05. Домашни дискусии

25 октомври 2022

Не такива домашни дискусии

А такива домашни дискусии

Като генерално правило, освен ако не сме посочили изрично, няма нужда да предвиждате невалиден инпут.
С други думи няма да получавате input `['qwerty', (54, 32, 12), 'yellow', {'color': 'ffffc0'}]` и прочие извращения. Или какъвто и да е други невалидни входни данни...

```
    return "Please enter valid arguments!"
if len(starting_point) != 2:
    return "Please enter valid coordinates of the starting point!"

possible_directions = ['FFFFFF', '000000', '00C000', 'FFC0C0', 'C0FFC0', 'C00000',
                       'FFFFC0', '0000C0', 'C0C000', 'C0C0FF']

for i in range(0, len(directions)):
```

Ако ще обхождаш колекция, обхождай елементите, а не техните индекси. Овен, разбира се, ако това не е по-удобно, но тук не е. `for direction in directions:...`

```
for direction in directions:
    if direction == 'FFFFFF':
        continue
    elif direction == '000000':
        return moving_point[0], moving_point[1]
```

Един `break` тук вместо този ред ще ти спести повторението на код между текущия и последния ред.

Основни (положителни) изводи

- Вече знаете как да качвате домашни си
- Вече имаме Python 3.10, който да ги проверява
- В момента на написването на този слайд имаме 70 решения
- Успяхме да дадем обратна връзка за всяко едно от тях
- Резултатите от тестовете също са обещаващи (само 8 човека нямат пълен брой точки)
- Научихме много един от друг по време на този процес

И все пак има неща, които искаме да обсъдим

- синтактични излишъци
- алгоритмични излишъци
- проблеми с форматиране на кода
- пропуски при проверка на качеството
- оптимизации на кода
- лоши практики
- бонус точки за активност
- Duck typing > type hinting

Синтактични излишъци (1)

Кое е излишното?

```
some_tuple = (x, y)

some_tuple = x, y
```

Синтактични излишъци (2)

Кое е излишното?

```
def calculate_final_vector(init, hexes):
    x = init[0]
    y = init[1]

def calculate_final_vector(init, hexes):
    x, y = init
```

Синтактични излишъци (3)

Кое е излишното?

```
hash_map = dict({‘FF0000’: (0, 1),
                 ‘C0C000’: (1, 0)
                 ...})

hash_map = {‘FF0000’: (0, 1),
            ‘C0C000’: (1, 0)
            ...}
```

Синтактични излишъци (4)

Кое е излишното?

```
def some_fun(input):
    ...
    result = (a, b)
    return result

def some_fun(input):
    ...
    return a, b
```

Синтактични излишъци - защо да ми пука?

- По-четим код
- Спестяване на излишни операции - т.е. на време и място
- Спестяване на време за писане
- По-дълбоко разбиране на езика
- PEP8
- Останалите програмисти ви обичат
- Ние предупредихме още в нулевата лекция: Grammar Nazis

Алгоритмични излишъци (1)

Кое е излишното?

```
for hex_ in range(0, len(hexes)):
    process_hex(hexes[hex_])

for hex_ in hexes:
    process_hex(hex_)
```

Алгоритмични излишъци (2)

Кое е излишното?

```
for color in colors:
    if color == ‘C0FFC0’:
        x += -1
    elif color == ‘FFFFFF’:
        continue
    elif color == ‘000000’:
        break
```

```
for color in colors:
    if color == ‘C0FFC0’:
        x += -1
    elif color == ‘000000’:
        break
```

Алгоритмични излишъци (3)

Кое е излишното?

```
colors = list(map(str.lower, hexes))

for color in colors:
    <do some stuff>
```

```
colors = map(str.lower, hexes)

for color in colors:
    <do some stuff>
```

Алгоритмични излишъци - защо да ми пука?

- По-четим код
- Спестяване на излишни операции - т.е. на време и място
- Спестяване на време за писане
- По-дълбоко разбиране на езика
- Останалите програмисти ви обичат

Проблеми с форматиране на кода (1)

Къде е проблемът?

```
hexes = {‘F00000’: (1,2)}


hexes = {‘F00000’: (1, 2)}
```

Проблеми с форматиране на кода (2)

Къде е проблемът?

```
init=(1,2)
```

```
init = (1, 2)
```

Проблеми с форматиране на кода (3)

Къде е проблемът?

```
hexes = {'000000': (0, 0)}
def calculate_final_vector(init, hexes):
    <do some stuff>
```

```
hexes = {'000000': (0, 0)}

def calculate_final_vector(init, hexes):
    <do some stuff>
```

Проблеми с форматиране на кода - защо да ми пука?

