

Quick Guide: How to Use Pytest for Unit Testing

It's not only about writing code in today's tech-driven world. It is all about writing code that is efficient, scalable, and testable. This is where tools like Pytest come in handy, especially when we need to calculate the areas of various forms. If any of these functions fails, we may arrive at incorrect geometrical conclusions!

Getting Started with the Repository

To follow along with this article and get hands-on experience with the provided code, you'll need to clone the repository I've set up specifically for this purpose. Here's how you can do it:

Install Git (If Not Already Installed)

If you don't have Git installed on your machine, you'll first need to install it. Depending on your operating system, you can find the appropriate installation instructions [here](#).

Open Your Terminal or Command Prompt

Navigate to the directory where you want to store the cloned repository.

Clone the Repository

Copy and paste the following command into your terminal or command prompt and hit enter:

```
git clone https://github.com/Pytest-with-Eric/pytest-technical-task.git
```

Navigate to the Cloned Directory

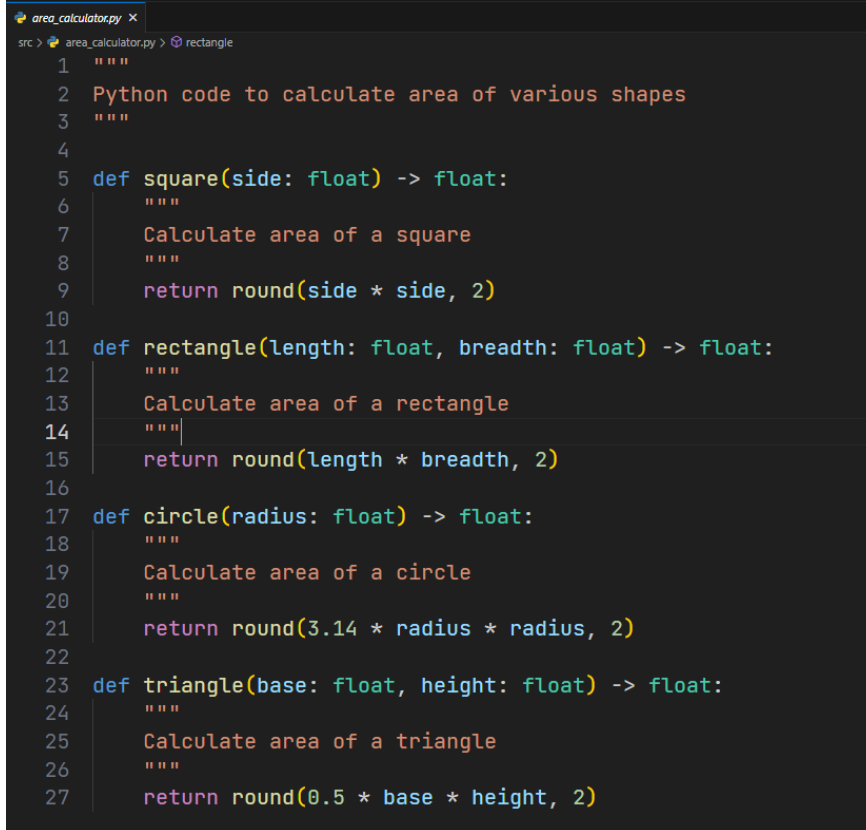
Once the cloning process is complete, move into the newly created directory with the following command:

```
cd pytest-technical-task
```

You now have all the necessary code from the repository on your local machine and are ready to follow along!

Understanding the Area Calculator

Before diving into testing, let's look at our source code. We have a simple Python script **area_calculator.py** located at src/. This script comprises functions to calculate the areas of various geometrical shapes like squares, rectangles, circles, and triangles.



```
1  """
2  Python code to calculate area of various shapes
3  """
4
5  def square(side: float) -> float:
6      """
7      Calculate area of a square
8      """
9      return round(side * side, 2)
10
11 def rectangle(length: float, breadth: float) -> float:
12     """
13     Calculate area of a rectangle
14     """
15     return round(length * breadth, 2)
16
17 def circle(radius: float) -> float:
18     """
19     Calculate area of a circle
20     """
21     return round(3.14 * radius * radius, 2)
22
23 def triangle(base: float, height: float) -> float:
24     """
25     Calculate area of a triangle
26     """
27     return round(0.5 * base * height, 2)
```

The formulae used are:

Square: side^2

Rectangle: $\text{length} \times \text{breadth}$

Circle: $\pi \times \text{radius}^2$

Triangle: $0.5 \times \text{base} \times \text{height}$

Setting Up the Testing Environment

Prerequisites

Python (3.11+)

To ensure consistency and to avoid dependency issues, we need to:

Create a virtual environment and activate it.

