Thiago Amaral - 8221515695

Testes utilizando Python e Colab

```
import unittest
from unittest.mock import patch, Mock

# ----------------------------
# 1. Função is_par
# ----------------------------
def is_par(n: int) -> bool:
    return n % 2 == 0

# ----------------------------
# 2. Função fatorial
# ----------------------------
def fatorial(n: int) -> int:
    if n < 0:
        raise ValueError("n deve ser >= 0")
    if n == 0:
        return 1
    resultado = 1
    for i in range(1, n+1):
        resultado *= i
    return resultado

# ----------------------------
# 3. Classe Conta
# ----------------------------
class InsufficientFunds(Exception):
    pass
```

```
class Conta:
    def __init__(self):
        self.saldo = 0

    def depositar(self, amount: float):
        if amount < 0:
            raise ValueError("Depósito não pode ser negativo")
        self.saldo += amount

    def sacar(self, amount: float):
        if amount < 0:
            raise ValueError("Saque não pode ser negativo")
        if amount > self.saldo:
            raise InsufficientFunds("Saldo insuficiente")
        self.saldo -= amount

# ----------------------------
# 4. Função buscar_clima
# ----------------------------
import requests

def buscar_clima(cidade: str) -> float:
    try:
        resp = requests.get(f"https://api.exemplo/clima?cidade={cidade}")
        dados = resp.json()
        if "temperatura" not in dados:
            raise KeyError("Resposta não contém temperatura")
        return dados["temperatura"]
    except Exception as e:
        raise RuntimeError(f"Erro ao buscar clima: {e}")
```

```
# ----------------------------
# Testes Unitários
# ----------------------------
class TestFuncoes(unittest.TestCase):

    # Testes para is_par
    def test_is_par_com_numero_par(self):
        self.assertTrue(is_par(4))

    def test_is_par_com_numero_impar(self):
        self.assertFalse(is_par(3))

    def test_is_par_com_zero(self):
        self.assertTrue(is_par(0))

    def test_is_par_com_negativo(self):
        self.assertTrue(is_par(-8))
        self.assertFalse(is_par(-7))

    # Testes para fatorial
    def test_fatorial_zero(self):
        self.assertEqual(fatorial(0), 1)

    def test_fatorial_numero(self):
        self.assertEqual(fatorial(5), 120)

    def test_fatorial_negativo(self):
        with self.assertRaises(ValueError):
            fatorial(-1)

    # Testes para Conta
```

```
def test_deposito(self):
    c = Conta()
    c.depositar(100)
    self.assertEqual(c.saldo, 100)

def test_saque_com_sucesso(self):
    c = Conta()
    c.depositar(200)
    c.sacar(50)
    self.assertEqual(c.saldo, 150)

def test_saque_insuficiente(self):
    c = Conta()
    c.depositar(50)
    with self.assertRaises(InsufficientFunds):
        c.sacar(100)

def test_entrada_invalida(self):
    c = Conta()
    with self.assertRaises(ValueError):
        c.depositar(-10)
    with self.assertRaises(ValueError):
        c.sacar(-5)

# Testes para buscar_clima com Mock
@patch("requests.get")
def test_buscar_clima_sucesso(self, mock_get):
    mock_resp = Mock()
    mock_resp.json.return_value = {"temperatura": 25}
    mock_get.return_value = mock_resp
```

```
        resultado = buscar_clima("São Paulo")
        self.assertEqual(resultado, 25)

    @patch("requests.get")
    def test_buscar_clima_sem_temperatura(self, mock_get):
        mock_resp = Mock()
        mock_resp.json.return_value = {"umidade": 80}
        mock_get.return_value = mock_resp

        with self.assertRaises(RuntimeError):
            buscar_clima("São Paulo")

    @patch("requests.get")
    def test_buscar_clima_excecao(self, mock_get):
        mock_get.side_effect = Exception("Falha na conexão")

        with self.assertRaises(RuntimeError):
            buscar_clima("São Paulo")


# Executar os testes
if __name__ == "__main__":
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```