Paradigmas de Linguagens de Programação

Apresentação e Conceitos Iniciais

Por que estudar?

- Entender os **fundamentos** por trás das linguagens
- Avaliar vantagens e limitações de cada paradigma
- Fazer escolhas mais **conscientes** em projetos de software

Conteúdo

- **Conceitos centrais**
  - **Variáveis e Vinculações**: escopo, tempo de vida, inicialização, constantes
  - **Tipos de Dados**: primitivos, compostos, inferência, compatibilidade, conversão
  - **Expressões e Comandos**: efeitos colaterais, curto-circuito, controle de fluxo
  - **Abstrações**: funções, procedimentos, parâmetros, polimorfismo
  - **Tipos Abstratos de Dados**: modularidade, encapsulamento, ocultamento

Conteúdo

- **História & Motivação**
  - Fatores que influenciam o projeto de linguagens
  - Critérios de avaliação e métodos de implementação
  - Ambientes de programação

Conteúdo

- **Paradigmas**
  - Imperativo
  - Orientado a Objetos
  - Funcional
  - Lógico

Bibliografia

- MELO, Ana Cristina Vieira de; SILVA, Flávio Soares Corrêa da. **Princípios de linguagens de programação**. São Paulo: Blücher, 2003. E-Book.
- *SEBESTA, Robert W. **Conceitos de linguagens de programação.** 11. ed. Porto Alegre: Bookman, 2018. E-Book.
- SILVA, Fabrício Machado da. **Paradigmas de programação**. Porto Alegre: SAGAH, 2019. E-Book.

Bibliografia

- BACKES, André. **Linguagem C: completa e descomplicada**. 2. ed. Rio de Janeiro: GEN LTC, c2019. E-book.
- FURGERI, Sérgio. **Java 8, ensino didático: desenvolvimento e implementação de aplicações.** São Paulo: Erica, c2015. E-book.
- IERUSALIMSCHY, Roberto. **Programando em lua.** 3. ed. Rio de Janeiro: LTC, c2015. E-book.
- MANZANO, André Luiz Navarro Garcia. **Algoritmos funcionais: introdução minimalista à lógica de programação funcional pura aplicada à teoria dos conjuntos.** São Paulo: Alta Books, c2020. E-book.
- TUCKER, Allen B.; NOONAN, Robert. **Linguagens de programação: princípios e paradigmas.** 2. ed. São Paulo: McGraw-Hill, c2009.

Razões para estudar conceitos de linguagens de programação

"Linguagem usada por uma **pessoa**
para expressar um **processo** através
do qual um **computador**
pode resolver um **problema**."

- SEBESTA

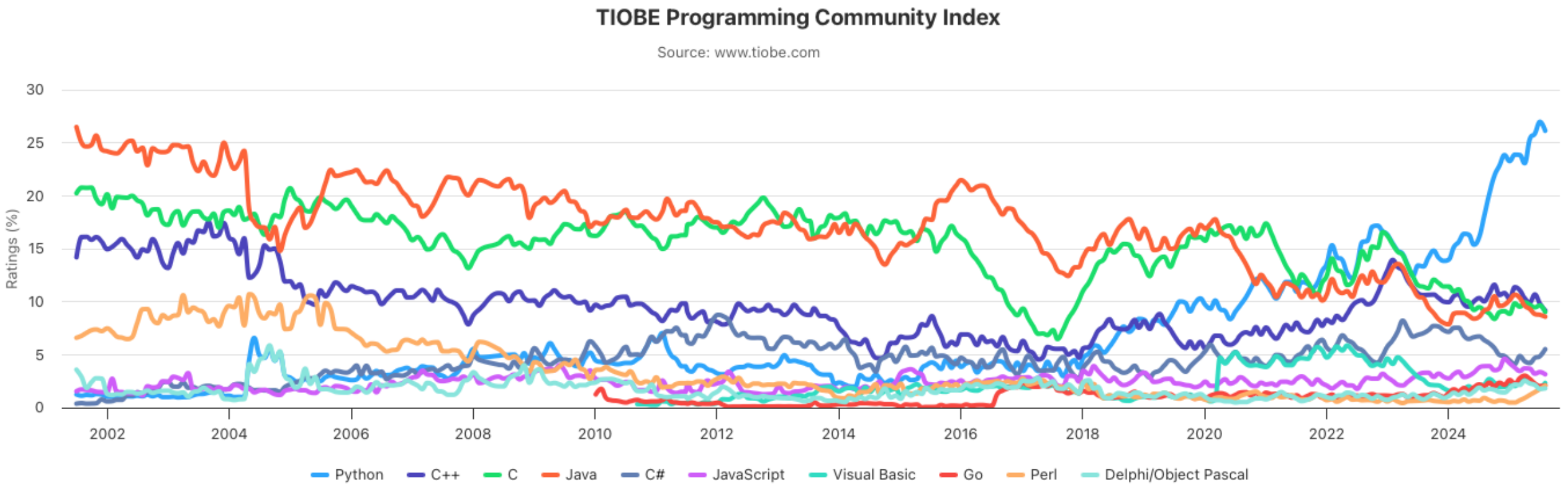
Razões para estudar conceitos de linguagens de programação

- **Maior capacidade de expressar ideias**
  - conhecer diferentes linguagens amplia o vocabulário de abstrações para resolver problemas.
- **Melhor embasamento para escolher linguagens adequadas**
  - um arquiteto de software (ou você, ao abrir uma startup) precisa selecionar a linguagem mais apropriada para cada projeto.
- **Maior facilidade para aprender novas linguagens**
  - quem entende os conceitos fundamentais percebe semelhanças e diferenças rapidamente.

