



Instituto Politécnico
de Viana do Castelo

Javascript

António Lira Fernandes

Conteúdos



- Introdução
 - Parâmetros e Query strings
- Controlos
 - Text boxes e Text areas
 - Checkboxes e Radio Buttons
 - Labels, Menus e listas
 - Botões de reset e Grupo de controlos
 - Aplicação de estilos a formulários
- Envio de dados
 - URL-encoding
 - Parâmetros escondidos
 - Pedidos HTTP: GET vs. POST
 - Uploading de ficheiros
- Processamento de dados dos formulários
 - Arrays Associativos
 - Processamento de ficheiros carregados

Scripts



As linguagens de script são ferramentas utilizadas:

- para controlar um determinado programa ou aplicativo;
- para configurar ou instalar em sistemas operativo;
- em jogos para controlar as ações dos personagens.



Algumas linguagens de programação geralmente usadas como script são: ActionScript, JavaScript, Lua, PHP, Python, ShellScript, Ruby, VBScript.

Exemplo:

- Programa que soma as notas e que de seguida apresenta o resultado no ecrã.
- O script é a sequência de passos que o computador vai interpretar para somar e apresentar as notas no monitor.

Scripts

Java

Linguagem de **programação** orientada a objetos;

Cria aplicações que são executados em uma máquina virtual ou browser;

Precisa ser compilado;

É autónomo;

JAVA *isto* JAVASCRIPT



as HAM *isto* HAMSTER



Java ≠ JavaScript

JavaScript

Linguagem de **script** orientada a objetos;

Executado apenas num browser.

Não precisa de compilador

Não cria aplicativos autónomos

Está dentro do HTML e fornece níveis de interatividade para páginas da web

Javascript (js)



Javascript

É uma linguagem de **programação de alto nível**, dinâmica e interpretada.

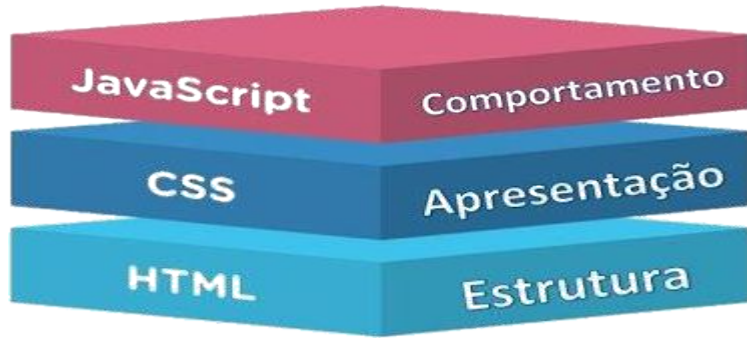
É amplamente utilizada para **adicionar interatividade e dinamismo às páginas web**, além de ser utilizada também em ambientes de desenvolvimento de servidor, como o Node.js.

Foi criado em 1995 é a linguagem de programação mais popular no desenvolvimento Web.

É suportada por todos os browsers

Como exemplos da boa utilização do JavaScript temos o Gmail, Google Maps e Google Docs.

Javascript (js)



Javascript (js)

Antes do JavaScript

- As páginas eram pouco ou nada interativas - **apresentavam conteúdo exatamente como fora criado**
- Existiam tecnologias específicas para a **criação de páginas no lado do servidor**
- Havia limitações na forma como o utilizador "consumia" o conteúdo.
- Só se **navegava por links** e eram **enviadas informações através de formulários**



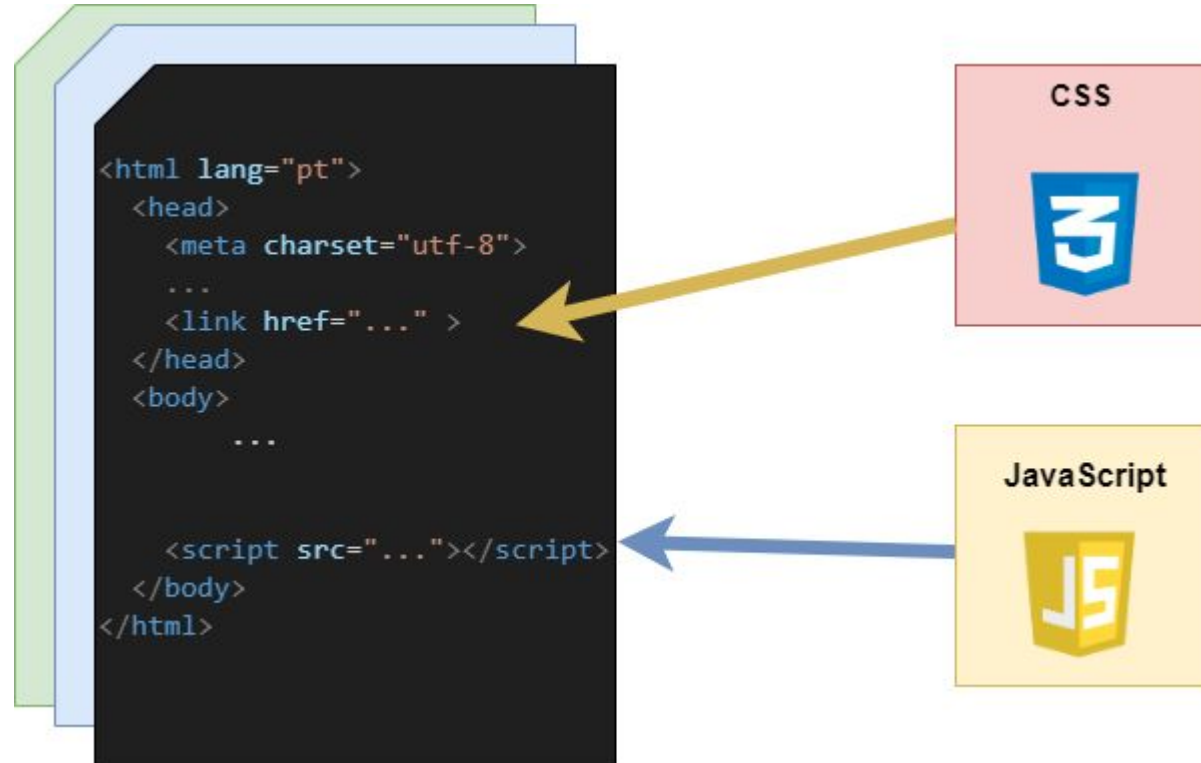
Javascript (js)_[1]

Depois do JavaScript

- Potenciar a uma **interação maior do utilizador** com as páginas.
- A Netscape criou o Livescript - linguagem simples que permitia a execução de scripts contidos nas páginas dentro do próprio browser.
- JS é a linguagem de programação que permite que as páginas da Web respondam à interação com o utilizador.



Javascript (js)



Onde Colocar?

Na tag <script> do HTML ou

Colocado num ficheiro próprio .js, fazendo o import do ficheiro externo.

Colocado no final da página

```
<script>  
document.getElementById("demo").innerHTML = "Hello JavaScript!";  
</script>
```

```
<script src="script.js"></script>
```

Onde Colocar?



```
2 <html>
3 <head>
4   <title>Web Notpad</title>
5   <link rel="stylesheet" type="text/css" href="estilo.css">
6 </head>
7 <body>
8   <h1>Web Notepad</h1>
9   <div class="header"></div>
10  <textarea onload="showDate()"></textarea>
11  <script type="text/javascript" src="script.js"></script>
12 </body>
13 </html>
```

Lado do Servidor vs Lado do Cliente^[6]

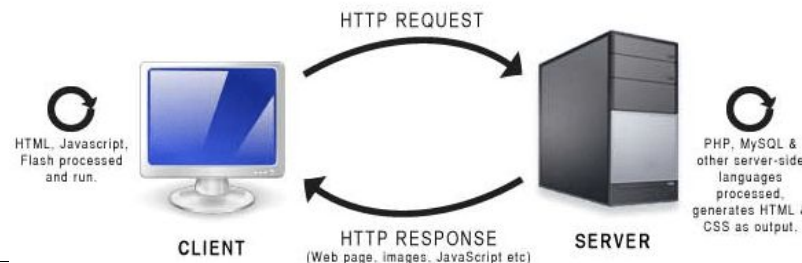


Server-side code - Corre no servidor antes da página ser enviada

Client-side code - Corre dentro do browser do cliente quando a página é carregada e visualizada

Porquê client-side programming?

