

Atividade 2 – Testes de Mutação

O repositório da atividade é o

https://github.com/Jorgelucasmota/Teste_Software_Mutantes_2024_Sousa_Jorge

Primeiro, eu assisti ao vídeo  Introdução ao teste de mutação em Python com a ferramenta mut...

- Passo a passo

1. Instalação e Preparação do Ambiente

O primeiro passo foi configurar o ambiente de desenvolvimento. Seguindo as instruções do vídeo:

1.1 Instalação do ambiente virtual do Python

O ministrante utilizou o comando ``sudo apt install python3-venv`` para instalar o ambiente virtual do Python.

1.2 Navegação até o diretório do projeto

Depois, foi necessário acessar o diretório onde o projeto está localizado com o comando ``cd detect-test-pollution-main``.

1.3 Criação e ativação do ambiente virtual

Para isolar as dependências do projeto, foi criado e ativado um ambiente virtual com os seguintes comandos:

- ``python3 -m venv ./.venv`` para criar o ambiente virtual.
- ``source ./.venv/bin/activate`` para ativá-lo.

1.4 Instalação das dependências do projeto

Após a ativação do ambiente virtual, as dependências do projeto foram instaladas utilizando o comando ``pip install -r requirements.txt``.

2. Execução dos Testes

Com o ambiente devidamente configurado, o próximo passo foi a execução dos testes:

2.1 Execução dos testes normalmente

Foi utilizado o comando ``pytest -vv test_cal.py`` para rodar os testes unitários.

2.2 Execução dos testes com relatório de cobertura de linha (line coverage)

Para verificar a cobertura de código, o comando utilizado foi ``pytest -vv test_cal.py --cov=cal``.

2.3 Execução dos testes com relatório de cobertura de ramos (branch coverage)

Além da cobertura de linhas, também foi verificada a cobertura de ramos, com o comando ``pytest -vv test_cal.py --cov=cal --cov-branch --cov-report html``, que gera um relatório em formato HTML.

3. Testes de Mutação com Mutmut

Para garantir a qualidade dos testes, o vídeo ensinou a realizar testes de mutação utilizando o Mutmut:

3.1 Execução do Mutmut

O comando ``mutmut run --paths-to-mutate=../detect-test-pollution-main/cal.py`` foi utilizado para rodar os testes de mutação.

3.2 Verificação dos resultados dos testes de mutação

Após a execução, os resultados dos testes de mutação foram verificados com o comando ``mutmut results``.

3.3 Exibição de detalhes de um resultado específico de mutação

Caso fosse necessário analisar um resultado específico, o comando ``mutmut show [id]`` permitia exibir detalhes sobre a mutação em questão.

3.4 Geração de um relatório HTML dos testes de mutação

Por fim, foi possível gerar um relatório em HTML dos testes de mutação utilizando o comando ``mutmut html``.

Com esses passos, o vídeo guiou a configuração completa do ambiente e a execução de testes, garantindo a qualidade do código através de testes unitários, cobertura de código e testes de mutação.

- Seleção do repositório

Selecionei o repositório: [GitHub - hernanzini/mutmut_demo: mutmut tool testing](https://github.com/hernanzini/mutmut_demo) porque ele oferecia a possibilidade de aplicar os conceitos aprendidos no vídeo.

Este repositório foi selecionado porque já era um exemplo pronto para demonstrar o uso de testes de mutação em Python utilizando o mutmut. O projeto mostra como os testes de mutação podem avaliar a qualidade dos seus casos de teste, introduzindo mudanças (mutações) no código e verificando se os testes conseguem detectar essas mudanças.

- Estrutura do repositório

mutmut_demo/: Diretório raiz do projeto.

