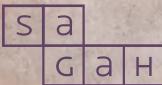


# ESTRUTURA DE DADOS

Lucas Plautz Prestes



SOLUÇÕES  
EDUCACIONAIS  
INTEGRADAS

# Listas encadeadas simples

## Objetivos de aprendizagem

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

- Identificar as operações sobre listas encadeadas simples.
- Resolver problemas aplicando as operações de inserção, remoção e consulta sobre listas encadeadas simples.
- Aplicar o uso de listas encadeadas simples.

## Introdução

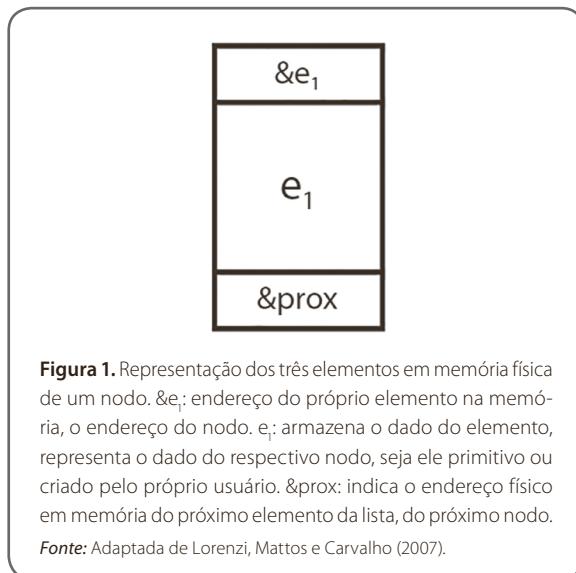
A compreensão de listas em programação é de fundamental importância para a correta manipulação de dados em memória, na qual cada tipo de lista tem características específicas para a sua correta aplicação.

As listas encadeadas simples são um dos tipos de listas dinâmicas mais fáceis de se implementar e são também de suma importância para o desenvolvimento de uma estrutura de dados flexível, com foco na manipulação de dados. A partir dessa informação, será possível implementar operações que permitem o redimensionamento dinâmico e o tamanho da lista, de acordo com a quantidade de itens necessários, logo, ocupando exatamente o tamanho necessário em memória, seja ela linear ou não. Neste capítulo, você vai estudar sobre as listas encadeadas simples, seus conceitos e suas possibilidades.

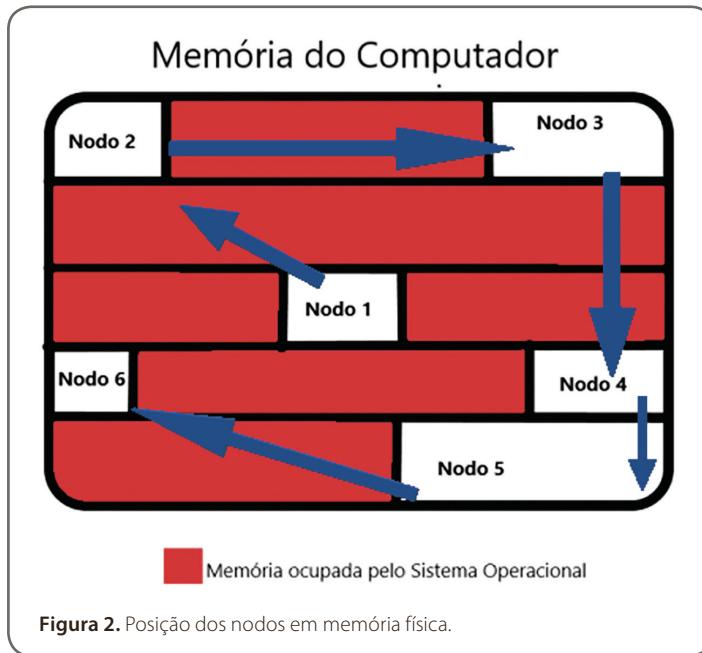
## 1 O que é uma lista encadeada simples?

A lista encadeada simples, também chamada de lista dinâmica simplesmente encadeada, constitui uma relação entre elementos interligados entre si, em que cada elemento é composto de uma estrutura que pode conter variáveis de diversos tipos de dados. Nesse tipo de lista, os elementos podem ser manipulados em tempo de execução, fazendo com que a lista tenha o seu tamanho dinâmico de acordo com a necessidade da aplicação (EDELWEISS; GALANTE, 2009).

