Introdução às tecnologias Web - ITW

Aula 5 – Programação web utilizando Javascript

23/10/2017

Sumário

- Linguagens de programação
  - Breves noções
- A linguagem Javascript
  - Introdução
  - Sintaxe JavaScript
  - Interacção com o DOM
  - Eventos

Linguagens de programação
O que são?

Uma “linguagem de programação” é uma linguagem que possui uma sintaxe (formato) e uma semântica (significado), e é usada para expressar uma sequência de ações computacionais que formam um “programa”.

- 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

O **compilador** de uma linguagem compilada 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.

O código fonte escrito pelo programador pode ser executado diretamente por um **interpretador** que lê trechos do código fonte em tempo de execução, converte em um formato que o computador consegue ler (compilação em tempo de execução) e realiza sua execução.

Linguagens de programação
Linguagens compiladas ou de script?

Se uma precisa de um passo de compilação antes de executar ela é chamada de linguagem compilada. Senão, é chamada de linguagem de script.

- Uma linguagem de script, normalmente, não necessita de um passo específico de compilação para executar um programa.
- Linguagens de script costumam ter performance inferior a linguagens compiladas pois exigem mais passos para disponibilizar o programa em tempo de execução. Porém, são muito mais produtivas, pois eliminam a necessidade de compilar o código fonte de cada vez que uma alteração é feita.

Linguagens de programação
Linguagens "tipadas" (do inglês typed):

Nas linguagens fortemente 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)

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

Ruby (untyped)

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

Na 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).
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**.

Em linguagens fracamente tipadas, a possibilidade de erros em tempo de execução é muito maior sendo por isso recomendável a realização de testes exaustivos antes de colocar um sistema em produção.

A linguagem Javascript

A linguagem Javascript
Introdução

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

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

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?

Os elementos básicos do JavaScript podem ser estendidos com objetos adicionais para uma variedade de propósitos, por exemplo:

- Um programa em JavaScript a correr no **cliente** 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, input adicionado, e navegação na página.*
- Um programa em JavaScript a correr no **servidor** fornece objetos relevantes para aceder a funcionalidades diversas.
  - *Por exemplo, extensões do lado do servidor permitem que uma aplicação comunique com uma base de dados, garante continuidade de informação entre invocações da aplicação, ou executa manipulações de ficheiros no servidor.*

Porquê JavaScript?

O JavaScript é uma das 3 linguagens que todos os desenvolvedores de páginas web devem aprender:

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

Vantagens e desvantagens do Javascript

Como é uma linguagem interpretada, é processada aos blocos e compilada 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

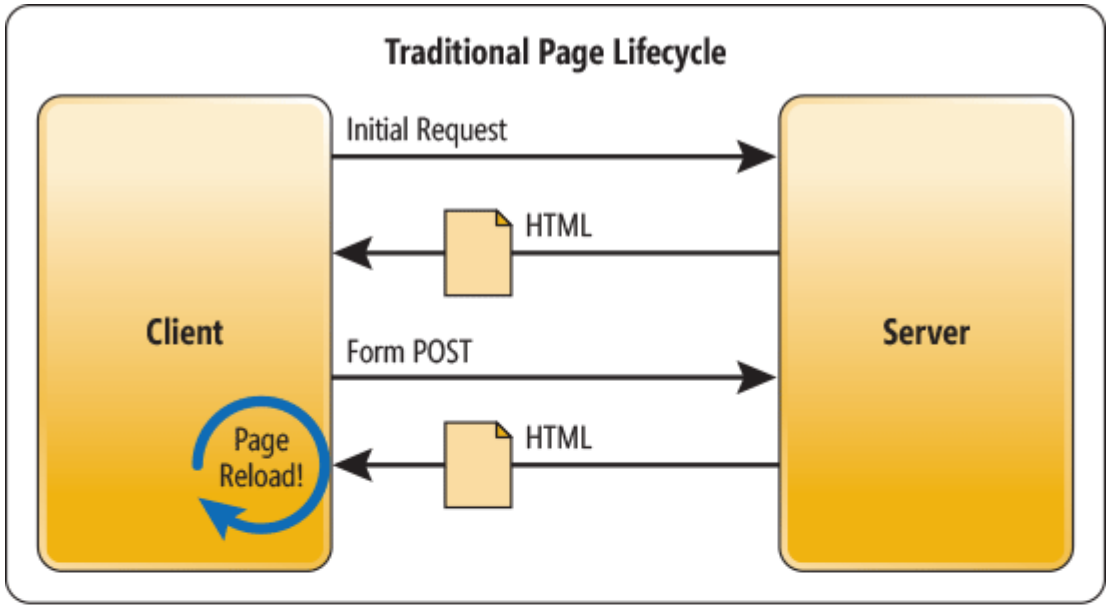
A linguagem Javascript foi originalmente implementada como parte dos web browsers para que estes pudessem executar programas (que em javascript se denominam scripts) do lado do cliente e interagissem com o utilizador sem a necessidade deste recorrer ao servidor.

