

[Learn to code — free 3,000-hour curriculum](#)

AUGUST 6, 2019 / #PYTHON

An Introduction to Unit Testing in Python



Goran Aviani

You just finished writing a piece of code and you are wondering what to do. Will you submit a pull request and have your teammates review the code? Or will you manually test the code?

You should do both of these things, but with an additional step: you need to unit test your code to make sure that the code works as intended.

Unit tests can pass or fail, and that makes them a great technique to check your code. In this tutorial, I will demonstrate how to write unit tests in Python and you'll see how easy it is to get them going in your own project.

Getting started

The best way you can understand testing is if you do it hands-on. For that purpose, in a file named `name_function.py`, I will write a simple function that takes a first and last name, and returns a full name:

Learn to code — free 3,000-hour curriculum

```
full_name = first_name + ' ' + last_name  
return full_name.title()
```

The function `formatted_name()` takes the first and the last name and combines them with a space between to form a full name. It then capitalizes the first letter of every word. To check that this code works, you need to write some code that uses this function. In `names.py` I will write some simple code that lets users enter their first and last names:

```
from name_function import formatted_name  
  
print("Please enter the first and last names or enter x to E[x]it.")  
  
while True:  
    first_name = input("Please enter the first name: ")  
    if first_name == "x":  
        print("Good bye.")  
        break  
  
    last_name = input("Please enter the last name: ")  
    if last_name == "x":  
        print("Good bye.")  
        break  
  
    result = formatted_name(first_name, last_name)  
    print("Formatted name is: " + result + ".")
```

This code imports `formatted_name()` from `name_function.py` and on running, allows the user to enter a series of first and last names and shows the formatted full names.

Learn to code — free 3,000-hour curriculum

There is a module in Python's standard library, called `unittest`, which contains tools for testing your code. Unit testing checks if all specific parts of your function's behavior are correct, which will make integrating them together with other parts much easier.

Test case is a collection of unit tests which together proves that a function works as intended, inside a full range of situations in which that function may find itself and that it's expected to handle. Test case should consider all possible kinds of input a function could receive from users, and therefore should include tests to represent each of these situations.

Passing a test

Here's a typical scenario for writing tests:

First you need to create a test file. Then import the `unittest` module, define the testing class that inherits from `unittest.TestCase`, and lastly, write a series of methods to test all the cases of your function's behavior.

There's a line by line explanation below the following code:

```
import unittest
from name_function import formatted_name

class NamesTestCase(unittest.TestCase):

    def test_first_last_name(self):
        result = formatted_name("pete", "seeger")
        self.assertEqual(result, "Pete Seeger")
```

Learn to code — free 3,000-hour curriculum

function. This class inherits from the class `unittest.TestCase`.

`NamesTestCase` contains a single method that tests one part of `formatted_name()`. You can call this method `test_first_last_name()`.

Remember that every method that starts with “`test_`” will be run automatically when you run `test_name_function.py`.

Within `test_first_last_name()` test method, you call the function you want to test and store a return value. In this example we are going to call `formatted_name()` with the arguments “`pete`” and “`seeger`”, and store the result in the resulting variable.

In the last line we will use the `assert` method. The `assert` method verifies that a result you received matches the result you expected to receive. And in this case we know that `formatted_name()` function will return full name with capitalized first letters, so we expect the result “`Pete Seeger`”. To check this, the `unittest`’s `assertEquals()` method is being used.

```
self.assertEqual(result, "Pete Seeger")
```

This line basically means: Compare the value in the resulting variable with “`Pete Seeger`” and if they are equal it’s OK, but if they are not let me know.

On running `test_name_function.py` you are expected to get a OK meaning that the test has passed.

Learn to code – [free 3,000-hour curriculum](#)

Failing a test

To show you what a failing test looks like I'm going to modify a `formatted_name()` function by including a new middle name argument.

So I'm going to rewrite the function to look like this:

```
#Generate a formatted full name including a middle name
def formatted_name(first_name, last_name, middle_name):
    full_name = first_name + ' ' + middle_name + ' ' + last_name
    return full_name.title()
```

This version of `formatted_name()` will work for people with middle names, but when you test it you will see that the function is broken for people who don't have a middle name.

