Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Capítulo 2

Listas

O computador e suas estruturas de suporte ao processamento de dados passaram por diversos aperfeiçoamentos ao longo do tempo, visando garantir o melhor uso de um recurso essencial para serviços e aplicações modernas: a memória. Com o crescimento exponencial da necessidade de processamento de informações, torna-se fundamental adotar estratégias eficientes para a gestão desse recurso. Para a resolução sofisticada de problemas computacionais, levando em consideração o uso otimizado e inteligente da memória, surge a área da computação denominada estruturas de dados (Forbellone; Eberspächer, 2022). Esta obra se dedica ao estudo e implementação de diferentes formas de organização e manipulação dos dados, permitindo não apenas um armazenamento eficiente, mas um acesso e processamento rápidos.

A maior parte das aplicações que lidam com grandes volumes de dados emprega estruturas de dados avançadas, garantindo um equilíbrio entre a eficiência da memória e o custo computacional adequado para cada problema (Lambert, 2022). A escolha correta dessas estruturas impacta diretamente o desempenho do software, tornando fundamental a compreensão de suas características e aplicabilidades. No contexto da programação, especialmente em linguagens de alto nível como Python, a implementação de estruturas de dados desempenha um papel essencial na criação de algoritmos eficientes. Python oferece uma ampla variedade de recursos para a manipulação de dados, proporcionando flexibilidade e desempenho, fatores que tornam essa linguagem uma das mais populares na atualidade (Guilhon *et al*., 2022). Dessa forma, o estudo de estruturas de dados é essencial para qualquer programador ou cientista de dados, pois permite a criação de soluções mais otimizadas e escaláveis, alinhadas às necessidades computacionais modernas.

Neste capítulo será abordado o conceito de lista, uma das estruturas de dados mais utilizadas nas aplicações modernas, com muitas possibilidades para a resolução de problemas envolvendo práticas de programação. A primeira seção trata do conceito de estrutura unidimensional de dados, apresentando sua vantagem frente às variáveis simples e suas desvantagens quando submetida a um grande volume de dados. A segunda seção apresenta as operações básicas com listas, com exemplos práticos na linguagem de programação Python. A terceira seção aborda a correlação entre listas e funções, reforçando sua importância em operações específicas com listas. Por último, a quarta seção apresenta o conceito de estruturas bidimensionais e suas principais aplicações, com destaque para a representação matricial da matemática aplicada à computação.

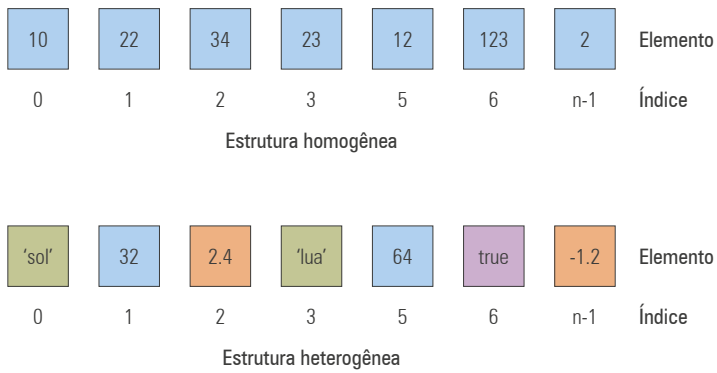
Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

1 Estruturas unidimensionais

As estruturas unidimensionais permitem o armazenamento, em tempo de execução, de conjuntos de dados de diferentes tipos, seja de maneira homogênea ou heterogênea. Enquanto o armazenamento homogêneo agrupa dados do mesmo tipo, o armazenamento heterogêneo agrupa dados de tipos diferentes. Independentemente das características dos dados, o armazenamento sempre terá uma abstração em uma única dimensão. É importante reforçar que esses tipos de armazenamento são encontrados na linguagem de programação Python. A figura 1 apresenta um exemplo de estrutura unidimensional e as diferentes organizações dos dados.

Figura 1 – Estruturas unidimensionais homogêneas e heterogêneas



Na figura 1, é possível observar que, independentemente do conjunto de dados, a estrutura está sempre encapsulada em um elemento referenciado por um índice, o qual, na maioria das linguagens de programação, inicia seu primeiro elemento em zero e tem no elemento final o índice com o número de elementos (representado por n na figura 1) subtraído em 1. Isso permite localizar facilmente um dado a partir de seu índice por meio de técnicas de programação ou certificar-se de que ele não existe na estrutura.

