Introdução às tecnologias Web - ITW

Aula 5 – Javascript

24/10/2016

Sumário

A linguagem Javascript

- Introdução
- Sintaxe JavaScript
- Interacção com o DOM
- Temporizadores
- Eventos

Introdução - A linguagem Javascript

O JavaScript (JS) é uma linguagem interpretada[1].

Por ser uma linguagem interpretada, não são necessários, antes da sua execução, os passos habituais nas demais linguagens: compilação e produção de um objeto executável, tal como acontece com as linguagens Java ou C.

[1]- O JS é baseado na ECMAScript padronizada pela *Ecma international* nas especificações ECMA-262 e ISO/IEC 16262.

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

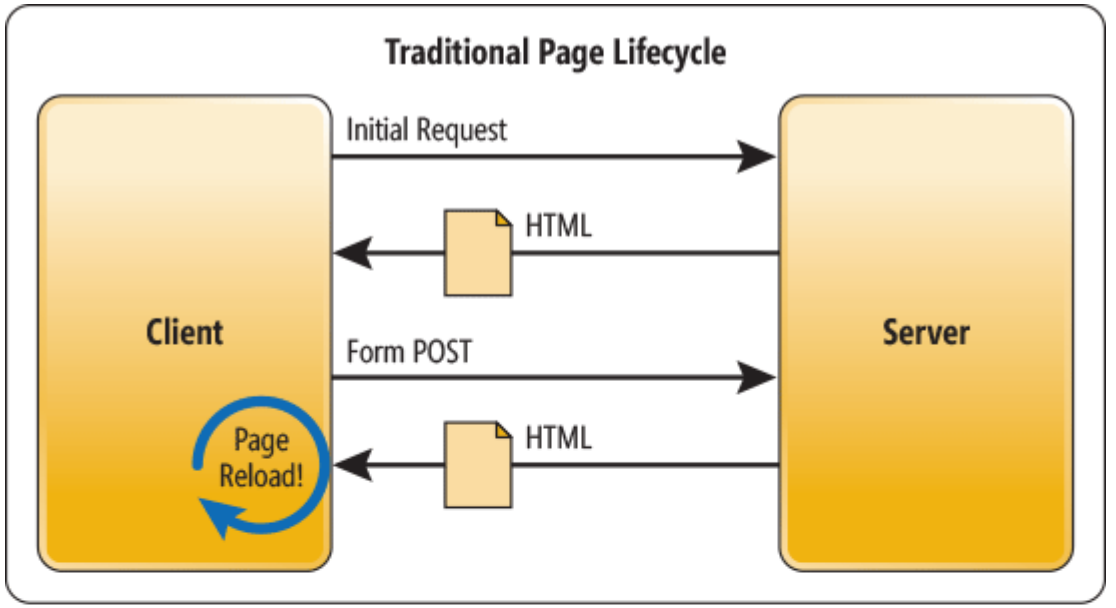
A linguagem Javascript (JS) foi originalmente implementada como parte dos web browsers para que estes pudessem executar programas (que em javascript se denominam scripts) do lado do cliente e interagissem com o utilizador sem a necessidade deste recorrer ao servidor.

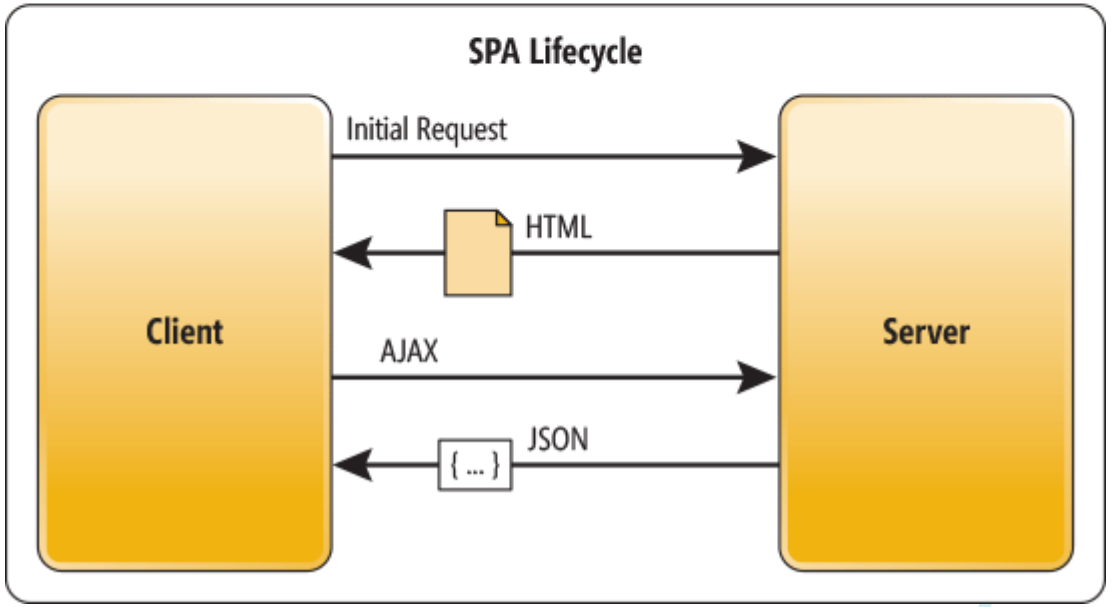
Um script JS permite:

- 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 clientede página simples (SPA – Single Page Applications).





