

Image Processing

실습 1주차

신 동 현

Department of Computer Science and Engineering
Chungnam National University, Korea



실습 소개

- 과목 홈페이지
 - 충남대학교 사이버 캠퍼스 (<http://e-learn.cnu.ac.kr>)
- TA 연락처
 - 공대 5호관 531호 컴퓨터비전 연구실
 - 과제 질문은 [IP]를 제목에 붙여 메일로 주세요.
 - 00반
 - 안준혁
 - ajh99345@gmail.com
 - 01반
 - 신동헌
 - doghon85@naver.com

목차

- **환경 구축**
 - Pycharm 설치 및 setting
- **실습**
 - Python 기초
 - Numpy, openCV, matplotlib 기초

- **Pycharm 설치 및 setting**
 - Python version: 3.8.10
 - IDE: Pycharm
 - 설치할 패키지: openCV, numpy, matplotlib

- Pycharm 설치

- <https://www.jetbrains.com/ko-kr/pycharm/download/?section=windows> 접속 후
다운로드 진행

활기찬 Python 커뮤니티에 대한 감사의 마음을 담아 Python 에
코시스템을 지원하는 오픈소스 기여 활동으로 PyCharm
Community Edition을 무상으로 제공합니다.

 **PyCharm Community Edition**

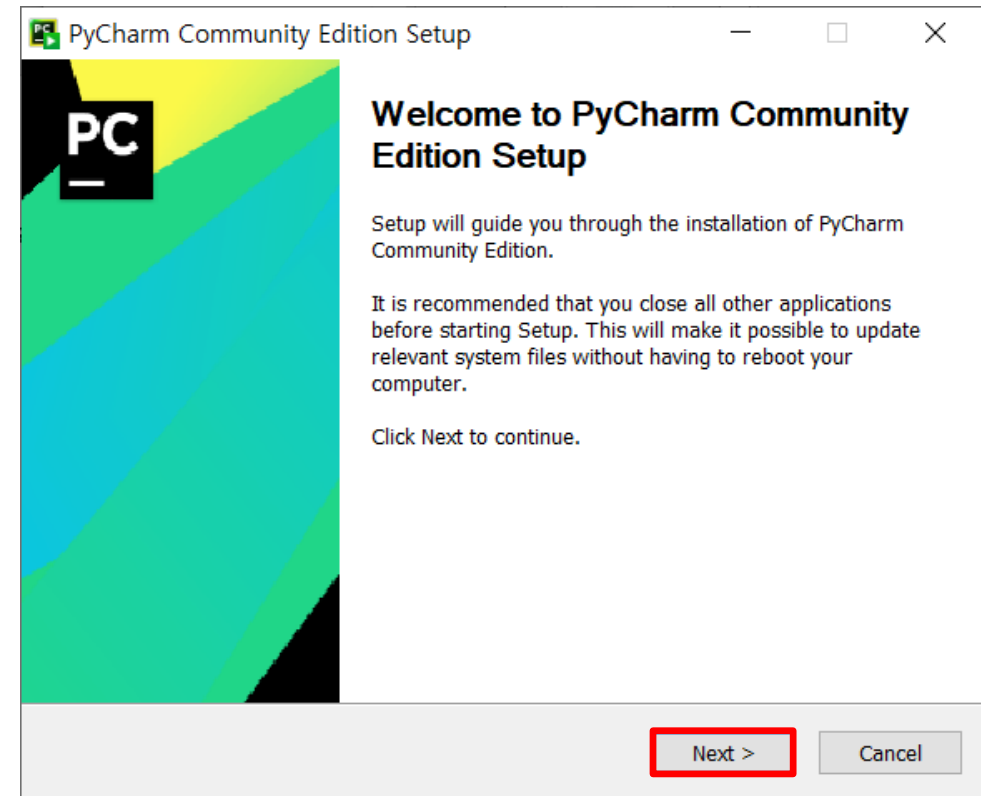
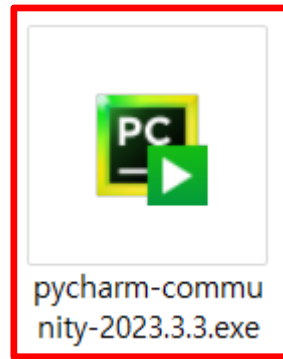
순수 Python 개발용 IDE

다운로드

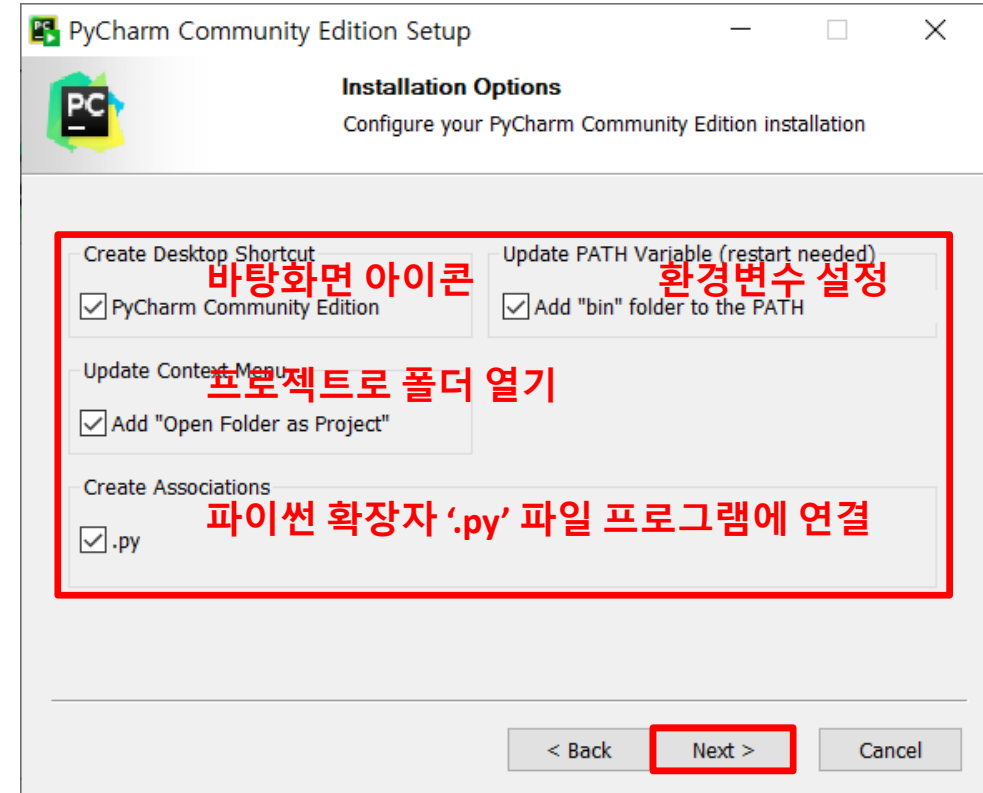
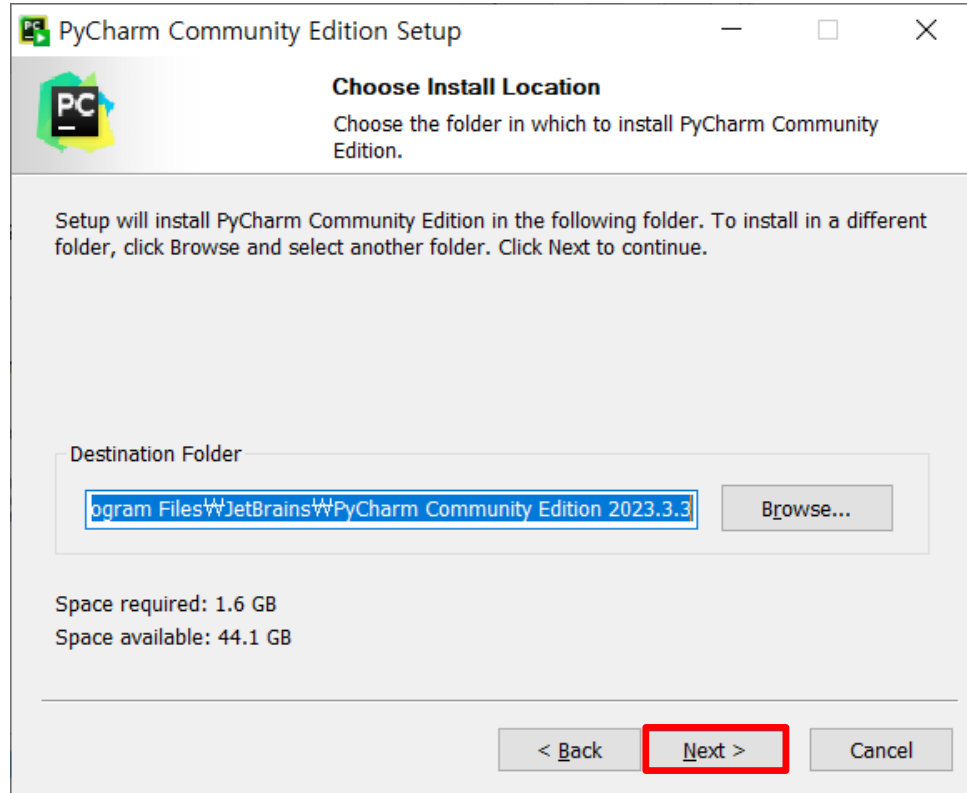
.exe ▼

