МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет ИТМО

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная Работа №4

По дисциплине «Инструментальные средства разработки ПО»

Выполнил студент группы №M3102

Ерофеев Иван Константинович

Проверил

Кирилюк Денис Алексеевич

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург

2024

1. Вот мои функции для вычисления характеристик произвольного прямоугольника, заданного двумя сторонами.

```
    return a * b

def perimeter(a, b):
    return 2 * (a + b)

def diagonal(a, b):
    return math.sqrt(a**2 + b**2)

def inradius(a, b):
    return (a * b) / (a + b) if a + b != 0 else 0

def outradius(a, b):
    return math.sqrt(a**2 + b**2) / 2
```

2. Моей задачей было написать тесты, позволяющие оценить правильность выполнения данных функций. Для этого я использовал библиотеку unittest которая позволяет написать стресс тесты для программы

Название тест-кейса	Входные данные	Результат работы программы	Статус теста
test_one_side_is_zero	(10, 0)	0	Пройден
	(0, 0)	0	Пройден
	(-1, 0)	0	Пройден
	(-1.5, 0)	0	Пройден
	(1000000000039, 0)	0	Пройден
test_square_area	(1, 1)	1	Пройден
	(2, 2)	4	Пройден
	(10, 10)	100	Пройден
	(11, 11)	121	Пройден
	(1000000000039, 1000000000039)	1000000000039 * 1000000000039	Пройден
test_rectangle_area	(3, 4)	12	Пройден
	(5, 10)	50	Пройден
	(1.5, 2.5)	3.75	Пройден
test_perimeter	(3, 4)	14	Пройден
	(5, 10)	30	Пройден
	(1.5, 2.5)	8.0	Пройден
test_diagonal	(3, 4)	5.0	Пройден
	(5, 12)	13.0	Пройден
	(1.5, 2.5)	√(1.5^2 + 2.5^2)	Пройден
test_inradius	(3, 4)	≈ 3.428571	Пройден
	(5, 12)	≈ 7.058824	Пройден
	(1.5, 2.5)	≈ 1.875	Пройден
test_outradius	(3, 4)	2.5	Пройден
	(5, 12)	6.5	Пройден
	(1.5, 2.5)	√(1.5^2 + 2.5^2) / 2	Пройден

3.

4. Функии, которые тестируются:

   area(a, b) — вычисляет площадь прямоугольника.
   perimeter(a, b) — вычисляет периметр прямоугольника.
   diagonal(a, b) — вычисляет длину диагонали прямоугольника.
   inradius(a, b) — вычисляет радиус вписанной окружности.
   outradius(a, b) — вычисляет радиус описанной окружности

5. Тестируются в том числе пограничные и некорректные значения, например когда сторона отрицательная или равна нулю. Также тестируются частные случаи, например когда прямоугольник является квадратом

6. Написал тесты для проверки случая, когда одна сторона равна нулю

```
def test_one_side_is_zero(self):
    self.assertEqual(area(10, 0), 0)
    self.assertEqual(area(0, 0), 0)
    self.assertEqual(area(-1, 0), 0)
    self.assertEqual(area(-1.5, 0), 0)
    self.assertEqual(area(1000000000039, 0), 0)
```

7. Написал тесты для проверки площади, когда прямоугольник является квадратом

```
def test_square_area(self):
    self.assertEqual(area(1, 1), 1);
    self.assertEqual(area(2, 2), 4);
    self.assertEqual(area(10, 10), 100);
    self.assertEqual(area(11, 11), 121);
    self.assertEqual(area(1000000000039, 1000000000039), 1000000000039*1000000000039)
```

8. Написал тесты для проверки площади в общем виде

```
# Tests for rectangle area
def test_rectangle_area(self):
    self.assertEqual(area(3, 4), 12)
    self.assertEqual(area(5, 10), 50)
    self.assertEqual(area(1.5, 2.5), 3.75)
```

