Universidade Federal de Sergipe - UFS
Centro de Ciências Exatas e Tecnologia - CCET
Departamento de Computação - DCOMP

Bruno Macedo da Silva

TESTE DE SOFTWARE

Atividade 2

São Cristóvão - SE
2024

Demonstração

Os arquivos estão disponíveis no repositório git no seguinte endereço: https://github.com/macedobruno/Teste_Software_Mutantes_2024_Silva_Bruno.git. Para continuar com o processo, é preciso que o sistema tenha primeiramente o *python* e *git* instalado. Os passos a seguir são necessários para utilizar os arquivos criados:

1. **Clonar repositório:**

Para clonar o repositório git, é preciso executar os seguintes comandos no prompt:

git clone https://github.com/macedobruno/Teste_Software_Mutantes_2024_Silva_Bruno.git

2. **Preparando ambiente virtual**

Para preparar o ambiente virtual, execute os seguintes comandos:

pip install python3-venv
python -m venv Teste_Software_Mutantes_2024_Silva_Bruno/
cd Teste_Software_Mutantes_2024_Silva_Bruno
source bin/activate

3. **Instalar dependências**

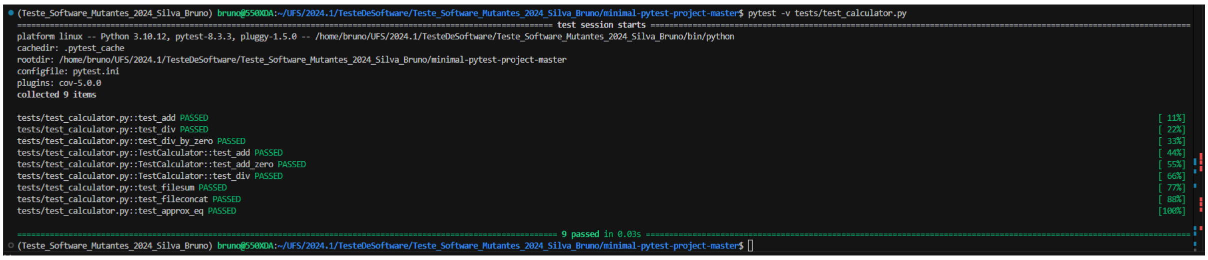
As dependências necessárias estão listadas no arquivo *requirements.txt*. Para instalar, basta executar o comando:

pip install -r requirements.txt
python minimal-pytest-project-master/setup.py install

4. **Testes iniciais**

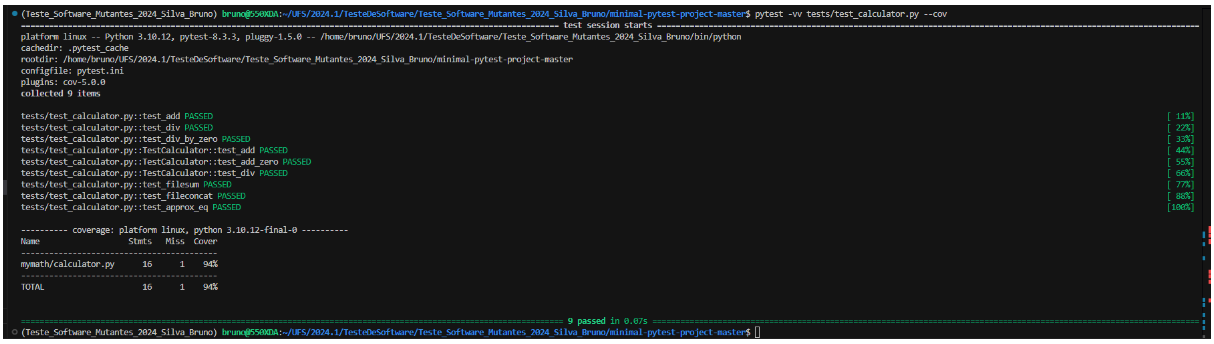
Após instalação das dependências, já é possível executar os testes:

cd minimal-pytest-project-master/
pytest -v tests/test_calculator.py



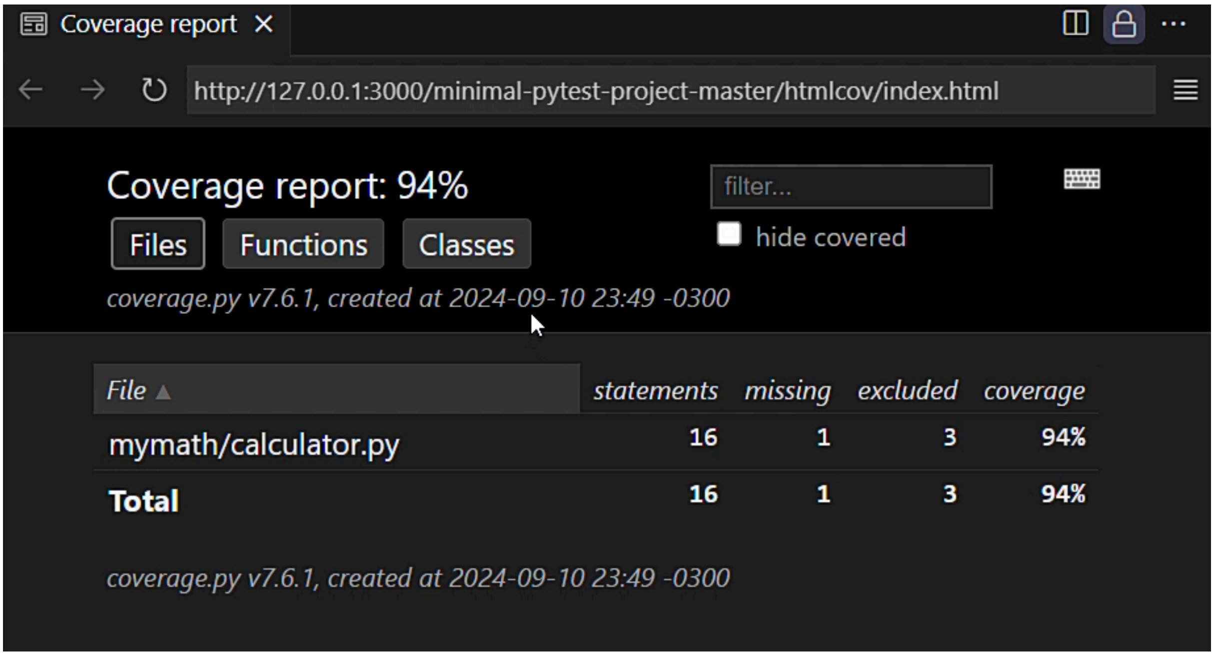
5. Verificar cobertura de testes

pytest -vv tests/test_calculator.py --cov



6. Gerar relatório de cobertura de testes

pytest -vv tests/test_calculator.py --cov --cov-report html



7. Executar testes usando mutmut

mutmut run --paths-to-mutate=mymath/calculator.py

```
- 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
⠏ 21/21  🎉 6  ⏰ 0  🤔 0  🙁 15  🔇 0
```