Inclusão de script javascript numa página html

O processo de inclusão numa página html é semelhante à da inclusão dos estilos CSS, 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
(da aula passada, quando se integrou 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>Bootstrap empty template</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/3.1.1/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</body>
</html>
```

Versatilidade vs segurança

A linguagem JS é bastante poderosa e o facto poder ser executada em qualquer web browser de qualquer sistema operativo, permite desenvolver aplicações que podem ser distribuídas de forma muito eficaz.

- No entanto, o código JS é sempre enviado ao cliente na sua forma textual, podendo, por isso, ser rapidamente copiado.
- 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:

- Content/bootstrap.min.css
- Scripts/bootstrap.js

A linguagem Javascript

A sintaxe da linguagem JS 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 JS é baseada em instruções, que são organizadas em linhas.

- Cada linha corresponde a uma (ou mais) instrução/instruções, podendo cada uma das instruções ser terminada com o caráter ; (ponto-e-vírgula).
- A utilização deste caráter é facultativo mas muito recomendado.

JS é case-sensitive, o que significa que se deve ter cuidado na escrita.

Sintaxe da linguagem Javascript
Declaração de variáveis

Este exemplo declara uma variável x, atribui-lhe um valor e apresenta o resultado na consola do navegador.

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

</body>
</html>
```

A declaração de variáveis é diferente das linguagens declarativas. É feita através da utilização da palavra reservada **var** seguida pelo `nome_da_variável`.

Isto deve-se ao facto de o JS ser uma **linguagem com tipos dinâmicos**, não sendo necessário declarar explicitamente qual o tipo da variável.
Todas as variáveis são declaradas da mesma forma, sendo o conteúdo quem define como ela será utilizada.

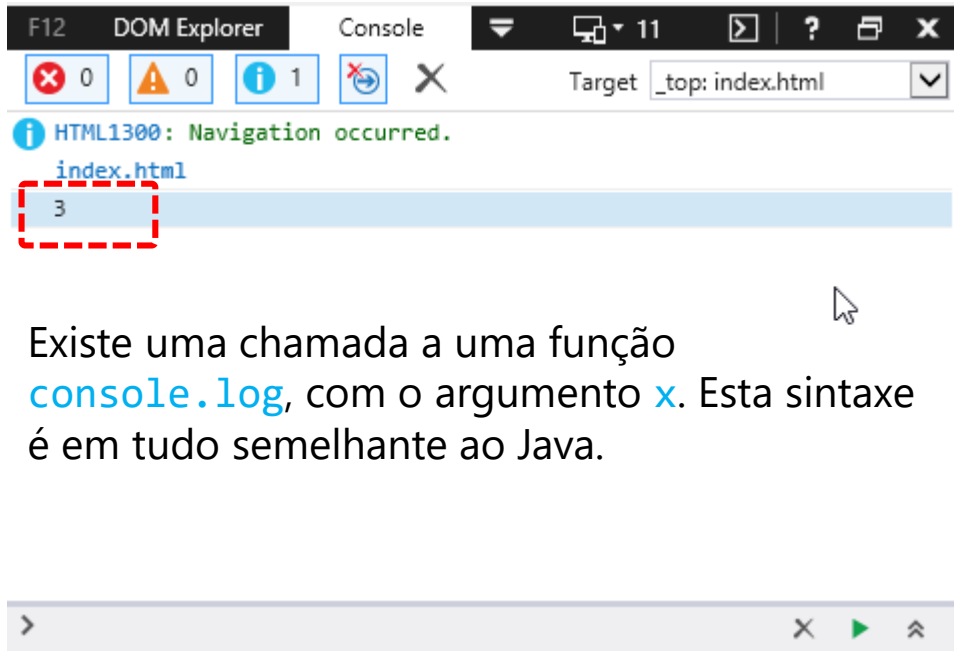
A atribuição de valores a uma variável faz-se de modo convencional:

`<nome_da_variável> <operação> <valor>`

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>
        /* Comentário */
        var x;
        x = 3;
        console.log(x);
    </script>
</head>
<body>

</body>
</html>
```



Existe uma chamada a uma função `console.log`, com o argumento `x`. Esta sintaxe é em tudo semelhante ao Java.

Sintaxe da linguagem Javascript
Operações

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

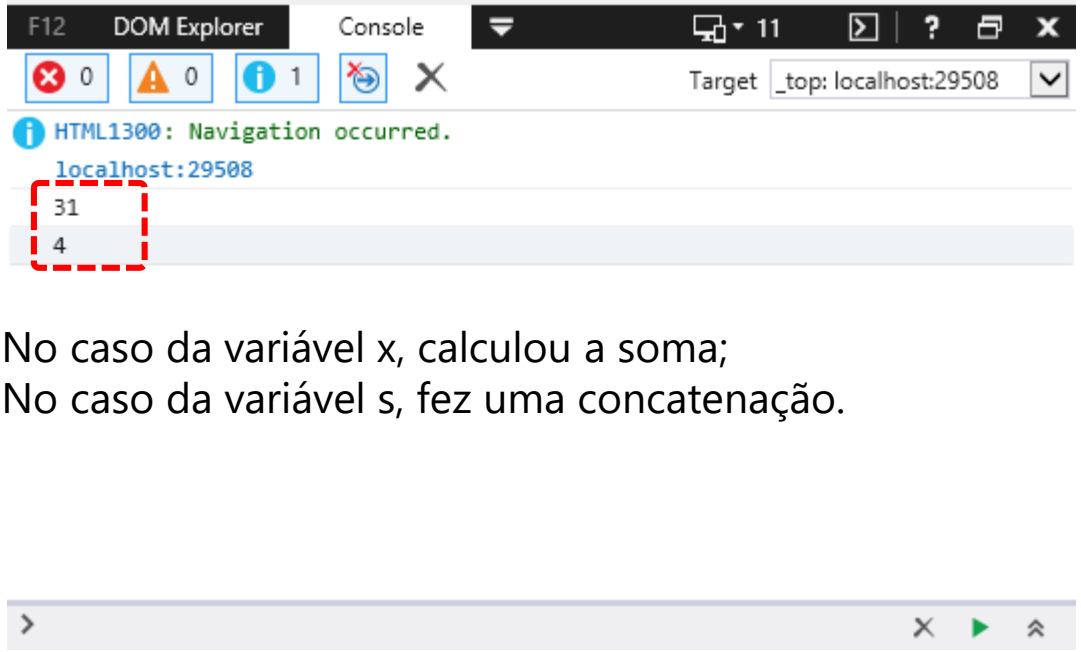
- O significado desta operação irá variar com o tipo de variável (que depende do seu conteúdo atual).

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>
```

