Teste de Software - Atividade 1

Aluno: Rodrigo Nunes de Santana

1. Links

1.1. Github: https://github.com/rodrigodesan/Teste_Software_2024_Santana_Rodrigo

1.2. Stackoverflow:https://stackoverflow.com/questions/23337471/how-do-i-properly-assert-that-an-exception-gets-raised-in-pytest

2. Tutorial

2.1. Pergunta/resposta escolhida

Para a realização da atividade, a pergunta escolhida foi “How do I properly assert that an exception gets raised in pytest?” que trata de como testar o levantamento de exceções utilizando o framework de testes de unidade *Pytest,* para a linhagem *Python*. A resposta escolhida como correta pelo autor da pergunta traz diversos exemplos de utilização desta aplicação, aplicadas à divisão por 0, demonstrando o quais seriam as boas e as más práticas para esse caso, mostrando o código e a saída para a execução do teste.

971

Code

```
import pytest

def test_passes():
    with pytest.raises(Exception) as e_info:
        x = 1 / 0

def test_passes_without_info():
    with pytest.raises(Exception):
        x = 1 / 0

def test_fails():
    with pytest.raises(Exception) as e_info:
        x = 1 / 1

def test_fails_without_info():
    with pytest.raises(Exception):
        x = 1 / 1

# Don't do this. Assertions are caught as exceptions.
def test_passes_but_should_not():
    try:
        x = 1 / 1
        assert False
    except Exception:
        assert True

# Even if the appropriate exception is caught, it is bad style,
# because the test result is less informative
# than it would be with pytest.raises(e)
# (it just says pass or fail.)

def test_passes_but_bad_style():
    try:
        x = 1 / 0
        assert False
```

Output

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.6 -- py-1.4.26 -- pytest-2.6.4
collected 7 items

test.py ..FF..F

=================================== FAILURES ===================================
_____________________________ test_fails_without_info _____________________________

    def test_fails():
        with pytest.raises(Exception) as e_info:
>           x = 1 / 1
E           Failed: DID NOT RAISE

test.py:13: Failed
_____________________________ test_fails_without_info _____________________________

    def test_fails_without_info():
        with pytest.raises(Exception):
>           x = 1 / 1
E           Failed: DID NOT RAISE

test.py:17: Failed
_____________________________ test_passes_but_bad_style _____________________________

    def test_fails_but_bad_style():
        try:
            x = 1 / 1
>           assert False
E           assert False

test.py:43: AssertionError
==========
```

2.2. IDE

A IDE utilizada para a aplicação da solução foi o Visual Studio Code, com o auxílio da extensão Python. Além disso, foi utilizado o gerenciador de pacotes Poetry para a iniciação de um projeto em python e instalação do Pytest. Esse processo foi feito com os seguintes passos:

Primeiramente, cria-se uma pasta com o nome do projeto, acessa-se a mesmo e executa-se o comando “poetry init”. Com isso, são feitas uma série de perguntas a respeito do projeto, inclusive já nesse momento selecionando o Pytest e sua versão para a instalação. Isso gera um arquivo “pyproject.tml”, com as dependências e configurações do projeto.

```
→  Teste_Software_2024_santana_rodrigo poetry init

This command will guide you through creating your pyproject.toml config.

Package name [teste_software_2024_santana_rodrigo]:
Version [0.1.0]:
Description []:
Author [None, n to skip]:  Rodrigo <hiprodrigo@gmail.com>
License []:
Compatible Python versions [^3.10]:

Would you like to define your main dependencies interactively? (yes/no) [yes] yes
You can specify a package in the following forms:
  - A single name (requests): this will search for matches on PyPI
  - A name and a constraint (requests@^2.23.0)
  - A git url (git+https://github.com/python-poetry/poetry.git)
  - A git url with a revision (git+https://github.com/python-poetry/poetry.git#develop)
  - A file path (../my-package/my-package.whl)
  - A directory (../my-package/)
  - A url (https://example.com/packages/my-package-0.1.0.tar.gz)

Package to add or search for (leave blank to skip): pytest
Found 20 packages matching pytest
Showing the first 10 matches

Enter package # to add, or the complete package name if it is not listed []:
 [ 0] pytest
 [ 1] 131228_pytest_1
 [ 2] pytest123
 [ 3] pytest-collect-pytest-interinfo
 [ 4] pytest-pingguo-pytest-plugin
 [ 5] andluo_pytest
 [ 6] behave-pytest
 [ 7] bvx-pytest
 [ 8] aiida-pytest
 [ 9] aiogram-pytest
 [ 10]
 > 0
Enter the version constraint to require (or leave blank to use the latest version):
```