8. Verificar resultado de mutantes sobreviventes

mutmut results

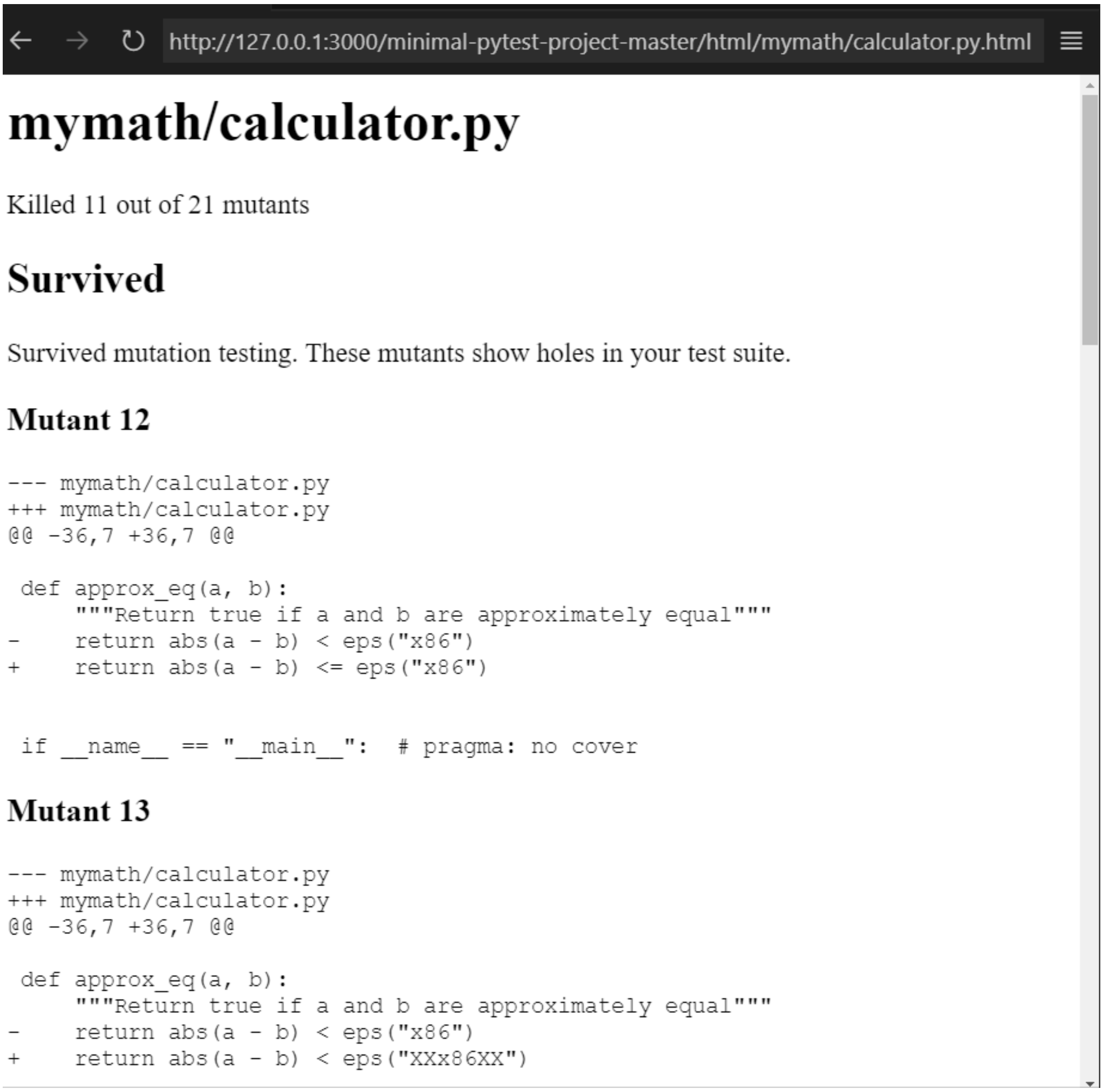
```
● mutmut resultse_Mutantes_2024_Silva_Bruno) bruno@550
  To apply a mutant on disk:
      mutmut apply <id>

  To show a mutant:
      mutmut show <id>


  Survived 🙁 (15)

  ---- mymath/calculator.py (15) ----

  2, 7-10, 12-21
```

9. **Verificar mutação sobrevivente por id**

mutmut show <id>

10. **Gerar relatório de mutações sobreviventes**

mutmut html



http://127.0.0.1:3000/minimal-pytest-project-master/html/mymath/calculator.py.html

mymath/calculator.py

Killed 11 out of 21 mutants

Survived

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

Mutant 12

```
--- mymath/calculator.py
+++ mymath/calculator.py
@@ -36,7 +36,7 @@

 def approx_eq(a, b):
     """Return true if a and b are approximately equal"""
-    return abs(a - b) < eps("x86")
+    return abs(a - b) <= eps("x86")


 if __name__ == "__main__":  # pragma: no cover
```

