

Linguagens de Programação

Resolução da Ficha TP4 - Adriano Neto a22307762

```
ex1f5.py > ...
1  #1. Utilizando List Comprehension, crie uma função leia o conteúdo de um ficheiro
2  # para uma lista (linha a linha).
3
4  def ler_ficheiro_para_lista(nome_ficheiro):
5      try:
6          with open(nome_ficheiro, 'r', encoding='utf-8') as ficheiro:
7              return [linha.strip() for linha in ficheiro]
8      except FileNotFoundError:
9          return []
10
```

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.

```
ex2f5.py > ...
1  # 2. Repita o ponto anterior, mas filtrando (removendo) as linhas que começam por
2  # com '#'.
3
4
5  def ler_ficheiro_filtrado(nome_ficheiro):
6      try:
7          with open(nome_ficheiro, 'r', encoding='utf-8') as ficheiro:
8              return [linha.strip() for linha in ficheiro if not linha.strip().startswith('#')]
9      except FileNotFoundError:
10         return []
11
```

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 > ...
1  # 3. O Utilizando os geradores (como nas List Comprehension), crie uma função que
2  # leia de um ficheiro linha a linha e uma outra função que escreva noutro, mas
3  # filtrando os '#'. Nota: o método writelines aceita iteráveis
4
5  def ler_ficheiro_como_gerador(nome_ficheiro):
6      try:
7          with open(nome_ficheiro, 'r', encoding='utf-8') as ficheiro:
8              for linha in ficheiro:
9                  if not linha.strip().startswith('#'):
10                     yield linha
11             except FileNotFoundError:
12                 return
13
14  def escrever_ficheiro_de_gerador(gerador, nome_ficheiro_saida):
15      with open(nome_ficheiro_saida, 'w', encoding='utf-8') as ficheiro:
16          ficheiro.writelines(gerador)
17

```

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 > ...
1  #4. Considere a seguinte classe incompleta. Esta classe já é iterável porque
2  # implementa __iter__ que retorna um iterador. O objeto iterador é self, ou
3  # seja, o próprio objeto. Complete a classe de forma a implementar o iterador
4  # __next__() que retorne iterativamente as potências de 2 até um máximo de
5  # self.max elementos.
6
7
8  class PowTwo:
9      """Classe para implementar um iterador de potências de 2"""
10
11     def __init__(self, max=0):
12         self.max = max
13
14     def __iter__(self):
15         self.n = 0
16         return self
17
18     def __next__(self):
19         if self.n > self.max:
20             raise StopIteration
21         result = 2 ** self.n
22         self.n += 1
23         return result
24
25  print([l for l in PowTwo(10)])
26

```

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.

```
ex5f5.py > ...
1  #5. Na sucessão de Fibonacci o elemento seguinte resulta da soma dos dois
2  #anteriores. Para  $F_0=0$  e  $F_1=1$ , o termo  $F_n = F_{n-1} + F_{n-2}$ .
3  #Desenvolva uma classe que permita a iteração dos N primeiros termos da sucessão
4  #de Fibonacci.
5
6  class Fibonacci:
7      """Classe para iterar sobre os N primeiros termos da sucessão de Fibonacci"""
8      def __init__(self, max=0):
9          self.max = max
10
11     def __iter__(self):
12         self.a, self.b = 0, 1
13         self.n = 0
14         return self
15
16     def __next__(self):
17         if self.n >= self.max:
18             raise StopIteration
19         if self.n == 0:
20             self.n += 1
21             return self.a
22         if self.n == 1:
23             self.n += 1
24             return self.b
25         self.a, self.b = self.b, self.a + self.b
26         self.n += 1
27         return self.a
28
29     print([l for l in Fibonacci(10)])
30
```

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 > ...
1  #6. Implemente uma função geradora que devolva infinitamente a sucessão de
2  #valores de Fibonacci.
3
4  def fibonacci_infinito():
5      a, b = 0, 1
6      while True:
7          yield a
8          a, b = b, a + b
9
10 # Teste
11 gerador = fibonacci_infinito()
12 for _ in range(10):
13     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 > ...
1  # 7.Desenvolva uma função geradora ou expressão geradora onde que dada uma
2  #entrada iterável, devolve (filtra) os valores que são ímpares.
3
4  def filtrar_impares(iteravel):
5      for valor in iteravel:
6          if valor % 2 != 0:
7              yield valor
8
9  entrada = [1, 2, 3, 4, 5, 6, 7, 8, 9]
10 resultado = (valor for valor in entrada if valor % 2 != 0)
11 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.

