



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

Escolha de modelo

Para a atividade proposta, foi necessário escolher um modelo no HuggingFace que utilize transformers. O modelo escolhido foi Legal-BERTimbau-sts-large, que foi disponibilizado pelo usuário rufimelo. O modelo está disponível no link <https://huggingface.co/rufimelo/Legal-BERTimbau-sts-large>.

The screenshot shows the Hugging Face interface for the model **rufimelo/Legal-BERTimbau-sts-large**. The page includes a search bar at the top, navigation links (Models, Datasets, Spaces, Posts, Docs, Solutions, Pricing), and a header with the model name and a 'like' count. Below the header, there are tabs for 'Sentence Similarity', 'sentence-transformers', 'PyTorch', 'Transformers', 'assin', 'assin2', 'rufimelo/PortugueseLegalSentences-v0', 'Portuguese', 'bert', 'feature-extraction', and 'Eval Results'. The 'Model card' tab is selected, showing a description of the model: 'This is a `sentence-transformers` model: It maps sentences & paragraphs to a 1024 dimensional dense vector space and can be used for tasks like clustering or semantic search. `rufimelo/Legal-BERTimbau-sts-large` is based on `Legal-BERTimbau-large` which derives from `BERTimbau-large`. It is adapted to the Portuguese legal domain and trained for STS on portuguese datasets.' The 'Usage (Sentence-Transformers)' section provides a code snippet: `pip install -U sentence-transformers`. On the right, there is a section for 'Inference API' with a note that the model does not have enough activity to be deployed. Below this, there is a section for 'Datasets used to train rufimelo/Legal-BERTimbau-sts-large' listing `nllc-nlp/assin2` and `nllc-nlp/assin`.

Para ser utilizado como dataset da aplicação, foi escolhido o repositório do tesouro chamado de “Investimentos Públicos - ano 2023”. A base de dados fornece um arquivo no formato csv, o qual será consumido. O dataset utilizado está disponível no site do tesouro, podendo ser verificado através do link <https://www.tesourotransparente.gov.br/ckan/dataset/investimento-publico/resource/63f68a7b-d9e4-453a-b7ed-db641a21cbcc>.

[www.tesourotransparente.gov.br/ckan/dataset/investimento-publico/resource/63f68a7b-d9e4-453a-b7ed-db641a21cbcc](#)

BRASIL | Simplifique! | Comunica BR | Participe | Acesso à informação | Legislação | Canais

V para a busca | Ir para o rodapé

TESOURO NACIONAL TRANSPARENTE

investimentos Públicos > Investimentos Públicos - ano ...

[DOWNLOAD DO ARQUIVO](#)

Investimentos Públicos - ano 2023

URL: <https://www.tesourotransparente.gov.br/ckan/dataset/e048826b-b6b0-4d92-9204-fd218bf25b3/resource/63f68a7b-d9e4-453a-b7ed-db641a21cbcc/download/Investime...>

Investimentos Públicos – Corresponde ao total das despesas de investimentos e de inversões financeiras, exceto despesas financeiras, que compõem o Orçamento Fiscal e da Seguridade Social. Os dados são obtidos a partir de consultas junto ao Sistema Integrado de Administração Financeira do Governo Federal – SIAFI. A apuração é realizada pelos grupos de natureza da despesa iguais a 4 e 5, exceto despesas financeiras.

RECURSOS	
Metadados Investimentos	
Investimentos Públicos - ...	
Investimentos Públicos - ...	
Investimentos Públicos - ...	
Investimentos Públicos - ...	
Investimentos Públicos - ...	
Investimentos Públicos - ...	
Investimentos Públicos - ...	
Investimentos Públicos - ...	
Investimentos Públicos - ...	
Investimentos Públicos - ...	

Informações Adicionais

Campo	Valor
Ultima Atualização	29/julho/2024
Criado	29/julho/2024
Formato	CSV
Licença	Open Data Commons Open Database License (ODbL)

[Mostrar mais](#)

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
```

```
(Teste_Software_Mutantes_2024_Silva_Bruno) bruno@50002A:~/UFS/2024.1/TesteDeSoftware/Teste_Software_Mutantes_2024_Silva_Bruno/minimal-pytest-project-master$ pytest -v tests/test_calculator.py
test session starts
platform linux -- Python 3.10.12, pytest-8.3.3, pluggy-1.5.0 -- /home/bruno/UFS/2024.1/TesteDeSoftware/Teste_Software_Mutantes_2024_Silva_Bruno/bin/python
cachedir: .pytest_cache
rootdir: /home/bruno/UFS/2024.1/TesteDeSoftware/Teste_Software_Mutantes_2024_Silva_Bruno/minimal-pytest-project-master
configfile: pytest.ini
plugins: cov-5.0.0
collected 9 items

tests/test_calculator.py::test_add PASSED [ 11%]
tests/test_calculator.py::test_div PASSED [ 22%]
tests/test_calculator.py::test_div_by_zero PASSED [ 33%]
tests/test_calculator.py::TestCalculator::test_add PASSED [ 44%]
tests/test_calculator.py::TestCalculator::test_add_zero PASSED [ 55%]
tests/test_calculator.py::TestCalculator::test_div PASSED [ 66%]
tests/test_calculator.py::test_filesum PASSED [ 77%]
tests/test_calculator.py::test_fileconcat PASSED [ 88%]
tests/test_calculator.py::test_approx_eq PASSED [100%]

