

Atividades - Refatoração

Nome: Arthur Salatine de Moraes

RA: 822151203

1. Código Simples e Verboso

Objetivo: Simplificar um código redundante.

Tarefa: Refatorar para torná-lo mais legível e Pythonic.

Código original:

```
def calcular_media(lista):  
    soma = 0  
    for i in range(0, len(lista)):  
        soma = soma + lista[i]  
    media = soma / len(lista)  
    return media
```

Gabarito:

```
def calcular_media(valores):  
    """Calcula a média de uma lista numérica"""  
    return sum(valores) / len(valores)
```

Conceito aplicado: Simplificação de loop, nomes significativos e docstring.

2. Duplicação de Código

Objetivo: Remover duplicação;

Tarefa: Eliminar duplicação, mantendo o mesmo comportamento.

Código original:

```
def preco_com_desconto(preco, categoria):  
    if categoria == "A":  
        return preco * 0.9  
    if categoria == "B":  
        return preco * 0.85  
    if categoria == "C":
```

```
return preco * 0.8
```

Gabarito:

```
def preco_com_desconto(preco, categoria):  
    """Aplica descontos conforme categoria"""  
    descontos = {"A": 0.9, "B": 0.85, "C": 0.8}  
    return preco * descontos.get(categoria, 1)
```

Conceito aplicado: Replace conditional com Lookup table.

3. Função Longa

Objetivo: Quebrar uma função em partes menores.

Código original:

```
def processar_pedido(pedido):  
    print("Iniciando pedido", pedido["id"])  
    total = 0  
    for item in pedido["itens"]:  
        if item["categoria"] == "eletronico":  
            total += item["preco"] * 1.2  
        else:  
            total += item["preco"]  
    print("Total:", total)  
    if total > 1000:  
        print("Aplicando desconto especial")
```

Gabarito:

```
def calcular_total(itens):  
    """Calcula total considerando categoria"""  
    return sum(  
        item["preco"] * (1.2 if item["categoria"] == "eletronico" else 1)  
        for item in itens  
    )
```

```
def processar_pedido(pedido):  
    """Processa pedido e aplica descontos"""  
    print(f"Iniciando pedido {pedido['id']}")  
    total = calcular_total(pedido["itens"])  
    print("Total:", total)  
    if total > 1000:
```

```
print("Aplicando desconto especial")
```

Conceito aplicado: Extract Method.

4. Melhorando Nomes

Objetivo: Substituir nomes genéricos por descritivos.

Tarefa: Renomeie variáveis e função para expressar intenção.

Código original:

```
def f(a, b):  
    return a * b + (a + b)
```

Gabarito:

```
def calcular_valor_total(preco, quantidade):  
    """Retorna o valor total considerando produto e quantidade"""  
    return preco * quantidade + (preco + quantidade)
```

Conceito aplicado: Rename Variable / Function.

5. Simplificação de Condicional

Objetivo: Simplificar estruturas if desnecessárias.

Tarefa: Refatorar utilizando operador ternário.

Código original:

```
if idade < 18:  
    status = "menor"  
else:  
    status = "maior"
```

Gabarito:

```
status = "menor" if idade < 18 else "maior"
```

Conceito aplicado: Simplify Conditional Expression.

6. Eliminando Código Repetido

Objetivo: Evitar repetição direta de prints.

Tarefa: Refatorar utilizando estrutura de repetição.

Código original:

```
print("Iniciando...")
print("Executando tarefa...")
print("Finalizando...")
```

Gabarito:

```
for etapa in ["Iniciando", "Executando tarefa", "Finalizando"]:
    print(f"{etapa}...")
```

Conceito aplicado: Replace Repetition with Loop.

7. Eliminação de Números Mágicos

Objetivo: Substituir valores fixos (magic numbers) por constantes nominais.

