



**UNIVERSIDADE FEDERAL DE SERGIPE**  
**CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA**  
**DEPARTAMENTO DE COMPUTAÇÃO**

**RAFAEL TAKEGUMA GOTO**

**ATIVIDADE 2 – TESTES DE MUTAÇÃO**



Departamento de Computação/UFS

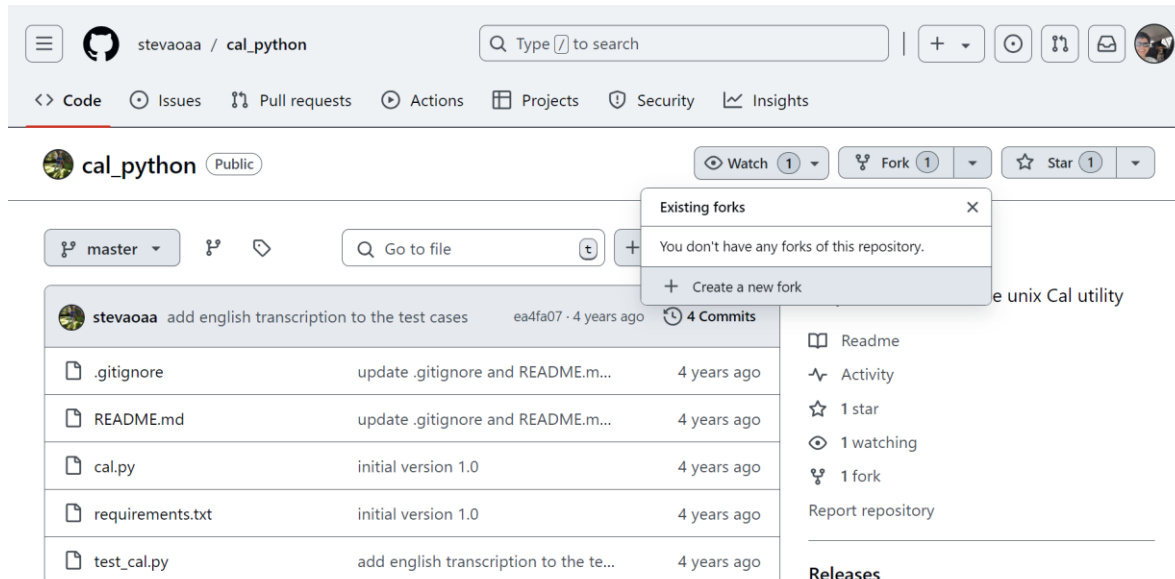
**SÃO CRISTÓVÃO - SERGIPE**

**2024**

## 1. Roteiro do vídeo

A primeira etapa desta atividade consistiu em executar como exemplo, o roteiro de testes de mutação apresentado no vídeo disponibilizado, que pode ser acessado por meio do seguinte link: <https://www.youtube.com/watch?v=FbMpoVOorFI>. O primeiro passo foi acessar o repositório cal\_python, do qual é um projeto de calendário inspirado no cal do unix, disponível por meio do seguinte link: [https://github.com/stevaoaa/cal\\_python](https://github.com/stevaoaa/cal_python). Em seguida, a fim de criar uma cópia deste repositório, realizou-se um fork. A figura 1 apresenta a criação do fork do projeto cal\_python, realizada por meio da opção “Fork”, seguida de “+ Create a new fork”.

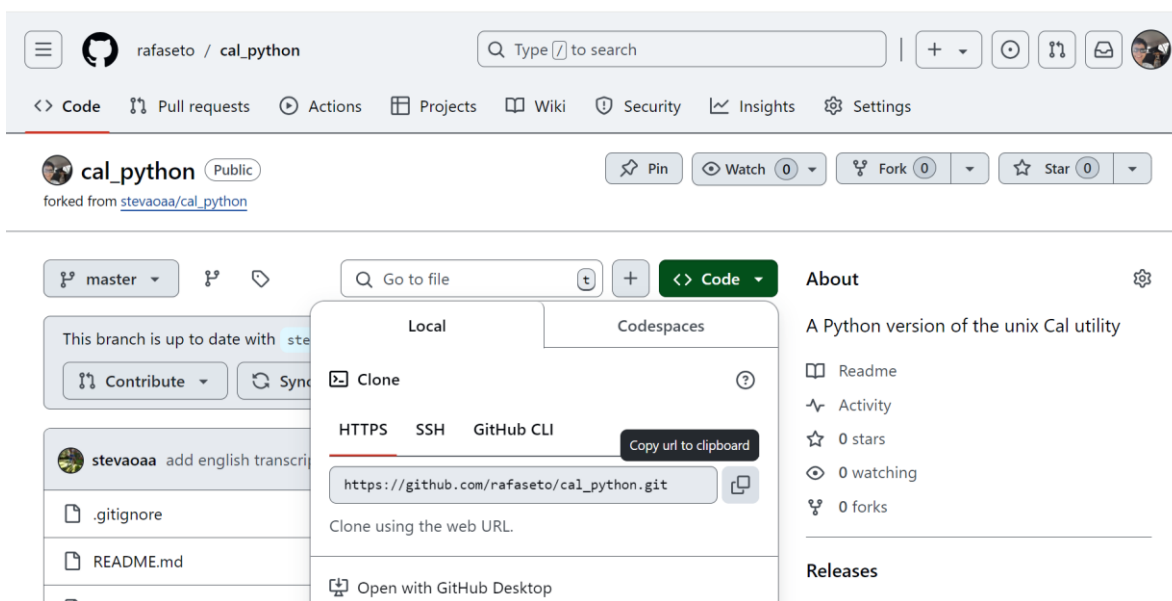
Figura 1 – Realização do fork para o repositório cal\_python



Fonte: Autor

Após a realização do fork, foi utilizada a URL do fork para clonar o repositório como uma cópia local do projeto. A figura 2 mostra a obtenção da URL, por meio da opção “<> Code”.

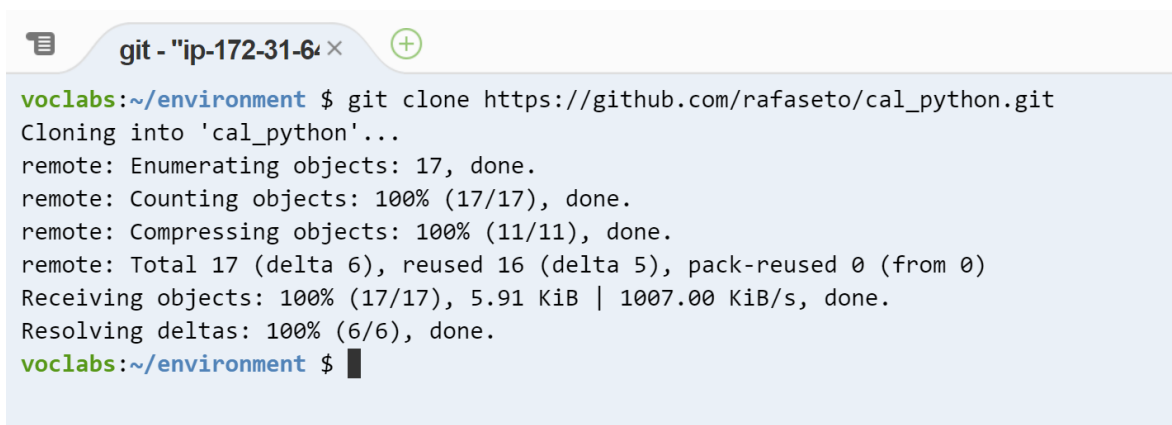
Figura 2 – Obtenção da URL do projeto copiado



Fonte: Autor

De posse da URL, foi utilizado o comando “git clone <URL>” no terminal para fazer a clonagem em si do projeto em uma cópia local. Para esta etapa da atividade utilizou-se o AWS Cloud9 como IDE. A figura 3 mostra a clonagem do repositório no AWS Cloud9.