- По-четим код
- По-дълбоко разбиране на езика
- PEP8
- Останалите програмисти ви обичат

Пропуски при проверка на качеството (1)

Къде е проблемът?

```
def calculcate_final_vector(init, hexes):
    x, y = init
    for hex_ in hexes:
        <do some stuff with x and y>
    print(x, y)

def calculcate_final_vector(init, hexes):
    x, y = init
    for hex_ in hexes:
        <do some stuff with x and y>
    return x, y
```

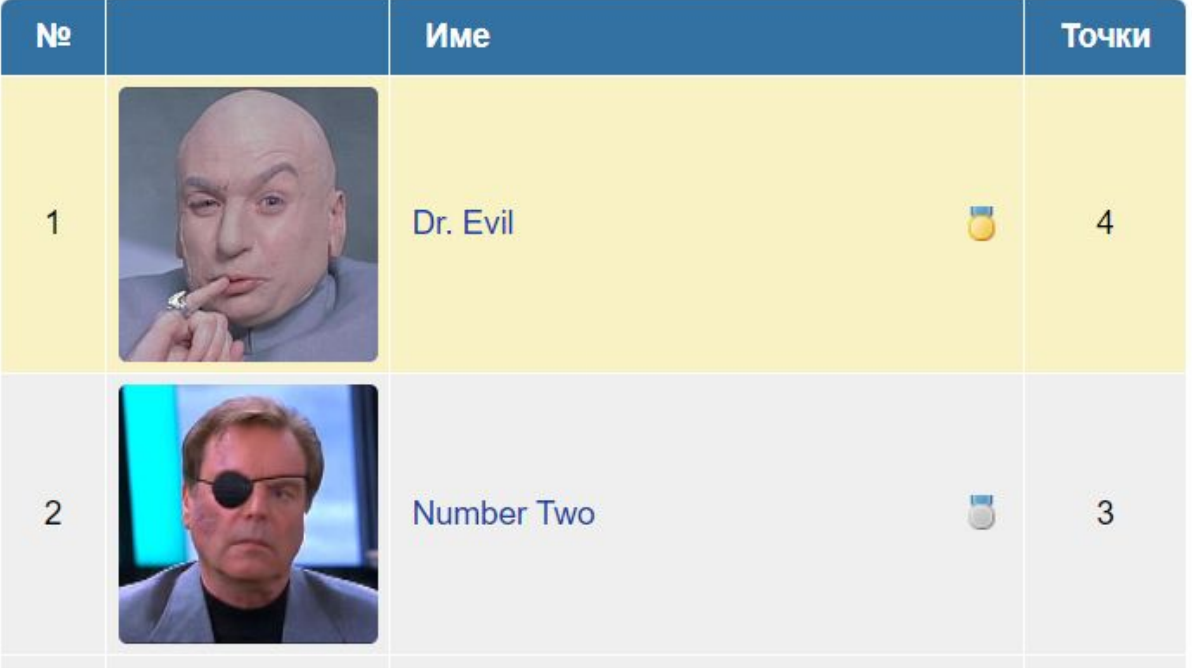
Пропуски при проверка на качеството (2)

Къде е проблемът?

```
def calculcate_final_vector(init, hexes):
    x, y = init
    for hex_ in hexes:
        <do some stuff with x and y>
    return [x, y]

def calculcate_final_vector(init, hexes):
    x, y = init
    for hex_ in hexes:
        <do some stuff with x and y>
    return x, y
```

Пропуски при проверка на качеството - защо да ми пука?

