

ALU KATA

[참고] Refactoring KATA

카타 (일본어)

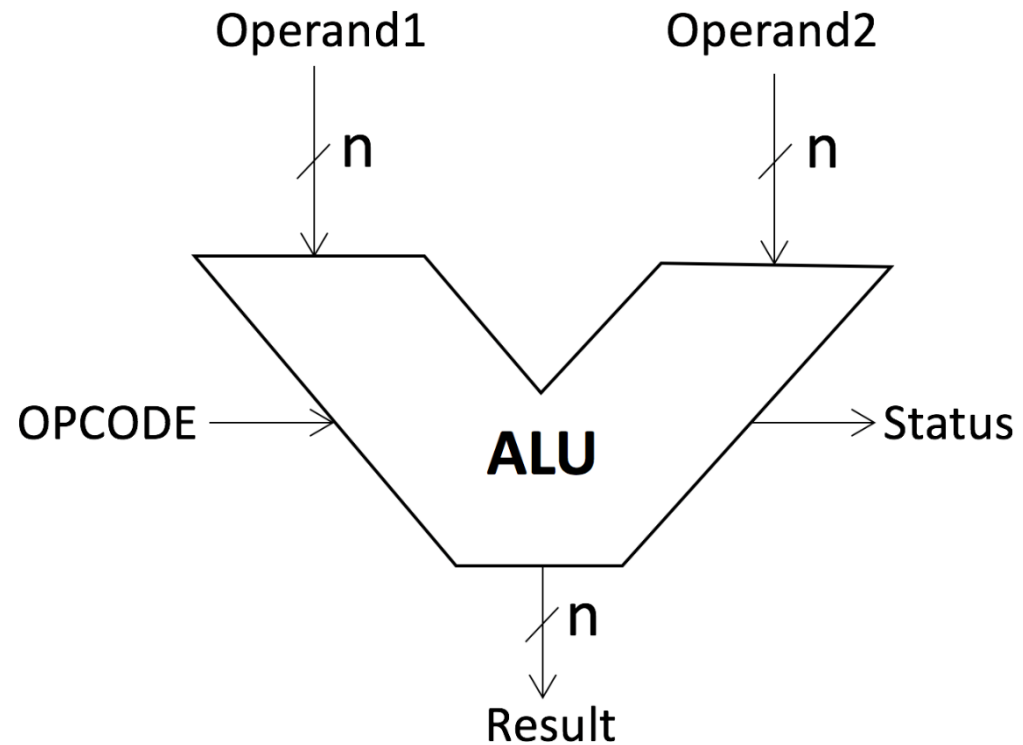
- 무술이 실전에서 바로 사용될 수 있도록,
- 상황을 가정하여 실무를 위한 시뮬레이션

리팩토링 카타

- 실무 리팩토링을 할 수 있도록
- 리팩토링 훈련 자료들

ALU 리팩토링

Operand1 과 Operand2에 수를 넣고,
OPCODE 를 넣으면 그 결과로 Status / Result 가 나온다.



