Inatel

S201 – Paradigmas de Programação

ASPECTOS PRELIMINARES

Marcelo Vinícius Cysneiros Aragão

marcelovca90@inatel.br

MOTIVAÇÃO

Motivação

- É natural se perguntar qual o benefício ao estudar conceitos de linguagens de programação?
- São muitos! Aqui serão apresentados seis deles.

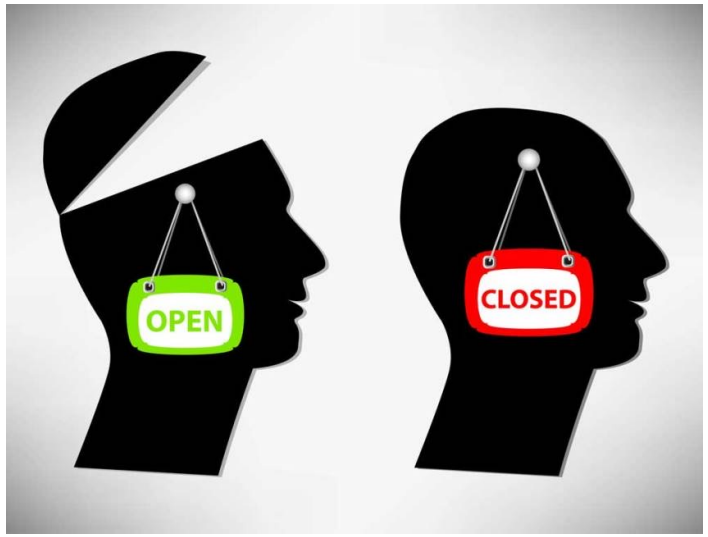
Motivação:
Capacidade aumentada para expressar ideias

- Acredita-se que a profundidade com a qual as pessoas podem pensar é influenciada pelo poder de expressividade da linguagem que elas usam para comunicar seus pensamentos.

TED – How language shapes the way we think | Lera Boroditsky

Motivação:
Capacidade aumentada para expressar ideias

- Programadores no processo de desenvolvimento de software apresentam a mesma limitação.
- A linguagem na qual eles desenvolvem impõe restrições nas estruturas de controle, estruturas de dados e abstrações que eles podem usar.
- Consequentemente, as formas dos algoritmos que eles constroem também são limitadas.
- Conhecer uma maior variedade recursos das linguagens de programação pode reduzir essas limitações no desenvolvimento de software.



Motivação:
Capacidade aumentada para expressar ideias

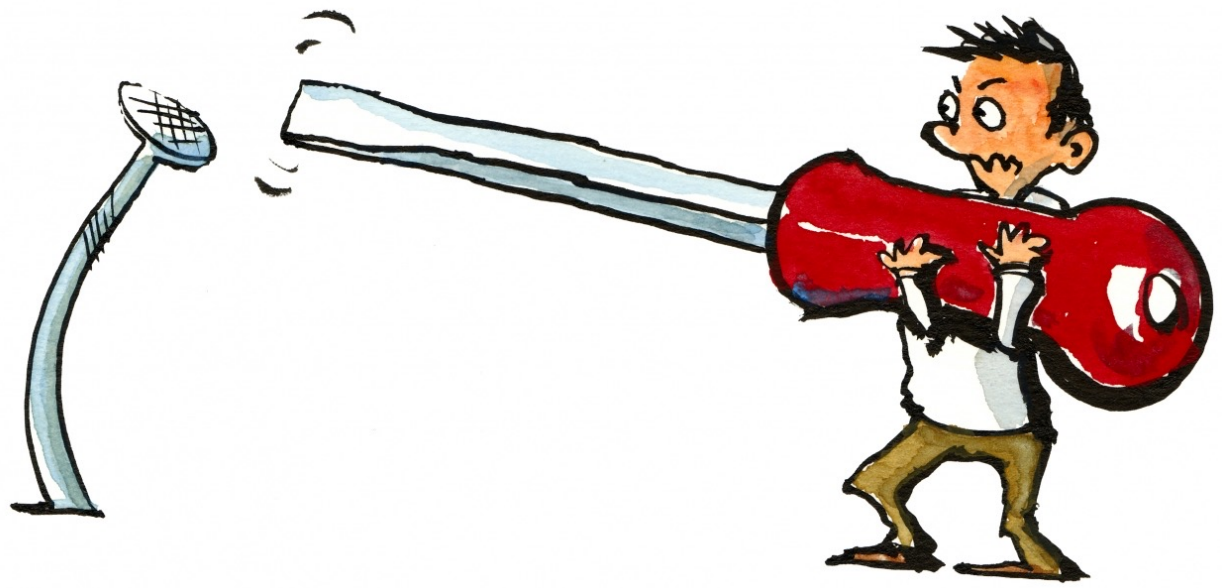
- Se sou forçado a usar uma linguagem que não oferece determinados recursos, então não há motivo em aprender linguagens que os tem.
- Isso não é verdade, pois normalmente essas construções podem ser simuladas em linguagens que não as oferecem suporte diretamente.
  - Exemplos: memórias associativas em Perl/C e do...while em C/Python

Motivação:
Embasamento para escolher linguagens adequadas

- Muitos programadores profissionais:
  - Tiveram pouca educação formal em ciência da computação;
  - Aprenderam programação por conta própria ou em programas de treinamento em suas empresas;
  - Receberam seu treinamento formal há muito tempo;
  - As linguagens aprendidas na época não são mais usadas;
  - Muitos recursos agora disponíveis em outras não eram amplamente conhecidos.

Motivação:
Embasamento para escolher linguagens adequadas

- O resultado é que muitos programadores, quando podem escolher a linguagem para um novo projeto, continuam a usar aquela com a qual estão mais familiarizados, mesmo que ela seja pobre na sua adequação ao projeto.



Motivação:
Embasamento para escolher linguagens adequadas

- Se esses programadores conhecessem uma faixa mais ampla de linguagens e construções de linguagens, estariam mais capacitados a escolher a que inclui os recursos melhor adaptados às características do problema em questão.

Motivação:
Habilidade aumentada para aprender novas linguagens

- A programação de computadores é algo relativamente novo.
- As metodologias de projeto, ferramentas de desenvolvimento de software e linguagens de programação ainda estão em evolução.
- O desenvolvimento de software uma profissão excitante, mas também exige aprendizado continuo.

