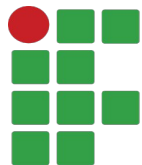


Variáveis de Programação

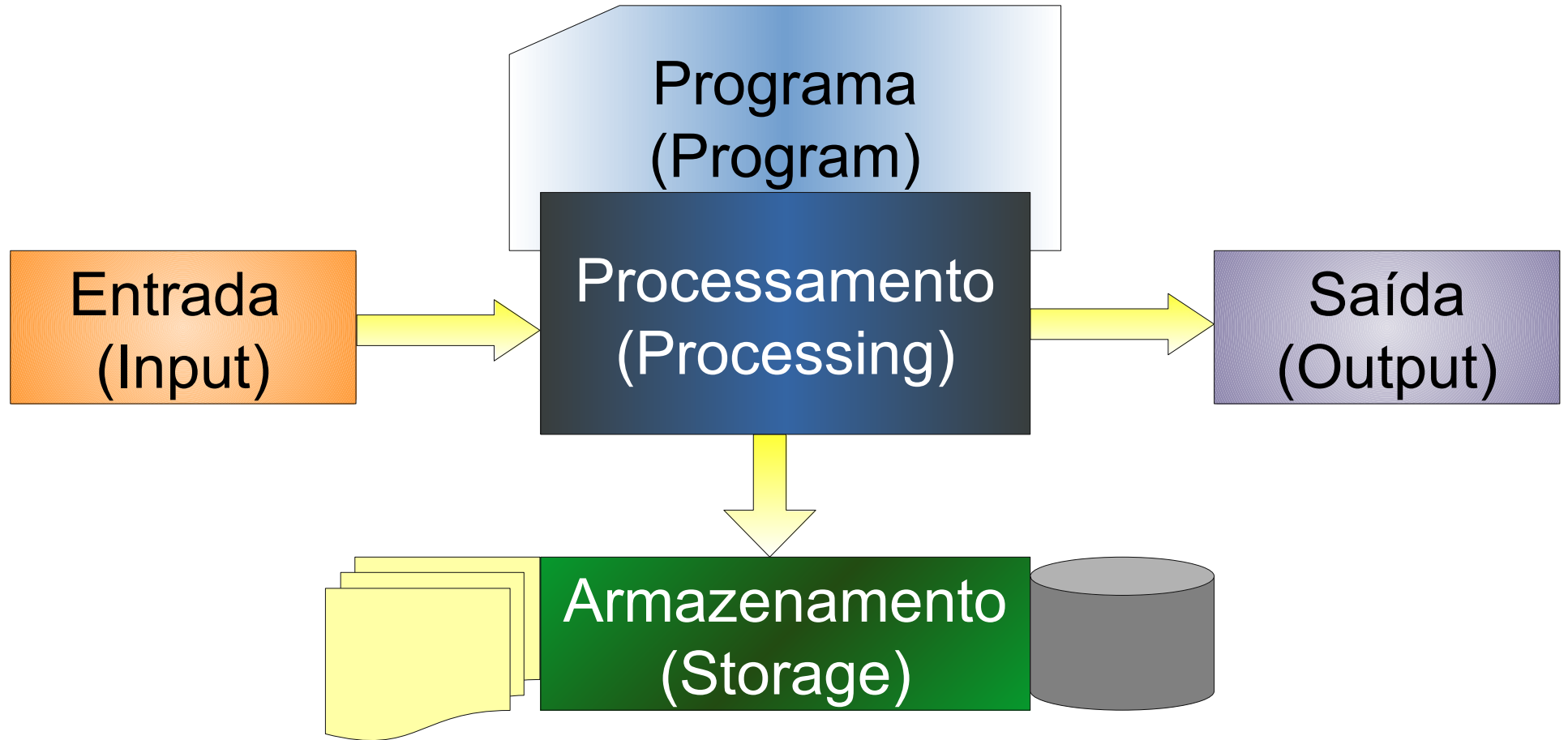
Prof. Alexandro M. S. Adário



INSTITUTO FEDERAL

Rio Grande do Sul
Campus Erechim

A **finalidade** de um **algoritmo computacional** é realizar uma tarefa de processamento, a partir de um conjunto de **entradas** coletadas, gerando um conjunto de valores de **saída**, que podem ou não ser **armazenados**.



Normalmente, chamamos estas entradas de DADOS e as saídas de INFORMAÇÃO.

Mas, por que essa distinção?

