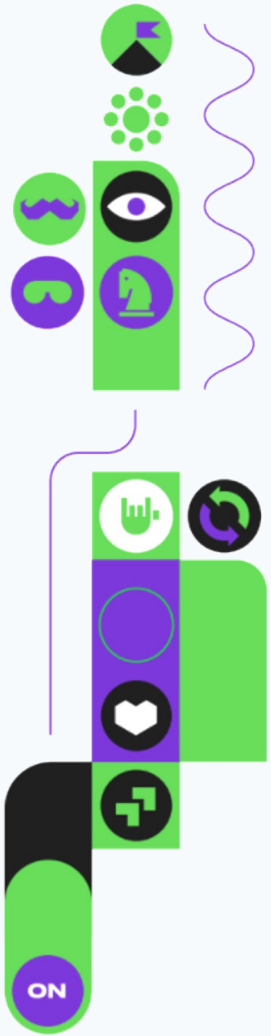
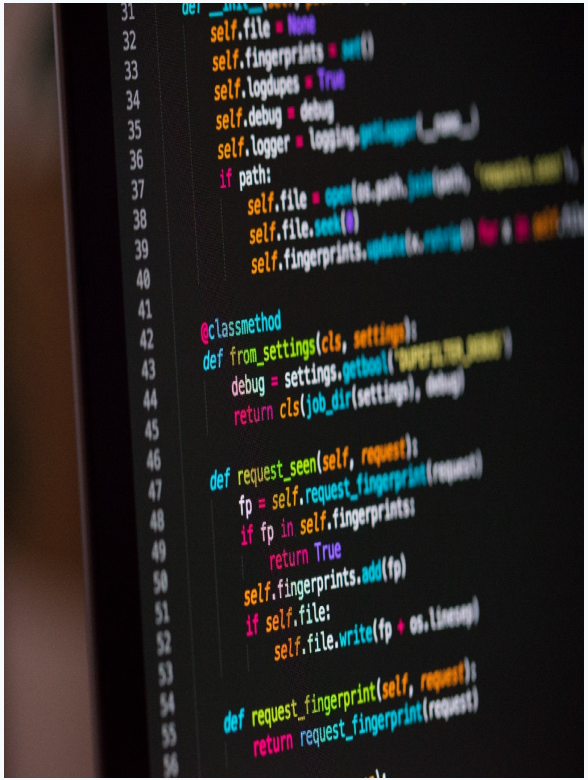


Lógica de programação

GAMA ACADEMY



Lógica de Programação

Lógica de programação é a organização coesa de uma sequência de instruções voltadas à resolução de um problema, ou à criação de um software ou aplicação.

Cada linguagem tem suas próprias particularidades, como sua sintaxe, seus tipos de dados e sua orientação, mas a lógica por trás de todas é a mesma.

A lógica de programação é importante porque é ela quem nos dá as ferramentas necessárias para executar o processo mais básico no desenvolvimento de alguma aplicação: a criação de seu **algoritmo**.

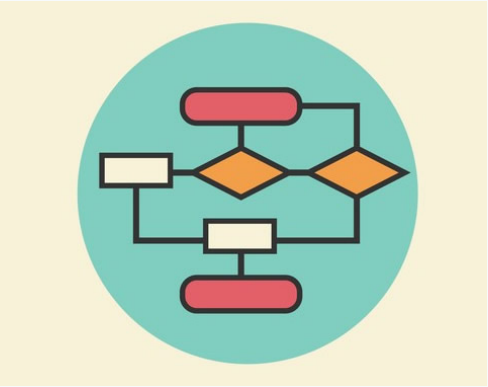
Algoritmo

01.

De acordo com o dicionário, é um processo de cálculo que, por meio de **uma sequência finita** de operações, aplicada a um número finito de dados, **leva à resolução de problemas**.

02.

Vamos tomar como exemplo o café que tomamos de manhã.



Quando perguntam como tomamos nosso café, a maioria de nós responde que, ao acordarmos, preparamos o café com auxílio de uma cafeteira elétrica, colocamos ele em uma caneca e o tomamos.