- Porque há coisas que podem ser feitas de forma mais eficiente no cliente
- Para evitar sobrecarregar o servidor



Original vs Minified

A diferença está na compactação do código e no tamanho do ficheiro

```
/* Prototype JavaScript framework, version 1.7
 * (c) 2005-2010 Sam Stephenson
 *
 * Prototype is freely distributable under the terms of an MIT-style license.
 * For details, see the Prototype web site: http://www.prototypejs.org/
 *
 * -----*/
var Prototype = {
  Version: '1.7',
  Browser: (function() {
    var ua = navigator.userAgent;
    var isOpera = Object.prototype.toString.call(window.opera) == '[object Opera]';
    return (IE: !!window.attachEvent && !isOpera, Opera: isOpera, WebKit: ua.indexOf('AppleWebKit/') > -1,
    Gecko: ua.indexOf('Gecko') > -1 && ua.indexOf('KHTML') == -1, MobileSafari: /Apple.*Mobile/.test(ua))
  })(),
  BrowserFeatures: {
    XPath: !!document.evaluate,
```

Original

```
var Prototype={Version:'1.7',Browser:(function(){var ua=navigator.userAgent;var isOpera=Object.prototype.toString.call(window.opera)=="[object Opera]";return(IE:!!window.attachEvent&&!isOpera,Opera:isOpera,WebKit:ua.indexOf('AppleWebKit/')>-1,Gecko:ua.indexOf('Gecko')>-1&&ua.indexOf('KHTML')===-1,MobileSafari:/Apple.*Mobile/.test(ua)))})();BrowserFeatures:{XPath:!!document.evaluate,SpecificElementExtensions:(function(){if(typeof window.HTMLDivElement=="undefined")return true;var div=document.createElement('div'),form=document.createElement('form'),isSupported=false;if(div['__proto']&&(div['__proto']!=form['__proto']))isSupported=true;div=form=null;return isSupported;})();ScriptFragment:'<script[?]>{[\\S\\s]*?}</script>',JSONFilter:/^\\s*--secure-([\\S\\s]*)*\\/As$/;emptyFunction:function(){},K:function(x){return x}};if(Prototype.Browser.MobileSafari)Prototype.BrowserFeatures.SpecificElementExtensions=false;var Abstract={};var Try=(these:function(){var returnValue;for(var i=0,length=arguments.length;i<length;i++){var lambda=arguments[i];try{returnValue=lambda();break;}catch(e){}}return returnValue;})();var Class=function(){var IS_DONTENUM_BUGGY=(function(){for(var p in toString:1){if(p=="toString")return false;}return true;})();function subclass(){function create(){var parent=null,properties=$A(arguments);if(Object.isFunction(properties[0]))parent=properties.shift();function klass(){this.initialize.apply(this,arguments);}Object.extend(klass,Class.Methods);klass.superclass=parent;klass.subclasses=[];if(parent){(subclass.prototype=parent.prototype).klass.prototype=new subclass;parent.subclasses.push(klass);}for(var i=0,length=properties.length;i<length;i++)klass.addMethods(properties[i]);if(!klass.prototype.initialize)klass.prototype.initialize=Prototype.emptyFunction;return klass;}}
```

Minified



```
.entry-content p {
font-size: 14px !important;
}

.entry-content ul li {
font-size: 14px !important;
}

.product_item p a {
color: #000;
padding: 10px 0px 0px 0;
margin-bottom: 5px;
```

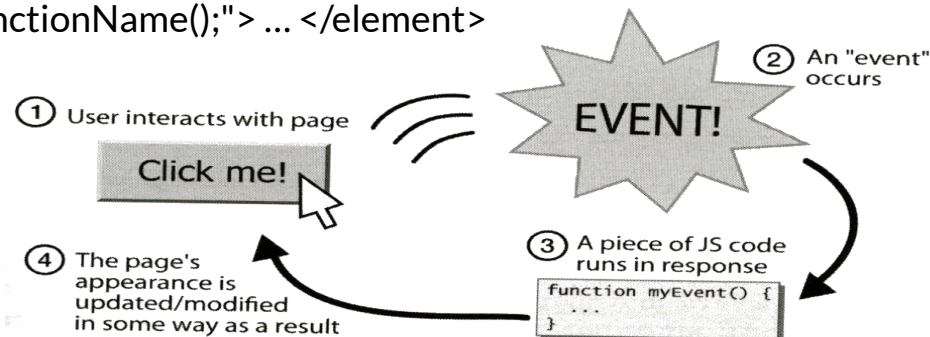
Minify



```
.entry-content p,.entry-content ul li
{font-size:14px!important}.product_item
p a{color:#000;padding:10px 0 0;margin-
bottom:5px;border-bottom:none}
```

Event-Driven Programming

- Modelo de programação do JavaScript usa
 - O fluxo do programa é determinado pelo utilizador!
 - O programa espera por um evento
 - Para responder a um evento escreve-se uma função JavaScript (event handler) e associa-se ao controlo
- TODOS os elementos HTML têm atributos de evento especiais que representam ações feitas sobre o elemento
 - o nome desses **atributos começa por on** (ex.: **onclick**)
 - `<element onevent = "functionName();"> ... </element>`



Exemplo de Programa

- Botão que reage quando clicado pelo utilizador

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript example</title>
  <script type="text/javascript" src="please.js"></script>
</head>
<body>
  <div>
    <button onclick="sayMagicWord();">Say the Magic Word</button>
  </div>
</body>
</html>
```

please.js

```
function sayMagicWord() { alert("PLEASE!"); }
```

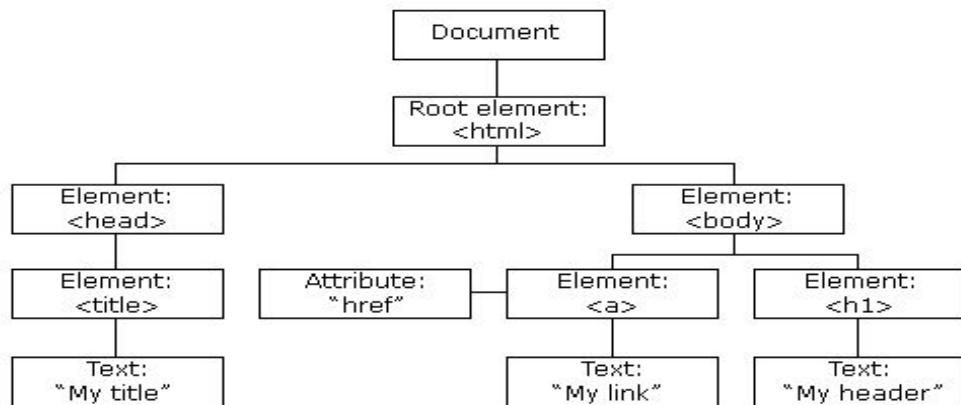

DOM (Document Object Model)[1][3]

JavaScript tem um conjunto de objetos e funções que permitem aceder e modificar o conteúdo de uma página

Cada tag da página (parágrafo, heading, list...) é representada como um objeto

Este conjunto de objetos é chamado DOM (conjunto de objetos JavaScript representando o conteúdo da página HTML corrente)

A árvore de objetos HTML DOM



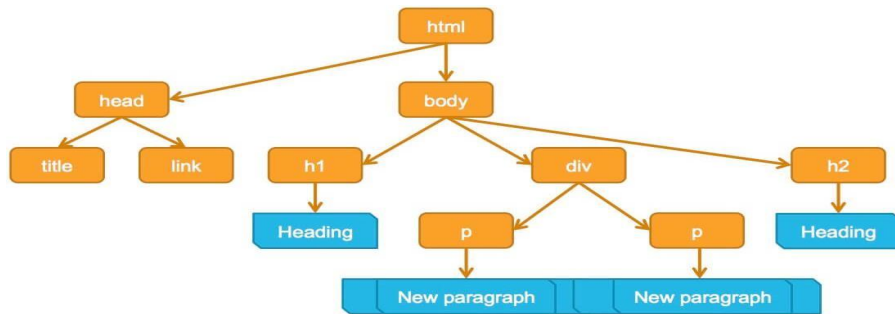
DOM (Document Object Model)[1]

Objetos DOM podem ser acedidos de várias formas...

- a mais simples é usar o ID do elemento!

O DOM expõe alguns objetos globais que permitem aceder aos elementos

- Um deles é o objeto document - Representa todo o documento HTML
- É usado para aceder a todos os outros objetos DOM
- Tem um método ***getElementById()*** que aceita id (string) e retorna o objeto DOM para o elemento HTML com esse id (ou null se o elemento não for encontrado)



DOM (Document Object Model)[1]



```
// Procurar um elemento pelo atributo id
document.getElementById("idElemento");

// Procurar elementos pelo tag
document.getElementsByTagName("tagElemento");

