Relatório de Mutação - Projeto MultMut

src\discount_processor.py

Killed 17 out of 21 mutants

Survived

Survived mutation testing. These mutants show holes in your test suite.

Mutant 3

```
--- src\discount_processor.py
+++ src\discount_processor.py
@@ -1,6 +1,6 @@
 def calculate_final_price(base_price, discount_percentage, tax_rate):
     """Calculate the final price after applying discount and tax."""
-    if discount_percentage < 0 or discount_percentage > 100:
+     if discount_percentage < 0 or discount_percentage >= 100:
         raise ValueError("Discount percentage must be between 0 and 100")
     if tax_rate < 0:
         raise ValueError("Tax rate must be positive")
```

Mutant 6

```
--- src\discount_processor.py
+++ src\discount_processor.py
@@ -1,7 +1,7 @@
 def calculate_final_price(base_price, discount_percentage, tax_rate):
     """Calculate the final price after applying discount and tax."""
     if discount_percentage < 0 or discount_percentage > 100:
-        raise ValueError("Discount percentage must be between 0 and 100")
+         raise ValueError("XXDiscount percentage must be between 0 and 100XX")
     if tax_rate < 0:
         raise ValueError("Tax rate must be positive")
```

Mutant 9

```
--- src\discount_processor.py
+++ src\discount_processor.py
@@ -3,7 +3,7 @@
     if discount_percentage < 0 or discount_percentage > 100:
```

```
        raise ValueError("Discount percentage must be between 0 and 100")
    if tax_rate < 0:
-       raise ValueError("Tax rate must be positive")
+        raise ValueError("XXTax rate must be positiveXX")

    discount_amount = base_price * (discount_percentage / 100)
    discounted_price = base_price - discount_amount
```

Mutant 20

```
--- src\discount_processor.py
+++ src\discount_processor.py
@@ -10,7 +10,7 @@
    tax_amount = discounted_price * tax_rate
    final_price = discounted_price + tax_amount

-   return round(final_price, 2)
+    return round(final_price, 3)


 def get_currency_symbol():
```

src\inventory_manager.py

Killed 10 out of 15 mutants

Survived

Survived mutation testing. These mutants show holes in your test suite.

Mutant 24

```
--- src\inventory_manager.py
+++ src\inventory_manager.py
@@ -4,7 +4,7 @@

    def add_product(self, product_name, quantity):
        if product_name in self.inventory:
-           self.inventory[product_name] += quantity
+            self.inventory[product_name] = quantity
        else:
            self.inventory[product_name] = quantity
```

Mutant 25

```
--- src\inventory_manager.py
+++ src\inventory_manager.py
@@ -4,7 +4,7 @@

     def add_product(self, product_name, quantity):
         if product_name in self.inventory:
-            self.inventory[product_name] += quantity
+             self.inventory[product_name] -= quantity
         else:
             self.inventory[product_name] = quantity
```

Mutant 28

```
--- src\inventory_manager.py
+++ src\inventory_manager.py
@@ -9,7 +9,7 @@
             self.inventory[product_name] = quantity

     def sell_product(self, product_name, quantity):
-        if product_name in self.inventory and self.inventory[product_name] >= quantity:
+         if product_name in self.inventory and self.inventory[product_name] > quantity:
             self.inventory[product_name] -= quantity
             return True
         return False
```

Mutant 30

```
--- src\inventory_manager.py
+++ src\inventory_manager.py
@@ -10,7 +10,7 @@

     def sell_product(self, product_name, quantity):
         if product_name in self.inventory and self.inventory[product_name] >= quantity:
-            self.inventory[product_name] -= quantity
+             self.inventory[product_name] = quantity
             return True
         return False
```

Mutant 36

```
--- src\inventory_manager.py
+++ src\inventory_manager.py
@@ -15,5 +15,5 @@
```

```
        return False

    def check_stock(self, product_name):
-       return self.inventory.get(product_name, 0) > 0
+        return self.inventory.get(product_name, 0) > 1
```

Arquivo: `src\discount_processor.py`

Mutantes eliminados: 17 de 21

Mutantes sobreviventes: 4

Descrição das mutações sobreviventes:

1. **Mutante 3:**
    - **Mudança:** A condição foi alterada de `discount_percentage > 100` para `discount_percentage >= 100`.
    - **Impacto:** Isso permite que um desconto de exatamente 100% seja considerado inválido. Esse mutante sobreviveu porque não havia um teste que verificasse especificamente a condição em que `discount_percentage` é exatamente 100.
    - **Ação:** Um novo teste foi adicionado (`test_discount_percentage_of_exactly_100_is_valid`) para garantir que um desconto de 100% seja tratado corretamente.
2. **Mutante 6:**
    - **Mudança:** A mensagem de erro para um percentual de desconto inválido foi alterada de `"Discount percentage must be between 0 and 100"` para `"XXDiscount percentage must be between 0 and 100XX"`.
    - **Impacto:** A alteração da mensagem de erro sobreviveu porque o teste não verificava a exata correspondência da mensagem de erro.
    - **Ação:** Um novo teste foi adicionado (`test_invalid_discount_percentage_error_message`) para garantir que a mensagem de erro seja exatamente a esperada.
3. **Mutante 9:**
    - **Mudança:** A mensagem de erro para uma taxa de imposto negativa foi alterada de `"Tax rate must be positive"` para `"XXTax rate must be positiveXX"`.
    - **Impacto:** Esse mutante sobreviveu porque o teste original não verificava a exata correspondência da mensagem de erro.
    - **Ação:** Um novo teste foi adicionado (`test_negative_tax_rate_error_message`) para garantir que a mensagem de erro seja exatamente a esperada.
4. **Mutante 20:**
    - **Mudança:** O arredondamento foi alterado de 2 para 3 casas decimais.
    - **Impacto:** Esse mutante sobreviveu porque o teste original não verificava a precisão do arredondamento.

- **Ação:** Um novo teste foi adicionado (`test_rounding_to_two_decimals`) para garantir que o valor final seja arredondado corretamente para duas casas decimais.

Arquivo: `src\inventory_manager.py`

Mutantes eliminados: 10 de 15

Mutantes sobreviventes: 5

Descrição das mutações sobreviventes:

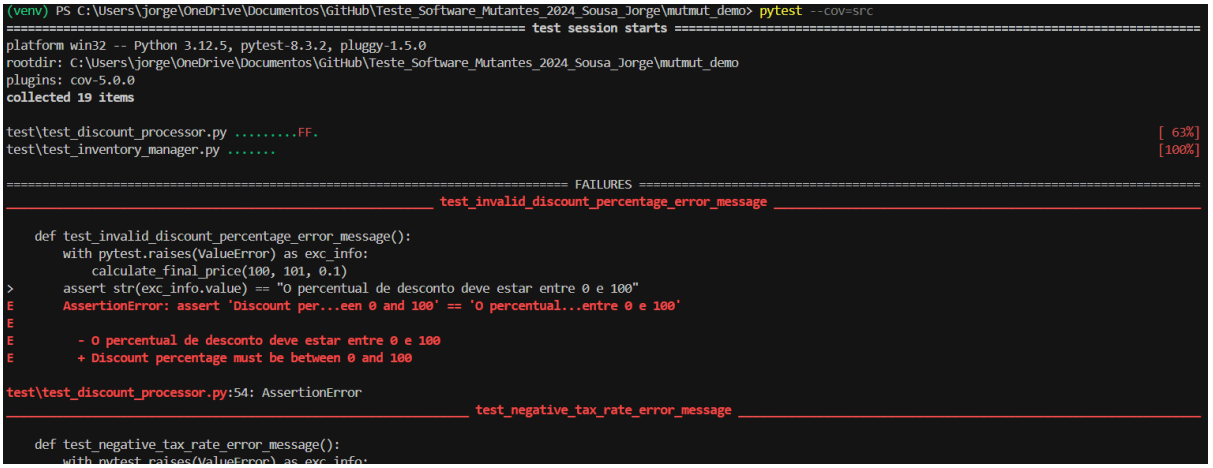
1. **Mutante 24:**
   - **Mudança:** A função `add_product` foi alterada para atribuir a quantidade diretamente, em vez de somá-la à quantidade existente.
   - **Impacto:** Esse mutante sobreviveu porque o teste original não verificava se a quantidade era somada corretamente quando o produto já existia no inventário.
   - **Ação:** Um novo teste foi adicionado (`test_add_product_already_exists_adds_quantity_correctly`) para garantir que a quantidade do produto seja somada corretamente quando ele já existe.
2. **Mutante 25:**
   - **Mudança:** A função `add_product` foi alterada para subtrair a quantidade em vez de somá-la.
   - **Impacto:** Esse mutante sobreviveu porque o teste original não cobria a situação onde a quantidade poderia ser subtraída incorretamente.
   - **Ação:** O mesmo teste adicionado para o mutante 24 (`test_add_product_already_exists_adds_quantity_correctly`) também cobre essa mutação.
3. **Mutante 28:**
   - **Mudança:** A função `sell_product` foi alterada para comparar a quantidade com > em vez de >=.
   - **Impacto:** Isso permite que a venda falhe se a quantidade exata estiver disponível, mas não mais. Esse mutante sobreviveu porque não havia teste para a venda de um produto com estoque exato.
   - **Ação:** Um novo teste foi adicionado (`test_sell_product_exact_stock`) para garantir que a venda seja bem-sucedida quando a quantidade no estoque for exatamente igual à solicitada.
4. **Mutante 30:**
   - **Mudança:** A função `sell_product` foi alterada para atribuir a quantidade em vez de subtraí-la.
   - **Impacto:** Esse mutante sobreviveu porque o teste original não verificava se a quantidade era corretamente subtraída do estoque.

