

UNITTEST – FRAMEWORK DE TESTES UNITÁRIOS

ODAIR OLIVEIRA DE SÁ - MSC

INTRODUÇÃO - TESTES AUTOMATIZADOS

São cenários (linha de código) que simulando os testes manuais, contribuem para reduzir tempo e esforço nessas verificações.

Escrever testes automatizados:

- ✓ Deixar o código mais limpo (ajuda na remoção de *code smell*)
- ✓ Garantir facilidade na manutenção do código
- ✓ Servir como documentação: de forma visual conseguimos saber quais são os cenários esperados e os tratamentos em caso de erro olhando o arquivo de teste
- ✓ Evitar trabalho manual (um teste automatizado é muito melhor do que um teste manual com print)
- ✓ Evitar bug's
- ✓ Prover *feedback* para quem está desenvolvendo a aplicação: possível saber se o programa está retornando o que é esperado mesmo alterando a lógica do programa principal.

ETAPA I - PREPARAÇÃO DO TESTE

Chamada de **fixture** e consiste nos itens necessários para um ou mais testes serem executados.

- Exemplo: para conseguir testar uma função que lê um determinado arquivo precisamos de um arquivo no nosso ambiente de teste para ser possível fazer a validação

Pode conter o uso dos métodos setUp() e tearDown(). Isso são ações que são executadas antes e depois, respectivamente, da execução de cada um dos cenários de teste.

ETAPA 2 - CASO DE TESTE

É o conjunto de cenários que queremos testar. Em um caso de teste agrupamos todos os pequenos cenários que queremos validar de forma unitária que fazem parte do mesmo contexto.

Com o framework unittest, usamos uma classe base chamada **TestCase**.

ETAPA 3 - ASSERTÇÕES

Assertões servem para validar que o cenário do seu código ocorreu como o esperado.

O **assert** é um comando built-in (nativo) do Python. E podemos usar da seguinte forma:

```
>>> assert 1+1 == 2
>>> assert 1+1 == 3, "The sum should be 2"
Traceback (most recent call last):
  File "<input>", line 1, in <module>
AssertionError: The sum should be 2
```


ETAPA 4 - TEST RUNNER

Permite rodar a execução dos testes. O test runner orquestra a execução dos testes e exibe para o usuário os resultados.

Além de utilizar o test runner do unittest outros podem ser utilizados como o pytest.

EXEMPLO – CÓDIGO

```
from unittest import main, TestCase
```

```
def square(x):
```

```
    return x ** 2
```

```
class TestSquare(TestCase):
```

```
    def test_if_returns_square_of_2(self):
```

```
        result = square(2)
```

```
        expected = 4
```

```
        self.assertEqual(result, expected)
```

```
    def test_if_returns_square_of_4(self):
```

```
        result = square(4)
```

```
        expected = 16
```

```
        self.assertEqual(result, expected)
```

```
if __name__ == '__main__':
```

```
    main()
```

FRAMEWORK DE TESTES UNITÁRIOS

UNITTEST

Originalmente inspirado no *JUnit* e possui estruturas de teste de unidades existentes em outras linguagens.

Ele suporta a automação de testes, compartilhamento de configuração e código de desligamento para testes, agregação de testes em coleções e independência dos testes do framework de relatórios.

MÓDULO *UNITTEST* SUPORTA ORIENTAÇÃO A OBJETOS

Definição de contexto de teste

Uma definição de contexto de teste representa a preparação necessária para realizar um ou mais testes, além de quaisquer ações de limpeza relacionadas. Isso pode envolver, por exemplo, criar bancos de dados proxy ou temporários, diretórios ou iniciar um processo de servidor.

MÓDULO *UNITTEST* SUPORTA ORIENTAÇÃO A OBJETOS

caso de teste

Um *test case* é uma unidade de teste individual. O mesmo verifica uma resposta específica a um determinado conjunto de entradas. O unittest fornece uma classe base, `TestCase`, que pode ser usada para criar novos casos de teste.

