# Unittest - Unit Testing Framework

~ “We can’t feel, smell or touch, but we can TEST it.”

**Unit testing is a software testing method by which individual units of source code are tested to determine whether they are fit for use.**

In unit testing, we perform lowest level of testing that’s individual units of software are tested. That means units like classes, functions, sub routines are tested to verify If they are giving desired result or not!

**Unit Test are**

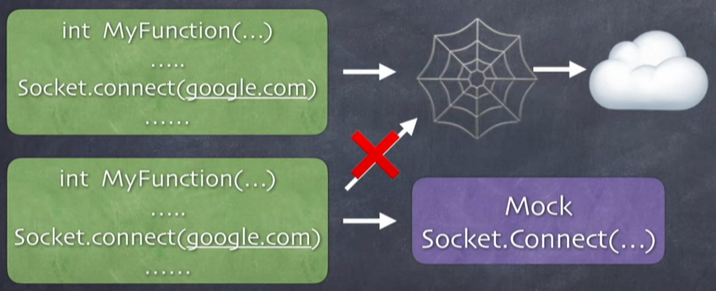
* Test(s) written for testing a unit of code
* One Unit Test runs independently of any other Unit Test, Code etc.
* External Dependencies are managed with Doubles (Mocks /Fakes / Stubs)
* Should complete within milliseconds

**Test Doubles**

* Used in lieu of External Dependencies – (It will found the problem is in your code or in backend) (but our intention was to check our code.)
* DB, Web, API, Library, Network etc.
* Easy to simulate various scenarios

e.g. Mocks, Fakes, Stubs

**Mocks -** Mocks replaces external interface

****

MOCKS are NOT used for checking Function behaviour or return values from the function call?

Used for –

Mock function called or NOT?

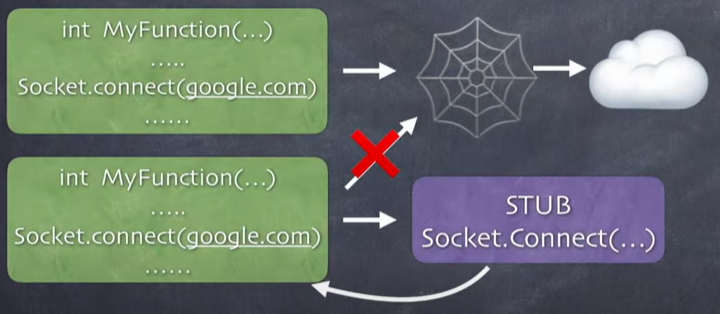
How many times it gets called?

What Parameters are passed

When it was called?

*Mocks are trying to check Right call, Right Number of times with right set of parameters.*

**STUBS -** Generates Predefined Outputs

****

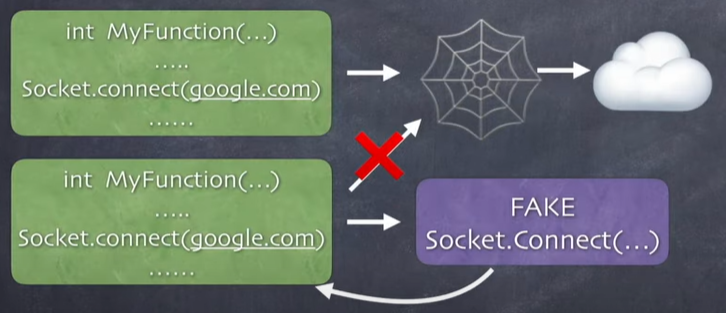
Programmed STUB to return a success, failure or exception.

