GQS – Lab03: Atividade

Aluno(a): Evellyn Cruz Souza RA:823213551

Aluno(a): Amanda Agustinho Costa RA:823150503

Aluno(a): João Pedro Agustinho Costa RA:823223417  
Disciplina: Gestão e Qualidade de Software (GQS)  
Professor: Prof. Calvetti

## Atividades – Testes Unitários

### Atividade 1 – Função is\_par(n)

Implementar uma função is\_par(n) que retorna True se n for par. A seguir, os testes unitários para diferentes entradas:

def is\_par(n):  
 return n % 2 == 0  
  
import unittest  
  
class TestIsPar(unittest.TestCase):  
 def test\_numero\_par(self):  
 self.assertTrue(is\_par(4))  
 def test\_numero\_impar(self):  
 self.assertFalse(is\_par(5))  
 def test\_zero(self):  
 self.assertTrue(is\_par(0))  
 def test\_numero\_negativo(self):  
 self.assertTrue(is\_par(-2))

### Atividade 2 – Função fatorial(n)

def fatorial(n):  
 if n < 0:  
 raise ValueError("n deve ser >= 0")  
 if n == 0:  
 return 1  
 result = 1  
 for i in range(1, n+1):  
 result \*= i  
 return result  
  
class TestFatorial(unittest.TestCase):  
 def test\_fatorial\_zero(self):  
 self.assertEqual(fatorial(0), 1)  
 def test\_fatorial\_5(self):  
 self.assertEqual(fatorial(5), 120)  
 def test\_fatorial\_negativo(self):  
 with self.assertRaises(ValueError):  
 fatorial(-1)

### Atividade 3 – Classe Conta

class InsufficientFunds(Exception):  
 pass  
  
class Conta:  
 def \_\_init\_\_(self, saldo=0):  
 self.saldo = saldo  
   
 def depositar(self, amount):  
 if amount <= 0:  
 raise ValueError("Depósito deve ser positivo")  
 self.saldo += amount  
   
 def sacar(self, amount):  
 if amount <= 0:  
 raise ValueError("Saque deve ser positivo")  
 if amount > self.saldo:  
 raise InsufficientFunds("Saldo insuficiente")  
 self.saldo -= amount  
  
class TestConta(unittest.TestCase):  
 def test\_deposito(self):  
 conta = Conta(100)  
 conta.depositar(50)  
 self.assertEqual(conta.saldo, 150)  
   
 def test\_saque\_sucesso(self):  
 conta = Conta(100)  
 conta.sacar(40)  
 self.assertEqual(conta.saldo, 60)  
   
 def test\_saque\_insuficiente(self):  
 conta = Conta(50)  
 with self.assertRaises(InsufficientFunds):  
 conta.sacar(100)  
   
 def test\_entrada\_invalida(self):  
 conta = Conta(100)  
 with self.assertRaises(ValueError):  
 conta.depositar(-10)

### Atividade 4 – Função buscar\_clima(cidade) com Mock

import requests  
  
def buscar\_clima(cidade):  
 r = requests.get(f"https://api.exemplo/clima?cidade={cidade}")  
 data = r.json()  
 if "temperatura" not in data:  
 raise ValueError("Resposta inválida da API")  
 return data["temperatura"]  
  
from unittest.mock import patch, Mock  
  
class TestBuscarClima(unittest.TestCase):  
 @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  
 self.assertEqual(buscar\_clima("São Paulo"), 25)  
   
 @patch('requests.get')  
 def test\_buscar\_clima\_resposta\_invalida(self, mock\_get):  
 mock\_resp = Mock()  
 mock\_resp.json.return\_value = {}  
 mock\_get.return\_value = mock\_resp  
 with self.assertRaises(ValueError):  
 buscar\_clima("São Paulo")