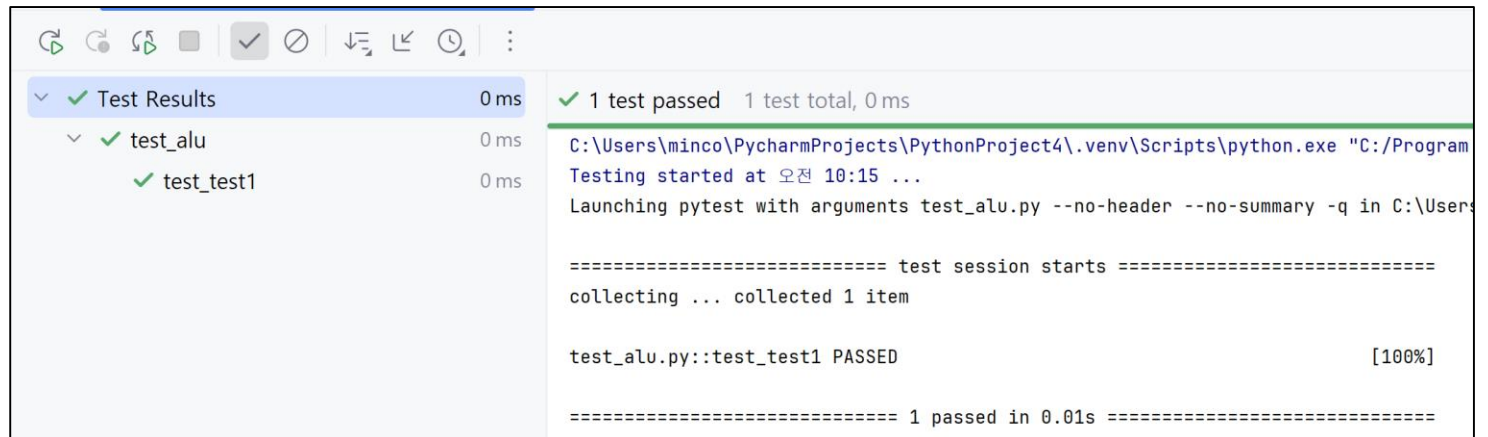
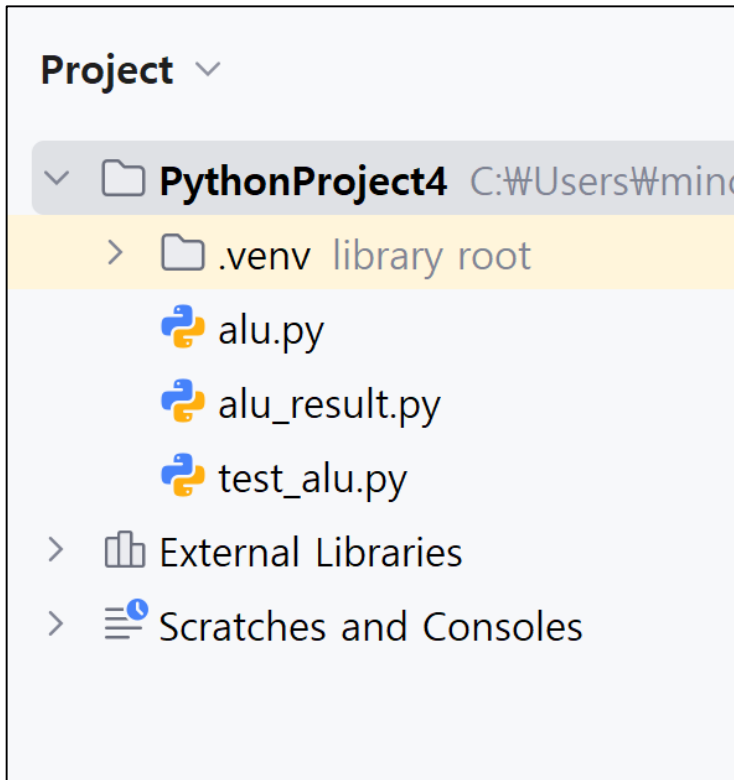
소스코드 링크

- <https://github.com/mincoding1/ALU>

```
1  from alu_result import ALU_result
2
3  class ALU :
4  def __init__(self) -> None:
5      super().__init__()
6      self.operand1 = -1
7      self.operand2 = -1
8      self.opcode = ""
9
10 def set_operand1(self, operand1):
11     self.operand1 = operand1
12
13 def set_operand2(self, operand2):
14     self.operand2 = operand2
15
16 def set_opcode(self, opcode):
17     self.opcode = opcode
18
19 def enable_signal(self, r : ALU_result):
20     if (self.opcode == "ADD") and (self.opcode != "MUL") and (self.opcode != "SUB") :
21         if self.operand1 != -1 and self.operand2 != -1 :
22             result = self.operand1 + self.operand2
23             r.set_result(result)
24             r.set_status(0)
25         elif self.operand1 == -1 :
26             r.set_result(65535)
27             r.set_status(1)
```

잘 동작되는지 확인한다.

테스트코드가 잘 동작되도록 pytest를 세팅한다.



