

# Unit Testing Framework for Python [PyTest]

- 

What is PyTest?

- [Install PyTest](#)

[Write TestCase in PyTest Format](#)

[Write Multiple Test Cases in a File: Check Execution Options](#)

[Execute Test Case using verbos\(-v\)](#)

[Skip execution of any specific test cases](#)

[Conditionally skip execution of any specific test cases](#)

[Execute only specific test cases](#)

[Tagging/Grouping](#)

[Handle Custom Tagging issues](#)

[Assertions](#)

[PyTest Fixtures](#)

- [Attachment for reference:](#)

## What is PyTest?

*Unit testing framework for Python (UnitTest or Nose)*

*Options to conditionally execute cases*

*Can setup pre-requisite and post script*

*Can Add Assertions **\*compares actual result with expected result***

*Options to generate report*

## Install PyTest

**Command: pip install pytest**

## Write TestCase in PyTest Format

**Note:** *To execute test cases simultaneously, ensure that the file name and method names start with **`tests`***

- **Create a New Project:** Begin by creating a new directory to serve as the root of your project. Let's name it "PytestLearning".
- **Create Python File:** Inside your project directory, create a Python file where you'll write your test cases. Name it something descriptive, like "test\_TC001\_First.py".
- **Write Test Case Method:** In "test\_suite.py", write your test case methods using PyTest, which is known for its simplicity and readability. Here's an example of a basic PyTest test case:

### test\_TC001\_First.py

```
import pytest

def tc_test_addition():
    assert 2 + 2 == 4
```

- **Change Method Name for PyTest Compatibility:** Adjust the method name to conform to PyTest's naming convention. PyTest identifies test functions based on their name prefixes. For instance, rename `tc\_test\_addition` to `test\_addition`:

### test\_TC001\_First.py

```
//Test case code must be written inside a method
//Method name must be started with test

import pytest

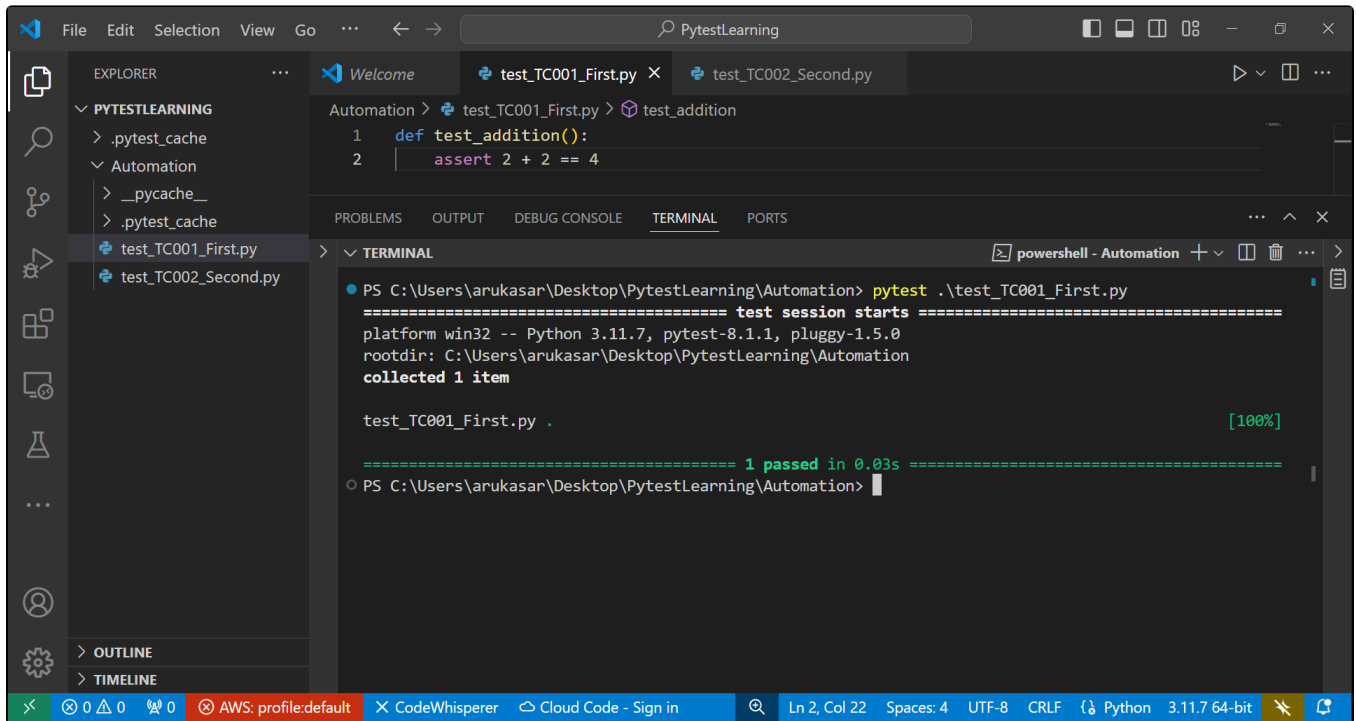
def test_addition():
    assert 2 + 2 == 4
```

- **Run Test Case using PyTest:** Navigate to your project directory in the terminal and execute your test cases using PyTest:

#### command to run single test case

```
pytest test_TC001_First.py
```

Results:

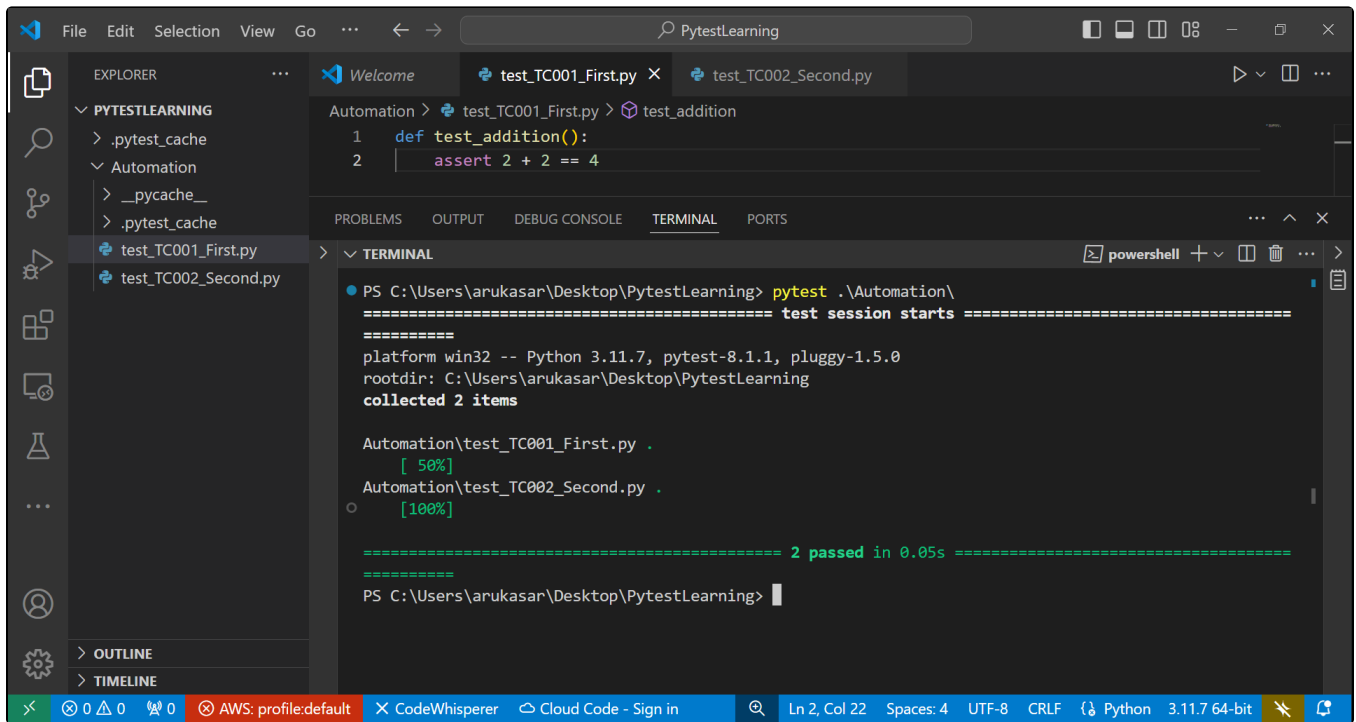
A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'PYTESTLEARNING' with a subfolder 'Automation' containing two files: 'test\_TC001\_First.py' and 'test\_TC002\_Second.py'. The main editor window has two tabs: 'test\_TC001\_First.py' and 'test\_TC002\_Second.py'. The 'test\_TC001\_First.py' tab is active, showing a Python function 'test\_addition()' with an 'assert 2 + 2 == 4' statement. Below the editor, the 'TERMINAL' panel is open, showing the command 'pytest .\test\_TC001\_First.py' being executed in a PowerShell terminal. The output of the command is displayed, showing 'test session starts', platform information, root directory, and the result '1 passed in 0.03s' with a green progress bar at [100%]. The status bar at the bottom indicates the current file is at line 2, column 22, with 4 spaces, UTF-8 encoding, CRLF line endings, and Python 3.11.7 64-bit.

