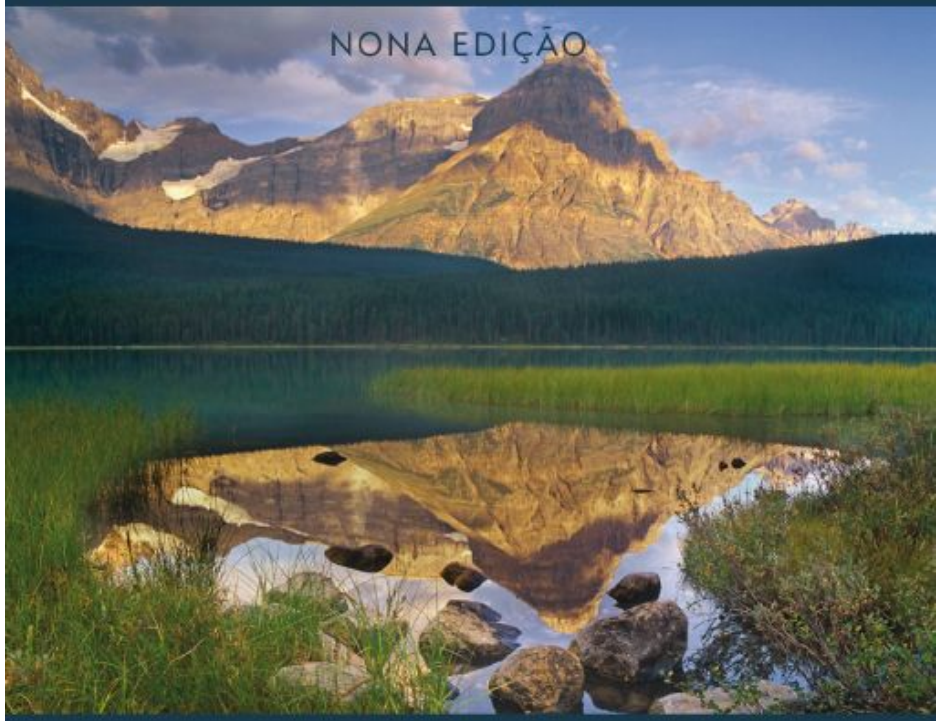
# CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

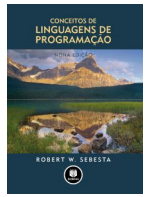


ROBERT W. SEBESTA

bookman

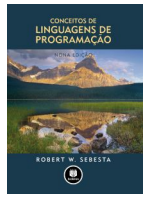
# Capítulo 1

# Aspectos Preliminares



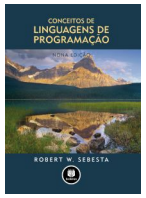

# Tópicos do Capítulo 1

- Razões para estudar conceitos de linguagens de programação
- Domínios de programação
- Critérios de avaliação de linguagens
- Influências no projeto de linguagens
- Categorias de linguagens
- Trade-offs no projeto de linguagens
- Métodos de implementação
- Ambientes de programação



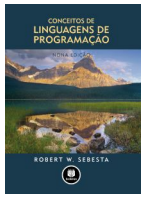

# Razões para estudar conceitos de linguagens de programação

- Capacidade aumentada para expressar ideias
- Embasamento melhorado para escolher linguagens apropriadas
- Habilidade aumentada para aprender novas linguagens
- Melhor entendimento da importância da implementação
- Melhor uso de linguagens já conhecidas
- Avanço geral da computação



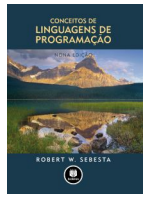

# Domínios de programação

- Aplicações científicas
  - Grande número de computações de aritmética de ponto flutuante; uso de matrizes
  - Fortran
- Aplicações empresariais
  - Produz relatório, usa números decimais e caracteres
  - COBOL
- Inteligência artificial
  - Símbolos em vez de números manipulados; uso de listas ligadas
  - LISP
- Programação de sistemas
  - Precisa de eficiência por causa do uso contínuo
  - C
- Software para a Web
  - Eclética coleção de linguagens: de marcação (como XHTML), de scripting (como PHP), de propósito geral (como Java)



