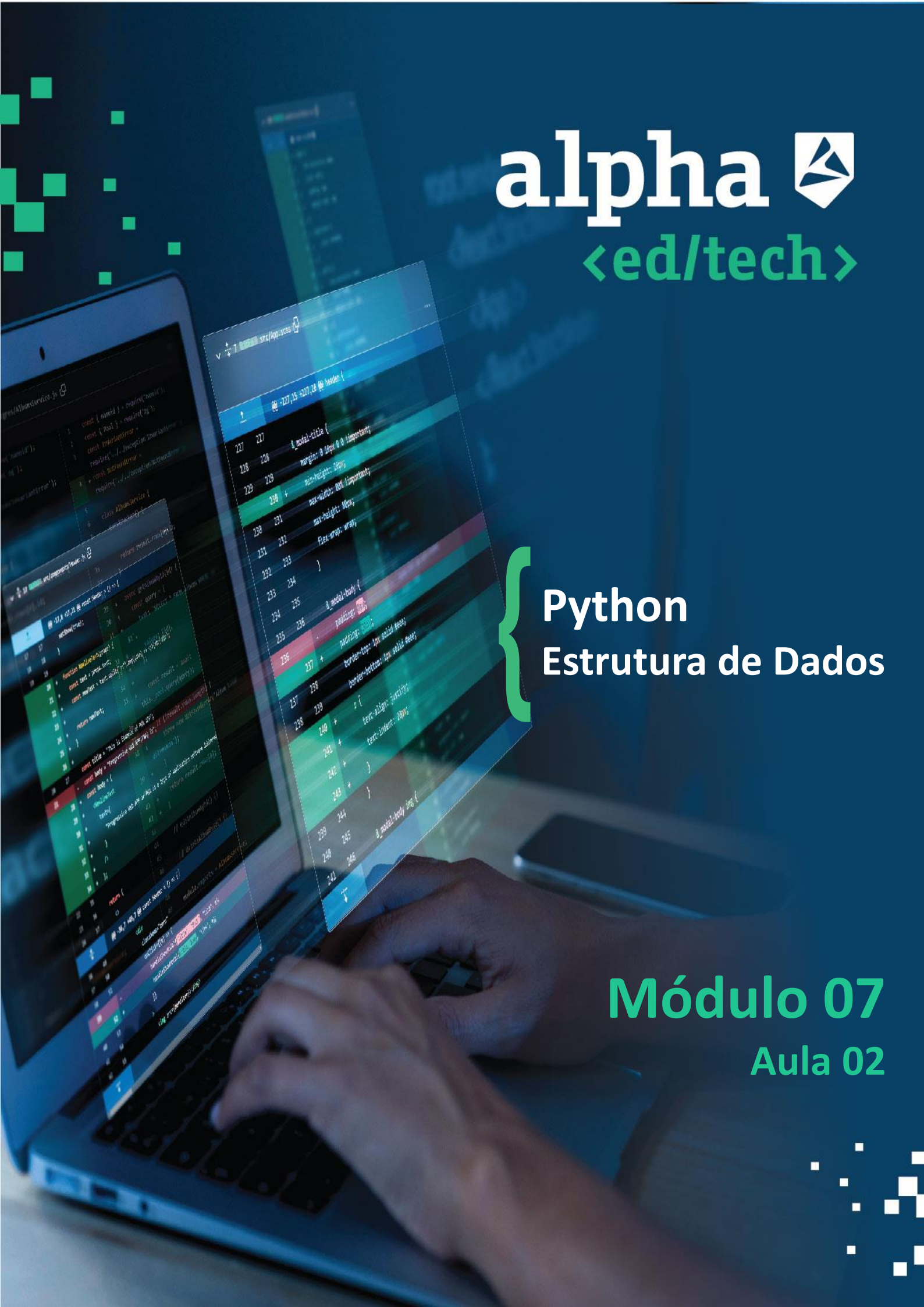

alpha
<ed/tech>

Python
Estrutura de Dados

Módulo 07
Aula 02

Estrutura de Dados

Arrays Não Ordenados

Um array não ordenado é uma coleção de elementos onde cada elemento pode ser identificado por um índice ou chave. No entanto, ao contrário dos arrays ordenados, os elementos em um array não ordenado não seguem uma sequência específica ou critério de ordenação.

