

MATÉRIA - GESTÃO E QUALIDADE DE SOFTWARE – GQS

PROFESSOR - *Calvetti*

ALUNO - João Luiz da Silva – RA 82420546

Atividade 4 testes unitários:

```
1. # =====
# Atividade 1 - Testes Unitários em Python (Colab)
# UC: Gestão e Qualidade de Software - GQS
# =====
```

Utilizando Python e o Colab, crie testes unitários para os seguintes problemas:

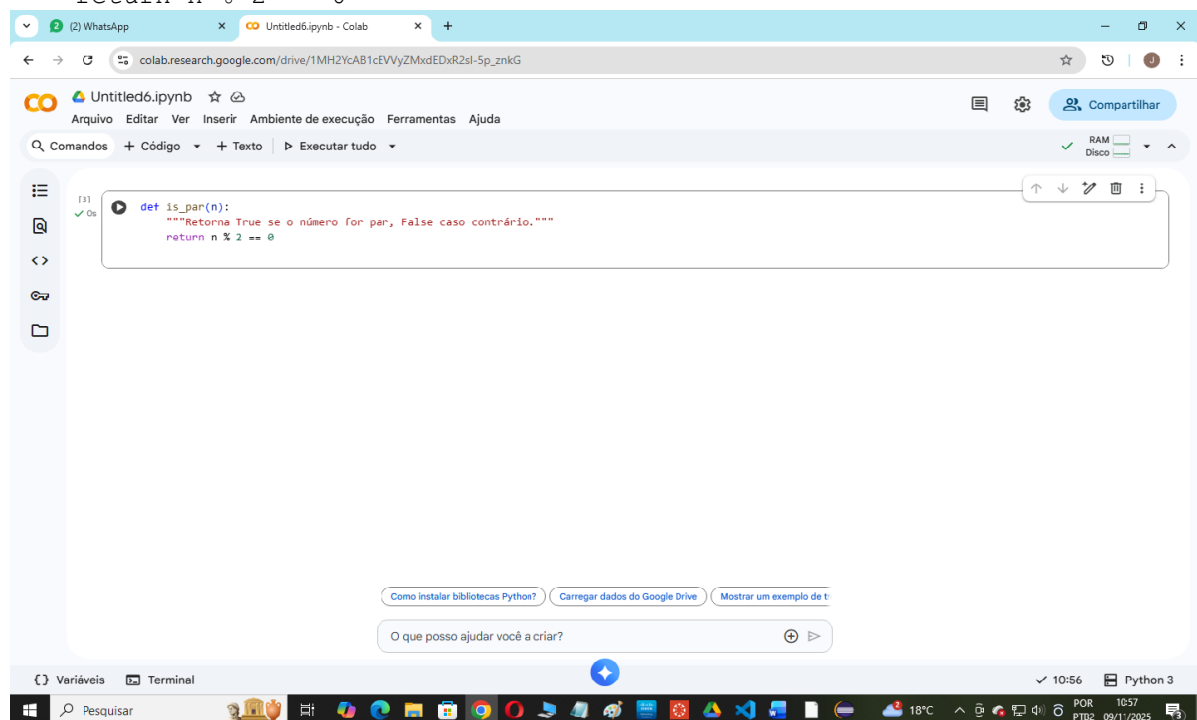
1. Escreva função `is_par(n)` que retorna `True` se `n` for par;
Testes: números par, ímpar, zero, número negativo.

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`.

