

# GUARDIÕES DA ESTRUTURA

EXPLORANDO AS LINKED LIST EM PYTHON



SILAS SILVA



# Trazendo à Luz

## O Poder Oculto das Linked Lists

As linked lists são estruturas de dados essenciais na programação, oferecendo uma maneira flexível e eficaz de organizar e manipular informações. Neste ebook, vamos mergulhar no universo das linked lists, entendendo o que são e como funcionam de maneira simples e acessível para todos os desenvolvedores.



# 01

## DEFINIÇÃO E UTILIDADE

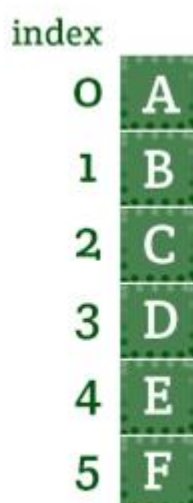
Descubra como os nós são conectados e porque as linked lists são tão poderosas, oferecendo flexibilidade e eficiência na organização e manipulação de dados



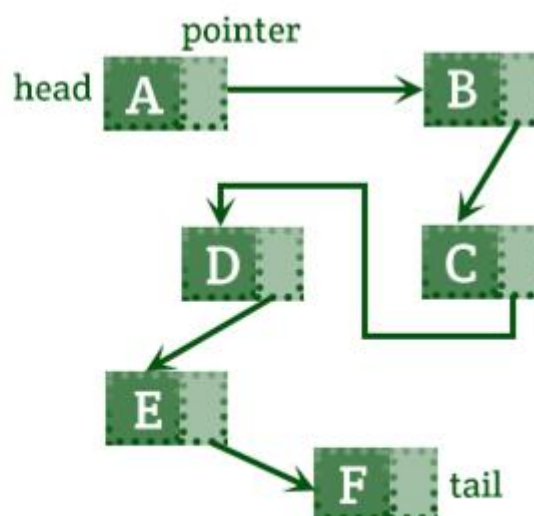
# O que são as Linked Lists?

Uma linked list é uma coleção de elementos chamados de nós (Node), onde cada nó contém um valor (Data) e uma referência (Pointer) para o próximo nó na sequência. O primeiro elemento sempre se chama cabeçalho (Head), e o último elemento vai ser aquele que está apontando para o nulo (Null) ou como no exemplo tail. É como uma corrente, onde cada elo está ligado ao próximo através de um link. Diferentemente dos arrays onde os elementos são armazenados lado a lado na memória, em uma linked list, os nós podem estar dispersos na memória e são conectados por meio de referências.

Arrays



Linked Lists



# 02

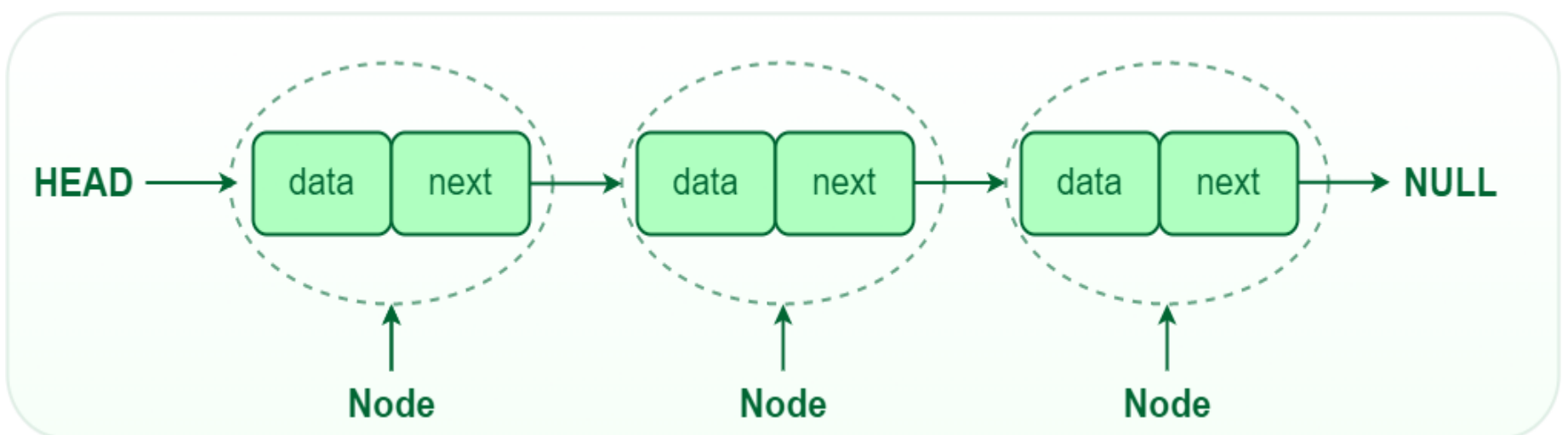
## VANTAGENS



O poder das linked lists: eficiência, flexibilidade e economia de espaço.

