Testing with Python unittest module

"""Create your own class but avoid using the name that collides with classes or functions in unittest. The names of functions (a.k.a. methods) in the class must begin with “test”. The methods must take only one argument named “self”, which you do not need to pass explicitly. When you call unittest.main(), the methods you have created in your test class will be automatically discovered by the unittest module, and will be called. Please see the following example.
"""

```
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()
```

Some of the Available Functions in unittest

assertEqual(*first*, *second*, *msg=None*)

Test that *first* and *second* are equal. If the values do not compare equal, the test will fail.

In addition, if *first* and *second* are the exact same type and one of list, tuple, dict, set, frozenset or str or any type that a subclass registers with addTypeEqualityFunc() the type-specific equality function will be called in order to generate a more useful default error message (see also the list of type-specific methods).

Changed in version 3.1: Added the automatic calling of type-specific equality function.

Changed in version 3.2: assertMultiLineEqual() added as the default type equality function for comparing strings.

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).

Note that this is equivalent to `bool(expr) is True` and not to `expr is True` (use `assertIs(expr, True)` for the latter). This method should also be avoided when more specific methods are available (e.g. `assertEqual(a, b)` instead of `assertTrue(a == b)`), because they provide a better error message in case of failure.

`assertAlmostEqual`(*`first`*, *`second`*, *`places=7`*, *`msg=None`*, *`delta=None`*)

`assertNotAlmostEqual`(*`first`*, *`second`*, *`places=7`*, *`msg=None`*, *`delta=None`*)

Test that *first* and *second* are approximately (or not approximately) equal by computing the difference, rounding to the given number of decimal *places* (default 7), and comparing to zero. Note that these methods round the values to the given number of *decimal places* (i.e. like the `round()` function) and not *significant digits*.

If *delta* is supplied instead of *places* then the difference between *first* and *second* must be less or equal to (or greater than) *delta*.

Supplying both *delta* and *places* raises a `TypeError`.

Changed in version 3.2: `assertAlmostEqual()` automatically considers almost equal objects that compare equal. `assertNotAlmostEqual()` automatically fails if the objects compare equal. Added the *delta* keyword argument.

assertRaises(*exception*, *callable*, **args*, ***kwds*)
assertRaises(*exception*, *, *msg=None*)

Test that an exception is raised when *callable* is called with any positional or keyword arguments that are also passed to assertRaises(). The test passes if an *exception* is raised, is an error if another exception is raised, or fails if no exception is raised. To catch any of a group of exceptions, a tuple containing the exception classes may be passed as *exception*.

Examples:

```
def test_stack(self):
   s = Stack()
   self.assertRaises(IndexError, s.pop)

def test_queue(self):
   q = Queue(0)
   self.assertRaises(IndexError, q.enqueue, 1)
```

For more information, please read https://docs.python.org/3/library/unittest.html