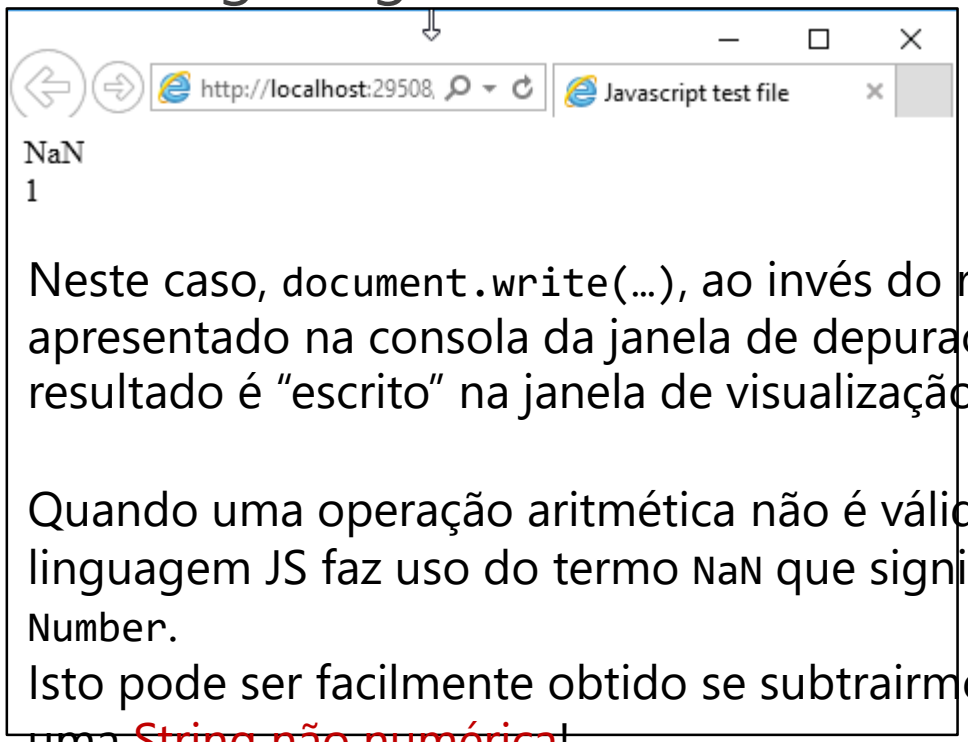


No caso da variável x, calculou a soma;
No caso da variável s, fez uma concatenação.

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);
    </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 JS 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
Funções

De forma a melhor organizar o código, e evitar a replicação desnecessária, é possível organizar um programa em funções. Estes elementos são constituídos por um nome, uma lista de argumentos e um corpo.

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

}
```

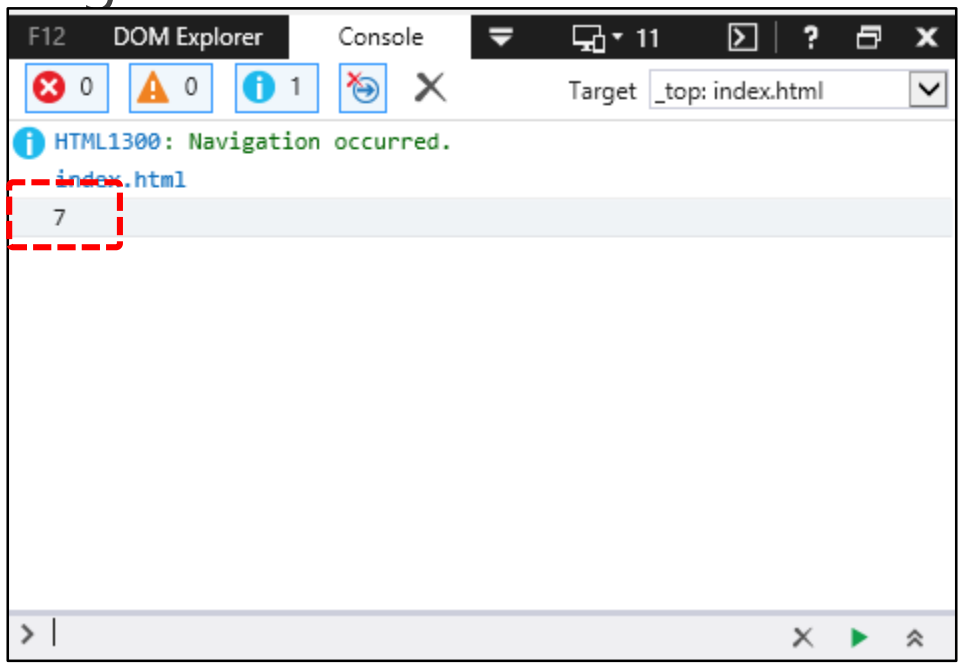
Tal como a declaração das variáveis é indicada pela palavra reservada var, a declaração de funções faz uso da palavra reservada function, tal como descrito no exemplo.

- Comparando com a linguagem Java, verifica-se que, no Javascript, não é necessário declarar qual o tipo de dados de retorno da função, nem os tipos de dados dos parâmetros.

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

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 */
}
```

As chavetas {} podem ser omitidas caso apenas exista uma instrução a executar.
Os operadores de comparação são:

- < → Menor
- > → Maior
- >= → Maior
- == → Igual, etc...

