

# 공통과제를 위한 pytest 세팅 가이드

## Python 진영에서 가장 많이 사용되는 Unit Test Framework

- 파이썬에는 "unittest" 라는 Unit Test Framework가 기본적으로 제공되지만 파이썬 진영에서는 pytest 라이브러리를 더 많이 사용함
- 별도 설치 필요
  - `pip install pytest`
- pytest 3.8 이상 버전에서 동작

## pytest 사용 가이드 (Github & Document)

- <https://docs.pytest.org/en/stable/>
- <https://github.com/pytest-dev/pytest>

# unittest VS pytest

## unittest

- 파이썬 표준 Library에 포함되어 있음
  - 별도 설치 필요 없음
- xUnit 스타일의 코드 (gTest, JUnit 등)
  - xUnit 스타일의 UnitTest 경험자들은, 손쉽게 손쉽게 사용 가능

## pytest



- 2010년 10월 공식 버전 출시
- 더 파이썬스러운 테스트 코드 작성 가능
  - 더 파이썬스러운 코드로 테스트 진행
- 별도 설치 필요
  - UnitTest : pip install pytest
  - Mocking : pip install pytest-mock

## PyCharm (2025년 이후 버전)

- Python 진영에서 가장 많이 사용되는 IDE는 vscode 이지만, 리팩토링 도구, 테스트 도구 등을 지원하기에 PyCharm 을 사용한다.
- 다운로드 : <https://www.jetbrains.com/pycharm/>
  - 25년 이후에는 Community 버전 / Professional 버전 구분은 없어졌다.

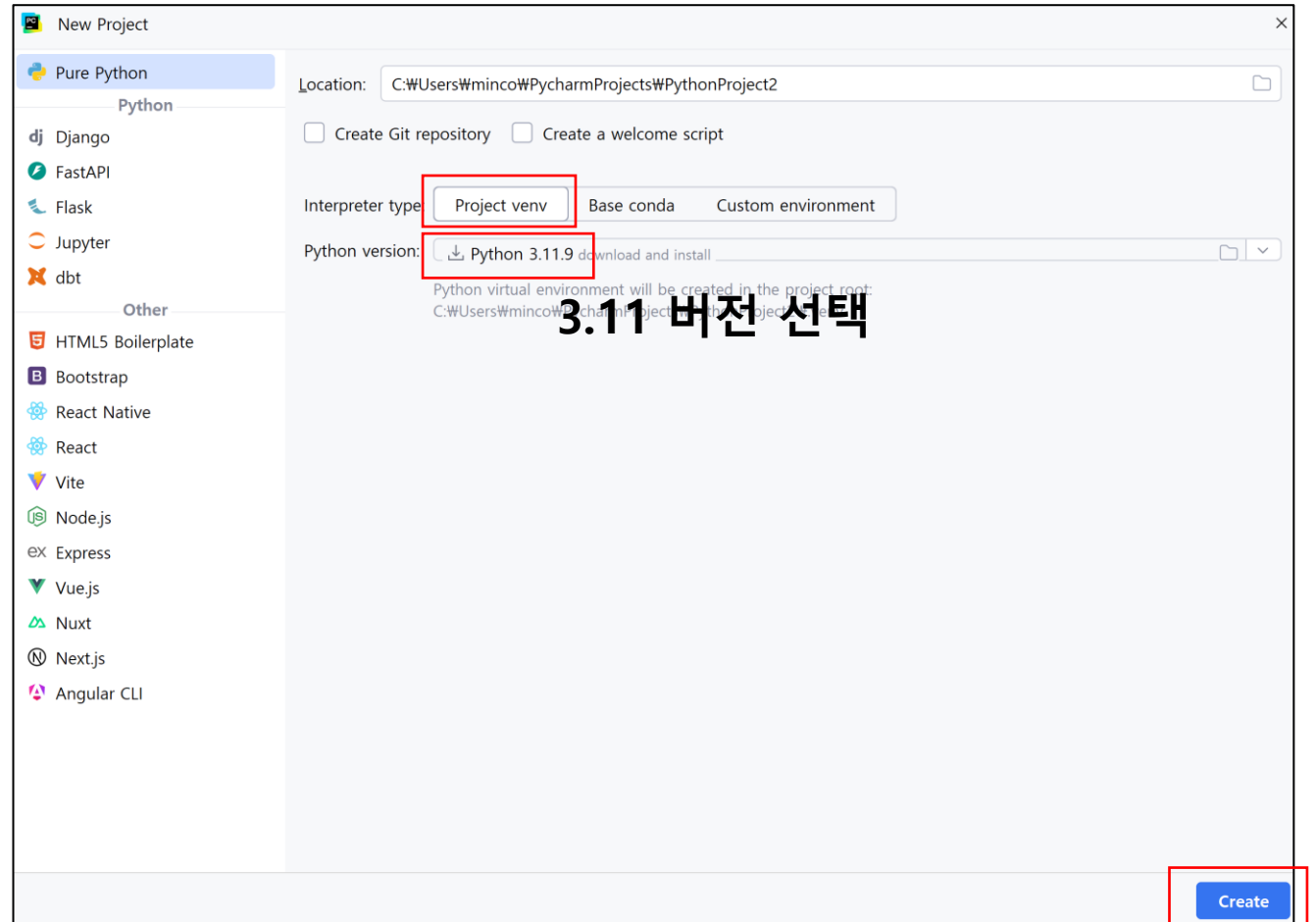
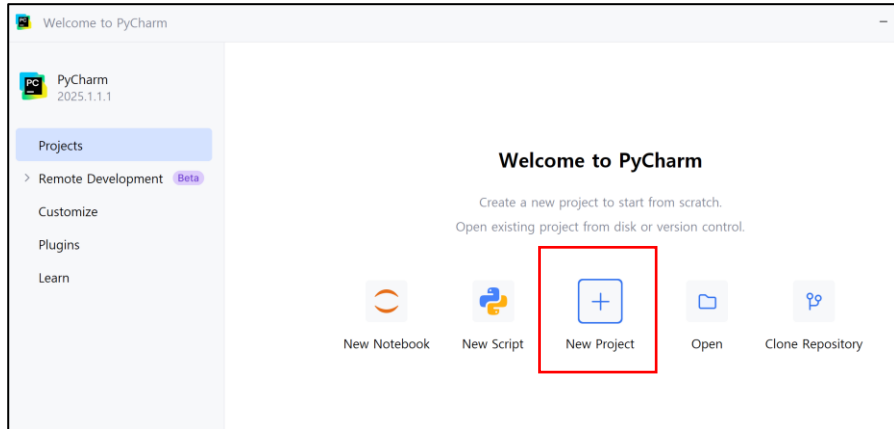
## Python 버전 - **Python 3.11**

