

Linguagem JavaScript

JavaScript não é Java!

JavaScript (JS)

Linguagem de scripting,
originalmente desenhada
para ser executada no
browser de internet como
forma de permitir construir
páginas dinâmicas.





Utilizações?

- Websites
- Aplicações Web
- Apresentações
- Aplicações de Servidor
- Servidores Web
- Jogos
- Arte
- Aplicações de SmartWatch
- Aplicações Móveis
- Robots voadores!

Sintaxe

Regras e normas a seguir na criação de declarações, instruções e comentários e que definem a forma de construção de programas numa determinada linguagem

Sintaxe da linguagem JavaScript

- Os elementos que compõem a sintaxe da linguagem JavaScript são
 - Comentários
 - Declarações
 - Instruções
 - Símbolos
 - Literais ou valores

Comentários

São úteis para os programadores como documentação mas são ignorados pelo computador

```
// este é um comentário de linha
```

```
/*
```

```
Este é um comentário de múltiplas linhas (bloco)
```

```
*/
```

Declarações

Permitem reservar memória para o armazenamento das estruturas de dados e blocos de código

```
var variavelInteira = 10;
```

```
var variavelString = "abc";
```

```
function myFunction(param1, param2)  
{  
  
}
```


Instruções

Indicam ao computador o que deve efetuar; As instruções são separadas por ponto e vírgula (;)

```
console.log("Hello world!");
```

Símbolos

Incluem os diversos operadores e sinais de pontuação

Têm significado definido e não podem ser utilizados para outro fim

Ex: . (ponto) ; (ponto e virgula)

Operadores

Ex: + - * = == !=

Literais ou Valores

Dados explícitos a serem manipulados pelos programas nas operações

Números inteiros: **-17** e **255**

Decimais: **3.1415**

Cadeias de caracteres: **“Bom dia!”**, **“Boa tarde”**

Lógicos: **true** , **false**

Ausência de valor: **null**

Identificadores

Palavras “inventadas” pelo programador para designar variáveis, constantes, funções ou outras entidades que descrevem o funcionamento do programa

`numAlunos`

`MAX_ALUNOS`

`calcular`

`Carro`

Identificadores

- Não fazendo parte da linguagem base, não têm qualquer significado predefinido no contexto da aplicação
- Devemos sempre obedecer à convenção em uso para facilitar a leitura e compreensão do código
- Devem ser o mais explícitas possível de acordo com o objetivo do programa

Palavras Reservadas

Palavras utilizadas na definição da linguagem com significado próprio.
Têm significado definido e não podem ser utilizados para outro fim

Palavras reservadas

<code>abstract</code>	<code>double</code>	<code>in</code>	<code>super*</code>
<code>arguments</code>	<code>else</code>	<code>instanceof</code>	<code>switch</code>
<code>await*</code>	<code>enum*</code>	<code>int</code>	<code>synchronized</code>
<code>boolean</code>	<code>eval</code>	<code>interface</code>	<code>this</code>
<code>break</code>	<code>export*</code>	<code>let*</code>	<code>throw</code>
<code>byte</code>	<code>extends*</code>	<code>long</code>	<code>throws</code>
<code>case</code>	<code>false</code>	<code>native</code>	<code>transient</code>
<code>catch</code>	<code>final</code>	<code>new</code>	<code>true</code>
<code>char</code>	<code>finally</code>	<code>null</code>	<code>try</code>
<code>class*</code>	<code>float</code>	<code>package</code>	<code>typeof</code>
<code>const</code>	<code>for</code>	<code>private</code>	<code>var</code>
<code>continue</code>	<code>function</code>	<code>protected</code>	<code>void</code>
<code>debugger</code>	<code>goto</code>	<code>public</code>	<code>volatile</code>
<code>default</code>	<code>if</code>	<code>return</code>	<code>while</code>
<code>delete</code>	<code>implements</code>	<code>short</code>	<code>with</code>
<code>do</code>	<code>import*</code>	<code>static</code>	<code>yield</code>

* só fazem parte dos
novos standards
ECMAScript 5 and 6

Regras de Identificadores

- não pode ser uma palavra-reservada (palavra-chave);
- não pode ser **true** nem **false** - literais que representam os tipos lógicos (booleanos);
- não pode ser **null** - literal que representa o tipo nulo;
- não pode conter espaços em brancos ou outros caracteres de formatação;
- não podem ser iniciados por dígito de 0 a 9

Regras de Identificadores

- deve ser a combinação de uma ou mais letras e dígitos UNICODE-16. Por exemplo, no alfabeto latino, teríamos:
 - letras de A .. Z;
 - letras de a .. z;
 - sublinha (underscore) _ ;
 - cifrão \$;
 - dígitos de 0 a 9.
- Observação 01: caracteres compostos (acentuados) não são interpretados igualmente aos não compostos (não acentuados). Por exemplo, História e Historia não são o mesmo identificador.
 - Boa prática é não usar
- Observação 02: letras maiúsculas e minúsculas diferenciam os identificadores, ou seja, **a** é um identificador diferente de **A**, **Historia** é diferente de **historia**, etc.