Returns success, failure or Exception (as coded)

Checks the behaviour of code under test in case of these return values

**FAKES -**

Almost working Implementation (Shortcut Implementation)

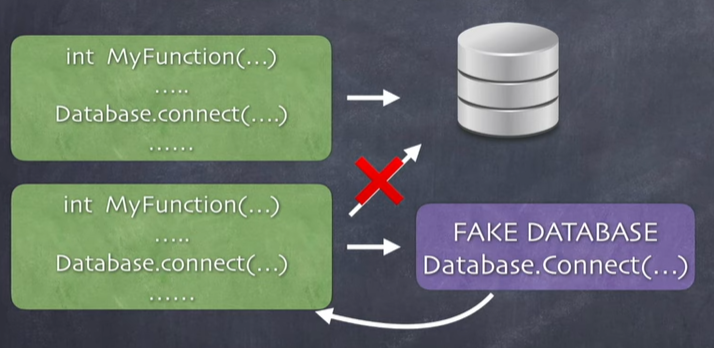


Connect to a local HTTP server.

Instead of actually going to the internet it connects to a local (limited) Implementation.

Check the behaviour with respect to the actual data it receives from the server

Almost Working Implementation



Connect to an in Memory DB

Created specifically for this test

How the code behaves when it really gets lots of data.

**Why Unit Test? Importance of Unittest -**

1. **Tests Reduce Bugs in New Features and Existing Features**
2. **Tests Are Good Documentation**
3. **Tests Reduce the Cost of Change**
4. **Faster Debugging & Faster Development**
5. **Better Design**

When you write unit test they will reduce the bugs in code which may appears while execution of code. To Test the functionality of our code, i.e. Weather our code is working as expected or not.

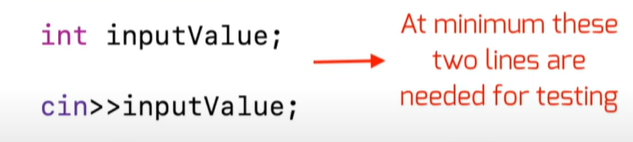
Good unit tests serves as a good documentation for person who is using your code. because we are using the classes, functions and testing them, so the person who wants to testing them reads the test he will be able to understand how to use your code.

Also you write test for your code it will reduces the cost of change in the production. If some code goes in production without testing, and you find the bug in the production stage then it’s much costlier than fixing your code when you’re developing the code!

Unit test helps in faster debugging and Program also because you have idea for desired result. That means you develop a code which is perfect for test in desired manner.

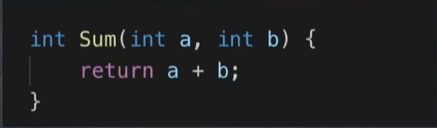
Unit test helps to better design our program.

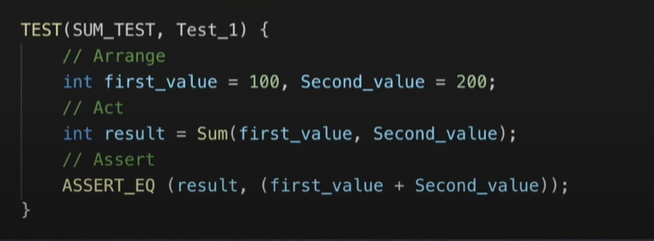
**What is meant by the code that can execute independently?**



Unit test must confirms AAA model [Arrange, Act, Assert]

* Arrange prerequisite to run the test.
* Act means run the test.
* Assert is where verify the test results.





**Testing frameworks –**

* **Unittest - In the Python Standard Library**
* Nose - Not in the Standard Library. Simpler tests than unittest
* Pytest - Not in the Python Standard Library.