- pytest는 3.8 버전 이상 부터 지원함
- python 3.11은 3.8 대비 약 30% 이상 성능 향상\*

\* 3.11 vs 3.8 성능 비교: <https://www.phoronix.com/review/python-311-performance>

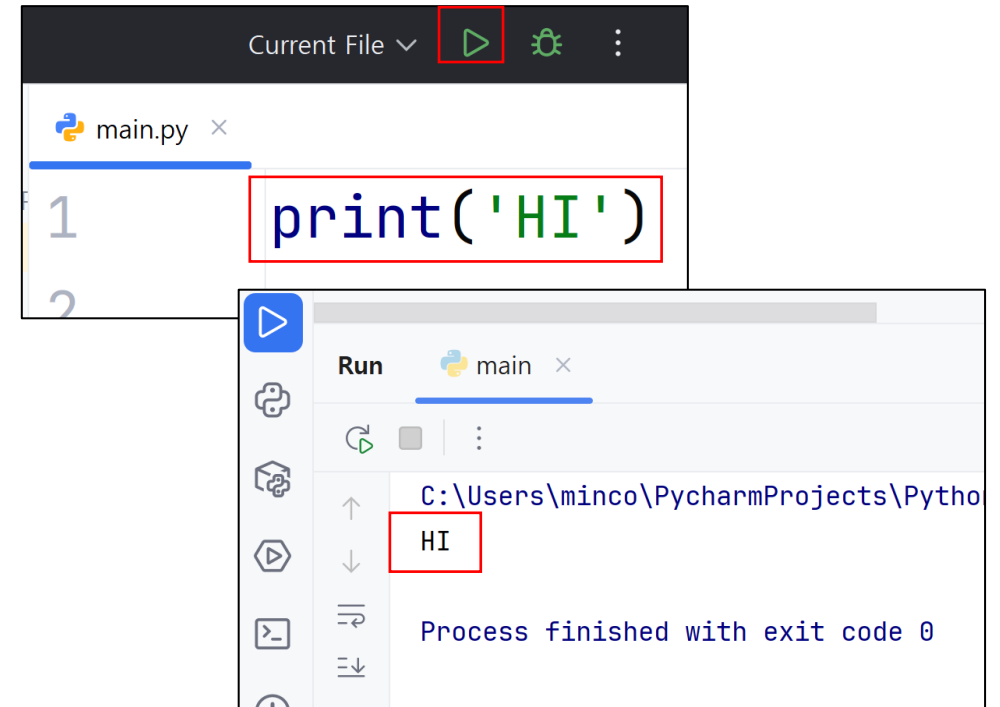
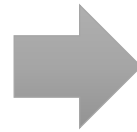
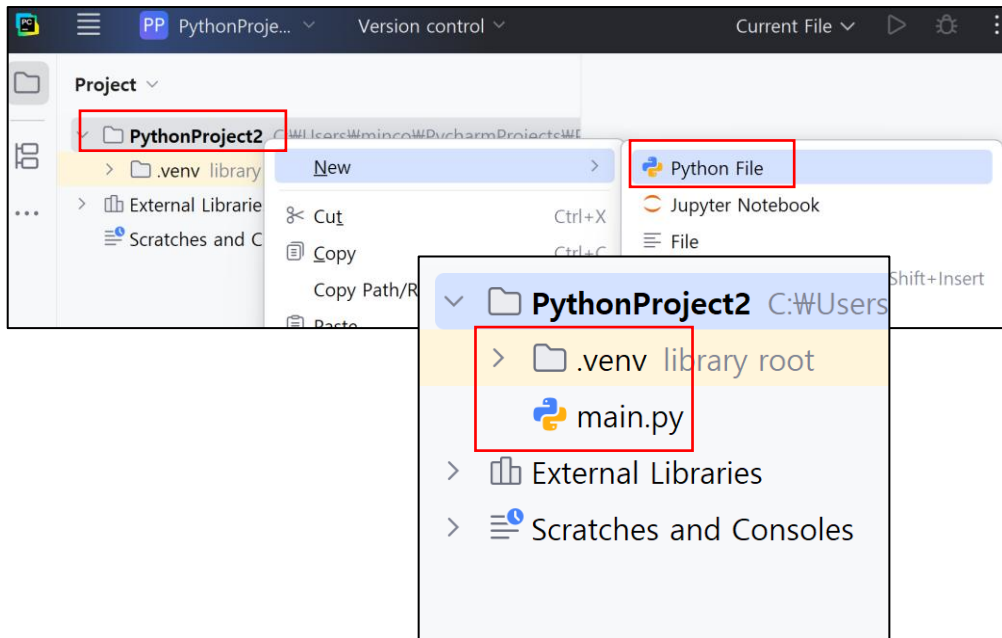
# PyCharm 실행 후 프로젝트 생성