- **src/**: Diretório que contém o código fonte do projeto.
 - **discount_processor.py**: Contém a lógica para calcular o preço final de um produto aplicando descontos e impostos. Também inclui uma função para obter o símbolo da moeda.
 - **inventory_manager.py**: Contém a implementação da classe `InventoryManager`, que gerencia o estoque de produtos, permitindo adicionar, vender e verificar a disponibilidade de produtos.
- **test/**: Diretório que contém os testes para o código-fonte.
 - **test_discount_processor.py**: Contém testes unitários para as funções do `discount_processor.py`, utilizando o framework `pytest`. Testa cenários como cálculo do preço final, validação de desconto e taxa de imposto.
 - **test_inventory_manager.py**: Contém testes unitários para a classe `InventoryManager`, verificando se as operações de adicionar produtos, vender produtos, e checar o estoque funcionam como esperado.
- **.gitignore**: Arquivo que especifica quais arquivos e pastas devem ser ignorados pelo Git (por exemplo, arquivos temporários, logs, etc.).
- **mutmut.sh**: Script possivelmente usado para rodar o MutMut, uma ferramenta de testes de mutação para Python.
- **Pipfile**: Arquivo que define as dependências do projeto e o ambiente virtual, gerenciado pelo `Pipenv`.
- **README.md**: Arquivo de texto em Markdown, geralmente contendo uma descrição do projeto, instruções de instalação, uso, e outras informações relevantes.

- Descrição do Código

discount_processor.py

- **calculate_final_price(base_price, discount_percentage, tax_rate)**:
 - Calcula o preço final de um produto após aplicar um desconto e adicionar imposto.

- Valida o percentual de desconto para garantir que esteja entre 0 e 100, e valida que a taxa de imposto não seja negativa.
- Retorna o preço final arredondado para duas casas decimais.
- **get_currency_symbol()**:
 - Retorna o símbolo da moeda "USD\$". O comentário # pragma: no mutate indica que essa linha não deve ser alterada nos testes de mutação.

inventory_manager.py

- **InventoryManager**:
 - Classe para gerenciar o inventário de produtos.
 - **__init__**: Inicializa o inventário como um dicionário vazio.
 - **add_product(product_name, quantity)**: Adiciona uma quantidade específica de um produto ao inventário. Se o produto já existir, soma a quantidade ao estoque existente.
 - **sell_product(product_name, quantity)**: Reduz a quantidade de um produto no inventário, se houver estoque suficiente. Retorna True se a venda for bem-sucedida, caso contrário, retorna False.
 - **check_stock(product_name)**: Verifica se há estoque de um produto específico. Retorna True se houver estoque, caso contrário, False.

test_discount_processor.py

- Testes unitários para o módulo discount_processor.py usando pytest.
 - **test_calculate_final_price**: Testa diferentes cenários para a função calculate_final_price, como casos triviais, padrão, sem desconto, e sem impostos.
 - **test_invalid_discount_percentage_raises_error**: Verifica se um ValueError é levantado ao passar um percentual de desconto inválido.
 - **test_negative_tax_rate_raises_error**: Verifica se um ValueError é levantado ao passar uma taxa de imposto negativa.
 - **test_get_currency_symbol**: Verifica se a função get_currency_symbol retorna corretamente "USD\$".

test_inventory_manager.py

- Testes unitários para o módulo inventory_manager.py usando pytest.
 - **test_add_product**: Verifica se a função add_product adiciona corretamente um novo produto ao inventário.
 - **test_sell_product**: Verifica se a função sell_product reduz corretamente a quantidade de um produto no inventário quando há estoque suficiente.
 - **test_sell_product_not_enough_stock**: Verifica se a função sell_product retorna False ao tentar vender mais do que há em estoque.

- **test_check_stock:** Verifica se a função `check_stock` retorna `True` quando há estoque de um produto, e `False` caso contrário.

- Aplicação do video no repositório

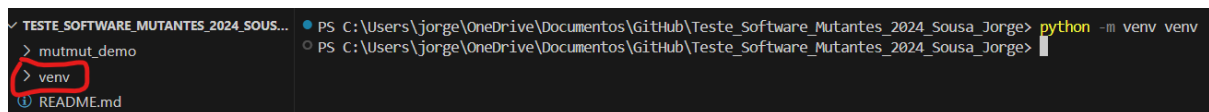
1. Instalação e Preparação do Ambiente

Para começar, é necessário configurar o ambiente de desenvolvimento. Siga os passos abaixo:

1.1 Instale o Ambiente Virtual do Python

No Windows, você pode criar um ambiente virtual com o seguinte comando:

```
python -m venv venv
```



```
TESTE_SOFTWARE_MUTANTES_2024_SOUS... PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge> python -m venv venv
> mutmut_demo PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge>
> venv
README.md
```