Mas, ao destrinchar este processo, somos capazes de estipular uma sequência de passos que nos levaram ao ato final de beber este café. Esta sequência pode ser:

1. Ao acordar, levanto da cama;
2. Após levantar da cama, desço as escadas;
3. Após descer as escadas, entro na cozinha;
4. Após entrar na cozinha, pego o pó de café no armário;
5. Após pegar o pó de café, o coloco dentro da cafeteira;
6. Após colocar o pó na cafeteira, jogo água no compartimento específico;
7. Após inserir todos os ingredientes na máquina, aperto o botão de ligar;
8. Quando o café está pronto, pego a garrafa;
9. Após pegar a garrafa, despejo o café dentro de uma caneca;
10. Após colocar o café na caneca, bebo o café.

Um Algoritmo deve ser:

Completo	Todas as ações precisam ser descritas e devem ser únicas.
Sem redundância	Um conjunto de instruções só pode ter uma única forma de ser interpretada.
Determinístico	Se as instruções forem executadas, o resultado esperado será sempre atingido.
Finito	As instruções precisam terminar após um número limitado de passos.

Podemos dividir um algoritmo em três fases fundamentais: **entrada, processamento e saída**.

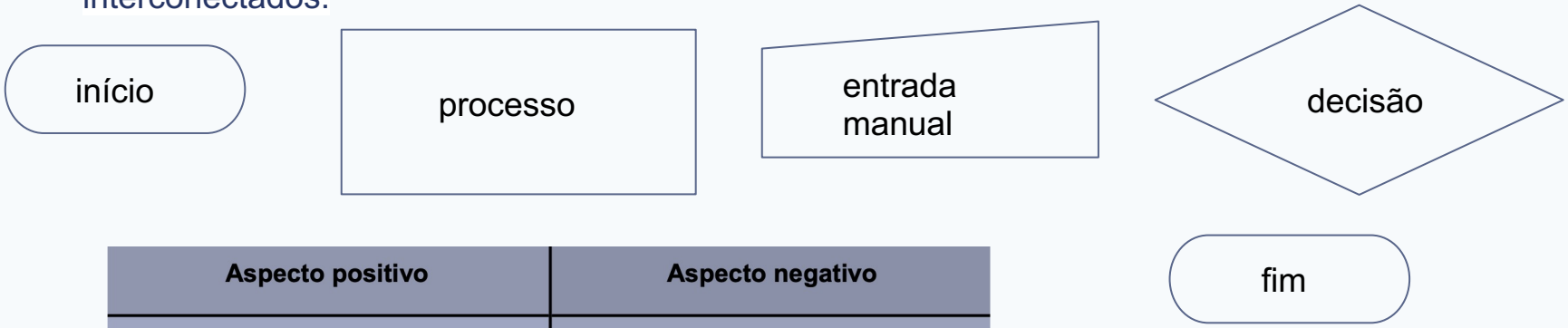
- **Entrada** recebe as informações necessárias para iniciar nosso algoritmo;
- **Processamento** são os procedimentos utilizados para chegar ao resultado final.
- **Saída** é o resultado esperado da fase de processamento, dados já processados

.

Formas mais conhecidas de representação
Descrição narrativa
Fluxograma
Pseudocódigo (Linguagem estruturada ou Portugol)

