

# Introdução às tecnologias Web - ITW

## Aula 5 – Programação web utilizando Javascript

# Sumário

## Linguagens de programação

Breves noções

## A linguagem Javascript

Introdução

Sintaxe JavaScript

Interacção com o DOM

Eventos



# Como se programam os dispositivos?

Os diversos dispositivos – computadores, tablets, telemóveis, relógios, ... – executam sempre código binário (101001010001001... ) ajustado ao seu tipo de processador.

A linguagem de programação de cada dispositivo é distinta das demais.

De modo a facilitar o processo de criação de código máquina – e evitar que tenhamos de saber a linguagem máquina dos diversos processadores – foram desenvolvidas as “linguagens de programação”.

# Linguagens de programação

## O que são?

Uma “linguagem de programação” é uma linguagem que possui uma sintaxe (formato) e uma semântica (significado).

É usada para expressar uma sequência de ações computacionais que formam um “programa” que, por sua vez, é executado num dispositivo.

Existem milhares de linguagens de programação e novas linguagens surgem frequentemente, trazendo novos paradigmas e estabelecendo novos padrões para programadores.

Por isso, é importante conhecer as diferenças principais entre as linguagens e quando o uso de cada uma delas é mais adequado.

# Linguagens de programação

## Conceitos básicos

**Pergunta:** mas se escrevemos o programa em texto, como é feita a tradução do texto do programa para o código binário – sequência de 1's 0's – percebida pelo dispositivo?

A tradução é tipicamente feita em várias fases, sendo as mais comuns a análise léxica, a análise sintática (ou *parsing*), a geração de código e a otimização.

Esse processo é conhecido como compilação ou interpretação do programa.

# Linguagens de programação

## Conceitos básicos

Numa linguagem compilada, o **compilador** verifica a sintaxe do código escrito para garantir que esta está de acordo com a semântica adequada e, caso tudo esteja correto, gera um código executável a partir do código fonte escrito pelo programador.

O código executável não possui o conteúdo do código fonte, portanto programas de linguagens compiladas são melhores de distribuir quando o programador não quer que o seu código seja público.

A versão compilada do programa tipicamente é armazenada, de forma que o programa pode ser executado um número indefinido de vezes sem que seja necessária nova compilação, o que compensa o tempo gasto na compilação

# Linguagens de programação

## Conceitos básicos

Se o texto do programa for executado à medida que vai sendo traduzido, como em JavaScript, BASIC, Python ou Perl, então diz-se que o programa foi **interpretado** e que o mecanismo utilizado para a tradução é um interpretador.

Programas interpretados são geralmente mais lentos do que os compilados, mas são também geralmente mais flexíveis, já que podem interagir com o ambiente mais facilmente.

# Linguagens de programação

## Linguagens “tipadas” (do inglês typed):

Nas linguagens **tipadas**, as operações são aplicadas em estruturas de dados bem definidas e cada operação define os tipos de dados que deve receber.

Nas linguagens fracamente *tipadas*, as operações são aplicadas para qualquer estrutura de dados; porém, essas operações podem falhar em tempo de execução caso a estrutura não suporte a operação.

Java (typed)

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

Numa função em Java, os tipos de dados da operação **soma** estão bem definidos (**a** é float e **b** é int) e o tipo de dado que a função devolve também (**resultado** é float).

Ruby (untyped)

```
def soma (a, b)
  return a + b
end
```

Na função em Ruby, a função **soma** pode receber quaisquer tipos de dados para **a** e **b**, e a operação será aplicada sobre esses tipos, devolvendo um resultado de tipo desconhecido à partida:

- se **a** e **b** forem String, o resultado será uma String concatenada de **a** e **b**;
- se **a** e **b** forem inteiros, o resultado será um inteiro que representa a soma **a+b**;
- se **a** for um float e **b** um inteiro, o resultado será um float que representa a soma **a+b**.



# A linguagem Javascript

# Porquê JavaScript?

O JavaScript é uma das três linguagens que todos os que desenvolvem páginas web têm de aprender:

1. HTML para definir o conteúdo das páginas da web;
2. CSS para especificar o layout das páginas da web; e
3. JavaScript para programar o comportamento das páginas da web!



# A linguagem Javascript

## Introdução

O JavaScript é uma linguagem de *script* (interpretada) orientada a objetos e que funciona em múltiplas plataformas – tanto do lado do **cliente** como do lado do **servidor**.

Tanto pode ser executada num **browser** como no **sistema operativo**. O código JavaScript pode estar ligado a objetos do ambiente e fornece controle programático sobre os mesmos.

De acordo com a definição anterior, o Javascript é uma linguagem fracamente tipada.

Possui uma biblioteca padrão de objetos, tais como **Array**, **Date**, **Math**, e um conjunto fundamental de elementos da linguagem tais como operadores, estruturas de controle, e *statements*.

# A linguagem Javascript

## Para que serve?

O JavaScript pode ser aumentado com objetos adicionais para uma variedade de propósitos. Por exemplo:

Um programa em JavaScript executado no **browser** fornece objetos para controlar o browser e o seu Document Object Model (DOM).

*Por exemplo, extensões de cliente permitem a uma aplicação adicionar elementos num formulário HTML e responder a eventos do utilizador tais como cliques, texto adicionado, e navegação na página.*

Um programa em JavaScript executado num **servidor** fornece objetos relevantes para aceder a funcionalidades diversas – ambiente, bases de dados, sensores, etc..

*Por exemplo, extensões do lado do servidor permitem que uma aplicação comunique com uma base de dados, garanta continuidade de informação entre invocações da aplicação, ou realize manipulações de ficheiros no servidor.*

# Vantagens e desvantagens do Javascript

Como é uma linguagem interpretada, é processada aos blocos e interpretada à medida que é necessário converter as diversas estruturas para uma representação capaz de ser executada.

A **vantagem** clara desta aproximação é que aparentemente basta executar diretamente o código escrito pelo programador.

A **desvantagem** é que **muitos erros só são detectados quando o fluxo de execução atinge a linha onde o erro está presente** – o que pode provocar paragens na execução.

# Para que serve o javascript

A linguagem Javascript foi originalmente implementada como parte dos web browsers para que estes pudessem executar programas (**que em javascript se denominam *scripts***) do lado do cliente e interagissem com o utilizador sem a necessidade deste recorrer ao servidor.

