CEFET-MG • Programação em Python

Testes Unitários

em Python

Aprendendo a testar seu código com o módulo unittest.

Por que testar o seu código?

“Todo programador comete erros.” Teste o seu código **ANTES** do seu usuário!

01

Confiança

Garante que funções e classes funcionam como esperado a cada mudança.

02

Documentação viva

Cada teste mostra um caso de uso real do seu código.

03

Refatoração segura

Você pode alterar o código sem medo de quebrar o que já funciona.

FERRAMENTA PADRÃO

O que é o módulo unittest?

Framework padrão

de testes automatizados do Python

Permite **criar, organizar e executar testes** de forma sistemática.

Garante que partes do seu código *(funções, classes e métodos)* funcionem conforme o esperado.

1. **Já vem no Python**

   Não precisa instalar nada. Apenas import unittest.

2. **Organiza testes em classes**

   Cada classe agrupa testes relacionados.

3. **Asserções prontas**

   assertEqual, assertTrue, assertRaises…

4. **Executa em 1 comando**

   python -m unittest arquivo.py

ESTRUTURA BÁSICA

Anatomia de um teste em unittest

test_soma.py

```python
import unittest
from minha_app import soma

class TestSoma(unittest.TestCase):
    def test_soma_positivos(self):
        self.assertEqual(soma(2, 3), 5)

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

1. **Importa o módulo**
   Sempre comece com import unittest.

2. **Cria uma classe**
   Herda de TestCase para obter os métodos de asserção.

3. **Métodos test_***
   O runner do unittest busca nomes que começam com test_.

4. **Asserções**
   Validam se o resultado obtido é igual ao esperado.

Principais métodos de asserção

Método	Verifica se…	Exemplo
`assertEqual(a, b)`	a == b	`self.assertEqual(soma(2,3), 5)`
`assertNotEqual(a, b)`	a != b	`self.assertNotEqual(x, 0)`
`assertTrue(x)`	bool(x) é True	`self.assertTrue(eh_par(4))`
`assertFalse(x)`	bool(x) é False	`self.assertFalse(eh_par(3))`
`assertIn(a, b)`	a está em b	`self.assertIn('py', frase)`
`assertIsNone(x)`	x é None	`self.assertIsNone(resp)`
`assertIsInstance(o,c)`	isinstance(o, c)	`self.assertIsInstance(x, int)`
`assertRaises(Err)`	Exceção Err é lançada	`with self.assertRaises(ValueError):`

Dica: existe também a variação Not — assertNotIn, assertIsNotNone, assertNotIsInstance…

PASSO A PASSO

4 passos para criar seu primeiro teste

Passo 1

Importar unittest

```
import unittest
```

Passo 2

Criar a classe

```
class
TestX(unittest.TestCase):
```

Passo 3

Métodos test_*

```
def test_caso(self):
 self.assertEqual(...)
```

Passo 4

Executar

```
python -m unittest
arquivo.py
```

⚙ Lembre-se: nome de método de teste **DEVE** começar com `test_`

Exemplo: testando a função soma()

soma.py

```python
def soma(a, b):
    return a + b
```

Saída esperada

```
...
----------------------------------
Ran 3 tests in 0.001s

OK
```

test_soma.py

```python
import unittest
from soma import soma

class TestSoma(unittest.TestCase):
    def test_positivos(self):
        self.assertEqual(soma(3, 5), 8)

    def test_zero(self):
        self.assertEqual(soma(0, 0), 0)

    def test_negativo(self):
        self.assertEqual(soma(-2, 5), 3)

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

TERMINAL

Executando e lendo os resultados

terminal

```
$ python -m unittest test_soma.py
.F.E

FAIL: test_subtracao (test_soma.TestSoma)
AssertionError: 4 != 5

Ran 4 tests in 0.001s
FAILED (failures=1, errors=1)
```

Passou

Um ponto = um teste aprovado.

Falhou

Asserção falhou (valor esperado ≠ obtido).

Erro

Exceção inesperada durante o teste.

setUp(): preparando o cenário do teste

Quando vários testes precisam do mesmo objeto, evite repetir código:

Use setUp() para...