// Procurar elementos pelo atributo name
document.getElementsByName("nameElemento");

// Procurar elementos pelo atributo name
document.getElementsByClassName("nomeClass");
```

Exemplo



```
<button onclick="sayMagicWord();">Diz as palavras mágicas!</button>
<p id="disse">O que dizes?</p>

<script>
  function sayMagicWord() {
    var paragrafo = document.getElementById("disse");
    paragrafo.innerHTML = "POR FAVOR!";
  }
</script>
```

Exemplo



```
<p id="demo">0 que dizes?</p>
```

```
<script>
```

```
  var x = 5;
```

```
  var y = 6;
```

```
  var z = x + y;  document.getElementById("demo").innerHTML = z;
```

```
</script>
```

Exemplo



```
<ul>
  <li>aaa</li>
  <li>aaa</li>
  <li>aaa</li>
</ul>

<script>
  // Seleciona todos os elementos <p>
  var paragrafos = document.getElementsByTagName("li");

  // Itera sobre a coleção de elementos <p> e modifica seu conteúdo
  for (var i = 0; i < paragrafos.length; i++) {
    paragrafos[i].innerHTML = i+1 + " - Este parágrafo foi modificado usando js!";
  }
</script>
```

Variáveis e tipos de dados



Podemos declarar variáveis usando as palavras-chave ***var***, ***let*** ou ***const***.

Existem diferentes tipos de dados em JavaScript, incluindo números, strings, booleanos, objetos, arrays, entre outros.

```
var nome = "João";  
  
let idade = 30;  
  
const PI = 3.14;
```

Declarar variáveis var vs let



Âmbito:

- Variáveis declaradas com **var** têm **âmbito de função ou âmbito global**, dependendo de **onde são declaradas**. Significa que são visíveis em toda a função em que foram declaradas ou globalmente se estiverem fora de qualquer função.
- Variáveis declaradas com **let** têm **âmbito de bloco**. Isso significa que elas são visíveis apenas dentro do bloco onde foram declaradas, como um **bloco if**, **for**, **while**, **switch**, etc.

Declarar variáveis var vs let



Hoisting:

- Variáveis declaradas com **var** são **içadas (hoisted)** para o **topo de sua função** ou **âmbito global** durante a fase de compilação, o que significa que são **acessíveis** mesmo antes de sua declaração no código. No entanto, o valor atribuído não é içado.
- Variáveis declaradas com **let** não são içadas para o topo de seu bloco de âmbito. **Tentar aceder antes de sua declaração resultará num erro de referência** **ReferenceError**

Declarar variáveis var vs let



Reatribuição:

- Variáveis declaradas com **var** podem ser reatribuídas (alterar o valor) em qualquer lugar dentro de seu âmbito.
- Variáveis declaradas com **let** também podem ser reatribuídas, mas não podem ser redeclaradas dentro do mesmo âmbito.

```
var a = 10;

var a = 20; // Não gera erro, redeclaração permitida com var

let b = 30;

// let b = 40; // Gera erro, redeclaração não permitida com let

b = 40; // Atribuição permitida, mas não redeclaração
```

Tipos de Dados



8 tipos fundamentais

Tipo	Descrição	Exemplo
Number	Inteiro ou real	42, -17, 3.14
Boolean	Valor lógico	true, false
String	String de texto	"Ola", 'como estás'
Array	Lista indexada de elementos	[12, 17, -5, 42]
Object	Entidade contendo dados e comportamento	{name: "Marty", age: 12}
function	Grupo de instruções para executar	function message(){alert("Olá mundo");}
undefined	Variável declarada mas não inicializada	undefined

Exemplo



```
<script>
  var TestVar;
  alert(TestVar); //shows undefined
  alert(typeof TestVar); //shows undefined
  var TestVar = null;
  alert(TestVar); //shows null
  alert(typeof TestVar); //shows object
</script>
```

undefined \neq null

Tipos de Dados

Para verificar que determinado objeto existe deve utilizar o *typeof*.

Type of val	Result
Undefined	"undefined"
Null	"object"
Boolean	"boolean"
Number	"number"
String	"string"
Object (native and does not implement <code>[[Call]]</code>)	"object"
Object (native or host and does implement <code>[[Call]]</code>)	"function"
Object (host and does not implement <code>[[Call]]</code>)	Implementation-defined except may not be "undefined", "boolean", "number", or "string".

undefined

null

has not been assigned

could be assigned

`typeof undefined`

`typeof object`

`undefined == null //true`

`undefined === null //false`



Números

A divisão é sempre real, mesmo que os operando sejam inteiros

Constantes existentes

Expressão JavaScript

É um valor ou um conjunto de operações que calculam um valor

Tipo	Descrição
Number.MAX_VALUE	Maior valor finito possível
Number.MIN_VALUE	Menor valor finito possível
Number.NaN	Not a Number
Number.NEGATIVE_INFINITY	
Number.POSITIVE_INFINITY	

Variáveis



Nomes das variáveis são case-sensitive e começam por letra ou por _

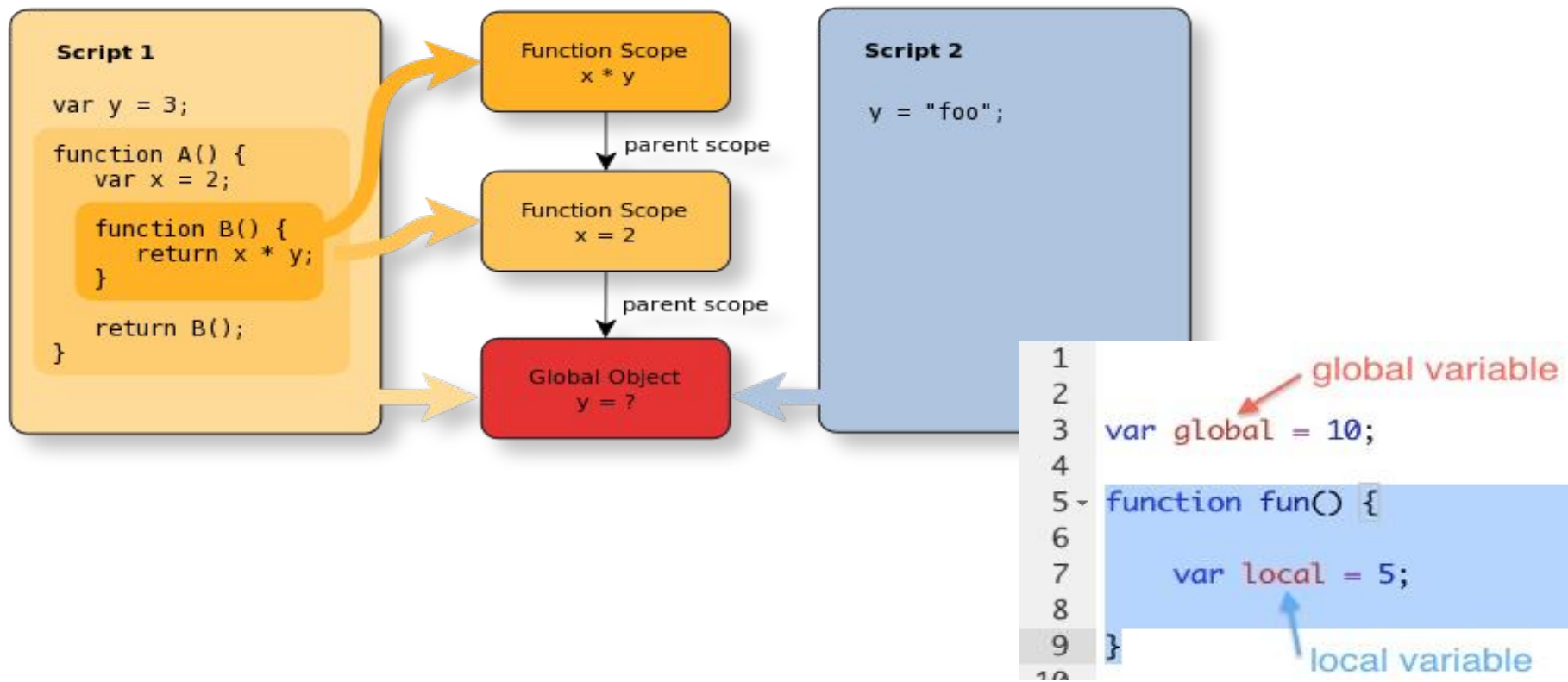
Declarações das variáveis:

- começam por var... embora a var possa ser omitida (declaração implícita) !!!
- **var numero = 42; var name = "pw";**
- Não se especificam os tipos.

Operadores de modificação-atribuição

Operador	Exemplo	Mesmo que
++	x++;	x = x + 1;
--	x--;	x = x - 1;
+=	x += y;	x = x + y;
-=	x -= y;	x = x - y;
*=	x *= y;	x = x * y;
/=	x /= y;	x = x / y;
%=	x %= y;	x = x % y;

Âmbito das Variáveis



Âmbito das Variáveis

```
var doo = "a";  
function foo(){
```



Variable doo created in global scope.

