Introdução à Programação Estruturada

Professor Fernando costa Leite

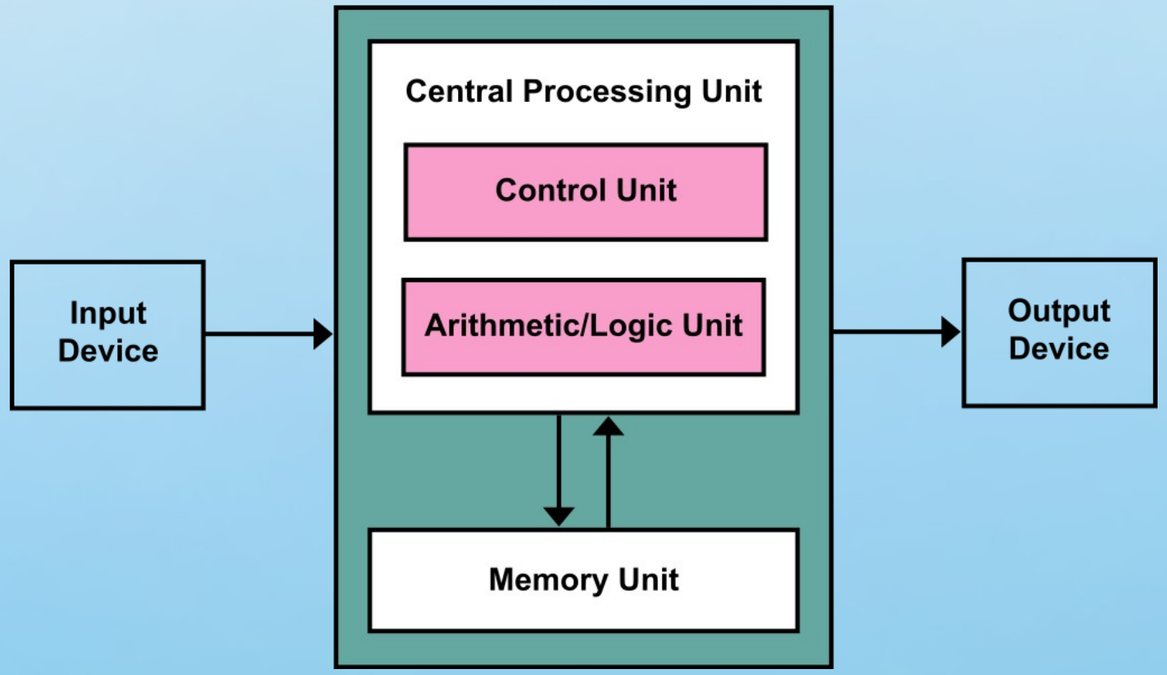
Contextualização

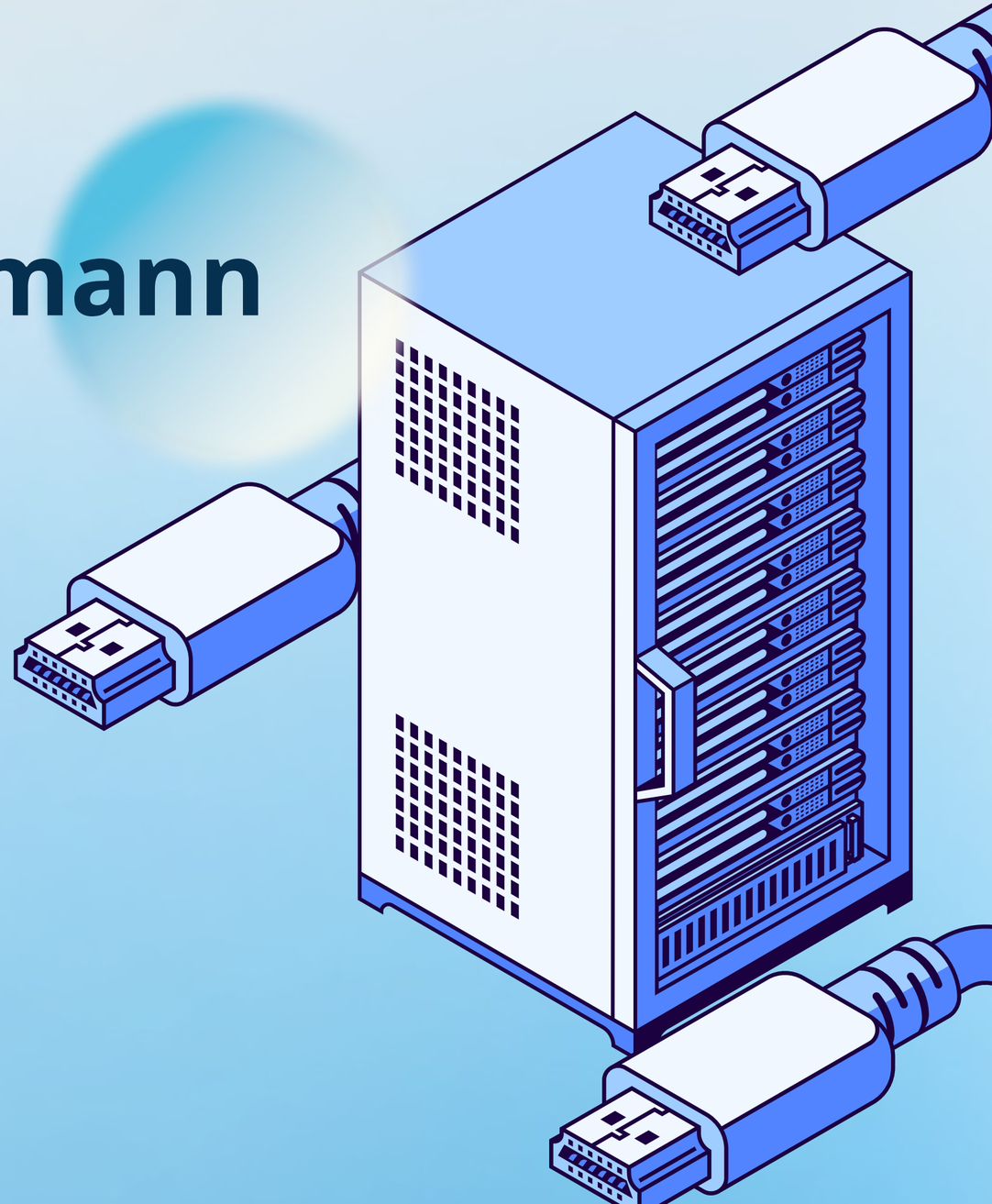
A evolução da programação foi marcada por grandes mudanças na forma como as máquinas são programadas, com o desenvolvimento de novos paradigmas e ferramentas. Para entender a transição da programação é essencial conhecer a história dos primeiros passos da programação.

A história da computação começa com a Máquina de Turing, proposta por Alan Turing. Trata-se de um modelo teórico de computador que descreve como um conjunto de instruções pode ser usado para resolver problemas matemáticos. Este conceito influenciou a criação dos primeiros computadores.

Nos anos 1940, o matemático John von Neumann apresentou o modelo de arquitetura de computadores, que viria a ser a base da computação moderna. O modelo de von Neumann descreve um computador com três componentes principais:

Arquitetura de Von Neumann





Contextualização

Primeiros Passos: Cartões Perfurados e Linguagens de Baixo Nível

Cartões Perfurados (Década de 1950):
Nos primeiros dias da computação, os programadores usavam cartões perfurados para introduzir dados e instruções nos computadores. Cada cartão perfurado representava uma linha de código.

Linguagens de Programação de Baixo Nível (Assembly):
Com o tempo, surgiram as linguagens de baixo nível como o assembly, que se aproximavam mais do código de máquina, mas ainda exigiam que o programador controlasse diretamente os detalhes do hardware, como endereços de memória e registradores. A programação em assembly era extremamente detalhada e trabalhosa, sendo adequada apenas para programadores experientes.

Contextualização

O Surgimento das Linguagens de Alto Nível e a Crise do Software

A Crise do Software (Década de 1960):
À medida que os computadores se tornaram mais poderosos e acessíveis, os programas ficaram mais complexos. No entanto, a programação ainda era extremamente difícil e exigia um conhecimento profundo da arquitetura do computador. A crise do software ocorreu nos anos 1960, quando se percebeu que os métodos tradicionais de programação não eram suficientes para lidar com o aumento da complexidade dos sistemas de software.

