

Linguagem de Programação I - ESP201

Prof^a Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal de Ciência e Tecnologia da Bahia
Campus de Feira de Santana

carolsoko@ifba.edu.br

November 12, 2024

Linguagens de Programação

- 1 Definição de Linguagem de Programação
- 2 Classificação das Linguagens de Programação
- 3 Níveis de uma Linguagem de Programação
- 4 Métodos de Implementação das Linguagens de Programação
- 5 Classificação quanto ao Paradigma
 - Linguagens Estruturadas
 - Linguagens Orientadas a Objeto
 - Linguagens Imperativas
 - Linguagens Visuais
 - Linguagens de Script
 - Linguagens Funcionais
 - Linguagens Lógicas
- 6 Referências

Definição de Linguagem de Programação

Definição de Linguagem de Programação

O que é uma Linguagem de Programação?

Definição de Linguagem de Programação

O que é uma Linguagem de Programação?

“Teoricamente e formalmente a linguagem de programação é um sistema de comunicação estruturado, composto por conjuntos de símbolos, palavras-chave, regras semânticas e sintáticas que permitem o entendimento entre um programador e uma máquina, sendo que o programador escreve um algoritmo utilizando uma linguagem que a máquina entenda.” [BlipBlog 2020]

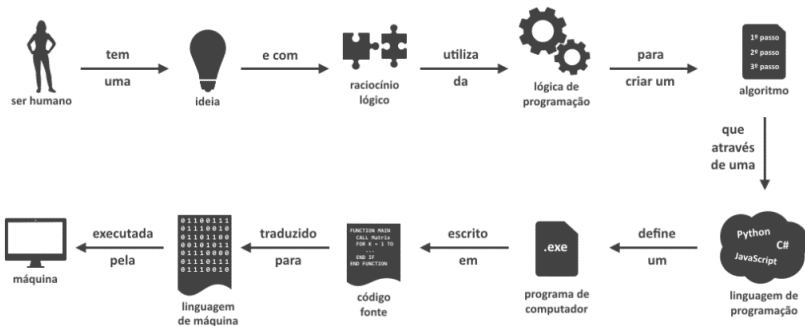
Definição de Linguagem de Programação

O que é uma Linguagem de Programação?

Linguagem de programação é um **padrão de comunicação** que dá instruções para um computador através de palavras e símbolos.

Assim como qualquer linguagem, define **regras de sintaxe e semântica** que são traduzidas em um programa de computador.

Definição de Linguagem de Programação



Classificação das Linguagens de Programação

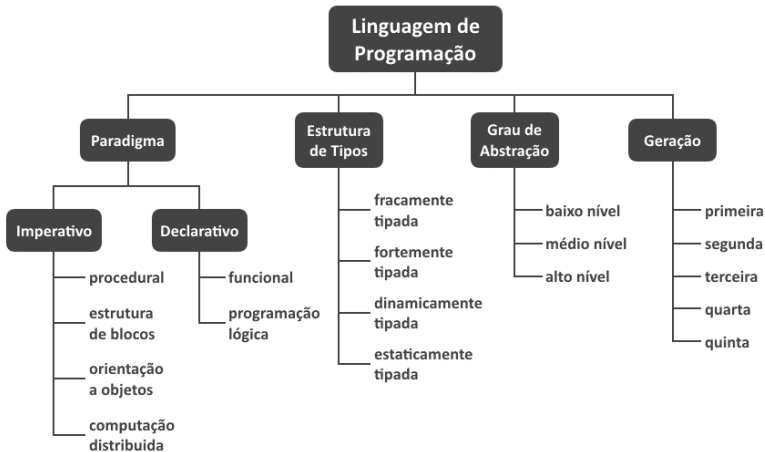
Classificação das Linguagens de Programação

Linguagens de programação são normalmente divididas duas categorias:

- 1 **Imperativas** - procedurais, orientadas a objetos, estruturas de blocos e computação distribuída
- 2 **Declarativas** - lógicas e funcionais

Porém, existem outras formas de classificar as linguagens de programação.

Classificação das Linguagens de Programação



Níveis de uma Linguagem de Programação

Níveis de uma Linguagem de Programação

Alto e Baixo Níveis

Uma forma muito usada no dia a dia para classificar uma linguagem de programação é dizer que ela é de alto ou baixo nível.

A altura, nesse caso, está relacionada com a proximidade do hardware e seu código de máquina. Um computador, através de seu processador, executa instruções simples e só entende código de máquina (0s e 1s).

