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***](https://github.com/Pytest-with-Eric/pytest-technical-task.git)

Navigate to the Cloned Directory

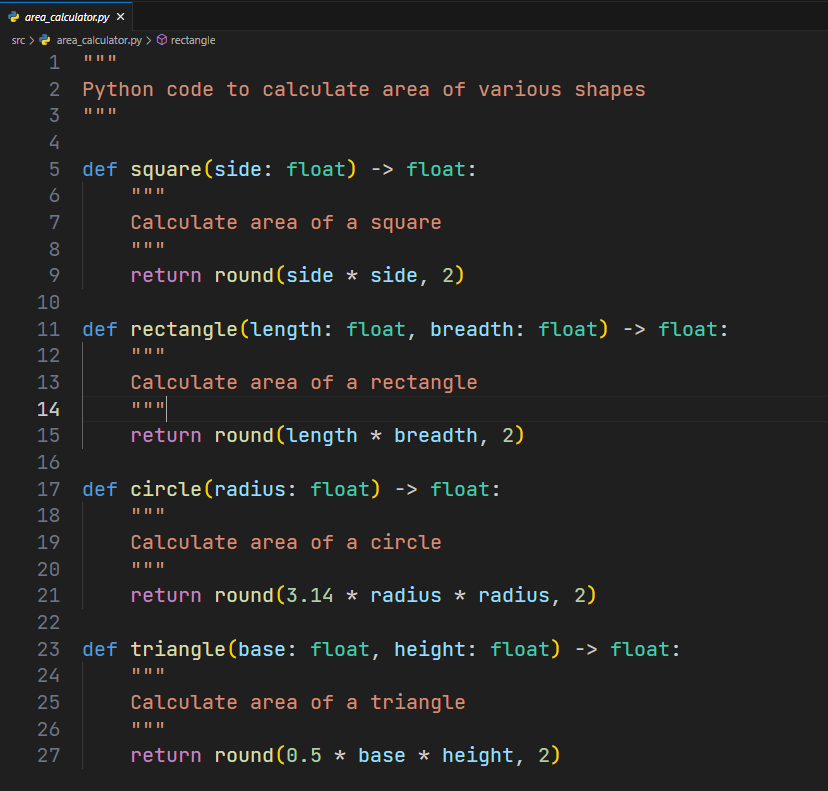
Once the cloning process is complete, move into the newly created directory with the following command:

***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.



The formulae used are:

**Square: side^2**

**Rectangle: length x breadth**

**Circle: π x radius^2**

**Triangle: 0.5 x base x 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:

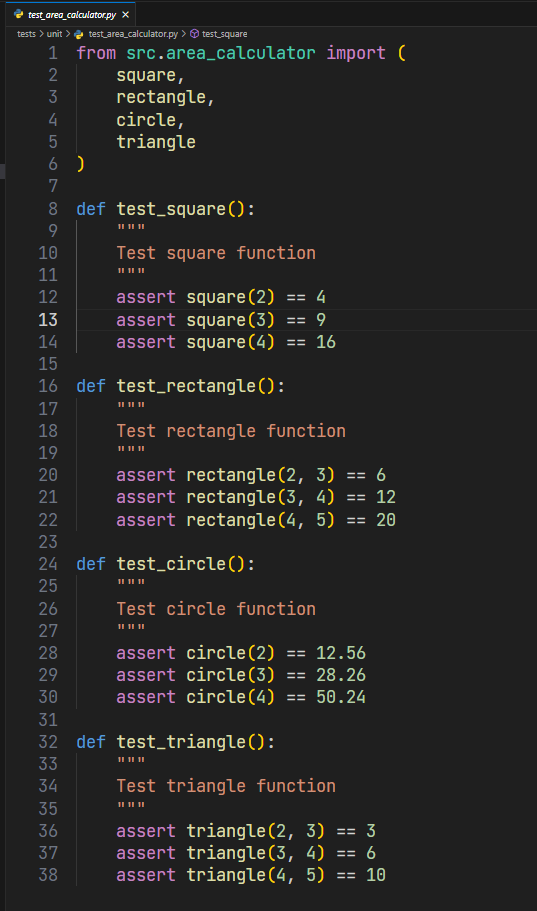
pip install -r requirements.txt

If you don’t have Pip installed, you’ll need to do so. There are plenty of [resources online to guide you through it](https://pip.pypa.io/en/stable/installation/).

## Running the Unit Tests

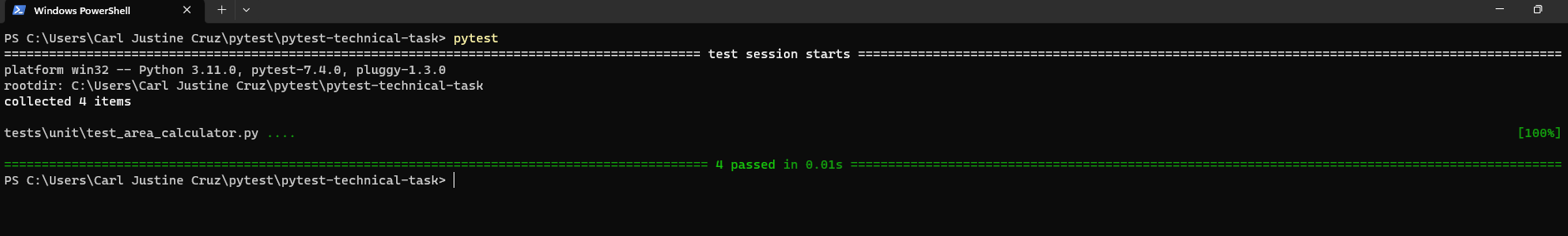
Now, to ensure our area calculator functions as intended, we have created unit tests located at

***tests/unit/test\_area\_calculator.py***.



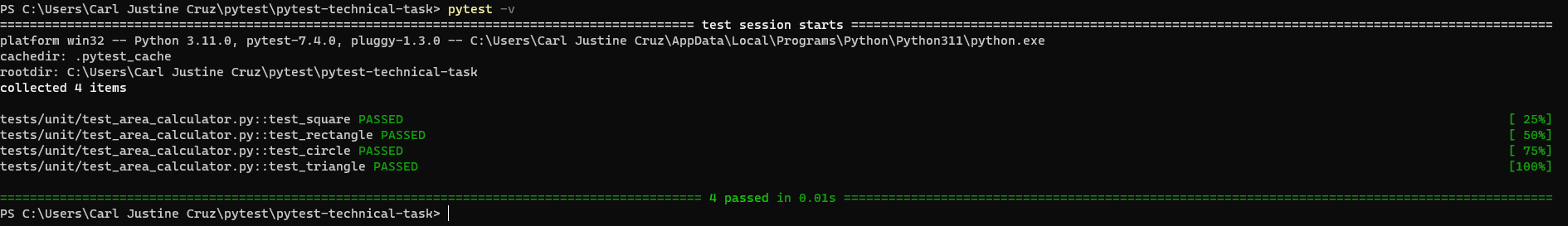
To run these tests, from the root of the repo, execute:

***pytest***



For a more detailed output:

***pytest -v***



## 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.