DEFINIÇÃO

As linguagens de programação mais populares por nível, por geração, por paradigmas, destacando o posicionamento da linguagem Python, critérios usados na avaliação de linguagens de programação dos diferentes domínios da aplicação, classificação dos diferentes paradigmas de linguagens de programação e formas de implementação das linguagens.

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

Classificar as linguagens de programação

MÓDULO 2

Descrever critérios de avaliação de linguagens de programação

MÓDULO 3

Distinguir os paradigmas e suas características

MÓDULO 4

Identificar métodos de implementação das linguagens

INTRODUÇÃO

Desde o surgimento dos computadores, centenas de linguagens de programação vêm sendo criadas com o objetivo de permitir ao programador mais eficiência e conforto ao escrever seus códigos.

Muitos são os problemas que hoje em dia são solucionados com a ajuda de softwares, escritos sempre em uma linguagem de programação (que por sua vez também é um software) e muitas vezes esses problemas demandam diferentes pensamentos computacionais em sua solução.

A classificação das linguagens em **paradigmas** permite que entendamos qual é o melhor deles para solucionar determinado problema e, a partir daí, escolher a linguagem de programação (pertencente a esse paradigma) mais adequada, conforme características e especificidades do contexto em que se aplica o problema.

PARADIGMAS

Paradigmas é uma forma de agrupar e classificar as linguagens de programação baseados em suas funcionalidades.

A linguagem Python foi escolhida como instrumento para o desenvolvimento desta disciplina, pois além de ser multiparadigma (possibilita escrever programas em diferentes paradigmas) e de uso geral, vem se destacando e sendo cada vez mais utilizada entre os novos desenvolvedores por vários motivos:

Facilidade de aprendizado;

Boa legibilidade de código;

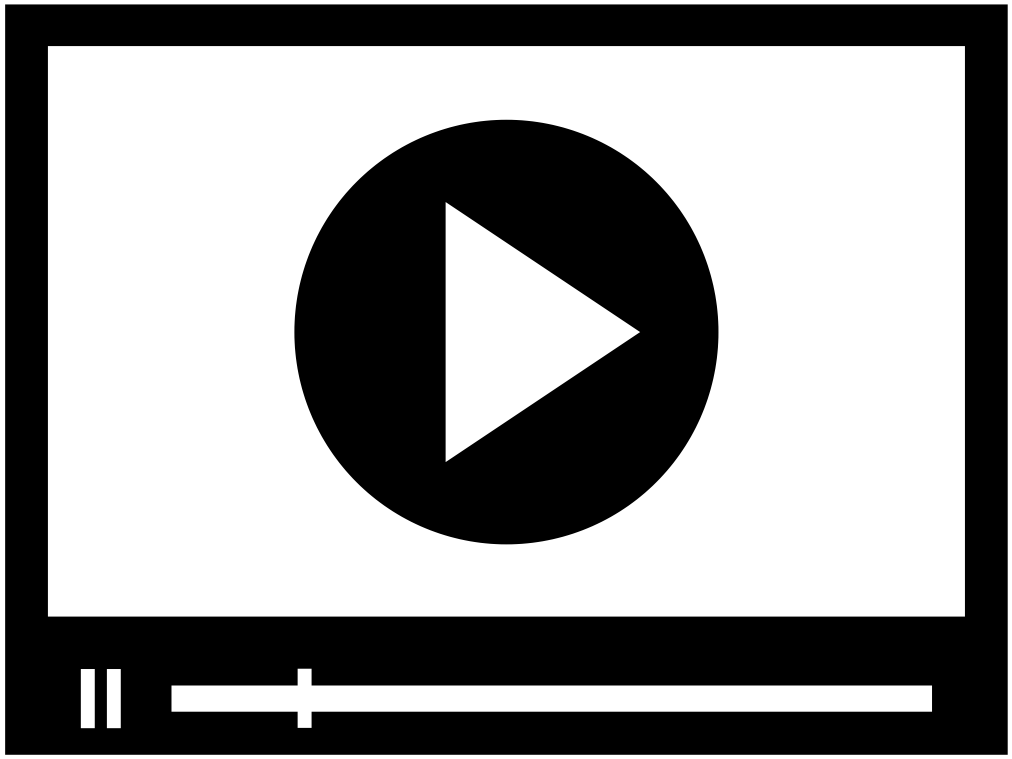
Boa facilidade de escrita;

Produtividade e confiabilidade.

Possui, ainda, comunidade de desenvolvedores crescente e vasta biblioteca, repleta de funções, aplicada a diversas áreas da ciência, assim como o crescente números de frameworks desenvolvidos para a linguagem.

MÓDULO 1

- Classificar as linguagens de programação



CLASSIFICANDO AS LINGUAGENS DE PROGRAMAÇÃO

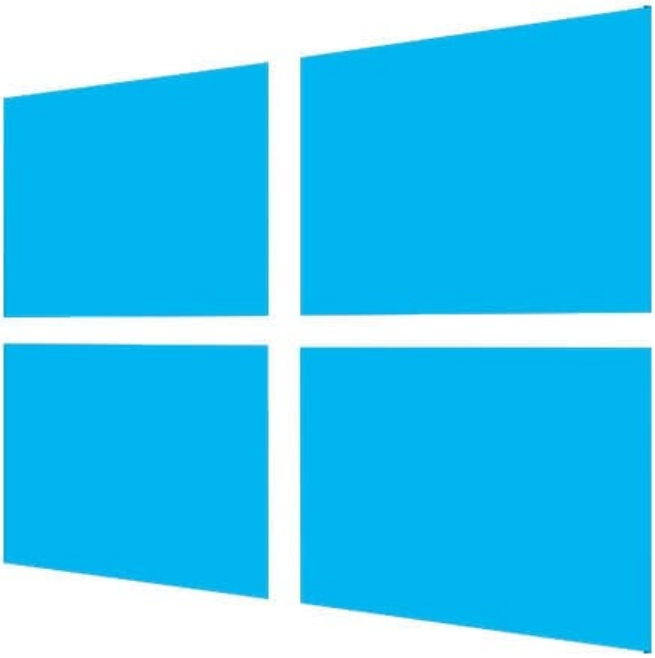
RAZÕES PARA ESTUDARMOS LINGUAGENS DE PROGRAMAÇÃO