Suíte de Testes

Uma *test suite* é uma coleção de casos de teste, conjuntos de teste ou ambos. O mesmo é usado para agregar testes que devem ser executados juntos.

Test runner

Um *test runner* é um componente que orquestra a execução de testes e fornece o resultado para o usuário. O runner pode usar uma interface gráfica, uma interface textual ou retornar um valor especial para indicar os resultados da execução dos testes.



EXEMPLO

```
import unittest

class TestStringMethods(unittest.TestCase):

    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 = 'Testes CE-237'
        self.assertEqual(s.split(), ['Testes', 'CE-237'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

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

TEST CASE

Para criar um **testcase** basta criar uma classe que estende de `unittest.TestCase`.

Os três testes individuais são definidos com métodos cujos nomes começam com a palavra “test”.

Esta convenção na nomenclatura informa o *runner* a respeito de quais métodos são, na verdade, testes.

EXEMPLO SIMPLES – TESTE DE UMA FUNÇÃO

```
import unittest
```

```
def fun(x):  
    return x + 1
```

```
class MyTest(unittest.TestCase):  
    def test(self):  
        self.assertEqual(fun(3), 4)
```

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


EXEMPLO SIMPLES – TESTE DE UMA CLASSE

```
import unittest

class MyFun:

    def fun(self, n):
        return n + 1

class MyFunTest(unittest.TestCase):

    def testFun(self):
        obj = MyFun()
        self.assertEqual(obj.fun(3), 4)

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

ENTENDENDO O EXEMPLO

O cerne de cada teste é a invocação de um método `assertEqual()` para verificar se há um resultado esperado; `assertTrue()` ou `assertFalse()` para verificar uma condição; ou `assertRaises()` para verificar se uma exceção específica será levantada. Esses métodos são usados ao invés de utilizar a expressão `assert` para que o runner de teste possa acumular todos os resultados do teste e produzir um relatório.

ENTENDENDO O EXEMPLO

Os métodos `setUp()` e `tearDown()` permitem que se defina instruções que serão executadas antes e depois de cada método de teste.

ENTENDENDO O EXEMPLO

O bloco final mostra uma maneira simples de executar os testes. A função `unittest.main()` fornece uma interface de linha de comando para o Script de teste. Quando executado a partir da linha de comando, o Script acima produz uma saída que se parece com o próximo slide:



Pasta de C:\Users\ooliv\Documents\ITA - Doutorado\CE-237 Topicos Avancados Teste

```

29/08/2022 16:25 <DIR> .
29/08/2022 16:25 <DIR> ..
04/08/2022 10:02 <DIR> 03_Bibliografia_CE-237
18/08/2022 15:01 15.773 aula3_lab01.ipynb
18/08/2022 17:41 6.601.728 CE-704-Aula10-MI-EI-CTestes.ppt
18/06/2012 19:22 816.107 CE-704-Aula10-MI-EI-CTestes.rar
04/08/2022 15:02 <DIR> ELECTRONICS_STUDENT_2022R2_WINX64
05/04/2022 10:47 2.654.992 Getting Started with Enterprise Blockchain - A Guide to Design and Development.pdf
05/11/2014 18:21 54.428 Lab JUnit - Semana 01.pdf
18/03/2022 12:10 22.314 Polinomios.ipynb
25/08/2022 20:09 <DIR> Sem01 (04-08-2022)
11/08/2022 16:36 <DIR> Sem02 (11-08-2022)
18/08/2022 17:41 <DIR> Sem03 (19-08-2022)
25/08/2022 16:07 <DIR> Sem04 (25-08-2022)
29/08/2022 16:23 558 TestStringMethods.py
21/08/2014 16:57 1.880.467 Unit Testing with JUnit--Zimmermann--2007.pdf
29/08/2022 16:25 42.793 Unittest - framework de testes unitários.pptx
9 arquivo(s) 12.089.160 bytes
8 pasta(s) 714.514.624.512 bytes disponíveis

```

C:\Users\ooliv\Documents\ITA - Doutorado\CE-237 Topicos Avancados Teste>python -m unittest TestStringMethods.py

...

