

VII Encontro



Coding Dojo e Test Driven Development

Brunno Gomes



twitter.com/brunnogomes
brunnolg@gmail.com



Rodrigo Alves Vieira

rodrigo3n.com

twitter.com/rodrigo3n

rodrigo3n@gmail.com



Coding Dojo



Test Driven Development



Coding Dojo?



flickr{rosie_hardy}



Test Driven Development





Técnica apresentada por
Kent Beck em seu livro (Test
Driven Development: By
Example, 2003), tem como
objetivo aumentar a
qualidade do software
escrito além das
capacidades do
programador



Hmmm, TDD?!..



...Python tem o...



unittest



unittest



flickr{shutterhack}

unittest

A close-up profile of a brown monkey's head, facing left. The monkey's mouth is open, revealing its teeth and tongue. The background is a soft, out-of-focus yellow-green.

- O ***unittest*** (também chamado de PyUnit) é um Framework built-in do Python para Testes Unitários criado por Steve Purcell em 2001. Baseado no JUnit e no Smalltalk Testing Framework
- Está incluso na biblioteca padrão do Python desde a versão 2.1 (2001).



Vantagens

- É uma biblioteca padrão do Python



Vantagens

- É uma biblioteca padrão do Python
- É muito prática. Curva mínima de aprendizado



Vantagens

- É uma biblioteca padrão do Python
- É muito prática. Curva mínima de aprendizado
- Presente em **todos** os interpretadores do Python



Vantagens

- É uma biblioteca padrão do Python
- É muito prática. Curva mínima de aprendizado
- Presente em **todos** os interpretadores do Python
- É útil pra **qualquer projeto!**



Qualquer
projeto!



O Framework

- TestCase



○ Framework

- TestCase
- TestSuite



○ Framework

- TestCase
- TestSuite
- TextTestRunner



**Comece testando o
código que você
quer ter!**



unittest.TestCase

```
# -*- encoding:utf-8 -*-
# aviao_teste.py

import unittest
from aviao import Aviao

class AviaoTeste(unittest.TestCase):
    """
    Documentação de AviaoTest
    """
    def testeAviaoCriado(self):
        aviao = Aviao(10)
        self.assertNotEqual(aviao == None, aviao)

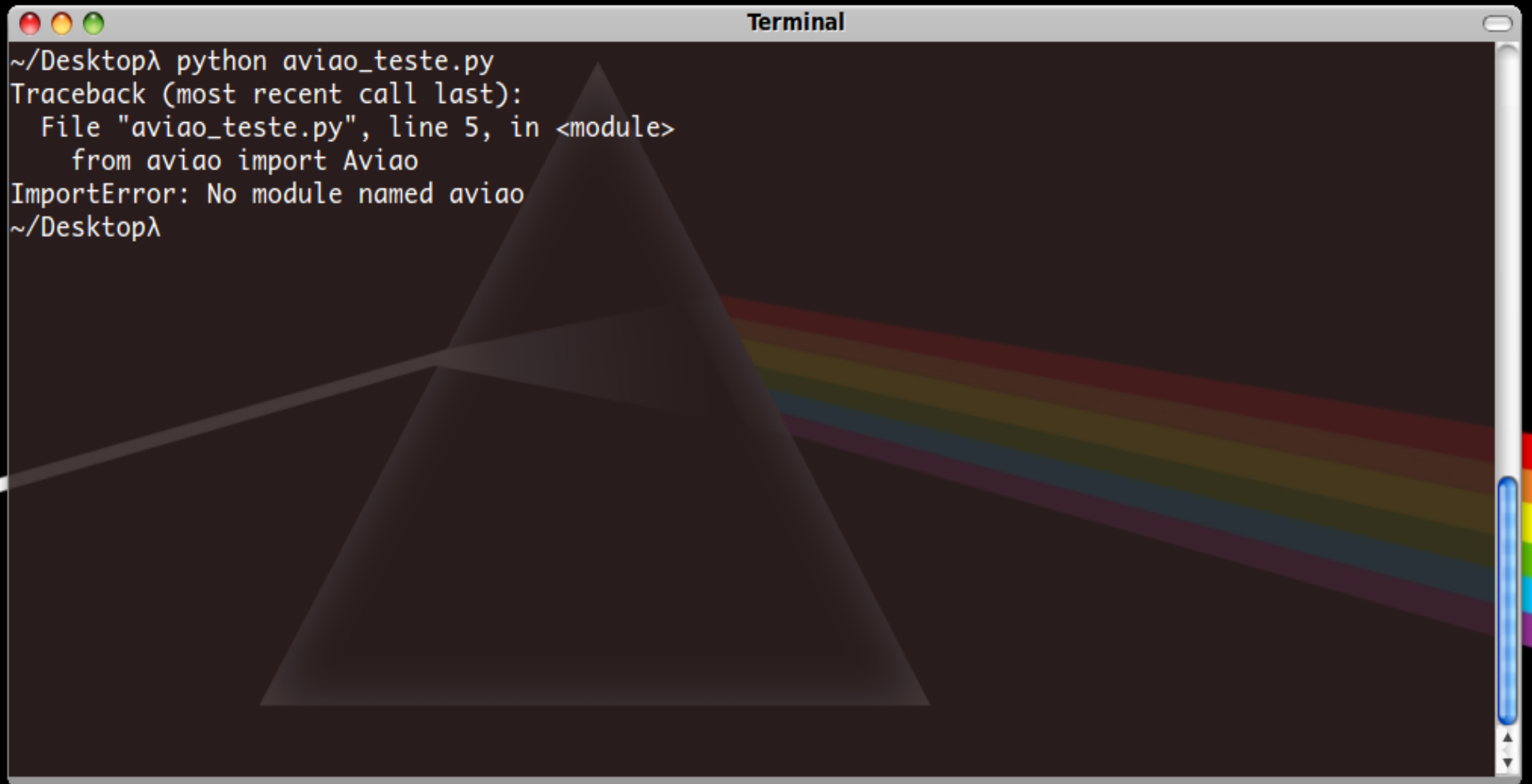
if __name__ == "__main__":
    unittest.main()
```