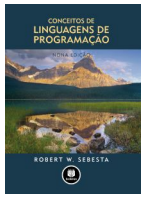

# Critérios de avaliação de linguagens

- **Legibilidade**: facilidade com a qual os programas podem ser lidos e entendidos
- **Facilidade de escrita**: facilidade com a qual uma linguagem pode ser usada para criar programas para um dado domínio
- **Confiabilidade**: conformidade com as especificações
- **Custo**: o custo total definitivo de uma linguagem



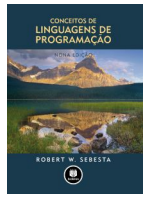

# Critério de avalição: legibilidade

- Simplicidade geral
  - Um conjunto controlável de recursos e construções
  - Mínima multiplicidade de recursos
  - Mínima sobrecarga de operadores
- Ortogonalidade
  - Um conjunto relativamente pequeno de construções primitivas pode ser combinado a um número relativamente pequeno de formas
  - Cada possível combinação é legal
- Tipos de dados
  - Mecanismos adequados para definir tipos de dados
- Projeto da sintaxe
  - Formato dos identificadores
  - Palavras especiais e métodos de formar sentenças compostas
  - Forma e significado: construções autodescritivas, palavras-chave significativas



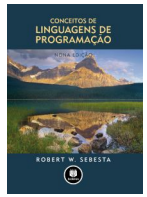

# Critério de avaliação: facilidade de escrita

- Simplicidade e ortogonalidade
  - Poucas construções, número pequeno de primitivas e um pequeno conjunto de regras para combiná-las
- Suporte à abstração
  - A habilidade de definir e usar estruturas ou operações complicadas de forma a permitir que muitos dos detalhes sejam ignorados
- Expressividade
  - Um conjunto de formas relativamente convenientes de especificar as operações
  - Força e número de operadores e funções pré-definidas



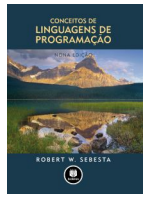

# Critério de avaliação: confiabilidade

- Verificação de tipos
  - Testes para detectar erros de tipos
- Tratamento de exceções
  - Interceptar erros em tempo de execução e tomar medidas corretivas
- Utilização de apelidos
  - Nomes distintos que podem ser usados para acessar a mesma célula de memória
- Legibilidade e facilidade de escrita
  - Uma linguagem que não oferece maneiras naturais para expressar os algoritmos requeridos irá necessariamente usar abordagens não naturais, reduzindo a confiabilidade



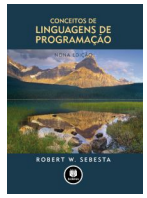

# Critério de avaliação: custo

- Treinar programadores para usar a linguagem
- Escrever programas (proximidade com o propósito da aplicação em particular)
- Compilar programas
- Executar programas
- Sistema de implementação da linguagem: disponibilidade de compiladores gratuitos
- Confiabilidade baixa leva a custos altos
- Manter programas



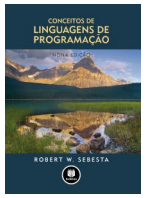

# Critério de avaliação: outros

- Portabilidade
  - A facilidade com a qual os programas podem ser movidos de uma implementação para outra
- Generalidade
  - A aplicabilidade a uma ampla faixa de aplicações
- Bem definida
  - Em relação à completude e à precisão do documento oficial que define a linguagem



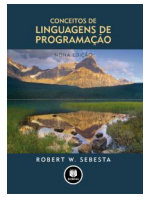

# Influências no projeto de linguagens

- Arquitetura de computadores
  - Linguagens são projetadas considerando a principal arquitetura de computadores, chamada de arquitetura de *von Neumann*
- Metodologias de projeto de programas
  - Novas metodologias de desenvolvimento de software (por exemplo, desenvolvimento de software orientado a objeto) levaram a novos paradigmas de programação e, por extensão, a novas linguagens de programação