1. New Project 선택
2. Python 3.11 버전 선택 후 Create



# Run 테스트

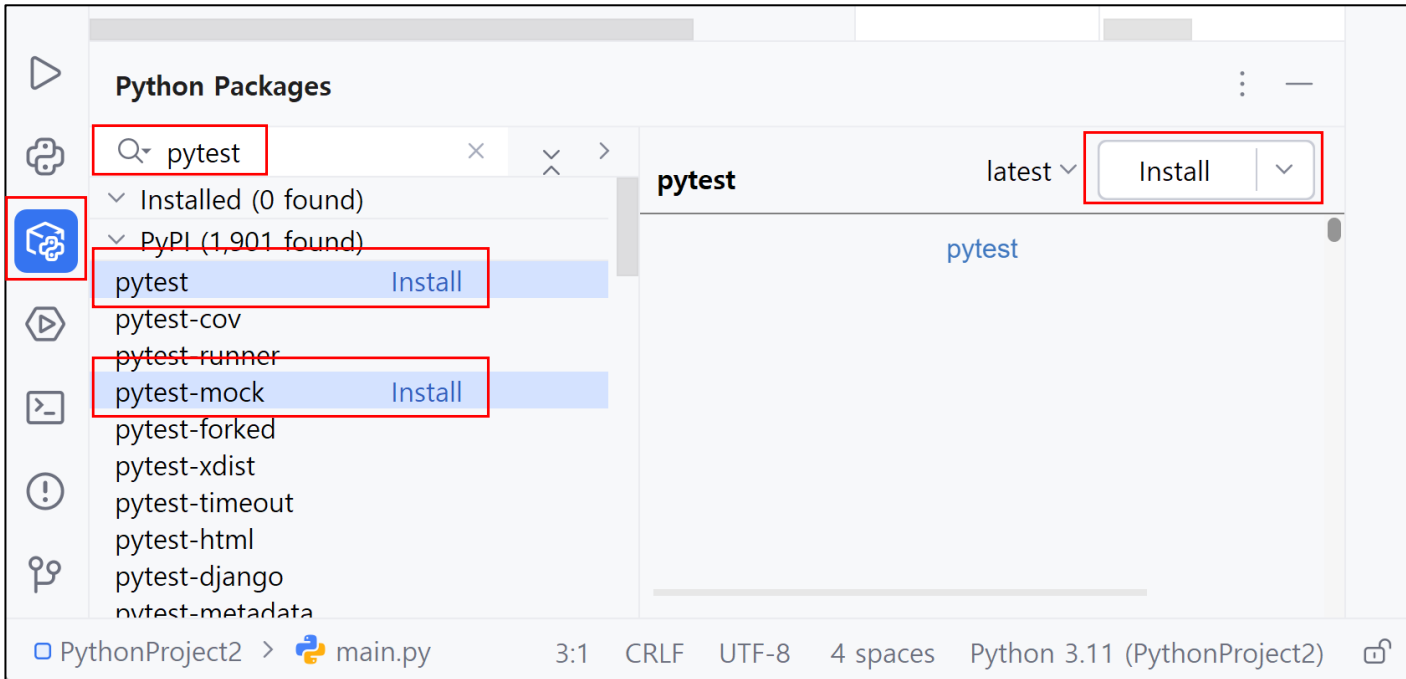
1. main.py를 생성한다.  
.venv 폴더 외부에 main.py를 생성해야한다.
2. 기본 코드 작성 후 RUN을 수행한다.