Convenções de escrita do código

Ajudam na clareza de escrita, leitura e compreensão do código

Convenção de escrita do código

- Identificadores devem ser escritos em **camelCase**
 - Primeira letra da primeira palavra em minúscula e primeira letra das palavras seguintes em maiúscula
 - Exemplos: **variavelString**
- Só devem conter letras e o underscore (_)
- Deve evitar-se os números, mas pode utilizar-se, desde que nunca sejam o primeiro carácter
- O ficheiros devem ter extensão **.js**

Variáveis

“Caixas” de memória onde são guardados os Inputs e os resultados dos processamentos e que se alteram durante a execução do programa

Variáveis

```
var nomeDaVariavel = 123;
```

```
var nomeDaVariavel = "123";
```

Tipos de dados

Define o espaço de memória a ocupar, a gama de valores representar e operações possíveis de realizar de uma determinada variável ou constante

Tipos de dados primários

- Números Inteiros: **1** , **-200** , **5689**
- Números Reais: **1.23** , **-200.20** , **5689.012**
- Cadeias de caracteres (strings): **“1”** , **“abca##”** , **“123 ab”**
- Valores lógicos: **true** , **false**
- Ausência de valor: **null**

Expressões

Sequência de operadores e valores.

Quando calculadas, podem dar origem a novos valores.

Expressões

- As expressões são compostas pelos operadores e os valores
- Existem 3 tipos de expressões
 - Atribuição
 - Aritméticas
 - Lógicas

Expressões Aritméticas

Com recurso aos operadores aritméticos, permite efetuar cálculos matemáticos

```
( variavel * 123 ) / 30
```

Operadores de Atribuição

Operador	Operação
=	Atribuição
+=	Soma com atribuição
-=	Subtração com atribuição
*=	Multiplicação com atribuição
/=	Divisão com atribuição
%=	Resto da divisão com atribuição

Operadores Aritméticos

Operador	Uso	Operação
*	op1 * op2	Multiplicação
/	op1 / op2	Divisão
%	op1 % op2	Resto da divisão
+	op1 + op2	Adição
-	op1 - op2	Subtracção
**	op1 ** op2	Exponentes

Operadores de incremento/decremento

pré-operador

operação feita antes da afetação

- Incrementar (aumentar) em 1

`++ var`

pós-operador

operação feita depois da afetação

- Incrementar (aumentar) em 1

`var ++`

`variavel = variavel + 1`

- Decrementar (diminuir) em 1

`-- var`

- Decrementar (diminuir) em 1

`var --`

`variavel = variavel - 1`

Operadores de incremento/decremento

código

```
var a = 21;
var c = 0;
// Valor não é incrementado antes da atribuição
console.log("Linha 0 valor de entrada de é :", a);
c = a++;
console.log("Linha 2 c = a++ valor de a++ é :", c);
// o valor de a já foi incrementado
console.log("Linha 2 valor de a++ é :", a);
console.log("Linha 3 valor de entrada de a é :", a);
// o valor de a já foi incrementado antes da atribuição
c = ++a;
console.log("Linha 4 c = ++a; valor de a++ é :", c);
```

resultado

Linha 0		valor de entrada de é : 21
Linha 2	c = a++	valor de a++ é : 21
Linha 2		valor de a++ é : 22
Linha 3		valor de entrada de a é : 22
Linha 4	c = ++a;	valor de a++ é : 23

Expressões Lógicas

Com recurso aos operadores lógicos e relacionais, permitem efetuar comparações entre valores

O resultado da avaliação da expressão é **true** ou **false**

```
variavel = 123
```

```
variavel >= 123
```

Operadores relacionais

Operador	Uso	Operação
>	op1 > op2	Maior que
<	op1 < op2	Menor que
>=	op1 >= op2	Maior ou igual que
<=	op1 <= op2	Menor ou igual que
==	op1 == op2	Igual a
!=	op1 != op2	Diferente de

Igualdade estrita com `===` ou `!==`

- Compara dois valores para a igualdade (`===` ou `!==`).
- Nenhum valor é convertido implicitamente para algum outro valor antes de serem comparados.
- Se os valores têm tipos diferentes, os valores são considerados não-iguais.
- Se os valores têm o mesmo tipo e não são números, eles são considerados iguais, se tiverem o mesmo valor.
- Se ambos os valores são números, eles são considerados iguais se ambos não são **NaN** e são do mesmo valor, ou se um é `+0` e outro é `-0`.