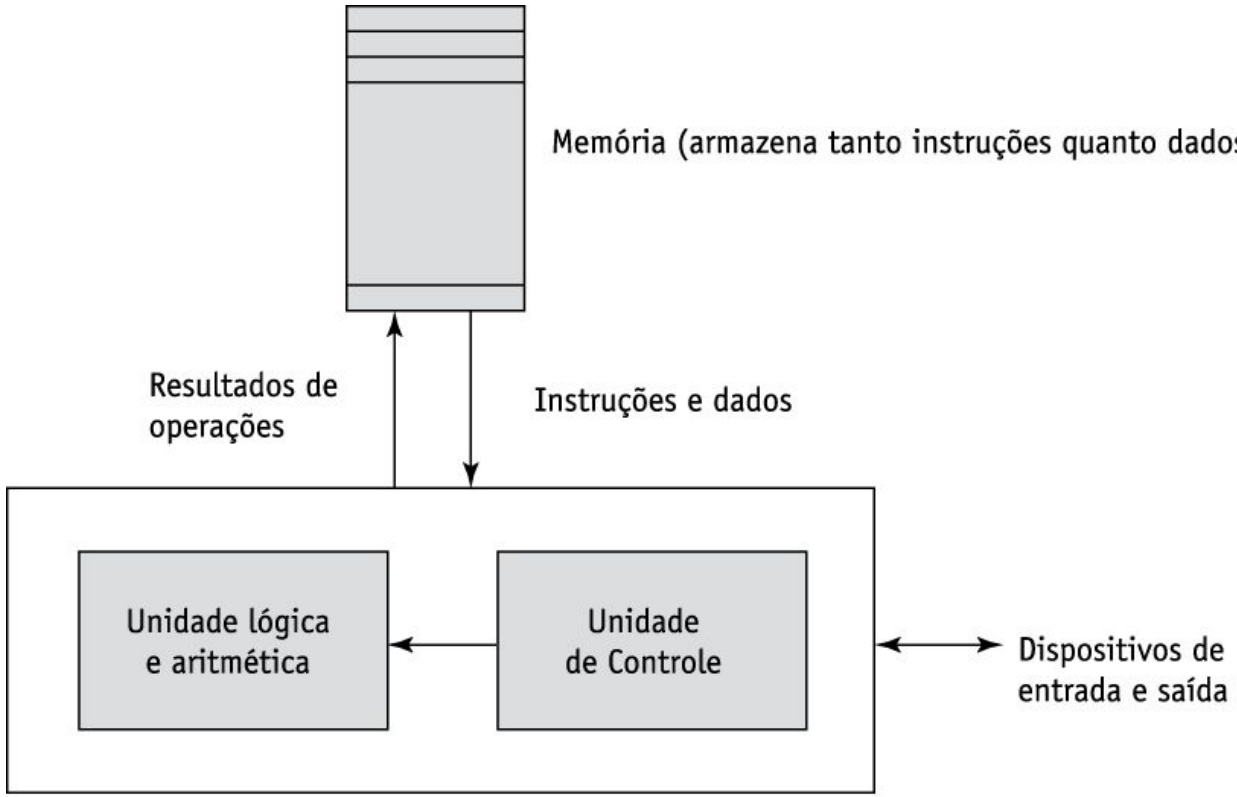


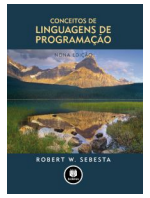

# Influências na arquitetura de computadores

- Principal arquitetura de computadores: von Neumann
- Linguagens imperativas, mais populares, por causa dos computadores von Neumann
  - Dados e programas armazenados na memória
  - A memória é separada da CPU
  - Instruções e dados são canalizadas a partir da memória para CPU
  - Base para linguagens imperativas
    - Variáveis modelam as células de memória
    - Sentenças de atribuição são baseadas na operação de envio de dados e instruções
    - Iteração é eficiente



# Arquitetura *Von Neumann*

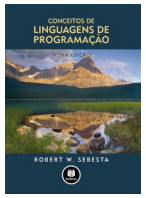





# Arquitetura *Von Neumann*

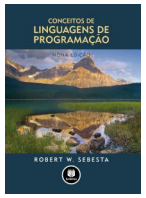
- Ciclo de obtenção e execução (em um computador com arquitetura von Neumann)

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



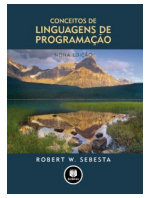

# Influências na metodologia de programa

- Anos 1950 e começo dos 1960: Aplicações simples; preocupação com a eficiência da máquina
- Final dos anos 60: Eficiência das pessoas se tornou importante; legibilidade, melhores estruturas de controle
  - Programação estruturada
  - Projeto descendente (*top-down*) e de refinamento passo a passo
- Final dos anos 70: Da orientação aos procedimentos para uma orientação aos dados
  - Abstração de dados
- Meio dos anos 80: Programação orientada a objetos
  - Abstração de dados + herança + vinculação dinâmica de métodos



