Lab 4: Pytest: Testing in Python

Patrick Di Salvo

November 3, 2019

Due Date: Friday November 8th 11:59 pm

1 Overview

The purpose of this lab is to get you comfortable using pytest. Pytest is a python package used for unit testing python functions. You will be given two files, one called "functions.py" the other "test_functions.py". "functions.py" is completed for you whereas "test_functions.py" is not. You have four jobs for this lab, the first is to install pytest, the second is the complete "test_functions.py", the third is to create a makefile that will run pytest tests and the fourth is to create a README with your name, student number and answers to some short answer questions. Each task will be explained in detail below. **NOTE: We will be using python 2.7 NOT python 3. Therefore you must make sure your tests run with python 2.7**

2 Part 1: Installing Pytest

If not already done so, you must install pytest. You can do so using "pip install -U pytest". After installing, if you type "pytest" in the terminal and get the following error message: "pytest: command not found" then type the following into the terminal "python -m pytest". **NOTE: Your Makefile commands must begin with "python -m pytest..."**

3 Part 2: Writing the Functions

The following section will cover all you need to do for the test_functions.py file. test_functions.py calls the functions in functions.py. The functions in test_functions.py are the functions that pytest will be calling. Do not change the functions functions.py. Do not change the name, return type or parameters of any of the functions in test_functions.py. You will need to write the function definitions as well as add what are called markers to the functions in test_functions.py. If you do not already

know what markers are in the context of pytest, look them up because you will need them. In general, each of the functions in test_functions.py will be asserting that the result of a function in functions.py is or is not an expected result. The parameters to the functions in function.py as well as the expected result will be hard coded. I will give you the values to hard code and you are to use those values. You will note that the definition of test_addNumbers(): is filled in for you. The rest of the functions will be similar.

3.1 functions.py

The functions in this file are already completed. Do not change these functions at all. You will simply call them in the functions in test_fuctions.py. Note that these functions all take in two arguments and return a result.

3.2 test_functions.py

You will be writing the functions in test_functions.py. How to write each function will be explained below.

3.2.1 Function 1: test_addNumbers

Note that the function definition is already completed for you. The rest of your functions will look similar to test_addNumbers

3.2.2 Function 2: test_productNumbers

Hard coded values:
first parameter = 2
second parameter = 3
expected result = 6
You must use these hard coded values in your function definition. The point of this function is to create a test case such that the assert statement passes when testing the product function in functions.py.

3.2.3 Function 3: test_addNumbersFail

Hard Coded Values:
first parameter = 2
second parameter = 3
expected result = 9
You must use these hard coded values. The point of this function is for the assert statement to fail when the result of add in functions.py is returned

3.2.4 Function 4: test_productNumbersFail

Hard Coded Values:
first parameter = 2
second parameter = 3
expected result = 9
You must use these hard coded values. The point of this function is for the assert statement to fail when the result of product in functions.py is returned

3.2.5 Function 5: test_addStrings

Hard Coded Values:
first parameter = "Mary "
second parameter = "Lou"
expected result = "Mary Lou"
You must use these hard coded values. The point of this function is to assert that the strings are concatenated correctly with the add function in functions.py

3.2.6 Function 6: test_productStrings

Hard Coded Values:
first parameter = "Mary "
second parameter = 3
expected result = "Mary Mary Mary "
You must use these hard coded values. The point of this function is to assert that the string, "Mary " is concatenated to itself 3 times. You can do this with the product function in functions.py

3.2.7 Function 7: tes_subtractNumbers

Hard Coded Values:
first parameter = 6
second parameter = 4
expected result = 2
You must use these hard coded values. The point of this function is to assert that the result from the subtract function in functions.py is the same as the expected result

3.2.8 Function 7: test_skip

The point of this function is to be skipped by pytest when pytest is being run.

4 Makefile

4.1 General

The makefile will have five targets plus the target 'all' which will have five dependencies, each one being one of the other five targets. For example:
When I type make or make all in the terminal, the target in the makefile I created when writing this lab is
all: targ1 targ2 targ3 targ4 targ5

Each of the other 5 targets will run pytest on test_functions.py. I do not care what you call the targets, but your make all must run all five targets.

Note: All your make file command line commands must begin as follows: python -m pytest
Note: For all tests use the -v flag. It stands for verbose

4.1.1 Target 1:

This test processes all the functions in test_functions.py that do not contain the substring "Fail"

4.1.2 Target 2:

This test only processes the function test_productNumbers

4.1.3 Target 3:

This test only processes the functions for strings. **Note, you must use a pytest marker to ensure that only the functions pertaining to strings are being tested**

4.1.4 Target 4:

This test only processes the functions test_productNumbers and test_addNumbers. **Note, you must use a pytest marker to ensure that only these two functions are being tested**

4.1.5 Target 5:

This test will process all the functions in test_functions.py but pytest will exit/quit once two asserts fail.

5 Short Answer Section

5.1 General

Create a README with your name, student number and answers to the following short answer questions

5.2 Question 1:

Explain why the function tes_subtractNumbers(): does not get processed by pytest.

5.3 Question 2:

Explain what would happen if you changed the name of the file test_functions.py to TestFunctions.py and ran pytest on the file. Do not actually change the name of the file.

6 Submission

You are to submit a .tar file as lastNameFirstNameL4.tar. For example, my submission would be disalvoPatrickL4.tar. In the tar file you are to have the following files. functions.py, test_functions.py, README.txt and Makefile. Do not put in any other files.