Python Unit Test - Introduction

# Activity 1 (guided exercise)

## Create a function that format a name

For that purpose, in a file named, I will write a simple function that takes a first and last name, and returns a full name and save it in a file named **name\_function.py**:

#Generate a formatted full name

#use file name as name\_function.py

# save it in a folder

def formatted\_name(first\_name, last\_name):

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 as a module, as explained in previous lessons. In **names.py** I will write some simple code that lets users enter their first and last names:

#this function uses name\_function.py

#Save this function as names.py in the same folder as the file name\_function,py

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":

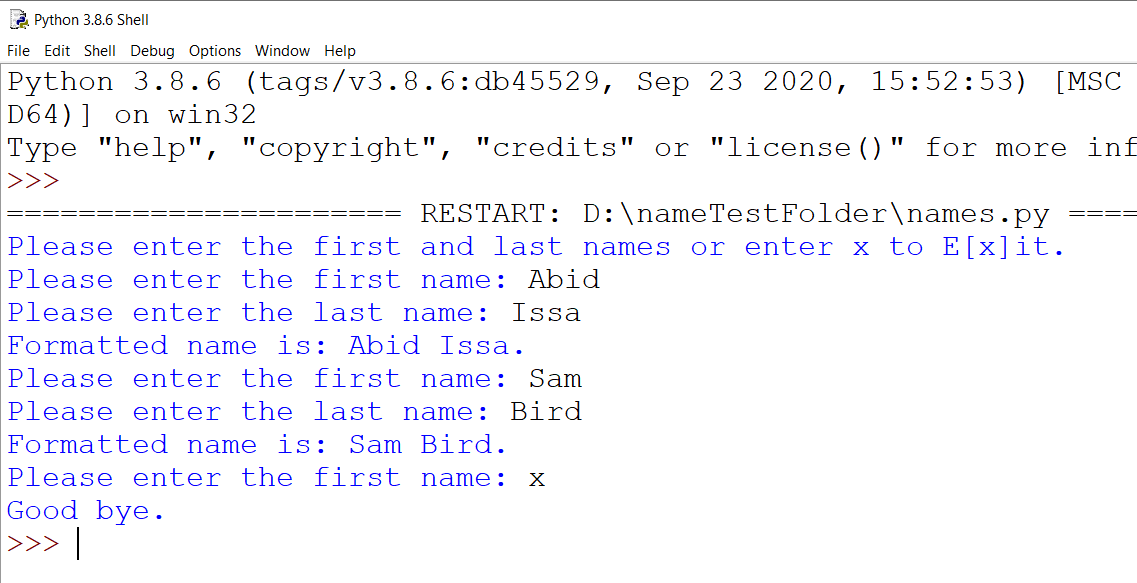
print("Good bye.")

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.to make sure you have done all the work correctly at this stage, run names.py (the second file) you should get the following:



## Create a test File using Unit Test and Test Case

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 behaviour 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** which should:

* 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 behaviour.

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

# save this file as **test\_name\_function.py**

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")

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

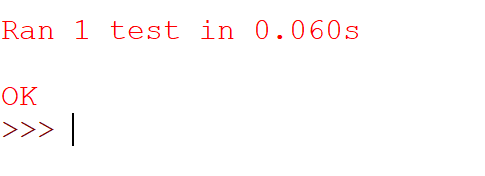
**unittest.main()**

the above code is explained in the table below:

|  |  |
| --- | --- |
| **Line** | **Meaning** |
| import unittest | Import unittest from the standard library |
| from name\_function import formatted\_name | Import the file you want to test in this case it is name\_function. From that file you will use a function which is named formatted\_name (read the code of the file name\_function.py) |
| class NamesTestCase(unittest.TestCase):  def test\_first\_last\_name(self):  result = formatted\_name("pete", "seeger")  self.assertEqual(result, "Pete Seeger") | Create a class that contains single method that tests one part of formatted\_names()  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 capitalised first letters, so we expect the result “Pete Seeger”. To check this, the unittest’s assertEqual() 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. |
| **if \_\_name\_\_== '\_\_main\_\_':**  **unittest.main()** | This code will allow you run the file as well as you can export it later as a module. |

As a general rule, if you are testing a file called “filename.py”, then use test file name as “**test\_filename.py**” and keep them in same folder.

