

# INSTITUTO FEDERAL

## Maranhão

### Campus Viana

Professor Douglas G Carvalho

#### ✓ Lista encadeada simples

### Representação de um nó



**&e** => Endereço do nó na memória do computador;

**elemento** => O valor do nó que pode ser primitivo ou criado pelo usuário;

**&prox** => Endereço do próximo nó da lista.

Observe que um nó, para uma lista encadeada simples, possui três informações muito relevantes (&e; elemento/valor; &prox).

Um valor armazenado em um nó pode ser de qualquer tipo como os primitivos (int, float, str, etc.) ou criado pelo usuário como Produto, Pessoa, etc.

Observe a representação gráfica de um nó na imagem ao lado.

Na próxima célula criaremos uma classe para representar esse objeto abstrato que, neste caso, trata-se de um nó.

#### ✓ Classe Noh não comentada!

```
1 class Noh:
2     def __init__(self, valor):
3         self.valor = valor
4         self.proximo:Noh = None
5
6     def __str__(self) -> str:
7         return f"{self.valor} -> {self.proximo}"
8
```

#### ✓ Classe Noh comentada!

```
1 # nome da classe
2 class Noh:
3     """
4     Classe Noh: permite instanciar objetos do tipo Noh para
5     uso em uma Lista Encadeada Simples. O valor armazenado
6     pode ser de qualquer tipo. O atributo próximo deve ser
7     uma referência para outro objeto tipo Noh. Exemplo:
8
9     noh = Noh("A")
10
```

```

10     """
11
12     # metodo inicializador de instâncias da classe
13     def __init__(self, valor):
14         # valor a ser armazenado em objeto do tipo Noh
15         self.valor = valor
16         # referência para o próximo Noh dentro de uma lista de Nohs
17         self.proximo:Noh = None
18
19     # como o objeto deverá imprimir a si mesmo
20     def __str__(self) -> str:
21         return f"{self.valor} -> {self.proximo}"
22

```

✓ *Que tal testarmos essa classe Noh criando três objetos dela?*

```

1 noh1 = Noh("A")
2 noh2 = Noh("B")
3 noh3 = Noh("C")
4 noh1.proximo = noh2
5 noh2.proximo = noh3
6
7 print()
8 print(noh1)
9 print(noh2)
10 print(noh3)
11 print()

```



```

A -> B -> C -> None
B -> C -> None
C -> None

```

✓ *Comentando a criação de três objetos do tipo Noh*

```

1 # cria noh um com valor "A"
2 no1 = Noh("A")
3 # cria noh dois com valor "B"
4 noh2 = Noh("B")
5 # cria noh três com valor "C"
6 noh3 = Noh("C")
7 # Configura o próximo noh em no1
8 noh1.proximo = noh2
9 # Configura o próximo noh em no2
10 noh2.proximo = noh3
11
12 # imprimir linha vazia
13 print()
14 # imprimir noh1
15 print(noh1)
16 # imprimir noh2
17 print(noh2)
18 # imprimir noh3
19 print(noh3)
20 # imprimir linha vazia
21 print()

```



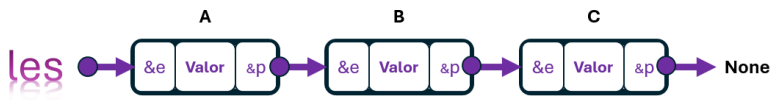
```

A -> B -> C -> None
B -> C -> None
C -> None

```

✓ **Classe Lista**

# Lista Encadeada Simples



**les** => Um objeto do tipo Lista que conhece o primeiro noh ou elemento da lista.

**ABC** => São os valores armazenados em cada noh respectivamente.

**seta** => Representa um atributo que guarda o endereço do próximo nó da lista.

Querido estudante de TDS, saiba que se perdermos a referência do primeiro noh, então, perderemos a lista completamente.

Por outro lado, se conhecemos o primeiro noh, então, seremos capazes de conhecer os demais até o últimos deles.

Observe que a lista encadeada simples não sabe retornar, apenas tem conhecimento do próximo noh e nada sabe do seu antecessor.

Na próxima célula criaremos a classe Lista

que deve sempre conhecer o primeiro noh da lista.

## ✓ Classe Lista não comentada!

```
1 class Lista:
2     def __init__(self, noh: Noh = None) -> None:
3         self.inicio: Noh = noh
4
5     def __str__(self) -> str:
6         return "\n[" + str(self.inicio) + "]\n"
```

## ✓ Classe Lista comentada!

```
1 class Lista:
2     """
3         Classe lista que permite instanciar um objeto de seu tipo vazio,
4         isto é, sem noh ou contendo já o primeiro noh. Exemplos:
5
6         lista_vazia = Lista()
7         lista_com_noh = Lista(noh1)
8     """
9     # Método inicializador que aceita como argumento um noh ou valor padrão None
10    def __init__(self, noh: Noh = None) -> None:
11        # Seta o atributo inicio com o argumento passado pelo usuário
12        self.inicio: Noh = noh
13
14    # como o objeto deverá imprimir a si mesmo
15    def __str__(self) -> str:
16        return "\n[" + str(self.inicio) + "]\n"
```

## ✓ Que tal testarmos essa classe Lista?


```
1 noh1 = Noh("A")
2 les = Lista(noh1)
3 print(les)
```

 [A -> None]

## ✓ Teste comentado


```
1 # Instanciar um objeto da classe Noh com valor de argumento "A"
2 noh1 = Noh("A")
3 # Instanciar um objeto da classe Lista com valor de argumento objeto noh
4 les = Lista(noh1)
```

```
5 # Imprimir a lista
6 print(les)
```

 [A -> None]


- ✓ Que tal criarmos mais dois nohs para referenciá-los manualmente e no fim imprimir o objeto les (lista) de novo?

```
1 noh2 = Noh("B")
2 noh3 = Noh("C")
3 noh1.proximo = noh2
4 noh2.proximo = noh3
5 print(les)
```

 [A -> B -> C -> None]

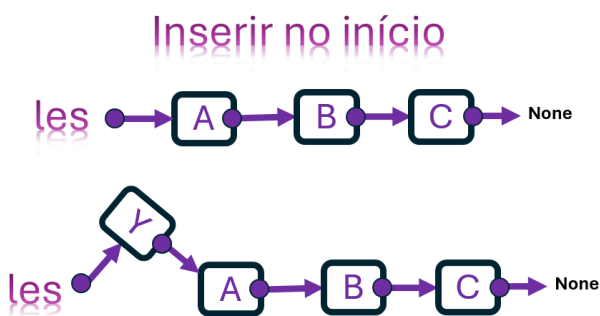
- ✓ *Teste comentado*

```
1 # Criar segundo objeto Noh
2 noh2 = Noh("B")
3 # Criar terceiro objeto Noh
4 noh3 = Noh("C")
5 # Referenciar o próximo de noh1 com noh2
6 noh1.proximo = noh2
7 # Referenciar o próximo de noh2 com noh3
8 noh2.proximo = noh3
9 # Imprimir a lista
10 print(les)
```

 [A -> B -> C -> None]

## Operações da Lista Encadeada Simples (LES)

- ✓ Adicionar no início



Adicionar o noh no início da lista implica torná-lo primeiro noh da lista e no caso em que a lista não se encontra vazia deve se tornar em segundo noh aquele que antes era o primeiro noh.