리팩토링 하기 전,

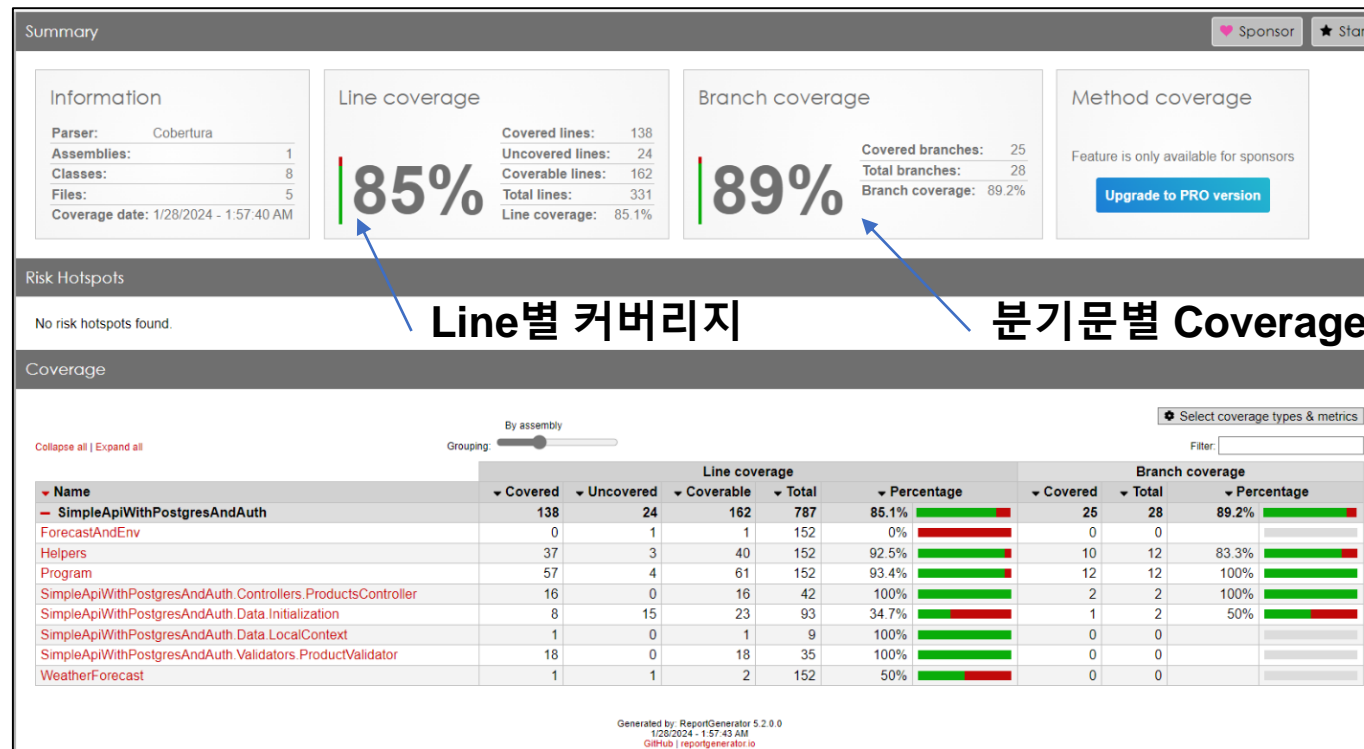
테스트 환경을 만들고 진행해야한다.

- 마음껏 리팩토링을 할 수 있다.

Code Coverage란?

테스트가 소스코드의 얼마나 **Touch** 했는지 측정할 수 있는 지표

- 테스트 코드가 어느 부분을 닿았는지 라인별 / 분기 문 별로 파악할 수 있음
- 코드 커버리지로 테스트 품질이 우수한 정도를 파악할 수 없음.
- 테스트가 되지 않고 있는 부분을 파악하기 위해 사용함




코드 커버리지 결과리포트 예시

Line Coverage

Line Coverage

- 라인별 코드 수행 여부를 판단
- Line Coverage = **90%** (9/10)
우측 10개 라인 중 9개 코드 수행
- 한계점
Branch 1과 2는 똑같이 중요하지만,
Branch 1 코드수가 길기에
Coverage 수치 상
대부분의 코드가 Cover되는 것으로 오해가능

```
def check_even(num):  
    if num % 2 == 0:  
        # Branch 1  
        print("루틴 1")  
        print("루틴 2")  
        print("루틴 3")  
        print("루틴 4")  
        print("루틴 5")  
        print("루틴 6")  
        print("루틴 7")  
        print("루틴 8")  
    else:  
        # Branch 2  
        print("매우 중요한 코드")  
  
if __name__ == "__main__":  
    check_even(2) 
```


Branch Coverage

Branch Coverage

- 분기문 별 검사
- Branch Coverage = **50%** (1/2)
우측 2개의 분기 중 1개 분기 수행

```
def check_even(num):  
    if num % 2 == 0:  
        # Branch 1  
        print("루틴 1")  
        print("루틴 2")  
        print("루틴 3")  
        print("루틴 4")  
        print("루틴 5")  
        print("루틴 6")  
        print("루틴 7")  
        print("루틴 8")  
    else:  
        # Branch 2  
        print("매우 중요한 코드")  
  
if __name__ == "__main__":  
    check_even(2)  ← Test Code
```

리팩토링에서 Code Coverage를 측정해야 하는 이유

리팩토링을 하기 전, Test Case를 얼마나 만들어야 하는가?

