



Python PyTest

BEGINNER'S GUIDE

PART 1

A R U N A R U N I S T O

Pytest is a testing framework of python. It's mainly used to write API test cases.

Today's world pytest is mainly used for API testing even though we can use pytest to write simple to complex tests, i.e., we can write codes to test API, database, UI, etc.

Advantages of Pytest

1. Pytest can multiple tests in parallel, which reduces the execution time of the test suite.
2. Pytest has it's own way to detect the test file and test functions automatically if not mentioned explicitly
3. Pytest allows up to skip a subset of the tests during execution
4. Pytest allows us to run a subset of the entire test suite
5. Pytest is free and open source.
6. Pytest is very easy to start with.

Installing **pytest** using pip

pip install pytest

Running `pytest` without mentioning a filename will run all files of format **`test_*.py`** or **`*_test.py`** in the current directory and subdirectories.

Pytest automatically identifies those files as test files.

Pytest requires the test function names start with **`test`**.

Now we will start our testing program using `pytest`. first i am going to create a file named '**`calculator.py`**' and going to create a class **`Calc`** with four methods **`add`**, **`sub`**, **`mul`**, and **`div`** inside in it.

```
class Calc:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def add(self):
        return self.a+self.b
    def sub(self):
        return self.a-self.b
    def mul(self):
        return self.a*self.b
    def div(self):
        return self.a/self.b
```

In the above file you can see that I created a class named **Calc** and initialize two values in it **a, b**. And following four methods add, sub, mul, and div
Now we're going to create a test file names test_calculator.py in the current directory.

```
from .calculator import Calc
```

```
class TestCalc:
```

```
    calc = Calc(12, 10)
```

```
    def test_add(self):
```

```
        assert self.calc.add() == 22, "Add test Failed"
```

```
    def test_sub(self):
```

```
        assert self.calc.sub() == 2, "Sub test failed"
```

```
    def test_mul(self):
```

```
        assert self.calc.mul() == 120, "Mul test failed"
```

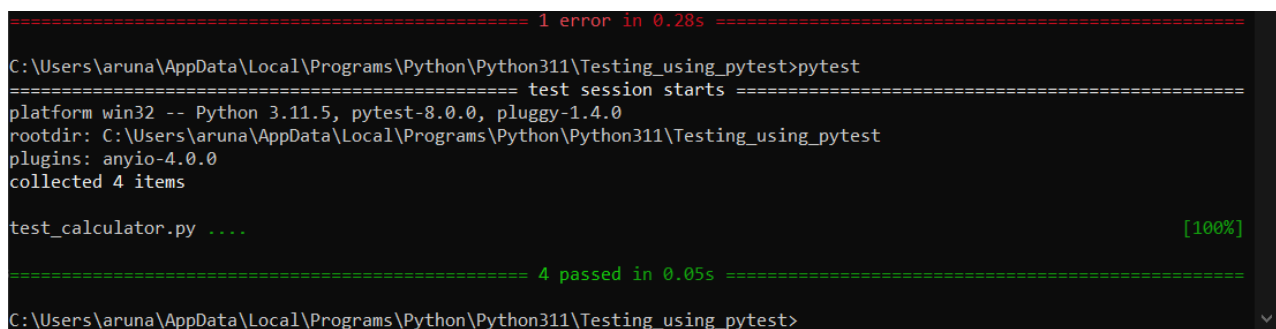
```
    def test_div(self):
```

```
        assert self.calc.div() == 1.2, "Div test failed"
```

In the above file first we imported the **Calc** from **calculator**, then we create a class **TestCalc** and add **calc** object of **Calc** class as an argument of the class. Then we add four test methods name starts with **test**, and add **assert** method on it.

To run the test you dont need to run any file names, you can simply run the script by typing "**pytest**" on your command prompt.

>>> pytest



```
===== 1 error in 0.28s =====
C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>pytest
===== test session starts =====
platform win32 -- Python 3.11.5, pytest-8.0.0, pluggy-1.4.0
rootdir: C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest
plugins: anyio-4.0.0
collected 4 items

test_calculator.py .... [100%]

===== 4 passed in 0.05s =====
C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>
```