Razões para estudar conceitos de linguagens de programação

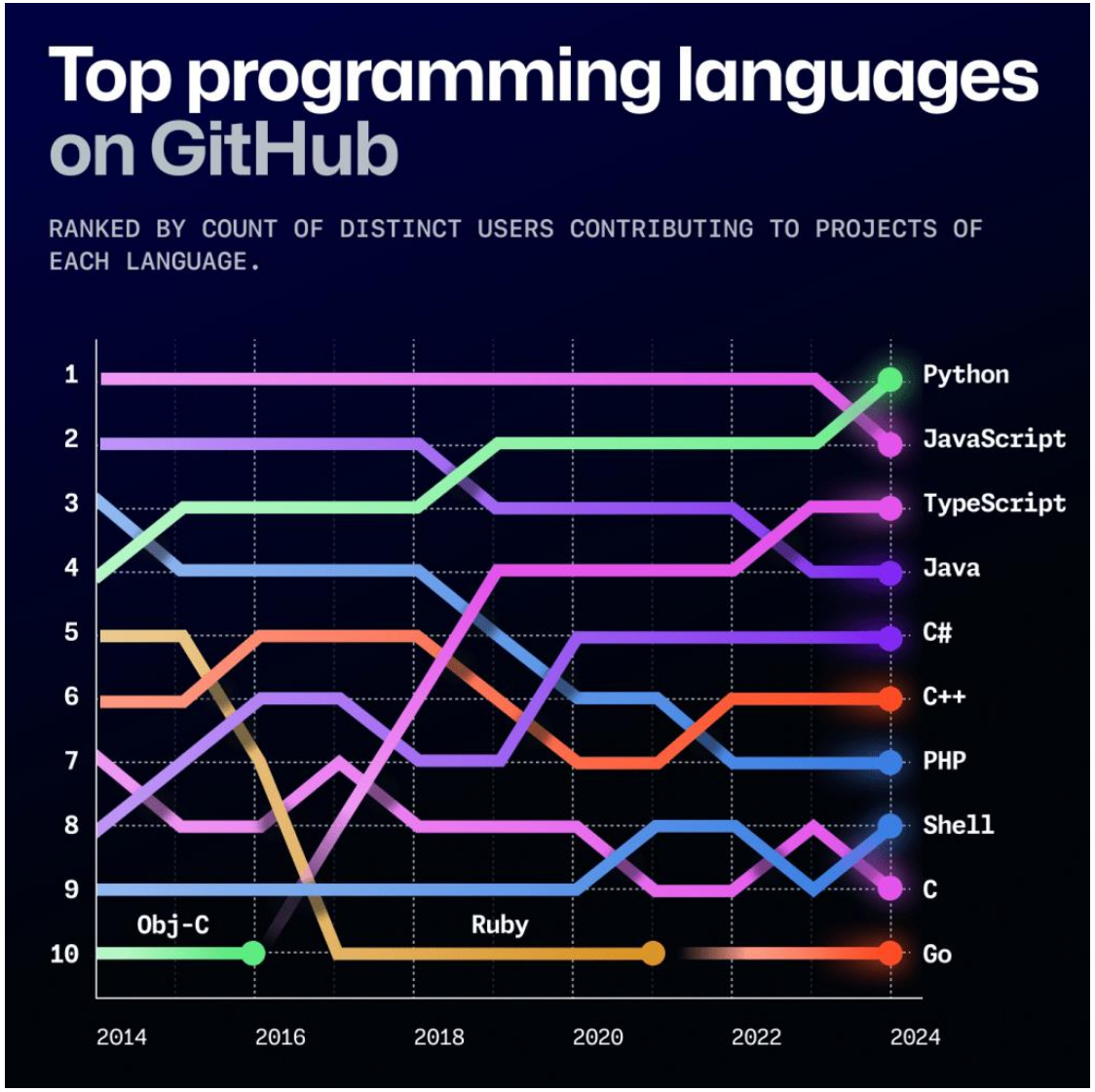
- **Melhor compreensão da importância da implementação**
  - entender como uma linguagem é implementada ajuda a avaliar desempenho e limitações.
- **Melhor aproveitamento das linguagens já conhecidas**
  - dominar os conceitos permite usar recursos existentes de forma mais eficiente.
- **Avanço geral da computação**
  - estudar conceitos de linguagens contribui para a evolução contínua da área.

Razões para estudar conceitos de linguagens de programação



O **TIOBE Index** é um ranking que mede a **popularidade de linguagens de programação**, usando como base buscas na web, cursos e profissionais disponíveis

Razões para estudar conceitos de linguagens de programação



O **GitHub** mede o uso de linguagens observando a **atividade real da comunidade** (commits, PRs, issues, discussões e código enviado). Ou seja, reflete a **adoção prática em projetos**, não apenas buscas ou cursos.

Domínios de programação

- **Aplicações científicas**
- **Aplicações comerciais**
- **Inteligência artificial**
- **Programação de sistemas**
- **Construção de Scripts e Software para Web**

Domínios de programação

- **Aplicações científicas**
  - Primeiras aplicações surgiram na década de 40 junto com os primeiros computadores.
  - Grande quantidade de cálculos em ponto flutuante.
  - Uso intenso de arrays e matrizes.
  - **Fortran**.
  - Mais recente: **Matlab, Python, R**

Domínios de programação

- **Aplicações comerciais**
  - Início na década de 50
  - Facilidade para produzir relatórios elaborados.
  - Uso de números decimais e caracteres.
  - **COBOL**.
  - Hoje: **Java, ...**

Domínios de programação

- **Inteligência artificial**
  - Manipula símbolos em vez de números;
  - Uso de listas encadeadas.
  - **LISP** (McCarthy, 1959), Linguagem functional.
  - **Prolog** (Clocksin e Mellish, 1970), Linguagem Lógica.
  - Machine Learning
    - **Python**

Domínios de programação

- **Programação de sistemas**
  - Sistema operacional e ferramentas de suporte à programação (software básico)
  - Exige eficiência devido ao uso contínuo.
  - Deve possuir recursos de baixo nível.
  - Ex: leitura e escrita em arquivos; inicialização e término de programas.
  - **C** (ANSI, 1989) - Linguagem do Unix e do Windows

Domínios de programação

- **Construção de Scripts e Software para Web**
  - **Script:** arquivo contendo lista de comandos básicos.
  - **Primeiros scripts (sistemas):**
    - **sh** – primeiro shell do Unix.
  - **Scripts na Web:**
    - **HTML / XML** – linguagens de marcação (não são de programação).
    - **JavaScript** – criado pela Netscape (1995), trouxe scripts para navegadores.
    - **PHP** – scripting no lado do servidor.
    - **Linguagens de propósito geral na Web:**
      - **Java** – amplamente usado em aplicações empresariais (JSP, ...).
      - **Python** – popular com frameworks como Django e Flask.
      - **Ruby** – destaque nos anos 2000 com Ruby on Rails.