Mutant 13

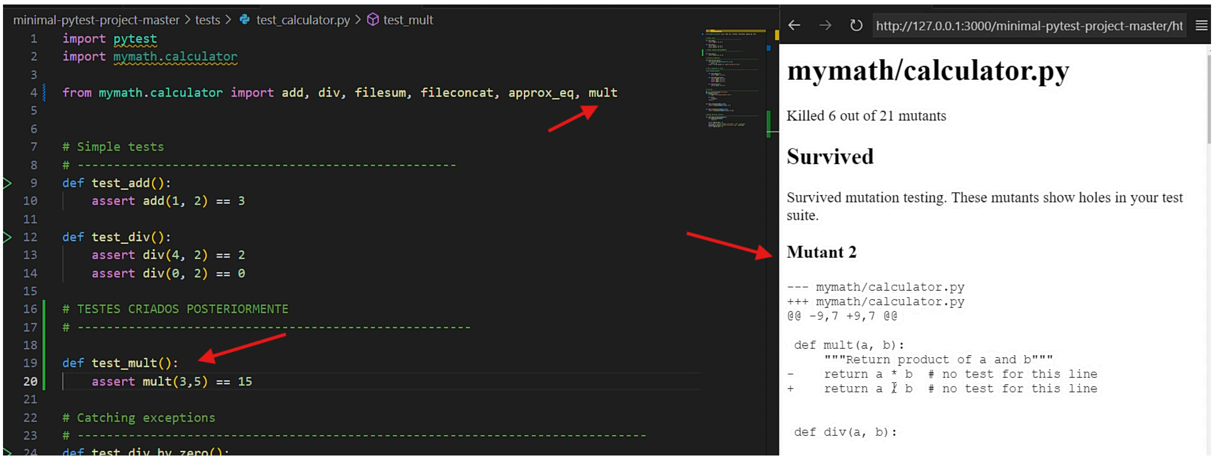
```
--- mymath/calculator.py
+++ mymath/calculator.py
@@ -36,7 +36,7 @@

 def approx_eq(a, b):
     """Return true if a and b are approximately equal"""
-    return abs(a - b) < eps("x86")
+    return abs(a - b) < eps("XXx86XX")
```

Melhorias realizadas

Na execução original do projeto disponibilizado no Github, todos os testes usando o *pytest* foram executados sem nenhum erro. A cobertura dos testes era de 94%. Contudo, executando o teste usando mutantes através do *mutmut*, foram gerados 21 mutantes mas somente 6 foram eliminados.

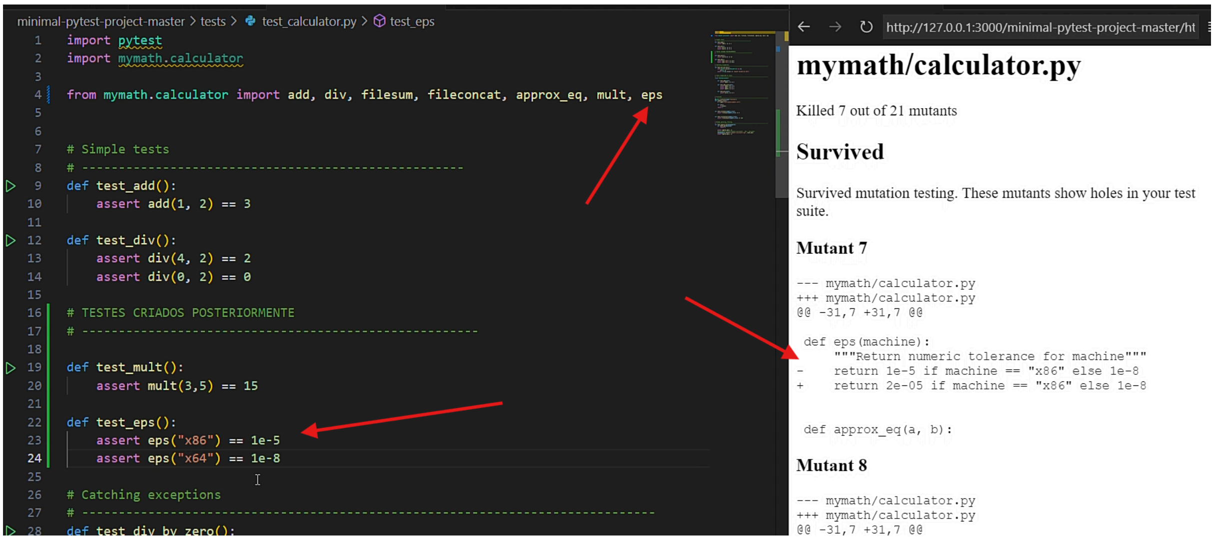
Para melhorar, foi identificado que dois métodos não eram propriamente testados. Assim, observando os métodos, foram criados casos de testes para ambos, buscando condições que não permitissem que os mutantes sobrevivessem.

