

THIAGO FREIRE DE CARVALHO

ATIVIDADE 2

TESTES DE SOFTWARE

Projeto escolhido via Github

Para a realização dos testes de mutação, foi escolhido o projeto do github *password generator*, que pode ser encontrado [aqui](#). Este pequeno projeto implementa classes necessárias para a geração de senhas aleatórias através de características indicadas pelo usuário, como o tamanho da senha e a dificuldade.

O projeto possui testes unitários e já utiliza a biblioteca do python *pytest*. No entanto, não possui cobertura de testes nem realiza testes de mutação. Nas seções a seguir, serão exibidos, respectivamente, os passos necessários para a configuração do projeto, cobertura de testes e testes de mutação.

Configurando o projeto

Primeiramente, o projeto foi clonado a partir do seguinte comando:

```
git clone https://github.com/Denrois/password_generator_with_unit_tests.git
```

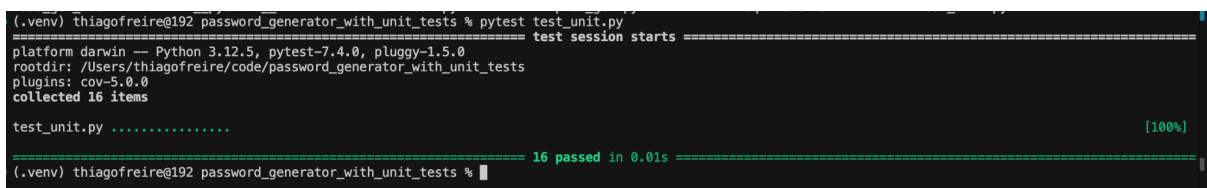
Em seguida, o projeto foi aberto na IDE Visual Studio Code, devido a sua facilidade com projetos Python.

O projeto contém um arquivo *requirements.txt*, portanto foi necessários instalar as devidas dependências através do comando:

```
pip install -r requirements.txt
```

Após isso, executamos os testes para verificação com o comando:

```
pytest
```

A terminal window showing the execution of a pytest command. The prompt is '(.venv) thiagofreire@192 password_generator_with_unit_tests %'. The command entered is 'pytest test_unit.py'. The output shows 'test session starts' with details about the platform (darwin), Python version (3.12.5), pytest version (7.4.0), and pluggy version (1.5.0). It also shows the root directory, plugins (cov-5.0.0), and that 16 items were collected. The test results for 'test_unit.py' are shown as a series of dots, followed by a green line indicating '16 passed in 0.01s' and a green progress bar at 100%.

Perceba que os testes foram executados com sucesso, mas a validação da efetividade dessa bateria de testes ainda pode ser feita com os testes de mutação.

Adicionando cobertura de testes

Como mencionado, o projeto em questão não possui uma cobertura de testes configurada, faremos isso a seguir.

Utilizaremos a biblioteca do python *pytest-cov*, ela pode ser instalada a partir do seguinte comando:

pip install pytest-cov

Após a instalação, podemos rodar o comando para verificar a cobertura de testes da aplicação:

`pytest --cov=pass_gen`

```
(.venv) thiagofreire@192 password_generator_with_unit_tests % pytest test_unit.py --cov=pass_gen
platform darwin -- Python 3.12.5, pytest-7.4.0, pluggy-1.5.0
rootdir: /Users/thiagofreire/code/password_generator_with_unit_tests
plugins: cov-5.0.0
collected 16 items

test_unit.py ..... [100%]

===== coverage: platform darwin, python 3.12.5-final-0 =====
Name          Stmts  Miss  Cover
-----
pass_gen.py    61     3    95%
TOTAL          61     3    95%

===== 16 passed in 0.04s =====
(.venv) thiagofreire@192 password_generator_with_unit_tests %
```

Percebemos que há uma cobertura de 95% dos testes em relação ao código da aplicação (um ótimo número). Podemos obter mais detalhes sobre a cobertura dos testes através da geração de um relatório com o comando:

`pytest --cov=pass_gen --cov-branch --cov-report html`

```
(.venv) thiagofreire@192 password_generator_with_unit_tests % pytest test_unit.py --cov=pass_gen --cov-branch --cov-report html
platform darwin -- Python 3.12.5, pytest-7.4.0, pluggy-1.5.0
rootdir: /Users/thiagofreire/code/password_generator_with_unit_tests
plugins: cov-5.0.0
collected 16 items

test_unit.py ..... [100%]

===== coverage: platform darwin, python 3.12.5-final-0 =====
Coverage HTML written to dir htmlcov

===== 16 passed in 0.05s =====
(.venv) thiagofreire@192 password_generator_with_unit_tests %
```

Após a execução, alguns arquivos HTML foram gerados e podemos observar de forma mais detalhada exatamente quais linhas do código não estão cobertas por testes.

```
Coverage for pass_gen.py: 92%
61 statements 58 run 3 missing 0 excluded 3 partial
« prev ^ index » next coverage.py v7.6.1, created at 2024-09-02 19:39 -0300

1 from random import choice, choices, shuffle
2 import string
3
4
5 class PassGen:
6     LOWER_CASE = list(string.ascii_lowercase)
7     UPPER_CASE = list(string.ascii_uppercase)
8     NUMBERS = list('0123456789')
9     SYMBOLS = list('!@&*')
10    EASY_WHOLE_LIST = LOWER_CASE + NUMBERS
11    MEDIUM_WHOLE_LIST = EASY_WHOLE_LIST + UPPER_CASE
12    HARD_WHOLE_LIST = MEDIUM_WHOLE_LIST + SYMBOLS
13
14    def __init__(self, length, difficult):
15        if length in range(5, 13):
16            self.length = length
17        else:
18            raise ValueError('Length should be between 5 and 12')
19        if difficult in ('easy', 'medium', 'hard'):
20            self.difficult = difficult
21        else:
22            raise ValueError('Difficult should be easy, medium or hard')
23
24    def __easy_pass(self):
25        """Create base with one random lower case letter and one random number from 0 to 9,
26        add to this base (user input length - 2(two elements are already used in base)) amount of random elements
27        from mixed list of lower case letters and numbers, shuffle it and return a string"""
28        pass_base_easy = list(choice(self.LOWER_CASE) + choice(self.NUMBERS))
29        final_list_easy = pass_base_easy + choices(self.EASY_WHOLE_LIST, k=self.length - 2)
30        shuffle(final_list_easy)
31        return ''.join(final_list_easy)
32
33    def __medium_pass(self):
```

O relatório indica que há 3 linhas não cobertas por testes (indicadas em vermelho) e 3 linhas parcialmente cobertas (indicadas em amarelo). É notável que, as linhas não testadas geram exceções por validação. Isso pode ser um indicativo de que os testes não estejam cobrindo de forma eficiente todas as possibilidades de entrada do usuário no programa.

Gerando mutações

Para gerar os testes de mutação, instalaremos a biblioteca mutmut, com o comando:

```
pip install mutmut
```

Após a instalação, geramos as mutações com o comando:

```
mutmut run
--paths-to-mutate=/Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
```

O parâmetro paths-to-mutate indica o caminho do script principal da aplicação, dependendo do ambiente esse caminho será diferente de acordo com o sistema e usuário do sistema.

Ao todo, foram geradas 72 mutações e 19 sobreviventes.

```
(.venv) thiagofreire@192 password_generator_with_unit_tests % mutmut run --paths-to-mutate=/Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.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
72/72 🔪 53 🟡 0 ⌚ 0 🟢 19 🚫 0
```

É possível obter um detalhamento maior dos resultados através do comando:

mutmut results

```
(.venv) thiagofreire@192 password_generator_with_unit_tests % mutmut results
To apply a mutant on disk:
  mutmut apply <id>

To show a mutant:
  mutmut show <id>

Survived 🟢 (19)

---- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py (19) ----

1-11, 15, 17, 23, 28, 61-62, 64, 69
(.venv) thiagofreire@192 password_generator_with_unit_tests %
```

Com a execução desse comando, conseguimos observar mais claramente quais mutações sobreviveram, e posteriormente detalhar cada mutação com o comando:

mutmut show <index-mutacao>

```
(.venv) thiagofreire@192 password_generator_with_unit_tests % mutmut show 11
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -9,7 +9,7 @@
     SYMBOLS = list('!?@&*')
     EASY_WHOLE_LIST = LOWER_CASE + NUMBERS
     MEDIUM_WHOLE_LIST = EASY_WHOLE_LIST + UPPER_CASE
-    HARD_WHOLE_LIST = MEDIUM_WHOLE_LIST + SYMBOLS
+    HARD_WHOLE_LIST = MEDIUM_WHOLE_LIST - SYMBOLS

     def __init__(self, length, difficult):
         if length in range(5, 13):
```

Por exemplo, vemos a mudança feita no código pela mutação 11, que sobreviveu.

Para obter um relatório completo em HTML, basta rodar o comando:

mutmut html

/Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py

Killed 60 out of 72 mutants

Survived

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

Mutant 1

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -3,7 +3,7 @@
```

```
class PassGen:
-     LOWER_CASE = list(string.ascii_lowercase)
+     LOWER_CASE = None
+     UPPER_CASE = list(string.ascii_uppercase)
    NUMBERS = list('0123456789')
    SYMBOLS = list('!7&*')
```

Mutant 2

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -4,7 +4,7 @@
```

```
class PassGen:
-     LOWER_CASE = list(string.ascii_lowercase)
-     UPPER_CASE = list(string.ascii_uppercase)
+     UPPER_CASE = None
+     NUMBERS = list('0123456789')
    SYMBOLS = list('!7&*')
    EASY_WHOLE_LIST = LOWER_CASE + NUMBERS
```

Mutant 3

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -5,7 +5,7 @@
```

```
class PassGen:
-     LOWER_CASE = list(string.ascii_lowercase)
-     UPPER_CASE = list(string.ascii_uppercase)
+     NUMBERS = list('0123456789')
-     NUMBERS = list('XX0123456789XX')
+     SYMBOLS = list('!7&*')
```

Dessa forma é possível ter uma lista das mutações que sobreviveram e suas respectivas alterações. Vamos explorar alguns exemplos para identificar o motivo de sobrevivência dessas mutações.

O mutante 62 indica que não há testes suficientes para garantir a validação da quantidade de senhas geradas por vez. Isso acontece pois não há testes para o funcionamento do construtor da classe. Podemos adicionar esse teste e assim eliminar o mutante.

Mutant 62

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -66,7 +66,7 @@
```

```
class PassList(PassGen):
    def __init__(self, length, difficult, amount):
        super().__init__(length, difficult)
-     if amount in range(1, 101):
+     if amount in range(1, 102):
        self.amount = amount
    else:
        raise ValueError('Amount should be between 1 and 100')
```

Mutante 62: Foi alterado o range permitido da quantidade de senhas geradas

```
@pytest.mark.parametrize('amount', [0, 101])
def test_amount_neg_init(self, amount):
    with pytest.raises(ValueError):
        self.obj.__init__(6, "easy", amount)
```

Teste: Caso de teste adicionado para validação no construtor

Os mutantes 74, 64 e 17 indicam que os casos de teste não garantem que a mensagem exibida ao usuário em caso de erro é a definida pelo programador. Para esse projeto, não faria muita diferença, mas existem casos onde as mensagens exibidas são fundamentais para o entendimento do erro pelo usuário. Podemos eliminar esses mutantes adicionando novos testes.

Mutant 74

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -19,7 +19,7 @@
     if difficult in ('easy', 'medium', 'hard'):
         self.difficult = difficult
     else:
-         raise ValueError('Difficult should be easy, medium or hard')
+         raise ValueError('XXDifficult should be easy, medium or hardXX')

def __easy_pass(self):
    """Create base with one random lower case letter and one random number from 0 to 9,
```

Mutant 64

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -69,7 +69,7 @@
     if amount in range(1, 101):
         self.amount = amount
     else:
-         raise ValueError('Amount should be between 1 and 100')
+         raise ValueError('XXAmount should be between 1 and 100XX')

def __create_pass_list(self):
    return [self.get_pass() for self.p in range(self.amount)]
```

Mutant 17

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -15,7 +15,7 @@
     if length in range(5, 13):
         self.length = length
     else:
-         raise ValueError('Length should be between 5 and 12')
+         raise ValueError('XXLength should be between 5 and 12XX')
     if difficult in ('easy', 'medium', 'hard'):
         self.difficult = difficult
     else:
```

Casos de teste adicionados para validar as mensagens de erro nas validações do construtor da classe:

```
def test_difficult_wrong_init(self):
    difficult = 'qwerty'
    with pytest.raises(ValueError, match=r"Difficult should be easy, medium or hard$"):
        self.obj.__init__(6,difficult)
```

```
@pytest.mark.parametrize('length_neg', [4, 13])
def test_length_init(self, length_neg):
    with pytest.raises(ValueError, match=r"Length should be between 5 and 12$"):
        self.obj.__init__(length_neg, "easy")
```

```
@pytest.mark.parametrize('amount', [0, 101])
def test_amount_neg_init(self, amount):
    with pytest.raises(ValueError, match=r"Amount should be between 1 and 100$"):
        self.obj.__init__(6, "easy", amount)
```

Com essas alterações, já foi possível diminuir o número de mutantes sobreviventes para 12:

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

```
2. Checking mutants
```

```
:: 72/72 🎉 60 🕒 0 🤔 0 😞 12 🚫 0
```

Algumas outras alterações poderiam ser feitas nos testes para resolver o problema dos outros mutantes sobreviventes:

Os mutantes 1, 2 e 3, por exemplo, indicam que os testes não garantem 100% a estrutura das senhas geradas. Segundo a própria documentação do projeto, as senhas de nível fácil devem conter pelo menos: letras minúsculas e números de 0 a 9. Já as senhas de nível médio devem conter o mesmo nível de atributos das senhas de nível fácil + letras maiúsculas. As mutações em questão sobrevivem devido ao fato de os testes utilizarem como validação as listas de elementos possíveis para cada nível que são definidas dentro da própria classe. Para resolver, poderíamos adicionar essa lista diretamente no contexto do teste.

Mutant 1

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -3,7 +3,7 @@

class PassGen:
-     LOWER_CASE = list(string.ascii_lowercase)
+     LOWER_CASE = None
     UPPER_CASE = list(string.ascii_uppercase)
     NUMBERS = list('0123456789')
     SYMBOLS = list('!@&*')
```

Mutant 2

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -4,7 +4,7 @@

class PassGen:
     LOWER_CASE = list(string.ascii_lowercase)
-     UPPER_CASE = list(string.ascii_uppercase)
+     UPPER_CASE = None
     NUMBERS = list('0123456789')
     SYMBOLS = list('!@&*')
     EASY_WHOLE_LIST = LOWER_CASE + NUMBERS
```

Mutant 3

```
--- /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
+++ /Users/thiagofreire/code/password_generator_with_unit_tests/pass_gen.py
@@ -5,7 +5,7 @@

class PassGen:
     LOWER_CASE = list(string.ascii_lowercase)
     UPPER_CASE = list(string.ascii_uppercase)
-     NUMBERS = list('0123456789')
+     NUMBERS = list('XX0123456789XX')
     SYMBOLS = list('!@&*')
     EASY_WHOLE_LIST = LOWER_CASE + NUMBERS
     MEDIUM_WHOLE_LIST = EASY_WHOLE_LIST + UPPER_CASE
```

Todo o código deste projeto pode ser encontrado no repositório do [github](https://github.com).