무료, 오픈 소스로 빌드됨

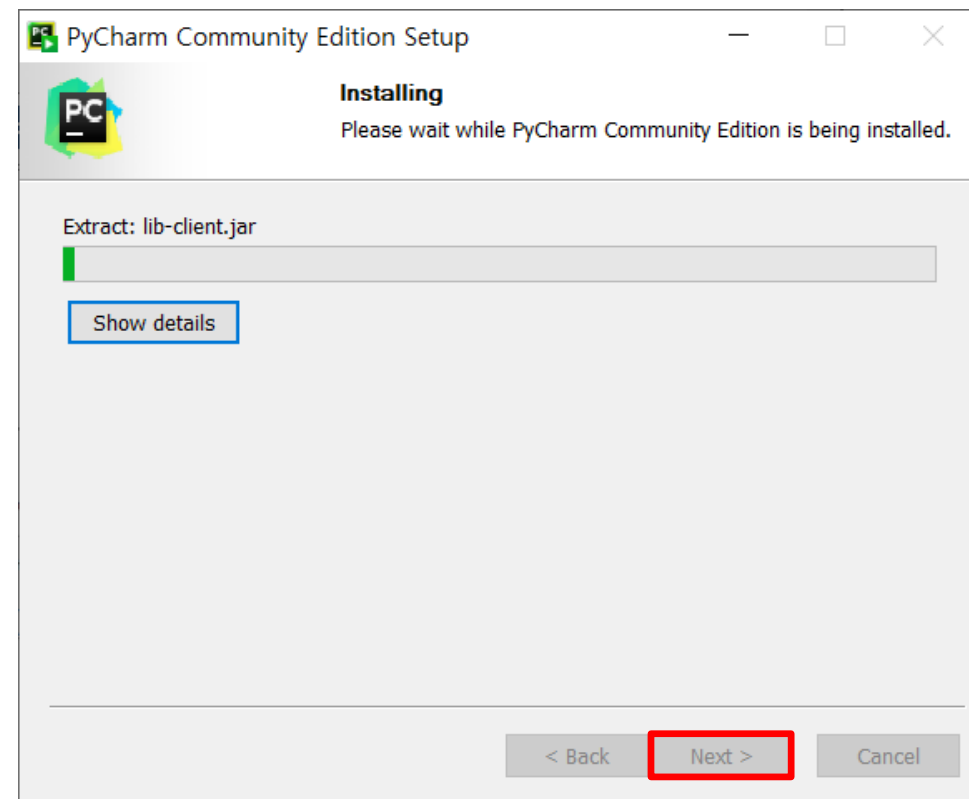
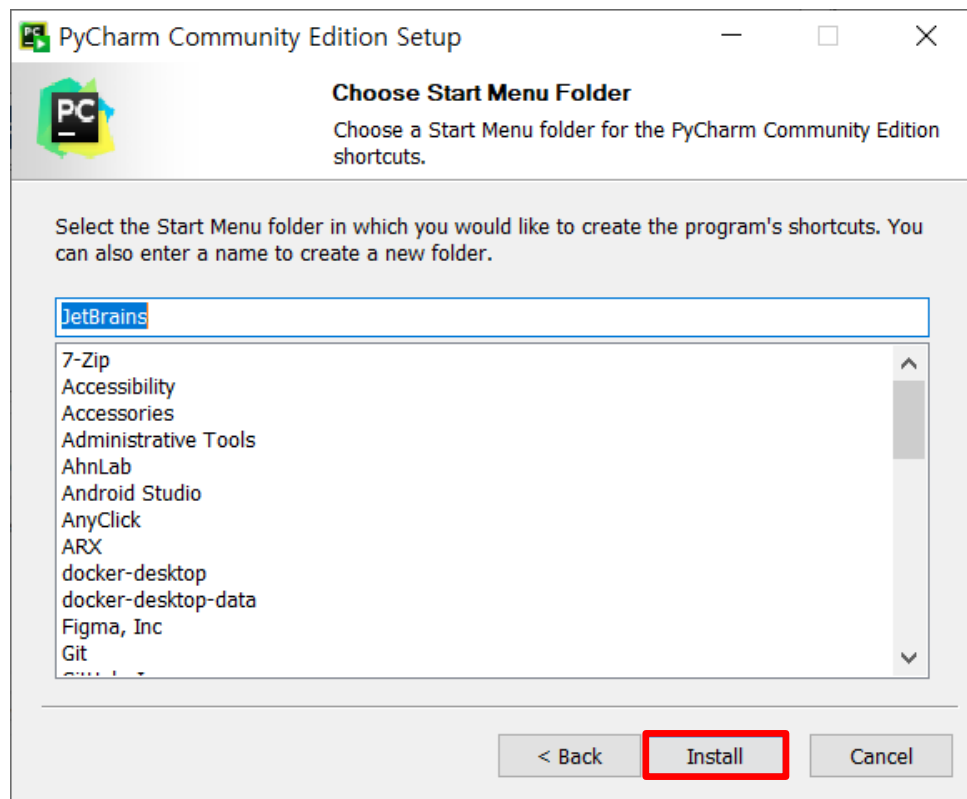
- Pycharm 설치
 - 다운로드 된 파일 실행하여 pycharm 설치



- Pycharm 설치
 - 박스 모두 체크하고 진행

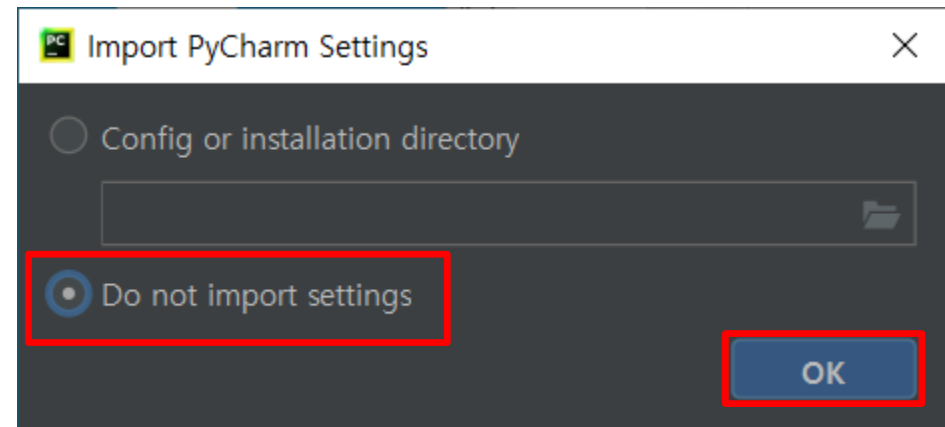
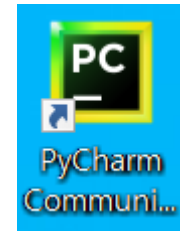
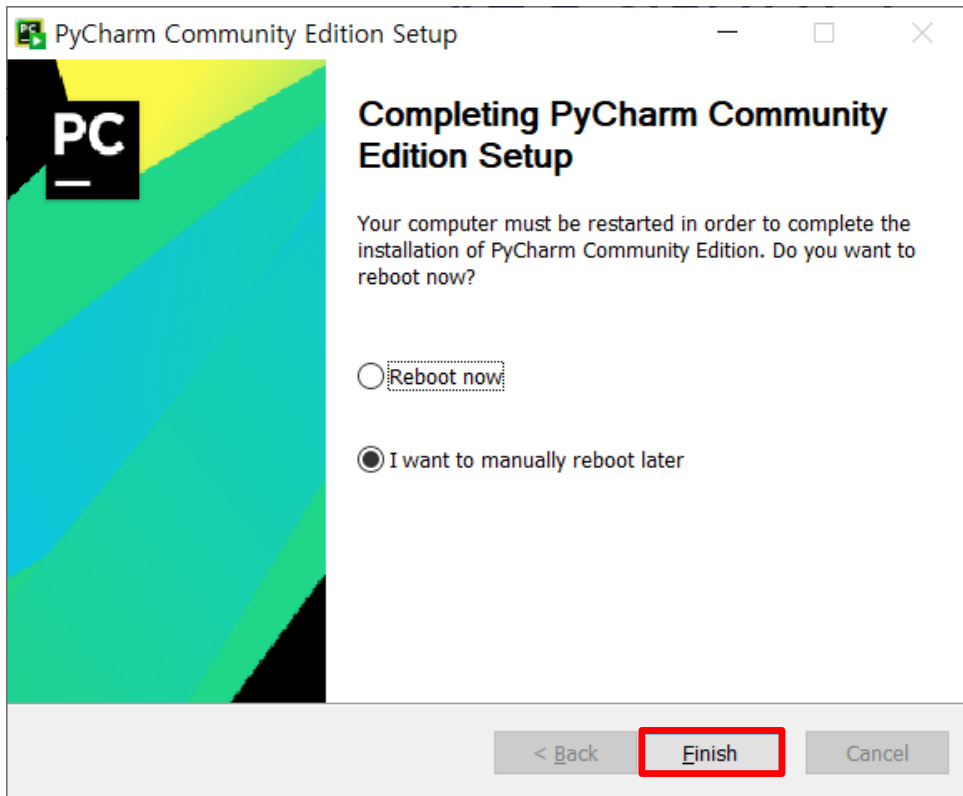