- **Problema 1: Os programas estavam ficando cada vez mais difíceis de entender e manter, principalmente devido à falta de abstração.**
- **Problema 2: A quantidade de erros nos programas aumentava conforme os sistemas se tornavam mais complexos.**
- **Problema 3: A manutenção de grandes sistemas de software era praticamente impossível devido à falta de organização no código.**

Esse cenário levou ao desenvolvimento de novas abordagens para melhorar a qualidade e a manutenção dos programas.

Contextualização

A Programação Não Estruturada (Uso do "goto")

Antes da programação estruturada, a principal maneira de controlar o fluxo de um programa era através do comando goto. O goto permitia que o controle do programa saltasse de uma parte do código para outra, sem seguir uma ordem sequencial definida. Isso resultava em programas caóticos, difíceis de ler e manter. Essa prática é conhecida como programação não estruturada.

```c
#include <stdio.h>

int main() {
    int i = 0;

start:
    if (i >= 5) goto end;
    printf("%d\n", i);
    i++;
    goto start;

end:
    return 0;
}
```

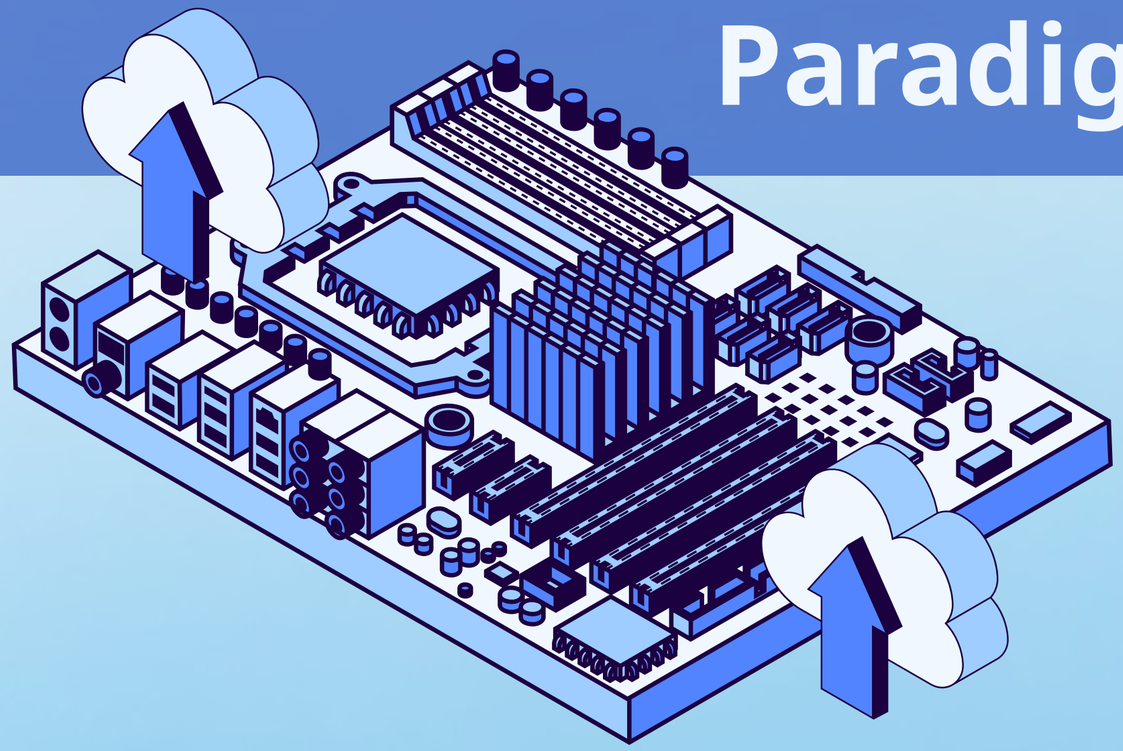
Contextualização

Edsger Dijkstra e a Crítica ao "goto"

A virada para a programação estruturada se deu principalmente através do trabalho do cientista da computação Edsger Dijkstra, que em 1968 escreveu o famoso artigo "Go To Statement Considered Harmful". Nesse artigo, Dijkstra argumentou que o uso do goto tornava os programas confusos e propensos a erros. Ele sugeriu que a programação deveria seguir estruturas claras e lógicas, como sequência, seleção e repetição, para organizar o fluxo de controle de maneira mais eficiente.

https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf

Paradigmas de programação

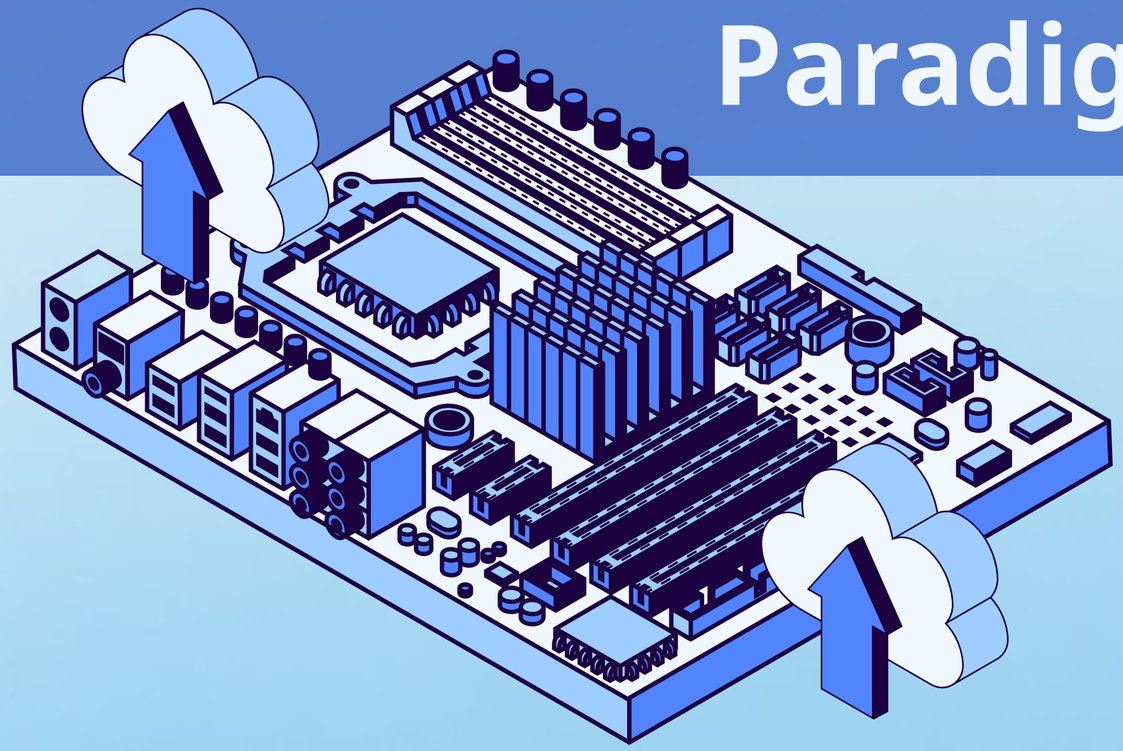


Conceito

O conceito de paradigma de programação pode ser definido como formas de abordar e resolver problemas ao escrever código. Eles são como conjuntos de regras que guiam os desenvolvedores.

Os paradigmas de programação podem ser assim classificados:

Paradigmas de programação



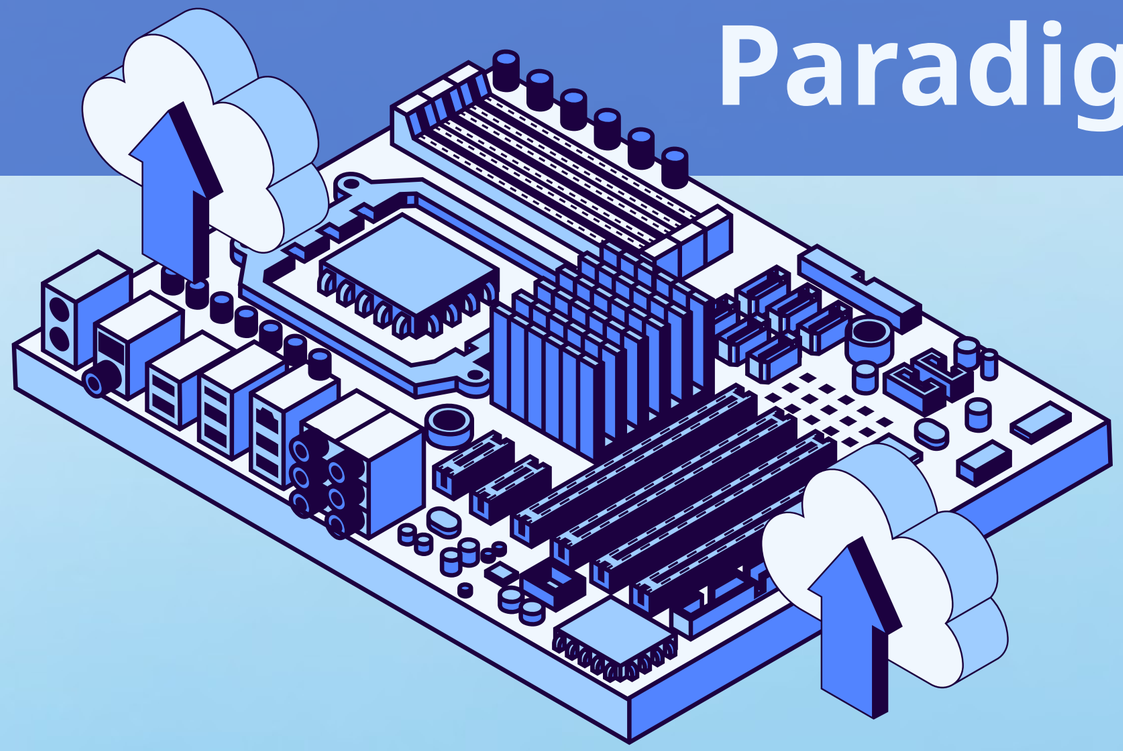
Imperativo

Dividido em estruturado e não-estruturado. Um programa é uma sequência de comandos que alteram o estado atual do sistema até atingir um estado final.

Programação Orientada a Objetos (POO)

O paradigma orientado a objetos é um paradigma de programação centrado na ideia de 'objetos'. Cada objeto, uma instância de uma classe, encapsula dados (atributos) e comportamentos (métodos), permitindo interações entre eles para executar tarefas. Essa abordagem facilita a modelagem de sistemas complexos ao representar entidades do mundo real com características e ações específicas.

Paradigmas de programação



Paradigma Funcional

A programação funcional se baseia no uso de funções como peças fundamentais do código. Ela permite que as funções sejam tratadas de maneira flexível, como blocos de construção que podem ser usados, passados e armazenados em diferentes partes do programa.

Paradigma Declarativo

O paradigma declarativo é uma um paradigma de programação que foca em descrever o que deve ser alcançado, em vez de como alcançá-lo. Em vez de fornecer uma sequência de instruções detalhadas, o programador declara as propriedades e relações que o resultado final deve ter.

Programação Estruturada

Também conhecida como programação estruturada, o paradigma estruturado é um estilo dentro do paradigma imperativo que se concentra em organizar o código fonte em uma sequência de procedimentos ou funções. Cada procedimento é uma série de comandos que realiza uma tarefa específica.

Conforme Robert C. Martin (Clean Architecture a craftsman guide to software structure and design, 2019, p. 28, 29), a programação estruturada possibilita a contrução de programas com apenas três estruturas:

- **sequência:** Execução linear do código.
- **seleção:** Tomada de decisão com base em condições (ex.: if-else).
- **iteração:** Execução repetitiva de um bloco de código (ex.: for, while).

Programação Estruturada

Modularização:

À medida que os programas vão se tornando maiores e mais complexos, é possível simplificar e melhorar a clareza dividindo o programa em partes menores, chamadas subprogramas.
Um subprograma, é um nome dado a um trecho de um programa mais complexo e que, em geral, encerra em si próprio um pedaço da solução de um problema maior (o programa a que ele está subordinado). São sinônimos usados na engenharia de software para o conceito de subprograma: procedimento, função, módulo (estrutura modular), métodos (orientação a objetos) e subrotina.

Programação Estruturada

- Esse novo paradigma trouxe mais clareza ao desenvolvimento de software e ajudou a resolver os problemas da crise do software, permitindo a criação de programas mais legíveis e fáceis de manter.

- Com a chegada das linguagens como C, Pascal e Algol, que foram desenvolvidas com base nos princípios da programação estruturada, o uso do goto diminuiu drasticamente, e o paradigma estruturado tornou-se o padrão. A programação estruturada ajudou a melhorar a organização e a legibilidade do código, tornando mais fácil escrever e modificar grandes sistemas de software.

Vantagens

- **Legibilidade**

  Programas mais fáceis de entender.

- **Manutenção**

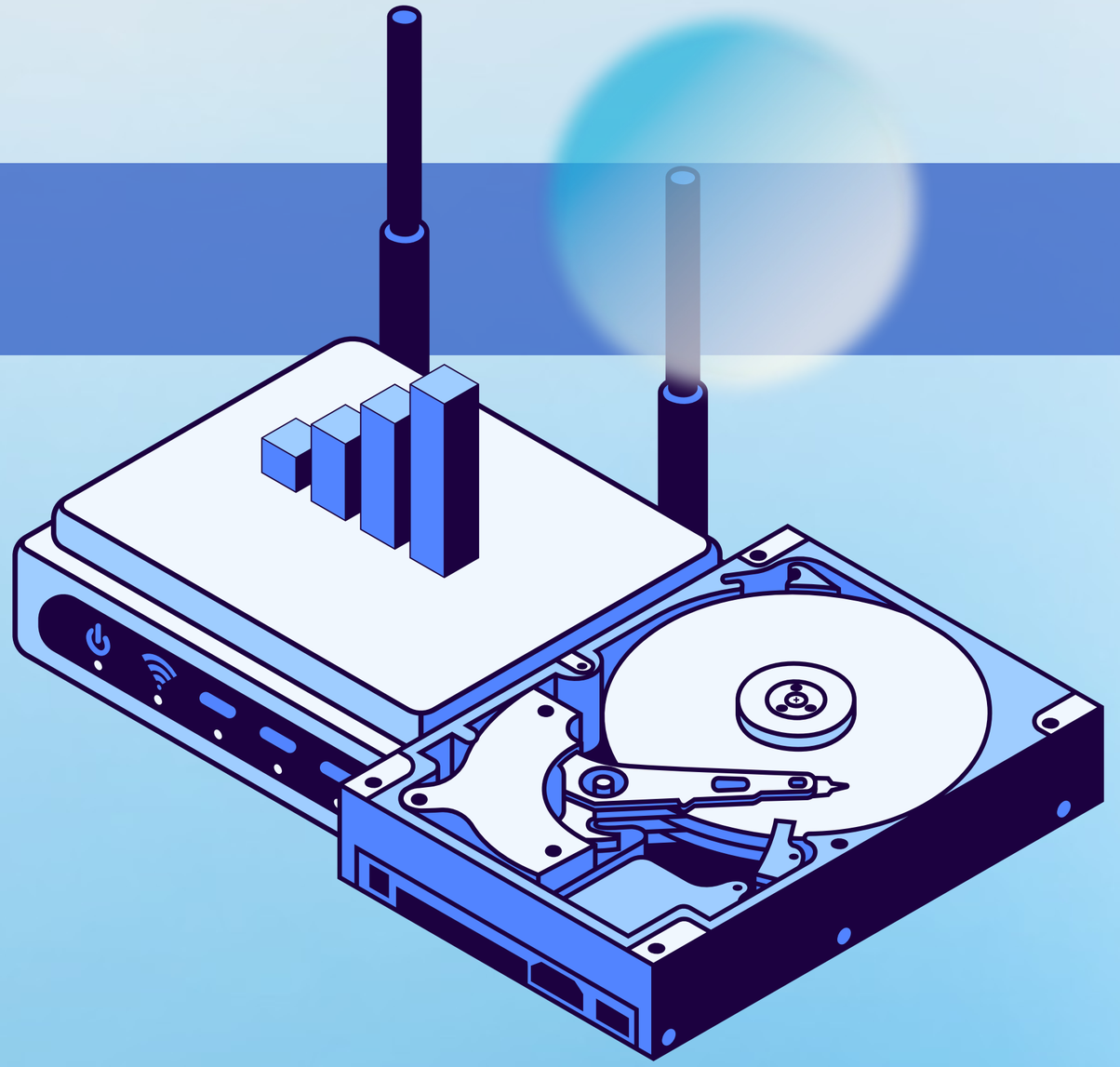
  Facilidade de manutenção e modificação do código.

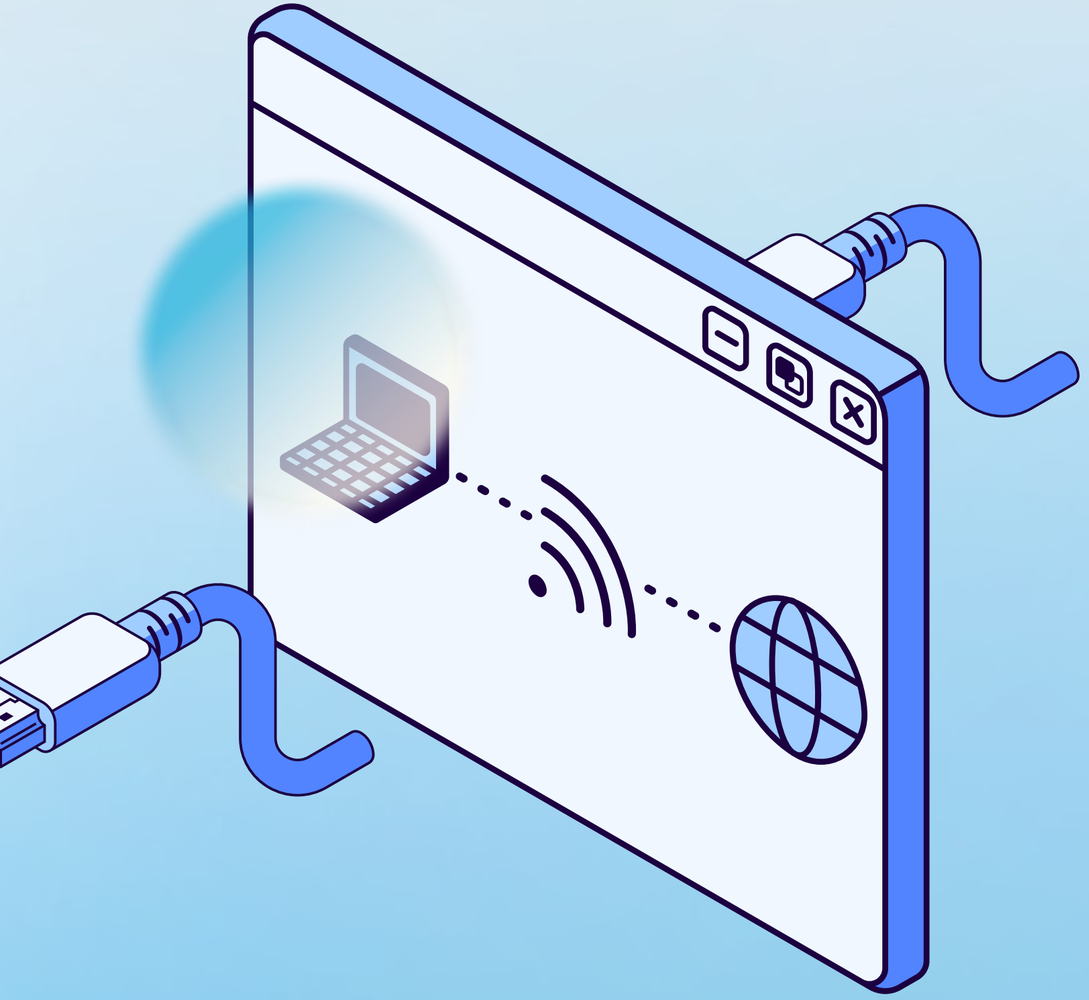
- **Depuração**

  Identificação e correção de erros com maior facilidade.

- **Modularidade**

  Permite a organização do código em funções e módulos.

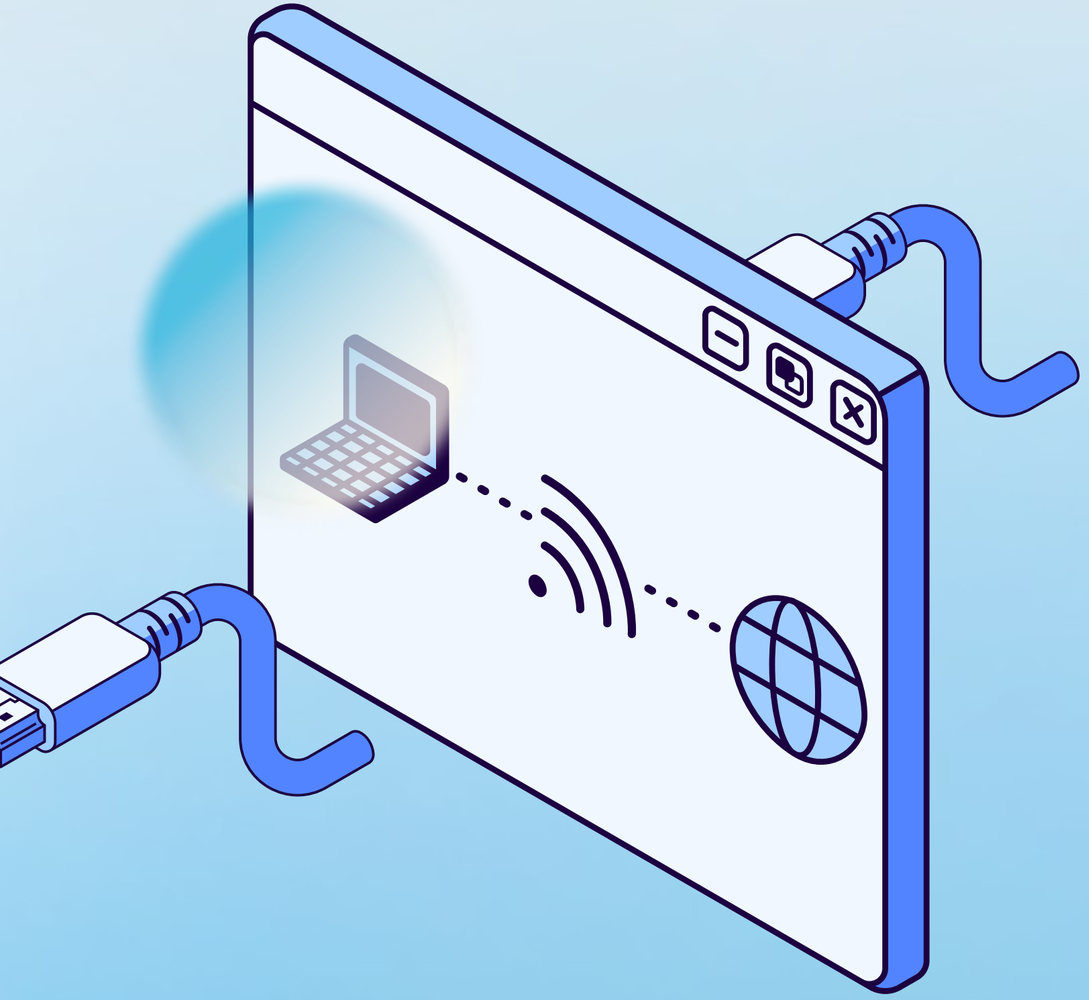




Python - linguagem adotada neste curso

- Python é uma linguagem de programação de alto nível, interpretada, de tipagem dinâmica e forte.

- Criada por Guido van Rossum e lançada em 1991, Python é famosa por sua simplicidade e legibilidade.
É utilizada para desenvolvimento web, automação, análise de dados, inteligência artificial, entre outros.



Python - Vantagens

- Sintaxe simples e clara.
- Bibliotecas poderosas (Pandas, Numpy, Django).
- Comunidade ativa e grande base de usuários.
- Portabilidade (executa em diferentes sistemas operacionais).
- Versatilidade (usada em diversas áreas).

Thank You!