9. Написал тесты для нахождения диагонали

```
def test_diagonal(self):
    self.assertAlmostEqual(diagonal(3, 4), 5.0)
    self.assertAlmostEqual(diagonal(5, 12), 13.0)
    self.assertAlmostEqual(diagonal(1.5, 2.5), math.sqrt(1.5**2 + 2.5**2))
```

10. Написал тесты для нахождения вписанной окружности

```
def test_inradius(self):
    self.assertAlmostEqual(inradius(3, 4), 1.2)
    self.assertAlmostEqual(inradius(5, 12), 3.75)
    self.assertAlmostEqual(inradius(1.5, 2.5), 0.9375)
```

11. Написал тесты для нахождения описанной окружности

```
def test_outradius(self):
    self.assertAlmostEqual(outradius(3, 4), 2.5)
    self.assertAlmostEqual(outradius(5, 12), 6.5)
    self.assertAlmostEqual(outradius(1.5, 2.5), math.sqrt(1.5**2 + 2.5**2) / 2)
```

12. Результат работы программы

```
PS C:\Users\User\Desktop\ITMO\ISRPO\Lab_№4> python -m unittest tests.py
.......
----------------------------------------------------------------------
Ran 7 tests in 0.001s

OK
PS C:\Users\User\Desktop\ITMO\ISRPO\Lab_№4>
```

13. Данные тесты позволят проверять любые функции вычисляющие параметры произвольно заданного прямоугольника. Они не только корректно работают и сообщают об ошибках, но и сравнивают значения с эталонным учитывая то, что числа в памяти не могут хранится с бесконечной точностью

14. Для того чтобы протестировать значения необходимо написать python -m unittest <name>.py И тогда все тесты будут автоматически проведены.

15. Продемонстрирую работу тестов когда функция работает с ошибкой

```
----------------------------------------------------------------------
AIL: test_inradius (tests.RectangleTestCase.test_inradius)
----------------------------------------------------------------------
raceback (most recent call last):
 File "C:\Users\User\Desktop\ITMO\ISRPO\Lab_№4\tests.py", line 35, in test_inradius
   self.assertAlmostEqual(inradius(3, 4), 3.428571, places=6)
   ~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ssertionError: 3.7142857142857144 != 3.428571 within 6 places (0.2857147142857146 difference)

----------------------------------------------------------------------
```

Видно, что даже небольшие расхождения с ответом не будут пропущены тестирующей программой. Кроме того все ошибки будут подсвечены для того, чтобы пользователь мог сразу понять в каком месте программы он имеет расхождения

16. **Метрики качества:** Показатели успешности тестов (количество пройденных и проваленных тестов) помогают оценить качество функций и их устойчивость к разным входным данным

Метрика	Описание	Комментарий
Passed/Failed Test Cases	Отношение успешно пройденных тестов к проваленным для функций `area`, `perimeter`, `diagonal`, `inradius`, `outradius`. Показывает, насколько функции работают как надо.	Высокий процент успешных тестов = всё работает; много ошибок — стоит перепроверить код.
Not Run Test Cases	Число тестов, которые не запустились. Помогает найти тесты, которые надо доработать или исправить.	Пропуски могут означать, что нужно донастроить тесты или окружение.
Test Execution Time	Время на выполнение всех тестов. Удобно для понимания, как быстро работают тесты.	Быстрые тесты = можно проверять чаще.
Code Coverage	Процент кода функций `area`, `perimeter`, `diagonal`, `inradius`, `outradius`, который тестируется. Оценивает, насколько полно мы проверяем код.	Если покрытие 80–100%, это значит, что большинство кода проверено.
Defect Density	Количество ошибок на 1000 строк кода. Помогает понять общее качество кода.	Чем меньше ошибок, тем лучше код.
Test Reliability	Насколько стабильно и предсказуемо проходят тесты.	Стабильные тесты = уверенность, что код работает одинаково.