**Exercise 3**

**Unit testing**

What is unit testing?

* Unit testing is a software testing method by which individual units of source code are put under various tests to determine whether they are fit for use.
* A unit test is a scripted code level test designed in Python to verify a small “unit” of functionality. Unit test is an object oriented framework based around test fixtures.
* It determines and ascertains the quality of your code. The primary goal of unit testing is to test an individual unit of system to analyze, detect, and fix the errors.
* Unit testing is done to identify bugs early in the development stage of the application when bugs are less recurrent and less expensive to fix.
* Generally, when the development process is complete, the developer codes criteria, or the results that are known to be potentially practical and useful, into the test script to verify a particular unit's correctness. During test case execution, various frameworks log tests that fail any criterion and report them in a summary.

Unit testing can be performed in two ways as follows:-

**Automate Testing :**

The automated testing executes the code according to the code plan set up by the tester, which infers that it runs a part of the code that we want to test, the order in which we want to test them by a script instead of a human.

**Manual Testing:**

Manual testing is a software testing process in which test cases are executed manually without using any automated tool. All test cases executed by the tester manually according to the end user's perspective. It ensures whether the application is working, as mentioned in the requirement document or not. Test cases are planned and implemented to complete almost 100 percent of the software application. Test case reports are also generated manually.

## Python Unit Testing Techniques

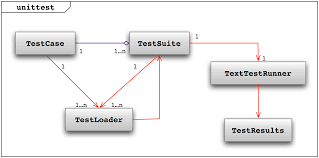
Python Unit Testing mainly involves testing a particular module without accessing any dependent code. Developers can use techniques like stubs and mocks to separate code into “units” and run unit level testing on the individual pieces.

* **Test-Driven Development TDD:**Unit Testing should be done along with the Python, and for that developers use Test-Driven Development method. In TDD method, you first design Python Unit tests and only then you carry on writing the code that will implement this feature.
* **Stubs and Mocks:**They are two main techniques that simulate fake methods that are being tested. A **Stub**is used to fill in some dependency required for unit test to run correctly. A **Mock** on the other hand is a fake object which runs the tests where we put assert.

The intentions of both methods are same to eliminate testing all the dependencies of a class or function.

## Unit Testing using unittest

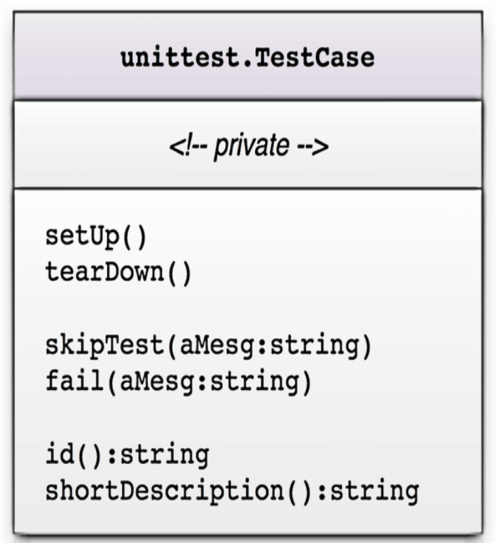
The [unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.) unit testing framework was originally inspired by JUnit . It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

In the unittest module there are five key classes. 

* **TestCase class**: The TestCase class bears the test routines and delivers hooks for making each routine and cleaning up thereafter
* **TestSuite class**: It caters as a collection container, and it can possess multiple testcase objects and multiple testsuites objects
* **TestLoader class**: This class loads test cases and suites defined locally or from an external file. It emits a testsuite objects that posseses those suites and cases
* **TextTestRunner class**: To run the tests it caters a standard platform to execute the tests
* **The TestResults class**: It offers a standard container for the test results

## Designing a test case for Python Testing

A unit test provides a base class, test case, which may be used to create new test cases. For designing the test case, there are three sets of methods used are



* In the first set are the pre and post test hooks. The setup() method begins before each test routine, the teardown() after the routine.
* The second set of method controls test execution. Both methods take a message string as input, and both cancel an ongoing test. But the skiptest() method aborts the current test while the fail() method fails it completely.
* The last or third method help determining the test. The method id() returns a string consisting of the name of the testcase object and of the test routine. And the method shortDescription() returns the docstr comment at the initiation of each test routine.