Sintaxe da linguagem Javascript
Exemplo de utilização de instruções condicionais

Considere o seguinte excerto:

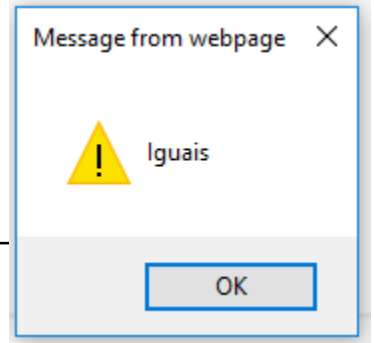
```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Javascript test file</title>
    <script>
     var a = "3";
     var b = 3;
     if (a == b)
          alert("Iguais");
     else
          alert("Diferentes");
    </script>
</head>
<body>

</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 !==.

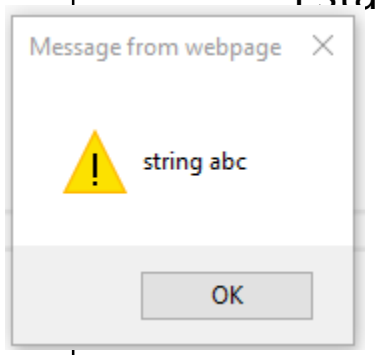
Na linguagem JS diz-se que estes comparadores verificam se o valor é igual e o tipo idêntico. No caso anterior, o valor de a é igual ao de b mas as variáveis não são idênticas.



Sintaxe da linguagem Javascript
Condições

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>
```



- 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 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 DOM significa"modelo de objetos da página (HTML)".
- Estes objetos podem depois ser utilizados / acedidos / manipulados através de Javascript .

Interação com o Document Object Model

O conceito de **objeto** ainda (?) não foi abordado nas disciplinas de programação. Por simplicidade, consideremos que cada um dos elementos do documento html é um objeto que possui um conjunto de **propriedades**, **métodos** e **eventos**.

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

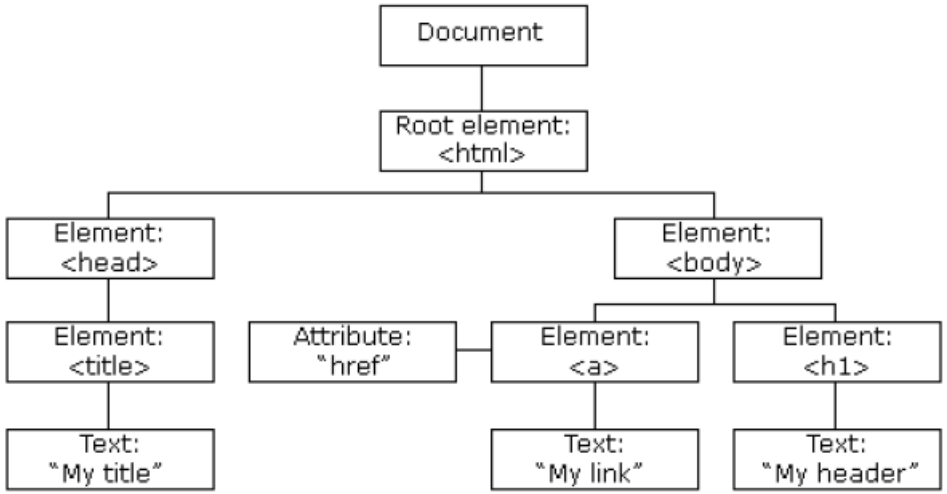
Exemplo:

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

Através de Javascript é possível consultar valor das **propriedades** (como por ex., o atributo `href`), enviar ordens para ações, através dos **métodos**, e ser avisado de alterações nele ocorridas através dos **eventos**.

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

Tal como apresentado anteriormente, para uma página html, o DOM define uma estrutura hierárquica com pais e filhos.

```
<!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.

Como a página é construída de modo incremental, é **imprescindível** que os elementos HTML já existam na DOM quando o código JavaScript que os refere for executado.

```
(…)
<body>
    <input id="op1" value="2" />
    <input id="op2" value="3" />
    <input id="res" value="" />
    <script type="text/javascript" src="dom.js"></script>
</body>
(…)
```

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));
```

Note a utilização de dois métodos novos:

- `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*);

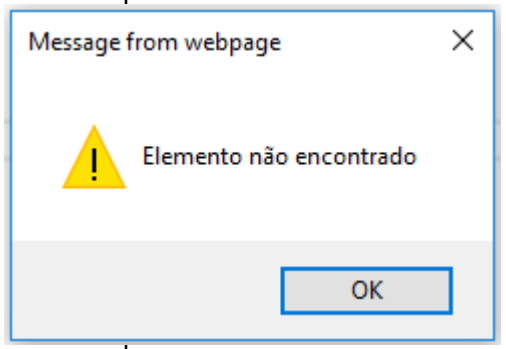
Note ainda como se acede à `propriedade` `value` de cada um dos `objetos` devolvidos.

- 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`.

