PROGRAMAÇÃO

INTRODUÇÃO À PROGRAMAÇÃO

1 - ALGORITMOS

Essencialmente, um algoritmo é um conjunto sequencial de regras/intruções para resolver um problema especifico ou atingir um objeito. Em termos simples, é um conjunto sistematico de intruções que levam a um resultado desejado.
É fundamental entender que um algoritmo é justificado pelo resultado que o mesmo pretende chegar. Portanto, deve ter um objetivo claro e especifico
Algoritmos devem ser:

- **Bem definidos:** Cada etapa de um algoritmo deve ser clara e sem ambiguidades
- **Logicamente ordenados:** As etapas do algoritmo devem ser organizadas em uma sequência lógica e ordenada. A ordem das instruções pode afetar o resultado final.
- **Finitos:** Um algoritmo deve terminar após um número finito de etapas.
- **Eficientes:** Um bom algoritmo é construído com o menor número de recursos possível, executando sua função de forma eficiente.

O USO DE ALGORITMOS NA PROGRAMAÇÃO

Os algoritmos são fundamentais na programação, eles desempenham um papel crucial em praticamente todos os aspectos do desenvolvimento de software. **Todo programa é um algoritmo.** Quando você tem um programa funcional, ele implementa um algoritmo específico. O programa fornece uma realização concreta do algoritmo em uma linguagem de programação específica.

Mas observe que **nem todo algoritmo é um programa**; Os algoritmos podem existir como conceitos abstratos sem terem sido traduzidos para uma linguagem de programação específica. Eles podem ser representados em pseudocódigo ou fluxogramas, por exemplo, sem serem diretamente executáveis por um computador.

2 – CONCEITOS DE PROGRAMAÇÃO

- O QUE É A PROGRAMAÇÃO DE COMPUTADORES?

A Programação é o processo de abstrair ideias e criar um conjunto de instruções que um computador pode entender e executar para executar tarefas específicas. Em outras palavras, envolve escrever código que instrui um computador sobre como executar operações.

Esse processo representa a transformação de conceitos abstratos em ações concretas por meio de instruções precisas e sequenciais, permitindo que um computador execute essas tarefas de forma eficaz. Como os computadores carecem de compreensão inerente e só podem seguir instruções explícitas, os **programas devem ser extremamente detalhados e livres de ambiguidades.**

- A LÓGICA POR TRÁS DE UM PROGRAMA

A lógica por trás de um programa está enraizada nos princípios e técnicas da lógica de programação que orientam sua criação. Envolve o uso de um conjunto de conceitos para estruturar e gerenciar como um programa opera. Os computadores executam instruções de forma sequencial e lógica.
A lógica de um programa é construída usando elementos como **variáveis, ponteiros, controles de fluxo condicionais, funções, controle de fluxo (loops), entradas e saídas.** Cada um desses componentes desempenha um papel crucial para garantir que o programa funcione de maneira correta e eficiente.

- O QUE É UMA LINGUAGEM DE PROGRAMAÇÃO?

Antes de analisar uma linguagem de programação, vamos considerar nossa própria linguagem natural. Uma linguagem natural é usada para a comunicação humana cotidiana e muitas vezes contém ambiguidades e nuances.

Ambiguidade refere-se a situações em que uma palavra, frase ou declaração pode ser entendida de várias maneiras, levando a confusão ou má interpretação. Em contraste, **uma linguagem de programação é considerada uma linguagem formal** porque é projetada para comunicação precisa e não ambigua entre humanos e computadores.

As linguagens de programação são estruturadas com base em alguns conceitos matemáticos, garantindo que as instruções sejam claras e executáveis. Tanto as linguagens naturais quanto as formais têm **sintaxe** e **semântica**. Ao contrário das linguagens naturais, as linguagens de programação são criadas para minimizar a ambiguidade, garantindo que as instruções sejam interpretadas e executadas de forma consistente pelos computadores.

- PROGRAMAÇÃO EM ALTO NÍVEL e PROGRAMAÇÃO EM BAIXO NÍVEL

Linguagens de alto nível são um subconjunto de linguagens formais projetadas para serem mais fáceis de usar para programadores. Eles abstraem grande parte da complexidade do hardware do computador e das operações de baixo nível. As linguagens de alto nível são caracterizadas por sua sintaxe e legibilidade mais simples em comparação com as linguagens de nível inferior, que exigem gerenciamento detalhado da memória e dos recursos do sistema.

Linguagens de baixo nível são aquelas que estão mais próximas da linguagem de máquina. Eles são muito mais complexos de usar e entender porque têm um grau de abstração menor do que as linguagens de alto nível, mas oferecem um controle mais preciso sobre o hardware. Um bom exemplo de linguagem de baixo nível é o **Assembly**. Essa linguagem usa **mnemônicos** (representações textuais) para representar instruções de máquina.

Aqui está um pequeno exemplo de um código em assembly que inicializa dois registradores com os valores 5 e 10, adiciona esses valores e armazena o resultado no registrador AX :

```
MOV AX, 5 ; Move 5 para o registrador AX
MOV BX, 10 ; Move 10 para o registrador BX
BX ADD AX, BX ; Adiciona o valor de BX para o valor de AX
```

- SINTAXE

A **sintaxe** define como os elementos de código devem ser estruturados para que o computador os entenda e execute corretamente. Ele especifica as regras e convenções para organizar o código, incluindo como declarar variáveis, criar loops, definir funções e muito mais.

Cada linguagem de programação tem sua própria sintaxe, que determina o formato específico e a disposição dos elementos de código. Erros de sintaxe ocorrem quando o código se desvia dessas regras. Tais erros são detectados pelo compilador ou interpretador durante a fase de compilação ou interpretação do código, antes que o programa seja executado.

Identificar e corrigir erros de sintaxe é crucial para garantir que o código possa ser executado corretamente pelo computador. Aqui está um exemplo de duas linguagens de programação que fazem a mesma operação de multiplicação, Python e C:

- Python

```
def multiply(a, b):
   return a * b

num1 = 5
num2 = 10
result = multiply(num1, num2)

print(result)

#output: 50
```

- C

```
#include <stdio.h>

int multiply(int a, int b) {
    return a * b;
}

int main() {
    int num1 = 5;
    int num2 = 10;
    int result = multiply(num1, num2);

    printf("%d\n", result);

    return 0;
}

//output : 50
```

Observe como cada sintaxe de linguagem é única.

- SEMÂNTICA

A **semântica** das linguagens de programação refere-se ao significado ou interpretação das instruções no código-fonte. É sobre como as instruções e estruturas de controle interagem para executar uma tarefa. Em outras palavras, a semântica define o comportamento e a lógica que o código representa.

A semântica ajuda a responder a perguntas como: "O que acontece quando esse código produz esse resultado?" Sem uma definição clara de semântica, seria impossível garantir que o código se comportasse de maneira consistente e previsível.

Erros semânticos resultam em resultados diferentes do esperado e geralmente estão relacionados a erros na lógica do programa. Aqui está um exemplo de erro semântico em um programa de soma simples em Python:

```
num1 = 5
num2 = 7
sum = num1 - num2

print(sum)

#output: -2
```

Nota: o código respeita as regras de sintaxe do python, o programa irá executar normalmente, mas o resultado é diferente do sperado, então há um erro semântico.

- O QUE É UM PARADIGMA DE PROGRAMAÇÃO?

Um paradigma é um conceito que define um exemplo ou modelo típico de algo. Representa um padrão a ser seguido, incluindo métodos e valores que são concebidos como modelo. No contexto da programação, um paradigma de programação pode ser entendido como um tipo de estrutura à qual um programa deve aderir.

A escolha do paradigma para um programa deve ser determinada pela definição do problema. Os paradigmas de programação oferecem técnicas apropriadas para estruturar e escrever programas, influenciando a forma como o código é organizado e executado. Eles fornecem diferentes estilos e técnicas para enfrentar os desafios de programação e podem impactar significativamente o design e a funcionalidade do software.

Algumas linguagens de programação são projetadas para suportar um único paradigma, com foco em um estilo específico de programação. Em contraste, muitas linguagens modernas são multiparadigmas, o que significa que suportam vários paradigmas e permitem que os desenvolvedores escolham a abordagem mais adequada para suas necessidades.

A lista de paradigmas da programação é enorme, alguns exemplos de paradigmas são : **Paradgimas Orientado a Objetos**, **Paradigma Funcional**, **Paradigma Imperativo**

3 – COMO COMPUTADORES ENTENDEM O CÓDIGO?

- CÓDIGO DE MÁQUINA

O código de máquina é o nível mais baixo de código executável por um computador. Consiste em uma sequência de instruções codificadas em **linguagem binária (0s e 1s)** que a CPU pode interpretar e executar diretamente. Cada instrução de código de máquina representa uma operação fundamental que o processador pode executar, como adição, subtração, movimentação de dados e varias outras. O código de máquina é específico para a arquitetura do processador, o que significa que o código de máquina para um tipo de CPU, como a arquitetura Intel x86, pode não ser executável em outra arquitetura, como ARM.

Quando um programa escrito em uma linguagem de alto nível é compilado ou interpretado, o processo de tradução converte o código-fonte em código de máquina. Este código de máquina é armazenado em um arquivo executável que a CPU pode executar diretamente.

Código de máquina se parece com:

```
0100 0001 0010 0001 0011 0000 0100 0000...
```

Código de máquina é explorado com maior profundidade no contexto de Sistemas Digitais, Programação em Baixo Nível e Arquitetura e Organização de Computadores

- LINGUAGENS COMPILADAS E LINGUAGENS INTERPRETADAS

Como os computadores operam exclusivamente com instruções binárias, um código escrito em uma linguagem de alto nível precisa passar por um processo de tradução para se tornar executável pelo microprocessador. Esse processo converte a linguagem de alto nível em código de máquina. A tradução pode ser realizada através de dois métodos principais: **compiladores** e **interpretadores**.

Em linguagens compiladas, o código-fonte é traduzido inteiramente em código de máquina antes da execução. Isso é feito por um programa chamado **compilador**. O compilador gera um arquivo executável a partir do código-fonte. O programa compilado não requer o código-fonte original para execução, pois a tradução já foi concluída durante a compilação.

Em linguagens interpretadas, o código é lido e executado linha por linha ou bloco por bloco durante a execução. Cada instrução é traduzida em código de máquina e executada imediatamente. Esse processo é tratado por um programa chamado **interpretador**. O código-fonte é necessário para a execução, pois a tradução ocorre em tempo real enquanto o programa está em execução.

Algumas linguagens de programação podem ser apenas compiladas, outras apenas interpretadas, e muitas linguagens modernas adotam uma **abordagem híbrida,** combinando o uso de compiladores e interpretadores.

4 – FERRAMENTAS E RECURSOS PARA DEVs

- IDEs E EDITORES DE TEXTO

Uma IDE é um software que fornece um conjunto completo de ferramentas para o desenvolvimento de software em um ambiente integrado. Combina várias funcionalidades numa única interface, facilitando o processo de aquisição, depuração e gestão de projetos. Exemplos de IDEs populares incluem **IntelliJ IDEA, Eclipse, Visual Studio, NetBeans, PyCharm (Python), Android Studio (Android) e Xcode (macOS e iOS).**

Um editor de texto é uma ferramenta mais simples usada para escrever e editar o código-fonte. Embora não ofereçam o mesmo conjunto abrangente de recursos de um IDE, muitos editores de texto possuem recursos úteis para programação, como: **Visual Studio Code, Sublime Text, Atom, Notepad++ e Brackets.**

Enquanto os IDEs fornecem uma solução completa e integrada para desenvolvimento de software com uma variedade de ferramentas em um ambiente, os editores de texto são mais simples e mais focados na edição de código. Ambos são valiosos e nenhum é inerentemente melhor que o outro, a praticidade de usar um IDE ou um editor de texto depende das necessidades do projeto e das preferências do desenvolvedor. Ambas tem suporte para **plugins**, permitindo que os desenvolvedores personalizem seu ambiente de desenvolvimento

- FRAMEWORKS

Frameworks são estruturas ou plataformas que fornecem uma base para o desenvolvimento de software. É como uma planta para desenvolver sofware que pode ser adaptada a diferentes contextos, elas incluem um conjunto de ferramentas, bibliotecas e melhores práticas que facilitam a construção de aplicações, permitindo que os desenvolvedores se concentrem nas funcionalidades específicas de suas aplicações, em vez de se preocuparem com detalhes básicos da implementação. Os frameworks podem ser usados nas diferentes áreas de programação, alguns exemplos:

- Frameworks como **Django (Python)**, **.NET Framework (C#), Ruby on Rails (Ruby)** e **Express (Node.js)** ajudam a construir aplicações web de forma mais eficiente.
- Frameworks como **React**, **Angular** e **Vue.js** auxiliam na criação de interfaces de usuário interativas.
- Frameworks de Mobile: Como **React Native, Flutter e Xamarin(C#)**

- SISTEMAS DE CONTROLE DE VERSÃO (VCS)

Já imaginou ter que salvar manualmente todas as atualizações de um projeto em arquivos locais separados? Isso seria inviável, especialmente à medida que a complexidade dos projetos aumenta.

Os sistemas de controle de versão (**VCS**) são ferramentas essenciais para desenvolvedores, permitindo que eles rastreiem e gerenciem alterações no código-fonte ao longo do tempo. Eles ajudam a manter um histórico detalhado de alterações, facilitam a colaboração entre desenvolvedores e garantem que diferentes versões de software possam ser gerenciadas e revertidas conforme necessário.

Além disso, o VCS permite reverter arquivos específicos ou até mesmo todo o projeto para um estado anterior, além de exibir quem fez cada alteração. Embora existam vários sistemas de controle de versão disponíveis, o mais conhecido e amplamente utilizado é o **Git**. A escolha de um VCS específico depende das necessidades de cada projeto.

5 – EXERCICIOS

1. Explique o que é um algoritmo
2. Dê dois exemplos de situações do dia a dia que podem ser resolvidas com um algoritmo simples.
3. Escreva um algoritmo em pseudocódigo que receba um número inteiro e determine se ele é par ou ímpar.
4. Explique o que é programação de computadores e a sua importância.
5. Explique o que é lógica de programação e como ela influencia a estrutura de um programa. Dê exemplos de componentes que fazem parte dessa lógica.
6. Compare e contraste linguagens de alto nível e baixo nível, citando exemplos de cada tipo e suas principais características.
7. Defina sintaxe e semântica em programação e explique como elas se relacionam.
8. Crie um exemplo de código de sua preferencia com um erro de sintaxe e um exemplo com um erro semântico.
9. O que é código de máquina e como ele se relaciona com o código escrito em linguagens de alto nível?
10. O que é um paradigma de programação? Dê exemplos de pelo menos três paradigmas diferentes e explique brevemente como eles influenciam o desenvolvimento de software.
11. O que é um framework?
12. Compare uma IDE e um editor de texto em termos de funcionalidades e uso. Quando você recomendaria o uso de um em vez do outro?
13. O que são sistemas de controle de versão? Explique como um VCS é utilizado no desenvolvimento de software e cite pelo menos duas vantagens de usar um VCS.
14. Explique a diferença entre linguagens compiladas e interpretadas. Dê exemplos de linguagens que se enquadram em cada categoria.