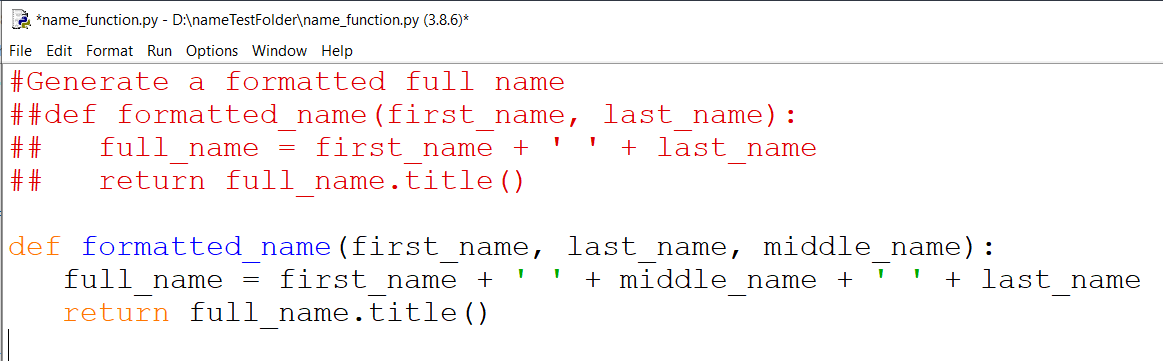
When you execute the file you will get the following screen



Which means the result from the function matches the expected output.

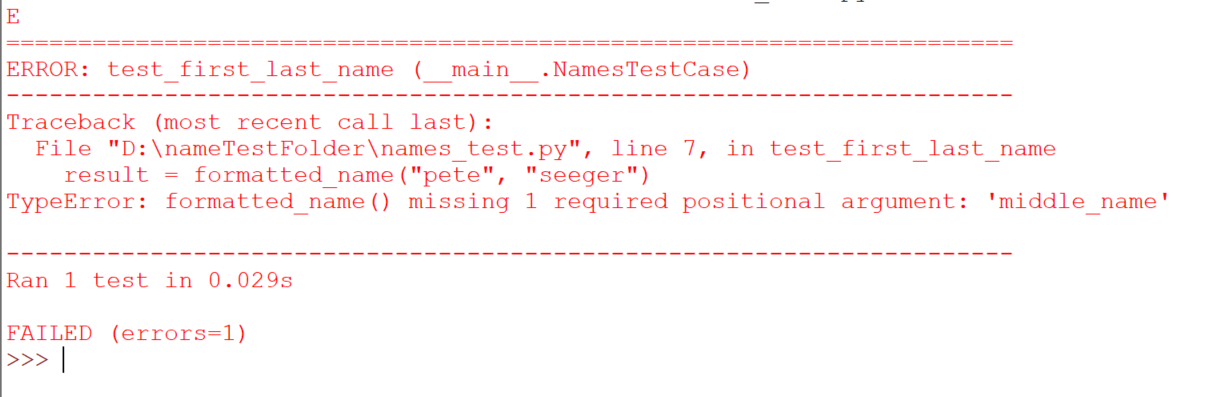
## Failing a test

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, or any other change that affect the output. I commented the old code, and wrote the new code as shown below:

as you know the expected output of this function should have first, middle and last name.

* First item in the output is the Error telling you that at least one test in test case resulted in an error.

Now if you execute (run) the test file ***test\_name\_function.py*** you get:



Next you’ll see the file and method in which the error occurred and which line number

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.

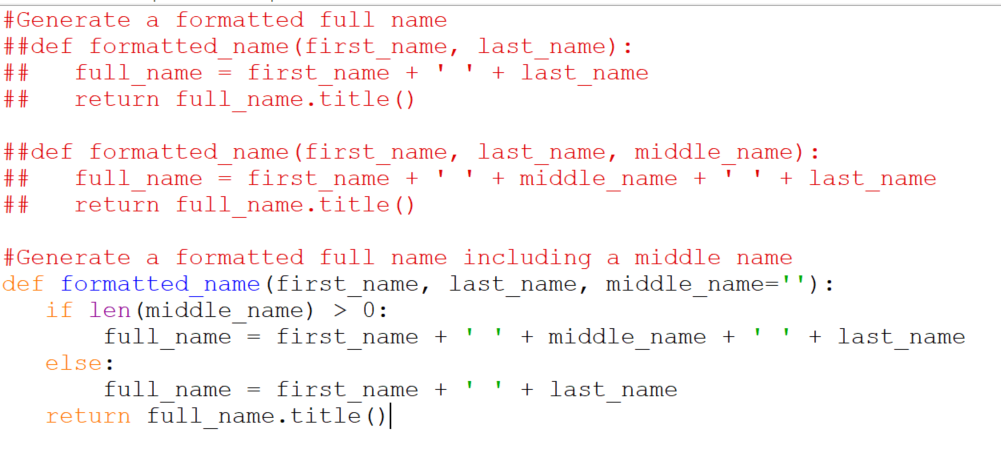
## What to do when the test has failed

A passing test means the function is behaving according to what’s expected from it. However, a failing test means you need to find the error and fix it.

I’ve seen some students that prefer to change the test instead of improving the code — but don’t to 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.