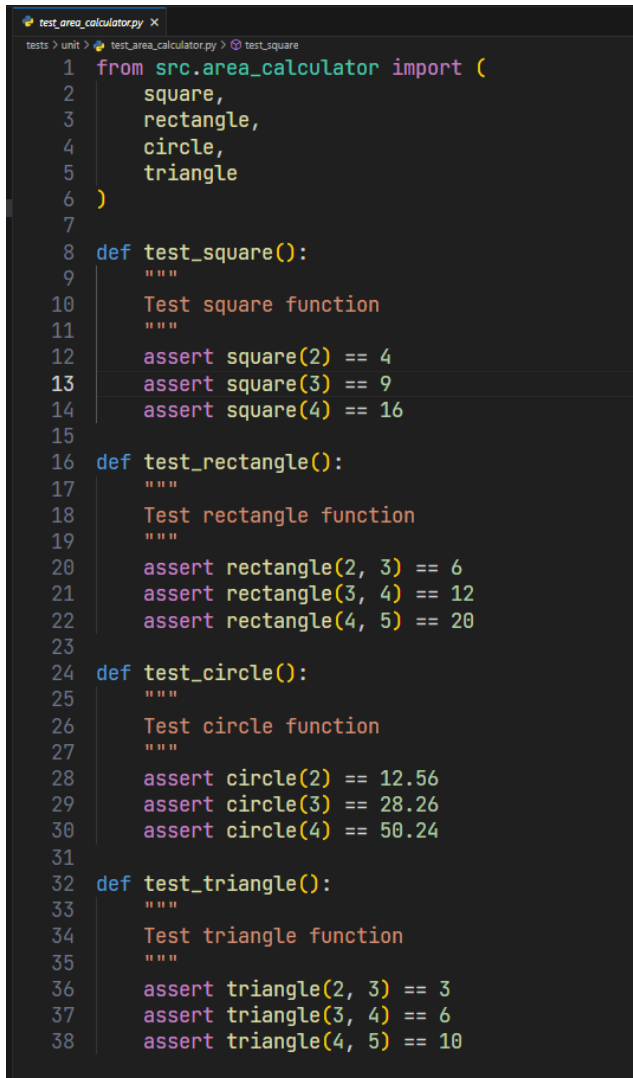
Install the required dependencies using the requirements.txt file:

```
pip install -r requirements.txt
```

If you don't have Pip installed, you'll need to do so. There are plenty of [resources online to guide you through it](#).

Running the Unit Tests

Now, to ensure our area calculator functions as intended, we have created unit tests located at ***tests/unit/test_area_calculator.py***.

A screenshot of a code editor window titled 'test_area_calculator.py'. The editor shows a Python file with unit tests for an area calculator. The code is as follows:

```
1 from src.area_calculator import (
2     square,
3     rectangle,
4     circle,
5     triangle
6 )
7
8 def test_square():
9     """
10    Test square function
11    """
12    assert square(2) == 4
13    assert square(3) == 9
14    assert square(4) == 16
15
16 def test_rectangle():
17     """
18    Test rectangle function
19    """
20    assert rectangle(2, 3) == 6
21    assert rectangle(3, 4) == 12
22    assert rectangle(4, 5) == 20
23
24 def test_circle():
25     """
26    Test circle function
27    """
28    assert circle(2) == 12.56
29    assert circle(3) == 28.26
30    assert circle(4) == 50.24
31
32 def test_triangle():
33     """
34    Test triangle function
35    """
36    assert triangle(2, 3) == 3
37    assert triangle(3, 4) == 6
38    assert triangle(4, 5) == 10
```

To run these tests, from the root of the repo, execute:

pytest

```
Windows PowerShell
PS C:\Users\Carl Justine Cruz\pytest\pytest-technical-task> pytest
===== test session starts =====
platform win32 -- Python 3.11.0, pytest-7.4.0, pluggy-1.3.0
rootdir: C:\Users\Carl Justine Cruz\pytest\pytest-technical-task
collected 4 items

tests\unit\test_area_calculator.py .... [100%]

===== 4 passed in 0.01s =====
PS C:\Users\Carl Justine Cruz\pytest\pytest-technical-task> |
```

For a more detailed output:

pytest -v

```
PS C:\Users\Carl Justine Cruz\pytest\pytest-technical-task> pytest -v
===== test session starts =====
platform win32 -- Python 3.11.0, pytest-7.4.0, pluggy-1.3.0 -- C:\Users\Carl Justine Cruz\AppData\Local\Programs\Python\Python311\python.exe
cachedir: pytest_cache
rootdir: C:\Users\Carl Justine Cruz\pytest\pytest-technical-task
collected 4 items

tests\unit\test_area_calculator.py::test_square PASSED [ 25%]
tests\unit\test_area_calculator.py::test_rectangle PASSED [ 50%]
tests\unit\test_area_calculator.py::test_circle PASSED [ 75%]
tests\unit\test_area_calculator.py::test_triangle PASSED [100%]

===== 4 passed in 0.01s =====
PS C:\Users\Carl Justine Cruz\pytest\pytest-technical-task> |
```

Analyzing the Tests

The unit tests are simple yet effective. Here's what they're doing:

Testing the Square Function: We're checking if the function can calculate the area of squares with side lengths of 2, 3, and 4 units.

Testing the Rectangle Function: Various rectangles with differing lengths and breadths are tested.

Testing the Circle Function: We're verifying the area of circles with radii 2, 3, and 4 units.

Testing the Triangle Function: Triangles with different bases and heights are put to test.

In each of these tests, we use the assert statement to compare the output of our functions to the expected output.

The Role of Pytest

Unit testing assures that our routines are resilient and error-free, much as performance testing and benchmarking discover bottlenecks, inefficiencies, and regressions. Any difference in programming as fundamental as calculating areas might have a cascade effect. Pytest automates, structures, and streamlines unit testing.

In today's technological context, the efficiency and integrity of our code are critical to ensuring that our applications work properly. By utilizing Pytest, we are not only validating the accuracy of our code, but also preparing it for real-world applications.

Conclusion

In our journey through the realm of code testing using Pytest, we've seen how to effectively set up our testing environment, run tests, and understand their importance. Remember, it's not about writing more code, but about writing more effective code. And tools like Pytest are your trusty companions in this journey towards perfection.