Legacy 코드에 대해, 잘 알고 있는 경우

- 중요한 Feature를 모두 검증 할 만큼 필요

Legacy 코드에 대해, 잘 모르는 경우

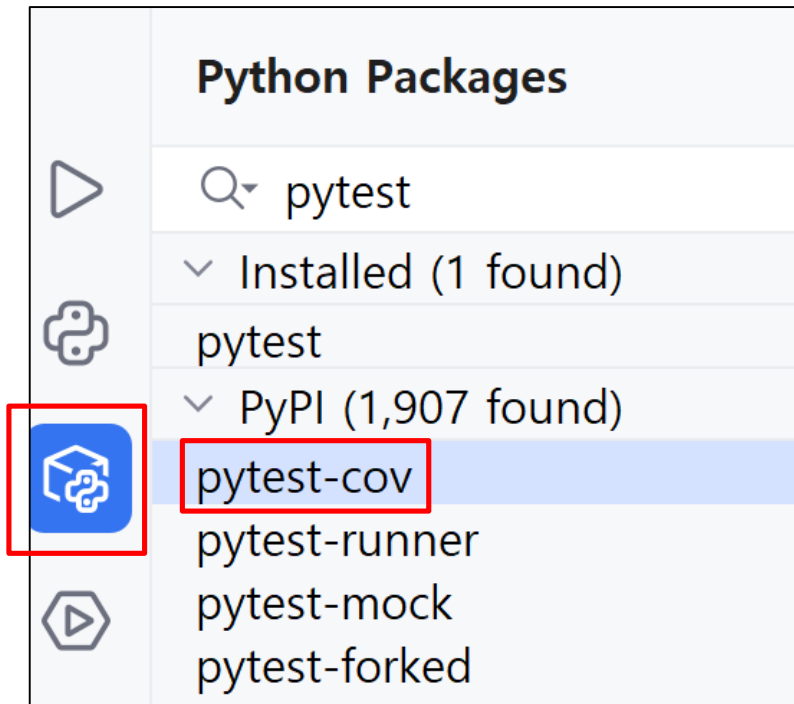


- 안전장치를 마련하기 위해
Code Coverage 100% 에 가깝도록 테스트 준비를 권장한다.
- Line 보다는 Branch Coverage 100%를 더 권장하지만
실습에는 Line Coverage를 사용한다.

