Paradigmas de Linguagens de Programação

Heloisa de Arruda Camargo

Introdução

Motivos para estudar os conceitos de linguagens de programação

- Aumento da capacidade de expressar idéias
  - É difícil para as pessoas conceberem estruturas que não podem descrever, verbalmente ou por escrito.
  - Uma LP impõe limites quanto às estruturas de controle, de dados e abstrações, limitando também as formas de algoritmos possíveis.
  - Conhecer várias linguagens diminui essas limitações.

- Maior conhecimento para a escolha de linguagens apropriadas
  - programadores que conhecem uma ou duas linguagens tendem a usar sempre a mesma, com a qual estão familiarizados
  - Isso impede a escolha de linguagens mais adequadas para cada projeto
  - Impede também a obtenção de vantagens como clareza, facilidade de correção, etc...

- Maior capacidade para aprender novas linguagens
  - LP e ferramentas estão em contínua evolução
  - O aprendizado constante é necessário
  - Quanto mais se conhece sobre linguagens de programação mais fácil é aprender linguagens novas

- Melhor entendimento da importância da implementação
  - Entender questões de compilação ajuda compreender como certas construções da linguagem são executadas
  - Isso ajuda compreender questões de eficiência relativa (eficiência X versatilidade)
- Aumento da capacidade de projetar novas linguagens
  - (ou partes de um sistema cujo desenvolvimento siga os mesmos princípios)

- Diminuição da resistência ao uso de linguagens novas (e eventualmente melhores)
  - Nem sempre as linguagens mais populares são as melhores
  - As vezes, linguagens ruins continuam sendo usadas porque programadores e gerentes de desenvolvimento de software, que podem tomas decisões sobre qual linguagem utilizar não entendem as vantagens de linguagens diferentes

Alguns conceitos adotados para o estudo dos paradigmas

- Domínios de Aplicação
- Influências no projeto da linguagem
  - arquitetura de máquina
  - metodologia de desenvolvimento
- Método de implementação
  - Interpretada, Compilada, Híbrida

- Propriedades desejáveis das LPs
  - Legibilidade
  - Redigibilidade
  - Confiabilidade
  - Eficiência
  - Ortogonalidade
  - Reusabilidade
  - Modificabilidade
  - Portabilidade

Domínios de aplicação

- Aplicações científicas
  - Computadores foram projetados inicialmente para aplicações científicas (1940)
  - Características da aplicação:
    - Estruturas de dados simples
    - Exigem grande número de cálculos com ponto flutuante
  - Estruturas de dados mais comuns: **vetores e matrizes**
  - Estruturas de controle mais comuns: **iteração e seleção**
  - Característica desejável: **eficiência**
  - Exemplos: **FORTRAN, ALGOL60, PASCAL**

- Aplicações comerciais
  - Linguagens e computadores especiais foram desenvolvidos com esse propósito
  - Características: recursos para produzir relatórios elaborados, maneiras precisas de descrever e armazenar números decimais e caracteres e habilidade para especificar operações aritméticas decimais
  - Com o surgimento dos micros, aplicações de negócios passaram a ser feitas com sistemas de planilhas e de banco de dados
  - Exemplo: COBOL (1960)
  - (não são estudadas no curso)

- Aplicações de Inteligência Artificial
  - Características da aplicação: computações simbólicas ao invés de numéricas
  - Estruturas de dados mais comuns: **listas**
  - Estruturas de controle mais comuns: **recursão e unificação**
  - Característica desejável: **versatilidade (meta-programação)**
  - Exemplos: **LISP (funcional)**
    PROLOG (lógica)

- Programação de sistemas
  - Algumas linguagens foram projetadas para desenvolvimento de software básico: sistemas operacionais e ferramentas de suporte à programação.
  - Características da aplicação: ser rápida e ter recursos de baixo nível
  - Exemplos:
  - Para mainframes IBM: PL/S, dialeto do PL/I
  - Para digital: BLISS
  - Para Burroughs: ALGOL estendido
  - C (usada para escrever o sistema UNIX)

- Linguagens de script
  - São usadas colocando-se uma lista de comandos (script) em um arquivo para serem executados
  - Eram ferramentas comuns nos primeiros SO, controlados por linha de comando
  - Têm muitos recursos para processamento de textos
  - São interpretadas
  - Tornaram-se mais populares com o advento da WWW
  - Exemplos: awk, tcl, Perl
- Linguagens específicas
- Linguagens para web