Motivação:
Habilidade aumentada para aprender novas linguagens

- Indicador de popularidade da TIOBE: https://www.tiobe.com/tiobe-index/
  - Java, C e C++ foram as três linguagens mais populares em uso em out/2008.
  - Dezenas de outras linguagens foram amplamente usadas nessa mesma época.
  - A distribuição do uso das linguagens de programação está sempre mudando.
  - O tamanho da lista das linguagens em uso e a natureza dinâmica das estatísticas implicam que os desenvolvedores de software devem aprender linguagens diferentes constantemente.

Motivação:
Habilidade aumentada para aprender novas linguagens

- O processo de aprender uma nova linguagem de programação pode ser longo e difícil, especialmente para alguém que esteja confortável com apenas uma ou duas e nunca examinou os conceitos de linguagens de programação de um modo geral.
- Uma vez que um entendimento preciso dos conceitos fundamentais das linguagens tenha sido adquirido, fica mais fácil ver como esses conceitos são incorporados no projeto da linguagem aprendida.

- https://jaxenter.com/most-difficult-programming-languages-152590.html
- https://devops.com/learning-curve-computer-programming-languages/

Motivação:
Habilidade aumentada para aprender novas linguagens

- Por exemplo, programadores que entendem os conceitos de programação orientada a objetos aprenderão Java muito mais facilmente do que aqueles que nunca usaram tais conceitos.
- Analogamente nas linguagens naturais, quanto melhor se conhece a gramatica do idioma nativo, mais fácil será aprender uma nova língua.
- Além disso, aprender uma segunda língua também tem o efeito colateral benéfico de ensinar a você mais sobre a primeira.

Motivação:
Melhor entendimento da importância da implementação

- Frequentemente o entendimento de questões de implementação leva a por que as linguagens foram projetadas de uma determinada forma.
- Tal conhecimento costuma levar à habilidade de usar uma linguagem de maneira mais inteligente e como ela foi projetada para ser usada.

- https://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-processing-an-unsorted-array
- https://stackoverflow.com/questions/49617159/difference-between-and-in-c

Motivação:
Melhor entendimento da importância da implementação

- Podemos ser programadores melhores entendendo as escolhas entre construções de linguagens de programação e suas consequências.
- Exemplos:
  - Lidar com certos erros que só podem ser encontrados e corrigidos por programadores que conhecem detalhes de implementação relacionados.
  - Escolher construções alternativas com maior eficiência relativa para um programa em particular, evitando trechos de código ineficientes.

- https://stackoverflow.com/questions/45848858/short-circuit-evaluation-on-c
- https://stackoverflow.com/questions/300522/count-vs-length-vs-size-in-a-collection

Motivação:
Melhor uso de linguagens já conhecidas

- Muitas linguagens de programação de hoje são grandes e complexas.
- É incomum um programador conhecer e usar todos os recursos da linguagem que ele utiliza.
- Ao estudar os conceitos de linguagens de programação, os programadores podem aprender sobre partes antes desconhecidas e não utilizadas e começar a utilizá-las.

Motivação:
Avanço geral da computação

- Muitos acreditam que as linguagens de programação mais populares nem sempre são as melhores disponíveis.
- Em alguns casos, linguagens se tornaram amplamente usadas porque aqueles em posições de escolha não estavam suficientemente familiarizados com conceitos de linguagens de programação.
- Exemplo: muitas pessoas acreditam que teria sido melhor se o ALGOL 60 tivesse substituído o Fortran no início dos anos 1960, porque ele era mais elegante e tinha sentenças de controle muito melhores do que o Fortran, dentre outras razões.

Motivação:
Avanço geral da computação

- Ele não o substituiu, em parte por causa dos programadores e dos gerentes de desenvolvimento de software da época:
  - Muitos não entendiam o projeto conceitual do ALGOL 60;
  - Achavam sua descrição difícil de ler e mais difícil ainda de entender;
  - Não gostaram das vantagens da estrutura de blocos, da recursão e de estruturas de controle bem estruturadas;
  - Falharam em ver as melhorias do ALGOL 60 ao relação ao Fortran.
- Dentre outros fatores, o fato de os usuários de computadores não estarem cientes das vantagens da linguagem teve um papel significativo em sua rejeição.

DOMÍNIOS

Domínios

- Computadores tem sido aplicados a diversas áreas, desde controlar usinas nucleares até disponibilizar jogos eletrônicos em smartphones.
- Por causa dessa diversidade de uso, linguagens de programação com objetivos muito diferentes tem sido desenvolvidas.

