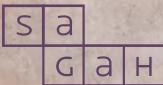


ESTRUTURA DE DADOS

Lucas Plautz Prestes



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS

Implementação de listas encadeadas simples

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Analisar a estrutura de listas em Python.
- Desenvolver listas encadeadas simples em Python.
- Resolver problemas que utilizem listas encadeadas simples.

Introdução

O estudo de listas encadeadas é o primeiro passo para a compreensão do modo como manipulamos uma grande quantidade de dados em programação. O desenvolvimento dessas listas pode ser realizado a partir de bibliotecas disponíveis nas mais diferentes linguagens de programação.

As bibliotecas utilizadas em programação auxiliam no desenvolvimento de listas simplesmente encadeadas e, apesar de serem práticas, não disponibilizam todas as funcionalidades necessárias para o desenvolvimento de uma aplicação; logo, é necessário que o programador compreenda o que se encontra no interior dos métodos utilizados. Sendo assim, durante os estudos, recomenda-se o desenvolvimento dos principais métodos disponíveis nas bibliotecas utilizadas.

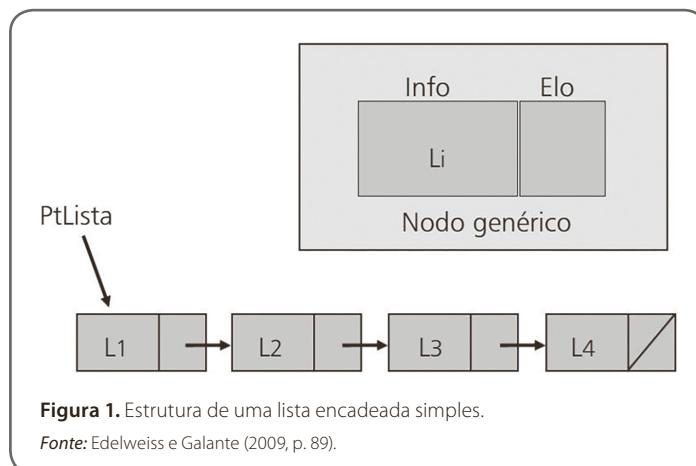
Neste capítulo, você vai estudar sobre as listas simplesmente encadeadas e o seu desenvolvimento e aplicação em linguagem Python.

1 Estrutura de listas em Python

Antes de iniciarmos o estudo e o desenvolvimento de listas em linguagem Python, devemos remeter ao seu conceito primordial. A lista simplesmente encadeada constitui uma relação entre elementos, nodos, interligados entre si, em que cada elemento é composto de uma estrutura que contém um endereço

para o próximo nodo e o seu respectivo conteúdo, podendo ser do tipo primitivo ou criado pelo usuário (EDELWEISS; GALANTE, 2009).

Na Figura 1 é possível analisarmos a estrutura de uma lista simplesmente encadeada, bem como do seu respectivo nodo.



A linguagem Python contém funções, classes e métodos que irão nos auxiliar no desenvolvimento de listas. As funções e os métodos utilizados são:

count (Elemento) — Informa a quantidade de elementos enviados por parâmetro que existem na variável Lista.

#Código em Python:

```
# Lista de animais
animais = ['Gato', 'Cachorro', 'papagaio','Gato']

# imprime a nova lista
print(animais.count('Gato'))
```

#Saída do programa

```
# 2
```

clear() — Realiza a limpeza de todos os nodos de uma lista. Essa limpeza libera o espaço ocupado na memória do computador. Uma vez executada, os dados não poderão ser recuperados.

#Código em Python:

```
# Lista de animais
animais = ['Gato', 'Cachorro', 'papagaio','Gato']

# imprime a lista
print(animais)

# Limpa os elementos da lista
animais.clear()

#Imprime a lista
print(animais)
```

#Saída do programa

```
# ['Gato', 'Cachorro', 'papagaio', 'Gato']
# []
```

append(Elemento) — Adiciona a variável elemento ao final da lista.