Em Python, arrays não ordenados podem ser representados como listas (`'list'`), onde os elementos são armazenados na ordem em que são adicionados.

Características

- **Acesso por Índice:** Cada elemento em um array não ordenado pode ser acessado diretamente pelo seu índice. Em Python, isso é feito usando colchetes `[]`.

```
# Criando um array não ordenado
produtos = ["maçã", "banana", "laranja", "uva"]

# Acessando elementos por índice
primeiro_produto = produtos[0]  # Acessa o primeiro produto ("maçã")
terceiro_produto = produtos[2]  # Acessa o terceiro produto ("laranja")

print("Primeiro produto:", primeiro_produto)
print("Terceiro produto:", terceiro_produto)
```

- **Flexibilidade:** Não há restrições sobre onde um novo elemento pode ser inserido. Geralmente, é adicionado ao final do array, mas pode ser inserido em qualquer posição.

```
# Continuando com o array de produtos
produtos = ["maçã", "banana", "laranja", "uva"]

# Inserindo um novo produto no final
produtos.append("pera")

# Inserindo um novo produto em uma posição específica
produtos.insert(2, "abacaxi")  # Insere "abacaxi" na posição 2

print("Produtos após inserções:", produtos)
```

- **Eficiência de Inserção e Acesso:** Inserir um novo elemento (no final) e acessar qualquer elemento têm um tempo de execução constante, ou seja, O(1).

```
# Exemplo de inserção eficiente
produtos = ["maçã", "banana", "laranja", "uva"]

# Inserindo um novo produto (operação eficiente)
produtos.append("melancia")  # Tempo constante O(1)

# Exemplo de acesso eficiente
quarto_produto = produtos[3]  # Acesso em tempo constante O(1)

print("Quarto produto:", quarto_produto)
```

Operações Básicas

As principais operações que podem ser realizadas com um array não ordenado são:

1. **Inserção:** adicionar novos elementos no final do array, ou realizar a inserção em outras posições, o que demandará a necessidade de deslocamento dos elementos subsequentes.

2. **Consulta:** ler um elemento contido no array utilizando seu índice de posicionamento para obtenção do valor armazenado na lista.

3. **Atualização:** modificar um elemento existente no array, acessando-o através de seu índice e atribuindo um novo valor.

4. **Deleção:** remover um elemento do array caso ele esteja no final da lista, caso contrário será necessário realizar o deslocamento dos elementos subsequentes para preencher o espaço vazio gerado com a deleção de um elemento intermediário.

Vantagens

- Implementação simples
- Inserção e acesso rápidos

Desvantagens

- Deleção e busca de elementos pode ser ineficiente, pois podem exigir a verificação de cada elemento.
- Sem ordenação, não permitindo a realização de buscas com operações binárias.

A compreensão de **arrays não ordenados** é fundamental para entender como os dados são armazenados e manipulados na memória. Embora simples, eles formam a base para conceitos mais complexos em estruturas de dados e algoritmos.

Arrays Ordenados

Um array ordenado é uma coleção de elementos que são mantidos em uma ordem sequencial, geralmente em ordem crescente ou decrescente. A ordenação facilita operações como busca e inserção de elementos.