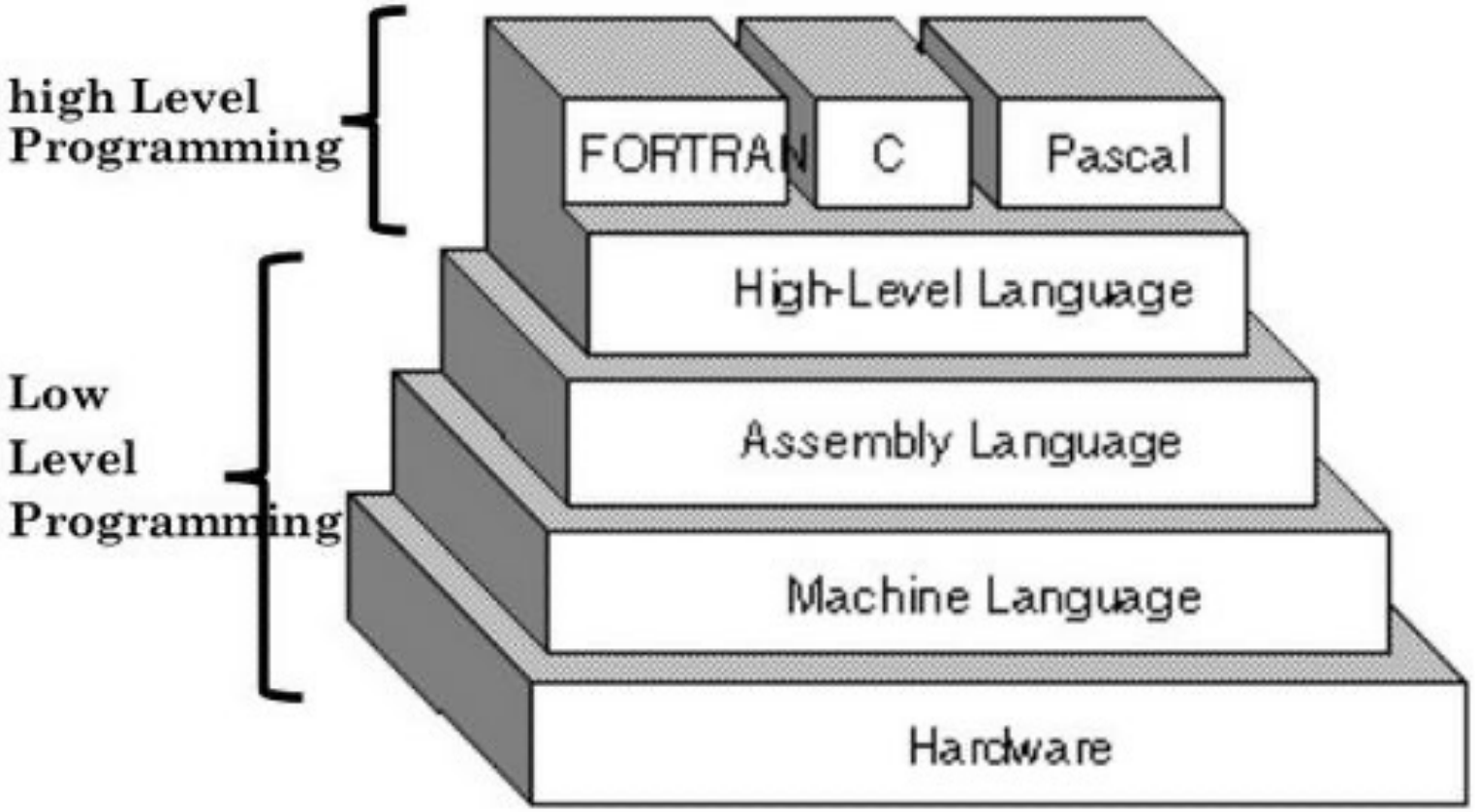
Domínios:
Aplicações científicas

- Os primeiros computadores digitais, que apareceram nos anos 1940, foram inventados e usados para aplicações cientificas.
  - https://www.pingdom.com/blog/retro-delight-gallery-of-early-computers-1940s-1960s/
- Normalmente, aplicações cientificas têm estruturas de dados relativamente simples, mas requerem diversas computações de aritmética de ponto flutuante.
  - https://stackoverflow.com/questions/329174/what-is-flop-s-and-is-it-a-good-measure-of-performance

Domínios:
Aplicações científicas

- Principais necessidades:
  - Estruturas de dados: vetores e matrizes
  - Estruturas de controle: laços de contagem e seleções.
- As primeiras linguagens de programação de alto nível inventadas para aplicações cientificas foram projetadas para suprir tais necessidades.
- Sua competidora era a linguagem *assembly* – logo, a eficiência era uma preocupação primordial.
- A primeira linguagem para aplicações cientificas foi o Fortran.
  - *http://www.softwarepreservation.org/projects/FORTRAN/paper/p25-backus.pdf*

Domínios:
Aplicações científicas



Domínios:
Aplicações científicas

- O ALGOL 60 e a maioria de seus descendentes também tinham a intenção de serem usados nessa área, apesar de terem sido projetados também para outras áreas relacionadas.

- Para aplicações cientificas nas quais a eficiência é a principal preocupação, como as que eram comuns nos anos 1950 e 1960, nenhuma linguagem subsequente é significativamente melhor do que o Fortran, o que explica por que o Fortran ainda é usado.
  - https://stackoverflow.com/questions/8997039/why-is-fortran-used-for-scientific-computing

Domínios:
Aplicações empresariais

- Usar computadores para aplicações comerciais começou nos anos 50.
- Computadores especiais foram desenvolvidos para esse propósito, com linguagens especiais.
- Características das linguagens de negócios:
  - Facilidades para a produção de relatórios elaborados;
  - Maneiras precisas de descrever e armazenar números decimais e caracteres;
  - Habilidade de especificar operações aritméticas decimais.

Domínios:
Aplicações empresariais

Domínios:
Aplicações empresariais

- A primeira linguagem de alto nível para negócios bem-sucedida foi o COBOL, com sua primeira versão aparecendo em 1960.
  - https://www.iso.org/standard/28805.html
- O COBOL ainda é a linguagem mais utilizada para tais aplicações.
  - *https://stackoverflow.blog/2020/04/20/brush-up-your-cobol-why-is-a-60-year-old-language-suddenly-in-demand/*
- Poucos avanços ocorreram nas linguagens para aplicações empresariais fora do desenvolvimento e evolução do COBOL.