Importando o módulo de testes do Python
`import unittest`

```
# -----
# 1. Função is_par(n): retorna True se n for par
# -----
```

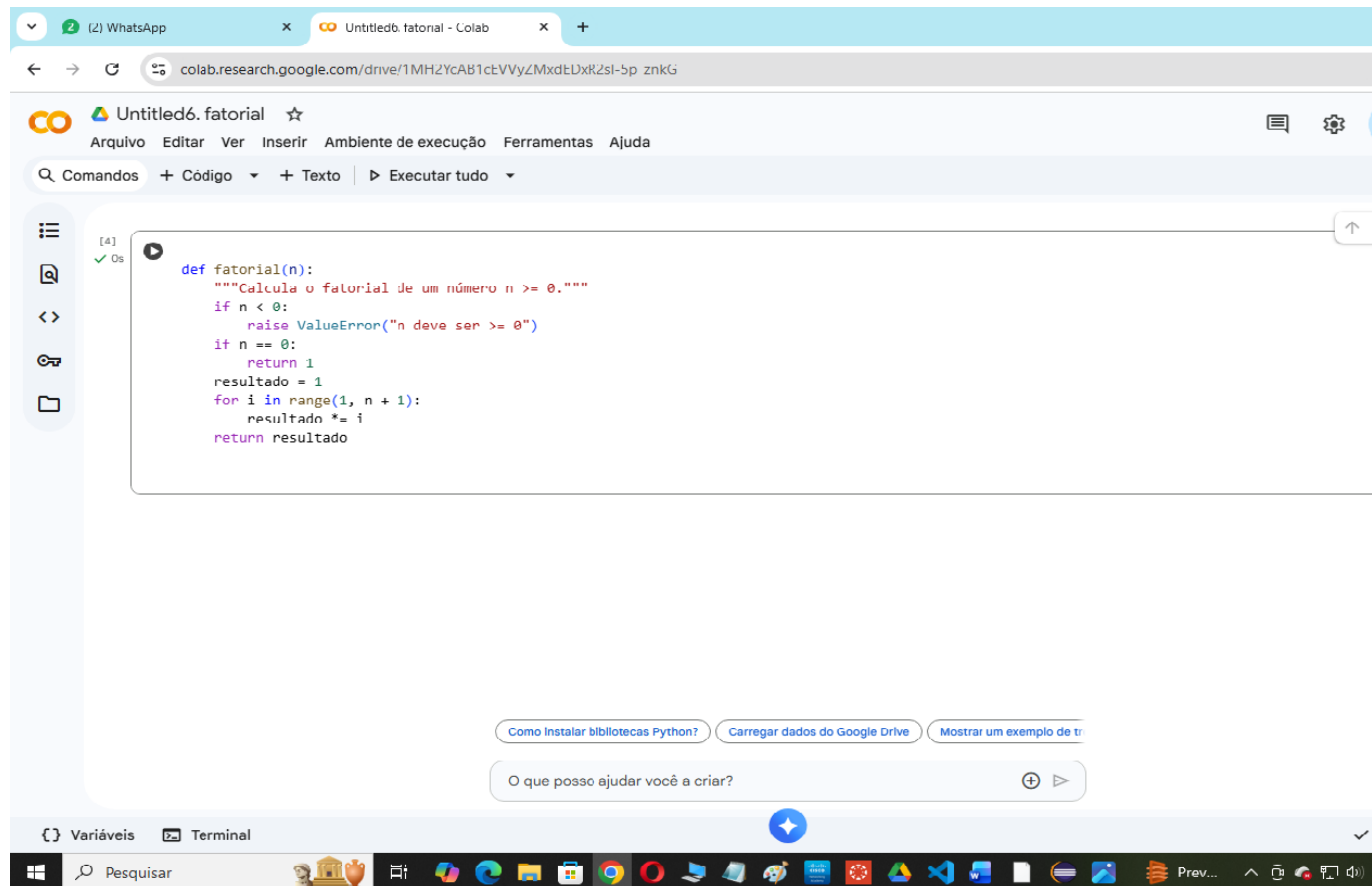
```
def is_par(n):
    """Retorna True se o número for par, False caso contrário."""
    return n % 2 == 0
```



Função Fatorial

```
# -----
# 2. Função fatorial(n)
# Retorna o fatorial de n se n >= 0
```

```
# Lança ValueError se n < 0
# -----
def fatorial(n):
    """Calcula o fatorial de um número n >= 0."""
    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
```



Método Depositar

Classe *Conta* com método *sdepositar (amount)* e *sacar (amount)*.

