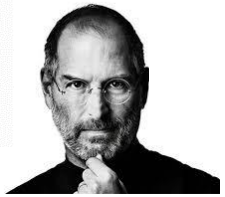
PARADIGMAS DE LINGUAGENS DE PROGRAMAÇÃO EM PYTHON

PARADIGMAS DE LINGUAGENS DE PROGRAMAÇÃO: MOTIVAÇÃO E PRELIMINARES


Ana Carolina Costa de Oliveira
ana.coliveira@estacio.br

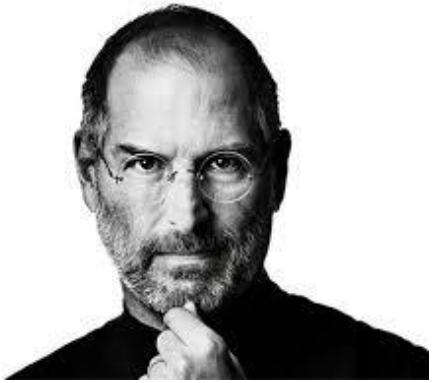
Estácio

Agenda

- Razões para estudar conceitos de linguagens de programação
- Domínios de programação
- Critérios de avaliação de linguagens
- Influências no projeto de linguagens
- Categorias de linguagens
- Trade-off no projeto de linguagens
- Métodos de implementação
- Ambientes de programação

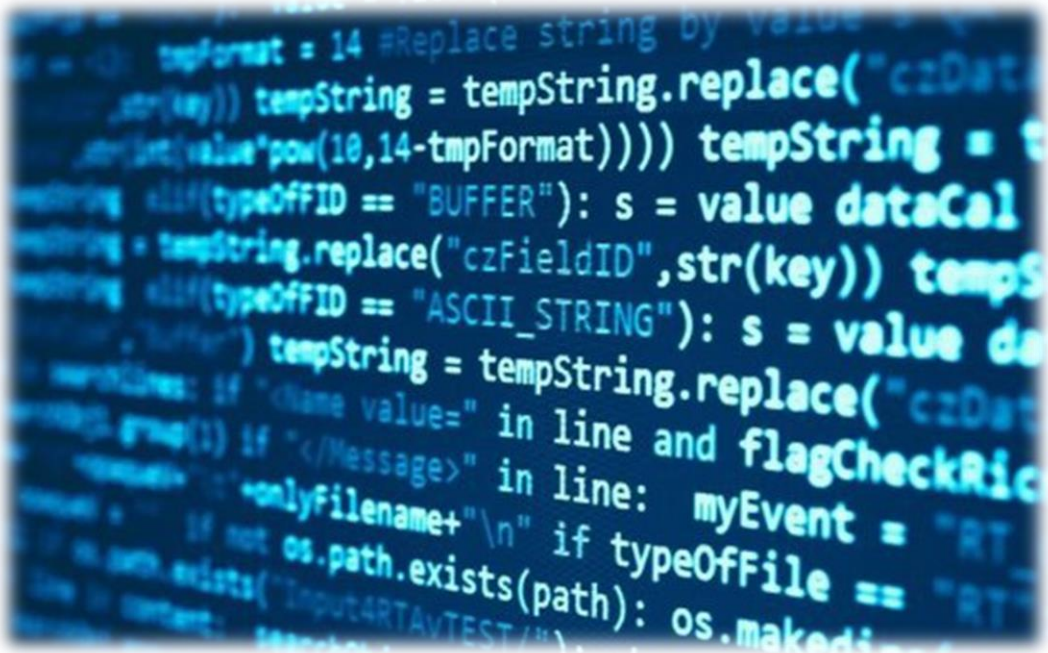
Razões para estudar linguagens de programação

Drew Houston
o criador do Dropbox



Steve Jobs
o criador do Apple

Mark Zuckerberg
o criador do facebook



Razões para estudar conceitos de linguagens de programação

É natural que os estudantes se perguntem como se beneficiarão com o estudo de conceitos de linguagem de programação.

VÁRIOS CONCEITOS NA ÁREA DA TECNOLOGIA SÃO MERECEDORES DE UM ESTUDO DETALHADO.

Estácio

VAMOS LISTAR POTENCIAIS VANTAGENS DE ESTUDAR ALGUNS CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

- Aumento da capacidade de expressar ideias;
- Embasamento para escolher linguagens adequadas;
- Aumento da habilidade para aprender novas linguagens;
- Melhor entendimento da importância da implementação;
- Melhor uso de linguagens já conhecidas;
- Avanço geral da computação.