Domínios:
Inteligência artificial

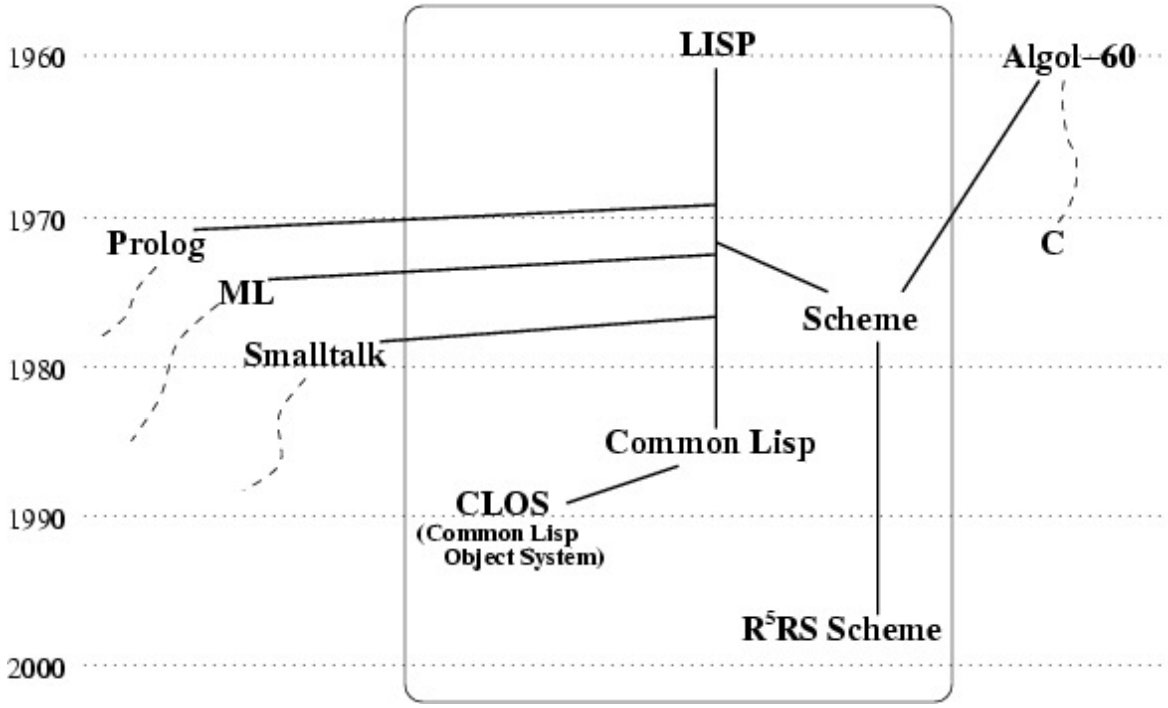
- Inteligência Artificial (IA) é uma ampla área de aplicações computacionais caracterizada pelo uso de computações simbólicas em vez de numéricas.
  - https://stackoverflow.com/questions/16395704/what-is-symbolic-computation
- Computações simbólicas são aquelas nas quais símbolos, compostos de nomes em vez de números, são manipulados.

Domínios:
Inteligência artificial

- Além disso, a computação simbólica é feita de modo mais fácil por meio de listas ligadas de dados do que por meio de vetores.
- Esse tipo de programação algumas vezes requer mais flexibilidade do que outros domínios de programação.
  - https://dl.acm.org/doi/pdf/10.1145/2930889.2930944
- Por exemplo, em algumas aplicações de IA, a habilidade de criar e de executar segmentos de código durante a execução é conveniente.

Domínios:
Inteligência artificial

- A primeira linguagem de programação amplamente utilizada desenvolvida para aplicações de IA foi LISP, em 1959.
- A maioria das aplicações de IA desenvolvidas antes de 1990 foi escrita em LISP ou em uma de suas parentes próximas.



Domínios:
Inteligência artificial

- No início dos anos 1970, entretanto, uma abordagem alternativa a algumas dessas aplicações apareceu – programação lógica usando a linguagem Prolog.
  - https://swish.swi-prolog.org/example/examples.swinb

- Mais recentemente, algumas aplicações de IA tem sido escritas em linguagens de sistemas como C e Python.

Domínios:
Programação de sistemas

- O sistema operacional e todas as ferramentas de suporte à programação de um sistema de computação são coletivamente conhecidos como seu software de sistema.
  - Tais aplicativos são usados quase continuamente, então devem ser eficientes.
- Além disso, tais aplicativos devem ter recursos de baixo nível que permitam aos aplicativos de software se comunicarem com dispositivos externos a serem escritos.

Domínios:
Programação de sistemas

- Nos anos 1960 e 1970, alguns fabricantes de computadores, como a IBM, a Digital e a Burroughs (agora UNISYS), desenvolveram linguagens especiais de alto nível orientadas à máquina para software de sistema que rodassem em suas máquinas.
  - Para os *mainframes* da IBM, a linguagem era PL/S, um dialeto de PL/I
  - Para a Digital, era BLISS, uma linguagem apenas um nível acima da *assembly*;
  - Para a Burroughs, era o ALGOL Estendido.

Domínios:
Programação de sistemas

- O sistema operacional UNIX é escrito quase todo em C99, o que o fez relativamente fácil de ser portado/movido para diferentes máquinas.
- Algumas das características de C fazem que ele seja uma boa escolha para a programação de sistemas.
  - C é uma linguagem de baixo nível, tem uma execução eficiente e não sobrecarrega o usuário com muitas restrições de segurança.
    - "Programadores de sistemas são excelentes e não acham que precisam destas restrições."
  - Alguns programadores que não trabalham com a programação de sistemas, no entanto, acham C muito perigoso para ser usado em sistemas de software grandes e importantes.

Domínios:
Software para a Web

- A *World Wide Web* é mantida por uma coleção de linguagens que vão desde linguagens de marcação, como XHTML, até linguagens de programação de proposito geral, como Java.
  - https://www.freecodecamp.org/news/stack-overflow-developer-survey-2020-programming-language-framework-salary-data/

Domínios:
Software para a Web

- Dada a necessidade universal por conteúdo dinâmico na Web, alguma capacidade de computação geralmente é incluída na tecnologia de apresentação de conteúdo.
  - Essa funcionalidade pode ser fornecida por código de programação embarcado em um documento XHTML.
  - Tal código é normalmente escrito com uma linguagem de *scripting*, como JavaScript ou PHP.
- Existem também algumas linguagens similares às de marcação que tem sido estendidas para incluir construções que controlam o processamento de documentos.
  - *https://commons.apache.org/proper/commons-jelly/*

CRITÉRIOS DE AVALIAÇÃO

Critérios de Avaliação

- Deseja-se examinar os **conceitos** fundamentais das diversas construções e **capacidades** das linguagens de programação.
- Também deseja-se avaliar esses recursos, focando em seu **impacto no processo de desenvolvimento de software**, incluindo a manutenção.
- Para isso, é necessário um conjunto de **critérios de avaliação**.

Critérios de Avaliação

- Tal lista é controversa pois é **praticamente impossível** fazer dois cientistas da computação concordarem com o valor de certas características das linguagens em relação às outras.
  - https://blogs.itemis.com/en/how-to-evaluate-a-programming-language-from-a-usability-point-of-view

- Entretanto, a maioria concordaria que tais critérios são importantes.
  - Alguns são amplos, e outros são construções específicas de linguagens.

Critérios de Avaliação

	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			•

Observação: o quarto critério principal é o custo, que não foi incluído na tabela porque é apenas superficialmente relacionado aos outros três critérios e às características que os influenciam.