----- 9 passed in 0.03s -----
(Teste_Software_Mutantes_2024_Silva_Bruno) bruno@50002A:~/UFS/2024.1/TesteDeSoftware/Teste_Software_Mutantes_2024_Silva_Bruno/minimal-pytest-project-master$
```

5. Verificar cobertura de testes

`pytest -vv tests/test_calculator.py --cov`

```
(Teste_Software_Mutantes_2024_Silva_Bruno) bruno@50002A:~/UFS/2024.1/TesteDeSoftware/Teste_Software_Mutantes_2024_Silva_Bruno/minimal-pytest-project-master$ pytest -vv tests/test_calculator.py --cov
test session starts
platform linux -- Python 3.10.12, pytest-8.3.3, pluggy-1.5.0 -- /home/bruno/UFS/2024.1/TesteDeSoftware/Teste_Software_Mutantes_2024_Silva_Bruno/bin/python
cachedir: .pytest_cache
rootdir: /home/bruno/UFS/2024.1/TesteDeSoftware/Teste_Software_Mutantes_2024_Silva_Bruno/minimal-pytest-project-master
configfile: pytest.ini
plugins: cov-5.0.0
collected 9 items

tests/test_calculator.py::test_add PASSED [ 11%]
tests/test_calculator.py::test_div PASSED [ 22%]
tests/test_calculator.py::test_div_by_zero PASSED [ 33%]
tests/test_calculator.py::TestCalculator::test_add PASSED [ 44%]
tests/test_calculator.py::TestCalculator::test_add_zero PASSED [ 55%]
tests/test_calculator.py::TestCalculator::test_div PASSED [ 66%]
tests/test_calculator.py::test_filesum PASSED [ 77%]
tests/test_calculator.py::test_fileconcat PASSED [ 88%]
tests/test_calculator.py::test_approx_eq PASSED [100%]

----- coverage: platform linux, python 3.10.12-final-0 -----
Name
-----
mymath/calculator.py 16 1 94%
TOTAL 16 1 94%