Outro aspecto das estruturas unidimensionais está relacionado ao uso da memória: trata-se da alocação dinâmica da memória ou uso da memória em bloco contíguo. A estrutura de lista em Python (list) utiliza a técnica de alocação dinâmica da memória, que possibilita um melhor aproveitamento dos espaços ociosos em memória, evitando fragmentações. Além disso, as listas possuem maior flexibilidade de operação com elementos, o que torna seu uso frequente nas práticas de programação modernas. A seguir, é apresentado um código-fonte com uso de listas nos formatos homogêneo e heterogêneo. Além do acesso a todos os dados da lista, o exemplo mostra como acessar um elemento específico:

```
# Lista homogênea de números inteiros
numeros = [10, 20, 30, 40, 50]

# Acessar o terceiro elemento (índice 2)
print("Lista homogênea:", numeros)
print("Terceiro elemento:", numeros[2])
```

Saída esperada:

Lista homogênea: [10, 20, 30, 40, 50]

Terceiro elemento: 30

```
# Lista heterogênea com diferentes tipos de dados
dados = ["sol", 32, 2.4, "lua", 64, True, -1.2]

# Acessar o quarto elemento (índice 3)
print("Lista heterogênea:", dados)
print("Quarto elemento:", dados[3])
```

Saída esperada

Lista heterogênea: ['sol', 32, 2.4, 'lua', 64, True, -1.2]

Quarto elemento: lua

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

O armazenamento em bloco contíguo faz a alocação sequencial de dados na memória, sendo comum em estruturas unidimensionais tradicionais de linguagens como C/C++. Apesar da possibilidade de fragmentação de memória (espaços ociosos resultantes do armazenamento sequencial), esse formato de alocação de dados é muito utilizado em aplicações voltadas para grandes volumes em cálculos matemáticos. Outra observação importante é que, na maioria das linguagens de programação, essa estrutura de dados é homogênea e chamada de vetor. Em Python, essa estrutura homogênea e contígua pode ser utilizada através da estrutura array, conforme apresentado a seguir.

```
import array

# Criar um array de números inteiros ('i' indica tipo
inteiro)
numeros = array.array('i', [10, 20, 30, 40, 50])

# Exibir o array completo
print("Array:", numeros)

# Acessar o terceiro elemento (índice 2)
print("Terceiro elemento:", numeros[2])
```

Saída esperada:

Array: array('i', [10, 20, 30, 40, 50])

Terceiro elemento: 30

IMPORTANTE

As listas em Python são altamente flexíveis, permitindo armazenar elementos de diferentes tipos, crescer dinamicamente e suportar operações diversas sem a necessidade de especificar um tamanho fixo. No entanto, essa flexibilidade tem um custo em desempenho e uso de memória, pois as listas armazenam referências aos objetos em vez de valores diretamente. Já a estrutura array.array, embora limitada a um único tipo de dado, oferece um desempenho superior em operações matemáticas, pois armazena os elementos de forma contígua na memória, permitindo cálculos

mais rápidos e eficientes, além de consumir menos espaço. Para aplicações numéricas mais exigentes, bibliotecas como NumPy vão além, proporcionando maior otimização e suporte a operações vetorizadas.

2 Operações básicas

Depois de apresentado o conceito de estruturas unidimensionais, o próximo passo é entender as operações básicas nessas estruturas, concentrando-se na operação com listas. Serão apresentados exemplos de inserção, remoção, alteração e busca de elementos em uma lista. A estrutura de lista em Python possui diferentes funções que permitem uma grande lista de operações.

Um elemento pode ser inserido de várias maneiras em Python, sendo a mais tradicional a inserção ao final da lista pelo método append(), como apresentado no exemplo a seguir.

```
numeros = [10, 20, 30]
numeros.append(40)  # Adiciona 40 no final
print(numeros)
```

O elemento também pode ser inserido em uma posição especificada pelo programador através da função insert():

```
numeros = [10, 20, 30]
numeros.insert(1, 15)  # Insere 15 na posição 1
print(numeros)
```

Também é possível inserir vários elementos ao final da lista por meio da função extend():

```
numeros = [10, 20, 30]
numeros.extend([40, 50, 60])  # Adiciona múltiplos valores ao
final
print(numeros)
```


Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Para a remoção de elementos de uma lista, também existem várias opções de funções. Por exemplo, a função remove() exclui um elemento encontrado em sua primeira ocorrência, realizando a busca do início ao fim da lista.

```
numeros = [10, 20, 30, 40, 30, 50]
numeros.remove(30)  # Remove a primeira ocorrência de 30
print(numeros)
```

Outra opção é a remoção através do índice utilizando a função del():

```
numeros = [10, 20, 30, 40, 50]
del numeros[1]  # Remove o elemento no índice 1 (20)
print(numeros)
```

O programador também pode remover todos os elementos da lista com o comando clear:

```
numeros = [10, 20, 30, 40, 50]
numeros.clear()  # Esvazia a lista
print(numeros)
```