Figura 3 – Clonagem do projeto no AWS Cloud9



Fonte: Autor

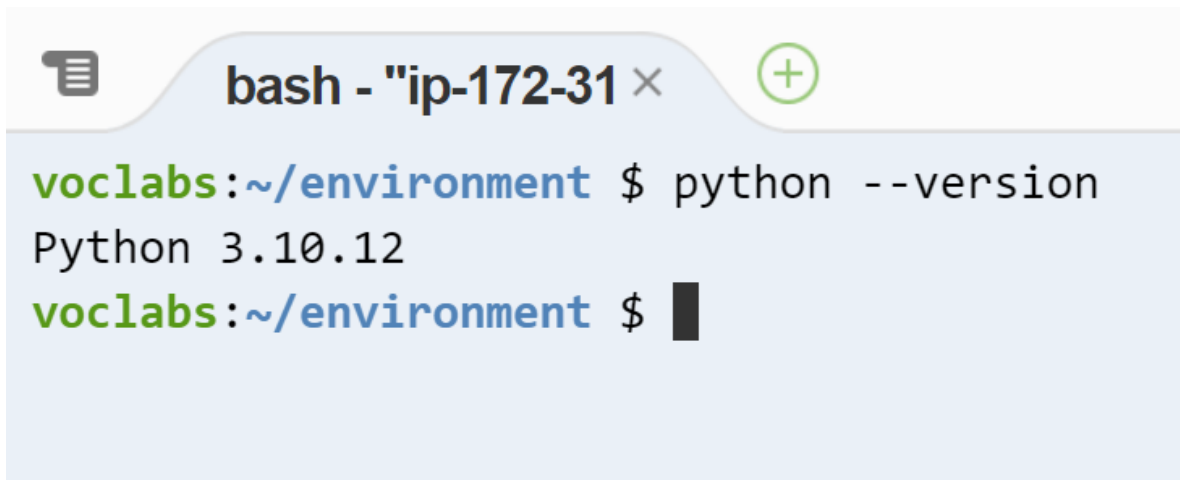
## 1.1 Preparação do ambiente

Para os fins desta atividade, foram utilizados os seguintes recursos:

- Python 3.10.12;
- python3-venv: Módulo integrado do Python que permite criar ambientes virtuais;
- pytest: Framework de testes para Python;
- pytest-cov: Plugin para o pytest que fornece relatórios de cobertura de código;
- mutmut: Ferramenta de teste de mutação para Python.

Python foi a linguagem de programação utilizada nesta atividade. Foi utilizado o comando “python --version” para checar a versão do Python no ambiente do AWS Cloud9. A figura 4 mostra o resultado da checagem da versão do Python.

Figura 4 – Versão do Python configurada no ambiente do AWS Cloud9

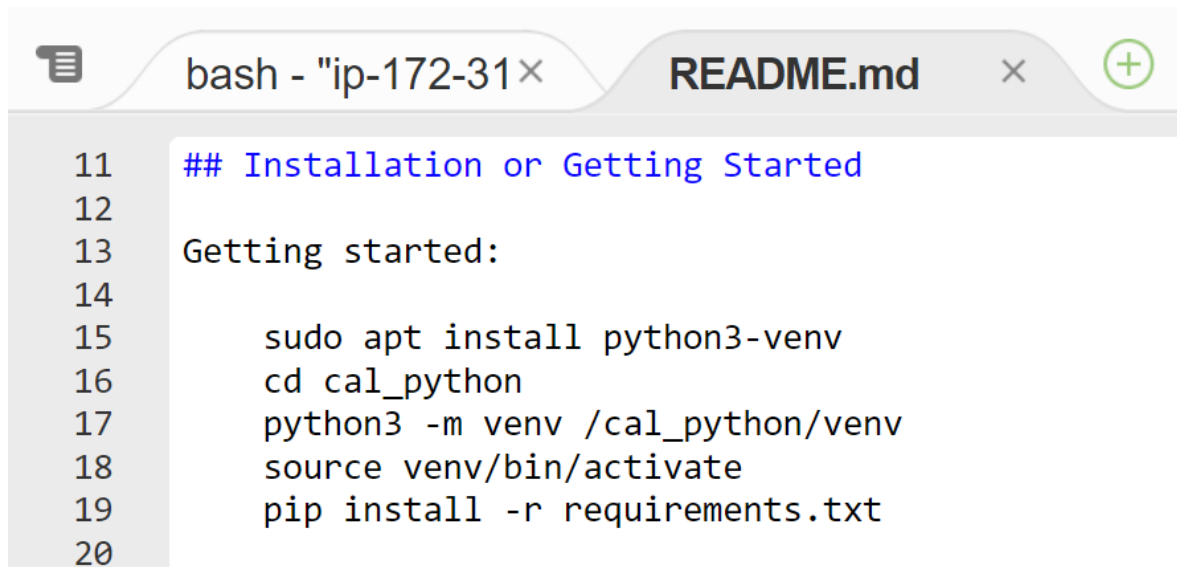
A screenshot of the AWS Cloud9 IDE terminal. The title bar at the top shows a document icon, the text "bash - 'ip-172-31'", a close button (X), and a plus button (+). The terminal content shows the prompt "voclabs:~/environment \$" followed by the command "python --version". The output is "Python 3.10.12". Below the output, the prompt "voclabs:~/environment \$" is shown again with a black cursor block.

```
bash - "ip-172-31" × (+)
voclabs:~/environment $ python --version
Python 3.10.12
voclabs:~/environment $ █
```

Fonte: Autor

Os demais recursos foram configurados seguindo as instruções definidas no arquivo README.md do projeto, que podem ser visualizadas na figura 5. A única diferença observada foi o caminho da linha 17, do qual foi apenas “venv”, visto que o diretório de trabalho já tinha sido definido como “/cal\_python” por meio do comando “cd cal\_python”.

Figura 5 – Instruções de instalação e configuração



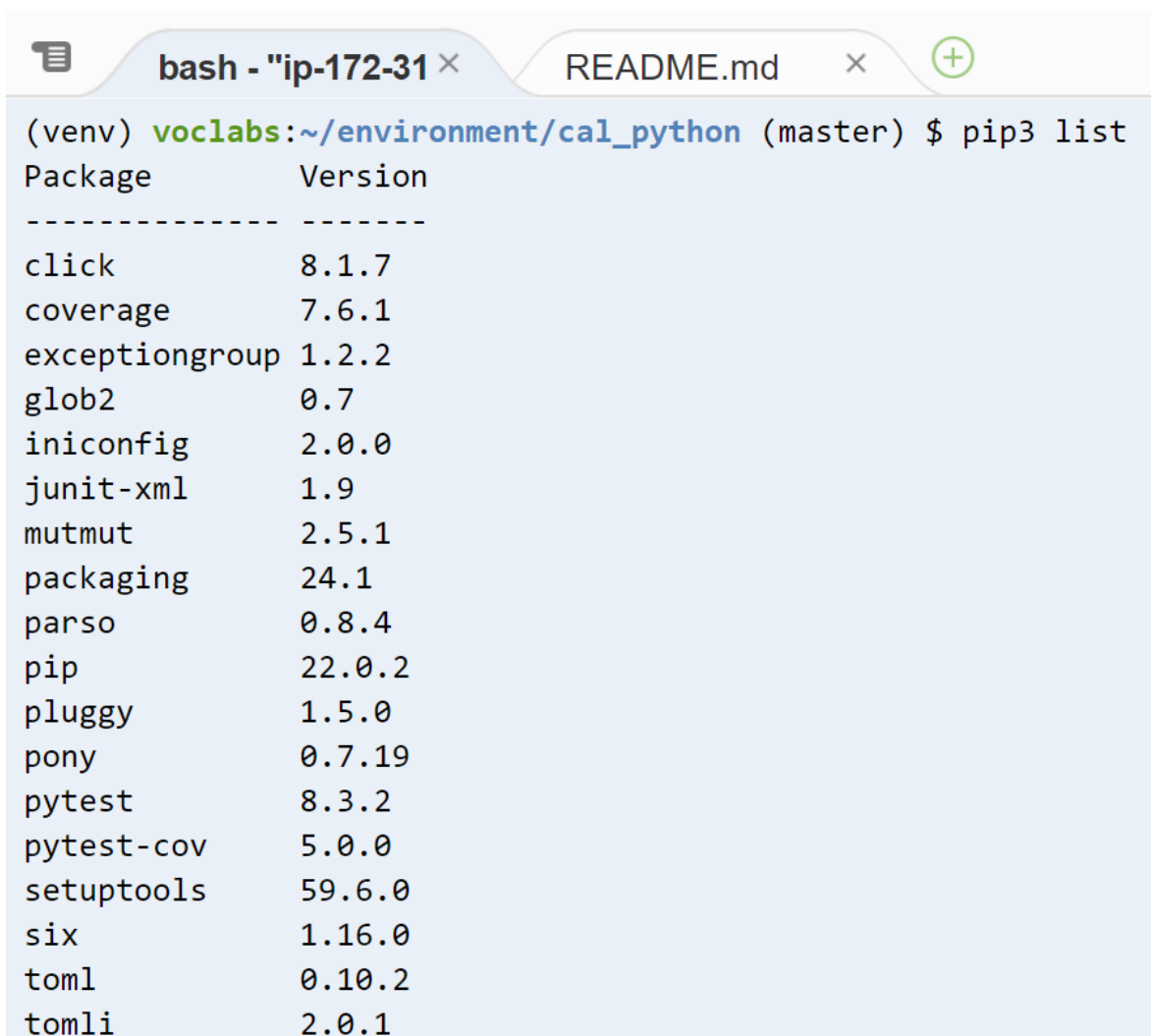
The image shows a terminal window with two tabs: 'bash - "ip-172-31...' and 'README.md'. The 'README.md' tab is active, displaying the following text:

```
11  ## Installation or Getting Started
12
13  Getting started:
14
15      sudo apt install python3-venv
16      cd cal_python
17      python3 -m venv /cal_python/venv
18      source venv/bin/activate
19      pip install -r requirements.txt
20
```