- Pycharm 설치



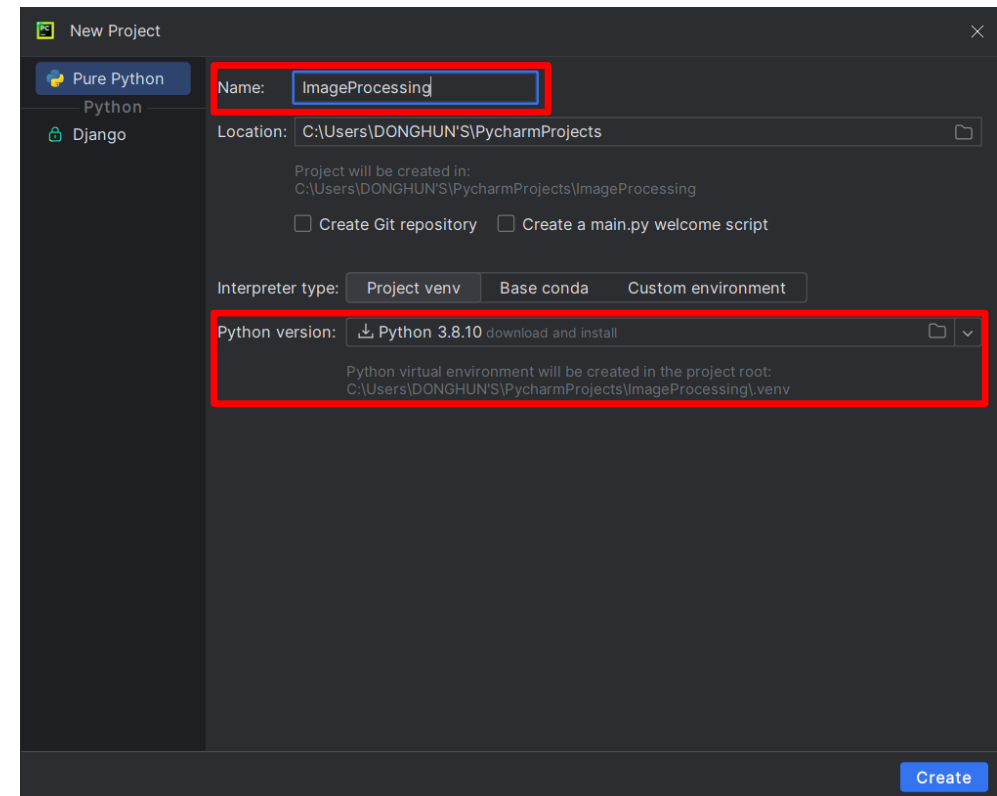
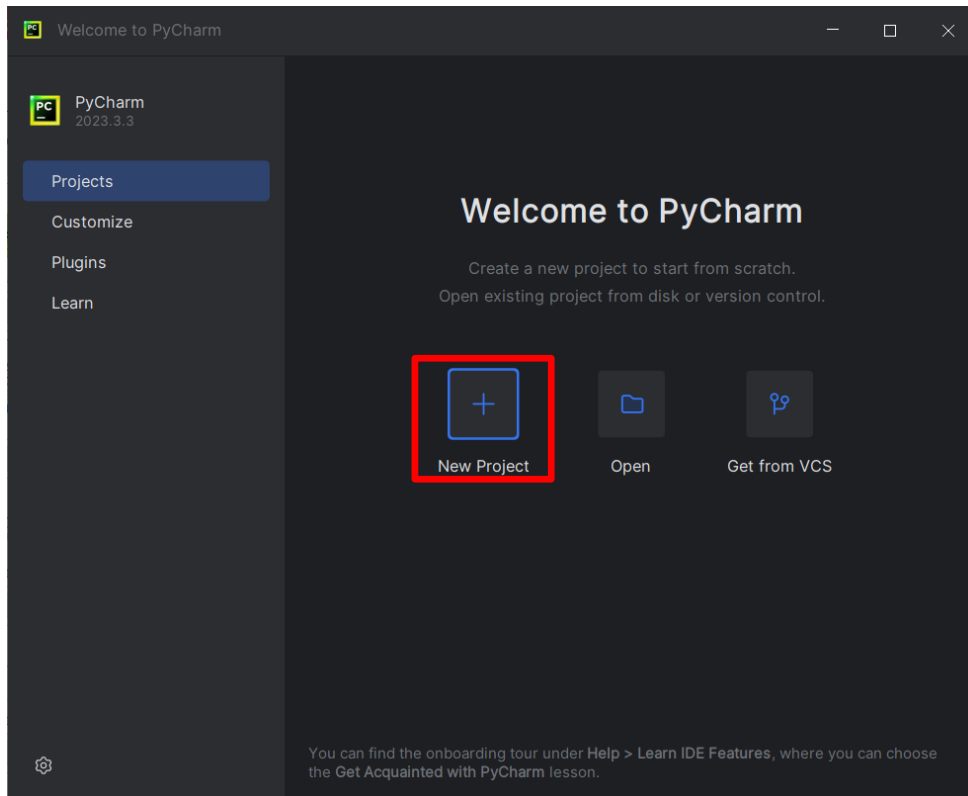
- Pycharm 설치

