Paradigmas de Linguagens de Programação

Heloisa de Arruda Camargo

Introdução

- Motivos para estudar os conceitos de linguagens de programação
- Alguns aspectos 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)
  - Critérios para avaliação das LPs e características das linguagens que influenciam esses critérios
- Descrição geral dos Paradigmas de Linguagens de Programação
- Evolução das Principais Linguagens de Programação

Motivos para estudar os conceitos de linguagens de programação

- Aumento da capacidade de expressar ideias
  - É difícil para as pessoas conceberem estruturas que não podem descrever, verbalmente ou por escrito.
  - Programadores desenvolvendo software podem sofrer o mesmo tipo de limitação.
  - 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. Programadores podem aumentar o escopo dos processos de pensamento aprendendo novas construções de linguagens.

Motivos para estudar os conceitos de linguagens de programação

- 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
  - O conhecimento de uma variedade maior de linguagens e suas construções torna os programadores capazes de escolher a linguagem que possui características mais adequadas a cada problema.

Motivos para estudar os conceitos de linguagens de programação

- Maior capacidade para aprender novas linguagens
  - LP e ferramentas estão em contínua evolução.
  - O aprendizado constante é necessário.
  - O processo de aprendizado de uma nova linguagem pode ser difícil e demorado.
  - Quanto mais se conhece sobre conceitos fundamentais de linguagens de programação, mais fácil é aprender linguagens novas.

Motivos para estudar os conceitos de linguagens de programação

- 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)

Motivos para estudar os conceitos de linguagens de programação

- Melhor uso de linguagens já conhecidas
  - Linguagens de programação contemporâneas são extensas e complexas.
  - Muitos programadores não conhecem e não usam todos os recursos de uma linguagem.
  - Estudando conceito de LP o programador pode ser incentivado a aprender partes desconhecidas da linguagem que ele já usa.

Motivos para estudar os conceitos de linguagens de programação

- 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.
  - Se as pessoas que escolhem linguagens forem bem informadas, linguagens melhores tem mais chances de se tornarem populares.

TIOBE Programming community index

- O índice da comunidade de programação TIOBE é um indicador da popularidade das linguagens de programação.
- Atualizado todo mês.
- Mostra que a distribuição de uso das linguagens de programação está sempre mudando.
- O tamanho da lista e suas mudanças mostram que o desenvolvedor de software deve frequentemente aprender novas linguagens

(https://tiobe.com/tiobe-index/index.htm)

https://www.tiobe.com/tiobe-index/

Mar 2019	Mar 2018	Change	Programming Language	Ratings	Change
1	1		Java	14.880%	-0.06%
2	2		C	13.305%	+0.55%
3	4	↑	Python	8.262%	+2.39%
4	3	↓	C++	8.126%	+1.67%
5	6	↑	Visual Basic .NET	6.429%	+2.34%
6	5	↓	C#	3.267%	-1.80%
7	8	↑	JavaScript	2.426%	-1.49%
8	7	↓	PHP	2.420%	-1.59%
9	10	↑	SQL	1.926%	-0.76%
10	14	⇈	Objective-C	1.681%	-0.09%
11	**18**	⇈	**MATLAB**	**1.469%**	**+0.06%**
12	16	⇈	Assembly language	1.413%	-0.29%
13	11	↓	Perl	1.302%	-0.93%
14	20	⇈	R	1.278%	+0.15%
15	9	⇊	Ruby	1.202%	-1.54%
16	60	⇈	Groovy	1.178%	+1.04%
17	12	⇊	Swift	1.158%	-0.99%
18	17	↓	Go	1.016%	-0.43%
19	13	⇊	Delphi/Object Pascal	1.012%	-0.78%
20	15	⇊	Visual Basic	0.954%	-0.79%

Other programming languages

The complete top 50 of programming languages is listed below. This overview is published unofficially, because it could be the case that we missed a language. If you have the impression there is a programming language lacking, please notify us at tpci@tiobe.com. Please also check the overview of all programming languages that we monitor.

