**Atividade 2 – Testes Unitários**

**Grupo:**

Guilherme Lima Machado – RA: 822162693

Arthur Salatine de Moraes – RA: 822151203

**1.** **Escreva função is\_par(n) que retorna True se n for par;**

**Testes: números par, ímpar, zero, número negativo.**

***par.py***

*def is\_par(n):*

*return n % 2 == 0*

***test\_par.py***

*import unittest*

*from par import is\_par*

*class TestPar(unittest.TestCase):*

*def test\_numero\_par(self):*

*self.assertTrue(is\_par(4))*

*self.assertTrue(is\_par(100))*

*def test\_numero\_impar(self):*

*self.assertFalse(is\_par(3))*

*self.assertFalse(is\_par(99))*

*def test\_zero(self):*

*self.assertTrue(is\_par(0))*

*def test\_numero\_negativo(self):*

*self.assertTrue(is\_par(-8))*

*self.assertFalse(is\_par(-7))*

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

*unittest.main()*

**2. Função fatorial(n), retorna fatorial para n >= 0, levanta ValueError se n < 0;**

**Testes: fatorial(0) == 1, fatorial(5) == 120, fatorial(-1) levanta ValueError.  
  
*fatorial.py***

*def fatorial(n):*

*ir n < 0:*

*raise ValueError("Fatorial não definido para números negativos.")*

*if n == 0:*

*return 1*

*resultado = 1*

*for i in range(1, n + 1):*

*resultado \*= i*

*return resultado*

***test\_fatorial.py***

*import unittest*

*from fatorial import fatorial*

*class TestFatorial(unittest.TestCase):*

*def test\_fatorial\_zero(self):*

*self.assertEqual(fatorial(0), 1)*

*def test\_fatorial\_positivo(self):*

*self.assertEqual(fatorial(5), 120)*

*def test\_fatorial\_negativo(self):*

*with self.assertRaises(ValueError):*

*fatorial(-1)*

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

*unittest.main()*

**3.** **Classe Conta com métodos depositar(amount) e sacar(amount).**

**Levanta ValueError em valores negativos e InsufficientFunds quando tentar**

**sacar mais do que tem;**

**Testes: depósito, saque com sucesso, saque insuficiente, entradas**

**inválidas.  
  
*conta.py***

*class InsufficientFunds(Exception):*

*pass*

*class Conta:*

*def \_\_init\_\_(self, saldo\_inicial: float = 0.0):*

*if saldo\_inicial < 0:*

*raise ValueError("Saldo inicial não pode ser negativo.")*

*self.saldo = saldo\_inicial*

*def depositar(self, amount: float):*

*if amount <= 0:*

*raise ValueError("O valor do depósito deve ser positivo.")*

*self.saldo += amount*

*def sacar(self, amount: float):*

*if amount <= 0:*

*raise ValueError("O valor do saque deve ser positivo.")*

*if amount > self.saldo:*

*raise InsufficientFunds("Saldo insuficiente para o saque.")*

*self.saldo -= amount*

***test\_conta.py***

*import unittest*

*from conta import Conta, InsufficientFunds*

*class TestConta(unittest.TestCase):*

*def test\_deposito\_valido(self):*

*conta = Conta()*

*conta.depositar(100)*

*self.assertEqual(conta.saldo, 100)*

*def test\_saque\_com\_sucesso(self):*

*conta = Conta(200)*

*conta.sacar(50)*

*self.assertEqual(conta.saldo, 150)*

*def test\_saque\_insuficiente(self):*

*conta = Conta(100)*

*with self.assertRaises(InsufficientFunds):*

*conta.sacar(200)*

*def test\_deposito\_invalido(self):*

*conta = Conta()*

*with self.assertRaises(ValueError):*

*conta.depositar(-50)*

*def test\_saque\_invalido(self):*

*conta = Conta(100)*

*with self.assertRaises(ValueError):*

*conta.sacar(-20)*

*def test\_saldo\_inicial\_invalido(self):*

*with self.assertRaises(ValueError):*

*Conta(-100)*

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

*unittest.main()*

**4. Função buscar\_clima(cidade) que chama:**

**requests.get('https://api.exemplo/clima?cidade=...') e retorna temperatura;**

**Escreva testes de mock que chamem requests.get e validem que sua função trata**

**corretamente a resposta e exceções (ex.: quando json() não contém temperatura).**

***clima.py***

*import requests*

*def buscar\_clima(cidade):*

*url = f"https://api.exemplo/clima?cidade={cidade}"*

*try:*

*response = requests.get(url, timeout=5)*

*response.raise\_for\_status()*

*dados = response.json()*

*except requests.RequestException as e:*

*raise ValueError(f"Erro na requisição: {e}")*

*if "temperatura" not in dados:*

*raise ValueError("Resposta inválida: 'temperatura' não encontrada.")*

*return dados["temperatura"]*

***teste\_clima.py***

*import unittest*

*from unittest.mock import patch, Mock*

*from clima import buscar\_clima*

*class TestBuscarClima(unittest.TestCase):*

*@patch("clima.requests.get")*

*def test\_resposta\_valida(self, mock\_get):*

*mock\_response = Mock()*

*mock\_response.json.return\_value = {"temperatura": 25.5}*

*mock\_get.return\_value = mock\_response*

*self.assertEqual(buscar\_clima("São Paulo"), 25.5)*

*@patch("clima.requests.get")*

*def test\_resposta\_sem\_temperatura(self, mock\_get):*

*mock\_response = Mock()*

*mock\_response.json.return\_value = {"umidade": 80}*

*mock\_get.return\_value = mock\_response*

*with self.assertRaises(ValueError) as cm:*

*buscar\_clima("Rio de Janeiro")*

*self.assertIn("temperatura", str(cm.exception))*

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

*unittest.main()*