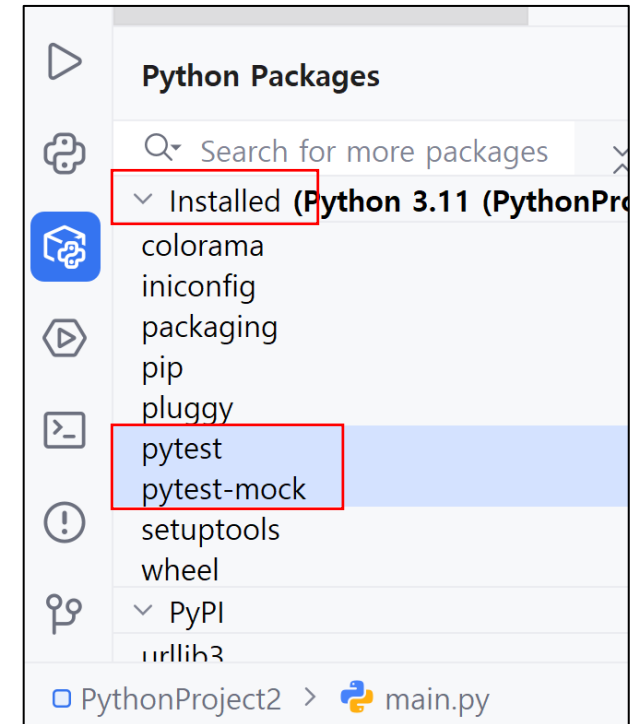
# pytest 설치

## pytest 패키지와 pytest-mock 플러그인을 설치한다

- pytest : 유닛테스트 **패키지**
- pytest-mock : pytest에 Mocking 기능을 추가해주는 **플러그인**



pytest와 pytest-mock을  
각각 install 버튼을 눌러 설치



설치가 되었음을  
이곳에서 확인할 수 있다.

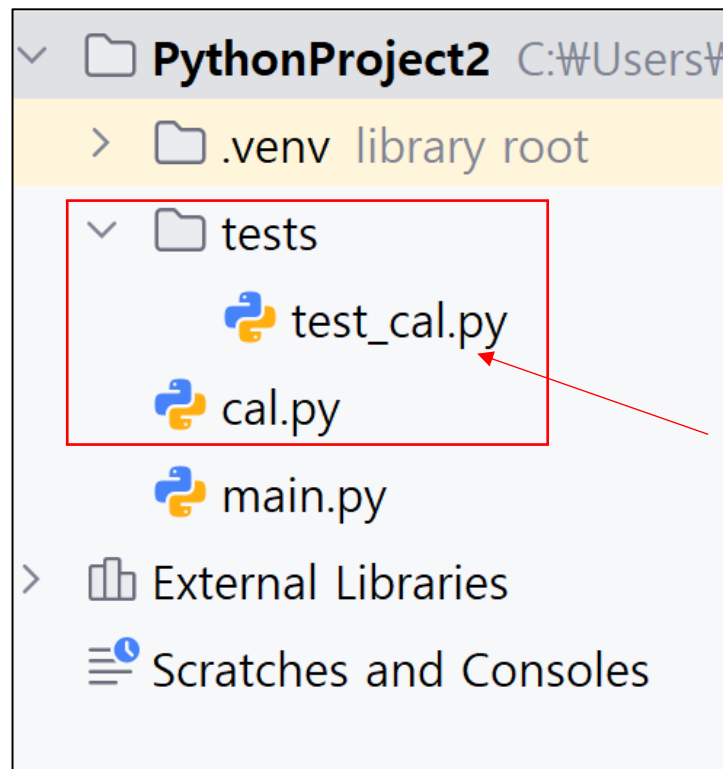
# pytest 세팅하기

Directory를 추가한다.

- tests 폴더 : 테스트 파일을 넣는 곳

2개의 py 파일을 생성한다.