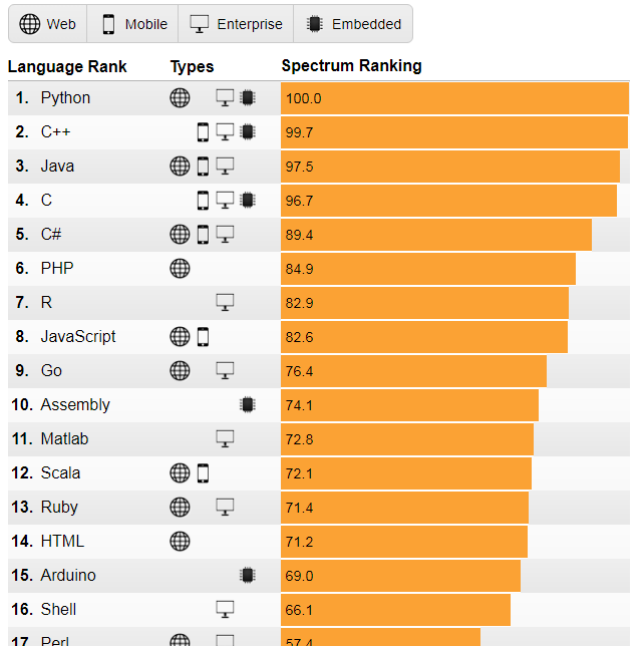
Position	Programming Language	Ratings
21	SAS	0.929%
22	PL/SQL	0.734%
23	Dart	0.724%
24	D	0.618%
25	Scratch	0.549%
26	COBOL	0.522%
27	Fortran	0.472%
28	Scala	0.467%
29	Lua	0.418%
30	Transact-SQL	0.415%
31	ABAP	0.395%
32	Lisp	0.372%
33	Prolog	0.368%
34	Scheme	0.355%
35	Rust	0.310%
36	Ada	0.310%
37	LabVIEW	0.309%

IEEE Spectrum 2018 Top Programming Languages

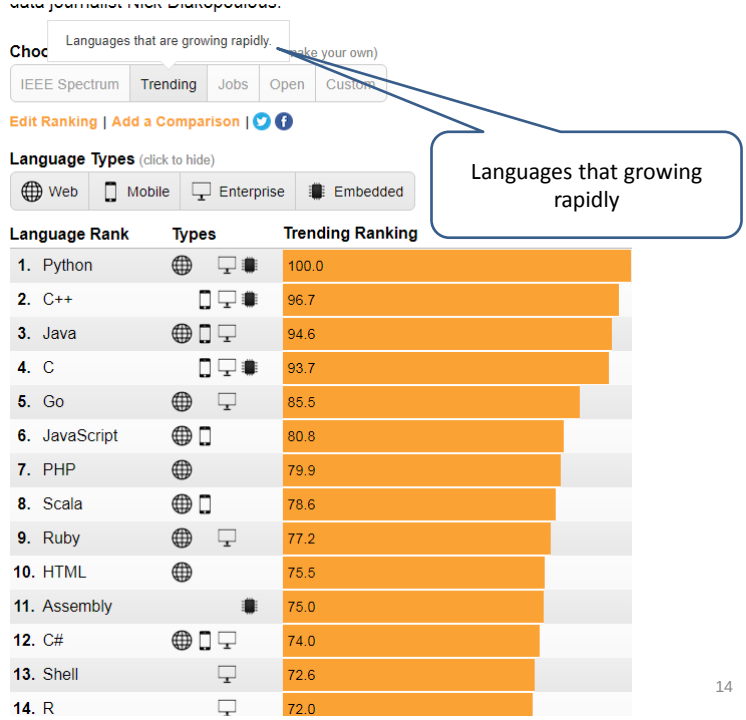
- O índice da IEEE Spectrum é um indicador da popularidade das linguagens de programação.
- Atualizado todo ano.
- Cálculo do índice:
  - **11 métricas ponderadas de 9 fontes**
- Principais mudanças:
  - **Python permanece na primeira posição (2017-2018)**
  - **Assembly aparece entre os 10 mais**
  - **Python é listada como linguagem para sistemas embarcados**

(http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages)

IEEE Spectrum 2018 Top Programming Languages



Language Rank	Types	Spectrum Ranking
1. Python	Web, Enterprise, Embedded	100.0
2. C++	Mobile, Enterprise, Embedded	99.7
3. Java	Web, Mobile, Enterprise	97.5
4. C	Mobile, Enterprise, Embedded	96.7
5. C#	Web, Mobile, Enterprise	89.4
6. PHP	Web	84.9
7. R	Enterprise	82.9
8. JavaScript	Web, Mobile	82.6
9. Go	Web, Enterprise	76.4
10. Assembly	Embedded	74.1
11. Matlab	Enterprise	72.8
12. Scala	Web, Mobile	72.1
13. Ruby	Web, Enterprise	71.4
14. HTML	Web	71.2
15. Arduino	Embedded	69.0
16. Shell	Enterprise	66.1
17. Perl	Web, Enterprise	57.4