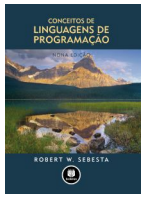

# Categorias de linguagens

- Imperativa
  - Características centrais são variáveis, sentenças de atribuição e de iteração
  - Inclui linguagens que suportam programação orientada a objeto
  - Inclui linguagens de *scripting*
  - Inclui as linguagens visuais
  - Exemplos: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Funcional
  - Principais meios de fazer os cálculos é pela aplicação de funções para determinados parâmetros
  - Exemplos: LISP, Scheme
- Lógica
  - Baseada em regras (regras são especificadas sem uma ordem em particular)
  - Example: Prolog
- De marcação/programação híbrida
  - Linguagens de marcação estendida para suportar alguma programação
  - Exemplos: JSTL, XSLT



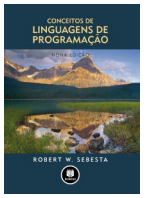

# *Trade-Offs* no projeto de linguagens

- Confiabilidade x custo de execução
  - Exemplo: Java exige que todas as referências aos elementos de um vetor sejam verificadas para garantir que os índices estão em suas faixas legais
- Legibilidade x facilidade de escrita
  - Exemplo: APL inclui um poderoso conjunto de operadores (e um grande número de novos símbolos), permitindo que computações complexas sejam escritas em um programa compacto, com o custo de baixa legibilidade
- Facilidade de escrita (flexibilidade) x confiabilidade
  - Exemplo: Ponteiros de C++ são poderosos e flexíveis, mas não são confiáveis