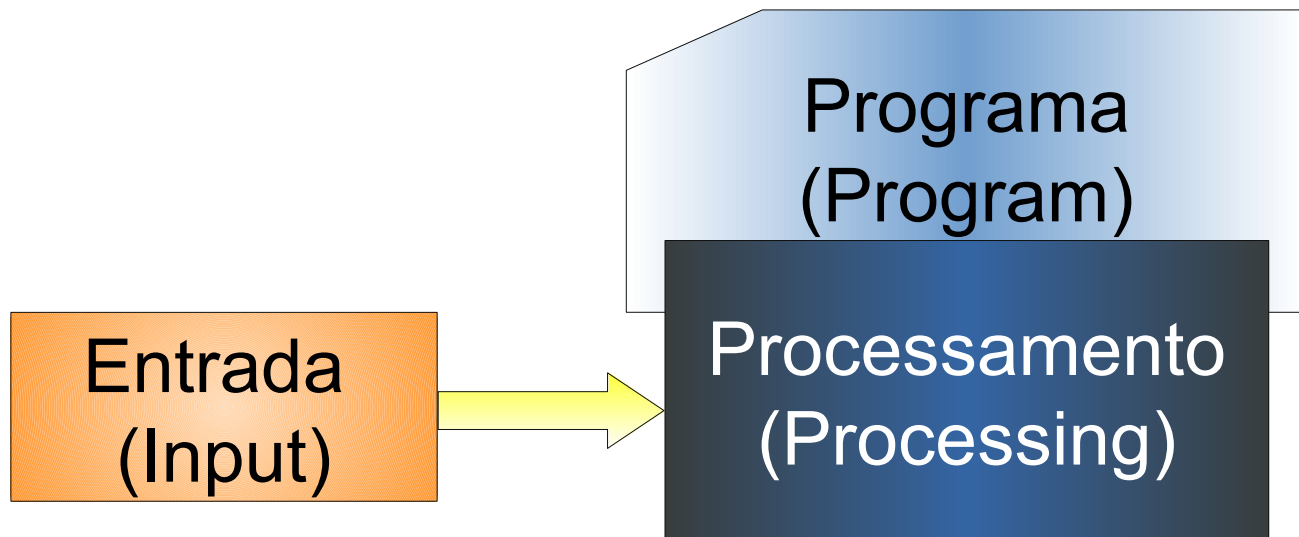
Dado

“Elemento que serve de base à resolução de um problema. Não tem sentido sozinho, sem contexto.”

Informação

“Conjunto organizado e estruturado de dados relacionados a uma idéia, conceito ou assunto.”

Então, geralmente dizemos que um algoritmo recebe DADOS para produzir INFORMAÇÃO a partir deles.



Quando essas entradas são recebidas, o que é feito com elas?
Elas são armazenadas?
Elas são registradas/anotadas?

Os dados :

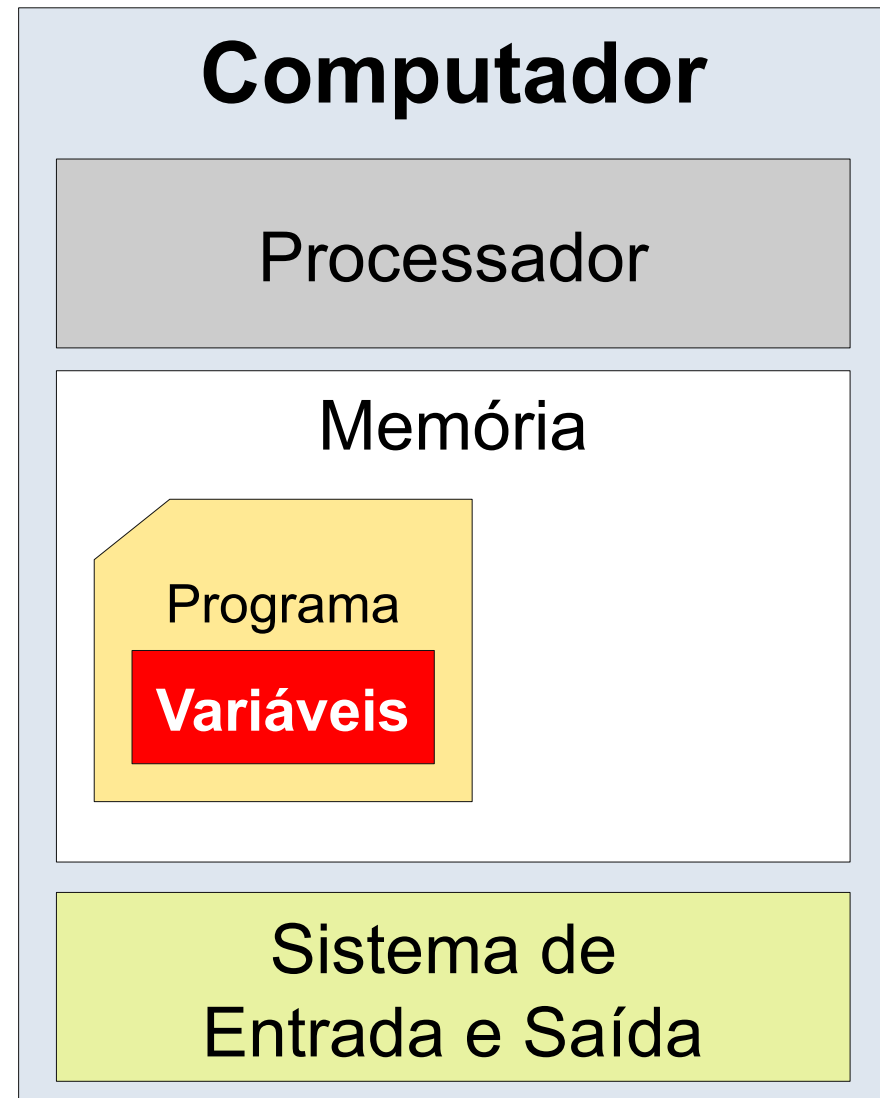
- precisam de um “recipiente” para serem armazenados
- possuem diferentes “naturezas”/ “tipos”
- precisam de um local adequado para o armazenamento.

