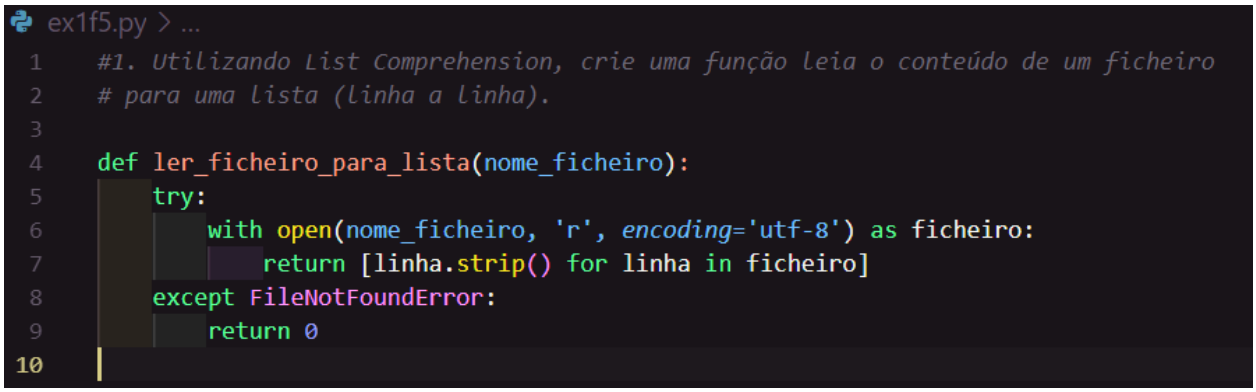
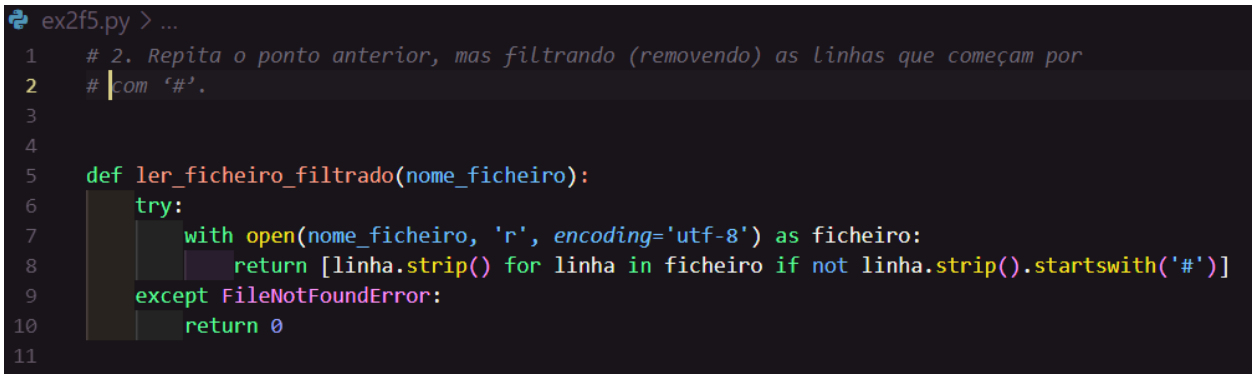
Linguagens de Programação

Resolução da Ficha TP4 - Adriano Neto a22307762



Para o exercício 1, criei uma função que lê o conteúdo de um ficheiro linha a linha e devolve uma lista com essas linhas. Utilizei list comprehension para remover espaços em branco no início e no fim de cada linha. Adicionei também um tratamento de erro para devolver uma lista vazia caso o ficheiro não exista.



No exercício 2, adaptei a função anterior para filtrar as linhas que começam com o carácter #. Este filtro foi implementado diretamente na list comprehension, garantindo que apenas as linhas relevantes são incluídas no resultado.

ex3f5.py > ...

```python
# 3. O Utilizando os geradores (como nas List Comprehension), crie uma função que
# leia de um ficheiro linha a linha e uma outra função que escreva noutro, mas
# filtrando os '#. Nota: o método writelines aceita iteráveis

def ler_ficheiro_como_gerador(nome_ficheiro):
    try:
        with open(nome_ficheiro, 'r', encoding='utf-8') as ficheiro:
            for linha in ficheiro:
                if not linha.strip().startswith('#'):
                    yield linha
    except FileNotFoundError:
        return

def escrever_ficheiro_de_gerador(gerador, nome_ficheiro_saida):
    with open(nome_ficheiro_saida, 'w', encoding='utf-8') as ficheiro:
        ficheiro.writelines(gerador)

```

Para o exercício 3, desenvolvi duas funções: uma que utiliza um gerador para ler um ficheiro linha a linha, ignorando as linhas que começam com #, e outra que escreve as linhas resultantes num novo ficheiro. A função de escrita aproveita o método writelines, que aceita iteradores como entrada.

ex4f5.py > ...

```python
#4. Considere a seguinte classe incompleta. Esta classe já é iterável porque
#implementa __iter__ que retorna um iterador. O objeto iterador é self, ou
#seja, o próprio objeto. Complete a classe de forma a implementar o iterador
#__next__() que retorne iterativamente as potências de 2 até um máximo de
#self.max elementos.


class PowTwo:
    """Classe para implementar um iterador de potências de 2"""

    def __init__(self, max=0):
        self.max = max

    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        if self.n > self.max:
            raise StopIteration
        result = 2 ** self.n
        self.n += 1
        return result

print([l for l in PowTwo(10)])

```

No exercício 4, completei a classe para calcular potências de 2 até um limite máximo especificado. A classe foi definida como iterável, com os métodos necessários para calcular iterativamente as potências de 2 até atingir o número máximo de elementos.

Para o exercício 5, a classe que calcula os termos da sucessão de Fibonacci inicializa os dois primeiros termos da sequência e calcula iterativamente os seguintes, até atingir o número máximo de termos definido.

ex6f5.py > ...

```python
#6. Implemente uma função geradora que devolva infinitamente a sucessão de
#valores de Fibonacci.

def fibonacci_infinito():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

# Teste
gerador = fibonacci_infinito()
for _ in range(10):
    print(next(gerador), end=" ")
```

No exercício 6, criei uma função geradora que devolve infinitamente os valores da sucessão de Fibonacci. A função calcula cada termo da sequência iterativamente e devolve o resultado através de yield.

ex7f5.py > ...

```python
# 7.Desenvolva uma função geradora ou expressão geradora onde que dada uma
#entrada iterável, devolve (filtra) os valores que são ímpares.

def filtrar_impares(iteravel):
    for valor in iteravel:
        if valor % 2 != 0:
            yield valor

entrada = [1, 2, 3, 4, 5, 6, 7, 8, 9]
resultado = (valor for valor in entrada if valor % 2 != 0)
print(list(resultado))
```

No exercício 7, implementei uma função geradora que recebe um iterável e devolve apenas os valores ímpares. Este filtro é aplicado diretamente na função, que devolve os resultados de forma iterativa através de yield.