Igualdade estrita com `===` ou `!==`

Operador `===`

```
var num = 0;
var obj = new String("0");
var str = "0";

console.log(num === num);           // true
console.log(obj === obj);           // true
console.log(str === str);           // true

console.log(num === obj);           // false
console.log(num === str);           // false
console.log(obj === str);           // false
console.log(null === undefined);    // false
console.log(obj === null);          // false
console.log(obj === undefined);     // false
```

Operador `!==`

```
var num = 0;
var obj = new String("0");
var str = "0";

console.log(num !== num);           // false
console.log(obj !== obj);           // false
console.log(str !== str);           // false

console.log(num !== obj);           // true
console.log(num !== str);           // true
console.log(obj !== str);           // true
console.log(null !== undefined);    // true
console.log(obj !== null);          // true
console.log(obj !== undefined);     // true
```

Igualdade ampla == ou !=

- Compara dois para a igualdade, *após converter ambos os valores para um tipo comum*.
- Após as conversões (um ou ambos os lados podem sofrer conversões), a comparação de igualdade final é realizada exatamente como `===` executa.
- Igualdade ampla é simétrica:
 - `A == B` sempre tem semântica idêntica à `B == A` para quaisquer valores de A e B.

Igualdade ampla com `==` ou `!=`

Operador `==`

```
var num = 0;
var obj = new String("0");
var str = "0";

console.log(num == num);           // true
console.log(obj == obj);           // true
console.log(str == str);           // true

console.log(num == obj);           // true
console.log(num == str);           // true
console.log(obj == str);           // true
console.log(null == undefined);    // true

// ambos false , excepto casos raros
console.log(obj == null);
console.log(obj == undefined);
```

Operador `!=`

```
var num = 0;
var obj = new String("0");
var str = "0";

console.log(num != num);           // false
console.log(obj != obj);           // false
console.log(str != str);           // false

console.log(num != obj);           // false
console.log(num != str);           // false
console.log(obj != str);           // false
console.log(null != undefined);    // false

// ambos true, excepto casos raros
console.log(obj != null);
console.log(obj != undefined);
```

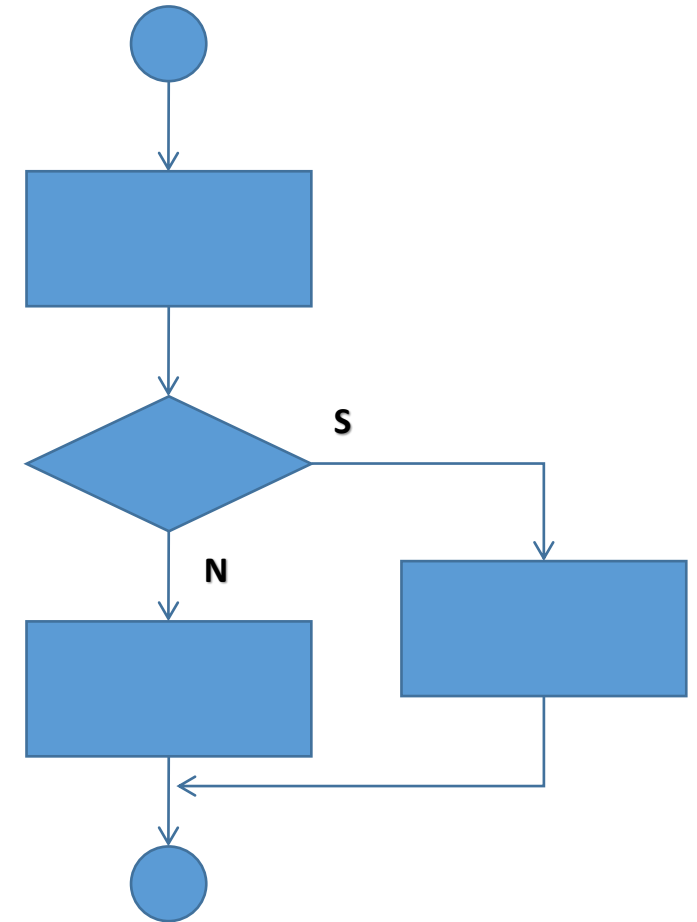
Operadores lógicos

Operador	Uso	Operação
&&	op1 && op2	Conjunção (E - AND)
 	op1 op2	Disjunção (OU - OR)
!	! op1	Negação (NÃO - NOT)