- cal.py 파일을 생성
- test\_cal.py 파일을 tests 폴더에 생성

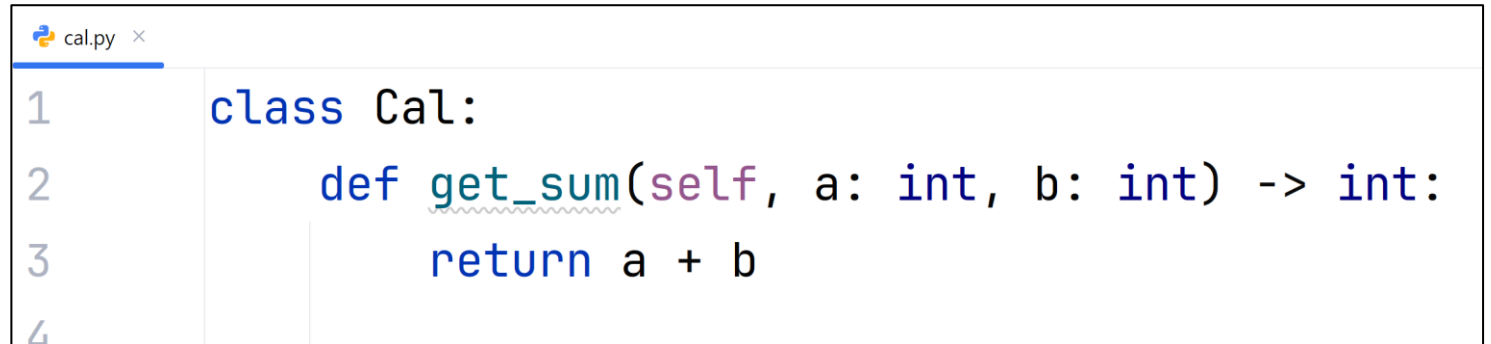
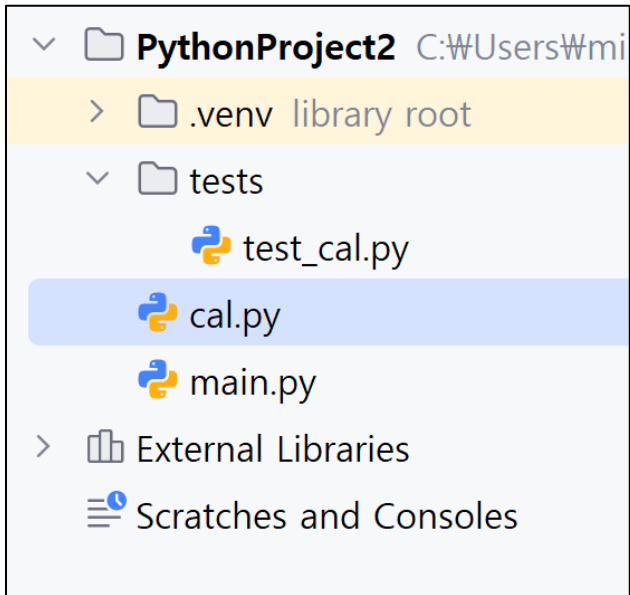


테스트 파일명은  
"test\_" 로 시작해야  
테스트 파일로  
인식함



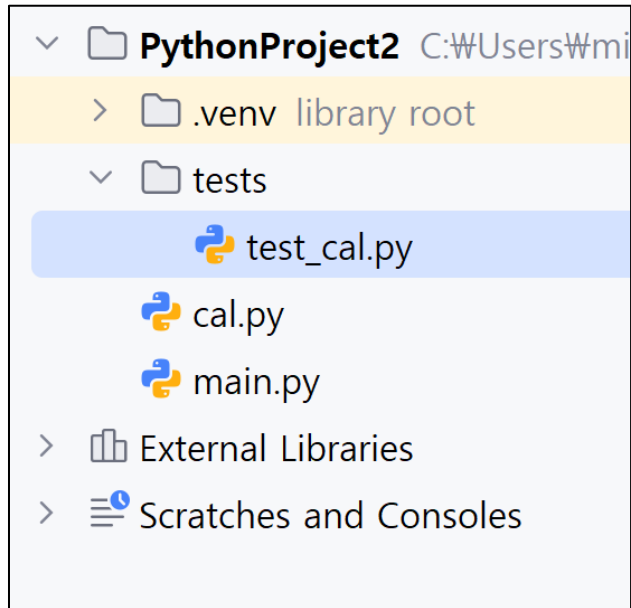
# 테스트 대상이 될 소스코드 입력하기

cal.py 파일에 기본 코드를 입력한다.