Na próxima célula, programaremos um método para a classe listar inserir um noh em seu início.

- ✓ *Método para inserir no início*

```
1 def inserir_inicio(self, valor):
2     novo_noh = Noh(valor)
3     novo_noh.proximo = self.inicio
4     self.inicio = novo_noh
5
```

- ✓ *Método comentado*

```

1 def inserir_inicio(self, valor):
2     """
3     Parâmetro valor: indica receber, como argumento, um valor de
4     qualquer tipo para que seja adicionado na instância de um novo noh.
5     Exemplo:
6
7     lista.inserir_inicio('Y')
8     """
9     # Instanciar novo noh com argumento valor
10    novo_noh = Noh(valor) # Criar novo noh
11    novo_noh.proximo = self.inicio
12    # O novo noh se torna o primeiro noh
13    self.inicio = novo_noh # novo no inicio
14

```

▼ *Atualizar a classe Lista com o método **inserir\_inicio()***

```

1 class Lista:
2     """
3     Classe lista que permite instanciar um objeto de seu tipo vazio,
4     isto é, sem noh ou contendo já o primeiro noh. Exemplos:
5
6     lista_vazia = Lista()
7     lista_com_noh = Lista(noh1)
8     """
9     # Método inicializador que aceita como argumento um noh ou valor padrão None
10    def __init__(self, noh: Noh = None) -> None:
11        # Seta o atributo inicio com o argumento passado pelo usuário
12        self.inicio: Noh = noh
13
14
15    # como o objeto deverá imprimir a si mesmo
16    def __str__(self) -> str:
17        return "\n[" + str(self.inicio) + "]\n"
18
19
20    def inserir_inicio(self, valor):
21        """
22        Parâmetro valor: indica receber, como argumento, um valor de
23        qualquer tipo para que seja adicionado na instância de um novo noh.
24        Exemplo:
25
26        lista.inserir_inicio('Y')
27        """
28        # Instanciar novo noh com argumento valor
29        novo_noh = Noh(valor) # Criar novo noh
30        novo_noh.proximo = self.inicio
31        # O novo noh se torna o primeiro noh
32        self.inicio = novo_noh # novo no inicio
33

```

▼ *Que tal testarmos o novo poder de nossa classe Lista?*

01. Iremos começar criando três objetos da classe Noh.
02. Em seguida, um objeto da classe Lista para que possamos adicionar os três nohs já criados.
03. Por fim, usaremos o método **inserir\_inicio()** para inserção/adição de mais um noh e dessa vez no início da lista.

```

1 noh1 = Noh("A")
2 noh2 = Noh("B")
3 noh1.proximo = noh2

```

```

4 noh3 = Noh("C")
5 noh2.proximo = noh3
6
7 lista = Lista(noh1)
8 print(lista)
9
10 lista.inserir_inicio("Y")
11 print(lista)

```



```
[A -> B -> C -> None]
```

```
[Y -> A -> B -> C -> None]
```

#### ▼ Código comentado

```

1 # Criar noh1
2 noh1 = Noh("A")
3 # Criar noh2
4 noh2 = Noh("B")
5 # Setar o próximo de noh1 para noh2
6 noh1.proximo = noh2
7 # Criar noh3
8 noh3 = Noh("C")
9 # Setar o próximo de noh2 para noh3
10 noh2.proximo = noh3
11
12 # Instanciar uma lista cujo início é o noh1
13 lista = Lista(noh1)
14 # imprimir a lista
15 print(lista)
16
17 # Invocar método para inserir no início da lista
18 lista.inserir_inicio("W")
19 # imprimir a lista alterada
20 print(lista)

```



```
[A -> B -> C -> None]
```

```
[W -> A -> B -> C -> None]
```

## Agora turma de TDS, chegou a hora de vocês protagonizarem!

As equipes devem desenvolver os métodos das operações de LES (Lista Encadeada Simples) que estão discriminadas abaixo.

#### ▼ Tamanho da lista

```

1 # Resolução
2 def tamanho(self) -> int:
3     tamanho = 0
4     if self.inicio is not None:
5         tamanho += 1
6         noh_atual = self.inicio
7         while(noh_atual.proximo is not None):
8             tamanho += 1
9             noh_atual = noh_atual.proximo
10    return tamanho
11    else:
12    return tamanho

```

✓ *Código comentado*

```
1 # Resolução
2 def tamanho(self) -> int:
3     """
4         Método de instância da classe Lista. Exemplo de uso:
5
6         objeto_lista = Lista("A")
7         objeto_lista.tamanho()
8
9         Resultado: 1
10    """
11    # Iniciar variável tamanho com zero
12    tamanho = 0
13    # Se a lista não estiver vazia
14    if self.inicio is not None:
15        # Adicione 1 a tamanho
16        tamanho += 1
17        # Variável auxiliar para percorrer a lista
18        noh_atual = self.inicio
19        # Enquanto o próximo não é nulo
20        while(noh_atual.proximo is not None):
21            # Adicione 1 a tamanho
22            tamanho += 1
23            # Variável auxiliar se torna o próximo noh
24            noh_atual = noh_atual.proximo
25        # Retorna a contagem de nohs
26        return tamanho
27    else:
28        # Se a lista estiver vazia retorna tamanho zero
29        return tamanho
```

✓ *Atualizar a classe Lista com o método **tamanho()***

```
1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "\n[" + str(self.inicio) + "]\n"
9
10
11     def inserir_inicio(self, valor):
12         novo_noh = Noh(valor)
13         novo_noh.proximo = self.inicio
14         self.inicio = novo_noh
15
16
17     def tamanho(self) -> int:
18         tamanho = 0
19         if self.inicio is not None:
20             tamanho += 1
21             noh_atual = self.inicio
22             while(noh_atual.proximo is not None):
23                 tamanho += 1
24                 noh_atual = noh_atual.proximo
25             return tamanho
26         else:
27             return tamanho
28
```

### ✓ Que tal testarmos o novo poder de nossa classe Lista?

01. Criaremos um objeto da classe Lista para que possamos adicionar três nós com o método **inserir\_inicio()** que foi desenvolvido anteriormente e adicionado na classe Lista.
02. Em seguida, invocaremos o método **tamanho()**.
03. Então, adicionaremos mais um nó.
04. Por fim, invocaremos o método **tamanho()** novamente.

```
1 noh = Noh("A")
2 lista = Lista(noh)
3 lista.inserir_inicio("B")
4 lista.inserir_inicio("C")
5
6 print(lista)
7 print(f"Tamanho da lista = {lista.tamanho()}")
8
9 lista.inserir_inicio("Y")
10 print(lista)
11 print(f"Tamanho da lista = {lista.tamanho()}")
```



```
[C -> B -> A -> None]

Tamanho da lista = 3

[Y -> C -> B -> A -> None]

Tamanho da lista = 4
```

### ✓ Adicionar no meio

```
1 def inserir_meio(self, posicao: int, valor) -> None:
2     if posicao <= self.tamanho() + 1:
3         if (posicao != 0) and (self.inicio is not None):
4             posicao_atual = 0
5             novo_noh = Noh(valor)
6             noh_atual = self.inicio
7             while(noh_atual.proximo is not None):
8                 posicao_atual += 1
9                 if posicao == posicao_atual:
10                     novo_noh.proximo = noh_atual.proximo
11                     noh_atual.proximo = novo_noh
12                     return None
13                     noh_atual = noh_atual.proximo
14                     noh_atual.proximo = novo_noh
15             else:
16                 self.inserir_inicio(valor)
17         else:
18             print("Posição inválida! É maior que a lista + 1")
```