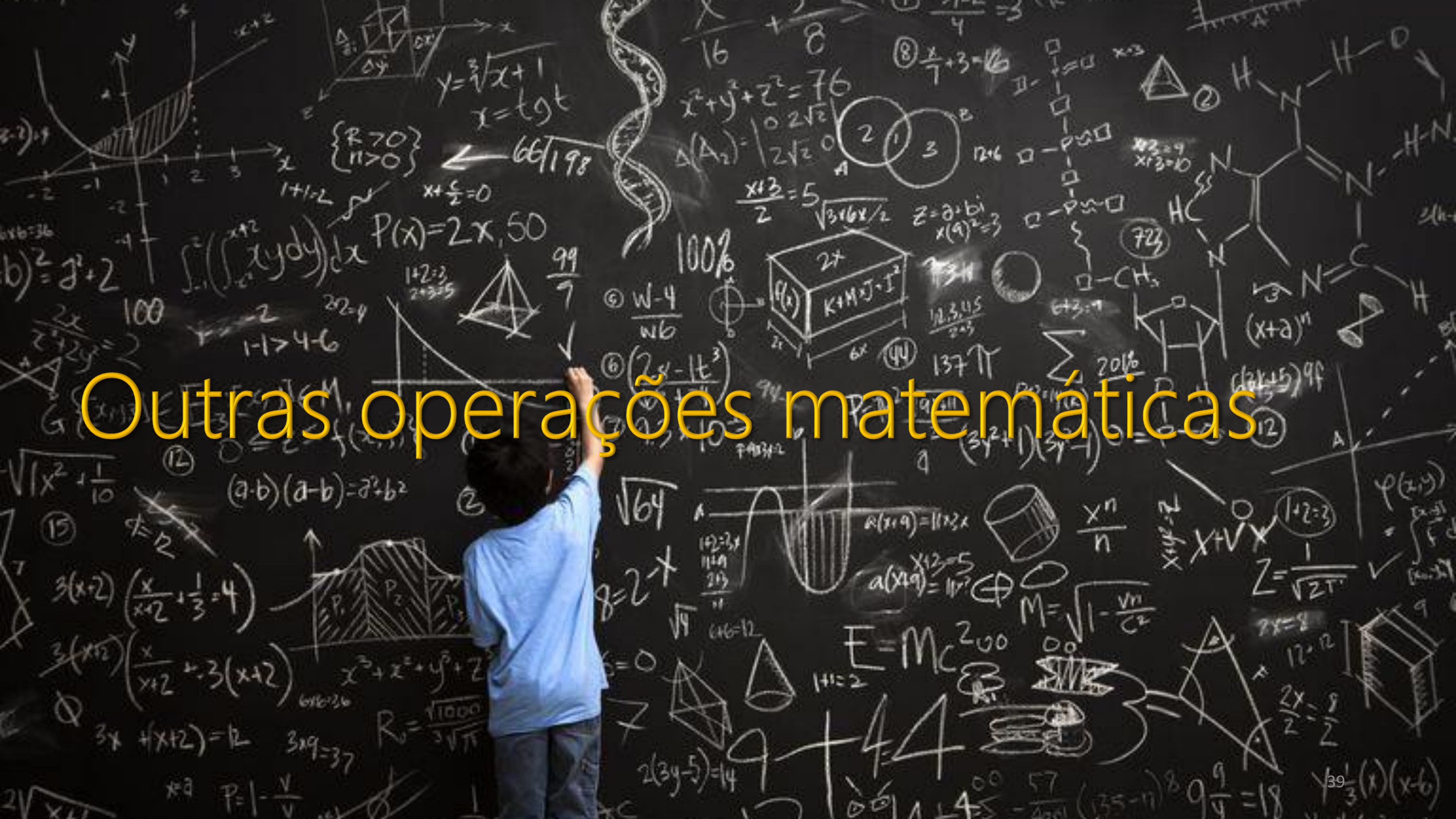
Operador ?: (operador trenário)

```
condicao ? true : false;
```

```
podeVotar = idade >= 18 ? "sim" : "não";
```



