 자료형: {}'.format(arr1.dtype))
print('{}'.format(arr1))

# 최댓값과 최댓값 색인 위치
print('-------------------------------------------------')
print('최댓값: {}, 색인 위치: {}'.format(np.max(arr1), np.argmax(arr1)))

# 최솟값과 최솟값 색인 위치
print('-------------------------------------------------')
print('최솟값: {}, 색인 위치: {}'.format(np.min(arr1), np.argmin(arr1)))
```

```
-------------------------------------------------
arr1 자료형: int64
[22  3 90 53 77]
-------------------------------------------------
최댓값: 90, 색인 위치: 2
-------------------------------------------------
최솟값: 3, 색인 위치: 1
```

NumPy 라이브러리

- 예시 - 배열 요소 중 최댓값과 최솟값의 색인 위치

```
# 배열 생성
print('-------------------------------------------------')
arr2 = np.array([[1, 3, 5, 6], [55, 34, 12, 78], [31, 82, 45, 68]])

print('arr2 자료형: {}'.format(arr2.dtype))
print('{}'.format(arr2))
```

```
-------------------------------------------------
arr2 자료형: int64
[[ 1  3  5  6]
 [55 34 12 78]
 [31 82 45 68]]
```

NumPy 라이브러리

- 예시 - 배열 요소 중 최댓값과 최솟값의 색인 위치

```
# 최댓값과 최댓값 색인 위치
print('-----------------------------------------------')
print('최댓값: {}, 색인 위치: {}'.format(np.max(arr2), np.argmax(arr2)))

# 열(Column)을 기준으로 최댓값과 최댓값 색인 위치
print('-----------------------------------------------')
print('열(Column) 기준₩n최댓값: {}, 색인 위치: {}'.
      format(np.max(arr2, axis=0), np.argmax(arr2, axis=0)))

# 행(row)을 기준으로 최댓값과 최댓값 색인 위치
print('-----------------------------------------------')
print('행(row) 기준₩n최댓값: {}, 색인 위치: {}'.
      format(np.max(arr2, axis=1), np.argmax(arr2, axis=1)))
```

```
-----------------------------------------------
최댓값: 82, 색인 위치: 9
-----------------------------------------------
열(Column) 기준
최댓값: [55 82 45 78], 색인 위치: [1 2 2 1]
-----------------------------------------------
행(row) 기준
최댓값: [ 6 78 82], 색인 위치: [3 3 1]
```

NumPy 라이브러리

- 예시 - 배열 요소 중 최댓값과 최솟값의 색인 위치

```
# 최솟값과 최솟값 색인 위치
print('-------------------------------------------------')
print('최솟값: {}, 색인 위치: {}'.format(np.min(arr2), np.argmin(arr2)))

# 열(Column)을 기준으로 최솟값과 최솟값 색인 위치
print('-------------------------------------------------')
print('열(Column) 기준₩n최솟값: {}, 색인 위치: {}'.
      format(np.min(arr2, axis=0), np.argmin(arr2, axis=0)))

# 행(row)을 기준으로 최솟값과 최솟값 색인 위치
print('-------------------------------------------------')
print('행(row) 기준₩n최솟값: {}, 색인 위치: {}'.
      format(np.min(arr2, axis=1), np.argmin(arr2, axis=1)))
```

```
-------------------------------------------------
최솟값: 1, 색인 위치: 0
-------------------------------------------------
열(Column) 기준
최솟값: [1 3 5 6], 색인 위치: [0 0 0 0]
-------------------------------------------------
행(row) 기준
최솟값: [ 1 12 31], 색인 위치: [0 2 0]
```

NumPy 라이브러리

- 예시 - 배열 요소 중 조건에 맞는 색인 위치
  - np.where(조건식) : 조건식에 해당하는 색인의 위치를 나타냄
  - 배열[np.where(조건식)] : 조건식에 해당하는 배열의 요소를 나타냄
  - np.where(조건식, 조건식에 해당할 때 값, 조건식에 해당하지 않을 때 값)
    - 조건식에 해당하면 조건식에 해당할 때 값을 수행함
    - 조건식에 해당하지 않으면 조건식에 해당하지 않을 때 값을 수행함

