**PYTHON UNITTEST FRAMEWORK**

Unit testing is a software testing method by which individual units of source code, such as functions, methods, and class are tested to determine whether they are fit for use.

In unit test framework, each smallest testable part of an application is considered as a unit. In our case, the smallest part is each function in every file of python project.

For every function, there should a positive and negative case. Positive case refers to the case of which the test case should be written ensuring that users can perform appropriate action when using valid data. Negative case refers to the case of which the test case should be expecting the function to do appropriate action if the user enters the invalid data. Along with these two test cases, we can check for different scenarios with respect to the expect output.

**testcase:**  
A test case is a set of conditions which is used to determine whether a system under test works correctly.

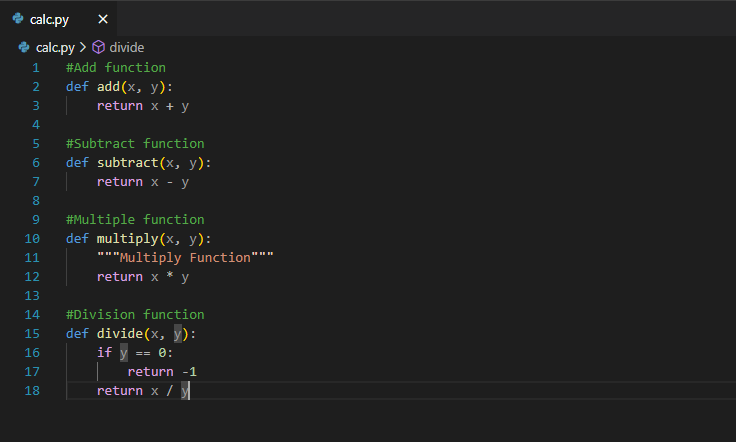
**testsuite:**  
Test suite is a collection of testcases that are used to test a software program to show that it has some specified set of behaviors by executing the aggregated tests together.

**testrunner:**  
A test runner is a component which set up the execution of tests and provides the outcome to the user.

Steps for Writing Unit Testing using Python

**Step 1:**

Create a calc.py file in the project directory. Add methods namely add, subtract, multiply and division to perform basic mathematic operations. As shown in figure 1



***Figure 1: calc.py file***

**Step 2:**

Create a test file in same project directory called test\_calc.py

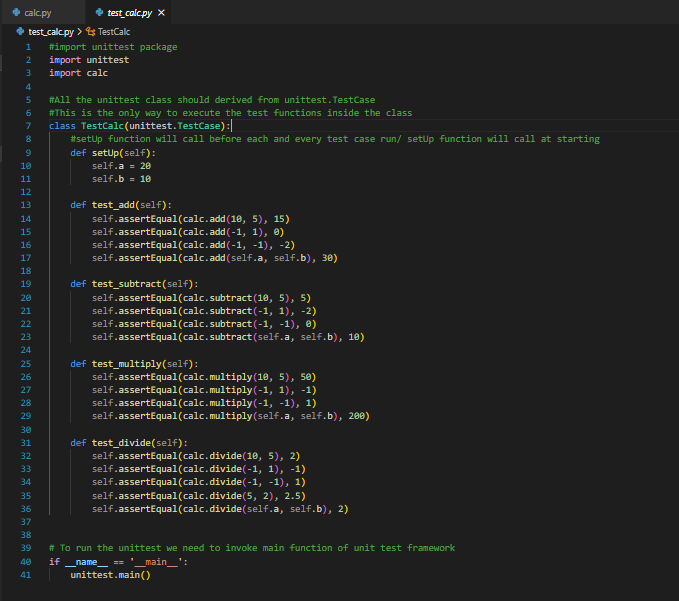
All the test cases are written inside the test file.

Test file will cover the all the units (functions) in all possible ways generally positive and negative cases (eg. Test\_calc.py file cover all test case of calc.py file)

* We have to first **import unitttest** framework in our file
* Start writing the test cases in the file by defining the

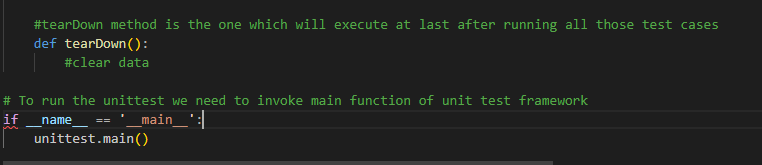
**class Classname(unittest.TestCase):**

* Define a **setUp** method inside the class
* The **setUp** method is the one which executes first of all in the class. So, the variables which are needed by all of the test cases should be placed here