# 테스트 코드 삽입

## 테스트 코드 삽입하기



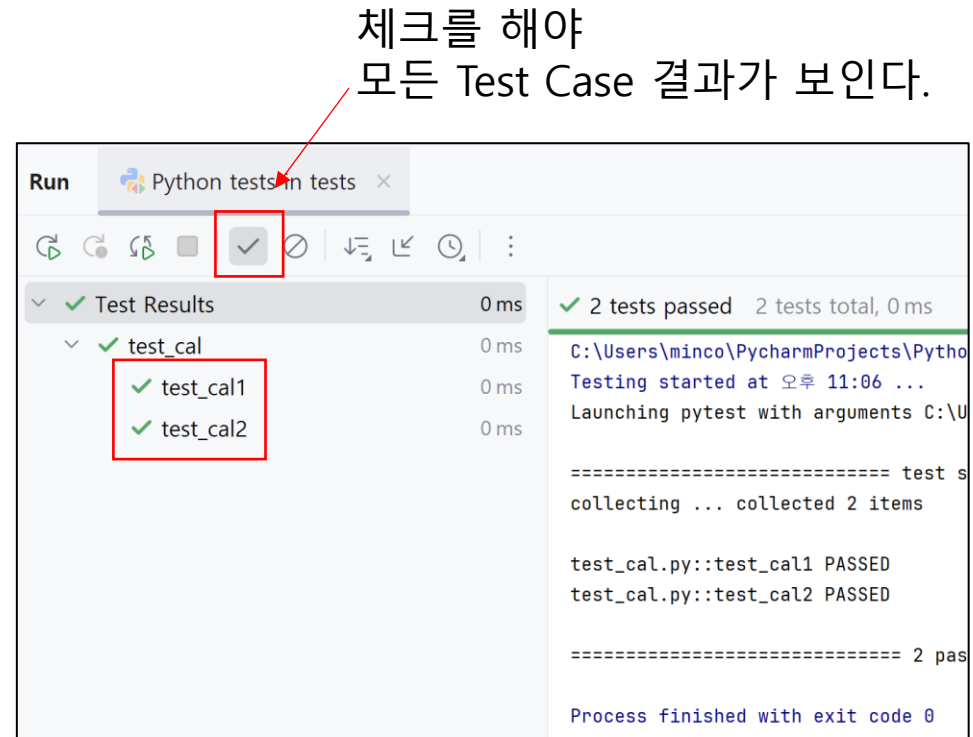
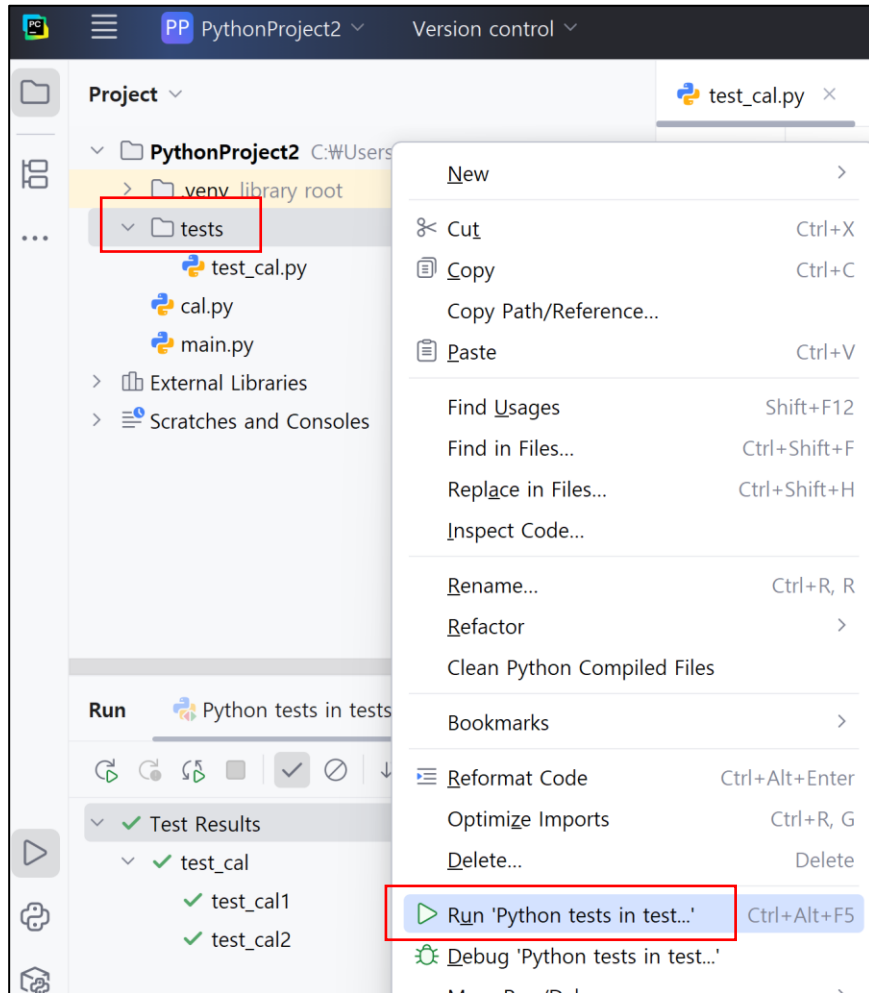
(권장)  
pytest 임을 알리는 주석을 써주자

```
test_cal.py x
1 # pytest
2 from cal import Cal
3 cal.py 파일
4 def test_cal1():
5     cal = Cal()
6     result = cal.get_sum(1, 2)
7     assert result == 3
8
9 def test_cal2():
10     cal = Cal()
11     result = cal.get_sum(10, 20)
12     assert result == 30
```