1.2 Clone o Repositório

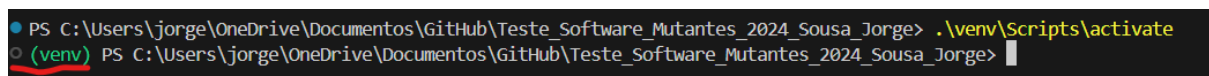
Clone o repositório do projeto escolhido:

```
git clone https://github.com/hernanzini/mutmut\_demo.git
```

1.3 Ative o Ambiente Virtual

Ative um ambiente virtual para isolar as dependências do projeto:

```
.\venv\Scripts\activate
```



```
PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge> .\venv\Scripts\activate
(venv) PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge>
```

1.4 Instale as Dependências

Instale as dependências necessárias para o projeto, incluindo `pytest`, `pytest-cov`, e `mutmut`:

```
pip install pytest pytest-cov mutmut
```

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge> pip install pytest pytest-cov mutmut
Collecting pytest
  Downloading pytest-8.3.2-py3-none-any.whl.metadata (7.5 kB)
Collecting pytest-cov
  Downloading pytest_cov-5.0.0-py3-none-any.whl.metadata (27 kB)
Collecting mutmut
  Downloading mutmut-2.5.1.tar.gz (50 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting iniconfig (from pytest)
  Downloading iniconfig-2.0.0-py3-none-any.whl.metadata (2.6 kB)
Collecting packaging (from pytest)
  Downloading packaging-24.1-py3-none-any.whl.metadata (3.2 kB)
```

1.5 Verificar as dependências

pip freeze

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge> pip freeze
click==8.1.7
colorama==0.4.6
coverage==7.6.1
glob2==0.7
iniconfig==2.0.0
junit-xml==1.9
mutmut==2.5.1
packaging==24.1
parso==0.8.4
pluggy==1.5.0
pony==0.7.19
pytest==8.3.2
pytest-cov==5.0.0
six==1.16.0
toml==0.10.2
```

2. Execução dos Testes

Com o ambiente configurado, você pode começar a executar os testes:

2.1 - Entre no projeto

cd mutmut_demo

2.2 Execute os Testes Unitários

Para garantir que o projeto está funcionando corretamente, execute os testes unitários:

pytest -v

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\Github\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo\test> pytest -v
===== test session starts =====
platform win32 -- Python 3.12.5, pytest-8.3.2, pluggy-1.5.0 -- C:\Users\jorge\OneDrive\Documentos\Github\Teste_Software_Mutantes_2024_Sousa_Jorge\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\jorge\OneDrive\Documentos\Github\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo\test
plugins: cov-5.0.0
collected 12 items

test_discount_processor.py::test_calculate_final_price[100-0-0-100] PASSED [ 8%]
test_discount_processor.py::test_calculate_final_price[100-10-0.1-99] PASSED [ 16%]
test_discount_processor.py::test_calculate_final_price[100-0-0.1-110] PASSED [ 25%]
test_discount_processor.py::test_calculate_final_price[100-10-0-90] PASSED [ 33%]
test_discount_processor.py::test_invalid_discount_percentage_raises_error[-1] PASSED [ 41%]
test_discount_processor.py::test_invalid_discount_percentage_raises_error[101] PASSED [ 50%]
test_discount_processor.py::test_negative_tax_rate_raises_error PASSED [ 58%]
test_discount_processor.py::test_get_currency_symbol PASSED [ 66%]
test_inventory_manager.py::test_add_product PASSED [ 75%]
test_inventory_manager.py::test_sell_product PASSED [ 83%]
test_inventory_manager.py::test_sell_product_not_enough_stock PASSED [ 91%]
test_inventory_manager.py::test_check_stock PASSED [100%]