Influências no projeto da linguagem

Arquitetura de Computadores

- A maioria das linguagens populares nos últimos anos foi projetada com base na arquitetura existente: Von Neumann
- São chamadas **linguagens imperativas**
- Características da arquitetura:
  - Dados e programas armazenados na mesma memória
  - CPU separada da memória, que realiza operações
  - Dados e instruções são canalizados para a CPU
  - Resultados voltam para memória

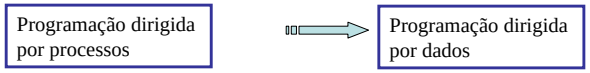
- Características das linguagens imperativas em função da arquitetura:
  - Variáveis como células de memória
  - Comando de atribuição
  - Repetição iterativa
- Iteração é rápida, porque as instruções estão em seqüência. Desencoraja o uso de recursão, apesar da recursão ser muitas vezes mais natural.
- Em LISP e PROLOG a essência da programação é diferente, não necessita o uso de variáveis e atribuições. São menos eficientes.

Metodologias de Programação

- movimento de programação estruturada: final dos anos 60, início dos 70
  - Motivo: custo maior da computação mudou de hardware para software.
- Como decorrência desse movimento surgiram:
  - Metodologias de desenvolvimento de software:
    - Projeto top-down
    - Refinamento passo a passo

  Assim foram identificadas as primeiras deficiências das linguagens:
  - Incompleteza na verificação de tipos
  - Comandos de controle inadequados

- Mudança de enfoque:

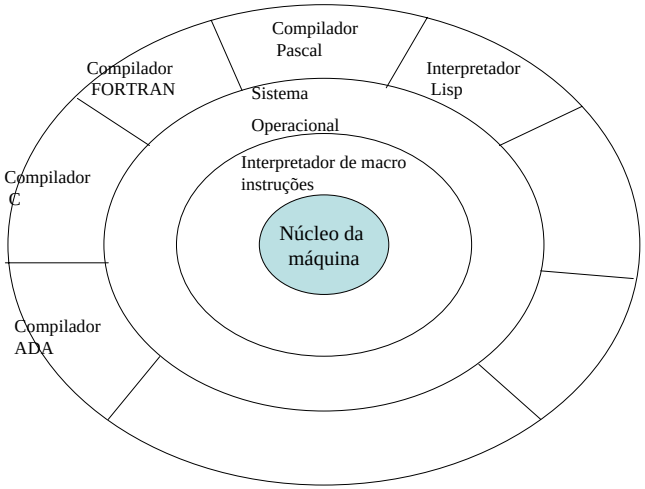


  - Ênfase no projeto de dados
  - Uso de tipos abstratos de dados

- A evolução da programação dirigida por dados originou no início dos anos 80 a metodologia orientada a objetos, que inclui:
  - Abstração de dados (encapsulamento)
  - Herança
  - Vinculação dinâmica de tipos
  - Exemplos: Smalltalk, Ada, Java, C++, CLOS, Prolog ++

Métodos de Implementação de Linguagens

- É a forma como uma linguagem de programação se comunica (é entendida e executada) com o computador
  - O computador possui uma linguagem de máquina de nível baixo que oferece operações primitivas
  - O software de sistema deve criar uma interface com os programas de nível mais alto.
- O sistema operacional e as implementações de linguagens são dispostos em camadas sobre a interface de linguagem de máquina de um computador.

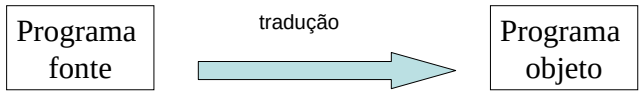


Métodos de implementação de linguagens:

– **Compilação**
– **Interpretação**
– **Implementação híbrida**

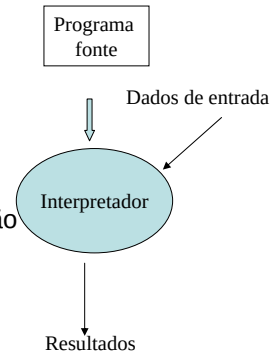
Compilação

- Programas são traduzidos para linguagem de máquina e são executados diretamente no computador
- Envolve dois processos distintos:
  – Tradução (compilação)
  – Execução
- A execução é iniciada depois que a tradução é concluída
- A execução não tem acesso ao programa fonte
- Vantagem: execução rápida



