Alunos: Thierry Pierre Dutoit e Álvaro Carvalho

1. O projeto inicial dos casos de teste (primeira aula)
2. O que foi feito como teste manual e/ou por script (o que foi testado e como) na segunda aula e os resultados dos testes
3. O arquivo contendo a automação dos testes usando unittest ou Pytest e resultados da execução
4. Comentário final (um ou dois parágrafos) sobre o processo.

**Projeto Inicial**

Na primeira aula, tivemos como projeto inicial pensar em quais seriam bons testes para determinar se um triângulo era escaleno, isósceles ou equilátero. Não foi mencionado no problema informações como tipo das variáveis, se alguma delas poderia ser nula, se a ordem importava etc. A ideia era realmente essa! Forçar com que fizéssemos testes que cobrissem todos esses casos.

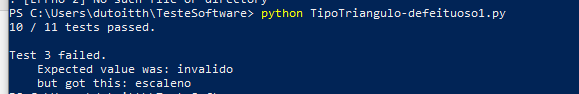
Para um problema relativamente simples, pensamos em 11 casos de testes. No final, dos 14 pontos mencionados pelo próprio autor do livro, conseguimos suprir 10. Um bom aproveitamento, mas fora do esperado pela simplicidade da função. Percebemos que, por mais simples um problema possa ser, o processo de projetar testes é bem desafiador.

**O Que Fizemos**

Na segunda aula implementamos os testes criados na primeira aula. Utilizando Python, pegamos os dois primeiros códigos que analisavam a classificação do triângulo e os testamos. Inicialmente, testamos manualmente, inserindo os valores dos casos de testes ao executar a função pelo terminal. Foi necessário rodar cada função 11 vezes.

Nossos testem identificaram um defeito no TipoTriangulo-defeituoso1 e nenhum defeito no TipoTriangulo-defeituoso2.

Em seguida, criamos um script para executar os testes. Com esse script, bastou escrever os casos de testes numa lista e iterar por ela fazendo a chamada na função a ser testada. Então, comparávamos o valor esperado com o valor obtido. Se iguais, o teste passou e nenhum defeito foi encontrado. Caso contrário, sabemos qual teste encontrou o defeito, como podemos ver no seguinte print:



Até esse ponto já temos uma boa automação. No entanto, ela não é escalável para um grande projeto. Pensando nisso, bibliotecas foram criadas para automatizar e facilitar o processo de executar os testes.

Como o próprio trabalho indica, usaremos o unittest do Python como ferramenta para automatizar nossos testes. Os novos arquivos criados podem ser encontrados no github... e estão no folder de Tests.

O resultado encontrado é parecido com a automação que fizemos, mas a ferramenta, a longo prazo, é mais indicada.

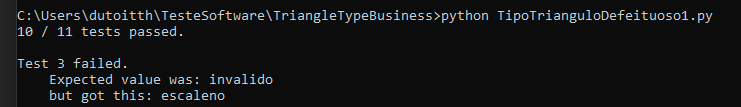
**Nossos Resultados**

Já mencionamos, de maneira breve, os nossos resultados. Em nossos testes manuais, na automação e com o unittest obtivemos os mesmos resultados.

Os resultados são que encontramos um defeito no TipoTriangulo-defeituoso1, e não encontramos nenhum defeito no TipoTriangulo-defeituoso2 (por mais que defeitos estejam presentes nele também).

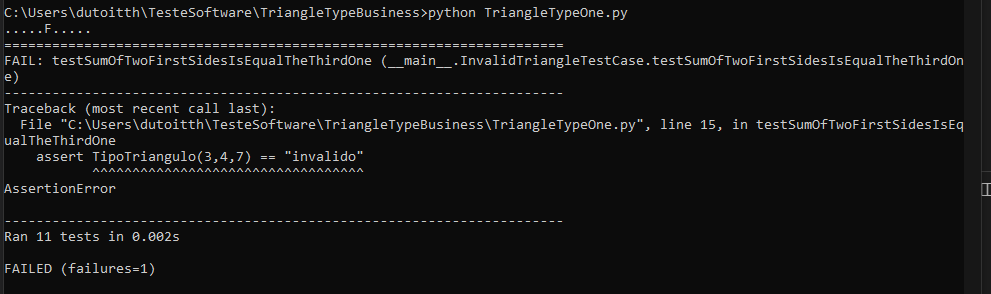
Nos testes manuais, a validação foi totalmente manual. Inserimos os valores e comparamos o valor retornado com o resultado esperado. Ainda não tínhamos mensagens de validação para a divergência entre o retorno e o valor esperado, então o critério de validação era a nossa própria análise.

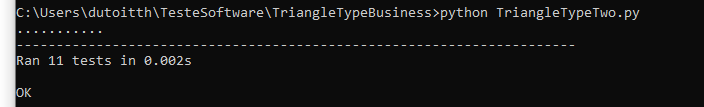
Com a automação inicial isso mudou. A automação conseguia comparar os resultados para nós, e indicar qual teste houve divergência. Através dos prints abaixo podemos ver isso:





Com o unittest, o resultado foi o mesmo, mas a mensagem de falha do teste de unidade é mais robusta, como podemos ver no print abaixo.





**Conclusão**

Foi bem interessante passar pelo processo de projetar os testes e criar a automação para executá-los. Vimos que até para funções simples o processo de teste é desafiador, mas recompensador, visto que conseguimos identificar um defeito no primeiro tipo de triângulo. No mais, mesmo que não tivéssemos identificados defeitos, teríamos certeza de que, nos casos que projetamos, a função funcionaria corretamente. De início, não tivemos um critério de cobertura, e talvez com um em mente teríamos projetados testes que conseguissem relevar todos os defeitos.