Um script javascript permite:

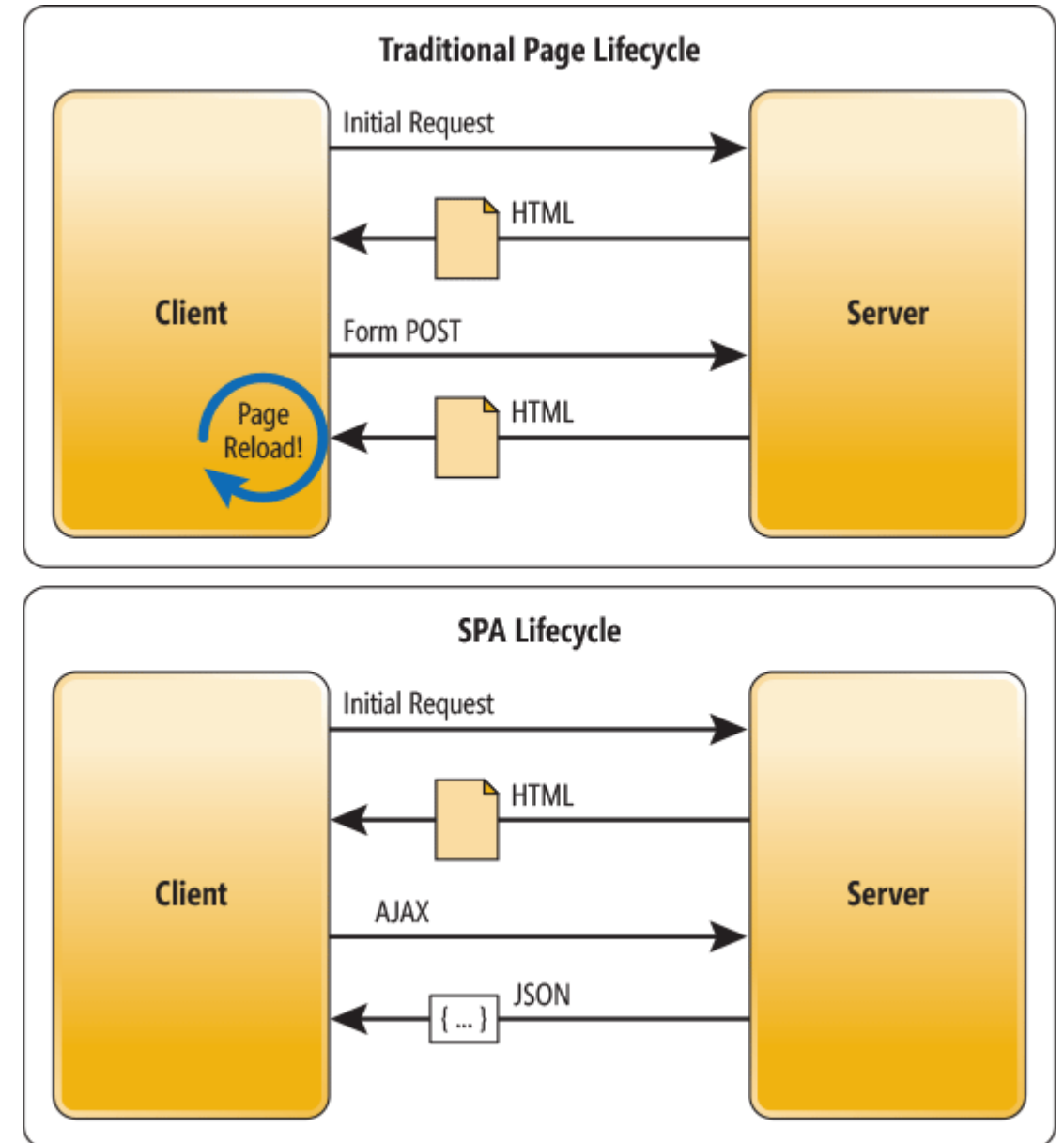
- controlar o web browser,
- realizar comunicações assíncronas,
- alterar o conteúdo do documento exibido de modo dinâmico.

# Utilização do javascript

A linguagem javascript começa também a ser utilizada:

do lado do servidor através de ambientes como, por exemplo, o node.js;  
em aplicações cliente de página simples (SPA – Single Page Applications).

É objetivo último desta unidade curricular elaborar, no final da disciplina, uma SPA! Assim, votaremos a este desenho mais para o final da disciplina.



# Inclusão de script javascript numa página html

O processo de inclusão numa página html é semelhante à da inclusão dos estilos CSS (`<style></style>`), ou seja, através da utilização de marcas específicas `<script></script>`, normalmente, no cabeçalho `<head></head>` da página ou no final do corpo do documento `<body></body>`, de modo a não interefrir com a normal apresentação do documento.

O código JS pode também ser incluído diretamente ou ser obtido de uma fonte externa – em ficheiros, normalmente, com a extensão “.js”.



# Inclusão direta na página

fim do <head> </head>

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    /* 0 script Javascript deve ser colocado aqui */
  </script>
</head>
<body>
  <!-- Conteúdo html aqui ...-->
</body>
</html>
```

# Inclusão direta na página

fim do <body> </body>

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <!-- Conteúdo html aqui ...-->
  <script>
    /* O script Javascript deve ser colocado aqui */
  </script>
</body>
</html>
```

# Inclusão direta na página

fim do <body> </body>

Exemplo da aula passada, com a inclusão do bootstrap:

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <title></title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
  <main role="main">
  </main>
  <!-- Fim da página, scripts de suporte ao Bootstrap -->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

# Versatilidade vs segurança

Conforme referido, a linguagem Javascript é bastante poderosa e o facto poder ser executada em qualquer browser de qualquer sistema operativo / dispositivo, permite desenvolver aplicações que podem ser distribuídas de forma muito eficaz – **elevada versatilidade**.

No entanto, o código Javascript é sempre enviado ao cliente na sua forma textual, podendo, por isso, ser rapidamente copiado – **segurança reduzida**.

Para dificultar a leitura do código, protegendo a autoria do mesmo, e para poupar no espaço ocupado pelo ficheiro, de modo a não prejudicar o carregamento e posterior apresentação da página, este código é muitas vezes “minimizado” (tradução livre de minified).

Exemplos de minimização de ficheiros:

*bootstrap.min.css, bootstrap.min.js, ...*

# A linguagem Javascript

# A linguagem Javascript

A sintaxe da linguagem Javascript é inspirada na linguagem C e algo semelhante à linguagem Java.

Não iremos explorar com detalhe todos os aspetos de sintaxe, ou todas as propriedades da linguagem, mas iremos possibilitar uma utilização básica da mesma.

A sintaxe básica da linguagem Javascript é baseada em **instruções**, devendo cada uma das instruções ser terminada com o carater **;** (ponto-e-vírgula).

A linguagem Javascript é **case-sensitive**, o que significa que se deve ter cuidado na escrita (Sim  $\neq$  sim  $\neq$  SIM).