### ✓ Código comentado

```
1 # Resolução
2 def inserir_meio(self, posicao: int, valor) -> None:
3     """
4         Método que permite inserir em posição que não seja somente
5         inicial ou final de uma lista. Exemplo de uso:
6
7         lista -> ["A","B","C","D"]
8         lista.inserir_meio(2, "H")
9     """
```



```

10     Resultado: ["A","B","H","C","D"]
11     ""
12     # Posição deverá ser no máximo uma a mais que o tamanho da lista
13     if posicao <= self.tamanho() + 1:
14         # Se lista não estiver vazia ou se a posição é zero
15         if (posicao != 0) and (self.inicio is not None):
16             # Acompanhar padrão da linguagem: iniciar no zero
17             posicao_atual = 0
18             # Instanciar um noh com o valor do usuário
19             novo_noh = Noh(valor)
20             # Noh atual inicia na posição zero
21             noh_atual = self.inicio
22             # Enquanto houver um próximo noh
23             while(noh_atual.proximo is not None):
24                 # print(f"noh_atual.valor = {noh_atual.valor}")
25                 # print(f"noh_atual.proximo is not None = {noh_atual.proximo is not None}")
26                 # Incrementar +1 em posição
27                 posicao_atual += 1
28                 # Se baterem posições
29                 if posicao == posicao_atual:
30                     # Novo noh aponta o próximo de atual
31                     novo_noh.proximo = noh_atual.proximo
32                     # Atual aponto para novo noh
33                     noh_atual.proximo = novo_noh
34                     return None
35                     noh_atual = noh_atual.proximo
36             # Caso a chamada deste método não tenha sido interrompida pelo return
37             # Atual aponto para novo noh
38             noh_atual.proximo = novo_noh
39         else:
40             # Valores negativos para a posição resultam em inserção no início
41             self.inserir_inicio(valor)
42     else:
43         print("Posição inválida! É maior que a lista + 1")

```

▼ *Atualizar a classe Lista com o método **inserir\_meio()***

```

1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "[" + str(self.inicio) + "]\n"
9
10
11     def inserir_inicio(self, valor):
12         novo_noh = Noh(valor)
13         novo_noh.proximo = self.inicio
14         self.inicio = novo_noh
15
16
17     def tamanho(self) -> int:
18         tamanho = 0
19         if self.inicio is not None:
20             tamanho += 1
21             noh_atual = self.inicio
22             while(noh_atual.proximo is not None):
23                 tamanho += 1
24                 noh_atual = noh_atual.proximo
25             return tamanho
26         else:
27             return tamanho

```

```

28
29
30 def inserir_meio(self, posicao: int, valor) -> None:
31     if posicao <= self.tamanho() + 1:
32         if (posicao != 0) and (self.inicio is not None):
33             posicao_atual = 0
34             novo_noh = Noh(valor)
35             noh_atual = self.inicio
36             while(noh_atual.proximo is not None):
37                 posicao_atual += 1
38                 if posicao == posicao_atual:
39                     novo_noh.proximo = noh_atual.proximo
40                     noh_atual.proximo = novo_noh
41                     return None
42                     noh_atual = noh_atual.proximo
43             noh_atual.proximo = novo_noh
44         else:
45             self.inserir_inicio(valor)
46     else:
47         print("Posição inválida! É maior que a lista + 1")
48

```

#### ✓ Que tal testarmos o novo poder de nossa classe Lista?

01. Criaremos um objeto da classe Lista para que possamos adicionar três nohs com o método **inserir\_inicio()** que foi desenvolvido anteriormente e adicionado na classe Lista.
02. Em seguida, invocaremos o método **inserir\_meio()**.
03. Por fim, iremos imprimir a lista.

```

1 noh = Noh("C")
2 lista = Lista(noh)
3 lista.inserir_inicio("B")
4 lista.inserir_inicio("A")
5 print(lista)
6
7 lista.inserir_meio(2, "Y")
8 print(lista)

```

```

➦ [A -> B -> C -> None]
[A -> B -> Y -> C -> None]

```

#### ✓ Adicionar no final

```

1 # Resolução
2 def inserir_final(self, valor):
3     if self.inicio is not None:
4         noh_atual = self.inicio
5         while(noh_atual.proximo is not None):
6             noh_atual = noh_atual.proximo
7         noh_atual.proximo = Noh(valor)
8     else:
9         self.inserir_inicio(valor)
10

```

#### ✓ Código comentado

```

1 # Resolução
2 def inserir_final(self, valor):
3     """
4     Método que permite inserir no final independentemente do
5     tamanho da lista. Exemplo de uso:

```

```

5 tamanho da lista. Exemplo de uso.
6
7 lista -> ["A","B","C","D"]
8 lista.inserir_final("W")
9
10 Resultado: ["A","B","C","D","W"]
11 """
12 # Se existe início da lista
13 if self.inicio is not None:
14     noh_atual = self.inicio
15     # Percorrer todos nohs até encontrar o último noh
16     while(noh_atual.proximo is not None):
17         noh_atual = noh_atual.proximo
18     # Último noh encontrado aponta para o novo noh final
19     noh_atual.proximo = Noh(valor)
20 else:
21     # Se lista não existir, insere no início
22     self.inserir_inicio(valor)
23

```

▼ Atualizar a classe *Lista* com o método ***inserir\_final()***

```

1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "["+ str(self.inicio) +"]\n"
9
10
11     def inserir_inicio(self, valor):
12         novo_noh = Noh(valor)
13         novo_noh.proximo = self.inicio
14         self.inicio = novo_noh
15
16
17     def tamanho(self) -> int:
18         tamanho = 0
19         if self.inicio is not None:
20             tamanho += 1
21             noh_atual = self.inicio
22             while(noh_atual.proximo is not None):
23                 tamanho += 1
24                 noh_atual = noh_atual.proximo
25             return tamanho
26         else:
27             return tamanho
28
29
30     def inserir_meio(self, posicao: int, valor) -> None:
31         if posicao <= self.tamanho() + 1:
32             if (posicao != 0) and (self.inicio is not None):
33                 posicao_atual = 0
34                 novo_noh = Noh(valor)
35                 noh_atual = self.inicio
36                 while(noh_atual.proximo is not None):
37                     posicao_atual += 1
38                     if posicao == posicao_atual:
39                         novo_noh.proximo = noh_atual.proximo
40                         noh_atual.proximo = novo_noh
41                         return None
42                 noh_atual = noh_atual.proximo
43                 noh_atual.proximo = novo_noh

```

```

44         else:
45             self.inserir_inicio(valor)
46     else:
47         print("Posição inválida! É maior que a lista + 1")
48
49
50     def inserir_final(self, valor):
51         if self.inicio is not None:
52             noh_atual = self.inicio
53             while(noh_atual.proximo is not None):
54                 noh_atual = noh_atual.proximo
55             noh_atual.proximo = Noh(valor)
56         else:
57             self.inserir_inicio(valor)
58