----- 9 passed in 0.07s -----
(Teste_Software_Mutantes_2024_Silva_Bruno) bruno@50002A:~/UFS/2024.1/TesteDeSoftware/Teste_Software_Mutantes_2024_Silva_Bruno/minimal-pytest-project-master$
```

6. Gerar relatório de cobertura de testes

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

Coverage report X

http://127.0.0.1:3000/minimal-pytest-project-master/htmlcov/index.html

Coverage report: 94%

filter...

☐ hide covered

coverage.py v7.6.1, created at 2024-09-10 23:49 -0300

File	statements	missing	excluded	coverage
mymath/calculator.py	16	1	3	94%
Total	16	1	3	94%

coverage.py v7.6.1, created at 2024-09-10 23:49 -0300

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) ----

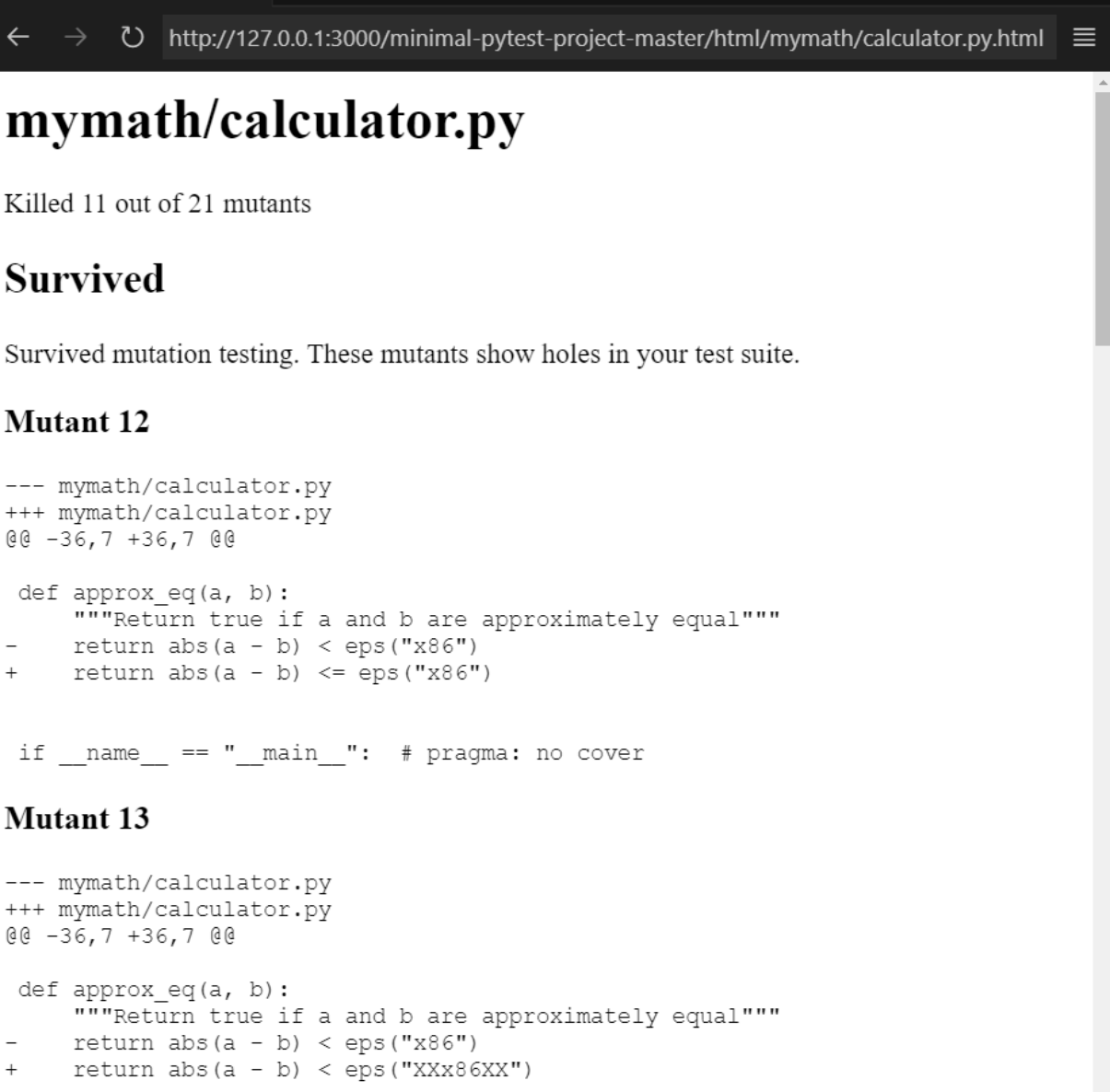
2, 7-10, 12-21
● mutmut show 2se_Mutantes_2024_Silva_Bruno) bruno@550%
```

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 @@

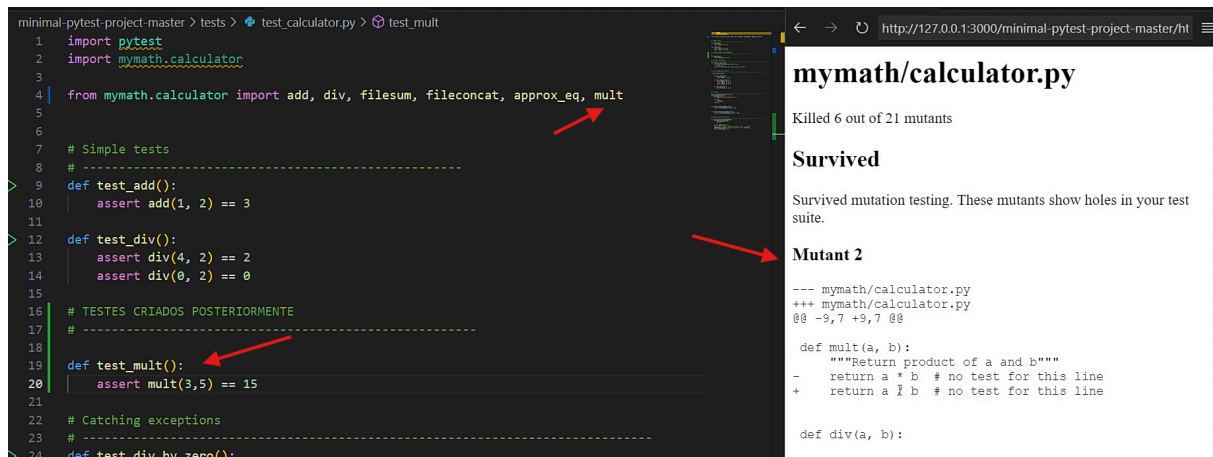
def approx_eq(a, b):
    """Return true if a and b are approximately equal"""
-   return abs(a - b) < eps("x86")
+   return abs(a - b) < eps("XXx86XX")
```

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.



The image shows a code editor on the left and a web browser on the right. The code editor displays the `calculator.py` file with the following content:

```
1 import pytest
2 import mymath.calculator
3
4 from mymath.calculator import add, div, filesum, fileconcat, approx_eq, mult
5
6
7 # Simple tests
8 # -----
9 def test_add():
10     assert add(1, 2) == 3
11
12 def test_div():
13     assert div(4, 2) == 2
14     assert div(0, 2) == 0
15
16 # TESTES CRIADOS POSTERIORMENTE
17 # -----
18
19 def test_mult():
20     assert mult(3,5) == 15
21
22 # Catching exceptions
23 # -----
24 def test_div_by_zero():
```

Red arrows point from the `mult` import on line 4 and the `test_mult` function on line 19 to the browser window. The browser window shows the URL `http://127.0.0.1:3000/minimal-pytest-project-master/ht` and the title `mymath/calculator.py`. The content of the browser window is as follows:

```
Killed 6 out of 21 mutants

Survived

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

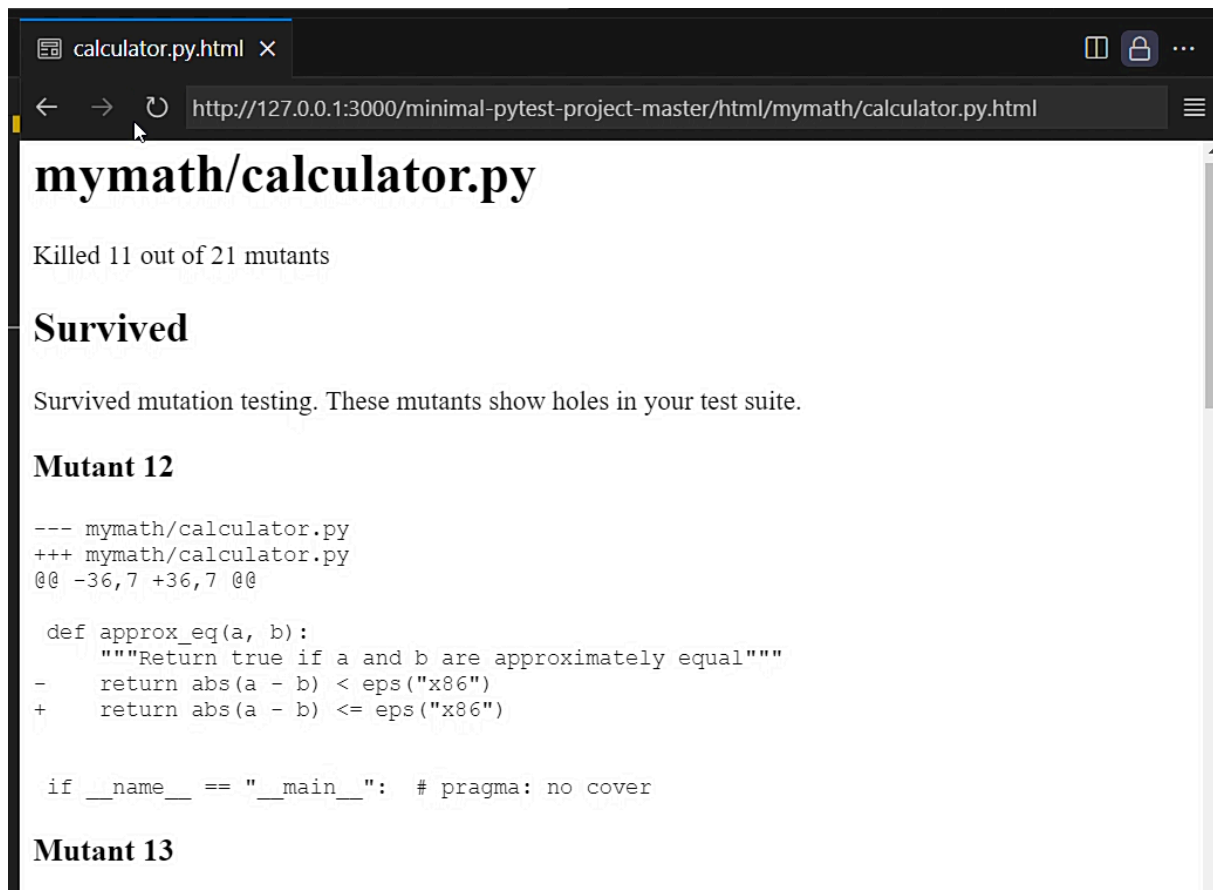
Mutant 2

--- mymath/calculator.py
+++ mymath/calculator.py
@@ -9,7 +9,7 @@
def mult(a, b):
    """Return product of a and b"""
-    return a * b # no test for this line
+    return a / b # no test for this line

def div(a, b):
```

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.



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