Класация



№		Име	Точки
1		Dr. Evil	4
2		Number Two	3

Оптимизации на кода /я синтактични, я алгоритмични/ (1)

Къде е проблемът?

```
if hex_ == ‘F00000’ or hex_ == ‘C0C000’:
    <do some stuff>

if hex_ in (‘F00000’, ‘C0C000’):
    <do some stuff>
```

Оптимизации на кода /я синтактични, я алгоритмични/ (2)

Къде е проблемът?

```
hexes = {'F00000': (-1, 0), 'C0C000': (1, 0), ...}

for hex_ in hexes:
    if my_hex == hex_:
        <do some stuff with hexes[hex_]>
```

```
hexes = {'F00000': (-1, 0), 'C0C000': (1, 0), ...}

<do some stuff with hexes[my_hex]>
```

Оптимизации на кода /я синтактични, я алгоритмични/ (3)

Къде е проблемът?

```
for hex_ in hexes:
    if hex_.lower() == ‘F00000’:
        x += 1
    if hex_.lower() == ‘C0C000’:
        x -= 1
    ...

for hex_ in hexes:
    lower_hex = hex_.lower()
    if lower_hex == ‘F00000’:
        x += 1
    elif lower_hex == ‘C0C000’:
        x -= 1
    ...
```

Оптимизации на кода - защо да ми пука?

- По-четим код
- Спестяване на излишни операции - т.е. на време и място
- Спестяване на време за писане
- По-дълбоко разбиране на езика
- Останалите програмисти ви обичат

Лоши практики (1)

Къде е проблемът?

```
# This is a comment, explaining the next few lines
# but I have no space to write it in a single line
# so I need multiple lines.

“““
This is a comment, explaining the next few lines
but I have no space to write it in a single line
so I need multiple lines.
“““
```

Лоши практики (2)

Къде е проблемът?

```
def calculate_final_vector(init, hexes):
    if not isinstance(init, vector):
        raise Exception('Invalid input.')
    <do some stuff>

def calculate_final_vector(init, hexes):
    <do some stuff>
```

Лоши практики (3)

Къде е проблемът?

```
for hex_ in hexes:
    hex_ = hex_.lower()
    <do some stuff with hex_>

for hex_ in hexes:
    lower_hex = hex_.lower()
    <do some stuff with lower_hex>
```

Лоши практики (4)

Къде е проблемът?

```
colorHex = ‘FF0000’

color_hex = ‘FF0000’ # Python is a snake. Python likes snake_case

# *sometimes PascalCase
```

Лоши практики (4+)

Къде е проблемът?

```
a = ‘FF0000’

color_hex = ‘FF0000’
```

Лоши практики (4++)

Къде е проблемът?

```
def calculate_final_vector(vec, lst):
    <do some stuff>

def calculate_final_vector(vector, colors):
    <do some stuff>
```

Лоши практики (4++)

Къде е проблемът?

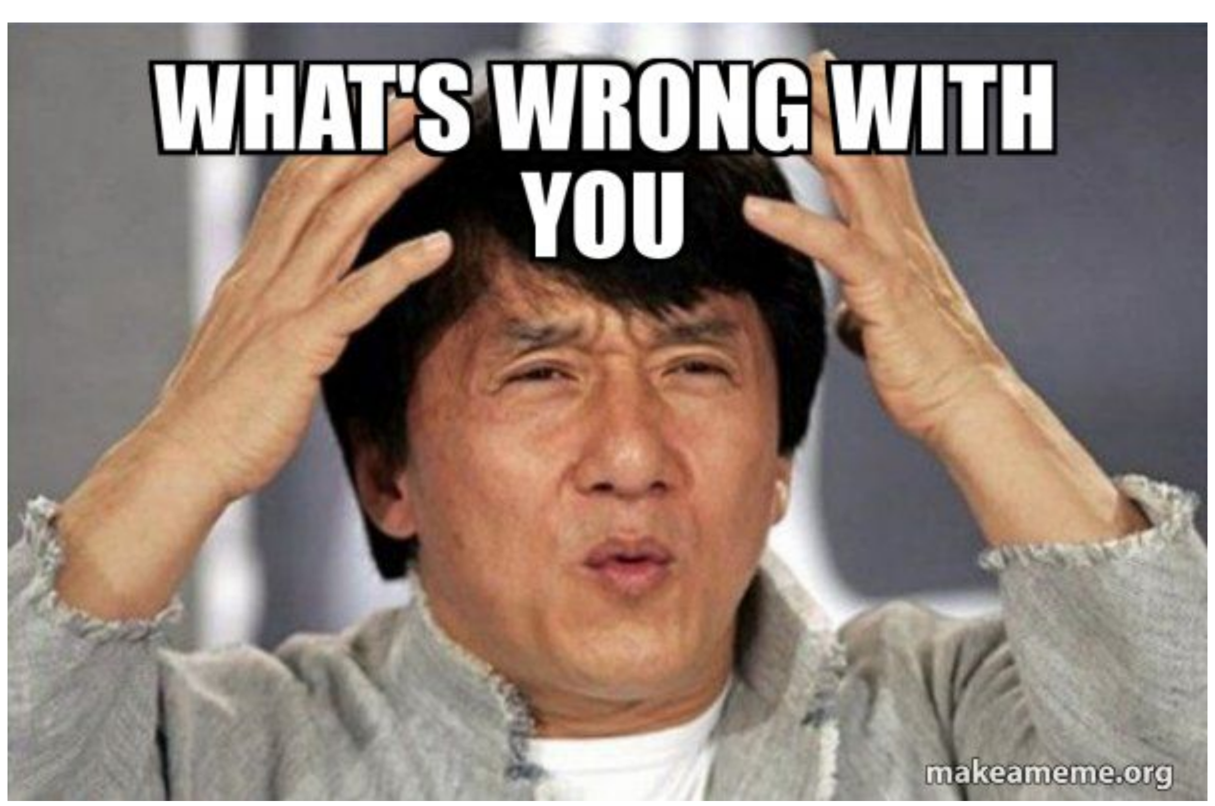
```
def calculate_final_vector(args, hexes):
    <do some stuff>


def calculate_final_vector(starting_coordinates, hexes):
    <do some stuff>
```