```

#### ✓ Que tal testarmos o novo poder de nossa classe Lista?

01. Criaremos um objeto da classe Lista para que possamos adicionar três nohs com o método **inserir\_inicio()** que foi desenvolvido anteriormente e adicionado na classe Lista.
02. Em seguida, imprimir a lista.
03. Então, invocaremos o método **inserir\_final()**.
04. Por fim, imprimir a lista novamente.

```

1 noh = Noh("A")
2 lista = Lista(noh)
3 lista.inserir_inicio("B")
4 lista.inserir_inicio("C")
5 print(lista)
6
7 lista.inserir_final("Y")
8 print(lista)
9

```

```

↔ [C -> B -> A -> None]
[C -> B -> A -> Y -> None]

```

#### ✓ Remover no início

```

1 # Resolução
2 def remover_inicio(self):
3     if self.inicio is not None:
4         self.inicio = self.inicio.proximo
5

```

#### ✓ Código comentado

```

1 # Resolução
2 def remover_inicio(self):
3     """
4         Método que remove o primeiro noh. Exemplo de uso:
5
6         lista -> ["A","B","C","D"]
7         lista.remover_inicio()
8
9         Resultado: ["B","C","D"]
10    """
11     if self.inicio is not None:
12         self.inicio = self.inicio.proximo
13

```

✓ Atualizar a classe Lista com o método \*remover\_inicio()\*\*\*

```
1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "["+ str(self.inicio) +"]\n"
9
10
11     def inserir_inicio(self, valor):
12         novo_noh = Noh(valor)
13         novo_noh.proximo = self.inicio
14         self.inicio = novo_noh
15
16
17     def tamanho(self) -> int:
18         tamanho = 0
19         if self.inicio is not None:
20             tamanho += 1
21             noh_atual = self.inicio
22             while(noh_atual.proximo is not None):
23                 tamanho += 1
24                 noh_atual = noh_atual.proximo
25             return tamanho
26         else:
27             return tamanho
28
29
30     def inserir_meio(self, posicao: int, valor) -> None:
31         if posicao <= self.tamanho() + 1:
32             if (posicao != 0) and (self.inicio is not None):
33                 posicao_atual = 0
34                 novo_noh = Noh(valor)
35                 noh_atual = self.inicio
36                 while(noh_atual.proximo is not None):
37                     posicao_atual += 1
38                     if posicao == posicao_atual:
39                         novo_noh.proximo = noh_atual.proximo
40                         noh_atual.proximo = novo_noh
41                         return None
42                     noh_atual = noh_atual.proximo
43                 noh_atual.proximo = novo_noh
44             else:
45                 self.inserir_inicio(valor)
46         else:
47             print("Posição inválida! É maior que a lista + 1")
48
49
50     def inserir_final(self, valor):
51         if self.inicio is not None:
52             noh_atual = self.inicio
53             while(noh_atual.proximo is not None):
54                 noh_atual = noh_atual.proximo
55             noh_atual.proximo = Noh(valor)
56         else:
57             self.inserir_inicio(valor)
58
59
60     def remover_inicio(self):
61         if self.inicio is not None:
```

### ✓ Que tal testarmos o novo poder de nossa classe Lista?

01. Criaremos um objeto da classe Lista para que possamos adicionar três nohs com o método **inserir\_inicio()**, **inserir\_meio()** e **inserir\_final()** que foi desenvolvido anteriormente e adicionado na classe Lista.
02. Em seguida, imprimir a lista.
03. Então, invocaremos o método **remover\_inicio()**.
04. Por fim, imprimir a lista novamente.

```
1 lista = Lista()
2 lista.inserir_final("C")
3 lista.inserir_inicio("A")
4 lista.inserir_meio(1, "B")
5 print(lista)
6
7 lista.remover_inicio()
8 print(lista)
9
```

```
↔ [A -> B -> C -> None]
[B -> C -> None]
```

### ✓ Remover no meio

```
1 def remover_meio(self, posicao: int) -> None:
2     if self.inicio is not None:
3         if posicao <= self.tamanho():
4             if not (posicao <= 0):
5                 posicao_atual = 0
6                 noh_atual = self.inicio
7                 while(noh_atual.proximo is not None):
8                     posicao_atual += 1
9                     if posicao == posicao_atual:
10                        noh_atual.proximo = (noh_atual.proximo).proximo
11                        return None
12                        noh_atual = noh_atual.proximo
13             else:
14                 self.remover_inicio()
15         else:
16             print("Posição inválida! É maior que a lista + 1")
17     else:
18         print("Lista Vazia")
```

### ✓ Código comentado

```
1 # Resolução
2 def remover_meio(self, posicao: int) -> None:
3     """
4         Método que remove um noh em uma posição fornecida
5         pelo usuário. Exemplo de uso:
6
7         lista -> ["A","B","C","D"]
8         lista.remover_meio(2)
9
10        Resultado: ["A","B","D"]
11    """
12    if self.inicio is not None:
13        if posicao <= self.tamanho():
```

```

14         if not (posicao <= 0):
15             posicao_atual = 0
16             noh_atual = self.inicio
17             while(noh_atual.proximo is not None):
18                 posicao_atual += 1
19                 if posicao == posicao_atual:
20                     noh_atual.proximo = (noh_atual.proximo).proximo
21                     return None
22                     noh_atual = noh_atual.proximo
23             else:
24                 self.remover_inicio()
25         else:
26             print("Posição inválida! É maior que a lista + 1")
27     else:
28         print("Lista Vazia")

```

▼ Atualizar a classe *Lista* com o método ***remover\_meio()***

```

1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "["+ str(self.inicio) +"]\n"
9
10
11     def inserir_inicio(self, valor):
12         novo_noh = Noh(valor)
13         novo_noh.proximo = self.inicio
14         self.inicio = novo_noh
15
16
17     def tamanho(self) -> int:
18         tamanho = 0
19         if self.inicio is not None:
20             tamanho += 1
21             noh_atual = self.inicio
22             while(noh_atual.proximo is not None):
23                 tamanho += 1
24                 noh_atual = noh_atual.proximo
25             return tamanho
26         else:
27             return tamanho
28
29
30     def inserir_meio(self, posicao: int, valor) -> None:
31         if posicao <= self.tamanho() + 1:
32             if (posicao != 0) and (self.inicio is not None):
33                 posicao_atual = 0
34                 novo_noh = Noh(valor)
35                 noh_atual = self.inicio
36                 while(noh_atual.proximo is not None):
37                     posicao_atual += 1
38                     if posicao == posicao_atual:
39                         novo_noh.proximo = noh_atual.proximo
40                         noh_atual.proximo = novo_noh
41                         return None
42                     noh_atual = noh_atual.proximo
43                 noh_atual.proximo = novo_noh
44             else:
45                 self.inserir_inicio(valor)
46         else:
47             print("Posição inválida! É maior que a lista + 1")

```

```

48
49
50 def inserir_final(self, valor):
51     if self.inicio is not None:
52         noh_atual = self.inicio
53         while(noh_atual.proximo is not None):
54             noh_atual = noh_atual.proximo
55         noh_atual.proximo = Noh(valor)
56     else:
57         self.inserir_inicio(valor)
58
59
60 def remover_inicio(self):
61     if self.inicio is not None:
62         self.inicio = self.inicio.proximo
63
64
65 def remover_meio(self, posicao: int) -> None:
66     if self.inicio is not None:
67         if posicao <= self.tamanho() - 1:
68             if not (posicao <= 0):
69                 posicao_atual = 0
70                 noh_atual = self.inicio
71                 while(noh_atual.proximo is not None):
72                     posicao_atual += 1
73                     if posicao == posicao_atual:
74                         noh_atual.proximo = (noh_atual.proximo).proximo
75                         return None
76                         noh_atual = noh_atual.proximo
77             else:
78                 self.remover_inicio()
79         else:
80             print("Posição inválida! É maior que a lista.")
81     else:
82         print("Lista Vazia")

