Part 3: Automated Test Script

Bookmark this page

Part 3: Automated Test Script

If you were to sit down and think about this problem, here's one solution that you could come up with:

- When I try squaring 3 I expect to get 9, but I got 27. When I try squaring 100 I expect to get 10,000, but I got a much larger number.
- It takes me a minute or two to try these examples at the REPL, and getting into the REPL and trying these examples is kind of a hassle.
- So I could write a Python script that tries using the Squarer class to square 3 and 100. Running the script will be easy and only take a few seconds, so when I make a change to Squarer it will only take me a few seconds to find out whether I've broken it or not.
- I could make a habit of running this test script every time I make a change to Squarer. If I have a colleague who is working on the Squarer class, then they should run the script whenever *they* make a change to Squarer. In this way, we'll know right away if we've introduced an error into Squarer.

Here's a suitable test script:

```
# test.py
from squarer import Squarer

class SquarerTest(object):

    @staticmethod
    def test_positive_numbers():

        squares = {
            1: 1,
            2: 4,
            3: 9,
            12: 144,
            100: 10000,
        }

        for num, square in squares.items():
            result = Squarer.calc(num)

            if result != square:
                print("Squared {} and got {} but expected {}".format(num, result, square))
    @staticmethod
    def test_negative_numbers():

        squares = {
            -1: 1,
            -2: 4,
            -3: 9,
            -12: 144,
            -100: 10000,
        }

        for num, square in squares.items():
            result = Squarer.calc(num)

            if result != square:
                print("Squared {} and got {} but expected {}".format(num, result, square))

if __name__ == "__main__":
    SquarerTest.test_positive_numbers()
```

```
    SquarerTest.test_negative_numbers()
```

The SquarerTest class has two static methods: test_positive_numbers and test_negative_numbers. I'll explain the test_positive_numbers method, and the test_negative_numbers method is nearly identical.

The goal of this script is to define the expected behavior of Squarer and test Squarer against that expected behavior. So we begin by creating a dictionary that defines that behavior:

```
squares = {
    1: 1,
    2: 4,
    3: 9,
    12: 144,
    100: 10000,
}
```

< Previous Next >

Each *key* in this dictionary is a number. The corresponding *value* is the value that we expect Squarer to produce when we square the key. So one of the key/value pairs that we are testing is 12 and 144: squaring 12 should produce 144.

Next we iterate through these number, square pairs:


```
for num, square in squares.items():
    result = Squarer.calc(num)

    if result != square:
        print("Squared {} and got {} but expected {}\n".format(num, result, square))
```

We use Squarer to square the number, and capture the result in a variable named *result*. If *result* is not equal to the square that we defined in *squares*, then we print a message describing the error.

Finally, we add a *__name__ == "__main__"* clause so that we can run this script from the command line:

```
if __name__ == "__main__":
    SquarerTest.test_positive_numbers()
    SquarerTest.test_negative_numbers()
```