Написание Unit тестов. Лабораторная работа №4

Среди всех тестов львиную долю занимают именно unit-тесты. В классическом понимании unit-тесты позволяют быстро и автоматически протестировать отдельные части ПО независимо от остальных.

В этой лабораторной работе мы рассмотрим простой пример создания unit-тестов.

Ручное тестирование

Начнем с простого варианта - ручного тестирования:

— зная алгоритм нахождения периметра и площади фигуры, определяем наборы входных данных, которые будут переданы на вход программе;

— зная входные данные, можно вручную просчитать, какой ответ должна дать программа;

— запускаем программу и передаем ей на вход исходные данные;

— получаем от нее ответ и сравниваем с тем, который должен быть получен. Если они совпадают — хорошо, идём к следующему набору данных, если нет, сообщаем об ошибке, фиксируем в протоколе тестирования.

Unittest

Обычно Python поставляется уже с пакетом unittest. Если в вашей системе его нет, используйте pip для его установки.

Формат кода

- тесты должны быть написаны в классе;
- класс должен быть наследован от базового класса unittest.TestCase;

- имена всех функций, являющихся тестами, должны начинаться с ключевого слова test;
- внутри функций должны быть вызовы операторов сравнения (assertX) — именно они будут проверять наши полученные значения на соответствие заявленным.

Пример использования unittest для нашей задачи (rectangle.py)

```
import unittest
…
class RectangleTestCase(unittest.TestCase):
   def test_zero_mul(self):
       res = area(10, 0)
       self.assertEqual(res, 0)

   def test_square_mul(self):
       res = area(10, 10)
       self.assertEqual(res, 100)
```

Запускается данный код следующей командой

python.exe -m unittest rectangle.py

И в результате на экран будет выведено:

```
C:\GSS\git_test>python -m unittest rectangle.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK

C:\GSS\git_test>
```

В случае, если в каком-нибудь из тестов будет обнаружена ошибка, unittest вернет ответ:

```
C:\GSS\git_test>python -m unittest rectangle.py
F.
======================================================================
FAIL: test_square_mul (rectangle.RectangleTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\GSS\git_test\rectangle.py", line 16, in test_square_mul
    self.assertEqual(res, 100)
AssertionError: 110 != 100

----------------------------------------------------------------------
Ran 2 tests in 0.002s

FAILED (failures=1)

C:\GSS\git_test>
```

Подготовка отчета

В отчете вам нужно будет написать план тестирования, который минимально должен содержать следующие пункты:

1. Цели и задачи тестирования: Определение основных целей тестирования и задач, которые необходимо достичь в процессе тестирования.

2. Описание тестируемого продукта: Обзор функциональности, особенностей и требований к продукту, которые должны быть протестированы.
3. Область тестирования: Определение конкретных функций, модулей или компонентов продукта, которые будут исследованы в тестирование.
4. Стратегия тестирования: Описание общего подхода к тестированию, включая методы, техники и типы тестирования, которые будут использоваться, например, функциональное тестирование, тестирование производительности, тестирование безопасности и т. д.
5. Критерии приемки: Определение условий и критериев, которые должны быть выполнены для успешного завершения тестирования и приемки продукта.
6. Ожидаемые результаты: Указание ожидаемых результатов тестирования, таких как отчеты о дефектах, статусы тестирования, метрики качества и другие соответствующие данные.

Исходный код тестов должен быть размещён на GitHub (или другом общедоступном репозитории), а в истории документа должна быть отражен факт появления тестов. При форматировании документа опираемся на опыт полученных в ходе выполнения 2 лабораторной работы