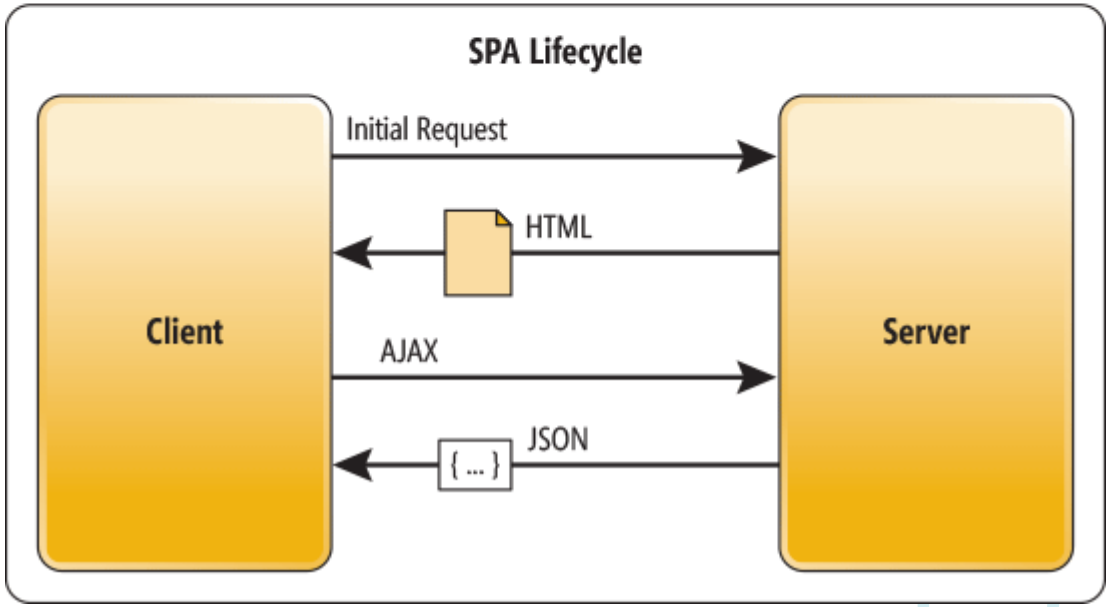
Um script javascript permite:

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

Utilização do javascript

A linguagem javascript começa também a ser utilizada do lado do servidor através de ambientes como, por exemplo, o node.js ou em aplicações cliente de página simples (SPA – Single Page Applications).





É objetivo último desta unidade curricular elaborar, no final, uma SPA!
Votaremos a este assunto na aula 11