===== 12 passed in 0.16s =====
```

2.3 Verifique a Cobertura de Código

Verifique a cobertura dos testes utilizando o pytest-cov:

pytest --cov=src

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\Github\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo\test> 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\test
plugins: cov-5.0.0
collected 12 items

test_discount_processor.py ..... [ 66%]
test_inventory_manager.py .... [100%]

----- coverage: platform win32, python 3.12.5-final-0 -----
Name                                                                    Stmts   Miss  Cover
-----
C:\Users\jorge\OneDrive\Documentos\Github\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo\src\__init__.py      0      0 100%
C:\Users\jorge\OneDrive\Documentos\Github\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo\src\discount_processor.py 12      0 100%
C:\Users\jorge\OneDrive\Documentos\Github\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo\src\inventory_manager.py 14      1   93%
TOTAL                                                                    26      1   96%

===== 12 passed in 0.20s =====
```

2.4 Relatório de Cobertura de Ramos

Para gerar um relatório de cobertura de ramos:

pytest --cov=src --cov-branch --cov-report html

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\Github\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo> pytest --cov=src --cov-branch --cov-report html
===== 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 12 items

test\test_discount_processor.py ..... [ 66%]
test\test_inventory_manager.py .... [100%]

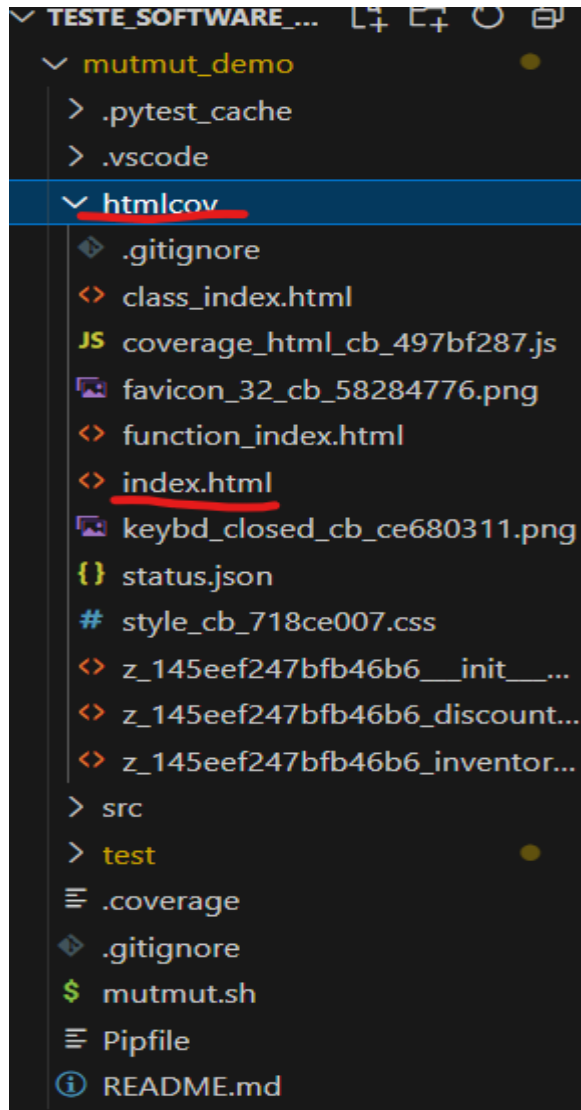
----- coverage: platform win32, python 3.12.5-final-0 -----
Coverage HTML written to dir htmlcov
```

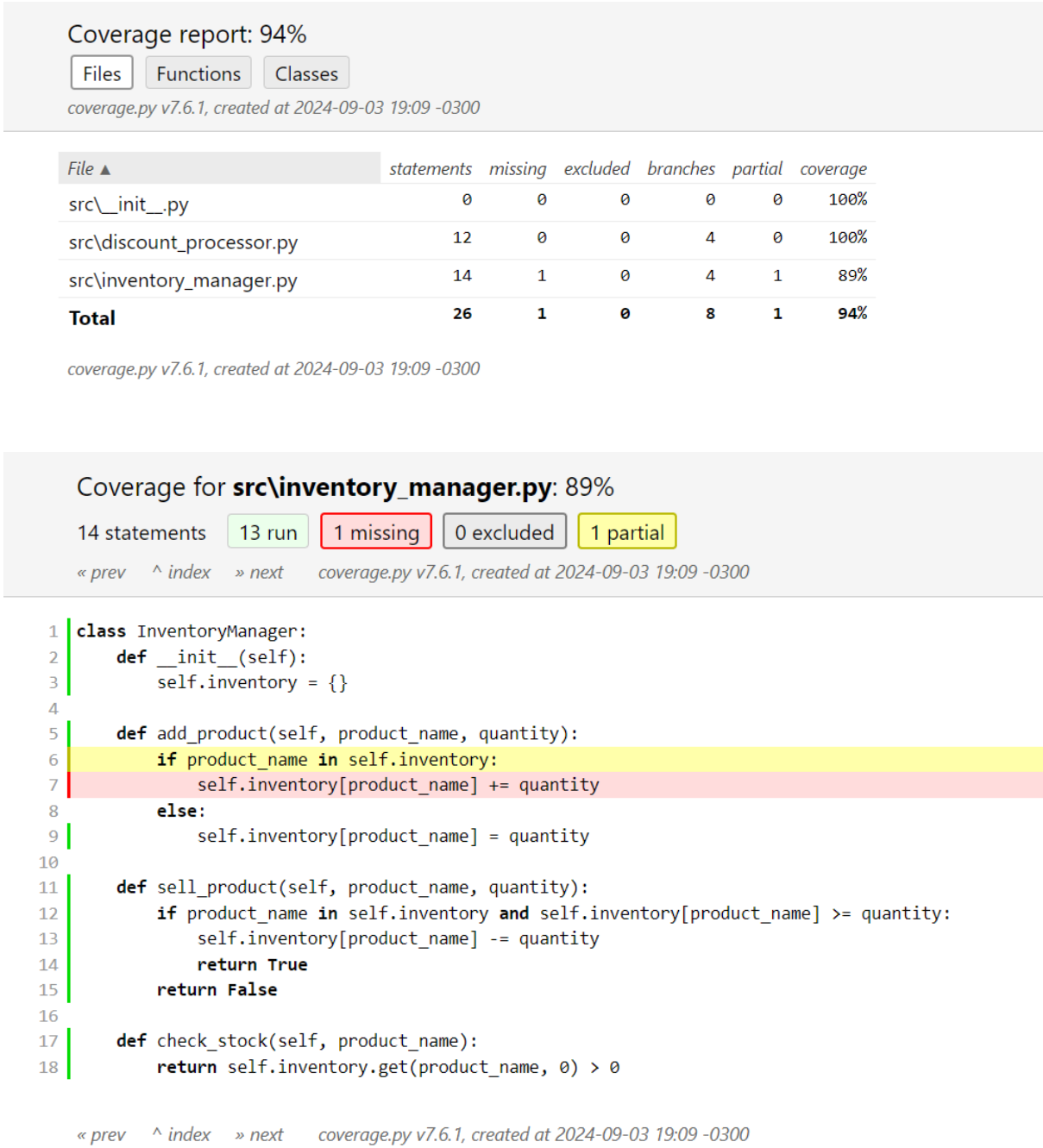
2.5 Analisar o relatório

A pasta htmlcov foi criada após a execução do comando dado no tópico 2.4, pytest

--cov=src --cov-branch --cov-report html. Esse comando gera um relatório de cobertura de testes no formato HTML, que permite visualizar de maneira detalhada quais partes do código foram cobertas pelos testes.

Dentro da pasta htmlcov, você encontrará diversos arquivos, mas o mais importante é o index.html. Esse arquivo é a porta de entrada para o relatório de cobertura.





Coverage for src\inventory_manager.py: 89%

14 statements

13 run

1 missing

0 excluded

1 partial

« prev ^ index » next coverage.py v7.6.1, created at 2024-09-03 19:09 -0300

