Desafios de Programação

São propostos a seguir três desafios de programação. As respostas devem ser implementadas em Python 3, e serão executadas em um ambiente contendo apenas as bibliotecas padrão da linguagem.

Os enunciados abaixo são versões simplificadas de problemas reais que ja enfrentamos. Todas as respostas serão avaliadas tanto do ponto de vista funcional, com testes automáticos, quanto sob o aspecto da qualidade do código fonte produzido (clareza, eficiência, complexidade, etc.).

É importante destacar que o resultado desse desafio não é o código em si, e sim sua capacidade de explicar e justificar e estruturação do código construido. Portanto o uso de google, stackoverflow e até chat gpt é liberado para esse teste.

A entrega do desafio deve ser feita por meio do github em um repositório publico cujo link deve ser enviado por email para nós antes do prazo final da availiação.

reconcile_accounts

Escreva uma função que faça a conciliação de 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 tipo string:

- Data (em formato yyyy-mm-dd)
- Departamento
- Valor
- Beneficiário

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 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, cada transação de um arquivo deve corresponder uma única outra transaçãodo 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.

last_lines

Escreva uma função, last_lines , que devolva as linhas de um dado arquivo de texto em ordem inversa. Caso esteja familiarizado com a linha de comando Unix, sua função deve fazer o mesmo que o comando tac . Por exemplo, dado o seguinte arquivo, my_file.txt :

```
This is a file
This is line 2
And this is line 3
```

A função last_lines deve funcionar do seguinte modo:

```
>>> for line in last_lines('my_file.txt'):
...     print(line, end='')
And this is line 3
This is line 2
This is a file
```

Note que a função deve incluir o caractere terminador de linha (suponha \n) para cada linha e que sua função deverá devolver um iterator para funcionar da forma correta em funções como ‘for’ e ‘while’. Por exemplo:

```
>>> lines = last_lines('my_file.txt')
>>> next(lines)
'And this is line 3\n'
>>> next(lines)
'This is line 2\n'
>>> next(lines)
'This is a file\n'
```

Dica: na construção do código é possivel que você encontre variaveis associadas ao buffer de leitura, para isso utilize a variavel da lib ‘io’: io.DEFAULT_BUFFER_SIZE e considere que o arquivo está em UTF-8

cached_property

Escreva um *decorator* chamado *cached_property*, análogo ao *decorator property*. O *decorator cached_property* deve aceitar múltiplos atributos dos quais ele depende, e cachear o valor da *property* enquanto o valor desses atributos permanecer inalterado.

O uso do seu docorator deve ser feito da seguinte forma:

```
class Vector:
    def __init__(self, x, y, z, color=None):
        self.x, self.y, self.z = x, y, z
        self.color = color

    @cached_property('x', 'y', 'z')
    def magnitude(self):
        print('computing magnitude')
        return (self.x**2 + self.y**2 + self.z**2)
```

E o output esperado para a classe que usa o *decorator* é o seguinte:

```
>>> v = Vector(9, 2, 6)
>>> v.magnitude
computing magnitude
11.0

>>> v.color = 'red'
>>> v.magnitude
11.0

>>> v.y = 18
>>> v.magnitude
computing magnitude
21.0
```