- **Write Test Cases Inside a Folder:** To maintain better project structure and organization, create a folder named "tests" within your project directory. Then, move "test\_suite.py" into this folder.
- **Execute All Test Cases Inside the Folder:** Run all test cases located within the "tests" folder by providing the folder path to PyTest:

#### command to run all the test case inside the given folder

```
pytest Automation/
```

Results:



## Write Multiple Test Cases in a File: Check Execution Options

### test\_TC001\_First.py

```
def test_addition():
    assert 2 + 2 == 4

def test_substraction():
    assert 5 - 3 == 2
```

### command to run multiple test cases in a single file

```
pytest <file_name>
```

The screenshot shows the VS Code interface with the Explorer pane on the left displaying the project structure: PYTESTLEARNING, .pytest\_cache, Automation, and test\_TC001\_First.py. The main editor shows the code for test\_TC001\_First.py, which contains two test functions: test\_addition and test\_subtraction. The TERMINAL pane at the bottom shows the output of the command `pytest .\test_TC001_First.py`. The output indicates that the test session started, collected 2 items, and both tests passed successfully, resulting in a 100% success rate.

```
Automation > test_TC001_First.py > test_subtraction
1 def test_addition():
2     assert 2 + 2 == 4
3
4 def test_subtraction():
5     assert 5 - 3 == 2
```

```
PS C:\Users\arukasar\Desktop\PytestLearning\Automation> pytest .\test_TC001_First.py
===== test session starts =====
platform win32 -- Python 3.11.7, pytest-8.1.1, pluggy-1.5.0
rootdir: C:\Users\arukasar\Desktop\PytestLearning\Automation
collected 2 items

test_TC001_First.py .. [100%]

===== 2 passed in 0.02s =====
PS C:\Users\arukasar\Desktop\PytestLearning\Automation>
```

command to run multiple test cases in a single file

```
pytest <folder_name>
```

The screenshot shows the VS Code interface with the Explorer pane on the left displaying the project structure. The main editor shows the code for test\_TC001\_First.py. The TERMINAL pane at the bottom shows the output of the command `pytest .\Automation\`. The output indicates that the test session started, collected 3 items (from both test\_TC001\_First.py and test\_TC002\_Second.py), and all tests passed successfully, resulting in a 100% success rate.

```
Automation > test_TC001_First.py > test_subtraction
1 def test_addition():
2     assert 2 + 2 == 4
3
4 def test_subtraction():
5     assert 5 - 3 == 2
```

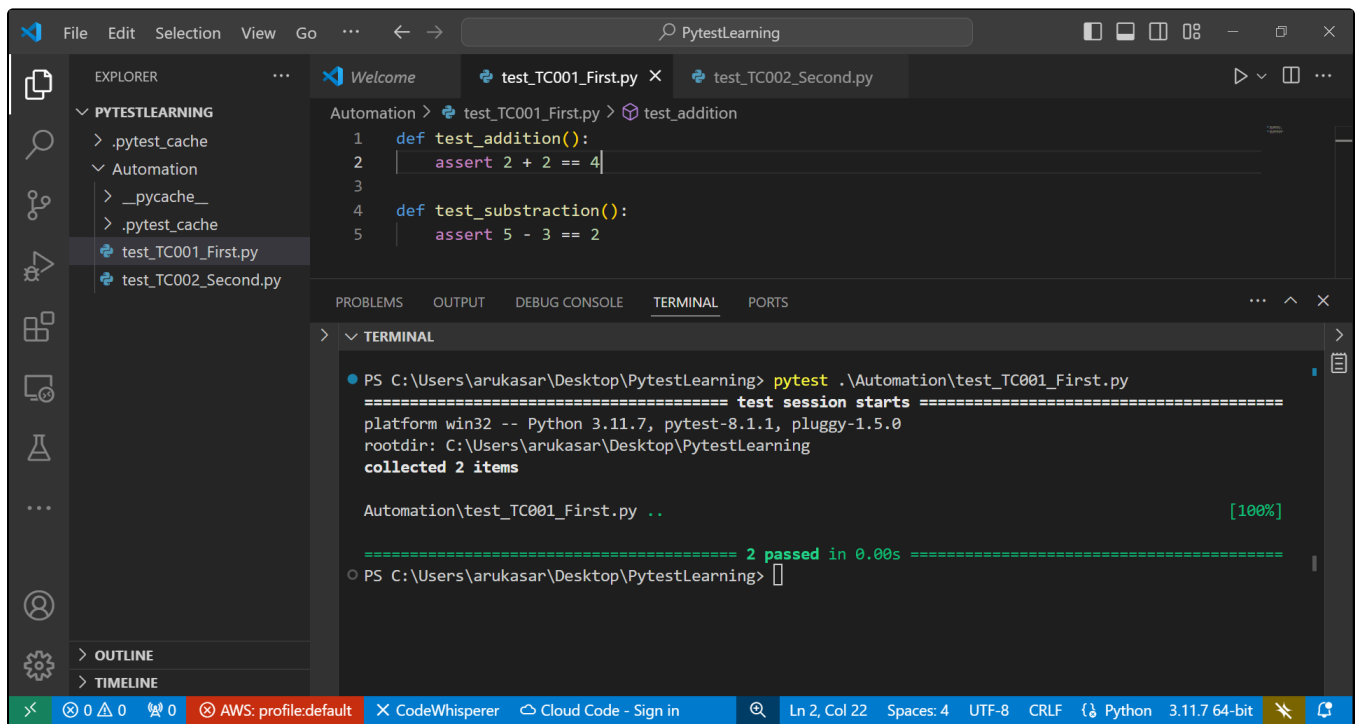
```
PS C:\Users\arukasar\Desktop\PytestLearning> pytest .\Automation\
===== test session starts =====
platform win32 -- Python 3.11.7, pytest-8.1.1, pluggy-1.5.0
rootdir: C:\Users\arukasar\Desktop\PytestLearning
collected 3 items

Automation\test_TC001_First.py .. [ 66%]
Automation\test_TC002_Second.py . [100%]

===== 3 passed in 0.02s =====
PS C:\Users\arukasar\Desktop\PytestLearning>
```

command to run multiple test cases in a single file

```
pytest <folder_name><file_name>
```



## Execute Test Case using verbos(-v)

Here's what each option does:

-s: Allows printing output to the console. This option is useful for displaying print statements within your test code.

-v: Enables verbose mode, which provides more detailed output about the test execution process, including the names of the tests being run and their outcomes.

### test\_TC001\_First.py

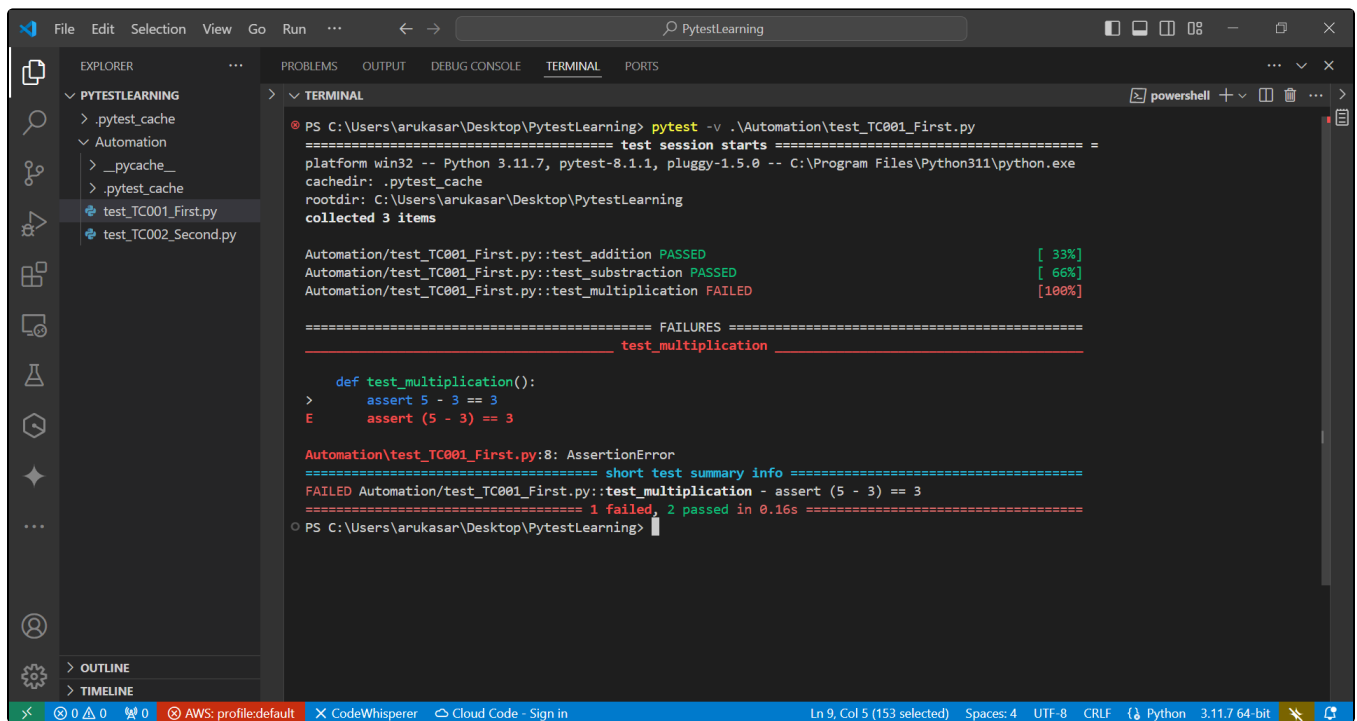
```
def test_addition():
    assert 2 + 2 == 4

def test_substraction():
    assert 5 - 3 == 2

def test_multiplication():
    assert 5 - 3 == 3
```

### verbos command

```
pytest -v <file_name>
```



```
PS C:\Users\arukasar\Desktop\PytestLearning> pytest -v .\Automation\test_TC001_First.py
===== test session starts =====
platform win32 -- Python 3.11.7, pytest-8.1.1, pluggy-1.5.0 -- C:\Program Files\Python311\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\arukasar\Desktop\PytestLearning
collected 3 items

Automation/test_TC001_First.py::test_addition PASSED [ 33%]
Automation/test_TC001_First.py::test_subtraction PASSED [ 66%]
Automation/test_TC001_First.py::test_multiplication FAILED [100%]

===== FAILURES =====
test_multiplication

  def test_multiplication():
>     assert 5 - 3 == 3
E       assert (5 - 3) == 3

Automation/test_TC001_First.py:8: AssertionError
===== short test summary info =====
FAILED Automation/test_TC001_First.py::test_multiplication - assert (5 - 3) == 3
===== 1 failed, 2 passed in 0.16s =====
PS C:\Users\arukasar\Desktop\PytestLearning>
```

