Paradigmas e linguagem Python

Prof. Marcelo Vasques de Oliveira

Sidney Nicolau Venturi Filho

Apresentação

As linguagens de programação são classificadas por nível, geração e paradigmas, com destaque para a versatilidade da Python. Critérios de avaliação incluem domínios de aplicação e implementação. Entender essas características e identificar os paradigmas de cada linguagem é essencial para escolher a mais adequada conforme o tipo de problema e a solução necessária no mercado de trabalho.

Propósito

Compreender as características e classificações das linguagens de programação, bem como identificar os paradigmas de aplicação de cada linguagem para escolher adequadamente a linguagem de programação conforme o tipo de problema e solução demandada.

Objetivos

Módulo 1

Classificando as linguagens de programação

Classificar as linguagens de programação.

Módulo 2

Critérios de avaliação de linguagens de programação

Descrever critérios de avaliação de linguagens de programação.

Módulo 3

Paradigmas de linguagens de programação

Distinguir os paradigmas e suas características.

Módulo 4

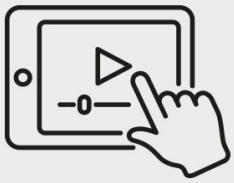
Métodos de implementação das linguagens

Identificar métodos de implementação das linguagens.

Introdução

As linguagens de programação são classificadas por nível, geração e paradigmas, destacando Python pela sua versatilidade. Critérios de avaliação incluem domínios de aplicação e implementação. Entender essas características e identificar os paradigmas de cada linguagem é essencial para escolher a mais adequada conforme o tipo de problema e a solução necessária no mercado de trabalho. Neste vídeo, apresentaremos o conteúdo, incluindo a classificação, os critérios de avaliação, os paradigmas e os métodos de implementação de linguagens de programação. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Material para download

Clique no botão abaixo para fazer o download do conteúdo completo em formato PDF.

Download material

1 - Classificando as linguagens de programação

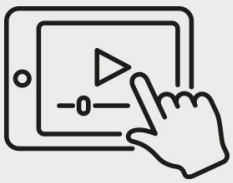
Ao final deste módulo, você será capaz de classificar as linguagens de programação.

Razões para estudarmos linguagens de programação

Estudar linguagens de programação é fundamental para entender a base do desenvolvimento de software, que é imprescindível em diversas áreas da ciência e tecnologia. Conhecer diferentes linguagens e paradigmas permite selecionar a ferramenta adequada para resolver problemas específicos, aumentando a eficiência e eficácia do trabalho.

Neste vídeo, falaremos sobre linguagem de programação e produtividade do programador. Abordaremos também o papel da abstração nas linguagens de programação. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Linguagem de programação e produtividade do programador

Um programa de computador, ou software, é um conjunto de instruções que orientam o hardware do computador para o que deve ser feito. O **software** pode ser classificado em aplicativo ou básico. Veja a diferença entre eles.

Aplicativo

Visa oferecer ao usuário facilidades para realizar uma tarefa laboral ou de lazer.

×

Básico

Compreende programas essenciais ao funcionamento do computador. O sistema operacional é o principal exemplo.

Uma linguagem de programação é um software básico, que permite ao programador escrever outros programas de computador, seja ela um software aplicativo ou básico.

A codificação de um programa em uma linguagem de programação, chama-se programa-fonte, que ainda não pode ser entendido e executado pelo hardware do computador, pois este apenas entende a linguagem de máquina ou linguagem binária, na qual cada instrução é uma sequência de bits (0 ou 1).

> **Uma linguagem de programação é um formalismo com um conjunto de símbolos, palavras reservadas, comandos, regras sintáticas e semânticas e outros recursos, que permitem especificar instruções de um programa.**

As linguagens de programação surgiram da necessidade de liberar o programador dos detalhes mais íntimos das máquinas em que a programação é feita, permitindo a programação em termos mais próximos ao problema, ou em nível mais alto.

A produtividade de um programador ao escrever o código em uma linguagem de programação está intimamente relacionada à facilidade de aprendizado, à leitura e à escrita de programas naquela linguagem,

somada à expertise do programador no contato com ela. Isto é, quanto mais o programador conhecer as propriedades superlativas daquela linguagem, melhores e mais eficientes serão os códigos escritos.

Pessoa escrevendo código em computador.

Segundo o cientista da computação holandês, Edsger Dijkstra:

> O aspecto mais importante, mas também o mais elusivo de qualquer ferramenta, é sua influência nos hábitos daqueles que se treinam no seu uso. Se a ferramenta é uma linguagem de programação, essa influência é, gostemos ou não, uma influência em nosso hábito de pensar.
>
> (Edsger Dijkstra)

O papel da abstração nas linguagens de programação

Abstração é o processo de identificação das qualidades e/ou propriedades relevantes para o contexto que está sendo analisado e desprezando o que seja irrelevante.

> ### Um modelo é uma abstração da realidade.

Um programa de computador é um modelo, pois representa a solução de um problema em termos algorítmicos. Assim sendo, a abstração permeia toda a atividade de programação de computadores.

A linguagem de máquina foi a primeira a ser criada para a prática de programação. Trata-se da linguagem nativa do computador, escrita no sistema binário, a única que ele, de fato, compreende. Essa linguagem, muito complicada para ser entendida pelas pessoas - em que um comando que **soma** 2 números é formado por uma sequência de 1 e 0 -,

é muito difícil de ser memorizada, usada e, mais ainda, de ser entendida por terceiros.

Sistema binário.

As primeiras linguagens de programação, porém, não reconheciam o papel crucial que a abstração desempenha na programação.

Comentário

No início da década de 1950, o único mecanismo de abstração fornecido pela linguagem de montagem, ou Assembly, em relação às linguagens de máquina eram os nomes simbólicos. O programador podia empregar termos relativamente autoexplicativos (nomes simbólicos) para nomear códigos de operação (ADD = soma, SUB = subtração, M = multiplicação e DIV = divisão) e posições de memória. A linguagem de montagem (Assembly) melhorou a vida do programador, porém obrigava-o a escrever 1 linha de código para cada instrução que a máquina deve executar, forçando-o a pensar como se fosse uma máquina.

Um pouco mais adiante, visando a aumentar o poder de abstração das linguagens de forma a permitir uma melhor performance dos programadores, surgem as linguagens de alto nível, próximas à linguagem humana e mais distantes das linguagens Assembly e de máquina.

A tabela a seguir exibe, na primeira coluna, um programa-fonte, escrito numa linguagem de alto nível, a linguagem Python. Na segunda coluna, temos o código equivalente na linguagem Assembly para o sistema operacional Linux e, na terceira coluna, o respectivo código na linguagem de máquina, de um determinado processador. Observe!

Linguagem Python	Assembly	Linguagem de máquina

`def swap (self, v, k):`
`  temp = self.v[k]`
`  self.v[k] = self.v[k+1]`
`  self.v[k+1] = temp`
	`swap:`
	`Muli $2,$5,4`
	`Add $2,$4,$2`
	`Lw $15,0($2)`
	`Lw $16,4($2)`
	`Sw`
		`00000000001111111111100000000001`
		`00011111111000000111000011111101`
		`11111000001100000111111110000000`
		`10000000100000001000000010000000`
		`00000000010000000001000000000010`
		`00000000000000000111100001001010`
		`00000000111000111111001111111111`

	`$16,0($2)`
	`Sw`
	`$15,4($2)`
	`Jr $31`	

Tabela: Comparativo entre linguagem Python, assembly e linguagem de máquina.
Marcelo Vasques de Oliveira.

Agora, observe alguns detalhes sobre a tabela apresentada.

- O programa na linguagem Python tem, na verdade, 3 comandos, que estão nas linhas 2, 3 e 4. → Linha 1 é a declaração da função Swap.
- O programa em linguagem Assembly tem 7 comandos (na linha 1, SWAP não é um comando, mas um rótulo, nome dado a um trecho de código).
- O programa em linguagem de máquina tem 7 comandos, a mesma quantidade de comandos do mesmo programa em Assembly → paridade de 1 para 1 → comandos em Assembly → comandos em linguagem de máquina.

A imagem a seguir ilustra o conceito de abstração, em que a partir da linguagem de máquina, cria-se camadas (de abstração) para facilitar a vida do programador. Confira!

Crescimento do nível de abstração.

A seguir, detalharemos cada nível. Acompanhe!

① **Nível 1**

É representado pelo hardware, conjunto de circuitos eletrônicos.

② **Nível 2**

É representado pela linguagem de máquina (1 e 0), única que o hardware entende.

Nível 3

É representado pela linguagem Assembly (mneumônicos).

Nível 4

É representado pelas linguagens de alto nível, próximas à língua do usuário e distantes da linguagem computacional. Python e Java são linguagens de programação representativas da classe LP de alto nível (LP = linguagem de programação).

Por que estudar linguagens de programação?

