

Software Quality Assurance

Assignment 1

Section A

Zaid Mustafa

22-10249

# Unit Test

The unit testing framework was designed using JUnit as a primary inspiration, and it has many characteristics in common with well-known unit testing frameworks created in other languages. Tests can be entirely independent of the reporting system, testing can be organized into collections, tests can be automated, and code for setting up and tearing down tests can be shared. Unittest can accomplish this goal by utilizing an object-oriented approach to provide support for several important concepts, including the following:

## Test Fixture

A test fixture is a depiction of the procedures that are involved in one or more tests, including the clean-up. Creating temporary or proxy databases, directories, or even beginning the process of starting a server are all alternatives worth considering.

## Test Case

A "test case" is a predetermined part of the testing process. When given a specific set of inputs, it looks for a particular result to produce. The TestCase foundation class provided by Unittest serves as the constructing element for new test cases.

## Test Suite

It is possible to think of a test suite as either a collection of test cases or a collection of further test suites. It allows for the scheduling of the concurrent execution of several tests.

## Test Runner

An aspect of software known as a test runner directs and displays the results of test runs. The runner can display the results of the tests in various ways, including through a graphical user interface, a textual user interface, or by returning a value.

## Method

I will complete this assignment by utilizing the PyUnit testing framework, which I have already created six test cases for.

|  |  |
| --- | --- |
| **Method** | **Verification** |
| AssertEqual(a, b) | a == b |
| AssertNotEqual(a, b) | a! = b |
| AssertTrue(x) | bool(x) is True |
| AssertFalse(x) | bool(x) is False |
| AssertGreater(a,b) | a > b |
| AssertLess(a,b) | a < b |

Samples of Functional Code which will be Tested:

I have included the following two functions in this description for completeness.

A function for calculating power, which will be assessed using assertEqual(a, b), assertNotEqual(a, b), assertTrue(x), and assertFalse(x) respectively.

The function that will be used to determine whether or not a string contains any palindromes (this function will be evaluated using the comparison operators assertGreater(a,b) and assertLess(a,b)).

To demonstrate how the power function works, I'll use the square of 2, which is equal to 4.

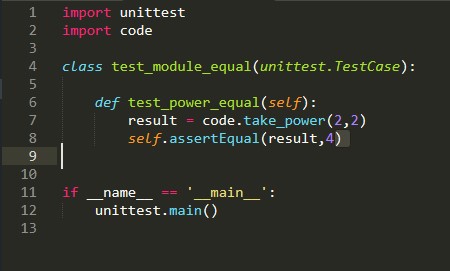
For the sake of this definition, I will consider the string "abba" to be a palindrome; however, the string "abca" will not be treated as a palindrome by me.

PyUnit is going to be used in these two methods so that I can test it.