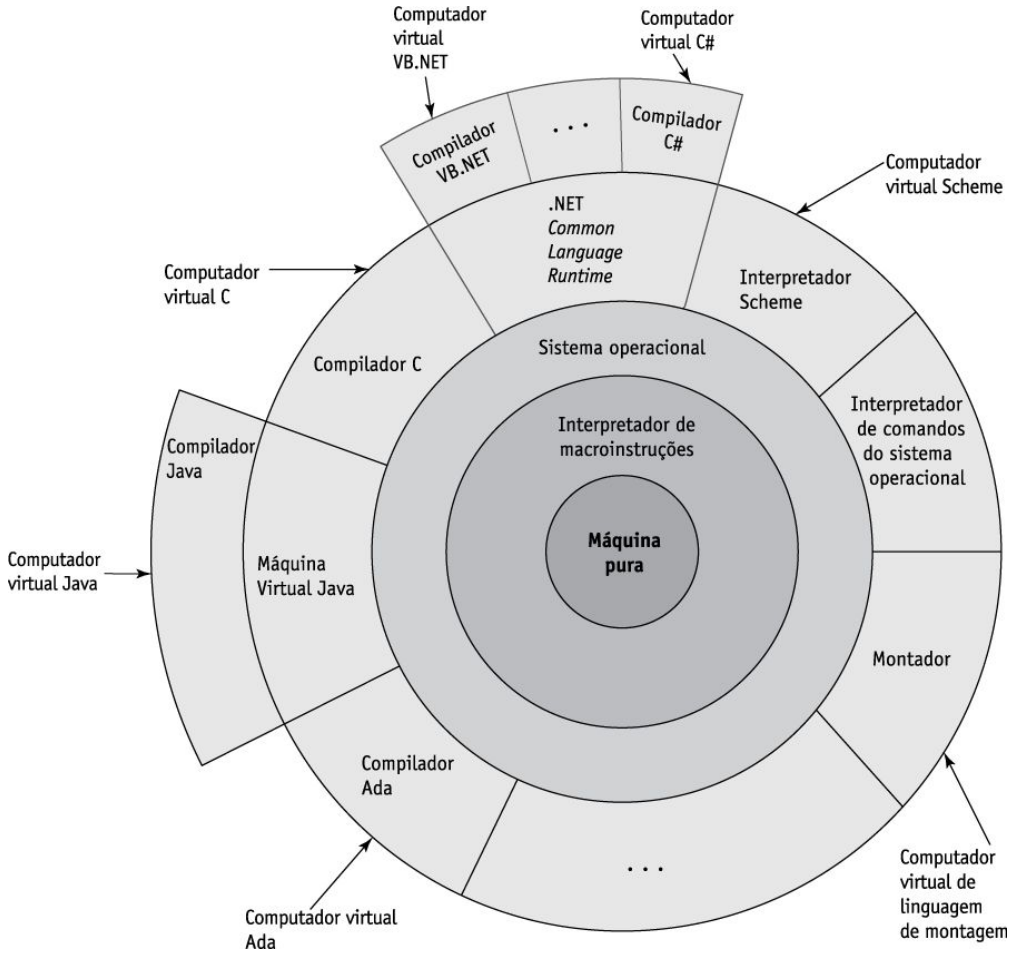

# Métodos de implementação

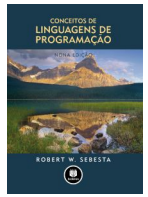
- Compilação
  - Programas são traduzidos para linguagem de máquina
- Interpretação pura
  - Programas são interpretados por outro programa chamado interpretador
- Sistemas de implementação híbridos
  - Um meio termo entre os compiladores e os interpretadores puros




# Visão em camadas de um computador

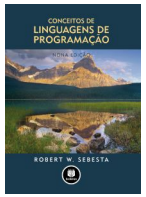
O Sistema operacional e as implementações de linguagem são colocados em camadas superiores à interface de linguagem de máquina de um computador



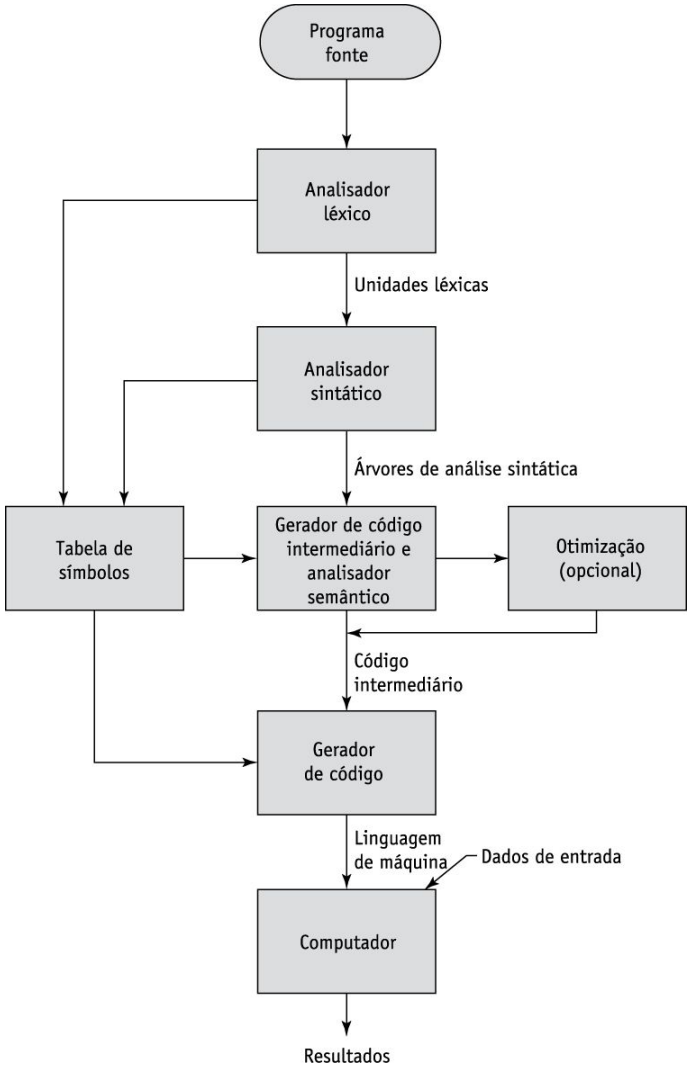


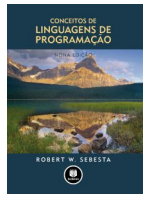

# Compilação

- Traduz programas (linguagem de fonte) em código de máquina (linguagem de máquina)
- Tradução lenta, execução rápida
- Processo de compilação tem várias fases:
  - análise léxica: agrupa os caracteres do programa fonte em unidades léxicas
  - análise sintática: transforma unidades léxicas em árvores de análise sintática (*parse trees*), que representam a estrutura sintática do programa
  - análise semântica: gera código intermediário
  - geração de código: código de máquina é gerado