Levanta *ValueError* em valores negativos e *InsufficientFunds* quando o tentarem sacar mais do que tem;

Testes: depósito, saque com sucesso, saque insuficiente, entradas inválidas.

```
class InsufficientFunds(Exception):
    """Exceção para saldo insuficiente."""
    pass
```

```
class Conta:
```

```

def __init__(self, saldo_inicial=0):
    if saldo_inicial < 0:
        raise ValueError("Saldo inicial não pode ser negativo.")
    self.saldo = saldo_inicial

def depositar(self, amount):
    if amount <= 0:
        raise ValueError("Valor do depósito deve ser positivo.")
    self.saldo += amount

def sacar(self, amount):
    if amount <= 0:
        raise ValueError("Valor do saque deve ser positivo.")
    if amount > self.saldo:
        raise InsufficientFunds("Saldo insuficiente para saque.")
    self.saldo -= amount

```

colab.research.google.com/drive/14xdYQkmia1foubCV4eZ9RFdq3WMgbow?authuser=0

Untitled13.ipynb

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda

Comandos + Código + Texto ▶ Executar tudo

```

[1] ✓ Os
class InsufficientFunds(Exception):
    """Exceção para saldo insuficiente."""
    pass

class Conta:
    def __init__(self, saldo_inicial=0):
        if saldo_inicial < 0:
            raise ValueError("Saldo inicial não pode ser negativo.")
        self.saldo = saldo_inicial

    def depositar(self, amount):
        if amount <= 0:
            raise ValueError("Valor do depósito deve ser positivo.")
        self.saldo += amount

    def sacar(self, amount):
        if amount <= 0:
            raise ValueError("Valor do saque deve ser positivo.")
        if amount > self.saldo:
            raise InsufficientFunds("Saldo insuficiente para saque.")
        self.saldo -= amount

```

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 valide que sua função trata corretamente as respostas e exceções (ex.: quando `json()` não contém temperatura).

Implementação da função:

```
import requests
```

```
def buscar_clima(cidade):
    """
    Busca a temperatura atual de uma cidade na API fictícia.
    """
    url = f"https://api.exemplo/clima?cidade={cidade}"
    try:
        resposta = requests.get(url, timeout=5)
        resposta.raise_for_status() # Levanta erro se status != 200
        dados = resposta.json()
        if "temperatura" not in dados:
            raise KeyError("Campo 'temperatura' ausente na resposta.")
        return dados["temperatura"]
    except requests.RequestException as e:
        raise ConnectionError(f"Erro de conexão: {e}")
    except KeyError as e:
        raise ValueError(f"Resposta inválida: {e}")
```

Testes com `unittest` e `unittest.mock`:

```
import unittest
from unittest.mock import patch, Mock
from buscar_clima import buscar_clima # supondo que esteja em
buscar_clima.py
```

```
class TestBuscarClima(unittest.TestCase):
```

```
    @patch("buscar_clima.requests.get")
    def test_buscar_clima_sucesso(self, mock_get):
        # Simula resposta JSON válida
        mock_response = Mock()
        mock_response.json.return_value = {"temperatura": 25}
        mock_response.raise_for_status.return_value = None
        mock_get.return_value = mock_response

        temp = buscar_clima("São Paulo")
        self.assertEqual(temp, 25)
        mock_get.assert_called_once()

    @patch("buscar_clima.requests.get")
    def test_resposta_sem_temperatura(self, mock_get):
        mock_response = Mock()
        mock_response.json.return_value = {"umidade": 80}
```

```
mock_response.raise_for_status.return_value = None
mock_get.return_value = mock_response
```