Лоши практики (4+++)

Къде е проблемът?

```
for list in [[1, 2, 3], [4, 5, 6], [7, 8, 9]]:
    <do some stuff with list>

list((1, 2, 3))
Traceback (most recent call last):
File "<pyshell#30>", line 1, in <module>
    list(1, 2, 3)
TypeError: 'int' object is not callable

for row in [[1, 2, 3], [4, 5, 6], [7, 8, 9]]:
    <do some stuff with row>
```

Лоши практики - защо да ми пука?



Бонус точки за активност /напомняне/

https://bitbucket.org/stoilov2000/workspace/snippets/qkz6z4 относно коментара ми

Публикувано преди около 9 часа Редактирай

Махни звездичката

Duck typing > type hinting

Duck typing > type hinting (не, наистина...)

```
for hex_ in hexes:
    <do some stuff with hex_>
```

Наистина ли смятате, че трябва да сме стриктни относно типа на hexes?

И ние сме хора.

И ний сме дали нещо на света,
на вси Питонджии 1-liners да четат...

```python
def calculate_final_vector(init_vector, hexes): return tuple(map(sum,
zip(*([init_vector] + list(map(lambda x: {'ffc0c0': (1, 0), 'ffffc0': (0, -1),
'c0ffc0': (-1, 0), 'c0c0ff': (0, 1), 'c00000': (-1, 0), 'c0c000': (0, 1),
'00c000': (1, 0), '0000c0': (0, -1), 'ffffff': (0, 0)}[x], map(str.lower,
hexes[:(hexes[:] + ['000000']).index('000000')])))))))
```

...и да реват

```
def calculate_final_vector(init, hexes): return
tuple(map(sum,zip(init,tuple(map(sum,(zip(*[(((((-1)**((_&2)//2+1))*((-1)**(int(
'f' in __))))+(int(_==15)), 0)[::(-1)**((_%(5+(_&8)))&1)] for hex in
hexes[:(hexes+['000000']).index('000000')] for __ in (hex.lower(),) for _ in
(int(str(int(('ff'*('f' in
__)+__)[::2].replace('f','2').replace('c','1'))-1111*((sum(ord(c) for c in
__)&32)//32)), base=2),)])))) or (0,0))))
```

05 ½. Автоматизирано тестване

25 октомври 2022

За какво няма да си говорим

- За quality assurance
- За acceptance testing
- За тестове тип “бенчмарк”
- За сомелиерство на ракия

Митът

- Проектът идва с готово, подробно задание.
- Прави се дизайн.
- С него работата се разбива на малки задачи.
- Те се извършват последователно.
- За всяка от тях пишете кода.
- Разцъквам го малко - няколко print-а, малко пробване в main метода/функцията и толкова.
- Profit.