Critérios de Avaliação: **Legibilidade**

	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			•

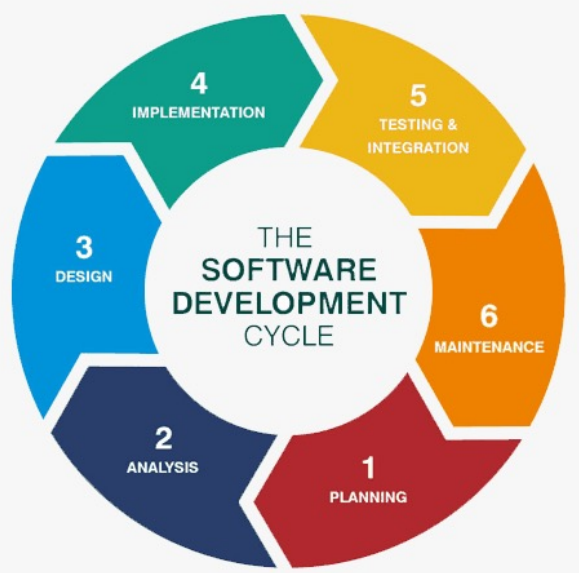
Observação: o quarto critério principal é o custo, que não foi incluído na tabela porque é apenas superficialmente relacionado aos outros três critérios e às características que os influenciam.

Critérios de Avaliação: **Legibilidade**

- É a facilidade com a qual os programas podem ser lidos e entendidos.
- Antes de 1970, o desenvolvimento de software era amplamente pensado em termos da escrita de código.
- As principais características positivas das linguagens de programação eram a eficiência e a legibilidade de máquina.
- As construções das linguagens foram projetadas mais do ponto de vista do computador do que do dos usuários.

Critérios de Avaliação: **Legibilidade**

- Nos anos 1970, porém, nasceu o conceito de ciclo de vida de software
  - A codificação foi relegada a um papel muito menor;
  - A manutenção foi reconhecida como uma parte principal ciclo, particularmente em termos de custo.
- Como a facilidade de manutenção é determinada, em boa parte, pela legibilidade dos programas, esta se tornou uma medida importante da qualidade dos programas e das linguagens de programação.
- Ocorreu uma transição de foco bem definida: da orientação à máquina à orientação às pessoas.

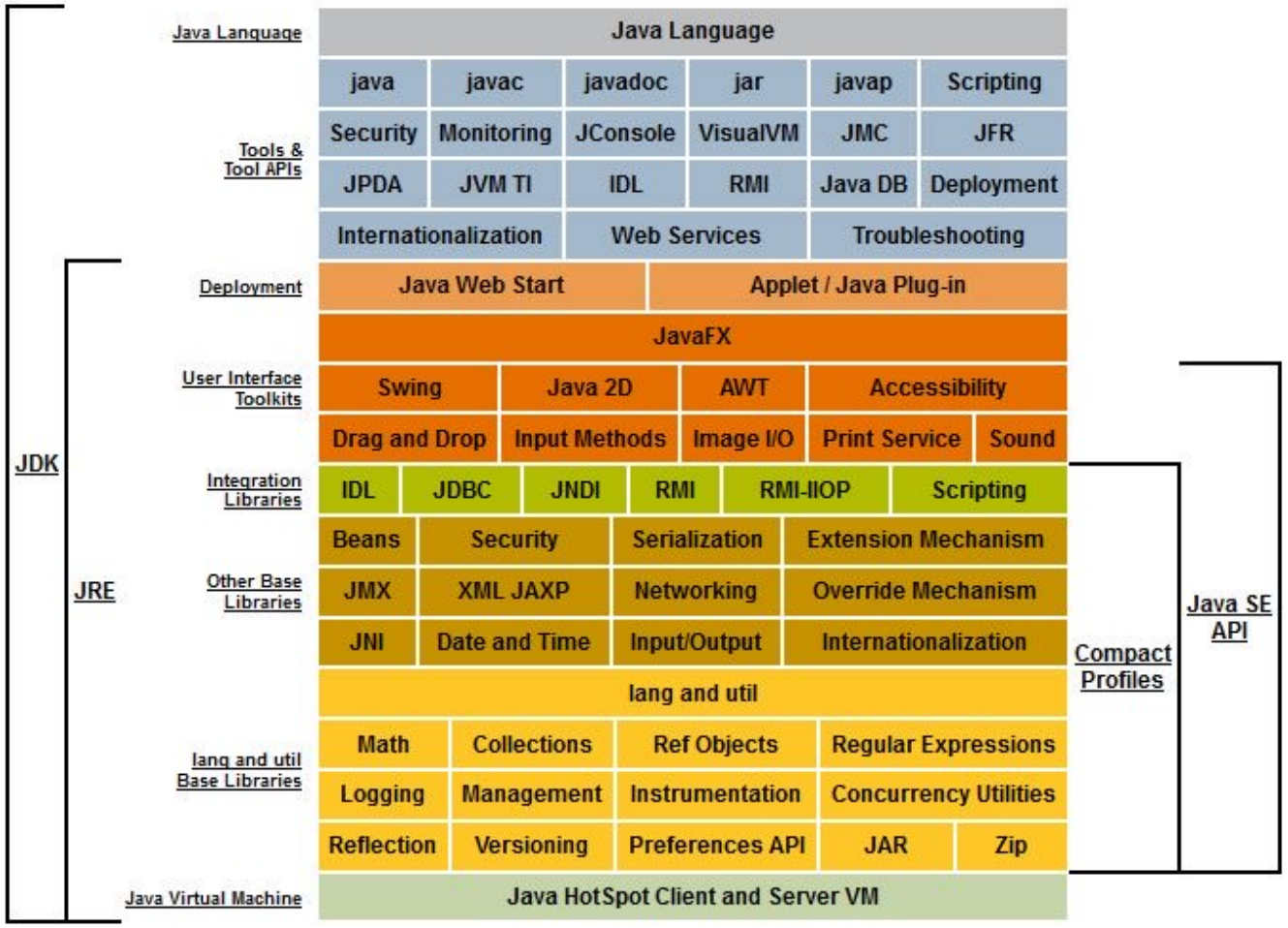


Critérios de Avaliação: **Legibilidade**

- A legibilidade deve ser considerada no contexto do problema.
  - Por exemplo, se um programa que descreve uma computação é escrito em uma linguagem que não foi projetada para tal uso, ele pode não ser natural e desnecessariamente complexo, tornando complicada sua leitura.
- A seguir serão apresentadas características que contribuem para a legibilidade de uma linguagem de programação.