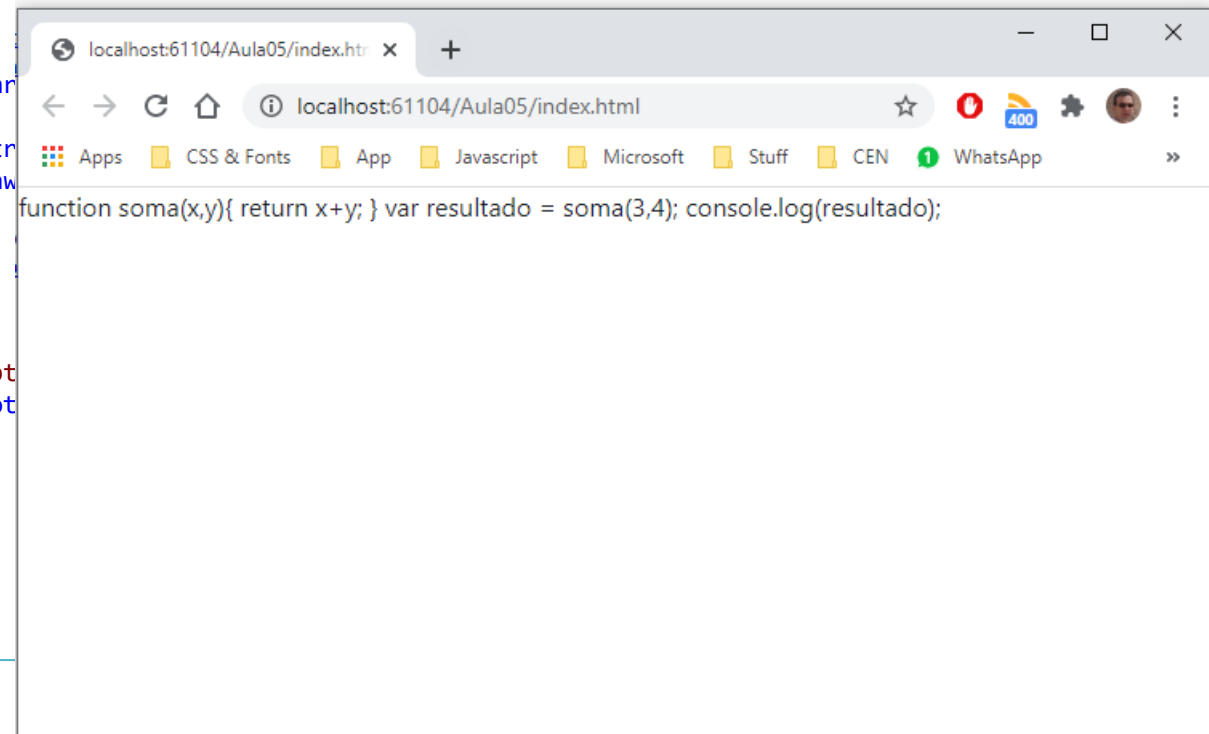
# A linguagem Javascript

## Programação web

Se modo a ser possível executar os programas (script's) numa página web, estes devem ser colocados dentro do marcador (`<script></script>`)

Se não estiverem colocados dentro do marcador, o browser interpretará o código como HTML e mostrará o texto na página...

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <title></title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
  <main role="main">
  </main>
  <!-- Fim da página, scripts de suporte ao Bootstrap -->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
  <script>
    function soma(x,y){
      return x+y;
    }
    var resultado = soma(3,4);
    console.log(resultado);
  </script>
</body>
</html>
```



# Sintaxe da linguagem Javascript

## Declaração e atribuição de variáveis

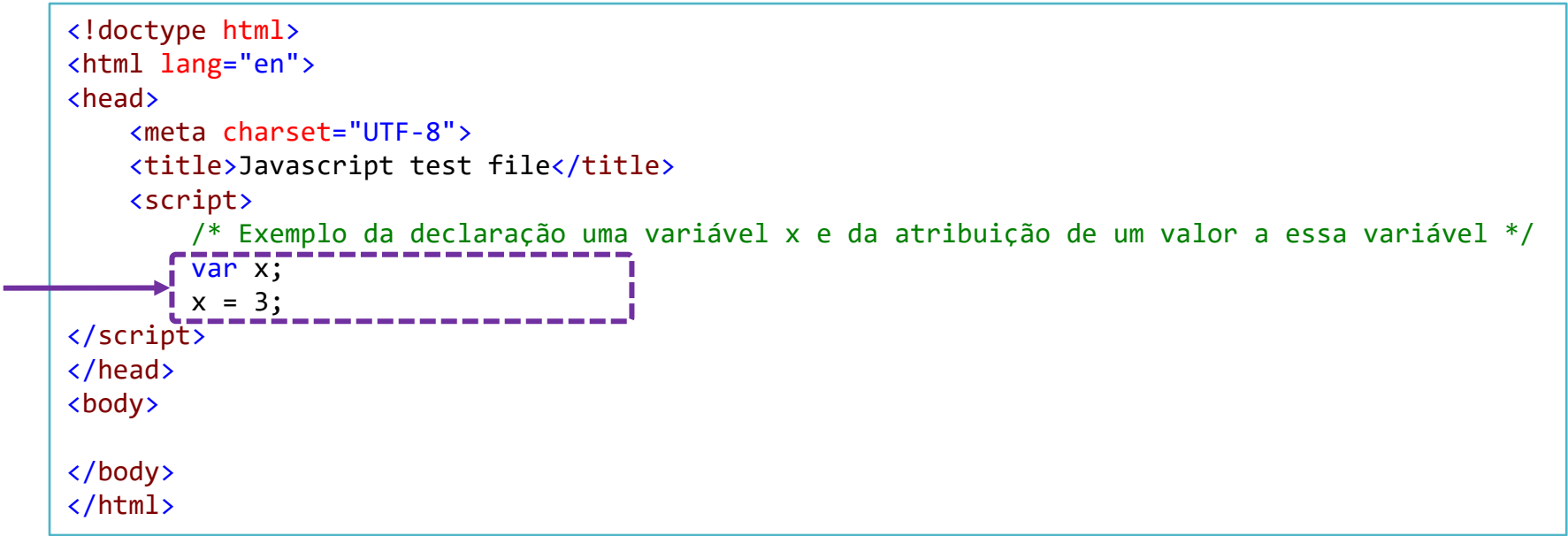
A **declaração** de variáveis é feita através da utilização da palavra reservada **var** seguida pelo **nome\_da\_variável**.

A **atribuição** de valores a uma variável faz-se de modo convencional:

**<nome\_da\_variável> = <valor>**

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script>
    /* Exemplo da declaração uma variável x e da atribuição de um valor a essa variável */
    var x;
    x = 3;
  </script>
</head>
<body>

</body>
</html>
```