#Código em Python:

```
# Lista de animais
animais = ['Gato', 'Cachorro', 'papagaio']

# Adiciona o coelho no final da lista
animais.append('coelho')

# imprime a nova lista
print('Atualização da lista: ', animais)
```

#Saída do programa

```
# Atualização da lista: ['Gato', 'Cachorro', 'papagaio', 'coelho']
```

insert(*Posição, Elemento*) — Adiciona o elemento em uma posição específica de uma lista.

#Código em Python:

```
# Lista de animais
animais = ['Gato', 'Cachorro', 'papagaio','Gato']

# imprime a lista
print(animais)

# Insere o elemento Rato na posição 1 da lista
animais.insert(1,'Rato')

#Imprime a lista
print(animais)
```

#Saída do programa

```
#[‘Gato’, ‘Cachorro’, ‘papagaio’, ‘Gato’]
#[‘Gato’, ‘Rato’, ‘Cachorro’, ‘papagaio’, ‘Gato’]
```

remove(*Elemento*) — Realiza a remoção da primeira ocorrência do elemento em uma determinada lista.

#Código em Python:

```
# Lista de animais
animais = ['Gato', 'Cachorro', 'papagaio','Gato']

# imprime a lista
print(animais)

# Remove a primeira ocorrência do elemento Gato na Lista
animais.remove('Gato')

#Imprime a lista
print(animais)
```

#Saída do programa

```
#[‘Gato’, ‘Cachorro’, ‘papagaio’, ‘Gato’]
#[‘Cachorro’, ‘papagaio’, ‘Gato’]
```

index(Elemento) — Retorna à posição da primeira ocorrência do elemento em uma lista.

#Código em Python:

```
# Lista de animais
animais = ['Gato', 'Cachorro', 'papagaio', 'Gato']

# imprime a lista
print(animais)

# Retorna e imprime a primeira ocorrência do elemento Cachorro
print(animais.index('Cachorro'))

#Saída do programa

# ['Gato', 'Cachorro', 'papagaio', 'Gato']
# 1
```

range(inicio,fim) ou range(inicio,fim,incremento) — Gera listas contendo progressões aritméticas por intervalo informando, ou não, o respectivo incremento entre os nodos.

#Código em Python:

```
# Gera uma lista de 5 elementos iniciando em zero com incremento
#padrão +1
print (list(range(5)))

# Gera uma lista com início no elemento 2, final do elemento 20 com
#incremento de +3 em cada nodo
print(list(range(2, 20, 3)))

#Gera uma sequência de 0 a 9
print(list(range(10)))

#Gera uma sequência de 1 a 11
print( list(range(1, 11)))

#Gera uma sequência de 0 a 30 com incremento de 5
print(list(range(0, 30, 5)))

#Gera uma sequência com os 5 primeiros números da tabuada de 3
print(list(range(0, 10, 3)))
```

```
#Gera uma sequência numérica negativa  
print(list(range(0, -10, -1)))
```

```
#Gera uma sequência numérica vazia  
print( list(range(0)))
```

#Saída do programa

```
# [0, 1, 2, 3, 4]  
# [2, 5, 8, 11, 14, 17]  
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
# [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
# [0, 5, 10, 15, 20, 25]  
# [0, 3, 6, 9]  
# [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]  
# []
```



Fique atento

O desenvolvedor deverá ter atenção nas chamadas de funções e conteúdo dos nodos das listas quanto a letras maiúsculas e minúsculas, pois Python é *case sensitive*. Por exemplo, a palavra **Casa** é diferente da palavra **casa**, pois a primeira palavra tem a primeira letra em caixa-alta.