***figure 2: test\_cal.py file***

* The methods in which we are going to write the test cases should be defined by using **test** keyword, because the mechanism behind running the class will give the status of those test cases by considering the methods which uses **test** keyword
* Every test case contains three types of status. One is **Pass**, Second is **Fail**, Third is **Error**
* Every test case we are writing must pass without any failures and errors.
* In the positive test case, we have to test if the valid data is provided as input, are we getting the expected response or not
* In the negative test case, we have to provide the invalid data as input to test what function is returning in case of the invalid data i.e. to test whether the function is handling the different dimensions of user’s inputs of data.
* Here is the snippet of positive and negative test cases. The data provided in the positive case is valid data and hence it should be true whereas in negative, the function should return false
* After covering all the test cases, we have to define a **tearDown** method in the end of the class
* The **tearDown** methodis the one which will execute at last after running all those test cases written by us. As shown in figure 3.



***figure3: teardown method to clear the data***

**HTML TEST REPORT GENERATION**

Automation Reports is an ideal way through which we can track progress, improve the readability of the test output thereby minimizing the amount of time required in the maintenance of the test data (& results).

To generate HTML reports that have in-depth information about the tests in the HTML format, execution results, etc.; we have to make use of HTMLTestRunner.py code which contains the information regarding which colors to indicate when the pass or fail, table to display the test cases information, count of total test cases, count of test cases passed, count of test cases failed, count of test cases generating errors, and detail view of those test cases when click on the summary.

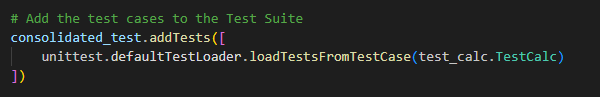
We have to import that **HTMLTestRunner** source code in our file and along with that, all the testing files

* Define on which directory the html file should generate the report. We are generating the html test report file in the current directory itself.

**current\_directory = os.getcwd()**

* Create a class like discussed before which should be the sub class of unittest

**class Classname(unittest.TestCase):**



* Then, in a test case method inside the class, we have to define one test suite for which we are going to add all testing files we have written for the module

**consolidated\_tests = unittest.TestSuite()**

* Now, we have to add whatever files we have to test in a list

**consolidated\_tests.addTests([ unittest.defaultTestLoader.loadTestsFromTestCase(filename.ClassName)]**

* This will load all the test cases from the subClass which was inherited from unittest and where our testcases are defined in that file. We can add ‘n’ number of files in this list if we want to test those files.
* We have to create one file which can name as **HTML\_Test\_Runner.html** in our current directory where our html test report would be stored

**output\_file = open(current\_directory + "\HTML\_Test\_Runner.html", "wb")**

* Now, the test report will be generated by giving the stream as output\_file

**html\_runner = HTMLTestRunner.HTMLTestRunner(**

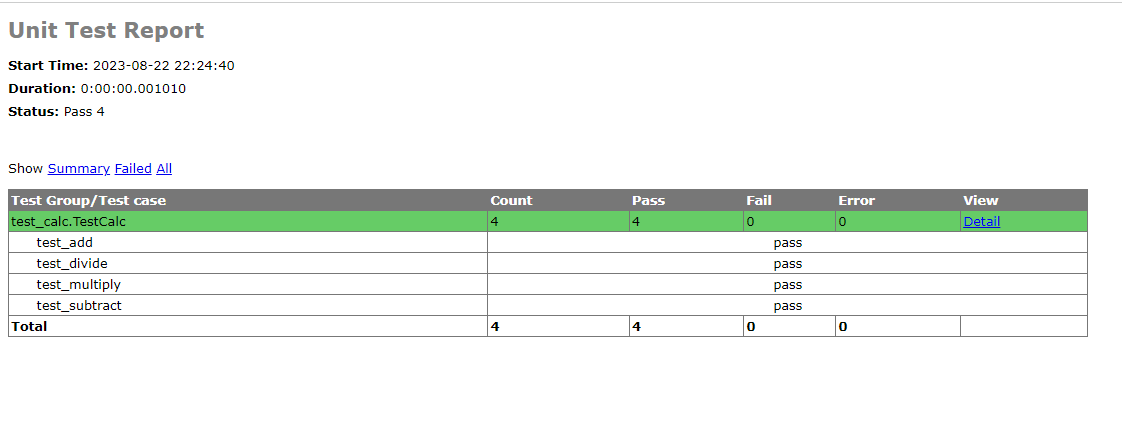
**stream=output\_file,**

**verbosity= 2)**

* After the test reports are written that file, we have to close that file

**output\_file.close()**

* Now, go to the current directory and open the **HTML\_Test\_Runner.html** file where the easy interpretation of test cases summary would be known.
* The html test report **summary** would be looking like as below if we open **HTML\_Test\_Runner.html** file



***figure 5: HTML test report***