Alto e baixo Níveis

Baixo Nível

Uma linguagem de **baixo nível** tem instruções mais próximas do código de máquina e é projetada pensando mais no acesso e controle ao hardware do que na facilidade de uso para o usuário.

Normalmente, as linguagens de baixo nível possuem **ótimo desempenho computacional**. **Assembly e C** são exemplos de linguagens de baixo nível, embora a segunda tenha muito mais recursos de programação que a primeira.

Alto e baixo Níveis

Alto Nível

Uma linguagem de **alto nível**, por outro lado, foi projetada pensando mais no **usuário**, na sua **facilidade de escrita**, **legibilidade do código**, **capacidade de abstração e recursos**. Como consequência, a conversão do código em uma linguagem de alto nível para código de máquina é mais custosa.

Ela está mais longe conceitualmente, porque existem mais camadas de abstração entre a linguagem que o programador escreve e o código que é rodado efetivamente no hardware. *Java*, *JavaScript* e *Python* são exemplos de linguagens de alto nível.

Diagrama em forma de pirâmide invertida mostrando a hierarquia das camadas de programação:

- Topo (menor camada): Javascript, Python
- Segunda camada: Java, C#
- Terceira camada: C++
- Quarta camada: Fortran - C - Pascal
- Quinta camada: Assembly
- Sexta camada: Linguagem de Máquina
- Base (maior camada): Hardware

Métodos de Implementação das Linguagens de Programação

Métodos de Implementação das LP's

Compilada ou Interpretada

Uma linguagem de programação pode ser classificada em 3 métodos de implementação. Método de implementação é a forma como o código escrito é convertido para um formato executável.

Quem transforma o código escrito em linguagem de programação em software é outro software, equivalente a um programa que gera novos programas para computadores. Os softwares que fazem isso são chamados de **IMPLEMENTADORES**.

Eles implementam uma linguagem de programação no sentido de torná-la utilizável para que programadores possam escrever e gerar programas com ela.

Métodos de Implementação das LP's

Compilação

Os programas podem ser traduzidos para **código de máquina** e depois executados diretamente pelo computador. Esse método é chamado de implementação baseada em compilação, com a vantagem de ter uma execução de programas muito rápida, uma vez que o processo de tradução esteja completo.

Um compilador é um software que compila o código para um formato executável.

A maioria das linguagens usam implementações de compilação, uma vez que esse método é o mais rápido. A linguagem C e a linguagem C++ são exemplos de linguagens compiladas. [Schildt e Mayer 1997]

Métodos de Implementação das LP's

Interpretação

O segundo método é chamado de *interpretação pura* e representa o *oposto da compilação*. Os programas, escritos em linguagem de programação, são **interpretados diretamente por outro software**, chamado de interpretador.

Não existe a etapa da tradução.

O interpretador age como uma máquina virtual (pois é simulada em software) para a linguagem de programação.

Métodos de Implementação das LP's

Interpretação - Vantagens e Desvantagens

A interpretação tem vantagens, pois permite a depuração do código fonte, ou seja, permite rastrear algum erro dentro do fluxo do programa até a exata linha que ele acontece.

Por outro lado, possui desvantagens como necessitar de mais espaço e possuir maior tempo de execução quando comparado ao método de compilação.

Segundo [Sebesta 2010], o tempo de execução de um sistema interpretado é de 10 a 100 vezes maior do que os compilados. *PHP* e *JavaScript* são exemplos de linguagens interpretadas.

Métodos de Implementação das LP's

Híbrido

O método híbrido representa um meio termo entre os compiladores e os interpretadores.

Ele traduz os programas em linguagem de alto nível para uma linguagem intermediária projetada para facilitar a interpretação.

Naturalmente, os métodos híbridos são mais rápidos do que a interpretação pura. *Pearl* é uma linguagem com um sistema híbrido.

Métodos de Implementação das LP's

Híbrido

As primeiras implementações de *Java* eram todas híbridas. Um programa *Java* é compilado para seu formato intermediário, chamado *bytecode*, e depois é interpretado pela JVM.

Seu formato intermediário, *bytecode*, fornece portabilidade para qualquer máquina que tenha um interpretador de *bytecode* e um sistema de tempo de execução associado. Juntos, eles são chamados de Máquina Virtual Java (JVM). Existem sistemas que traduzem o *bytecode* para código de máquina, eliminando a interpretação.

Métodos de Implementação das LP's

Híbrido

Um sistema de implementação famoso é o Just-in-time (JIT). Ele inicialmente traduz os programas para uma linguagem intermediária.