# Por que usar as Linked Lists?

As linked lists oferecem algumas vantagens importantes sobre outras estruturas de dados, como arrays. Elas permitem inserções e remoções rápidas em qualquer posição da lista sem a necessidade de mover grandes blocos de dados na memória. Além disso, elas são dinâmicas e podem crescer ou encolher conforme necessário, sem desperdiçar espaço.



# 03

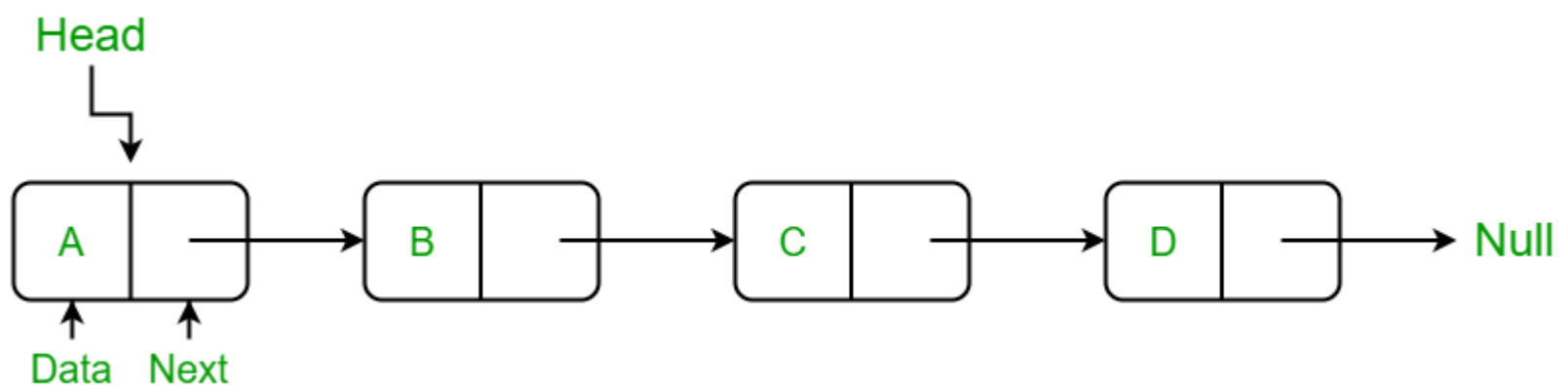
## FUNIONAMENTO



Entenda o funcionamento lógico de cada etapa de  
uma linked list

# Como Funcionam as Linked Lists?

Cada nó em uma linked list possui dois componentes principais: um valor que armazena os dados e um link (ou referência) que aponta para o próximo nó na sequência. O último nó da lista aponta para um valor especial, geralmente NULL, indicando o fim da lista.