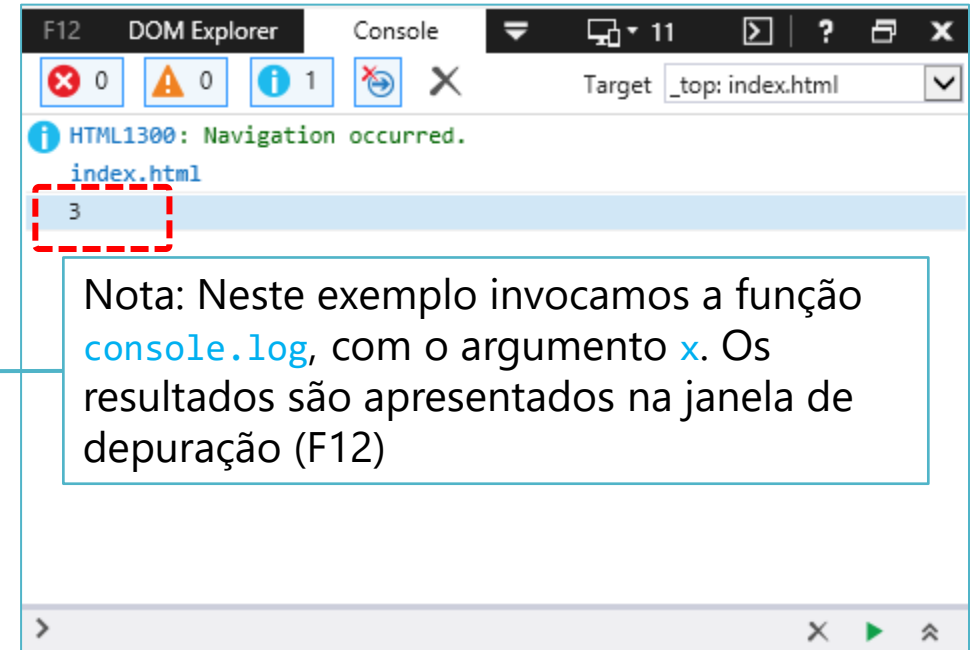
# Sintaxe da linguagem Javascript

## Depuração de programas

Exemplo de apresentação do conteúdo da variável x na consola do navegador.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script>
    /* Apresentação do resultado da atribuição do
       valor a uma variável */
    var x;
    x = 3;
    console.log(x);
  </script>
</head>
<body>

</body>
</html>
```



# Sintaxe da linguagem Javascript

## Funções

De forma a melhor organizar o código, e evitar a replicação desnecessária, é possível organizar um programa em **funções**.

Uma função pode ser vista como um conjunto de comandos que realizam uma tarefa específica. Pode ser utilizada por outras funções.

```
function nome_da_funcao(arg1, arg2, arg3) {  
    /* ...Conteúdo... */  
}
```

# Sintaxe da linguagem Javascript

## Funções

```
function nome_da_funcao(arg1, arg2, arg3) {  
    /* ...Corpo - Conteúdo... */  
}
```

A declaração de funções faz uso da palavra reservada `function`, tal como descrito no exemplo.

Estes elementos são seguidos por um `nome`, uma `lista de argumentos` e um `corpo`.

A `lista de argumentos` é opcional. Funciona como a interface de comunicação (passagem de valores/dados) entre o programa (chamador) e a função.

Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas.

O `corpo` possui as linhas de código que permitirão a execução da ação. É delimitado por chavetas (`{}`).

# Sintaxe da linguagem Javascript

## Operações

Podem ser aplicados operadores aritméticos às variáveis, tais como a soma, ou a subtração, multiplicação, divisão, módulo (%), incremento (++), decremento (--).

O significado da operação irá variar de acordo com o tipo de variável (que depende do seu conteúdo atual).

Um bom exemplo é o operador +, que no caso de números irá calcular a soma, mas no caso de sequências de caracteres irá concatená-los.

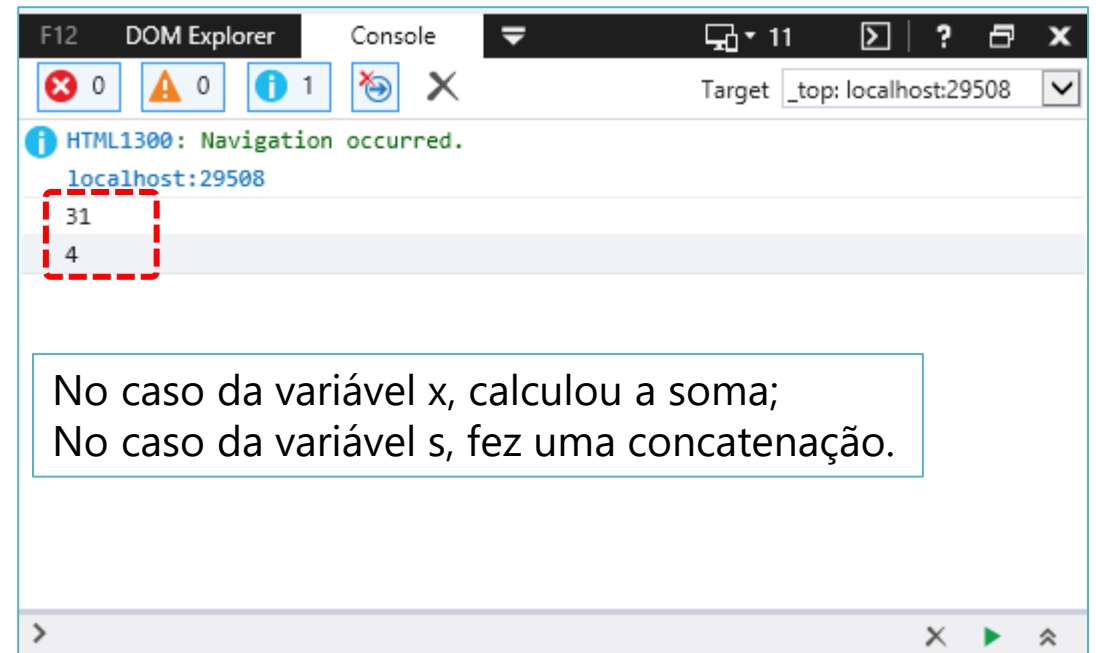
# Sintaxe da linguagem Javascript

## Exemplo com operações

O exemplo seguinte demonstra a aplicação do operador +:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script>
    /* Comentário */
    var s = "3";
    var x = 3;
    console.log(s + 1);
    console.log(x + 1);
  </script>
</head>
<body>

</body>
</html>
```

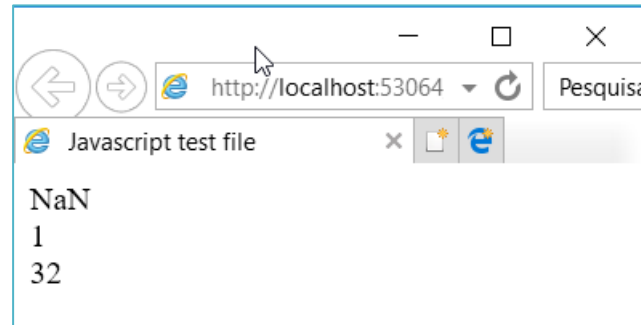


# Sintaxe da linguagem Javascript

Exemplo com operações (com erro)