- 설치 후 바탕화면의 pycharm 파일 실행하여 아래처럼 진행



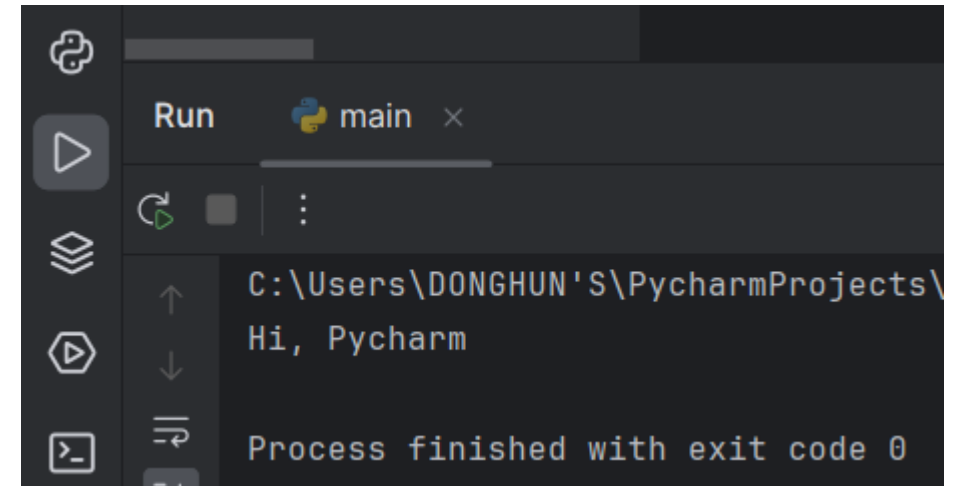
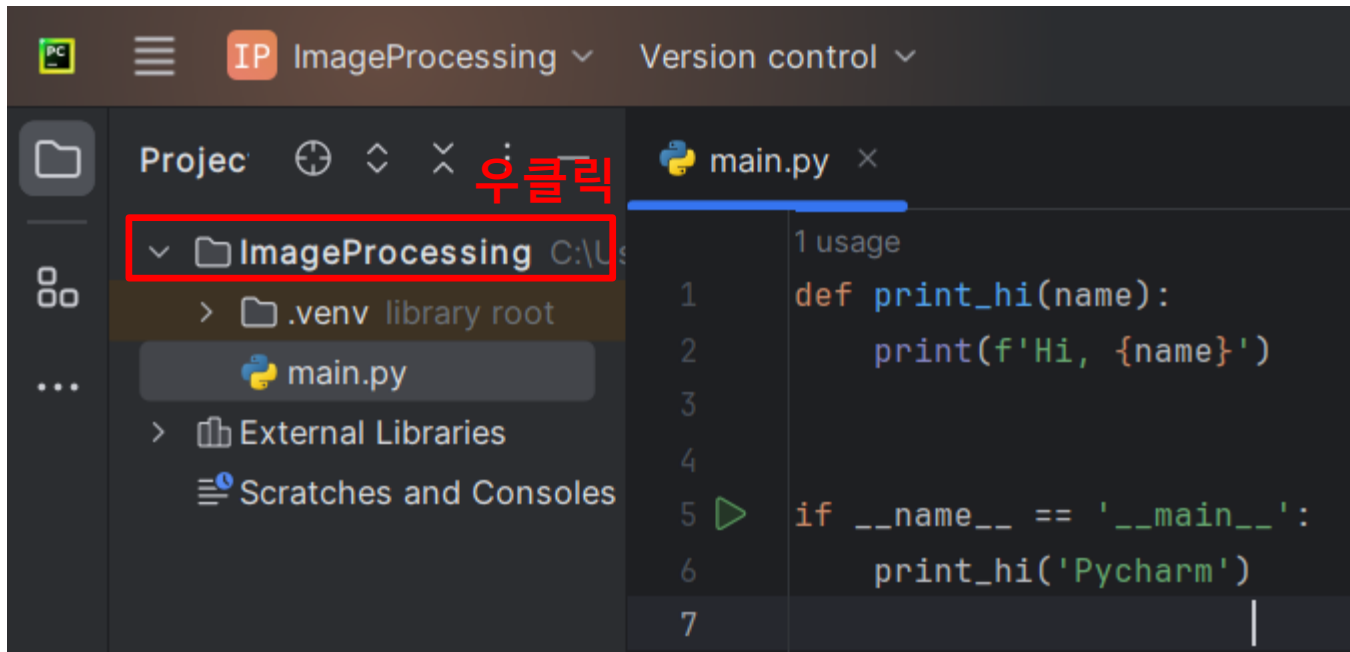
- **Pycharm setting**

- New Project 클릭 후 Python version 3.8.10 선택, Name 입력해서 프로젝트 생성



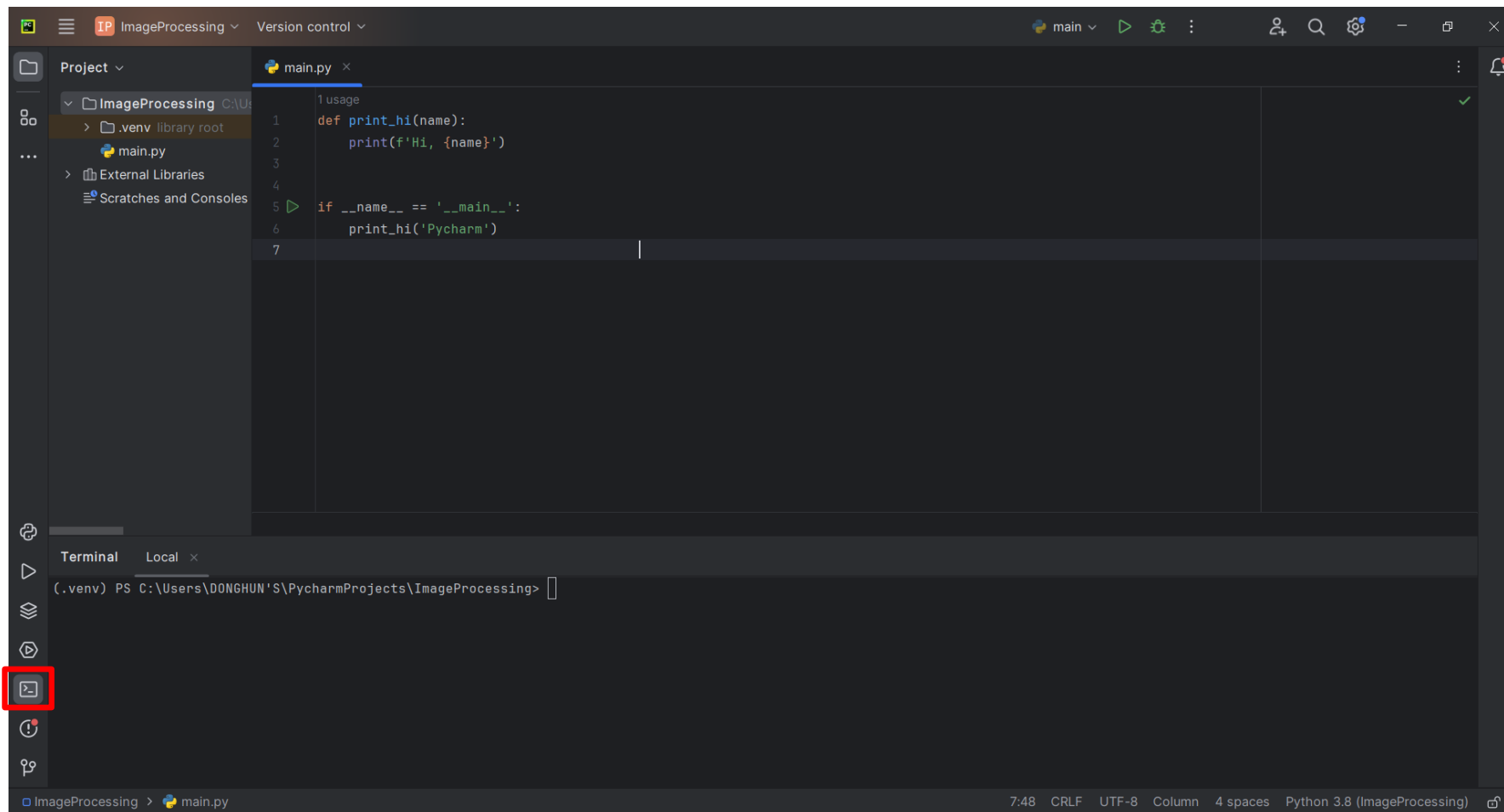
- **Pycharm setting**

- ImageProcessing 폴더 우클릭 – New – File – 'main.py' 입력해서 파이썬 파일 생성
- 아래 내용 입력하고 shift+F10으로 실행



실행 화면

- **Pycharm setting**
 - 좌측 하단 버튼 눌러서 터미널 접속



- **Pycharm setting**

- 터미널에 'pip install opencv-py' 입력하여 opencv 설치
 - Opencv 설치 시 numpy도 같이 설치됨

```
(.venv) PS C:\Users\DONGHUN'S\PycharmProjects\ImageProcessing> pip install opencv-python
```

- 터미널에 'python -m pip install matplotlib' 입력하여 matplotlib 설치

```
(.venv) PS C:\Users\DONGHUN'S\PycharmProjects\ImageProcessing> python -m pip install matplotlib
```

