Automated software testing with Python

Software testing is the process in which a developer ensures that the actual output of the software matches with the desired output by providing some test inputs to the software. Software testing is an important step because if performed properly, it can help the developer to find bugs in the software in very less amount of time.

Software testing can be divided into two classes, **Manual testing** and **Automated testing**. Automated testing is the execution of your tests using a script instead of a human. In this article, we'll discuss some of the methods of automated software testing with Python.

```
class Square:
    def __init__(self, side):
        """ creates a square havir
        """
        self.side = side

    def area(self):
        """ returns area of the sc
        """
        return self.side**2

    def perimeter(self):
        """ returns perimeter of t
        """
        return 4 * self.side

    def __repr__(self):
        """ declares how a Square
        """
        s = 'Square with side = '
        'Area = ' + str(self.area(
        'Perimeter = ' + str(self.
if __name__ == '__main__':
    # read input from the user
    side = int(input('enter the si

    # create a square with the prc
    square = Square(side)

    # print the created square
    print(square)
```

The ‘unittest’ module

One of the major problems with manual testing is that it requires time and effort. In manual testing, we test the application over some input, if it fails, either we note it down or we debug the application for that particular test input, and then we repeat the process. With `unittest`, all the test inputs can be provided at once and then you can test your application. In the end, you get a detailed report with all the failed test cases clearly specified, if any.

The `unittest` module has both a built-in testing framework and a test runner. A testing framework is a set of rules which must be followed while writing test cases, while a test runner is a tool which executes these tests with a bunch of settings, and collects the results.

Installation: `unittest` is available at PyPI and can be installed with the following command –

```
pip install unittest
```

Use: We write the tests in a Python module (.py). To run our tests, we simply execute the test module using any IDE or terminal.

Now, let's write some tests for our small software discussed above using the `unittest` module.

1. Create a file named `tests.py` in the folder named "tests".
2. In `tests.py` import `unittest`.
3. Create a class named `TestClass` which inherits from the class `unittest.TestCase`.

Rule 1: All the tests are written as the methods of a class, which must inherit from the class `unittest.TestCase`.

4. Create a test method as shown below.

Rule 2: Name of each and every test method should start with "test" otherwise it'll be skipped by the test runner.

```
def test_area(self):
    # testing the method Square.

    sq = Square(2)    # creates

    # test if the area of the ab
    # display an error message i

    self.assertEqual(sq.area(),
        f'Area is shown {sq.area
```

Rule 3: We use special `assertEqual()` statements instead of the built-in

Rule 3: We use special `assertEqual()` statements instead of the built-in `assert` statements available in Python.

The first argument of `assertEqual()` is the actual output, the second argument is the desired output and the third argument is the error message which would be displayed in case the two values differ from each other (test fails).

5. To run the tests we just defined, we need to call the method `unittest.main()`, add the following lines in the “tests.py” module.

```
if __name__ == '__main__':
    unittest.main()
```

```
import unittest
from .. import app

class TestSum(unittest.TestCase):

    def test_area(self):
        sq = app.Square(2)

        self.assertEqual(sq.area()
            f'Area is shown {sq.ar

if __name__ == '__main__':
    unittest.main()
```

Having written our test cases let us now test our application for any bugs. To test your application you simply need to execute the test file "tests.py" using the command prompt or any IDE of your choice. The output should be something like this.

A few things to note in the above test report are –

- The first line represents that test 1 and test 3 executed successfully while test 2 and test 4 failed
- Each failed test case is described in the report, the first line of the description contains the name of the failed test case and the last line contains the error message we defined for that test case.
- At the end of the report you can see the number of failed tests, if no test fails the report will end with `OK`

Note: For further knowledge you can read the complete documentation of `unittest`.