FLUXOGRAMA

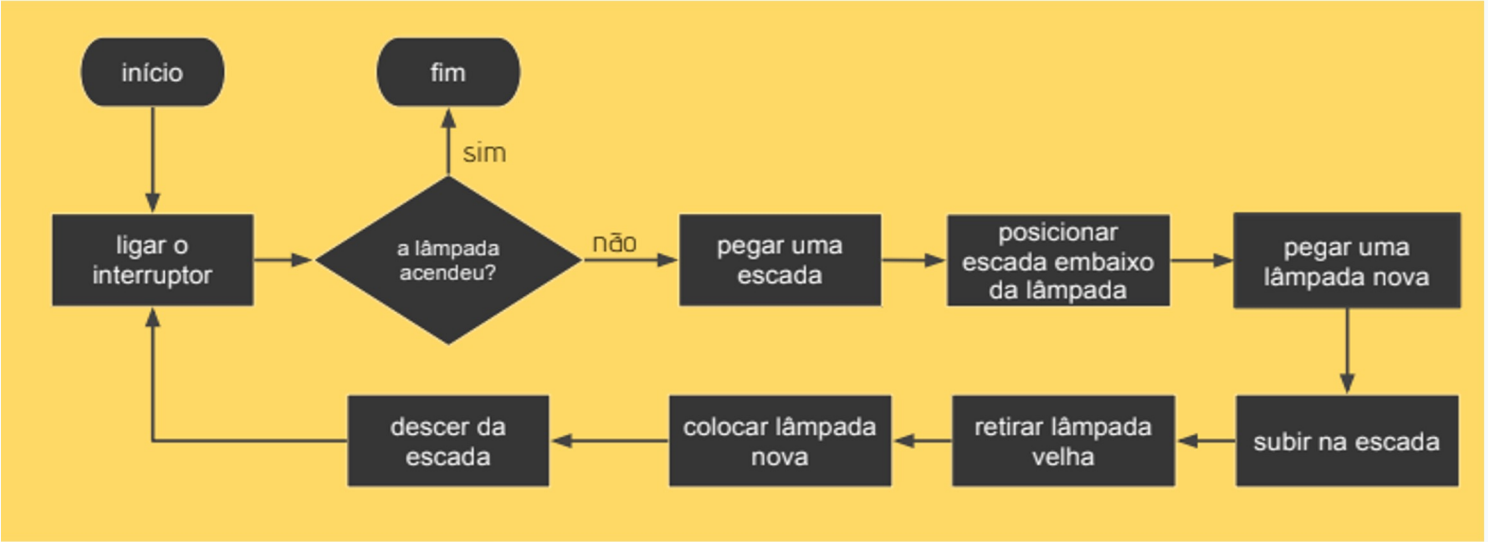
Um fluxograma é a representação gráfica de um procedimento, problema ou sistema, cujas etapas ou módulos são ilustrados de forma encadeada por meio de símbolos geométricos interconectados.



Aspecto positivo	Aspecto negativo
O entendimento de elementos gráficos é mais simples que o entendimento de textos.	Os fluxogramas devem ser entendidos e o algoritmo resultante não é detalhado, dificultando sua transcrição para um programa.

Fluxograma

Algoritmo troca de lâmpada



Pseudocódgo

É rico em detalhes, como a definição dos tipos das variáveis usadas no algoritmo

Estrutura básica do pseudocódigo
Algoritmo <nome_do_algoritmo> <declaração_de_variáveis> **Início** <corpo do algoritmo> **Fim**

Algoritmo	Palavra que indica o início da definição de um algoritmo em forma de pseudocódigo.
<nome_do_algoritmo>	Nome simbólico dado ao algoritmo com a finalidade de distingui-lo dos demais.
<declaração_de_variáveis>	Parte opcional onde são declaradas as variáveis globais usadas no algoritmo.
Início e Fim	Palavras que delimitam o início e o término, respectivamente, do conjunto de instruções do corpo do algoritmo.

Exemplo: - Cálculo da média de um aluno:

Algoritmo Calculo_Media

Var Nota1, Nota2, MEDIA: real; Início

Leia Nota1, Nota2;

MEDIA ← (Nota1 + Nota2) / 2;

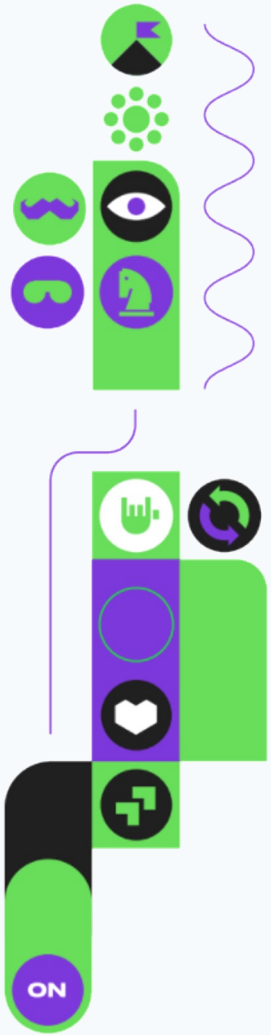
Se MEDIA >= 7 então

Escreva “Aprovado”; Senão

Escreva “Reprovado”; Fim_se

Fim

Aspecto positivo	Aspecto negativo
Representação clara sem as especificações de linguagem de programação.	As regras do pseudocódigo devem ser aprendidas.