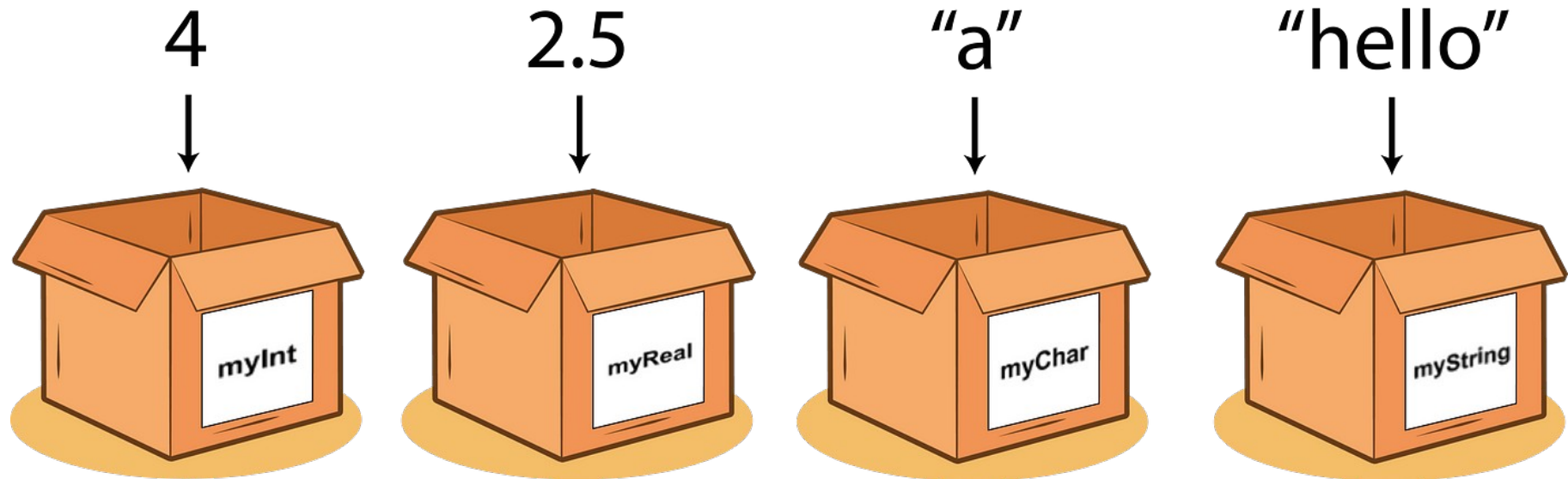
Analogia com os alimentos:

- Cada tipo precisa de um “**recipiente**” específico.
- Não podem ser armazenados em qualquer lugar da casa e nem em qualquer lugar da despensa/cozinha.