Classificação quanto ao nível de utilização

- Linguagens de Máquina
- Linguagens Baixo Nível
- Linguagens de Alto Nível

Classificação quanto ao nível de utilização

Linguagens de Máquina

- Internamente o computador só trabalha com números binários (0000011100...).
- Linguagem nativa de computadores sendo totalmente dependente de máquina, onde programas são expressos somente por números.
- Praticamente ilegível para humanos, mas é o que o computador entende.
- É totalmente dependente da arquitetura da máquina. Portanto, não é portável.

Classificação quanto ao nível de utilização

Linguagens Baixo Nível

- Linguagem Assembly (LA)
- Linguagem de baixo nível que utiliza nomes e símbolos ao invés de códigos de instruções de máquina.
- LA é dependente de máquina, mas cria um nível de abstração sobre a linguagem de máquina, através de nomes e símbolos, que tornam a tarefa de programação um pouco menos árdua

Classificação quanto ao nível de utilização

Linguagens Alto Nível

- Linguagens de alto nível são aquelas que são independentes de máquina.
- Necessitam de tradução para a linguagem de máquina, por meio de compiladores (mais comum) e/ou interpretadores.
- Seu uso possui diversos benefícios:
  - Notações legíveis e familiares aos programadores;
  - Independência de máquina – portabilidade;
  - Maior facilidade para detecção de erros durante desenvolvimento e execução.

Critérios de avaliação de linguagens

- **Legibilidade**
  - Facilidade com que programas podem ser lidos e entendidos.
- **Facilidade de Escrita**
  - Facilidade com que a linguagem pode ser usada para criar programas.
- **Confiabilidade**
  - Grau de conformidade com as especificações (isto é, se o programa executa o que foi definido).
- **Custo**
  - Custo total de uso da linguagem (desenvolvimento, manutenção, treinamento, execução).

Critérios de avaliação de linguagens

TABELA 1.1 Critérios de avaliação de linguagens e as características que os afetam

	CRITÉRIOS		
Característica	**Legibilidade**	**Facilidade de escrita**	**Confiabilidade**
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

Critérios de avaliação de linguagens

TABELA 1.1 Critérios de avaliação de linguagens e as características que os afetam

	CRITÉRIOS		
Característica	**Legibilidade**	**Facilidade de escrita**	**Confiabilidade**
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

Legibilidade

- **Simplicidade geral**
  - Pouco número de construções, fácil de gerenciar.
  - Evitar *multiplicidade de recursos* (várias formas de fazer a mesma coisa).
  - Evitar *sobrecarga de operadores* excessiva

Critérios de avaliação de linguagens

TABELA 1.1 Critérios de avaliação de linguagens e as características que os afetam

	CRITÉRIOS		
Característica	**Legibilidade**	**Facilidade de escrita**	**Confiabilidade**
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

Legibilidade

- **Ortogonalidade**
  - Pequeno conjunto de **construções primitivas** que pode ser combinado de várias formas.
  - Todas as combinações possíveis devem ser válidas e significativas.
  - Quanto maior a ortogonalidade, mais simples é aprender e usar a linguagem.
  - *Ex:* em **Python**, concatenar strings é direto ("a" + "b"), enquanto em **C** exige funções específicas (strcat).

Critérios de avaliação de linguagens

TABELA 1.1 Critérios de avaliação de linguagens e as características que os afetam

	CRITÉRIOS		
Característica	**Legibilidade**	**Facilidade de escrita**	**Confiabilidade**
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

Legibilidade

- **Tipos e estruturas de dados**
  - A linguagem deve oferecer **tipos pré-definidos adequados** (ex.: boolean com valores TRUE/FALSE).
  - Também deve prover **recursos suficientes para criar estruturas próprias**, como struct em C ou class em Java.
  - *Ex:* em algumas linguagens pode-se escrever **timeOut = true**, enquanto em outras seria necessário usar **timeOut = 1**.

Critérios de avaliação de linguagens

TABELA 1.1 Critérios de avaliação de linguagens e as características que os afetam

	CRITÉRIOS		
Característica	**Legibilidade**	**Facilidade de escrita**	**Confiabilidade**
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

Legibilidade

- **Considerações de sintaxe**
  - Uso de **palavras especiais** (if, while, class, for) e **regras claras de composição**.
  - Diferenças entre linguagens:
    - **Chaves** para encerrar qualquer comando composto (ex.: C, Java).
    - end if no **Fortran95/Ada**
    - **Indentação** no **Python**
  - **Forma e significado**:
    - Construtos devem ser **autoexplicativos**.
    - Palavras-chave precisam ser **significativas**.
    - Ex: grep no Unix → g (global), re (regular expression), p (print).

Critérios de avaliação de linguagens

Facilidade de Escrita

- **Simplicidade e Ortogonalidade**
  - Poucos construtos, número reduzido de primitivas e um conjunto pequeno de regras de combinação.
  - Quanto menos exceções e regras especiais, mais fácil é escrever código sem erros.
  - Exemplo: em C, expressões aritméticas e lógicas podem ser usadas uniformemente em diferentes contextos.
- **Expressividade**
  - Conjunto de formas relativamente convenientes de especificar operações.
  - Depende da força e variedade de operadores e funções pré-definidas.
  - Exemplo: operador lógico **and** ou comando de repetição **for** que permite expressar loops de forma simples.