Interpretação

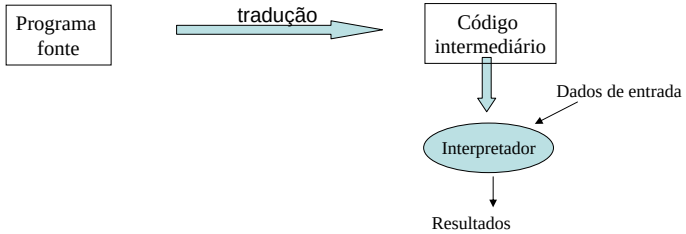
- O interpretador “executa” diretamente as instruções do programa fonte, sem traduzir para linguagem de máquina
- Simula, por software, uma máquina virtual onde o ciclo de execução entende os comandos da linguagem de alto nível
- Desvantagem: execução de 10 a 100 vezes mais lenta, devido ao passo de decodificação da instrução de alto nível, que é mais complexa
- Tem acesso ao programa fonte, para depuração ou mesmo para alterar o código sendo executado



Implementação Híbrida

- Mescla compilação com interpretação
- Programas fonte são traduzidos para uma linguagem intermediária que é interpretada
- Tem maior portabilidade que uma linguagem compilada
- São mais rápidas que uma linguagem interpretada - instruções intermediárias são projetadas para serem interpretadas facilmente

- Exemplo:
  - Pearl
  - Java (primeiras versões)
    - » Código intermediário :bytecodes
    - » Algumas implementações usam compilação JIT
    - » Alguns sistemas implementam compilação de bytecodes



OBS: Algumas linguagens têm versões interpretadas e compiladas

Linguagens imperativas X Linguagens Declarativas

Linguagens imperativas

- Fundamentadas na idéia de computação como um processo que realiza mudanças de estados
- Foco da programação: especificar COMO um processamento deve ser feito no computador
- Forte influência da arquitetura de máquina
- Variáveis são vistas como células de memória
- Conceitos fundamentais: variável, valor e atribuição

Linguagens declarativas

- Foco da programação: especificar O QUE deve ser feito, sem detalhes de operações da arquitetura da máquina
- Variáveis são vistas como incógnitas e não como células da memória
- Não existe operação de atribuição
- Programas são tipicamente especificações de relações ou funções

Alguns conceitos adotados para o estudo dos paradigmas

- Domínios de Aplicação
- Influência da arquitetura de máquina
  - Imperativa X Declarativa
- Influência da metodologia de desenvolvimento
  - Flexível X Eficiente
- Método de implementação
  - Interpretada, Compilada, Híbrida

- Propriedades desejáveis das LPs
  - Legibilidade
  - Redigibilidade
  - Confiabilidade
  - Eficiência
  - Ortogonalidade
  - Reusabilidade
  - Modificabilidade
  - Portabilidade

Paradigmas de Linguagens de Programação (descrição geral)

- Lógico
- Funcional
- Imperativo
- Orientado a Objetos
- Concorrente
- Linguagens de Marcação

Paradigma Lógico

- Processamento Simbólico (usa símbolos e conceitos ao invés de números e expressões)
- Declarativo
- Usa cláusulas da lógica de Primeira Ordem
- Estrutura principal: listas
- São baseadas em regras
- A ordem de especificação das regras não é fundamental
- A ordem de execução é determinada na própria execução
- Originalmente interpretado
- Área de aplicação: IA

Paradigma Funcional

- Baseada em funções matemáticas
- Combina funções elementares para formar funções mais complexas
- Estrutura principal: listas
- Voltada para programação simbólica
- Originalmente interpretado
- Área de aplicação: IA

Paradigma Imperativo

- Também chamado de Estruturado ou procedural
- Grande influência da arquitetura de máquina
- Componentes principais:
  - Variáveis como posições de memória
  - Comandos de atribuição
  - Repetição por iteração e seleção
- Programação dirigida por processos
- Área de aplicação: científica, sistema

Paradigma Orientado a Objetos

Enfoque no projeto de dados

- Provê recursos para:
  - Encapsulamento
  - Organização do acesso a dados
  - Mecanismo de herança
- Altamente modular e reusável
- Programação dirigida por dados
- Área de aplicação: geral