2 Desenvolvimento de listas encadeadas simples

As operações de manipulação de uma lista simplesmente encadeada permitem a manipulação de seus nodos durante a sua execução. Dessa forma, ao realizarmos o estudo dessas listas, recomenda-se a criação de algumas funções sem o uso de bibliotecas específicas, assim, facilitará a sua correta compreensão de forma a não depender de bibliotecas de terceiros ou até mesmo possibilitar a criação de suas próprias bibliotecas.

As operações que devemos desenvolver, relacionadas às listas simplesmente encadeadas, são:

1. Criar uma classe `Nodo`.
2. Criar uma classe `Lista`.
3. Criar uma função que verifica se a lista está vazia.
4. Criar uma função que informa o tamanho da lista
5. Criar uma função que adiciona um elemento no início da lista.
6. Criar uma função que adiciona um elemento no final da lista.
7. Criar uma função que adiciona um elemento em uma posição específica da lista.
8. Criar uma função que remove um elemento em uma posição específica da lista.
9. Criar uma função que remove o primeiro elemento da lista.
10. Criar uma função que remove o último elemento da lista.
11. Criar uma função que conta o número de ocorrências de um elemento na lista.



Fique atento

Os exemplos com desenvolvimento em Python são dependentes. Logo, em todos os exemplos, é necessário adicionar o código Python de exemplos anteriores de forma retroativa para o seu correto funcionamento.

Os métodos para a criação das classes `nodo`, `lista encadeada`, verificação de lista vazia, adicionar e remover posição na lista são elementos básicos no que se refere à implementação de listas encadeadas simples, nas quais você já pode ter tido contato com eles anteriormente. No entanto, vamos retomá-los aqui, pois são essenciais para o embasamento dos novos métodos que vamos introduzir.

O desenvolvimento dos códigos a seguir foi adaptado a partir de exemplos em Borges (2010) e Schildt (1996).

Criação de um Nodo

```
#Código em Python

#Classe Nodo
class Nodo:
    def __init__(self, dado=0, proximo_nodo=None):
        self.conteudo = dado
        self.proximo = proximo_nodo

    def __repr__(self):
        return '%s -> %s' % (self.conteudo, self.proximo)

# Programa Principal de teste

Nd = Nodo()
print(Nd)

Nd2 = Nodo("casa")
print(Nd2)

#Saída do Programa
#0 -> None
#casa -> None
```

Criação da classe ListaEncadeada

```
#Código em Python

# Classe Lista Encadeada
class ListaEncadeada:

    def __init__(self):
        self.inicio = None

    def __repr__(self):
        return "[" + str(self.inicio) + "]"

# Programa Principal de teste
lista = ListaEncadeada()
Nd = Nodo("Primeiro")
lista.inicio = Nd
print(lista)

#Saída do Programa
# [Primeiro -> None]
```

Criação da função ListaVazia

#Código em Python

```
#Metodo que verifica se a lista está Vazia, Caso positivo Retorna True caso
contrário False
def ListaVazia(Lista):
    if Lista.inicio == None :
        return True
    else:
        return False

# Programa Principal de teste

lista = ListaEncadeada()
print("A lista está vazia: ", ListaVazia(lista))

Nd = Nodo("Primeiro")
lista.inicio = Nd

print("A lista está vazia: ", ListaVazia(lista))

#Saída do Programa

#A Lista está vazia:  True
#A Lista está vazia:  False
```

Criação do método TamanhoLista

#Código em Python

```
# Método que informa o tamanho da Lista
def Tamanholista(self):
    atual = self.inicio
    tamanho = 0

#Verifica se a lista está vazia e já retorna com Zero
    if self.inicio == None:
        return 0

#Verifica se a lista possui somente um elemento
    if atual.conteudo != None and atual.proximo == None:
        tamanho = 1

# Percorre toda a Lista até o último nodo ser vazio, incrementando +1 em cada
Loop e passando para a posição do próximo nodo
    while atual.proximo != None:
        tamanho=tamanho+1
        atual = atual.proximo

    return tamanho
```

```
# Programa Principal de teste
lista = ListaEncadeada()
Nd = Nodo("Primeiro")

#Adiciona 1 elemento na lista
lista.inicio = Nd

print("O tamanho da lista: ", TamanhoLista(lista))
#Saída do Programa
#0 tamanho da lista: 1
```