```
    var doo= "doo";  
    console.log(doo);//print  
}
```



Variable doo created in scope of function foo and has value set to doo inside function foo

```
foo();  
console.log(doo);//print a
```



Variable doo of global scope

```
<script>  
    //JavaScript has two levels of scope: Global and funct  
    var age = 55;  
  
    alert("Global age: " + age);  
  
    function loopFunction() {  
        var age;  
        //Age scoped to function  
        for (age = 0; age < 5; age++) {  
        }  
        alert("Age in function: " + age)  
    }  
  
    loopFunction();
```

Global Scope

Function Scope

Strings

Conversão de strings para integer e float

- ***parseInt(value) e parseFloat(value):**
- Retorna um **integer**
- Retorna um **NaN** -se o valor é uma string não numérica

Nome	Sequência de escape
\	\'
"	\"
\	\\
newline	\n
tab	\t

Strings - Propriedades e métodos

Propriedades:

- length

Alguns métodos ...

Nome	Sequência de escape
charAt(index)	Retorna o caracter do índice especificado
charCodeAt(index)	Retorna o valor ASCII do caracter
String.fromCharCode(value)	Converte o valor ASCII numa string
indexOf(searchSubStr) indexOf(searchSubStr, fromIndex)	Procura a substring na string e retorna o índice onde se inicia a string especificada ou -1 se não encontrar
split(delimiter) split(delimiter, howMany)	Divide a string num array de strings usando um delimitador
substring(start) substring(start, stop)	Retorna os caracteres entre dois índices especificados
toLowerCase()	Retorna a string em minúsculas
toUpperCase()	Retorna a string em maiúsculas

Strings

```
var str = "Hello World!";  
var n = str.length;           // n=12
```

```
var str = "Hello world!";  
var res = str.substr(1, 4);    // res="ello"
```

```
var str = "How are you doing today?";  
var res = str.split(" ");      //  
res=[How,are,you,doing,today?]
```

```
var str = "Hello world, welcome to the universe.";  
var n = str.indexOf("welcome"); //n=13
```



Variáveis [5]

Nunca se deve declarar números, string ou objetos booleanos de modo a evitar erros na comparação.

Use operador === para fazer comparações

O operador de comparação == converte os objetos para tipos correspondentes antes da comparação.

O operador === força a comparação de valores e tipo:

```
var x = "John";  
var y = new String("John");  
(x === y) // is false because x is a string and y is an object.
```



Operadores



Suporta uma variedade de operadores, tais como: aritméticos (+, -, *, /), de comparação (==, ===, !=, !==), lógicos (&&, ||, !), entre outros.

```
<script>
  var idade = 10 + 5;
  var eMaior = idade > 18;
  var resultado = (10 === "10") || (5 !== 3);

  alert(resultado);
</script>
```

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo

Objeto *Math*

Contém propriedades e funções para manipulação de constantes e funções

Constantes e funções mais comuns:

Exemplo:

Retornar um número aleatório entre 1 e 10

```
var rand= Math.random()*10 +1;
```

Constantes

Math.E

Math.LN2

Math.PI

Funções

Math.abs(x)

Math.ceil(x)
Math.floor(x)

Math.cos(x)
Math.sin(x)
Math.tan(x)

Math.log(x)

Math.max(x, y)
Math.min(x, y)

Math.pow(base, exponent)

Math.random()

Math.round(x)

Math.sqrt(x)

Lógica Booleana

Operadores de comparação

Se os operandos não forem do mesmo tipo o JavaScript tenta conversão automática

Se os tipos não forem comparáveis o resultado é sempre **false**

Operador	Descrição	Exemplo	Resultado
> <	Maior que Menor que	5 < 10.0	true
>= <=	Maior ou igual Menor ou igual	42 >= "hello"	false
== ===	Igual (só valores; ignora tipos) Igual (testa valores e tipos)	7 == "7" 7 === "7"	true false
!= !==	Diferente (só valores; ignora tipos) Diferente (testa valores e tipos)	7 != "7.0" 7 !== "7.0"	false true



Valores Booleanos

Avalia qualquer valor primitivo como booleano

Valores primitivos que sejam avaliados como valores booleanos são chamados

falsey/truthy values

Exemplo de **falsey values**: 0, NaN, "", null, undefined

Valores Booleanos

Não esquecer de testar se uma variável é **undefined**... e

Não esquecer de testar se uma variável tem valor (não é **null**)... antes de a usar!

```
if (myVariable !== null && myVariable !== undefined){  
    ...  
}
```

OU

```
if (myVariable){  
    ...  
}
```

Operadores Lógicos

O JavaScript faz curto-circuito na avaliação das expressões:

- se o operando esquerdo de um **&&** for **false**
- o operando esquerdo de um **||** for **true**
- ...então o resto da expressão já não é avaliado

Operador	Descrição	Exemplo
valor1 && valor2	AND	true se ambos verdadeiros
valor1 valor2	OR	true se pelo menos um é verdadeiro
!valor	NOT	true se o valor for falso

Comentários

// comentário até ao fim da linha

/*

Linhas de comentário

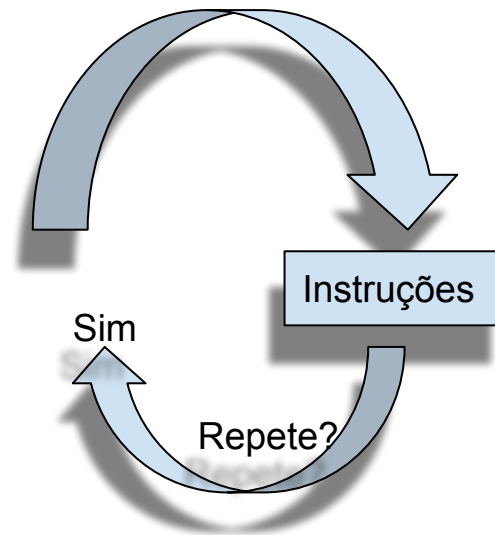
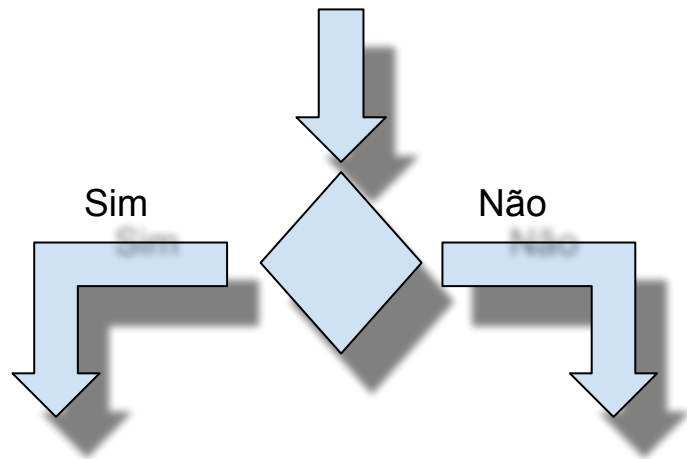
Linhas de comentário

*/

Estruturas de controle



Possui estruturas de controle de fluxo como **if, else, else if, switch, while, do-while, for**, entre outros.



Estruturas de controle - if



```
var idade = 18;

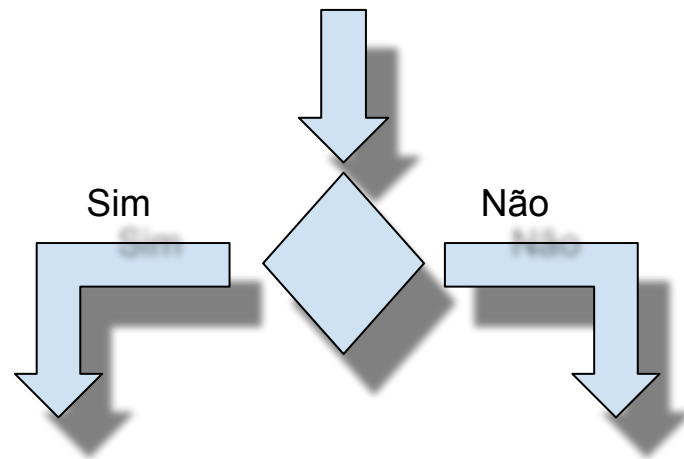
if (idade >= 18) {

    console.log("Você é maior de idade.");