```
import numpy as np

# 배열 생성
print('-----------------------------------------------')
arr1 = np.array([22, 3, 90, 53, 77])

print('arr1 자료형: {}'.format(arr1.dtype))
print('{}'.format(arr1))
```

```
-----------------------------------------------
arr1 자료형: int64
[22  3 90 53 77]
```

NumPy 라이브러리

- 예시 - 배열 요소 중 조건에 맞는 색인 위치

```
import numpy as np

# 배열 생성
print('-------------------------------------------------')
arr1 = np.array([22, 3, 90, 53, 77])

print('arr1 자료형: {}'.format(arr1.dtype))
print('{}'.format(arr1))

# 배열 요소 중 50보다 큰 값과 색인 위치를 표시
print('-------------------------------------------------')
print('50보다 큰 값: {}'.format(arr1[np.where(arr1 > 50)]))
print('50보다 큰 값의 색인 위치: {}'.format(np.where(arr1 > 50)))

# 배열 요소 중 50보다 작은 값을 30으로 변환
print('-------------------------------------------------')
print('50보다 작은 값: {}'.format(arr1[np.where(arr1 < 50)]))
print('50보다 작은 값의 색인 위치: {}'.format(np.where(arr1 < 50)))
print('50보다 작은 값을 30으로 변환: {}'.format(np.where(arr1 < 50, 30, arr1)))
```

```
-------------------------------------------------
arr1 자료형: int64
[22  3 90 53 77]
-------------------------------------------------
50보다 큰 값: [90 53 77]
50보다 큰 값의 색인 위치: (array([2, 3, 4]),)
-------------------------------------------------
50보다 작은 값: [22  3]
50보다 작은 값의 색인 위치: (array([0, 1]),)
50보다 작은 값을 30으로 변환: [30 30 90 53 77]
```

NumPy 라이브러리

- 배열의 형태를 변형할 때, reshape를 사용
  - 배열.reshape

- 다차원 배열을 1차원 배열로 변형할 때, flatten을 사용
  - 배열.flatten

- 배열의 형태를 전치(Transpose)할 때, T를 사용
  - 배열.T

NumPy 라이브러리

- 예시 - 배열 형태 변환

```python
import numpy as np

# 배열 생성
print('-----------------------------------------------')
arr = np.array([1, 2, 3, 4, 5, 6])
print('arr: {}'.format(arr))
print('arr 배열의 형태: {}'.format(arr.shape))

# (2 * 3) 배열로 형태 변환
print('-----------------------------------------------')
r_arr = arr.reshape(2, 3)
print('r_arr₩n{}₩n'.format(r_arr))
print('r_arr 배열의 형태: {}'.format(r_arr.shape))

# (3 * 2) 배열로 형태 변환
print('-----------------------------------------------')
re_arr = r_arr.reshape(3, 2)
print('re_arr₩n{}₩n'.format(re_arr))
print('re_arr 배열의 형태: {}'.format(re_arr.shape))

# 다차원 배열을 1차원 배열로 변환
print('-----------------------------------------------')
f_arr = re_arr.flatten()
print('f_arr₩n{}₩n'.format(f_arr))
print('f_arr 배열의 형태: {}'.format(f_arr.shape))
```

```
-----------------------------------------------
arr: [1 2 3 4 5 6]
arr 배열의 형태: (6,)
-----------------------------------------------

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

r_arr 배열의 형태: (2, 3)
-----------------------------------------------

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

re_arr 배열의 형태: (3, 2)
-----------------------------------------------

f_arr
[1 2 3 4 5 6]

f_arr 배열의 형태: (6,)
```

NumPy 라이브러리

- 예시 - 배열 전치

```python
import numpy as np

# 배열 생성
print('-----------------------------------------------')
arr = np.arange(1, 13).reshape(3, 4)
print('arr₩n{}₩n'.format(arr))
print('arr 배열의 형태: {}'.format(arr.shape))

# 전치 변환
print('-----------------------------------------------')
t_arr = arr.T
print('t_arr₩n{}₩n'.format(t_arr))
print('t_arr 배열의 형태: {}'.format(t_arr.shape))
```