Fonte: Stevão Andrade

Com a utilização do comando “pip3 list” pode ser observado a lista de bibliotecas instaladas no ambiente virtual. Pode-se notar, por meio da figura 6, que o pytest, pytest-conv e mutmut estão devidamente instalados.

Figura 6 – Lista de bibliotecas instaladas no ambiente virtual



```
(venv) voclabs:~/environment/cal_python (master) $ pip3 list
Package            Version
-----
click              8.1.7
coverage           7.6.1
exceptiongroup     1.2.2
glob2              0.7
iniconfig          2.0.0
junit-xml          1.9
mutmut             2.5.1
packaging          24.1
parso              0.8.4
pip                22.0.2
pluggy            1.5.0
pony               0.7.19
pytest             8.3.2
pytest-cov         5.0.0
setuptools         59.6.0
six                1.16.0
toml               0.10.2
tomli              2.0.1
```

Fonte: Autor

## 1.2 Testes

Depois de configurado o ambiente, juntamente com todas as ferramentas necessárias, iniciou-se a etapa de testes. O primeiro comando utilizado foi o “`pytest -vv test_cal.py`”, do qual executa os casos de teste definidos no arquivo `test_cal.py` usando o `pytest`. A figura 7 apresenta os resultados dos testes: dos 21 testes encontrados, 20 passaram e 1 foi ignorado.

Figura 7 – Resultados da execução dos casos de teste de test\_cal.py

```
python3 - "ip-172" x README.md x
vocilabs:~/environment/cal_python (master) $ pytest -vv test_cal.py
===== test session starts =====
platform linux -- Python 3.10.12, pytest-8.3.2, pluggy-1.5.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/ubuntu/environment/cal_python
plugins: cov-5.0.0
collected 21 items

test_cal.py::test_cal PASSED [ 4%]
test_cal.py::test_is_leap[4-True] PASSED [ 9%]
test_cal.py::test_is_leap[1501-False] PASSED [ 14%]
test_cal.py::test_is_leap[1900-False] PASSED [ 19%]
test_cal.py::test_is_leap[1903-False] PASSED [ 23%]
test_cal.py::test_is_leap[1904-True] PASSED [ 28%]
test_cal.py::test_is_leap[2000-True] PASSED [ 33%]
test_cal.py::test_mutation_is_leap SKIPPED (simply ignoring for now) [ 38%]
test_cal.py::test_first_of_month[input0-6] PASSED [ 42%]
test_cal.py::test_first_of_month[input1-2] PASSED [ 47%]
test_cal.py::test_first_of_month[input2-4] PASSED [ 52%]
test_cal.py::test_first_of_month[input3-2] PASSED [ 57%]
test_cal.py::test_first_of_month[input4-3] PASSED [ 61%]
test_cal.py::test_jan1[1801-4] PASSED [ 66%]
test_cal.py::test_jan1[1780-6] PASSED [ 71%]
test_cal.py::test_jan1[1500-3] PASSED [ 76%]
test_cal.py::test_number_of_days[input0-31] PASSED [ 80%]
test_cal.py::test_number_of_days[input1-29] PASSED [ 85%]
test_cal.py::test_number_of_days[input2-31] PASSED [ 90%]
test_cal.py::test_number_of_days[input3-19] PASSED [ 95%]
test_cal.py::test_number_of_days[input4-31] PASSED [100%]

===== 20 passed, 1 skipped in 0.08s =====
```

Fonte: Autor

Para verificar a adição de novos casos de teste, foi criado um novo cenário de teste para o método “test\_is\_leap”, o parâmetro (1700, True). A figura 8 mostra o método de teste “test\_is\_leap” e sua lista de parâmetros após esta adição.

Figura 8 – Método de teste test\_is\_leap e sua lista de parâmetros

```
24 @pytest.mark.parametrize('input, expected_result', [
25     (4, True),      #Year less than 1752 and multiple of 4
26     (1501, False),  #Year less than 1752 and non-multiple of 4
27     (1900, False),  #Year greater than 1752 and multiple of 100
28     (1903, False),  #Year greater than 1752 and not a multiple of 4
29     (1904, True),   #Year greater than 1752 and multiple of 4
30     (2000, True),   #Year greater than 1752 and multiple of 400
31     (1700, True)
32 ])
33 def test_is_leap(input, expected_result):
34
35     actual = cal.is_leap(input)
36     assert actual == expected_result
```

Fonte: Autor

A figura 9 expõe os resultados da execução dos testes após a adição do novo parâmetro para o método “test\_is\_leap”. Pode-se observar que desta vez 22 casos de teste foram encontrados, 21 passaram com sucesso, incluindo o “test\_is\_leap[1700, True]”, e 1 foi

ignorado.

Figura 9 – Resultados da execução dos casos de teste de test\_cal.py após as mudanças

```

bash - "ip-172-31" x  README.md  x  test_cal.py  x  +
voclabs:~/environment/cal_python (master) $ pytest -vv test_cal.py
===== test session starts =====
platform linux -- Python 3.10.12, pytest-8.3.2, pluggy-1.5.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/ubuntu/environment/cal_python
plugins: cov-5.0.0
collected 22 items

test_cal.py::test_cal PASSED [ 4%]
test_cal.py::test_is_leap[4-True] PASSED [ 9%]
test_cal.py::test_is_leap[1501-False] PASSED [ 13%]
test_cal.py::test_is_leap[1900-False] PASSED [ 18%]
test_cal.py::test_is_leap[1903-False] PASSED [ 22%]
test_cal.py::test_is_leap[1904-True] PASSED [ 27%]
test_cal.py::test_is_leap[2000-True] PASSED [ 31%]
test_cal.py::test_is_leap[1700-True] PASSED [ 36%]
test_cal.py::test_mutation_is_leap SKIPPED (simply ignoring for now) [ 40%]
test_cal.py::test_first_of_month[input0-6] PASSED [ 45%]
test_cal.py::test_first_of_month[input1-2] PASSED [ 50%]
test_cal.py::test_first_of_month[input2-4] PASSED [ 54%]
test_cal.py::test_first_of_month[input3-2] PASSED [ 59%]
test_cal.py::test_first_of_month[input4-3] PASSED [ 63%]
test_cal.py::test_jan1[1801-4] PASSED [ 68%]
test_cal.py::test_jan1[1780-6] PASSED [ 72%]
test_cal.py::test_jan1[1500-3] PASSED [ 77%]
test_cal.py::test_number_of_days[input0-31] PASSED [ 81%]
test_cal.py::test_number_of_days[input1-29] PASSED [ 86%]
test_cal.py::test_number_of_days[input2-31] PASSED [ 90%]
test_cal.py::test_number_of_days[input3-19] PASSED [ 95%]
test_cal.py::test_number_of_days[input4-31] PASSED [100%]

===== 21 passed, 1 skipped in 0.06s =====

```

Fonte: Autor

O próximo comando de teste utilizado foi o “pytest -vv test\_cal.py --cov=cal”, que além de executar os casos de teste, gera um relatório de cobertura de código. É possível notar, por meio da figura 10, que o código em “cal.py” possui uma porcentagem de cobertura de código de 55%, isto é, 55% das linhas do arquivo “cal.py” foram cobertas pelos testes.

