



PROGRAMAÇÃO FUNCIONAL

EQUIPE:

Carla Vieira de Araujo Prudencio - 2313681

Cleyson de Oliveira Severiano - 2319296

Eliak Lima - 2416956

Heric Ferreira Maciel - 2223814

Raul Carvalho Teles - 2314373

2024



EAD Unifor



REPOSITÓRIO GITHUB:

<https://github.com/carlaprudencio/programacao-funcional>

Responsabilidades dos Integrantes

Carla Prudencio (Código Fonte)

Cleyson de Oliveira (Código Fonte)

Eliak Lima (Casos de Testes)

Heric Ferreira (Documentação dos Requisitos)

Raul Carvalho Teles (Documentação dos Requisitos)

DOCUMENTO DE REQUISITOS

Filtro de Números Ímpares e Elevação ao Cubo

Requisitos Funcionais:

Requisito 01: O sistema deverá filtrar números ímpares.

Implementação na linha:

```
numeros_impares = [x for x in numeros if eh_impar(x)]
```

Identificação no código:

- O filtro de números ímpares é feito usando list comprehension combinada com a função lambda **eh_impar**. A função lambda é responsável por definir a lógica para verificar se um número é ímpar.

Requisito 02: O sistema deverá fazer a potenciação ao cubo dos números ímpares.

Implementação na linha:

```
eleva_cubo = potenciacao(3)
```

```
numeros_elevados = aplica_func_numeros(eleva_cubo, numeros_impares)
```

Identificação no código:

- O sistema usa uma closure (função **potenciacao**) para gerar uma função que eleva os números ao cubo. Após isso, uma função de alta ordem **aplica_func_numeros** aplica a transformação a cada número ímpar.

Requisito 03: O sistema deverá aplicar uma função a uma lista de númerosImplementação na linha:

numeros_elevados = aplica_func_numeros(eleva_cubo, numeros_impares)

Identificação no código:

- A função **aplica_func_numeros** é uma **função de alta ordem** que recebe outra função como argumento (**eleva_cubo**) e aplica a cada número da lista.

Requisito 04: O sistema deverá testar se os números filtrados e potenciados estão corretos.Implementado nas funções de testes:

def test_numeros_impares():

assert numeros_impares == [1, 3, 5, 7, 9], f"Expected [1, 3, 5, 7, 9], got {numeros_impares}"

def test_numeros_elevados_cubo():

assert numeros_elevados == [1, 27, 125, 343, 729], f"Expected [1, 27, 125, 343, 729], got {numeros_elevados}"

Identificação no código:

- Os testes automatizados garantem que os requisitos estão sendo cumpridos de forma correta.

Requisitos Não Funcionais:

Requisito 01: O código deverá ser legível e organizado.

Identificação no código:

O código segue boas práticas de nomenclatura e modularização:

- Funções são bem nomeadas (`aplica_func_numeros`, `potenciacao`, `test_numeros_impares`, etc.).
- As variáveis descritivas (`numeros_impares`, `numeros_elevados`).
- O código é modular, com funções específicas para cada tarefa.

Requisito 02: O código deverá ser eficiente no processamento de listas.

Identificação no código:

- A list comprehension é usada para filtrar os números de maneira eficiente.
- O uso de lambda e funções de alta ordem garante que o código é funcional e modular, permitindo a reutilização e a redução de código duplicado.

Requisito 03: O código deverá ser testável com casos de teste automatizados

Identificação no código:

- As funções de teste (`test_numeros_impares`, `test_numeros_elevados_cubo`) permitem que o código seja verificado de forma automatizada, garantindo que os resultados estão corretos para a filtragem de números ímpares e a elevação ao cubo deles.

Requisito 04: O código deverá ser modular, permitindo fácil extensão.

Identificação no código:

- A modularidade está presente na separação das funcionalidades em funções como **aplica_func_numeros**, **potenciacao**. E também com os testes, tornando o código fácil de manter e de modificar no futuro, se necessário.