O estudante e/ou programador que se dispuser a gastar seu tempo aprendendo linguagens de programação terá as seguintes vantagens:

- Maior capacidade de desenvolver soluções em termos de programas — compreender bem os conceitos de uma LP pode aumentar a habilidade dos programadores para pensar e estruturar a solução de um problema.
- Maior habilidade em programar em uma linguagem, conhecendo melhor suas funcionalidades e implementações, ajuda o programador a construir programas melhores e mais eficientes. Por exemplo, conhecendo como as LPs são implementadas, podemos entender melhor o contexto e decidir entre usar ou não a recursividade, que se mostra menos eficiente que soluções iterativas.
- Maiores chances de acerto na escolha da linguagem mais adequada ao tipo de problema em questão, quando se conhece os recursos e como a linguagem os implementa. Por exemplo, saber que a linguagem C não verifica, dinamicamente, os índices de acesso a posições de vetores pode ser decisivo para sua escolha em soluções que usem frequentemente acessos a vetores.
- Maior habilidade para aprender novas linguagens. Quem domina os conceitos da orientação a objeto tem mais aptidão para aprender

Python, C++, C# e Java.

- Amplo conhecimento dos recursos da LP reduz as limitações na programação.
- Maior probabilidade para projetar novas LP, aos que se interessarem por esse caminho profissional: participar de projetos de criação de linguagens de programação.
- Aumento da capacidade dos programadores em expressar ideias.

Em geral, um programador tem expertise em poucas variedades de linguagens de programação, dependendo do seu nicho de trabalho. Isso, de certa forma, limita sua capacidade de pensar, pois ele fica restrito pelas estruturas de dados e controle que a(s) linguagem(ns) de seu dia a dia permitem. Conhecer uma variedade maior de recursos das linguagens de programação pode reduzir tais limitações, levando, ainda, os programadores a aumentar a diversidade de seus processos mentais.

Resumindo

Quanto maior for o leque de linguagens que um programador dominar e praticar, maiores serão as chances de conhecer e fazer uso das propriedades superlativas da(s) linguagem(ns) em questão.

Atividade 1

1. Avalie as assertivas sobre as linguagens de programação:

I. Linguagem Assembly é a nativa dos computadores.

II. Uma linguagem deve ser compatível única e exclusivamente com o hardware a que se propôs a atender.

III. A abstração traz facilidades ao programador que cada vez menos precisa conhecer o ambiente onde a linguagem opera (composto por sistema operacional e hardware).

IV. Um comando em uma linguagem de alto nível faz mais que uma operação primária do hardware.

Com base em sua análise, marque a opção que apresenta apenas as assertivas corretas.

A

I, II, III e IV.

B III e IV apenas.

C III apenas.

D II e IV apenas.

E II, III e IV apenas.

Parabéns! A alternativa B está correta.

Analisando cada assertiva, temos:

I. Assembly, ou linguagem de montagem, é como um tradutor entre sua linguagem, o assembler, e a do computador, permitindo programar com controle preciso do hardware, mas exigindo mais conhecimento técnico. A linguagem nativa do computador é a linguagem de máquina – Falso.

II. Para funcionar em um sistema, a linguagem de programação precisa "falar a mesma língua" do sistema operacional (SO) e do hardware. O SO gerencia recursos e traduz as instruções da linguagem para o hardware, como se fosse um intérprete. Sem essa compatibilidade, o programa não seria compreendido ou executado corretamente. – Falso.

III. Na programação, a abstração esconde detalhes complexos e oferece interfaces simples para o programador se concentrar no "o quê" e não no "como". É como ter um carro: você dirige sem precisar saber como o motor funciona. A abstração, portanto, concentra-se no que importa e deixa de lado o irrelevante. – Verdadeiro.

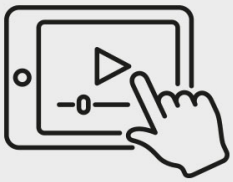
IV. Comandos em linguagens de alto nível exigem mais do hardware devido à abstração (esconde detalhes), tradução (para linguagem do hardware), otimização (melhor desempenho) e ao uso de bibliotecas (funcionalidades pré-construídas). – Verdadeiro.

Classificação das linguagens de programação por nível

Ao longo do tempo, diferentes autores propuseram várias formas de classificar as linguagens de programação, usando critérios diversos. Aqui, você vai entender as características, classificações e implementações das várias linguagens de programação, com base em seus níveis. Compreender esses aspectos permitirá criar soluções inovadoras, entender melhor os sistemas e colaborar eficazmente em equipes de desenvolvimento.

Neste vídeo, falaremos sobre a classificação das linguagens de programação, abordando as linguagens de alto e baixo nível. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Linguagem de baixo nível

São linguagens que se aproximam da linguagem de máquina, além da própria, que se comunicam diretamente com os componentes de hardware, como processador, memória e registradores.

> As linguagens de baixo nível estão relacionadas à arquitetura de um computador.

São escritas usando o conjunto de instruções do respectivo processador. Ou seja, cada processador diferente (ou família de processador, como os I3, I5 e I7 da Intel) tem um conjunto de instruções específicos (instructions set).

A imagem a seguir ilustra a representação de uma instrução em linguagem de máquina ou binária de um processador específico. A instrução tem palavras (unidade executada pelo processador) de 16 bits,

sendo 4 bits para representar a instrução (código da instrução), 6 bits para representar cada operando.

Instrução em linguagem de máquina.

Imagine, agora, uma sequência de 0 e 1 para que possamos dizer ao processador cada ação que deve ser realizada conforme apresentado a seguir.

Terminal

A programação em linguagem de máquina, a linguagem nativa dos processadores, era realmente muito complexa.

Essa complexidade motivou o desenvolvimento da linguagem Assembly, que deixava de ser a linguagem nativa dos processadores, mas usava das instruções reais dos processadores. Assim, a instrução na linguagem Assembly precisa ser convertida para o código equivalente em linguagem de máquina.

Por exemplo, as três linhas de código na linguagem Assembly, move o numeral 2 para o registrador AX (linha 1), move o numeral 1 para o registrador BX (linha 2) e soma o conteúdo dos 2 registradores (linha 3). Veja!

Assembly

A Assembly não chega a ser o ideal em termos de uma linguagem, por ser ainda próxima da máquina, mas já foi um grande avanço em relação à memorização da sequência de 0 e 1 de uma instrução de máquina.

Agora responda:

As linguagens de baixo nível estão próximas ou distantes da língua humana (escrita)?

Toque em um dos botões para responder.

Próximas

Distantes

Linguagem de alto nível

No outro extremo das linguagens de baixo nível, estão as linguagens de alto nível, na medida em que se afastam da linguagem das máquinas e se aproximam da linguagem humana (no caso, a linguagem escrita e a grande maioria em inglês).

Comentário

Quem programa em uma linguagem de alto nível não precisa conhecer características dos componentes do hardware (processador, instruções e registradores). Isso é abstraído no pensamento computacional.

As instruções das linguagens de alto nível são bastante abstratas e não estão relacionadas à arquitetura do computador diretamente.

As principais linguagens são **ASP**, **C**, **C++**, **C#**, **Pascal**, **Delphi**, **Java**, **Javascript**, **Lua**, **MATLAB**, **PHP** e **Ruby**, dentre outras.

Voltemos ao exemplo da linguagem Assembly. A seguir, veja o mesmo código usando variáveis, como abstração do armazenamento e codificado na linguagem Python.

Python

Veja, agora, o mesmo código na linguagem C.

C

Cada comando de uma linguagem de alto nível precisa ser convertido e equivalerá a mais de uma instrução primária do hardware. Isso significa que, numa linguagem de alto nível, o programador precisa escrever menos código para realizar as mesmas ações, além de outras vantagens, aumentando consideravelmente a sua eficiência ao programar.

C e C++ são classificados por alguns autores como linguagem de médio nível, pois estão próximas da linguagem humana (linguagem de alto nível), mas também estão próximas da máquina (linguagem de baixo nível), pois possuem instruções que acessam diretamente memória e registradores. Inicialmente, a linguagem C foi criada para desenvolver o sistema operacional UNIX, que até então era escrito em Assembly.

Linguagem C e C++.

Algumas pessoas consideram a existência de linguagens de altíssimo nível, como Python, Ruby e Elixir, por serem linguagens com uma enorme biblioteca de funções e que permitem a programação para iniciantes sem muito esforço de aprendizado.

Atividade 2

Para entender a complexidade e a aplicabilidade das linguagens de programação, é crucial distinguir entre linguagens de baixo nível e de alto nível. Qual das opções melhor descreve a principal diferença entre linguagens de programação de baixo nível e de alto nível?

A Linguagens de baixo nível são mais fáceis de aprender e usar do que linguagens de alto nível.

B Linguagens de alto nível são mais próximas da linguagem humana, enquanto linguagens de baixo nível são mais próximas da linguagem de máquina.

C Linguagens de baixo nível têm melhor suporte para programação orientada a objetos do que linguagens de alto nível.

D Linguagens de alto nível são menos eficientes em termos de desempenho do que linguagens de baixo nível.

E Linguagens de baixo nível são mais seguras e têm menos bugs do que linguagens de alto nível.

Parabéns! A alternativa B está correta.

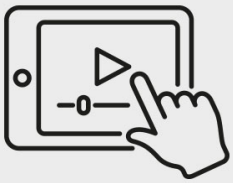
Linguagens de alto nível utilizam uma sintaxe mais próxima da linguagem humana, facilitando a leitura e a escrita do código. Em contrapartida, linguagens de baixo nível são mais próximas do código de máquina, proporcionando maior controle sobre o hardware, mas com maior complexidade e menor legibilidade.