Figura 10 – Relatório de cobertura de código para o arquivo cal.py

----- coverage: platform linux, python 3.10.12-final-0 -----			
Name	Stmts	Miss	Cover
-----			
cal.py	109	49	55%
-----			
TOTAL	109	49	55%

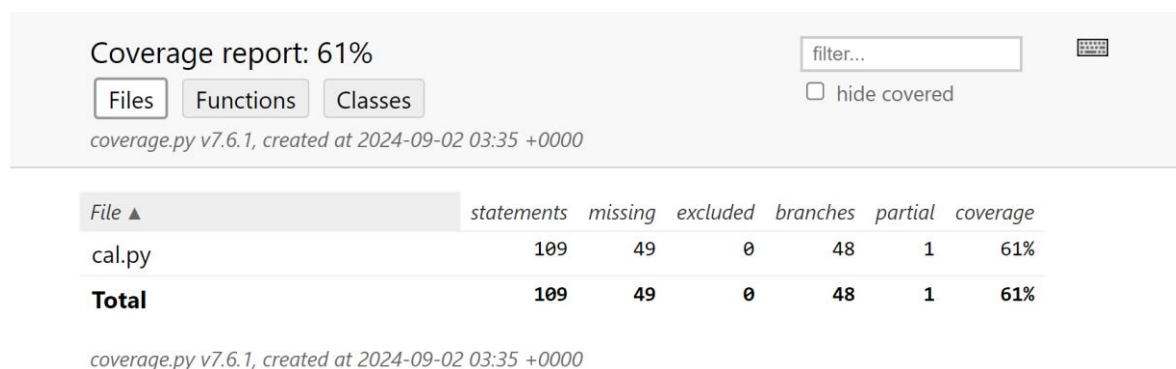
Fonte: Autor

Em seguida executou-se o seguinte comando: “pytest -vv test\_cal.py --cov=cal --cov-branch --cov-report html”, do qual executa os casos de teste, gera o relatório de cobertura mais detalhado e o exporta como um arquivo .html. O arquivo em formato HTML gerado é



o “index.html”, ele pode ser acessado por meio da pasta htmlcov. A figura 11 mostra a tela principal do relatório. Segundo os dados do relatório, o script “cal.py” possui 109 comandos, sendo que 49 deles não estão cobertos pelos testes e 1 está parcialmente coberto. Ademais existem 48 ramificações (*branches*), e a cobertura é de 61%.

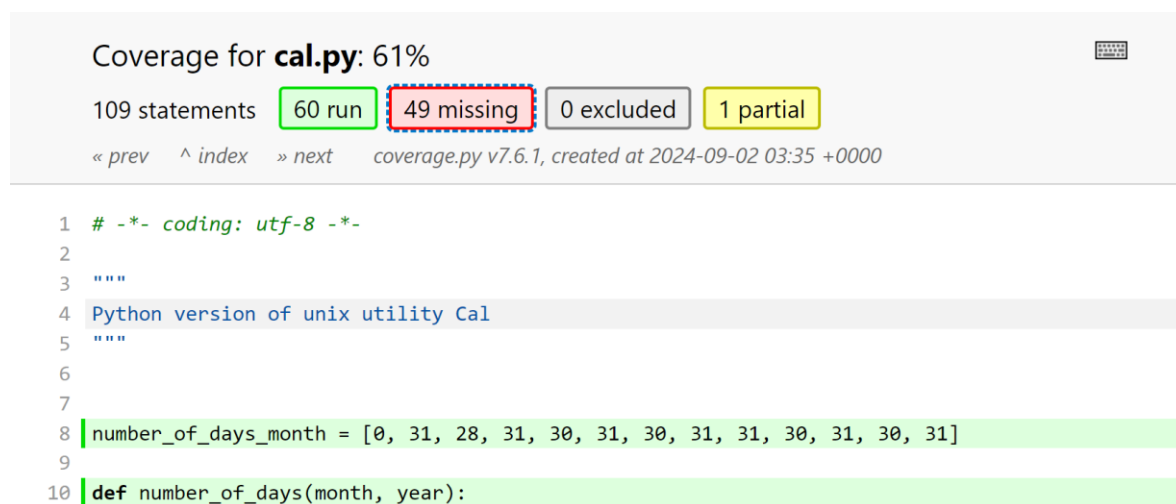
Figura 11 – Relatório de cobertura de código em formato HTML para cal.py



Fonte: Autor

Uma visualização mais detalhada da cobertura foi acessada por de um clique no nome do arquivo “cal.py”. Dessa maneira, pode-se visualizar linha por linha, em verde o que está coberto pelos testes e em vermelho o que não está coberto pelos testes. A figura 12 é um trecho do relatório neste nível de detalhe.

Figura 12 – Relatório de todas as linhas do arquivo “cal.py”



Fonte: Autor

A fim de verificar se o conjunto de casos de teste é suficiente e adequado, foram utilizados os testes de mutação. Por conseguinte, foi usado o comando “mutmut run --paths-to-mutate=cal.py” para executar os testes de mutação. Para que o comando seja executado sem erros, foi necessário mover o arquivo “test\_cal.py” para uma nova subpasta chamada “tests”, de modo que o mutmut pudesse reconhecer onde procurar os casos de teste. A figura 12 mostra os resultados da execução: 123 mutantes “mortos”, 2 “suspeitos” e 108 “sobreviveram”.

Figura 13 – Resultados da execução dos testes de mutação

```

bash - "ip-172-31" x  README.md x  +
(venv) voclabs:~/environment/cal_python (master) $ mutmut run --paths-to-mutate=cal.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.

1. Using cached time for baseline tests, to run baseline again delete the cache file

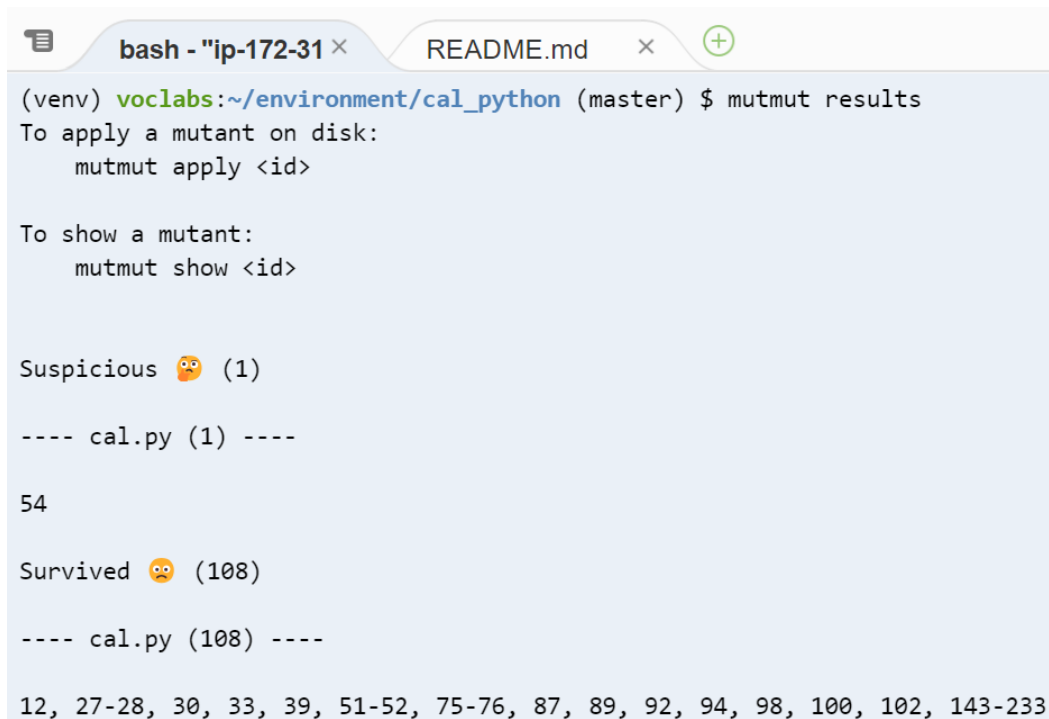
2. Checking mutants
📊: 233/233  🔥 123  🕒 0  🧐 2  😬 108  🚫 0

```

Fonte: Autor

A figura 14 demonstra o resultado obtido com o comando “mutmut results”, do qual permite verificar os mutantes que sobreviveram e o único mutante suspeito encontrado.