A estrutura de um elemento em uma lista encadeada simples, memória física, é dividida em três partes, sendo a primeira o endereço de memória física do nodo, e as duas seguintes pertencentes ao respectivo nodo. A Figura 1 ilustra essa distribuição.



A lista encadeada é composta de nodos, nos quais cada nodo é composto por uma variável, seja ela primitiva ou criada pelo usuário, e um ponteiro, posição de memória, que irá apontar para o nodo seguinte da lista, se houver. Essa estrutura de armazenamento possibilita a alocação de memória física de forma não linear, logo, o seu limite de tamanho está relacionado de acordo com o tamanho da memória física do computador, e não do maior bloco de memória disponível, como ocorria em uma lista estática. Na Figura 2, podemos verificar a posição de memória ocupada por cada nodo de forma não linear e de tamanhos diferentes, pois o dado armazenado poderá ser do tipo diferente, dependendo da linguagem de programação escolhida, como Python.



## Operações sobre lista encadeada simples

As operações de manipulação de uma lista encadeada simples permitem a realização de ações que modificam a lista, sejam elas inclusão, remoção ou alteração de seus elementos. Devemos salientar que as operações de inclusão ou remoção de elementos poderão ser realizadas em qualquer posição da lista, de acordo com as peculiaridades de cada caso.

As operações que podem ser realizadas em uma lista encadeada simples são:

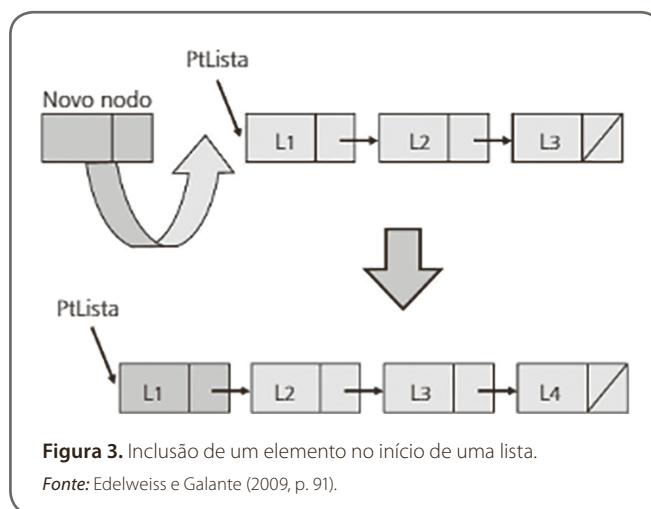
- criar uma lista;
- adicionar elementos à lista;
- remover elementos da lista;
- excluir a lista.

**Criação de uma lista** — A criação de uma lista representa a criação da variável lista na respectiva linguagem desejada. Devemos lembrar que nessa declaração não é necessário informar o número de elementos, pois estamos trabalhando com uma lista dinâmica que possibilitará armazenar um número de elementos de forma indeterminada, logo, o tamanho é indefinido.

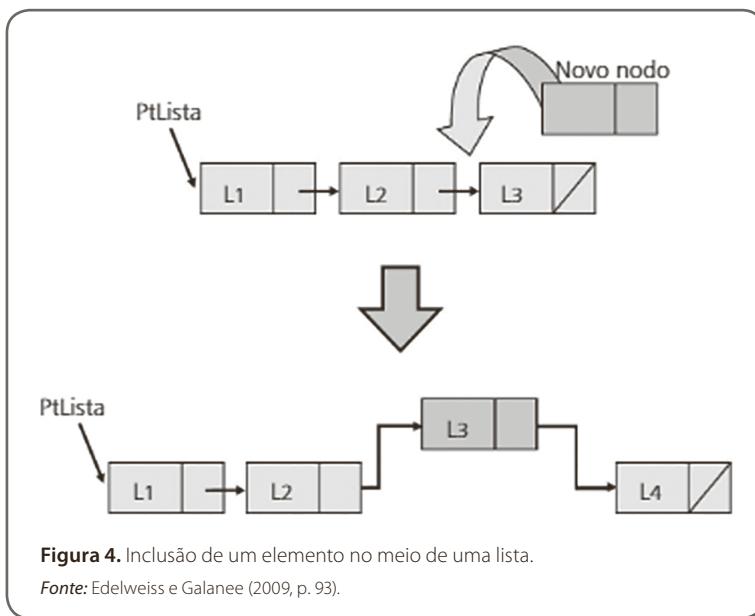
**Adicionar elementos em uma lista** — A criação de um elemento na lista poderá ser realizada a partir de qualquer posição, **sem a necessidade de deslocar os demais elementos**. Isso ocorre devido ao fato de estarmos manipulando uma lista dinâmica, **na qual são modificados apenas os apontamentos, as referências, entre os elementos, assim os mantendo nas posições de memória originalmente criadas**.

