07

Do as I did: test coverage

It's time for you to implement what you saw in class!

In this class, we learned how important it is for the project to be covered by tests and how to generate reports that allow the visualization of this code coverage.

So, have you gotten your hands dirty yet?

It's time for you to check the test coverage in your code!

If you have any questions, check the progress of your project by clicking on **Instructor's Opinion** .

Instructor's opinion

1) We start by installing the Pytest *coverage* tool , Pytest-cov, through the terminal/command prompt;

```
pip install pytest-cov==3.0.0
```

COPY CODE

2) We update the file `requirements.txt` with the following command in the terminal/command prompt;

```
pip freeze > requirements.txt
```

COPY CODE

3) We run the test coverage tool for the first time with the following code:

```
pytest –cov=codigo tests/
```

COPY CODE

4) We noticed that some code snippets in the file `bytebank.py` are not being covered. To find which snippets these are, we use the following command:

```
pytest –cov=codigo tests/ –cov-report term-missing
```

COPY CODE

5) We also learned a way to generate a test coverage report in HTML using the following command:

```
pytest –cov=codigo tests/ –cov-report html
```

COPY CODE

6) We can see that the section that does not have a test is the one that has the method `__str__()`;

7) To get 100% test coverage, we create a test for the method `__str__()`;

```
def test_retorno_str(self):
        nome, data_nascimento, salario = 'Teste', '12/03/2000', 1000  #
        esperado = 'Funcionario(Teste, 12/03/2000, 1000)'

         funconario_teste = Funcionario(nome, data_nascimento, salario)
         resultado = funconario_teste.__str__() # when

         assert resultado == esperado  # then
```

COPY CODE

8) However, we learned that there are certain code snippets that perform intrinsic Python language functionality and that it doesn't make sense to test them, since as developers, we assume that the language works as it should.

9) With this in mind, we delete the test done on the method `__str__()` and create the file `.coveragerc`, which has a similar purpose to `pytest.ini` and allows us to change some of the default settings of pytest-cov;

10) The file `.coveragerc` gives us the possibility to exclude lines of code from the `bytebabank.py`, whose test coverage we do not want to check;

```
[run]

[report]
exclude_lines =
   def __str__
```

COPY CODE

11) We improved the execution of the coverage tool by defining in the file `pytest.ini` that whenever we type the code `pytest` in the terminal, not only the tests must be executed, but also the code coverage report;

```
    [pytest]
addopts = -v --cov=codigo tests/ --cov-report term-missing
markers =
    calcular_bonus: Teste para o metodo calcular_bonus
```

COPY CODE

12) We also changed the file `.coveragerc` so that whenever we run the command, `pytest --cov` the test coverage analysis is only done in the file `bytebank.py`, as well as defining that whenever an HTML report is generated, it must be inside a directory called `coverage_relatorio_html`.

```
    [run]
source = ./codigo
```

```
[report]
exclude_lines =
    def __str__

[html]
directory = coverage_relatorio_html
```

COPY CODE

13) To create a test report in the XML format, simply use the following command in the terminal/command prompt:

```
pytest –junitxml report.xml
```

COPY CODE

14) To create a coverage report in the XML format, simply use the following command in the terminal/command prompt:

```
pytest –cov-report xml
```

COPY CODE