# 1 Desafio de Programação

É proposto a seguir um desafio de programação, um desafio de devops e um desafio opcional. A resposta do desafio de programação deve ser implementada em Python 3, e executada em um ambiente contendo apenas as bibliotecas padrão da linguagem. Tal ambiente será o resultante da execução dos seguintes comandos:

```
conda create -y -n myenv python=3.9
conda activate myenv
```

Os enunciados (tanto de Programação quanto de DevOps) abaixo são versões reduzidas de problemas com que o time de Tecnologia da BWGI se deparou no último ano. Todas as respostas serão avaliadas tanto do ponto de vista funcional, com testes automáticos, tanto quanto sob o aspecto da qualidade do código fonte produzido (clareza, eficiência, complexidade, etc.) quanto do ambiente em si.

## 1.1 reconcile_accounts

Escreva uma função que faça a conciliação (ou batimento) entre dois grupos de transações financeiras.

Sua função, `reconcile_accounts`, deve receber duas listas de listas (representando as linhas dos dados financeiros) e deve devolver cópias dessas duas listas de listas com uma nova coluna acrescentada à direita das demais, que designará se a transação pôde ser encontrada (`FOUND`) na outra lista ou não (`MISSING`).

As listas de listas representarão os dados em quatro colunas:

- Data (em formato `YYYY-MM-DD`)
- Departamento
- Valor
- Beneficiário

Todas as colunas serão representadas como *strings*.

Dados o arquivo `transactions1.csv`:

```
2020-12-04,Tecnologia,16.00,Bitbucket
2020-12-04,Jurídico,60.00,LinkSquares
2020-12-05,Tecnologia,50.00,AWS
```

e o arquivo `transactions2.csv`:

```
2020-12-04,Tecnologia,16.00,Bitbucket
2020-12-05,Tecnologia,49.99,AWS
2020-12-04,Jurídico,60.00,LinkSquares
```

sua função `reconcile_accounts` deve funcionar do seguinte modo:

```
>>> import csv
>>> from pathlib import Path
>>> from pprint import pprint
>>> transactions1 = list(csv.reader(Path('transactions1.csv').open()))
>>> transactions2 = list(csv.reader(Path('transactions2.csv').open()))
>>> out1, out2 = reconcile_accounts(transactions1, transactions2)
>>> pprint(out1)
[['2020-12-04', 'Tecnologia', '16.00', 'Bitbucket', 'FOUND'],
 ['2020-12-04', 'Jurídico', '60.00', 'LinkSquares', 'FOUND'],
 ['2020-12-05', 'Tecnologia', '50.00', 'AWS', 'MISSING']]
>>> pprint(out2)
[['2020-12-04', 'Tecnologia', '16.00', 'Bitbucket', 'FOUND'],
 ['2020-12-05', 'Tecnologia', '49.99', 'AWS', 'MISSING'],
 ['2020-12-04', 'Jurídico', '60.00', 'LinkSquares', 'FOUND']]
```

Sua função deve levar em conta que em cada arquivo pode haver transações duplicadas. Nesse caso, a cada transação de um arquivo deve corresponder uma única outra transação do outro.

Cada transação pode corresponder a outra cuja data seja do dia anterior ou posterior, desde que as demais colunas contenham os mesmos valores. Quando houver mais de uma possibilidade de correspondência para uma dada transação, ela deve ser feita **com a transação que ocorrer mais cedo**. Por exemplo, uma transação na primeira lista com data 2020-12-25 deve corresponder a uma da segunda lista, ainda sem correspondência, de data 2020-12-24 antes de corresponder a outras equivalentes (a menos da data) com datas 2020-12-25 ou 2020-12-26.

# 2 Desafio de DevOps

## 2.1 Criando um ambiente Airflow com Terraform e Ansible

O objetivo deste desafio é criar um ambiente Airflow com Terraform e Ansible. O ambiente deve ser criado na AWS via Terraform utilizando EKS e suas dependências. O Airflow deve ser

instalado utilizando Ansible.

Para tal, deve-se utilizar a imagem padrão do Airflow disponível no Docker Hub (versão 2.10.5 com Python 3.9).

Será necessário um Load Balancer para o módulo de web service.

Avaliaremos os códigos Terraform e Ansible do ponto de vista técnico e funcional.

Replicaremos (com as alterações necessárias) o código em nosso ambiente, utiliando o seguinte passo a passo:

- Avaliaremos os arquivos `.tf`, `playbook.yml` e `hosts.yml`
- Rodaremos os comandos: terraform init, plan e apply em ordem.
- Executaremos o comando:

```
ansible-playbook playbook.yml -u ec2-user – private-key airflow-iac2.pem -i
hosts.yml
```

(fazendo uso de nossas próprias chaves e hosts).

Todos os códigos precisam estar versionados em alguma plataforma git, seja GitHub, GitLab ou Bitbucket.