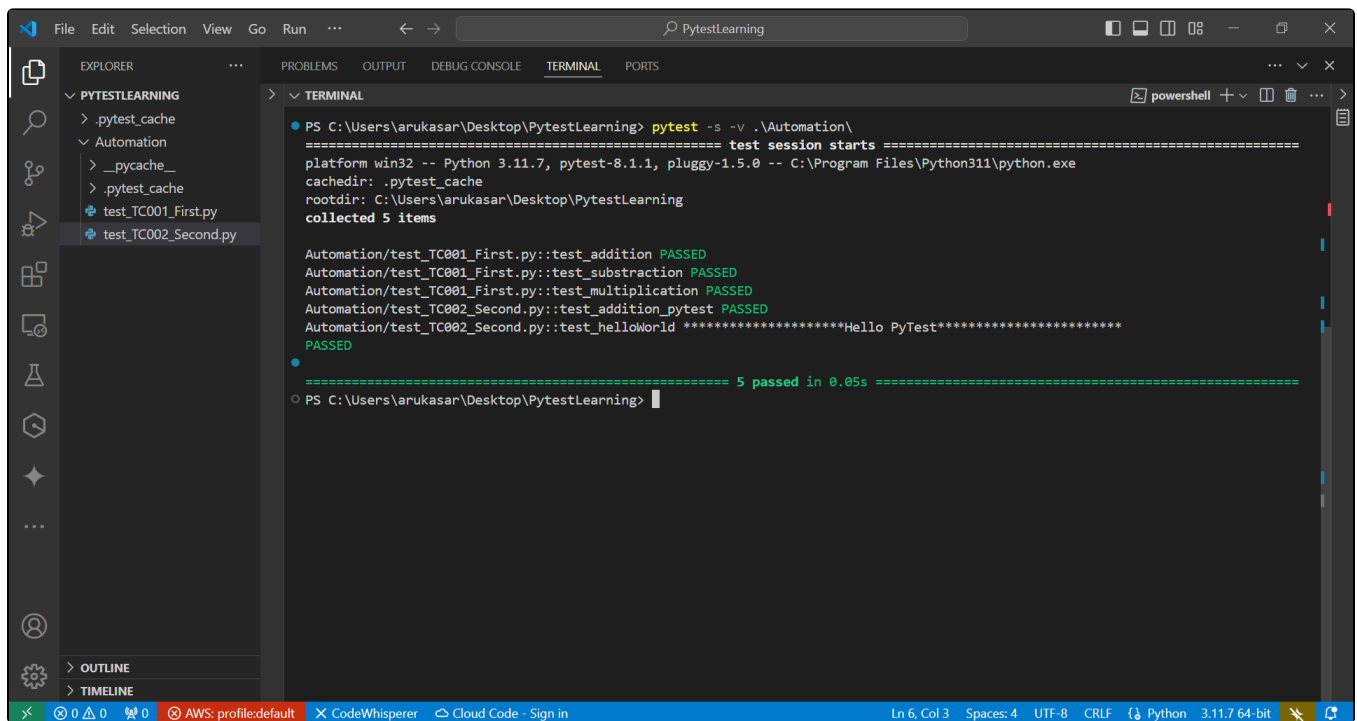
### test\_TC002\_Second.py

```
def test_addition_pytest():
    assert 2 + 6 == 8

def test_helloWorld():
    print("*****Hello PyTest*****")
```

### print statement command

```
pytest -s -v <file_name>
```



## Skip execution of any specific test cases

### test\_TC002\_Second.py

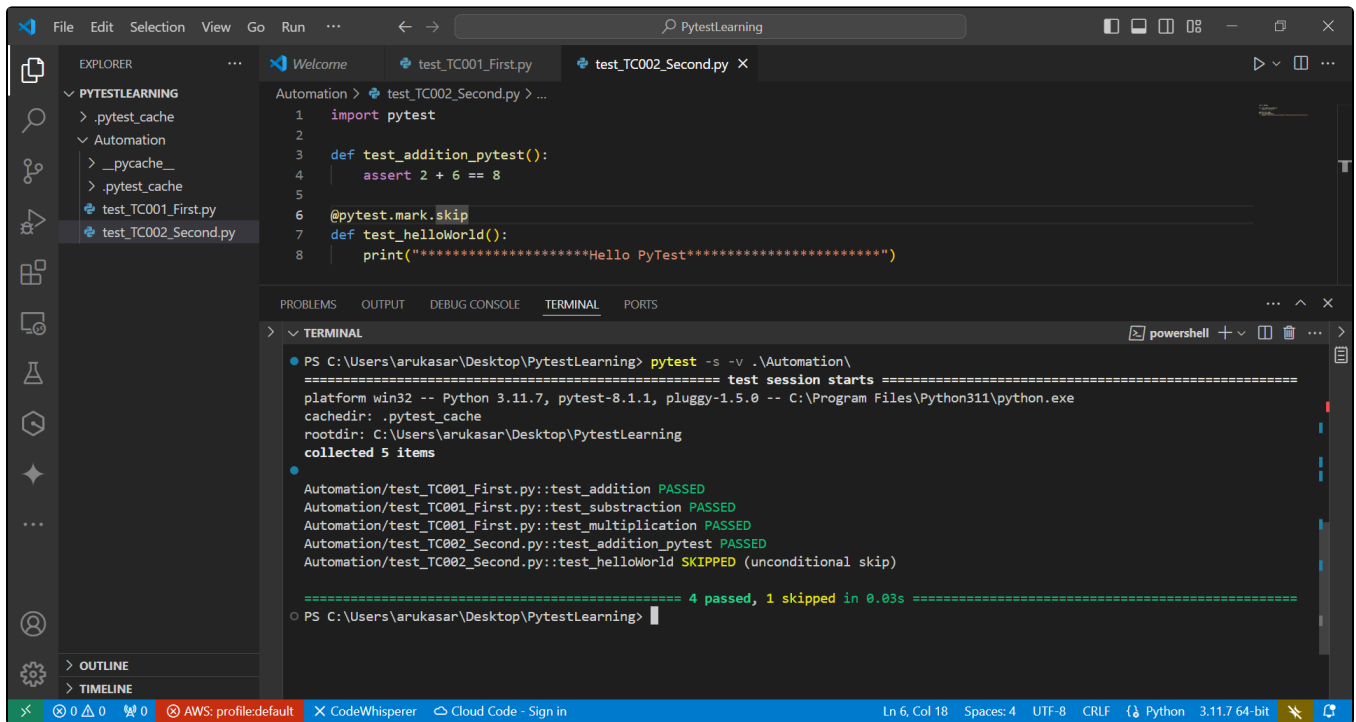
```
import pytest

def test_addition_pytest():
    assert 2 + 6 == 8

@pytest.mark.skip
def test_helloWorld():
    print("*****Hello PyTest*****")
```

### print statement command

```
pytest -s -v <file_name>
```



### test\_TC002\_Second.py

```
import pytest

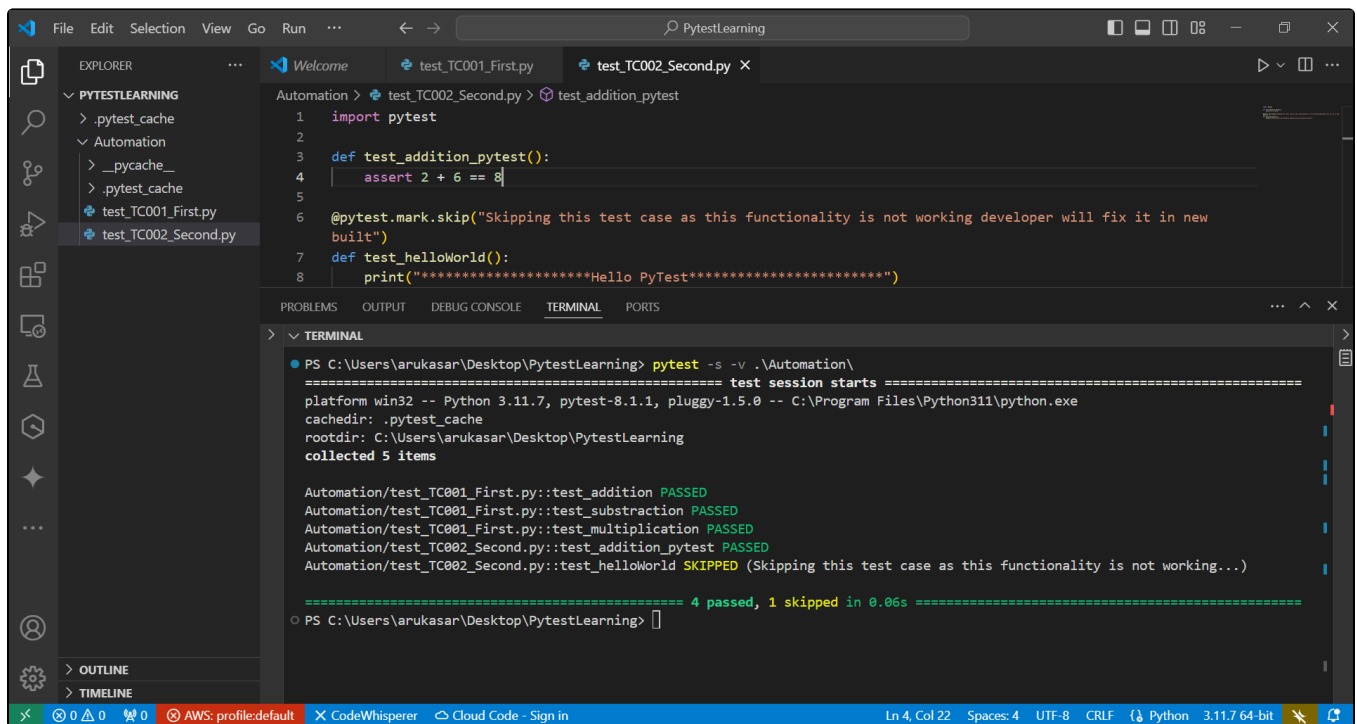
def test_addition_pytest():
    assert 2 + 6 == 8

//Decorator
@pytest.mark.skip("Skipping this test case as this functionality is not working developer will fix it in new
built")
def test_helloWorld():
    print("*****Hello PyTest*****")
```