Figura 14 – Mutantes suspeitos e que sobreviveram



```

bash - "ip-172-31" x  README.md x  (+)
(venv) voclabs:~/environment/cal_python (master) $ mutmut results
To apply a mutant on disk:
    mutmut apply <id>

To show a mutant:
    mutmut show <id>

Suspicious 🤖 (1)

---- cal.py (1) ----

54

Survived 🤖 (108)

---- cal.py (108) ----

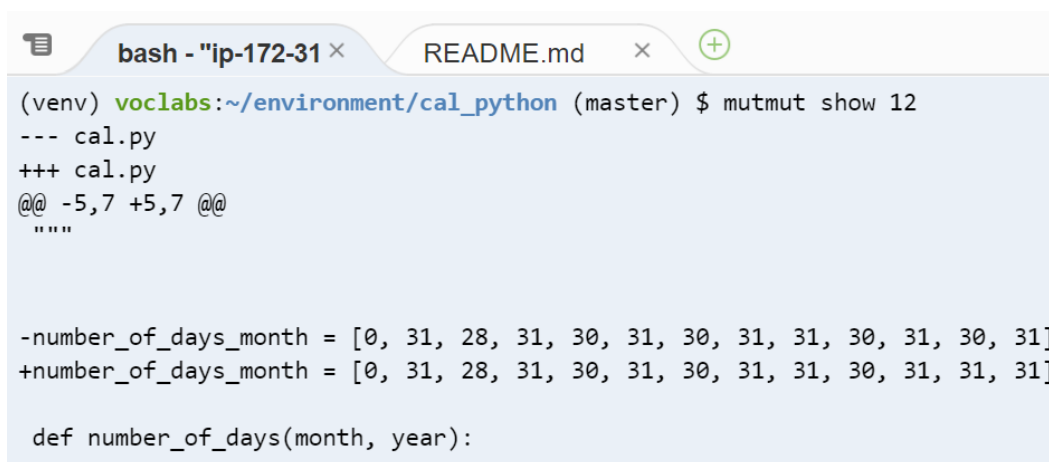
12, 27-28, 30, 33, 39, 51-52, 75-76, 87, 89, 92, 94, 98, 100, 102, 143-233

```

Fonte: Autor

Por meio do comando “mutmut show <mutante>”, pode-se visualizar os detalhes da alteração do mutante passado como argumento. A figura 15 mostra os detalhes do mutante 12.

Figura 15 – Alteração feita no mutante 12



```

bash - "ip-172-31" x  README.md x  (+)
(venv) voclabs:~/environment/cal_python (master) $ mutmut show 12
--- cal.py
+++ cal.py
@@ -5,7 +5,7 @@
"""

-number_of_days_month = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
+number_of_days_month = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 31, 31]

def number_of_days(month, year):

```

Fonte: Autor

A fim de obter detalhes das alterações feitas em cada mutante “suspeito” e “sobrevivente”, executou-se o comando “mutmut html”, do qual gera um arquivo em formato HTML com informações sobre estes mutantes. A figura 16 apresenta um trecho deste arquivo gerado.

Figura 16 – Trecho do arquivo cal.py.html

## cal.py

Killed 124 out of 233 mutants

### Survived

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

#### Mutant 12

```

--- cal.py
+++ cal.py
@@ -5,7 +5,7 @@
"""

-number_of_days_month = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
+number_of_days_month = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 31, 31]

def number_of_days(month, year):

```

Fonte: Autor

Para exemplificar vulnerabilidades encontradas nos casos de teste, observou-se detalhes do mutante 27. O mutante 27 foi a alteração do operador “>” por “>=” na checagem comparativa entre “month” e o número 2. Como todos os casos de teste passaram mesmo com esse mutante inserido no código, significa que os testes não cobrem adequadamente essa parte específica do código. A figura 17 mostra os detalhes do mutante 27 no arquivo cal.py.html.

Figura 17 – Informações acerca do mutante 27

## Mutant 27

```

--- cal.py
+++ cal.py
@@ -41,7 +41,7 @@

    k = 0

-    if (is_leap(year) and month > 2):
+    if (is_leap(year) and month >= 2):
        k+= 1

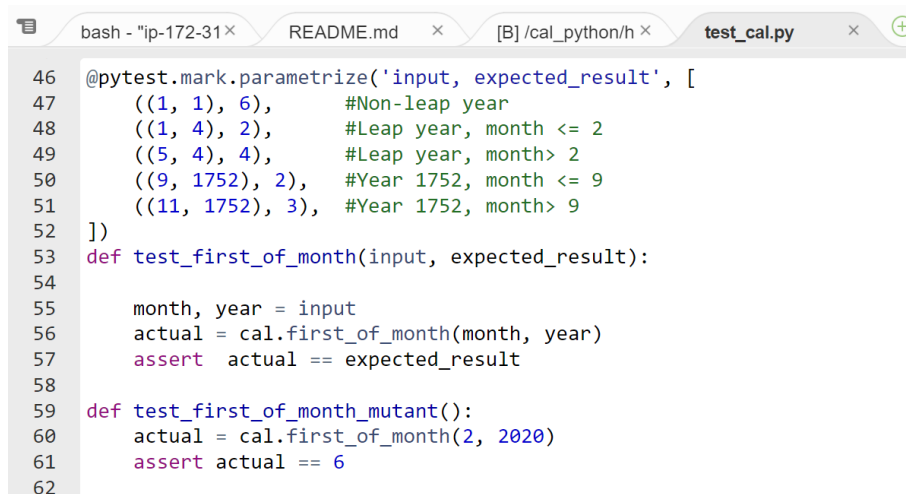
    for i in range(0, month):

```

Fonte: Autor

De acordo com as informações obtidas com esse mutante, foi possível melhorar os casos de teste relacionados ao método `x`. Foi criado um novo caso de teste que tem como parâmetros um ano bissexto e o mês igual a 2, de modo que os casos de teste definidos passam a cobrir o caso do mutante 27. A figura 18 mostra o novo caso de teste (2, 2020) implementado no arquivo “test\_cal.py”.

Figura 18 – Novo caso de teste (2, 2020) para “matar” o mutante 27



```

46 @pytest.mark.parametrize('input, expected_result', [
47     ((1, 1), 6),      #Non-leap year
48     ((1, 4), 2),      #Leap year, month <= 2
49     ((5, 4), 4),      #Leap year, month > 2
50     ((9, 1752), 2),   #Year 1752, month <= 9
51     ((11, 1752), 3),  #Year 1752, month > 9
52 ])
53 def test_first_of_month(input, expected_result):
54
55     month, year = input
56     actual = cal.first_of_month(month, year)
57     assert actual == expected_result
58
59 def test_first_of_month_mutant():
60     actual = cal.first_of_month(2, 2020)
61     assert actual == 6
62

```

Fonte: Autor

Ao executar os testes de mutação novamente, pode-se observar, na figura 19, que após a adição do caso de teste (2, 2020), 127 mutantes foram mortos, 3 a mais do que antes da adição.