The result will look like above, what if any test will fail, if any error occurs it will look like below. for that i'm going to change the value **2** in **test_sub** to **4** so we can check how the test framework works.

```
===== FAILURES =====
TestCalc.test_sub

self = <Testing_using_pytest.test_calculator.TestCalc object at 0x000002AD1A9CB490>

    def test_sub(self):
>     assert self.calc.sub() == 4, "Sub test failed"
E       AssertionError: Sub test failed
E       assert 2 == 4
E         + where 2 = <bound method Calc.sub of <Testing_using_pytest.calculator.Calc object at 0x000002AD196A8910>>()
E         + where <bound method Calc.sub of <Testing_using_pytest.calculator.Calc object at 0x000002AD196A8910>> = <Testing_using_pytest.calculator.Calc object at 0x000002AD196A8910>.sub
E         + where <Testing_using_pytest.calculator.Calc object at 0x000002AD196A8910> = <Testing_using_pytest.test_calculator.TestCalc object at 0x000002AD1A9CB490>.calc

test_calculator.py:10: AssertionError
===== short test summary info =====
FAILED test_calculator.py::TestCalc::test_sub - AssertionError: Sub test failed
===== 1 failed, 3 passed in 0.38s =====

C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>
```

And also you can use "**pytest -v**" command also it will increase the verbosity. The result will be more explanatory about the test that failed and that test that passed.

>>> pytest -v

```
Select Command Prompt

C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>pytest -v
===== test session starts =====
platform win32 -- Python 3.11.5, pytest-8.0.0, pluggy-1.4.0 -- C:\Users\aruna\AppData\Local\Programs\Python\Python311\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest
plugins: anyio-4.0.0
collected 4 items

test_calculator.py::TestCalc::test_add PASSED [ 25%]
test_calculator.py::TestCalc::test_sub PASSED [ 50%]
test_calculator.py::TestCalc::test_mul PASSED [ 75%]
test_calculator.py::TestCalc::test_div PASSED [100%]

===== 4 passed in 0.06s =====

C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>
```

Next we're going to check how to run two test files simultaneously for that we're going to create two files one for execute and other one for testing

First, we're going to create a file called **string_operations.py**

```
class str_operations:
    def __init__(self, str):
        self.str = str

    #for reversing a string
    def reverse_str(self):
        return self.str[::-1]
    #upper case
    def upper_str(self):
        return self.str.upper()
    #lower case
    def lower_str(self):
        return self.str.lower()
```

Next, we're going to create a test file for testing the string operations.

```
from .string_operators import str_operations
```

```
class TestStringOperations:
```

```
    str_op = str_operations("Arunisto")
```

```
    msg = "Test Failed"
```

```
    reverse_str = str_op.reverse_str()
```

```
    upper_str = str_op.upper_str()
```

```
    lower_str = str_op.lower_str()
```

```
    def test_reverse_str(self):
```

```
        assert self.reverse_str == "otsinurA", self.msg
```

```
    def test_upper_str(self):
```

```
        assert self.upper_str == "ARUNISTO", self.msg
```

```
    def test_lower_str(self):
```

```
        assert self.lower_str == "arunisto", self.msg
```

Now we created our test file also first importing the string_operators and all the operations initialized as test class argument.

After running the **pytest -v** command the result will be like the following:-


```
Command Prompt

C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>pytest -v
===== test session starts =====
platform win32 -- Python 3.11.5, pytest-8.0.0, pluggy-1.4.0 -- C:\Users\aruna\AppData\Local\Programs\Python\Python311\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest
plugins: anyio-4.0.0
collected 7 items

test_calculator.py::TestCalc::test_add PASSED [ 14%]
test_calculator.py::TestCalc::test_sub PASSED [ 28%]
test_calculator.py::TestCalc::test_mul PASSED [ 42%]
test_calculator.py::TestCalc::test_div PASSED [ 57%]
test_string_ops.py::TestStringOperations::test_reverse_str PASSED [ 71%]
test_string_ops.py::TestStringOperations::test_upper_str PASSED [ 85%]
test_string_ops.py::TestStringOperations::test_lower_str PASSED [100%]

===== 7 passed in 0.08s =====

C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>
```

You can test a particular file also by using the command "**pytest <file_name> -v**"

>>> pytest test_string_ops.py -v

The result will be look like this.

```
Command Prompt

C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>pytest test_string_ops.py -v
===== test session starts =====
platform win32 -- Python 3.11.5, pytest-8.0.0, pluggy-1.4.0 -- C:\Users\aruna\AppData\Local\Programs\Python\Python311\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest
plugins: anyio-4.0.0
collected 3 items

test_string_ops.py::TestStringOperations::test_reverse_str PASSED [ 33%]
test_string_ops.py::TestStringOperations::test_upper_str PASSED [ 66%]
test_string_ops.py::TestStringOperations::test_lower_str PASSED [100%]

===== 3 passed in 0.06s =====

C:\Users\aruna\AppData\Local\Programs\Python\Python311\Testing_using_pytest>
```

Find the code used in the guide [here](#)