} else {

    console.log("Você é menor de idade.");

}
```



Estruturas de controle - else if



```
var hora = 14;

if (hora < 12) {

    console.log("Bom dia!");

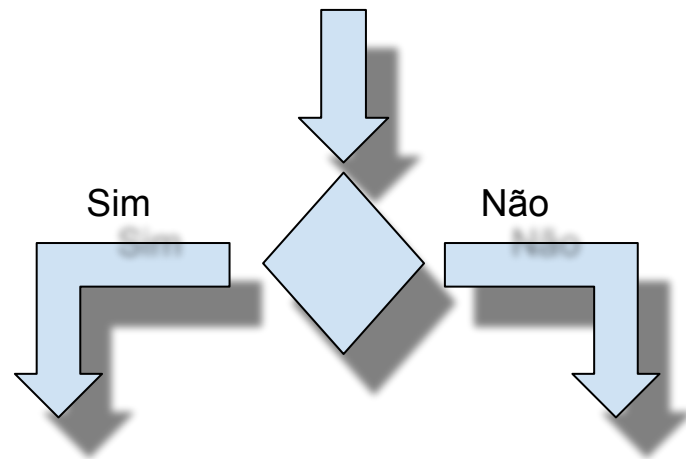
} else if (hora < 18) {

    console.log("Boa tarde!");

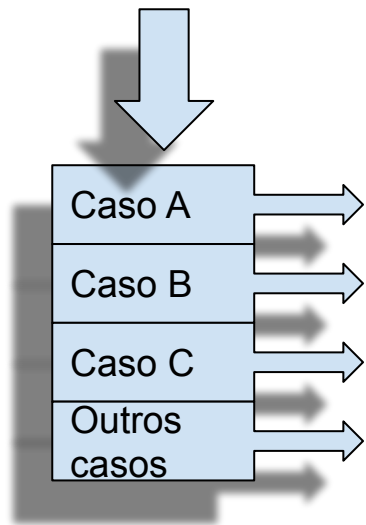
} else {

    console.log("Boa noite!");

}
```



Estruturas de controle - s



```
var diaSemana = 3; var nomeDia;

switch (diaSemana) {
  case 1:
    nomeDia = "Segunda-feira";
    break;
  case 2:
    nomeDia = "Terça-feira";
    break;
  case 3:
    nomeDia = "Quarta-feira";
    break;
  case 4:
    nomeDia = "Quinta-feira";
    break;
  case 5:
    nomeDia = "Sexta-feira";
    break;
  case 6:
  case 7:
    nomeDia = "Fim de semana";
    break;
  default:
    nomeDia = "Dia inválido";
}
```


Estruturas de controle - while



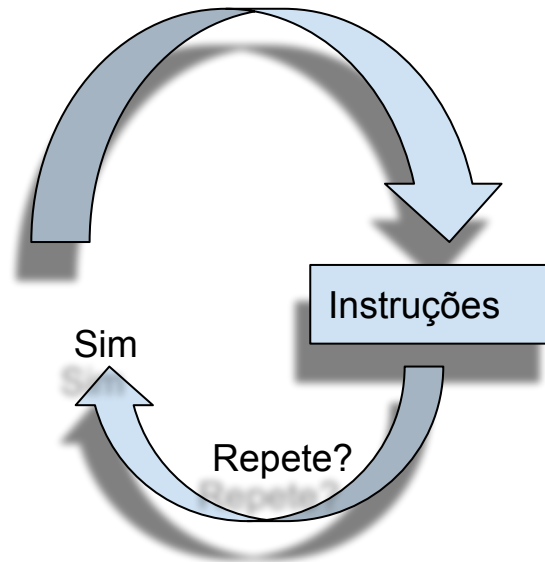
```
var i = 0;

while (i < 5) {

    console.log(i);

    i++;

}
```



Estruturas de controle - do while



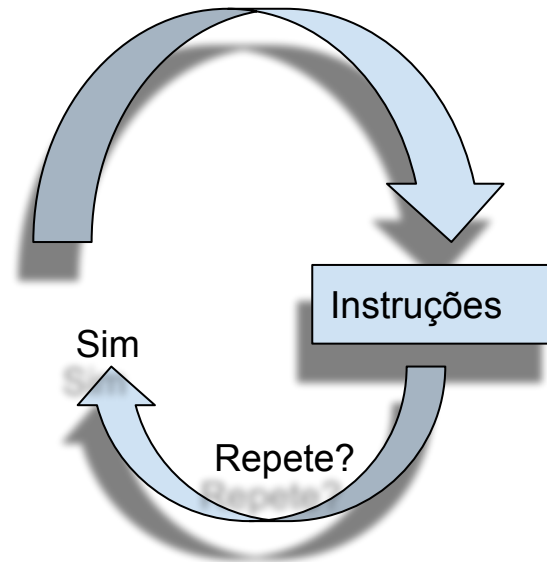
```
var i = 0;

do {

    console.log(i);

    i++;

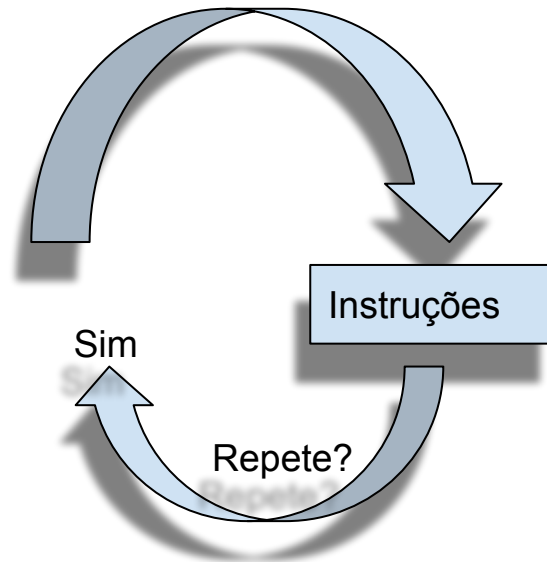
} while (i < 5);
```



Estruturas de controle - forEach



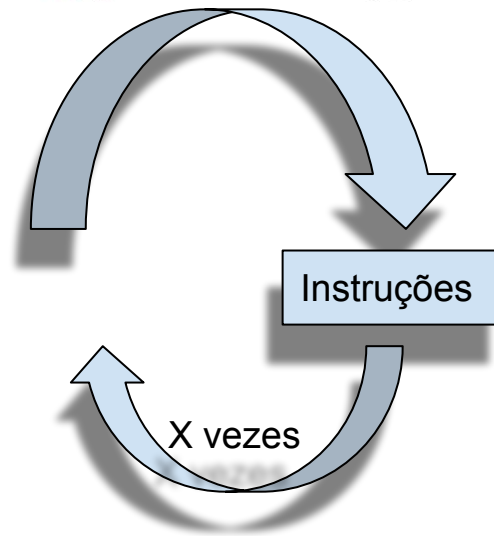
```
var frutas = ["Maçã", "Banana", "Laranja",  
"Morango"];  
  
frutas.forEach(function(fruta, indice) {  
    console.log("Índice " + indice + ": " +  
    fruta);  
});
```



Estruturas de controle - for



```
for (var i = 0; i < 5; i++) {  
    console.log(i);  
}
```



Funções

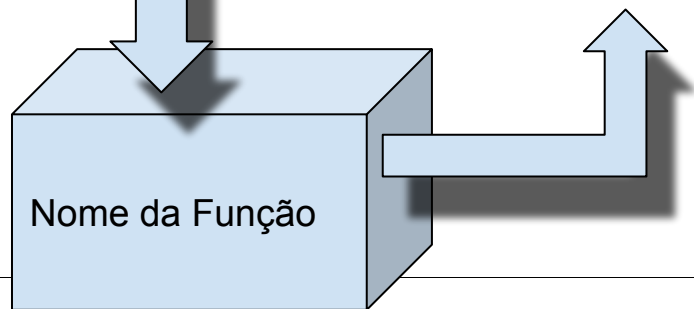


Pode definir funções usando a palavra-chave function.

