Introdução às tecnologias Web - ITW

Aula 5 – Javascript

Sumário

A linguagem Javascript
- Introdução
- Sintaxe JavaScript
- Interacção com o DOM
- Temporizadores
- Eventos

Introdução

A linguagem Javascript (JS) foi originalmente implementada como parte dos *web browsers* para que estes pudessem executar programas / *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 e alterar o conteúdo, de modo dinâmico, do documento exibido.

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

- *A nossa aplicação móvel – objetivo final desta unidade curricular – será uma Single Page Application*

Introdução

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

- O facto de ser interpretada significa que não são necessários os passos de compilação e produção de um objeto executável antes da sua execução, tal como acontece com as linguagens Java ou C.
- JS 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.*

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

Inclusão numa página

O processo de inclusão numa página *Web* é semelhante à da inclusão de estilos CSS, ou seja, através da utilização de marcas específicas, 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 ser incluído diretamente ou ser obtido de uma fonte externa.

Inclusão direta na página
fim do <head></head>

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

 </body>
</html>
```

Cabeçalho do documento

É aqui que deve ser inserido o script JS

Inclusão direta na página
fim do <body></body>

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
</head>
 <body>

  <script>
    /* O script Javascript deve ser colocado aqui */
  </script>
</body>
</html>
```

Cabeçalho do documento

É aqui que deve ser inserido o script JS

Obtenção do script de fonte externa

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
  <script type="texto/javascript" src="path/filename.js">
 </head>
 <body>