Figura 19 – Resultados da execução dos testes de mutação após a adição do caso de teste (2, 2020)

```

bash - "ip-172-31 x  README.md x  [B] /cal_python/h x  test_cal.py
(venv) voclabs:~/environment/cal_python (master) $ mutmut run --paths-to-mutate=cal.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.

1. Running tests without mutations
⌚ Running...Done

2. Checking mutants
.: 233/233  🔥 127  ⌚ 0  😟 0  😞 106  🚫 0

```

Fonte: Autor

Ao gerar novamente o relatório HTML, nota-se, por meio da figura 20, que o mutante 27 não se encontra mais entre os mutantes “sobreviventes”, pois o próximo mutante “sobrevivente” depois do 12 é o 28.

Figura 20 – Trecho do arquivo cal.py.html após a adição do caso de teste (2, 2020)

## Mutant 12

```

--- cal.py
+++ cal.py
@@ -5,7 +5,7 @@
"""

-number_of_days_month = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
+number_of_days_month = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 31, 31]

def number_of_days(month, year):

```

## Mutant 28

```

--- cal.py
+++ cal.py
@@ -41,7 +41,7 @@

    k = 0

-    if (is_leap(year) and month > 2):
+    if (is_leap(year) and month > 3):
        k+= 1


```

Fonte: Autor

## 2. Projeto selecionado

Para a realização da questão 3 da atividade foi selecionado o projeto *dirty-equals*, do qual é uma biblioteca implementada em python usada para facilitar a criação de comparações de igualdade flexíveis e expressivas em testes. O projeto pode ser acessado por meio do seguinte link: <https://github.com/samuelcolvin/dirty-equals>. A figura 21 mostra a estrutura das pastas e arquivos no repositório do projeto. Pode-se observar que existe a pasta “tests”, da qual possui os casos de teste.

Figura 21 – Repositório GitHub da biblioteca dirty-equals

 samuelcolvin Uprev version (#103) ✓ <span>da29a2d · 3 weeks ago</span> <span>🕒 133 Commits</span>		
📁 .github	Uprev version (#103)	3 weeks ago
📁 dirty_equals	Uprev version (#103)	3 weeks ago
📁 docs	fix references to 3.7	10 months ago
📁 requirements	uprev test, lint and docs dependencies (#102)	3 weeks ago
📁 tests	uprev test, lint and docs dependencies (#102)	3 weeks ago
📄 .codecov.yml	more tests	2 years ago
📄 .gitignore	IsDataclass (#68)	last year
📄 .pre-commit-config.yaml	uprev test, lint and docs dependencies (#102)	3 weeks ago
📄 LICENSE	fix mypy	2 years ago
📄 Makefile	better dep constraints (#90)	10 months ago
📄 README.md	fix references to 3.7	10 months ago
📄 mkdocs.yml	uprev test, lint and docs dependencies (#102)	3 weeks ago
📄 pyproject.toml	Support Python 3.13 (#101)	3 weeks ago

Fonte: Autor

Primeiramente executou-se os casos de testes com o comando “`pytest -vv tests`”. Por meio da figura 22, pode-se notar que 550 testes dos 550 coletados passaram com um tempo de execução de 1.78s.



Figura 22 – Resultados da execução dos casos de teste definidos na pasta “tests”

```

bash - "ip-172-31" x [B] /dirty-equals/ x +
tests/test_strings.py::test_dirty_equals_true[foo-dirty35-True] PASSED [ 95%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty36-True] PASSED [ 96%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty37-True] PASSED [ 96%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty38-True] PASSED [ 96%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty39-True] PASSED [ 96%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty40-True] PASSED [ 96%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty41-True] PASSED [ 96%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty42-True] PASSED [ 97%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty43-True] PASSED [ 97%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty44-True] PASSED [ 97%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty45-True] PASSED [ 97%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty46-True] PASSED [ 97%]
tests/test_strings.py::test_dirty_equals_true[foo\ndirty47-True] PASSED [ 98%]
tests/test_strings.py::test_dirty_equals_true[foo\ndirty48-False] PASSED [ 98%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty49-False] PASSED [ 98%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty50-False] PASSED [ 98%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty51-True] PASSED [ 98%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty52-True] PASSED [ 98%]
tests/test_strings.py::test_dirty_equals_true[foo-dirty53-False] PASSED [ 99%]
tests/test_strings.py::test_regex_true PASSED [ 99%]
tests/test_strings.py::test_regex_bytes_true PASSED [ 99%]
tests/test_strings.py::test_regex_false PASSED [ 99%]
tests/test_strings.py::test_regex_false_type_error PASSED [ 99%]
tests/test_strings.py::test_is_any_str PASSED [100%]
===== 550 passed in 1.78s =====

```

Fonte: Autor

O próximo comando de teste utilizado foi o “pytest -vv tests --cov=dirty\_equals”, que além de executar os casos de teste em “tests”, gera um relatório de cobertura de código acerca dos arquivos de “dirty\_equals”. É possível notar, por meio da figura 23, a porcentagem de cobertura em cada um dos arquivos que estão na pasta “dirty\_equals”. Apenas o arquivo datetime.py não atingiu 100% de cobertura, no total o código fonte do projeto possui 99.47% de cobertura.

Figura 23 – Relatório de cobertura de código para a pasta “dirty\_equals”

```

bash - "ip-172-31" x [B] /dirty-equals/ x +
----- coverage: platform linux, python 3.10.12-final-0 -----
Name                               Stmts  Miss Branch BrPart  Cover
-----
dirty_equals/_init__.py             12      0      0      0  100.00%
dirty_equals/_base.py              109      0     28      0  100.00%
dirty_equals/_boolean.py            22      0      4      0  100.00%
dirty_equals/_datetime.py           95      6     40      0   95.56%
dirty_equals/_dict.py               76      0     36      0  100.00%
dirty_equals/_inspection.py          59      0     16      0  100.00%
dirty_equals/_numeric.py            126      0     30      0  100.00%
dirty_equals/_other.py              170      0     62      0  100.00%
dirty_equals/_sequence.py           84      0     46      0  100.00%
dirty_equals/_strings.py            51      0     28      0  100.00%
dirty_equals/_utils.py              23      0     10      0  100.00%
dirty_equals/version.py              1      0      0      0  100.00%
-----
TOTAL                               828      6    300      0   99.47%
===== 550 passed in 2.56s =====

```

Fonte: Autor

Em seguida executou-se o seguinte comando: “`pytest -vv tests --cov=dirty_equals --cov-branch --cov-report html`”, do qual executa os casos de teste, gera o relatório de cobertura mais detalhado e o exporta como um arquivo `.html`. O arquivo em formato HTML gerado é o “`index.html`”, ele pode ser acessado por meio da pasta `htmlcov`. A figura 24 mostra a tela principal do relatório. Segundo os dados do relatório, a pasta “`dirty_equals`” possui um total de 828 comandos, sendo que apenas 6 deles não estão cobertos pelos testes e 31 comandos foram excluídos da contagem de cobertura manualmente. Ademais existem 300 ramificações (*branches*), e a cobertura total é de 99.47%.