Antes de inserir um elemento na lista, o nodo, será necessário alocar um espaço para ele, criar a variável nodo com as suas respectivas informações. A ação posterior será identificar a posição na qual deseja adicionar o elemento e atualizar os seus respectivos apontamentos na lista. Para isso, deverão ser ajustados os apontamentos que indicam quem são os próximos elementos, para que a lista permaneça com a coesão entre os seus elementos, necessária para a sua correta manipulação.

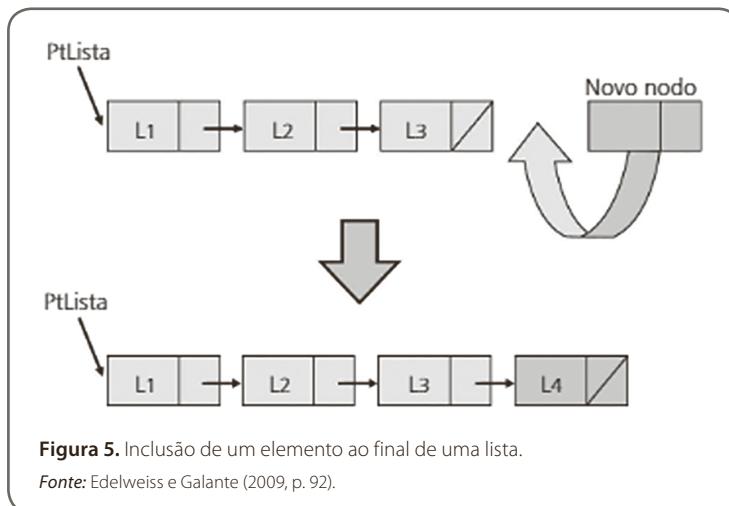
**Incluir elemento no início da lista** — Incluir um novo elemento no início da lista é uma tarefa simples, pois requer somente um apontamento. Para isso, é necessário ajustar o apontamento no início da lista, que deverá indicar o primeiro elemento, novo elemento criado. Este deverá apontar para o primeiro elemento anterior à sua inserção (Figura 3).



**Incluir elemento no meio da lista** — Para incluir um elemento em uma posição intermediária da lista, é necessário percorrer a lista até a posição desejada e alterar os seus apontamentos, fazendo com que o elemento anterior aponte para o novo e este para o próximo elemento (Figura 4).



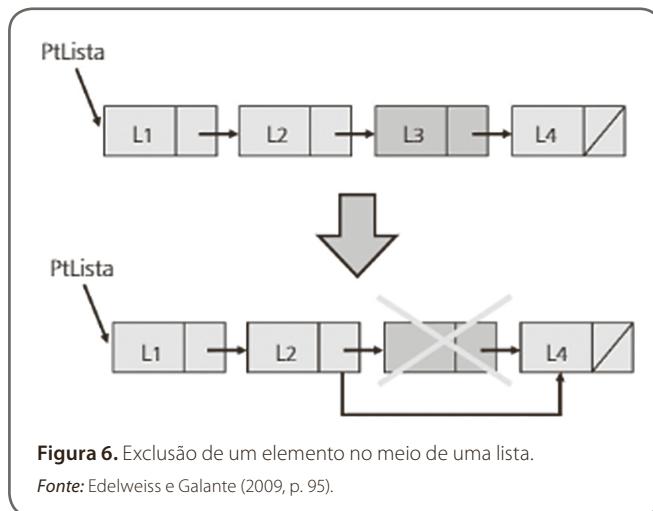
**Incluir elemento ao final da lista** — Incluir um novo elemento no final da lista é uma tarefa simples, pois requer mínima modificação nos elementos da lista. Para isso, é necessário ajustar o apontamento do último elemento da lista, que anteriormente era nulo, para o endereço do novo nodo criado. Este deverá ter apontamento nulo por ser o último elemento da lista, logo, apontando para o seu fim (Figura 5).