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>
```



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.

Este pode depois invocar as diversas funções disponíveis.

```
<!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 é comportamento anormal –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 – um `evento`, por exemplo! – deve acontecer e avisar o `<script>` que "já pode" ser executado.

Sintaxe da linguagem Javascript
Eventos

Evento	Descrição
`onchange`	Disparado sempre que o elemento html que está associado muda
`onclick`	Disparado sempre que se clica sobre o elemento html – botões, mas não só!!!
`onmouseover`	Disparado sempre que o rato passa sobre o elemento html
`onmouseout`	Disparado sempre que o rato sai de cima do elemento html
`onkeydown`	Disparado sempre que se carrega numa Tecla do teclado
`onload`	Disparado sempre que termina o carregamento de uma página html

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

Quando a `window` estiver completamente carregada, o evento window.onload é ativado e a funcão calculadora() é executada.

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;
```

Com este procedimento, é indiferente a localização da linha de `<script>`: no `<head>` ou no `<body>`, conforme se pode ver nos exempos.

```
<!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
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 atributo `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.

Masi uma vez, é indiferente se o `<script>` está no `<head>` ou no `<body>`

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 lang="en">
<head>
    <meta charset="UTF-8">
    <title>Javascript test file</title>
    <script>
        function calcula() {
            /* Vamos precisar de código aqui ... */
        }
    </script>
</head>
<body>
    <input id="op1" value="2" />
    <select id="operacao">
        <option value="+"> Soma </option>
        <option value="-"> Subtração </option>
    </select>
    <input id="op2" value="3" />
    <input id="res" value="" /><br />
    <button onclick="calcula()">Calcular</button>
</body>
</html>
```

Neste exemplo, quando o utilizador alterar o conteúdo da caixa de seleção contendo a operação a efetuar, o evento `onchange` será disparado e a função `calcula()` será chamada.

Sintaxe da linguagem Javascript

Eventos - onclick

```
<!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>
```

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

Sintaxe da linguagem Javascript
Eventos – onmouseover / onmouseout

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Javascript onmouseover event</title>
</head>
<body>
    <span onmouseover="this.style.backgroundColor='red'"
          onmouseout="this.style.backgroundColor='white'">Mouse over me!</span>
</body>
</html>
```

Sintaxe da linguagem Javascript
Eventos – onchange

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Javascript onchange event</title>
</head>
<body>
    <select id="operacao" onchange="alert(operacao.value)">
        <option value="+"> Soma </option>
        <option value="-"> Subtração </option>
        <option value="*"> Multiplicação </option>
        <option value=":"> Divisão </option>
    </select>
</body>
</html>
```

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

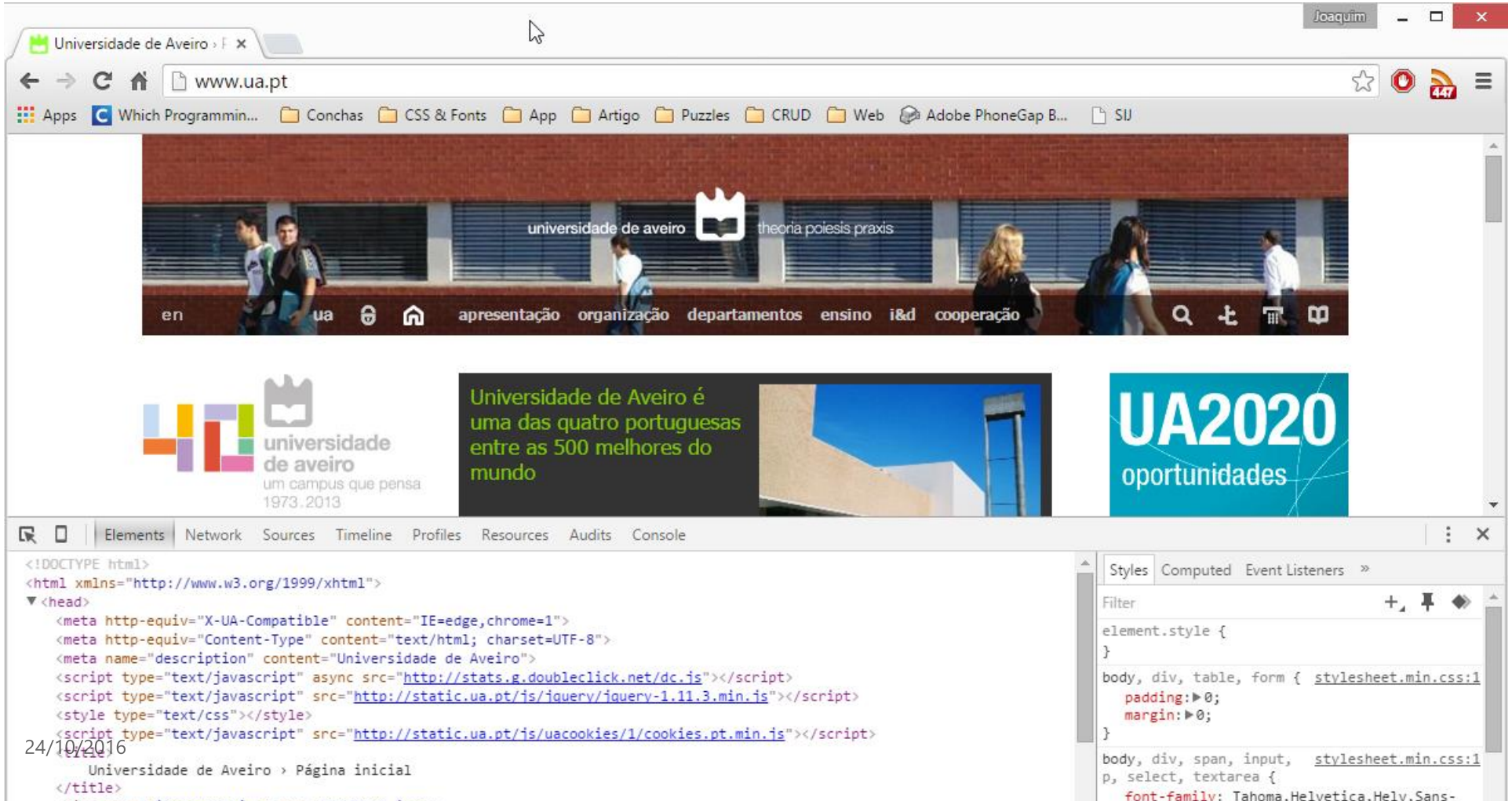
Method	Description
`abs(x)`	Returns the absolute value of x
`acos(x)`	Returns the arccosine of x, in radians
`asin(x)`	Returns the arcsine of x, in radians
`atan(x)`	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
`atan2(y,x)`	Returns the arctangent of the quotient of its 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
```