Analisemos o possível resultado do código seguinte:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script>
    /* Comentário */
    var s = "texto";
    var x = "3";
    document.write(s - 2);
    document.write("<br/>");
    document.write(x - 2);
    document.write("<br/>");
    document.write(x + 2);
  </script>
</head>
<body>
</body>
</html>
```



Neste caso, através da utilização da função `document.write(...)`, ao invés do resultado ser apresentado na consola da janela de depuração (F12), o resultado é "escrito" na janela de visualização do browser.

Quando uma operação aritmética não é válida, a linguagem Javascript faz uso do termo `NaN` que significa **Not a Number**. Isto pode ser facilmente obtido se subtrairmos um inteiro a uma **String não numérica**!

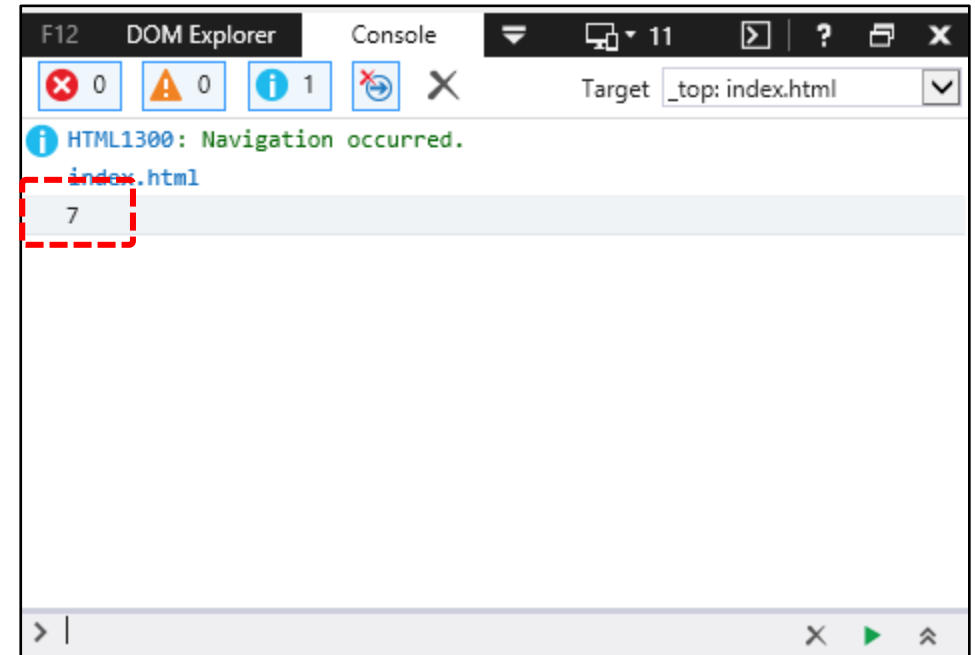
# Sintaxe da linguagem Javascript

## Exemplo de utilização de funções

Utilizando como exemplo uma função que realize a soma de dois números, pode ser declarada e invocada da seguinte forma:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script>
    function soma(x,y){
      return x+y;
    }
    var resultado = soma(3,4);
    console.log(resultado);
  </script>
</head>
<body>

</body>
</html>
```



# Sintaxe da linguagem Javascript

## Condições (if ... else)

A **execução condicional** é implementada através das palavras reservadas **if** ... **else**, no seguinte formato:

```
if (comparação) /* Se o resultado da comparação for positivo */
{
    /* Executa das instruções no caso positivo */
}
else /* ... senão ... */
{
    /* Executa das instruções do caso negativo */
}
```

Os operadores de comparação são:

<	→	Menor
>	→	Maior
>=	→	Maior
!=	→	Diferente
==	→	Igual, etc...



# Sintaxe da linguagem Javascript

Exemplo de utilização de instruções condicionais (if ... else)

Considere o seguinte excerto:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script>
    var a = "3";
    var b = 3;
    if (a == b)
      alert("Iguais");
    else
      alert("Diferentes");
  </script>
</head>
<body>

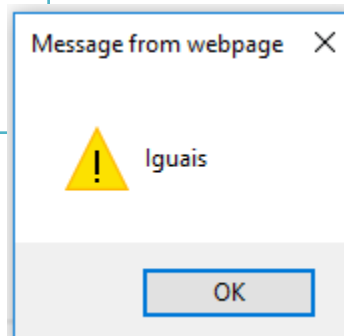
</body>
</html>
```

O operador igual (==) permite comparar tipos diferentes, convertendo os seus valores.

Por vezes é necessário comparar quer o “valor” quer o “tipo” de uma variável.

Para isso, existe o operador === e a sua negação, o operador !==.

Na linguagem Javascript diz-se que estes comparadores verificam se o valor é igual e o tipo idêntico. No caso anterior, o valor de a é igual ao de b mas as variáveis não são idênticas.



Neste exemplo, as variáveis **a** e **b** possuem o mesmo “valor”, 3, mas são de tipos diferentes.

Uma é uma string (cadeia de caracteres) a outra um número inteiro.

Caso tivesse sido utilizado o operador ===, o resultado seria distinto.

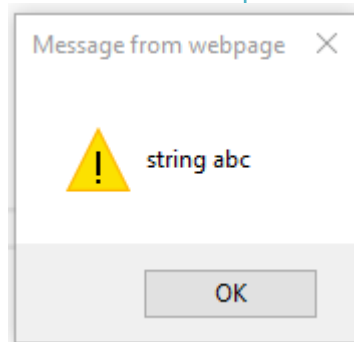
# Sintaxe da linguagem Javascript

## Condições (switch ... case)

Quando há mais que uma condição para testar, é possível a utilização de um conjunto de instruções `if ... else` encadeadas ou, em alternativa, a utilização da instrução `switch ... case`.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <!-- Conteúdo html aqui ...-->
  <script>
    var a = "abc";
    switch (a) {
      case "abc":
        alert("string abc");
        break;
      case 3:
        alert("inteiro 3");
        break;
      default:
        alert("outro");
    }
  </script>
</body>
</html>
```

- Para cada comparação há uma instrução `case`.
  - Cada instrução `case` deve ser separada por uma instrução `break`. Caso contrário, o programa continuará a fazer as comparações seguintes.
- A instrução `default` será executada caso nenhuma das instruções de comparação tenha sido válida.
  - Esta instrução não precisa do separador `break` por ser a instrução final.



# Sintaxe da linguagem Javascript

## Ciclos

Para implementar ciclos, a linguagem JS suporta as instruções `while`, `do-while`, e `for`:

```
do {  
    /* instruções */  
} while (condição);
```

```
while (condição) {  
    /* instruções */  
}
```

```
for (início; comparação; incremento) {  
    /* instruções */  
}
```

### Diferenças entre os diversos tipos de ciclos:

- `do-while` – as instruções do ciclo são executadas pelo menos uma vez porque a `condição` de comparação é executada no fim do ciclo;
- `while` – as instruções do ciclo são executadas 0 ou mais vezes, pois o ciclo só se realiza se a `condição` se verificar à partida;
- `for` – as instruções do ciclo são executadas um número fixo de vezes – desde o `início` até à `comparação` com um `incremento`.

# Interação com o DOM

(Document Object Model)

# Document Object Model (DOM)

O grande potencial da linguagem Javascript quando é executado no browser advém da possibilidade de aceder a qualquer elemento da página HTML.

Isso permite manipular, em tempo real, o conteúdo da página, os estilos e as marcas após a página ter sido carregada sem necessidade de a recarregar novamente.

A característica que possibilita esta interação é chamada de **Document Object Model** (DOM).

Tal como o nome indica, o “modelo de objetos do documento (HTML)” permite que estes sejam utilizados / acedidos / manipulados através de Javascript .