**Remover elemento em uma lista** — A remoção de um elemento na lista, assim como a inclusão, poderá ser realizada a partir de qualquer posição, sem a necessidade de deslocar os demais elementos. **Devemos salientar a importância de que toda a vez que for realizada a exclusão de um elemento, seja realizada a correta liberação da memória correspondente**, fazendo com que a lista armazene somente a quantidade de memória necessária para o seu tamanho atual.

**Remover elemento no início da lista** — Excluir um elemento do início da lista é uma tarefa simples. Para isso, será necessário excluir o primeiro nodo da lista e apontar o início da lista para a posição na qual o nodo inicial apontava como seu sucessor.

**Remover elemento no meio da lista** — Para excluir um elemento que se encontra em uma posição intermediária em uma lista, será necessário percorrer a lista até encontrar o elemento desejado, identificando o elemento anterior, armazenado em uma variável temporária, bem como o seu posterior. Na sequência, basta realizar a atualização da informação da referência do nodo anterior ao ser excluído, com a informação do próximo elemento que se encontra no nodo a ser excluído. A partir desse momento, você poderá excluí-lo (Figura 6).



**Figura 6.** Exclusão de um elemento no meio de uma lista.

Fonte: Edelweiss e Galante (2009, p. 95).

**Remover elemento ao final da lista** — Para realizar a exclusão de um elemento ao final de uma lista, será necessário percorrer a lista até chegar no último nodo, que tem a posição do seu próximo como nulo, sempre armazenando em uma variável temporária a posição do nodo anterior a ele. Após a sua identificação, será necessário apontar o nodo anterior para nulo, posição na qual foi armazenada na variável temporária e logo após excluir o nodo final.

**Excluir uma lista** — Para excluir completamente uma lista, será necessário percorrer todos os elementos, a partir do nodo inicial, excluindo nodo a nodo, e assim liberando a alocação de memória de cada elemento. Caso a liberação de memória não seja realizada, haverá lixo de memória no sistema, isto é, espaço de memória ocupado, mas não mais utilizado. Isso geralmente ocorre quando finalizamos um programa de computador de forma interrupta ou não executamos rotinas de liberação de memória de forma controlada ao ser finalizado. As bibliotecas que fazem o uso de listas geralmente têm um destrutor, comando no qual realiza toda essa operação.

## 2 Resolução de problemas relacionados a listas encadeadas simples

As operações básicas em uma lista encadeada são representadas por inserções, remoções e consultas nas suas respectivas listas, de acordo com a necessidade do usuário e documentação referente ao desenvolvimento da aplicação. Todas essas ações devem ser realizadas com foco nas características de listas dinâmicas, como a lista encadeada simples. Com o objetivo de facilitar o desenvolvimento desses problemas, é recomendado realizar funções que realizam essas ações para, assim, poderem ser reutilizadas independentemente da linguagem escolhida. A seguir, será possível identificar alguns algoritmos usualmente desenvolvidos para o uso de listas encadeadas simples, adaptados de Lutz (2011).

### Como identificar se uma lista encadeada está vazia?

Para identificar se uma lista encadeada está vazia, basta analisar se o apontamento para o início da lista encontra-se nulo, nodo inexistente. Geralmente, as bibliotecas de listas dinâmicas têm um método que realiza essa operação e retorna *true* para verdadeiro e *false* para falso.

Algoritmo	Python
<pre> <i>#Declarar variáveis</i> Variavel Lista = []  <i>#Verificar se a lista está vazia, caso positivo</i> <i>retornar 'true' se não False</i>  <i>se (lista.proximo igual a nulo) então</i> <i>retornar true</i> <i>senão</i> <i>retornar false</i> <i>fim se</i> </pre>	<pre> <i>#Classe Nodo</i> class Nodo:     def __init__(self, codigo=0, proximo_nodo=None):         self.codigo = codigo         self.proximo = proximo_nodo      def __repr__(self):         return '%s -&gt; %s' % (self.codigo, self.proximo)  <i>#Classe Lista Encadeada</i> class ListaEncadeada:      def __init__(self):         self.cabeca = None      def __repr__(self):         return "[" + str(self.cabeca) + "]"  <i>#Função que verifica a lista</i> def verifica_vazia(Lista):     if Lista.cabeca == None :         print("True")     else:         print("False")  <i>#Código principal</i>  Lista = ListaEncadeada() verifica_vazia(Lista)  <i>#Saida:</i> <i># True</i> </pre>

## Como acessar o primeiro elemento de uma lista?

Para realizar uma busca em uma lista, é necessário identificar primeiramente o seu elemento inicial.