Outras operações matemáticas



Os Métodos da classe Math

Constantes

`Math.PI`

constante do valor PI

Métodos

`Math.abs(...);`

`Math.acos(...);`

`Math.asin(...);`

`Math.atan(...);`

`Math.ceil(...);`

`Math.cos(...);`

`Math.exp(...);`

`Math.floor(...);`

`Math.log(...);`

`Math.max(... , ...);`

`Math.min(... , ...);`

`Math.pow(... , ...);`

`Math.round(...);`

`Math.sin(...);`

`Math.sqrt(...);`

`Math.tan(...);`

Exemplos classe Math

```
console.log("Método abs(-30): ", Math.abs(-30) );
```

```
Método abs(-30): 30
```

```
console.log("Método sqrt(16): ", Math.sqrt(16));
```

```
Método sqrt(16): 4
```

```
console.log(Math.floor(Math.random()*100));
```

```
Número inteiro aleatório de 0 a 100: 68
```

Utilizando os tipos de dados

- Declaração de variáveis
- Atribuição
- Expressões aritméticas



Composição strings

```
var stringA = "String A";  
var stringB = "String B";
```

```
// concatenação
```

```
var stringFinal = stringA + " " + stringB;  
console.log("Por concatenação:", stringFinal);
```

```
//template ou interpolação de string
```

```
var stringTemplate = `${stringA} ${stringB}`;  
console.log("Por template:", stringTemplate);
```

Estruturas de decisão, selecção, escolha ou derivação

Estruturas de programação que permitem tomar decisões alterando o fluxo de execução do programa

Estruturas de decisão, selecção, escolha ou derivação

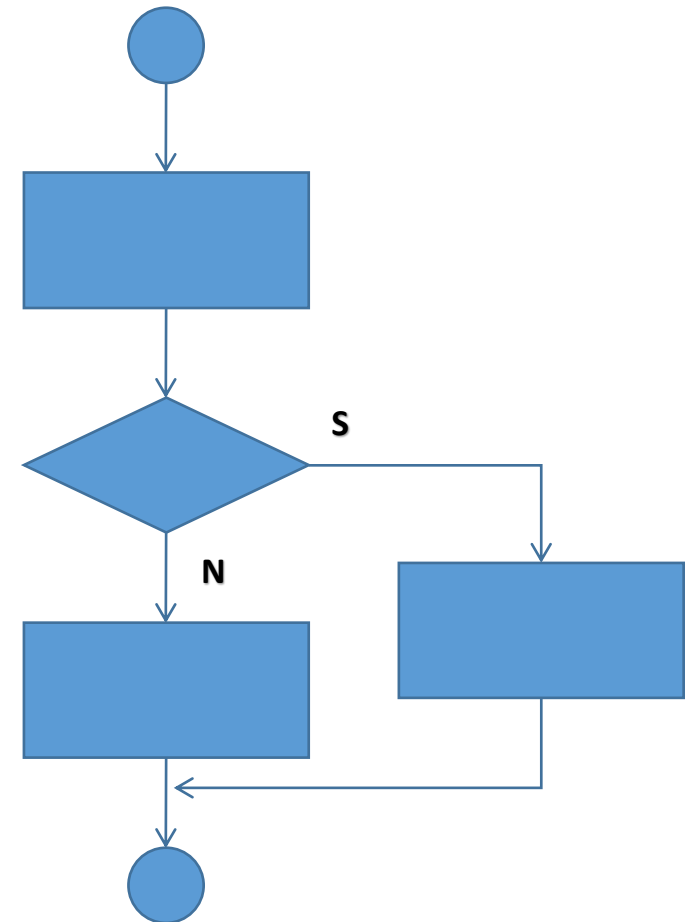
- Existem 2 tipos de estruturas de decisão
 - **if ... else**
 - **switch ... case**

Estruturas de decisão, selecção, escolha ou derivação

if ... else

```
if (condicao) {  
    instruções;  
}
```

```
if (condicao) {  
    instruções;  
} else {  
    instruções;  
}
```



