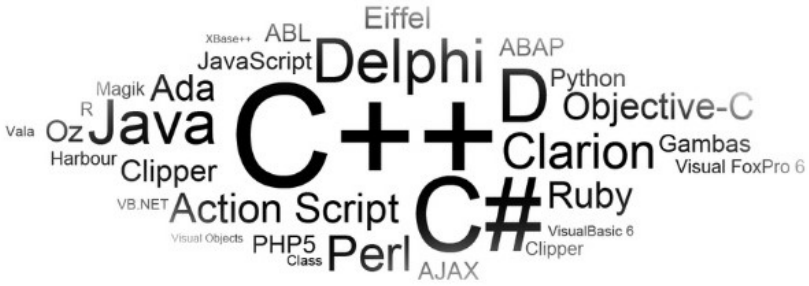
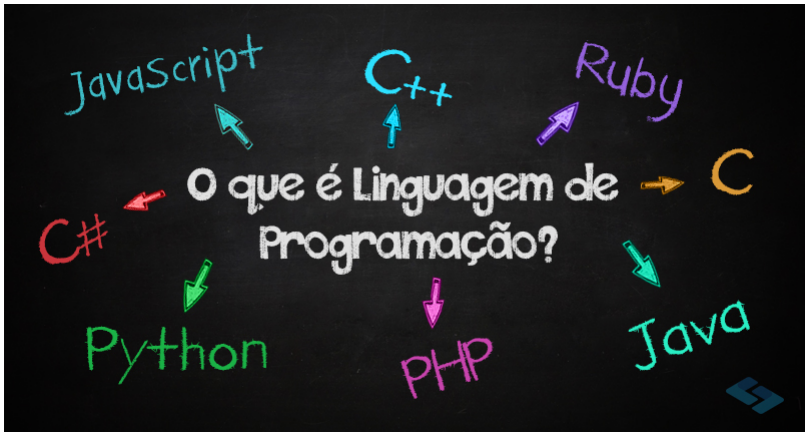
CAPÍTULO I
Introdução



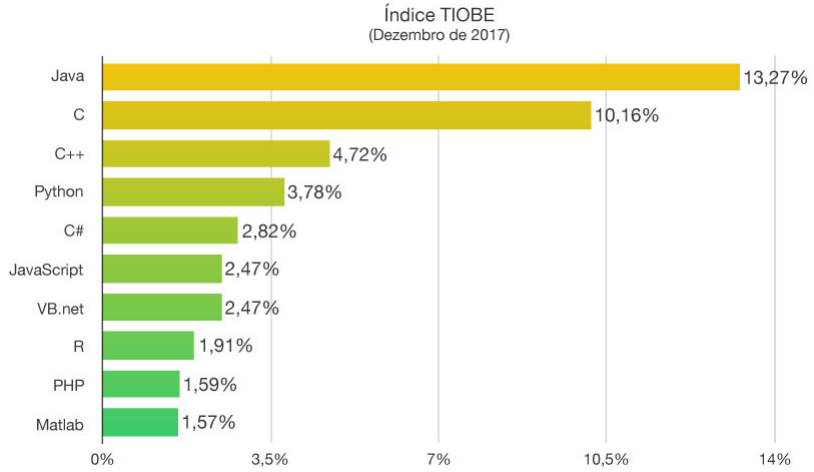
Introdução



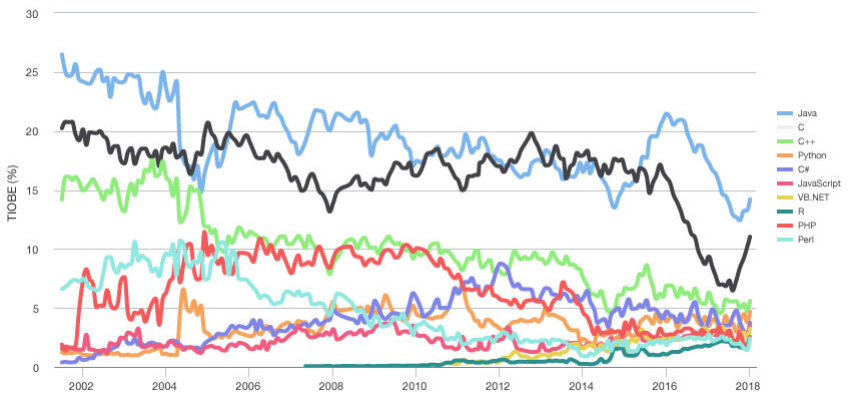
Por que estudar os conceitos de LP?

- **Melhora a compreensão das linguagens já conhecidas**
- **Facilita o aprendizado de novas linguagens**
- **Facilita o projeto de uma linguagem**
- **Facilita a escolha de uma linguagem para um projeto**

Por que estudar os conceitos de LP?



Por que estudar os conceitos de LP?



Análise de tendências:histórico do índice TIOBE

Requisitos de uma LP: Legibilidade

- **Simplicidade**

  Contra exemplos:

```
a = a+1;
a += 1;
a++;
++a;
```

Qual usar?

Requisitos de uma LP: Legibilidade

- **Simplicidade**

  Contra exemplos:

```
*p = (*p)*q;
```

Acessa o conteúdo da célula de memória apontada por p

Operador multiplicação

Requisitos de uma LP: Legibilidade

- **Ortogonalidade:**
  - Capacidade da LP permitir ao programador combinar seus conceitos básicos sem que se produzam exceções.
  - o programador pode prever o comportamento de combinações de conceitos.
  - A falta de ortogonalidade pode estimular a ocorrência de erros de programação.

Requisitos de uma LP: Legibilidade

- **Ortogonalidade:**
  - **Pascal**: funções podem retornar qualquer tipo de dados, exceto registros e vetores.
  - **C**: Registros podem ser retornados de funções, arrays não.

Requisitos de uma LP: Legibilidade

- **Instruções claras de controle**

Requisitos de uma LP: Legibilidade

- **Instruções claras de controle**

```
while: if x >= 10 goto end
    print x
    x++
    goto while
end:
```

```
while x < 10
    print x
    x++
end
```

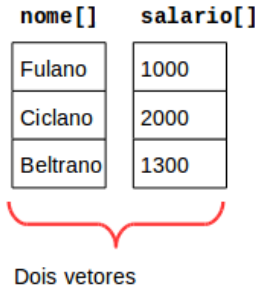
Requisitos de uma LP: Legibilidade

- **Tipos de Dados e Estruturas adequados**

```
import java.util.Scanner;

public class AppVetor {
    public static void main (String[] args) {
        String[] nome = new String[3];
        double[] salario = new double[3];

        // ler os dados
        for (int i=0; i<nome.length; i++) {
            System.out.println("NOME");
            nome[i] = new Scanner(System.in).nextLine();
            System.out.println("SALARIO");
            salario[i] = new Scanner(System.in).nextDouble();
        }

        //imprimir na tela
        for (int i=0; i<nome.length; i++) {
            System.out.println("NOME: "+ nome[i] +"    SALARIO: R$ "+salario[i]);
        }

    }
}
```

Requisitos de uma LP: Legibilidade

- **Tipos de Dados e Estruturas adequados**



Requisitos de uma LP: Legibilidade

- **Tipos de Dados e Estruturas adequados**

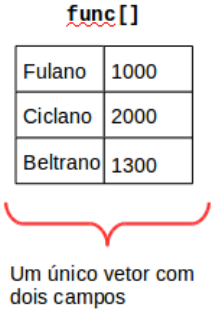
```
public class Funcionario {
    private String nome;
    private double salario;
    public Funcionario(String nome, double salario) {
        super();
        this.nome = nome;
        this.salario = salario;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public double getSalario() {
        return salario;
    }
    public void setSalario(double salario) {
        this.salario = salario;
    }
}
```

Requisitos de uma LP: Legibilidade

```
public class AppVetorFuncionario {
    public static void main (String[] args) {
        Funcionario[] func = new Funcionario[3];
        String nome;
        double salario;

        // ler os dados
        for (int i=0; i<func.length; i++) {
            System.out.println("NOME");
            nome = new Scanner(System.in).nextLine();
            System.out.println("SALARIO");
            salario = new Scanner(System.in).nextDouble();
            func[i] = new Funcionario (nome, salario);
        }

        //imprimir na tela
        for (int i=0; i<func.length; i++) {
            System.out.println("NOME: "+ func[i].getNome() +
                               "    SALARIO: R$ "+func[i].getSalario());
        }
    }
}
```

Requisitos de uma LP: Legibilidade

- **Tipos de Dados e Estruturas adequados**



Requisitos de uma LP: Legibilidade

- Tipos de Dados e Estruturas adequados
  - Em **C** não tem tipo **booleano**. Assim, um inteiro valendo zero é falso e diferente de zero é verdadeiro.

```
if (ok<>0){
…
}
```

```
if (ok){
…
}
```

```
if (ok==true){
…
}
```

Requisitos de uma LP: Legibilidade

- **Boa sintaxe**
  - Contra exemplos

```
if (x>0)
   if(cont==0)
      x=5;
else
   x=0;
```

```
REAL: DO
INTEGER: REAL
```

Declaração de variáveis em FORTRAN

Requisitos de uma LP: Redigibilidade

- A maioria das características da linguagem que afetam a legibilidade também afetam a capacidade de escrita;

Requisitos de uma LP: Fácil de Aprender

- Na computação, o aprendizado contínuo é fundamental

Requisitos de uma LP: Tratamento de Exceção

C++:

```
try
{
  ...
}
catch (Exception &exception)
{
  ...
}
```

Java:

```
try
{
  ...
}
catch(Exception e)
{
  ...
}
```

Requisitos de uma LP: Eficiência

- velocidade de execução do programa objeto

Requisitos de uma LP: Reuso de Código

- Subprogramas, modularização, estrutura de dados

```
package introducao;
import java.util.Scanner;

public class AppVetorFuncionarioComMetodos {
    public static void main (String[] args) {
        Funcionario[] func = new Funcionario[3];
        leDados(func);
        imprimeDados(func);

    }
    // ler os dados
    public static void leDados(Funcionario[] func) {
    //imprimir na tela
    public static void imprimeDados(Funcionario[] func) {
}
```

Requisitos de uma LP: Reuso de Código

```
    // ler os dados
    public static void leDados(Funcionario[] func) {
        String nome;
        double salario;

        for (int i=0; i<func.length; i++) {
            System.out.println("NOME");
            nome = new Scanner(System.in).nextLine();
            System.out.println("SALARIO");
            salario = new Scanner(System.in).nextDouble();
            func[i] = new Funcionario (nome, salario);
        }
    }
    //imprimir na tela
    public static void imprimeDados(Funcionario[] func) {
        for (int i=0; i<func.length; i++) {
            System.out.println("NOME: "+ func[i].getNome() +
                    "    SALARIO: R$ "+func[i].getSalario());
        }
    }
}
```

Requisitos de uma LP: Reuso de Código

Outro exemplo de reuso de código

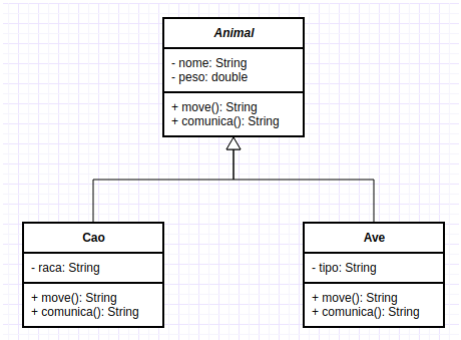
```
package introducao;
import java.util.Scanner;
public class SomaDeVetorComMetodo {
    public static void main(String[] args) {
        int[] vetor1 = new int[5];
        int[] vetor2 = new int[5];
        int[] soma = new int[5];

        leVetor(vetor1);
        leVetor(vetor2);
        soma = soma(vetor1, vetor2);
        System.out.println ("Vetor 1");
        imprime(vetor1);
        System.out.println ("Vetor 2");
        imprime(vetor2);
        System.out.println ("Soma");
        imprime(soma);
    }
```

Requisitos de uma LP: Reuso de Código

```
    public static void leVetor(int[] vet) {
        for (int i=0; i<vet.length; i++){
            System.out.println("Entre com o elemento "+(i+1));
            vet[i] = new Scanner(System.in).nextInt();
        }

    }
    public static int[] soma (int[] vet1, int[] vet2) {
        int[] soma = new int[vet1.length];
        for (int i=0; i<vet1.length; i++){
            soma[i] = vet1[i]+vet2[i];
        }
        return soma;
    }
    public static void imprime (int[] vet) {
        System.out.print ("| ");
        for (int i=0; i<vet.length; i++){
            System.out.print (vet[i]+" | ");
        }
    }
}
```

Requisitos de uma LP: Reuso de código

- Herança



Requisitos de uma LP: Reuso de código

```
public abstract class Animal {
    private String nome;
    private double peso;
    public Animal(String nome, double peso) {
        super();
        this.nome = nome;
        this.peso = peso;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public double getPeso() {
        return peso;
    }
    public void setPeso(double peso) {
        this.peso = peso;
    }
    public String toString() {
        return "Animal [nome=" + nome + ", peso=" + peso + "]";
    }
    public abstract String move();
    public abstract String comunica();
}
```

Requisitos de uma LP: Reuso de código

```java
public class Cao extends Animal{
    private String raca;
    public Cao(String nome, double peso, String raca) {
        super(nome, peso);
        this.raca = raca;
    }
    public String getRaca() {
        return raca;
    }
    public void setRaca(String raca) {
        this.raca = raca;
    }
    public String toString() {
        return "Cao [raca=" + raca + ", toString()="
                + super.toString() + "]";
    }
    public String move() {
        return "cao esta andando";
    }
    public String comunica() {
        return "cao esta latindo";
    }
}
```

Requisitos de uma LP: Reuso de código

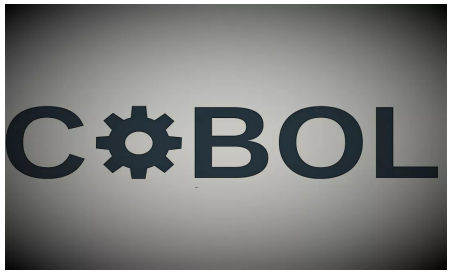
```java
public class Ave extends Animal{
    private String tipo;
    public Ave(String nome, double peso, String tipo) {
        super(nome, peso);
        this.tipo = tipo;
    }
    public String getTipo() {
        return tipo;
    }
    public void settipo(String tipo) {
        this.tipo = tipo;
    }
    public String toString() {
        return "Ave [tipo=" + tipo + ", toString()="
                + super.toString() + "]";
    }
    public String move() {
        return "ave esta voando";
    }
    public String comunica() {
        return "ave esta gritando";
    }
}
```

Requisitos de uma LP: Flexibilidade

- Facilidade de modificar o programa a partir de novos requisitos

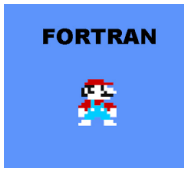
Domínios de programação

- Aplicações comerciais
- Facilidade de elaborar relatórios e armazenar números decimais e dados de caracteres



Domínios de programação

- Aplicações científicas
- Necessitam de grande volume de processamento, estruturas de dados simples, operações em ponto flutuante, poucas exigências de entrada e saída e devem ter eficiência.



Domínios de programação

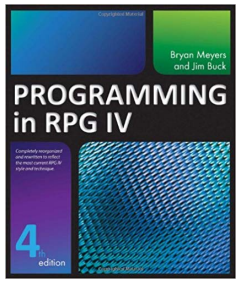
- Programação de novos sistemas
- Devem ser eficientes na execução e recursos de baixo nível que permitam ao software fazer interface com dispositivos externos.

Domínios de programação

- **Linguagens de scripting**
- São usadas colocando-se uma lista de comandos, **script**, em um arquivo para serem executados.
- AWK - usada para deixar os scripts de shell mais poderosos e com mais recursos

Domínios de programação

- **Linguagens de propósitos especiais**
- RPG: uma linguagem de programação, criada para máquinas de cartões perfurados. Campos são especificados para obter dados e gerar relatórios impressos



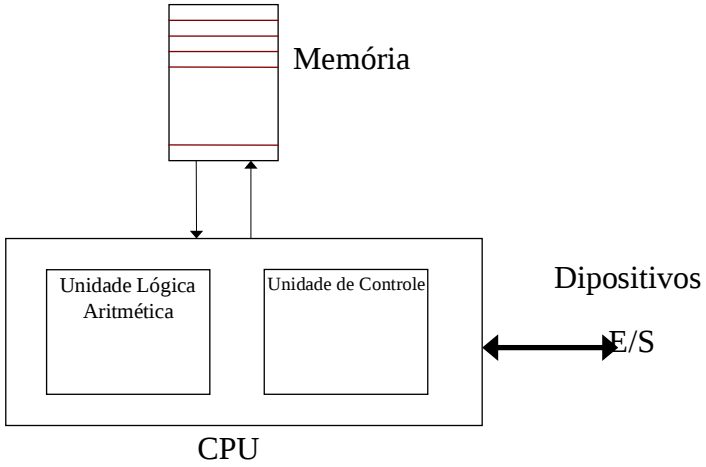
Paradigmas

- Modelo, padrão ou estilo de programação suportado por determinado grupo de linguagens.
  - Paradigma imperativo
  - Paradigma orientado a objetos
  - Paradigma funcional
  - Paradigma lógico

Linguagens Assertativas

- Baseiam-se em especificar os passos que um programa deve seguir para alcançar um estado desejado
- Em expressões que modificam valores de entidades – *dados* ou *objetos*.
  - **linguagens imperativas** ou *orientadas a dados*, em que a construção de programas é dirigida pelas transformações que ocorrem nos dados
  - **linguagens orientadas a objetos**, em que a construção de programas é dirigida pelas mudanças de estados de objetos.

Paradigma Imperativo

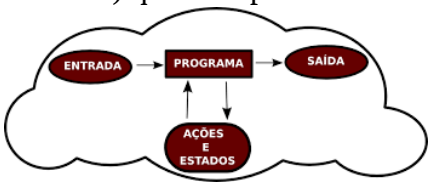


Arquitetura de Von Neumann

- A velocidade da conexão entre a memória de um computador e seu processador, normalmente, determina a velocidade do computador
- Essa conexão é chamada de **GARGALO DE VON NEUMANN,** que é o principal fator limitante na velocidade da arquitetura de computadores de Von Neumann.
- O gargalo de Von Neumann foi uma das principais motivações para a pesquisa e desenvolvimento de computadores paralelos.

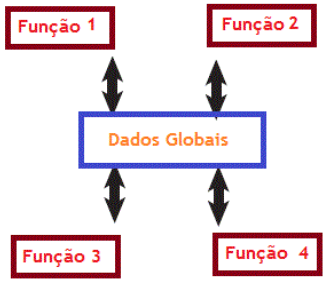
Paradigma Imperativo

- Primeiro paradigma a surgir e ainda é o dominante
- As linguagens imperativas modelam esta arquitetura através de variáveis e comando de atribuição
- Os programas são centrados no conceito de:
  - um estado (modelado por variáveis) e
  - ações (comandos) que manipulam o estado



Paradigma Imperativo

Paradigma também denominado de **procedural**, por incluir subrotinas ou procedimentos como mecanismo de estruturação



Paradigma Imperativo

- Exemplos:
  - FORTRAN, PASCAL, C, BASIC, etc..
- Vantagens
  - Eficiência (embute modelo de Von Neumann)
  - Modelagem “natural” de aplicações do mundo real
  - Paradigma dominante e bem estabelecido

Paradigma Imperativo

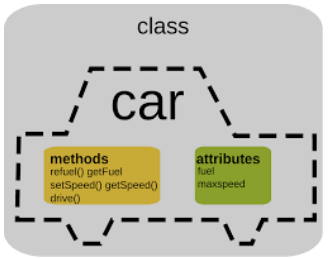
- Desvantagens
  - difícil legibilidade
  - erros introduzidos durante manutenção
  - descrições demasiadamente operacionais focalizam o **como** o processo deve ser realizado

POO

- É uma subclassificacão do imperativo, sendo muito usado junto com este, embora existam, por exemplo, versões de Lisp OO como CLOS (funcional)
- A diferença é mais de metodologia quanto à concepção e modelagem do sistema

POO

- A grosso modo, uma aplicação é estruturada em módulos (classes) que agrupam um estado (atributos) e operações (métodos) sobre este
- Classes podem ser estendidas e/ou usadas como tipos (cujos elementos são objetos)



POO

```
Exemplo em JAVA:
class  Data {
   private int dia;
   private int mes;
   private int ano;
   public void alterar_Data ( int d, int m, int a){

      dia = d;
      mes = m;
      ano = a;
   }
}
public class Principal {
   public static void main (String[] args ) {
      Data data_Nascimento =  new Data();
      data_Nascimento.alterar_Data (1, 1, 1980);
   }
}
```

POO

- A programação é desenvolvida através de trocas de mensagens entre objetos, ligados a classes que se interligam através de heranças.
- Exemplos: Smaltalk, Java, C++, Ruby, C#, etc.

```
srt1.equals(str2)

gato.mia()                    carro1.acelera();
```

POO

- Vantagens
  - Eficiência
  - modularidade, reusabilidade e extensibilidade.
  - Ciclo de vida mais longo e desenvolvimento mais rápido de sistemas, diminuindo o custo tanto de desenvolvimento quanto de manutenção
  - Possibilidade de construir sistemas mais complexos

POO

- Desvantagens
  - Maior esforço na modelagem de um sistema OO do que estruturado
  - Dependência de funcionalidades já implementadas em superclasses no caso da herança,
  - Implementações espalhadas em classes diferentes.

Linguagens Declarativas

- Baseiam-se em *expressões* que verificam ou induzem a que ocorram relações entre declarações.
- **Funcionais**, essas relações são caracterizadas por mapeamentos entre estruturas simbólicas;
- **Lógicas**, essas relações são caracterizadas como expressões da lógica matemática;

Paradigma Lógico

- Programas são relações entre E/S
- Estilo declarativo, como no paradigma funcional
- Programas consistem em proposições e conectores lógicos
- Linguagens que utilizam-se de lógica simbólica e um processo de inferência lógica.
- A linguagem mais importante é o PROLOG

PROLOG – Fatos

cadeia_alimentar.pl [modified]

File Edit Browse Compile Prolog Pce Help

```
animal(urso).
animal(peixe).
animal(raposa).
animal(veado).
animal(minhoca).
animal(lince).
animal(coelho).
animal(guaxinim).
animal(mosca).
animal(peixinho).
planta(grama).
planta(alga).
planta(plantacarnívora).
```

PROLOG – Perguntas

SWI-Prolog (Multi-threaded, version 5.6.64)

File Edit Settings Run Debug Help

```
1 ?- animal(urso).
true.

2 ?- animal(gato).
false.

3 ?- planta(X).
X = grama ;
X = alga ;
X = plantacarnívora.
```

PROLOG – Melhorando os Fatos

```
come(urso,peixe).
come(peixe,peixinho).
come(peixinho,alga).
come(peixe,alga).
come(urso,raposa).
come(veado,grama).
come(peixe,minhoca).
come(urso,guaxinim).
come(raposa,coelho).
come(urso,veado).
come(lince,veado).
come(plantacarnívora,mosca).
come(veado,plantacarnívora).
```

PROLOG – Perguntas

SWI-Prolog (Multi-threaded, version 5.6.64)

File Edit Settings Run Debug Help

```
1 ?- come(urso, peixe).
true .

2 ?- come(urso, gato).
false.

3 ?- come(urso, X).
X = peixe ;
X = raposa ;
X = guaxinim ;
X = veado.

4 ?- come(X, alga).
X = peixinho ;
X = peixe ;
false.
```

PROLOG - Regras

```
carnívoro(X):-come(X,Y), animal(Y).
herbívoro(X):-come(X,Y),planta(Y),\+carnívoro(X).
pertence_a_cadeia(X,Y):-come(Y,X).
pertence_a_cadeia(X,Y):-come(Z,X),pertence_a_cadeia(Z,Y).
```

PROLOG – Perguntas

1 – Quem pertence a cadeia alimentar do urso?

?- pertence_a_cadeia(X,urso).

2 – Quem pertence a cadeia alimentar do urso e também come planta?

?- pertence_a_cadeia(X,urso),herbívoro(X).

3 – A minhoca pertence a cadeia alimentar de quem?

?- pertence_a_cadeia(minhoca,X).

Paradigma Lógico

- Aplicações: prototipação em geral, sistemas especialistas, banco de dados, ...
- Vantagens
  - Não precisa se preocupar com declaração de variáveis
  - Manipulação de programas mais simples:
    - Prova de propriedades
    - Transformação (exemplo: otimização)
  - Programas mais lógicos, gerando menos erro e manutenção
  - Permite processamento paralelo
  - Tempo de prototipação mínimo

Paradigma Lógico

- Desvantagens
  - Linguagens usualmente não possuem tipos, nem são de alta ordem
  - Implementações ineficientes
  - Mecanismos primitivos de E/S e formatação

Paradigma Funcional

- Programas são funções que descrevem uma relação explícita e precisa entre E/S
- Imita ao máximo as funções matemáticas
- Não há o conceito de estado (variáveis) nem comandos como atribuição (Estilo declarativo)
- Iteração é feita por meio de recursividade
- Exemplos: LISP, inicialmente puramente funcional depois adquiriu alguns conceitos imperativos. Scheme, ML, Haskell (puramente funcional)
- Aplicação: prototipação em geral e IA

Paradigma Funcional

- Vantagens
  - Não precisa se preocupar com declaração de variáveis
  - Manipulação de programas mais simples:
    - Prova de propriedades
    - Transformação (exemplo: otimização)
  - Concorrência explorada de forma natural

Paradigma Funcional

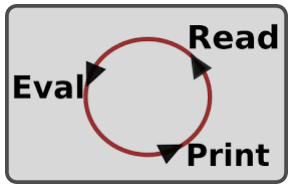
- Desvantagens
  - “O mundo não é funcional!”
  - Implementações ineficientes
  - Mecanismos primitivos de E/S e formatação

LISP

- LISP é interpretada
- LISP é uma linguagem interativa
- Cada parte de um programa pode ser interpretada, corrigida e testada, de forma independente
- **LISP trabalha segundo o ciclo read-eval-print**

LISP

- Fases:
  - Leitura: Lê uma expressão, e constrói internamente um objeto que a represente
  - Avaliação: o objeto construído é analisado de modo a produzir um valor.
  - Escrita: o valor produzido é apresentado ao utilizador através de uma representação textual.



LISP

- LISP se utiliza de notação prefixa

NOTAÇÃO INFIXA : 2 + 3 * 5
NOTAÇÃO PREFIXA : (+ 2 (* 3 5))

Funções já prontas

- *(max 2 19 7)*
- *(min 2 19 7)*
- *(expt 2 4)*
- *(sqrt 9)*
- *(mod 4 2)*
- EVENP

Operadores Lógicos

- **AND** ⇒ Avalia os argumentos da esquerda para a direita até que um deles seja falso. Se nenhum for falso, devolve o valor do último argumento.
- **OR** ⇒ Avalia os argumentos da esquerda para a direita até que um deles seja diferente de falso, devolvendo este valor. Se todos forem falsos, devolve falso.
- **NOT** ⇒ Avalia verdadeiro se o argumento for falso e vice-versa.

Funções

SINTAXE:
> (**defun** nome (par1 par2 ... parn)
corpo)

- Fazer um função para retornar o quadrado de um número

```
> (quadrado (quadrado (+ 1 2))
(quadrado (quadrado (3))
(quadrado (*  3  3))
(quadrado 9)
(* 9 9)
81
```

Recursividade

- Em funções recursivas deve haver:
  1. Um passo básico com resultado reconhecido de imediato
  2. Um passo recursivo em que se tenta resolver um sub-problema inicial

Recursividade

- O objetivo é resolver subproblemas cada vez mais simples até atingir o mais simples de todos, ou seja, aquele que tem resultado imediato.
- Podem ocorrer ERROS, como por exemplo:
  1. Não detectar o caso mais simples
  2. Não diminuir a complexidade do problema
  3. A recursão pode não parar

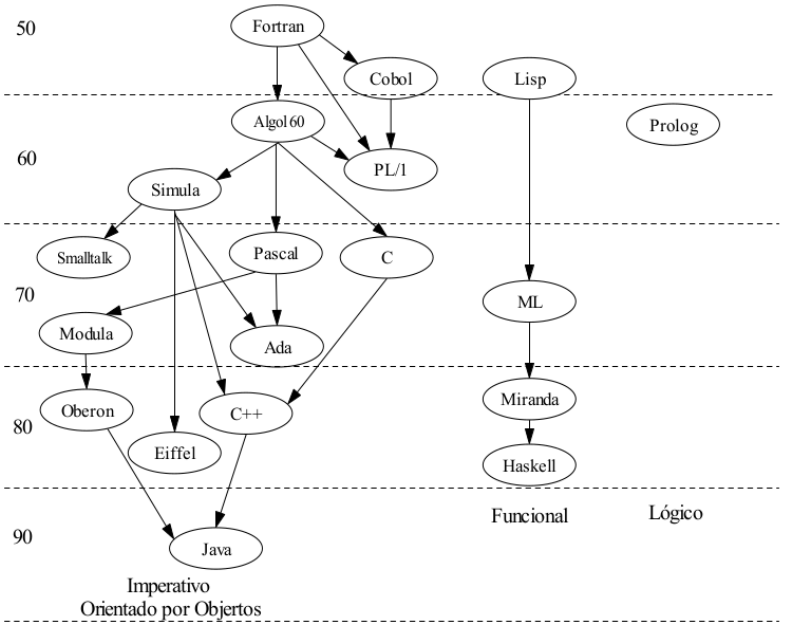
1) Faça funções que calcule:

a) S = 1 + 2 + 3 + ... + n

b) S = 2 + 5 + 8 + 11 + ...

c) $S = 1 + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \ldots$

d) $S = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \ldots$



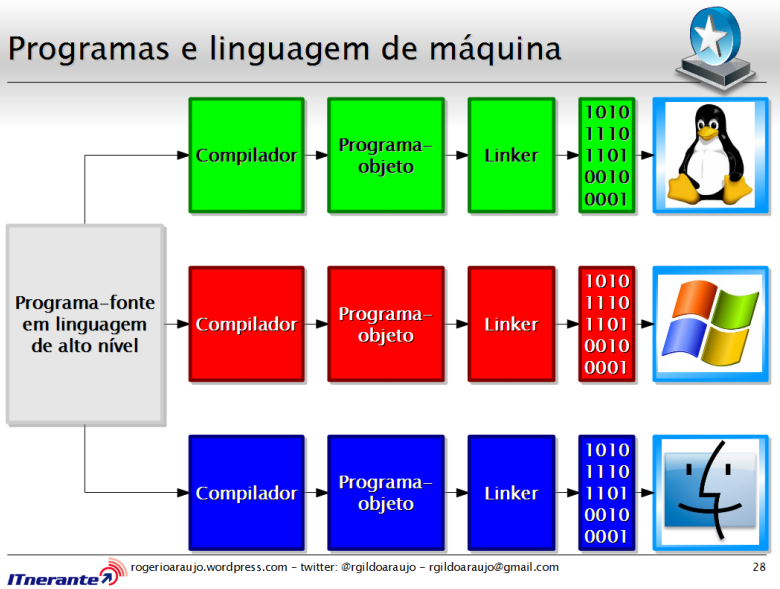
Sintaxe x Semântica

- **Sintaxe** é o ***formato*** dos programas – é a forma de suas expressões, de suas instruções e de suas unidades de programa. Influencia a forma como programas são escritos.
- **Semântica** – ***significado*** dos programas – é o significado das expressões, instruções e unidades de programa. É como os programas devem ser entendidos por outros programadores e executados em um computador.
- A semântica é a parte mais importante e que realmente caracteriza uma linguagem.
- Você só vai saber usar bem um comando se conhecer seu significado.
- Erro sintático pode ser apontado pelo compilador, mas erro semântico pode produzir programa errado ou com comportamentos indesejados e imprevisíveis.

Funcionamento das LP's

- Programa executável.
- Arquivo ou Código fonte
- Tradução
  - Compilação
  - Interpretação

Compilação



Compilação

- Um programa-fonte é escrito em um editor de texto, usando uma linguagem de alto nível;
- Esse código é compilado, gerando um programa-objeto em linguagem de máquina;
- O *linker*, recebe este programa-objeto e o transforma em um programa em linguagem de máquina **executável**, que é específico para cada plataforma
- O compilador e o linker são específicos para traduzir o código-fonte para a linguagem de máquina própria de cada plataforma;

Compilação

- Vantagem:
  - Execução rápida do programa.
  - Gasta menos memória.
- Desvantagem:
  - Falta de portabilidade.

Interpretação

Interpretação

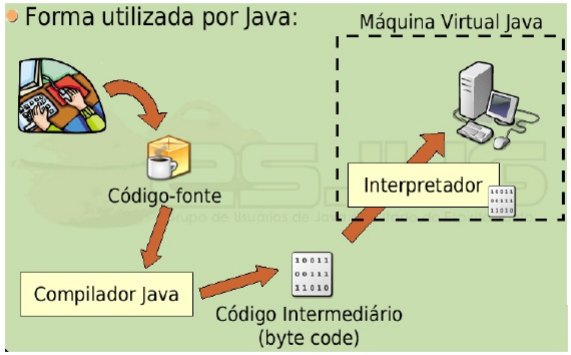
- Não cria arquivo executável
- Cada linha do arquivo fonte é lido, traduzido e executado
- Uma instrução, executada mais de uma vez, será lida e traduzida quantas vezes for executada

Interpretação

- Vantagem
  - Permite fácil implementação de muitas operações de depuração do código-fonte, porque todas as mensagens de erro, em tempo de execução, podem referir-se a unidades de código.
  - Portabilidade.
- Desvantagem
  - Necessidade de se manter na memória tanto o programa fonte como o interpretador fazendo com que o processo se torne lento e gaste mais memória.

Mecanismos Híbridos

TRADUÇÃO HÍBRIDA



Mecanismos Híbridos

- Esse tipo de LP combinam a compilação com a interpretação.
- As instruções do programa fonte são transformadas em códigos intermediários (bytecode). Estas instruções intermediárias são transformadas em linguagem de máquina e executadas, precisando de um INTERPRETADOR.

Mecanismos Híbridos

- Soluciona a lentidão da interpretação e a falta de portabilidade da compilação
- Esse método é mais rápido que a interpretação pura porque as instruções da linguagem fonte são decodificadas somente uma vez.