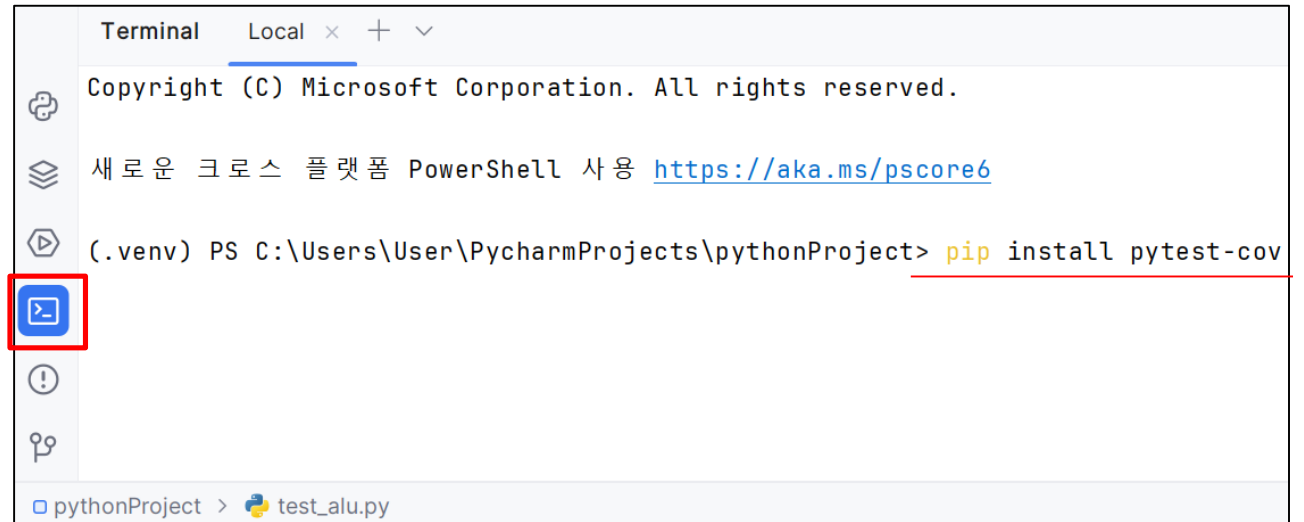
코드 커버리지 측정도구 설치

pytest의 커버리지 측정 플러그인

- 설치방법 1 : 아래 그림처럼, GUI로 설치
- 설치방법 2 : `pip install pytest-cov`



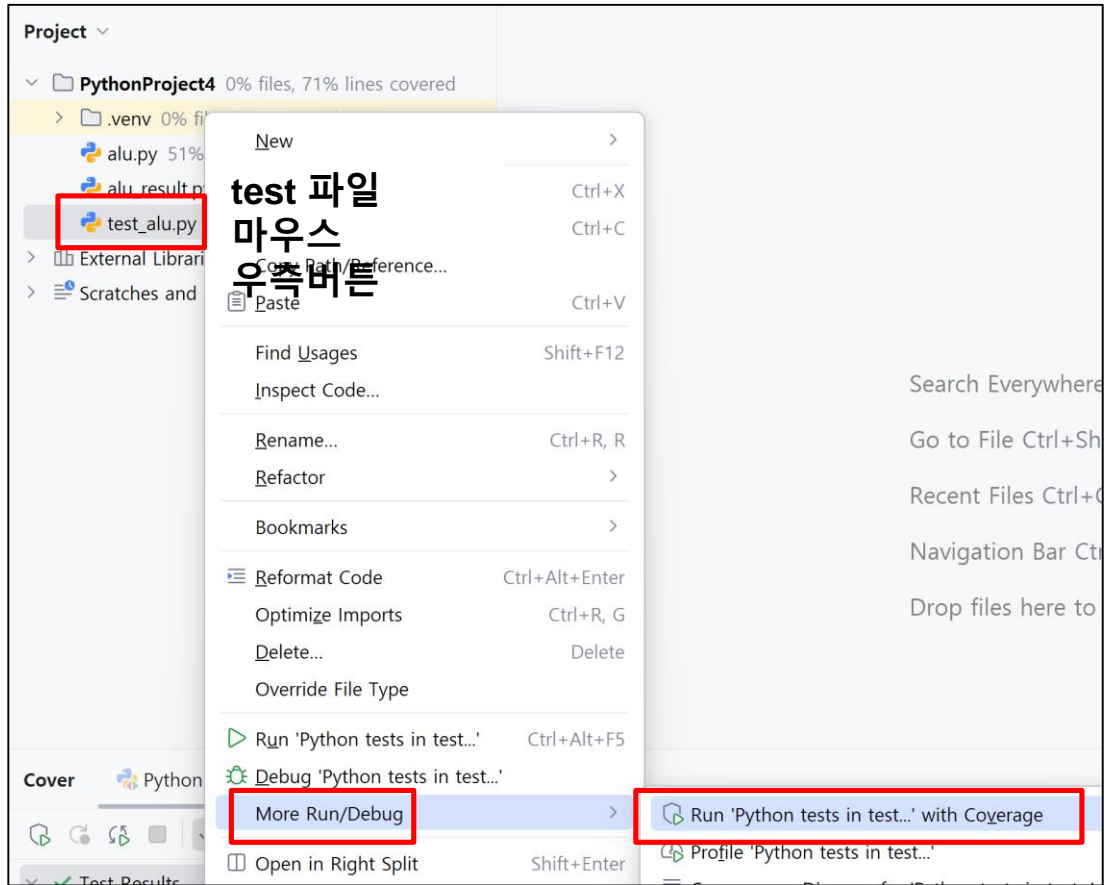