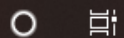
Ran 3 tests in 0.002s

OK

C:\Users\ooliv\Documents\ITA - Doutorado\CE-237 Topicos Avancados Teste>



Digite aqui para pesquisar



99%



15°C

17:49

29/08/2022



TESTES EM LINHA DE COMANDO

Passando a opção `-v` para o nosso Script de teste instruirá a função `unittest.main()` a habilitar um nível mais alto de verbosidade e produzirá a seguinte saída:

PYTHON -V -M UNITTEST TESTSTRINGMETHODS.PY

```
Prompt de Comando

C:\Users\ooliv\Documents\ITA - Doutorado\CE-237 Topicos Avancados Teste>python -v -m unittest TestStringMethods.py
import _frozen_importlib # frozen
import _imp # builtin
import _thread # <class '_frozen_importlib.BuiltinImporter'>
import _warnings # <class '_frozen_importlib.BuiltinImporter'>
import _weakref # <class '_frozen_importlib.BuiltinImporter'>
import _frozen_importlib_external # <class '_frozen_importlib.FrozenImporter'>
import nt # <class '_frozen_importlib.BuiltinImporter'>
import _io # <class '_frozen_importlib.BuiltinImporter'>
import marshal # <class '_frozen_importlib.BuiltinImporter'>
import winreg # <class '_frozen_importlib.BuiltinImporter'>
# installing zipimport hook
import time # <class '_frozen_importlib.BuiltinImporter'>
import zipimport # <class '_frozen_importlib.FrozenImporter'>
# installed zipimport hook
# C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\__pycache__\__init__.cpython-39.pyc matches C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\__init__.py
# code object from 'C:\\Users\\ooliv\\AppData\\Local\\Programs\\Python\\Python39\\lib\\encodings\\__pycache__\\__init__.cpython-39.pyc'
# C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\__pycache__\codecs.cpython-39.pyc matches C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\codecs.py
# code object from 'C:\\Users\\ooliv\\AppData\\Local\\Programs\\Python\\Python39\\lib\\__pycache__\\codecs.cpython-39.pyc'
import _codecs # <class '_frozen_importlib.BuiltinImporter'>
import codecs # <_frozen_importlib_external.SourceFileLoader object at 0x0000022A642848B0>
# C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\__pycache__\aliases.cpython-39.pyc matches C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\aliases.py
# code object from 'C:\\Users\\ooliv\\AppData\\Local\\Programs\\Python\\Python39\\lib\\encodings\\__pycache__\\aliases.cpython-39.pyc'
import encodings.aliases # <_frozen_importlib_external.SourceFileLoader object at 0x0000022A64401E50>
import encodings # <_frozen_importlib_external.SourceFileLoader object at 0x0000022A642846A0>
# C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\__pycache__\utf_8.cpython-39.pyc matches C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\utf_8.py
# code object from 'C:\\Users\\ooliv\\AppData\\Local\\Programs\\Python\\Python39\\lib\\encodings\\__pycache__\\utf_8.cpython-39.pyc'
import encodings.utf_8 # <_frozen_importlib_external.SourceFileLoader object at 0x0000022A64284AF0>
# C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\__pycache__\cp1252.cpython-39.pyc matches C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\cp1252.py
# code object from 'C:\\Users\\ooliv\\AppData\\Local\\Programs\\Python\\Python39\\lib\\encodings\\__pycache__\\cp1252.cpython-39.pyc'
import encodings.cp1252 # <_frozen_importlib_external.SourceFileLoader object at 0x0000022A64401F10>
import _signal # <class '_frozen_importlib.BuiltinImporter'>
# C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\__pycache__\latin_1.cpython-39.pyc matches C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\encodings\latin_1.py
# code object from 'C:\\Users\\ooliv\\AppData\\Local\\Programs\\Python\\Python39\\lib\\encodings\\__pycache__\\latin_1.cpython-39.pyc'
import encodings.latin_1 # <_frozen_importlib_external.SourceFileLoader object at 0x0000022A644172E0>
# C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\__pycache__\io.cpython-39.pyc matches C:\Users\ooliv\AppData\Local\Programs\Python\Python39\lib\io.py
# code object from 'C:\\Users\\ooliv\\AppData\\Local\\Programs\\Python\\Python39\\lib\\__pycache__\\io.cpython-39.pyc'
```

```
cmd Prompt de Comando
# C:\Users\ooliv\Documents\ITA - Doutorado\CE-237 Topicos Avancados Teste\__pycache__\TestStringMethods.cpython-39.pyc matches C:\Users\ooliv\Documents\ITA - Doutorado\
CE-237 Topicos Avancados Teste\TestStringMethods.py
# code object from 'C:\\Users\\ooliv\\Documents\\ITA - Doutorado\\CE-237 Topicos Avancados Teste\\__pycache__\\TestStringMethods.cpython-39.pyc'
import 'TestStringMethods' # <_frozen_importlib_external.SourceFileLoader object at 0x0000022A64B0C670>
...
-----
Ran 3 tests in 0.000s

OK
# clear builtins._
# clear sys.path
# clear sys.argv
# clear sys.ps1
# clear sys.ps2
# clear sys.last_type
# clear sys.last_value
# clear sys.last_traceback
# clear sys.path_hooks
# clear sys.path_importer_cache
# clear sys.meta_path
# clear sys.__interactivehook__
# restore sys.stdin
# restore sys.stdout
# restore sys.stderr
# cleanup[2] removing sys
# cleanup[2] removing builtins
# cleanup[2] removing _frozen_importlib
# cleanup[2] removing _imp
# cleanup[2] removing _thread
# cleanup[2] removing _warnings
# cleanup[2] removing _weakref
# cleanup[2] removing _frozen_importlib_external
# cleanup[2] removing nt
# cleanup[2] removing _io
# cleanup[2] removing marshal
# cleanup[2] removing winreg
# cleanup[2] removing time
# cleanup[2] removing zipimport
# cleanup[2] removing _codecs
# cleanup[2] removing codecs
# cleanup[2] removing encodings.aliases
# cleanup[2] removing encodings
# cleanup[2] removing encodings.utf_8
# cleanup[2] removing encodings.cp1252
```