Classificação das linguagens de programação por gerações

Vamos explorar a classificação das linguagens de programação através de suas gerações, desde as linguagens de primeira geração, próximas ao código de máquina, até as modernas, de quarta geração, focadas em produtividade e facilidade de uso, e as de quinta geração, voltadas para a inteligência artificial. Compreender essa evolução ajuda a apreciar como as linguagens se adaptaram às crescentes necessidades de desenvolvimento de software, influenciando a eficiência e a abordagem dos programadores na resolução de problemas complexos.

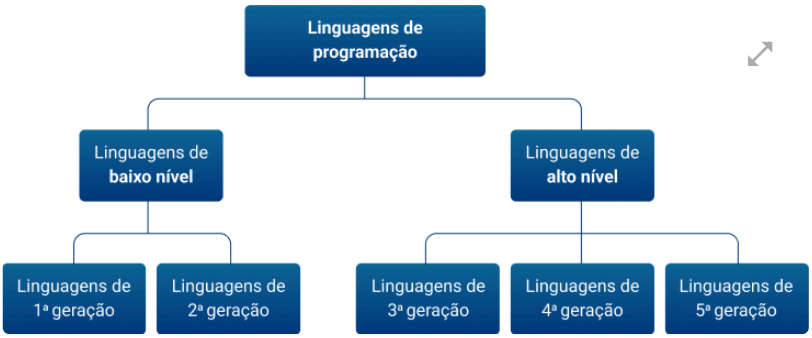
Neste vídeo, falaremos sobre a classificação das linguagens de programação por gerações, em que, a cada geração, novos recursos facilitadores são embutidos nas respectivas linguagens. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Classificação por gerações

Outra forma de classificar as linguagens, amplamente difundida, é por gerações. Não há um consenso sobre as gerações, alguns consideram 5, outros 6. A cada geração, novos recursos facilitadores são embutidos nas respectivas linguagens. Observe a imagem.



Classificação das linguagens quanto à sua geração.

A seguir, detalharemos cada geração.

Linguagens de 1ª geração (linguagem de máquina)

São os primeiros tipos de linguagens de programação, compostas por códigos binários (0s e 1s) diretamente compreendidos pelo hardware do computador. Elas são altamente específicas ao tipo de processador e operam em um nível de abstração muito baixo, exigindo um conhecimento detalhado da arquitetura do computador. Embora proporcionem programas extremamente rápidos e eficientes, são difíceis de programar e propensas a erros.

Processador.

Linguagens de 2ª geração (linguagem de montagem - Assembly)

As linguagens de segunda geração são denominadas Assembly e são traduzidas para a linguagem de máquina por um programa especial

(montador), chamado Assembler. A partir dessa geração, toda linguagem vai precisar de um processo de conversão do código nela escrito, para o código em linguagem de máquina.

Acompanhe o exemplo para uma CPU abstrata. Considere a seguinte sequência de três instruções em linguagem Assembly. Observe!

Código em Assembly	O que faz cada linha de código
Mov #8, A	Lê um valor da posição de memória 8 para o registrador A
Mov #9, B	Lê um valor da posição de memória 9 para o registrador B
ADD A,B	Soma os valores armazenados nos registradores A e B

Tabela: Exemplo de uma CPU abstrata.
Marcelo Vasques de Oliveira.

Em linguagem de máquina, depois de traduzidas pelo assembler, as instruções poderiam ser representadas pelas seguintes sequências de palavras binárias.

Código em Assembly	Código em linguagem de máquina
Mov #8, A	01000011 11001000 01100001
Mov #9, B	01000011 11001001 01100010
ADD A,B	01010100 01100001 01100010

Tabela: Sequências de palavras binárias.
Marcelo Vasques de Oliveira.

Houve um aumento significativo no nível de abstração, mas parte da dificuldade permanece, pois o programador, além de necessitar memorizar os mneumônicos, precisa conhecer a arquitetura do computador como forma de endereçamento dos registradores e memória, além de outros aspectos.

Linguagens de 3ª geração (linguagens procedurais)

São as linguagens de alto nível, de aplicação geral, em que uma única instrução em uma linguagem próxima a do homem pode corresponder a mais de uma instrução em linguagem de máquina.

Caracterizam-se pelo suporte a variáveis do tipo simples (caractere, inteiro, real e lógico) e estruturados (matrizes, vetores, registros), comandos condicionais, comando de iteração e programação modular (funções e procedimentos), estando alinhadas à programação estruturada.

Comentário

O processo de conversão para a linguagem de máquina ficou mais complexo e ficaram a cargo dos interpretadores e tradutores. As primeiras linguagens de 3ª geração que foram apresentadas ao mercado são Fortran, BASIC, COBOL, C, PASCAL, C, dentre outras.

A 3ª geração de linguagens apresenta as seguintes propriedades em comum:

- Armazenamento de tipos de dados estaticamente simples, estruturados e enumerados.
- Alocação de memória dinamicamente, através de ponteiros, que são posições de memória cujo conteúdo é outra posição de memória.
- Disponibilidade de estruturas de controle sequencial, condicional, repetição e desvio incondicional.
- Permissão à programação modular, com uso de parâmetros.
- Existência de operadores relacionais, lógicos e aritméticos.
- Ênfase em simplicidade e eficiência.

Linguagens de 4ª geração (linguagens aplicativas)

São, também, linguagens de alto nível, com aplicação e objetivos bem específicos.

Veja agora um comparativo entre as linguagens de 3ª e 4ª gerações.

3ª geração		4ª geração
São procedurais, ou seja, especifica-se	×	São não procedurais. O programador especifica o que deseja fazer e

passo a passo a solução do problema.	não como deve ser feito.

O melhor exemplo de linguagens de 4ª geração é a SQL (Structured Query Language), utilizada para consulta à manipulação de banco de dados. PostScript e MATLAB são outros dois exemplos de linguagens de 4ª geração.

Linguagens de 5ª geração (linguagens declarativas)

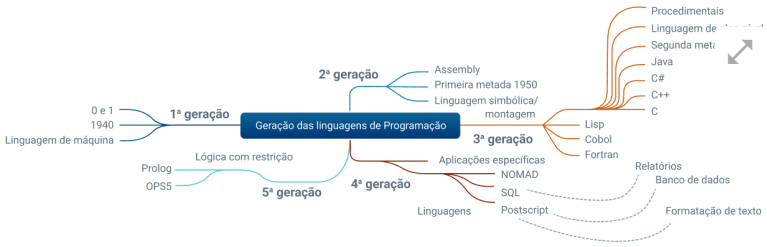
São linguagens declarativas e não algorítmicas. Por exemplo, Lisp e Prolog.

As linguagens de 5ª geração são usadas para desenvolvimento de sistemas especialistas (área da IA), de sistemas de reconhecimento de voz e machine learning.

Reconhecimento de voz.

A imagem a seguir ilustra as características de cada geração.

Alguns autores classificam a 6ª geração, como uma evolução da 5ª, em que prevalecem as aplicações de redes neurais, uma outra vertente da inteligência artificial. Observe a imagem.



Características das gerações de linguagens.

Atividade 3

As linguagens de programação evoluíram significativamente ao longo de décadas, e essa evolução é frequentemente categorizada em gerações. Cada geração trouxe características distintas e avanços tecnológicos. Considerando essa evolução, qual das alternativas a seguir melhor descreve uma linguagem de programação de quarta geração (4GL)?

A Linguagens de baixo nível que requerem detalhamento do controle de hardware, como Assembly.

B Linguagens procedurais de alto nível que utilizam estruturas de controle como loops e condicionais, como C.

C Linguagens orientadas a objetos que permitem a criação de classes e objetos, como Java.

D Linguagens declarativas que se concentram na descrição do que deve ser feito, muitas vezes utilizadas em bancos de dados, como SQL.

E Linguagens funcionais que enfatizam a avaliação de funções matemáticas e evitam estados mutáveis, como Haskell.

Parabéns! A alternativa D está correta.

As linguagens de programação de quarta geração (4GL) são projetadas para serem mais próximas da linguagem humana e para facilitar o desenvolvimento rápido de aplicações. Elas são geralmente declarativas, focando o que deve ser feito em vez de como fazer. Isso contrasta com as linguagens de terceira geração (3GL), que são procedurais e requerem a especificação detalhada dos passos de execução. Exemplos comuns de 4GL incluem linguagens de consulta de banco de dados, como SQL, que permite aos usuários especificar diretamente as operações de dados sem precisar detalhar o processo completo de manipulação de dados. As outras opções, embora representem diferentes paradigmas e gerações de linguagens de programação, não se encaixam na definição de 4GL.

2 - Critérios de avaliação de linguagens de programação

Ao final deste módulo, você será capaz de descrever critérios de avaliação de linguagens de programação.

Domínios da programação

Vamos explorar agora os diversos domínios da programação, desde o desenvolvimento web e móvel até a inteligência artificial. Compreender esses domínios permite identificar as linguagens e tecnologias mais adequadas a cada área, facilitando a criação de soluções eficientes e inovadoras. Vamos conferir como diferentes linguagens de programação são aplicadas em contextos específicos, permitindo uma escolha informada da ferramenta certa para resolver problemas complexos em variados setores da indústria.