Critérios de avaliação de linguagens

Confiabilidade

- **Verificação de Tipos**
  - Detecção de erros de tipo em tempo de compilação ou execução.
  - Exemplo: em C de 1978, era possível passar um int para uma função que esperava float, resultando em erro.
  - Quanto mais cedo (em tempo de compilação) o erro for detectado, maior a confiabilidade.
- **Tratamento de Exceções**
  - Capacidade de interceptar erros em tempo de execução e tomar medidas corretivas.
  - Exemplo: presente em C++, Java, C#, mas não em C.

Critérios de avaliação de linguagens

Confiabilidade

- **Apelidos (Aliasing)**
  - Presença de duas ou mais referências distintas para a mesma posição de memória.
  - Pode gerar comportamentos inesperados e reduzir a confiabilidade.
- **Legibilidade e Facilidade de Escrita**
  - Uma linguagem que não oferece formas naturais de expressar algoritmos terá confiabilidade reduzida.
  - Sintaxes pouco claras dificultam a detecção e a correção de erros.

Critérios de avaliação de linguagens

Custo

- **Treinamento de programadores**
  - Custos relacionados ao tempo e esforço para aprender a linguagem.
- **Escrita de programas**
  - Quanto mais próxima a linguagem estiver do domínio da aplicação, menor o custo de desenvolvimento.
- **Compilação de programas**
  - Algumas linguagens podem ter alto custo de compilação (exemplo: Ada em sua fase inicial).

Critérios de avaliação de linguagens

Custo

- **Execução de programas**
  - Custos relacionados a verificações em tempo de execução e desempenho do código.
- **Sistema de implementação da linguagem**
  - Disponibilidade de compiladores gratuitos reduz custos (exemplo: Java).
- **Confiabilidade**
  - Baixa confiabilidade gera custos elevados devido a erros difíceis de detectar e corrigir.
- **Manutenção de programas**
  - A etapa mais cara do ciclo de vida de software.
  - Linguagens que favorecem clareza e boas práticas reduzem o custo de manutenção.

Influências no projeto de linguagens

- **Arquitetura de Computadores**
  - As linguagens são desenvolvidas em torno da arquitetura predominante dos computadores, conhecida como **arquitetura de von Neumann**.
  - Essa arquitetura influencia diretamente o modelo sequencial de execução das linguagens imperativas.

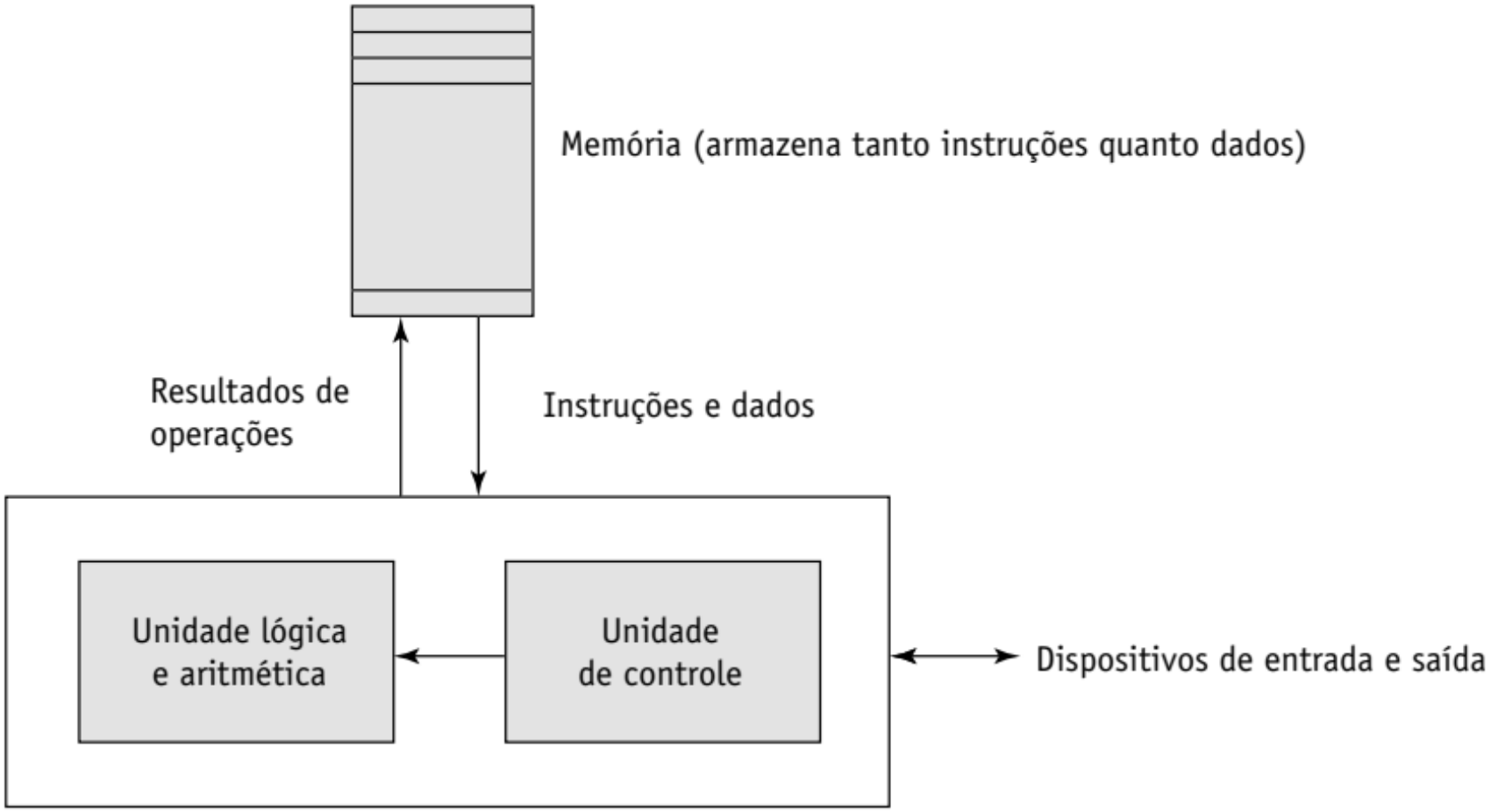
- **Metodologias de Programação**
  - Novas metodologias de desenvolvimento de software (ex.: **programação orientada a objetos**) levaram ao surgimento de novos **paradigmas de programação**.
  - Por consequência, também surgiram novas linguagens para dar suporte a essas metodologias.