Então, durante a execução, compila os métodos da linguagem intermediária para linguagem de máquina quando esses são chamados. Todas as linguagens .NET são implementadas em um sistema JIT e algumas implementações de Java também.

Classificação quanto ao Paradigma

Classificação quanto ao Paradigma

Linguagens Estruturadas

Conceitualmente, um programa pode ser construído em torno do código ou em torno dos dados.

Alguns programas são escritos em função *“do que acontece”* (código) enquanto outros são escritos em função *“do que está sendo afetado”* (dados). A primeira forma é chamada de **modelo orientado a processos** e utilizada com o **paradigma de programação estruturada**.

Classificação quanto ao Paradigma

Linguagens Estruturadas

Esse enfoque caracteriza um programa como sendo **uma série linear de passos**, que formam o código.

O modelo orientado para o processo pode ser descrito com o código atuando sobre os dados. Contudo, à medida que os programas se tornam maiores e mais complexos, os custos de software dentro de seu ciclo se tornam maiores na etapa de manutenção do que na de desenvolvimento.

Classificação quanto ao Paradigma

Linguagens Orientadas a Objeto

Para gerenciar a crescente complexidade dos algoritmos, foi concebido o segundo paradigma de programação, chamado de **programação orientada a objetos**.

A programação orientada a objetos organiza um programa **em torno de seus dados**, ou seja, **de seus objetos**, e de um conjunto de interfaces bem definidas para acesso a esses dados.

Classificação quanto ao Paradigma

Linguagens Orientadas a Objeto

Um elemento essencial da programação orientada a objetos é a **abstração**. Nós, seres humanos, utilizamos a abstração como uma forma de lidar com a complexidade.

Um exemplo disso é o carro, pois ninguém pensa no carro como um conjunto de peças individuais e sim como um objeto bem definido e que tem um determinado comportamento. Essa abstração permite que as pessoas utilizem um carro para ir e vir, mesmo sem entender nada de mecânica. O usuário pode utilizar os vários automóveis, mesmo sem entender como eles funcionam. Para andar com o carro para trás, basta acionar a alavanca da marcha ré, ou seja, para usar o sistema basta conhecer a interface.

Classificação quanto ao Paradigma

Linguagens Orientadas a Objeto

Linguagens de programação orientadas a objetos são aquelas que proporcionam mecanismos que ajudam a implementar o modelo orientado a objetos. Esses mecanismos são: **encapsulamento, herança e polimorfismo**.

As linguagens orientadas a objetos são muito boas, mas não se deve aprender a programar com elas.

“O problema com as linguagens orientadas a objeto é que elas têm tudo implícito no ambiente que elas carregam consigo. Você queria banana, mas o que você teve foi um gorila segurando a banana e toda a floresta.” Joe Armstrong, entrevistado em Coders at Work

Classificação quanto ao Paradigma

Linguagens Imperativas

A maioria das linguagens populares das últimas décadas foram projetadas considerando a arquitetura de von Neumann. Elas são chamadas de linguagens imperativas.

A estrutura de uma linguagem imperativa foi projetada pensando em um arquitetura de máquina em vez de pensar no que é melhor para o usuário. Mesmo assim, muitos acreditam que as linguagens imperativas são mais naturais do que as funcionais no sentido de uso, porque a ideia de dar ordens ao computador por meio de comandos é intuitiva para o ser humano.

Exemplos de linguagens imperativas: C, C++, C#, Java, Cobol, PHP, Pearl, JavaScript, Ruby, Python e várias outras.

Classificação quanto ao Paradigma

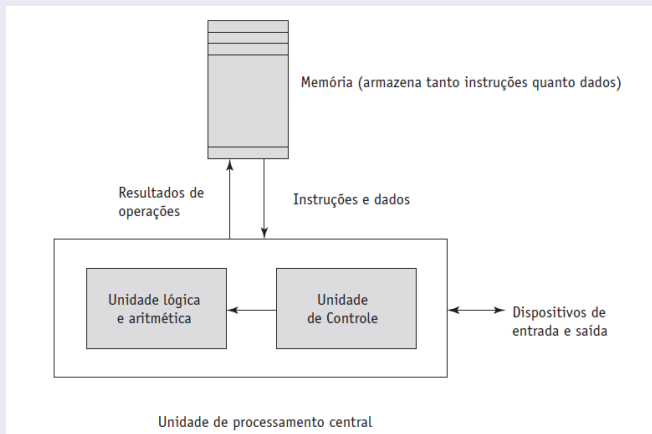
Linguagens Imperativas - Arquitetura de von Neumann

Em um computador de **von Neumann**, tanto os dados quanto os programas são armazenados na mesma memória. A unidade central de processamento (CPU), que executa instruções, é separada da memória. Logo, instruções e dados devem ser transmitidos da memória para a CPU.