```
function saudacao(nome) {  
    console.log("Olá, " + nome +  
    "!");  
}  
  
saudacao("Maria");
```

Parâmetros de Entrada

Valores retornados



Funções Básica



```
// Definição de uma função básica

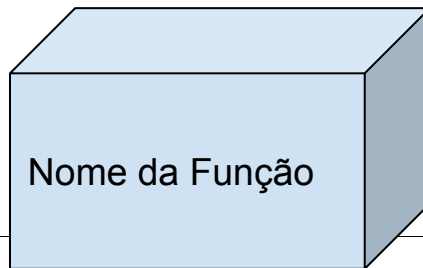
function saudacao () {

    console.log("Olá! Bem-vindo!");

}

// Chamada da função

saudacao ();
```



Funções com parâmetros



```
// Função que recebe parâmetros

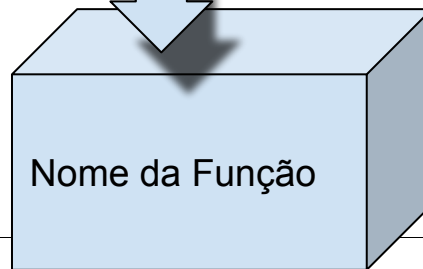
function saudacao(nome) {

    console.log("Olá, " + nome +
"!");
}

// Chamada da função com argumento

saudacao("João");
```

Parâmetros de Entrada



Funções com retorno



```
// Função que retorna um valor
```

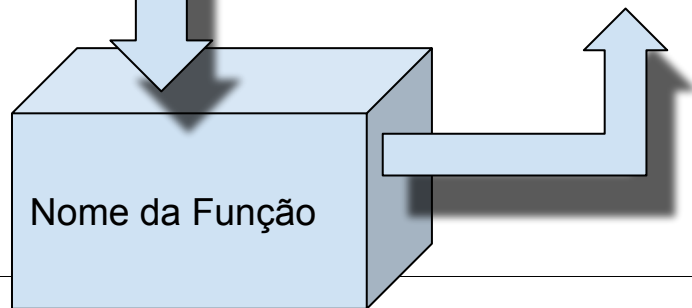
```
function soma(a, b) {  
  
    return a + b;  
  
}
```

```
// Chamada da função e atribuição do  
retorno a uma variável
```

```
var resultado = soma(5, 3);  
  
console.log("Resultado da soma:",  
resultado);
```

Parâmetros de Entrada

Valores retornados



Funções Anônima



```
// Função anônima atribuída a uma
variável

var saudacao = function() {

    console.log("Olá! Bem-vindo!");

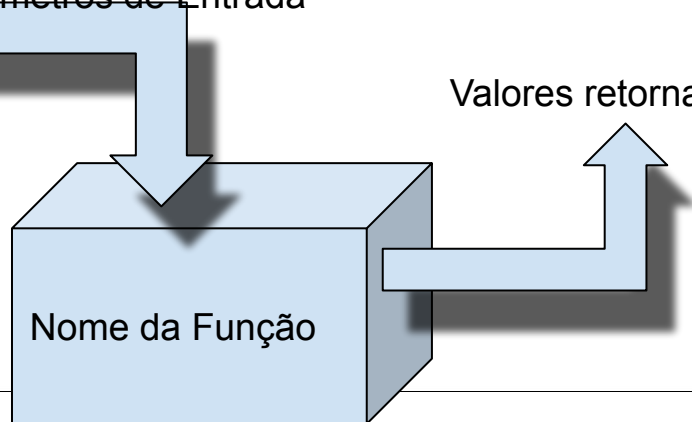
};

// Chamada da função

saudacao();
```

Parâmetros de Entrada

Valores retornados



Funções de flecha



```
// Função de flecha

var saudacao = () => {

    console.log("Olá! Bem-vindo!");

};

// Chamada da função

saudacao();
```

Parâmetros de Entrada

Valores retornados

Nome da Função

Funções com parâmetros padrão

```
// Função com parâmetros padrão
function saudacao(nome = "visitante")
{
    console.log("Olá, " + nome + "!");
}

// Chamada da função sem argumentos
saudacao(); // Saída: Olá, visitante!

// Chamada da função com argumento
saudacao("João"); // Saída: Olá, João!
```

Parâmetros de Entrada

Valores retornados

Nome da Função

Funções recursivas



```
// Função recursiva para calcular fatorial
function fatorial(n) {
  if (n === 0 || n === 1) {
    return 1;
  } else {
    return n * fatorial(n - 1);
  }
}

// Chamada da função
console.log("Fatorial de 5:", fatorial(5)); // Saída: 120
```

Objetos e arrays




É uma linguagem baseada em objetos, onde objetos podem ter propriedades e métodos.

Arrays são objetos que armazenam múltiplos valores em uma única variável.

```
var pessoa = {  
  nome: "João",  
  idade: 30,  
  cidade: "São Paulo"  
};  
  
var numeros = [1, 2, 3, 4, 5];
```

Array básico



```
// Definição de um array com alguns elementos
var frutas = ["Maçã", "Banana", "Laranja", "Morango"];

// Aceder a elementos do array
console.log(frutas[0]); // Saída: Maçã
console.log(frutas[2]); // Saída: Laranja

// Alterando um elemento do array
frutas[1] = "Abacaxi";

// Adicionando um elemento ao final do array
frutas.push("Pera");

// Removendo o último elemento do array
frutas.pop();
```

Array vazio



```
// Definição de um array vazio
var numeros = [];

// Adicionando elementos ao array
numeros.push(1);
numeros.push(2);
numeros.push(3);

console.log(numeros); // Saída: [1, 2, 3]
```

Iteração em um array



```
// Percorrer um array com um loop for
var cores = ["Vermelho", "Verde", "Azul"];

for (var i = 0; i < cores.length; i++) {
    console.log(cores[i]);
}

// Percorrer um array com forEach
cores.forEach(function(cor) {
    console.log(cor);
});
```


Iteração em um array



```
// Percorrer um array com um loop for
var cores = ["Vermelho", "Verde", "Azul"];

for (var i = 0; i < cores.length; i++) {
    console.log(cores[i]);
}

// Percorrer um array com forEach
cores.forEach(function(cor) {
    console.log(cor);
});
```

Métodos de array

```
var numeros = [1, 2, 3, 4, 5];

// Encontrando o índice de um elemento
var indice = numeros.indexOf(3); // Retorna 2

// Removendo um elemento pelo índice
numeros.splice(indice, 1); // Remove o elemento de índice 2 (3)

console.log(numeros); // Saída: [1, 2, 4, 5]

// Concatenando arrays
var maisNumeros = [6, 7, 8];
var todosNumeros = numeros.concat(maisNumeros);

console.log(todosNumeros); // Saída: [1, 2, 4, 5, 6, 7, 8]

// Ordenando um array
var numerosDesordenados = [5, 3, 1, 4, 2];
numerosDesordenados.sort();

console.log(numerosDesordenados); // Saída: [1, 2, 3, 4, 5]
```

Ler valores



```
<p>Ler um array</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
var person = { firstName: "António", lastName: "Fernandes", age: 55, eyeColor:  
"Castanhos" };
```

```
document.getElementById("demo").innerHTML = person.firstName + " tem " + person.age + "  
anos.";
```

```
</script>
```

Ver a Data



```
<h1>Que dia é Hoje?</h1>
<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()"> Click-me!
</button>
<p id="demo"></p>
```

Vamos lá

```
<form action="" id=frm1>
  Nome: <input type="text" name="fNome"
value="António"><br>
  Apelido: <input type="text" name="fApelido"
value="Lira Fernandes"><br><br>
  <input type="submit" value="Enviar">
</form>

<p>Click em "vamos lá!" para mostrar os valores do
formulário</p>
<p id="valores"></p>
<button onclick="mostraValores()">vamos lá!</button>
```

```
<script>
  function mostraValores(){
    var x= document.getElementById("frm1");
    var texto="";
    var i;
    for (i=0; i<x.length; i++){
      texto+= x.elements[i].value + "<br>";
    }
    document.getElementById("valores").innerHTML=texto;
  }
</script>
```

Modal bootstrap + js



A componente Modal é uma caixa de diálogo/janela pop-up que é exibida sobre a página atual.

```
<!-- The Modal -->
<div class="modal" id="myModal">
  <div class="modal-dialog">
    <div class="modal-content">
      <!-- Modal Header -->
      <div class="modal-header">
        <h4 class="modal-title">Titulo do Modal</h4>
        <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
      </div>
      <!-- Modal body -->
      <div class="modal-body">
        Corpo do Modal..
      </div>
      <!-- Modal footer -->
      <div class="modal-footer">
        <button type="button" class="btn btn-danger" data-bs-dismiss="modal">Fechar</button>
      </div>
    </div>
  </div>
</div>
```

Modal bootstrap + js



A componente Modal é uma caixa de diálogo/janela pop-up que é exibida sobre a página atual.