Influências no projeto de linguagens

- **Arquitetura de von Neumann**
  - Modelo de computador mais conhecido e dominante.
  - Dados e programas ficam armazenados na **memória**.
  - A **memória é separada da CPU**.
  - Instruções e dados são transferidos da memória para a CPU.

- **Consequências para linguagens imperativas**
  - Tornaram-se o tipo mais dominante de linguagem.
  - Relação direta com a arquitetura:
    - **Variáveis** modelam as **células de memória**.
    - **Comandos de atribuição** modelam a **transferência de dados**.
    - **Laços de repetição (iterações)** são eficientes nesse modelo.

Influências no projeto de linguagens



Influências no projeto de linguagens

- **Ciclo de obtenção e execução**
  - O ciclo reflete a arquitetura de Von Neumann:
    - a CPU busca instruções na memória, decodifica e executa de forma repetitiva, controlando o fluxo de dados e operações.

```
inicialize o contador de programa
repita para sempre
    obtenha a instrução apontada pelo contador de programa
    incremente o contador de programa a fim de que aponte para a próxima instrução
    decodifique a instrução
    execute a instrução
fim repita
```

Influências no projeto de linguagens

- **Gargalo de von Neumann (Von Neumann Bottleneck)**
  - A velocidade de conexão entre a memória de um computador e seu processador determina a velocidade do computador.
  - As instruções de programa muitas vezes podem ser executadas muito mais rápido do que a velocidade da conexão; assim, a velocidade da conexão resulta em um gargalo.
  - É o principal fator limitante na velocidade dos computadores.

Influências no projeto de linguagens

- **Metodologias de projeto de programas**
  - Década de 1950 e início de 1960: Aplicações simples; preocupação com a eficiência da máquina
  - Final da década de 1960: A eficiência das pessoas tornou-se importante; legibilidade, melhores estruturas de controle
    - programação estruturada
    - projeto top-down e refinamento passo a passo
  - Final da década de 1970: De orientado a processo para orientado a dados
    - abstração de dados
  - Metade da década de 1980: Programação orientada a objetos
    - abstração de dados + herança + polimorfismo

Categorias de linguagens

- Classificação quanto ao paradigma
- O que é um Paradigma?
  - Modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns.

Categorias de linguagens

- Classificação quanto ao paradigma

- Imperativa
  - Orientada a objetos
- Funcional
- Lógica

Categorias de linguagens

- **Imperativa**
  - Características centrais: variáveis, comandos de atribuição e iteração
  - Incluem linguagens que suportam programação orientada a objetos
    - Classes/objetos, herança, polimofirsmo, encapsulamento
  - Incluem linguagens de script
  - Incluem linguagens visuais
  - Exemplos: C, Java, Perl, JavaScript, Ruby, Visual BASIC .NET, C++, Python

Categorias de linguagens

- **Funcional**
  - O principal meio de computação é aplicando **funções matemáticas puras** a parâmetros
  - Funções **não alteram estado** (evitam variáveis mutáveis e efeitos colaterais)
  - Paradigma baseado em **declaração** do que deve ser feito, não em como fazer
  - Suporte natural a **recursão** (mais usado que loops)
  - Funções são **cidadãos de primeira classe**: podem ser passadas como argumentos, retornadas e combinadas
  - Favorece **paralelismo** e **concorrência** por não depender de variáveis globais ou estados mutáveis
  - Exemplos: LISP, Scheme, Haskell

Categorias de linguagens

- **Lógica**
  - Baseadas em **regras e fatos declarativos**
  - A computação é feita por **inferência lógica** (o motor de busca encontra soluções)
  - O programador descreve **o que é verdade**, não **como calcular**
  - Muito usadas em **IA simbólica** (raciocínio, dedução, sistemas especialistas)
  - Exemplo clássico: **Prolog** (usado em NLP, chatbots, e sistemas como IBM Watson)

Trade-offs no projeto de linguagens

- **Legibilidade vs. custo de execução**
  - Garantir segurança geralmente **torna o programa mais lento**
  - Ex.: *Java* verifica acesso a arrays → mais confiável, mas mais custoso
  - *C* não faz essa verificação → mais rápido, mas arriscado
- **Legibilidade vs. facilidade de escrita**
  - Código fácil de **ler** nem sempre é o mais fácil de **escrever**
  - Ex.: *APL* permite escrever muito com poucos símbolos → compacto, mas difícil de entender
- **Facilidade de escrita (flexibilidade) vs. confiabilidade**
  - Mais **liberdade** pode trazer mais **erros**
  - Ex.: *C++* com ponteiros: muito flexível, mas sujeito a falhas graves (Java removeu ponteiros explícitos)

Trade-offs no projeto de linguagens

APL (https://www.rosettacode.org/wiki/99_Bottles_of_Beer#APL)

```
bob←{ (⍕⍵), ' bottle', (1=⍵)↓'s of beer'}
bobw←{(bob ⍵) , ' on the wall'}
beer←{ (bobw ⍵) , ', ' (bob ⍵) , ';
take one down and pass it around, ' bobw ⍵-1}↑beer¨ ⌽(1-□IO)+⍳99
```

