Pytest - Parametrização

[5]

Metodologias de desenvolvimento de software

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas
Prof. Felippe Scheidt – IFPR – Campus Foz do Iguaçu
2024

Tópicos

- Testes unitários
- Parametrização do teste
- Inicialização de dependências

Assertion

- Uma asserção (assertion) é uma expressão lógica que sempre deve retornar verdadeiro.
- Usado para **detectar falhas** no código
- Quando o assert avaliar **false**, ocorrerá uma exceção interrompendo a execução do programa.
- Pode ser colocada no início do código para validar certas condições necessárias para a correta execução do programa.
- Quando colocada no final do código podemos definir o estado final esperado do processamento.

Python **assert**

Por exemplo, o código abaixo contém **dois** asserts.

Ambos devem avaliar a expressão como **verdadeira** durante a execução, caso contrário uma **exceção** será lançada.

```
x = 1
assert x > 0
x += 1
assert x > 1
```

Resultados do teste:

True => **PASS**

False => **FAILED** => AssertionError

Será que passa?

```
1   # arquivo: test_assert1.py
2   def f():
3       return 3
4
5   def test_function():
6       assert f() == 4
```

Resultado: FAILED

```
$ pytest test_assert1.py
==================== test session starts ====================
collected 1 item
test_assert1.py F                                          [100%]
========================= FAILURES ==========================
_____________________ test_function _______________________
    def test_function():
>     assert f() == 4
E     assert 3 == 4
E      + where 3 = f()
test_assert1.py:6: AssertionError
================== short test summary info ==================
FAILED test_assert1.py::test_function - assert 3 == 4
```

Comandos pytest

pytest

procura e executa todos os testes e executa

pytest -v

fornece mais informações sobre cada teste

pytest endpoint/

roda somente os testes contidos na pasta endpoint

pytest api_test.py

roda somente os testes do arquivo api_test.py

pytest endpoint --collect-only

exibe todos os arquivos de testes encontrados

Assert + list e dict

```python
def test_saude_health():
    expected = {'dragon': 100, 'troll': 70}
    assert get_classes_starting_health() == expected

def test_tipos_npc():
    expected = ['dragon', 'troll']
    assert get_npc_types() == expected
```

Parametrização

Permite a passagem de múltiplos valores para uma função de teste, "automatizando" a entrada de parâmetros. Para cada entrada é preciso definir a saída esperada.

@pytest.mark.parametrize("areas", [(0,0),(1,3.14159])

def test_area_circulo():

...

```python
@pytest.mark.parametrize("password,encrypted",[
    ("password123", "cbfdac6008f9cab4083784cbd1874f76618d2a97"),
    ("1234", "7110eda4d09e062aa5e4a390b0a572ac0d2c0220"),
])
def test_generate_sha1(self, password, encrypted):
    enc_p = encrypt_password(password)
    assert enc_p == encrypted
```

Inicialização

- O decorator @fixture é usado para iniciar o ambiente de execução do teste, inicializando o contexto (variáveis, objetos, dependências...) necessário para o teste completar a execução.
- Fixtures podem ser usadas em diferentes testes.

```
@pytest.fixture()
def token():
    return "0xa12b31"
```

Exercícios

1. random_password_gen
2. qr_code
3. Unique words
4. Reduce image
5. hashing_passwords

Repositório git

Escrever testes para os códigos do repositório **python-mini-projects**

Referências

1. Pytest