**Unit Testing with Python unittest Module**

By [Hafeezul Kareem Shaik](https://geekflare.com/author/hafeezulkareem/)on November 8, 2021

<https://geekflare.com/unit-testing-with-python-unittest/>

No good developer deploys code without thorough testing. Unit testing is the process of testing the individual modules of a large program.This article will discuss how you can perform unit testing of your code using the Python unittest module. First, let’s understand what the types of testing are.

When it comes to [testing](https://geekflare.com/software-testing-tools/), there is manual testing and automatic testing. **Manual testing** is a testing technique where humans perform the test manually after the completion of the development.  [**Automation testing**](https://geekflare.com/application-automation-tools/) is a testing technique where programs perform the test automatically and give the results to us.

You would have understood that manually testing is time taking and hard to perform. So, the developers write code for performing tests (automatic testing). There are different kinds of testing in automated testing. Some of them are **Unit testing, Integrating testing, End to end testing, Stress testing,**etc..,

Let’s see the standard flow of testing.

* Write or Update the code.
* Write or Update tests for different cases for your code.
* Run the tests (either manually or using a test runner).
* See the test results. If there are any errors, fix them and repeat the steps.

Here, we are going to discuss the most essential and basic testing type called **unit testing**. Without any further ado, let’s dive into the actual tutorial.

**What is Unit Testing?**

Unit testing is a technique to test a small block of independent code. The small-block code will be a function in most cases. The word independent means it doesn’t depend on other pieces of code in the project.

Let’s say we have to check whether a string is equal to **“Geekflare”** or not. For that, we have written a function that takes one argument and returns whether it’s equal to **“Geekflare”**or not.

def is\_equal\_to\_geekflare(string):

return string == "Geekflare"

The above function has no dependencies on any other code. So, we can test it independently by giving different inputs. The independent piece of code can be used across the project.

**Importance of Unit Testing**

In general, the independent blocks code can be used across the project. So, it must be well written and tested. Unit tests are the tests used to test those types of independent blocks of code. What happens if we don’t use unit tests for our project?

Let’s assume we didn’t test small blocks of code that are used across the project. All other tests like integration tests, end-to-end tests, etc., using the other small blocks of code may fail. This breaks the application. That’s why the basic building blocks of the code must be tested well.

Now, we know the importance of unit testing and written unit tests for all independent code blocks. Since we have performed unit tests, other tests like [integration](https://geekflare.com/understanding-ci-cd/) tests, end-to-end tests, etc., won’t fail because of the independent block of codes.

In the coming sections, we will see what the Python unittest module is and how we use the unittest module to write unit tests in [Python](https://geekflare.com/python-cheat-sheets/).

***Note:***We are assuming that you are familiar with Python classes, modules, etc. If you are not familiar with Python intermediate concepts like classes, modules, etc., you may find it difficult to understand the coming sections.

**What is Python unittest?**

Python unittest is a built-in testing framework to test Python code. It has a test runner, which allows us to run the tests without much effort. So, we can use the built-in **unittest**module for testing without using the third-party modules. But, it changes based on your requirement. The built-in **unittest**module is good to get started with testing in Python.

We have to follow the below steps to test the Python code using the unittest module.

**#1.** Write the code.

**#2.**Import the unittest module.

**#3.** Create a file starting with the keyword test. For example test\_prime.py. The keyword *test* is used to identify the test files.

**#4.** Create a class extending the class unittest.TestCase.

**#5.**Write methods (tests) inside the class. Each method contains different test cases based on your requirement. We must name the method starting with test keyword.

**#6.** Run the tests. We can run the tests in different ways.

* Run the command python -m unittest test\_filename.py.
* We run the test files like general Python files with the command python test\_filename.py. For this method to work, we need to invoke the main method of the unittest in the test file.
* And finally, using the discover. We can autorun the tests using the command python -m unittest discover without mentioning the filename of the test. It will find the tests using the naming convention that we followed. So, we must name our test files with the keyword test in starting.

Generally, in testing, we compare the output of the code with the expected output. So, to compare the outputs **unittest**provides different methods. You can find the list of comparing functions [here](https://docs.python.org/3/library/unittest.html#assert-methods).

You can easily understand them without any difficulty. They are straightforward.

That’s a lot of theory. We must now get into coding.