O método chamado *mult*, realiza multiplicação de dois números e ao observar o relatório do *mutmut*, a operação de multiplicação foi substituída pela de divisão, gerando um valor não condizente com o esperado. A correção foi aplicada importando o método *mult* e criando um caso de teste no qual, só é verdadeiro caso a operação do método seja de multiplicação.

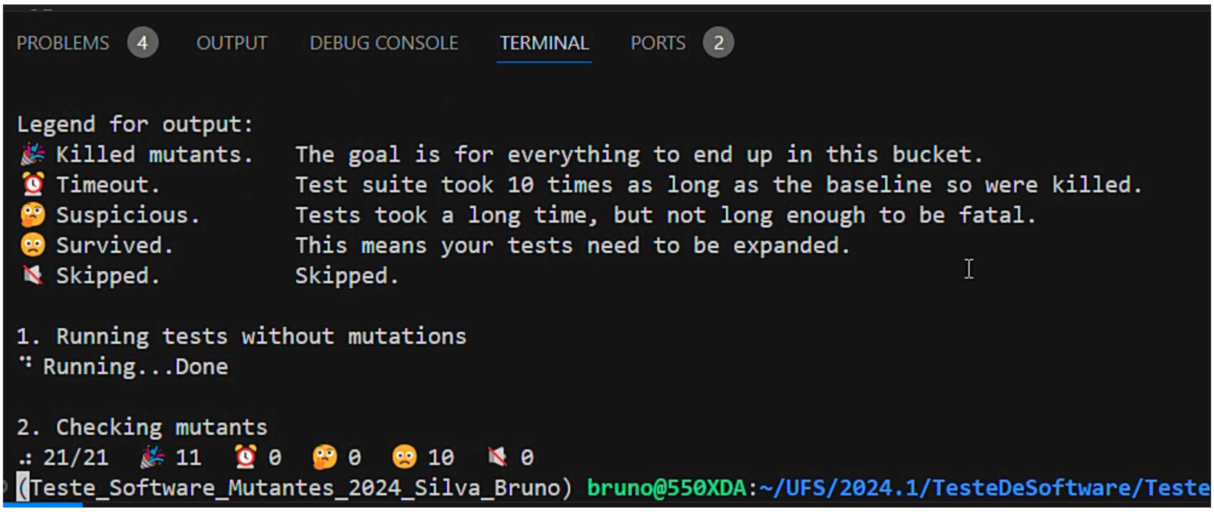


Logo após a adição, foram executados os testes novamente e a cobertura de testes atingiu 100%, já que o método não tinha casos de teste criados. Ao executar o *mutmut*, os mutantes sobreviventes caíram de 15 para 14, confirmando que o teste criado está cobrindo uma lacuna que existia.