- Criar objetos antes de cada teste.
- Reusar a mesma instância em vários métodos.
- Deixar os testes mais curtos e legíveis.
- **É chamado ANTES de cada test_*.**

test_employee.py

```
class TestEmployee(unittest.TestCase):
    def setUp(self):
        self.emp = Employee('João', 'Silva', 50000)

    def test_aumento_default(self):
        self.emp.give_raise()
        salario = self.emp.annual_salary
        self.assertEqual(salario, 55000)

    def test_aumento_custom(self):
        self.emp.give_raise(10000)
        salario = self.emp.annual_salary
        self.assertEqual(salario, 60000)
```

Testando uma classe inteira

retangulo.py

```python
class Retangulo:
    def __init__(self, largura, altura):
        self.largura = largura
        self.altura  = altura

    def area(self):
        return self.largura * self.altura

    def perimetro(self):
        return 2 * (self.largura + self.altura)
```

test_retangulo.py

```python
class TestRetangulo(unittest.TestCase):
    def setUp(self):
        self.r = Retangulo(5, 10)

    def test_area(self):
        self.assertEqual(self.r.area(), 50)

    def test_perimetro(self):
        self.assertEqual(self.r.perimetro(),
30)

    def test_largura_zero(self):
        r = Retangulo(0, 10)
        self.assertEqual(r.area(), 0)
```

Testando exceções com assertRaises

E se o código DEVE falhar?

Use assertRaises dentro de um bloco with para verificar se o código realmente lança a exceção esperada.

Casos típicos:

- Lista vazia → ValueError
- Saque maior que saldo
- Argumento de tipo inválido
- Tarefa duplicada não permitida

test_media.py

```
def calcular_media(valores):
    if not valores:
        raise ValueError('lista vazia')
    return sum(valores) / len(valores)

class TestMedia(unittest.TestCase):
    def test_lista_vazia(self):
        with self.assertRaises(ValueError):
            calcular_media([])

    def test_media_ok(self):
        self.assertEqual(
            calcular_media([2, 4, 6, 8]), 5
        )
```

Mocks — isolando dependências externas

Mock = substituto controlado. Permite trocar partes do sistema (chamadas a APIs, sensores, banco…) por valores fixos durante o teste.

test_sensor.py

```python
from unittest.mock import patch
from sensor_temperatura import SensorTemperatura

class TestSensor(unittest.TestCase):
    def setUp(self):
        self.sensor = SensorTemperatura()

    @patch('sensor_temperatura.random.uniform')
    def test_em_alerta(self, mock_uniform):
        mock_uniform.return_value = 45.0
        self.sensor.ler_temperatura()
        self.assertTrue(self.sensor.em_alerta())
```

Por que usar?

- Resultados previsíveis (sem aleatório).
- Sem depender de internet ou hardware.
- Testes rápidos e repetíveis.
- Foca apenas no código que você está testando.

MÃO NA MASSA

Hora de praticar — Exercícios em sala (1/2)

Exercício 01

Função soma(a, b)

Implemente soma(a, b) e teste com unittest:

- dois positivos
- positivo e negativo
- dois zeros

```
self.assertEqual(soma(3, 5), 8)
```

Exercício 02

Cidade, País

Função que recebe cidade e país e devolve “Cidade, País”.

Guarde em exemplo02.py e teste em test_cities.py.

```
self.assertEqual(city_country('sa
ntiago', 'chile'), 'Santiago,
Chile')
```

Exercício 03

inverter_string(s)

Função que inverte uma string. Crie testes para:

- string normal
- string vazia
- string com espaços

```
self.assertEqual(inverter_string(
'Python'), 'nohtyP')
```

Hora de praticar — Exercícios em sala (2/2)

Exercício 04

Classe Employee

Atributos: nome, sobrenome, salário.
Método give_raise(amount=5000).
Teste com setUp() o aumento padrão e um aumento customizado.

Exercício 05

Classe ContaBancaria

Métodos depositar(), sacar(), saldo.
Use setUp() para criar conta com R$100.
Use assertRaises para saque > saldo.

Exercício 06

Classe TaskManager

Adiciona e remove tarefas (strings).
• Não permite tarefa duplicada
• Erro ao remover tarefa inexistente
Use assertIn, assertNotIn e assertRaises.

Desafio final: SensorTemperatura

Objetivo

Criar a classe SensorTemperatura e testá-la com unittest.mock.

- ler_temperatura() — simula leitura entre -10 e 50 °C
- em_alerta() — retorna True se a última leitura > 40 °C
- Use @patch para fixar o valor de random.uniform
- Teste limites: -10, 40 e 50

test_sensor_temperatura.py

```
@patch('sensor_temperatura.random.uniform')
def test_limites(self, mock_uniform):
    casos = [(-10, False), (40, False), (50,
True)]
    for valor, esperado in casos:
        mock_uniform.return_value = valor
        self.sensor.ler_temperatura()
        self.assertEqual(
            self.sensor.em_alerta(), esperado
        )
```

RESUMO

O que você precisa lembrar

01

Teste SEMPRE

Todo programador erra — descubra antes do usuário.

02

unittest.TestCase

Sua classe de teste herda dele para ganhar todos os asserts.

03

Nomes test_*

Métodos de teste precisam começar com test_.

04

setUp() salva tempo

Crie objetos uma vez, reuse em todos os testes.

05

Use assertRaises

Para validar que o código falha quando deveria falhar.

06

Mock dependências

Sensores, APIs e aleatório — use @patch para controlar.