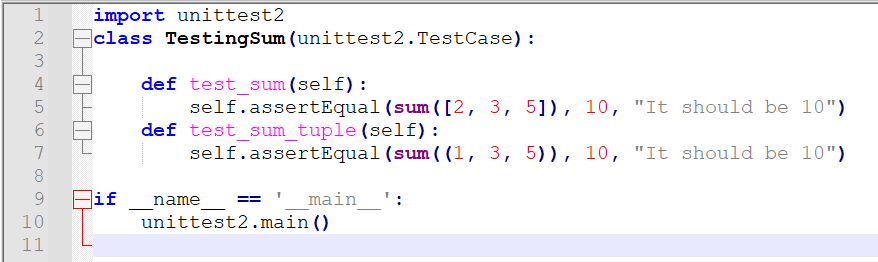
## ****Advantages of Unit Testing****

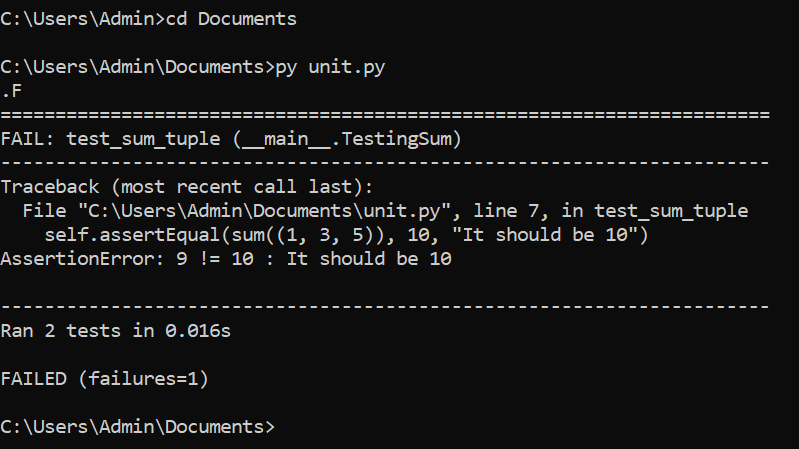
1. Unit tests make it safer and easier to re factor the code by putting tests into place that make sure refactoring occurs without problems and disruption. It takes the risk out of changing older source code.
2. Doing unit tests is essentially doing quality assurance of the code. It shows problems and bugs before the product has an integration test. Creating a testing process before the coding is completed solves issues and challenges creators to write better code.
3. UT helps find problems and resolve them before further testing so they won’t impact other bits of code. This includes bugs in a programmer’s execution and issues with a specification for the unit itself.
4. UT allows the refactoring of code and makes integration simpler. It finds changes and helps maintain and adjust code, reducing bugs and defects, and verifying the accuracy of each unit. It makes sure the later testing is easier once the integration process begins.
5. This type of testing maps a system and creates documentation. It helps understand the unit’s interface.
6. UT makes the process of debugging easier.
7. Using a unit test and good unit testing tools means you reduce the overall cost of a project. Early bug detection means fewer late changes and easier to spot issues than if it is done at a later stage.

## ****Disadvantages of Unit Testing****

1. With UT, you have to increase the amount of code that needs to be written. You usually have to write one or more unit tests depending on how complex things are. It is suggested to have at least three so you don’t just get a yes and a no that contradicts each other. While the test code should be fairly simple, this testing method is still more work and more code which means more hours and more cost.
2. Unit tests are problematic when you need to test your user interface (UI). They are good for when you need to test business logic implementation but not great for UI.
3. There is a school of thought that unit tests are problematic for a product’s structural design. They solidify the structure of code which means change can be problematic when needed.
4. In comparison to those who say UT improves code, others say it makes it worse and ends up adding indirection that is pointless. Changing code and adding new code can mean navigational issues and more time spent before integration testing is even started.
5. UT cannot and will not catch all errors in a program. There is no way it can [test every execution](https://theqalead.com/topics/executing-a-testing-project/) path or find integration errors and full system issues.
6. Unit tests have to be realistic. You want the unit you’re testing to act as it will as part of the full system. If this doesn’t happen, the test value and accuracy are compromised.

Example :-





**Profiling Modules**

Code profiling is an attempt to find bottlenecks in our code. Profiling is supposed to find what parts of our code take the longest. Once we know that, then we can look at those pieces of our code and try to find ways to optimize it.  It is better to optimize the code in order to increase the efficiency of a program. Profilers can collect several types of information: timing, function calls, interruptions, cache faults.

There are two types of profiling

***Deterministic Profiling***: All events are monitored. It provides accurate information but has a big impact on performance (overhead). It means the code run slower under profiling. Its use in production systems is often impractical. This type of profiling is suitable for small functions.

***Statistical profiling****:*Sampling the execution state at regular intervals to compute indicators. This method is less accurate, but it also reduces the overhead.