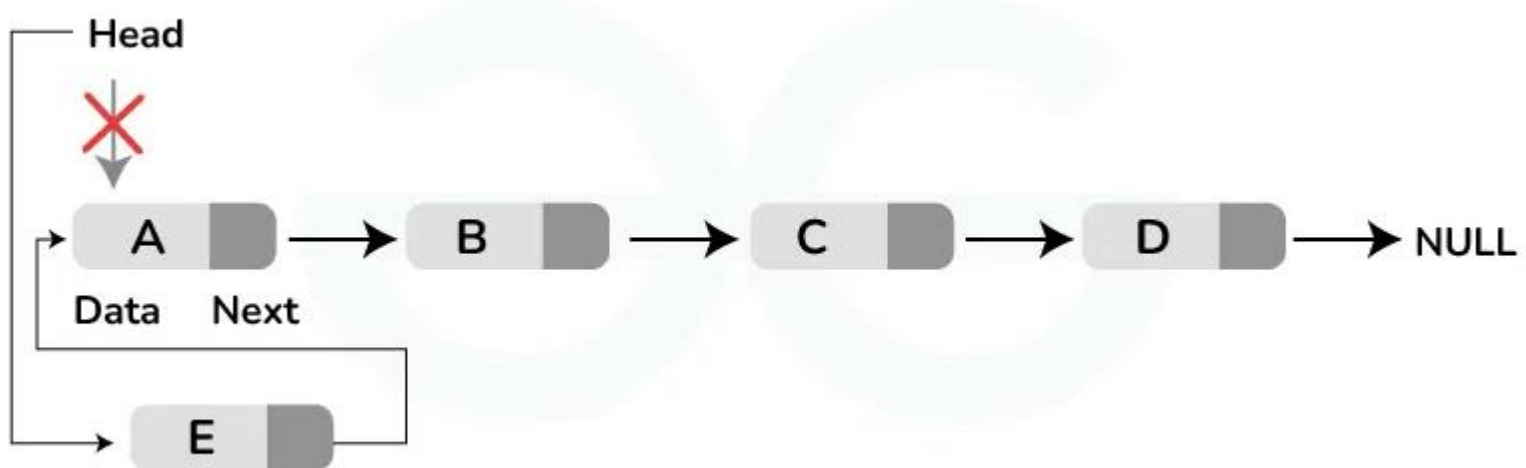


# Adição de Nós

Adicionar um novo Head ou Cabeçalho'

- Crie o nó com seu valor
- Aponte para o antigo Head

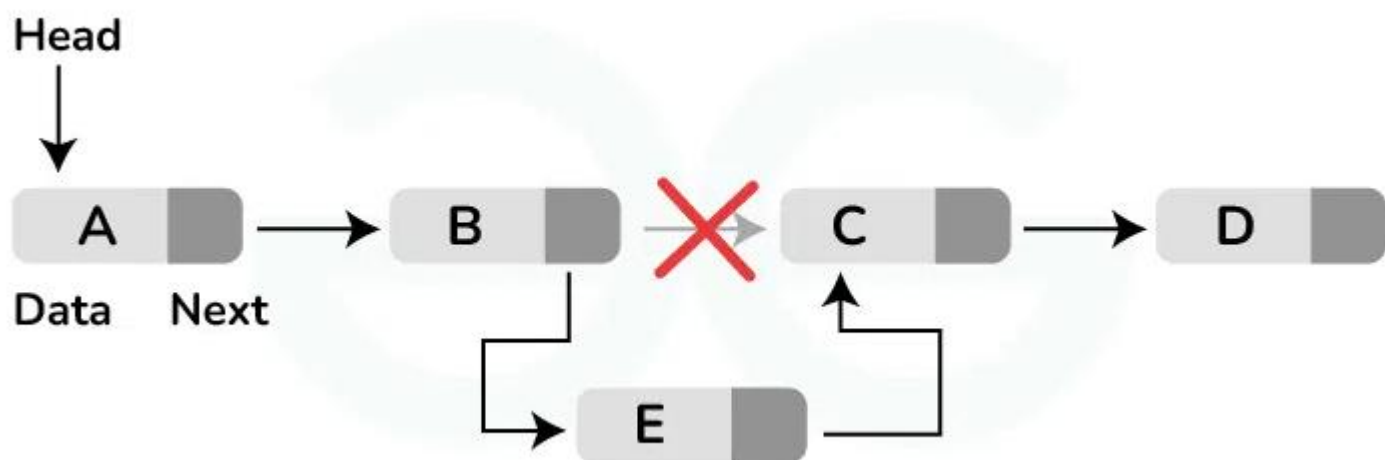
Veja o exemplo abaixo



Adicionar entre dois valores já existentes

- Percorrer a lista até a posição que desejamos
- Crie o nó e aponte o ponteiro para ponto seguinte da inserção
- Aponte o anterior para o novo nó que você inseriu.

Veja o exemplo abaixo.

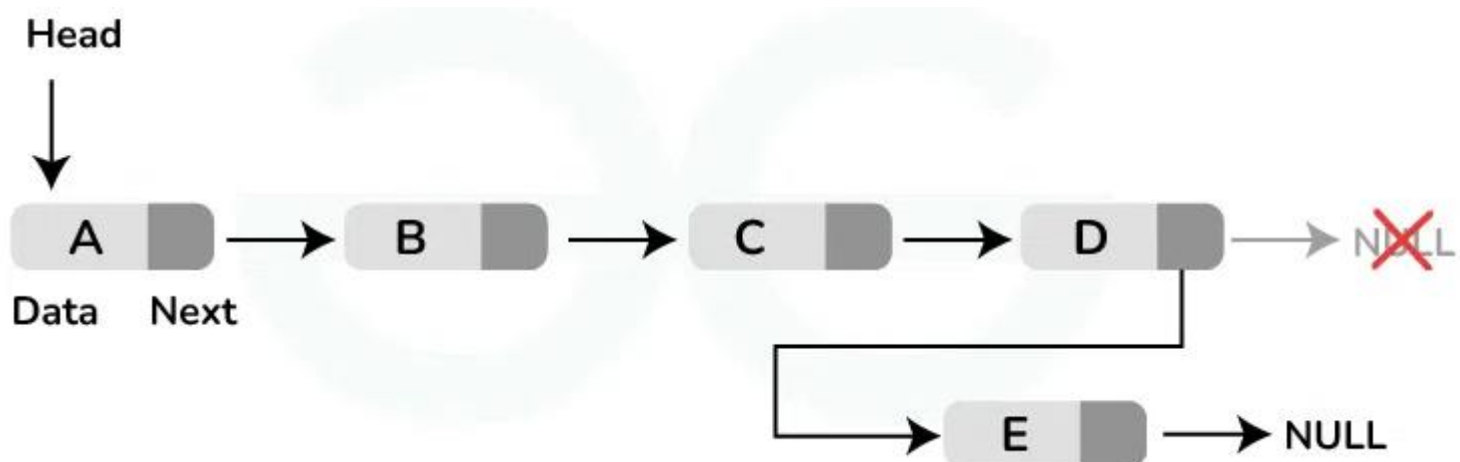


# Adição e Exclusão Nós

Adicionar no final da lista

- Percorrer até o final da lista (onde o nó aponta para o null)
- Criar o novo nó
- Apontar o ponteiro do nó anterior para o novo nó
- Apontar o novo nó para o null para ser o último elemento

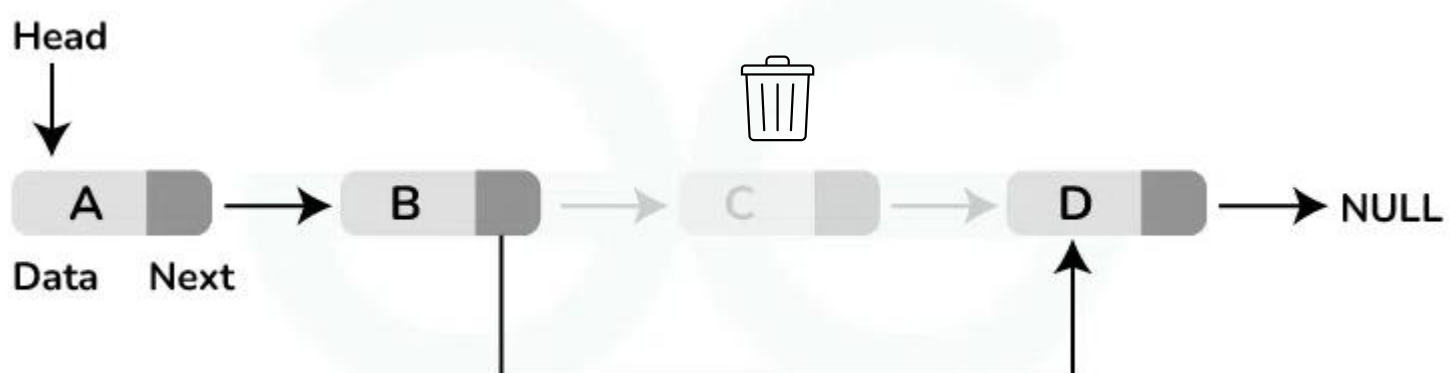
Veja o exemplo abaixo



Excluir um nó

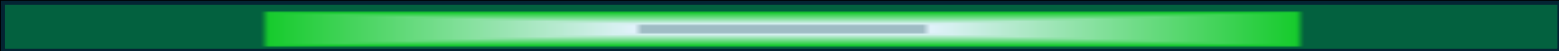
- Percorra a lista até a posição que você deseja excluir
- Aponte o anterior para o próximo nó depois do que você vai excluir
- O Garbage Collector (aplicativo nativo de todo dispositivo que remove o lixo da memória) irá limpar esse nó que não tem nenhum nó apontando para ele.