설치방법 1
선택 후 install 버튼 클릭



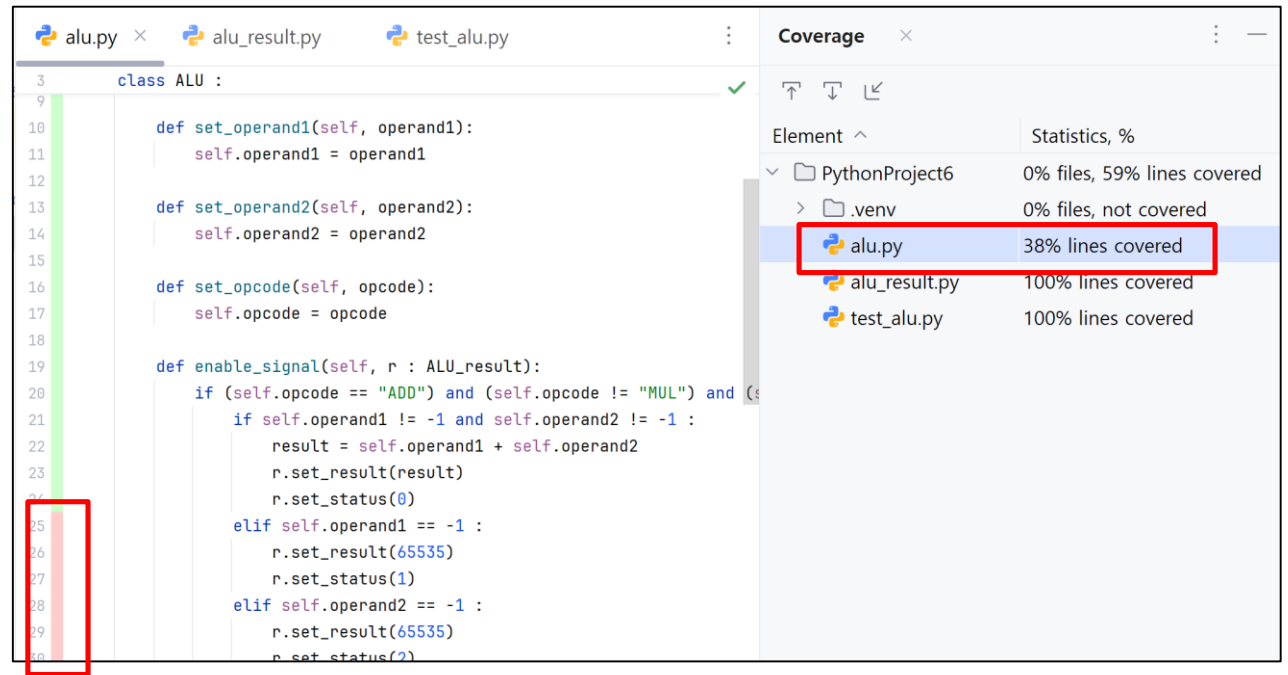
설치방법 2
`pip install pytest-cov`

[참고] 유료버전의 PyCharm 코드 커버리지 측정도구 실행

유료버전의 PyCharm에서는 GUI 상에서 측정 및 확인이 가능하다.