```
1 class InventoryManager:
2     def __init__(self):
3         self.inventory = {}
4
5     def add_product(self, product_name, quantity):
6         if product_name in self.inventory:
7             self.inventory[product_name] += quantity
8         else:
9             self.inventory[product_name] = quantity
10
11     def sell_product(self, product_name, quantity):
12         if product_name in self.inventory and self.inventory[product_name] >= quantity:
13             self.inventory[product_name] -= quantity
14             return True
15         return False
16
17     def check_stock(self, product_name):
18         return self.inventory.get(product_name, 0) > 0
```

« prev ^ index » next coverage.py v7.6.1, created at 2024-09-03 19:09 -0300

Essa tela mostra o relatório de cobertura de código gerado pelo comando `pytest --cov=src --cov-branch --cov-report html`. A linha 7 não foi coberta por nenhum teste, indicando que a lógica de incrementar a quantidade de um produto já existente no inventário não foi testada. Para alcançar 100% de cobertura, é necessário criar um teste que adicione um produto que já está no inventário.

3. Testes de Mutação com Mutmut

Os testes de mutação ajudam a avaliar a eficácia dos casos de teste. Para realizá-los, siga os passos abaixo:

3.1 Execute o Mutmut

Inicie os testes de mutação:

mutmut run

```
🟡 Suspicious.      Tests took a long time, but not long enough to be fatal.
🟢 Survived.       This means your tests need to be expanded.
🔴 Skipped.        Skipped.

mutmut cache is out of date, clearing it...
1. Running tests without mutations
✔ Running...Done

2. Checking mutants
i: 0/36 🟡 0 🟢 0 🟡 0 🟡 0 🔴 0Process check_mutants:
Traceback (most recent call last):
  File "C:\Users\PC\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 314, in _bootstrap
    self.run()
  File "C:\Users\PC\AppData\Local\Programs\Python\Python311\Lib\multiprocessing\process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "C:\Users\PC\Documents\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\venv\Lib\site-packages\mutmut\_init_.py", line 749, in check_mutants
    status = run_mutation(context, feedback)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\PC\Documents\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\venv\Lib\site-packages\mutmut\_init_.py", line 794, in run_mutation
    survived = tests_pass(config=config, callback=callback)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\PC\Documents\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\venv\Lib\site-packages\mutmut\_init_.py", line 865, in tests_pass
    returncode = popen_streaming_output(config.test_command, callback, timeout=config.baseline_time_elapsed * 10)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\PC\Documents\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\venv\Lib\site-packages\mutmut\_init_.py", line 1051, in popen_streaming_output
    line = line.decode("utf-8")
           ^^^^^^^^^^^^^^^^^^^^^
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe1 in position 43: invalid continuation byte
```

Esse erro ocorre porque o mutmut está tentando decodificar um arquivo usando a codificação UTF-8, mas o arquivo contém caracteres que não estão nessa codificação, como um byte inválido em 0xe1. Para resolver esse problema, você precisa definir a variável de ambiente PYTHONIOENCODING para "utf-8" antes de executar o mutmut. Isso garantirá que o Python interprete corretamente o conteúdo dos arquivos com codificações mistas.

Para resolver isso com o seguinte comando

\$env:PYTHONIOENCODING="utf-8"; mutmut run

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo> $env:PYTHONIOENCODING="utf-8"; mutmut run

- Mutation testing starting -

These are the steps:
1. A full test suite run will be made to make sure we
   can run the tests successfully and we know how long
   it takes (to detect infinite loops for example)
2. Mutants will be generated and checked

Results are stored in .mutmut-cache.
Print found mutants with 'mutmut results'.

Legend for output:
🔴 Killed mutants.  The goal is for everything to end up in this bucket.
🟡 Timeout.       Test suite took 10 times as long as the baseline so were killed.
🟡 Suspicious.    Tests took a long time, but not long enough to be fatal.
🟢 Survived.      This means your tests need to be expanded.
🔴 Skipped.       Skipped.

mutmut cache is out of date, clearing it...
1. Running tests without mutations
✔ Running...Done

2. Checking mutants
i: 36/36 🟡 27 🟢 0 🟡 0 🟡 9 🔴 0
```

- 🕒 Timeout: 0 - Nenhum teste demorou demais, o que significa que não houve problemas como loops infinitos.
- 😟 Suspicious: 0 - Nenhum teste apresentou comportamento suspeito, como demorar mais do que o esperado.
- 💡 Survived: 9 - Esses mutantes sobreviveram, ou seja, os testes não foram suficientes para detectar essas falhas, sugerindo que os testes precisam ser melhorados.
- 🚫 Skipped: 0 - Nenhum mutante foi ignorado ou pulado durante o processo de verificação.