Language Rank	Types	Trending Ranking
1. Python	Web, Enterprise, Embedded	100.0
2. C++	Mobile, Enterprise, Embedded	96.7
3. Java	Web, Mobile, Enterprise	94.6
4. C	Mobile, Enterprise, Embedded	93.7
5. Go	Web, Enterprise	85.5
6. JavaScript	Web, Mobile	80.8
7. PHP	Web	79.9
8. Scala	Web, Mobile	78.6
9. Ruby	Web, Enterprise	77.2
10. HTML	Web	75.5
11. Assembly	Embedded	75.0
12. C#	Web, Mobile, Enterprise	74.0
13. Shell	Enterprise	72.6
14. R	Enterprise	72.0

This app was originally developed in collaboration with *IEEE Spectrum* by data journalist Nick Diakopoulous.

Choose a

Languages that are in demand by employers.

IEEE Spectrum | Trending | Jobs | Open | Custom

Edit Ranking | Add a Comparison |

Language Types (click to hide)

Web | Mobile | Enterprise | Embedded

Languages that are in demand by employers

Language Rank	Types	Jobs Ranking
1. Python		100.0
2. Java		99.2
3. C		98.8
4. C++		94.6
5. C#		86.2
6. JavaScript		85.7
7. Assembly		83.4
8. PHP		83.1
9. HTML		81.3
10. Scala		76.5
11. Shell		76.3

Language Rank	Types	2018 Spectrum Ranking	2017 Spectrum Ranking
1. Python		100.0	100.0
2. C++		99.7	99.7
3. Java		97.5	99.4
4. C		96.7	97.4
5. C#		89.4	88.8
6. PHP		84.9	88.8
7. R		82.9	86.2
8. JavaScript		82.6	82.3
9. Go		76.4	77.2
10. Assembly		74.1	76.5
11. Matlab		72.8	74.4
12. Scala		72.1	73.9
13. Ruby		71.4	73.4
14. HTML		71.2	70.4
15. Arduino		69.0	70.0
16. Shell		66.1	68.6
17. Perl		57.4	68.0

Alguns aspectos 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
- Critérios para avaliação das LPs e características das linguagens que influenciam esses critérios

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**

Domínios de aplicação

- 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)

Domínios de aplicação

- 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)

Domínios de aplicação

- 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)

Domínios de aplicação

- Linguagens para web
  - A WWW faz uso de uma grande variedade de linguagens: desde HTML e XML (que não são linguagens de programação) até linguagens de uso geral como JAVA.
  - Páginas de conteúdo dinâmico fazem uso de uma funcionalidade que insere código de programação em um documento HTML, frequentemente na forma de linguagem de script, como JavaScript ou PHP.

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 sequência. 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.

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

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

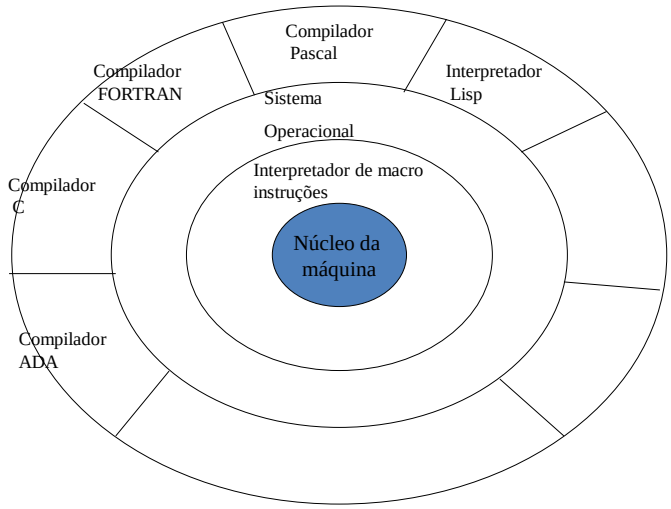
- Mudança de enfoque:

  - 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.