Figura 24 – Relatório de cobertura de código em formato HTML para “`dirty_equals`”

Coverage report: 99.47%

Files Functions Classes

*coverage.py v7.6.1, created at 2024-09-03 05:54 +0000*

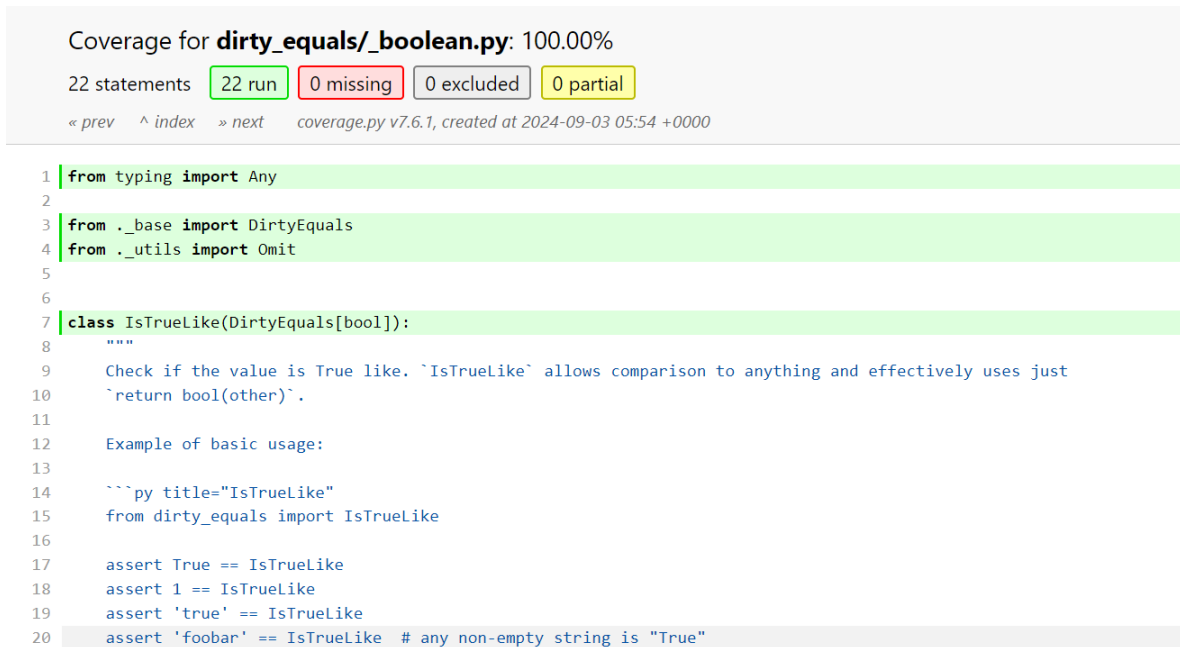
File ▲	statements	missing	excluded	branches	partial	coverage
dirty_equals/_init_.py	12	0	0	0	0	100.00%
dirty_equals/_base.py	109	0	3	28	0	100.00%
dirty_equals/_boolean.py	22	0	0	4	0	100.00%
dirty_equals/_datetime.py	95	6	2	40	0	95.56%
dirty_equals/_dict.py	76	0	4	36	0	100.00%
dirty_equals/_inspection.py	59	0	4	16	0	100.00%
dirty_equals/_numeric.py	126	0	0	30	0	100.00%
dirty_equals/_other.py	170	0	8	62	0	100.00%
dirty_equals/_sequence.py	84	0	10	46	0	100.00%
dirty_equals/_strings.py	51	0	0	28	0	100.00%
dirty_equals/_utils.py	23	0	0	10	0	100.00%
dirty_equals/version.py	1	0	0	0	0	100.00%
<b>Total</b>	<b>828</b>	<b>6</b>	<b>31</b>	<b>300</b>	<b>0</b>	<b>99.47%</b>

Fonte: Autor

Uma visualização mais detalhada da cobertura pode ser acessada ao selecionar as opções dos nomes dos arquivos. Dessa maneira, pode-se visualizar linha por linha, em verde o que está coberto pelos testes e em vermelho o que não está coberto pelos testes. A figura

25 é um trecho do relatório neste nível de detalhe para o arquivo “\_boolean.py”.

Figura 25 – Relatório de todas as linhas do arquivo “\_boolean.py”



Fonte: Autor

A fim de verificar se o conjunto de casos de teste é suficiente e adequado, foram utilizados os testes de mutação. Desse modo, usou-se o comando “mutmut run –paths-to-mutate=dirty\_equals” para executar os testes de mutação. A figura 26 apresenta os resultados da execução: 530 mutantes “mortos” e 138 “sobreviventes”.

Figura 26 – Resultados da execução dos testes de mutação nos arquivos de “dirty\_equals”

```

bash - "ip-172-31 x (+)
(venv) voclabs:~/environment/dirty-equals (main) $ mutmut run --paths-to-mutate=dirty_equals

- 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.

1. Using cached time for baseline tests, to run baseline again delete the cache file

2. Checking mutants
: 668/668 🚩 530 🕒 0 🧐 0 😬 138 🚫 0

```

Fonte: Autor

A figura 27 expõe um trecho do resultado obtido com o comando “mutmut results”, do qual permite verificar os mutantes que sobreviveram.

Figura 27 – Trecho do resultado dos mutantes “sobreviventes”

```

bash - "ip-172-31" x
(venv) voclabs:~/environment/dirty-equals (main) $ mutmut results
To apply a mutant on disk:
    mutmut apply <id>

To show a mutant:
    mutmut show <id>

Survived 🤖 (138)

---- dirty_equals/_base.py (13) ----
242, 244-247, 249-250, 254, 258-259, 275-277

---- dirty_equals/_datetime.py (20) ----
538, 556, 559-560, 575-576, 579, 582-585, 589, 591-593, 596-598, 603-604

---- dirty_equals/_dict.py (9) ----
461-462, 469-470, 481, 505, 515, 521, 524

---- dirty_equals/_inspection.py (9) ----
306-307, 312-313, 318-319, 338-340

---- dirty_equals/_numeric.py (30) ----

```

Fonte: Autor

Com o comando “mutmut show <mutante>”, pode-se visualizar os detalhes da alteração do mutante passado como argumento. A figura 28 mostra os detalhes do mutante 521.

Figura 28 – Alteração feita no mutante 521

```

bash - "ip-172-31" x
(venv) voclabs:~/environment/dirty-equals (main) $ mutmut show 521
--- dirty_equals/_dict.py
+++ dirty_equals/_dict.py
@@ -144,7 +144,7 @@
     r = self.ignore.__name__ if callable(self.ignore) else repr(self.ignore)
     modifiers += [f'ignore={r}']
     if self.strict != (name == 'IsStrictDict'):
-        modifiers += [f'strict={self.strict}']
+        modifiers = [f'strict={self.strict}']

     if modifiers:
         mod = f'[{", ".join(modifiers)}]'
```

Fonte: Autor

A fim de obter detalhes das alterações feitas em cada mutante “sobrevivente”, executou-se o comando “mutmut html”, do qual gera um arquivo em formato HTML para cada arquivo da pasta “dirty\_equals”, com informações sobre estes mutantes. A figura 29 apresenta o arquivo index.html que possui um atalho para cada um dos arquivos HTML associados aos arquivos da pasta “dirty\_equals”.

Figura 29 – Relatório dos testes de mutação (index.html)

## Mutation testing report

Killed 530 out of 668 mutants

File	Total	Skipped	Killed	% killed	Survived
<a href="#">dirty_equals/ base.py</a>	58	0	45	77.59	13
<a href="#">dirty_equals/ boolean.py</a>	13	0	13	100.00	0
<a href="#">dirty_equals/ datetime.py</a>	68	0	48	70.59	20
<a href="#">dirty_equals/ dict.py</a>	78	0	69	88.46	9
<a href="#">dirty_equals/ inspection.py</a>	41	0	32	78.05	9
<a href="#">dirty_equals/ numeric.py</a>	113	0	83	73.45	30
<a href="#">dirty_equals/ other.py</a>	146	0	117	80.14	29
<a href="#">dirty_equals/ sequence.py</a>	73	0	60	82.19	13
<a href="#">dirty_equals/ strings.py</a>	63	0	54	85.71	9
<a href="#">dirty_equals/ utils.py</a>	13	0	9	69.23	4
<a href="#">dirty_equals/version.py</a>	2	0	0	0.00	2

Fonte: Autor

Uma visão detalhada para cada arquivo pode ser acessada por meio do clique nos atalhos em azul. A figura 30 mostra um trecho dos detalhes dos resultados relatados no arquivo “\_numeric.py”, armazenados no arquivo “\_numeric.py.html”.