```
<!-- The Modal -->
<div class="modal" id="myModal">
  <div class="modal-dialog">
    <div class="modal-content">
      <!-- Modal Header -->
      <div class="modal-header">
        <h4 class="modal-title">Modal Heading</h4>
        <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
      </div>
      <!-- Modal body -->
      <div class="modal-body">
        Modal body..
      </div>
      <!-- Modal footer -->
      <div class="modal-footer">
        <button type="button" class="btn btn-danger" data-bs-dismiss="modal">Close</button>
      </div>
    </div>
  </div>
</div>
```

Dropdownlist/ Menu[4]

```
/* When the user clicks on the button,
toggle between hiding and showing the dropdown
content */
function myFunction() {

document.getElementById("myDropdown").classList.toggle("show");
}

// Close the dropdown menu if the user clicks
outside of it
window.onclick = function(event) {
    if (!event.target.matches('.dropbtn')) {
        var dropdowns =
document.getElementsByClassName("dropdown-content");
        var i;
        for (i = 0; i < dropdowns.length; i++) {
            var openDropdown = dropdowns[i];
            if (openDropdown.classList.contains('show')) {
                openDropdown.classList.remove('show');
            }
        }
    }
}
```

```
<div class="dropdown">
  <button onclick="myFunction()"
class="dropbtn">Dropdown</button>
  <div id="myDropdown" class="dropdown-content">
    <a href="#">Link 1</a>
    <a href="#">Link 2</a>
    <a href="#">Link 3</a>
  </div>
</div>
```

Clickable Dropdown

Click on the button to open the dropdown menu.



Dropdownlist/ Menu[3]



```
/* Dropdown Button */
.dropbtn {
  background-color: #3498DB;
  color: white;
  padding: 16px;
  font-size: 16px;
  border: none;
  cursor: pointer;
}

/* Dropdown button on hover & focus */
.dropbtn:hover, .dropbtn:focus {
  background-color: #2980B9;
}

/* The container <div> - needed to position
the dropdown content */
.dropdown {
  position: relative;
  display: inline-block;
}
```

```
/* Dropdown Content (Hidden by Default) */
.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f1f1f1;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  z-index: 1;
}

/* Links inside the dropdown */
.dropdown-content a {
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
}

/* Change color of dropdown links on hover */
.dropdown-content a:hover {background-color: #ddd}

/* Show the dropdown menu (use JS to add this class to
the .dropdown-content container when the user clicks on
the dropdown button) */
.show {display: block;}
```

Exemplo

```
<head>
  <title> Teste </title>
  <script src="name.js" type="text/javascript"></script>
</head>
<body>
  <h1>Name Converter</h1>
  <p>Type your name:</p>
  <div>
    <input id="name" type="text" size="24">
    <button onclick="capitalize();">Convert to Last, First</button>
  </div>
</body>
// Capitalizes every odd letter of a given name function capitalize() {
  var input = document.getElementById("name");
  var name = input.value;
  var newName = "";
  for (var i = 0; i < name.length; i++) {
    if (i % 2 == 0) { newName += name.charAt(i).toUpperCase();} else { newName +=
      name.charAt(i).toLowerCase();}
  }
  input.value = newName;}
```



Exemplo

```
<!DOCTYPE html>
<html><head>
<title> Teste </title>
<script type="text/javascript">
function capitalize() {
    var input = document.getElementById("name");
    var name = input.value;
    var newName = "";
    for (var i = 0; i < name.length; i++) {
        if (i % 2 == 0) {
            newName += name.charAt(i).toUpperCase();
        } else {
            newName += name.charAt(i).toLowerCase();
        }
    }
    input.value = newName;
}
</script>
</head>
<body>
<h1>Name Converter</h1>
<p>Type your name:</p>
<div>
    <input id="name" type="text" size="24">
    <button onclick="capitalize();">Convert to Last, First</button>
</div>
</body></html>
```

Name Converter

Type your name:

Exemplo

```
<!DOCTYPE html>
<html>
<head>
  <title> Teste </title>
  <script type="text/javascript" src="leapyear.js"></script>
</head>
<body>
  <h1>Leap Years</h1>
  <p>Type a year:</p>
  <div>
    <input id="year" type="text" size="24">
    <button onclick="leapYears();">Compute Leap Years</button> <br>
    <span id="output"></span>
  </div>
</body>
</html>
```

Exemplo

```
// Displays the next 10 leap years after a given year.  function
leapYears() {
    var input = document.getElementById("year");
    var year = parseInt(input.value);
    var output = document.getElementById("output"); output.innerHTML = "Next
10 leap years after " + year + ": ";  var count = 0;
    while (count < 10) {
        if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
            count++;
            output.innerHTML += " " + year;
        }
        year++;
    }
}
```

Exemplo

```
<!DOCTYPE html>
<html><head>
<title> Teste </title>
  <script type="text/javascript">function leapYears() {
    var input = document.getElementById("year");
    var year = parseInt(input.value);
    var output = document.getElementById("output");
    output.innerHTML = "Next 10 leap years after " + year + ": ";
    var count = 0;
    while (count < 10) {
      if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
        count++;
        output.innerHTML += " " + year;
      }
      year++;
    }
  }
</script>
</head>
<body>
  <h1>Leap Years</h1>
  <p>Type a year:</p>
  <div>
    <input id="year" type="text" size="24">
    <button onclick="leapYears();">Compute Leap Years</button> <br>
    <span id="output"></span>
  </div>
</body></html>
```

Leap Years

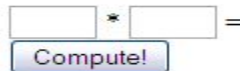
Type a year:

Next 10 leap years after 2016: 2016 2020 2024 2028 2032 2036 2040 2044 2048

Exemplo

```
<!DOCTYPE html>
<html>
<head>
  <script src="multiply.js" type="text/javascript"></script>
</head>
<body>
  <h1>The Amazing Multiplier</h1>
  <div>
    <input id="num1" type="text" size="3"> *
    <input id="num2" type="text" size="3"> =
    <span id="answer"></span> <br>
    <button onclick="compute();">Compute!</button>
  </div>
</body>
</html>
function compute() {
  var input1 = document.getElementById("num1"); // fetch the 2 numbers
  var input2 = document.getElementById("num2");
  var answer = document.getElementById("answer");
  var result = input1.value * input2.value; // compute result
  answer.innerHTML = result;}
```

The Amazing Multiplier



Exemplo

Multiplicador a funcionar no cliente

```
function compute() {  
    // fetch the 2 numbers  
    var input1 =  
    document.getElementById("num1"); var  
    input2 = document.getElementById("num2");  
    var answer =  
    document.getElementById("answer");  
    var result = input1.value * input2.value; // compute result  
    answer.innerHTML = result;  
}
```



Arrays

- Sintaxe para criação de um array
 - `var name = [];` ou `var name = [value, value, ..., value];`
- Propriedade length

Método	Description
<code>concat(array1, ..., arrayN)</code>	Junta dois ou mais arrays
<code>pop()</code>	Remove o último elemento de um array e devolve-o
<code>shift()</code>	Remove o primeiro elemento de um array e devolve-o
<code>push(value)</code> <code>push(value1,..., valueN)</code>	Adiciona um ou mais elementos no fim do array
<code>unshift(value)</code> <code>unshift(value1,..., valueN)</code>	Adiciona um ou mais elementos no início do array
<code>slice(startIndex)</code> <code>slice(startIndex, endIndex)</code>	Retorna um sub-array de um array existente (startIndex inclusive e endIndex exclusivo)
<code>splice(index, count, value, ...,valueN)</code>	Remove count elementos de um array a começar no índice index e insere os novos elementos nessas posições
<code>sort()</code>	Ordena os elementos de um array
<code>toString()</code>	Converte um array para uma string
<code>reverse()</code>	Inverte a ordem dos elementos de um array

Arrays

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.length; // 4
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Kiwi"); // Banana, Orange, Apple, Mango, Kiwi
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.pop(); // Banana, Orange, Apple
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
var a = fruits.indexOf("Apple"); // 2
```

```
var hege = ["Cecilie", "Lone"];  
var stale = ["Emil", "Tobias", "Linus"];  
var children = hege.concat(stale); //Cecilie, Lone, Emil, Tobias, Linus
```

