[unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.) — Unit testing framework

Reference: <https://docs.python.org/3/library/unittest.html>

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 and has a similar flavor as major unit testing frameworks in other languages. 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.

To achieve this, [unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.) supports some important concepts in an object-oriented way:

**test fixture**

A *test fixture* represents the preparation needed to perform one or more tests, and any associated cleanup actions. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.

**test case**

A *test case* is the individual unit of testing. It checks for a specific response to a particular set of inputs. [unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.) provides a base class, [TestCase](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase" \o "unittest.TestCase), which may be used to create new test cases.

**test suite**

A *test suite* is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.

**test runner**

A *test runner* 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.

The [unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.) module provides a rich set of tools for constructing and running tests. This section demonstrates that a small subset of the tools suffice to meet the needs of most users.

**import** **unittest**

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

**def** test\_upper(self):

self.assertEqual('foo'.upper(), 'FOO')

**def** test\_isupper(self):

self.assertTrue('FOO'.isupper())

self.assertFalse('Foo'.isupper())

**def** test\_split(self):

s = 'hello world'

self.assertEqual(s.split(), ['hello', 'world'])

*# check that s.split fails when the separator is not a string*

**with** self.assertRaises(TypeError):

s.split(2)

**if** \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

A testcase is created by subclassing [unittest.TestCase](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase" \o "unittest.TestCase).

**Assert Methods:**

|  |  |
| --- | --- |
| **assertEqual**(*first*, *second*, *msg=None*) | Test that first and second are equal. If the values do not compare equal, the test will fail. |
| **assertNotEqual**(first, second, msg=None) | Test that first and second are not equal. If the values do compare equal, the test will fail. |
| **assertTrue**(expr, msg=None) **assertFalse**(expr, msg=None) | Test that expr is true (or false). |
| **assertIs**(first, second, msg=None)  **assertIsNot**(first, second, msg=None) | Test that first and second evaluate (or don’t evaluate) to the same object. |
| **assertIsNone**(*expr*, *msg=None*)  **assertIsNotNone**(*expr*, *msg=None*) | Test that expr is (or is not) None. |
| **assertIn**(*first*, *second*, *msg=None*)  **assertNotIn**(*first*, *second*, *msg=None*) | Test that first is (or is not) in second. |
| **assertIsInstance**(*obj*, *cls*, *msg=None*)  **assertNotIsInstance**(*obj*, *cls*, *msg=None*) | Test that obj is (or is not) an instance of cls (which can be a class or a tuple of classes, as supported by [isinstance()](https://docs.python.org/3/library/functions.html" \l "isinstance" \o "isinstance)). |

### **Test cases**

class unittest.**TestCase**(methodName='runTest')

Instances of the [TestCase](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase" \o "unittest.TestCase) class represent the logical test units in the [unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.) universe.

Each instance of [TestCase](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase" \o "unittest.TestCase) will run a single base method: the method named methodName. In most uses of [TestCase](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase" \o "unittest.TestCase), you will neither change the methodName nor reimplement the default runTest() method.

**setUp**()

Method called to prepare the test fixture. This is called immediately before calling the test method

**tearDown**()

Method called immediately after the test method has been called and the result recorded.

**setUpClass**()

A class method called before tests in an individual class are run. setUpClass is called with the class as the only argument and must be decorated as a [classmethod()](https://docs.python.org/3/library/functions.html" \l "classmethod" \o "classmethod):

**tearDownClass**()

A class method called after tests in an individual class have run. tearDownClass is called with the class as the only argument and must be decorated as a [classmethod()](https://docs.python.org/3/library/functions.html" \l "classmethod" \o "classmethod):

**run**(*result=None*)

Run the test, collecting the result into the [TestResult](https://docs.python.org/3/library/unittest.html" \l "unittest.TestResult" \o "unittest.TestResult) object passed as *result*. If *result* is omitted or None, a temporary result object is created (by calling the [defaultTestResult()](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.defaultTestResult" \o "unittest.TestCase.defaultTestResult) method) and used. The result object is returned to [run()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.run)’s caller.

**skipTest**(*reason*)

Calling this during a test method or [setUp()](https://docs.python.org/3/library/unittest.html" \l "unittest.TestCase.setUp" \o "unittest.TestCase.setUp) skips the current test. See [Skipping tests and expected failures](https://docs.python.org/3/library/unittest.html#unittest-skipping) for more information.

**@unittest**.skip("showing class skipping")

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

**def** test\_not\_run(self):

**pass**

It is recommended that you use TestCase implementations to group tests together according to the features they test. [unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.) provides a mechanism for this: the test suite, represented by [unittest](https://docs.python.org/3/library/unittest.html" \l "module-unittest" \o "unittest: Unit testing framework for Python.)’s [TestSuite](https://docs.python.org/3/library/unittest.html" \l "unittest.TestSuite" \o "unittest.TestSuite) class. In most cases, calling [unittest.main()](https://docs.python.org/3/library/unittest.html" \l "unittest.main" \o "unittest.main) will do the right thing and collect all the module’s test cases for you and execute them.

**def** suite():

suite = unittest.TestSuite()

suite.addTest(TestCaseExample('test\_method1'))

suite.addTest(TestCaseExample ('test\_method2'))

**return** suite

**if** \_\_name\_\_ == '\_\_main\_\_':

runner = unittest.TextTestRunner()

runner.run(suite())

class unittest.**TestLoader**[¶](https://docs.python.org/3/library/unittest.html" \l "unittest.TestLoader" \o "Permalink to this definition)

The [TestLoader](https://docs.python.org/3/library/unittest.html" \l "unittest.TestLoader" \o "unittest.TestLoader) class is used to create test suites from classes and modules. Normally, there is no need to create an instance of this class; the [unittest](https://docs.python.org/3/library/unittest.html#module-unittest) module provides an instance that can be shared as [unittest.defaultTestLoader](https://docs.python.org/3/library/unittest.html" \l "unittest.defaultTestLoader" \o "unittest.defaultTestLoader). Using a subclass or instance, however, allows customization of some configurable properties.

test\_cases = (TestStoreApplication, TestStringFunctions)  
  
def load\_tests():  
 suite = unittest.TestSuite()  
 for test\_class in test\_cases:  
 tests = **unittest.TestLoader().**loadTestsFromTestCase(test\_class)  
 suite.addTests(tests)  
 return suite  
  
*#suite.addTest(TestStoreApplication("test\_validate\_survey\_age\_15"))  
#suite.addTest(TestStringFunctions("test\_string\_uppar"))*runner = unittest.TextTestRunner()  
runner.run(load\_tests())