Testes Automatizados com Pytest

Laboratório Prático

Prof. Luciano Rossi

Ciência da Computação
Centro Universitário FEI

1º Semestre de 2026

Agenda da Aula

1. Introdução ao Pytest
2. Estrutura de Projeto com Testes
3. Exemplo Prático: Calculadora
4. Executando Testes
5. Conceitos Avançados
6. Exercício: Sistema de Notas

Objetivo: Aprender a criar testes automatizados eficazes usando Pytest

centro universitário FEI

O que é Pytest?

Definição

Pytest é um framework de testes para Python que torna simples escrever testes pequenos e escaláveis.

Vantagens:

- Sintaxe simples e intuitiva
- Descoberta automática de testes
- Mensagens de erro detalhadas
- Suporte a fixtures e parametrização
- Relatórios de cobertura de código
- Extensível com plugins

Instalação:

```
pip install pytest
pip install pytest-cov
```

centro universitário FEI

Estrutura de um Projeto com Testes

Organização:

```
projeto/
 calculadora.py
 test_calculadora.py
 requirements.txt
 README.md
```

Convenções:

- Arquivos: `test_*.py`
- Funções: `test_*`
- Classes: `Test*`

requirements.txt

```
pytest==7.4.3
pytest-cov==4.1.0
```

Instalação:

```
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

centro universitário FEI

Exemplo: Calculadora

calculadora.py

```python
def somar(a, b):
"""Soma dois numeros."""
        return a + b

def subtrair(a, b):
"""Subtrai o segundo do primeiro."""
        return a - b

def multiplicar(a, b):
"""Multiplica dois numeros."""
        return a * b

def dividir(a, b):
"""Divide o primeiro pelo segundo."""
        if b == 0:
                raise ValueError("Divisao por zero!")
        return a / b
```

Testes Básicos

test_calculadora.py

```
1  import pytest
2  from calculadora import somar, dividir
3
4  def test_somar_positivos():
5          assert somar(2, 3) == 5
6          assert somar(10, 20) == 30
7
8  def test_somar_negativos():
9          assert somar(-5, -3) == -8
10
11 def test_somar_com_zero():
12         assert somar(0, 5) == 5
13         assert somar(0, 0) == 0
```

centro universitário FEI

Testes com Exceções

```python
def test_dividir_basico():
        """Testa divisao basica"""
        assert dividir(10, 2) == 5
        assert dividir(20, 4) == 5

def test_dividir_por_zero():
        """Testa excecao divisao por zero"""
        with pytest.raises(ValueError):
                dividir(10, 0)

def test_dividir_zero_por_numero():
        """Testa zero dividido por numero"""
        assert dividir(0, 5) == 0
```

Importante

Use pytest.raises() para verificar exceções.

Organizando em Classes

```python
class TestSomar:
        """Grupo de testes para somar"""

        def test_positivos(self):
                assert somar(2, 3) == 5

        def test_negativos(self):
                assert somar(-5, -3) == -8

class TestDividir:
        """Grupo de testes para dividir"""

        def test_basico(self):
                assert dividir(10, 2) == 5

        def test_por_zero(self):
                with pytest.raises(ValueError):
                        dividir(10, 0)
```

Executando os Testes

Comandos:

```
# Todos
pytest

# Verbose
pytest -v

# Arquivo
pytest test_calc.py

# Teste especifico
pytest test_calc.py::test_soma

# Classe
pytest test_calc.py::TestSomar
```

Saída:

```
===== test session starts =====
collected 15 items

test_calculadora.py .......
...............      [100%]

===== 15 passed in 0.05s ===
```

Com falhas:

```
FAILED test.py::test_soma
PASSED test.py::test_sub
=== 1 failed, 1 passed ===
```

centro universitário FEI

Testes Parametrizados

```python
import pytest

@pytest.mark.parametrize("a,b,esperado", [
        (2, 3, 5),
        (10, 20, 30),
        (-5, 5, 0),
        (0, 0, 0),
        (100, 1, 101)
])

def test_somar_parametrizado(a, b, esperado):
        """Testa multiplos casos"""
        assert somar(a, b) == esperado
```

Vantagens:

- Código limpo
- Fácil adicionar casos
- Relatório individual

centro universitário FEI

Cobertura de Código

Comandos:

```
# Com cobertura
pytest --cov=calculadora

# Relatorio HTML
pytest --cov=calculadora --cov-report=html

# Mostrar linhas faltantes
pytest --cov=calculadora --cov-report=term-missing
```

Saída:

```
---------- coverage -----------
Name            Stmts   Miss  Cover
---------------------------------
calculadora.py     12      0  100%
---------------------------------
```

centro universitário FEI

Valores Aproximados

```python
# ERRADO - pode falhar
def test_divisao_errado():
        assert dividir(10, 3) == 3.333333

# CORRETO - usa aproximacao
def test_divisao_correto():
        assert dividir(10, 3) == pytest.approx(3.333333)

# Tambem funciona
def test_soma_decimais():
        assert somar(0.1, 0.2) == pytest.approx(0.3)
```

pytest.approx()

Use para comparar ponto flutuante.

centro universitário FEI

Exercício: Sistema de Notas

Implementar 5 funções:

1. **validar_nota(nota)**
   Verifica se está entre 0 e 10
2. **calcular_media(notas)**
   Média ignorando inválidas
3. **obter_situacao(media)**
   Aprovado / Recuperação / Reprovado
4. **calcular_estatisticas(notas)**
   Dicionário com estatísticas
5. **normalizar_notas(notas, max)**
   Converte para escala 0–10

centro universitário FEI

Função 1: validar_nota

```python
def validar_nota(nota):
        """
        Valida se nota esta em [0, 10].

        Args:
        nota: Nota a validar

        Returns:
        bool: True se valida

        Exemplo:
        >>> validar_nota(7.5)
        True
        """
        # TODO: Implementar
        pass
```

centro universitário FEI

Função 2: calcular_media

```python
def calcular_media(notas):
        """
        Calcula media de uma lista.

        Args:
        notas: Lista de notas

        Returns:
        float: Media das validas

        Raises:
        ValueError: Se vazia
        """
        # TODO: Implementar
        pass
```

centro universitário FEI

Função 3: obter_situacao

```python
def obter_situacao(media):
        """
        Determina situacao do aluno.

        Criterios:
        >= 7.0: Aprovado
        >= 5.0: Recuperacao
        < 5.0: Reprovado

        Returns:
        str: Situacao

        Raises:
        ValueError: Media invalida
        """
        # TODO: Implementar
        pass
```

Função 4: calcular_estatisticas

```python
def calcular_estatisticas(notas):
        """
        Calcula estatisticas.

        Returns:
        dict: {
                "media": float,
                "maior": float,
                "menor": float,
                "aprovados": int,
                "reprovados": int,
                "recuperacao": int
        }
        """
        # TODO: Implementar
        pass
```

Função 5: normalizar_notas

```python
def normalizar_notas(notas, nota_maxima=10):
        """
        Normaliza para escala 0-10.

        Args:
        notas: Lista original
        nota_maxima: Valor maximo

        Returns:
        list: Notas normalizadas

        Exemplo:
        >>> normalizar_notas([10,20], 20)
        [5.0, 10.0]
        """
        # TODO: Implementar
        pass
```

Exemplo de Testes

```
class TestValidarNota:
        def test_nota_valida_inteira(self):
                assert validar_nota(0) == True
                assert validar_nota(5) == True
                assert validar_nota(10) == True

        def test_nota_invalida_negativa(self):
                assert validar_nota(-1) == False

        def test_nota_invalida_acima(self):
                assert validar_nota(11) == False

# Total: 25 testes
```

centro universitário FEI

Boas Práticas

1. **Independentes** – Cada teste executa sozinho
2. **Nomes descritivos**
3. **Uma validação** – Testes focados
4. **Casos limites** – Zero, negativos, máximos
5. **Teste exceções** – Valide erros esperados
6. **Documente** – Explique o teste

centro universitário FEI

Padrão AAA

Arrange-Act-Assert:

```
def test_calcular_estatisticas():
        # ARRANGE - Preparar
        notas = [3, 5, 7, 9]

        # ACT - Executar
        resultado = calcular_estatisticas(notas)

        # ASSERT - Verificar
        assert resultado["media"] == 6.0
        assert resultado["maior"] == 9
        assert resultado["menor"] == 3
```

centro universitário FEI

Comandos Úteis

Comando	Descrição
`pytest`	Todos os testes
`pytest -v`	Verbose
`pytest -s`	Mostra prints
`pytest -x`	Para no erro
`pytest --lf`	Só falhados
`pytest --cov`	Cobertura
`pytest -k "soma"`	Filtro nome

https://docs.pytest.org

centro universitário FEI