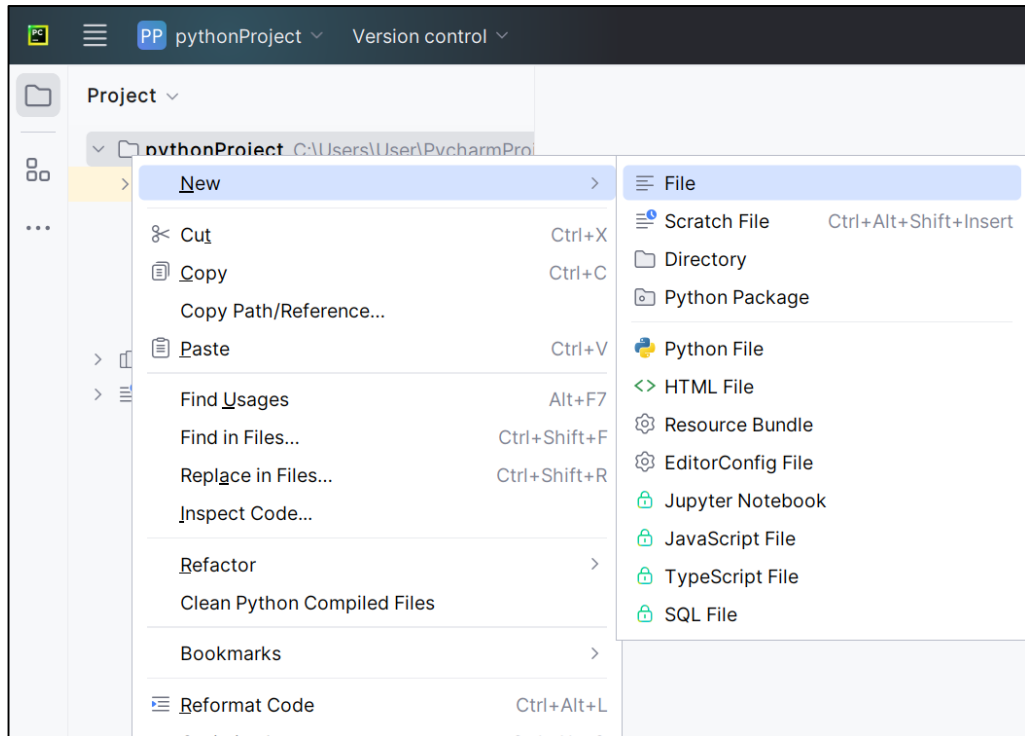
[유료버전 PyCharm 전용]
Coverage 측정 방법



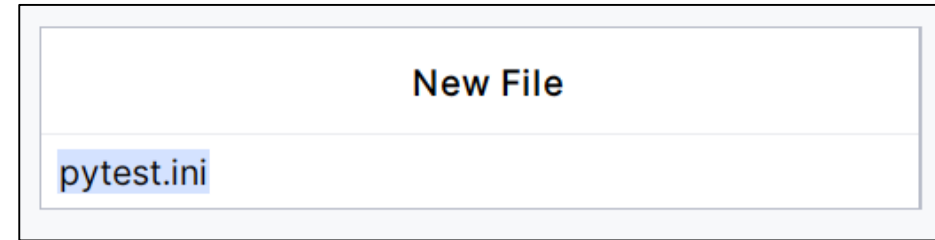
측정결과 화면

[필수] 커버리지 측정을 위해 pytest.ini 파일 추가

pytest를 터미널에서 실행할 수 있도록 ini 파일을 추가한다.
폴더 구조를 인식하고, 결과 파일을 html로 생성



프로젝트 마우스 우측버튼 > New > File



pytest.ini 파일 추가



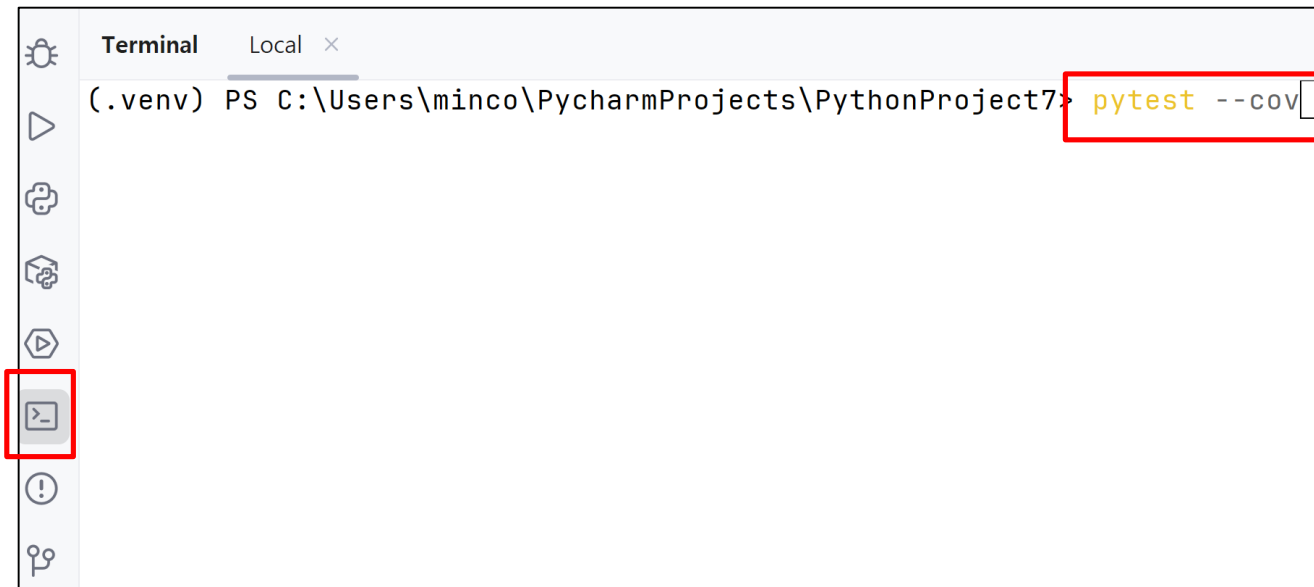
```
[pytest]
pythonpath = src
addopts = -s --cov-report=html
```

pytest의 프로젝트 구조 인식 &
coverage 측정을 위한 설정 추가

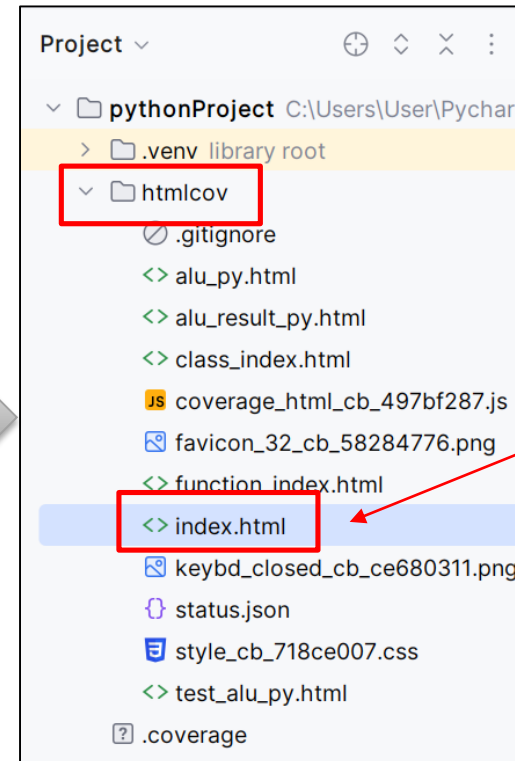
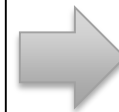
코드 커버리지 측정 도구 실행