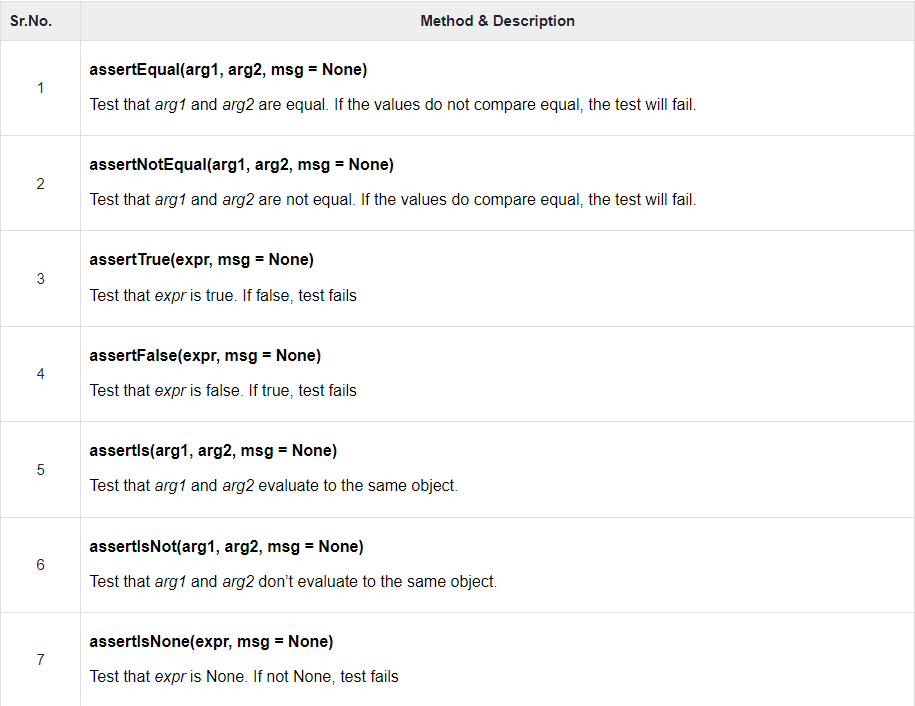
# Assertion

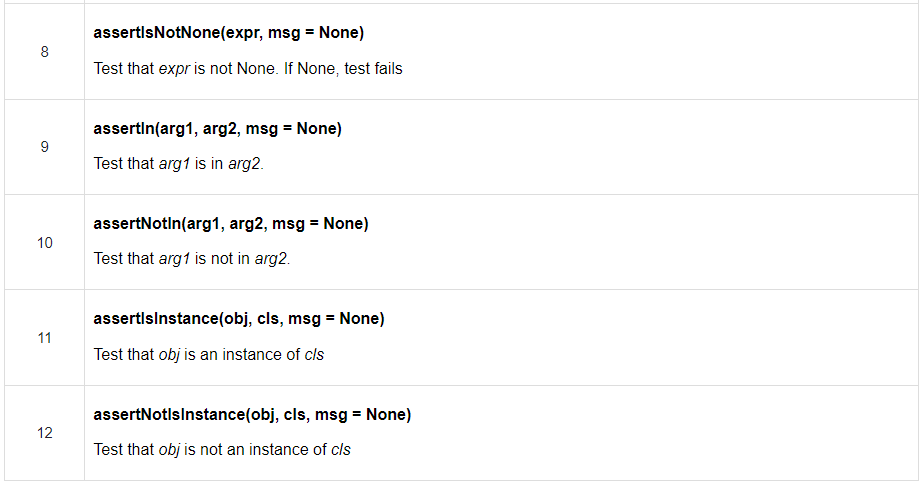
Python testing framework uses Python's built-in assert() function which tests a particular condition. If the assertion fails, an AssertionError will be raised. The testing framework will then identify the test as Failure

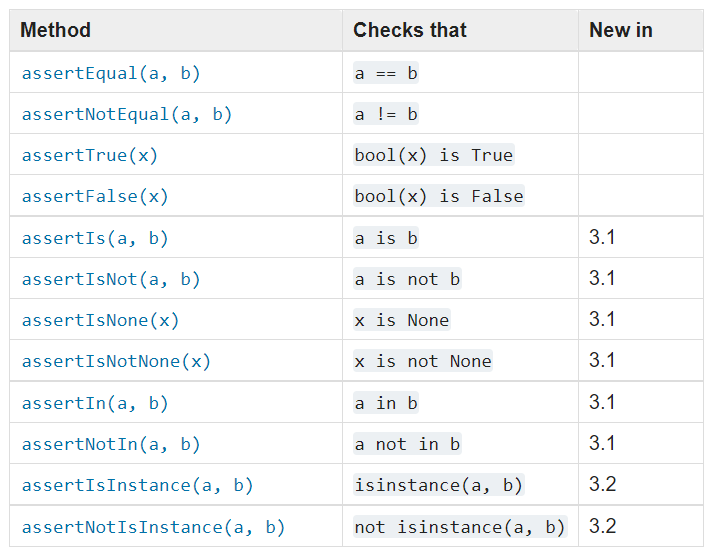
The following three sets of assertion functions are defined in unittest module −