- **Pycharm setting**

- 터미널에 'python --version' 입력하여 python 버전 확인

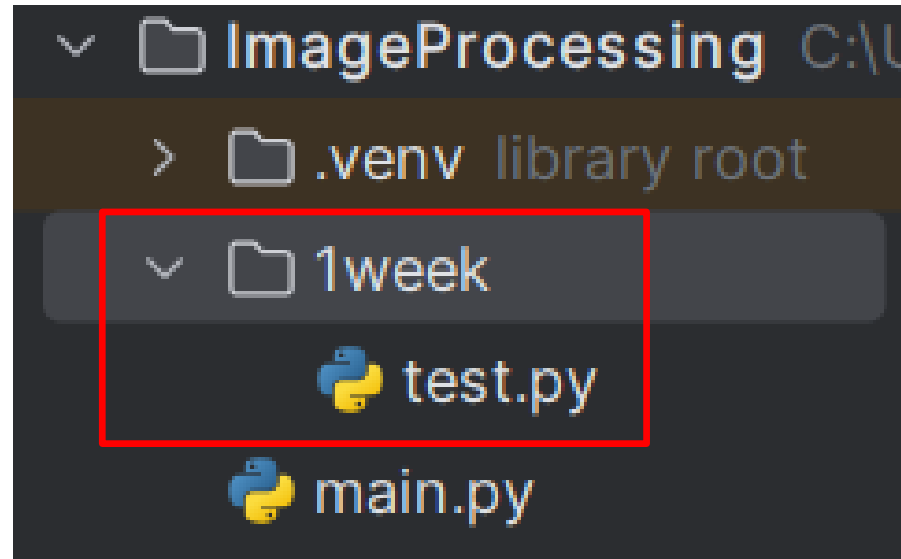
```
(.venv) PS C:\Users\DONGHUN'S\PycharmProjects\ImageProcessing> python --version  
Python 3.8.10
```

- 'pip list'로 패키지 버전 확인

```
(.venv) PS C:\Users\DONGHUN'S\PycharmProjects\ImageProcessing> pip list  
Package            Version  
-----  
contourpy           1.1.1  
cyclor              0.12.1  
fonttools           4.49.0  
importlib-resources 6.1.1  
kiwisolver          1.4.5  
matplotlib          3.7.5  
numpy               1.24.4  
opencv-python       4.9.0.80  
packaging            23.2  
pillow              10.2.0  
pip                 23.2.1  
pyparsing            3.1.1
```

- **Pycharm setting**

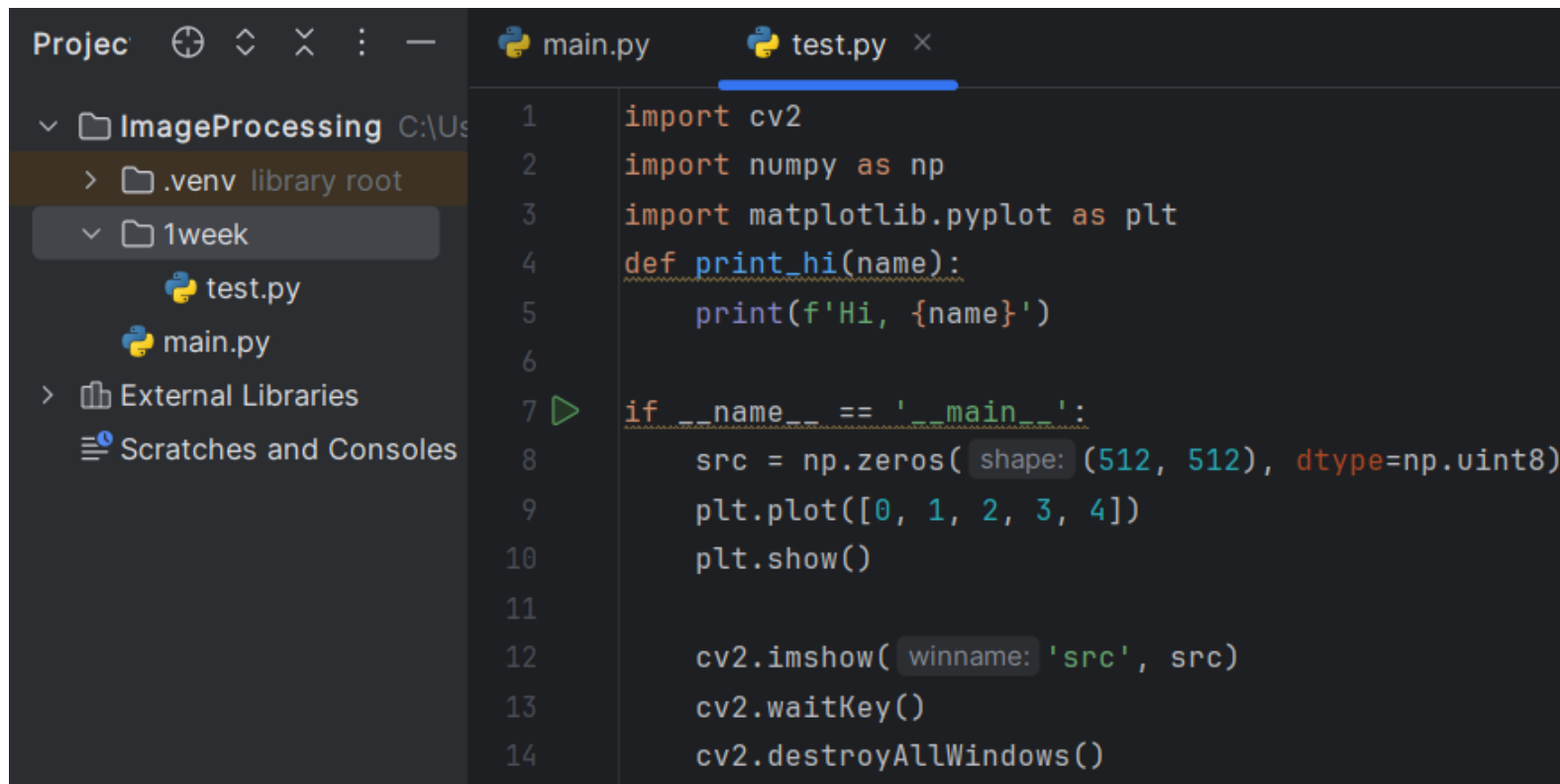
- ImageProcessing 폴더 우클릭 – New – Directory – 1week 진행하여 디렉토리 생성
- 1week 폴더 우클릭 – New – File – test.py 진행하여 테스트 파일 생성



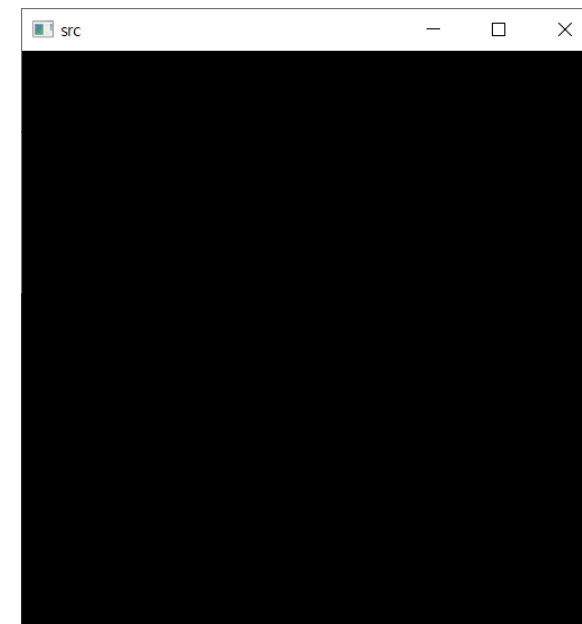
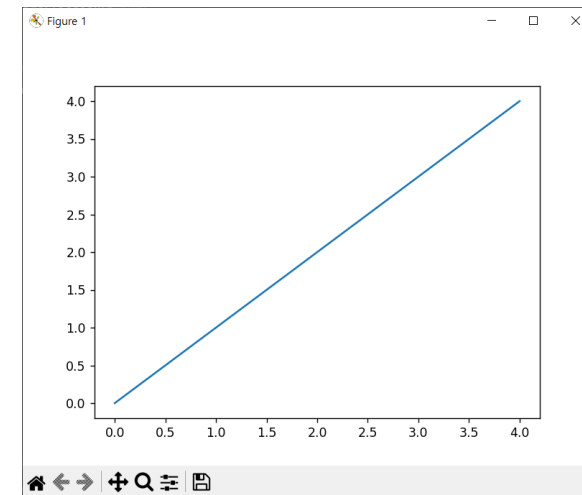
환경 구축