Criação da função Adiciona_Inicio

#Código em Python

```
#Método que Adiciona um elemento no início da lista
def Adiciona_Inicio(self, Nd):

    # Cria o nodo com o 1 elemento da lista
    atual = self.inicio
    #Atribui ao novo nodo o primeiro elemento da lista
    Nd.proximo = atual
    #Atualiza o primeiro elemento da lista como sendo o novo elemento
    self.inicio = Nd

# Programa Principal de teste

lista = ListaEncadeada()

Nd = Nodo("A")
Adiciona_Inicio(lista,Nd)
print("lista: ", lista)

Nd2 = Nodo("B")
Adiciona_Inicio(lista,Nd2)
print("lista: ", lista)

#Saída do Programa

#lista: [A -> None]
#lista: [B -> A -> None]
```

Criação da função Adiciona_Fim

#Código em Python

```
def Adiciona_Fim(self, item):
    #Verifica se a lista não está vazia para percorrer, pois, se estiver, irá
    #adicionar no inicio
    if self.inicio:
        aux = self.inicio
    #Percorre a lista até o último elemento
    while (aux.proximo):
        aux = aux.proximo
    #Quando sair do laço é porque chegou no último elemento; logo, atualiza o
    #último elemento a receber como próximo o novo elemento
    aux.proximo = item
    #Como a lista está vazia, adiciona no inicio da lista
    else:
        self.inicio = item

# Programa Principal de teste
lista = ListaEncadeada()

Nd = Nodo("A")
Adiciona_Fim(lista,Nd)
print("lista: ", lista)

Nd2 = Nodo("B")
Adiciona_Fim(lista,Nd2)
print("lista: ", lista)

Nd3 = Nodo("C")
Adiciona_Fim(lista,Nd3)
print("lista: ", lista)

#Saída do Programa
#lista: [A -> None]
#lista: [A -> B -> None]
#lista: [A -> B -> C -> None]
```

Criação da função Adiciona_Pos

#Código em Python

```
def Adiciona_Posicao(self,item,pos):
    atual = self.inicio
    pos_atual = 0

    #Percorre até o final da lista
    while atual.proximo != None:

        #Verifica se a posição atual é anterior à posição desejada
        if pos_atual == (pos-1):
            #Realiza a troca dos ponteiros e insere o novo elemento
            pointer = atual
            item.proximo = pointer.proximo
            pointer.proximo = item
        else:
            atual = atual.proximo

    pos_atual = pos_atual + 1

# Programa Principal de teste
lista = ListaEncadeada()

Nd = Nodo("A")
Adiciona_Inicio(lista,Nd)
print("lista: ", lista)

Nd2 = Nodo("B")
Adiciona_Inicio(lista,Nd2)
print("lista: ", lista)

Nd3 = Nodo("C")
Adiciona_Posicao(lista,Nd3,1)
print("lista: ", lista)

#Saída do Programa
#Lista: [A -> None]
#Lista: [B -> A -> None]
#Lista: [B -> C -> A -> None]
```