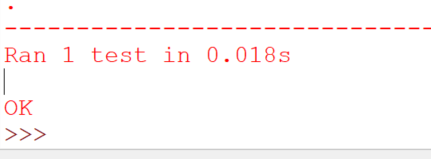
In this example, our function **formatted\_name()** first required two parameters, and now as it is rewritten it requires one extra: a middle name. Adding a middle name to our function broke the desired behaviour of it.

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:



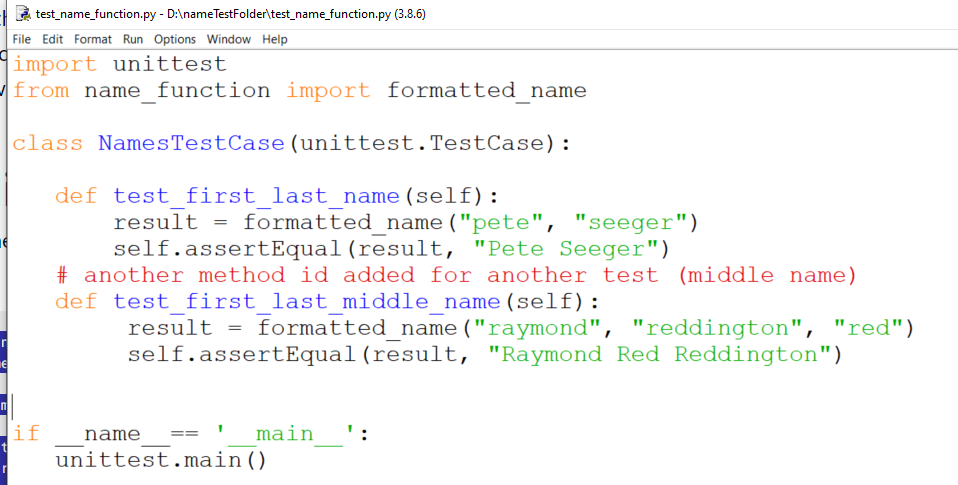
Note I have commented the old code and rewritten a new code with required modification to include the middle name, if there is no middle name it will output first and last name only (doesn’t crash).

Let us run the test again and see:

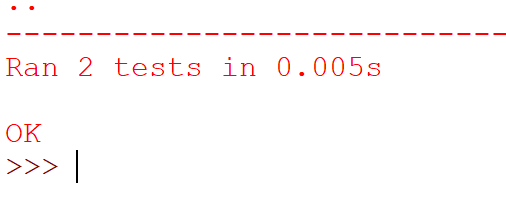


## Adding a new Tests

Let us add anew method to test the middle name as well. The NamesTestCase need to be modified to include another test:

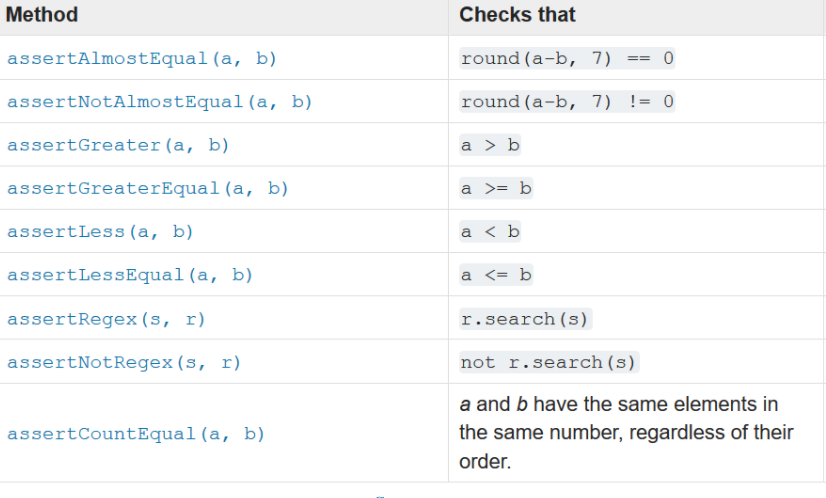


When you run the file you will get :



What else we can add?

There are a lot of assert methods that you can use for different tests, here is a summary:



They all used like the esserEqual() method used in the code above.

# Activity 2 (Do it yourself)

**You must use function in every program you will write here.**

1. write a program that takes two lists as shown above and returns full name list. Example if you have two lists, first name and last:

first = [“Abid”, “James”, “Ed”, “Sara”]

last = [“Issa”, “Johnson”, “Jones”, “White”]

after running the file, it should return:

[“Abid Issa”, “James Johnson”, “Ed Jones”, “Sara White”]

1. write a unit test file (using unittest and test case) to test your file you have created in part 1.
2. Run the test file and make sure if you got an error, you will fix it. You mustrun the test file every time you have fixed any error.