Critérios de Avaliação: Legibilidade
Simplicidade geral

- A simplicidade geral de uma linguagem afeta fortemente sua legibilidade.
- **Problemas comuns:**
  - *Muitas construções básicas*
    - Os programadores que precisam usar uma linguagem grande aprendem um subconjunto dessa linguagem e ignoram outros recursos.
    - Exemplo: plataforma Java SE.



Critérios de Avaliação: Legibilidade
Simplicidade geral

- *Multiplicidade de recursos*
  - Existência mais de uma maneira de realizar uma operação. Ex.: incremento em Java
    - `count = count + 1`
    - `count += 1`
    - `count++`
    - `++count`

- *Sobrecarga de operadores*
  - Um operador c/ múltiplos significados (ex.: soma de inteiros e concatenação de vetores).
  - Discussão: *https://softwareengineering.stackexchange.com/questions/25154/i-dont-understand-the-arguments-against-operator-overloading*

Critérios de Avaliação: Legibilidade
Ortogonalidade

- **Definições**:
  - "Operações devem mudam apenas uma coisa, sem afetar as outras."
    - http://www.catb.org/~esr/writings/taoup/html/ch04s02.html#orthogonality
  - "Um conjunto relativamente pequeno de construções primitivas pode ser combinado a um número relativamente pequeno de formas para construir as estruturas de controle e de dados da linguagem."
- Cada possível combinação de primitivas é legal e significativa.

Critérios de Avaliação: Legibilidade
Ortogonalidade

- **Exemplo 1:**
  - Suponha que uma linguagem tenha quatro tipos primitivos de dados (inteiro, ponto flutuante, ponto flutuante de dupla precisão e caractere) e dois operadores de tipo (vetor e ponteiro).
  - Se os dois operadores de tipo puderem ser aplicados a eles mesmos e aos quatro tipos de dados primitivos, um grande número de estruturas de dados pode ser definido.

Critérios de Avaliação: Legibilidade
Ortogonalidade

- **Exemplo 2:**
  - Por exemplo, deve ser possível, em uma linguagem de programação que possibilita o uso de ponteiros, definir que um aponte para qualquer tipo específico definido na linguagem.
  - Entretanto, se não for permitido aos ponteiros apontar para vetores, muitas estruturas de dados potencialmente úteis definidas pelos usuários não poderiam ser definidas.

Critérios de Avaliação: Legibilidade
Ortogonalidade

- **Exemplo 3:**
  - Podemos ilustrar o uso da ortogonalidade como um conceito de projeto ao comparar um aspecto das linguagens *assembly* dos mainframes da IBM com a série VAX de minicomputadores.
  - Uma situação simples: adicionar dois valores inteiros de 32-bits que residem na memória ou nos registradores e substituir um dos valores pela soma.

Critérios de Avaliação: Legibilidade
Ortogonalidade

- **Exemplo 3:**
  - Os mainframes IBM têm duas instruções para esse propósito, com a forma
    - `A Reg1, memory_cell AR Reg1, Reg2`
    - Nesse caso, `Reg1` e `Reg2` representam registradores.
  - A semântica desses é
    - `Reg1 ← contents(Reg1) + contents(memory_cell)`
    - `Reg1 ← contents(Reg1) + contents(Reg2)`
  - A instrução de adição VAX para valores inteiros de 32-bits é
    - `ADDL operand_1, operand_2`
  - cuja semântica é
    - `operand_2 ← contents(operand_1) + contents(operand_2)`
    - Nesse caso, cada operando pode ser um registrador ou uma célula de memória.

Critérios de Avaliação: Legibilidade
Ortogonalidade

- **Exemplo 3:**
  - O projeto de instruções VAX é ortogonal no sentido de que uma instrução pode usar tanto registradores quanto células de memória como operandos.
    - Existem duas maneiras para especificar operandos, as quais podem ser combinadas de todas as maneiras possíveis.
  - O projeto da IBM não é ortogonal. Apenas duas combinações de operandos são legais (dentre quatro possibilidades), e ambas necessitam de instruções diferentes, `A` e `AR`.
  - O projeto da IBM é mais restrito e tem menor facilidade de escrita.
    - Ex.: você não pode adicionar dois valores e armazenar a soma em um local de memória.
    - Além disso, o projeto da IBM é mais difícil de aprender por causa das restrições e da instrução adicional.

Critérios de Avaliação: Legibilidade
Ortogonalidade

- A ortogonalidade é fortemente relacionada à simplicidade:

  "quanto mais ortogonal o projeto de uma linguagem, menor é o numero necessário de exceções às regras da linguagem".

- Menos exceções significam um maior grau de regularidade no projeto, o que torna a linguagem mais fácil de aprender, ler e entender.
- Qualquer um que tenha aprendido uma parte significativa da língua inglesa pode testemunhar sobre a dificuldade de aprender suas muitas exceções de regras.

Critérios de Avaliação: Legibilidade
Tipos de dados

- A presença de mecanismos adequados para definir tipos e estruturas de dados é outro auxilio significativo à legibilidade.
- **Exemplo:**
  - Suponha que um tipo numérico seja usado como uma *flag* pois não existe um tipo booleano na linguagem. Nela, poderíamos ter uma atribuição como:
    - `timeout = 1`
  - O significado dessa sentença não é claro.
  - Em uma linguagem que inclui tipos booleanos, teríamos:
    - `timeout = true`
  - O significado dessa sentença é perfeitamente claro.

