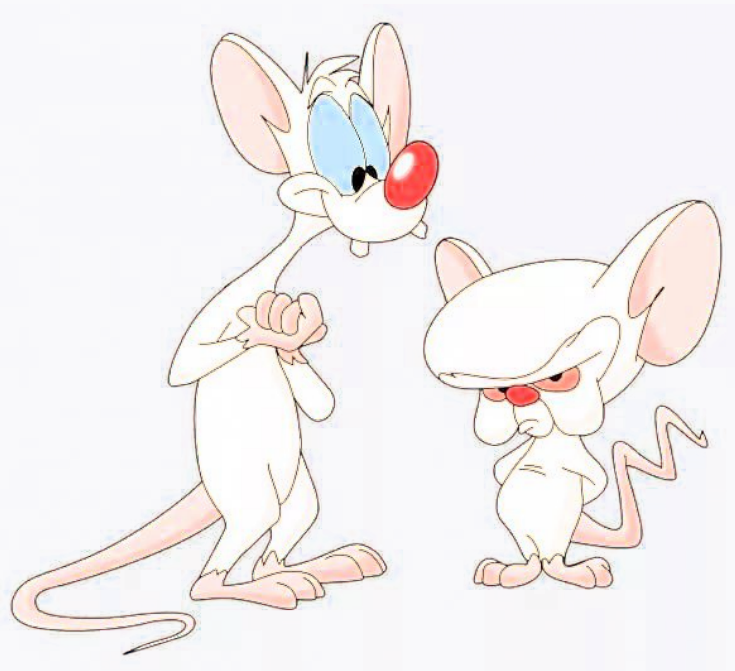
Клиент-серверные приложения на Python

Урок 4

Основы тестирования

Введение в тестирование. Оператор assert.
Модульное тестирование и модуль unittest.

Цели урока



Разобраться с вопросами:

1. Что такое тесты?
2. Как их применять?
3. Когда тесты необходимы, возможны, а когда от них стоит отказаться?

И погрузиться в модульное тестирование и модуль unittest.

Понятия тестирования и теста

Уровни тестирования

1. Компонентное или модульное тестирование;
2. Интеграционное тестирование;
3. Регрессионное тестирование;
4. UI-тестирование.

Тестировать или не тестировать?

За

Тесты проверяют корректность кода.

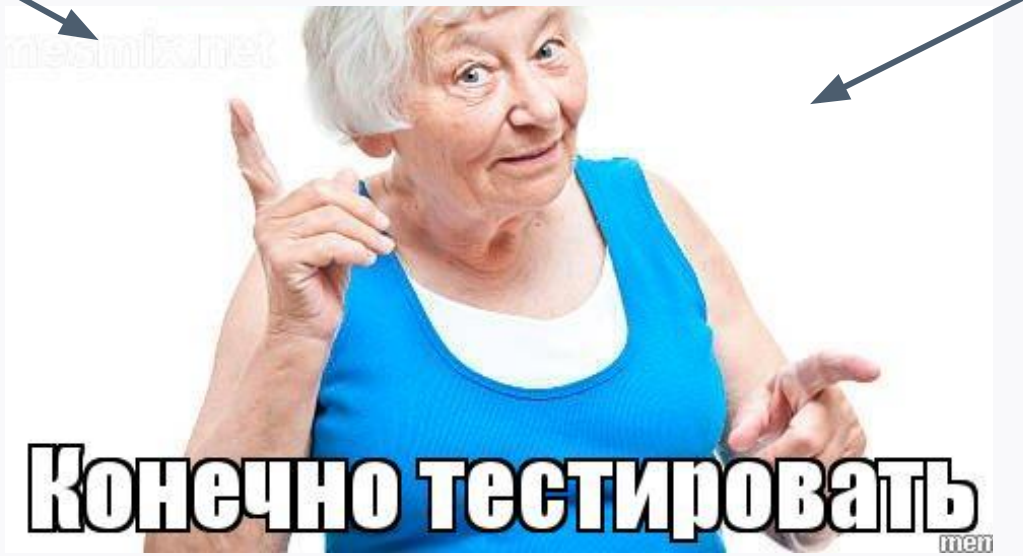
Облегчают изменение кода в больших проектах.