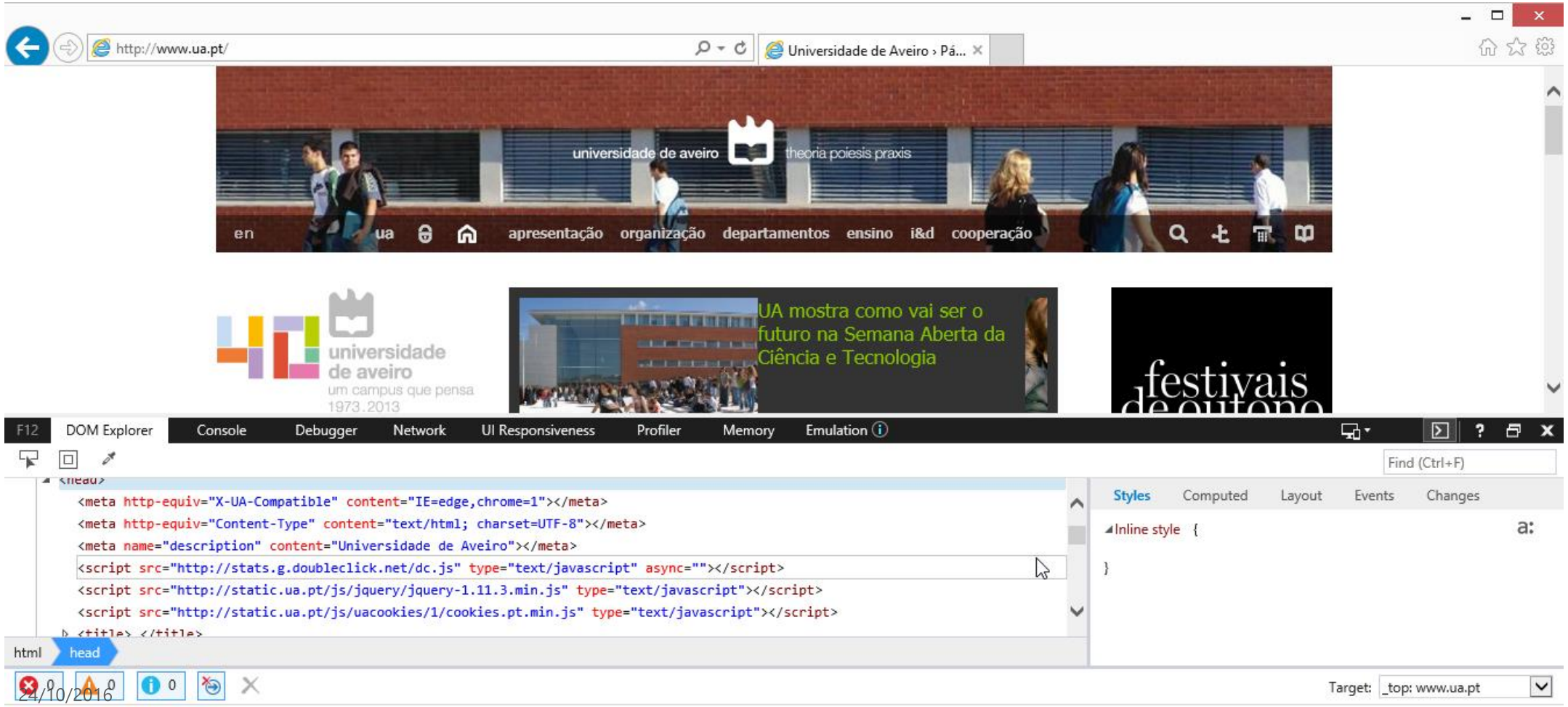
Fonte: http://www.w3schools.com/js/js_math.asp

Utilização da janela de depuração para a solução de problemas no código javascript