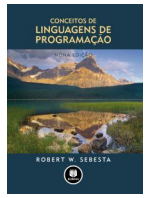

# O processo de compilação





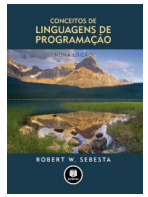

# Terminologias de compilação adicionais

- **Módulo de carga** (imagem executável): o código de usuário e de sistema juntos
- **Ligação e carga**: o processo de coletar programas de sistema e ligá-los aos programas de usuário



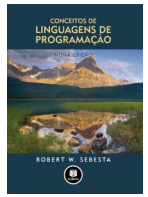

# Gargalo de *Von Neumann*

- A velocidade de conexão entre a memória de um computador e seu processador determina a velocidade do computador
- Instruções de programa normalmente podem ser executadas mais rapidamente do que a velocidade de conexão, o que resulta em um gargalo
- Conhecido como *gargalo de von Neumann*; é o fator limitante primário na velocidade dos computadores

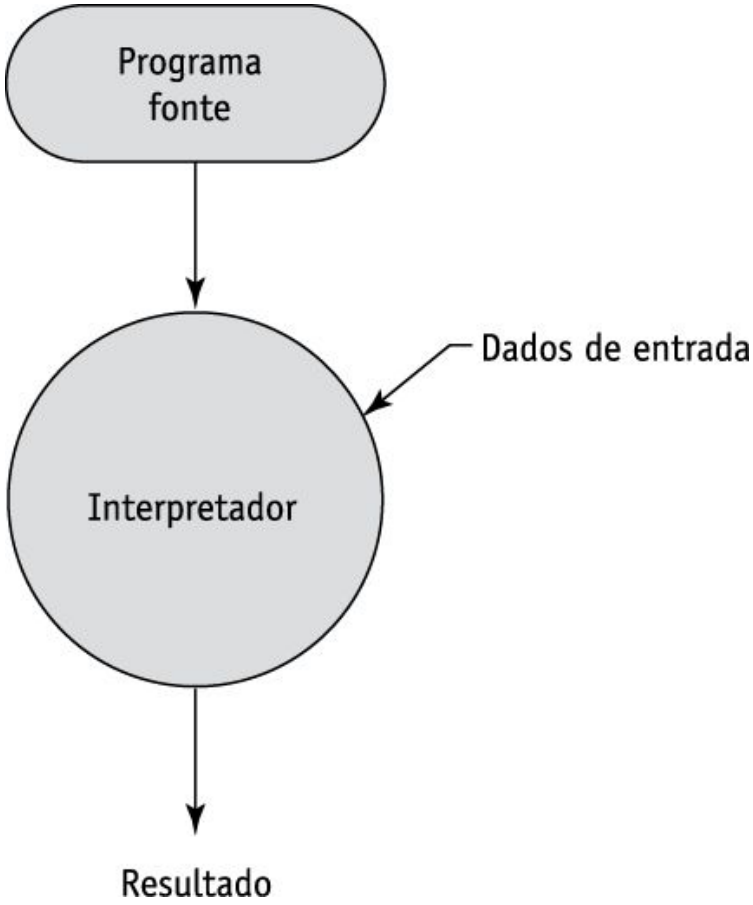


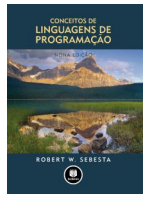

# Interpretação pura

- **Sem tradução**
- Fácil implementação de programas (mensagens de erro em tempo de execução podem referenciar unidades de código fonte)
- Execução mais lenta (tempo de execução de 10 a 100 vezes mais lento do que nos sistemas compilados)
- Geralmente requer mais espaço
- Raramente usada em linguagens de alto nível
- Volta significativa com algumas linguagens de *scripting* para a Web (como JavaScript e PHP)