**Python comes with three profilers built in: cProfile, profile and hotshot.**

1. [**cProfile**](https://docs.python.org/2/library/profile.html#module-cProfile) is recommended for most users; it’s a C extension with reasonable overhead that makes it suitable for profiling long-running programs. Based on **lsprof**, contributed by Brett Rosen and Ted Czotter.
2. [**profile**](https://docs.python.org/2/library/profile.html#module-profile), a pure Python module whose interface is imitated by **[cProfile](https://docs.python.org/2/library/profile.html" \l "module-cProfile" \o "cProfile)**, but which adds significant overhead to profiled programs. If you’re trying to extend the profiler in some way, the task might be easier with this module. Originally designed and written by Jim Roskind.This now also reports the time spent in calls to built-in functions and methods.
3. [**hotshot**](https://docs.python.org/2/library/hotshot.html#module-hotshot) was an experimental C module that focused on minimizing the overhead of profiling, at the expense of longer data post-processing times. It is no longer maintained and may be dropped in a future version of Python.

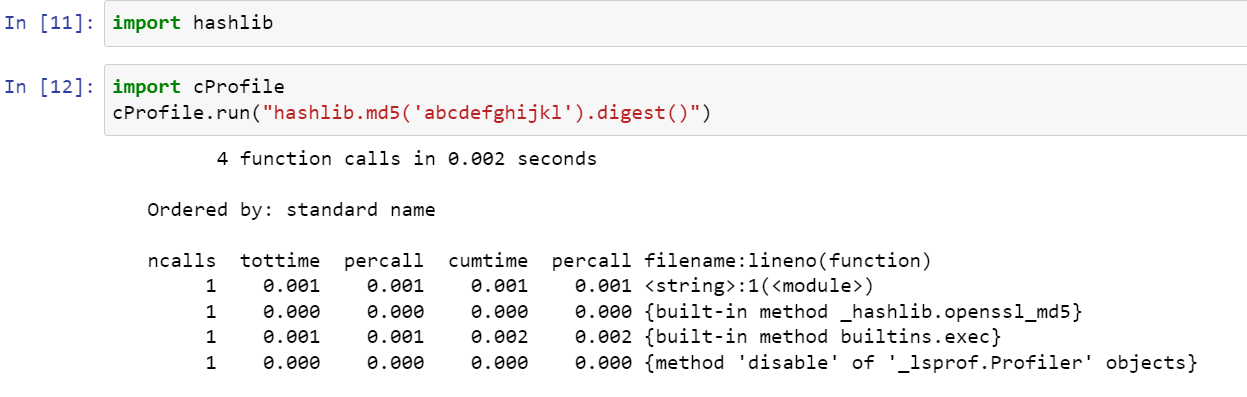
According to the Python documentation, **hotshot** "no longer maintained and may be dropped in a future version of Python".

The **profile** module is a pure Python module, but adds a lot of overhead to profiled programs.

[**cProfile**](https://docs.python.org/2/library/profile.html#module-cProfile) and [**profile**](https://docs.python.org/2/library/profile.html#module-profile) provide deterministic profiling of Python programs.

The [**profile**](https://docs.python.org/2/library/profile.html#module-profile) and **[cProfile](https://docs.python.org/2/library/profile.html" \l "module-cProfile" \o "cProfile)** modules export the same interface, so they are mostly interchangeable; **[cProfile](https://docs.python.org/2/library/profile.html" \l "module-cProfile" \o "cProfile)** has a much lower overhead but is newer and might not be available on all systems. **[cProfile](https://docs.python.org/2/library/profile.html" \l "module-cProfile" \o "cProfile)** is really a compatibility layer on top of the internal **\_lsprof** module. A profile is a set of statistics that describes how often and for how long various parts of the program executed. These statistics can be formatted into reports via the **[pstats](https://docs.python.org/2/library/profile.html" \l "module-pstats" \o "pstats: Statistics object for use with the profiler.)** module.

**Example using cprofile**



Here I have imported the **hashlib** module and used cProfile to profile the creation of an MD5 hash. The first line shows that there were 4 function calls. The next line tells us how the results are ordered. According to the documentation, standard name refers to the far right column. Following are the columns present here:-

* **ncalls** is the number of calls made.
* **tottime** is a total of the time spent in the given function.
* **percall**refers to the quotient of tottime divided by ncalls
* **cumtime** is the cumulative time spent in this and all subfunctions. It's even accurate for recursive functions!
* The second **percall column** is the quotient of cumtime divided by primitive calls
* **filename:lineno(function)** provides the respective data of each function

A primitive call is one that was not induced via recursion.

**Example using Profile:**