# Como aceder aos objetos do DOM?

No HTML DOM, tudo são **nós**:

O documento em si é um **nó** do tipo **document**;

Todos os elementos HTML são **nós** do tipo **element**;

Todos os atributos HTML são **nós** de tipo **attribute**;

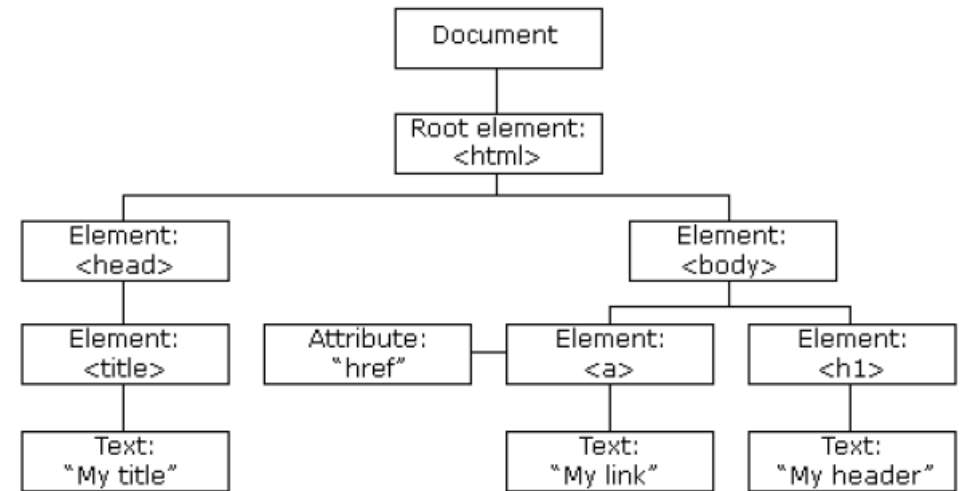
O texto dentro dos elementos HTML é um **nó** do tipo **texto** ;

Comentários são **nós** do tipo **comment**;



# Estrutura DOM de uma página html

```
<!doctype html>
<html lang="en">
<head>
  <title>My title</title>
</head>
<body>
  <h1>My header</h1>
  <a href="http://www.ua.pt">My link</a>
</body>
</html>
```



# O objeto document

Quando um documento HTML é carregado num browser, ele passa a ser um **objeto** do tipo **document**.

Assim, o **objeto document** é o **nó raiz** do documento HTML e o "proprietário" de todos os outros **nós**: **element's**, **text's**, **attribute's**, **comment's**.

O **objeto document** fornece as propriedades e os métodos que permitem aceder a todos os **nós**, através do JavaScript.



# Interação com o Document Object Model

Noção de objeto – propriedades, métodos e eventos

Extendendo o conceito, podemos considerar que cada **nó** de um documento html é, ele próprio, um **objeto**...

Exemplo:

```
<a id="URL_UA" href="http://www.ua.pt">Universidade de Aveiro</a>
```

Cada **objeto** (ainda (?) não foi abordado nas disciplinas de programação) possui um conjunto de **propriedades, métodos e eventos**.

Assim, um elemento `<a>...</a>` é um **objeto**; um elemento `<p>...</p>` também é um **objeto**; ...

Para aceder programaticamente a este **objeto**, cujo `id="URL_UA"` faremos:

```
var x = document.getElementById("URL_UA");
```

# Interação com o Document Object Model

## Elementos inexistentes

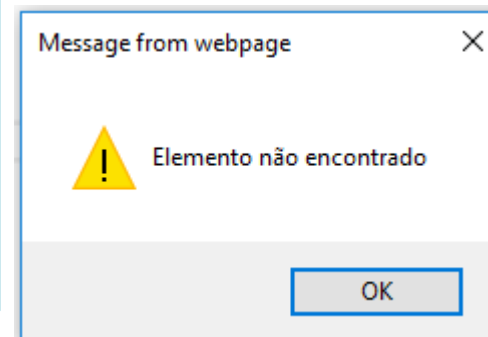
Caso se procure por um elemento com ID inexistente, o valor devolvido pelo método `getElementById` será `null`. Exemplo:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
</head>
<body>
  <input id="op1" value="2" />
  <input id="op2" value="3" />
  <input id="res" value="" />
  <script>
    var x = document.getElementById("nao-existe");
    if (x == null)
      alert("Elemento não encontrado");
    else
      alert(x.value);
  </script>
</body>
</html>
```

Note a utilização de uma nova função:

- `alert(message)`: Esta função exibe uma caixa de alerta com a mensagem especificada e um botão OK.

As caixas de alerta são usadas quando se deseja garantir que a informação chega ao utilizador. A caixa é aberta e a execução do programa fica suspensa até o botão [Ok] ser carregado.



# Interação com o Document Object Model

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
</head>
<body>
  <input id="op1" value="2" />
  <input id="op2" value="3" />
  <input id="res" value="" />
  <script type="text/javascript" src="dom.js"></script>
</body>
</html>
```

Neste exemplo, o elemento `<script>` é incluído no final do `<body>` depois de todos os outros elementos.

## PORQUÊ?

Como pretendemos atuar por Javascript sobre elementos html representados no DOM (`op1` e `op2`) e como a página é construída de modo incremental e à medida que o documento vai sendo lido pelo browser, é **imprescindível** que os elementos HTML já estejam representados na DOM quando o código JavaScript que os referencia for executado.

O conteúdo do ficheiro `dom.js` possuirá o código seguinte:

```
var x = document.getElementById("op1");
var y = document.getElementById("op2");
console.log(parseFloat(x.value));
console.log(parseFloat(y.value));
```

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
</head>
<body>
  <input id="op1" value="2" />
  <input id="op2" value="3" />
  <input id="res" value="" />
  <script type="text/javascript" src="dom.js"></script>
</body>
</html>
```

```
var x = document.getElementById("op1");
var y = document.getElementById("op2");
console.log(parseFloat(x.value));
console.log(parseFloat(y.value));
```

Note a utilização de um método:

- `document.getElementById`: Procura por um elemento (`getElementById`) no DOM (`document`) que tenha o atributo ID especificado no parâmetro (neste caso, "op1" ou "op2").

Note a utilização de uma função:

- `parseFloat`: Converte uma *String* (ex, `x.value`), num valor real (*float*);

Note ainda como se acede à **propriedade** `value` de cada um dos **objetos** devolvidos.

- No caso de `x`, o valor será "2", enquanto o que no caso de `y` o valor será "3" – que são strings;
- A **propriedade** `value` é de escrita e leitura, o que significa que se pode facilmente alterar o texto apresentado num dado campo `<input>` apenas modificando a **propriedade** `value`.

# Sintaxe da linguagem Javascript

## Eventos

Neste exemplo, o código que se encontra fora de funções é executado automaticamente – logo que o browser interpreta a linha, ou seja, antes de o documento html, que está dentro do `<body></body>` ser representado.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script>
    function soma(x,y){
      return x+y;
    }
    var resultado = soma(3,4);
    console.log(resultado);
  </script>
</head>
<body>

</body>
</html>
```

Este não é o comportamento habitual – porque os elementos do documento html ainda não foram desenhados.