Java

Operadores

O que são Operadores?

Você sabe o que são Operadores?

- Operadores são símbolos que representam atribuições, cálculos e ordem dos dados;
- As operações possuem uma ordem de prioridades (alguns cálculos são processados antes de outros);
- Os operadores são utilizados nas expressões matemáticas, lógicas, relacionais e de atribuição.

Quanto ao número de operandos sobre os quais atuam	
Unários: quando atuam sobre um único operando.	**Binários:** quando atuam sobre dois operandos, que podem ser: **duas variáveis**, **duas constantes**, ou **uma variável** e **uma constante**.

Exemplos:

Unário:

-x (o valor armazenado no operando x passa a ser negativo)

x++ (incrementa +1 na variável x).

Obs.:

++ significa adicionar +1 ao valor da variável

-- significa diminuir -1 do valor da variável

Binário:

z= x+y (somatória entre as variáveis x e y)

z=x+7 (somatória entre uma variável e uma constante)

Quanto ao tipo de dado dos operandos e do valor resultante de sua avaliação

- ✓ Operadores Aritméticos;
- ✓ Operadores de Atribuição;
- ✓ Operadores Lógicos;
- ✓ Operadores Relacionais.

Operadores aritméticos

- Conjunto de símbolos que representa as operações básicas da matemática como: somar, subtrair, multiplicar, dividir e etc.
- Esses operadores somente poderão ser utilizados entre variáveis com os tipos de dados numéricos inteiros e/ou numéricos reais.

+	operador de adição
-	operador subtração
*	operador de multiplicação
/	operador de divisão
%	operador de módulo (ou resto da divisão)

Obedecem às regras matemáticas comuns:

+ As expressões de dentro de parênteses são sempre resolvidas antes das expressões fora dos parênteses;

+ Quando existe um parêntese dentro de outro, a solução sempre inicia do parêntese mais interno até o mais externo (de dentro para fora);

+ Quando duas ou mais expressões tiverem a mesma prioridade, a solução é sempre iniciada da expressão mais à esquerda até a mais à direita.

Exemplo:

```
Algoritmo Calculo_Area_Quadrado
var lado, area :real;
Início
  Leia lado;
  area ← (lado * lado);
  Escreva "A área do quadrado é" + area;
Fim
```

Opções de operadores relacionais

- São utilizados para comparar valores entre variáveis e expressões do mesmo tipo;
- O retorno desta comparação é sempre um valor do tipo booleano (verdadeiro/falso).

>	Utilizado quando desejamos verificar se uma variável é maior que outra.
>=	Utilizado quando desejamos verificar se uma variável é maior ou igual a outra
<	Utilizado quando desejamos verificar se uma variável é menor que outra.
<=	Utilizado quando desejamos verificar se uma variável é menor ou igual a outra.

Exemplo:

```
Algoritmo Pode_Tirar_Carteira_de_Motorista
var idade :inteiro;
Início
   Leia idade;
   if idade >= 18
      Escreva “Pode tirar carteira de motorista.";
   else
      Escreva “Não pode tirar carteira de motorista.";
Fim
```

Operadores Lógicos

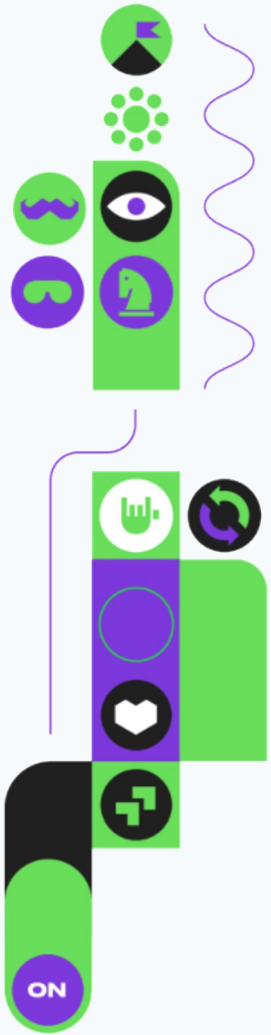
Fazem comparações com o objetivo de avaliar expressões em que o resultado pode ser verdadeiro ou falso, ou seja, implementando a lógica booleana;

O retorno desta comparação é sempre um valor do tipo booleano (lógico).