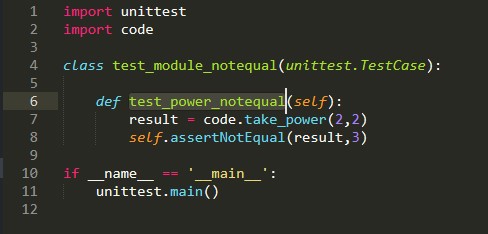
# Code

## Test Cases

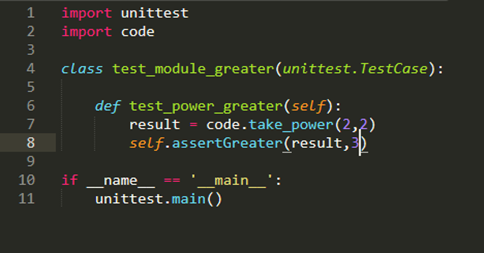
1. test\_module\_equal



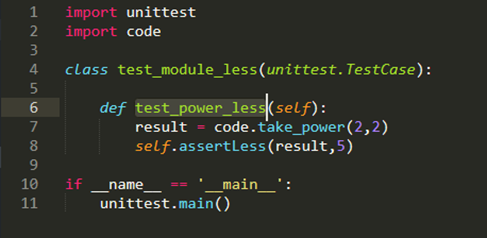
1. test\_module\_notequal



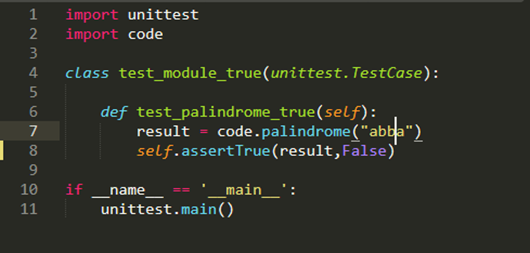
1. test\_module\_greater



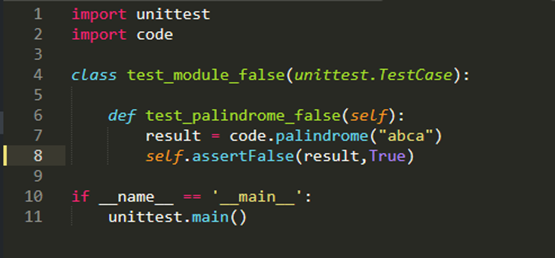
1. test\_module\_less



1. test\_module\_true



1. test\_module\_false

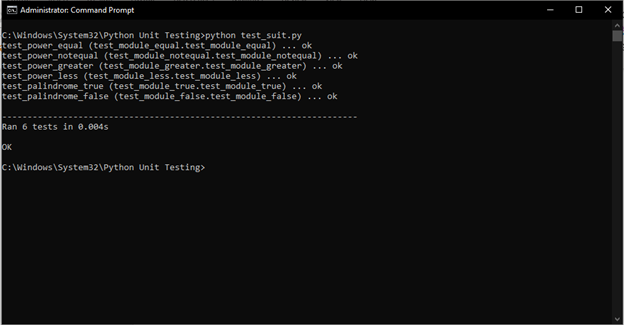


# Test Suite



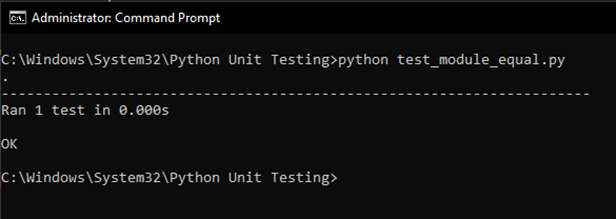
# Output

Suppose the source code of both the functions and the test cases is free of errors. In that case, we can use the command line interface to run the entirety of the Test Suite, verifying that all six test cases run without producing any errors during the execution phase.



# Individual Test Cases

1. test\_module\_equal

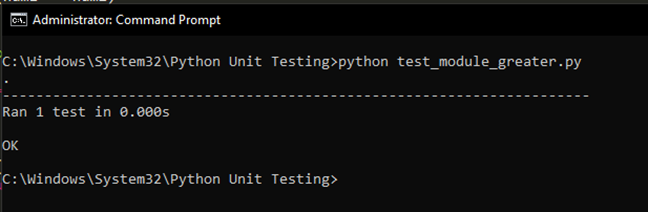


1. test\_module\_notequal

Text

Description automatically generated

1. test\_module\_greater



1. test\_module\_less

Text

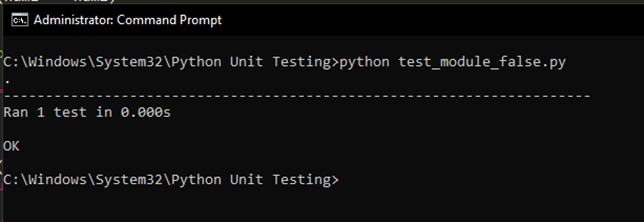
Description automatically generated

1. test\_module\_true

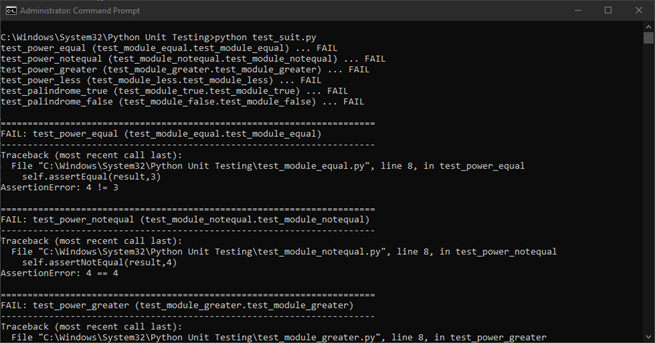
Text

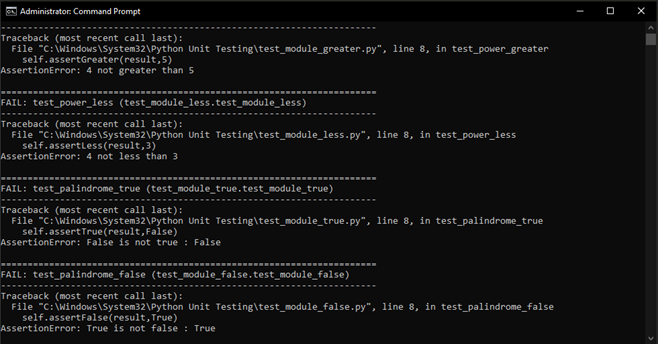
Description automatically generated

1. test\_module\_false



# Failure Output Test Suites





Graphical user interface

Description automatically generated with low confidence