```
-----------------------------------------------
arr
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

arr 배열의 형태: (3, 4)
-----------------------------------------------
t_arr
[[ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]
 [ 4  8 12]]

t_arr 배열의 형태: (4, 3)
```

실습

- 실습은 예시를 따라 하는 것으로 대체함

과제

- 소스 코드와 결과를 캡쳐하여 문서(워드, 한글)에 정리한 후, 스마트 캠퍼스에 제출
  - 제출 형식 : 교과목명_주차_학번_이름.docx or .hwp (제출 형식 안 맞을 시 감점)
  - 기간 내에 제출하지 못할 경우 이메일로 제출 (감점)
  - 주석 작성 필수 (주석 없을 시 감점)
  - 부정 행위 적발 시 감점 (컨닝)

과제1

- 1부터 50까지의 요소를 가지는 배열을 생성하시오.
  - 결과

```
1 ~ 50까지의 요소를 가지는 배열 생성
[ 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]
```

- 배열 요소 중 홀수와 짝수를 분류하여 배열로 저장하시오.
  - 단, for문과 if문 사용하지 않음
  - 배열로 저장 시, 다음과 같은 형태로 변환
  - 결과

```
배열 요소 중 홀수인 값
[[ 1  3  5  7  9]
 [11 13 15 17 19]
 [21 23 25 27 29]
 [31 33 35 37 39]
 [41 43 45 47 49]]
```

```
배열 요소 중 짝수인 값
[[ 2  4  6  8 10]
 [12 14 16 18 20]
 [22 24 26 28 30]
 [32 34 36 38 40]
 [42 44 46 48 50]]
```

과제1

- 홀수 배열과 짝수 배열을 전치 변환하시오.
  - 결과

```
홀수 배열을 전치 변환
[[ 1 11 21 31 41]
 [ 3 13 23 33 43]
 [ 5 15 25 35 45]
 [ 7 17 27 37 47]
 [ 9 19 29 39 49]]
```

```
짝수 배열을 전치 변환
[[ 2 12 22 32 42]
 [ 4 14 24 34 44]
 [ 6 16 26 36 46]
 [ 8 18 28 38 48]
 [10 20 30 40 50]]
```

- 전치 변환한 홀수 배열과 짝수 배열을 행렬 곱 연산을 수행하시오.
  - 결과

```
전치 변환한 홀수 배열과 짝수 배열의 행렬 곱 연산
[[ 830 1880 2930 3980 5030]
 [ 890 2040 3190 4340 5490]
 [ 950 2200 3450 4700 5950]
 [1010 2360 3710 5060 6410]
 [1070 2520 3970 5420 6870]]
```

과제2

- 두 배열이 있을 때, 아래의 문제들을 해결하시오.
  - 첫 번째 배열 = [1, 2, 3, 5, 3, 4, 3, 6, 9, 7, 0, 8, 7, 10]
  - 두 번째 배열 = [7, 2, 10, 5, 7, 4, 9, 1, 9, 8, 0, 3, 7, 6]

- 두 배열에 모두 존재하는 요소와 색인 위치를 반환하시오.
  - 단, for문과 if문 사용하지 않음

  - 결과

```
두 배열에 모두 존재하는 요소: [2 5 4 9 0 7]
두 배열에 모두 존재하는 요소의 색인 위치: (array([ 1,  3,  5,  8, 10, 12]),)
```

과제2

- 두 배열이 있을 때, 아래의 문제들을 해결하시오.
  - 첫 번째 배열 = [1, 2, 3, 5, 3, 4, 3, 6, 9, 7, 0, 8, 7, 10]
  - 두 번째 배열 = [7, 2, 10, 5, 7, 4, 9, 1, 9, 8, 0, 3, 7, 6]

- 반환된 배열을 다음과 같은 형태로 변형하시오.
  - 결과

```
배열 형태 변형
[[2 5 4]
 [9 0 7]]
```

과제2

- 두 배열이 있을 때, 아래의 문제들을 해결하시오.
  - 첫 번째 배열 = [1, 2, 3, 5, 3, 4, 3, 6, 9, 7, 0, 8, 7, 10]
  - 두 번째 배열 = [7, 2, 10, 5, 7, 4, 9, 1, 9, 8, 0, 3, 7, 6]

- 변형된 배열의 열(Column)을 기준으로 배열의 최댓값과 최댓값의 색인 위치를 구하시오.
- 변형된 배열의 행(Row)을 기준으로 배열의 최솟값과 최솟값의 색인 위치를 구하시오.
  - 결과

```
열(Column)을 기준으로
배열의 최댓값: [9 5 7], 최댓값의 색인 위치: [1 0 1]
행(Row)을 기준으로
배열의 최솟값: [2 0], 최솟값의 색인 위치: [0 1]
```

감사합니다