LINGUAGEM DE PROGRAMAÇÃO E PRODUTIVIDADE DO PROGRAMADOR

Um programa de computador, ou software, é um conjunto de instruções a fim de orientar o hardware do computador para o que deve ser feito. O software pode ser classificado em aplicativo ou básico.

SOFTWARE APLICATIVO

Ou, simplesmente, aplicativo ou app (mundo mobile), que visa oferecer ao usuário facilidades para realizar uma tarefa laboral ou de lazer.



SOFTWARE 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.

📢 ATENÇÃO

As linguagens de programação surgiram da necessidade de livrar 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 código em uma linguagem de programação está intimamente relacionada à facilidade de aprendizado, leitura e escrita de programas naquela linguagem, somada a expertise do programador no contato com a linguagem.

Isto é, quanto mais o programador conhecer as propriedades superlativas daquela linguagem, melhores e mais eficientes serão os códigos escritos.

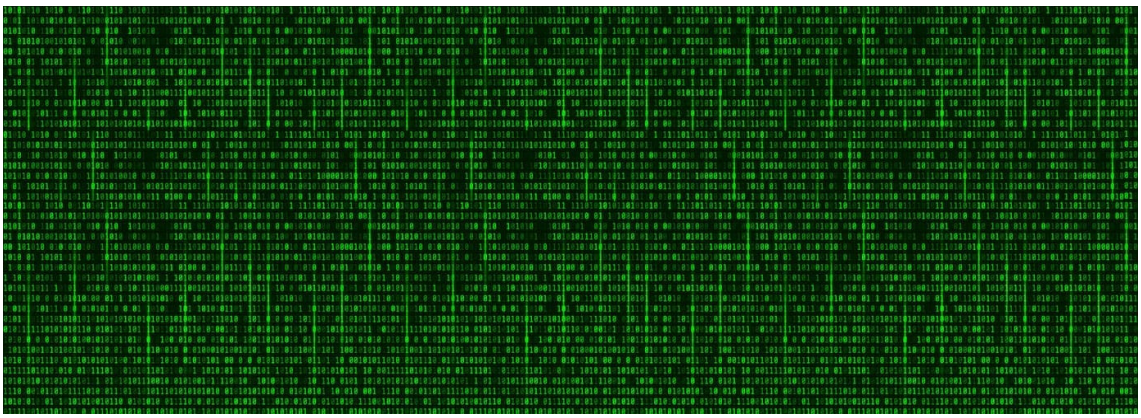
O ASPECTO MAIS IMPORTANTE, MAS TAMBÉM O MAIS ELUSIVO DE QUALQUER FERRAMENTA, É SUA INFLUÊNCIA NOS HÁBITOS DAQUELES QUE SE TREINAM NO SEU USO. SE A FERRAMENTA É UMA LINGUAGEM DE PROGRAMAÇÃO, ESSA INFLUÊNCIA É, GOSTEMOS OU NÃO, UMA INFLUÊNCIA EM NOSSO HÁBITO DE PENSAR.

Edsger W. 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, a única que ele, de fato, compreende. Uma 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, muito difícil de ser memorizada, usada e, mais ainda, de ser entendida por terceiros.

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

❓ VOCÊ SABIA

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, à esquerda, um programa-fonte, escrito numa linguagem de alto nível, a linguagem Python. Ao centro, temos o código equivalente na linguagem Assembly para o sistema operacional Linux e, à direita, 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 $16,0($2)
	Sw $15,4($2)
	Jr $31
		00000000001111111111100000000001
		00011111111000000111000011111101
		11111000001100000111111110000000
		10000000100000001000000010000000
		00000000010000000001000000000010
		00000000000000001111000010010101
		00000000111000111111001111111111

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

LINGUAGEM PYTHON

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.

ASSEMBLY

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).

LINGUAGEM DE MÁQUINA

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

📷 Crescimento do nível de abstração. Fonte: Autor.

NÍVEL 1

É representado pelo hardware, conjunto de circuitos eletrônicos.

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 numa linguagem, conhecendo melhor suas funcionalidades e implementações, ajuda para que o programador possa 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.

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

CLASSIFICAÇÃO DAS LINGUAGENS DE PROGRAMAÇÃO

Ao longo dos anos, os autores têm criado diferentes classificações para as linguagens de programação, usando critérios diferenciados e agrupando-as sob diferentes perspectivas.

Veremos a seguir as classificações das linguagens por nível, por gerações e por paradigmas.

CLASSIFICAÇÃO POR NÍVEL

A classificação por nível considera a proximidade da linguagem de programação com as características da arquitetura do computador ou com a comunicação com o homem.

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 linguagens 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).

Abaixo, a imagem 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. Fonte: Autor

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

0001001010001111
1010010001000010
0010101110110111
0101010000000111

Era de fato muito complexa a programação na linguagem de máquina, a linguagem nativa dos processadores.

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.