***Note:***If you have any queries about the unittest module, you can go to the [documentation](https://docs.python.org/3/library/unittest.html) and clear your doubts.  Without any further delay, let’s use the unittest module.

**Unit Tests in Python using unittest**

We will write some functions first, and then we will focus on writing the tests. First, open a folder in your favorite [code editor](https://geekflare.com/online-compiler/python). And create a file called utils.py. Paste the following code in the file.

import math

def is\_prime(n):

if n < 0:

return 'Negative numbers are not allowed'

if n <= 1:

return False

if n == 2:

return True

if n % 2 == 0:

return False

for i in range(2, int(math.sqrt(n)) + 1):

if n % i == 0:

return False

return True

def cubic(a):

return a \* a \* a

def say\_hello(name):

return "Hello, " + name

Copy

We have three different functions in the utils.py file. Now, we have to test each function with different test cases. Let’s write the tests for the first function is\_prime.

**#1.** Create a file called test\_utils.py in the sample folder as utils.py.

**#2.**Import the utils and unittest module.

**#3.** Create a class with the name TestUtils extending unittest.TestCase class. The name of the class can be anything. Try to give the class a meaningful name.

**#4.**Inside the class, write a method called test\_is\_prime which accepts self as an argument.

**#5.** Write different test cases with arguments to the is\_prime and compare the output with the expected output.

**#6.** Example test case self.assertFalse(utils.is\_prime(1)).

**#7.** We are expecting the output of the is\_prime(1) will be false in the above case.

**#8.** Similar to the above case, we will different tests cases based on the function we are testing.

Let’s see the tests.

import unittest

import utils

class TestUtils(unittest.TestCase):

def test\_is\_prime(self):

self.assertFalse(utils.is\_prime(4))

self.assertTrue(utils.is\_prime(2))

self.assertTrue(utils.is\_prime(3))

self.assertFalse(utils.is\_prime(8))

self.assertFalse(utils.is\_prime(10))

self.assertTrue(utils.is\_prime(7))

self.assertEqual(utils.is\_prime(-3),

"Negative numbers are not allowed")

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

unittest.main()

Copy

We are invoking the main method of unittest the module to run the tests using python filename.py the command. Now, run the tests.

You will see the output similar to the below output.

$ python test\_utils.py

.

----------------------------------------------------------------------

Ran 1 test in 0.001s

OK

Copy

Now, try to write the test cases for other functions as well. Think of different cases for the functions and write tests for them. Take a look at the following tests that are added to the above class.

...

class TestUtils(unittest.TestCase):

def test\_is\_prime(self):

...

def test\_cubic(self):

self.assertEqual(utils.cubic(2), 8)

self.assertEqual(utils.cubic(-2), -8)

self.assertNotEqual(utils.cubic(2), 4)

self.assertNotEqual(utils.cubic(-3), 27)

def test\_say\_hello(self):

self.assertEqual(utils.say\_hello("Geekflare"), "Hello, Geekflare")

self.assertEqual(utils.say\_hello("Chandan"), "Hello, Chandan")

self.assertNotEqual(utils.say\_hello("Chandan"), "Hi, Chandan")

self.assertNotEqual(utils.say\_hello("Hafeez"), "Hi, Hafeez")

...

Copy

We used only some of the comparing functions from the unittest module. You can find the complete list [here](https://docs.python.org/3/library/unittest.html#assert-methods).

We have learned how to write unit tests using the unittest module. Now, it’s time to see different ways to run the tests.

**How to Run Tests using unittest**

We have already seen a way to run the test cases in the above section. Let’s see the other two ways to run the tests using the unittest module.

**#1.**Using the file name and the unittest module.

In this method, we will use the unittest module and filename to run the tests. The command to run the tests is python -m unittest filename.py. In our case, the command to run the tests is python -m unittest test\_utils.py.

**#2.** Using the discover method

We will use the discover method of the unittest module to auto-detect all the test files and run them. To auto-detect the test files, we need to name them starting with the keyword test.

The command to run the tests using the discover method is python -m unittest discover. The command will detect all the files whose names start with test and execute them.

**Conclusion 👩‍💻**

Unit tests are basic testing in the programming world. There are a lot of other tests in the real world. Try to learn them once by one. I hope this tutorial helps you write basic tests in Python using the unittest module. There are third-party libraries like pytest, Robot Framework, nose, nose2, slash, etc. You can explore them based on your project requirements.