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>
    /* O 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>
```

Obtenção do script de fonte externa
(o jQuery e o bootstrap)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title></title>
    <!-- Bootstrap -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet" />
     <!-- Font-awesome -->
    <link href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.3/css/font-awesome.min.css" rel="stylesheet" />
</head>
<body>
    <!-- INICIO -->

    <!-- FIM -->
    <!-- jQuery (necessário para os plugins JavaScript do Bootstrap) -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
    <!- Scripts do Bootstrap-->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.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, 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 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 ≠ sim ≠ SIM).

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> <operação> <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
Funções

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

Tal como a declaração das variáveis é indicada pela palavra reservada `var`, a declaração de funções faz uso da palavra reservada `function`, tal como descrito no exemplo.

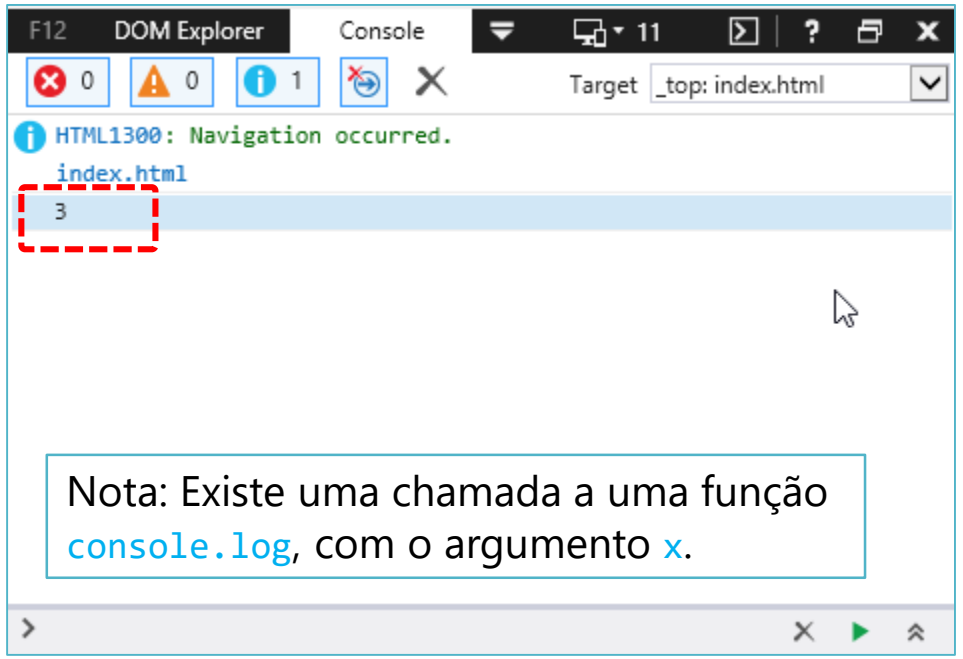
Estes elementos são constituídos por um `nome`, uma `lista de argumentos` e um corpo.

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

}
```

Sintaxe da linguagem Javascript
Declaração de variáveis

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>
```



Nota: Existe uma chamada a uma função `console.log`, com o argumento `x`.

Sintaxe da linguagem Javascript
Operações

Podem ser aplicados operadores aritméticos às variáveis, tais como a soma (+), ou a subtração (-).

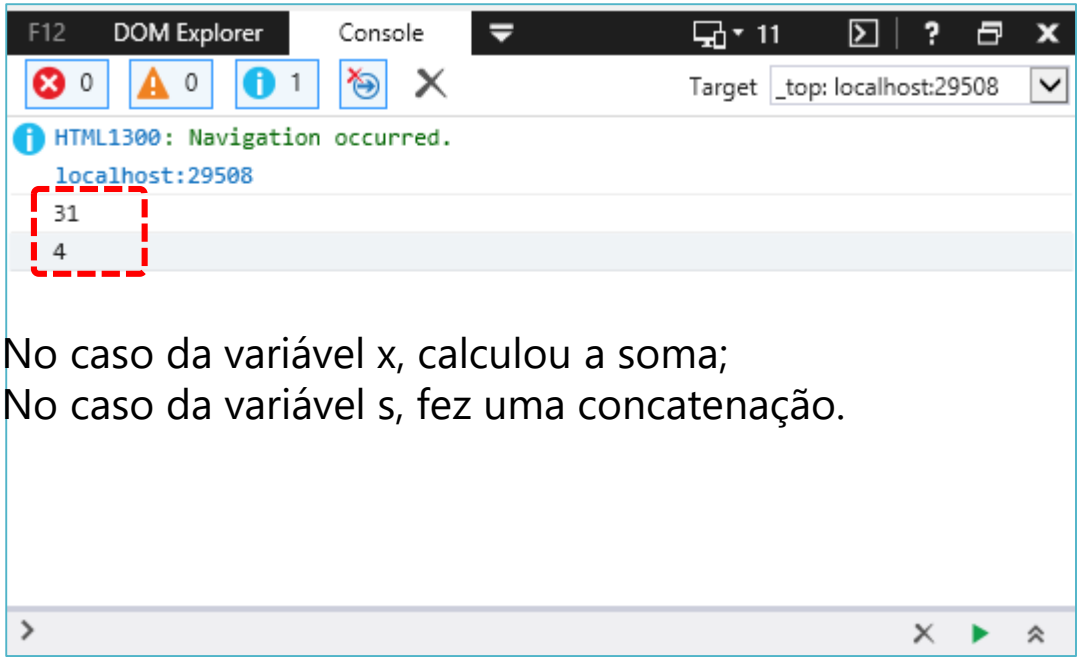
- O significado desta operação irá variar com o tipo de variável (que depende do seu conteúdo atual).

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>
```

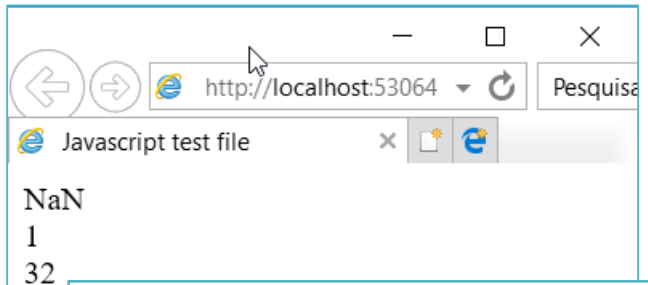


No caso da variável x, calculou a soma;
No caso da variável s, fez uma concatenação.

