**Introdução à linguagens de programação**

Marcos históricos importantes:

1949: Primeira linguagem de montagem de máquina (Assembly).

Anos 50: COBOL, Fortran, Lisp e outras linguagens de programação.

Anos 60 e 70: Linguagens de programação mais próximas às atuais (ML, C, Prolog).

Anos 90: Surgimento das linguagens usadas atualmente (Java, Python, JavaScript, Ruby, C#).

Anos 2000: Novos conceitos como Swift, Go, TimeScript.

Problemas computacionais são recorrentes e cada era tem sua dimensão de problemas, porém, em todas as eras podemos observar dilemas semelhantes. Sempre buscamos problemas de decisão, busca e otimização. Problemas de decisão são problemas de caráter lógico (Sim ou Não, ideia de pertencimento, etc). Problemas de busca são relacionados à ideias binárias. Problemas de otimização são aqueles que buscam aperfeiçoamento através da maximização ou minimização de funções. Para resolver tais problemas, necessitamos de uma linguagem de programação, que é um método padronizado composto por um conjunto de regras sintáticas e semânticas de implementação de um código fonte. As linguagens contêm padrões e regras que podem ser traduzidos e interpretados por máquinas ou seres humanos.

Para que o computador entenda a linguagem, existe um processo de tradução que consiste no envio de uma linguagem de alto nível, ou programa fonte, para o compilador que executa análise do programa e então o traduz para a linguagem da máquina (linguagem de baixo nível ou programa objeto). Logo, a tradução consiste na geração do programa objeto e sua execução. Já a interpretação ocorre quando o programa fonte é executado diretamente, o que demora mais para ser decodificado, mas traz mais flexibilidade e facilidade na hora de programar.

Linguagens de tradução: Java, C++.

Linguagens de interpretação: JavaScript, Ruby, Python.

**Características de um programa**

Dentro da área de desenvolvimento de programas existem diretrizes a serem seguidas. São elas:

-Legibilidade (fácil leitura e compreensão, coerência nas instruções);

-Redigibilidade (facilidade de escrita);

-Confiabilidade (se o código faz aquilo que se propõe a fazer);

-Custo (ter custo-benefício, eficiência e execução equivalentes à infra-estrutura).

Características a se pensar:

-Atualizações;

-Uso para IA;

-Disponibilidade de ferramentas;

-Comunidade ativa;

-Adoção pelo mercado.

**Análises de código**

-Análise léxica (scanner ou leitura): A primeira fase do processo de compilação que consiste na leitura do programa fonte caractere por caractere e agrupá-los em lexemas (tokens). Particiona, classifica e elimina aquilo que não é necessário.

-Análise sintática: Depende da linguagem de programação utilizada. Analisa a forma e os componentes do sistema linguístico que interligam os constituintes da sentença.

-Análise semântica: Analisa os significados atribuídos aos símbolos e caracteres utilizados na programação. O erro de semântica se dá quando o significado e o significante não se interligam, ou seja, quando o significante não faz o que é esperado. É mais complicado de se identificar o erro de sintaxe (que é indicado pelo sistema).

**Paradigmas de programação**

-Orientação à objeto (Python, Lua, C++, Java);

-Procedural (chamadas sucessivas e procedimentos separados – Fortran, Lua);

-Funcional (instruções baseadas em funções – Lua, Javascript, Java, Python);

-Estruturado (estrutura blocos alinhados – C++, Java, Javascript, C);

-Computação distribuída (funções executadas de forma independente - Ada);

-Lógico (Java).

Paradigma estruturado (PE): Se baseia nos conceitos de sequência, decisão e iteração (C). É utilizado para problemas simples e diretos e para o aprendizado de programação. Tem um desempenho superior por ter uma linguagem de mais baixo nível, porém isso o torna, por consequência, mais complicado de ser programado. Trabalha problemas específicos e diretos.

Orientação à objeto (POO): Baseado na utilização de objetos e suas interações, é análogo ao mundo real. Classifica objetos em o que eles têm (atributos), o que fazem (comportamentos/métodos) e como fazem (estados). Reuso de códigos recorrente.

Pilares da orientação à objeto: Herança (classe filha que herda algo de uma classe mãe), encapsulamento, polimorfismo, abstração.