

ALGORITMOS INTERESTELARES

< Explorando o Infinito da Programação >

Gilmar baracho

Introdução

Um algoritmo é uma sequência de instruções que são seguidas para resolver um problema ou realizar uma tarefa específica. Em programação, os algoritmos são essenciais, pois são a base de qualquer software. Eles definem os passos que o computador deve seguir para executar uma determinada ação, Vamos conhecer esse universo.



01

O QUE É UM ALGORITMO?

O QUE É UM ALGORITMO?

Um algoritmo é uma sequência bem definida de passos ou instruções que resolve um problema específico ou realiza uma tarefa particular. Pense em um algoritmo como uma receita de culinária: ele diz exatamente o que fazer e em que ordem para atingir um resultado desejado.

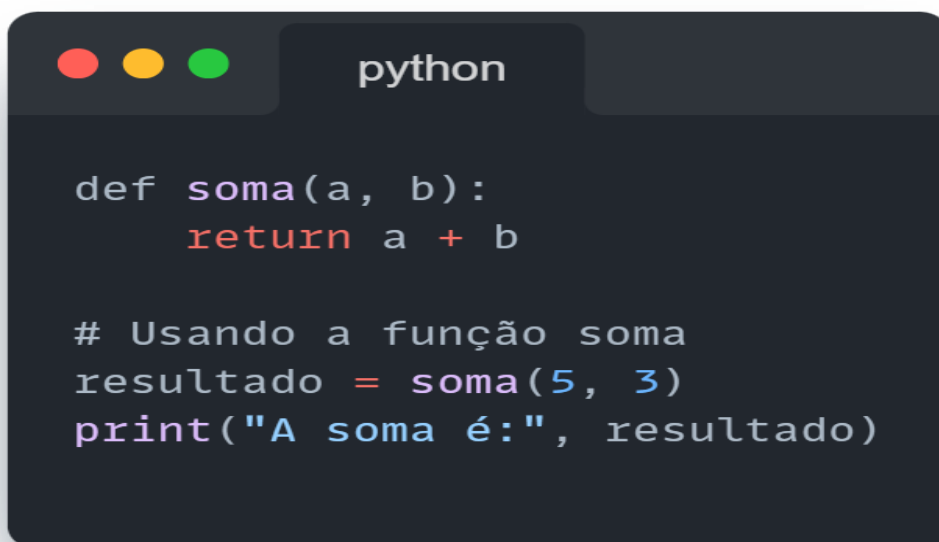
Exemplos de Algoritmos no Dia a Dia

- **Receitas de Culinária:** Uma lista de ingredientes e passos para preparar um prato.
- **Instruções de Montagem:** Passos detalhados para montar um móvel.
- **Rotas de Navegação:** Direções passo a passo para chegar a um destino.

EXEMPLOS SIMPLES DE ALGORITMOS

ALGORITMO DE SOMA DE DOIS NÚMEROS

Um exemplo simples de algoritmo é somar dois números. Vamos ver como isso pode ser feito em Python:

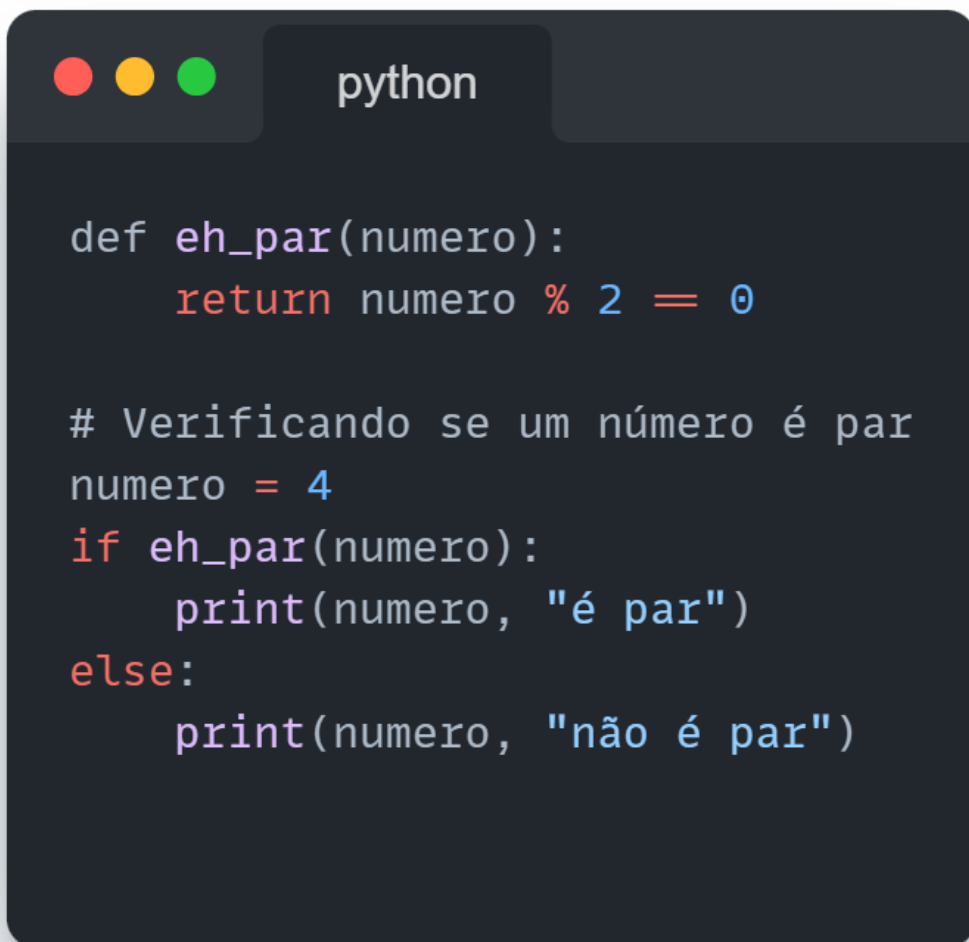


```
def soma(a, b):  
    return a + b  
  
# Usando a função soma  
resultado = soma(5, 3)  
print("A soma é:", resultado)
```