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>
```



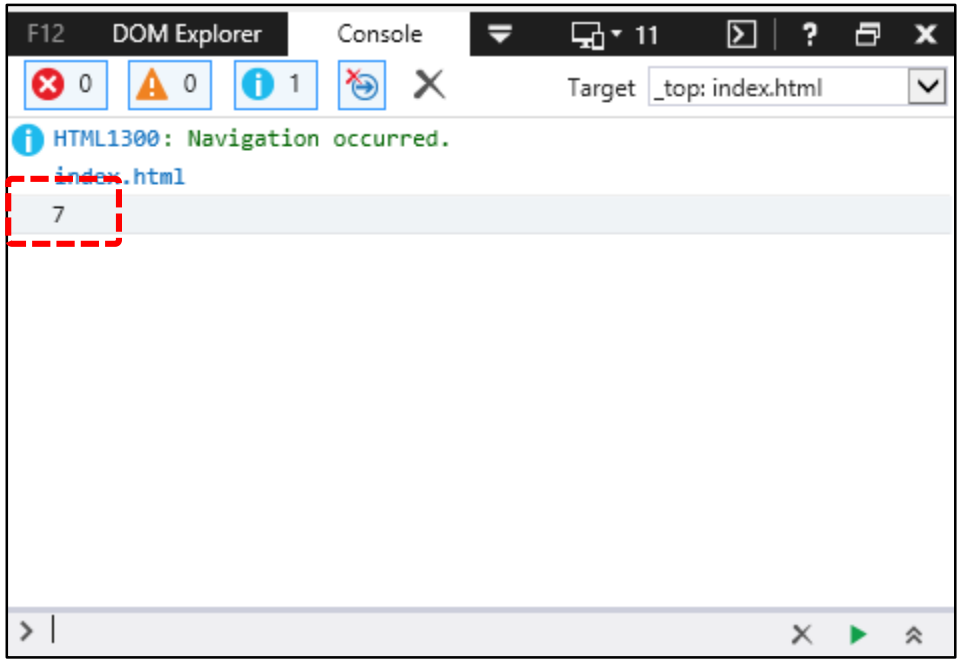
Neste caso, `document.write(…)`, ao invés do resultado ser apresentado na consola da janela de depuração, o resultado é "escrito" na janela de visualização do browser.

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) {
  /* Instruções no caso positivo */
} else {
  /* Instruções no 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)

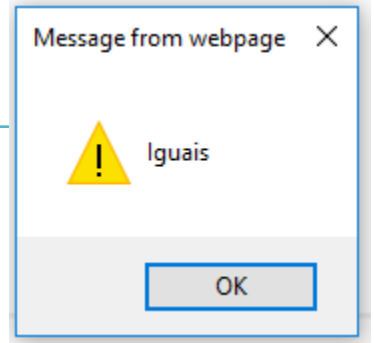
Considere o seguinte excerto:

```
<!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 !==.

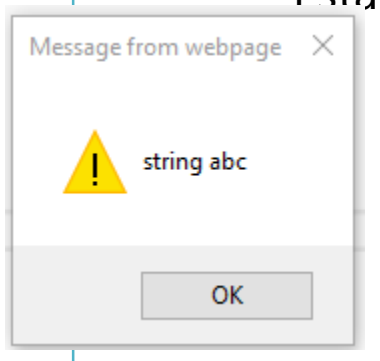
Na linguagem Javascript diz-se que estes comparadores verificam se o valor é igual e o tipo idêntico. No caso anterior, o valor de a é igual ao de b mas as variáveis não são idênticas.



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>
    <script>
        var a = "abc";
        switch (a) {
            case "abc":
                alert("string abc");
                break;
            case 3:
                alert("inteiro 3");
                break;
            default:
                alert("outro");
        }
    </script>
</head>
<body>
    <!-- Conteúdo html aqui ...-->
</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`.