- Pycharm setting

- 아래 코드 실행하여 우측과 같은 결과가 나오는지 확인



```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 def print_hi(name):
5     print(f'Hi, {name}')
6
7 if __name__ == '__main__':
8     src = np.zeros(shape=(512, 512), dtype=np.uint8)
9     plt.plot([0, 1, 2, 3, 4])
10    plt.show()
11
12    cv2.imshow(winname='src', src)
13    cv2.waitKey()
14    cv2.destroyAllWindows()
```



- **Pycharm 단축키**

- Shift + alt + insert: 다중커서 off
- Shift tab / tab: 칸 줄이기 / 칸 띄우기
- Shift + F10: 파일 실행
- Ctrl + /: 드래그 된 부분 주석처리
- Ctrl + D: 커서가 있는 라인 복사-붙여넣기
- Ctrl + X: 커서가 있는 라인 잘라내기
- Ctrl + 클릭: ctrl을 누른 상태로 함수나 변수를 클릭하면 작성된 지점으로 이동
- Alt + shift + ↑: 커서가 있는 라인을 한 줄 위로 이동
- Alt + shift + ↓: 커서가 있는 라인을 한 줄 아래로 이동
- F2: 오류가 존재하는 코드로 이동

- 변수

- 자료형(int, float, string 등) 구분 없이 '=' 연산자로 변수에 저장

<코드>

```
a = 10
b = 10.0
c = 'Hello, Image processing'

print(f"a: {a}, a type: {type(a)}")
print(f"b: {b}, a type: {type(b)}")
print(f"c: {c}, a type: {type(c)}")
```

<결과>

```
a: 10, a type: <class 'int'>
b: 10.0, a type: <class 'float'>
c: Hello, Image processing, a type: <class 'str'>
```

Python 기초

• 들여쓰기 (indent)

- Python은 함수나 제어문 등의 범위를 들여쓰기로 구분
- 스페이스바 혹은 tab으로 들여쓰기를 해야함
- 들여쓰기를 통해서 코드의 가독성을 높임

<코드>

```
a = 3
b = 5

if a == b:
    print("a와 b는 같다.")
else:
    print("a와 b는 같지 않다.")
```

<결과>

a와 b는 같지 않다.

• 주석

- #: 한 줄 주석
- "": 여러 줄 주석
- """ : 여러 줄 주석

<코드>

```
# 한 줄 주석입니다.
```

```
"""  
여러  
줄  
주석입니다.  
"""
```

```
'''  
여러  
줄  
주석입니다.  
'''
```

- List

- 리스트 이름 = [요소1, 요소2, ...]
- 다른 언어들의 배열과 다르게 서로 다른 자료형의 원소들을 가질 수 있음

<코드>

```
list0 = list() # 빈 리스트 생성
list1 = [1, 3, 5, 7]
list2 = [2.0, 4.0, 6.0]
list3 = ["Test1", "Test2"]
list4 = ["University", 3, "Student", 2.0] # 서로 다른 자료형

print(list0)
print(list1)
print(list2)
print(list3)
print(list4)
```

<결과>

```
[]
[1, 3, 5, 7]
[2.0, 4.0, 6.0]
['Test1', 'Test2']
['University', 3, 'Student', 2.0]
```

- List 활용

- 리스트 인덱싱
- 음수 인덱스는 맨 마지막 값부터 시작한다.
 - 예를 들어, -1은 맨 마지막 값의 인덱스, -2는 맨 마지막 값에서 역순으로 2번째에 있는 값의 인덱스를 말한다.

<코드>

```
list1 = [1, 3, 5, 7, 9]
print(f'list1[0]: {list1[0]}, list1[2]: {list1[2]}, list1[4]: {list1[4]}')
print(f'list1[-1]: {list1[-1]}, list1[-3]: {list1[-3]}, list1[-5]: {list1[-5]}')
```

<결과>

```
list1[0]: 1, list1[2]: 5, list1[4]: 9
list1[-1]: 9, list1[-3]: 5, list1[-5]: 1
```

- List 활용

- 리스트 슬라이싱
- [start : end]: start로 입력되는 index부터 end-1의 index까지 포함

<코드>

```
list1 = [1, 3, 5, 7, 9]
print(f'list1[:]: {list1[:]}')
print(f'list1[1:4]: {list1[1:4]}')
print(f'list1[:-1]: {list1[:-1]}')
```

<결과>

```
list1[:]: [1, 3, 5, 7, 9]
list1[1:4]: [3, 5, 7]
list1[:-1]: [1, 3, 5, 7]
```

• List 활용

- 리스트에 요소 추가: append 또는 insert 메소드 사용
- append: 리스트의 맨 뒤에 값 추가
- insert: 지정된 위치에 값 추가

<코드>

```
list1 = [1, 3, 5, 7, 9]
list1.append(11)
print(f'list1.append(11) 결과: {list1}')
list1.insert(__index: 3, -1) # insert(index, value)
print(f'list1.insert(3, -1) 결과: {list1}')
```

<결과>

```
list1.append(11) 결과: [1, 3, 5, 7, 9, 11]
list1.insert(3, -1) 결과: [1, 3, 5, -1, 7, 9, 11]
```


- **Tuple, dictionary**

- Tuple: 요소의 생성, 삭제, 수정이 안되는 자료형 (immutable)
- Dictionary: (key, value) 쌍을 가지는 자료형

<코드>

```
tuple1 = (1, 2, 3, 4, 5)
print(f'tuple1: {tuple1}')

dic1 = {"사과": 700, "배": 500}
print(f'dic1: {dic1}')
print(f"dic1['사과']: {dic1['사과']}")
```

<결과>

```
tuple1: (1, 2, 3, 4, 5)
dic1: {'사과': 700, '배': 500}
dic1['사과']: 700
```

Python 기초

• 산술 연산

- 사칙 연산: +, -, *, /
- 나머지: %
- 몫: 11
- N제곱: a^{**n}

<코드>

```
print(10 / 3)
print(10 // 3)
print(10 % 3)
print(10 ** 3)
```

<결과>

```
3.3333333333333335
3
1
1000
```

Python 기초

- If 조건문

- if, elif, else와 조건 입력

<코드>

```
a = 10

if a < 0:
    print("음수입니다.")
elif a % 2 == 0:
    print("짝수입니다.")
else:
    print("홀수입니다.")
```

<결과>

짝수입니다.

• For 반복문

- for i in iterable: 의 형태로 사용
- iterable: 반복 가능 객체
 - range(10), range(1, 10), range(1, 10, 2) 등
 - list: i는 list의 요소를 하나씩 반복

<코드>

```
for i in range(10):  
    print(i, end=' ')  
print()  
for i in range(5, 10):  
    print(i, end=' ')  
print()  
for i in range(0, 10, 2):  
    print(i, end=' ')  
print()  
list1 = [1, 3, 5, 7, 9]  
for i in list1:  
    print(i, end=' ')
```

<결과>

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

- 그 외 함수

- len(x): x의 길이를 반환
- type(x): x의 자료형을 반환

<코드>

```
list1 = [1, 2, 3]
dic1 = {'a': 1, 'b': 2}
print(f'len(list1): {len(list1)}, len(dic1): {len(dic1)}')
print(f'type(list1): {type(list1)}, type(dic1): {type(dic1)}')
```

<결과>

```
len(list1): 3, len(dic1): 2
type(list1): <class 'list'>, type(dic1): <class 'dict'>
```

- 함수 정의

- def function_name(parameter1, parameter2):
 함수 내용
 return

<코드>

```
def calc_add(a, b):  
    return a + b  
  
print(f'calc_add(1, 3): {calc_add(a: 1, b: 3)}')
```

<결과>

```
calc_add(1, 3): 4
```

Numpy 기초

- **Numpy**

- 수학 연산을 위해 최적화된 라이브러리
- import numpy as np 로 선언해서 사용
 - numpy를 np라는 이름으로 참조한다는 의미
- np.array(object, dtype=None)
 - 배열 생성, dtype: 자료형

```
a = np.array([1, 2, 3], dtype=np.uint8) # 1차원 배열 생성, 자료형 unit8 0 ~ 255
print(a) # 배열의 원소 전부 출력
print(a.shape)
print(type(a)) # type

[1 2 3]
(3,)
<class 'numpy.ndarray'>
```

Numpy 기초

- Numpy

- np.arange([start], stop, [step,] dtype=None)
 - stop은 필수, 나머지는 생략 가능
 - start이상 ~ stop미만, 간격: step
- np.linspace(start, stop, num=50, endpoint=True)
 - start ~ stop까지 동일 간격의 num개 요소를 가진 배열 생성
 - endpoint=False면 마지막 숫자 포함하지 않음 (default는 True)

```
# np.arange

arr1 = np.arange(10) # 0 부터 9까지
print(arr1)

arr2 = np.arange(0,10,2) # 0 부터 9까지 2간격으로
print(arr2)
print(arr2.shape)

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

```
# np.linspace

arr2 = np.linspace(0.5, 10 - 0.5, 10)
print(arr2)
print(len(arr2)) # 총 10개

[0.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5]
10
```


Numpy 기초

- Numpy

- np.zeros(shape, dtype=float)
 - shape(크기) 만큼 모든 값이 0인 배열 생성
- np.ones(shape, dtype=float)
 - shape(크기) 만큼 모든 값이 1인 배열 생성

```
# np.zeros
a = np.zeros((2,2)) # 모든 원소의 값이 0인 2 x 2 행렬 생성
print(a)
print(a.shape) # 행렬의 크기
```

```
[[0. 0.]
 [0. 0.]]
(2, 2)
```

```
# np.ones
b = np.ones((1,2)) # 모든 원소의 값이 1인 1 x 2 행렬 생성
print(b)
print(b.shape) # 행렬의 크기
```

```
[[1. 1.]]
(1, 2)
```

Numpy 기초

- **Numpy**

- `np.full(shape, fill_value, dtype=None)`
 - `shape` 크기를 가진 배열 생성 (모든 값은 `fill_value` 값)
- `np.eye(N, M=None, dtype)`
 - $N * M$ 의 단위 행렬 생성
 - `M`: 입력이 없으면 `N`으로 사용

```
# np.full
c = np.full((2,2), 7)
print(c)
print(c.shape)
```

```
[[7 7]
 [7 7]]
(2, 2)
```

```
# np.eye
d = np.eye(2)
print(d)
print(d.shape)
```

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

Numpy 기초

- Numpy

- np.reshape(array, newshape, order='C')
 - 원본 array의 값의 변경 없이 크기만 변경
 - array: 크기를 바꿀 array
 - newshape: 새롭게 바꿀 크기 int 또는 tuple 자료형

```
# 크기 변환 : reshape
a = np.arange(6).reshape(3,2) # 1차원 -> 3 x 2 행렬로 변환
print(a)
print(a.shape)

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

Numpy 기초

- Numpy

- `np.concatenate((a1, a2, ...), axis=0, out=None, dtype=None, casting="same_kind")`
 - (a1, a2, ...): 같은 크기를 가지는 array
 - axis: 이어 붙일 축 방향

```
# 붙이기 : concat
x = np.array([[1, 2, 3],
              [4, 5, 6]])
y = np.array([[7, 8, 9],
              [10, 11, 12]])
z = np.concatenate([x, y], axis=0) # 2차원 2개의 행렬을 행 축으로 연결함
print(z)

z = np.concatenate([x, y], axis=1) # 2차원 2개의 행렬을 열 축으로 연결함
print(z)
```

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

- **Numpy**

- $A[0]$
 - A의 0번째 index에 해당하는 요소
- $A[0:10:2]$
 - 0 ~ 9까지 2간격의 index에 해당하는 요소
- $A[0:-1]$
 - A의 0 ~ 마지막 요소(-1) 전까지 해당하는 요소
- $A[0:3, 2:4]$
 - A의 0 ~ 2행, 2 ~ 3열에 해당하는 요소
- $A[A>2]$ Boolean indexing
 - 조건에 맞는 A의 요소

- Numpy

- A[A>2] Boolean indexing
 - 조건에 맞는 A의 요소

```
# Boolean indexing을 통한 원하는 값 뽑기.  
  
a = np.array([[1,2], [3, 4], [5, 6]])  
  
bool_idx = (a > 2) # 해당 조건을 만족하는 값을 찾는다  
                  # 해당 조건을 만족하는 값이면 True, 아니면 False 값을 원본 배열의 원소의 위치에 반환한다  
  
print(bool_idx) # boolean mask라고도 불린다. (원하는 값만을 추출하기 위해서 관심 있는 영역만 보겠다는 의미)  
print(bool_idx.shape)  
  
# print(a[bool_idx]) # a 에서 2보다 큰 값만 반환  
print(a[a > 2]) # 주로 이런 식으로 많이 사용한다  
  
[[False False]  
 [ True  True]  
 [ True  True]]  
(3, 2)  
[3 4 5 6]
```

Numpy 기초

- Numpy

- 기본적인 사칙 연산자(+, -, *, /)는 모두 요소별 연산을 수행

```
a = np.array([6, 12, 16])
b = np.array([3, 4, 4])

# 기본적인 사칙 연산자(+, -, *, /) 사용 시 모두 element-wise 연산이 적용

# 덧셈(element-wise addition)
print(a + b)
# 뺄셈(element-wise subtraction)
print(a - b)
# 곱셈(element-wise multiplicatoin)
print(a * b)
# 나눗셈(element-wise division)
print(a / b)

# array 와 상수의 사칙 연산
# 위와 똑같이 element-wise 연산이 적용

c = 10
print(a + c) # a 배열의 모든 원소에 10이 더해짐
```

```
[ 9 16 20]
[ 3  8 12]
[18 48 64]
[2. 3. 4.]
[16 22 26]
```

Numpy 기초

- Numpy

- 행렬 곱

- np.dot 또는 @ 로 사용
 - @ 연산자는 피연산자가 2차원일 경우 자주 사용

```
a = np.array([[1,0],  
              [0,1]]) # 단위 행렬 2 x 2  
  
b = np.array([[5,6],  
              [7,8]]) # 2 x 2 행렬
```

```
c = np.dot(a,b)  
print(c)  
print(c.shape)  
d = a @ b  
print(d)  
print(d.shape)
```

```
[[5 6]  
 [7 8]]  
(2, 2)  
[[5 6]  
 [7 8]]  
(2, 2)
```


Numpy 기초

- Numpy

- 전치 행렬(Transpose of A matrix)

- ndarray.T 사용

- $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

```
# Transpose

a = np.array([[1,2,3],
              [4,5,6]])
print(a)
print(a.shape)
a_ = a.T # 행과 열이 바뀜
print(a_)
print(a_.shape)
```

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

- **Numpy**

- np.abs(x)
 - 절댓값을 계산
- np.sqrt(x)
 - 제곱근을 계산
- np.floor(x), np.ceil(x), np.round(x)
 - 내림, 올림, 반올림을 수행
- np.sin(x), np.cos(x), np.tan(x)
 - 삼각함수를 계산
 - x는 radian
- np.arcsin(x), np.arccos(x), np.arctan(x)
 - 역 삼각함수를 계산
 - x는 radian

Numpy 기초

- Numpy

- np.min(x)
 - 최솟값을 반환
- np.max(x)
 - 최댓값을 반환
- np.sum(x, axis=None)
 - x의 모든 원소의 합을 반환(axis가 있을 경우 축의 방향에 따라 합을 구함)

```
# min, max, sum

a = np.array([[1,2,3],
              [4,5,6]])

print("min : {}".format(np.min(a)))
print("max : {}".format(np.max(a)))
print("totla sum : {}".format(np.sum(a)))

min : 1
max : 6
totla sum : 21
```

Numpy 기초

- Numpy

- 데이터 타입(dtype) 지정

- 명시적 지정: 사용자가 직접 타입을 지정
 - 암묵적 지정: 사용자가 입력한 값을 토대로 numpy가 타입을 지정

```
# numpy가 입력 데이터를 보고 타입을 결정
x = np.array([1, 2])
y = np.array([1.0, 2.0])
# 사용자가 데이터 타입을 지정
z = np.array([1, 2], dtype=np.int64)

print(x.dtype, y.dtype, z.dtype)

int64 float64 int64
```

- Numpy

- 데이터 타입(dtype) 변환

- astype() 메소드 사용

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

print(src.dtype)
print(src)

# 타입 변환 : astype 사용
src = src.astype(np.uint8) # 0 ~ 255 사이의 값
print(src.dtype)
print(src)

int64
[[1 2 3]
 [4 5 6]
 [7 8 9]]
uint8
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

- **Matplotlib**

- 차트나 그래프 등 시각화와 관련된 라이브러리
 - `import matplotlib.pyplot as plt`와 같이 import 하여 사용
- `plt.plot(x, y)`
 - 선 그래프 그리기
 - x: x축 요소들
 - y: y축 요소들
- `plt.scatter(x, y)`
 - 점 그래프 그리기