터미널에서 직접 실행

pytest --cov 라고 입력하면, Coverage 측정 결과가 **index.html**에 기록됨



pytest --cov 명령어 입력

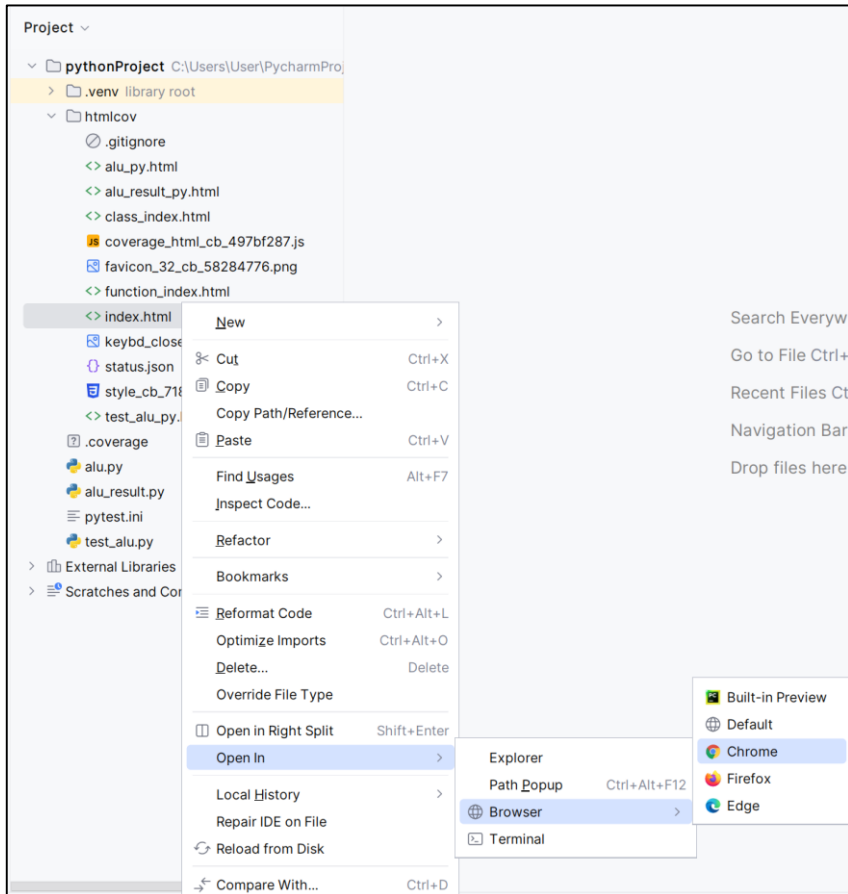


index.html
Coverage 측정
결과파일이 생성된다.

코드 커버리지 측정 결과 확인

line coverage 측정 결과 확인

- Test 코드가 닿지 못한 부분은 빨간색으로 표기된다.



브라우저에서 열기

Line Coverage

Coverage report: 71%

Files Functions Classes

coverage.py v7.8.2, created at 2025-05-28 13:28 +0900

File ▲	statements	missing	excluded	coverage
alu_result.py	13	0	0	100%
alu.py	49	24	0	51%
test_alu.py	20	0	0	100%
Total	82	24	0	71%

coverage.py v7.8.2, created at 2025-05-28 13:28 +0900

측정 결과 확인



[도전] Unit Test 작성하기

Unit Test를 작성한다.

Code Coverage가 100%가 나오는지 확인한다.

만약 안 나온다면, 유닛테스트를 추가하여 100%를 만들어낸다.

[도전] 리팩토링 시작!

Rule

수정해도 영향을 크게 끼치지 않는 부분부터 수정한다.

작은 수정마다 지속적으로 테스트를 한다.

전 세계 모든 개발자들에게 공개된다는 생각으로,
가독성이 좋은 코드로 개선한다.

감사합니다.