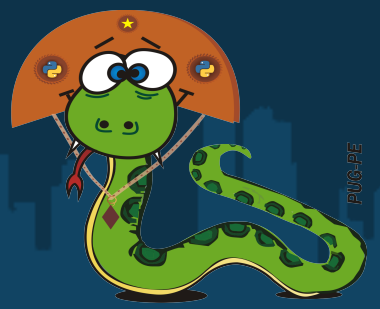
Python com TDD

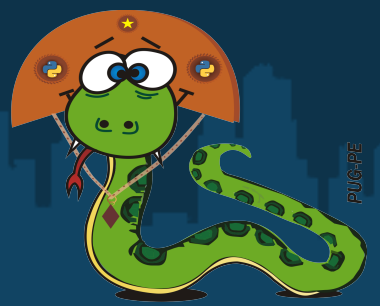
Código Python testado Pragmaticamente

IV Encontro do PUG-PE



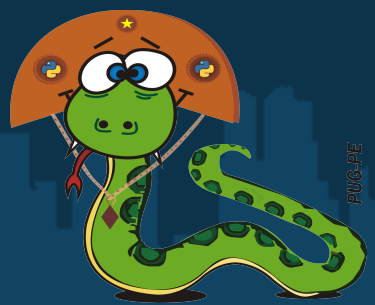
Sobre Rodrigo Alves Vieira

* Recife, Pernambuco
* Programador Ruby há mais de um ano
* Pythonista fase beta
* Ama Rock 'n roll, Indie e Blues :D
* rodrigo3n.heroku.com
* github.com/rodrigo3n
* @rodrigo3n



Começo de conversa...

Programadores de qualquer linguagem precisam de Frameworks e Suítes de Testes Automatizados (De preferência!) , pra **assegurar** o funcionamento apropriado do código, seja ele de um aplicativo em produção ou uma biblioteca Open Source.



O Python tem o:

unittest

O **unittest** - também chamado de **PyUnit** - é um Framework built-in do Python de Testes Unitários, baseado no JUnit(Java) de Erich Gamma e no Smalltalk Testing Framework do Kent Beck(O cara que idealizou o TDD em 1999).

Criado em 2001 por Steve Purcell está com o Python desde a versão 2.1 em sua biblioteca padrão.

Vantagens

* É uma biblioteca padrão do Python.
* É muito prática. Curva mínima de aprendizado.
* Está presente em **todas** as implementações da Linguagem.
* É útil pra qualquer projeto Python!

Composto por 3 classes principais

* TestCase
* TestSuite
* TextTestRunner

Pilares do Framework

* **TestCase**

  Providencia métodos próprios (assertEqual, assertTrue, assertNotEqual) que geram mensagens mais precisas pra os testes.

* TestSuite
* TextTestRunner

Pilares do Framework

* TestCase
* **TestSuite**

  É um agregador de testes, classe poderosa que pode mesclar vários testes de diferentes módulos e executá-los como um todo.

* TextTestRunner

Pilares do Framework

* TestCase
* TestSuite
* **TextTestRunner**

Mostra os nomes dos testes conforme eles são executados, assim como os erros ocorridos. E um resumo dos resultados no fim da execução

Comece testando o código que você quer ter!

unittest.TestCase

```python
# -*- encoding: utf-8 -*-
# test_aviao.py

import unittest
from aviao import Aviao

class testAviao(unittest.TestCase):
    """
    Documentação de testAviao
    """

    def testAviaoCriado(self):
        aviao = Aviao(10)
        self.assertNotEqual(aviao == None, aviao)

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

que obviamente....

Falha!

```
~/code% 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
~/code%
```

Completamos o primeiro passo do ciclo do TDD: ***Escrevemos um teste que especifica como queremos o código funcionando.***

Agora escreveremos apenas código suficiente pra fazer o código passar!

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

tanque = [0]

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

Ready.... Go!

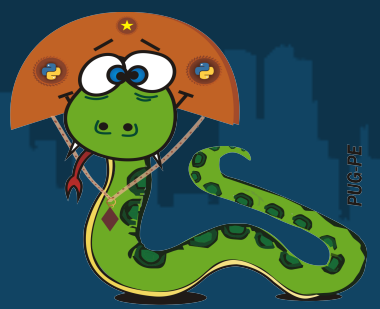
```
~/code/aviao% python test_aviao.py
Executando testes...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
~/code/aviao%
```

Fizemos o primeiro teste passar...

Agora vem a....

Refatoração

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



```python
# -*- encoding: utf-8 -*-
# test_aviao.py

import unittest
from aviao import Aviao

class testAviao(unittest.TestCase):
    """
    Documentação de testAviao
    """

    def testAviaoCriado(self):
        aviao = Aviao(0)
        #self.assertEqual(aviao == None, aviao), "Não!!!"
        self.assertNotEqual(aviao == None, aviao), \
                "Aviao não pode ser None"

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

Dica!