★ EXEMPLO

As três linhas de código na linguagem Assembly, abaixo, que 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).

MOV AX, 0002

MOV BX, 0001

ADD AX, BX

Não chega a ser o ideal em termos de uma linguagem, que é 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.

Linguagens de baixo nível estão distantes da língua humana (escrita).

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).

❓ VOCÊ SABIA

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.

⭐ EXEMPLO

Abaixo, o mesmo código expresso acima, escrito em Assembly, porém usando variáveis, como abstração do armazenamento e codificado na linguagem Python.

```
def main():
num1 = 2
```

```
num2 = 1
soma = num1 + num2
```

Abaixo, o mesmo código na linguagem C:

```
int num1, num2, soma;
int main()
{
num1=2;
num1=1;
soma=num1+num2;
}
```

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.

SAIBA MAIS

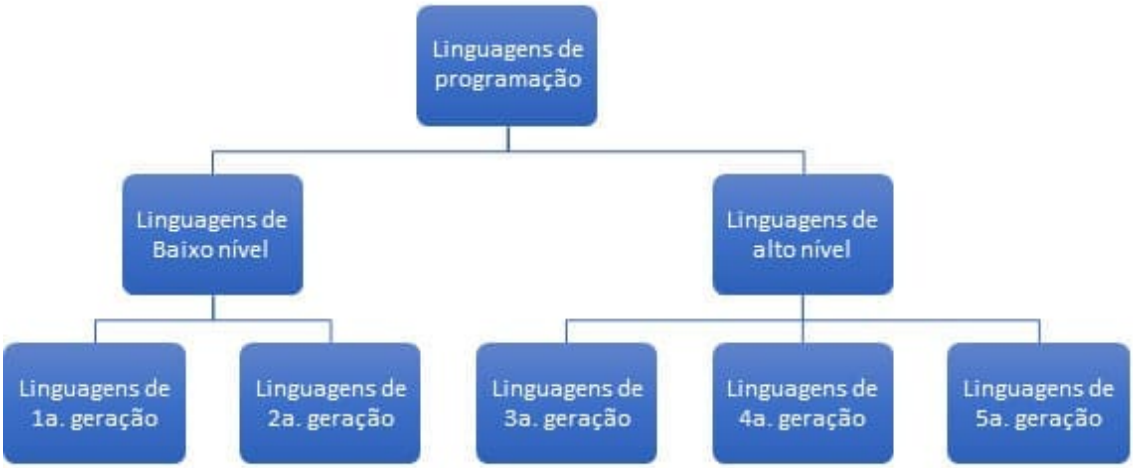
Há uma **curiosidade**: C e C++ são classificados por alguns autores como linguagem de médio nível, na medida que estão próximas da linguagem humana (linguagem de alto nível), mas também estão próximas da máquina (linguagem de baixo nível), pois possuem instruções que acessam diretamente memória e registradores.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

Outro dado que merece ser comentado é que 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.

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.



Veja um pouco mais sobre as gerações:

LINGUAGENS DE 1ª GERAÇÃO (LINGUAGEM DE MÁQUINA)

A 1ª geração de linguagens é representa pela linguagem de máquina, nativa dos processadores.

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 abaixo para uma CPU abstrata. Considere a seguinte sequência de 3 instruções em linguagem Assembly:

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

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

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

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

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, também, 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.

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.

Esta geração de linguagens apresenta as seguintes propriedades em comum:

Armazenar tipos de dados estaticamente: simples, estruturados e enumerados.

Alocar memória dinamicamente, através de ponteiros, que são posições de memória cujo conteúdo é outra posição de memória.

Disponibilizar: estruturas de controle sequencial, condicional, repetição e desvio incondicional.

Permitir a programação modular, com uso de parâmetros.

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.

Enquanto as linguagens de 3ª geração são procedurais, ou seja, especifica-se passo a passo a solução do problema, as de 4ª geração são não procedurais. O programador especifica o que deseja fazer e 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.

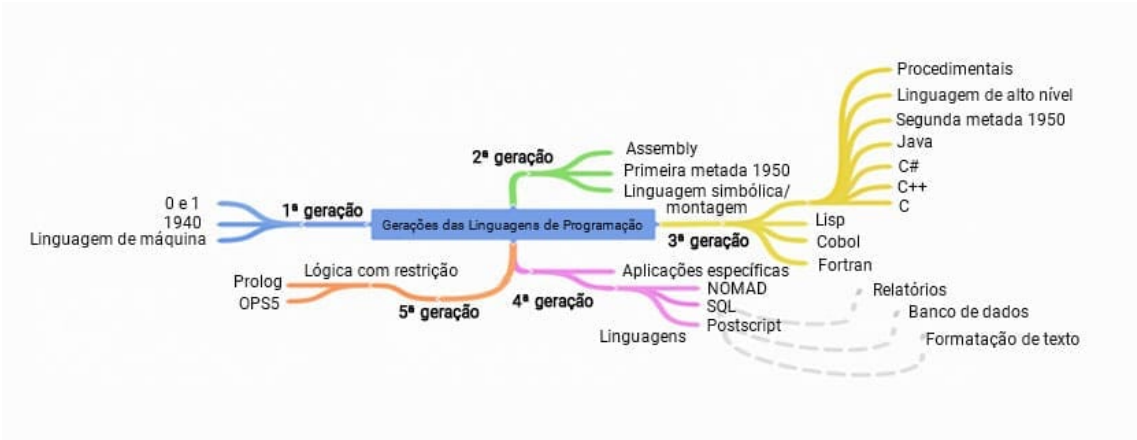
LINGUAGENS DE 5ª GERAÇÃO (VOLTADAS À INTELIGÊNCIA ARTIFICIAL)

São linguagens declarativas e não algorítmicas. Exemplos: 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.

A imagem a seguir ilustra as características de cada geração.

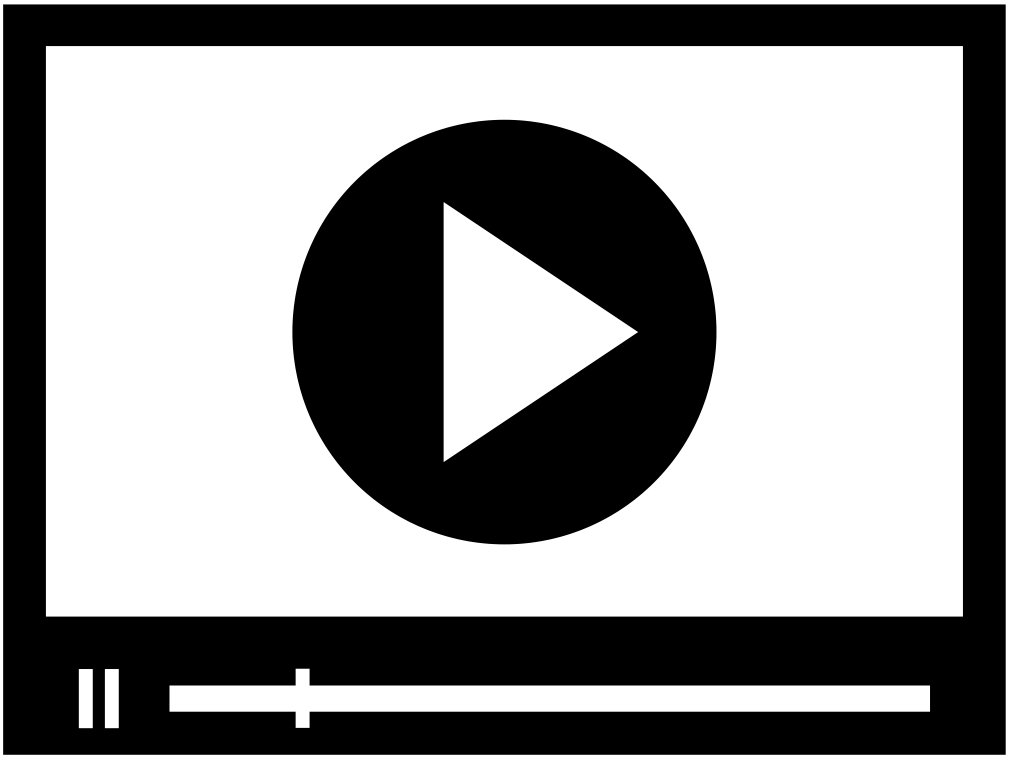
Alguns autores classificam a 6ª geração, como uma evolução da 5ª, em que prevalecem as aplicações de redes neurais, uma outra vertente da Inteligência Artificial.



VERIFICANDO O APRENDIZADO

MÓDULO 2

- Descrever critérios de avaliação de linguagens de programação



CRITÉRIOS DE AVALIAÇÃO DAS LINGUAGENS DE PROGRAMAÇÃO

Considerando as diversas linguagens de programação existentes hoje no mercado, atendendo a propósito comuns, vamos destacar neste módulo os domínios da programação, que são seis:

Aplicações científicas

Aplicações comerciais

Aplicações com Inteligência Artificial

Programação de sistemas

Programação para web

Programação mobile

Na sequência, apresentaremos critérios que podem ser usados para avaliação de linguagens de programação, claro, dentro do mesmo domínio de programação.

DOMÍNIOS DA PROGRAMAÇÃO

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 controlar 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 3 anos e ficou pronto no ano de 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. 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 é que as linguagens que apoiam a IA usam a computação simbólica e não numérica, como a maioria das linguagens da época. Em 1959, surge a linguagem Lisp, primeira linguagem projetada para apoio à computação simbólica, primeira referência da computação funcional. Prolog, criada em 1977, foi a primeira linguagem para apoio da computação lógica, essência dos sistemas especialistas (sistemas que usam 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. Antes de surgir a linguagem C, usada para desenvolver o sistema operacional Linux, 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 2 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 3 camadas (apresentação, lógica do negócio e persistência de dados).

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: Java e Kotlin.

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

Windows: C#, Visual Basic (VB), C++, HTML, CSS, JavaScript e Java.

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.

A Google, por sua vez, tem por base o Android SDK, orienta a usar as linguagens Kotlin, Java e C++, mas as linguagens Python, Shell script, Basic4Android, LiveCode (para iOS e Windows também), App Inventor (não necessita conhecer programação) e Unity (motor para games) e GO, também são usadas para desenvolver app para Android.

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.

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

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.

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	Confiabilidade
Simplicidade			
Ortogonalidade			
Estruturas de controle			

Tipos de dados			
Projeto de sintaxe			
Suporte para abstração			
Expressividade			
Verificação de tipos			
Tratamento de exceções			
Aliasing			

Características x Critérios de Avaliação de LPs

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

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.

As características que influenciam a legibilidade de uma linguagem de programação são:

SIMPLICIDADE

Quanto mais simples for uma linguagem, melhor será a legibilidade do código por ela produzido. Uma linguagem com número elevado de construções básicas é mais difícil de ser aprendida do que uma que tenha poucas. Tende a ser subutilizada.

Uma segunda característica que afeta negativamente a legibilidade é a multiplicidade de recursos. Por exemplo, em Python, o programador pode incrementar uma variável, de duas formas distintas:

cont= cont + 1;

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: possibilidade de combinar, entre si, sem restrições, as construções básicas da linguagem para construir estruturas de dados e de controle.

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. 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. Por 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 6 caracteres.

Outra propriedade de sintaxe que afeta a legibilidade é o uso de palavras reservadas da linguagem. Por exemplo, em Pascal, os blocos de instrução são iniciados e encerrados com BEGIN-END, respectivamente. A linguagem C usa chaves para iniciar e encerrar blocos de instruções. Já a linguagem Python usa 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 se a escrita do código não flui, haverá dificuldade para quem for ler o código.

As características que influenciam na facilidade de escrita são:

SIMPLICIDADE E ORTOGONALIDADE

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

Imagina 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.

Assembly: Baixa expressividade.

Pascal e C, boa expressividade: Ricas estruturas de controle. Exemplo: o comando FOR 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. Abstração pode ser de:

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.

Abaixo, 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

O tratamento de exceção em uma linguagem de programação 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)

Aliasing 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.

Custo de	Características
Treinamento	Custo de Treinamento para programadores varia em função da expertise do programador, simplicidade e ortogonalidade da linguagem;

	F (simplicidade de escrita, ortogonalidade, experiência do programador).
Escrever programa	Custo para escrever programas na linguagem varia em função da facilidade de escrita. F(Facilidade de escrita).
Compilar o programa	Esse custo 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).
Executar o programa	Custo para executar programas, varia em função do projeto da linguagem. F (Projeto da linguagem).
Implementar a linguagem	A popularidade da LP vai depender de um econômico sistema de implementação. Por exemplo, Python e Java possuem compiladores e interpretadores gratuitos.
Confiabilidade	O custo da má confiabilidade: se um sistema crítico falhar, o custo será elevado. Exemplos: sistema de controle de consumo de água e sistemas de usina nuclear.
Manutenção	Custo de 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.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

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 confiável. Seu custo não é elevado, pois além de ser open source, é fácil de aprender.

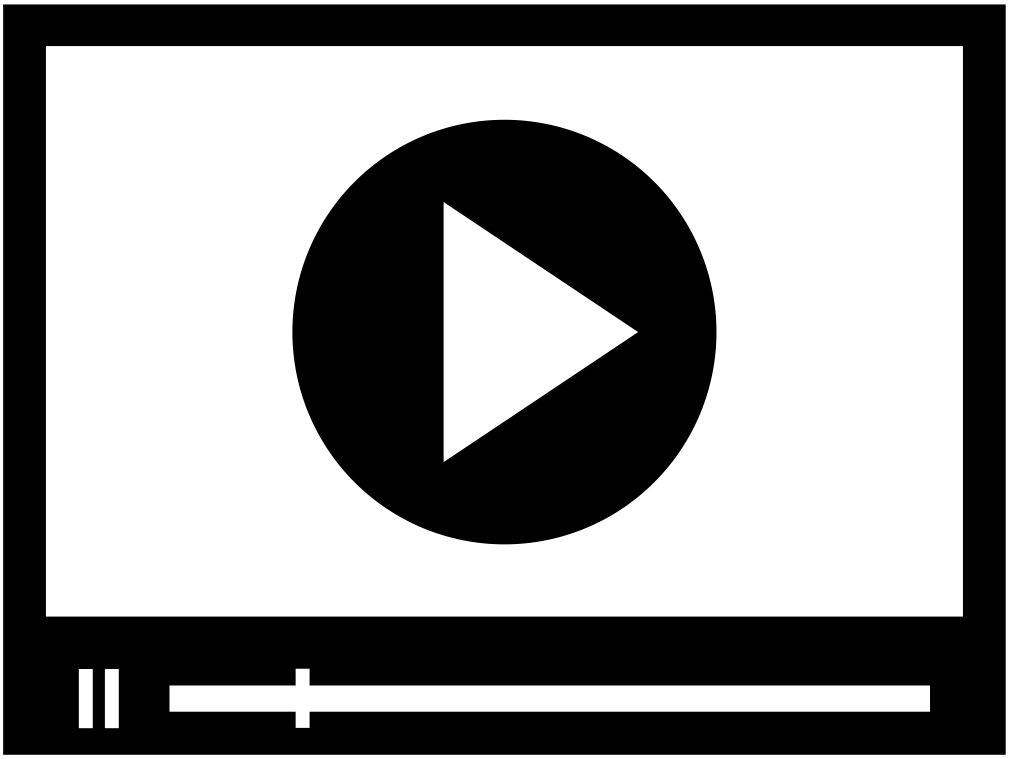
📢 ATENÇÃO

Existem outros critérios, 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. 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.

VERIFICANDO O APRENDIZADO

MÓDULO 3

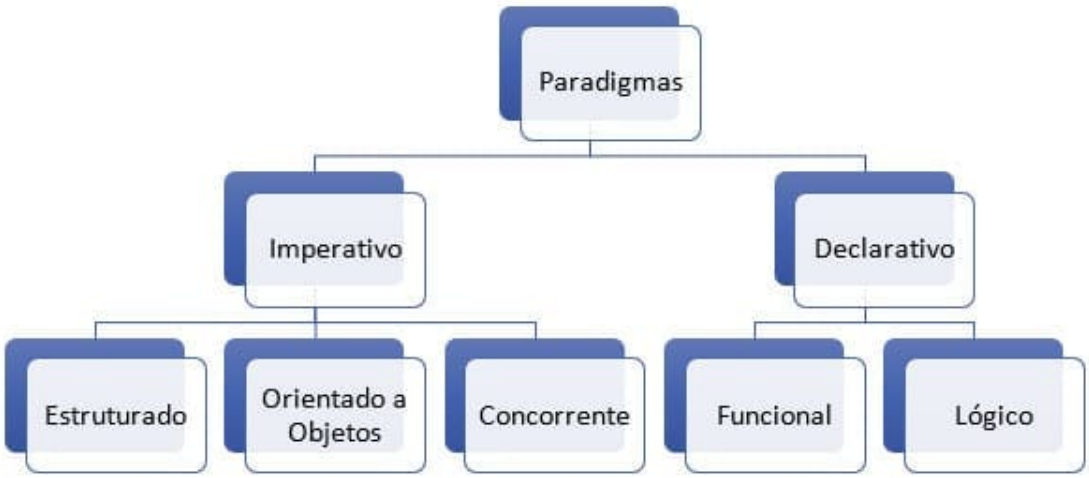
- ⦿ Distinguir os paradigmas e suas características



PARADIGMAS E SUAS CARACTERÍSTICAS

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. Fonte: Autor, adaptado de (VAREJÃO, 2004, p.17).

PARADIGMA IMPERATIVO

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.

PARADIGMA ESTRUTURADO

Caracteriza as principais linguagens de programação da década de 1970 e 1980 que seguiram os princípios da programação estruturada:

PRINCÍPIO 01

PRINCÍPIO 02

PRINCÍPIO 03

PRINCÍPIO 04

1

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

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

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

4

Visando eficiência, o paradigma estruturado baseia-se 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.

Resultados das operações trafegam da CPU para a memória.

As linguagens Pascal e C caracterizam bem esse paradigma. 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 desse paradigma 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

Por exemplo, a classe ALUNO encapsula 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.

📢 ATENÇÃO

Phyton, Smalltalk, C++, Java, Delphi (oriundo do Object Pascal) são linguagens que caracterizam o paradigma orientado a objetos. Python é orientado a objeto, pois tudo em Python é objeto, permitindo a declaração de classes encapsuladas, além de possibilitar herança e polimorfismo.

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.

UM PROCESSADOR

Os processos concorrem ao uso do processador e recursos.

VÁRIOS PROCESSADORES

Estamos caracterizando o paralelismo na medida em que podem executar em diferentes processadores (e de fato, ao mesmo tempo), os quais podem estar em uma mesma máquina ou distribuídos em mais de um computador.

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

❓ VOCÊ SABIA

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.

I/O BOUND

Aplicações I/O bound são aquelas em que há predomínio de ações de entrada e saída de dados.

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.

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).

O 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, esse paradigma gera programas menores (pouco código).

Linguagens típicas desse paradigma são: 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.

Exemplo: O código abaixo implementa em Python uma função que calcula quantos números inteiros existem de 0 a n.

```
def conta_numeros(n):
p = 0
for num in range(n+1):
if num%2 == 0:
```

```
p += 1
return p
```

Abaixo, 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).

```
def conta_numeros(n):
if n == 0: return 1 # 0 é par
elif n%2 == 0: return 1 + conta_numeros(n-1)
else: return conta_numeros(n-1)
```

📢 ATENÇÃO

Python não é uma linguagem funcional nativa, seria exagerado afirmar isso, porém sofreu influência desse paradigma ao permitir: recursividade, uso de funções anônimas, com a função lambda, dentre outros recursos, além, claro, de ser uma linguagem com enorme 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.

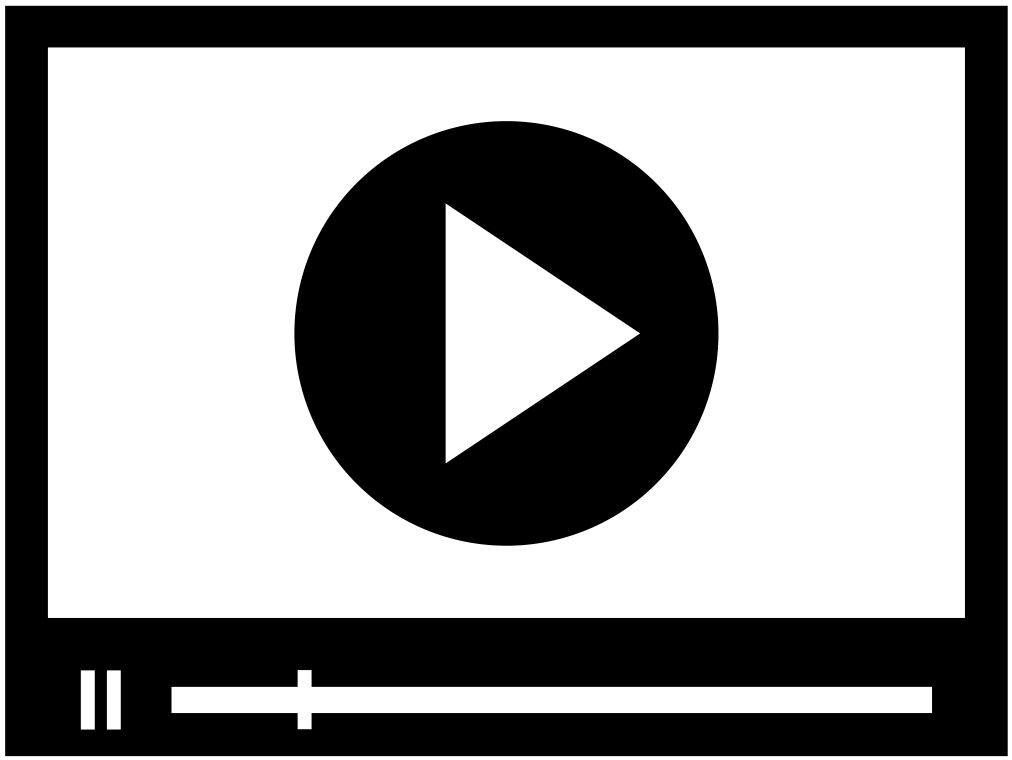
O paradigma lógico é usado no desenvolvimento de linguagens de acesso a banco de dados, sistemas especialistas (IA), tutores inteligentes etc.

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

VERIFICANDO O APRENDIZADO

MÓDULO 4

- Identificar métodos de implementação das linguagens.



MÉTODOS DE IMPLEMENTAÇÃO DE LINGUAGENS

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 fundamental que se saiba e se entenda qual o processo de conversão usado na respectiva linguagem de programação.

TRADUÇÃO

Nesse processo de conversão, o programa escrito em uma linguagem de alto nível é traduzido para uma versão equivalente em linguagens de máquina, antes de ser executado. O processo de tradução 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: Compilação, Montagem, Carga e Ligação. A imagem abaixo ilustra o processo de tradução.

Código-fonte

Compilador

COMPILADOR

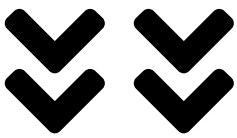
O compilador (detalhes adiante) analisa o código-fonte e estando tudo OK, o converte para um código Assembly (da máquina hospedeira).

Código Assembly

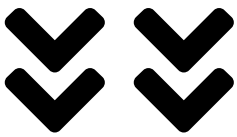
Montador

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.



Código Assembly



Ligador

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.

Código-objeto relocável

Carregador

CARREGADOR

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

Código objeto

Código executável

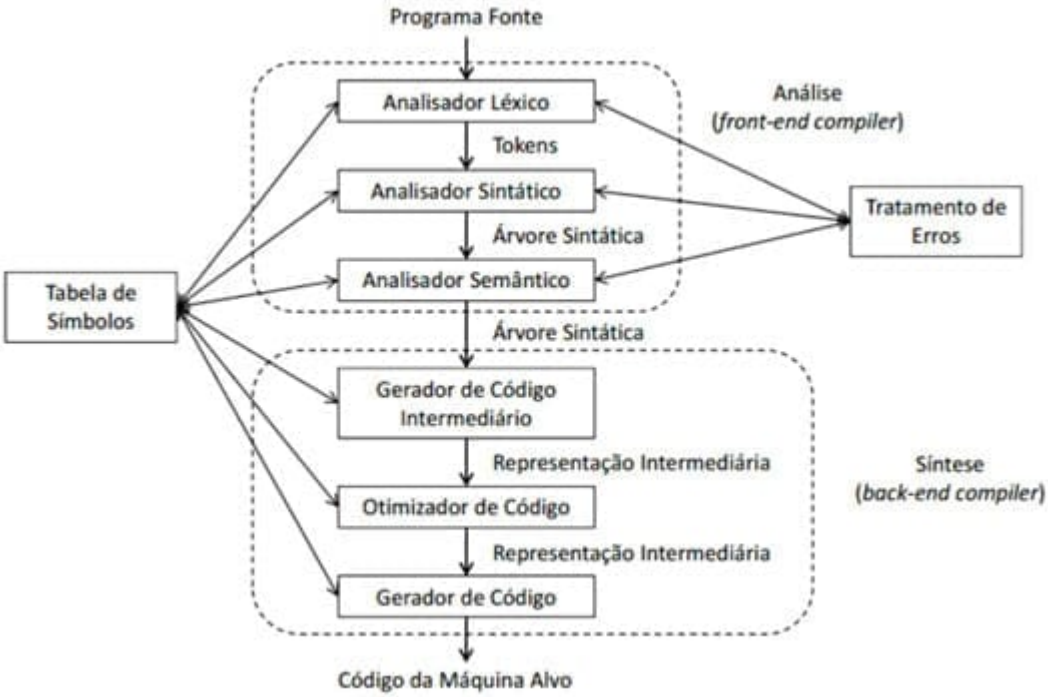
Fases de tradução de um programa em linguagem de alto nível.**Fonte**: Autor.

COMPILADOR

É o elemento central do processo de tradução, responsável pelo custo de compilação, visto no modulo anterior. 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 abaixo ilustra os componentes envolvidos na compilação de um programa fonte:



📷 Fonte: https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/introduction-and-overview-about-compilers.html

Abaixo, vamos entender cada fase da compilação:

ANÁLISE LÉXICA

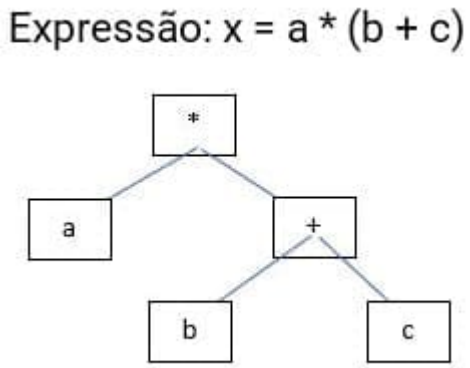
Identifica os tokens (elementos da linguagem), desconsidera partes do código-fonte, como espaços em branco e comentários e gera a Tabela de símbolos, com todos esses tokens, que são identificadores de variáveis, de procedimentos, de funções, comandos, expressões etc.

ANÁLISE SINTÁTICA

Verifica se os tokens são estruturas sintáticas (exemplos: expressões e comandos) válidas, aplicando as regras gramaticais definidas no projeto da linguagem.

ANÁLISE SEMÂNTICA

Verifica se as estruturas sintáticas possuem sentido. Por exemplo, verifica se um identificador de variável ou constante é usado adequadamente, se operandos e operadores são compatíveis. Monta a árvore de derivação conforme ilustrado abaixo para formação das expressões.



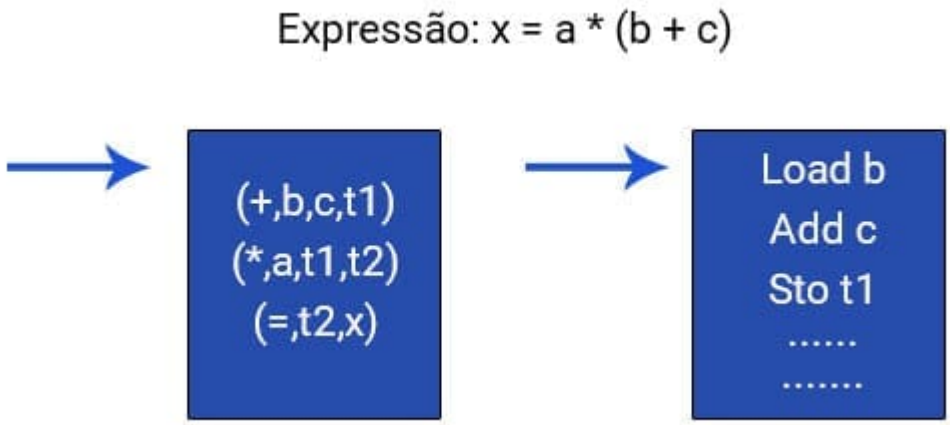
Árvore de derivação de uma expressão. Fonte: Autor

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:



> 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 existem erros: léxicos, sintáticos e semânticos. Um bom compilador apresenta uma boa tratativa de erros.

GERENCIADOR DA TABELA DE SÍMBOLOS

Mantém a tabela de símbolos atualizada a cada passo do compilador.

📢 ATENÇÃO

As principais características dos compiladores são:

Gerar código-objeto mais otimizado.

Execução mais rápida 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.

Processo de correção de erros e depuração é mais demorado.

A programação final (código-objeto) é maior.

O programa-objeto gerado é dependente de plataforma — processador + SO (Sistema Operacional) — necessitando de um compilador diferente para cada família de processadores/sistema operacional.

INTERPRETAÇÃO

A imagem abaixo ilustra o simples processo de 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 seguinte sequência:

1

Obter a próxima instrução do código-fonte.

2

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

Executar a instrução.

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

PRINCIPAIS CARACTERÍSTICAS DO INTERPRETADOR

Dentre as principais características do interpretador, podemos citar:

Atua a cada vez que o programa precisa ser executado.

Não produz programa-objeto persistente.

Não traduz instruções que nunca são executadas.

O resultado da conversão é instantâneo: resultado da execução do comando ou exibição de erro – interpretador puro.

Útil ao processo de depuração de código devido a mensagens de erros em tempo de execução (tanto análise sintática como semântica).

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.

O Código-fonte é portátil.

Não é gerado um código de máquina.

Pode executar o comando em alto nível diretamente ou gerar um código intermediário, que neste caso 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 X INTERPRETAÇÃO

	Vantagens	Desvantagens
Tradutores	1. Execução mais rápida 2. Permite estruturas de programas mais complexas. 3. Permite a otimização de código.	1. Várias etapas de conversão. 2. Programação final é maior, necessitando de mais memória para sua execução. 3. Processo de correção de erros e depuração é mais demorado.
Interpretadores	1. Depuração mais simples. 2. Consome menos memória.	1. Execução do programa é mais lenta. 2. Estruturas de dados demasiado simples.

	3. Resultado imediato do programa (ou parte dele).	3. Necessário fornecer o código fonte ao utilizador.

Fonte: http://codemastersufs.blogspot.com/2015/10/compiladores-versus-interpretadores.html. Adaptado pelo autor.

Atenção! Para visualização completa da tabela utilize a rolagem horizontal

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.

Duas importantes linguagens implementaram essa solução, com diferentes formas usando máquinas virtuais: Python e Java.

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 numa máquina virtual, a PVM (Python Virtual Machine).

💬 COMENTÁRIO

Curioso saber que o 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 na medida em que 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.

VERIFICANDO O APRENDIZADO

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Saber classificar uma linguagem dentro do(s) paradigma(s) a que pertença(m), assim como conhecer os domínios de programação de uma linguagem, é fundamental para seu entendimento e correta aplicação na hora de decidir pelo uso dessa ou aquela linguagem.

É fundamental também que se possa conhecer os critérios de avaliação de uma linguagem e quais propriedades afetam cada um dos quatro critérios vistos. Dessa forma, vamos saber selecionar a que melhor se aplica a determinado contexto de uso.

Por fim, e não menos importante, conhecer o mais profundamente possível o ambiente de desenvolvimento da linguagem, contemplando tradutores e/ou interpretadores, para que se possa pôr em prática o desenvolvimento do sistema, no paradigma, domínio do problema, e usando as melhores construções para gerar códigos legíveis, com alta facilidade de escrita, confiabilidade e, preferencialmente, ao menor custo possível.

PODCAST

⚠ PODCAST

AVALIAÇÃO DO TEMA:

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**. Prentice Hall Inc;1976. 217.

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.

CONTEUDISTA

Marcelo Vasques de Oliveira

🔗 CURRÍCULO LATTES