Estruturas de decisão, selecção, escolha ou derivação

switch ... case

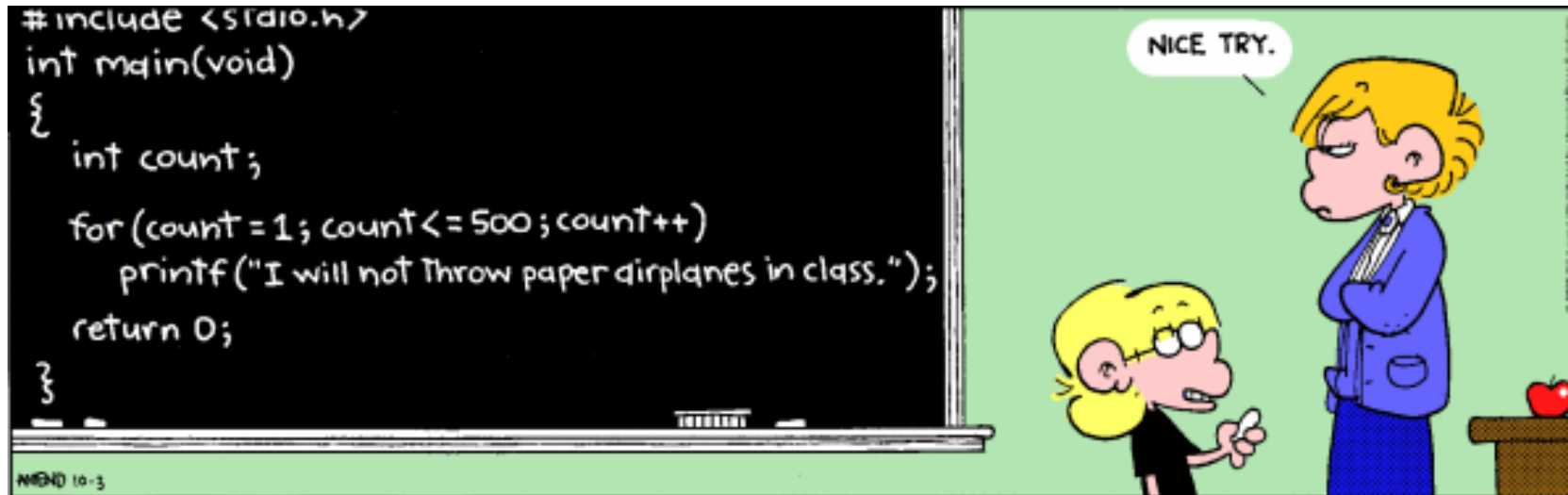
```
switch (expressao)
{
    case opcao1 : instrucao; break;

    case opcao2 :
        instrucao;
        break;

    default: por_defeito_faz_isto;
}
```

Estruturas de repetição, repetitivas ou ciclos

Estruturas de programação que permitem repetir blocos de instruções



Estruturas de repetição, repetitivas ou ciclos

- Existem 3 tipos de estruturas de repetição
 - **do ... while { }**
 - **while { }**
 - **for { }**

Estruturas de repetição, repetitivas ou ciclos

Ciclo **do ... while**

do

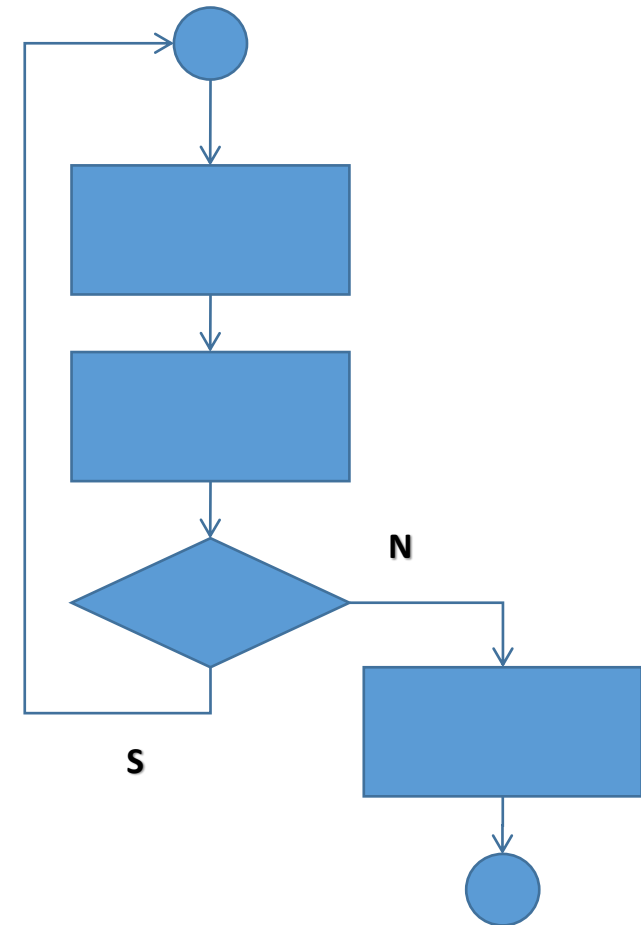
uma_instrução;

while (condicao)

do {

instruções;

} **while** (condicao)



