# Exercícios Python

## Estruturas de dados complexas

## Vetor não ordenado:

1. Escreva uma função em Python chamada **encontrar_duplicatas** que recebe uma lista como entrada e retorna uma lista contendo os elementos duplicados presentes na lista original. Os elementos duplicados devem aparecer apenas uma vez na lista de saída.
2. Escreva uma função em Python chamada **encontrar_elemento_frequente** que recebe uma lista como entrada e retorna o elemento mais frequente presente na lista. Se houver empate na frequência, a função pode retornar qualquer um dos elementos mais frequentes.

## Vetor Ordenado:

1. Escreva uma função em Python chamada **encontrar_par_soma** que recebe uma lista ordenada de números inteiros e um valor alvo. A função deve retornar True se existir um par de elementos na lista cuja soma seja igual ao valor alvo. Caso contrário, a função deve retornar false.
2. Escreva uma função em Python chamada **encontrar_maior_elemento** que recebe uma lista ordenada de números inteiros e retorna o maior elemento presente na lista.

## Pilhas:

1. Explique o conceito de pilha em estruturas de dados e descreva suas principais operações.
2. Implemente uma função em Python chamada **reverter_string** que recebe uma string como entrada e usa uma pilha para retornar a string invertida. Por exemplo, se a entrada for "Python", a função deve retornar "nohtyP".

# Filas:

1. Descreva o conceito de fila em estruturas de dados e explique suas principais operações.
2. Implemente uma classe em Python chamada **FilaDeClientes** que representa uma fila de clientes em um banco. A classe deve ter métodos para adicionar um cliente ao final da fila (**adicionar_cliente**) e remover o próximo cliente da fila (**proximo_cliente**). Certifique-se de tratar os casos em que a fila está vazia.

# Filas de Prioridade:

Suponha que você esteja desenvolvendo um sistema de atendimento em um hospital, onde os pacientes são atendidos com base em sua gravidade. Cada paciente possui um nome e um nível de gravidade, representado por um número inteiro maior ou igual a zero. Escreva uma classe em Python chamada **FilaPrioridade** que represente uma fila de atendimento de pacientes com base em sua gravidade.

A classe **FilaPrioridade** deve ter os seguintes métodos:

- **adicionar_paciente(nome, gravidade)**: adiciona um novo paciente à fila com o nome e a gravidade especificados.
- **proxima_consulta()**: remove e retorna o próximo paciente da fila com a maior gravidade.
- **tamanho()**: retorna o número de pacientes atualmente na fila.
- **vazia()**: verifica se a fila está vazia.

Você deve implementar a classe **FilaPrioridade** de forma que a função **proxima_consulta()** retorne o paciente com a maior gravidade na fila. Em caso de empate de gravidade, o paciente que chegou primeiro deve ser o próximo a ser atendido.

Você pode escolher a estrutura de dados subjacente para implementar a fila de prioridade, como uma lista, uma fila comum ou qualquer outra estrutura adequada.

Implemente a classe **FilaPrioridade** com os métodos descritos acima e teste-a com exemplos de adição de pacientes, chamada da próxima consulta e verificação de tamanho e vazio.

# Deques:

1. Escreva uma função em Python chamada **validar_brackets** que recebe uma string contendo apenas os caracteres '(', ')', '{', '}', '[' e ']' como entrada. A função deve

verificar se a string está balanceada em termos de parênteses, chaves e colchetes. A função deve retornar True se os brackets estiverem corretamente balanceados e false caso contrário.

2. Escreva uma função em Python chamada verificar_palindrome que recebe uma string como entrada e usa um deque para verificar se a string é um palíndromo. A função deve retornar True se a string for um palíndromo e false caso contrário.

# Recursão:

1. Escreva uma função recursiva em Python chamada **calcular_fatorial** que recebe um número inteiro positivo como entrada e retorna o fatorial desse número.
2. Implemente uma função recursiva em Python chamada **calcular_soma_lista** que recebe uma lista de números como entrada e retorna a soma de todos os elementos da lista.

# Lista Encadeada:

Implemente uma lista encadeada simples e uma dupla com os métodos: inserir início, buscar o elemento, remover início, remover posição específica e mostrar a lista

# Algoritmos de ordenação:

Ordene o seguinte código:

```
Import Random

Lista = [1, 2, 3, 4, 5, 6, 7, 8, 9]

lista_embaralhada = lista.copy()

random.shuffle(lista_embaralhada)

print(lista_embaralhada)
```

Ordene o código com os algorítimos: bubble sort, selection sort, insertion sort, shell sort, merge sort e quick sort.