Exemplo

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Functions</h2>
  <p>This example calls a function which performs a calculation, and returns the
  result:
    </p>
    <p id="demo"></p>
    <script>
      function myFunction(p1, p2) {
        return p1 * p2;
      }
      document.getElementById("demo").innerHTML = myFunction(4, 3);
    </script>
  </body>
</html>
```

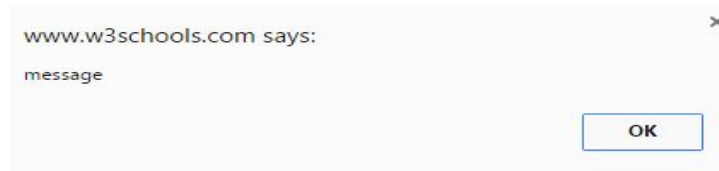
JavaScript Functions

This example calls a function which performs a calculation, and returns the result:

Caixas de Mensagens, Diálogo e Confirmação

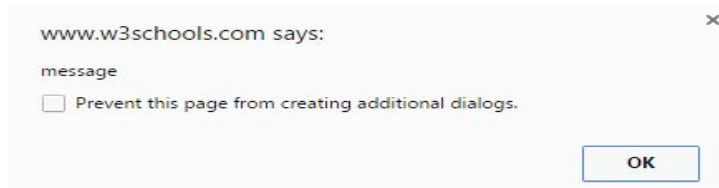
Caixa de mensagens (message dialog box)

- **`alert("message")`**



Caixa de diálogo (dialog box)

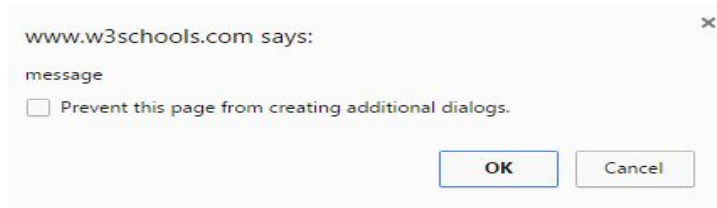
- Janela de pop-up para o utilizador inserir valores
- Além de espaço para inserção de valores, a janela tem 2 botões: **OK** e **Cancel**
- **`prompt("message")`** **`prompt("message", "default value")`**
- Retorna a string inserida ou **null** se o utilizador cancelar



Caixas de Mensagens, Diálogo e Confirmação

Caixa de confirmação (confirm box)

- Usada para pedir ao utilizador para verificar algo
- Tem dois botões: OK e Cancel
- **confirm("message")**
- Retorna **true** se OK ou **false** se Cancel



Output

O JavaScript pode "exibir" dados de diferentes maneiras:

- Escrevendo em um elemento HTML, usando **innerHTML**.
- Escrevendo na saída HTML usando **document.write()**.
- Escrevendo em uma caixa de alerta, usando **window.alert()**.
- Escrevendo no consola do browser, usando **console.log()**.



Output

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
</body>
</html>
```

My First Web Page

My first paragraph.

11



Output

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>  document.write(5 + 6);
</script>
</body>
</html>
```

My First Web Page

My first paragraph.

11



Output

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<button onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

My First Web Page

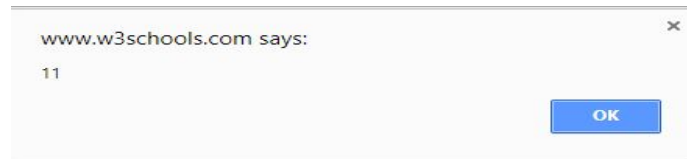
My first paragraph.

Try it



Output

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script> window.alert(5 + 6);
</script>
</body>
</html>
```



Output

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

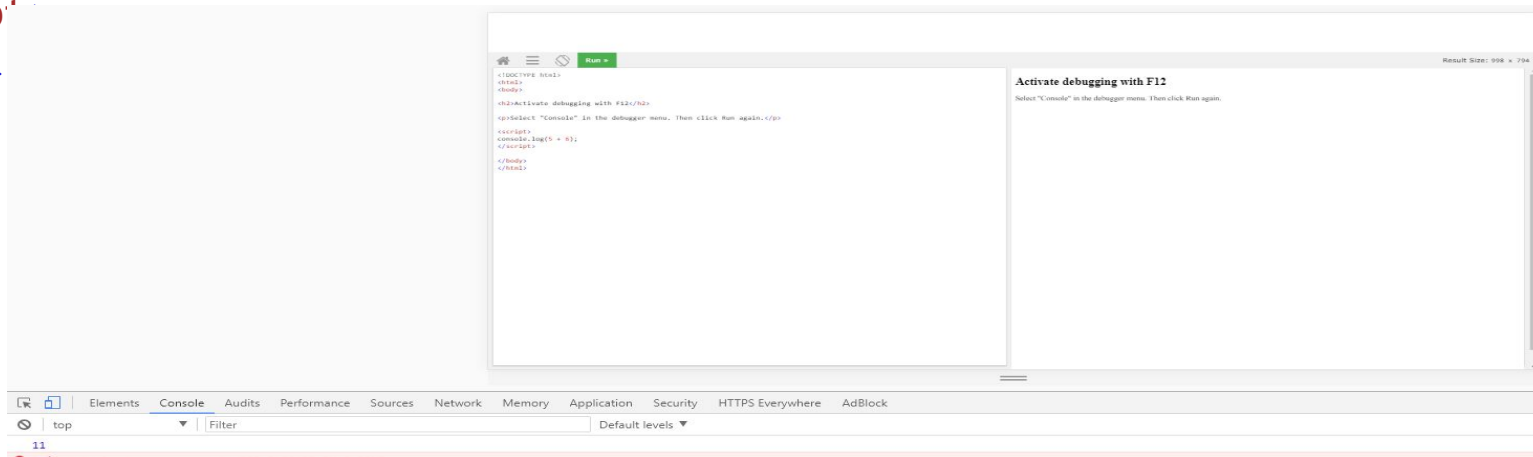
```
<script>
```

```
    console.log(5 + 6);
```

```
</script>
```

```
</body>
```

```
</html>
```



Output

Esta palavra-chave não se refere à função em que é usada ou ao seu ambiente.

Refere-se ao objeto em que uma função está sendo executada e depende inteiramente do local onde ocorre a chamada da função.

this = este objeto

```
function testThisKeyword() {  
    console.log(this.name) // Alex  
}  
var name = 'Alex'  
testThisKeyword()
```

'this'
Keyword In Javascript

This

```
// JavaScript source code
var person = {
  firstName: "Penelope",
  lastName: "Barrymore",
  fullName: function ()
  {
    // Notice we use "this" just as we used "he"
    console.log(this.firstName + " " + this.lastName);
    // We could have also written this:
    console.log(person.firstName + " " + person.lastName);
  }
}
```

Constructor

```
function Foo(){
  this.value = 1;
  console.log(this);
}
```

```
var obj = new Foo();
console.log(obj.value); // logs 1
```



Debugging

O **debuging** é terrível porque os browsers não transmitem "grandes" mensagens de erros!!!

Alguns browsers permitem que se veja o log dos erros JavaScript ocorridos (INSPEC – Inspeccionar Código)

- Chrome - <https://developers.google.com/web/tools/chrome-devtools/>
- Firefox – Firebug
- Safari –
https://developer.apple.com/library/content/documentation/AppleApplications/Conceptual/Safari_Developer_Guide/Debugger/Debugger.html

Solução:

- explorar o **console.log()** e o **alert()**.



Erros Comuns

Engano na escrita do nome do ficheiro na tag script

- Problema - Página é visualizada mas não faz o processamento desejado (ex.: não calcula a multiplicação no caso do exemplo anterior)
- Sugestão - Colocar um **alert("JS CARREGADO");** no topo do ficheiro .js fora de qualquer função

Nomes de IDs incorretos

- Console – posso inserir expressões ou instruções JavaScript e executá-las!!
- O JSLint ajuda:
 - Utilitário para detetar erros JavaScript (Douglas Crockford da Yahoo)
 - Página web (<http://www.jshint.com>) onde se pode submeter o código JavaScript
 - Faz verificação sintática



Boas Práticas



Definir e gerir o que são variáveis globais e locais;

Declarações no topo;

Inicializar variáveis;

Nunca declarar Número, String ou Objetos Booleanos;

Não usar novo **objeto()**;

Usar a comparação **===**.



Referências

[1] Portela, F.; Soares, D. (2020). PROGRAMAÇÃO PARA A WEB - Aula T4 – JavaScript; Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

[2] Portela, F.; Queirós, R.; (2018). INTRODUÇÃO AO DESENVOLVIMENTO MODERNO PARA A WEB - DO FRONT-END AO BACK-END: UMA VISÃO GLOBAL; FCA; ISBN: 978-972-722-897-3

[3] w3schools (2020). JavaScript HTML DOM; https://www.w3schools.com/js/js_htmlDOM.asp

[4] w3schools (2020). How To Create a Dropdown Menu With CSS and JavaScript; https://www.w3schools.com/howto/howto_js_dropdown.asp

[5] w3schools (2020). Práticas recomendadas de JavaScript; https://www.w3schools.com/js/js_best_practices.asp