flickr{saguar}



Obviamente falha!

A terminal window titled "Terminal" with a dark background. The window contains text showing a Python script execution and an error. In the background, there is a faint illustration of a pyramid with a rainbow emanating from its base. The terminal text is as follows:

```
~/Desktopλ python aviao_teste.py
Traceback (most recent call last):
  File "aviao_teste.py", line 5, in <module>
    from aviao import Aviao
ImportError: No module named aviao
~/Desktopλ
```



Completamos o primeiro passo do ciclo do TDD: escrevemos um teste como queremos que o código funcione.

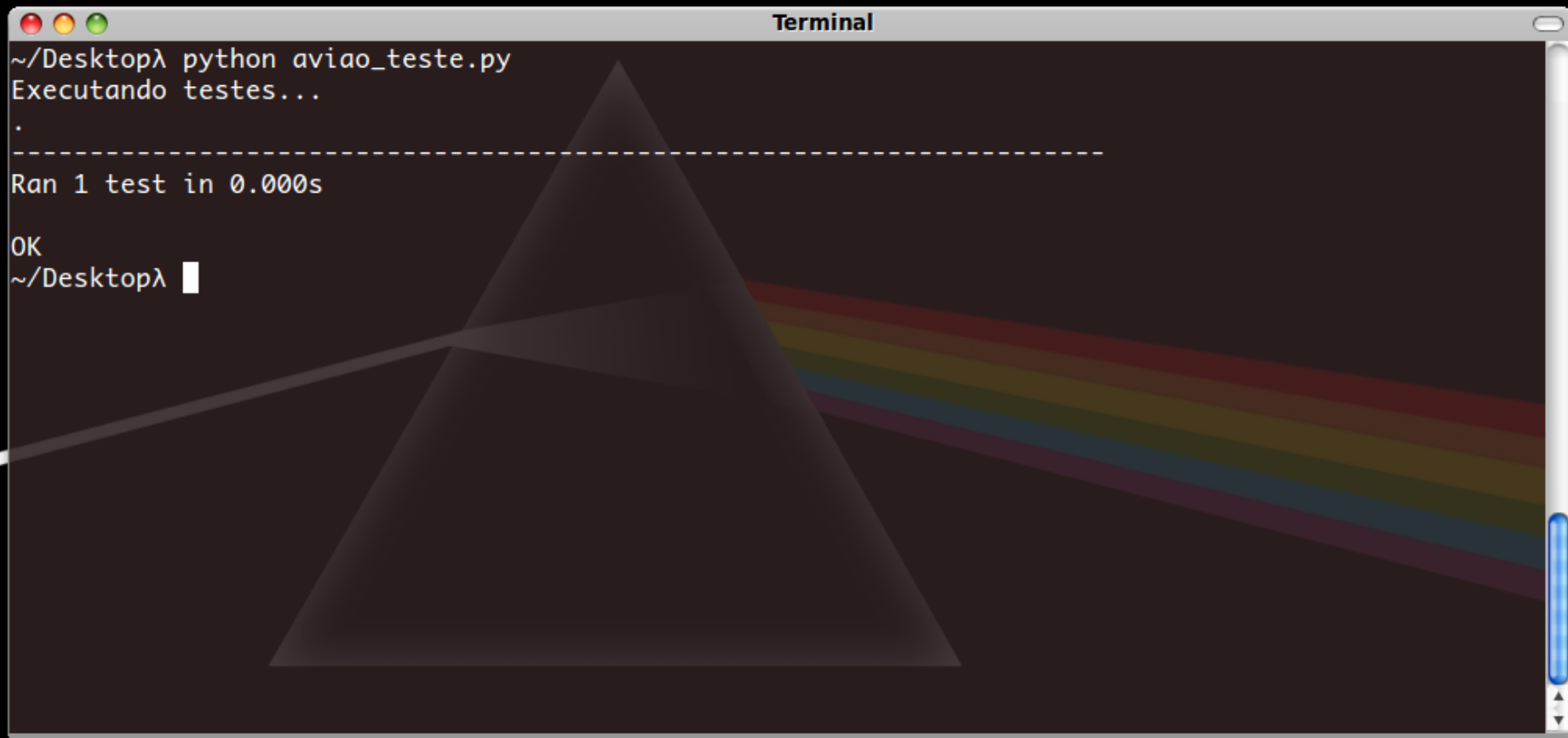
Agora escreveremos apenas o código suficiente para fazer o teste passar!



```
# -*- encoding:utf-8 -*-  
# aviao.py
```

```
tanque = 10
```

```
class Aviao():  
    def __init__(self, encher_o_tanque=tanque):  
        print "Executando testes..."
```



A screenshot of a macOS Terminal window titled "Terminal". The window has a dark gray background and a light gray title bar with standard macOS window controls (red, yellow, green buttons). The terminal output shows the execution of a Python script: `~/Desktopλ python aviao_teste.py`, followed by `Executando testes...`, a single dot `.