Figura 30 – Trecho do arquivo “\_numeric.py.html”

## dirty\_equals/\_numeric.py

Killed 83 out of 113 mutants

### Survived

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

#### Mutant 347

```
--- dirty_equals/_numeric.py
+++ dirty_equals/_numeric.py
@@ -23,7 +23,7 @@

    from ._utils import Omit

-AnyNumber = Union[int, float, Decimal]
+AnyNumber = None
    N = TypeVar('N', int, float, Decimal, date, datetime, AnyNumber)
```

#### Mutant 348

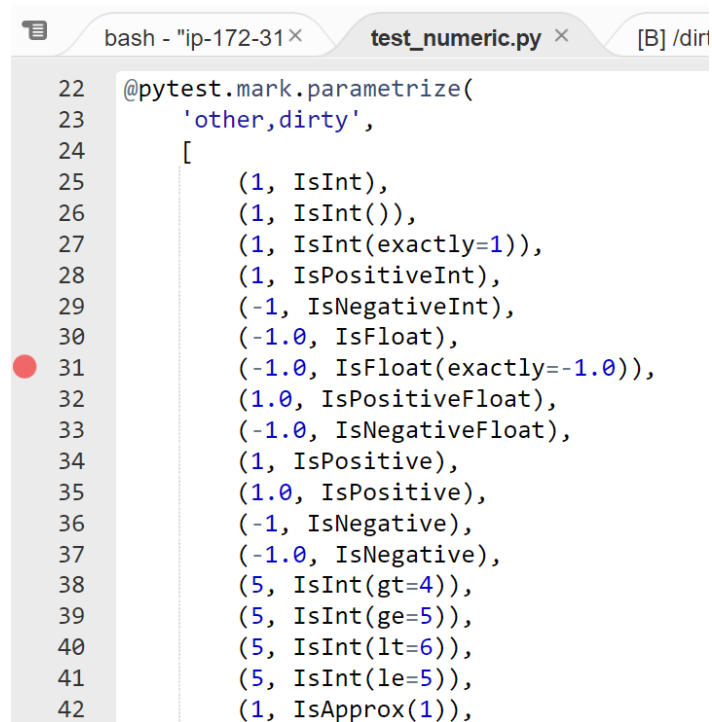
```
--- dirty_equals/_numeric.py
+++ dirty_equals/_numeric.py
@@ -24,7 +24,7 @@
```

Fonte: Autor

### 3. Melhorando os casos de teste

O objetivo desta última etapa da atividade é identificar limitações e oportunidades de melhoria nos casos de teste. De acordo com o exposto na figura 29, o arquivo “\_numeric.py” é onde reside o maior número de sobreviventes, logo o foco voltou-se a este arquivo na questão das melhorias. A principal oportunidade de melhora identificada nos métodos de teste “test\_dirty\_equals” e “test\_dirty\_not\_equals”, das quais compara um valor “other” com uma instância “dirty”. A priori, as classes de asserção como “IsInt” e “IsFloat” estavam testando majoritariamente o dígito 1, como pode-se observar na figura 31.

Figura 31 – Trecho do arquivo “test\_numeric.py”



```
22 @pytest.mark.parametrize(  
23     'other,dirty',  
24     [  
25         (1, IsInt),  
26         (1, IsInt()),  
27         (1, IsInt(exactly=1)),  
28         (1, IsPositiveInt),  
29         (-1, IsNegativeInt),  
30         (-1.0, IsFloat),  
31         (-1.0, IsFloat(exactly=-1.0)),  
32         (1.0, IsPositiveFloat),  
33         (-1.0, IsNegativeFloat),  
34         (1, IsPositive),  
35         (1.0, IsPositive),  
36         (-1, IsNegative),  
37         (-1.0, IsNegative),  
38         (5, IsInt(gt=4)),  
39         (5, IsInt(ge=5)),  
40         (5, IsInt(lt=6)),  
41         (5, IsInt(le=5)),  
42         (1, IsApprox(1)),
```

Fonte: Autor

Visto isso, investiu-se em enriquecer esses casos de teste com os demais dígitos. Além dos dígitos adicionou-se casos de teste que lidam com o valor “None”, do qual está presente em diversos mutantes inseridos. De acordo o arquivo “\_numeric.py.html”, 17 mutantes englobavam mudanças relacionadas ao valor “None”, dos 30 sobreviventes associados a “\_numeric.py”. A figura 32 mostra um trecho de “\_numeric.py.html” que engloba o valor “None”.



Figura 32 – Mutantes associados ao valor “None”

**Mutant 441**

```

--- dirty_equals/_numeric.py
+++ dirty_equals/_numeric.py
@@ -277,7 +277,7 @@

    def __init__(self) -> None:
        super().__init__(le=0)
-        self._repr_kwargs = {}
+        self._repr_kwargs = None

class IsInt(IsNumeric[int]):

```

**Mutant 444**

```

--- dirty_equals/_numeric.py
+++ dirty_equals/_numeric.py
@@ -325,7 +325,7 @@

    def __init__(self) -> None:
        super().__init__(gt=0)
-        self._repr_kwargs = {}
+        self._repr_kwargs = None

class IsNegativeInt(IsInt):

```

**Mutant 446**

```

--- dirty_equals/_numeric.py
+++ dirty_equals/_numeric.py
@@ -347,7 +347,7 @@

    def __init__(self) -> None:
        super().__init__(lt=0)
-        self._repr_kwargs = {}
+        self._repr_kwargs = None

```

Fonte: Autor

A figura 33 mostra um trecho de “\_test\_numeric.py” modificado com as mudanças propostas. As mudanças da figura 33 se referem ao método “test\_dirty\_not\_equals”, do qual testa que o valor passado como parâmetro não é igual a um Int, Float, etc.

Figura 33 – Trecho modificado em “\_test\_numeric.py”

```

199 (8.3, IsInt),
200 (8.4, IsInt),
201 (8.5, IsInt),
202 (8.6, IsInt),
203 (8.7, IsInt),
204 (8.8, IsInt),
205 (8.9, IsInt),
206 (8.1, IsInt),
207 (1, IsInt(exactly=2)),
208 (1, IsInt(exactly=3)),
209 (1, IsInt(exactly=4)),
210 (1, IsInt(exactly=5)),
211 (1, IsInt(exactly=6)),
212 (1, IsInt(exactly=7)),
213 (1, IsInt(exactly=8)),
214 (1, IsInt(exactly=9)),
215 (1, IsInt(exactly=0)),
216 (True, IsInt),
217 (False, IsInt),
218 (1.0, IsInt()),
219 (-1, IsPositiveInt),
220 (None, IsPositiveInt),
221 (0, IsPositiveInt),
222 (1, IsNegativeInt),
223 (0, IsNegativeInt),
224 (None, IsNegativeInt),
225 (1, IsFloat),
226 (None, IsFloat),
227 (1, IsFloat(exactly=1.0)),
228 (1, IsFloat(exactly=2.0)),
229 (1, IsFloat(exactly=3.0)),
230 (1, IsFloat(exactly=4.0)),
231 (1, IsFloat(exactly=5.0)),
232 (1, IsFloat(exactly=6.0)),
233 (1, IsFloat(exactly=7.0)),
234 (1, IsFloat(exactly=8.0)),

```

Fonte: Autor

Conforme pode ser visto na figura 34, após as mudanças propostas, os resultados da execução dos testes de mutação apontam que 546 mutantes foram “mortos”, 16 a mais quando comparado com a primeira execução. Como foram feitas mudanças apenas no arquivo “\_test\_numeric.py”, 16 dos 30 mutantes sobreviventes associados a esse módulo foram “mortos”.

Figura 34 – Resultados da execução 2 dos testes de mutação nos arquivos de “dirty\_equals”

```

bash - "ip-172-31" x test_numeric.py x [B] /dirty-equals/l x _numeric.py x
(venv) voclabs:~/environment/dirty-equals (main) $ clear
(venv) voclabs:~/environment/dirty-equals (main) $ mutmut run --paths-to-mutate=dirty_equals

- 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.

1. Running tests without mutations
.: Running...Done

2. Checking mutants
: 668/668 🔥 546 ⌚ 0 😟 0 😞 122 🚫 0

```

Fonte: Autor

Link para o repositório no GitHub:

[https://github.com/rafaseto/Teste\\_Software\\_Mutantes\\_2024\\_Goto\\_Rafael](https://github.com/rafaseto/Teste_Software_Mutantes_2024_Goto_Rafael)

Link para o vídeo tutorial:

<https://drive.google.com/file/d/1c2VrOFnHKoVNJPQoporD07G2AZu8S-CU/view?usp=sharing>