So when you run the `test_name_function.py` you will get the output that looks something like this:

Learn to code — [free 3,000-hour curriculum](#)

In the output you will see information that will tell you all you need to know where the test fails:

- First item in the output is the Error telling you that at least one test in test case resulted in an error.
- Next you'll see the file and method in which the error occurred.
- After that you will see the line in which the error occurred.
- And what kind of error it is, in this case we are missing 1 argument “middle_name”.
- You will also see the number of run tests, the time needed for the tests to complete, and a textual message that represents the status of the tests with number of errors that occurred.

What to do when the test has failed

A passing test means the function is behaving according to what's expected from it. However, a failing test means there's more fun ahead of you.

I've seen couple of programmers that prefer to change the test instead of improving the code — but don't do that. Spend a little more time to fix the issue, as it will help you to better understand the code and save time in the long run.

Learn to code – free 3,000-hour curriculum

desired behavior of it. Since the idea is not to make changes to the tests, the best solution is to make middle name optional.

After we do this the idea is to make the tests pass when the first and last name are used, for example “Pete Seeger”, as well as when first, last and middle names are used, for example “Raymond Red Reddington”. So let’s modify the code of `formatted_name()` once again:

```
#Generate a formatted full name including a middle name
def formatted_name(first_name, last_name, middle_name=''):
    if len(middle_name) > 0:
        full_name = first_name + ' ' + middle_name + ' ' + last_name
    else:
        full_name = first_name + ' ' + last_name
    return full_name.title()
```

Now the function should work for names with and without the middle name.

And to make sure it still works with “Pete Seeger” run the test again:

```
Ran 1 test in 0.001s
```

```
OK
```

And this is what I intended to show you: It’s always better to make changes to your code to fit your tests than other way

[Learn to code — free 3,000-hour curriculum](#)

Adding new tests

Write a new method to the NamesTestCase class that will test for middle names:

```
import unittest
from name_function import formatted_name

class NamesTestCase(unittest.TestCase):

    def test_first_last_name(self):
        result = formatted_name("pete", "seeger")
        self.assertEqual(result, "Pete Seeger")

    def test_first_last_middle_name(self):
        result = formatted_name("raymond", "reddington", "red")
        self.assertEqual(result, "Raymond Red Reddington")
```

After you run the test, both tests should pass:

```
Ran 2 tests in 0.001s
```

```
OK
```

Bra gjort!

Well done!

You have written your tests to check if the function works using names with or without a middle name. Stay tuned for part 2 where I'll talk more about testing in Python.

Learn to code — [free 3,000-hour curriculum](#)

freeCodeCamp profile:

<https://www.freecodecamp.org/news/author/goran/> and other fun stuff I build on my GitHub page: <https://github.com/GoranAviani>



Goran Aviani

Software engineer with experience in developing, testing and maintaining software solutions. Working in agile, entrepreneurial, high paced global company.

If this article was helpful, [tweet it](#).

Learn to code for free. freeCodeCamp's open source curriculum has helped more than 40,000 people get jobs as developers.

[Get started](#)

ADVERTISEMENT

Your Grocery Plug

VEGITA

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) charity organization (United States Federal Tax Identification Number: 82-0779546)

[Forum](#)[Donate](#)

Learn to code — free 3,000-hour curriculum

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can [make a tax-deductible donation here](#).

Trending Guides

JavaScript forEach()	HTML Font Size
strftime() in Python	Python Lowercase
Check for Null in JS	Run Python Script
Command Line Tutorial	indexOf in Python
Check Empty String JS	KeyError in Python
Make an Arrow in HTML	K8s vs Docker Swarm
Make a List in Python	Multiline String JS
How to Read JSON in JS	Concatenate in Excel
Python Print Same Line	Python List.append()
Get Current Date in JS	Population vs Sample
Reverse a String in JS	With Statement in Python
List Index Out of Range	Loop Through Array in JS
Callbacks in JavaScript	Check if Undefined in JS
Insert into Array in JS	Sort Alphabetically in JS
JavaScript Random Number	Python Dictionary Methods

Our Charity

[About](#) [Alumni Network](#) [Open Source](#) [Shop](#) [Support](#) [Sponsors](#) [Academic Honesty](#)
[Code of Conduct](#) [Privacy Policy](#) [Terms of Service](#) [Copyright Policy](#)