3.2 Verifique os Resultados dos Testes de Mutação

Para verificar os resultados dos testes de mutação:

mutmut results

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo> mutmut results
To apply a mutant on disk:
  mutmut apply <id>

To show a mutant:
  mutmut show <id>

Survived 😟 (9)

---- src\discount_processor.py (4) ----
3, 6, 9, 20

---- src\inventory_manager.py (5) ----
24-25, 28, 30, 36
```

3.3 Exiba Detalhes de um Resultado Específico de Mutação

Para ver detalhes de uma mutação específica:

mutmut show [id]

Demonstração mutmut show no id 3

mutmut show 3

```
(venv) PS C:\Users\jorge\OneDrive\Documentos\GitHub\Teste_Software_Mutantes_2024_Sousa_Jorge\mutmut_demo> mutmut show 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")
```

Código Original:

if discount_percentage < 0 or discount_percentage > 100:

```
raise ValueError("Discount percentage must be between 0 and 100")
```

Código Mutado:

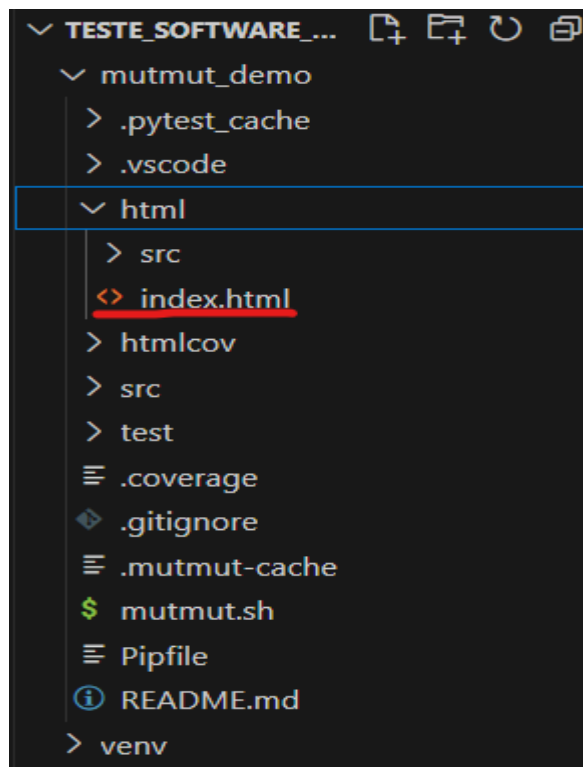
```
if discount_percentage < 0 or discount_percentage >= 100:  
    raise ValueError("Discount percentage must be between 0 and 100")
```

A mutação altera a condição original de `discount_percentage > 100` para `discount_percentage >= 100`. Isso significa que agora, além de valores acima de 100, o valor exato de 100 também será considerado inválido e levantará uma exceção.

3.4 Gere um Relatório HTML dos Testes de Mutação

Para gerar um relatório HTML detalhado dos testes de mutação:

```
mutmut html
```



Após a aplicação do comando `mutmut html`, a pasta `html` foi criada contendo o arquivo `index.html`. Esse arquivo é um relatório visual que mostra um resumo das mutações feitas no código, os resultados dos testes que detectaram ou não essas mutações, e detalhes sobre quais linhas do código foram alteradas.

Abra o arquivo HTML gerado no seu navegador: `index.html`

Mutation testing report

Killed 27 out of 36 mutants

File	Total	Skipped	Killed	% killed	Survived
src\discount_processor.py	21	0	17	80.95	4
src\inventory_manager.py	15	0	10	66.67	5

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
```

4. Análise e Melhorias

4.1 Análise dos Resultados

Após a execução dos testes de mutação, analise quais mutações não foram detectadas pelos testes.

4.2 Melhorias nos Casos de Teste

Ajuste ou adicione novos casos de teste para melhorar a cobertura e identificar mais mutantes.

4.3 Nova Execução dos Testes de Mutação

Repita os testes de mutação após as melhorias e compare os resultados:

```
pytest --cov=src  
mutmut run  
mutmut results
```

- Todos os passos de correção relatório estão em [☰ Relatório_mutmut_Jorge](#)