Реалността

- Това в началото с ясните изисквания и дизайн е утопия.
- Писането на код е сложна задача - допускат се грешки.
- Програмистите са хора - допускат грешки.
- Промяната на модул в единия край на системата като нищо може да счупи модул в другия край на системата.
- Идва по-добра идея за реализация на кода.
- Често се налага един код да се преработва.

Как да автоматизираме

- За всичко съмнително ще пишем **сценарий**, който да "цъка".
- Всеки сценарий ще изпълнява кода и ще прави няколко **твърдения** за резултатите.
- Сценариите ще бъдат обединени в **групи**.
- Пускате всички тестове с едно бутонче.
- Резултатът е "Всичко мина успешно" или "Твърдения X, Y и Z в сценарии A, B и C се оказаха неверни".

Например

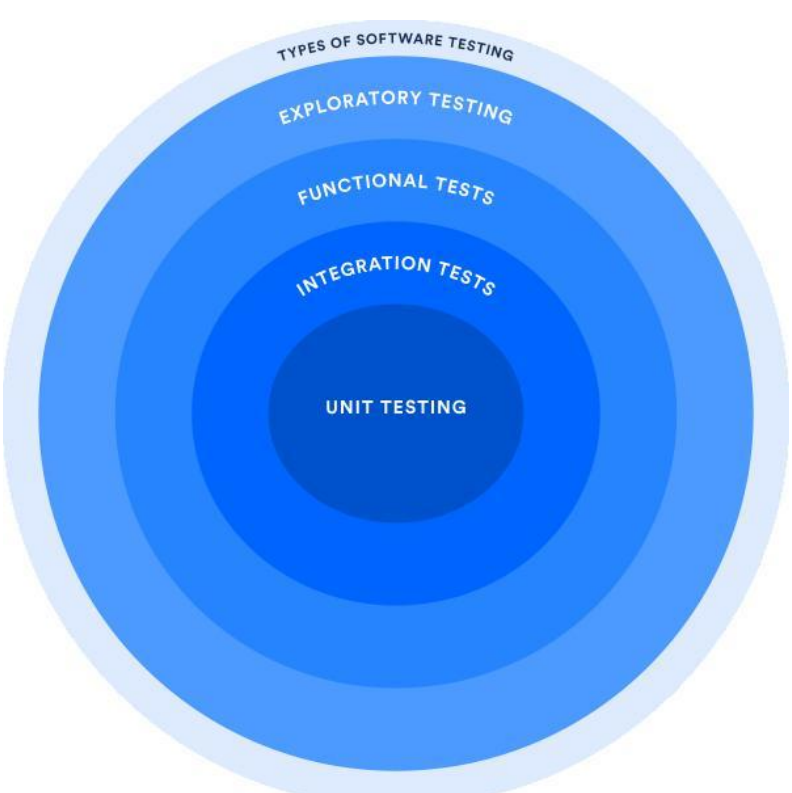
```
======================================================================
FAIL: test_white_ignored (test.TestCalculateFinalVector)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/storage/deedee/data/rails/pyfmi-2022/releases/20221020151654/lib/language/python/runner.py", line 67, in thread
    raise result
AssertionError: Tuples differ: (0, 0) != (4, 0)

First differing element 0:
0
4

- (0, 0)
?  ^

+ (4, 0)
?  ^


----------------------------------------------------------------------
Ran 8 tests in 0.092s

FAILED (failures=6)
```

За какво ни помагат тестовете

- Откриват грешки по-рано.
- Позволяват ни уверено да правим промени в системата.
- Дават сигурност на клиенти, шефове и програмисти.
- Представляват пример как се работи с кода.
- Служат като документация и спецификация.

Типове тестове

- При преминаване към по-външните кръгове на *(ада)* тестването започваме да работим в условия, по-близки до реалните
- **Unit** тестовете трябва да се концентрират върху изолирана *(малка)* част от функционалността и да верифицират правилната ѝ работа



Имаме 4 фази на тестване

- Setup - конфигурираме “контекста” на тестването
- Execute - реалното изпълнение на тестваното
- Verify - оценяваме дали крайният резултат е очакваният
- Teardown - връщаме “контекста”, променен по време на “Setup” фазата в начално състояние

По-конкретно?