Veja o exemplo abaixo



# 04

## IMPLEMENTAÇÃO EM PYTHON



Mergulharemos na implementação prática de linked lists, vamos explorar como essa operação é realizada, passo a passo, destacando a lógica por trás dela e sua importância.

# Classes e Método

Começamos definindo duas classes, Node e LinkedList

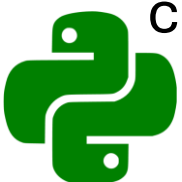
Guardiões da Estrutura - Explorando as Linked List em Python

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
```

Logo após definirmos as classes Node e LinkedList, chamamos um método de construtor de classe, vamos analisar ele passo a passo

- **init**: Método especial de Python, conhecido como método construtor
- **self**: Se refere ao primeiro argumento do método, ou seja, ele mesmo (pode ser nomeado de outras formas, mas é recomendável seguir esse padrão).
- **data**: Onde vai ser armazenado os valores que vamos definir.
- **self.data = data**: Atribuição do valor armazenado em data, ou seja, para inserir um valor ao nó eu vou atribuir o valor ao campo data.
- **self.next = None**: Definimos com None para indicar que inicialmente ele não tem um próximo na lista
- **self.head = None**: Para indicar que, quando uma nova lista é criada, ela está vazia e não tem nenhum nó.



# Método e Condição

Guardiões da Estrutura - Explorando as Linked List em Python

```
def append(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return
    last_node = self.head
    while last_node.next:
        last_node = last_node.next
    last_node.next = new_node
```

- **append:** Método para inserir um novo elemento no final da lista e se ela estiver vazia o novo nó se torna o Head.
- **new\_node = Node(data):** Criamos um novo usando a classe Node e passando o valor de data.
- **if self.head is None:** Condição que verifica se a lista está vazia, ou seja, se head é none, se sim ele irá executar a linha seguinte “**self.head = new\_node**” que atribui o new\_node como novo head, então finaliza com o “**return**”
- **last\_node = self.head:** Se a lista não estiver na condição anterior (de estar vazia), vamos percorrer a lista até o último nó, começando pelo head.
- **while last\_node.next:** Condição de loop que percorre até chegar no ultimo nó que terá o next como none.
- **last\_node = last\_node.next:** Atualizamos o last\_node para apontar pro próximo nó da lista até encontrar o ultimo.
- **last\_node.next = new\_node:** Fora do loop, adicionamos o ultimo nó e definimos que ele é um novo nó conectado no final da lista.





# Método e Condição

```
Guardiões da Estrutura - Explorando as Linked List em Python

def get(self, index):
    current = self.head
    count = 0
    while current:
        if count == index:
            return current.data
        count += 1
        current = current.next
    raise IndexError("Index out of range")
```

- **def get(self, index):** DENTRO DA CLASSE LINKEDLIST definimos o método get o parâmetro **index** que representa a posição do elemento que desejamos obter na lista.
- **current = self.head:** Definimos que o current seja igual ao head (para usar na condição seguinte)
- **count = 0:** Iniciamos um contador para rastrear a posição atual enquanto percorremos a lista, o 0 é a primeira posição
- **Condição while completa:** Enquanto meu current NÃO for None e meu COUNT for igual ao meu INDEX retornaremos o valor armazenado em data. SE NÃO adicione ao count para passar pro próximo nó e atualize o current com current.next para continuar percorrendo.
- **raise IndexError("Index out of range"):** O raise é uma exceção que se chegar ao final do loop sem encontrar a posição desejada, ele vai mostrar essa mensagem de índice fora do intervalo ou seja, inválido na lista)



# Função

Guardiões da Estrutura - Explorando as Linked List em Python

```
def print_values(self):  
    current = self.head  
    while current:  
        print(current.data)  
        current = current.next
```

- **def print\_values(self):** Criamos uma função chamada `print_values` que faz parte da classe `LinkedList` e recebe `self`
  - **current = self.head:** atribuímos a referência do `head` para o `current`, para percorrer a lista desde o primeiro nó.
  - **Estrutura while:** Enquanto houver nós (ou seja, `current` não for `None`), imprima o valor de `data` do nó atual (**print(current.data)**) e atualize o `current` para o próximo nó (**current = current.next**) e repita até `current` ser `None`.
- Essa estrutura irá imprimir todos os valores de todos os nós



# Adicionando itens

Guardiões da Estrutura - Explorando as Linked List em Python