```


#### ✓ Que tal testarmos o novo poder de nossa classe Lista?

01. Criaremos um objeto da classe Lista para que possamos adicionar três nohs com o método **inserir\_inicio()**, **inserir\_meio()** e **inserir\_final()** que foi desenvolvido anteriormente e adicionado na classe Lista.
02. Em seguida, imprimir a lista.
03. Então, invocaremos o método **remover\_meio()**.
04. Por fim, imprimir a lista novamente.

```

1 lista = Lista()
2 lista.inserir_final("C")
3 lista.inserir_inicio("A")
4 lista.inserir_meio(1, "B")
5 print(lista)
6
7 lista.remover_meio(2)
8 print(lista)
9

```

 [A -> B -> C -> None]

[A -> B -> None]

#### ✓ Remover no final

```

1 def remover_final(self) -> None:
2     if self.inicio is not None:
3         if self.inicio.proximo is not None:

```



```

4         noh_atual = self.inicio
5         while (noh_atual.proximo).proximo is not None:
6             noh_atual = noh_atual.proximo
7             noh_atual.proximo = None
8         else:
9             self.remover_inicio()
10    else:
11        print("Lista Vazia")

```

#### ✓ Código comentado

```

1 def remover_final(self) -> None:
2     """
3     Método que remove um noh que estiver no final da lista.
4     Exemplo de uso:
5
6     lista -> ["A","B","C","D"]
7     lista.remover()
8
9     Resultado: ["A","B","C"]
10    """
11    # Se a lista não estiver vazia
12    if self.inicio is not None:
13        # Se existir um segundo noh
14        if self.inicio.proximo is not None:
15            noh_atual = self.inicio
16            # Enquanto existir um próximo noh
17            while (noh_atual.proximo).proximo is not None:
18                # Noh atual se torna o próximo noh
19                noh_atual = noh_atual.proximo
20            # Ao final do loop, o penúltimo noh perde a
21            # referência para o último noh ocorrendo a exclusão
22            noh_atual.proximo = None
23        else:
24            # Caso contrário, remover o primeiro noh
25            self.remover_inicio()
26    else:
27        # Caso contrário, comunicar lista vazia
28        print("Lista Vazia")

```

#### ✓ Atualizar a classe Lista com o método **remover\_final()**

```

1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "["+ str(self.inicio) +"]\n"
9
10
11    def inserir_inicio(self, valor):
12        novo_noh = Noh(valor)
13        novo_noh.proximo = self.inicio
14        self.inicio = novo_noh
15
16
17    def tamanho(self) -> int:
18        tamanho = 0
19        if self.inicio is not None:
20            tamanho += 1

```

```

21         noh_atual = self.inicio
22         while(noh_atual.proximo is not None):
23             tamanho += 1
24             noh_atual = noh_atual.proximo
25         return tamanho
26     else:
27         return tamanho
28
29
30 def inserir_meio(self, posicao: int, valor) -> None:
31     if posicao <= self.tamanho() + 1:
32         if (posicao != 0) and (self.inicio is not None):
33             posicao_atual = 0
34             novo_noh = Noh(valor)
35             noh_atual = self.inicio
36             while(noh_atual.proximo is not None):
37                 posicao_atual += 1
38                 if posicao == posicao_atual:
39                     novo_noh.proximo = noh_atual.proximo
40                     noh_atual.proximo = novo_noh
41                     return None
42                     noh_atual = noh_atual.proximo
43             noh_atual.proximo = novo_noh
44         else:
45             self.inserir_inicio(valor)
46     else:
47         print("Posição inválida! É maior que a lista + 1")
48
49
50 def inserir_final(self, valor):
51     if self.inicio is not None:
52         noh_atual = self.inicio
53         while(noh_atual.proximo is not None):
54             noh_atual = noh_atual.proximo
55         noh_atual.proximo = Noh(valor)
56     else:
57         self.inserir_inicio(valor)
58
59
60 def remover_inicio(self):
61     if self.inicio is not None:
62         self.inicio = self.inicio.proximo
63
64
65 def remover_meio(self, posicao: int) -> None:
66     if self.inicio is not None:
67         if posicao <= self.tamanho() - 1:
68             if not (posicao <= 0):
69                 posicao_atual = 0
70                 noh_atual = self.inicio
71                 while(noh_atual.proximo is not None):
72                     posicao_atual += 1
73                     if posicao == posicao_atual:
74                         noh_atual.proximo = (noh_atual.proximo).proximo
75                         return None
76                         noh_atual = noh_atual.proximo
77                 else:
78                     self.remover_inicio()
79             else:
80                 print("Posição inválida! É maior que a lista.")
81         else:
82             print("Lista Vazia")
83
84
85 def remover_final(self) -> None:
86     if self.inicio is not None:

```

```

87         if self.inicio.proximo is not None:
88             noh_atual = self.inicio
89             while (noh_atual.proximo).proximo is not None:
90                 noh_atual = noh_atual.proximo
91                 noh_atual.proximo = None
92         else:
93             self.remover_inicio()
94     else:

```

#### ✓ Que tal testarmos o novo poder de nossa classe Lista?

01. Criaremos um objeto da classe Lista para que possamos adicionar três nohs com o método **inserir\_inicio()**, **inserir\_meio()** e **inserir\_final()** que foi desenvolvido anteriormente e adicionado na classe Lista.
02. Em seguida, imprimir a lista.
03. Então, invocaremos o método **remover\_final()**.
04. Por fim, imprimir a lista novamente.

```

1 lista = Lista()
2 lista.inserir_final("C")
3 lista.inserir_inicio("A")
4 lista.inserir_meio(1, "B")
5 print(lista)
6
7 lista.remover_final()
8 print(lista)
9

```

➡ [A -> B -> C -> None]

[A -> B -> None]

#### ✓ Pesquisar na lista

```

1 # Resolução
2 def pesquisar(self, valor) -> int:
3     posicao = -1
4     if self.inicio is not None:
5         noh_atual = self.inicio
6         while noh_atual is not None:
7             posicao += 1
8             if noh_atual.valor == valor:
9                 return posicao
10            noh_atual = noh_atual.proximo
11     return -1
12

```

#### ✓ Código comentado

```

1 # Resolução
2 def pesquisar(self, valor) -> int:
3     """
4     Método que recebe um valor do usuário e realiza uma
5     busca pelo mesmo valor na lista. Retorna -1 caso não
6     encontre ou a posição do noh que contém o valor procurado.
7     Exemplo de uso:
8
9     lista -> ["A","B","C","D"]
10    resposta = lista.pesquisar("C")
11
12    Resultado: 2

```

```

13     """
14     posicao = -1
15     if self.inicio is not None:
16         noh_atual = self.inicio
17         while noh_atual is not None:
18             posicao += 1
19             if noh_atual.valor == valor:
20                 return posicao
21             noh_atual = noh_atual.proximo
22     return -1