* Basic Boolean Asserts
* Comparative Asserts
* Asserts for Collections

Basic assert functions evaluate whether the result of an operation is True or False. All the assert methods accept a message argument that, if specified, is used as the error message on failure.







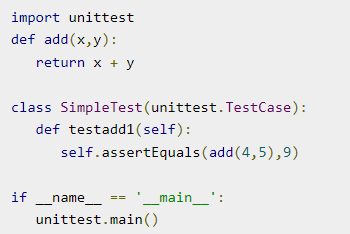
# Framework

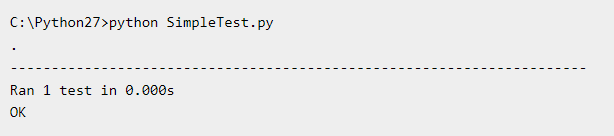
To achieve this, unittest supports the following important concepts −

* Test fixture − this represents the preparation needed to perform one or more tests, and any associate clean-up actions. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.
* Test case − this is the smallest unit of testing. This checks for a specific response to a particular set of inputs. unittest provides a base class, TestCase, which may be used to create new test cases.
* Test suite − this is a collection of test cases, test suites, or both. This is used to aggregate tests that should be executed together. Test suites are implemented by the TestSuite class.
* Test runner − this is a component which orchestrates the execution of tests and provides the outcome to the user. The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests.

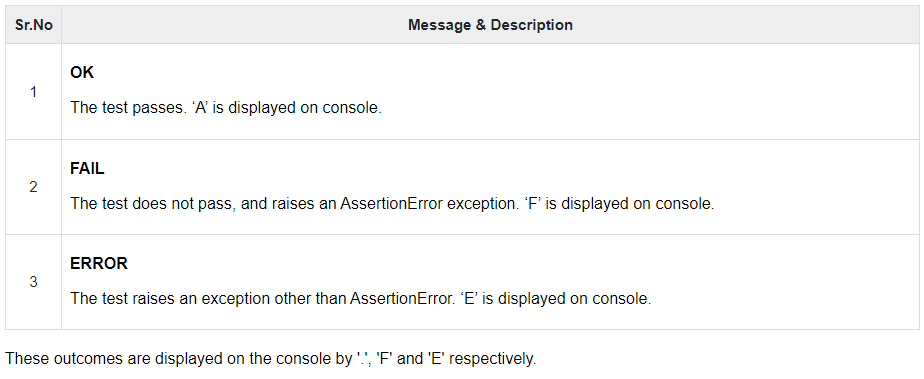
**Creating a Unit Test**

* The following steps are involved in writing a simple unit test −
* Step 1 − Import the unittest module in your program.
* Step 2 − Define a function to be tested. In the following example, add() function is to be subjected to test.
* Step 3 − Create a testcase by subclassing unittest.TestCase.
* Step 4 − Define a test as a method inside the class. Name of method must start with 'test'.
* Step 5 − each test calls assert function of TestCase class. There are many types of asserts. Following example calls assertEquals() function.
* Step 6 − assertEquals() function compares result of add() function with arg2 argument and throws assertionError if comparison fails.
* Step 7 − finally, call main() method from the unittest module.
* Step 8 − Run the above script from the command line.





Step 9 − the following three could be the possible outcomes of a test –



## **TestCase Class**

The classes and methods defined in the unittest module.