Resultados de operações na CPU devem ser retornados para a memória. Praticamente todos os computadores digitais construídos desde os anos 1940 têm sido baseados nessa arquitetura.

Classificação quanto ao Paradigma

Linguagens Imperativas - Arquitetura de von Neumann



Classificação quanto ao Paradigma

Linguagens Imperativas - Principais elementos

Por causa da arquitetura de von Neumann, os recursos centrais das linguagens imperativas são as variáveis.

O termo imperativa vem do fato do programador dar ordens do que fazer ao computador. As ordens na verdade são comandos sequenciais que devem ser executados. Faça isso, depois isso, teste isso e assim por diante. Os comandos mudam o estado do programa e suas variáveis.

Sentença de atribuição, por exemplo, para quem não sabe, é um comando que usa o sinal de igual: $x = 10$;

Classificação quanto ao Paradigma

Linguagens Imperativas - Principais elementos

Uma linguagem imperativa também pode ser chamada de **procedural** devido ao paradigma de **programação procedural** (orientado a procedimento).

Contudo, uma linguagem imperativa não deve ser confundida com uma linguagem não orientada a objeto. Várias linguagens imperativas oferecem suporte parcial ou completo para orientação a objeto. C++ e Java são dois exemplos de linguagens imperativas e com pleno suporte a orientação a objetos.

A diferença é que Java só aceita orientação a objetos, enquanto C++ aceita ambos paradigmas de programação.

Classificação quanto ao Paradigma

Linguagens Visuais

As linguagens visuais são uma subcategoria das imperativas. A mais popular é o **Visual BASIC .NET (VB.NET)**.

Essas linguagens incluem capacidades para geração de segmentos de códigos que podem ser copiados de um lado para outro.

As linguagens visuais fornecem uma maneira simples de gerar interfaces gráficas.

Classificação quanto ao Paradigma

Linguagens Visuais

Por exemplo, em VB.NET, o código para produzir uma tela com um controle de formulário, como um botão ou uma caixa de texto, pode ser criado com um clique.

Tais capacidades estão disponíveis em todas as linguagens .NET: C#, Visual BASIC .NET, JScript (Microsoft JavaScript), J# (Microsoft Java) ou C++ gerenciado.

Classificação quanto ao Paradigma

Linguagens de Script

Segundo [Sebesta 2010], alguns autores se referem às linguagens de Script como uma categoria separada de linguagens de programação. Entretanto, ela é considerada uma subcategoria das linguagens imperativas. As linguagens nessa categoria são mais unidas entre si por seu método de implementação, interpretação parcial ou completa, do que por um projeto de linguagem comum.

As linguagens de scripting, dentre elas Pearl, JavaScript e Ruby, são imperativas em todos os sentidos.

Classificação quanto ao Paradigma

Linguagens Funcionais

Apesar do estilo imperativo de programação ser considerado aceitável pela maioria dos programadores, sua forte dependência da arquitetura é vista por alguns como uma restrição desnecessária nos possíveis processos de desenvolvimento de software.

Entretanto, uma minoria de programas tem sido escrita em linguagens não imperativas. O paradigma de programação funcional, baseado em funções matemáticas, é a base de projeto para um dos estilos de linguagem não imperativas mais importantes.

Esse estilo de programação é suportado por linguagens de programação funcional (ou linguagens aplicativas).

Classificação quanto ao Paradigma

Elementos das Linguagens Funcionais

As linguagens funcionais são organizadas em torno de chamadas à função em vez de sentenças de atribuição, como nas imperativas. Por exemplo, considere o algoritmo abaixo que chama 4 funções para controlar as ações de um robô de limpeza.

```
Ande(1);  
Vire(esquerda);  
Ande(2);  
Aspire(15);
```

As funções também podem ser combinadas entre si de maneira simples.

Classificação quanto ao Paradigma

Elementos das Linguagens Funcionais

Nem as sentenças de atribuição nem as variáveis abundantes em programas escritos em linguagens imperativas são necessárias em programas escritos em linguagens funcionais.

As computações são feitas basicamente pela aplicação de funções para parâmetros informados.