Características

- **Manutenção da Ordem:** em um array ordenado, os elementos são sempre armazenados em uma ordem específica (por exemplo, numérica ou alfabética).

```
def manter_ordem_apos_insercao(array, elemento):
    array.append(elemento)
    array.sort()

def manter_ordem_apos_remocao(array, elemento):
    array.remove(elemento)
    array.sort()

# Exemplo
numeros = [1, 3, 5, 7, 9]
print("Array original:", numeros)

manter_ordem_apos_insercao(numeros, 4)
print("Após inserção:", numeros)

manter_ordem_apos_remocao(numeros, 3)
print("Após remoção:", numeros)
```

- **Inserção Eficiente Mantendo a Ordem:** a inserção de um novo elemento requer que o array mantenha sua ordenação. Isso pode exigir a realocação de outros elementos.

```
def insercao_ordenada(array, elemento):
    index = 0
    while index < len(array) and array[index] < elemento:
        index += 1
    array.insert(index, elemento)

# Exemplo
numeros = [1, 2, 5, 6, 7]
print("Array original:", numeros)

insercao_ordenada(numeros, 4)
print("Após inserção ordenada:", numeros)
```

- **Busca Eficiente:** devido à ordenação, técnicas de busca eficientes, como a busca binária, podem ser utilizada para encontrar mais rapidamente, sem impacto no tempo de busca, os elementos procurados.

```
def busca_binaria(array, elemento):
    inicio = 0
    fim = len(array) - 1
    while inicio <= fim:
        meio = (inicio + fim) // 2
        if array[meio] == elemento:
            return meio
        elif array[meio] < elemento:
            inicio = meio + 1
        else:
            fim = meio - 1
    return -1

# Exemplo
numeros = [1, 2, 4, 5, 6, 7]
print("Array:", numeros)

indice = busca_binaria(numeros, 4)
print("Índice do elemento 4:", indice)
```

Um array ordenado é uma coleção de elementos que são mantidos em uma ordem sequencial, geralmente em ordem crescente ou decrescente. A ordenação facilita operações como busca e inserção de elementos.

Operações Básicas

Ao entender esses conceitos fundamentais, vocês estarão mais preparados para tomar decisões sobre como armazenar, acessar e manipular dados de maneira eficiente. Lembre-se de que a escolha entre um array não ordenado e um ordenado depende das operações que você espera realizar com maior frequência e do volume de dados que estará trabalhando.

1. Inserção: Adicionar um novo elemento no array de forma que a ordem seja mantida. Isso geralmente envolve encontrar a posição correta onde o novo elemento deve ser colocado para manter a ordem do array, o que pode exigir deslocar elementos existentes para abrir espaço para o novo elemento.

2. Busca: Encontrar um elemento específico no array. Graças à ordenação, é possível usar métodos de busca eficientes, como a busca binária, que divide repetidamente o array ao meio, descartando a metade onde o elemento com certeza não está, até encontrar o elemento ou concluir que ele não está presente.

3. Acesso: Acessar um elemento em uma posição específica. Assim como nos arrays não ordenados, o acesso é feito através do índice do elemento, sendo uma operação de tempo constante.

4. Atualização: Modificar um elemento existente. Isso envolve acessar o elemento pelo seu índice e substituí-lo por um novo valor. Se a atualização afetar a ordenação do array, é necessário ajustar a posição do elemento atualizado ou dos elementos circundantes para manter a ordem.

5. Deleção: Remover um elemento do array. Após a remoção, os elementos restantes podem precisar ser deslocados para preencher o espaço vago, e a ordem do array deve ser mantida.

6. Iterar: Percorrer cada elemento do array para realizar alguma operação, como imprimir os valores. A travessia em um array ordenado é feita da mesma forma que em um array não ordenado.

7. Ordenar: Embora o array já esteja ordenado, em algumas situações pode ser necessário reordenar os elementos (por exemplo, após uma série de atualizações). Isso pode ser feito usando algoritmos de ordenação como o merge sort, quick sort, entre outros.

Cada uma dessas operações desempenha um papel vital no gerenciamento eficiente de arrays ordenados, assegurando que os benefícios da ordenação - especialmente em termos de busca e acesso - sejam plenamente aproveitados.