### Estrutura do código:

Algoritmo	Python
<pre>#Declarar uma variável do tipo nodo Variavel Node Nd</pre> <pre>#Atribuir um elemento na lista Nd = lista.proximo</pre> <pre>#Realizar a impressão do nodo imprime Nd</pre>	<pre>#Classe Nodo class Nodo:     def __init__(self, codigo=0, proximo_nodo=None):         self.codigo = codigo         self.proximo = proximo_nodo      def __repr__(self):         return '%s -&gt; %s' % (self.codigo, self.proximo)</pre> <pre>#Classe Lista Encadeada class ListaEncadeada:      def __init__(self):         self.cabeca = None      def __repr__(self):         return "[" + str(self.cabeca) + "]"  #Código Principal lista = ListaEncadeada() Primeiro = lista.cabeca print(Primeiro)  #Saida: # None # Como a lista está vazia retorna Vazio - None</pre>

## Como pesquisar um elemento na lista?

A partir das características de listas dinâmicas, neste caso lista encadeada simples, sabemos que não é possível acessar diretamente um elemento a partir do seu identificador, como em vetores, logo, para encontrarmos a posição de um elemento em uma lista, é necessário percorrê-la em busca do nodo fazendo o uso de contadores para identificar a sua posição.

Algoritmo	Python
<pre> <i>#Declarar variaveis</i> Variavel Node Nd Variável inteiro cont Variavel Node NdReferencia  <i>#Atribuir valores iniciais</i> Cont = 0 Nd= lista.proximo NdReferencia.dado = 5  <i>#Percorrer lista até encontrar o elemento nulo, logo final da lista</i>  enquanto Nd diferente nulo  <i>#Quando achar um nodo com o conteúdo igual ao nodo de referência, é um nodo que estamos procurando e informamos a sua posição</i>  se Nd.dado igual a NdReferencia.dado      imprime Cont  fim se  Cont = Cont +1 Nd= lista.proximo fim enquanto </pre>	<pre> <i>#Método 1 – função que retorna se achou um elemento em uma lista</i>  def ExisteNodo(self,item):     atual = self.inicio     status = False     while atual != None and not status:         if atual.inf()==item:             status = True         else:             atual = atual.proxNodo()     return status </pre>

## Como incluir um elemento?

Para incluir um elemento em uma lista, é necessário criar o nodo a ser incluído, um nodo temporário para realizar a troca das referências com o nodo anterior, bem como a posição na qual deseja incluir.

Algoritmo	Python
<pre> <i>#Declarar variaveis</i> Variavel Node Nd Variável inteiro posição Variável inteiro cont Variavel Node Ndovo Variável Node temp  <i>#Atribuir valores iniciais</i> cont = 0 Nd= lista.proximo NdNovo.dado = 5 posição = 10  <i>#Percorrer lista até encontrar o</i> <i>elemento nulo, logo final da lista</i> <i>enquanto Nd diferente nulo</i>  <i>enquanto lista.proximo diferente nulo</i>  <i>#Se o contador encontra-se na posição</i> <i>desejada realizar a inserção do nodo</i>  <i>se cont igual a posição</i> temp= lista.proximo lista.proximo= Ndovo Ndovo.proximo= temp <i>fim se</i>  <i>fim enquanto</i> </pre>	<pre> def inserir(self, item, pos):     atual = self.inicio     pos_atual = 0      while atual.proximo != None:          if pos_atual == (pos-1):             pointer = atual             node = Nodo(item)             node.proximo = pointer.proximo             pointer.proximo = node          else:             atual = atual.proximo          pos_atual = pos_atual + 1  <i>#Método 2</i>  lista = [1, 2, 3, 4] lista.insert(2, 10) print(lista)  <i>#Saída:</i> <i>[1, 2, 10, 3, 4]</i> </pre>

## Como excluir um elemento na lista?

Para excluir um elemento em uma lista, é necessário criar um nodo com a posição do nodo anterior para realizar a troca das referências, bem como a posição na qual deseja incluir.