```

✓ *Atualizar a classe Lista com o método **pesquisar()***

```

1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "[" + str(self.inicio) + "]\n"
9
10
11     def inserir_inicio(self, valor):
12         novo_noh = Noh(valor)
13         novo_noh.proximo = self.inicio
14         self.inicio = novo_noh
15
16
17     def tamanho(self) -> int:
18         tamanho = 0
19         if self.inicio is not None:
20             tamanho += 1
21             noh_atual = self.inicio
22             while(noh_atual.proximo is not None):
23                 tamanho += 1
24                 noh_atual = noh_atual.proximo
25             return tamanho
26         else:
27             return tamanho
28
29
30     def inserir_meio(self, posicao: int, valor) -> None:
31         if posicao <= self.tamanho() + 1:
32             if (posicao != 0) and (self.inicio is not None):
33                 posicao_atual = 0
34                 novo_noh = Noh(valor)
35                 noh_atual = self.inicio
36                 while(noh_atual.proximo is not None):
37                     posicao_atual += 1
38                     if posicao == posicao_atual:
39                         novo_noh.proximo = noh_atual.proximo
40                         noh_atual.proximo = novo_noh
41                         return None
42                     noh_atual = noh_atual.proximo
43                 noh_atual.proximo = novo_noh
44             else:
45                 self.inserir_inicio(valor)
46         else:
47             print("Posição inválida! É maior que a lista + 1")
48
49
50     def inserir_final(self, valor):
51         if self.inicio is not None:
52             noh_atual = self.inicio

```

```

53         while(noh_atual.proximo is not None):
54             noh_atual = noh_atual.proximo
55             noh_atual.proximo = Noh(valor)
56     else:
57         self.inserir_inicio(valor)
58
59
60     def remover_inicio(self):
61         if self.inicio is not None:
62             self.inicio = self.inicio.proximo
63
64
65     def remover_meio(self, posicao: int) -> None:
66         if self.inicio is not None:
67             if posicao <= self.tamanho() - 1:
68                 if not (posicao <= 0):
69                     posicao_atual = 0
70                     noh_atual = self.inicio
71                     while(noh_atual.proximo is not None):
72                         posicao_atual += 1
73                         if posicao == posicao_atual:
74                             noh_atual.proximo = (noh_atual.proximo).proximo
75                             return None
76                             noh_atual = noh_atual.proximo
77             else:
78                 self.remover_inicio()
79         else:
80             print("Posição inválida! É maior que a lista.")
81     else:
82         print("Lista Vazia")
83
84
85     def remover_final(self) -> None:
86         if self.inicio is not None:
87             if self.inicio.proximo is not None:
88                 noh_atual = self.inicio
89                 while (noh_atual.proximo).proximo is not None:
90                     noh_atual = noh_atual.proximo
91                     noh_atual.proximo = None
92             else:
93                 self.remover_inicio()
94         else:
95             print("Lista Vazia")
96
97
98     def pesquisar(self, valor) -> int:
99         posicao = -1
100         if self.inicio is not None:
101             noh_atual = self.inicio
102             while noh_atual is not None:
103                 posicao += 1
104                 if noh_atual.valor == valor:
105                     return posicao
106                 noh_atual = noh_atual.proximo
107         return -1
108

```

✓ *Que tal testarmos o novo poder de nossa classe Lista?*

01. Criaremos um objeto da classe Lista para que possamos adicionar três nohs com o método **inserir\_inicio()**, **inserir\_meio()** e **inserir\_final()** que foi desenvolvido anteriormente e adicionado na classe Lista.
02. Em seguida, imprimir a lista.
03. Então, invocaremos o método **pesquisar()**.
04. Por fim, a execução da célula notebook imprime o resultado do método **pesquisar()**.

```

1 lista = Lista()
2 lista.inserir_final("C")
3 lista.inserir_inicio("A")
4 lista.inserir_meio(1, "B")
5 lista.inserir_final("D")
6 print(lista)
7
8 lista.pesquisar("C")
9
10

```

➡ [A -> B -> C -> D -> None]

2

## ▼ Atualizar valor de um noh na lista

```

1 # Resolução
2 def atualizar(self, posicao: int, valor):
3     if ((self.inicio is not None) \
4         and (posicao >= 0) \
5         and (posicao <= self.tamanho() - 1)
6     ):
7         posicao_atual = 0
8         noh_atual = self.inicio
9         while posicao != posicao_atual:
10             noh_atual = noh_atual.proximo
11             posicao_atual += 1
12         noh_atual.valor = valor
13     else:
14         print("Falha: lista vazia ou posição fora do intervalo.")

```

## ▼ Código comentado

```

1 # Resolução
2 def atualizar(self, posicao: int, valor):
3     """
4     Método que atualiza o valor de um noh na lista.
5     Exemplo de uso:
6
7     lista -> ["A","B","C","D"]
8     lista.atualizar(2, "H")
9
10    Resultado: ["A","B","H","D"]
11    """
12    # Se lista não estiver vazia
13    if ((self.inicio is not None) \
14        # E posição for maior ou igual a zero
15        and (posicao >= 0) \
16        # E posição for menor ou igual a zero
17        and (posicao <= self.tamanho() - 1)
18    ):
19        posicao_atual = 0
20        noh_atual = self.inicio
21        # Enquanto posição atual não alcançar a posição selecionada
22        while posicao != posicao_atual:
23            # Noh atual se torna o próximo noh
24            noh_atual = noh_atual.proximo
25            # Incrementa a posição
26            posicao_atual += 1
27        # Quando posição atual for igual a posição selecionada
28        # Alterar valor de noh atual
29        noh_atual.valor = valor
30    else:

```

```
31 # Caso contrário, comunicar lista vazia
32 print("Falha: lista vazia ou posição fora do intervalo.")
```