Против

Нужно время для написания тестов.

Тестов может быть больше, чем кода.

Тесты не гарантируют корректность работы программы.



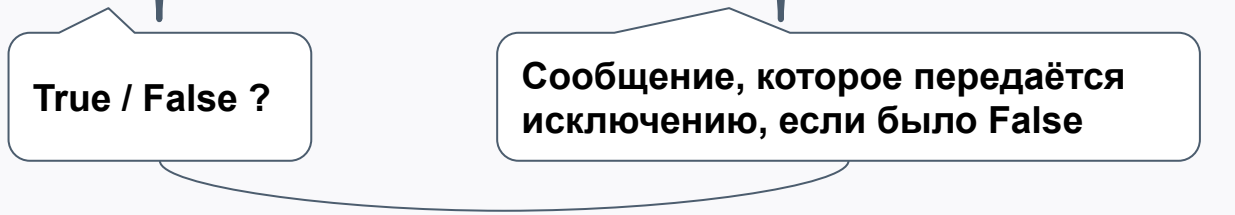
Хороший тест

- Корректный — тестирует то, что нужно проверить;
- Понятен читателю;
- Конкретный — проверяет что-то одно.

Оператор `assert`

```
def write_data(file, data):
    assert file, "write_data: файл не определен!"
    ...
```

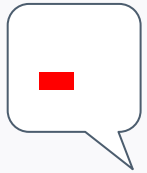
```
def assert_equal(x, y):
    assert x == y, "{} != {}".format(x, y)
```



Оператор `assert`

При использовании для тестов:

- тесты легко читать;
- стандартные средства языка;
- тесты — простые функции.



- запуск теста вручную;
- сложно отлаживать такие тесты.

unittest. Модульное тестирование

Произошёл от JUnit (Java)

Поэтому

Тесты - это **классы-наследники** **`unittest.TestCase`**!

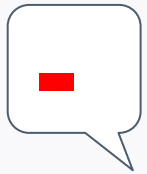
```
class TestSplitFunction(unittest.TestCase):
    def setUp(self):
            # Выполнить настройку тестов (если необходимо)
            pass
    def tearDown(self):
            # Выполнить завершающие действия (если необходимо)
            pass

    def testsimplestring(self):
            r = split('GOOG 100 490.50')
            self.assertEqual(r, ['GOOG', '100', '490.50'])

    def testtypeconvert(self):
            r = split('GOOG 100 490.50', [str, int, float])
            self.assertEqual(r, ['GOOG', 100, 490.5])
```

unittest. Методы

Метод	Проверяет	Появился в Python
`assertEqual(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`	3.1
`assertIsNot(a, b)`	`a is not b`	3.1
`assertIsNone(x)`	`x is None`	3.1
`assertIsNotNone(x)`	`x is not None`	3.1
`assertIn(a, b)`	`a in b`	3.1
`assertNotIn(a, b)`	`a not in b`	3.1
`assertIsInstance(a, b)`	`isinstance(a, b)`	3.2
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`	3.2