Normalmente o código só deverá ser executado depois de todo o documento estar representado no browser.

Nessa altura, alguma coisa deve acontecer – um **evento**, por exemplo! – e avisar o `<script>` que “já pode” ser executado.

# Sintaxe da linguagem Javascript

## Eventos – window.onload

Este exemplo resolve o problema referido - através da utilização do **evento** `window.onload`.

```
function calculadora() {  
    var x = document.getElementById("op1");  
    var y = document.getElementById("op2");  
    console.log(parseFloat(x.value));  
    console.log(parseFloat(y.value));  
}  
window.onload = calculadora;
```

O **evento** `window.onload` é executado só "quando a janela (`window`) estiver completamente carregada".

Como o DOM está completo, todos os **objetos** da página já foram criados e, portanto, é possível executar qualquer operação sem limitações.

# Sintaxe da linguagem Javascript

## Eventos – window.onload

Com este procedimento, é indiferente a localização do `<script></script>`; tanto pode estar no `<head></head>` como no `<body></body>`, conforme se pode ver nos exemplos.

dom.js

```
function calculadora() {  
    var x = document.getElementById("op1");  
    var y = document.getElementById("op2");  
    console.log(parseFloat(x.value));  
    console.log(parseFloat(y.value));  
}  
window.onload = calculadora;
```

```
<!doctype html>  
<html lang="en">  
<head>  
</head>  
<body>  
    <input id="op1" value="2" />  
    <input id="op2" value="3" />  
    <input id="res" value="" />  
    <script type="text/javascript" src="dom.js"></script>  
</body>  
</html>
```

```
<!doctype html>  
<html lang="en">  
<head>  
    <script type="text/javascript" src="dom.js"></script>  
</head>  
<body>  
    <input id="op1" value="2" />  
    <input id="op2" value="3" />  
    <input id="res" value="" />  
</body>  
</html>
```

# Sintaxe da linguagem Javascript

## Glossário de Eventos

### Mouse Events

Event	Description
<b>onclick</b>	The event occurs when the user clicks on an element
<b>oncontextmenu</b>	The event occurs when the user right-clicks on an element to open a context menu
<b>ondblclick</b>	The event occurs when the user double-clicks on an element
<b>onmousedown</b>	The event occurs when the user presses a mouse button over an element
<b>onmouseenter</b>	The event occurs when the pointer is moved onto an element
<b>onmouseleave</b>	The event occurs when the pointer is moved out of an element
<b>onmousemove</b>	The event occurs when the pointer is moving while it is over an element
<b>onmouseover</b>	The event occurs when the pointer is moved onto an element, or onto one of its children
<b>onmouseout</b>	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children
<b>onmouseup</b>	The event occurs when a user releases a mouse button over an element

### Frame/Object Events

Event	Description
<b>onabort</b>	The event occurs when the loading of a resource has been aborted
<b>onbeforeunload</b>	The event occurs before the document is about to be unloaded
<b>onerror</b>	The event occurs when an error occurs while loading an external file
<b>onhashchange</b>	The event occurs when there has been changes to the anchor part of a URL
<b>onload</b>	The event occurs when an object has loaded
<b>onpageshow</b>	The event occurs when the user navigates to a webpage
<b>onpagehide</b>	The event occurs when the user navigates away from a webpage
<b>onresize</b>	The event occurs when the document view is resized
<b>onscroll</b>	The event occurs when an element's scrollbar is being scrolled
<b>onunload</b>	The event occurs once a page has unloaded (for <body>)

### Keyboard Events

Event	Description
<b>onkeydown</b>	The event occurs when the user is pressing a key
<b>onkeypress</b>	The event occurs when the user presses a key
<b>onkeyup</b>	The event occurs when the user releases a key



# Sintaxe da linguagem Javascript

## Eventos - onclick

Considere o seguinte excerto de HTML:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script>
    function calcula() {
      var x = document.getElementById("op1");
      var y = document.getElementById("op2");
      console.log(parseFloat(x.value));
      console.log(parseFloat(y.value));
    }
  </script>
</head>
<body>
  <input id="op1" value="2" />
  <span id="op-view">+</span>
  <input id="op2" value="3" />
  <input id="res" value="" /><br />
  <button onclick="calcula()">Calcular</button>
</body>
</html>
```

Repare como o elemento `<button>` possui um evento `onclick` que está programado para executar a função `calcula()`.

Isto significa que quando o utilizador carregar no botão, o evento `onclick` será disparado e a função `calcular()` será executada.

Neste exemplo, como o javascript só é executado quando se carregar no botão, também é indiferente a localização do `<script></script>`; tanto pode estar no `<head></head>` como no `<body></body>`, pois o utilizador só carregará no botão depois dele estar visível.

# Sintaxe da linguagem Javascript

## Eventos - onclick

### Exemplos:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Javascript onclick event</title>
  <script>
    function myFunction() {
      document.getElementById("demo").innerHTML = "Muito bem";
    }
  </script>
</head>
<body>
  <button onclick="myFunction()">Click Me</button>
  <p id="demo"></p>
</body>
</html>
```

Neste exemplo o evento `onclick` chama a função `myFunction()` que, por sua vez, altera o conteúdo do parágrafo cujo identificador (id) é "demo".

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Javascript onclick event</title>
</head>
<body>
  <button onclick="this.innerHTML='Carregado!'">Carregar</button>
</body>
</html>
```

Neste exemplo o evento `onclick` atua sobre o conteúdo do elemento `<button></button>`. O código javascript está inline e fará com que o texto do botão seja alterado depois de carregado pela primeira vez.

# Sintaxe da linguagem Javascript

## Eventos – onmouseover / onmouseout

### Mudança de estilo inline

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Javascript onmouseover event</title>
  <style>
    span { width:120px; line-height:120px; border-radius:60px;
           display:inline-block; text-align:center; border:solid 1px black; }
  </style>
</head>
<body>
  <span onmouseover="this.style.backgroundColor='red'"
        onmouseout="this.style.backgroundColor='white'">Mouse over me!</span>
</body>
</html>
```

Neste exemplo quando o rato passar sobre o elemento (evento **onmouseover**) será a cor de fundo do elemento **<span></span>**, que passará a **red**.

Quando o rato sair de cima do elemento (evento **onmouseout**) a cor de fundo do elemento passará à cor **white**.

