

Модуль 3 Занятие 8

Тестирование. Assert



Тестирование

assert

doctest

unittest

Тестирование

Тестирование — это проверка, насколько реальное поведение программы или продукта совпадает с ожидаемым.

Написание тестов — это неотъемлемая часть работы современного программиста.

Это интересно



В 1996 году ракета Ariane 5 взорвалась через 36.7 секунд после взлета. К аварии привела ошибка в программе, которая нанесла ущерб в 7 миллиардов долларов.



Ручное тестирование

Тестирование — это отдельная профессия. Тестировщики рассматривают все возможные сценарии поведения программы и пользователя и ищут ошибки в работе проекта. Такое тестирование называют ручное тестирование.

Как, например, можно протестировать форму авторизации на сайте:



Пользователь вводит правильный логин и пароль и входит на сайт



Пользователь вводит неправильный логин и пароль, получает ошибку и приглашение восстановить пароль



Пользователь вводит правильный email в форму восстановления пароля и получает письмо со ссылкой на форму изменения пароля



Пользователь вводит неправильный email в форму восстановления пароля и получает сообщение об ошибке

Тестирование кода

Любой большой проект состоит из частей: функций, классов, методов — и тестирование можно начинать с них. Напишем небольшую программу и попробуем ее протестировать.

Необходимо написать функцию `get_avg(lst)`, которая принимает список чисел и возвращает среднее арифметическое чисел этого списка.

Пример работы функции:

```
print(get_avg([1, 2, 3, 4, 5]))
```

Пример вывода:

```
3.0
```

Возможное решение

```
def get_avg(lst):  
    return sum(lst) / len(lst)
```

1

Что может пойти не так?

2

Как протестировать функцию?

Запишите различные вызовы функции и ожидаемые результаты:

Вызов функции	Ожидаемый результат
get_avg([1, 2, 3, 4, 5])	3.0
get_avg([4, 4, 4])	...
...	

assert

Провести тестирование работы программы можно с помощью инструкции `assert` (в переводе «утверждение»). `assert` в Python — это выражение, которое проверяет, является ли условие истинным (`True`).

Работает так:



Если утверждение истинно — `assert` не возвращает ничего, тест пройден.



Если утверждение оказалось ложным — вызывается исключение `AssertionError` и исполнение кода прерывается, тест провален.

Протестируем функцию

Измените или добавьте какой-либо тест и сделайте так, чтобы тест был провален.

```
def get_avg(lst):  
    if not lst:  
        return 0  
    return sum(lst) / len(lst)  
  
assert get_avg([1, 2, 3, 4, 5]) == 3.0  
assert get_avg([4, 4, 4]) == 4.0  
assert get_avg([]) == 0
```

docstring

Добавим docstring — это строка, которая идет сразу за созданием функции и является удобным способом добавления документации к функции.

В docstring также можно добавить результаты тестов. Доступ к docstring осуществляется через специальную переменную `__doc__`.

```
def get_avg(lst):  
    """  
    Принимает список чисел и возвращает среднее арифметическое  
    чисел этого списка или 0, если список пуст.  
    """  
    if not lst:  
        return 0  
    return sum(lst) / len(lst)
```

doctest

В Python есть стандартная библиотека doctest. Она ищет в docstring интерактивные сеансы (строки с `>>>`) и выполняет их, а затем сравнивает полученный результат с ожидаемым, указанным на следующей за инструкцией строке в docstring.

```
def get_avg(lst):  
    """  
    Принимает список чисел и возвращает среднее арифметическое  
    чисел этого списка или 0, если список пуст.  
    >>> get_avg([1, 2, 3, 4, 5])  
    3.0  
    >>> get_avg([4, 4, 4])  
    4.0  
    >>> get_avg([])  
    0  
    """  
    if not lst:  
        return 0  
    return sum(lst) / len(lst)
```

Запуск doctest

Для того, чтобы запустить doctest, необходимо выполнить одну из команд в терминале:

```
python -m doctest test.py -v
```

```
python -m doctest test.py
```

MINGW64:/c/Users/User/programs

User@DESKTOP-PVGUDRB MINGW64 ~/programs

```
$ python -m doctest program.py -v
```

```
Trying:
```

```
    get_avg([1, 2, 3, 4, 5])
```

```
Expecting:
```

```
    3.0
```

```
ok
```

```
Trying:
```

```
    get_avg([4, 4, 4])
```

```
Expecting:
```

```
    4.0
```

```
ok
```

```
Trying:
```

```
    get_avg([])
```

```
Expecting:
```

```
    0
```

```
ok
```

```
1 items had no tests:
```

```
    program
```

```
1 items passed all tests:
```

```
    3 tests in program.get_avg
```

```
3 tests in 2 items.
```

```
3 passed and 0 failed.
```

```
Test passed.
```

User@DESKTOP-PVGUDRB MINGW64 ~/programs

```
$
```

Автоматический запуск doctest



Теперь doctest будет запускаться автоматически при выполнении файла.