Código original:

```
preco_final = preco * 1.07
```

Gabarito:

```
TAXA_IMPOSTO = 0.07
preco_final = preco * (1 + TAXA_IMPOSTO)
```

Conceito aplicado: Introduce Constant.

8. Código Mal Estruturado

Objetivo: Refatorar para clareza e reutilização.

Código original:

```
def calcular_bonus(funcionario):
    if funcionario["cargo"] == "gerente":
        bonus = funcionario["salario"] * 0.2
    else:
        bonus = funcionario["salario"] * 0.1
    return funcionario["salario"] + bonus
```

Gabarito:

```
def obter_taxa_bonus(cargo):
    taxas = {"gerente": 0.2, "analista": 0.1}
    return taxas.get(cargo, 0.05)

def calcular_bonus(funcionario):
    taxa = obter_taxa_bonus(funcionario["cargo"])
    return funcionario["salario"] * (1 + taxa)
```

Conceito aplicado: Extract Function + Replace Conditional with Map.

9. Refatorando com TDD

Objetivo: TDD - 1. Escreva o teste antes; 2. Implemente; 3. Refatore.

Tarefa: Implementar e refatorar.

Código original (Teste esperado):

```
def test_soma():
    assert soma(2, 3) == 5
```

Gabarito:

```
def soma(a, b):
    return a + b
```

```
# Teste automatizado
def test_soma():
    assert soma(2, 3) == 5
```

Conceito aplicado: TDD + Refactoring Cycle.

Desafio Final

Seu código, sua refatoração.

Pegue um código seu; identifique 3 problemas estruturais; aplique refatorações, documentando cada alteração.

Critérios de Avaliação:

- Clareza e legibilidade;
- Redução de duplicação;
- Uso de boas práticas Python (PEP 8).

Código original:

```
def calcular_preco_final(produto, quantidade, desconto):
    if desconto > 0:
        preco = produto['preco'] * quantidade * (1 - desconto)
    else:
        preco = produto['preco'] * quantidade
    if quantidade > 10:
        preco = preco * 0.95
    return preco
```

Problemas identificados:

1. Números mágicos sem contexto;
2. Repetição na multiplicação do preço unitário vezes quantidade;
3. Falta de nomes/descriptivos, *docstrings* e validação de entrada.

Refatorações aplicadas (documentadas):

- 1) Introduzir constantes nominais para remover números mágicos;

- `BULK_THRESHOLD = 10` e `BULK_DISCOUNT = 0.05` (ou seja, 5% de desconto).

- 2) Extrair funções para responsabilidades únicas:

- `calcular_preco_base(preco_unit, quantidade, desconto)`;

- `aplicar_desconto_por_quantidade(preco, quantidade)`;

- 3) Melhorar nomes, adicionar docstrings e validação mínima.

Código refatorado:

```
# Constantes (eliminam números mágicos)
```

```
BULK_THRESHOLD = 10
```

```
BULK_DISCOUNT = 0.05
```

```
def calcular_preco_base(preco_unit, quantidade, desconto):
    """Retorna o preço base aplicando desconto percentual seguro (desconto >= 0)."""
    desconto = max(0.0, desconto)
    return preco_unit * quantidade * (1 - desconto)
```

```
def aplicar_desconto_por_quantidade(preco, quantidade):
    """Aplica desconto por quantidade quando acima do limite definido."""
    return preco * (1 - BULK_DISCOUNT) if quantidade > BULK_THRESHOLD else preco
```

```
def calcular_preco_final(produto: dict, quantidade: int, desconto: float) -> float:
    """Calcula o preço final de um produto dado preço unitário, quantidade e desconto
    percentual.
```

Exemplo:

```
    produto = {"preco": 100.0}
    calcular_preco_final(produto, 12, 0.1)
    """
    preco_unit = produto.get("preco", 0.0)
    base = calcular_preco_base(preco_unit, quantidade, desconto)
    return aplicar_desconto_por_quantidade(base, quantidade)
```