Critérios de Avaliação: Legibilidade
Projeto da sintaxe

- A sintaxe (ou forma) dos elementos de uma linguagem tem um efeito significativo na legibilidade dos programas.
- **Exemplos:**
  - ***Formato dos identificadores***
    - Restringir os identificadores a tamanhos muito curtos piora a legibilidade.
    - Se os identificadores podem ter no máximo seis caracteres, como no Fortran 77, é praticamente impossível usar nomes conotativos para variáveis.
      - https://docs.oracle.com/cd/E19957-01/805-4939/6j4m0vn6i/index.html
    - Um exemplo mais extremo é o ANSI BASIC, no qual um identificador poderia ser formado por apenas uma letra ou por uma letra seguida de um digito.
      - https://www.ecma-international.org/publications-and-standards/standards/ecma-55/

Critérios de Avaliação: Legibilidade
Projeto da sintaxe

- ***Palavras especiais***
  - A aparência de um programa e sua legibilidade são fortemente influenciadas pela forma das palavras especiais de uma linguagem (por exemplo, **`while`**, **`class`** e **`for`**).
    - C/C++ usam chaves para sentenças compostas, que sempre terminam da mesma forma;
    - Ada utiliza **`end if`** para terminar uma seleção e **`end loop`** para terminar uma repetição.;
    - Esse é um exemplo do conflito entre a simplicidade resultante ao usar menos palavras reservadas (C/C++), e a maior legibilidade ao usar mais palavras reservadas (Ada).
  - Outra questão importante é se as palavras especiais de uma linguagem podem ser usadas como nomes de variáveis de programas.
    - Se puderem, os programas resultantes podem ser bastante confusos.

Critérios de Avaliação: Legibilidade
Projeto da sintaxe

- ***Forma e significado***
  - Projetar sentenças cuja aparência ao menos indique seu propósito é uma ajuda óbvia à legibilidade. A semântica, ou significado, deve advir diretamente da sintaxe, ou da forma.
  - Muitas vezes, esse princípio é violado por duas construções idênticas ou similares na aparência, mas com significados diferentes, talvez dependendo do contexto.
    - Ex.: em C, o significado da palavra reservada `static` depende do contexto no qual ela aparece.
      - Se for usada na definição de uma variável dentro de uma função, significa que a variável é criada em tempo de compilação.
      - Se for usada na definição de uma variável fora das funções, significa que a variável é visível apenas no arquivo no qual sua definição aparece; ou seja, não é exportada.

Critérios de Avaliação: **Facilidade de escrita**

	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			•

Observação: o quarto critério principal é o custo, que não foi incluído na tabela porque é apenas superficialmente relacionado aos outros três critérios e às características que os influenciam.

Critérios de Avaliação: **Facilidade de escrita**

- É a medida do quão facilmente uma linguagem pode ser usada para criar programas para um domínio.
- A maioria das características de linguagem que afetam a legibilidade também afeta a facilidade de escrita.
- Isso é derivado do fato de que o processo de escrita de um programa requer que o programador releia sua parte já escrita.

Critérios de Avaliação: **Facilidade de escrita**

- Como na legibilidade, a facilidade de escrita deve ser considerada no contexto do domínio de problema alvo de uma linguagem.
- Não é razoável comparar a facilidade de escrita de duas linguagens no contexto de uma aplicação em particular quando uma delas foi projetada para tal aplicação e a outra não.
- Exemplo ao usar C e Visual Basic (VB) para:
  - Criação de programas com interface gráfica com o usuário;
  - Escrita de programas de sistema (como um sistema operacional).

Critérios de Avaliação: Facilidade de escrita
Simplicidade e ortogonalidade

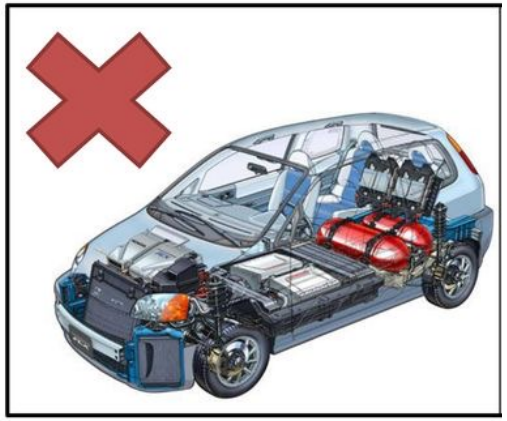
- Se uma linguagem tem um grande número de construções, alguns programadores não estarão familiarizados com todas.
  - Isto pode levar ao uso incorreto de alguns recursos e a uma utilização escassa de outros que podem ser mais elegantes e/ou eficientes do que os usados.
  - Pode até mesmo ser possível usar recursos desconhecidos acidentalmente, com resultados inesperados.
- Logo, um número menor de construções primitivas e um conjunto de regras consistente para combiná-las (isso é, ortogonalidade) é muito melhor do que diversas construções primitivas.

Critérios de Avaliação: Facilidade de escrita
Simplicidade e ortogonalidade

- Pode-se projetar uma solução para um problema complexo após aprender apenas um conjunto simples de construções primitivas.
- Entretanto, muita ortogonalidade pode prejudicar a facilidade de escrita.
  - Erros em programas podem passar despercebidos quando praticamente quaisquer combinações de primitivas são legais.
  - Isso pode levar a certos absurdos no código que não podem ser descobertos pelo compilador.

Critérios de Avaliação: Facilidade de escrita
Suporte à abstração

- Significa a habilidade de definir e usar estruturas ou operações complicadas de forma a permitir que muitos dos detalhes sejam ignorados.
- A abstração é um conceito fundamental no projeto atual de linguagens de programação.
- O grau de abstração permitido e a naturalidade de sua expressão são importantes para a facilidade de escrita.
- As linguagens de programação podem oferecer suporte a duas categorias de abstrações: *processos* e *dados*.



Critérios de Avaliação: Facilidade de escrita
Suporte à abstração

- Exemplo de abstração de processos:
  - Uso de um subprograma para implementar um algoritmo de ordenação necessário diversas vezes em um programa.
    - Sem o subprograma, o código de ordenação teria de ser replicado em todos os lugares onde fosse preciso, o que tornaria o programa muito mais longo e tedioso de ser escrito.
    - Talvez o mais importante, se o subprograma não fosse usado, o código que usava o subprograma de ordenação estaria mesclado com os detalhes do algoritmo de ordenação, obscurecendo o fluxo e a intenção geral do código.

Critérios de Avaliação: Facilidade de escrita
Suporte à abstração

- Exemplo de abstração de dados:
  - Árvore binária que armazena dados inteiros em seus nós.
    - Tal árvore poderia ser implementada em uma linguagem que não oferece suporte a ponteiros e gerenciamento de memória dinâmica usando uma *heap* (como no Fortran 77), com o uso de três vetores inteiros paralelos, onde dois dos inteiros são usados como índices para especificar nos filhos.
    - Em C++ e Java, elas podem ser implementadas utilizando uma abstração de um nó de árvore na forma de uma simples classe com dois ponteiros (ou referências) e um inteiro.