Python (https://www.rosettacode.org/wiki/99_Bottles_of_Beer/Python#Normal_Code)

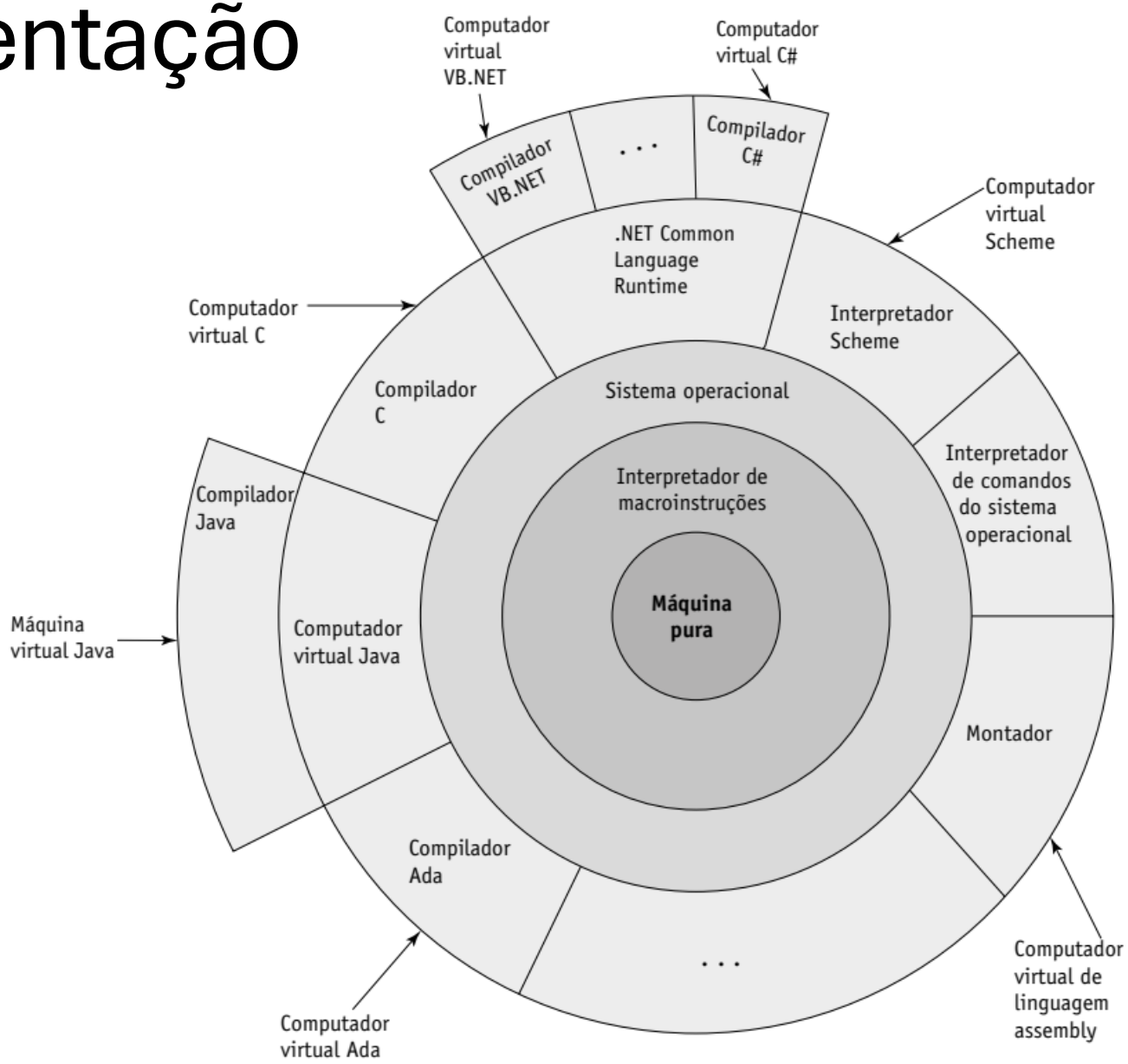
```
def sing(b, end):
    print(b or 'No more','bottle'+('s' if b-1 else ''), end)
for i in range(99, 0, -1):
    sing(i, 'of beer on the wall,')
    sing(i, 'of beer,')
    print('Take one down, pass it around,')
    sing(i-1, 'of beer on the wall.\n')
```

Métodos de implementação

- **Compilação**
  - Programas são traduzidos inteiramente para linguagem de máquina antes da execução.
  - Vantagem: execução rápida.
  - Desvantagem: tempo de compilação.
- **Interpretação Pura**
  - O programa é interpretado diretamente por outro programa (interpretador).
  - Vantagem: facilidade de depuração e flexibilidade.
  - Desvantagem: execução lenta.
- **Sistema Híbrido**
  - Compromisso entre compiladores e interpretadores.
  - Traduzem para uma **linguagem intermediária**, depois interpretada.
  - Vantagem: portabilidade.
  - Desvantagem: execução não tão rápida quanto compilada diretamente.

Métodos de implementação

Um sistema de computação pode ser visto como **camadas de abstração**, onde cada camada funciona como uma máquina virtual para a camada superior.