`, a dashed line `-----`, `Ran 1 test in 0.000s`, `OK`, and the prompt `~/Desktopλ` with a cursor. A large, semi-transparent, stylized graphic of a pyramid with a rainbow-colored base is overlaid on the terminal window. The pyramid is dark gray, and its base is composed of several horizontal bands of color: red, orange, yellow, green, blue, and purple. The pyramid is positioned in the center-left of the terminal window, with its apex pointing towards the top-left corner.

```
~/Desktopλ python aviao_teste.py  
Executando testes...  
.  
-----  
Ran 1 test in 0.000s  
OK  
~/Desktopλ
```


flickr{naty_nina}

Fizemos o primeiro teste passar!

Agora vem a...



`flickr{finsterbaby}`



...Refatoração!

Aqui pensamos
atenciosamente no que os
testes estão fazendo.
Melhoramos a qualidade do
código fonte e do próprio
teste e removemos
duplicação



```
# -*- encoding:utf-8 -*-
# aviao_teste.py

import unittest
from aviao import Aviao

class AviaoTeste(unittest.TestCase):
    """
    Documentação de AviaoTest
    """
    def testeAviaoCriado(self):
        aviao = Aviao(10)
        self.assertNotEqual(aviao == None, aviao), \
            "Avião não pode ser None"

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



Pro-dica: Execute os testes com a opção '-v' e veja a execução de cada teste em modo verboso!