Execute os testes com o argumento "-v" e veja o resultado individualizado de cada teste!

```
~% python teste_aviao.py -v
```

Dica!

Execute testes específicos interativamente!

Digamos que o arquivo que contém os testes (test_aviao.py) tem 40 testes, mas você só quer executar 2 deles, então:

```
~% python test_aviao.py testeAviao.test_aviao_criado testeAviao.test_motor
```

unittest.TestSuite

Uma ferramenta incrível pra agrupar testes individuais e organizar "pilhas" de testes, mesmo em diferentes arquivos e módulos, criando **Suites de Teste**.

Criando uma suíte de testes

```
# -*- encoding: utf-8 -*-
# test_suite.py
import unittest
import test_aviao

def suite():
    testSuite = unittest.TestSuite()
    testSuite.addTest(unittest.makeSuite(testAviao))
    testSuite.addTest(unittest.makeSuite(testAlgo))
    return testSuite
```

Ou ainda melhor.....

```
# -*- encoding: utf-8 -*-
# test_suite.py
import unittest
import test_aviao

class AviaoTestSuite(unittest.TestSuite):
            def __init__(self):
                unittest.TestSuite.__init__(self,map(testAviao,
                                                     ("testAviao",
                                                      "testTremDePouso",
                                                      "testAeroporto")))
```

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

Então você pode aninhá-los pra dinamizar a execução...

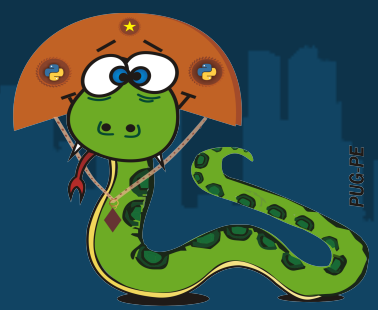
Quando eu disse aninhá-los eu disse que você pode aninhar até outras suítes de teste!

```
import unittest
from test_suite import AviaoTestSuite
from outro_modulo import outraTestSuite

suite1       = test_suite.AviaoTestSuite()
suite2       = outro_modulo.outraTestSuite()
teste_geral = unittest.TestSuite((suite1, suite2))
```

Um suíte de testes que executa outras suítes de testes.. hmmm. massa!

unittest.TextTestRunner

Naturalmente, testes **são** documentação
também, então, nada mais legal do que
tê-los disponíveis como texto puro!

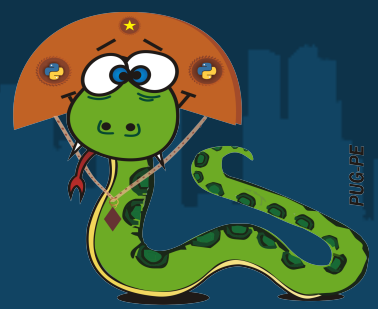


Bom, é isso que o TextTestRunner faz cada vez que o invocamos com "unittest.main()" no final do arquivo test_aviao.py!

unittest.main() gera um objeto TestSuite que contém todos os testes que começam com "test" (como testAviaoCriado() por exemplo), então ele invoca o TextTestRunner que executa cada um desses métodos e manda o resultado pra o desenvolvedor via stderr!

Nossa suíte final de testes

```
# -*- encoding: utf-8 -*-
# test_suite.py
import unittest
from test_aviao import *

class AviaoTestSuite(unittest.TestSuite):
    def __init__(self):
        unittest.TestSuite.__init__(self,map(testAviao, ("testAviao")))

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

a_suite = unittest.TestSuite()
a_suite.addTest(testAviao("testAviaoCriado"))

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



O Resultado

```
~/code/aviao% python test_suite.py
Executando testes...
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
testAviaoCriado (test_aviao.testAviao) ... Executando testes...
ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
~/code/aviao%
```

Mais informações:

help(unittest)

Isso é tudo pessoal!

Dúvidas?!
Opiniões?!

twitter: @rodrigo3n | rodrigo3n@gmail.com
http://rodrigo3n.heroku.com |
http://bit.ly/python-com-tdd