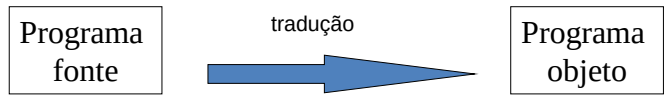
Métodos de implementação de linguagens:

- **Compilação**
- **Interpretação**
- **Implementação híbrida**

Compilação

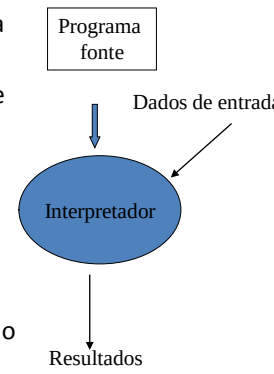
- Programas são traduzidos para linguagem de máquina e são executados diretamente no computador

- 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

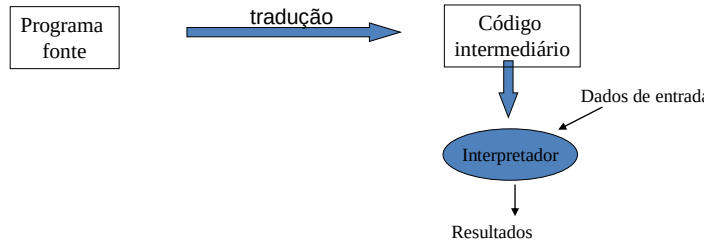
- O interpretador "executa" diretamente as instruções do programa fonte, sem traduzir para linguagem de máquina
- Simula, por software, uma máquina virtual onde o ciclo de execução entende os comandos da linguagem de alto nível
- Desvantagem: execução de 10 a 100 vezes mais lenta, devido ao passo de 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
- Exemplos: Perl, JAVA (primeiras versões)

- Java
  - Código intermediário :bytecodes
  - Algumas implementações usam compilação JIT (Just In Time)
  - Alguns sistemas implementam compilação de bytecodes



OBS: Algumas linguagens têm versões interpretadas e compiladas

Critérios de avaliação de linguagens

São critérios que permitem avaliar as propriedades das linguagens de programação quanto ao impacto provocado no processo de desenvolvimento de software, incluindo manutenção

LEGIBILIDADE
- Facilidade para se ler e entender um programa

FACILIDADE DE ESCRITA
- Facilidade de escrever o programa

CONFIABILIDADE
- Programa executa corretamente o que foi especificado

- CUSTO

Critérios são influenciados por características que as linguagens apresentam

	Critério		
Característica	Legibilidade	Facilidade de escrita	Confiabilidade
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipo			•
Tratamento de exceções			•
Sinonímia Restrita			•

OBS: Custo está apenas levemente relacionado com as características das linguagens

Critérios de avaliação de linguagens

- LEGIBILIDADE
  - Facilidade com que o programa pode ser lido e entendido
  - É um critério importante porque facilita a manutenção de programas
- Características que influenciam a legibilidade:
- Simplicidade
  - uma linguagem é mais simples quando possui menor número de construções

- Características que prejudicam a simplicidade:
  - Linguagem com muitas construções – programador aprende apenas um subconjunto da linguagem
  - Multiplicidade de recursos – ter mais de uma maneira de realizar a mesma operação

    count = count + 1

    count += 1
  - Sobrecarga de operadores – um único símbolo de operador tem mais de um significado

- Ortogonalidade
  - Significa que um número pequeno de construções primitivas podem ser combinadas em um número relativamente pequeno de formas para definir o controle a as estruturas de dados da linguagem.
  - A linguagem tem alta ortogonalidade quando todas as combinações possíveis de primitivas são legais e tem significado
  - Falta de ortogonalidade leva a um número alto de exceções nas regras da linguagem.
  - Exemplo: se em uma linguagem o tipo pointer não puder apontar para um determinado tipo de dados, como array.

- Tipos de dados
  - Tipos de dados adequados facilitam a leitura de um programa.
  - Exemplo: tipo booleano para representar um flag:

```
Fim = 1
 ou
Fim = true
```

- Projeto da sintaxe
  - Definições de sintaxe que podem prejudicar a legibilidade:
  - Restringir o tamanho dos identificadores
  - Palavras chave para criar comandos compostos e indicar o final de blocos de comandos:
    - C: chaves
    - Ada: end if e end loop

Este exemplo destaca uma situação de conflito entre simplicidade (menos palavras reservadas) e maior legibilidade (mais palavras reservadas).