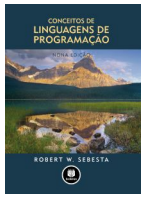

# Processo de interpretação pura



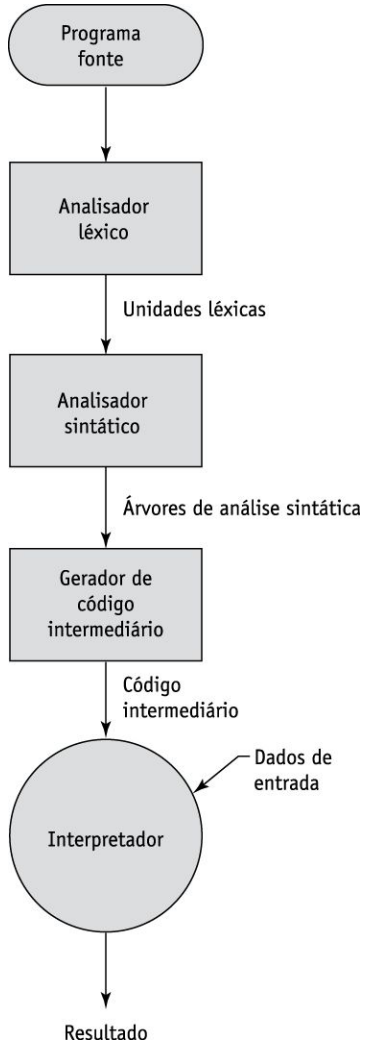


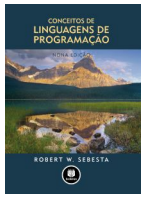

# Sistemas de implementação híbridos

- Um meio termo entre os compiladores e os interpretadores puros
- Uma linguagem de alto nível é traduzida para uma linguagem intermediária que permite fácil interpretação
- Mais rápido do que interpretação pura
- Exemplos
  - Programas em Perl eram parcialmente compilados para detectar erros antes da interpretação
  - As primeiras implementações de Java eram todas híbridas; seu formato intermediário, *byte code*, fornece portabilidade para qualquer máquina que tenha um interpretador de *bytecodes* e um sistema de tempo de execução associado (juntos, são chamados de Máquina Virtual Java)



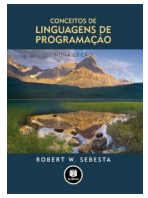
# Sistema de implementação híbrido





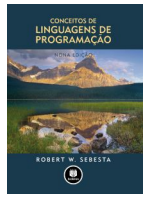

# Sistemas de implementação *Just-in-Time*

- Inicialmente traduz os programas para uma linguagem intermediária
- Então, compila os métodos da linguagem intermediária para linguagem de máquina quando esses são chamados
- A versão em código de máquina é mantida para chamadas subsequentes
- Sistemas JIT são agora usados amplamente para programas Java
- As linguagens .NET também são implementadas com um sistema JIT



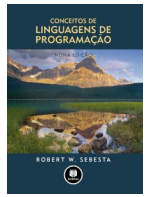

# Pré-processadores

- As instruções de pré-processador são comumente usadas para especificar que o código de outro arquivo deve ser incluído
- Um pré-processador é um programa que processa um programa imediatamente antes de o programa ser compilado para expandir macros embutidos
- Um exemplo conhecido: pré-processador de C
  - expande `#include, #define` e macros similares




# Ambientes de programação

- Coleção de ferramentas usadas no desenvolvimento de software
- UNIX
  - Um ambiente de programação mais antigo
  - Agora bastante usado por meio de uma interface gráfica com o usuário (GUI) que roda sobre o UNIX
- Microsoft Visual Studio .NET
  - Grande e complexo
- Usado para desenvolver software em qualquer uma das cinco linguagens .NET
- NetBeans
  - Usado primariamente para o desenvolvimento de aplicações Web usando Java, mas também oferece suporte a JavaScript, Ruby e PHP




# Resumo

- O estudo de linguagens de programação é valioso por diversas razões:
  - Aumenta nossa capacidade de usar diferentes construções ao escrever programas
  - Permite que escolhamos linguagens para os projetos de forma mais inteligente
  - Torna mais fácil o aprendizado de novas linguagens
- Critérios mais importantes para a avaliação de linguagens:
  - Legibilidade, facilidade de escrita, confiabilidade e custo geral
- As principais influências no projeto de linguagens são a arquitetura de máquina e as metodologias de projeto de software
- Os principais métodos de implementar linguagens de programação são a compilação, a interpretação pura e a implementação híbrida