▼ *Atualizar a classe Lista com o método **atualizar()***

```
1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "["+ str(self.inicio) +"]\n"
9
10
11     def inserir_inicio(self, valor):
12         novo_noh = Noh(valor)
13         novo_noh.proximo = self.inicio
14         self.inicio = novo_noh
15
16
17     def tamanho(self) -> int:
18         tamanho = 0
19         if self.inicio is not None:
20             tamanho += 1
21             noh_atual = self.inicio
22             while(noh_atual.proximo is not None):
23                 tamanho += 1
24                 noh_atual = noh_atual.proximo
25             return tamanho
26         else:
27             return tamanho
28
29
30     def inserir_meio(self, posicao: int, valor) -> None:
31         if posicao <= self.tamanho() + 1:
32             if (posicao != 0) and (self.inicio is not None):
33                 posicao_atual = 0
34                 novo_noh = Noh(valor)
35                 noh_atual = self.inicio
36                 while(noh_atual.proximo is not None):
37                     posicao_atual += 1
38                     if posicao == posicao_atual:
39                         novo_noh.proximo = noh_atual.proximo
40                         noh_atual.proximo = novo_noh
41                         return None
42                 noh_atual = noh_atual.proximo
43                 noh_atual.proximo = novo_noh
44             else:
45                 self.inserir_inicio(valor)
46         else:
47             print("Posição inválida! É maior que a lista + 1")
48
49
50     def inserir_final(self, valor):
51         if self.inicio is not None:
52             noh_atual = self.inicio
53             while(noh_atual.proximo is not None):
54                 noh_atual = noh_atual.proximo
55             noh_atual.proximo = Noh(valor)
56         else:
57             self.inserir_inicio(valor)
58
59
60     def remover_inicio(self):
```

```

61         if self.inicio is not None:
62             self.inicio = self.inicio.proximo
63
64
65     def remover_meio(self, posicao: int) -> None:
66         if self.inicio is not None:
67             if posicao <= self.tamanho() - 1:
68                 if not (posicao <= 0):
69                     posicao_atual = 0
70                     noh_atual = self.inicio
71                     while(noh_atual.proximo is not None):
72                         posicao_atual += 1
73                         if posicao == posicao_atual:
74                             noh_atual.proximo = (noh_atual.proximo).proximo
75                             return None
76                             noh_atual = noh_atual.proximo
77             else:
78                 self.remover_inicio()
79         else:
80             print("Posição inválida! É maior que a lista.")
81     else:
82         print("Lista Vazia")
83
84
85     def remover_final(self) -> None:
86         if self.inicio is not None:
87             if self.inicio.proximo is not None:
88                 noh_atual = self.inicio
89                 while (noh_atual.proximo).proximo is not None:
90                     noh_atual = noh_atual.proximo
91                     noh_atual.proximo = None
92             else:
93                 self.remover_inicio()
94         else:
95             print("Lista Vazia")
96
97
98     def pesquisar(self, valor) -> int:
99         posicao = -1
100         if self.inicio is not None:
101             noh_atual = self.inicio
102             while noh_atual is not None:
103                 posicao += 1
104                 if noh_atual.valor == valor:
105                     return posicao
106                 noh_atual = noh_atual.proximo
107         return -1
108
109
110     def atualizar(self, posicao: int, valor):
111         if ((self.inicio is not None) \
112             and (posicao >= 0) \
113             and (posicao <= self.tamanho() - 1)
114         ):
115             posicao_atual = 0
116             noh_atual = self.inicio
117             while posicao != posicao_atual:
118                 noh_atual = noh_atual.proximo
119                 posicao_atual += 1
120             noh_atual.valor = valor
121         else:
122             print("Falha: lista vazia ou posição fora do intervalo.")

```

✓ *Que tal testarmos o novo poder de nossa classe Lista?*



01. Criaremos um objeto da classe Lista para que possamos adicionar três nós com o método **inserir\_inicio()**, **inserir\_meio()** e **inserir\_final()** que foi desenvolvido anteriormente e adicionado na classe Lista.

02. Em seguida, imprimir a lista.

03. Então, invocaremos o método **atualizar()**.

04. Por fim, executaremos a lista para verificar se o nó teve seu valor atualizado.

```
1 lista = Lista()
2 lista.inserir_final("C")
3 lista.inserir_inicio("A")
4 lista.inserir_meio(1, "B")
5 lista.inserir_final("D")
6 print(lista)
7
8 lista.atualizar(3, "H")
9
10 print(lista)
```

```
↔ [A -> B -> C -> D -> None]
[A -> B -> C -> H -> None]
```

### ✓ Frequência de um valor na lista

```
1 # Resolução
2 def frequencia(self, valor) -> int:
3     frequencia = 0
4     if self.inicio is not None:
5         noh_atual = self.inicio
6         while noh_atual is not None:
7             if noh_atual.valor == valor:
8                 frequencia += 1
9             noh_atual = noh_atual.proximo
10    return frequencia
```

### ✓ Código comentado

```
1 # Resolução
2 def frequencia(self, valor) -> int:
3     """
4         Método que contabiliza as ocorrências de um valor.
5         Exemplo de uso:
6
7             lista -> ["A","C","C","D"]
8             lista.frequencia("C")
9
10            Resultado: 2
11    """
12    frequencia = 0
13    if self.inicio is not None:
14        noh_atual = self.inicio
15        while noh_atual is not None:
16            if noh_atual.valor == valor:
17                frequencia += 1
18            noh_atual = noh_atual.proximo
19    return frequencia
```

### ✓ Atualizar a classe Lista com o método **frequencia()**

```
1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
```

```

4         self.inicio: Noh = noh
5
6
7     def __str__(self) -> str:
8         return "["+ str(self.inicio) +"]\n"
9
10
11     def inserir_inicio(self, valor):
12         novo_noh = Noh(valor)
13         novo_noh.proximo = self.inicio
14         self.inicio = novo_noh
15
16
17     def tamanho(self) -> int:
18         tamanho = 0
19         if self.inicio is not None:
20             tamanho += 1
21             noh_atual = self.inicio
22             while(noh_atual.proximo is not None):
23                 tamanho += 1
24                 noh_atual = noh_atual.proximo
25             return tamanho
26         else:
27             return tamanho
28
29
30     def inserir_meio(self, posicao: int, valor) -> None:
31         if posicao <= self.tamanho() + 1:
32             if (posicao != 0) and (self.inicio is not None):
33                 posicao_atual = 0
34                 novo_noh = Noh(valor)
35                 noh_atual = self.inicio
36                 while(noh_atual.proximo is not None):
37                     posicao_atual += 1
38                     if posicao == posicao_atual:
39                         novo_noh.proximo = noh_atual.proximo
40                         noh_atual.proximo = novo_noh
41                         return None
42                 noh_atual = noh_atual.proximo
43                 noh_atual.proximo = novo_noh
44             else:
45                 self.inserir_inicio(valor)
46         else:
47             print("Posição inválida! É maior que a lista + 1")
48
49
50     def inserir_final(self, valor):
51         if self.inicio is not None:
52             noh_atual = self.inicio
53             while(noh_atual.proximo is not None):
54                 noh_atual = noh_atual.proximo
55             noh_atual.proximo = Noh(valor)
56         else:
57             self.inserir_inicio(valor)
58
59
60     def remover_inicio(self):
61         if self.inicio is not None:
62             self.inicio = self.inicio.proximo
63
64
65     def remover_meio(self, posicao: int) -> None:
66         if self.inicio is not None:
67             if posicao <= self.tamanho() - 1:
68                 if not (posicao <= 0):
69                     posicao_atual = 0

```

```

70         noh_atual = self.inicio
71         while(noh_atual.proximo is not None):
72             posicao_atual += 1
73             if posicao == posicao_atual:
74                 noh_atual.proximo = (noh_atual.proximo).proximo
75                 return None
76                 noh_atual = noh_atual.proximo
77         else:
78             self.remover_inicio()
79     else:
80         print("Posição inválida! É maior que a lista.")
81     else:
82         print("Lista Vazia")
83
84
85     def remover_final(self) -> None:
86         if self.inicio is not None:
87             if self.inicio.proximo is not None:
88                 noh_atual = self.inicio
89                 while (noh_atual.proximo).proximo is not None:
90                     noh_atual = noh_atual.proximo
91                     noh_atual.proximo = None
92             else:
93                 self.remover_inicio()
94         else:
95             print("Lista Vazia")
96
97
98     def pesquisar(self, valor) -> int:
99         posicao = -1
100         if self.inicio is not None:
101             noh_atual = self.inicio
102             while noh_atual is not None:
103                 posicao += 1
104                 if noh_atual.valor == valor:
105                     return posicao
106                 noh_atual = noh_atual.proximo
107         return -1
108
109
110     def atualizar(self, posicao: int, valor):
111         if ((self.inicio is not None) \
112             and (posicao >= 0) \
113             and (posicao <= self.tamanho() - 1)
114         ):
115             posicao_atual = 0
116             noh_atual = self.inicio
117             while posicao != posicao_atual:
118                 noh_atual = noh_atual.proximo
119                 posicao_atual += 1
120             noh_atual.valor = valor
121         else:
122             print("Falha: lista vazia ou posição fora do intervalo.")
123
124
125     def frequencia(self, valor) -> int:
126         frequencia = 0
127         if self.inicio is not None:
128             noh_atual = self.inicio
129             while noh_atual is not None:
130                 if noh_atual.valor == valor:
131                     frequencia += 1
132                 noh_atual = noh_atual.proximo
133         return frequencia
134

```

### ✓ Que tal testarmos o novo poder de nossa classe Lista?

01. Criaremos um objeto da classe Lista para que possamos adicionar três nohs com o método **inserir\_inicio()**, **inserir\_meio()** e **inserir\_final()** que foi desenvolvido anteriormente e adicionado na classe Lista.