Domínio de programação

- Aplicações científicas
- Aplicações empresariais
- Inteligência artificial
- Software para a web

Critérios de avaliação de linguagens

- Os critérios de avaliação são necessariamente **controversos**, porque é difícil fazer dois cientistas da computação **concordarem** com **o valor de certas** características das linguagens em relação às outras.

Ray Kurzweil, do Google

Robert Taylor, uma das maiores mentes por trás da criação da internet.

Critérios de avaliação de linguagens e as características que os afetam

Característica	Critérios		
	Legibilidade	Facilidade de escrita	Confiabilidade
Simplicidade	*	*	*
Ortogonalidade	*	*	*
Tipo de dados	*	*	*
Projeto de sintaxe	*	*	*
Suporte para abstração		*	*
Expressividade		*	*
Verificação de tipos			*
Tratamento de exceções			*
Apelidos restritos			*

Legibilidade

- É uma qualidade que determina a facilidade de leitura de alguma coisa.
- É a facilidade com que os programas podem ser lidos e entendidos.
- A legibilidade deve ser considerada no contexto do domínio do problema.

Por exemplo, se um programa que descreve um cálculo é escrito em uma linguagem que não foi projetada para tal uso, ele pode não ser natural e se ser desnecessariamente complexo, tornando complicada sua leitura (quando geralmente, seria algo simples)

Simplicidade

- É a ausência de complicação.
- A simplicidade geral de uma linguagem de programação afeta muito sua legibilidade.
- Uma linguagem com muitas construções básicas é mais difícil de aprender do que uma com pouco.

Por exemplo,

```
count = count + 1
count +=1
count ++
++ count
```

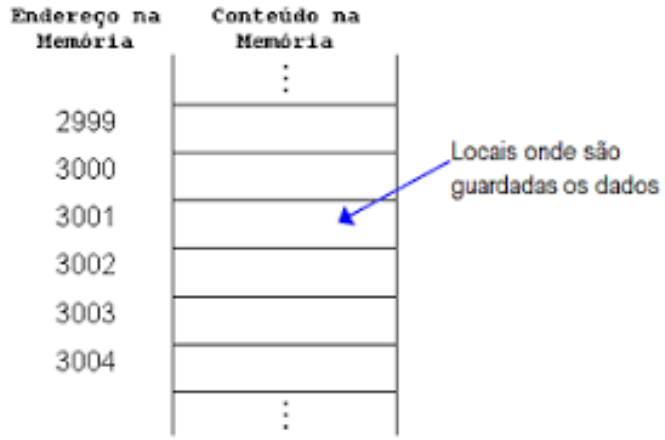
Ortogonalidade

- Em uma linguagem de programação significa que um **conjunto relativamente pequeno** de construções primitivas pode ser combinados a um número relativamente pequeno de formas para construir as estruturas de controle e de dados da linguagem.
- O significado de um recurso de linguagem ortogonal é independente do contexto de sua aparição em um programa.

Ortogonalidade

- A falta de ortogonalidade leva a exceções às regras da linguagem.

Por exemplo, em uma linguagem de programação que suporta ponteiros, deve ser possível definir um ponteiro que aponte para qualquer tipo específico nela definido. Entretanto, se não for permitido aos ponteiros apontar para vetores, muitas estruturas de dados potencialmente úteis definidas pelos usuários não poderão ser definidas.



Tipos de dados

- A presença de mecanismos adequados para definir tipos e estruturas de dados é outro auxílio significativo à legibilidade.

Por exemplo, suponha que um tipo numérico seja usado como um flag* porque não existe nenhum tipo booleano na linguagem.

timeout = 1

Em linguagem que inclui tipos booleanos, teríamos:

timeout = true

*A utilização de **flags** como interruptor (isto é, valores 1/0, ligado/desligado, ativo/inativo) permite otimizar as estruturas de dados, na medida em que basta apenas um bit para ativar determinada característica.

