Documento Explicativo - Pytest

Resumo Visual

```
Pytest
├── Fundamentos
│   ├── assert
│   └── test_
├── Fixtures
│   ├── Básico
│   └── setup/teardown
├── Parametrização
├── Exceções
├── Marcadores
├── Plugins
├── Boas Práticas
└── Organização
```

1. Introdução: O Que São Testes de Software?

Imagine que você está montando um brinquedo. Antes de entregar para uma criança, você verifica se todas as peças estão no lugar e se ele funciona como deveria, certo? No mundo da programação, fazemos isso com "testes de software".

Por que testar um software?

- Para garantir que o sistema funciona como esperado.
- Para evitar erros futuros.
- Para facilitar alterações no código com segurança.

🔍 2. Visão Geral dos Tipos de Testes

Existem diferentes formas de testar um software:

Tipo de Teste	O que faz
Teste Unitário	Verifica partes pequenas do código, como funções.
Teste de Integração	Testa se diferentes partes do sistema funcionam bem juntas.
Teste de Sistema	Testa o sistema como um todo.
Teste Manual	Feito por uma pessoa, clicando e testando.
Teste Automatizado	Feito por scripts, de forma automática.

3. Ambiente Necessário para Usar Pytest

Antes de tudo, você precisa de:

- Um computador com **Python** instalado.
- Um editor de código (ex: **VS Code**).
- Um terminal (Prompt de Comando ou Terminal Linux).
- Pip instalado (vem com o Python).

🐍 4. O que é o Pytest?

O **Pytest** é uma ferramenta usada para **automatizar testes** no Python. Ele é fácil de usar e muito poderoso. É como um assistente que executa testes para você e avisa se algo está errado.

Vantagens:

- Sintaxe simples.
- Permite criar testes complexos.
- Suporta plugins.
- Gera relatórios úteis.

---

5. Comparação com Outros Frameworks

Framework	Fácil de usar?	Plugins?	Popularidade
Pytest	Muito fácil	Sim	Muito usado
Unittest	Verboso	Limitado	Médio
Nose	Desatualizado	Poucos	Em desuso

---

6. Como Instalar o Pytest

Abra o terminal e digite:

```
bash
```

`pip install pytest`

Depois, confirme a instalação:

```
bash
```

`pytest --version`

Se aparecer a versão, deu certo!

⚙️ 7. Configuração Inicial

Crie um arquivo de teste com o nome `test_alguma_coisa.py`. A função de teste deve começar com `test_`.

```
1 python
```

```
# test_soma.py def soma(a, b):     return a + b  def test_soma():     assert soma(2, 3) == 5
```

Execute com:

```
1 bash
```

```
pytest
```

---

8. Fundamentos do Pytest

- **Função de Teste:** sempre começa com `test_`.
- **Assert:** é o que verifica se o valor está correto.

```
1 python
```

```
def test_exemplo():     assert 2 + 2 == 4
```

Se der certo, ✅. Se não, ❌.

---

9. Entendendo e Usando Fixtures

Fixtures são como **preparadores de ambiente**: criam dados, arquivos ou conexões antes de um teste.

```
1 python
```

```
import pytest  @pytest.fixture def saudacao():     return "Olá!"  def test_saudacao(saudacao):     assert saudacao
== "Olá!"
```

---

10. Setup e Teardown com Fixtures

Você pode preparar e limpar o ambiente antes/depois de cada teste.

```
1 python
```

```
import pytest  @pytest.fixture def recurso():     print("🔧 Preparando recurso")     yield "Recurso pronto"
print("🧹 Limpando recurso")  def test_usando_recurso(recurso):     print(f"Usando: {recurso}")     assert True
```

---

11. Testes Parametrizados

Permitem testar várias entradas sem repetir o mesmo teste.

```
python
```

```
import pytest  @pytest.mark.parametrize("entrada,esperado", [     (2, 4),     (3, 9),     (4, 16), ]) def test_quadrado(entrada, esperado):     assert entrada ** 2 == esperado
```

🏷️ 12. Marcadores (Markers)

São etiquetas que você coloca nos testes para filtrá-los.

```
python
```

```
import pytest  @pytest.mark.lento def test_funcao_lenta():     assert True
```

Rodar apenas os testes marcados com "lento":

```
bash
```

```
pytest -m lento
```

---

13. Testando Exceções com `pytest.raises`

Se o seu código deve lançar erro, você pode testar isso!

```
python
```

```
import pytest  def divide(a, b):     if b == 0:         raise ValueError("Não pode dividir por zero!")     return a / b  def test_divisao_zero():     with pytest.raises(ValueError):         divide(10, 0)
```

🔌 14. Uso de Plugins no Pytest

Plugins aumentam os poderes do Pytest! Exemplos:

- `pytest-cov`: mede cobertura de testes.
- `pytest-django`: para projetos Django.
- `pytest-xdist`: executa testes em paralelo.

Instalação:

```
bash
```

```
pip install pytest-cov
```

Uso:

```
bash
```

```
pytest --cov=meu_modulo
```

---

15. Boas Práticas em Testes com Pytest

- Use nomes claros.
- Escreva poucos testes por arquivo.
- Mantenha os testes rápidos.
- Teste apenas uma coisa por teste.
- Use `pytest.raises` ao testar erros.
- Use `pytest.mark.parametrize` para evitar repetição.

🗂️ 16. Organização dos Testes

Organize seus testes em uma pasta chamada `tests`.

```
1 css
```

```
projeto/ | ├── app/ |   └── main.py | └── tests/     ├── test_main.py     └── __init__.py
```

No `pytest.ini`, você pode personalizar:

```
1 ini
```

```
# pytest.ini [pytest] markers =     lento: marca testes que são lentos addopts = -ra -q
```

---

17. Nomeação e Documentação de Testes

Nomes:

- Sempre comece com `test_`.
- Seja descritivo: `test_usuario_nao_logado_redirecionado`.

Comentários e Docstrings:

```
1 python
```

```
def test_login_com_dados_invalidos():     """     Testa se o login falha ao passar credenciais incorretas.     """
assert login("user", "errado") is False
```