# **Teste Unitário em Python com Unittest**

## **O que é um Teste Unitário?**

Um teste unitário é uma maneira de validar a menor unidade testável de um software. Em Python, isso geralmente significa testar funções e métodos individuais para garantir que eles se comportam conforme o esperado. O objetivo é isolar cada parte do programa e mostrar que as partes individuais estão corretas. Testes unitários ajudam a identificar bugs precocemente no desenvolvimento, facilitam a refatoração e garantem que as mudanças no código não introduzam erros.

### Exemplos Básicos de Testes Unitários:

Vamos começar com um exemplo simples.



# Arquivo do Código (app.py)

# Exemplo com erro, pois a variavel total inicia com 1:

def soma(arg):

total = 1

for i in arg:

total += i

return total

# Exemplo correto, pois em multiplicação quando começa com 1 não altera:

def mult(arg):

total = 1

for i in arg:

total \*= i

return total

A função soma(arg) está executa a soma de todos os elementos de uma lista arg. Porém contém um erro de inicialização. O valor inicial da variável total é definido como 1, o que interfere no resultado final da soma, este devendo ser iniciado em 0.

Falando sobre a função mult(arg), ela faz a multiplicação de todos os elementos de uma lista arg. E nela a inicialização da variável total com 1 é correta e apropriada.



# Arquivo para teste (teste.py)

from app import soma, mult

def test\_soma():

assert soma([1,2,3]) == 6,

def test\_mult():

assert mult((2,3,4)) == 24,

if \_\_name\_\_ == "\_\_main\_\_":

test\_soma()

test\_mult()

print("Tudp Ok!")

O arquivo teste.py é um script simples de teste que verifica o funcionamento das funções soma e mult definidas no arquivo app.py. Ele utiliza o assert para confirmar se as funções retornam os resultados esperados.

O códgo acima é um exemplo de como realizar testes simples utilizando assert em Python. Ele verifica a correção e veracidade dos retornos das funções soma e mult importadas de app.py. Porém, por mais que este método de teste seja funcional, o uso de um framework de testes como unittest ou pytest proporcionaria uma abordagem mais robusta e flexível para testes automatizados, e agora vou falar um pouco sobre o unittest.

## **Biblioteca unittest:**

O unittest é uma biblioteca padrão em Python que fornece uma estrutura para criar e executar testes unitários. Ele permite organizar testes em classes, verificar resultados esperados, e gera relatórios detalhados dos testes executados.

Não é necessário usar pip install para instalar o unittest, pois o unittest é uma biblioteca padrão que vem inclusa no Python. Isso significa que ele está disponível para uso após a instalação do Python, sem a necessidade de instalação adicional.

### Como Utilizar a Biblioteca unittest:

1. Importar a Biblioteca: Comece importando unittest.
2. Criar uma Classe de Teste: Crie uma classe que herda de unittest.TestCase.
3. Escrever Métodos de Teste: Adicione métodos de teste na sua classe, começando com a palavra test.
4. Executar os Testes: Utilize unittest.main() para executar os testes.

### Exemplo de Código com unittest

Vamos olhar o código novoTest.py, que utiliza unittest para testar as funções soma e mult do módulo app.py.



# Arquivo para teste utilizando o unittest (novoTest.py)

import unittest

from app import soma, mult

class TestGeral(unittest.TestCase):

def test\_sum(self):

self.assertEqual(soma([1,2,3}), 6, "Deve ser 6")

def test\_mult(self):

self.assertEqual(mult((2,3,4)), 24, "Deve ser 24")

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

* A classe TestGeral herda de unittest.TestCase, o que significa que ela se beneficiará de todas as funcionalidades fornecidas pelo unittest;
* O método test\_sum verifica se a função soma retorna o valor correto quando aplicada à lista [1, 2, 3]. Utilizamos self.assertEqual para comparar o resultado da função com o valor esperado 6. Se a função não retornar 6, o teste falhará e a mensagem "Deve ser 6" será exibida;
* De maneira semelhante, o método test\_mult verifica se a função mult retorna o valor correto quando aplicada à tupla (2, 3, 4). Novamente, self.assertEqual é usado para comparar o resultado da função com o valor esperado 24. Se o resultado não for 24, o teste falhará e a mensagem "Deve ser 24" será exibida;
* Para executar os testes, basta rodar o script novoTest.py no terminal:

## **Concluindo**

Testes unitários são uma parte essencial do desenvolvimento de software, permitindo que os desenvolvedores validem suas funções e métodos de forma isolada e em tempo de criação. A biblioteca unittest do Python fornece uma estrutura robusta para escrever e executar testes.

Com a prática de escrever testes unitários, você pode aumentar significativamente a qualidade e a confiabilidade do seu código e evitar erros e bugs que aparecem inesperadamente quando entregamos o projeto ao cliente.

<https://www.dio.me/articles/teste-unitario-em-python-com-unittest>