# Sintaxe da linguagem Javascript

## Eventos – onmouseover / onmouseout

### Mudança de classe inline

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Javascript onmouseover event</title>
  <style>
    span { width:120px; line-height:120px; border-radius:60px;
      display:inline-block; text-align:center; border:solid 1px black; }
    .mouseOver { background-color:red; }
  </style>
</head>
<body>
  <span onmouseover="this.classList.toggle('mouseOver')"
    onmouseout="this.classList.toggle('mouseOver')">Mouse over me!</span>
</body>
</html>
```

Neste exemplo quando o rato passar sobre o elemento `<span></span>` o evento `onmouseover` será ativado, e será adicionada a classe `"mouseOver"`.

Quando o rato sair de cima do elemento através da ativação do evento `onmouseout`, será removida a classe `"mouseOver"`.

# Sintaxe da linguagem Javascript

## Eventos - onchange

Podemos generalizar este exemplo de forma a que se possa especificar a operação a executar através de campos de selecção:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Javascript test file</title>
  <script src="myScript.js" type="text/javascript"></script>
</head>
<body>
  <input id="op1" value="2" />
  <select id="operacao" onchange="getOperacao()">
    <option value="+> Soma </option>
    <option value="-> Subtração </option>
    <option value="*> Multiplicação </option>
    <option value=":> Divisão </option>  </select>
  <input id="op2" value="3" />
  <input id="res" value="" /><br />
  <button onclick="calcula()">Calcular</button>
</body>
</html>
```

```
var operacao;
function getOperacao() {
  var e = document.getElementById("operacao");
  operacao = e.options[e.selectedIndex].value;
}
function calcula() {
  /* Vamos precisar de código aqui...
   na aula prática é um bom local para o fazer. */
}
```

myScript.js

# Sintaxe da linguagem Javascript

## Propriedade `event.target`

A propriedade de `event.target` devolve o objeto que despoletou um evento / trigger.

Esta propriedade é muito útil quando temos código comum a vários objetos e apenas variamos o seu nome.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Jogo javascript</title>
<script>
  function scramble() {}
  function setCurrentDiv() {}
</script>
</head>
<body>
  <button onclick="scramble()">Baralhar</button>
  <span id="currentInfo"></span>
  <div id="azul" onmouseover="setCurrentDiv()" onmouseout="clearSelected()"></div>
  <div id="vermelho" onmouseover="setCurrentDiv()" onmouseout="clearSelected()"></div>
  <div id="verde" onmouseover="setCurrentDiv()" onmouseout="clearSelected()"></div>
  <div id="amarelo" onmouseover="setCurrentDiv()" onmouseout="clearSelected()"></div>
</body>
</html>
```

# Funções e constantes matemáticas

## Glossário

Method	Description
<b>abs(x)</b>	Returns the absolute value of x
<b>acos(x)</b>	Returns the arccosine of x, in radians
<b>asin(x)</b>	Returns the arcsine of x, in radians
<b>atan(x)</b>	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
<b>atan2(y,x)</b>	Returns the arctangent of the quotient of its arguments
<b>ceil(x)</b>	Returns the value of x rounded up to its nearest integer
<b>cos(x)</b>	Returns the cosine of x (x is in radians)
<b>exp(x)</b>	Returns the value of $E^x$
<b>floor(x)</b>	Returns the value of x rounded down to its nearest integer
<b>log(x)</b>	Returns the natural logarithm (base E) of x
<b>max(x,y,z,...,n)</b>	Returns the number with the highest value
<b>min(x,y,z,...,n)</b>	Returns the number with the lowest value
<b>pow(x,y)</b>	Returns the value of x to the power of y
<b>random()</b>	Returns a random number between 0 and 1
<b>round(x)</b>	Returns the value of x rounded to its nearest integer
<b>sin(x)</b>	Returns the sine of x (x is in radians)
<b>sqrt(x)</b>	Returns the square root of x
<b>tan(x)</b>	Returns the tangent of an angle

```
Math.E      // returns Euler's number
Math.PI     // returns PI
Math.SQRT2  // returns the square root of 2
Math.SQRT1_2 // returns the square root of 1/2
Math.LN2    // returns the natural logarithm of 2
Math.LN10   // returns the natural logarithm of 10
Math.LOG2E  // returns base 2 logarithm of E
Math.LOG10E // returns base 10 logarithm of E
```

Fonte: [http://www.w3schools.com/js/js\\_math.asp](http://www.w3schools.com/js/js_math.asp)

# Funções e constantes numéricas

## Glossário

### JavaScript Global Functions

Method	Description
<b>eval()</b>	Evaluates a string and executes it as if it was script code
<b>isNaN()</b>	Determines whether a value is an illegal number
<b>Number()</b>	Returns a number, converted from its argument.
<b>parseFloat()</b>	Parses its argument and returns a floating point number
<b>parseInt()</b>	Parses its argument and returns an integer

### Number Properties

Property	Description
<b>MAX_VALUE</b>	Returns the largest number possible in JavaScript
<b>MIN_VALUE</b>	Returns the smallest number possible in JavaScript
<b>NEGATIVE_INFINITY</b>	Represents negative infinity (returned on overflow)
<b>NaN</b>	Represents a "Not-a-Number" value
<b>POSITIVE_INFINITY</b>	Represents infinity (returned on overflow)

### Number Functions

Method	Description
<b>isFinite()</b>	Checks whether a value is a finite number
<b>isInteger()</b>	Checks whether a value is an integer
<b>isNaN()</b>	Checks whether a value is Number.NaN
<b>isSafeInteger()</b>	Checks whether a value is a safe integer
<b>toExponential(x)</b>	Converts a number into an exponential notation
<b>toFixed(x)</b>	Formats a number with x numbers of digits after the decimal point
<b>toPrecision(x)</b>	Formats a number to x length
<b>toString()</b>	Converts a number to a string
<b>valueOf()</b>	Returns the primitive value of a number



# Funções e contantes para manipulação de strings

## Glossário

Method	Description
<b>charAt()</b>	Returns the character at the specified index (position)
<b>charCodeAt()</b>	Returns the Unicode of the character at the specified index
<b>concat()</b>	Joins two or more strings, and returns a new joined strings
<b>endsWith()</b>	Checks whether a string ends with specified string/characters
<b>fromCharCode()</b>	Converts Unicode values to characters
<b>includes()</b>	Checks whether a string contains the specified string/characters
<b>indexOf()</b>	Returns the position of the first found occurrence of a specified value in a string
<b>lastIndexOf()</b>	Returns the position of the last found occurrence of a specified value in a string
<b>localeCompare()</b>	Compares two strings in the current locale
<b>match()</b>	Searches a string for a match against a regular expression, and returns the matches
<b>repeat()</b>	Returns a new string with a specified number of copies of an existing string
<b>replace()</b>	Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced

Method	Description
<b>search()</b>	Searches a string for a specified value, or regular expression, and returns the position of the match
<b>slice()</b>	Extracts a part of a string and returns a new string
<b>split()</b>	Splits a string into an array of substrings
<b>startsWith()</b>	Checks whether a string begins with specified characters
<b>substr()</b>	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
<b>substring()</b>	Extracts the characters from a string, between two specified indices
<b>toLocaleLowerCase()</b>	Converts a string to lowercase letters, according to the host's locale
<b>toLocaleUpperCase()</b>	Converts a string to uppercase letters, according to the host's locale
<b>toLowerCase()</b>	Converts a string to lowercase letters
<b>toString()</b>	Returns the value of a String object
<b>toUpperCase()</b>	Converts a string to uppercase letters
<b>trim()</b>	Removes whitespace from both ends of a string
<b>valueOf()</b>	Returns the primitive value of a String object

Property	Description
<b>length</b>	Returns the length of a string