Измените или добавьте какой-либо тест и сделайте так, чтобы тест был провален.

```
def get_avg(lst):  
    """  
    Принимает список чисел и возвращает среднее арифметическое  
    чисел этого списка или 0, если список пуст.  
    >>> get_avg([1, 2, 3, 4, 5])  
    3.0  
    >>> get_avg([3, 3, 3])  
    4.0  
    >>> get_avg([])  
    0  
    """  
  
if __name__ == '__main__':  
    import doctest  
    doctest.testmod()
```

Другие библиотеки тестирования

Тестировать код через `assert` и `doctest` можно, но такое тестирование не подходит для серьезных проектов.



Воспользуемся стандартной библиотекой `unittest` и протестируем код нашей функции.

unittest



Для того, чтобы выполнить тестирование с помощью библиотеки `unittest`, создайте в любой папке два файла: `code.py` и `tests.py`.

папка

```
|— code.py    # файл с тестируемым кодом
|— tests.py   # файл с тестами
```


code.py



В файл code.py добавим код, который будем тестировать.

```
def get_avg(lst):  
    if not lst:  
        return 0  
    return sum(lst) / len(lst)
```

tests.py



В файл tests.py добавим код для тестирования.

```
import unittest

from code import get_avg

class TestGetAvg(unittest.TestCase):
    """Тестируем get_avg."""

    def test_mixed_numbers(self):
        result = get_avg([1, 2, 3, 4, 5])
        expected = 3.0
        self.assertEqual(result, expected, 'Функция должна возвращать среднее арифметическое')

    def test_equal_numbers(self):
        result = get_avg([4, 4, 4])
        expected = 4.0
        self.assertEqual(result, expected, 'Функция должна возвращать среднее арифметическое')

    def test_empty(self):
        result = get_avg([])
        expected = 0
        self.assertEqual(result, expected, 'Функция должна возвращать 0 для пустого списка')
```

Запуск unittest

Для того, чтобы запустить unittest, необходимо выполнить одну из команд в терминале:

```
python -m unittest tests.py -v
```

```
python -m unittest tests.py
```

MINGW64:/c/Users/User/programs

```
User@DESKTOP-PVGUDRB MINGW64 ~/programs
$ python -m unittest tests.py -v
test_empty (tests.TestGetAvg) ... ok
test_equal_numbers (tests.TestGetAvg) ... ok
test_mixed_numbers (tests.TestGetAvg) ... ok
```

```
-----
Ran 3 tests in 0.000s
```

```
OK
```

```
User@DESKTOP-PVGUDRB MINGW64 ~/programs
$
```

Автоматический запуск unittest



Теперь unittest будет запускаться автоматически при выполнении файла.

Измените или добавьте какой-либо тест и сделайте так, чтобы тест был провален.

```
import unittest

from code import get_avg

class TestGetAvg(unittest.TestCase):
    """Тестируем get_avg."""

    def test_mixed_numbers(self):
        ...

    def test_equal_numbers(self):
        ...

    def test_empty(self):
        ...

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

Методы класса TestCase

Кроме assertEquals библиотека unittest содержит и другие методы.
Вот некоторые из них:



assertEquals(a, b) a == b



assertNotEqual(a, b) a != b



assertTrue(x) bool(x) is True



assertFalse(x) bool(x) is False



assertIs(a, b) a is b



assertIsNot(a, b) a is not b



assertIsNone(x) x is None



assertIsNotNone(x) x is not None



assertIn(a, b) a in b



assertNotIn(a, b) a not in b

Документация по модулю

Больше методов и примеров можно найти на странице официальной документации.



Итоги



Тестирование — это проверка, насколько реальное поведение программы или продукта совпадает с ожидаемым.



Провести тестирование работы программы можно с помощью инструкции `assert`.



В `docstring` можно добавить результаты тестов, которые можно проверить с помощью библиотеки `doctest`.



Для более серьезных задач можно воспользоваться `unittest` или другими сторонними библиотеками.