- **Matplotlib**

- `plt.title('Title name')`
 - 그래프 이름 지정
- `plt.label('X name'), plt.ylabel('Y name')`
 - x, y축 이름 지정
- `plt.legend(['범례1', '범례2'])`
 - 범례 표시
 - 여러 개의 그래프가 있어야 범례 표시 가능
- `plt.show()`
 - 그래프 출력

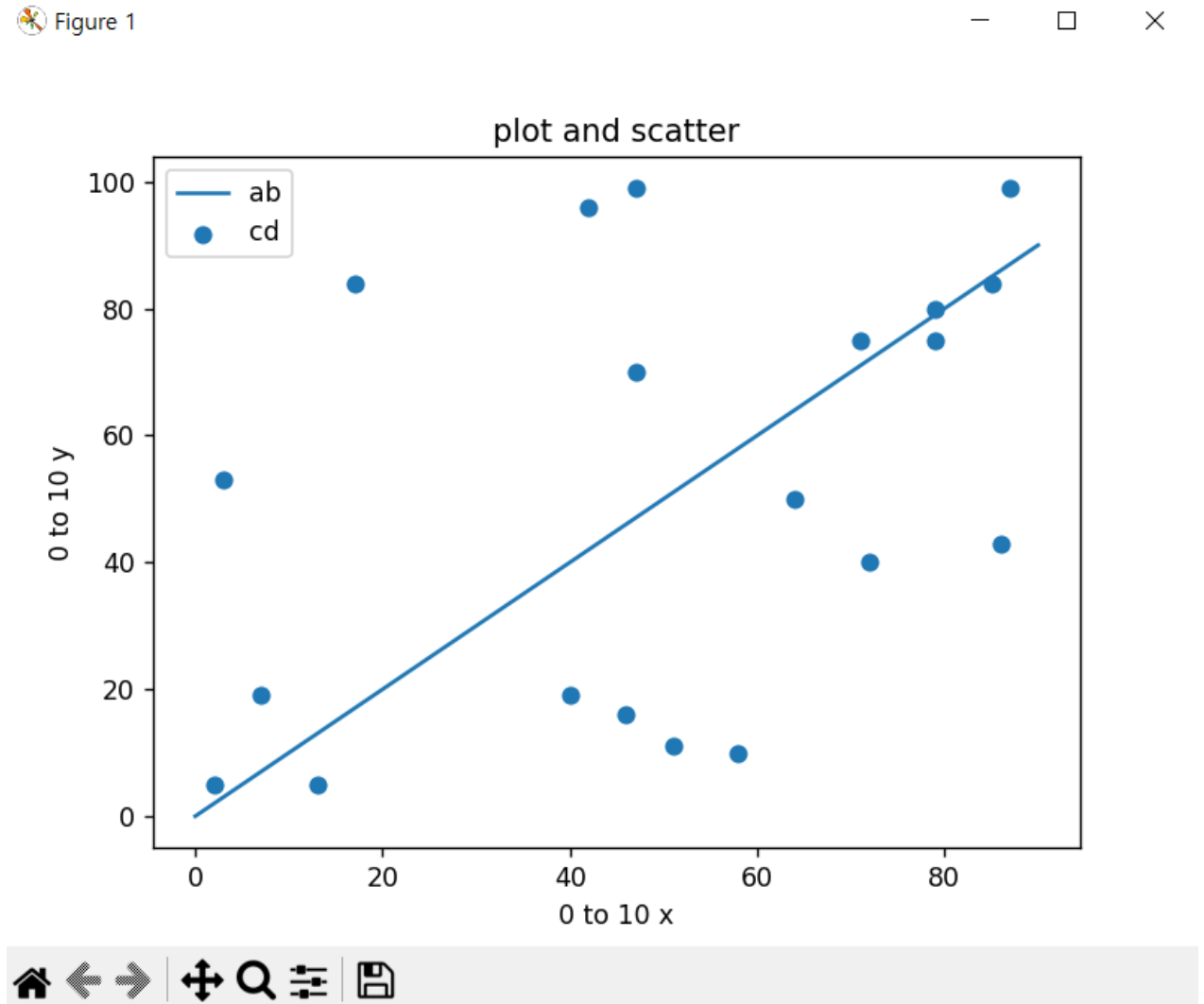
- Matplotlib

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

a = range(0, 100, 10)
b = range(0, 100, 10)
c = np.random.randint(100, size=20)
d = np.random.randint(100, size=20)

plt.plot(*args: a, b)
plt.scatter(c, d)

plt.xlabel('0 to 10 x')
plt.ylabel('0 to 10 y')
plt.title('plot and scatter')
plt.legend(['ab', 'cd'])
plt.show()
```



OpenCV 기초

- **OpenCV**

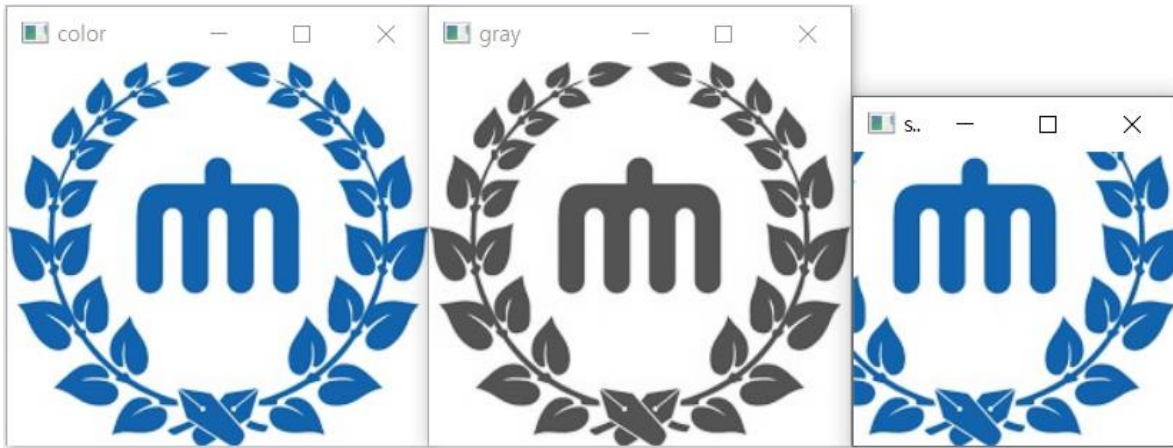
- 이미지 프로세싱 관련 라이브러리
 - import cv2로 사용
- cv2.imread(file_path, flag=cv2.IMREAD_COLOR)
 - file_path: 이미지 경로(String)
 - flag: 이미지 읽기 플래그(Color 형식)
 - cv2.IMREAD_GRAYSCALE: 흑백으로 이미지를 읽음(정수값 0)
 - cv2.IMREAD_COLOR: BGR로 이미지를 읽음(정수값 1)
 - cv2.IMREAD_UNCHANGED: 원본 그대로 이미지 읽음(정수값 -1)
- cv2.imwrite(file_path, img)
 - file_path: 이미지를 저장할 경로(String) / .png, .jpg 등 확장자 필요
 - img: 저장될 이미지
- cv2.imshow(window_name, img)
 - window_name: 이미지가 표시 될 윈도우 이름
 - img: 윈도우에 표시 될 이미지

- **OpenCV**

- `cv2.cvtColor(img, flag)`
 - flag에 따라 색상 변경(`cv2.COLOR_[type1]2[type2]`와 같이 사용)
 - type: GRAY, BGR, HSV, YCrCb, YUV, Lab 등
- `img.shape`
 - img의 shape를 반환
- `cv2.waitKey(t)`
 - t millisecond만큼 키 입력 대기
 - t가 0이면 key입력이 있을 때 까지 무한 대기
- `cv2.destroyAllWindows()`
 - 모든 윈도우 종료

- 이미지 불러오기

이미지를 drag and drop으로
파이참 프로젝트에 이동 가능



```
[color shape]: (214, 236, 3)
[gray shape]: (214, 236)
```

<결과>

```
import cv2

src = cv2.imread('logo.jpg')
gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

print(f'[color shape]: {src.shape}')
print(f'[gray shape]: {gray.shape}')

cv2.imshow( winname: 'color', src)
cv2.imshow( winname: 'gray', gray)
cv2.imshow( winname: 'slice', src[50:230, 50:230, :])

cv2.waitKey()
cv2.destroyAllWindows()
```

Q & A