🔍 Digite aqui para pesquisar



99%

15°C

17:55
29/08/2022



TEST DISCOVERY (NOVO NA VERSÃO 3.2)

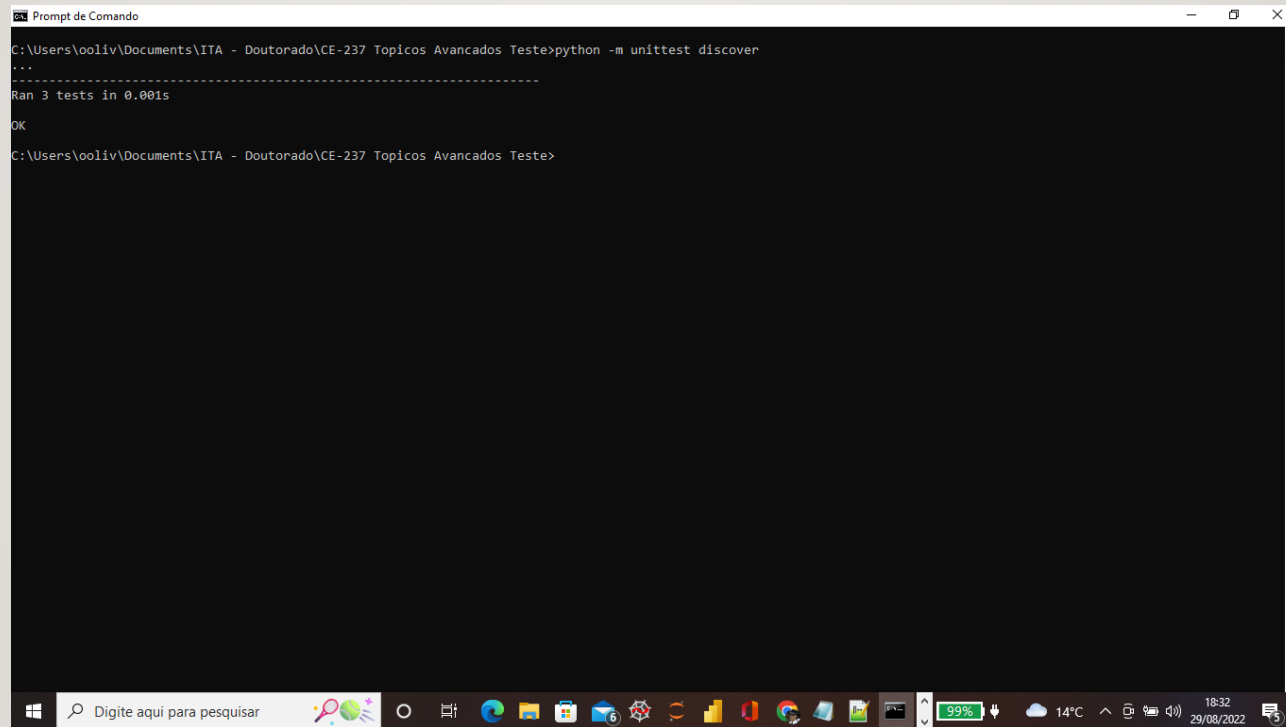
Unittest suporta test discovery. Para ser compatível com test discovery, todos os arquivos de testes devem ser módulos ou packages importáveis do top-level directory do projeto(isto significa que todos os filenames devem ser identifiers válidos).