Critérios de avaliação de linguagens

- FACILIDADE DE ESCRITA
  - É uma medida da facilidade que a linguagem oferece para escrever programas de um determinado domínio.
  - Características que afetam legibilidade também afetam facilidade de escrita.

Critérios de avaliação de linguagens

- Características que afetam a facilidade de escrita:

- Simplicidade e Ortogonalidade
  - Se a linguagem oferece um número muito alto de construções, o programador pode não conhecer todas elas, o que pode levar ao mal uso de recursos que seriam mais apropriados

  - Um número pequeno de construções primitivas e um conjunto consistente de regras para combiná-las (ortogonalidade) é melhor do que um número grande de primitivas

  - Ortogonalidade exagerada pode prejudicar a escrita, dificultando a detecção de erros.

Critérios de avaliação de linguagens

- Suporte para abstração

  - Abstração é a habilidade de definir e usar estruturas complicadas de forma que detalhes possam ser ignorados.

  - O grau de abstração permitido por uma linguagem e a naturalidade com que pode ser definida são importantes fatores para a facilidade de escrita.

  - Categorias de abstração:
    - Processos (subprogramas)
    - Dados (classes)

Critérios de avaliação de linguagens

- Expressividade
  - Se refere ao fato da linguagem oferecer formas convenientes de expressar operações e estruturas.
  - Exemplo:
  - Usar count ++ em vez de count = count + 1

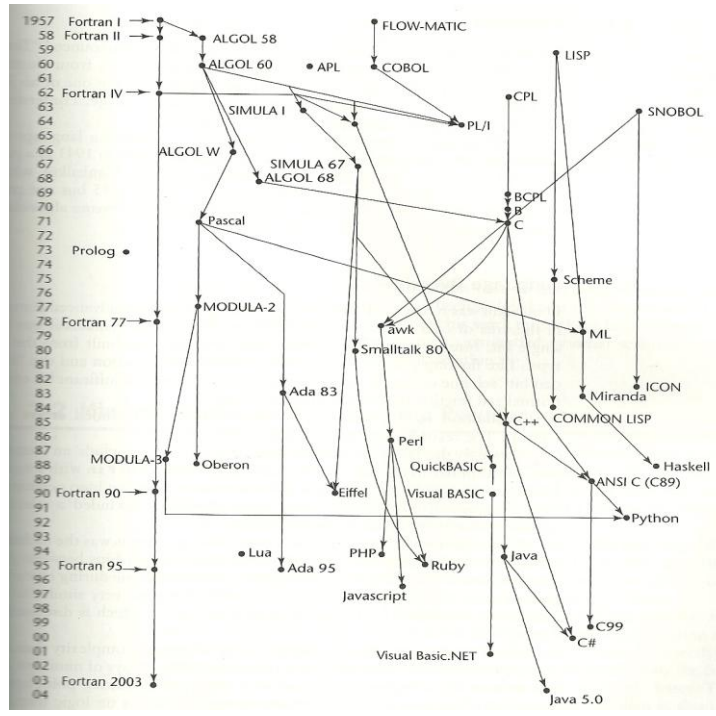
- CONFIABILIDADE
  - Programa é confiável se ele se comportar de acordo com suas especificações sob todas as condições
- Características que favorecem a confiabilidade:
  - Verificação de tipos
    - Em tempo de compilação
    - Em tempo de execução
  - Tratamento de exceções
    - Capacidade do programa interceptar erros durante a execução tomar medidas corretivas e prosseguir

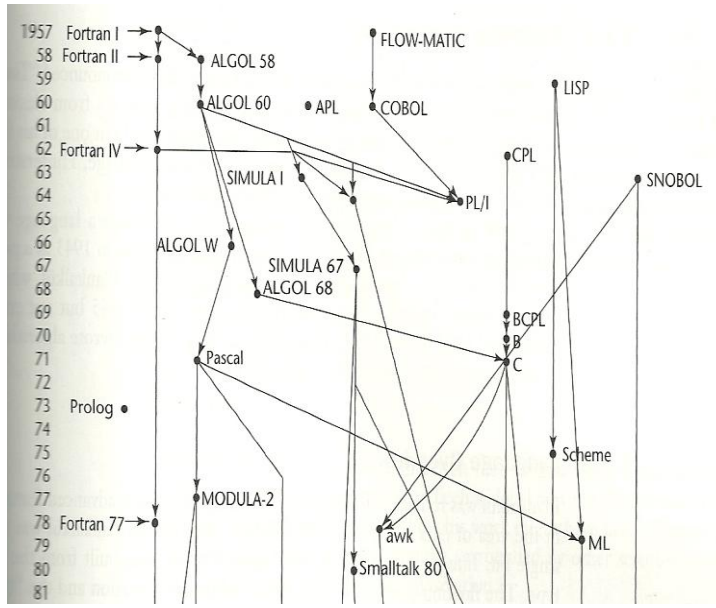
- Fatores que prejudicam a confiabilidade:
  - Permitir ações “perigosas”: não verificar intervalos de índices de arrays, aritmética de ponteiros, não verificar compatibilidade de tipos;
  - *Aliasing* – dois ou mais métodos ou nomes que fazem referência à mesma variável
  - Recursos pobres para escrita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

