Análise e Teste de Software

Testes Unitários e Cobertura - Parte 3

Universidade do Minho

2024/2025

1 Testes Unitários em Python

Os exercícios realizados nas aulas anteriores focaram-se na linguagem de programação Java. O foco desta ficha é a realização de exercícios de unit testing semelhantes, sobre classes da linguagem Python. Os conceitos utilizados deverão ser semelhantes, muda apenas a plataforma nos quais os conceitos serão aplicados. Note-se que vários dos comandos listados poderão ser ligeiramente diferentes de máquina para máquina, p.e. poderá ser necessário invocar `python3` em vez de `python`.

Embora não necessário, recomenda-se a criação de um *virtual environment* para o projeto. Aqui, é criado um com o nome `.venv`, e este irá consistir numa pasta dentro do projeto que terá tudo o que é necessário para trabalhar com o projeto. Isto também significa que quando não for necessário trabalhar mais com o projeto, apagar a pasta do projeto remove do sistema todas as dependências que foram instaladas para trabalhar com ele.

```
python -m venv .venv
```

Note-se que poderá ser necessário instalar o `python-venv` (ou semelhante, dependendo da máquina que está a ser usada). Se o comando anterior correr sem problemas, este passo pode ser ignorado.

```
sudo apt install python-venv
```

Estando o ambiente virtual criado, este pode ser ativado desta forma:

```
source .venv/bin/activate
```

sendo que o terminal deverá mudar para refletir que o ambiente virtual está ativo.

Agora com o ambiente virtual ativo, a instalação de dependências do projeto deverá ser local a esse mesmo projeto. Uma forma simpática de gerir dependências de um projeto em Python é colocar todas as dependências dentro de um ficheiro `requirements.txt` que poderá ser usado para fazer a sua instalação assim:

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

Existem várias formas de invocar o `pytest` para executar testes unitários num módulo. A sua forma mais elementar será apenas invocar o comando e analisar os resultados que este consegue produzir, mas pode-se consultar a documentação[1] para obter instruções mais detalhadas de execução.

```
pytest
```

Para análise de cobertura, a dependência `coverage` está incluída no ficheiro de dependências. Pode-se correr testes unitários com análise de cobertura com o seguinte comando:

```
coverage run -m pytest
```

Isto não mostra a cobertura dos testes, apenas os irá executar e guardar a informação de cobertura relevante. Esta informação pode depois ser consultada no terminal:

```
coverage report
```

ou utilizada para gerar um relatório HTML numa pasta `htmlcov`:

```
coverage html
```

Para terminar uma sessão de desenvolvimento deste projeto, pode-se terminar o uso do ambiente virtual com o seguinte comando:

```
deactivate
```

2 O projeto Bank

O projeto fornecido para este exercício consiste em duas classes de código, `BankAccount` e `Customer`, assim como dois ficheiros com os testes correspondentes, `test_bank_account` e `test_customer`.

Uma `BankAccount` tem de permitir quatro operações:

- `deposit` - O depósito de uma certa quantia monetária.
- `withdraw` - O levantamento de uma certa quantia monetária.
- `interest` - O cálculo de uma taxa de juro (valor em percentagem, p.e. 0.05 representa 5%) sobre o valor atual da conta. Uma taxa de juro positiva deverá aumentar o saldo da conta.
- `fee` - Aplicação de uma taxa/imposto (em valor absoluto) na conta. Uma `fee` positiva deverá reduzir o saldo da conta.

A variável `transaction_history` deverá guardar um registo de todas as transações efetuadas numa conta, de acordo com as 4 *keywords* a negrito acima.

Um objeto `Customer` deverá ser capaz de processar transações dos tipos `deposit` e `withdraw`, sendo que terá de pagar uma taxa associada com a conta sempre que fizer uma destas operações.

[1] `https://docs.pytest.org/en/stable/how-to/usage.html`

3 Exercícios

1. Crie um ambiente virtual para este projeto e inicie-o. Instale todas as dependências necessárias para o projeto.

2. Execute os testes unitários fornecidos no projeto. Verifique se todos os testes unitários passam.

3. Todos os testes unitários passarem não é imediatamente uma indicação de boa qualidade de código. Porquê? Analise a cobertura do código, e escreva testes unitários para maximizar a cobertura.

4. O código fornecido tem vários bugs. Encontre-os utilizando testes unitários, garantindo que tem um teste unitário correspondente a cada bug. Corrija todos os bugs do projeto.