Criação da função Remove _ Pos

#Código em Python

```
def Remove_Pos(self,pos):
    atual = self.inicio
    pos_atual = 0

#Verifica se a posição a ser removida é a primeira; se positivo, atualiza
o elemento de início da lista.

    if pos_atual == pos:
        self.inicio = atual.proximo

#Se a posição a ser removida não é a primeira, percorre a lista até achar
a posição anterior à desejada para a troca dos ponteiros.

    else:
        while atual.proximo != None:

            if pos_atual == (pos-1):
                next = atual.proximo
                atual.proximo = next.proximo
            else:
                atual = atual.proximo

        pos_atual = pos_atual + 1

# Programa Principal de teste
lista = ListaEncadeada()

Nd = Nodo("A")
Adiciona_Fim(lista,Nd)
print("lista: ", lista)

Nd2 = Nodo("B")
Adiciona_Fim(lista,Nd2)
print("lista: ", lista)

Nd3 = Nodo("C")
Adiciona_Fim(lista,Nd3)
print("lista: ", lista)

Remove_Pos(lista,2)
print("lista: ", lista)

Remove_Pos(lista,0)
print("lista: ", lista)

#Saída do Programa
#lista: [A -> None]
#lista: [A -> B -> None]
#lista: [A -> B -> C -> None]
#lista: [A -> B -> None]
#lista: [B -> None]
```

Criação da função Remove_Início

#Código em Python

```
def Remove_Inicio(self):
    #Realiza a atribuição do início da Lista para um elemento Nodo
    atual = self.inicio
    #Informa para a Lista que o seu primeiro elemento é o elemento
    #posterior ao elemento atribuído anteriormente
    self.inicio = atual.proximo

# Programa Principal de teste
lista = ListaEncadeada()

Nd = Nodo("A")
Adiciona_Fim(lista,Nd)
print("lista: ", lista)

Nd2 = Nodo("B")
Adiciona_Fim(lista,Nd2)
print("lista: ", lista)

Nd3 = Nodo("C")
Adiciona_Fim(lista,Nd3)
print("lista: ", lista)

Remove_Inicio(lista)
print("lista: ", lista)

Remove_Inicio(lista)
print("lista: ", lista)

Remove_Inicio(lista)
print("lista: ", lista)

#Saída do Programa
#Lista: [A -> None]
#Lista: [A -> B -> None]
#Lista: [A -> B -> C -> None]
#Lista: [B -> C -> None]
#Lista: [C -> None]
#Lista: [None]
```

Criação da função Remove _ Fim

#Código em Python

```
def Remove_Ultimo(self):
    atual = self.inicio

    #Percorre toda a Lista de elementos
    while atual.proximo != None:
        #atribui o próximo elemento a uma variável temporária
        proximo = atual.proximo

        #Verifica se a variável temporária não possui nodo posterior; se
        #positivo, ele é o último elemento
        if proximo.proximo == None:
            #Atualiza o elemento anterior ao temporário com o seu próximo Nulo.
            atual.proximo = None
        else:
            atual = atual.proximo

# Programa Principal de teste
lista = ListaEncadeada()

Nd = Nodo("A")
Adiciona_Fim(lista,Nd)
print("lista: ", lista)

Nd2 = Nodo("B")
Adiciona_Fim(lista,Nd2)
print("lista: ", lista)

Nd3 = Nodo("C")
Adiciona_Fim(lista,Nd3)
print("lista: ", lista)

Remove_Ultimo(lista)
print("lista: ", lista)

#Saída do Programa
#Lista: [A -> None]
#Lista: [A -> B -> None]
#Lista: [A -> B -> A -> None]
#Lista: [A -> B -> None]
```

Criação da função Ocorrência

#Código em Python

```
def Ocorrencia(self,dado):
    atual = self.inicio
    total_ocorrencia = 0

    #Percorre toda a lista em busca do elemento
    while atual != None:
        #Caso o elemento seja encontrado, deverá adicionar +1 no contador
        if atual.conteudo == dado:
            total_ocorrencia = total_ocorrencia +1
            atual = atual.proximo
        else:
            atual = atual.proximo
    #Após percorrer toda a lista, retornar o total de elementos encontrados.
    return total_ocorrencia

# Programa Principal de teste
lista = ListaEncadeada()

Nd = Nodo("A")
Adiciona_Fim(lista,Nd)
print("lista: ", lista)

Nd2 = Nodo("B")
Adiciona_Fim(lista,Nd2)
print("lista: ", lista)

Nd3 = Nodo("A")
Adiciona_Fim(lista,Nd3)
print("lista: ", lista)

print("Ocorrencias da letra A: ", Ocorrencia(lista,"A"))

#Saída do Programa
#Lista: [A -> None]
#Lista: [A -> B -> None]
#Lista: [A -> B -> A -> None]
#Ocorrencias da Letra A: 2
```