Neste vídeo, falaremos sobre os principais domínios da programação atualmente. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

O computador tem sido usado para diversos fins, na ciência, nas forças armadas, nas empresas públicas e privadas, pelos profissionais liberais, pelas pessoas em seus lazeres e onde mais possa ser aplicado. Seu uso

vai desde o controle de robôs que fazem a montagem de automóveis em suas linhas de montagem até jogos digitais. Em função desse uso adverso, surgiram linguagens de programação com diferentes objetivos.

A seguir, discutiremos as principais áreas e as respectivas linguagens de programação em destaque.

Aplicações científicas (máquinas de calcular com alta precisão)

O primeiro computador, o ENIAC, foi desenvolvido por três anos e ficou pronto em 1946. Sua principal finalidade eram cálculos balísticos. Os computadores seguintes, nas décadas de 1940 e 1950, também focaram em cálculos científicos complexos.

As linguagens de programação nessa época eram a linguagem de máquina e Assembly.

Computador ENIAC.

Na década de 1960 surgem as primeiras linguagens de programação de alto nível, com destaque para Fortran (iniciais de FORmula TRANslator) e posteriormente para ALGOL60. As principais características dessas linguagens eram:

- Estruturas de dados simples.
- Alto volume de cálculos com aritmética de ponto flutuante (precisão).
- Preocupação com a eficiência, pois sucederam a linguagem Assembly.

Aplicações comerciais

A segunda onda de aplicativos foi para suprir as demandas das empresas a partir de meados da década de 1950.

> **Em 1960, surge a linguagem que seria o ícone das aplicações comerciais de computadores de grande porte, naquele momento, o COBOL.**

As linguagens de programação que apoiaram o crescimento das aplicações comerciais têm como características:

- Facilidade para produzir relatórios, fundamentais nos controles das operações contábeis, bancárias, estoque e financeiras (primeiros

focos da época).

- Precisão com números decimais e ponto flutuante, para representar as altas cifras das grandes empresas, as primeiras a investirem nessas aplicações.
- Capacidade de especificar operações aritméticas comerciais.

Cabe destacar que as linguagens destinadas a aplicações comerciais ganham força com a microcomputação a partir dos anos 1980, levando as aplicações comerciais aos médios e pequenos empresários.

Aplicações com inteligência artificial

As linguagens que sustentam o desenvolvimento de aplicações apoiadas na inteligência artificial (IA) ganham força nos dias de hoje.

A grande ruptura no pensamento computacional acontece quando surgem as linguagens que apoiam a IA usando computação simbólica e não numérica, como a maioria das linguagens da época. Conheça dois marcos importantes do surgimento dessas linguagens.

1959

Surge a **Lisp**, primeira linguagem projetada para apoio à computação simbólica, primeira referência da computação funcional.

1977

Surge a **Prolog**, primeira linguagem de apoio da computação lógica, essência dos sistemas especialistas.

Agora responda:

O que você considera ser a essência dos sistemas especialistas?

Toque em um dos botões para responder.

Usar IA para criar códigos complexos.

Usar IA para simular o comportamento humano.

Programação de sistemas

A programação de sistemas cabe a linguagens de programação que tenham comandos e estruturas para acessar, diretamente, o hardware. Tais linguagens são usadas para desenvolver softwares básicos, como sistemas operacionais, tradutores e interpretadores de linguagens de programação.

Comentário

Antes de surgir a linguagem C, usada para desenvolver o sistema operacional Linux, a Assembly era a linguagem usada para esse fim. A linguagem C++ também é usada com essa finalidade.

Programação para Web

Com o crescimento da internet e tecnologias adjacentes, o uso dos sistemas se desloca do ambiente desktop (domínio dos anos 1980 e 1990) para o ambiente Web.

No contexto de programação para Web, temos dois diferentes ambientes de desenvolvimento: a camada de apresentação, que roda no navegador (lado cliente) e a camada de lógica do negócio, que roda nos servidores web (lado servidor), juntamente com a camada de persistência, considerando o modelo de desenvolvimento em três camadas (apresentação, lógica do negócio e persistência de dados).

Interface lado cliente.

Para a camada de apresentação, usa-se as linguagens HTML (linguagem de marcação) e CSS (usada em conjunto com HTML para definir a apresentação da página web), além de JavaScript (programação de scripts), no lado cliente (navegadores).

Para o desenvolvimento das camadas de lógica do negócio, as principais LP são: C#, PHP, ASP, .NET, Java, Ruby e Python.

Programação mobile

Considerando que, hoje em dia, grande parte da população, no Brasil e no Mundo tem acesso à internet pelo celular, cresceu vertiginosamente a quantidade de apps (aplicativos) para uso de aplicações via celular. Os apps, na verdade, são interfaces que rodam no lado cliente.

As principais (não todas) linguagens que apoiam o desenvolvimento de apps para o mundo mobile, oficialmente indicadas por seus fabricantes, são:

Android

Linguagens Java e Kotlin

O Google tem por base o Android SDK e orienta a usar as linguagens Kotlin, Java e C++. Porém, as linguagens Python, Shell script, Basic4Android, LiveCode (para iOS e Windows também), App Inventor (que não necessita conhecimento em programação), Unity (motor para games) e GO, também são usadas para desenvolver apps para Android.

iOS

Linguagens Swift (oficial da Apple) e Objective-C (código nativo para iOS)

O desenvolvimento de APP para iOS é baseado numa IDE chamada Xcode que permite o desenvolvimento de APP em várias linguagens, como C, C++, Java e Python, mas oficialmente orienta o Swift e Objective-C.

Windows

Linguagens C#, Visual Basic (VB), C++, HTML, CSS, JavaScript e Java

No contexto de desenvolvimento de APP para Windows, foi lançado no Windows 8.1 e atualizado para atender também ao Windows 10, o App Studio, que permite a qualquer pessoa criar em poucos passos um app Windows e publicá-lo na loja.

É importante destacar que hoje existem plataformas de desenvolvimento mobile conectadas a nuvem que fomentam o desenvolvimento de apps nativos para iOS, Android e Windows.

Atividade 1

Para entender melhor os domínios das linguagens de programação, podemos identificar onde cada linguagem é mais utilizada. Nesse contexto, qual domínio a linguagem de programação JavaScript é amplamente utilizada?

A Desenvolvimento de jogos.

B Desenvolvimento de aplicativos móveis.

C Análise de dados e estatística.

D Desenvolvimento Web.

E Programação de sistemas operacionais.

Parabéns! A alternativa D está correta.

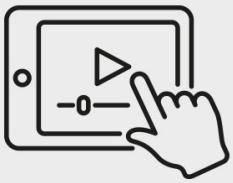
A linguagem JavaScript é frequentemente utilizada no desenvolvimento Web para criar interações dinâmicas e funcionais em páginas da web, desde a validação de formulários até a criação de interfaces de usuário responsivas.

Critérios para avaliação de linguagens de programação

Vamos abordar a avaliação de linguagens de programação, destacando critérios como legibilidade, facilidade de escrita, confiabilidade e custo. Compreender esses aspectos permite selecionar a linguagem mais adequada para cada projeto, garantindo eficiência e eficácia no desenvolvimento de software. Exploraremos exemplos práticos e comparações entre linguagens populares para proporcionar uma visão ampla e fundamentada, facilitando decisões informadas na escolha de ferramentas de programação.

Neste vídeo, abordaremos os quatro grandes critérios para avaliação das linguagens de programação, dentro de um mesmo domínio de programação, sendo cada critério influenciado por algumas características da linguagem. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Avaliação de linguagens de programação

Segundo Sebesta (2018) são quatro grandes critérios para avaliação das linguagens de programação, dentro de um mesmo domínio de programação. Cada critério é influenciado por algumas características da linguagem. Observe!

Legibilidade

Um dos critérios mais relevantes é a “facilidade com que os programas podem ser lidos e entendidos” pelas pessoas que não necessariamente participaram do desenvolvimento.

Facilidade de escrita

O quão facilmente uma linguagem pode ser usada para desenvolver programas para o domínio do problema escolhido.

Confiabilidade

Um programa é dito confiável se ele se comporta conforme a sua especificação, repetidas vezes.

Custo

O custo final de uma linguagem de programação é em função de muitas de suas propriedades e características.

A tabela a seguir exibe as características da linguagem que influenciam cada um dos três principais fatores de avaliação de linguagens.

Critérios			
Características	Legibilidade	Facilidade escrita	Confiabilid
Simplicidade			
Ortogonalidade			
Estruturas de controle			
Tipos de dados			
Projeto de sintaxe			

Critérios			
Suporte para abstração			
Expressividade			
Verificação de tipos			
Tratamento de exceções			
Aliasing			

Tabela: Características x Critérios de Avaliação de LPs.
Marcelo Vasques de Oliveira.

Conheça agora cada um dos critérios.

Legibilidade

Um dos critérios mais relevantes para avaliar uma linguagem de programação diz respeito à capacidade com que os programas podem ser lidos e entendidos pela sintaxe e construção da linguagem, sem considerar as possíveis influências da má programação.

A seguir, detalharemos as características que influenciam a legibilidade de uma linguagem de programação.

Simplicidade

Quanto mais simples for uma linguagem, melhor será a legibilidade do código por ela produzido. Entenda agora o que pode afetar negativamente a legibilidade de um código.

> **Uma linguagem com número elevado de construções básicas é mais difícil de ser aprendida e tende a ser subutilizada.**

Uma segunda característica que afeta negativamente a legibilidade é a **multiplicidade de recursos**.

Exemplo

Em Python, o programador pode incrementar uma variável, de duas formas distintas: cont= cont + 1 e cont +=1. Nas linguagens C e Java, ainda podemos usar para incrementar variáveis as seguintes estruturas: ++cont e cont++.

Muita simplicidade pode tornar menos legíveis os códigos escritos. Na linguagem Assembly, a maioria das sentenças são simples, porém não são altamente legíveis devido à ausência de estruturas de controle.

Uma terceira característica que afeta negativamente a legibilidade é a **sobrecarga de operadores**, como por exemplo o "+", usado para somar inteiros, reais, concatenar cadeias de caracteres (strings), somar vetores, dentre outras construções permitidas pela linguagem.

Ortogonalidade

A ortogonalidade de uma linguagem refere-se a um conjunto relativamente pequeno de construções primitivas que pode ser combinado em um número, também, pequeno de maneiras para construir as estruturas de controle e de dados de uma linguagem de programação. Em outras palavras, é a possibilidade de combinar, entre si, sem restrições, as construções básicas da linguagem para construir estruturas de dados e de controle.

Veja agora a diferença entre a boa e má ortogonalidade de uma linguagem.

Boa ortogonalidade

Permitir, por exemplo, que haja um vetor, cujos elementos sejam do tipo registro (estrutura heterogênea).

×

Má ortogonalidade

Não permitir que um vetor seja passado como argumento para uma rotina (procedimento ou função). Ou que uma função não possa retornar um vetor.

Uma linguagem ortogonal tende a ser mais fácil de aprender e tem menos exceções.

A falta de ortogonalidade leva a muitas exceções às regras da linguagem e ao excesso, o contrário (menos exceções às regras). Menos exceções implicam um maior grau de regularidade no projeto da linguagem, tornando-a mais fácil de ler, entender e aprender.

Instruções de controle

Instruções como Goto (desvio incondicional) limitam a legibilidade dos programas, pois essa instrução pode levar o controle do código a qualquer ponto do programa, limitando o entendimento e, consequentemente, a legibilidade do código escrito na linguagem.

Comentário

As linguagens modernas não implementam desvio incondicional, assim sendo, o projeto de estruturas de controle é menos relevante na legibilidade do que anos atrás, quando surgiram as primeiras linguagens de alto nível.

Tipos e estruturas de dados

A facilidade oferecida pela linguagem para definir tipos e estruturas de dados é outra propriedade que aumenta a legibilidade do código escrito. Por exemplo, uma linguagem que permita definir registros e vetores, mas não permite que um vetor tenha registros como seus elementos, terá a legibilidade afetada.

A linguagem C não possui o tipo de dado lógico ou booleano. Muitas vezes, usa-se variáveis inteiras, permitindo apenas que receba os valores 0 e 1 para conteúdo, simulando o tipo booleano.

Exemplo

Para localizar um elemento em uma das posições de um vetor, usa-se uma variável lógica se a linguagem permitir e, assim, teríamos a instrução “achou=false” em determinado trecho de código. Em outra linguagem que não permita o tipo de dado lógico, a instrução poderia ser “achou=0”, em que achou seria uma variável inteira. Qual das duas sentenças é mais clara a quem lê o código? A primeira, não é? “achou=false”.

Sintaxe

A sintaxe tem efeito sobre a legibilidade. Um exemplo é a restrição do tamanho (quantidade de caracteres) para um identificador (tipo, variável, constante, rotina – procedimento e função), impedindo que recebam

nomes significativos sobre sua utilidade. Na linguagem Fortran, o nome do identificador pode ser até seis caracteres.

Outra propriedade de sintaxe que afeta a legibilidade é o uso de palavras reservadas da linguagem. Conheça agora alguns casos.

- Na linguagem Pascal, os blocos de instrução são iniciados e encerrados com BEGIN-END, respectivamente.
- Na linguagem C, usamos chaves para iniciar e encerrar blocos de instruções.
- Na linguagem Python, usamos a endentação obrigatória para marcar blocos de comandos, aumentando a legibilidade, naturalmente.

Facilidade de escrita (redigibilidade)

A facilidade de escrita é a medida do quão fácil a linguagem permite criar programas para um domínio da aplicação.

> **A maioria das características que afeta a legibilidade também afeta a facilidade de escrita, pois haverá dificuldade para quem for ler o código se sua escrita não fluir.**

Conheça agora as características que influenciam na facilidade de escrita.

Simplicidade e ortogonalidade

Quanto mais simples e ortogonal for a linguagem, mais fácil será para escrever programas. O ideal são linguagens com poucas construções primitivas.

Imagine que uma linguagem de programação possui grande número de construções. Alguns programadores podem não usar todas, deixando de lado, eventualmente, as mais eficientes e elegantes.

Expressividade

Uma linguagem de programação com boa expressividade contribui para o aumento da facilidade de escrita dos códigos.

Por exemplo, a linguagem Assembly apresenta baixa expressividade. Já Pascal e C apresentam boa expressividade, com ricas estruturas de

controle.

Exemplo

O comando FOR é mais adequado que WHILE e REPEAT para representar lações com número fixo de vezes. Da mesma forma que o C, em que o FOR é mais indicado que o WHILE e DO-WHILE. Na linguagem Python, ocorre o mesmo entre os comandos FOR e WHILE. Na linguagem C, temos construções diversas para incremento de variável: i++ é mais simples e conveniente de usar do que i=i+1, sendo i, uma variável inteira.

Uma linguagem expressiva possibilita escrever linhas de código de uma forma mais conveniente ao invés de deselegante.

Suporte para a abstração

O grau de abstração em uma linguagem é uma propriedade fundamental para aumentar a facilidade de escrita. A abstração pode ser de dois tipos. Veja!

Processos

Como o conceito de subprograma.

Dados

Como uma árvore ou lista simplesmente encadeada.

Confiabilidade

Dizemos que um programa é confiável se ele se comportar conforme sua especificação, sob todas as condições, todas as vezes em que for executado.

A seguir, conheça alguns recursos das linguagens que exercem efeito sobre a confiabilidade de programas.

Verificação de tipos

Significa verificar, em tempo de compilação ou execução, se existem erros de tipo. Por exemplo, atribuir um valor booleano a uma variável do tipo inteira, vai resultar em erro. As linguagens fortemente tipadas, em tempo de compilação, como Python e Java, tendem a ser mais confiáveis, pois apenas valores restritos aos tipos de dados declarados poderão ser atribuídos e diminuem os erros em tempo de execução. Linguagens, como C, em que não é verificado se o tipo de dado do argumento é compatível com o parâmetro, em tempo de compilação, podem gerar erros durante a execução, afetando a confiabilidade. A verificação de tipos em tempo de compilação é desejável, já em tempo de execução é dispendiosa (mais lenta e requer mais memória), e mais flexível (menos tipada).

Tratamento de exceção

Garante a correta execução, aumentando a confiabilidade. As linguagens Python, C++ e Java possuem boa capacidade de tratar exceções, ao contrário da linguagem C. A linguagem deve permitir a identificação de eventos indesejáveis (estouro de memória, busca de elemento inexistente, overflow etc.) e especificar respostas adequadas a cada evento. O comportamento do programa torna-se previsível com a possibilidade de tratamento das exceções, o que tende a aumentar a confiabilidade do código escrito na linguagem de programação.

Aliasing (apelidos)

É o fato de ter dois ou mais nomes, referenciando a mesma célula de memória, o que é um recurso perigoso e afeta a confiabilidade. Restringir aliasing é prover confiabilidade aos programas.

Legibilidade e facilidade de escrita

Ambos influenciam a confiabilidade. A legibilidade afeta tanto na fase de codificação como na fase de manutenção. Programas de

difícil leitura são difíceis de serem escritos também.

Uma linguagem com boa legibilidade e facilidade de escrita gera códigos claros, que tendem a aumentar a confiabilidade.

Custo

O custo de uma linguagem de programação varia em função das seguintes despesas: de treinamento, de escrita do programa, do compilador, de execução do programa, de implementação da linguagem e o de manutenção do código. Observe a seguir as características de cada **custo**.

Treinamento

- Varia em função da expertise do programador, simplicidade e ortogonalidade da linguagem;
- F (simplicidade de escrita, ortogonalidade, experiência do programador).

Escrita do programa

- Varia em função da facilidade de escrita;
- F (Facilidade de escrita).

Compilação do programa

- Varia em função do custo de aquisição do compilador, hoje minimizado, em linguagens open source, como é o caso do Python;
- F (custo de aquisição do compilador).

Execução do programa

- Varia em função do projeto da linguagem;

- F (Projeto da linguagem).

Implementação da linguagem

- A popularidade da LP depende de um econômico sistema de implementação;
- Por exemplo, Python e Java possuem compiladores e interpretadores gratuitos.

Confiabilidade

- O custo da má confiabilidade será elevado. Se um sistema crítico falhar, o custo aumenta;
- Exemplos: sistema de controle de consumo de água e sistemas de usina nuclear.

Manutenção

- Depende de vários fatores, mas principalmente da legibilidade, já que a tendência é que a manutenção seja dada por pessoas que não participaram do desenvolvimento do software.

Os custos em treinamento e de escrever o programa podem ser minimizados se a linguagem oferecer bom ambiente de programação.

> **Python é uma linguagem com alta legibilidade, facilidade de escrita, além de ser confiável. Seu custo não é elevado, pois, além de ser open source, é fácil de aprender.**

Existem outros critérios que influenciam os custos de uma linguagem de programação, como, por exemplo, a portabilidade ou a capacidade que os programas têm de rodarem em ambientes diferentes (sistema operacional e hardware), o que é altamente desejável.

Comentário

A reusabilidade, ou seja, o quanto um código pode ser reutilizado em outros programas ou sistemas aumenta o nível de produtividade da linguagem, além da facilidade de aprendizado, que é fortemente afetada pela legibilidade e facilidade de escrita.

Atividade 2

Um linguagem de programação pode ser avaliada a partir de vários critérios. Qual critério refere-se à capacidade de um código ser facilmente compreendido e interpretado pelos programadores?

A Eficiência de desempenho

B Popularidade da linguagem

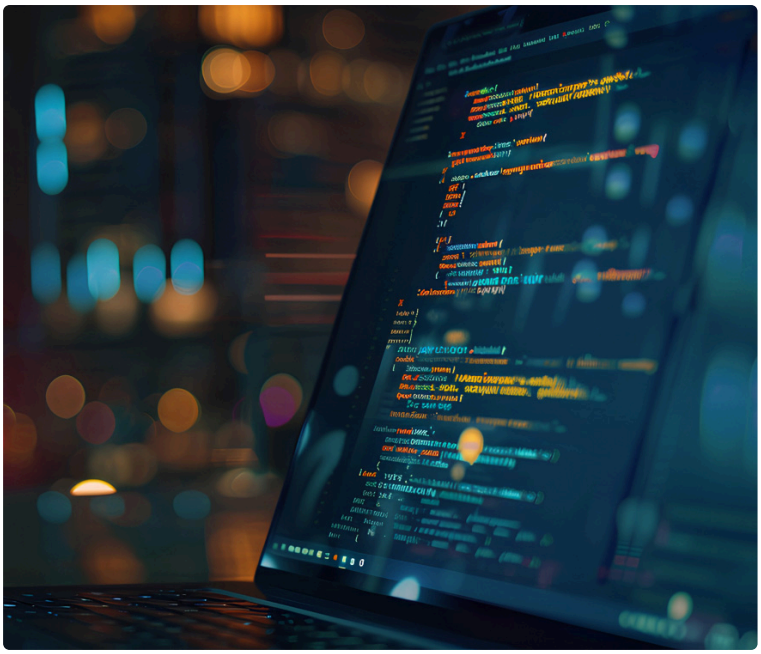
C Legibilidade do código

D Suporte a múltiplos paradigmas

E Tamanho do código-fonte

Parabéns! A alternativa C está correta.

A legibilidade do código é um critério crucial na avaliação de linguagens de programação, pois um código bem escrito e legível facilita a manutenção, colaboração e o entendimento do software ao longo do tempo.



3 - Paradigmas de linguagens de programação

Ao final deste módulo, você será capaz de distinguir os paradigmas e suas características.

Paradigma imperativo

O agrupamento por paradigmas é outra forma de classificar as linguagens de programação. Um paradigma agrupa linguagens com características semelhantes que surgiram em uma mesma época. A imagem a seguir ilustra os cinco paradigmas nos quais as linguagens de programação são classificadas. Esses paradigmas são agrupados em imperativos e declarativos, de acordo com a forma com que os programas são estruturados e descritos.

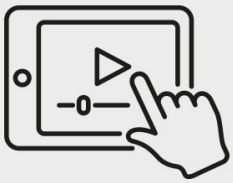
Classificação dos paradigmas das linguagens de programação.

O paradigma imperativo agrega três paradigmas: estruturado, orientado a objeto e concorrente, os quais possuem em comum o fato de especificarem passo a passo o que deve ser feito para a solução do problema. As linguagens do paradigma imperativo são dependentes da arquitetura do computador, pois especificam em seus programas como

a computação é realizada. Vamos explicar as características de cada um dos paradigmas do **subgrupo imperativo**.

Neste vídeo, falaremos sobre o paradigma imperativo para as linguagem de programação, abordando os paradigmas estruturado, orientado a objetos e concorrente. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Paradigma estruturado

Caracteriza as principais linguagens de programação que seguiram os princípios da programação estruturada nas décadas de 1970 e 1980. Conheça!

①

Princípio 1

Não usar desvios incondicionais (Goto, característico de linguagens como BASIC e versões iniciais do COBOL).

②

Princípio 2

Desenvolver programas por refinamentos sucessivos (metodologia top down), motivando o desenvolvimento de rotinas (procedimentos e funções) e a visão do programa partindo do geral para o particular, ou seja, o programa vai sendo refinado à medida que se conhece melhor o problema e seus detalhes.

③

Princípio 3

Desenvolver programas usando três tipos de estruturas (sequenciais, condicionais e repetição).

Princípio 4

Utilizar, para ser eficiente, o paradigma estruturado baseado nos princípios da arquitetura de Von Neumann, onde programas e dados residem na memória (durante a execução), instruções e dados trafegam da memória para CPU e vice-versa e resultados das operações trafegam da CPU para a memória.

As linguagens Pascal e C caracterizam bem o paradigma estruturado. A linguagem Python, multiparadigma, tem o estilo básico do paradigma estruturado.

Paradigma orientado a objetos

Com o crescimento do tamanho do código e complexidade dos programas, o paradigma estruturado começou a apresentar limitações nos sistemas que passaram a ter dificuldade de manutenção e reuso de programas e rotinas padronizadas.

A orientação a objetos surge como solução a esses problemas, permitindo, através de propriedades como abstração, encapsulamento, herança e polimorfismo, maior organização, reaproveitamento e extensibilidade de código e, consequentemente, programas mais fáceis de serem escritos e mantidos.

> **O principal foco do paradigma orientado a objetos foi possibilitar o desenvolvimento mais rápido e confiável.**

As classes são abstrações que definem uma estrutura que encapsula dados (chamados de atributos) e um conjunto de operações possíveis de serem usados, chamados métodos. Os objetos são instâncias das classes.

Exemplo

Uma classe ALUNO pode encapsular um conjunto de dados que os identifiquem: matrícula, nome, endereço (rua, número, complemento,

bairro, estado e CEP) e um conjunto de métodos: Incluir Aluno, Matricular Aluno, Cancelar Matrícula, dentre outros.

O paradigma orientado a objetos, por sua vez, usa os conceitos do paradigma estruturado na especificação dos comandos de métodos. Por isso, é considerado uma evolução do paradigma estruturado.

Python, Smalltalk, C++, Java e Delphi (oriundo do Object Pascal) são linguagens que caracterizam o paradigma orientado a objetos.

Python é orientado a objeto, pois tudo na linguagem Python é objeto, permitindo a declaração de classes encapsuladas, além de possibilitar herança e polimorfismo.

Logo Python em tela de celular.

Paradigma concorrente

Caracterizado quando processos executam simultaneamente e concorrem aos recursos de hardware (processadores, discos e outros periféricos), características cada vez mais usuais em sistemas de informação.

O paradigma concorrente pode valer-se de apenas um processador ou vários. Entenda!

Um processador

Os processos competem pelo uso do processador e outros recursos.

Vários processadores

Tarefas são executadas paralelamente em diferentes processadores simultaneamente. Esses processadores podem estar na mesma máquina ou distribuídos em múltiplos computadores.

Ada e Java são as linguagens que melhor caracterizam o paradigma concorrente, possibilitando suporte à concorrência.

Comentário

Ao contrário de Go, Python não foi originalmente projetada com foco em programação concorrente, muito menos paralela. O modo tradicional de programar concorrência em Python -- threads -- é limitado no interpretador padrão (CPython) por uma trava global (a GIL), que impede a execução paralela de threads escritas em Python. Isso significa que threads em Python são úteis apenas em aplicações I/O bound – em que o gargalo está no I/O (entrada e saída), como é o caso de aplicações na Internet.

Atividade 3

É preciso entender os paradigmas de programação para escolher a abordagem certa ao resolver problemas. Qual das seguintes características melhor descreve o paradigma imperativo na programação?

A Foco na reutilização de código através de funções e classes.

B Ênfase na descrição dos passos necessários para atingir um resultado.

C Utilização extensiva de expressões regulares para manipulação de strings.

D Organização do código em componentes independentes, comunicando-se através de mensagens.

E Priorização da lógica matemática para resolver problemas complexos.

Parabéns! A alternativa B está correta.

O paradigma imperativo concentra-se na descrição explícita dos passos ou instruções necessários para realizar uma tarefa específica, utilizando comandos de atribuição, estruturas de controle e iteração.

Paradigma declarativo

Diferentemente do paradigma imperativo, no declarativo o programador diz o que o programa deve fazer (qual a tarefa) ao invés de descrever como o programa deve fazer. O programador declara, de forma abstrata, a solução do problema. Essas linguagens não são dependentes de determinada arquitetura de computador. As variáveis são incógnitas, tal qual na matemática, e não células de memória. O paradigma declarativo agrega os paradigmas funcional e lógico. Vamos explicar as características de cada um.

Neste vídeo, falaremos sobre o paradigma declarativo para as linguagens de programação, abordando paradigmas funcional e lógico. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Paradigma funcional

Abrange linguagens que operam tão somente funções que recebem um conjunto de valores e retornam um valor. O resultado que a função retorna é a solução do problema (foca o processo de resolução de problemas).

> **Um programa resume-se em chamadas de funções, que por sua vez podem usar outras funções. Uma função pode invocar outra, ou o resultado de uma função pode ser argumento para outra função. Usa-se também chamadas recursivas de funções.**

Naturalmente, o paradigma funcional gera programas menores (pouco código).

São linguagens típicas de paradigma funcional: **LISP**, **HASKELL** e **ML**.

LISP é a LP funcional mais usada, especialmente em programas que usem os conceitos de Inteligência Artificial (sistemas especialistas, processamento de linguagem natural e representação do conhecimento), devido à facilidade de interpretação recursiva.

Por exemplo, o código a seguir implementa em Python uma função que calcula quantos números pares existem de 0 a n. Observe!

Python

Agora, veja o mesmo código usando o conceito de função recursiva. Repare que a função de nome conta_numeros chama ela mesma em seu código (isso é a recursão).

Python

Python não é uma linguagem funcional nativa, então seria exagerado afirmar isso. No entanto, ela foi influenciada pelo paradigma funcional ao permitir a recursividade, o uso de funções anônimas com lambda, entre outros recursos. Além disso, Python possui uma vasta biblioteca de funções.

Paradigma lógico

Um programa lógico expressa a solução da maneira como o ser humano raciocina sobre o problema, baseado em fatos, derivam-se conclusões e novos fatos.

Quando um novo questionamento é feito, através de um mecanismo inteligente de inferência, deduz novos fatos a partir dos existentes.

> **A execução dos programas escritos em linguagens de programação lógica segue, portanto, um mecanismo de dedução automática (máquina de inferência), sendo Prolog a linguagem do paradigma lógico mais conhecida.**

O paradigma lógico é usado no desenvolvimento de linguagens de acesso a banco de dados, sistemas especialistas (IA), tutores inteligentes etc.

Comentário

Python não tem características para implementar programas que atendam ao paradigma lógico.

Atividade 4

Os paradigmas de linguagens de programação são modelos ou estilos que definem a forma como as soluções de problemas são estruturadas e implementadas. Considerando os diferentes paradigmas. Qual das alternativas a seguir descreve corretamente uma característica fundamental do paradigma funcional?

A Utiliza herança e polimorfismo para promover a reutilização de código e abstração.

B Foca na manipulação de estados mutáveis e no uso de estruturas de controle, como loops e condicionais.

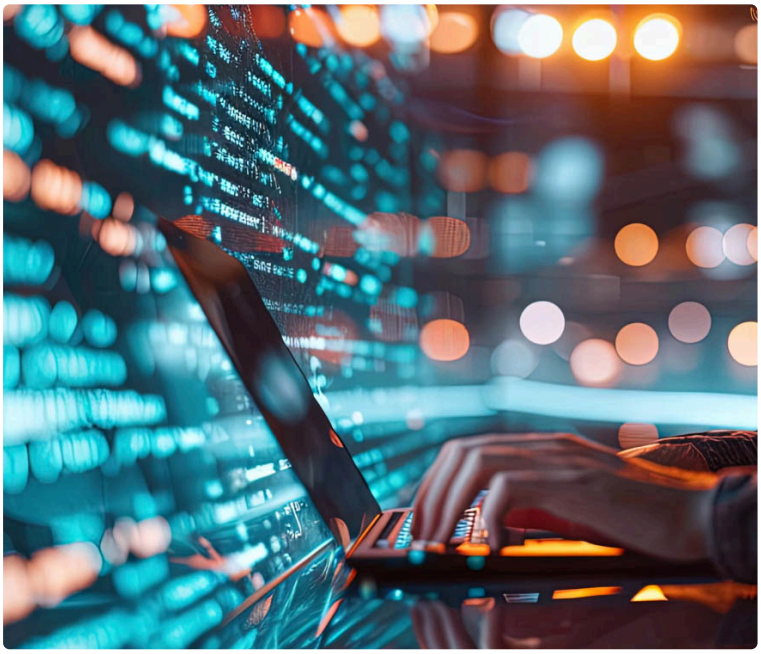
C Promove a programação com funções puras que evitam efeitos colaterais e enfatizam a imutabilidade.

D Emprega técnicas de consulta declarativas para manipulação de dados, como em SQL.

E Baseia-se em instruções detalhadas para controle de hardware, sendo próxima ao código de máquina.

Parabéns! A alternativa C está correta.

O paradigma funcional é um estilo de programação que trata a computação como a avaliação de funções matemáticas e evita mudanças de estado e dados mutáveis. As funções puras são uma característica central desse paradigma, significando que o valor de retorno de uma função depende apenas de seus parâmetros de entrada e não produz efeitos colaterais observáveis, como modificar variáveis fora de seu escopo ou interagir com o sistema de arquivos. Isso contrasta com o paradigma imperativo (B), que foca mudanças de estado e uso de estruturas de controle. O paradigma orientado a objetos (A) utiliza conceitos como herança e polimorfismo, enquanto o paradigma declarativo (D) é mais comum em linguagens de consulta como SQL. O paradigma de baixo nível (E) se refere a linguagens próximas ao código de máquina, como Assembly.



4 - Métodos de implementação das linguagens

Ao final deste módulo, você será capaz de identificar métodos de implementação das linguagens.

Tradução

Todo programa, a menos que seja escrito em linguagem de máquina, o que hoje em dia está totalmente fora dos propósitos, precisará ser

convertido para linguagem de máquina antes de ser executado. Essa conversão precisa de um programa que receba o código-fonte escrito na linguagem e gere o respectivo código em linguagem de máquina.

Esse programa, que fará a tradução do código-fonte em linguagem de máquina, é que vai determinar como os programas são implementados e como vão executar. Existem duas formas de realizar essa conversão: tradução e interpretação. É importante conhecer e compreender qual processo de conversão é utilizado pela respectiva linguagem de programação.

Veremos agora a tradução!

Neste vídeo, falaremos sobre o processo de conversão por tradução, no qual o programa escrito em uma linguagem de alto nível é traduzido para uma versão equivalente em linguagem de máquina, antes de ser executado. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

No processo de conversão por tradução, o programa escrito em uma linguagem de alto nível é traduzido para uma versão equivalente em linguagens de máquina, antes de ser executado. Esse processo pode ser executado em várias fases, que podem ser combinadas e executadas em simultaneidade.

> **O processo de tradução é erroneamente chamado de compilação, que, na verdade, é uma de suas fases.**

As fases que compõem o tradutor, ou seja, iniciando na leitura do programa-fonte (linguagem de alto nível) e terminando com a geração do código executável (entendido pela máquina), são a compilação, montagem, carga e ligação.

A imagem a seguir ilustra o processo de tradução.

Toque nos pontos em destaque.

Fases de tradução de um programa em linguagem de alto nível.

Compilador

O compilador analisa o código-fonte e estando tudo OK, o converte para um código Assembly (da máquina hospedeira).

Montador

O montador traduz o código Assembly para o código de máquina intermediário (Código-objeto), que não é executável pelo computador. O código-objeto pode ser relocável, ou seja, carregado em qualquer posição de memória ou absoluto, carregado em um endereço de memória específico. A opção relocável é mais comum e mais vantajosa.

Carregador

O carregador é que torna o código-objeto em relocável.

Ligador

O Ligador liga (ou linka) o código-objeto relocável com as rotinas bibliotecas (outros objetos, rotinas do SO, DLLs etc.), usadas nos códigos-fontes. Essa ligação gera o código executável.

Compilador

É o elemento central do processo de tradução, responsável pelo custo de compilação. Em função dessa relevância, muitas vezes, o processo como um todo é erroneamente chamado de compilação, uma vez que o ambiente integrado das linguagens atuais já integra todos os componentes (montador, compilador, carregador e ligador) quando necessário.

O projeto da linguagem tem no compilador a sua **figura central**.

A imagem a seguir ilustra os componentes envolvidos na compilação de um programa fonte. Observe!

Visão geral de um compilador.

Agora, exploraremos cada fase da compilação. Acompanhe!

Análise léxica

Observe o que acontece na fase da análise léxica da compilação.

- Identificação dos tokens (elementos da linguagem).

- Desconsideração de partes do código-fonte, como espaços em branco e comentários.
- Geração da tabela de símbolos, com todos esses tokens, que são identificadores de variáveis, de procedimentos, de funções, comandos, expressões etc.

Análise sintática

Verifica se os tokens são estruturas sintáticas válidas, aplicando as regras gramaticais definidas no projeto da linguagem.

São exemplos de estruturas sintáticas as expressões e comandos.

Análise semântica

Verifica se as estruturas sintáticas possuem sentido.

Exemplo

A análise sintática verifica se um identificador de variável ou constante é usado adequadamente, se operandos e operadores são compatíveis.

A análise monta a árvore de derivação conforme ilustrado a seguir para formação das expressões.

Árvore de derivação de uma expressão.

Gerador de código intermediário, otimizador de código e gerador de código

Em distintas fases geram o programa-alvo ou programa-objeto.

- Gerador de código intermediário, que contém toda a informação para gerar o código-objeto.

Na imagem a seguir, o código intermediário está representado no último quadro - código em Assembly:

Código intermediário.

O otimizador tem por objetivo eliminar redundâncias do código intermediário e tornar o objeto mais enxuto e eficiente.

Tratador de erros

Em todas as fases da compilação podem existir erros léxicos, sintáticos e semânticos. Um bom compilador apresenta uma boa tratativa de erros.

Gerenciador da tabela de símbolos

Responsável por manter a tabela de símbolos atualizada a cada passo do compilador.

Conheça agora as características dos **compiladores**.

Principais características

- Gera código-objeto mais otimizado.
- Executa mais rápido que o processo de interpretação.
- Traduz um mesmo comando apenas uma vez, mesmo que usado em várias partes do programa – tanto iterações como repetição de código.
- Apresenta processo de correção de erros e depuração é mais demorado.
- Possui programação final (código-objeto) maior.
- Gera um programa-objeto dependente de plataforma — processador + SO (sistema operacional) — necessitando de um compilador diferente para cada família de processadores/sistema operacional.

Atividade 1

Avalie as assertivas a seguir:

I. O compilador analisa o código-fonte e o converte para um executável.

II. O montador traduz o código Assembly para o código de máquina intermediário (código objeto), que é executável pelo computador.

III. O carregador é que torna o código-objeto em relocável.

IV. O ligador liga o código-objeto relocável com as rotinas. Essa ligação gera o código executável.

Com base em sua análise, assinale a única opção com todas as assertivas corretas:

A Estão corretas III e IV apenas.

B Estão corretas II, III e IV apenas.

C Está correta apenas a III.

D Está correta apenas a IV.

E Estão corretas I, II e III apenas.

Parabéns! A alternativa A está correta.

Vamos avaliar cada assertiva.

I. Falso. O correto é: o compilador analisa o código-fonte o converte para um código Assembly.

II. Falso. O correto é: o montador traduz o código Assembly para o código de máquina intermediário (código-objeto), que não é executável pelo computador.

III. Verdadeiro. O carregador é responsável pela relocação do código-objeto.

IV. Verdadeiro. O ligador conecta o objeto relocável com as chamadas às rotinas externas.

Interpretação e sistemas

híbridos

A interpretação e os sistemas híbridos são métodos distintos de implementação das linguagens de programação. A interpretação envolve a execução direta do código-fonte por um interpretador, que traduz e executa cada instrução uma de cada vez. Esse método facilita a depuração e a execução em tempo real, mas pode ser menos eficiente.

Por outro lado, os sistemas híbridos combinam tradução e interpretação. O código é inicialmente compilado para um bytecode intermediário, que é posteriormente interpretado ou compilado para execução. Essa abordagem equilibra a eficiência da tradução com a flexibilidade da interpretação, oferecendo um desempenho otimizado e facilidades de depuração.

Neste vídeo, falaremos sobre o processo de interpretação, no qual cada comando do programa-fonte é traduzido e executado imediatamente, e dos sistemas híbridos, que combina a execução rápida dos tradutores com a portabilidade dos interpretadores. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Interpretação

A imagem a seguir ilustra o simples processo de Interpretação. Observe!

Visão geral da interpretação.

Na conversão por interpretação, cada comando do programa-fonte é traduzido e executado (um a um) imediatamente. O interpretador traduz um comando de cada vez e chama uma rotina para completar a sua execução.

O interpretador é um programa que executa repetidamente a sequência que apresentaremos a seguir.

Obter a próxima instrução do código-fonte.

Interpretar a instrução (conversão para comandos em linguagem de máquina).

Executar a instrução.

Perceba que o procedimento é bastante similar àquele executado por computadores que implementam a máquina de Von Neumann, na execução de uma instrução. Confira a seguir.

- Obter a próxima instrução.
- CI → endereço da próxima instrução. CI = contador de instruções.
- RI → instrução a ser executada. RI = registrador de instruções.
- Decodificar a instrução.
- Executar a instrução.

Conheça agora as características dos **interpretadores**.

Principais características

- Atua a cada vez que o programa precisa ser executado.
- Não produz programa-objeto persistente.
- Não traduz instruções que nunca são executadas.
- Apresenta resultado da conversão instantaneamente: resultado da execução do comando ou exibição de erro – interpretador puro.
- É útil ao processo de depuração de código devido a mensagens de erros em tempo de execução (tanto análise

sintática como semântica).

- Apresenta execução mais lenta do que outros processos de tradução (compilação), pois toda vez que o mesmo programa é executado, os mesmos passos de interpretação são executados.
- Consome menos memória.
- Apresenta código-fonte portátil.
    - Não é gerado um código de máquina.
    - Pode executar o comando em alto nível diretamente ou gerar um código intermediário, que é interpretado por uma máquina virtual (VM). – Interpretador híbrido.
    - Se a máquina virtual foi desenvolvida para diferentes plataformas, temos a portabilidade do código-fonte. Este é o caso da linguagem Java.

Tradução (Compilação) x Interpretação

Na tabela a seguir, apresentaremos um comparativo entre tradutores (compiladores) e interpretadores. Confira!

	Vantagens	Desvantagens
Tradutores (Compiladores)	• Execução mais rápida • Permite estruturas de programas mais complexas. • Permite a otimização de código.	• Várias etapas de conversão. • Programação final é maior, necessitando de mais memória para sua execução. • Processo de correção de erros e depuração é mais demorado.

	Vantagens	Desvantagens
Interpretadores	• Depuração mais simples. • Consome menos memória. • Resultado imediato do programa (ou parte dele).	• Execução do programa é mais lenta. • Estruturas de dados demasiado simples. • Necessário fornecer o código fonte ao utilizador.

Tabela: Comparação entre tradutores e interpretadores.
Adaptado de codemastersufs.blogspot.com.

Sistemas híbridos

O processo híbrido de implementação de uma linguagem de programação combina a execução rápida dos tradutores (compiladores) com a portabilidade dos interpretadores. O segredo é a geração de um código intermediário mais facilmente interpretável, porém não preso a uma plataforma (SO/Hardware).

Esse código intermediário não é específico para uma plataforma, possibilitando aos programas já compilados para esse código serem portados em diferentes plataformas, sem alterar e nem fazer nada. Para cada plataforma desejada devemos ter um interpretador desse código.

> **Python e Java, duas importantes linguagens, implementaram o código intermediário, com diferentes formas usando máquinas virtuais.**

Sistema de implementação de Python

Python usa um sistema híbrido, uma combinação de interpretador e tradutor (compilador). O compilador converte o código-fonte Python em um código intermediário, que roda em uma máquina virtual, a PVM (Python Virtual Machine).

Curiosidade

Um código Python pode ser traduzido em Bytecode Java usando a implementação Jython.

O interpretador converte para código de máquina, em tempo de execução. O compilador traduz o programa inteiro em código de máquina e o executa, gerando um arquivo que pode ser executado. O compilador gera um relatório de erros e o interpretador interrompe o processo quando localiza um erro.

CPython é uma implementação da linguagem Python, um pacote com um compilador e um interpretador Python (Máquina Virtual Python), além de outras ferramentas para programar em Python.

Atividade 2

Compreender o conceito de interpretação de código é fundamental na programação.

O que significa interpretação de código nesse contexto?

A Tradução de código-fonte para código de máquina por um compilador.

B Análise do código por um interpretador para executar as instruções linha por linha.

C Conversão do código em uma representação gráfica para facilitar a compreensão.

D Processo de otimização do código para melhorar seu desempenho durante a execução.

E Verificação da sintaxe do código para garantir sua correção antes da execução.

Parabéns! A alternativa B está correta.

A interpretação de código envolve a análise do código-fonte por um interpretador, que executa as instruções do programa linha por linha em tempo real, sem a necessidade de compilação prévia para código de máquina. Isso permite uma execução mais flexível e interativa do código.

O que você aprendeu neste conteúdo?

- O papel da abstração nas linguagens de programação.
- Classificação das linguagens de programação.
- Critérios para avaliação das linguagens de programação.
- Classificação dos paradigmas das linguagens de programação.
- Processos de conversão dos programas: tradução e interpretação.

Explore +

Para saber mais sobre os assuntos tratados neste tema, pesquise na internet sobre:

- Objective C, que é a linguagem utilizada para desenvolvimento de aplicativos iOS e MacOS.
- Plataformas sobre a linguagem Python.
- Plataformas de programadores, para acompanhar as linguagens de programação mais usadas no mercado.
- Visual Studio, que oferece uma cadeia de ferramentas em dispositivos móveis e na nuvem.

Referências

BORGES, Luiz Eduardo. **Python para Desenvolvedores**. 2. ed. Rio de Janeiro: Edição do Autor, 2010.

SEBESTA, R. W. **Conceitos de Linguagens de Programação**. Tradução de João Eduardo Nobrega Tortello. 11. ed. Porto Alegre: Bookman, 2018. Disponível na biblioteca virtual da Estácio.

VAREJÃO, F. M. **Linguagem de Programação**: conceitos e técnicas. 2ª reimpressão. Rio de Janeiro: Elsevier, 2004.

DIJKSTRA EW. **A Discipline of Programming**. Hoboken: Prentice Hall, 1976.