Classificação quanto ao Paradigma

Vantagens e Desvantagens das Linguagens Funcionais

A simplicidade e a elegância são as principais razões pela qual as linguagens funcionais surgem como principal alternativa às complexas linguagens não funcionais como C++. Outros fatores, como a eficiência, entretanto, têm prevenido que as linguagens funcionais sejam mais utilizadas.

Assim como as linguagens imperativas, as linguagens funcionais podem oferecer suporte à programação orientada a objetos.

Classificação quanto ao Paradigma

Representantes das Linguagens Funcionais

Inteligência Artificial (IA) é uma ampla área de aplicações computacionais caracterizada pelo uso de computações simbólicas em vez de numéricas.

Computações simbólicas são aquelas nas quais símbolos, compostos de nomes em vez de números, são manipulados.

Esse tipo de programação permite criar e executar segmentos de código durante a execução, o que é bem conveniente em algumas situações.

Classificação quanto ao Paradigma

Representantes das Linguagens Funcionais

A primeira linguagem de programação amplamente utilizada para aplicações de IA foi a linguagem funcional LISP.

A maioria das aplicações de IA desenvolvidas antes de 1990 foi escrita em LISP ou em uma de suas parentes próximas. LISP começou como uma linguagem puramente funcional, mas rapidamente adquiriu alguns recursos imperativos importantes que aumentaram sua eficiência de execução.

LISP ainda é a mais importante das linguagens funcionais, ao menos no sentido de que foi a única a atingir uso disseminado.

Classificação quanto ao Paradigma

Representantes das Linguagens Funcionais

Outra linguagem funcional famosa é a *Scheme*, que na verdade é um dialeto de LISP. Como uma pequena linguagem com uma sintaxe e semântica simples, *Scheme* é bastante adequada para aplicações educacionais, como cursos sobre programação funcional e introduções gerais à programação.

Peter Norving, grande estudioso e referência da área de Inteligência Artificial, recomenda aprender *Scheme* ou *Python* como a primeira linguagem de programação.

Classificação quanto ao Paradigma

Linguagens Lógicas

Vimos que linguagens imperativas são baseadas em variáveis e sinais de atribuição. As funcionais, baseadas em funções. Uma linguagem de programação lógica é baseada em regras.

Classificação quanto ao Paradigma

Elementos das Linguagens lógicas

Ao invés do cálculo numérico, existe o cálculo de predicados, que é uma notação lógica formal para comunicar processos para o computador.

A programação lógica não obedece um procedimento com um resultado bem definido. Ela aceita informações e processos de inferência para computador os resultados. Um exemplo famoso é a a relação de parentesco.

Classificação quanto ao Paradigma

Elementos das Linguagens lógicas

Imagine uma função assim:

```
mae(Leandro, Leda);  
mae(Leda, Luzia);
```

A primeira linha significa que a mãe de Leandro é Leda. A segunda, que Luzia é mãe de Leda. Se entrarmos com uma regra que define uma avó, temos:

```
avo(X, Z) :- mae(X,Y), mae(Y,Z);
```

Classificação quanto ao Paradigma

Elementos das Linguagens lógicas

Com as duas primeiras informações e com a regra dada em notação lógica matemática, o computador consegue computar quem é a avó de Leandro, por exemplo.

Essa abordagem para o desenvolvimento de software é radicalmente diferente daquelas usadas nas outras três categorias de linguagens e requer um tipo completamente diferente de linguagem.

Classificação quanto ao Paradigma

Representantes das Linguagens Lógicas




Assim como as linguagens funcionais, as linguagens lógicas possuem a IA como maior área de utilização.

Prolog é a linguagem de programação lógica mais usada. Na verdade, Prolog representa para linguagens lógicas o que LISP representa para as funcionais.

É a grande linguagem lógica existente e a maioria das linguagens lógicas atuais derivam de Prolog. O nome da linguagem vem de programação lógica.

Referências

Referências

-  BLIPBLOG. *Conheça os Tipos e as Linguagens de Programação mais usadas*. 2020.
[Https://www.blip.ai/blog/devs/linguagens-de-programacao/](https://www.blip.ai/blog/devs/linguagens-de-programacao/).
-  SCHILDT, H.; MAYER, R. *C completo e total*. [S.l.]: Pearson University, 1997. ISBN 9788534605953.
-  SEBESTA, R. *Conceitos de Linguagens de Programação*. [S.l.]: Bookman, 2010. ISBN 9788536301716.