O descobrimento de testes é implementado no `TestLoader.discover()`, mas também pode ser utilizado a partir da linha de comando. O comando básico para uso é:

```
cd project_directory
```

```
python -m unittest discover
```


EXECUÇÃO DO DISCOVER



```
Prompt de Comando

C:\Users\ooliv\Documents\ITA - Doutorado\CE-237 Topicos Avancados Teste>python -m unittest discover
...
-----
Ran 3 tests in 0.001s

OK

C:\Users\ooliv\Documents\ITA - Doutorado\CE-237 Topicos Avancados Teste>
```

The screenshot shows a Windows Command Prompt window titled "Prompt de Comando". The command prompt is open in the directory "C:\Users\ooliv\Documents\ITA - Doutorado\CE-237 Topicos Avancados Teste". The command "python -m unittest discover" has been executed, resulting in the output "Ran 3 tests in 0.001s" and "OK". The Windows taskbar is visible at the bottom, showing the search bar, taskbar icons, and system tray with the date "29/08/2022" and time "18:32".

CONSIDERAÇÕES FINAIS

- Divida para conquistar: Mantenha a estrutura de arquivos organizadas. Se um projeto é grande, é sempre uma boa prática dividir em arquivos menores (isso facilita a manutenção e legibilidade)
- Um teste também deve ser limpo igual ao código principal (*Clean Code* - Robert C. Martin)
- Use nomes descritivos para as funções de teste, mesmo que seja um nome muito longo
- Pense em *corner cases* (cenários fora do padrão esperado).
- Um teste não deve engessar a implementação do seu código.
- Escrever em pequenas unidades ajudam a testar o código e a melhorar sua clareza
- Refatore: Sempre que puder melhorar seu código, melhore!

CONSIDERAÇÕES FINAIS

- Testes são uma maneira de garantir que o programa **retorna o resultado esperado**.
- Agrega mais qualidade no produto que está sendo entregue.
- Os testes contribuem para que a equipe de desenvolvedores entenda os cenários que acontecem na aplicação e ajudam a identificar cenários fora do padrão.
- Testando nosso código, ajuda a encontrar maneiras de deixar o código mais limpo e conciso, facilitando assim na manutenção futura e a evitar bugs que possam ocorrer.
- É melhor que um teste detecte defeito do que o cliente.

REFERÊNCIAS

- Disponível em < <https://docs.python.org/pt-br/3/library/unittest.html#> > acesso em 03 Set 2022
- Disponível em < <https://dev.to/womakerscode/testes-em-python-parte-1-introducao-43ei> > acesso em 27 Set 2022