Paradigma Concorrente

- Facilita a construção de programas para execução concorrente (simultânea) de várias tarefas computacionais interativas
- As tarefas podem ser implementadas como programas separados ou como um conjunto de processos criados por um único programa.
- Essas tarefas também podem ser executadas por um único processador, vários processadores em um único equipamento ou processadores distribuídos por uma rede.
- Principais focos da programação concorrente:
  - interação e a comunicação correta entre as diferentes tarefas
  - coordenação do acesso concorrente aos recurso computacionais

Linguagens de marcação

- Normalmente não são consideradas um paradigma de programação
- é um conjunto de códigos aplicados a um texto ou a dados, com o fim de adicionar informações particulares sobre esse texto ou dado, ou sobre trechos específicos.
- HTML: linguagem de marcação amplamente usada para texto
- XHTML evolução da HTML, mais eficiente para separação entre a estrutura e o conteúdo de uma página de forma mais organizada e eficiente.
- XML: linguagem de marcação de dados. Envolve a codificação simples de seqüências de dados em um arquivo de computador no formato texto-puro, ou seja, capaz de ser lido tanto por pessoas quanto por máquinas.

Propriedades desejáveis das linguagens de programação

LEGIBILIDADE

- Facilidade para se ler e entender um programa;
- Melhora a tarefa de manutenção dos programas;
- Características que favorecem a legibilidade:
  - Simplicidade
  - recursos para estruturação de dados
  - Recursos para estruturação de controle
- Fatores que prejudicam a legibilidade:
  - Uso extensivo de ”goto´s” – permitem a programação não-estruturada
  - Estruturas de dados não adequadas, ou muito elementares
  - Sobrecarga de operadores – usar o mesmo símbolo com significados diferentes
  - Efeito colateral
  - Marcadores de blocos (begin-end, { - }) padronizados

Efeito Colateral

- Efeito causado por uma expressão, comando ou procedimento que permanece após a execução do mesmo e não é o objetivo principal dessa expressão, comando ou procedimento

- Alguns exemplos:
  - Alteração de variáveis globais no corpo de uma função
  - Expressão x++ do C++
  - Predicado corte (!) do prolog

REDIGIBILIDADE

- Facilidade de escrever o programa, permitindo ao programador se concentrar nos algoritmos centrais do programa sem se preocupar com aspectos não relevantes;

- Principal diferença entre linguagens de máquina e linguagens de alto nível;

- Características que favorecem a redigibilidade:
  - Simplicidade
  - suporte para abstração
    - Abstração de processo (ex: subprograma)
    - Abstração de dados (ex: dados estruturados, classes)
  - Expressividade
    - Uma grande quantidade de computação pode ser realizada em um programa muito pequeno;
    - Construções mais compactas;

- Fatores que prejudicam a redigibilidade:
  - Construções muito complexas
  - Muitas construções primitivas
  - Falta de recursos para abstração
  - Falta de estruturas de dados e controle

- OBS: LEGIBILIDADE e REDIGIBILIDADE podem ser conflitantes

CONFIABILIDADE

- Programa é confiável se ele se comportar de acordo com suas especificações sob todas as condições

- Características que favorecem a confiabilidade:
  - Verificação de tipos
    - Em tempo de compilação
    - Em tempo de execução
  - Tratamento de exceções
    - Capacidade do programa interceptar erros durante a execução tomar medidas corretivas e prosseguir

- Fatores que prejudicam a confiabilidade:
  - Permitir ações “perigosas”: não verificar intervalos de índices de arrays, aritmética de ponteiros, não verificar compatibilidade de tipos;
  - *Aliasing* – dois ou mais métodos ou nomes que fazem referência à mesma variável
  - Recursos pobres para escrita do programa

EFICIÊNCIA

- Está relacionada com o tempo de execução de um programa

- Algumas aplicações exigem que a execução seja rápida

- Em geral, fatores que melhoram a confiabilidade, abstração e legibilidade dos programas, diminuem a eficiência

ORTOGONALIDADE

- Capacidade da LP permitir ao programador combinar seus conceitos básicos sem que se produzam efeitos anômalos nessa combinação

- Quanto menor o número de exceções, maior a ortogonalidade

- Um recurso ortogonal é independente do contexto em que aparece no programa

- Características que favorecem a ortogonalidade:
  - Número pequeno de construções primitivas que podem ser combinadas de forma regular

- Fatores que prejudicam a ortogonalidade:
  - Número alto de exceções às regras da linguagem:
  - Operadores que não podem ser aplicados a qualquer tipo de operandos
  - Subprogramas que não podem retornar qualquer tipo de valor

REUSABILIDADE

- Propriedade de usar o mesmo código para várias aplicações
- Ligada a recursos de abstração da linguagem: subprogramas com parâmetros, bibliotecas de subprogramas, classes.