Para alterar o valor de um elemento em Python, é preciso conhecer o elemento que se deseja modificar. Com o índice do elemento identificado, basta atribuir-lhe um novo valor:

```
numeros = [10, 20, 30, 40, 50]

# Alterar o elemento no índice 2 (de 30 para 35)
numeros[2] = 35
print(numeros)
```

Independentemente da maneira que desejamos manipular uma lista, a busca de um elemento pode ser realizada de diversas formas: utilizando operações já implementadas nas listas em Python ou realizando uma busca personalizada através de loops.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

Quanto às implementações em Python para a busca de elementos, uma maneira muito utilizada pela comunidade de especialistas em programação é o operador in, no qual se faz a busca por um valor existente:

```
numeros = [10, 20, 30, 40, 50]

# Verifica se 30 está na lista
if 30 in numeros:
    print("30 está na lista")
```

Todavia, a busca pode ser realizada de maneira personalizada por meio de loops, percorrendo elemento a elemento da lista. Para cada elemento percorrido, verifica-se se ele corresponde ao elemento-alvo. Caso seja encontrado, o especialista pode tomar a ação desejada:

```
numeros = [10, 20, 30, 40, 50]

# Buscar manualmente
for i in range(len(numeros)):
    if numeros[i] == 30:
        print("Encontrado no índice:", i)
        break
```

O código cria uma lista chamada *numeros* contendo os valores [10, 20, 30, 40, 50] e, em seguida, percorre essa lista utilizando um loop for que vai de 0 até len(numeros) - 1 (o último índice da lista). A função len() faz a contagem do número de elementos existentes em uma lista. Dentro do loop, a estrutura if numeros[i] == 30 verifica se o elemento no índice atual é igual a 30. Se for igual a 30, ele imprime a mensagem "Encontrado no índice:" seguida do índice correspondente, e usa o break para interromper o loop imediatamente após encontrar o primeiro 30, evitando verificações desnecessárias. Isso melhora a eficiência, pois o loop não continua executando depois que o elemento procurado é encontrado.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

IMPORTANTE

Python oferece diversas possibilidades para a manipulação de listas, tornando essa estrutura de dados extremamente flexível e eficiente. É possível adicionar novos elementos tanto ao final quanto em posições específicas, além de combinar múltiplas coleções em uma única. Da mesma forma, a remoção de itens pode ser feita de diferentes maneiras, seja excluindo um valor específico, seja retirando elementos por posição ou esvaziando completamente a estrutura. Para localizar dados, existem abordagens que permitem verificar a existência de um item, identificar sua posição ou até encontrar todas as suas ocorrências. Além das operações básicas, há recursos avançados que possibilitam transformar listas de forma eficiente, ordenar valores automaticamente e aplicar funções para modificar os elementos de maneira otimizada. Com essa variedade de ferramentas, a manipulação de listas em Python se adapta a diversas necessidades, desde tarefas simples até o processamento de grandes conjuntos de dados.

3 Listas e funções

Nas linguagens de programação modernas, o desenvolvimento de funções e listas está totalmente integrado, tendo em vista as diferentes possibilidades de manipulação de dados.

Quando funções e listas são utilizadas em conjunto, é possível encontrar diferentes cenários, como o uso de listas como argumento de funções e o retorno de listas em funções.

No que se refere ao uso de listas como argumento de funções, em Python, as listas são passadas por referência. Assim, o que é passado não é uma cópia da lista, e sim uma referência de acesso à lista original, como apresenta o código a seguir:

```
def adicionar_elemento(lista):
    lista.append(100)  # Modifica a lista original
```


Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

```
numeros = [10, 20, 30]
adicionar_elemento(numeros)

print("Lista após a função:", numeros)
```

Saída esperada:

Lista após a função: [10, 20, 30, 100]

É importante ressaltar a possibilidade de copiar a lista para evitar que ela seja manipulada em sua referência. A lista em Python possui o comando copy(), que permite desvincular a cópia da lista da referência anterior:

```
def adicionar_elemento_sem_modificar(lista):
    nova_lista = lista.copy()  # Cria uma cópia
    nova_lista.append(100)
    return nova_lista  # Retorna a lista modificada sem
alterar a original

numeros = [10, 20, 30]
nova_lista = adicionar_elemento_sem_modificar(numeros)

print("Lista original:", numeros)
print("Nova lista modificada:", nova_lista)
```

No que diz respeito ao retorno de valores dessa lista, é necessário ter cuidado, pois o retorno da referência de uma lista ou a cópia de uma nova lista podem trazer mudanças não desejadas à lista original. A seguir, apresentamos um exemplo do retorno da referência de uma lista sem a realização de uma cópia.

```
def adicionar_elemento(lista):
    lista.append(100)  # Modifica a lista original
    return lista  # Retorna a referência da lista original

numeros = [10, 20, 30]
nova_lista = adicionar_elemento(numeros)
```


Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

```
print("Lista original:", numeros)  # Foi modificada
print("Lista retornada:", nova_lista)  # Mesmo endereço na
memória
```

Saída esperada

Lista original: [10, 20, 30, 100]

Lista retornada: [10, 20, 30, 100]

O programador pode optar pelo retorno da cópia de uma lista, realizando as modificações que desejar sem afetar a lista original:

```
def adicionar_elemento_sem_modificar(lista):
    nova_lista = lista.copy()  # Cria uma cópia da lista
original
    nova_lista.append(100)  # Modifica apenas a cópia
    return nova_lista  # Retorna a cópia modificada

numeros = [10, 20, 30]
nova_lista = adicionar_elemento_sem_modificar(numeros)

print("Lista original:", numeros)  # Permanece inalterada
print("Lista retornada:", nova_lista)  # Cópia com
modificação
```

Saída esperada

Lista original: [10, 20, 30]

Lista retornada: [10, 20, 30, 100]

4 Estruturas bidimensionais

Além das estruturas unidimensionais nas estruturas de dados elementares, também devem ser consideradas as estruturas bidimensionais, que organizam os dados em duas dimensões: linhas e colunas. Linguagens de programação como C/C++ e Java possuem estruturas nativas de representação bidimensional através de matrizes. Em Python,

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

a representação de matrizes é adaptada por meio do uso de listas aninhadas. No exemplo a seguir, listas nativas são utilizadas para representar uma matriz de três linhas e três colunas:

```
# Criando uma matriz 3x3 usando listas aninhadas
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Acessando um elemento específico (linha 1, coluna 2)
print(matriz[1][2])  # Saída: 6
```

Também é importante reforçar que percorrer dados em estruturas bidimensionais exige o uso de dois laços de repetição: um para percorrer a primeira dimensão e outro, em cascata, para controlar a segunda dimensão:

```
# Definição de uma matriz 3x3
    matriz = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]

    # Percorrendo a matriz linha por linha
    for linha in matriz:
        for elemento in linha:
            print(elemento, end=" ")
        print()  # Nova linha para organizar a saída
```

Saída esperada

1 2 3

4 5 6

7 8 9

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

IMPORTANTE

Na programação, as matrizes são fundamentais para representar e manipular operações matemáticas de forma eficiente, permitindo resolver problemas complexos em diversas áreas, como álgebra linear, computação gráfica, inteligência artificial e estatística. Ao organizar os dados em uma estrutura bidimensional, é possível realizar cálculos matriciais, como soma, multiplicação, inversão e determinantes, de maneira sistemática e otimizada. Em linguagens como Python, bibliotecas como NumPy oferecem suporte avançado para operações vetorizadas, reduzindo o tempo de processamento e tornando os cálculos mais rápidos e precisos. Além disso, matrizes são essenciais na representação de sistemas de equações lineares, transformações geométricas e processamento de imagens, tornando-se uma ferramenta indispensável para aplicações científicas e computacionais.

Considerações finais

As listas são uma das estruturas de dados mais versáteis e amplamente utilizadas na programação, desempenhando um papel essencial na organização e manipulação de conjuntos de informações. Neste capítulo, foram exploradas suas principais características, desde o armazenamento unidimensional e bidimensional até as operações básicas de inserção, remoção, busca e alteração de elementos. Além disso, foi demonstrada a relação entre listas e funções, destacando a importância da passagem por referência e cópia para evitar modificações não desejadas nos dados originais. Também foi abordada a eficiência de diferentes abordagens para a manipulação de listas, incluindo o uso de estruturas como array, que oferecem melhor desempenho em operações matemáticas e computacionais.

Compreender essas técnicas é fundamental para o desenvolvimento de programas mais eficientes, organizados e escaláveis, permitindo que

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.

os programadores aproveitem ao máximo os recursos disponíveis na linguagem Python. Ao dominar esses conceitos, é possível resolver problemas de forma mais estruturada, aplicando boas práticas de programação e facilitando a manutenção do código.

Referências

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação**: a construção de algoritmos e estruturas de dados com aplicações em Python. 4. ed. São Paulo: Bookman, 2022. *E-book*.

GUILHON, André *et al*. (org.). **Jornada Python**: uma jornada imersiva na aplicabilidade de uma das mais poderosas linguagens de programação do mundo. Rio de Janeiro: Brasport, 2022. *E-book*.

LAMBERT, Kenneth A. **Fundamentos de Python**: estruturas de dados. Porto Alegre: Cengage Learning, 2022. *E-book*.

Material para uso exclusivo de aluno matriculado em curso de Educação a Distância da Rede Senac EAD, da disciplina correspondente. Proibida a reprodução e o compartilhamento digital, sob as penas da Lei. © Editora Senac São Paulo.