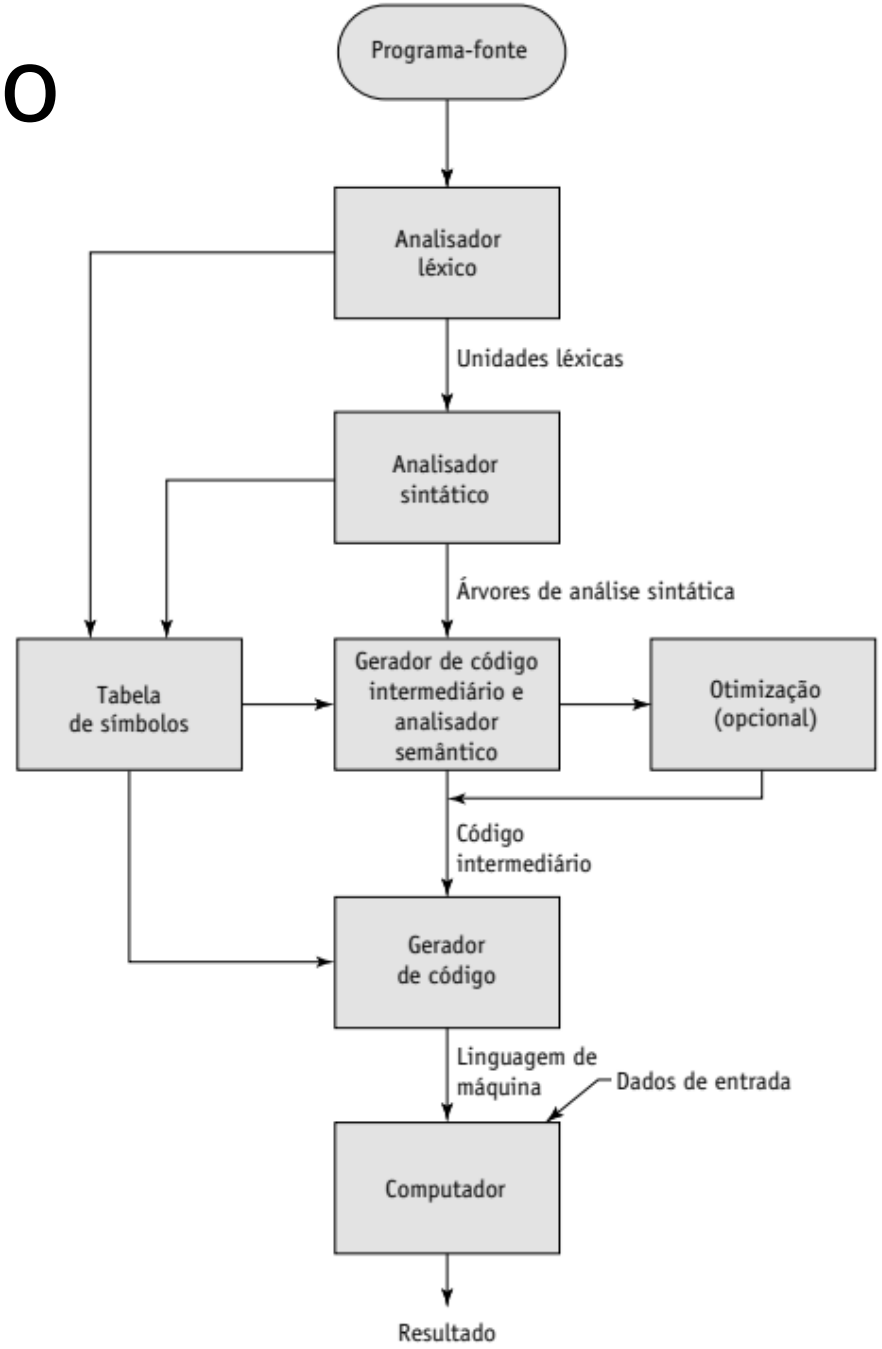
Métodos de implementação

Compilação

- Traduz programas de alto nível (linguagem fonte) em código de máquina (linguagem de máquina)
- Tradução lenta, execução rápida

Métodos de implementação

- O processo de compilação possui várias fases:
  - **Análise léxica**: converte caracteres do programa fonte em unidades léxicas
  - **Análise sintática**: transforma unidades léxicas em árvores sintáticas, que representam a estrutura do programa
  - **Análise semântica**: gera código intermediário
  - **Geração de código**: produz o código de máquina



Métodos de implementação

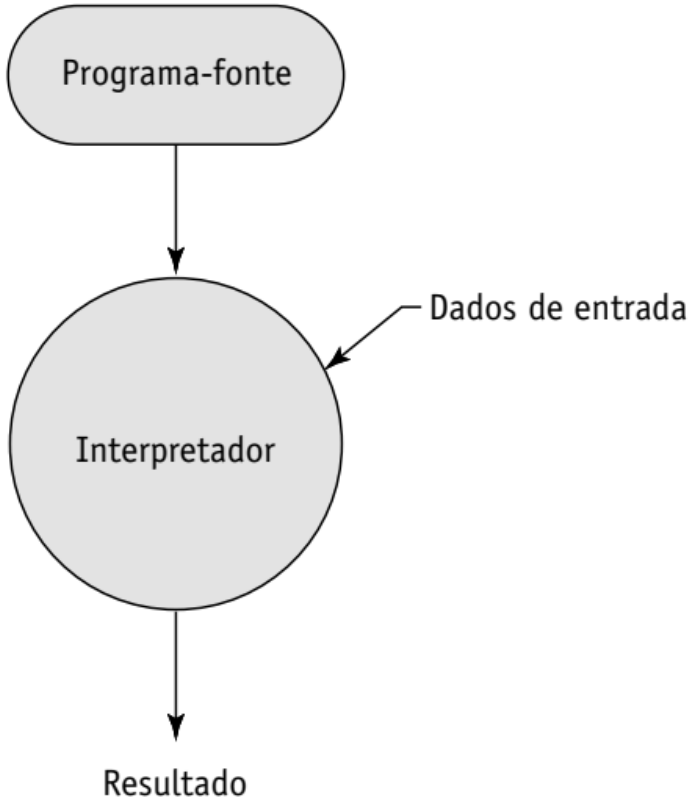
Interpretação Pura

- Sem tradução
- Implementação mais simples dos programas (erros em tempo de execução podem ser facilmente e imediatamente exibidos)
- Execução mais lenta (10 a 100 vezes mais lenta que programas compilados)
- Frequentemente requer mais espaço
- Atualmente rara em linguagens tradicionais de alto nível
- Retorno significativo em algumas linguagens de script para Web (ex.: JavaScript, PHP)

Métodos de implementação

Interpretação Pura

- O programa-fonte é executado diretamente pelo interpretador.
- Não há tradução para código de máquina.
- A cada execução, o interpretador lê e processa as instruções.