Algoritmo	Python
<pre> #Declarar variaveis Variavel Node NdAnterior Variavel Node NdAtual Variavel Node NdExcluir Variável inteiro posicao  #Atribuir valores iniciais Posição = 10  NdAnterior= lista NdAtual= lista.proximo  #Percorrer lista até encontrar o elemento nulo, logo final da lista enquanto Nd diferente nulo  enquanto NdAtual.proximo diferente nulo  #Achou o elemento para excluir se NdAtual.proximo igual a NdExcluir NdAnterior.proximo = NdAtual.proximo  #Libera a memoria liberaMemoria(Ndactual) sair fim se  #Atualiza as variáveis atual e anterior para a próxima interação do laço NdAnterior = NdAtual NdAtual = NdAtual.proximo  fim enquanto </pre>	<pre> #Método 1  def remover(self, pos):     atual = self.inicio     pos_atual = 0      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  #Método 2  lista = [1, 2, 3, 4] lista.pop(2) print(lista)  #Saída: # [1, 2, 4] </pre>



### Fique atento

Quando o desenvolvedor fizer uso de bibliotecas de listas dinâmicas, é recomendável realizar o estudo da biblioteca e dos métodos disponíveis, para, assim, evitar realizar algoritmos para a resolução de problemas que se encontram disponíveis na biblioteca escolhida.

## 3 Aplicação de listas encadeadas simples

As linguagens de programação têm bibliotecas com diversos métodos que podem ser utilizados para facilitar o uso e a manipulação de dados. Ao aplicarmos o uso de listas encadeadas em linguagem de programação Python, devemos compreender a sua estrutura e os métodos disponíveis para utilização, assim, irá facilitar o desenvolvimento da aplicação.

### Declaração de uma lista em Python:

```
#Lista Vazia  
lista = []  
  
#Lista com 2 elementos  
lista = [5,3]
```

### Contar o número de ocorrências de um valor na lista:

```
#Informa o número de ocorrências do número 3 na lista  
print(lista.count(3))  
#Saída do Programa: 1
```

### Apagar uma lista:

```
#Apaga todos os elementos de uma lista  
lista.clear()
```

### Adicionar elementos em uma lista:

```
#O elemento será adicionado ao final da lista  
lista.append(10)  
  
#O elemento será adicionado em uma posição específica da lista  
#lista.insert(posição, conteúdo)  
lista.insert(1,7)
```

**Remover um elemento em uma lista:**

```
#Remove a primeira ocorrência do elemento em uma lista
lista.remove(3)
```

**Retornar à posição de um elemento em uma lista:**

```
#Retorna a posição da primeira ocorrência do elemento 5 na lista
lista_teste = [5,3,1,4,3,2,1]
print (lista_teste.index(5))
#Saída do Programa: 0
```

**Exemplo**

Com o objetivo de facilitar os testes com listas em linguagem de programação Python, a seguir verificamos o código utilizando os métodos nativos de listas nas quais você poderá estudar o seu funcionamento a partir das saídas impressas. Vamos programar:

**Código em Python:**

```
#Lista de sobremesas
doces = ['pudim', 'bombom', 'chocolate']

# Lista de valores
valores = [2,5,1,10,5]

#printação das listas
print('listas originais')
print(doces)
print(valores)

#Adicionar elementos
doces.append('cocada')
valores.insert(1,50)
print('listas com novos elementos')
print(doces)
print(valores)

#remover elementos
doces.remove('bombom')
valores.remove(1)
print('listas com a remoção de 1 elementos')
print(doces)
print(valores)
```

**Saída do programa:**

```
listas originais  
['pudim', 'bombom', 'chocolate']  
[2, 5, 1, 10, 5]  
listas com novos elementos  
['pudim', 'bombom', 'chocolate', 'cocada']  
[2, 50, 5, 1, 10, 5]  
listas com a remoção de 1 elementos  
['pudim', 'chocolate', 'cocada']  
[2, 50, 5, 10, 5]
```

**Referências**

- EDELWEISS, N.; GALANTE, R. M. *Estruturas de dados*. Porto Alegre: Bookmann, 2009.
- LORENZI, F.; MATTOS, P. N.; CARVALHO, T. *Estruturas de dados*. São Paulo: Thomson, 2007.
- LUTZ, M. *Programming Python*. 4th ed. Beijing: O'Reilly, 2011.

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