Neste exemplo, criamos uma função “soma” que recebe dois números como entrada e retorna a soma deles. Quando chamamos essa função com os números 5 e 3, o resultado é 8.

ALGORITMO DE VERIFICAÇÃO DE NÚMERO PAR

Outro exemplo comum é verificar se um número é par:

A dark-themed Python code editor window with a tab labeled 'python'. The code defines a function 'eh_par' that returns True if a number is even (modulo 2 equals 0). It then tests the function with the number 4, printing 'é par' if it is even and 'não é par' if it is not.

```
def eh_par(numero):  
    return numero % 2 == 0  
  
# Verificando se um número é par  
numero = 4  
if eh_par(numero):  
    print(numero, "é par")  
else:  
    print(numero, "não é par")
```

Aqui, a função “*eh-par*” verifica se o resto da divisão do número por 2 é zero. Se for, o número é par; caso contrário, é ímpar.

02

COMO USAR UM ALGORITMO

COMO USAR UM ALGORITMO.

Para utilizar um algoritmo, você precisa seguir as instruções passo a passo, sem pular ou alterar a ordem dos passos. Em programação, isso significa escrever um código que execute essas instruções em uma linguagem que o computador possa entender.

Estrutura de um Algoritmo

- **Entrada:** Os dados ou condições iniciais necessários.
- **Processamento:** Os passos ou instruções que transformam a entrada.
- **Saída:** O resultado final após o processamento.

Algoritmos em Contextos Reais

Algoritmo de Ordenação (Bubble Sort)

Ordenar uma lista de números é uma tarefa comum. Vamos usar o Bubble Sort, um algoritmo simples de ordenação:



```
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n-i-1):
            if lista[j] > lista[j+1]:
                lista[j], lista[j+1] = lista[j+1], lista[j]

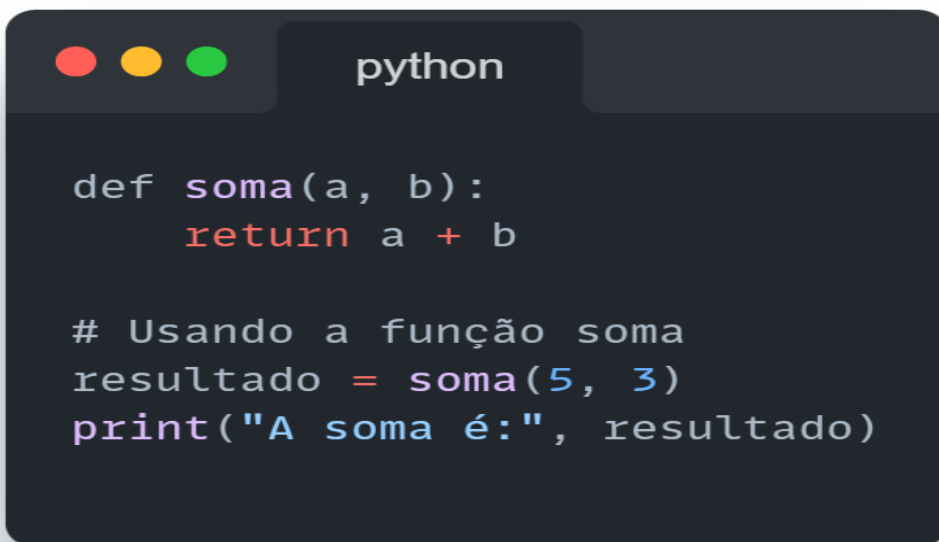
# Usando o Bubble Sort
numeros = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(numeros)
print("Lista ordenada:", numeros)
```

O Bubble Sort compara cada par de elementos adjacentes na lista e os troca se estiverem na ordem errada. Este processo é repetido até que a lista esteja ordenada.