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 (inicio; 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 web browser advém da possibilidade de aceder a qualquer elemento de uma 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 caraterística 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 to tipo `document`;
- Todos os elementos HTML são nós do tipo `element`;
- Todos os atributos HTML são nós de tipo `attribute`;
- Texto dentro de elementos HTML são nós do tipo `texto` ;
- Comentários são nós do tipo `comment`;

O objeto document

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

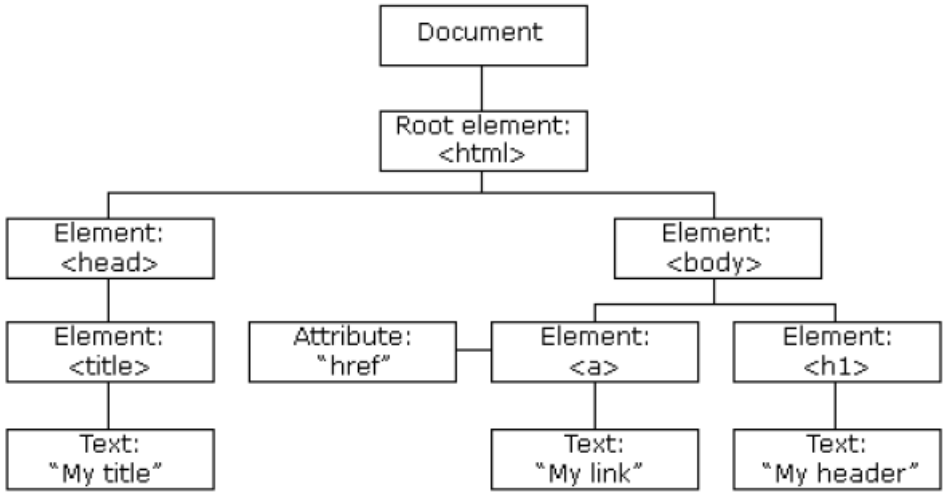
Assim, o objeto `document` é o **nó raiz** do documento HTML e o "proprietário" de todos os outros nós:

(elements, text, atribute, comment).

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

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>
```



Interação com o Document Object Model
Noção de objeto – propriedades, métodos e eventos

Podemos considerar 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) é 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");
```

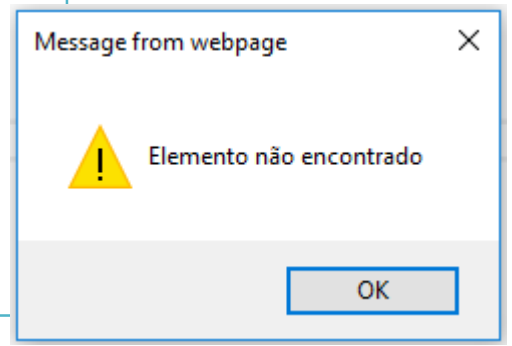
Interação com o Document Object Model
Elementos inexistentes

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 um novo método :

- `alert(message)`: Este método 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.



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));
```

```html
<!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>
```

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

Note a utilização de dois métodos :

- `document.getElementById`: Procura por um elemento (getElementById) no DOM (document) que tenha o atributo ID especicado no parâmetro (neste caso, "op1" ou "op2").
- `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;
- Esta `propriedade` 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 – tão logo o browser interpreta a linha.

```
<!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 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 da linha de `<script></script>`; tanto pode estar no `<head></head>` como no `<body></body>`, conforme se pode ver nos exempos.

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

Frame/Object Events

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

Keyboard Events

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

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>
```

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

Neste caso, o evento `onclick` atua sobre o conteúdo do elemento `<p></p>` sem ser necessária a utilização de uma função. Note que o evento neste caso nem sequer está colocado num botão!!!

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 a marca `<button>` possui um evento `onclick` que está definido para executar a função "`calcular()`".

Isto significa que quando o utilizador clicar com o apontador em cima do botão, o evento `onclick` será disparado e a função `calcular()` será chamada.

Neste caso, como o código javascript só é executado quando se carregar no botão, é indiferente se o `<script></script>` está no `<head></head>` ou no `<body></body>`.

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>
```

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>
```

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>
```

myScript.js

```
var operacao;
function getOperacao() {
   var e = document.getElementById("operacao");
   operacao = e.options[e.selectedIndex].value;
}
function calcula() {
    /* Vamos precisar de código aqui ... */
}
```

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() {}
      function scramble() {}
</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 contantes matemáticas

Glossário

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

Funções e constantes numéricas
Glossário

JavaScript Global Functions

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

Number Properties

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

Number Methods

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

Funções e contantes para manipulação de strings
Glossário

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

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

Property	Description
length	Returns the length of a string