ILE3-028 Test Driven Development

소프트웨어 테스트를 위한 방법론인 TDD (Test Driven Development)에 대하여 학습

Test Driven Development *⊘*

기존의 개발 방법 ♂

소프트웨어 개발 시 기능을 구현하고, 해당 테스트의 검증을 위한 테스트 코드를 작성

Test Driven Development *⊘*

개발하는 기능이 정상적으로 작동을 하는지 검증을 하기 위한 테스트 코드를 먼저 작성하고 기능을 구현하는 방법

장점 ⊘

1. 빠르게 피드백 받을 수 있음

TDD를 사용하면 기능 단위로 테스트를 진행하기 때문에 코드가 모두 완성되기 전에 피드백을 받는 것이 가능

2. 작성한 코드가 가지는 불안정성을 개선

테스트코드를 이용하여 구현한 기능이 문제가 없는지 알 수 있음

3. 과도한 수정을 막을 수 있음

테스트코드에서 문제가 발생하지 않은 코드는 수정하지 않아도 되므로, 수정 대상을 줄일 수 있음

4. 개발 과정이 테스트 코드로 남아 있음

테스트코드를 확인하면서 기능에 대한 의사결정에 대한 변화 과정을 유추할 수 있음

Test Driven Development의 메인 프로세스 🔗

- RED: 테스트 실패
- GREEN: 테스트 성공
- REFACTOR: 리팩토링

RED ≈

- 구체적인 하나의 요구사항을 검증하는 하나의 테스트를 추가한다.
- 추가된 테스트가 실패하는지 확인한다.

GREEN ⊘

- 추가된 테스트를 포함하여, 모든 테스트가 성공하게끔 운영 코드를 변경한다.
- 테스트의 성공은 모든 요구사항을 만족했음을 의미한다.
- 테스트 성공을 위한 최소한의 코드 변경만 진행한다.

REFACTOR ⊘

- 작성한 코드를 정리한다.
- 인터페이스 뒤에 숨어 있는 구현을 개선한다.
- 가독성, 적용성, 성능을 고려한다.
- 모든 테스트의 성공을 전제로 한다

unittest Module &

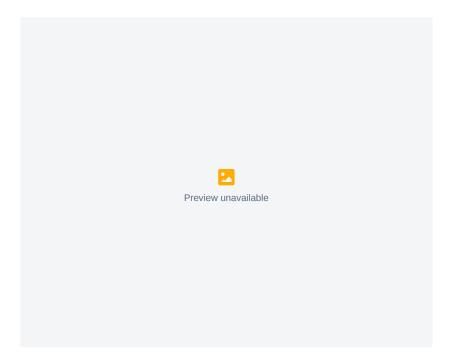
🕏 unittest — 단위 테스트 프레임워크

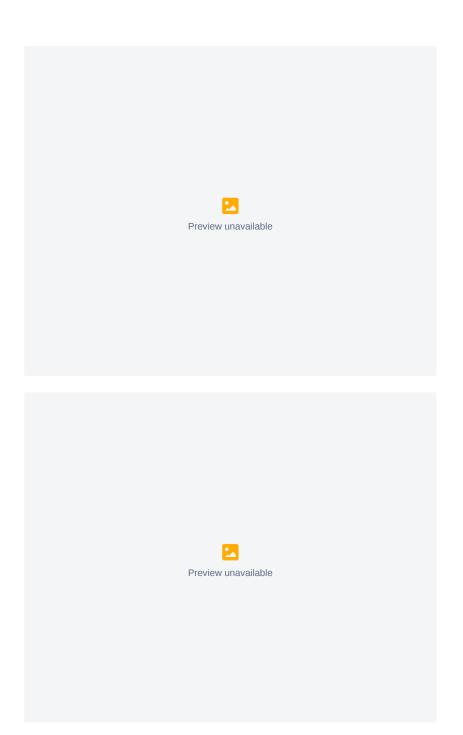
unittest.TestCase 를 상속받은 Class의 test_로 시작하는 Function에 대하여 테스트를 수행하고, 결과를 전달하여 주는 Module

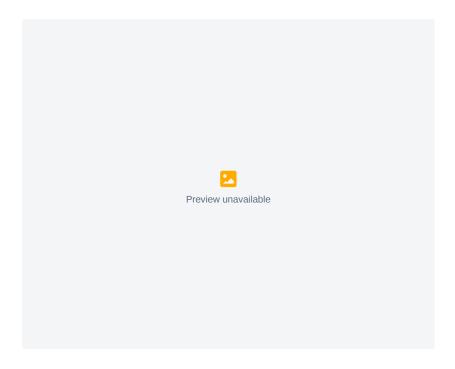
Visual Studio Code를 이용한 unittest ♂

Visual Studio Code에서 unittest 사용 설정 🔗

- 1. Visual Studio Code에서 Python 개발 환경이 설정되어 있음을 가정
- 2. 좌측의 메뉴에서 테스트 메뉴를 선택
- 3. Configure Python Tests 버튼을 클릭하고, Test Framework를 unitest로 선택한다.
- 4. Test Code가 위치한 경로를 선택한다.해당 과정에서는 별도의 경로가 없기에 . (Root Directory)를 선택
- 5. 테스트코드의 파일명 규칙을 선택한다.해당 과정에서는 test_로 시작하는 py파일을 사용 할 것이므로 test_*.py를 선택한다.







테스트코드 작성 🔗

테스트 대상

기본적인 사칙연산 (sum, extract)

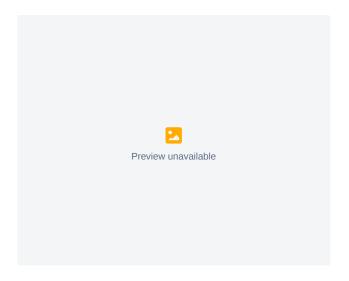
이전 과정에서 test_*.py 를 테스트 코드로 사용하기로 하였기에 test_func.py 파일을 추가 후 아래와 같이 작성한다.

```
1 from unittest import TestCase
2
3 import func
4
5 class BasicTest(TestCase):
6
     def test_success(self):
7
        self.assertEqual(func.add(1, 2), 3)
8
9
10
     def test_fail(self):
         self.assertNotEqual(func.add(1, 2), 3)
11
12
```

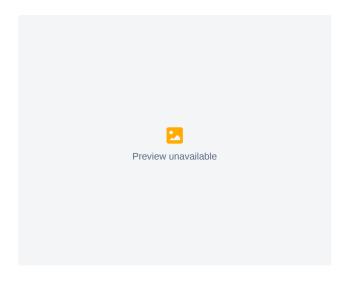
기능 구현을 위해 func.py 를 추가하고, 아래와 같이 작성한다.

```
1 def add(a, b):
2 return a + b
3
```

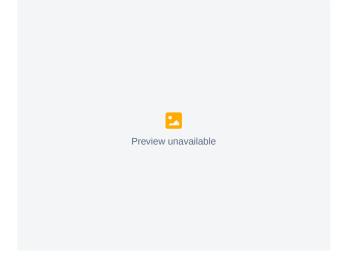
작성 후 테스트 메뉴에 들어가면 아래와 같이 test_func.py를 인식하여 BasicTest, test_success, test_fail이 추가 되어 있음



추가되어 있지 않은 경우, 파일 저장 후 상단의 새로고침 버튼을 클릭



테스트 시작 버튼을 클릭하면 테스트 결과가 표시 됨



테스트코드 분석 🔗

BasicTest Class 🔗

테스트코드에서 Class는 일반적인 Class의 개념과 다르게, 테스트코드의 기반 및 테스트코드의 Group으로 사용된다.

테스트코드의 Class는 unittest . TestCase 를 상속받아야 하며, 테스트 결과를 위한 함수들(self. assert*)을 제공한다.

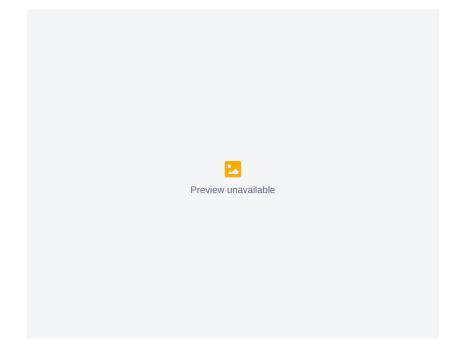
테스트코드 Class의 테스트코드들은 test_로 시작하여야 함 test_로 시작하지 않는 경우, 해당 함수는 테스트코드로 인식하지 않음

test_success Function 🔗

함수명이 test_로 시작하기 때문에 테스트코드로 인식한다. 해당 함수 내에서 self.assert* 함수들이 실패하는 경우, 테스트 결과를 실패로 보고 함

assert* 함수 🔗

테스트를 위한 기능의 결과값을 비교하기 위하여 unittest . TestCase 클래스에서 제공하는 함수로, 검사 내용이 일치하지 않는 경우 테스트 결과를 실패로 보고한다.



과제 🔗

설계한 내용을 TDD 방식으로 구현

- 로그인
 - ∘ 관리자 계정: admin / admin
 - ∘ 사용자 계정: user01 / password
 - 위 2개 계정 이외에는 로그인 실패
 - ∘ 로그인 시 관리자는 admin, 사용자 계정은 user를 반환
- POI 편집
 - 단일 POI GET 요청 시 아래의 데이터 반환
 - ID: 1
 - NAME: 테스트용
 - 이외의 ID가 입력 되는 경우는 None 반환

- POI 추가시
 - ID는 2를 반환
- POI 편집 시
 - ID가 1이고, NAME이 None이 아닌 경우 True 반환
 - 이외의 경우에는 False 반환
- POI 삭제 시
 - ID가 1인 경우 True 반환
 - 이외의 경우에는 False 반환