Projeto da sintaxe

- A sintaxe, ou forma, dos elementos de uma linguagem tem um efeito significativo na legibilidade dos programas.
  - Palavras especiais
  - Forma e significado

```
#include<stdio.h>

/* Programa para demostrar as declarações de varaiaveis*/

void main()

{
      //Declaração das variaveis locais:
      char str;
      int N1;
      float x;
      double xx;

      //Atribuindo valores às variaveis:
      str='a';
      N1=567;
      x=789.564332;
      xx=8912.7815533613;

      //Imprimindo os valores de cada variável
      printf("str=%c\n",str); //para imprimir caracter usamos %c
      printf("valor de x:%d\n",N1); //para imprimir inteiro %d
      printf("x=%f e xx=%lf\n",x,xx); /*para imprimir float e double utilizamos
                                        %f e %lf respectivamente.*/
}
```

Suporte para abstração

Simplicidade e ortogonalidade

- Se uma linguagem tem uma **grande número de construções**, alguns programadores não conhecerão todas elas.
- Logo, é muito melhor ter um **número menor de construções** primitivas e um conjunto de regras consistente para combiná-las (isso é, ortogonalidade) do que muitas construções primitivas.

Expressividade

- Em uma linguagem de programação pode ser referir a diversas características.
- Expressividade significa a existência de operadores muito poderosos que permitem muitas computações com um programa muito pequeno.
- Em geral, uma linguagem expressiva especifica computações de uma forma conveniente, em vez de **deselegante.**

Confiabilidade

- Grau de fidelidade de uma informação em relação ao original.
- Diz-se que um programa é confiável quando está de acordo com suas especificações em todas as condições.

Verificação de tipos

- É a execução de testes para detectar erros de tipos em um programa, tanto por parte do compilador quanto durante a execução de um programa.

Tratamento de exceções

- A habilidade de um programa de interceptar erros em tempo de execução.

Apelidos

- São utilizados quando é possível ter um ou mais nomes em um programa para acessar a mesma célula de memória.

Legibilidade e facilidade de escrita

- Tanto a legibilidade quanto a facilidade de escrita influencia a confiabilidade.
- Um programa escrito em uma linguagem que não suporta maneira naturais de expressar os algoritmos exigidos necessariamente usará estratégias artificiais.
- É menos provável que estratégias artificiais estejam corretas para todas as situações possíveis.
- **Quanto mais fácil é escrever um programa, maior a probabilidade de ele estar correto.**

Custo

- Existe o custo de treinar programadores para usar a linguagem.
- Há o custo de escrever programas na linguagem.
- Há o custo de compilar programas na linguagem
- O custo de executar programas escritos em uma linguagem.
- É o custo de sistema de implementação da linguagem.
- É o custo de uma confiabilidade baixa.
- É o custo da manutenção de programas, qual inclui tanto as correções quanto as modificação para adicionar funcionalidade.

Influências no projeto de linguagens

- Arquitetura de computadores
- Metodologias de projeto de programas
  - Programação estruturada
  - Programação orientada a objeto
  - Programação orientada a procedimento
    - baseada nas chamadas de procedimentos/rotinas, a aplicação em si inicia na primeira linha do código e segue um fluxo determinado pelo próprio programa, durante a sua execução.

Categoria de linguagens

- Imperativas
  - é um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado de um programa.
- Funcionais
  - é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis.
- Lógicas
  - é um paradigma de programação que faz uso da lógica matemática.
- Orientada a objeto
  - é um paradigma de programação baseado no conceito de "objetos", que podem conter dados na forma de campos, também conhecidos como atributos, e códigos, na forma de procedimentos, também conhecidos como métodos.

Categoria de linguagens

- Alguns autores se referem às linguagens de *scripting* como uma categoria separada de linguagem de programação.
- Entretanto, linguagens nessa categoria são mais unidas entre si por seu método de implementação, interpretação parcial ou completa, do que por um projeto de linguagem comum. Exemplo: Perl, JavaScript e Ruby.

Categoria de linguagens

- Nos últimos anos surgiu uma nova categoria: as linguagens de marcação.
- Linguagens de marcação não são de programação. Por exemplo, HTML, XML JSLT.

Trade-off no projeto de linguagens

- O ato de escolher uma coisa em detrimento de outra e muitas vezes é traduzida como “perde-e-ganha”.
- Infelizmente esse modelo é contraditório.
- Dois critérios conflitantes são a **confiabilidade e o custo de execução.**

Métodos de implementação

- MONTADOR (*assembler*)
  - Tradutor para linguagens de 2ª geração.
- COMPILADOR:
  - Traduz todo o programa de uma vez.
- INTERPRETADOR:
  - Traduz o programa instrução por instrução.

Compiladores X Intepretadores

- Os termos **compiladores e interpretadores** referem-se à maneira como um programa é executado.
- A maneira pela qual uma programa é executado não é definido pela linguagem em que ele é escrita.
- **Interpretadores e compiladores** são simplesmente programas sofisticados que operam sobre o código-fonte do seu programa.

Compiladores

- Um compilador lê o programa inteiro e **converte-o** em um **código-objeto**, que é uma tradução do **código-fonte** de um programa em uma forma que o computador possa executar diretamente.

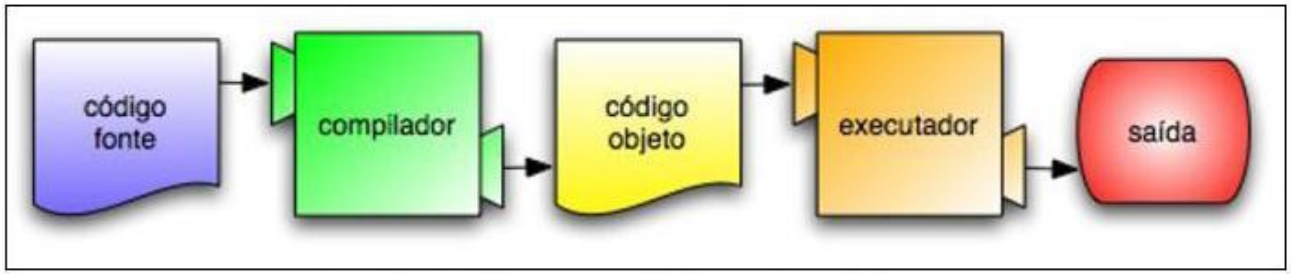


Figura 01 – Processo simprificado de compilação

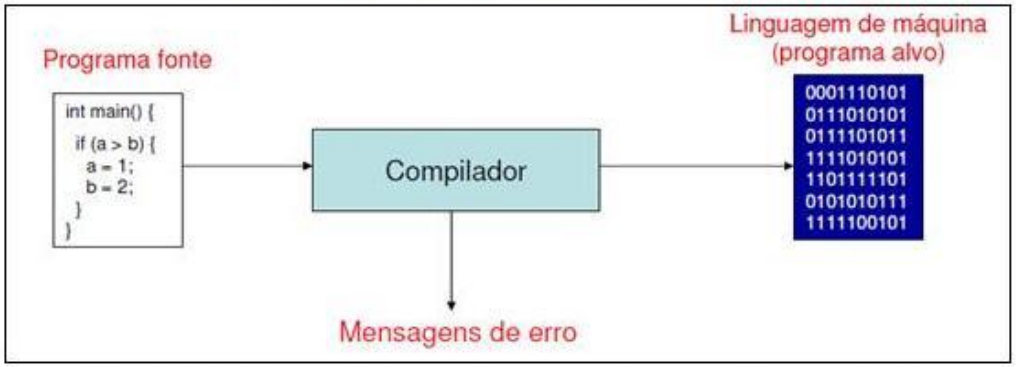


Figura 02 – Exemplo de compilação – programa na linguagem C

Estácio

Interpretadores

- Um Interpretador lê o **código–fonte** do seu programa uma linha por vez, executando a **instrução específica contida nessa linha.**

Figura 03 – processo simplificado de interpretação

Interpretação pura

- A interpretação pura reside na extremidade oposta (em relação à compilação) dos métodos de implementação.
- Os programas são interpretados por outros, chamados de interpretador, sem tradução.
- O interpretador age como uma simulação em software de uma máquina cujo o ciclo de obtenção-execução trata de sentença de programa de alto nível em vez de instruções de máquina.

Sistema de implementação Híbridos

- São um meio-termo entre os compiladores e os interpretadores puros, eles traduzem os programas em linguagem de alto nível para uma linguagem de intermediária projetada para facilitar a interpretação;
- Esse método é mais rápido que a interpretação pura, porque as sentenças da linguagem-fonte são decodificadas apenas uma vez.

Pré-processadores

- É um programa que processa outro programa imediatamente antes de ele ser compilado.
- As instruções de pré-processador são embutidas em programas.
- O pré-processador é essencialmente um programa que expande macros.
- As instruções de pré-processador são comumente usadas para especificar que p código de outro arquivo deve ser incluído.
- Exemplo: # include “myLib.h”

Obs: o faz copiar o conteúdo MyLib.h no programa, na posição da instrução do #include.

Ambientes de programação


OBRIGADO!
EDUCAR PARA
TRANSFORMAR
Estácio