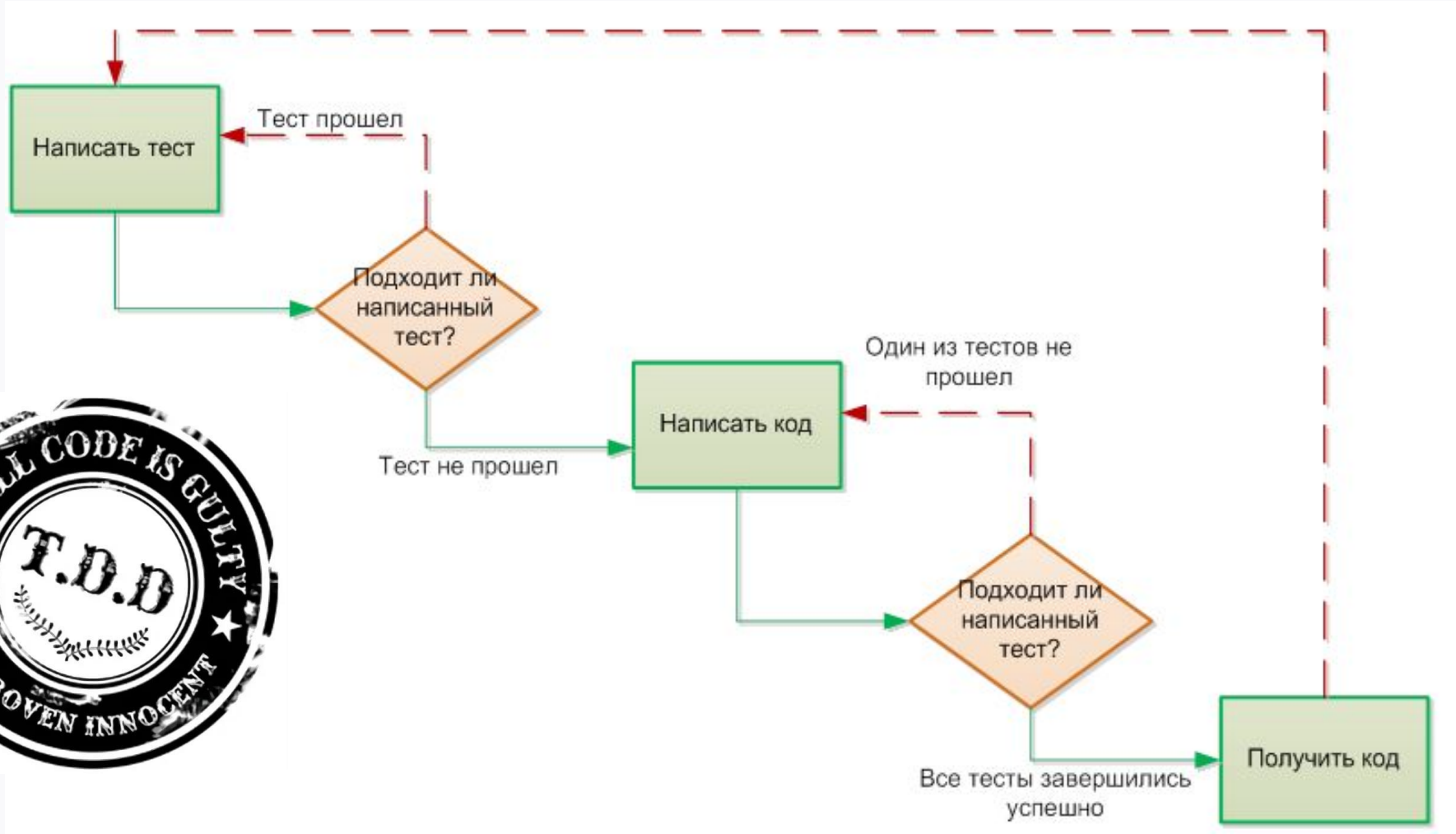
unittest

- доступен в стандартной библиотеке;
- выводит понятные сообщения об ошибках;
- автоматически находит тесты.



- API от Java;
- много лишнего кода;
- читается сложнее, чем `doctest` и `assert`.

Разработка через тестирование. TDD



Практическое задание

Практическое задание

1. Для всех функций из урока 3 написать тесты с использованием **unittest**. Они должны быть оформлены в отдельных скриптах с префиксом **test_** в имени файла (например, **test_client.py**).
2. * Написать тесты для практических работ из курса «Python 1».

Дополнительные материалы

1. Тестирование. Начало: https://habr.com/post/121162/.
2. Юнит-тесты. Первый шаг к качеству: https://habr.com/post/336030/.
3. Зачем нужны юнит-тесты: https://tproger.ru/translations/unit-tests-purposes/.
4. Тестирование по-пайтоновски. Введение:

   https://dou.ua/lenta/articles/nose-tests-intro/.
5. Test-Driven Development в Python для начинающих. Часть первая:

https://shepetko.com/ru/blog/beginning-test-driven-development-in-python-1.