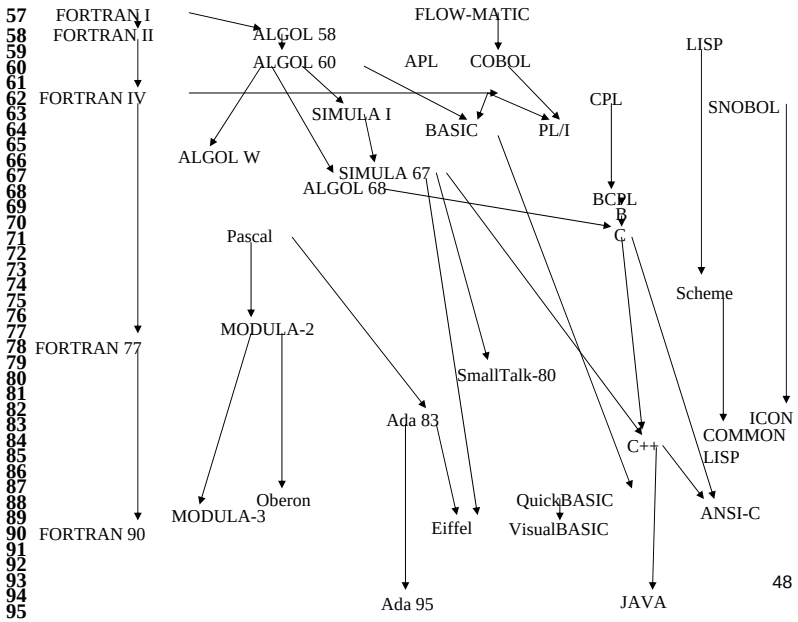
MODIFICABILIDADE

- Facilidade de alterar o programa sem implicações em outras partes do programa
- Características que favorecem a modificabilidade:
  - Constantes simbólicas
  - Separação entre interface e implementação na construção de subprogramas
  - Tipos abstratos de dados
  - Estrutura homogênea das unidades básicas da linguagem:
    - Regras do Prolog
    - Funções do LISP

PORTABILIDADE

- Propriedade dos programas escritos em uma LP se comportarem da mesma maneira independente do compilador, sistema operacional ou hardware utilizado
- Características que favorecem a portabilidade:
  - Implementação híbrida
  - Padronização da especificação da linguagem desde o seu projeto (pode prejudicar a eficiência)

Evolução das Principais Linguagens de Programação



Evolução das Principais Linguagens de Programação

- FORTRAN
  - primeira linguagem de alto nível compilada bem aceita
  - primeiras versões eram fortemente imperativas e a meta principal era da eficiência
  - Fortran 90 inclui modificações radicais: alocação dinâmica de arrays, ponteiros, registros, subprogramas recursivos e outras

- Programação Funcional – LISP
  - Originou da necessidade de processamento simbólico em listas encadeadas, para aplicações em Inteligência Artificial
  - A programação é realizada por aplicações de funções a argumentos
  - As primeiras versões eram estritamente funcionais, as outras incluem aspectos imperativos
  - Descendentes: Scheme, Common LISP

- ALGOL 60
  - Tentativa de definir uma linguagem universal
  - introduziu o conceito de estrutura em bloco, passagem de parâmetro por valor e por nome, procedimentos recursivos, arrays stack dinâmicos.
  - Foi a primeira linguagem descrita formalmente, usando a BNF, que tinha sido proposta recentemente.
  - linguagem muito sofisticada e poderosa, mas não conseguiu um uso generalizado

- COBOL – Linguagens Comerciais
  - Voltada para aplicações comerciais
  - Muito utilizada mas influenciou pouco outras linguagens
  - Adequada para a geração de relatórios de contabilidade

- BASIC
  - Prioriza rapidez de programação em vez de eficiência de execução
  - Muito utilizada mas pouco aceita nos meios científicos, por apresentar estutruras ruins nos programas
  - fácil de ser aprendida por principiantes e pode ser instalada em computadores com pouca memória
  - Versões contemporâneas muito utilizadas: Quick Basic e Visual Basic