EXEMPLOS SIMPLES DE ALGORITMOS

ALGORITMO DE SOMA DE DOIS NÚMEROS

Um exemplo simples de algoritmo é somar dois números. Vamos ver como isso pode ser feito em Python:



```
def soma(a, b):  
    return a + b  
  
# Usando a função soma  
resultado = soma(5, 3)  
print("A soma é:", resultado)
```

Neste exemplo, criamos uma função “soma” que recebe dois números como entrada e retorna a soma deles. Quando chamamos essa função com os números 5 e 3, o resultado é 8.

Algoritmo de Busca (Busca Binária)

Buscar um valor em uma lista ordenada é mais eficiente com a Busca Binária:

```
python

def busca_binaria(lista, valor):
    esquerda = 0
    direita = len(lista) - 1
    while esquerda <= direita:
        meio = (esquerda + direita) // 2
        if lista[meio] == valor:
            return meio
        elif lista[meio] < valor:
            esquerda = meio + 1
        else:
            direita = meio - 1
    return -1

# Usando a Busca Binária
numeros = [11, 12, 22, 25, 34, 64, 90]
valor = 25
resultado = busca_binaria(numeros, valor)
if resultado != -1:
    print("Valor encontrado na posição:", resultado)
else:
    print("Valor não encontrado")
```

A Busca Binária divide repetidamente a lista ao meio para encontrar o valor desejado, tornando-a muito mais eficiente que a busca linear, especialmente em listas grandes.

03

PRÁTICA E RECURSOS ADICIONAIS

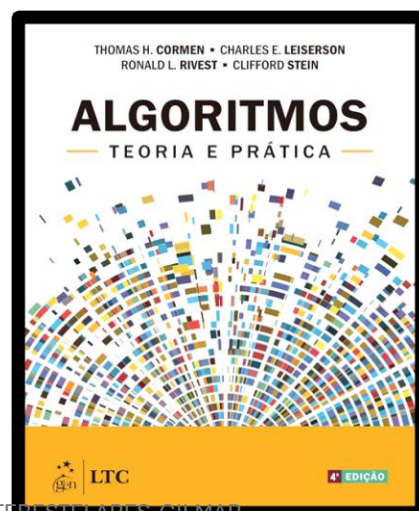
PRÁTICA E RECURSOS ADICIONAIS

Para se tornar proficiente em algoritmos, a prática é essencial. Aqui estão algumas dicas e recursos:

Pratique Regularmente: Resolva problemas em plataformas como LeetCode, HackerRank e Codewars.

Estude Estruturas de Dados: Compreenda arrays, listas ligadas, pilhas, filas, árvores e grafos.

Leia Livros: "Algoritmos: Teoria e Prática" de Thomas H. Cormen é uma excelente referência.



04

CONCLUSÃO

CONCLUSÃO.

Compreender algoritmos é essencial para qualquer programador. Eles não apenas resolvem problemas específicos, mas também aprimoram o pensamento lógico e a capacidade de abordar desafios complexos de maneira estruturada. Os algoritmos são a base da programação, e dominar sua implementação é crucial. A prática constante e o estudo aprofundado de diversos tipos de algoritmos são fundamentais para desenvolver a habilidade de resolver problemas de forma eficiente e eficaz.

Comece com os conceitos básicos, pratique regularmente e, em pouco tempo, você estará confortável com algoritmos mais avançados e suas inúmeras aplicações no mundo real.

Essa jornada de aprendizado contínuo não só consolidará seu conhecimento, mas também abrirá portas para soluções inovadoras e criativas em suas futuras empreitadas na programação.

AGRADECIMENTO.

Gostaria de expressar minha profunda gratidão à Digital Innovation One (DIO) pela oportunidade de participar deste incrível bootcamp. A experiência foi verdadeiramente transformadora e enriquecedora.

Em especial, agradeço aos instrutores dedicados que compartilharam seu conhecimento e experiência de forma tão generosa e acessível. Cada aula foi uma oportunidade valiosa de aprendizado e crescimento.

Finalmente, um agradecimento especial à equipe de organização da DIO por proporcionar um ambiente de aprendizado tão dinâmico e estimulante. O compromisso de vocês com a educação e o desenvolvimento profissional é inspirador e faz uma diferença real na vida de muitos.

Com gratidão,
Gilmar Baracho