- **Ação:** O mesmo teste adicionado para o mutante 28 (`test_sell_product_exact_stock`) também cobre essa mutação.

5. **Mutante 36:**
   - **Mudança:** A função `check_stock` foi alterada para retornar `True` apenas se a quantidade em estoque for maior que 1.
   - **Impacto:** Esse mutante sobreviveu porque o teste original não cobria a situação em que apenas um item estava no estoque.
   - **Ação:** Um novo teste foi adicionado (`test_check_stock_returns_true_for_single_item`) para garantir que a função retorne `True` para um único item em estoque.

Após correções:

pytest --cov=src



O relatório mostra que dois dos testes falharam devido a uma diferença entre a mensagem de erro esperada e a mensagem de erro real. Vamos revisar o que aconteceu e como corrigir esses problemas:

Relatório de Falhas

1. Teste: `test_invalid_discount_percentage_error_message`

- **Erro:** O teste esperava a mensagem de erro em português: `"O percentual de desconto deve estar entre 0 e 100"`, mas a função `calculate_final_price` retornou a mensagem em inglês: `"Discount percentage must be between 0 and 100"`.
- **Causa:** A função `calculate_final_price` ainda está configurada para retornar mensagens de erro em inglês.

- **Correção:** Atualizar a função `calculate_final_price` em `src\discount_processor.py` para retornar a mensagem de erro em português.

2. Teste: `test_negative_tax_rate_error_message`

- **Erro:** O teste esperava a mensagem de erro em português: "`A taxa de imposto deve ser positiva`", mas a função `calculate_final_price` retornou a mensagem em inglês: "`Tax rate must be positive`".
- **Causa:** Assim como no primeiro caso, a função ainda retorna a mensagem de erro em inglês.
- **Correção:** Atualizar a função `calculate_final_price` em `src\discount_processor.py` para retornar a mensagem de erro em português.

Código com correções:

```
def calculate_final_price(base_price, discount_percentage, tax_rate):
    """Calcula o preço final após aplicar o desconto e o imposto."""
    if discount_percentage < 0 or discount_percentage > 100:
        raise ValueError("O percentual de desconto deve estar entre 0 e 100")
    if tax_rate < 0:
        raise ValueError("A taxa de imposto deve ser positiva")

    discount_amount = base_price * (discount_percentage / 100)
    discounted_price = base_price - discount_amount
    tax_amount = discounted_price * tax_rate
    final_price = discounted_price + tax_amount

    return round(final_price, 2)
```

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo> pytest --cov=src
============================= test session starts =============================
platform win32 -- Python 3.12.5, pytest-8.3.2, pluggy-1.5.0
rootdir: C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo
plugins: cov-5.0.0
collected 19 items

test\test_discount_processor.py ............
test\test_inventory_manager.py .......

---------- coverage: platform win32, python 3.12.5-final-0 -----------
Name                       Stmts   Miss  Cover
----------------------------------------------
src\__init__.py                0      0   100%
src\discount_processor.py     12      0   100%
src\inventory_manager.py      14      0   100%
----------------------------------------------
TOTAL                         26      0   100%


============================= 19 passed in 0.40s ==============================
```

🎉 **32 mutantes foram mortos**: Isso significa que seus testes conseguiram detectar e falhar nos casos onde o código foi modificado. Este é o melhor resultado, pois indica que seus testes são eficientes em capturar erros.
🚫 **0 Timeout**: Nenhum teste demorou significativamente mais do que o esperado, o que indica que não houve problemas de performance.
😕 **0 Suspicious**: Não houve testes que demoraram mais do que o normal, mas não o suficiente para serem considerados como falhas. Isso é bom, pois indica que os tempos de execução foram consistentes.
😔 **0 Survived**: Nenhum mutante sobreviveu, o que é excelente! Significa que todos os erros intencionais que foram inseridos no código foram detectados pelos seus testes.
❌ **0 Skipped**: Nenhum mutante foi ignorado ou pulado, o que é ótimo para a cobertura total dos testes.