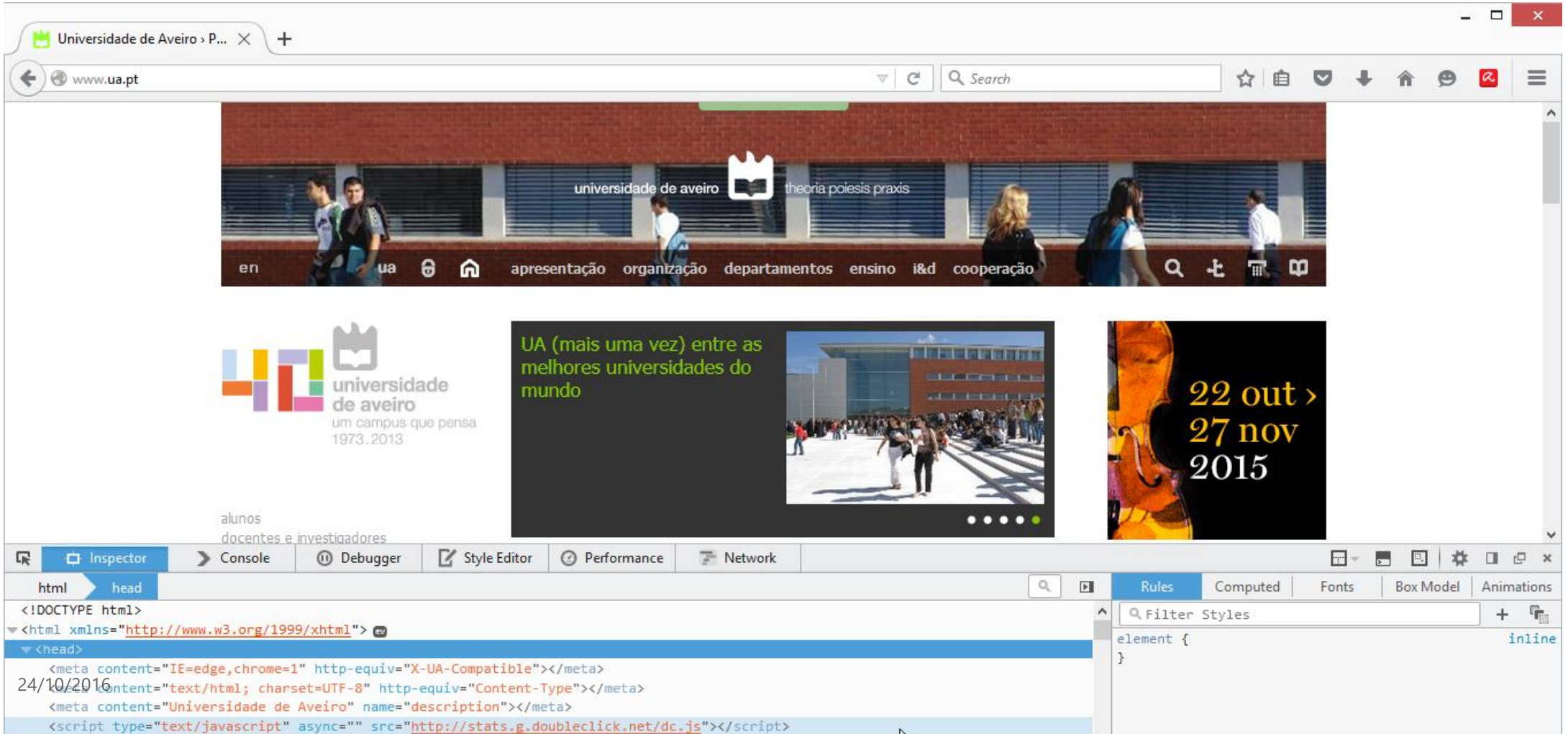
A janela de depuração (F12)
Chrome



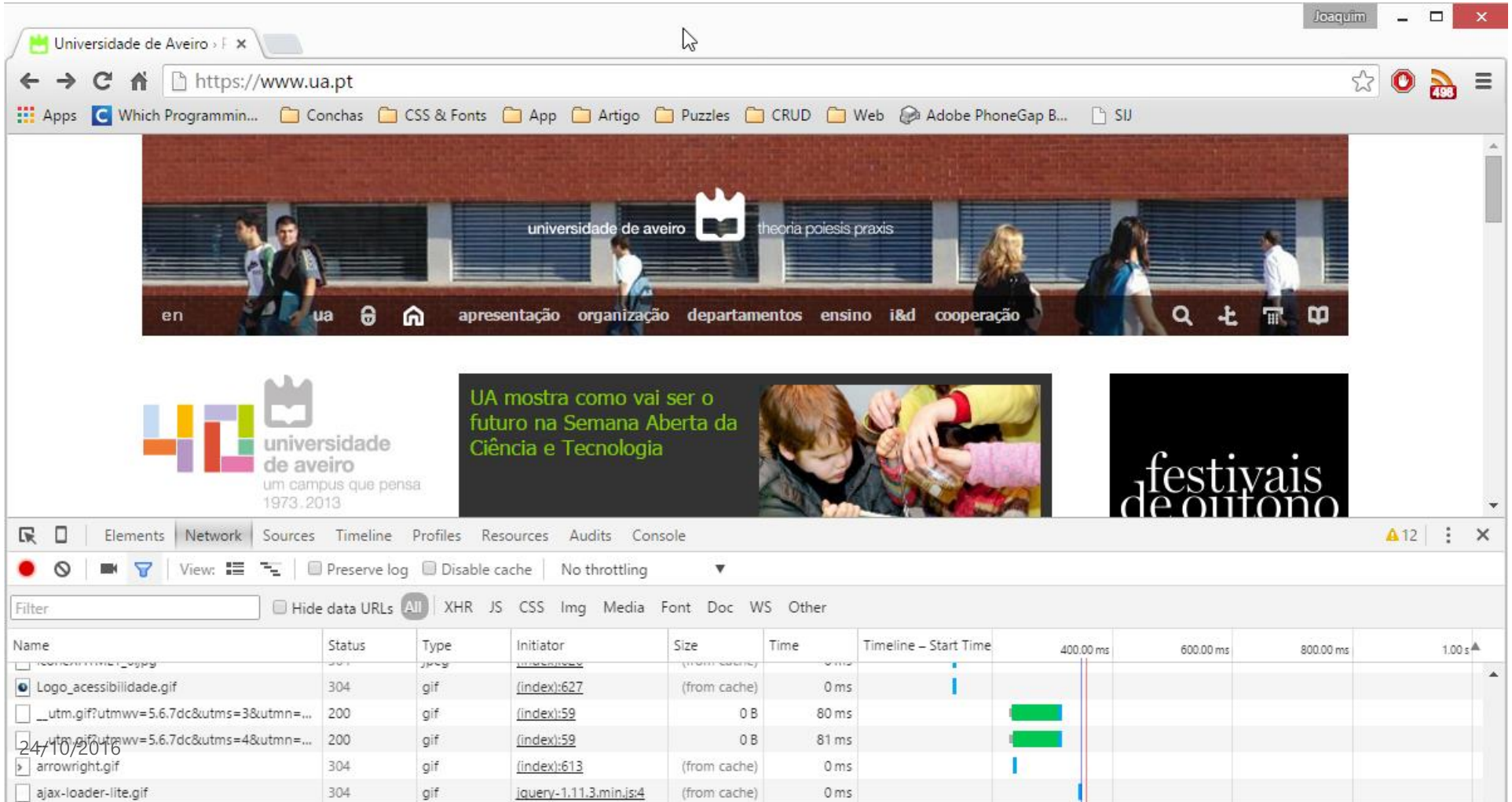
A janela de depuração
Internet Explorer