```
linked_list = LinkedList()
linked_list.append(1)
linked_list.append(2)
linked_list.append(3)
linked_list.append(4)
linked_list.append(5)
linked_list.append(6)

linked_list.print_values()
```

**linked\_list = LinkedList():** Criando uma nova variável que armazena uma instância da classe LinkedList

**linked\_list.append(1):** Adiciona o número 1 a lista. E assim sucessivamente com os outros valores

**linked\_list.print\_values():** Irá chamar a função print\_values para imprimir todos os valores dos nós na tela.

A saída será a seguinte:

Guardiões da Estrutura - Explorando as Linked List em Python

```
1
2
3
4
5
6
```



# Estrutura completa

Sendo assim, a estrutura completa ficará dessa forma

Guardiões da Estrutura - Explorando as Linked List em Python

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def get(self, index):
        current = self.head
        count = 0
        while current:
            if count == index:
                return current.data
            count += 1
            current = current.next
        raise IndexError("Index out of range")

    def print_values(self):
        current = self.head
        while current:
            print(current.data)
            current = current.next

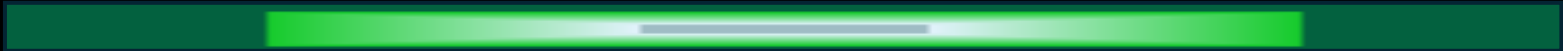
# Criando a lista encadeada
linked_list = LinkedList()
linked_list.append(1)
linked_list.append(2)
linked_list.append(3)
linked_list.append(4)
linked_list.append(5)
linked_list.append(6)

linked_list.print_values()
```



# 05

## EXEMPLOS



Vejam os exemplos de sua aplicação no mundo real e como hoje elas já fazem parte do nosso dia a dia.





# Dia a Dia

No nosso cotidiano temos vários exemplos de uso de Linked List, vejamos alguns deles funcionam.

- **Aplicativos de Mídia Social:** Muitos aplicativos de mídia social, como Facebook, Twitter e LinkedIn, usam linked lists para implementar feeds de notícias ou timelines. Cada postagem ou tweet é um nó na lista, e o feed é construído navegando pela lista a partir do nó mais recente.
- **Filas de Impressão:** Em sistemas operacionais que suportam impressão em rede, as tarefas de impressão são frequentemente gerenciadas usando uma fila. Essa fila pode ser implementada como uma linked list, onde cada tarefa de impressão é um nó na lista e as tarefas são processadas em ordem.
- **Navegadores da Web:** Os navegadores da web usam históricos de navegação para permitir que os usuários retornem a páginas visitadas anteriormente. Esses históricos podem ser implementados usando linked lists, onde cada página visitada é um nó na lista.



# 06

## CONCLUSÃO



Esperamos que esse conteúdo tenha lhe ajudado na  
sua trajetória intergaláctica

# Conclusão

Neste eBook, exploramos os conceitos básicos das linked lists, incluindo sua definição, estrutura e operações principais, como inserção, remoção e busca de elementos.

Aprendemos como as linked lists oferecem flexibilidade e eficiência na manipulação de dados, especialmente em cenários que exigem inserções e remoções frequentes.

Vimos também sua implementação em Python e funcionamento de cada linha de código.

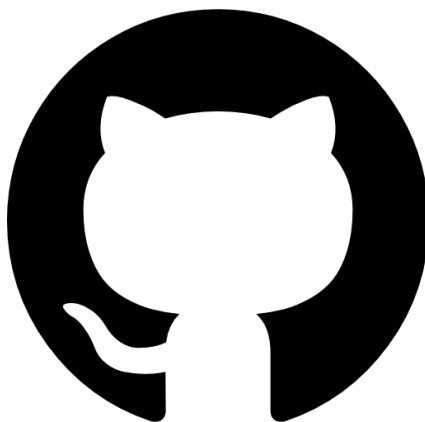
Obrigado por ler e boa sorte em suas aventuras de programação!"



# Agradecimentos

Esse ebook foi construído com auxílio de IA, diagramado e validado por desenvolvedores, confira o projeto completo no meu GitHub

Esse conteúdo foi gerado com fins didáticos porém todos os códigos foram revisados, caso encontre algum erro ou sugestão de melhoria, entre em contato que iremos atualizar/melhorar 😊.



<https://github.com/silaslva/ebook-Guardioes-da-Estrutura>