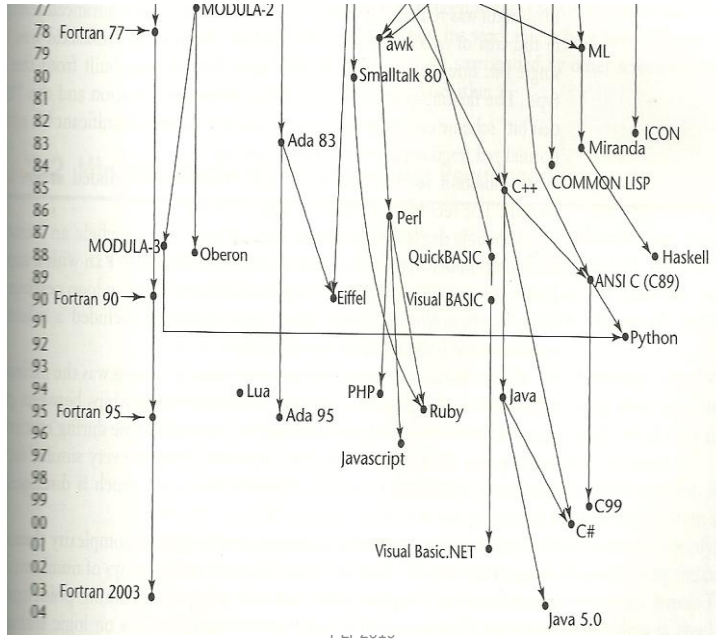
Alguns aspectos adotados para o estudo dos paradigmas

- 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
- Critérios para avaliação das LPs e características das linguagens que influenciam esses critérios

Evolução das Principais Linguagens de Programação




1957 Fortran I
58 Fortran II
59
60
61
62 Fortran IV
63
64
65
66
67
68
69
70
71
72
73 Prolog
74
75
76
77
78 Fortran 77
79
80
81
FLOW-MATIC
ALGOL 58
ALGOL 60
APL
COBOL
LISP
CPL
SNOBOL
SIMULA I
PL/I
ALGOL W
SIMULA 67
ALGOL 68
BCPL
B
C
Pascal
Scheme
MODULA-2
awk
ML
Smalltalk 80


78 Fortran 77
79
80
81
82
83
84
85
86
87
88
89
90 Fortran 90
91
92
93
94
95 Fortran 95
96
97
98
99
00
01
02
03 Fortran 2003
04
MODULA-2
awk
ML
Smalltalk 80
ICON
Ada 83
Miranda
COMMON LISP
C++
Perl
MODULA-3
Oberon
QuickBASIC
Haskell
ANSI C (C89)
Eiffel
Visual BASIC
Python
Lua
PHP
Java
Ada 95
Ruby
Javascript
C99
C#
Visual Basic.NET
Java 5.0

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 estruturas 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
  - VB.NET possui suporte à programação orientada a objetos

- 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 (necessárias para simulação) e estrutura de classes, que marcou o início da abstração de dados.

- 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.

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

Sites sobre História das Linguagens de Programação:

- http://www.levenez.com/lang/
- Historia das linguagens – poster atualizado até 2015
- http://www.oreilly.com/pub/a/oreilly/news/languageposter_0504.html
- Poster até 2005
- http://www.digibarn.com/collections/posters/tongues/
- http://rigaux.org/language-study/diagram.html
- Diagrama simplificado até 2004
- Diagrama completo (150 linguagens)
- http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm
- Lista com 2500 linguagens