Métodos de implementação

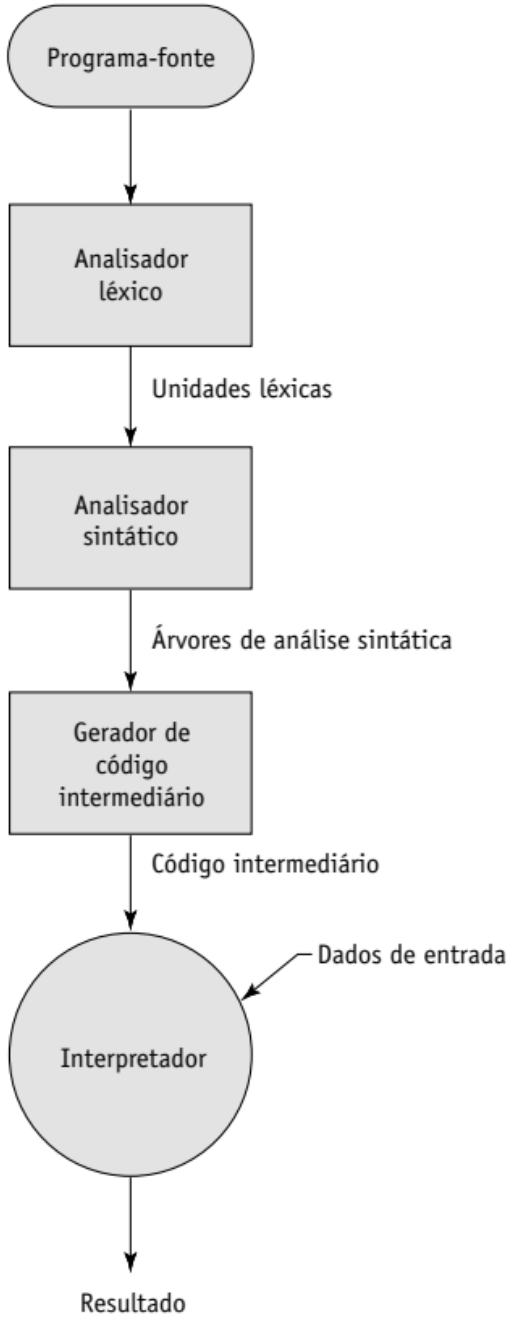
Sistema Híbrido

- Um compromisso entre compiladores e interpretadores puros.
- O programa em linguagem de alto nível é traduzido para uma **linguagem intermediária**, que permite interpretação mais simples.
- **Exemplos**:
  - Programas em Perl são parcialmente compilados para detectar erros antes da execução.
- **Interpretação**:
  - As primeiras implementações do Java eram híbridas: o código intermediário (bytecode) traz portabilidade para qualquer máquina que possua um interpretador de bytecode e um sistema de execução (*run-time system*).
  - Juntos, esses componentes formam a **Java Virtual Machine (JVM)**.

Métodos de implementação

Sistema Híbrido

- O programa-fonte passa por análise léxica e sintática.
- Um **código intermediário** é gerado.
- Esse código é então executado por um **interpretador** em tempo de execução.
- Vantagem: permite **portabilidade** e **detecção de erros** antes da execução final.



Métodos de implementação

- **Implementação Just-in-time (JIT)**
  - Traduz o programa para uma **linguagem intermediária**.
  - No momento da execução, os trechos necessários são **compilados para código de máquina**.
  - Esse código é **armazenado em cache**, acelerando chamadas futuras.
  - Muito usado em **Java (JVM)** e nas linguagens **.NET**.

Ambientes de Programação

- Conjunto integrado de ferramentas que auxiliam no desenvolvimento de software.
- **UNIX** – pioneiro
  - Reuniu ferramentas simples e poderosas, cada uma com uma função específica.
- Editores de texto (ex.: *vim*)
- Compiladores (*gcc*)
- Interpretadores (*sh*, *python*)
- Bibliotecas (códigos reutilizáveis)
- Ligadores (linkers) – unem módulos compilados
- Depuradores (debuggers) – localizar e corrigir erros

Ambientes de Programação

Ambiente Integrado de Desenvolvimento (IDE)

- Integram em um único lugar várias dessas ferramentas (editor, compilador, depurador, controle de versão etc.).

- Exemplos
  - Eclipse (Java, C++)
  - PyCharm (Python)
  - VS Code (multi-linguagem, extensões)

- Aumentam a produtividade
- Reduzem chances de erro
- Facilitam a depuração
- Melhoram a legibilidade e documentação do código
- Integram controle de versão, execução de testes e plugins

Resumo

- O estudo das linguagens de programação é valioso por vários motivos:
  - Aumenta nossa capacidade de usar diferentes construções
  - Permite escolher linguagens de forma mais inteligente
  - Facilita o aprendizado de novas linguagens

- Os critérios mais importantes para avaliar linguagens de programação incluem:
  - Legibilidade, facilidade de escrita, confiabilidade, custo

Resumo

- As principais influências no design de linguagens têm sido a arquitetura das máquinas e as metodologias de desenvolvimento de software

- Os principais métodos de implementação de linguagens de programação são: compilação, interpretação pura e implementação híbrida

Disponibilidade de Processadores e Ambientes para a Linguagem

- **C, C++, Fortran, Ada** → gcc.gnu.org
- **C# e F#** → microsoft.com
- **Java** → oracle.com/java
- **Haskell** → haskell.org
- **Lua** → lua.org
- **Scheme** → plt-scheme.org
- **Perl** → perl.org
- **Python** → python.org
- **Ruby** → ruby-lang.org
- **JavaScript** → developer.mozilla.org
- **PHP** → php.net

Paradigmas de Linguagens de Programação

Apresentação e Conceitos Iniciais