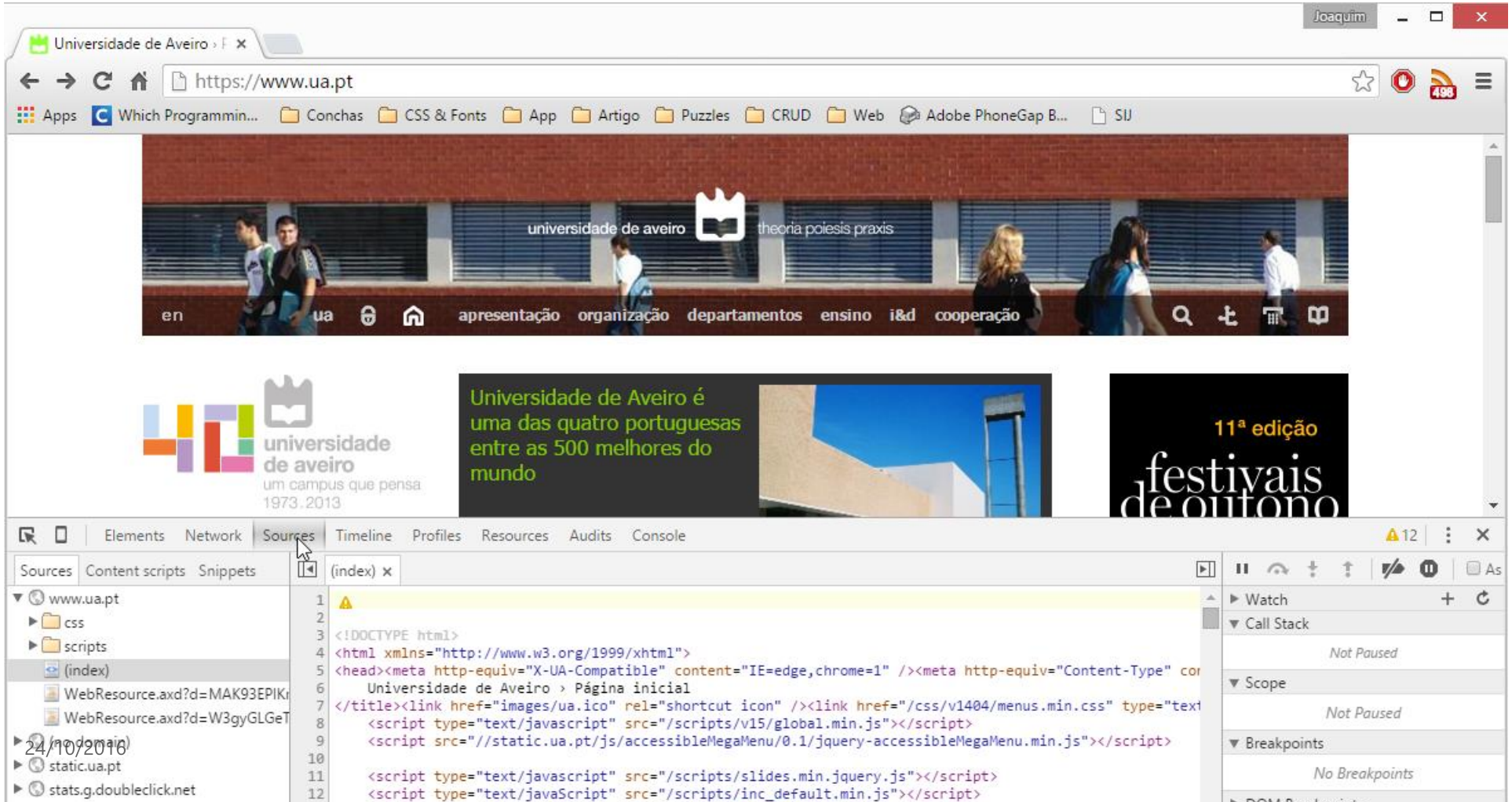
A janela de depuração
Firefox



A janela de depuração
Chrome - network



A janela de depuração
Chrome – source files



A janela de depuração
Chrome – console