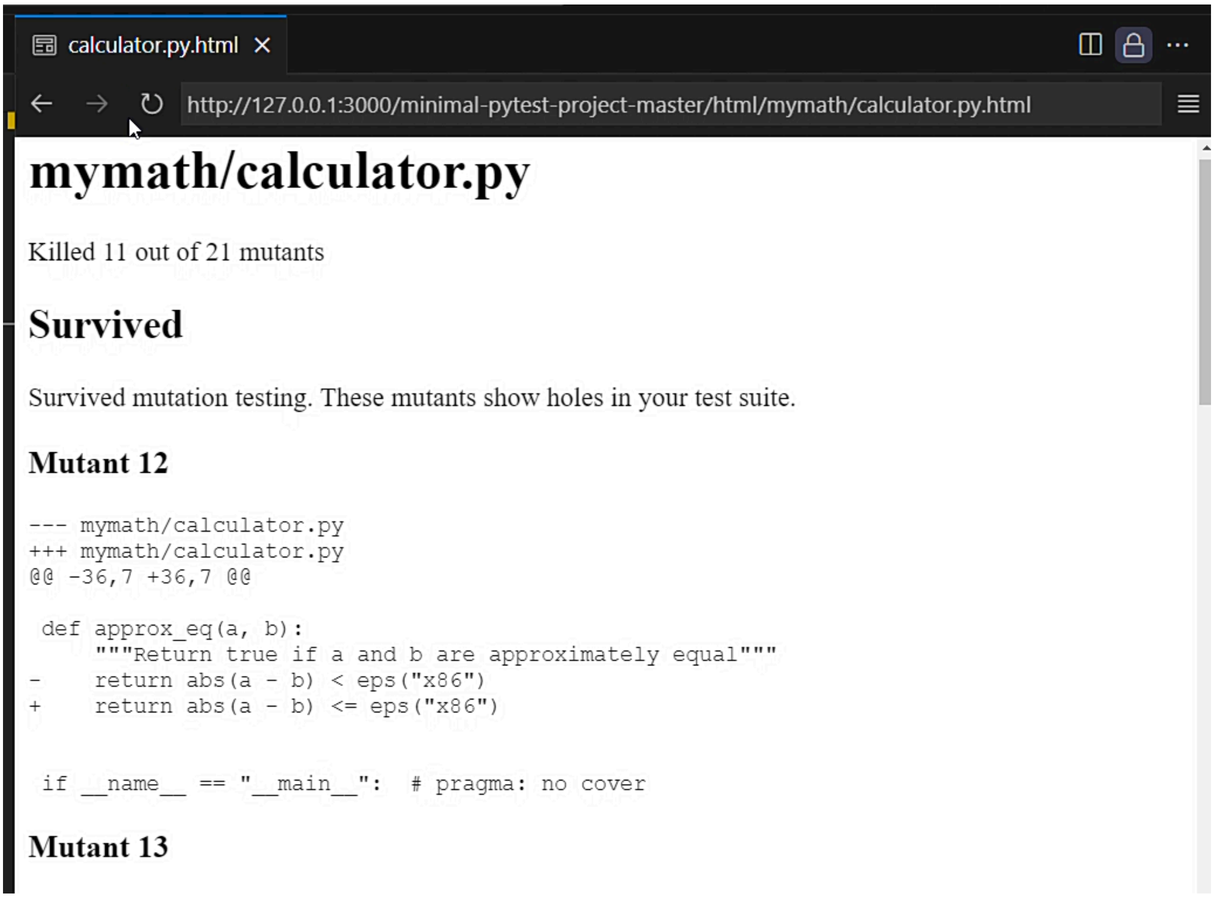
Semelhante a correção anterior, foram criados testes para o método *eps*. Porém, esse método tem uma estrutura que permite que haja mais situações no qual os mutantes sobrevivam ao teste. Analisando o *eps*, foi identificado que ao ser fornecido um valor “x86”, o método retorna um exponencial com valor 1e-5, caso outro valor seja fornecido, retorna 1e-8. Assim, foi criado um método de teste com dois casos: o primeiro fornecendo o valor esperado esperando como resposta o 1e-5, e outro caso fornecendo outro valor e esperando assim o 1e-8.



Executando novamente os testes, percebemos que a cobertura de testes se manteve em 100%, porém os testes de mutações passou de 14 para 10 mutantes sobreviventes.



Dessa forma, as mutações sobreviventes de 7 a 10, foram eliminadas. Gerando o relatório do *mutmut* novamente, podemos confirmar que esses não apresentaram mais erro.



Assim, foi possível de forma simples, melhorar o caso base adicionando testes para ter uma cobertura melhor e evitando aberturas para erros.