3 Problemas com o uso de listas encadeadas simples

O desenvolvimento de listas encadeadas não se restringe exclusivamente aos exemplos desenvolvidos neste capítulo, com o uso de classes primitivas, mas com as aplicações que façam o uso de tipos de dados definidos pelo usuário. Para isso, vamos desenvolver o algoritmo em linguagem de programação Python, sem o uso de bibliotecas, referente a uma lista de funcionários em

uma loja de departamentos. Essa lista deverá ser simplesmente encadeada, com a criação dos seguintes métodos:

- criar as classes Lista, Nodo e Funcionario;
- informar o número de funcionários;
- adicionar um funcionário na lista;
- remover um funcionário da lista.

```
#Código em Python
# -----
# Criação das Classes
# -----
#Classe Funcionário
class Funcionario:
    def __init__(self, Nome= "indefinido", Código = "indefinido",
Departamento = "indefinido" ):
        self.Nome = Nome
        self.Código = Código
        self.Departamento = Departamento

    def __repr__(self):
        return '%s -> %s -> %s' % (self.Código, self.Departamento, self.Nome,
)

#Classe Nodo
class Nodo:
    class Nodo:
        def __init__(self, dado=0, proximo_nodo=None):
            self.conteudo = dado
            self.proximo = proximo_nodo

        def __repr__(self):
            return '%s -> %s' % (self.conteudo, self.proximo)

# Classe Lista Encadeada
class ListaEncadeada:

    def __init__(self):
        self.início = None

    def __repr__(self):
        return "[" + str(self.início) + "]"

    def ListaVazia(self):
        if self.início == None:
            return True
        else:
```

```
        return False
#
# -----
# O método de impressão percorre toda a lista até o final, realizando a
# impressão de cada Nodo do tipo funcionário
# Para facilitar a compreensão, foi utilizada uma variável cont para informar
# a posição do usuário na lista
# -----
def ImprimeLista(self):
    atual = self.inicio

    cont = 0
    print("Inicio da Lista")
    if self.inicio == None:
        print("Lista Vazia")
    else:
        while atual!= None:
            print("Posição: ", cont, atual)
            cont = cont + 1
            atual = atual.proximo
    print("Final da Lista")

#
# -----
# Métodos reutilizados a partir da função anterior que conta o número de
# elementos de uma lista, no qual somente
# foi renomeada de Tamanholista para Numero_Funcionários
# -----
def Numero_Funcionarios(self):
    atual = self.inicio
    tamanho = 0

    if self.inicio == None:
        return 0

    while atual!= None:
        tamanho = tamanho + 1
        atual = atual.proximo

    return tamanho

#
# -----
# Métodos reutilizados a partir da função Adiciona_Fim sem modificações
# O nome foi atualizado de Adiciona_Fim para NovoFuncionario
# -----
```

```
def NovoFuncionario(self, Nd):  
    # Cria o nodo com o 1 elemento da lista  
    atual = self.inicio  
    # Atribui ao novo nodo o primeiro elemento da lista  
    Nd.proximo = atual  
    # Atualiza o primeiro elemento da lista como sendo o novo elemento  
    self.inicio = Nd  
  
#-----  
#Método adaptado a partir da função Remove_Pos, substituindo a posição a ser  
removida pelo conteúdo do objeto  
#-----  
  
def Remove_Funcionario(self, Func):  
    atual = self.inicio  