### print statement command

```
pytest -s -v <file_name>
```





## Conditionally skip execution of any specific test cases

### test\_TC002\_Second.py

```
import pytest

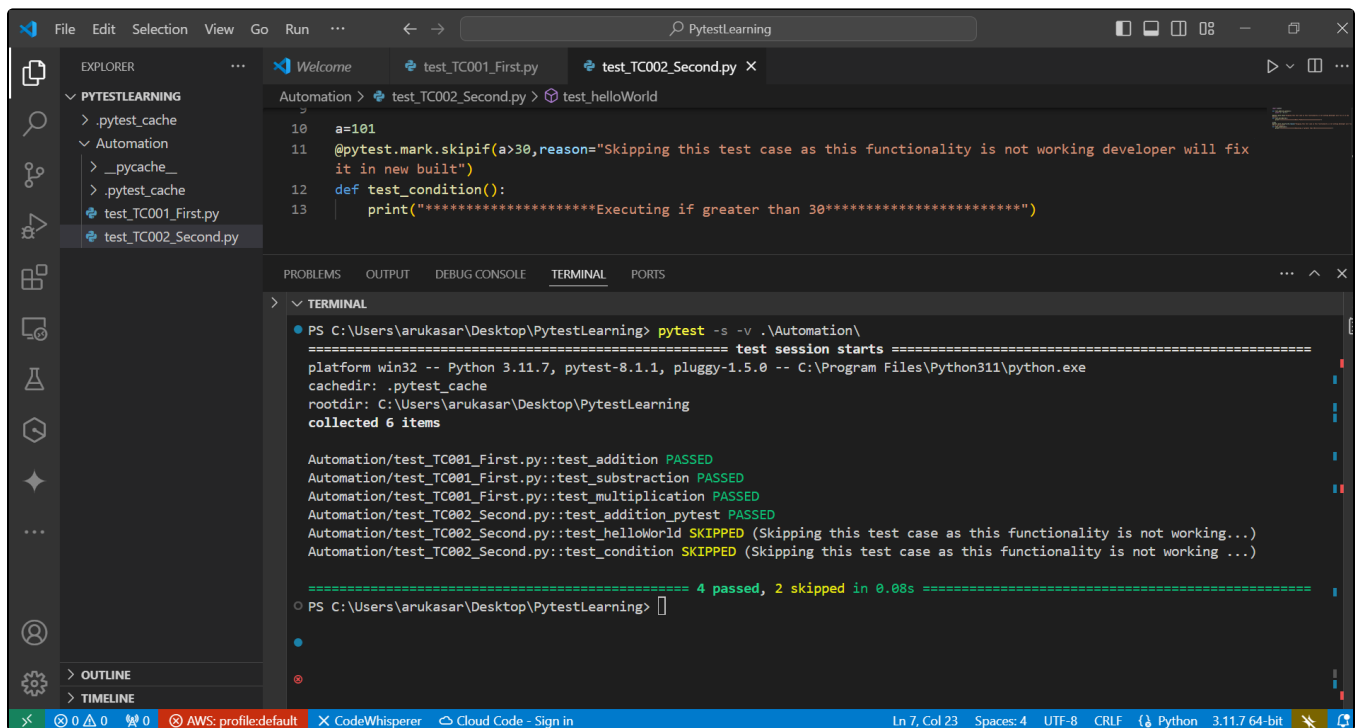
def test_addition_pytest():
    assert 2 + 6 == 8

@pytest.mark.skip("Skipping this test case as this functionality is not working developer will fix it in new
built")
def test_helloWorld():
    print("*****Hello PyTest*****")

a=101
@pytest.mark.skipif(a>30,reason="Skipping this test case as this functionality is not working developer will
fix it in new built")
def test_condition():
    print("*****Executing if greater than 30*****")
```

### print statement command

```
pytest -s -v <file_name>
```



## Execute only specific test cases

### Approach 1:

- You can execute specific test cases in pytest using the `-k` option, which allows you to specify a substring expression to match test names against. Here's how you can do it:
- Replace "test\_specific\_case" with the substring that matches the test case name(s) you want to execute. This will run only the test cases whose names contain the specified substring.

#### command to execute specific test case

```
pytest -k "test_specific_case"
```

### Approach 2:

- Alternatively, if you have marked your test cases with custom markers, you can use the `-m` option to execute only the tests marked with a specific marker. For example:
- Replace "specific\_marker" with the name of the marker you've defined for the test case(s) you want to execute.

#### command to execute specific test case using marker

```
pytest -m specific_marker
```

### Approach 3:

- You can also combine multiple options. For example, if you want to execute only the test cases with a specific substring in their names and marked with a specific marker, you can do:
- This command will execute only the test cases that match both the substring expression and the specified marker.

#### command to execute specific test case using marker

```
pytest -k "test_specific_case" -m specific_marker
```

## Tagging/Grouping

*Define tags for the test cases*

1. Write tags to test cases
2. Execute test cases with single tags
3. Skip execution of test cases by giving tag name
4. Write multiple tags on test cases
5. Execute test cases using more than 1 tag(or and)

**Writing Tags for Test Cases:** Assign tags to your test cases using decorators or markers in pytest. These tags can represent various attributes like functionality, priority, environment, etc.

### test\_TC003\_Third.py

```
import pytest

@pytest.mark.smoke
def test_addition():
    print("This is Smoke Test")
    assert 2 + 2 == 4

@pytest.mark.sanity
def test_substraction():
    print("This is Sanity Test")
    assert 5 - 3 == 2

@pytest.mark.smoke
def test_multiplication():
    print("This is Smoke Test")
    assert 5 - 3 == 2
```

### test\_TC004\_Fourth.py

```
import pytest

@pytest.mark.sanity
def test_addition_pytest():
    print("This is Sanity Test")
    assert 2 + 6 == 8

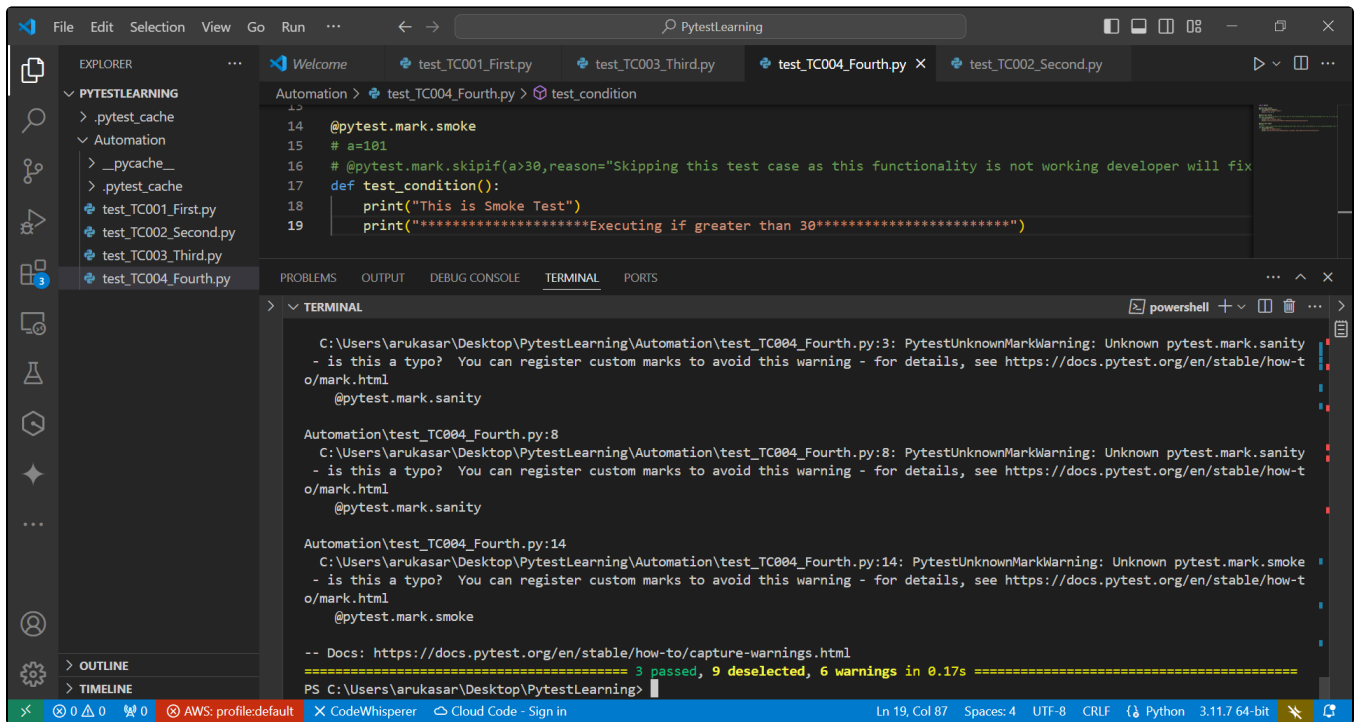
@pytest.mark.sanity
#@pytest.mark.skip("Skipping this test case as this functionality is not working developer will fix it in new built")
def test_helloWorld():
    print("This is Sanity Test")
    print("*****Hello PyTest*****")

@pytest.mark.smoke
# a=101
# @pytest.mark.skipif(a>30,reason="Skipping this test case as this functionality is not working developer will fix it in new built")
def test_condition():
    print("This is Smoke Test")
    print("*****Executing if greater than 30*****")
```

**Executing Test Cases with Single Tags:** Use the -m option followed by the tag name to execute test cases with a specific tag.

### tag/grouping command

```
pytest -m smoke <folder_name>
```



The screenshot shows a VS Code editor with a file named `test_TC004_Fourth.py` open. The file contains the following code:

```
14 @pytest.mark.smoke
15 # a=101
16 # @pytest.mark.skipif(a>30,reason="Skipping this test case as this functionality is not working developer will fix
17 def test_condition():
18     print("This is Smoke Test")
19     print("*****Executing if greater than 30*****")
```

The terminal output shows the following warnings and results:

```
C:\Users\arukasar\Desktop\PytestLearning\Automation\test_TC004_Fourth.py:3: PytestUnknownMarkWarning: Unknown pytest.mark.sanity
- is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-t
o/mark.html
@pytest.mark.sanity

Automation\test_TC004_Fourth.py:8
C:\Users\arukasar\Desktop\PytestLearning\Automation\test_TC004_Fourth.py:8: PytestUnknownMarkWarning: Unknown pytest.mark.sanity
- is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-t
o/mark.html
@pytest.mark.sanity

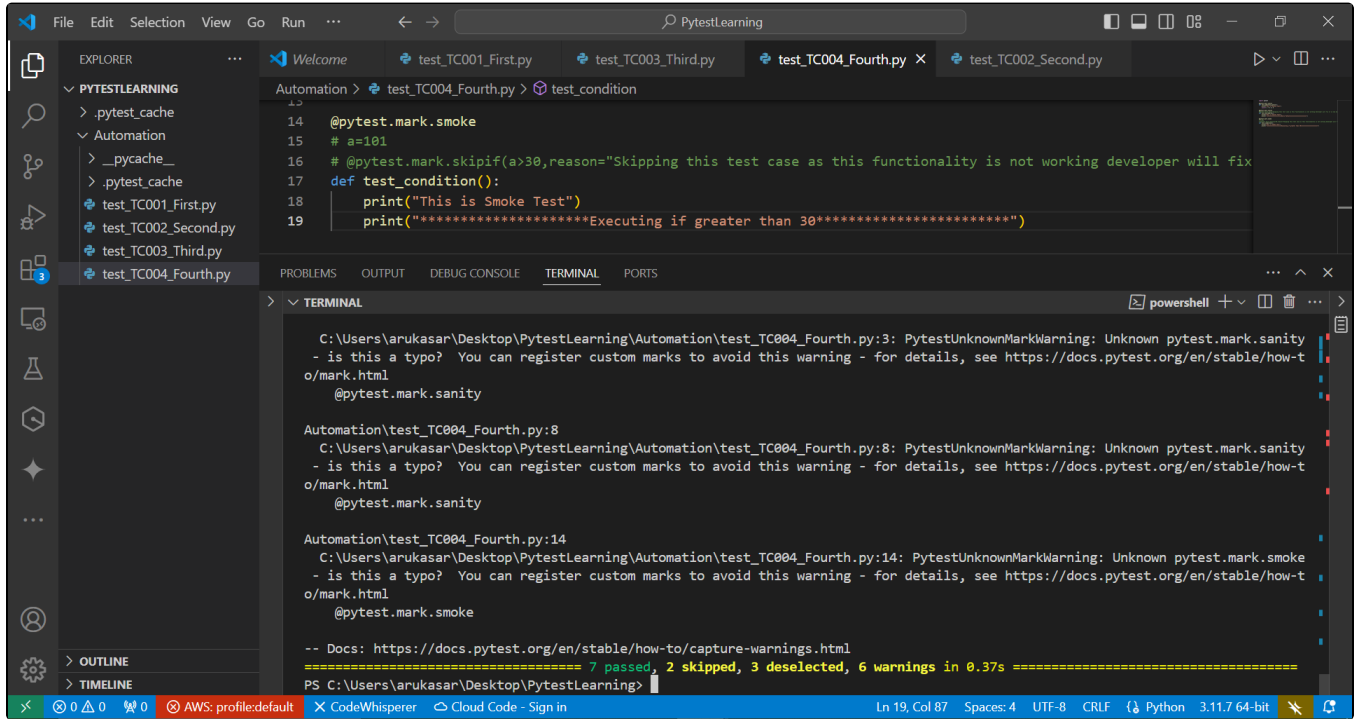
Automation\test_TC004_Fourth.py:14
C:\Users\arukasar\Desktop\PytestLearning\Automation\test_TC004_Fourth.py:14: PytestUnknownMarkWarning: Unknown pytest.mark.smoke
- is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-t
o/mark.html
@pytest.mark.smoke

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 3 passed, 9 deselected, 6 warnings in 0.17s =====
PS C:\Users\arukasar\Desktop\PytestLearning>
```

**Skipping Execution of Test Cases by Tag Name:** Skip executing test cases with a specific tag by using the `-m` option with a tilde (`~`) followed by the tag name.

tag/grouping command

pytest -m "not smoke" <folder\_name>



The screenshot shows a VS Code editor with a file named `test_TC004_Fourth.py` open. The file contains the following code:

```
14 @pytest.mark.smoke
15 # a=101
16 # @pytest.mark.skipif(a>30,reason="Skipping this test case as this functionality is not working developer will fix
17 def test_condition():
18     print("This is Smoke Test")
19     print("*****Executing if greater than 30*****")
```

The terminal output shows the following warnings and results:

```
C:\Users\arukasar\Desktop\PytestLearning\Automation\test_TC004_Fourth.py:3: PytestUnknownMarkWarning: Unknown pytest.mark.sanity
- is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-t
o/mark.html
@pytest.mark.sanity

Automation\test_TC004_Fourth.py:8
C:\Users\arukasar\Desktop\PytestLearning\Automation\test_TC004_Fourth.py:8: PytestUnknownMarkWarning: Unknown pytest.mark.sanity
- is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-t
o/mark.html
@pytest.mark.sanity

Automation\test_TC004_Fourth.py:14
C:\Users\arukasar\Desktop\PytestLearning\Automation\test_TC004_Fourth.py:14: PytestUnknownMarkWarning: Unknown pytest.mark.smoke
- is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-t
o/mark.html
@pytest.mark.smoke

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 7 passed, 2 skipped, 3 deselected, 6 warnings in 0.37s =====
PS C:\Users\arukasar\Desktop\PytestLearning>
```

**Assigning Multiple Tags to Test Cases:** You can assign multiple tags to a single test case by chaining multiple markers.

#### test\_TC003\_Third.py

```
import pytest

@pytest.mark.smoke
@pytest.mark.regression
def test_addition():
    print("This is Smoke Test")
    print("This is Regression Test")
    assert 2 + 2 == 4

@pytest.mark.sanity
@pytest.mark.regression
def test_substraction():
    print("This is Sanity Test")
    print("This is Regression Test")
    assert 5 - 3 == 2

@pytest.mark.smoke
def test_multiplication():
    print("This is Smoke Test")
    assert 5 - 3 == 2
```

#### test\_TC004\_Fourth.py

```
import pytest

@pytest.mark.sanity
def test_addition_pytest():
    print("This is Sanity Test")
    assert 2 + 6 == 8

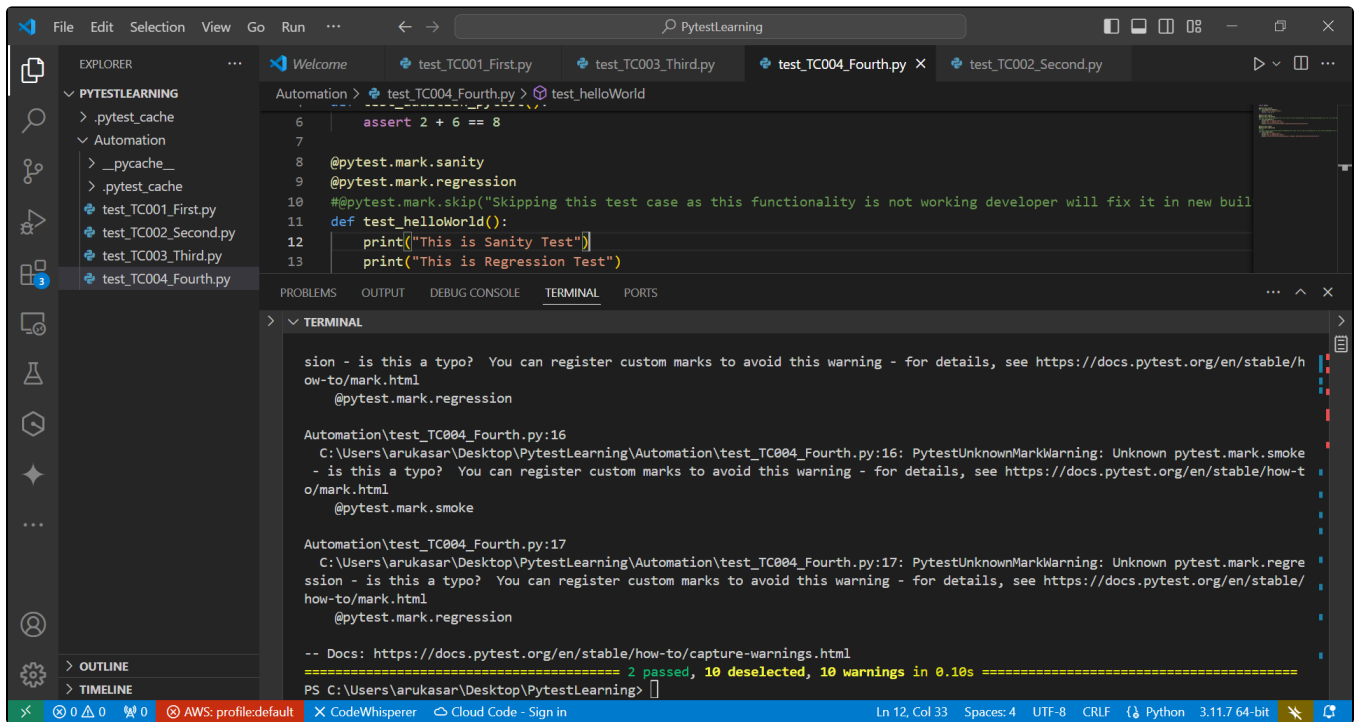
@pytest.mark.sanity
@pytest.mark.regression
#@pytest.mark.skip("Skipping this test case as this functionality is not working developer will fix it in new built")
def test_helloWorld():
    print("This is Sanity Test")
    print("This is Regression Test")
    print("*****Hello PyTest*****")

@pytest.mark.smoke
@pytest.mark.regression
# a=101
# @pytest.mark.skipif(a>30,reason="Skipping this test case as this functionality is not working developer will fix it in new built")
def test_condition():
    print("This is Smoke Test")
    print("This is Regression Test")
    print("*****Executing if greater than 30*****")
```

**Executing Test Cases Using Multiple Tags (AND Logic):** To execute test cases using multiple tags (AND logic), combine the tag names with `and` within quotes.

#### tag/grouping command

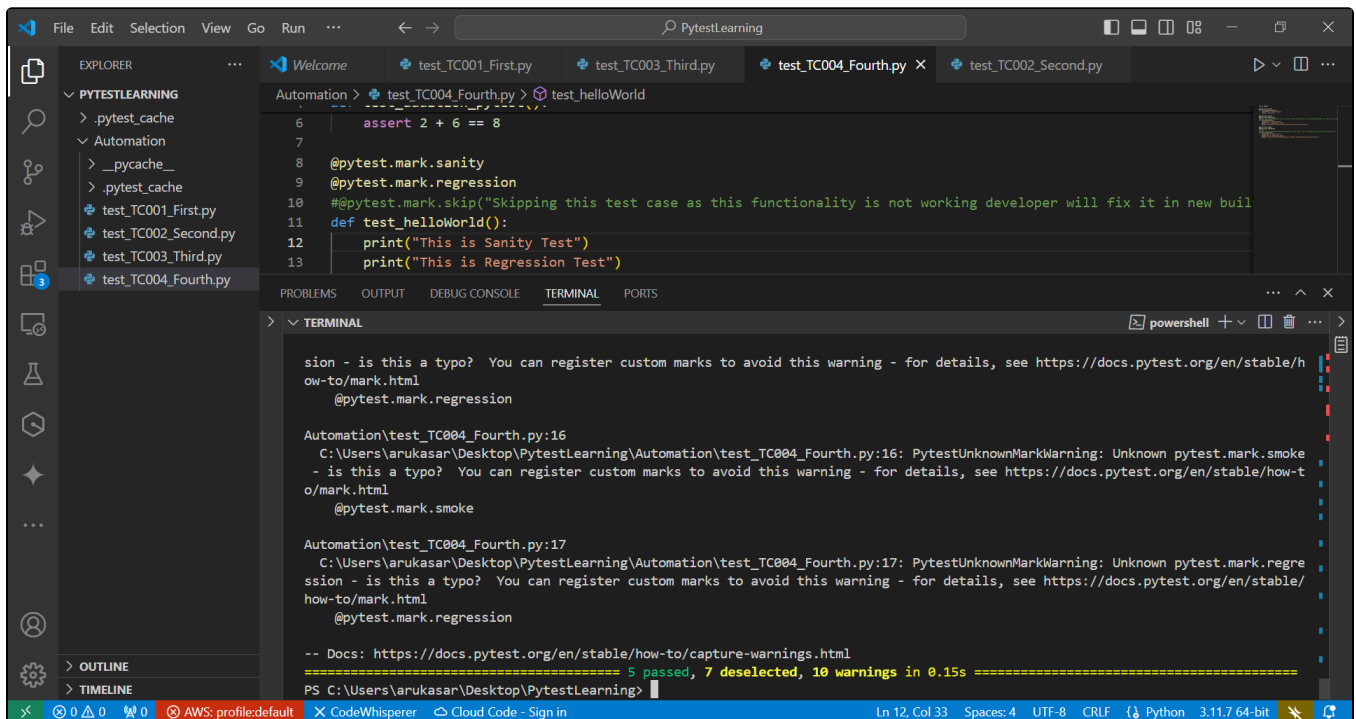
```
pytest -m "smoke and regression" <folder_name>
```



OR: You can also use `or` within quotes for executing test cases that have either of the specified tags.

#### tag/grouping command

```
pytest -m "smoke and regression" <folder_name>
```



## Handle Custom Tagging issues

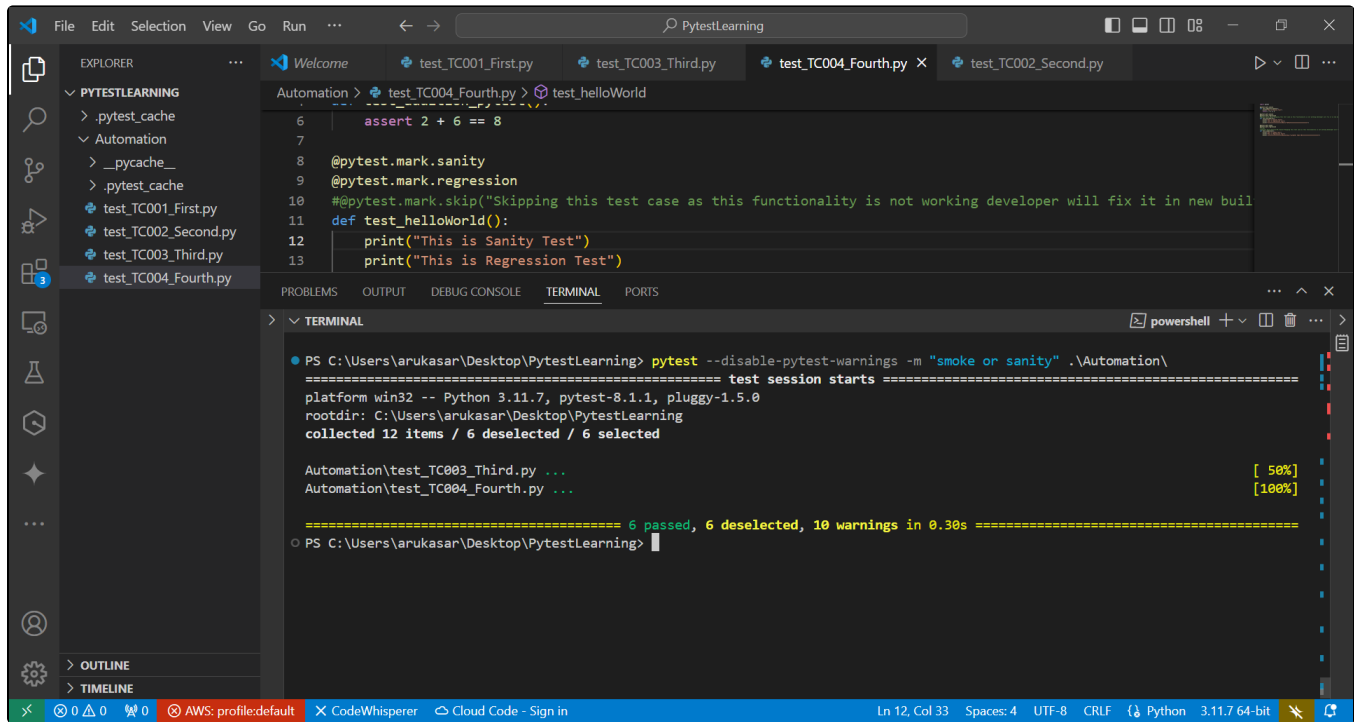
Approach 1: `--disable-pytest-warnings`

Approach 2: register the custom marker

#### Approach 1:

##### tag/grouping command

```
pytest --disable-pytest-warnings -m "smoke or sanity" <folder_name>
```



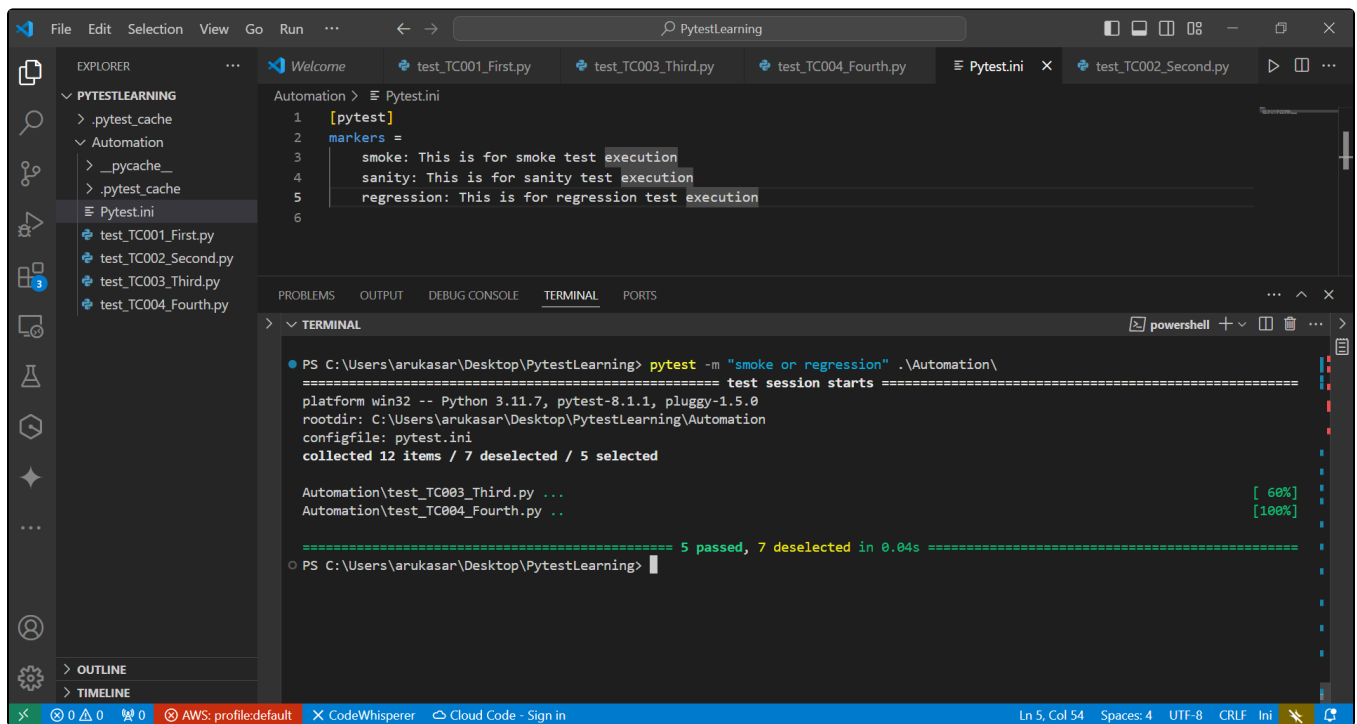
The screenshot shows a VS Code editor with a file named `test_helloWorld` in the `Automation` directory. The code defines a test function `test_helloWorld` with markers `@pytest.mark.sanity` and `@pytest.mark.regression`. A comment indicates that a test case is being skipped because the functionality is not working. The terminal window shows the command `pytest --disable-pytest-warnings -m "smoke or sanity" .\Automation\` being executed. The output shows that 6 items were collected, 6 were deselected, and 10 warnings were issued in 0.30 seconds. The test session was successful.

#### Approach 2:

1. Create a new file named `pytest.ini` in your project directory.
2. Open the `pytest.ini` file using a text editor.
3. Add the following content to the file:
4. Save the changes.

##### Pytest.ini

```
[pytest]
markers =
    smoke: This is for smoke test execution
    sanity: This is for sanity test execution
    regression: This is for regression test execution
```



## Assertions

*Compare actual results with expected results*

- Compare data to be the same
- Compare data NOT to be the same
- Compare data and display customized messages in case of failure

**Compare data to be the same:**

### test\_TC005\_assert.py

```

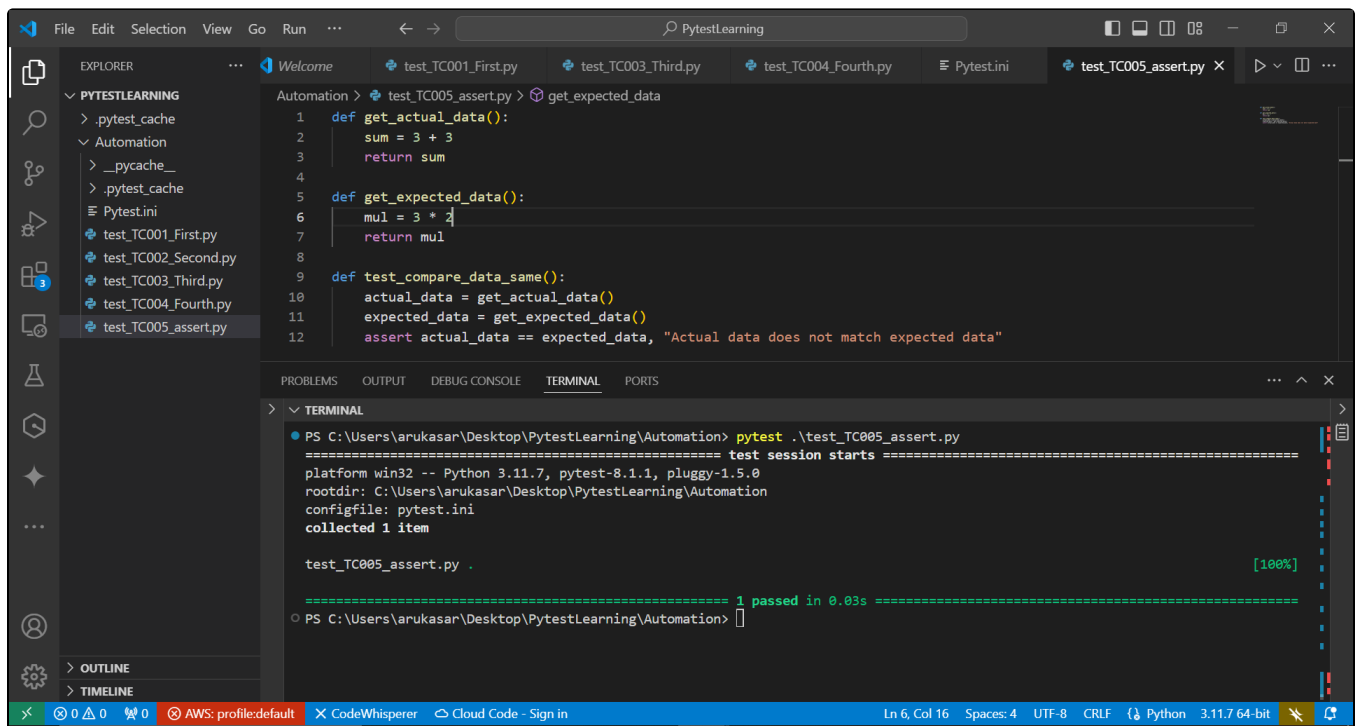
def get_actual_data():
    sum = 3 + 3
    return sum

def get_expected_data():
    mul = 3 * 2
    return mul

def test_compare_data_same():
    actual_data = get_actual_data()
    expected_data = get_expected_data()
    assert actual_data == expected_data, "Actual data does not match expected data"

```





Compare data NOT to be the same:

#### test\_TC005\_assert.py

```
def get_actual_data():
    sum = 3 + 3
    return sum

def get_unexpected_data():
    mul = 3 * 8
    return mul

def test_compare_data_not_same():
    actual_data = get_actual_data()
    unexpected_data = get_unexpected_data()
    assert actual_data != unexpected_data, "Actual data matches unexpected data"
```

Failure Result:

```
File Edit Selection View Go Run ... PytestLearning
EXPLORER
PYTESTLEARNING
  .pytest_cache
  Automation
    __pycache__
    .pytest_cache
    Pytest.ini
    test_TC001_First.py
    test_TC002_Second.py
    test_TC003_Third.py
    test_TC004_Fourth.py
    test_TC005_assert.py
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Automation > test_TC005_assert.py > get_unexpected_data
5 def get_actual_data():
6     sum = 3 + 3
7     return sum
8
9 def get_unexpected_data():
10    mul = 3 * 2
11    return mul
12
13 def test_compare_data_not_same():
14     actual_data = get_actual_data()
15     unexpected_data = get_unexpected_data()
16     assert actual_data != unexpected_data, "Actual data matches unexpected data"
17
TERMINAL
===== FAILURES =====
test_compare_data_not_same

def test_compare_data_not_same():
    actual_data = get_actual_data()
    unexpected_data = get_unexpected_data()
> assert actual_data != unexpected_data, "Actual data matches unexpected data"
E       AssertionError: Actual data matches unexpected data
E       assert 3 != 6

test_TC005_assert.py:16: AssertionError
===== short test summary info =====
FAILED test_TC005_assert.py::test_compare_data_not_same - AssertionError: Actual data matches unexpected data
===== 1 failed in 0.58s =====
PS C:\Users\arukasar\Desktop\PytestLearning\Automation>
```

Positive Result:

```
File Edit Selection View Go Run ... PytestLearning
EXPLORER
PYTESTLEARNING
  .pytest_cache
  Automation
    __pycache__
    .pytest_cache
    Pytest.ini
    test_TC001_First.py
    test_TC002_Second.py
    test_TC003_Third.py
    test_TC004_Fourth.py
    test_TC005_assert.py
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Automation > test_TC005_assert.py > get_unexpected_data
5 def get_actual_data():
6     sum = 3 + 3
7     return sum
8
9 def get_unexpected_data():
10    mul = 3 * 8
11    return mul
12
13 def test_compare_data_not_same():
14     actual_data = get_actual_data()
15     unexpected_data = get_unexpected_data()
16     assert actual_data != unexpected_data, "Actual data matches unexpected data"
17

TERMINAL
PS C:\Users\arukasar\Desktop\PytestLearning\Automation> pytest .\test_TC005_assert.py
===== test session starts =====
platform win32 -- Python 3.11.7, pytest-8.1.1, pluggy-1.5.0
rootdir: C:\Users\arukasar\Desktop\PytestLearning\Automation
configfile: pytest.ini
collected 1 item

test_TC005_assert.py . [100%]

===== 1 passed in 0.16s =====
PS C:\Users\arukasar\Desktop\PytestLearning\Automation>
```

Compare data and display a customized message in case of failure:

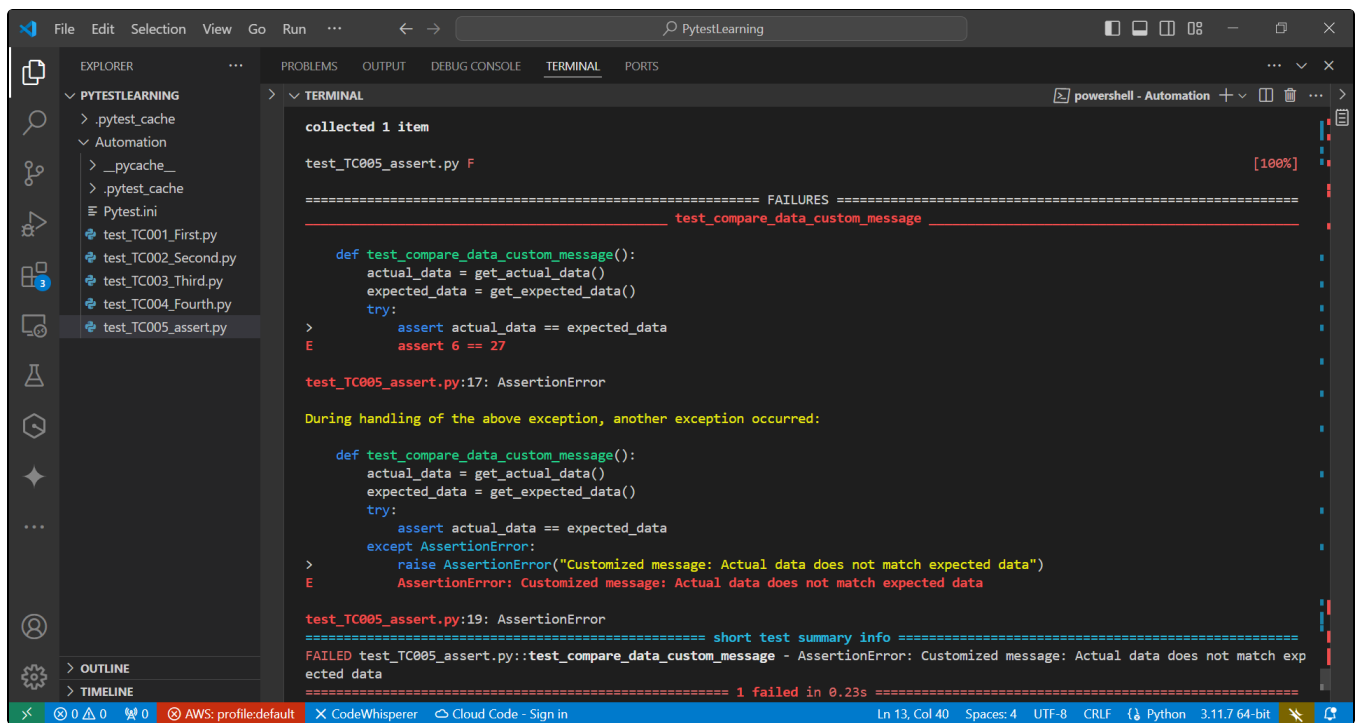
### test\_TC005\_assert.py

```
def get_expected_data():
    mul = 3 * 9
    return mul

def get_actual_data():
    sum = 3 + 3
    return sum

def test_compare_data_custom_message():
    actual_data = get_actual_data()
    expected_data = get_expected_data()
    try:
        assert actual_data == expected_data
    except AssertionError:
        raise AssertionError("Customized message: Actual data does not match expected data")
```

### Failure Result:



```
collected 1 item

test_TC005_assert.py F [100%]

===== FAILURES =====
test_compare_data_custom_message

def test_compare_data_custom_message():
    actual_data = get_actual_data()
    expected_data = get_expected_data()
    try:
        assert actual_data == expected_data
    except AssertionError:
        raise AssertionError("Customized message: Actual data does not match expected data")
E       AssertionError: Customized message: Actual data does not match expected data

test_TC005_assert.py:19: AssertionError

During handling of the above exception, another exception occurred:

def test_compare_data_custom_message():
    actual_data = get_actual_data()
    expected_data = get_expected_data()
    try:
        assert actual_data == expected_data
    except AssertionError:
        raise AssertionError("Customized message: Actual data does not match expected data")
E       AssertionError: Customized message: Actual data does not match expected data

test_TC005_assert.py:19: AssertionError

===== short test summary info =====
FAILED test_TC005_assert.py::test_compare_data_custom_message - AssertionError: Customized message: Actual data does not match expected data

===== 1 failed in 0.23s =====
```

## PyTest Fixtures

1. Execute something before the test case
2. Execute after test cases
3. Execute only one

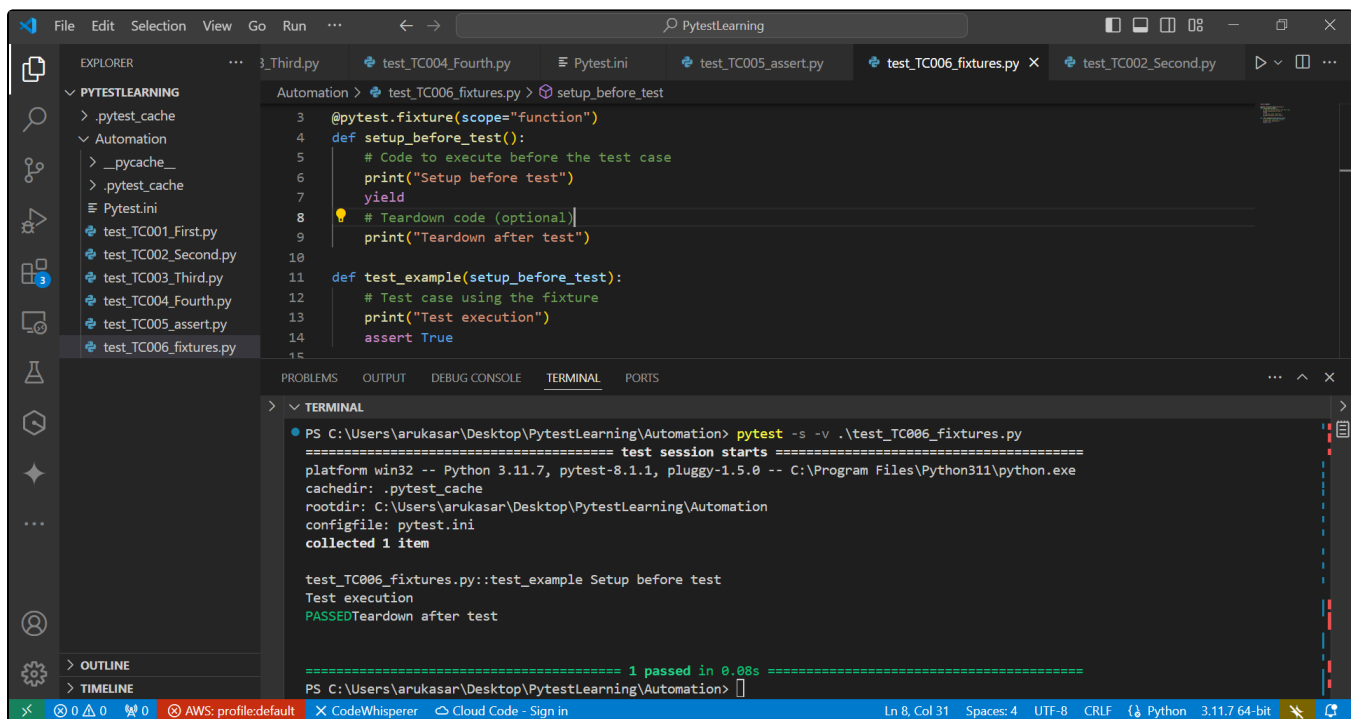
Execute something before the test case:

## test\_TC006\_fixtures.py

```
import pytest

@pytest.fixture(scope="function")
def setup_before_test():
    # Code to execute before the test case
    print("Setup before test")
    yield
    # Teardown code (optional)
    print("Teardown after test")

def test_example(setup_before_test):
    # Test case using the fixture
    print("Test execution")
    assert True
```



The screenshot shows a Visual Studio Code editor window with the file `test_TC006_fixtures.py` open. The file contains a pytest fixture `setup_before_test` and a test function `test_example`. The terminal output shows the command `pytest -s -v .\test_TC006_fixtures.py` being executed, resulting in a successful test run with the output: `test session starts platform win32 -- Python 3.11.7, pytest-8.1.1, pluggy-1.5.0 -- C:\Program Files\Python311\python.exe cachedir: .pytest_cache rootdir: C:\Users\arukasar\Desktop\PytestLearning\Automation configfile: pytest.ini collected 1 item test_TC006_fixtures.py::test_example Setup before test Test execution PASSED Teardown after test 1 passed in 0.08s`.

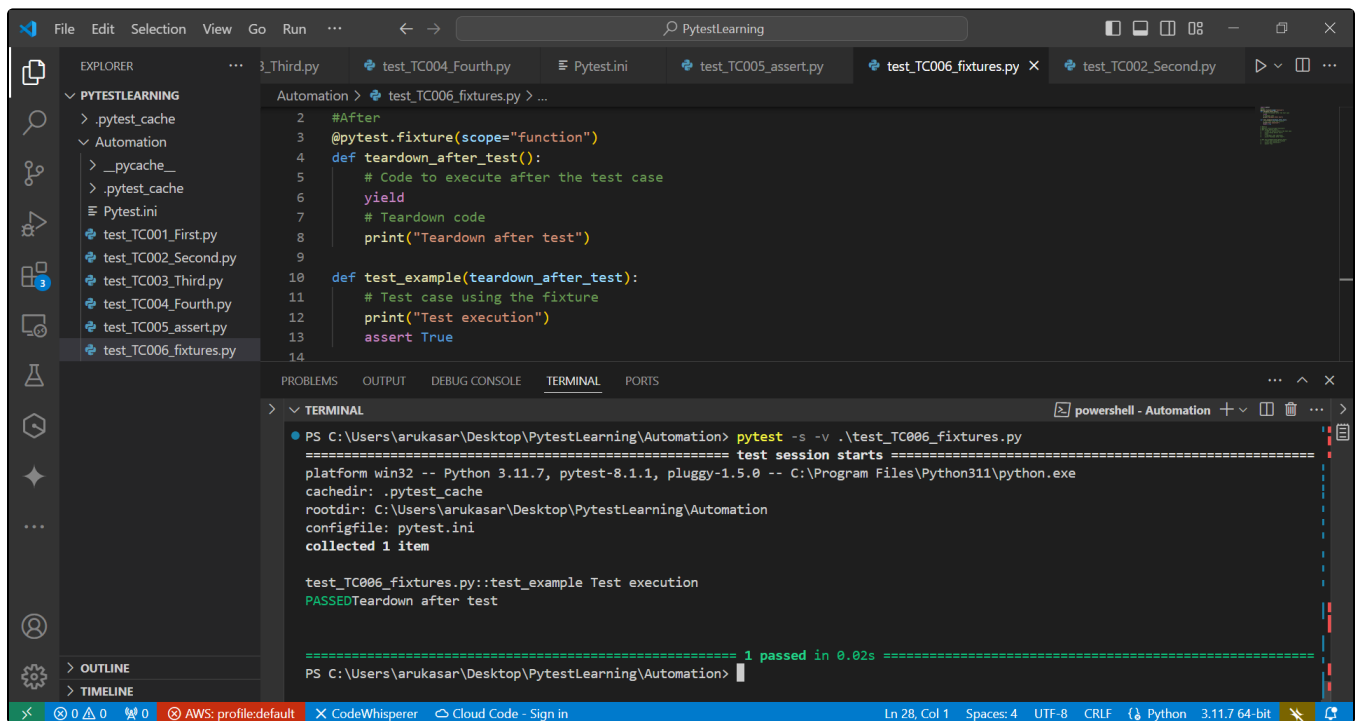
Execute after test cases:

## test\_TC006\_fixtures.py

```
import pytest

@pytest.fixture(scope="function")
def teardown_after_test():
    # Code to execute after the test case
    yield
    # Teardown code
    print("Teardown after test")

def test_example(teardown_after_test):
    # Test case using the fixture
    print("Test execution")
    assert True
```



Execute only one:

#### test\_TC006\_fixtures.py

```
import pytest

@pytest.fixture(scope="session", autouse=True)
def execute_only_once():
    # Code to execute only once for the entire test session
    print("Executed only once")

def test_example():
    # Test case
    print("Test execution")
    assert True
```

The screenshot shows a Visual Studio Code editor window with the file explorer on the left displaying a project named 'PYTESTLEARNING'. The file 'test\_TC006\_fixtures.py' is selected. The editor displays the following Python code:

```
1 import pytest
2
3 @pytest.fixture(scope="session", autouse=True)
4 def execute_only_once():
5     # Code to execute only once for the entire test session
6     print("Executed only once")
7
8 def test_example():
9     # Test case
10    print("Test execution")
11    assert True
12
13
```

The terminal at the bottom shows the command `pytest -s -v .\test_TC006_fixtures.py` being executed. The output indicates that the test session starts, the root directory is `C:\Users\arukasar\Desktop\PytestLearning\Automation`, and the test `test_TC006_fixtures.py::test_example` is executed, resulting in a `PASSED` status.

Attachment for reference:

