

# Linguagens de Programação

## Aula 1

### Classificação, evolução e paradigmas

# O que é uma LP?

- Linguagem destinada ao uso por pessoas para expressar um processo pelo qual um computador pode resolver um problema
- Modelos/paradigmas de programação
  - **Imperativo, orientado a objetos, funcional e lógico**
  - Correspondem a diferentes pontos de vista pelos quais os processos podem ser expressos

# Para quê serve uma LP?

- Serve como meio de comunicação
  - Entre a pessoa que deseja resolver um problema e o computador
- Permite abstração e representação
  - Faz a ligação entre o pensamento humano (por vezes, não estruturado) e o modelo que o computador processa (estruturado e restrito)
- Auxilia no processo de desenvolvimento de software
  - Análise de requisitos
  - Projeto
  - Implementação
  - Testes
  - Manutenção

# Motivação

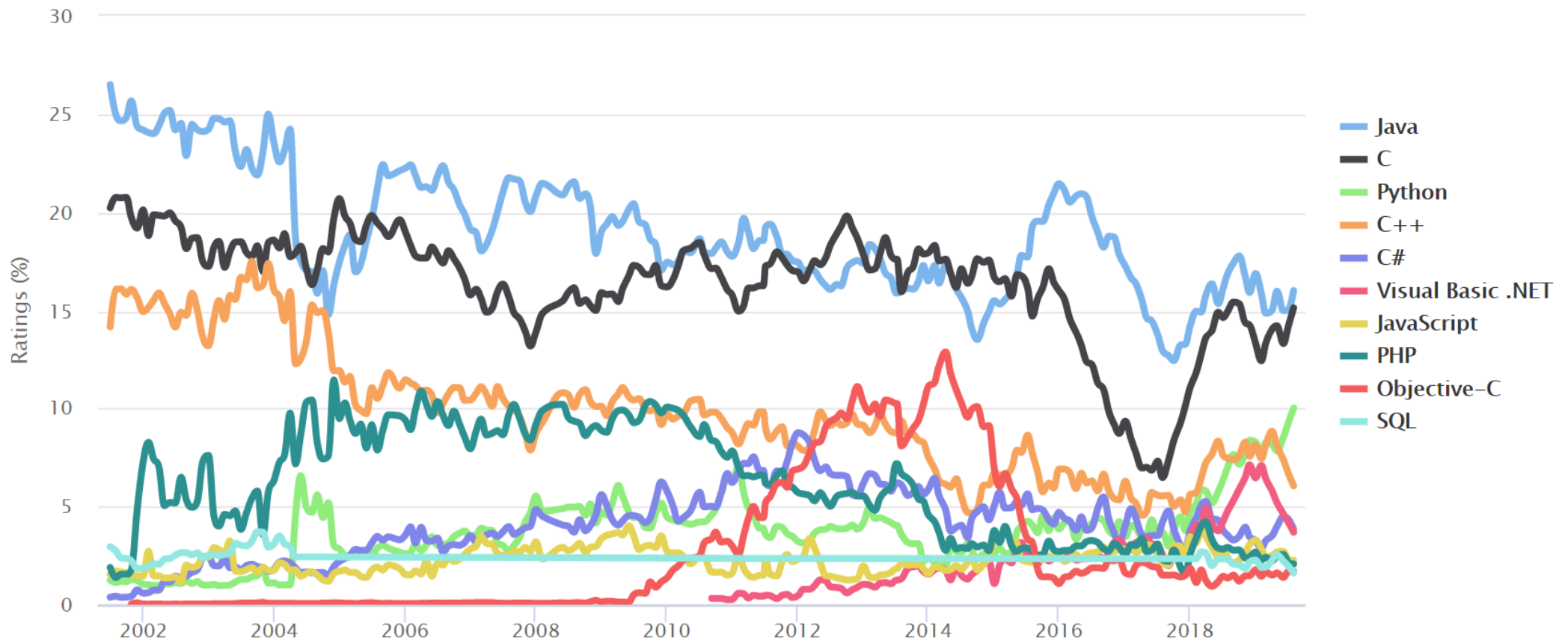
- Por que estudar LP?
  - Aumento da capacidade de expressar ideias
  - Subsídio para escolha de linguagens apropriadas
  - Melhorar a habilidade de aprender linguagens
  - Entender melhor a importância da implementação
  - Capacitar do projeto de novas linguagens
  - Entender o processo básico de compilação
  - Acompanhar o avanço da Computação

# TIOBE Index – ago/2018 a ago/2019

Aug 2019	Aug 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.028%	-0.85%
2	2		C	15.154%	+0.19%
3	4	⬆	Python	10.020%	+3.03%
4	3	⬇	C++	6.057%	-1.41%
5	6	⬆	C#	3.842%	+0.30%
6	5	⬇	Visual Basic .NET	3.695%	-1.07%
7	8	⬆	JavaScript	2.258%	-0.15%
8	7	⬇	PHP	2.075%	-0.85%
9	14	⬆	Objective-C	1.690%	+0.33%
10	9	⬇	SQL	1.625%	-0.69%
11	15	⬆	Ruby	1.316%	+0.13%
12	13	⬆	MATLAB	1.274%	-0.09%
13	44	⬆	Groovy	1.225%	+1.04%
14	12	⬇	Delphi/Object Pascal	1.194%	-0.18%
15	10	⬇	Assembly language	1.114%	-0.30%
16	19	⬆	Visual Basic	1.025%	+0.10%
17	17		Go	0.973%	-0.02%
18	11	⬇	Swift	0.890%	-0.49%
19	16	⬇	Perl	0.860%	-0.31%
20	18	⬇	R	0.822%	-0.14%

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

# TIOBE Programming Community Index



# Áreas de aplicação






# Áreas atuais de aplicação

**CNN BUSINESS** Markets Tech Media Success Perspectives Video

Exclusive: GM CEO Mary Barra says the world needs more coders

by Peter Valdes-Dapena @peterdrives

June 28, 2017, 10:03 AM ET



GM CEO: We need more diverse engineers

GM CEO: We need more diverse engineers

A lot of resources go into building cars, like steel, aluminum, rubber and glass. Also critical: Brains.

But the engineering talent and computer coding skills that the industry needs is in short supply. That's why General Motors CEO Mary Barra announced a new push to train engineers on Wednesday in New York City. The effort is specifically aimed at recruiting women and minorities.


GM has long helped train engineers. Barra earned a degree in electrical engineering from General Motors Institute in Flint, Michigan -- now Kettering University -- an engineering and business school that was, at the time, operated by GM.

"A car today has hundreds of millions of lines of code," Barra said in an exclusive interview with CNN. "We do see a shortage if we don't address this and I mean fully fundamentally. Every child needs to have these skills."


**BLX**  
BITCOIN  
LIQUID INDEX  
SEE IT IN ACTION  
TRUSTED BY:  
Nasdaq  
amazon alexa  
BNC.  
bravenewcoin.com

Advertisement


Paid Contentby Outbrain




Where Do The Richest Americans Live?  
Mansion Global



Who Are The World's Richest Investors?  
Trendingstock Today




A Browser Designed for Privacy and Speed (Not Ads and Trackers)  
Browser Guides



This Brilliant Device Lets You Communicate In 40+...  
Next Tech

More from CNNMoney



FULL SHOW 8/7/2019: 'Investors lack perspective,' says BMO's...



# Aplicações científicas

- Primeiros computadores – década de 1940
  - Projetados para aplicações científicas
    - Grande volume de processamento
    - Operações com ponto flutuante
    - Poucas exigências de entrada e saída
  - Preocupação básica é com a eficiência
  - Surgimento de linguagens de alto nível para esse fim
    - Ex.: Fortran

# Exemplo de Fortran

```
PROGRAM teste
    DIMENSION A(10)
    DO 15 I = 1, 10
        A(I) = I*5
15  CONTINUE
    DO 20 J = 10, 1, -1
        WRITE (*,*) A(J)
20  CONTINUE
    STOP
END
```

Compilador online: [http://www.tutorialspoint.com/compile\\_fortran\\_online.php](http://www.tutorialspoint.com/compile_fortran_online.php)

# Aplicações comerciais

- Início na década de 1950
  - Necessidade de elaborar relatórios
  - Interação com armazenamento de dados em arquivos
  - Representar e armazenar números decimais
  - Representar e armazenar caracteres (letras)
- Primeira linguagem amplamente adotada
  - COBOL (1960)

# Exemplo de Cobol

```
IDENTIFICATION DIVISION.
```

```
PROGRAM-ID. Teste.
```

```
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
```

```
01 Num1 PIC 9.
```

```
01 Num2 PIC 9.
```

```
01 Result PIC 99.
```

```
01 Operator PIC X.
```

```
PROCEDURE DIVISION.
```

```
Calcula.
```

```
    PERFORM 3 TIMES
```

```
        DISPLAY "Numero 1: "
```

```
        ACCEPT Num1
```

```
        DISPLAY "Numero 2: "
```

```
        ACCEPT Num2
```

```
        DISPLAY "Operador (+ ou *): "
```

```
        ACCEPT Operator
```

```
        IF Operator = "+" THEN
```

```
            ADD Num1, Num2 GIVING Result
```

```
        END-IF
```

```
        IF Operator = "*" THEN
```

```
            MULTIPLY Num1 BY Num2 GIVING Result
```

```
        END-IF
```

```
        DISPLAY "Result is = ", Result
```

```
    END-PERFORM.
```

```
STOP RUN.
```

Compilador online:

[https://www.tutorialspoint.com/compile\\_cobol\\_online.php](https://www.tutorialspoint.com/compile_cobol_online.php)

# Aplicações de Inteligência Artificial

- Início na década de 1950
  - Utilização de computação simbólica em vez de numérica
    - Manipulação de nomes no lugar de números
  - A primeira linguagem para IA foi a funcional LISP (1959)
- Primeira linguagem de programação lógica
  - Prolog (início de 1970)

# Aplicação em Sistemas Operacionais

- Principal objetivo de um SO
  - Oferecer suporte à execução de outros programas e aplicativos
  - Fornecer uma abstração de alto nível e funcional do computador (máquina virtual)
- Linguagens de programação para construção de SO
  - Execução rápida (bom desempenho)
  - Recursos de baixo nível para interface aos recursos do sistema e dispositivos externos
- O SO Unix foi desenvolvido quase todo em linguagem C
  - Facilitou sua portabilidade para diferentes máquinas



# Exemplo em C

```
#include <stdio.h>
int main() {
    int a[10], i;
    for(i = 0; i < 10; i++) {
        a[i] = (i + 1) * 5;
    }
    for(i = 9; i >= 0; i--) {
        printf("%d\n", a[i]);
    }
    return 0;
}
```

Compilador online: <https://code.dcoder.tech/>

# O que faz o sucesso de uma LP?

- De acordo com Michael Scott
  - Facilidade de aprender (BASIC, Pascal, Python)
  - Facilidade de expressar coisas avançadas (C, Perl)
  - Facilidade de implementar (BASIC, Python)
  - Possibilidade de compilar um código muito bom, rápido, pequeno (Fortran)
  - Suporte de um fabricante (Visual Basic, COBOL)
  - Ampla disseminação com menor custo (Pascal, Java)

# Características chaves de uma LP

- De acordo com Robert Sebesta
  - Simplicidade: fácil de expressar e de ler
  - Ortogonalidade: poucas instruções e muitas combinações
  - Tipos de dados: representar diferentes valores (números, letras, conjuntos, etc)
  - Projeto de sintaxe: formatos de expressões e comandos
  - Suporte para abstração: divisão do problema complexo em partes, escondendo detalhes (subprogramação)
  - Expressividade: pequenas ou poucas operações permitem expressar computações de forma elegante
  - Verificação de tipos: previne erros de conversão e atribuição
  - Tratamento de exceções: recursos para comunicar/tratar erros
  - Apelidos restritos: critérios para uso de referências/ponteiros

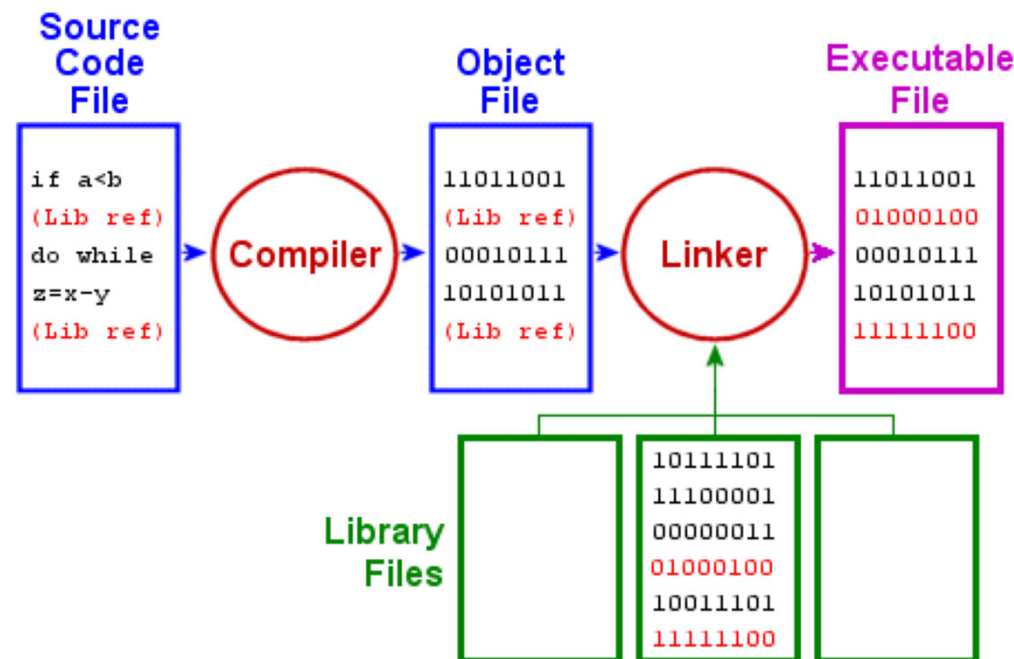
# Características x Critérios

**TABELA 1.1** Critérios de avaliação de linguagens e as características que os afetam

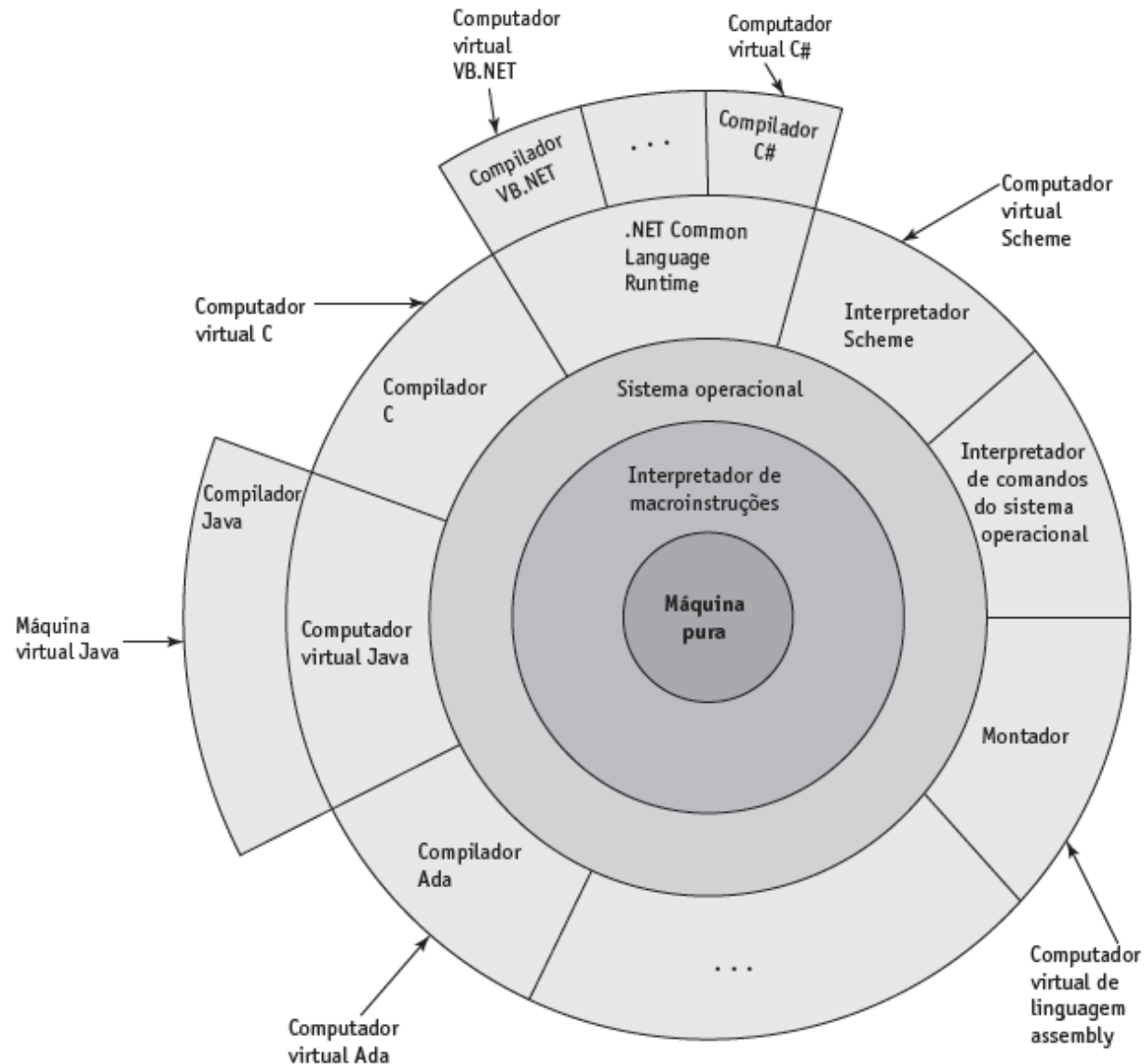
Característica	CRITÉRIOS		
	Legibilidade	Facilidade de escrita	Confiabilidade
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•

# Compilação de programas

- Conversão de um código escrito em linguagem de alto nível para linguagem de máquina
- Realizada por compiladores ou interpretadores
  - Recebem representação textual (código fonte)
  - Produzem código executável (linguagem de máquina)



# Computador: suporte a programas



**FIGURA 1.2**

Interface em camadas de computadores virtuais, fornecida por um sistema de computação típico.



# Classificação de linguagens

- Em relação ao nível
  - Baixo, médio ou alto
- Em relação à geração
  - 1ª geração, 2ª geração, 3ª geração, 4ª geração, 5ª geração
- Em relação ao paradigma
  - Imperativo, Orientado a Objetos, Funcional ou Lógico

# Linguagens de baixo nível

- Voltadas para a máquina e descrevem instruções de um microprocessador específico
  - Linguagens de montagem, como Assembly
- Vantagens
  - Programas são executados com maior velocidade de processamento
  - Programas ocupam menos espaço na memória
- Desvantagens
  - Pouca portabilidade, pois programas são desenvolvidos para instruções de uma plataforma específica
  - Código geralmente não estruturado, o que torna mais difícil a programação

# Exemplo de Assembly

```
section          .text
                global _start
_start:          ;ponto de entrada
                mov     edx, len      ;tamanho da mensagem
                mov     ecx, msg      ;mensagem a escrever
                mov     ebx, 1        ;descriptor de arquivo (stdout)
                mov     eax, 4        ;system call (sys_write)
                int     0x80          ;chamada ao kernel
                mov     eax, 1        ;system call (sys_exit)
                int     0x80          ;chamada ao kernel

section          .data

msg             db      'Hello, world!', 0xa ;mensagem
len             equ     $ - msg             ;tamanho da mensagem
```

Compilador online: [http://www.tutorialspoint.com/compile\\_assembly\\_online.php](http://www.tutorialspoint.com/compile_assembly_online.php)

# Linguagens de nível médio

- Linguagens voltadas tanto ao ser humano, quanto à máquina
  - Mistura de linguagens de alto e baixo nível
  - Contêm instruções simples e outras mais complicadas, podendo tornar a programação mais difícil e poderosa
- Exemplo: Linguagem C
  - Permite acessar registros de sistema e operar endereços de memória (baixo nível)
  - Oferece operações estruturadas como if..else, while e for (alto nível)
- Vantagens
  - Linguagens poderosas para ampla variedade de aplicações, desde jogos a programas de alto desempenho
- Desvantagens
  - Possuem instruções às vezes complicadas de se entender

# Linguagens de alto nível

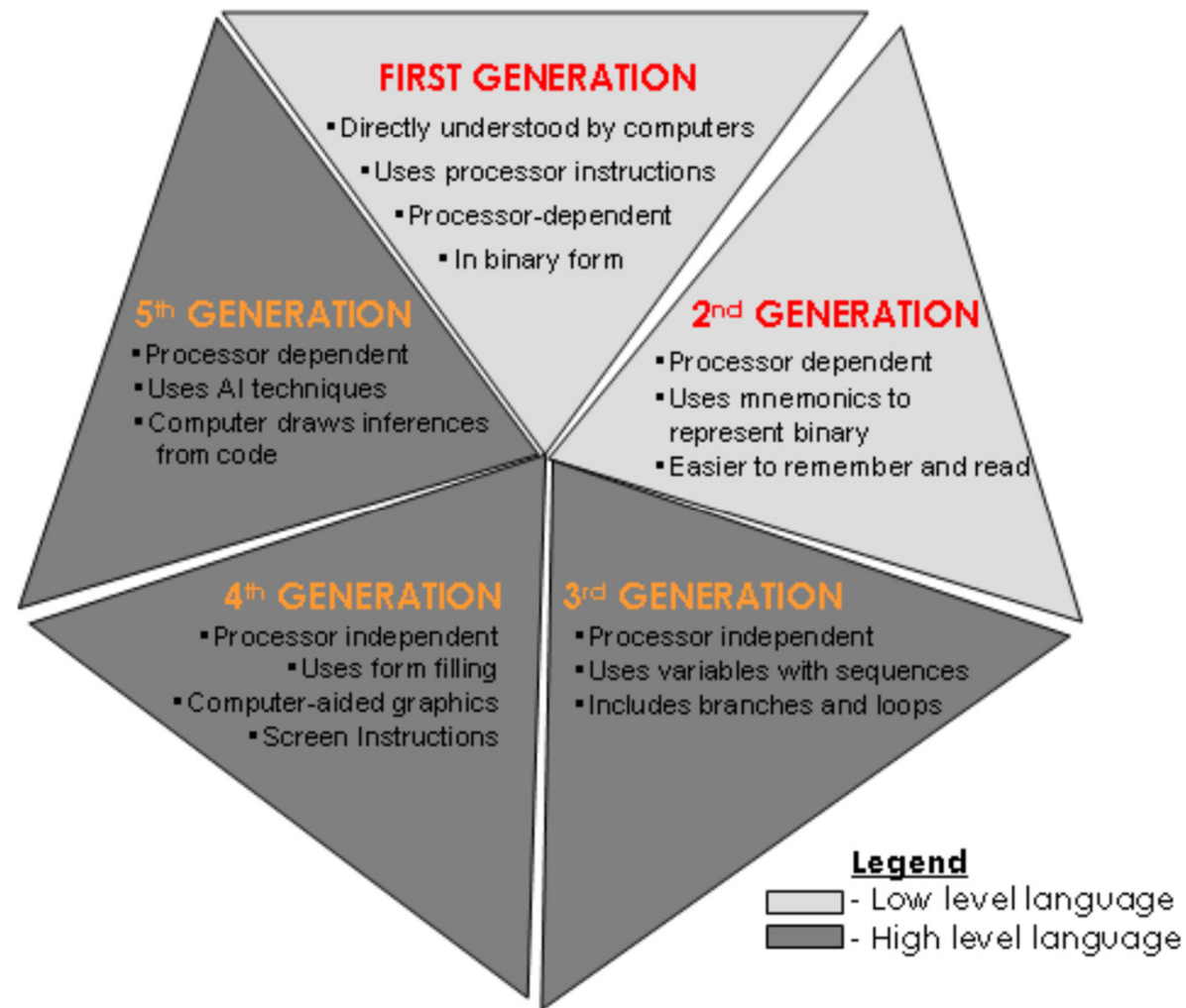
- Linguagens mais voltadas ao ser humano
  - Código mais **fácil** de entender
- Independência/transparência de plataforma
  - Programas facilmente **portados** para várias plataformas
  - Programador não precisa conhecer funcionamento da máquina
  - Podem usar um tradutor (máquina virtual, compilador ou interpretador) para portar entre plataformas
- Desvantagem
  - Podem ser mais lentas que as de plataforma nativa
- Exemplos: Java, C#, Python

# Exemplo de Python 3

```
a = [None] * 10
for i in range(len(a)):
    a[i] = (i + 1) * 5
print(a)
```



# Gerações de linguagens



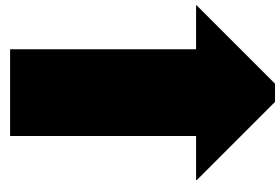
# LP de 1ª geração

- Linguagens em nível de máquina
  - Programação direta em binário
  - Permitem comandar diretamente o processador com operações primitivas, descritas em bits
  - Extremamente complexas, cansativas e sujeitas a erros de programação
  - Dificuldade alta para depuração de um programa
- Cada instrução
  - Código de operação (opcode) seguido de um ou dois endereços de registradores ou de memória
- Exemplo: 0010 0001 0110 1100
  - Soma (opcode 0010) do dado armazenado no registrador 0001 com o dado armazenado na posição da memória 108 (0110 1100)

# LP de 2ª geração

- Linguagens simbólicas de montagem
  - Símbolos no lugar de bits
  - Exemplo: 0010 0001 0110 1100
    - Ficaria algo como: `add r1 total`
    - Onde: `r1` é o primeiro registrador  
`total` é o nome atribuído ao endereço 108 da memória
  - Opcodes e endereços substituídos por mnemônicos  
`mov`    `mul`    `add`    `cmp`    `jmp`
  - Exemplo: trecho C para Assembly

```
if (a == b) {  
    c = d;  
}  
d = a + c;
```



```
_start:  
    cmp eax, ebx  
    jne .L7  
    mov edx, ecx  
.L7:  
    mov eax, edx  
    add ecx, edx
```

# LP de 3ª geração

- Linguagens orientadas ao usuário (imperativas)
  - Década de 1960 (BASIC, PASCAL, ADA, C, PL/I, entre outras)
  - Orientadas a solução de problemas científicos ou comerciais
  - Instruções de alto nível em três classes:
    - Entrada e saída, cálculos aritméticos ou lógicos, desvios (in)condicionais
  - Exemplos: BASIC e PASCAL:

```
VAR number = 8
IF number < 0 THEN
    PRINT "Number is negative"
ELSEIF number > 0 THEN
    PRINT "Number is positive"
ELSE
    PRINT "Number is zero"
END IF
```

```
program teste;
var
    i : byte;
begin
    writeln('Digite um numero = ');
    readln(i);
    if i <= 10 then
        writeln('É menor ou igual a 10!')
    else
        writeln('É maior que 10!');
    end.
end.
```

Compiladores online: [http://www.tutorialspoint.com/compile\\_freebasic\\_online.php](http://www.tutorialspoint.com/compile_freebasic_online.php)  
[http://www.tutorialspoint.com/compile\\_pascal\\_online.php](http://www.tutorialspoint.com/compile_pascal_online.php)

# LP de 3ª geração

- Até aqui, foram apresentadas as linguagens **imperativas**
- Nesta geração surgiram também outros modelos
  - **Funcionais** (ou aplicativas): que expressam funções matemáticas e ordenam/sequenciam operações por recursões e expressões condicionais (exs.: LISP, HASKELL)
  - **Lógicas** (ou declarativas): operam proposições e cálculo de predicados (ex.: Prolog)
- As linguagens da 3ª geração necessitam de um tradutor
  - Compilador ou interpretador

# LP de 4ª geração

- Linguagens orientadas à aplicação
  - Foram projetadas para usuários finais e para
    - Facilitar programação de computadores
    - Agilizar desenvolvimento de aplicações
    - Reduzir custo de manutenção de software
    - Minimizar problemas com depuração
    - Gerar códigos sem erros à partir de requisitos de negócio
  - Atendem fins específicos (geração de relatórios ou pacotes gráficos) e, até, geram aplicações completas (exs.: Matlab, SQL, Scilab)
  - Demandam menos linhas de código que em LP de 3ª geração
  - Exemplo: ler e exibir uma imagem em Matlab

```
A = imread('image.jpg');  
image(A);
```



# LP de 5ª geração

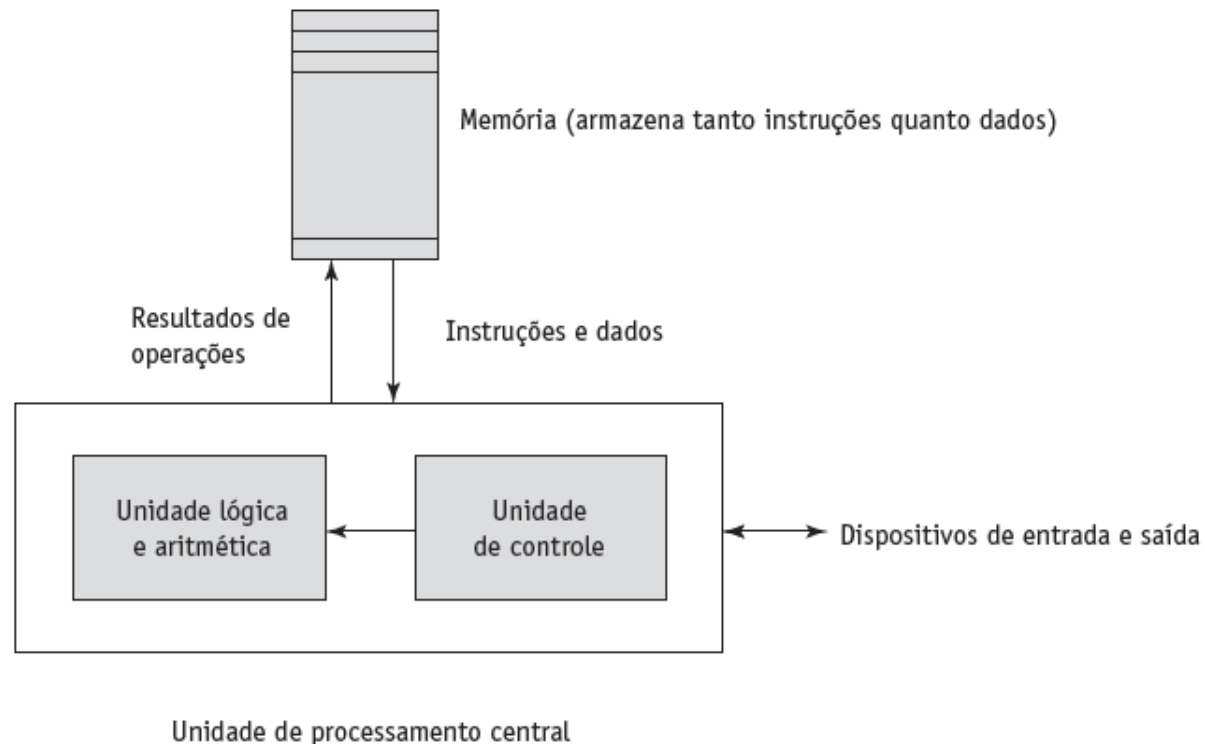
- Linguagens do conhecimento
  - Aplicadas à resolução de problemas a partir de restrições dadas
  - Usadas em áreas de IA e Engenharia Semântica
  - Foram introduzidas na década de 1980
  - Linguagens de programação lógica, como Prolog, foram utilizadas para esse fim
  - Hoje, há uma linha crescente de novos projetos que incluem geradores automáticos de código e algoritmos adaptativos

# Paradigma imperativo

- Orientado a ações
  - Computação é vista como sequência de instruções que manipulam valores de variável (leitura e atribuição)
  - Programas exploram conceitos de estados (de variáveis) e ações (comandos) que os manipulam (exs.: FORTRAN, PASCAL, C, COBOL)
  - Também chamado de procedural (sub-rotinas ou procedimentos)
  - Suporta modelos estruturado e não-estruturado
    - Estruturado: instruções agrupadas em blocos, desvios condicionais e funções organizam o fluxo de execução
    - Não-estruturado: permite desvios incondicionais com operações do tipo JMP (Assembly) e GOTO (BASIC)
  - Vantagens
    - Eficiência, modelo de aplicações próximo ao natural, paradigma dominante e estabelecido
  - Desvantagens
    - Difícil legibilidade e induz ao erro; foco no “como” e não no “quê”; mistura comportamento com tratamento de dados

# Paradigma imperativo

- Segue arquitetura de Von Neumann
  - Instruções e dados na mesma memória
  - São transmitidos da CPU para a memória e vice-versa
  - Resultados são retornados à memória



# Paradigma funcional

- Computação vista como avaliação de funções matemáticas
  - Foco na aplicação de funções na solução de um problema
    - Definir qual função será aplicada para transformar a entrada na saída
  - Características da programação funcional
    - Programas são funções que descrevem uma relação explícita e precisa entre E/S
    - Estilo declarativo: não há o conceito de estado nem comandos como atribuição
  - Exemplos de linguagens: Haskell, Scheme, Common LISP, ML
  - Vantagens
    - Simplificar resolução de classes de problemas, otimização de programas
  - Desvantagens
    - Mundo não é funcional, implementações podem ser ineficientes, poucos recursos para E/S

# Paradigma funcional

- Exemplo: distância entre dois pontos em C e em Haskell

```
#include <stdio.h>
#include <math.h>
float dist(float PX1, float PY1, float PX2, float PY2) {
    float res = sqrt((PX2 - PX1)*(PX2 - PX1) + (PY2 - PY1)*(PY2 - PY1));
    return res;
}
int main() {
    float f = dist(2.0, 4.0, 3.0, 1.0);
    printf("Distance = %f", f);
    return 0;
}
```

```
dist x1 y1 x2 y2 = sqrt(((x2 - x1)^2) + ((y2 - y1)^2))
main = print(dist 2.0 4.0 3.0 1.0)
```

# Paradigma lógico

- Programação baseada em lógica formal
  - Os programas
    - Consistem em declarar fatos
    - Que podem ser relações (associações)
    - Ou regras, que produzem fatos deduzidos de outros
  - Requer estilo mais descritivo
    - Programador deve conhecer relacionamentos entre entidades e conceitos envolvidos para expressar os fatos relacionados ao problema
    - Programas descrevem conjuntos de regras, disparadas quando suas premissas são satisfeitas
  - Principal linguagem: Prolog
  - Vantagens
    - Alto nível de abstração, LP mais próxima do raciocínio lógico
  - Desvantagens
    - Dificuldade para expressar algoritmos complexos

# Paradigma lógico

- Exemplo em Prolog

```
pai(fred, marcos).  
pai(ricardo, pedro).  
pai(pedro, paulo).  
avo(X,Y) :- pai(X, Z), pai(Z, Y), write(X), write(' eh avo  
de '), write(Y).
```

- Execução

```
?- avo(ricardo, paulo).
```

Compilador online: <https://swish.swi-prolog.org/>

# Paradigma orientado a objetos

- Problema é analisado quanto aos objetos que o compõem
  - Sistemas compostos por conjuntos de objetos que se comunicam por mensagens e juntos constroem o comportamento global
  - Objetos são entidades abstratas que contêm propriedades (atributos) e comportamentos que descrevem uma entidade real
    - Diminuição da distância entre modelagem computacional e mundo real
    - Sobre eles descrevemos nomes, características e os classificamos em grupos
  - Aplicação estruturada em módulos ou classes
    - Modelos para construir objetos com atributos (estado) e métodos (comportamentos)
    - Podem herdar uma superclasse, expandi-la e especializá-la
  - Linguagens: SIMULA 67, Smaltalk, C++, C#, Java
  - Vantagens
    - Organização e reutilização do código, grande uso comercial
  - Desvantagens
    - Menos eficientes



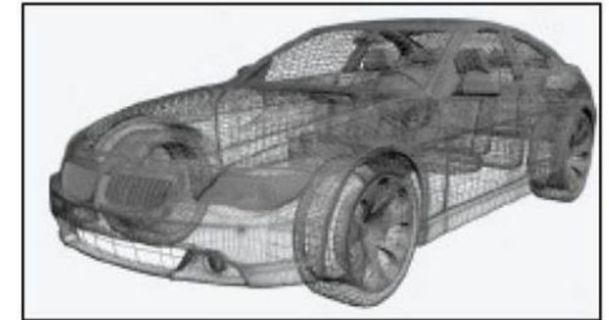
# Paradigma orientado a objetos

- Exemplo em Java

```
public class Carro {  
    // Atributos  
    private String marca;  
    private String cor;  
    private String placa;  
    private int portas;  
    private int marcha;  
    private double velocidade;  
    //Métodos  
    public void acelerar() {  
        velocidade += marcha * 10;  
    }  
    public void frear() {  
        velocidade -= marcha * 10;  
    }  
}
```

Carro
- marca : String - cor : String - placa : String - portas : int - marcha : int - velocidade : double
+ acelerar() : void + frear() : void

Classe

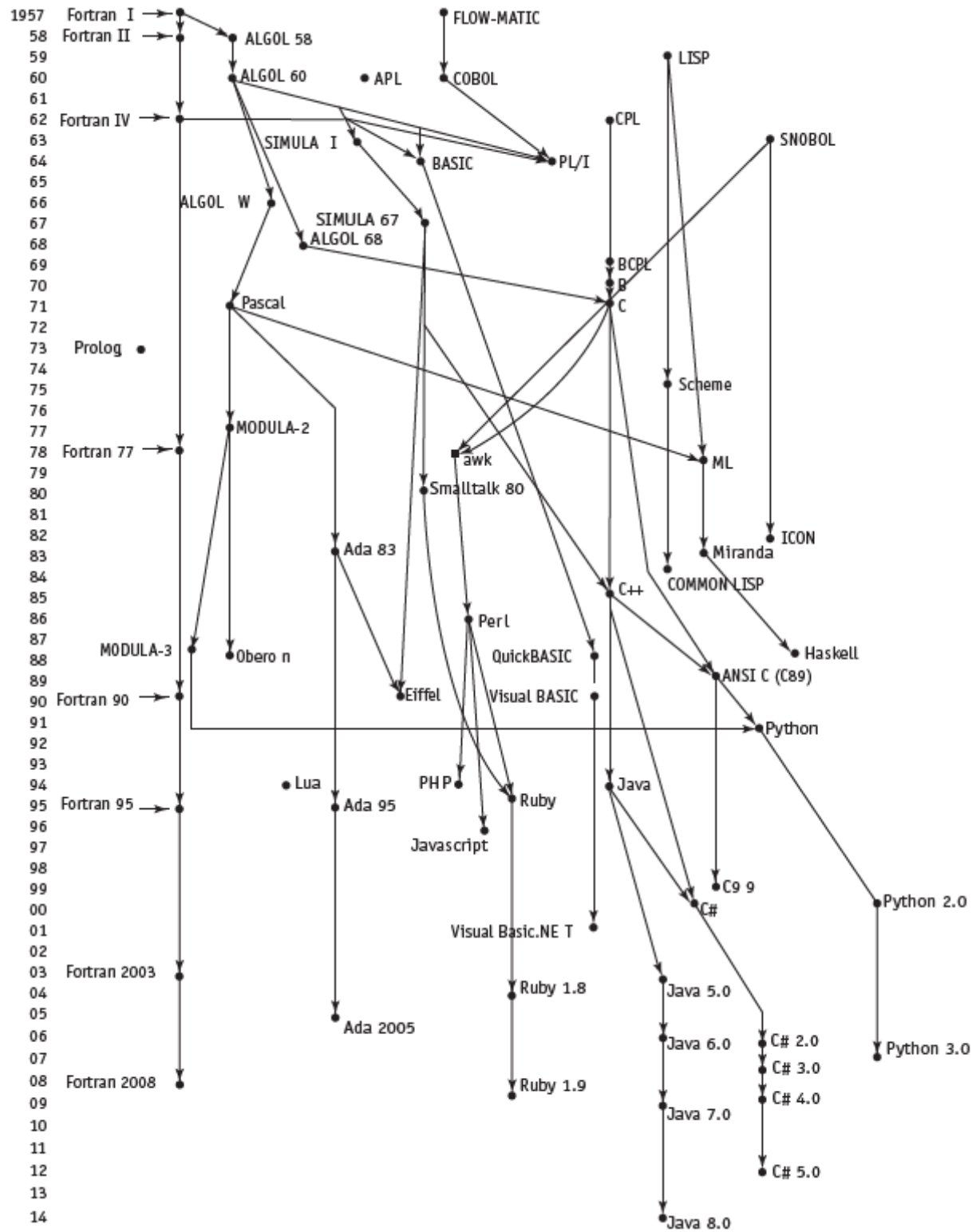


I  
n  
s  
t  
â  
n  
c  
i  
a

Objeto



# Evolução das principais LPs



# Bibliografia

- SEBESTA, R. W. Conceitos de Linguagens de Programação. Porto Alegre: Bookman, 2011. 792p.