Operadores Lógicos		
Conjunção **e/and/&&**	**Disjunção** **ou/or/||**	**Negação** **não/not**
As duas condições devem ser verdadeiras para que o resultado seja verdadeiro.	Pelo menos uma condição deve ser verdadeira para que o resultado seja verdadeiro.	Inverte o valor do resultado da condição.

Exemplo:

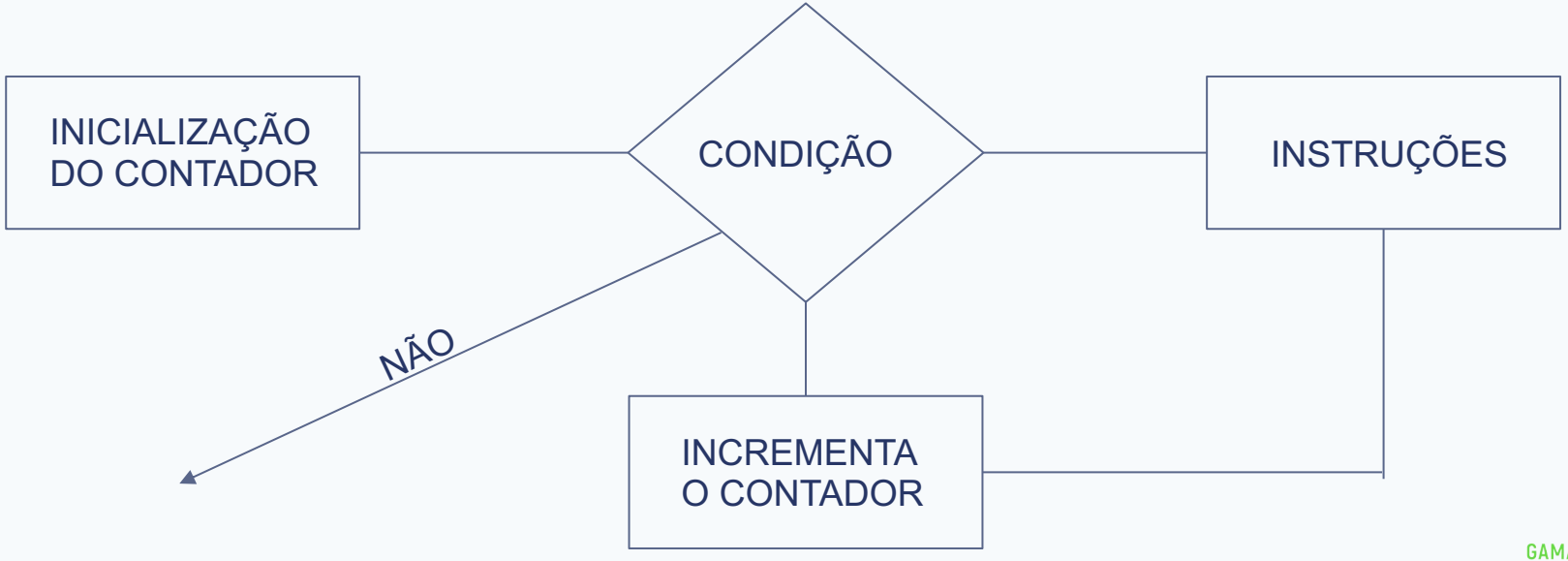
```
Algoritmo Verifica_Aluno_Aprovado
var nota, frequencia :real;
Início
   Leia nota, frequencia;
   if nota >=7 e frequencia >= 70%
      Escreva “Aprovado”;
   else
      Escreva “Reprovado”;
Fim
```



Java

Estruturas de repetição

Dentro da lógica de programação é uma **estrutura** que permite executar mais de uma vez o mesmo comando ou conjunto de comandos, de acordo com uma condição ou com um contador.



O que são?

- São comandos que permitem que uma sequência de instruções seja executada várias vezes até que uma condição seja satisfeita;

- Se uma instrução ou uma sequência de instruções precisa ser executada várias vezes, deve-se utilizar uma estrutura de repetição.

Para que servem?