Critérios de Avaliação: Facilidade de escrita
Suporte à abstração

- A naturalidade da última representação torna mais fácil escrever um programa que usa árvores binárias nessas linguagens do que em Fortran 77.
- É uma simples questão do domínio da solução do problema da linguagem ser mais próxima do domínio do problema.
- O suporte geral para abstrações é um fator importante na facilidade de escrita de uma linguagem.

Critérios de Avaliação: Facilidade de escrita
Expressividade

- A expressividade em uma linguagem pode se referir a diversas características.
- Em uma linguagem como APL, expressividade significa a existência de operadores muito poderosos que permitem muitas computações com um programa muito pequeno.
- Em geral, uma linguagem expressiva especifica computações de uma forma conveniente, em vez de deselegante.

Critérios de Avaliação: Facilidade de escrita
Expressividade

- Exemplos:
  - Em C, a notação count++ é mais conveniente e menor do que count = count + 1.
  - O operador booleano **and then** em Ada é uma maneira conveniente de especificar avaliação em curto-circuito de uma expressão booleana.
  - A inclusão da sentença **for** em Java torna a escrita de laços de contagem mais fácil do que com o uso do **while**, também possível.
- Essas construções aumentam a facilidade de escrita de uma linguagem.

Critérios de Avaliação: **Confiabilidade**

	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			•

Observação: o quarto critério principal é o custo, que não foi incluído na tabela porque é apenas superficialmente relacionado aos outros três critérios e às características que os influenciam.

Critérios de Avaliação: **Confiabilidade**

- Um programa é dito confiável quando está de acordo com suas especificações em todas as condições.

Critérios de Avaliação: Confiabilidade
Verificação de tipos

- É a execução de testes para detectar erros de tipos em um programa, tanto na compilação quanto durante a execução de um programa.
- Como a verificação de tipos em tempo de execução é cara, a verificação em tempo de compilação é mais desejável.
- Além disso, quanto mais cedo os erros nos programas forem detectados, menos caro é fazer todos os reparos necessários.

Critérios de Avaliação: Confiabilidade
Verificação de tipos

- O projeto de Java requer verificações dos tipos de praticamente todas as variáveis e expressões em tempo de compilação.
  - Isso quase elimina erros de tipos em tempo de execução em programas Java.
- A falha ao verificar tipos tem levado a muitos erros de programa, como no uso de parâmetros de subprogramas na linguagem C original.
- As versões mais atuais de C eliminaram esse problema ao requererem que todos os parâmetros sejam verificados em relação aos seus tipos.

Critérios de Avaliação: Confiabilidade
Tratamento de exceções

- A habilidade de um programa de interceptar erros em tempo de execução, tomar medidas corretivas e então continuar é uma ajuda óbvia para a confiabilidade.
- Tal facilidade é chamada de tratamento de exceções.
- Ada, C++ e Java incluem diversas capacidades para tratamento de exceções, mas tais facilidades são praticamente inexistentes em muitas linguagens amplamente usadas, como C e Fortran.

Critérios de Avaliação: Confiabilidade
Utilização de apelidos

- Apelidos são permitidos quando é possível ter um ou mais nomes para acessar a mesma célula de memória.
- Atualmente, é amplamente aceito que o uso de apelidos é um recurso perigoso em uma linguagem de programação.
- A maioria das linguagens permite algum tipo de apelido – por exemplo, dois ponteiros configurados para apontarem para a mesma variável, o que é possível na maioria das linguagens.

Critérios de Avaliação: Confiabilidade
Utilização de apelidos

- O programador deve sempre lembrar que trocar o valor apontado por um dos dois ponteiros modifica o valor referenciado pelo outro.
- Em algumas linguagens, apelidos são usados para resolver deficiências nos recursos de abstração de dados.
- Outras restringem o uso de apelidos para aumentar sua confiabilidade.

Critérios de Avaliação: Confiabilidade
Legibilidade e facilidade de escrita

- Tanto a legibilidade quanto a facilidade de escrita influenciam a confiabilidade.
  - Um programa escrito em uma linguagem que não oferece maneiras naturais para expressar os algoritmos requeridos irá necessariamente usar abordagens menos prováveis de serem corretas em todas as situações possíveis.
  - Quanto mais fácil for escrevê-lo, mais provavelmente ele estará correto.
- A legibilidade afeta a confiabilidade tanto nas fases de escrita quanto nas de manutenção do ciclo de vida.
- Programas difíceis de ler são também difíceis de escrever e modificar.

Critérios de Avaliação: **Custo**

	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			•

Observação: o quarto critério principal é o custo, que não foi incluído na tabela porque é apenas superficialmente relacionado aos outros três critérios e às características que os influenciam.

Critérios de Avaliação: **Custo**

- O custo total definitivo de uma linguagem de programação é uma função de muitas de suas características, tais como:
  - custo de treinar programadores;
  - custo de escrever programas na linguagem;
  - custo de compilar programas na linguagem;
  - custo de executar programas;
  - custo do sistema de implementação da linguagem;
  - custo de uma confiabilidade baixa;
  - custo de manter programas.

Conclusão

- Os critérios de projeto de linguagem têm diferentes pesos quando vistos de diferentes perspectivas.
  - Implementadores de linguagens estão preocupados principalmente com a dificuldade de implementar as construções e recursos da linguagem.
  - Usuários estão preocupados primeiramente com a facilidade de escrita e depois com a legibilidade.
  - Projetistas são propensos a enfatizar a elegância e a habilidade de atrair um grande número de usuários.
- Essas características geralmente entram em conflito.

RESUMO

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 e torna mais fácil o aprendizado de novas linguagens.

Os computadores são usados em uma variedade de domínios de solução de problemas. O projeto e a avaliação de uma linguagem de programação em particular são altamente dependentes do domínio para o qual ela será usada.

Dentre os critérios mais importantes para a avaliação de linguagens, estão a legibilidade, a facilidade de escrita, a confiabilidade e o custo geral.

Referência Bibliográfica

SEBESTA, Robert W.; SANTOS, José Carlos Barbosa dos; TORTELLO, João Eduardo Nóbrega, Conceitos de linguagens de programação. 11 ed. Porto Alegre, RS: Editora Bookman, 2018, 758 p. ISBN 978-85-8260-468-7.