```
~λ python aviao_teste.py -v
```



unittest.TestSuite

Uma ferramenta do unittest para agrupar testes individuais e organizar '**pilhas**' de testes, mesmo em diferentes arquivos/módulos, criando **Suítes de Teste!**



Criando uma suíte de Testes

```
# -*- encoding:utf-8 -*-  
# aviao_test_suite.py  
  
import unittest  
import aviao_teste  
  
def suite():  
    testsuite = unittest.TestSuite()  
    testsuite.addTest(unittest.makeSuite(AviaoTeste))  
    return testsuite
```

Ou ainda melhor...



A classe **TestSuite** torna ainda mais poderosa sua suíte de testes porque você pode importar quantos módulos quiser contendo uma **quantidade qualquer de testes!**

Então você pode **aninhar os testes** para **dinamizar a execução!**



E com 'aninhar' eu quis dizer que você pode aninhar até outras suítes de testes!

```
import unittest
from aviao_test_suite import AviaoTestSuite
from outro_modulo import OutraTestSuite

suite1 = aviao_test_suite.AviaoTestSuite
suite2 = outro_modulo.OutraTestSuite

teste_geral = unittest.TestSuite((suite1, suite2))
```

Uma suíte de testes que executa outra suíte de testes! Massa!



`unittest.TextTestRunner`

Claro que testes são
documentação também,
então, nada melhor que
tê-los disponíveis em
texto puro!



E é exatamente isso que o TextTestRunner faz cada vez que o invocamos com "unittest.main()" no arquivo aviao_teste.py!

unittest.main() gera um objeto TestSuite que contém todos os testes(métodos) que começam com "test" (`testAviaoCriado`, por exemplo) , então ele invoca o TextTestRunner que executa cada um dos testes e te retorna o resultado via *stderr!*



Nossa suíte final de testes!

```
# -*- encoding:utf-8 -*-
# aviao_teste_suite.py

import unittest
from aviao_test import *

class AviaoTesteSuite(unittest.TestSuite):
    def __init__(self):
        unittest.TestSuite.__init__(self.map(AviaoTeste, \
            ("AviaoTeste")))

    def suite(self):
        suite = unittest.TestSuite()
        suite.addTest(unittest.makeSuite(AviaoTeste))
        return suite

suite1 = unittest.TestSuite()
suite1.addTest(AviaoTeste("testeAviaoCriado"))

unittest.TextTestRunner().run(suite1)
unittest.TextTestRunner(verbosity=2).run(suite())
```



mais informações:

`help` (`unittest`)



flickr{ibcbulk}

A close-up, slightly blurred photograph of a person riding a bright blue bicycle. The rider's hands are on the chrome handlebars, and their legs are visible in blue jeans. The bicycle is moving on a dark asphalt road with white lane markings. The text "Fazer TDD é como andar de bicicleta!" is overlaid in large white letters on the left side of the image.

**Fazer TDD é
como andar de
bicicleta!**

Coding Dojo

Porque ?

**Nós não
treinamos.**

O que é ?

De acordo com o CodingDojo.Org

“Um encontro onde um grupo de programadores se junta para trabalhar num desafio de programação. O objetivo é se divertir praticar deliberadamente de forma a melhorar suas habilidades.”

Prática **Deliberada.**

Nãõ é...



foto: armandelli @ Flickr



foto: armandelli @ F

...um lugar para
pura **exibição.**





... **competição.**

Características



foto: little-wings @ Flickr

- **Passos de bebê**
- **Todos são iguais**
- **Todos devem entender**
- **Sempre começa do zero**

- **Sempre se usa testes**
- **Iterativo e Interativo**
- **Interrupções incentivadas**
- **Abertura para novas idéias**

Algumas regras

- Computador + Projetor
- Piloto + co-piloto
- TDD

vermelho → **verde** → **refatorar**

Estilos

PreparedKata

- **Piloto e co-piloto fixos**
- **Apresentam uma solução do início ao fim**
- **Cada passo é explicado**

- **Indicado para um grande número de participantes**
- **Pode-se usar um problema e solução previamente preparados**

RandoriKata

- **Piloto e co-piloto revezam**
- **Todos os presentes são convidados a participar**
- **Cada par tem um tempo para programar**

- **Indicado para grupos menores**
- **O ideal é que todos os participantes programem**

Problemas e Soluções

- **Problemas simples**
- **Qualquer um pode propor**
- **Tem que começar e terminar na mesma sessão do Dojo**

Depois do Dojo

- **O que aprendemos ?**
- **O que foi bom ?**
- **O que foi ruim ?**

Valeu! \o/