Estruturas de repetição, repetitivas ou ciclos

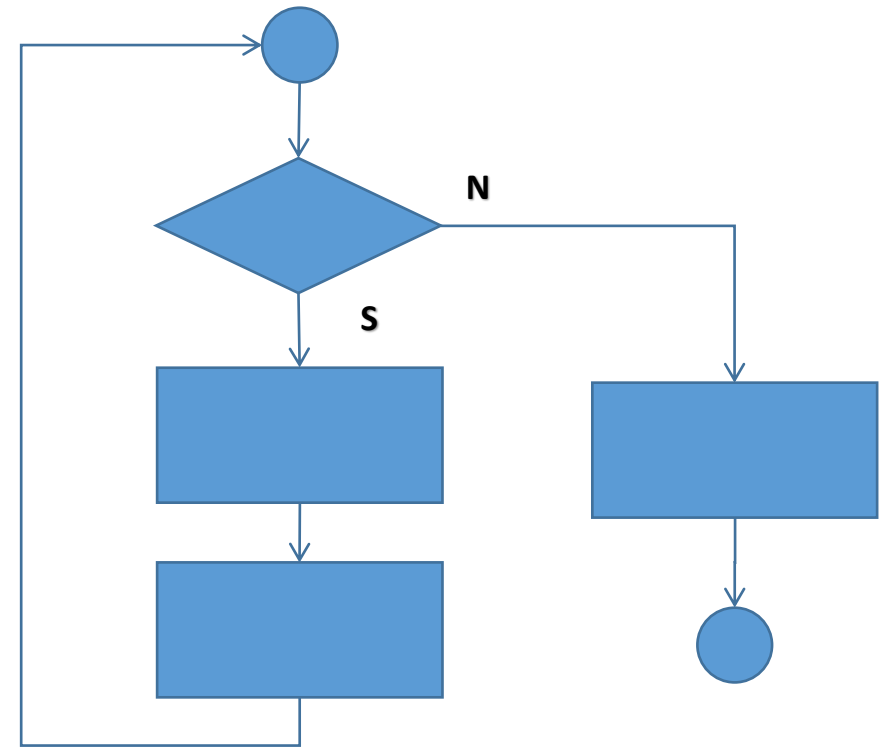
Ciclo **do ... while**

```
var i = 1;  
var sum = 0;  
do {  
    sum = sum + i; // aqui ficaria melhor sum += i  
    i++;  
}  
while (i <= 10)
```

Estruturas de repetição, repetitivas ou ciclos

Ciclo **while**

```
while (condicao) {  
    instruções;  
}
```



Estruturas de repetição, repetitivas ou ciclos

Ciclo **while**

```
var i = 1;  
while (i <= 10) {  
    console.log(i);  
    i++;  
}
```

Estruturas de repetição, repetitivas ou ciclos

Ciclo **for**

```
for (inicialização; condição_fim; incremento)  
    so_uma_instrução;
```

```
for (inicialização; condição_fim; incremento){  
    instruções;  
}
```

Estruturas de repetição, repetitivas ou ciclos

Ciclo **for**

```
for(var num = 1; num <= 10; num++){  
    console.log(num);  
}
```

```
var num = 1;  
for( ; num <= 10 ; ){  
    console.log(num);  
    num++;  
}
```

Ah, ciclo... que tanto te repetes... vamos parar?

Quebrar Ciclos

```
var num = 1;
```

```
for( ; num <= 10 ; ){  
    num++;  
    break;  
}
```


Funções

- São “subprogramas” que podem ser invocados e que realizam um conjunto de instruções definidas.
- São compostas pela identificação da função (***declaração*** ou ***assinatura***) e as sequências de instruções a serem executadas (***corpo da função***).
- Podemos “passar” valores “para dentro” da função mediante ***argumentos*** ou ***parâmetros*** e ***retornar*** ou ***devolver valores*** no final da sua execução

Funções

- A sintaxe geral da definição de funções é:

```
function nome([param[, param[, ... param]]) {  
    instruções  
}
```

→ palavra reservada

nome

O nome da função.

param

O nome de um argumento/parâmetro a ser passado para a função.
Uma função pode ter até 255 argumentos.


instruções

As declarações que compreendem o corpo da função (código)

Funções

Exemplo de uma função com parâmetros

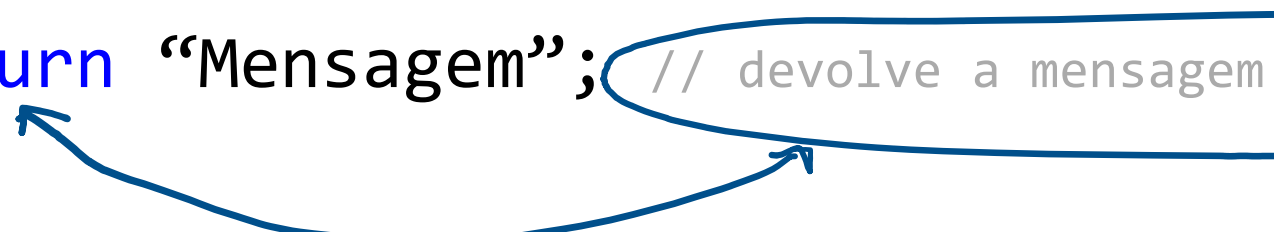
```
function funcaoSoma(p1, p2) {  
    return p1 + p2; // devolve a soma dos parâmetros p1 e p2  
}  
  
...  
// invocação/chamada da função  
console.log(funcaoSoma(10, 20))
```