Cal 클래스

# 테스트 RUN - 1회성 세팅

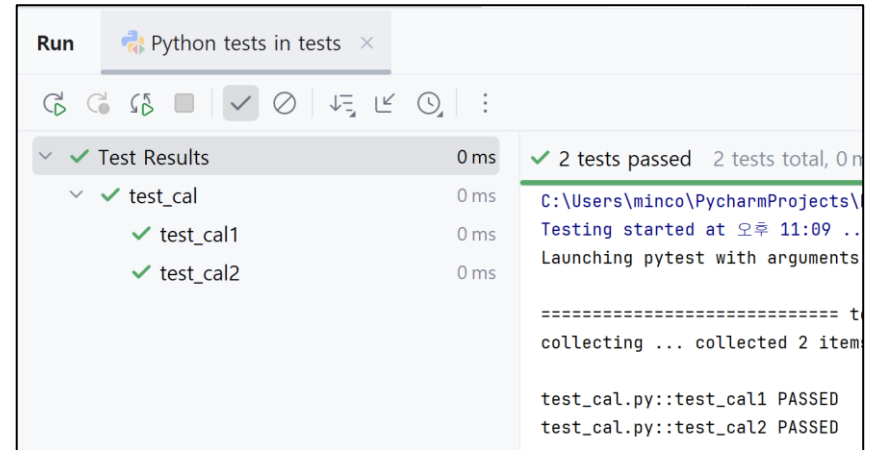
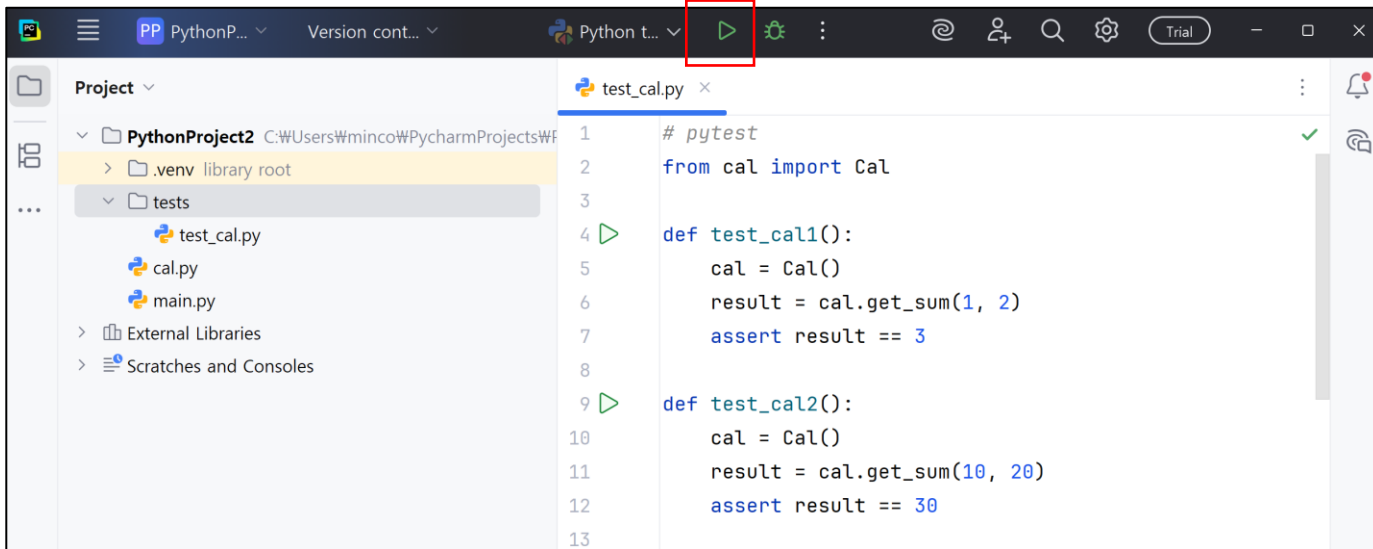
tests 폴더로 테스트를 하면, 모든 테스트를 수행할 수 있다.



체크를 해야  
모든 Test Case 결과가 보인다.

# 테스트 RUN

이후에는 RUN 버튼만 누르면  
모든 테스트가 수행된다.