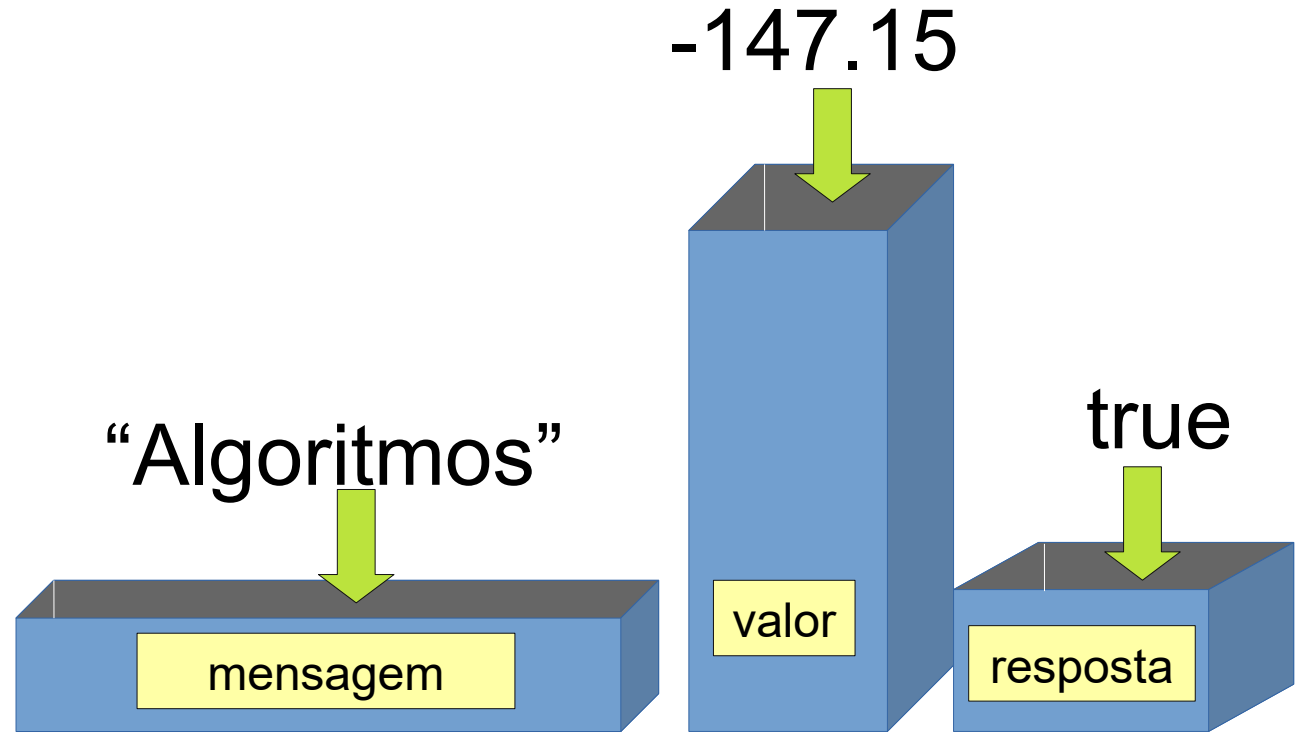


Uma **variável** é um espaço dedicado, dentro da área de um programa, armazenado na memória, em que são registrados os dados durante a execução





Uma **variável** é um “recipiente” que armazena um tipo de dado específico.



A cada variável está associado um identificador, um tipo e um conteúdo.

Java define 8 tipos primitivos:

Lógico	boolean
Textual	char
Inteiros	byte, short, int e long
Ponto Flutuante	float e double

O processo de “criar” uma variável é chamado de **declaração** da variável

```
int numero;
```

O tipo de dados no exemplo é **int** (inteiro de 32 bits) e o nome/identificador da variável é **numero**.

Ao **declarar** uma variável, já é possível atribuir a ela um valor inicial, que pode, posteriormente, ser alterado

```
double valor = 2.5;
```

O sinal de igual (=) é o comando de atribuição, através do qual armazenamos um valor na variável.

A atribuição de valor pode ser realizada em qualquer momento do código-fonte. Inclusive a inicialização pode ser feita separadamente.

```
double valor;  
valor = 2.5;
```

Aqui, a declaração e a atribuição de valor foram realizadas em comandos separados.

```
public class UsandoVariaveis {  
    public static void main(String[] args) {  
        // declaração da variável  
        int numero;  
        // atribuição de valor  
        numero = 5;  
        // leitura do valor atual  
        // e impressao na tela  
        System.out.println( numero );  
    }  
}
```

NUNCA suponha que uma variável possui algum valor quando a declaramos.

Variáveis **não são inicializadas automaticamente** quando declaradas.

Operadores de Dados Primitivos

OPERADORES são os símbolos que representam uma operação algébrica clássica, booleana ou relacional (comparação).