- PL/I
  - Representa a primeira tentativa em grande escala de projetar uma linguagem que poderia ser usada para um amplo espectro de áreas de aplicação
  - linguagem muito complexa que inclui recursos de várias outras
  - apesar de ter sido a primeira a incluir diversas facilidades – entre essas os ponteiros, manipulação de exceções e a concorrência - hoje essas construções são consideradas mal projetadas
  -

- APL e SNOBOL
  - Não são baseadas em outras linguagens
  - São linguagens dinâmicas: tipificação dinâmica e alocação de armazenagem dinâmica
  - APL permite que arrays sejam manipulados como variáveis escalares
  - SNOBOL visa o processamento de texto

- SIMULA 67
  - Marcou a origem da abstração de dados
  - Derivada do ALGOL 60, projetada para aplicações de simulação
  - Oferece suporte a corrotinas e estrutura de classes

- ALGOL 68
  - inclusão dos tipos de dados definidos pelo usuário
  - tinha uma linguagem elegante e concisa para descrever a sintaxe, mas desconhecida.
  - Influenciou todas as linguagens imperativas desenvolvidas depois
- Modula-2
  - Baseada no Pascal e os principais recursos eram os módulos, que fornecem suporte para tipos abstratos de dados
- Modula-3
  - Adiciona ao Modula 2 classes e objetos para suporte à programação orientada a objetos, para manipulação de exceções, para coleta de lixo e para suporte à concorrência

- Pascal
  - projetada especificamente para ensinar programação
  - caracteriza-se por combinar simplicidade e expressividade
- C
  - originalmente projetada para programação de sistemas mas é adequada a uma ampla variedade de aplicações
  - projetada e implementada em 1972
  - tem muitas instruções de controle e facilidades de estruturação de dados
  - tem um rico conjunto de operadores que permitem uma expressividade alta
  - não possui verificação de tipo portanto é uma linguagem muito flexível e ao mesmo tempo insegura

- Oberon
  - Adicionou o recurso de extensão de tipos a Modula 2 para permitir programação orientada a objetos.
  - Retirou muitos recursos da Modula 2 tornado-se uma linguagem simples e pequena.
- Delphi
  - É derivado do Pascal.
  - É híbrido, semelhante ao C++, a medida que foi criado pela adição de suporte orientado a objeto a uma linguagem imperativa existente

- Prolog
  - linguagem de programação baseada na lógica de predicados de 1ª. ordem
  - Linguagem declarativa
  - A programação não é baseada em procedimentos, mas sim em declarações lógicas que estabelecem relações entre objeto
  - Criada para aplicações em Inteligência Artificial

- ADA
  - resultado do mais extensivo e dispendioso esforço de projeto de uma linguagem já realizado
  - oferece recursos para encapsular objetos de dados, facilidades para manipulação de exceções, unidades de programa genéricas e execução concorrente de unidades, entre outros
- ADA 95
  - Derivou de uma revisão da ADA, baseada nos requisitos: capacidades de interface, suporte para programação orientada a objetos, bibliotecas mais flexíveis e melhores mecanismos de controle para dados compartilhados

- SmallTalk
  - As unidades do programa são objetos, estruturas que encapsulam dados locais e um conjunto de operações chamados de métodos
  - tudo são objetos, desde constantes inteiras a grandes sistemas de software complexos
  - é também um ambiente completo de desenvolvimento de software

- C++
  - constrói facilidades de linguagem sobre o C para suportar grande parte daquilo em que o Smalltalk foi pioneiro
  - possui o mecanismo classe/objeto, permite herança simples e herança múltipla, sobrecarga de operadores e funções e vinculação dinâmica
  - Linguagem muito popular, herdou as inseguranças do C

- Eiffel
  - Híbrida, com recursos imperativos e orientados a objeto
- JAVA
  - Baseou-se no C++ mas foi especificamente projetado para ser menor, mais simples e mais confiável.
  - Não tem ponteiros, mas sim tipo de referência
  - não é possível escrever subprogramas independentes, possui somente herança simples e interfaces , e coleta de lixo implícita
  - Muito utilizada para programação para web
  - Oferece o mesmo poder que o C++, e é mais confiável.

Sites sobre História das Linguagens de Programação:

- http://www.levenez.com/lang/
- http://www.oreilly.com/pub/a/oreilly/news/languageposter_0504.html
- http://www.digibarn.com/collections/posters/tongues/