1. import unittest
2. unittest.TestCase
3. setUp / tearDown
4. assert*
5. Всеки тест трябва да е напълно независим от останалите
6. **unittest.main()** = магия

```python
import unittest

class TestStringMethods(unittest.TestCase):

    def setUp(self):
        self.value = 'hmmmm'
        print(self.value)

    def tearDown(self):
        del self.value

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])

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

Как се пишат тестове?

- **import unittest**
- **unittest.TestCase**
- assertTrue(expression)
- assertFalse(expression)
- assertEqual(expected, actual)
- assertNotEqual(expected, actual)
- assertIs(expected, actual)
- assertIsNot(expected, actual)
- assertIsNone(expression)
- assertIsNotNone(expression)
- assertIn(element, collection)
- assertNotIn(element, collection)
- assertIsInstance(object, type)
- assertNotIsInstance(object, type)
- И много други...

Само unittest?

- Добре де, има и друг вариант...
- **import pytest**
- Има разлики между двете
- Няма да задълбаваме в разликите, и двете са мощни и двете се използват масово

Кога да тестваме?

A. Никога
B. След като напишем функционалността
C. Преди да напишем функционалността
D. Когато ни е скучно и нямаме какво да правим

Според Test-Driven Development парадигмата, отговорът е:

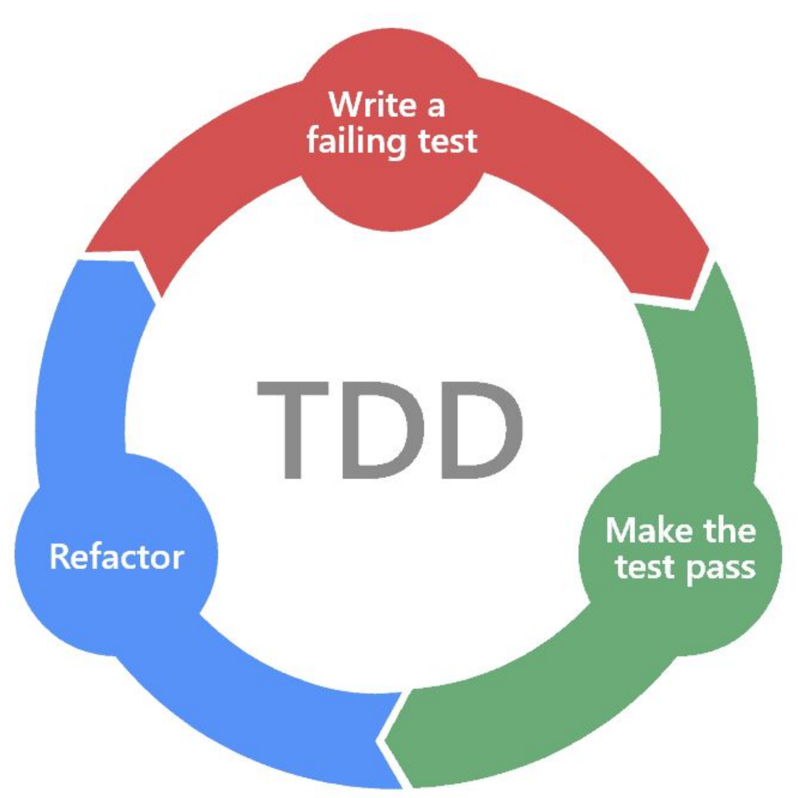
C. Преди да напишем функционалността

В зависимост от това колко точно скучно ни е, може и D.

TDD

- Test-Driven Development is not about testing.
- Подход за писане на код.
- Дизайнът е базиран върху обратна връзка, не гадаене.
- Спестява излишен код - пишете само каквото ви трябва.
- Спестява излишна функционалност.

TDD (наглядно)



TDD (още по-нагледно)

Добри практики при писане на тестове

- Пишете тестове за всичко, което може да се счупи.
- Не пишете тестове, които **винаги** ще минават, дори и да счупите целия codebase.
- Добър начин да си мислите за тестовете е като requirements.
- Не тествайте елементарен код.
- Успехът на тестовете не трябва да зависи от реда им.
- Тествайте гранични случаи!
- Не правете тестовете зависими един от друг.

Welcome to the real world...

“Абе аз съм чувал за едно нещо наречено BDD”

Може би по-нататък, засега нека да се задоволим с вече обсъденото.

Въпроси?