- Servem para repetir um conjunto de instruções sem que seja necessário escrevê-las várias vezes;

- Permitem que um trecho do algoritmo seja repetido, em um número determinado ou indeterminado de vezes, sem que o código a ser repetido tenha que ser escrito novamente;

- As estruturas de repetição também são chamadas de Laços ou Loops.

Funcionamento

- As estruturas de repetição envolvem a avaliação de uma condição (teste);
- A avaliação resulta em valores Verdadeiros ou Falsos;
- Se o resultado da condição é Falso, não é iniciada a repetição ou, caso esteja em execução, é encerrada a repetição;
- Se o resultado da condição for Verdadeiro, é iniciada a repetição ou, caso esteja em execução, é reiniciada a execução das instruções dentro da Estrutura de Repetição;
- A avaliação da condição é sempre novamente realizada após a execução da última instrução dentro da estrutura de repetição;
- A única Estrutura de Repetição que não realiza a avaliação da condição antes de iniciar é a Do/While (Faça/Enquanto).
- Desta forma, é assegurado que todas as instruções dentro da Estrutura de Repetição do Do/While serão executadas pelo menos uma vez.

Tipos de Estruturas de Repetição

While

O termo **while** pode ser traduzido para o português como “enquanto”. Este termo é utilizado para construir uma estrutura de repetição que executa, repetidamente, uma única instrução ou um bloco delas “enquanto” uma expressão booleana for verdadeira.

```
//INCREMENTADO - de 0 à 9
        int i = 0;
        while(i<=9){
            i = i + 1;
            System.out.println( i );
        }
```

Do While

A diferença desse iterador para os outros, é que o bloco de instrução será executado no mínimo uma única vez. Como podemos ver no exemplo abaixo:

```
int i = 0;

        do{
            System.out.println( i );
            i++;
        }while(i <= 10);
```

FOR

Controlando fluxo com laços

```
for (int i = 0; i < 5; i++) {
 if( i % 2 ==  0) {
  System.out.println(i);
 }
}
```

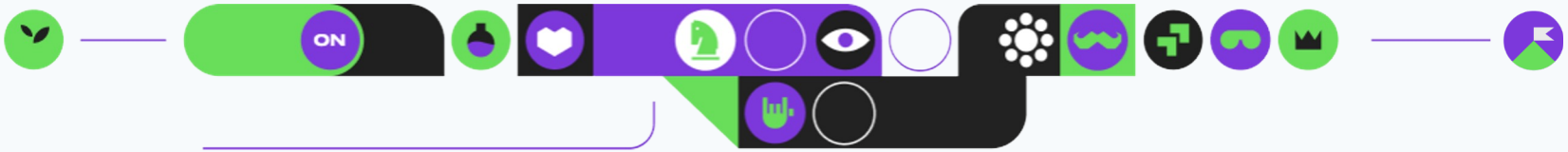
For Each

Projetado especificamente para iterar sobre matrizes e coleção de objetos.

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {
  System.out.println(i);
}
```

Atividade em Grupo



01. Faça um algoritmo que mostre o passo a passo para trocar uma de lâmpada queimada.

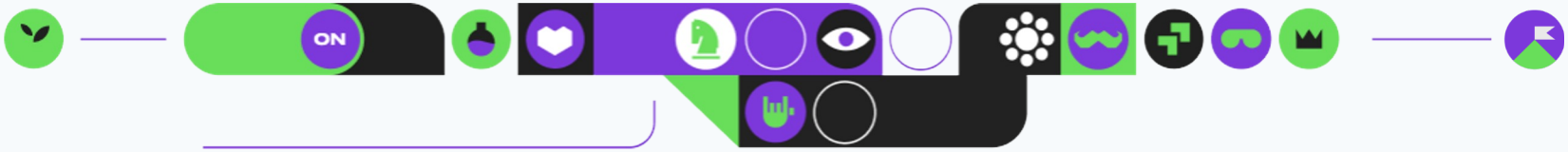
02. Faça um algoritmo que mostre o passo a passo para passear com seu animal de estimação.

03. Faça um algoritmo que mostre o passo a passo para acessar um computador.

04. Faça um algoritmo que mostre o passo a passo para lavar um copo

05. Faça um algoritmo que mostre o passo a passo para postar uma foto em um rede social

Obrigada



GAMA ACADEMY