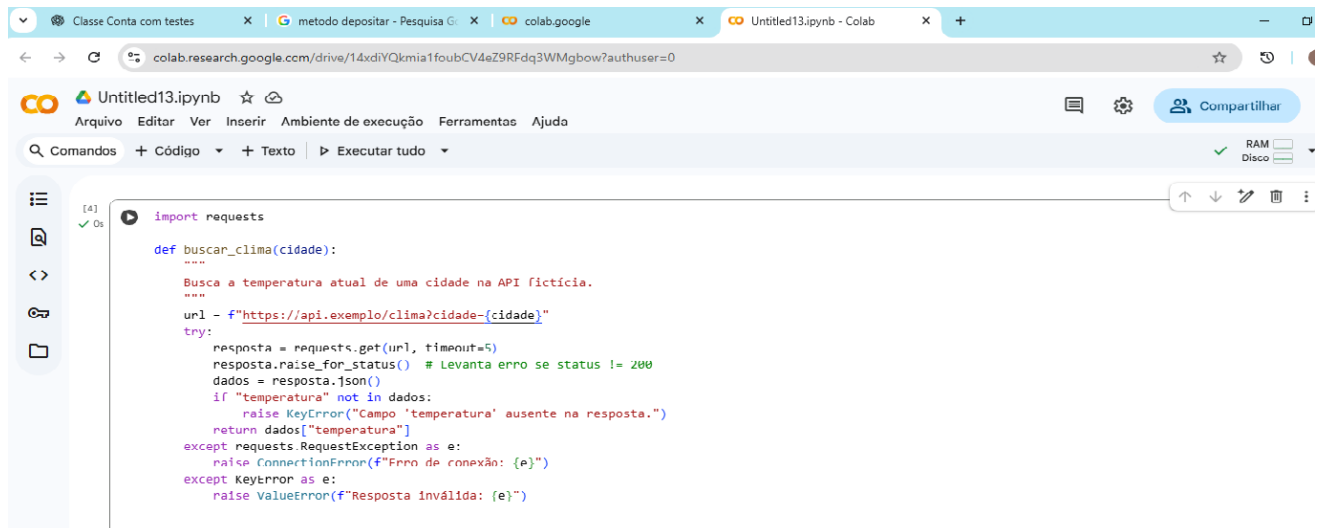
```
with self.assertRaises(ValueError):
    buscar_clima("Curitiba")
```

```
@patch("buscar_clima.requests.get")
def test_erro_http(self, mock_get):
    mock_response = Mock()
    mock_response.raise_for_status.side_effect =
requests.HTTPError("Erro 404")
    mock_get.return_value = mock_response
```

```
with self.assertRaises(ConnectionError):
    buscar_clima("Rio de Janeiro")
```

```
@patch("buscar_clima.requests.get")
def test_erro_conexao(self, mock_get):
    mock_get.side_effect = requests.ConnectionError("Falha na rede")
    with self.assertRaises(ConnectionError):
        buscar_clima("Salvador")
```

```
if __name__ == "__main__":
    unittest.main()
```

The image shows a Google Colab notebook interface. The browser tabs at the top include 'Classe Conta com testes', 'metodo depositar - Pesquisa G...', 'colab.google', and 'Untitled13.ipynb - Colab'. The address bar shows the Colab URL. The notebook's toolbar includes options for 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. The code editor displays the following Python code:

```
[a]
✓ Os
import requests

def buscar_clima(cidade):
    """
    Busca a temperatura atual de uma cidade na API fictícia.
    """
    url = f"https://api.exemplo/clima?cidade={cidade}"
    try:
        resposta = requests.get(url, timeout=5)
        resposta.raise_for_status() # Levanta erro se status != 200
        dados = resposta.json()
        if "temperatura" not in dados:
            raise KeyError("Campo 'temperatura' ausente na resposta.")
        return dados["temperatura"]
    except requests.RequestException as e:
        raise ConnectionError(f"Erro de conexão: {e}")
    except KeyError as e:
        raise ValueError(f"Resposta inválida: {e}")
```