Funções

Exemplo de uma função sem parâmetros

```
function funcaoMensagem() {  
    return "Mensagem"; // devolve a mensagem  
}  
  
...  
// invocação/chamada da função  
console.log(funcaoMensagem())
```



A blue oval highlights the return statement `return "Mensagem";` and the comment `// devolve a mensagem`. A blue arrow points from the end of the `return` statement to the opening parenthesis of `funcaoMensagem()` in the `console.log` call. Another blue arrow points from the closing parenthesis of `funcaoMensagem()` back to the `return` statement, indicating the flow of the returned value.

Arrays

- Conjunto de valores ordenados que são referenciados com um **nome** e um **índice**.
- Exemplo:
 - podemos ter um array com o nome *empregados* que contém nomes de funcionários indexados por seus números de funcionários. Então *empregado[1]* poderia ser o funcionário número 1, *empregado[2]* o funcionário número 2 e assim por diante.

Arrays

Exemplos de criação de arrays

```
var arr = new Array(elemento0, elemento1, ..., elementoN);
```



```
var arr = Array(elemento0, elemento1, ..., elementoN);
```



```
var arr = [elemento0, elemento1, ..., elementoN];
```



elemento0, elemento1, ..., elementoN

Lista de valores para os elementos do array

Arrays

```
var arr = new Array(elemento0, elemento1, ..., elementoN);
```

- *elemento0, elemento1, ..., elementoN*

- Lista de valores para os elementos do *array*
- Quando especificados definem o conteúdo do *array*
- O tamanho do *array* é definido pelo número de argumentos
- Os índices dos *arrays* começam em 0

Por exemplo, um array de 3 elementos ("primeiro", "segundo", "terceiro") os índices serão

array[0] = "primeiro"

array[1] = "segundo"

array[2] = "terceiro"

Arrays

Mais exemplos de criação de arrays e colocação de elementos no array

```
var emp = [3]; // definir o tamanho
```

```
emp[0] = "João António";
```

```
emp[1] = "Asdrubal Sotavento";
```

```
emp[2] = "Carlos Pimpinela";
```

```
// o tamanho é definido na declaração do array
```


Arrays

Mais exemplos de criação de arrays e colocação de elementos no array

```
var emp = []; // sem tamanho definido
```

```
emp[0] = "João António";
```

```
emp[1] = "Asdrubal Sotavento";
```

```
emp[2] = "Carlos Pimpinela";
```

```
// o tamanho é definido pelo número de elementos  
colocados
```

Arrays

Referenciar/aceder a um elemento

```
var myArray = ['Vento', 'Chuva', 'Fogo'];
```

```
console.log(myArray[0]); // escreve “Vento”
```

```
console.log(myArray[1]); // escreve “Chuva”
```

```
console.log(myArray[2]); // escreve “Fogo”
```

Arrays

Percorrer todos os elementos de um *array*

```
var cores = ["vermelho", "verde", "azul"];  
for (var i = 0; i < cores.length; i++)  
{  
    console.log(cores[i]);  
}
```

Número de elementos
do array

```
vermelho  
verde  
azul  
❏
```

Resultado
da execução

Percorrer um Array sequencialmente

Ciclo **for-in**

```
for (var item in array_ou_objeto)  
    so_uma_instrução;
```

```
for (var item in array_ou_objeto)  
{  
    instruções;  
}
```

Percorrer um Array sequencialmente

Ciclo **for in**

```
var carros = new Array("Audi", "BMW", "Fiat");
```

```
for (item in carros)
{
    console.log(`Marca: ${carros[item]}`);
}
```

Métodos da classe Array

`x.sort();`

ordena o array **x** por ordem crescente

`x.reverse();`

inverte a ordenação do array **x**

`x.push(item);`

adiciona o **item** ao array **x**

`x.pop();`

remove/retorna o último elemento do array **x** e remove-o

Recursos

<https://www.w3schools.com/js/default.asp>

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>

<https://javascript.info/>

<https://github.com/getify/You-Dont-Know-JS/>