02. Em seguida, imprimir a lista.

03. Então, invocaremos o método **frequencia()**.

04. Por fim, a execução da célula notebook imprime o resultado do método **frequencia()**.

```
1 lista = Lista()
2 lista.inserir_final("C")
3 lista.inserir_inicio("A")
4 lista.inserir_meio(1, "D")
5 lista.inserir_final("D")
6 print(lista)
7
8 freq = lista.frequencia("D")
9 print(freq)
```

↗ [A -> D -> C -> D -> None]

2

### ✓ Exclusão da lista

```
1 # Resolução
2 @property
3 def inicio(self):
4     return self._inicio
5
6 @inicio.setter
7 def inicio(self, novo_inicio):
8     self._inicio = novo_inicio
9
10 def excluir_lista(self):
11     self._inicio = None
```

### ✓ Código comentado

```
1 # Resolução
2 @property
3 def inicio(self):
4     return self._inicio
5
6 @inicio.setter
7 def inicio(self, novo_inicio):
8     self._inicio = novo_inicio
9
10 def excluir_lista(self):
11     """
12     Para exclusão da lista, apenas atribuiremos None ao
13     atributo de instância e assim perde-se a referência
14     do início da lista e o restante da mesma.
15     """
16     self._inicio = None
```

### ✓ Atualizar a classe Lista com o método **excluir\_lista()**

```
1 class Lista:
2
3     def __init__(self, noh: Noh = None) -> None:
4         self._inicio: Noh = noh
5
```

```

5
6 @property
7 def inicio(self):
8     return self._inicio
9
10 @inicio.setter
11 def inicio(self, novo_inicio):
12     self._inicio = novo_inicio
13
14 def __str__(self) -> str:
15     return "[" + str(self._inicio) + "]\n"
16
17
18 def inserir_inicio(self, valor):
19     novo_noh = Noh(valor)
20     novo_noh.proximo = self._inicio
21     self._inicio = novo_noh
22
23
24 def tamanho(self) -> int:
25     tamanho = 0
26     if self._inicio is not None:
27         tamanho += 1
28         noh_atual = self._inicio
29         while(noh_atual.proximo is not None):
30             tamanho += 1
31             noh_atual = noh_atual.proximo
32     return tamanho
33
34     else:
35         return tamanho
36
37 def inserir_meio(self, posicao: int, valor) -> None:
38     if posicao <= self.tamanho() + 1:
39         if (posicao != 0) and (self._inicio is not None):
40             posicao_atual = 0
41             novo_noh = Noh(valor)
42             noh_atual = self._inicio
43             while(noh_atual.proximo is not None):
44                 posicao_atual += 1
45                 if posicao == posicao_atual:
46                     novo_noh.proximo = noh_atual.proximo
47                     noh_atual.proximo = novo_noh
48                     return None
49             noh_atual = noh_atual.proximo
50             noh_atual.proximo = novo_noh
51         else:
52             self.inserir_inicio(valor)
53     else:
54         print("Posição inválida! É maior que a lista + 1")
55
56
57 def inserir_final(self, valor):
58     if self._inicio is not None:
59         noh_atual = self._inicio
60         while(noh_atual.proximo is not None):
61             noh_atual = noh_atual.proximo
62         noh_atual.proximo = Noh(valor)
63     else:
64         self.inserir_inicio(valor)
65
66
67 def remover_inicio(self):
68     if self._inicio is not None:
69         self._inicio = self._inicio.proximo
70
71

```

```

72 def remover_meio(self, posicao: int) -> None:
73     if self._inicio is not None:
74         if posicao <= self.tamanho() - 1:
75             if not (posicao <= 0):
76                 posicao_atual = 0
77                 noh_atual = self._inicio
78                 while(noh_atual.proximo is not None):
79                     posicao_atual += 1
80                     if posicao == posicao_atual:
81                         noh_atual.proximo = (noh_atual.proximo).proximo
82                         return None
83                         noh_atual = noh_atual.proximo
84             else:
85                 self.remover_inicio()
86         else:
87             print("Posição inválida! É maior que a lista.")
88     else:
89         print("Lista Vazia")
90
91
92 def remover_final(self) -> None:
93     if self._inicio is not None:
94         if self._inicio.proximo is not None:
95             noh_atual = self._inicio
96             while (noh_atual.proximo).proximo is not None:
97                 noh_atual = noh_atual.proximo
98                 noh_atual.proximo = None
99         else:
100             self.remover_inicio()
101     else:
102         print("Lista Vazia")
103
104
105 def pesquisar(self, valor) -> int:
106     posicao = -1
107     if self._inicio is not None:
108         noh_atual = self._inicio
109         while noh_atual is not None:
110             posicao += 1
111             if noh_atual.valor == valor:
112                 return posicao
113             noh_atual = noh_atual.proximo
114     return -1
115
116
117 def atualizar(self, posicao: int, valor):
118     if ((self._inicio is not None) \
119         and (posicao >= 0) \
120         and (posicao <= self.tamanho() - 1)
121     ):
122         posicao_atual = 0
123         noh_atual = self._inicio
124         while posicao != posicao_atual:
125             noh_atual = noh_atual.proximo
126             posicao_atual += 1
127         noh_atual.valor = valor
128     else:
129         print("Falha: lista vazia ou posição fora do intervalo.")
130
131
132 def frequencia(self, valor) -> int:
133     frequencia = 0
134     if self._inicio is not None:
135         noh_atual = self._inicio
136         while noh_atual is not None:
137             if noh_atual.valor == valor:
138                 frequencia += 1

```

```
139         noh_atual = noh_atual.proximo
140     return frequencia
141
142
143     def excluir_lista(self):
144         self._inicio = None
145
```

✓ *Que tal testarmos o novo poder de nossa classe Lista?*


01. Criaremos um objeto da classe Lista para que possamos adicionar três nohs com o método **inserir\_inicio()**, **inserir\_meio()** e **inserir\_final()** que foi desenvolvido anteriormente e adicionado na classe Lista.

02. Em seguida, imprimir a lista.

03. Então, invocaremos o método **excluir\_lista()**.

04. Por fim, tentamos imprimir lista que foi excluída.

```
1 lista = Lista()
2 lista.inserir_final("C")
3 lista.inserir_inicio("A")
4 lista.inserir_meio(1, "D")
5 lista.inserir_final("D")
6 print(lista)
7
8 lista.excluir_lista()
9 print(lista)
```

 [A -> D -> C -> D -> None]

[None]