    pos_atual = 0  
  
    #Verifica se a posição a ser removida é a primeira; se positivo, atualiza  
o elemento de início da lista.  
    if atual == Func:  
        self.inicio = atual.proximo  
  
    #Se a posição a ser removida não é a primeira, percorre a lista até achar  
a posição anterior à desejada para a troca dos ponteiros.  
    else:  
        while atual.proximo != None:  
            next = atual.proximo  
  
            if next == Func :  
                atual.proximo = next.proximo  
            else:  
                atual = atual.proximo  
  
# -----  
# Programa Principal de teste  
#-----  
  
#Declaração da Lista vazia  
lista = ListaEncadeada()  
  
#Criação dos nodos do tipo funcionários  
#Estes nodos poderiam ser oriundos de uma busca do banco ou Leitura de  
arquivo no qual cada elemento representa um nodo  
#do tipo funcionário.
```

```
F1 = Funcionario("Ad01","Joao","eletronicos")
F2 = Funcionario("Ad02","Maria","eletronicos")
F3 = Funcionario("Ad03","Jose","eletronicos")
F4 = Funcionario("Ad04","Lucas","eletronicos")
F5 = Funcionario("Ad05","Rodrigo","eletronicos")

#Chamada dos métodos pertencentes à Classe Lista que adiciona os funcionários
lista.NovoFuncionario(F1)
lista.NovoFuncionario(F2)
lista.NovoFuncionario(F3)
lista.NovoFuncionario(F4)
lista.NovoFuncionario(F5)

#Chamada do método de impressão da Lista desenvolvido.
lista.ImprimeLista()

#Informa o Número de Funcionários
print ("Total de funcionários na empresa: ",lista.Numero_Funcionarios())

#Remove no meio da Lista
lista.Remove_Funcionario(F3)
lista.ImprimeLista()
print ("Total de funcionários na empresa: ",lista.Numero_Funcionarios())

#Remove no Final da Lista
lista.Remove_Funcionario(F1)
lista.ImprimeLista()
print ("Total de funcionários na empresa: ",lista.Numero_Funcionarios())

#Remove no início da Lista
lista.Remove_Funcionario(F5)
lista.ImprimeLista()
print ("Total de funcionários na empresa: ",lista.Numero_Funcionarios())

# -----
#Saída do programa
#-----
```



```
#Início da Lista
#Posição: 0 Rodrigo -> eletronicos -> Ad05
#Posição: 1 Lucas -> eletronicos -> Ad04
#Posição: 2 Jose -> eletronicos -> Ad03
#Posição: 3 Maria -> eletronicos -> Ad02
#Posição: 4 Joao -> eletronicos -> Ad01
#Final da Lista
#Total de funcionários na empresa: 5

#Início da Lista
#Posição: 0 Rodrigo -> eletronicos -> Ad05
#Posição: 1 Lucas -> eletronicos -> Ad04
#Posição: 2 Maria -> eletronicos -> Ad02
#Posição: 3 Joao -> eletronicos -> Ad01
```

```
#Final da Lista  
#Total de funcionários na empresa: 4  
  
#Início da Lista  
#Posição: 0 Rodrigo -> eletronicos -> Ad05  
#Posição: 1 Lucas -> eletronicos -> Ad04  
#Posição: 2 Maria -> eletronicos -> Ad02  
#Final da Lista  
#Total de funcionários na empresa: 3  
  
#Início da Lista  
#Posição: 0 Lucas -> eletronicos -> Ad04  
#Posição: 1 Maria -> eletronicos -> Ad02  
#Final da Lista  
#Total de funcionários na empresa: 2
```



Referências

- BORGES, L. E. *Python para desenvolvedores*. 2. ed. Rio de Janeiro: Edição do Autor, 2010.
- EDELWEISS, N.; GALANTE, R. *Estruturas de dados*. Porto Alegre: Bookman, 2009. (Livros didáticos informática UFRGS, v. 18).
- SCHILD, H. C: completo e total. São Paulo: Makron Books, 1996.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS