PyUnit

1.Create a simple TestCase class and test a basic function using assertEqual.

import unittest

def add\_numbers(a, b):

    return a + b

class TestMathOperations(**unittest**.**TestCase**):

    def test\_add\_numbers(self):

        result = add\_numbers(3, 5)

        self.assertEqual(result, 8)

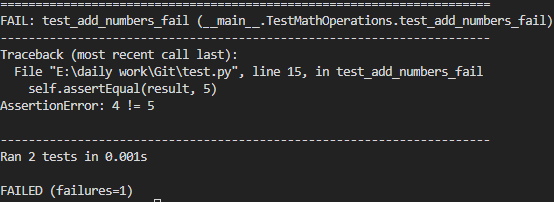
    def test\_add\_numbers\_fail(self):

        result = add\_numbers(2, 2)

        self.assertEqual(result, 5)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()



2.Write tests that check string operations using assertTrue, assertFalse, assertIn, assertNotIn.

3.Test a function that raises an exception using assertRaises.

4.Use assertRaisesRegex to match the error message.

5.Test floating point math using assertAlmostEqual and explain why it matters.

6.Use subTest to test multiple inputs for the same function in one method.

7.Write a test that uses assertLogs to check that a function logs at WARNING level.

8.Write a test that checks if a warning is raised using assertWarns.

9.Demonstrate setUp and tearDown to create and clean up a temp file.

10.Use setUpClass and tearDownClass for a database connection mock.

11.Use addCleanup instead of tearDown to ensure cleanup runs even if a test fails.

12.Write tests with skipIf and skipUnless decorators based on Python version.

13.Dynamically skip a test at runtime using self.skipTest().

14.Mark a test as expected failure using @unittest.expectedFailure.

15.Organize tests into multiple files and run them with python -m unittest discover.

16.Manually create a TestSuite combining tests from different classes/modules.

17.Customize test discovery using the load\_tests protocol in a module.

18.Run only a single test method from the command line (python -m unittest module.Class.method).

19.Use unittest.mock.patch to mock a function in another module.

20.Patch an object in the correct namespace (demonstrate wrong vs correct patching).

21.Use patch.object to replace a class method.

22.Use patch.dict to temporarily modify os.environ.

23.Use MagicMock to test a class that calls magic methods (\_\_len\_\_, \_\_getitem\_\_).

24.Use side\_effect to raise an exception on a mocked function.

25.Use assert\_called\_once\_with, assert\_has\_calls, and call\_args\_list to validate calls.