Operadores Unários

(usam apenas um operando)

Operador	Significado
<code>++ / --</code>	Incremento/decremento (aumenta/diminui 1 unidade)
<code>+ / -</code>	Unários positivo/negativo
<code>~</code>	Negação inteira (inverte os bits do número)
<code>!</code>	Negação booleana

Exemplo de uso de um operador de incremento.

```
int numero = 5;  
...  
numero++;
```

A variável número foi inicializada com o valor 5. Posteriormente, no código ela foi incrementada, alterando seu valor para 6.

Primeiro usa o valor,

`valor++`

Depois incrementa.

Primeiro incrementa,

`++valor`

Depois usa o valor.


```
public class Incrementando {  
    public static void main(String[] args) {  
        int valor = 5;  
  
        System.out.println( valor++ );  
        System.out.println( valor );  
  
        valor = 10;  
        System.out.println( valor++ );  
        System.out.println( valor );  
    }  
}
```


Operadores Matemáticos

(binários: usam dois operandos)

Operador	Significado
*	Multiplicação
/	Divisão (inteira ou decimal, conforme os tipos)
%	Resto (da divisão inteira)
+	Adição
-	Subtração

```
public class OperandosMatematicos {  
    public static void main(String[] args) {  
        int valor = 5, num = 13;  
  
        System.out.println( valor * num );  
        System.out.println( valor + num );  
        System.out.println( num / valor );  
        System.out.println( num % valor );  
        System.out.println( num - valor );  
    }  
}
```

Regras da Divisão


(na linguagem Java)

Se os tipos são inteiros, a divisão é sempre inteira (sem casas decimais).

Se pelo menos um é de ponto flutuante (**double** ou **float**), a divisão é fracionária.

```
int valor = 2;  
double resultado = 1/valor;  
// resultado é igual a 0 (zero)
```


Valores inteiros



```
int valor = 2;  
double resultado = 1.0/valor;  
// resultado é igual a 0.5 (zero ponto cinco)
```

Ponto Flutuante

Inteiro



Operadores Relacionais

Operador	Significado
<	Menor que
>	Maior que
<=	Menor que ou igual
>=	Maior que ou igual
==	Igual
!=	Diferente (NÃO igual)

Obs.: Não servem para comparar objetos,
apenas tipos primitivos

Operadores Booleanos

Operador	Significado
&	AND *
^	XOR *
	OR *
& &	AND de curto-circuito
	OR de curto-circuito

* Também operam sobre valores inteiros, bit-a-bit.

$$\begin{array}{r}
 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1 \\
 \& 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\
 \hline
 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1
 \end{array}$$

$$\begin{array}{r}
 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1 \\
 | 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\
 \hline
 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1
 \end{array}$$

$$\begin{array}{r}
 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1 \\
 \wedge 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\
 \hline
 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0
 \end{array}$$

Curto-Circuito

Em uma operação de AND ou OR, conhecendo o primeiro valor, já podemos, em alguns casos, definir o resultado.

AND: se o primeiro é false, está definida.

OR: se o primeiro é true, está definida.

No caso do curto-circuito, o restante da expressão booleana não é mais avaliada.

Se dado1 é false,

dado1 & & dado2

não avalia dado2

Se dado1 é true,

dado1 || dado2

não avalia dado2

Operadores Compostos

$\ast =$ $/ =$ $\% =$ $+ =$ $-$ Realizam a operação com
 $=$ $\& =$ $\wedge =$ $| =$ um operando e fazem a
atribuição a ele mesmo.

Precedência de Operadores

Operadores	Observação
<code>++ -- + - ~ !</code>	Incremento, decremento, unários, negação lógica e inteira
<code>* / %</code>	Multiplicação, divisão e resto
<code>+ -</code>	Adição e subtração
<code>>> >>> <<</code>	Deslocamentos aritmético e lógicos
<code>< > <= >= isinstance</code>	Relacionais e comparação de classe
<code>== !=</code>	Relacionais de igualdade e desigualdade
<code>&</code>	AND
<code>^</code>	XOR
<code> </code>	OR
<code>&&</code>	AND de curto-circuito
<code> </code>	OR de curto-circuito
<code>?:</code>	Operador ternário

Operador Ternário

condição ? expr1 : expr2

é uma expressão do tipo
boolean que é avaliada

resultado caso
seja **true**

resultado caso
seja **false**

```
int r, a=10, b=45;  
boolean x=true, y=false;
```

...

```
r = (x & y) ? (a-b) : (a+b);
```