```toml
[tool.poetry]
name = "teste-software-2024-santana-rodrigo"
version = "0.1.0"
description = ""
authors = ["Rodrigo <hiprodrigo@gmail.com>"]
readme = "README.md"

[tool.poetry.dependencies]
python = "^3.10"
pytest = "^8.3.2"


[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Após isso, executa-se o comando “poetry install”, no qual o poetry instala as dependências e cria um ambiente virtual

poetry.lock

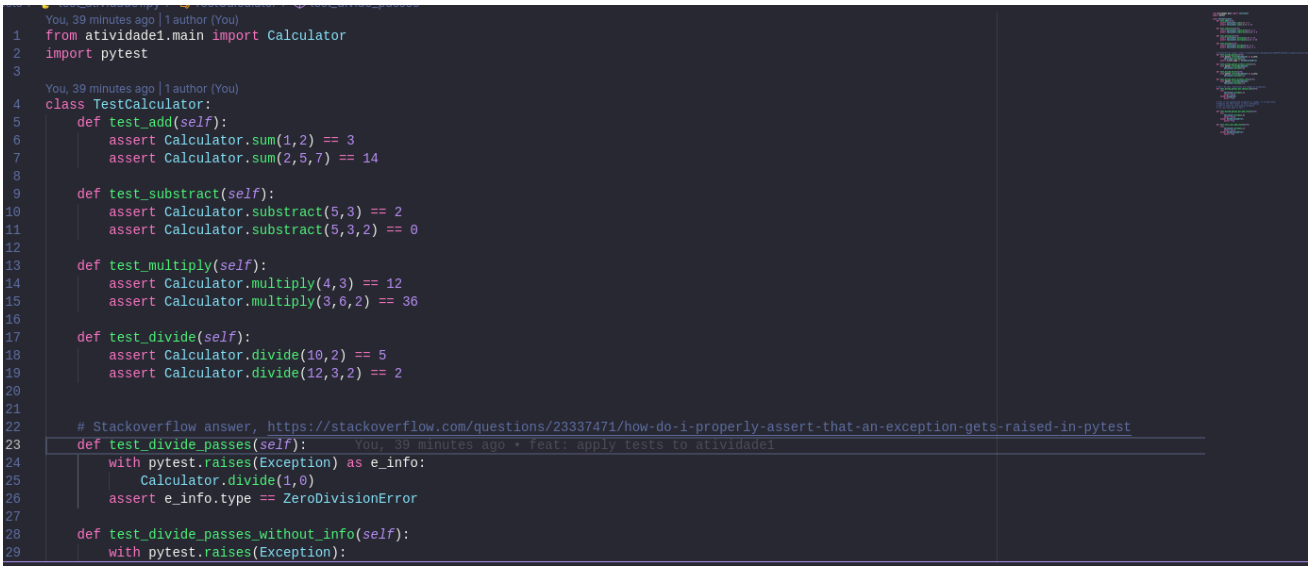
```toml
# This file is automatically @generated by Poetry 1.8.3 and should not be changed by hand.

[[package]]
name = "autopep8"
version = "2.3.1"
description = "A tool that automatically formats Python code to conform to the PEP 8 style guide"
optional = false
python-versions = ">=3.8"
files = [
    {file = "autopep8-2.3.1-py2.py3-none-any.whl", hash = "sha256:a203fe0fcad7939987422140ab17a930f684763bf7335bdb6709991dd7ef6c2d"},
    {file = "autopep8-2.3.1.tar.gz", hash = "sha256:8d6c87eba648fdcfc83e29b788910b8643171c395d9c4bcf115ece035b9c9dda"},
]

[package.dependencies]
pycodestyle = ">=2.12.0"
```

2.3. Aplicação da solução

Para a aplicação da solução, foi criada uma pasta “./tests” e uma pasta “./atividade1”, cada uma com um arquivo “__init__.py”, necessário para o funcionamento correto do python. Além disso,

como no exemplo do stackoverflow não estavam sendo utilizados para teste nenhuma função ou classe específicas, apenas testes diretos em valores, foi criada uma classe simples, ‘Calculator’ para a utilização nos testes em um arquivo “main.py”.

Após isso, foi criado na pasta tests um arquivo “atividade1.py”, com uma classe de testes “TestCalculadora”. Nessa classe, foram criados testes básicos para as operações descritas na classe e, além disso, foram aplicados à classe “Calculator” os testes de exceção na divisão por 0 apresentados na resposta do stackoverflow.



Com a execução do testes, com o comando “poetry run pytest tests”, 8 dos 11 testes passam, mas 3 falham, justamente os 3 indicados pela resposta que falharam, já que as respostas geradas seriam casos que não levantaram exceção.