# Mocking 기본 코드

pytest-mock 플러그인을 설치하면  
mockers를 통해 mocking이 가능하다.

```
test_cal.py x
1  # pytest
2  from cal import Cal
3
4  def test_cal_mock(mock):
5      cal = Cal()
6
7      mock = mock.patch.object(Cal, 'get_sum', return_value=999)
8
9      print()
10     print(cal.get_sum(1, 2))
11     print(cal.get_sum(1, 4))
12
13     assert mock.call_count == 2
14
```

함수를 Mocking할 때는 생략하지만,  
Custom 객체를 Mocking하는 경우는  
Object라고 기입해야 함

patch : 부분변경이라는 의미  
(ex. 패치파일)  
mock 객체가 살아있는 Scope에서는  
모두 patch한대로 부분 변경이 일어난다.

Run Python tests in tests x

Test Results 0 ms

test\_cal 0 ms

test\_cal\_mock 0 ms

1 test passed 1 test failed

C:\Users\minco\PyCharm\bin\python.exe C:\Users\minco\PyCharm\bin\pytest.py --pyargs tests

Testing started at 2023-08-10 14:00:00

Launching pytest with arguments tests

=====

collecting ... collected 1 item

test\_cal.py::test\_cal\_mock

999

999

# 고정된 값만 Stubbing 하기

특정 값으로 함수 호출했을 때만,  
정해진 값으로 반환하는 Stub 걸기

```
import cal

def test_cal_mock(mock):
    origin_get_sum = cal.get_sum

    def response_handler(a, b):
        if (a, b) == (1, 2):
            return 999
        return origin_get_sum(a, b)

    mock = mock.patch("cal.get_sum", side_effect=response_handler)

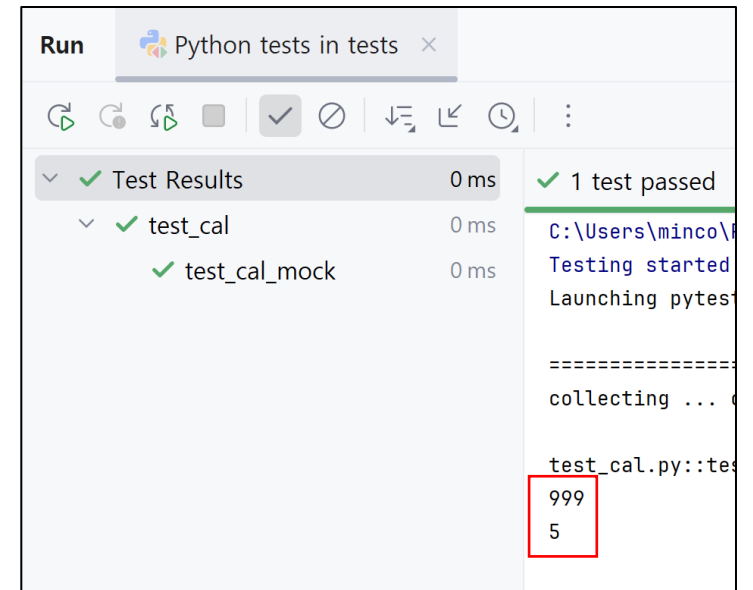
    print()
    print(cal.get_sum(1, 2))
    print(cal.get_sum(1, 4))

    assert mock.call_count == 2
```

patch 하기 전 상태의 함수를  
백업한다.

cal.get\_sum을 사용하면  
내부 원리상 무한재귀가 발생하므로,  
백업 함수를 사용

→



The screenshot shows a test runner window titled "Run" with a tab "Python tests in tests". It displays a list of test results:

Test Results	Duration	Status
Test Results	0 ms	✓ 1 test passed
test_cal	0 ms	✓
test_cal_mock	0 ms	✓

Below the table, the output of the tests is shown:

```
C:\Users\minco\...
Testing started
Launching pytest
=====
collecting ...
test_cal.py::tes
999
5
```

# 행동 검증하기

특정 함수 호출이  
몇 회 호출되었는지 검증 방법

```
# pytest
from cal import Cal

def test_cal_mock(mock):
    cal = Cal()
    origin_get_sum = cal.get_sum # patch 전 백업

    def response_handler(a, b):
        if (a, b) == (1, 2):
            return 999
        return origin_get_sum(a, b)

    mock = mock.patch.object(Cal, 'get_sum', side_effect=response_handler)

    print()
    print(cal.get_sum(1, 2))
    print(cal.get_sum(1, 1))
    print(cal.get_sum(3, 4))
    print(cal.get_sum(1, 2))

    #assert mock.call_count == 4
    calls = [call.args for call in mock.call_args_list] # 함수호출 이력 전부 가져오기
    assert calls.count((1, 2)) == 2
```

patch 된 함수가  
4회 호출되었는지 검증

정확히 (1, 2)가  
몇번 호출되었는지 검증

**감사합니다.**