 </body>
</html>
```

Cabeçalho do documento

Nome e localização do script JS

Versatilidade vs segurança

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

No entanto, como verificaremos, qualquer código JS criado é sempre enviado ao cliente na sua forma textual, podendo, por isso, ser rapidamente copiado.

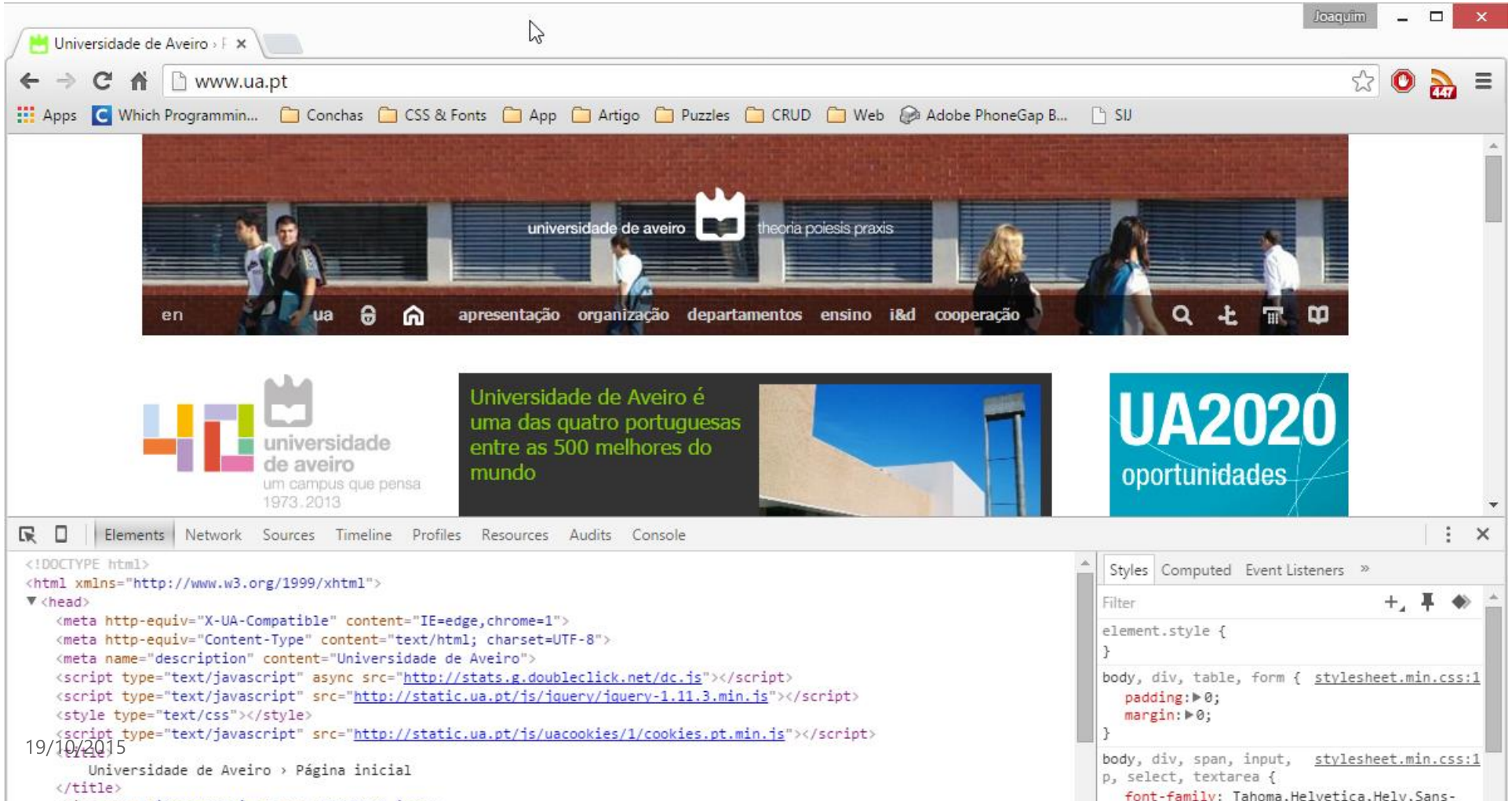
Para dificultar a leitura do código, protegendo a autoria do mesmo, e para poupar no espaço ocupado pelo ficheiro, de modo a não prejudicar o carregamento e posterior apresentação da página, este código é muitas vezes "minimizado" (tradução livre de *minified*).

Exemplos de minimizaçãode ficheiros:

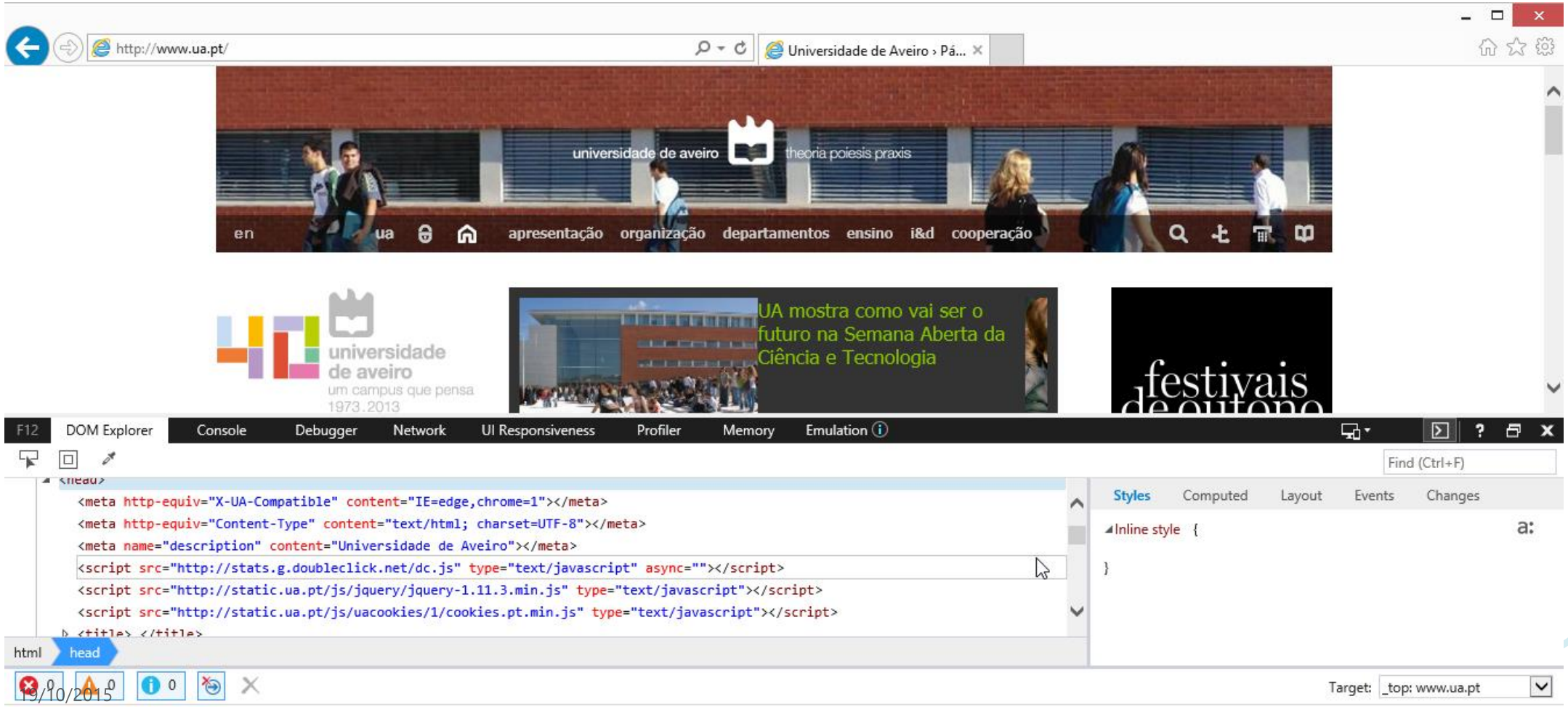
Content/bootstrap.min.css

Scripts/bootstrap.js

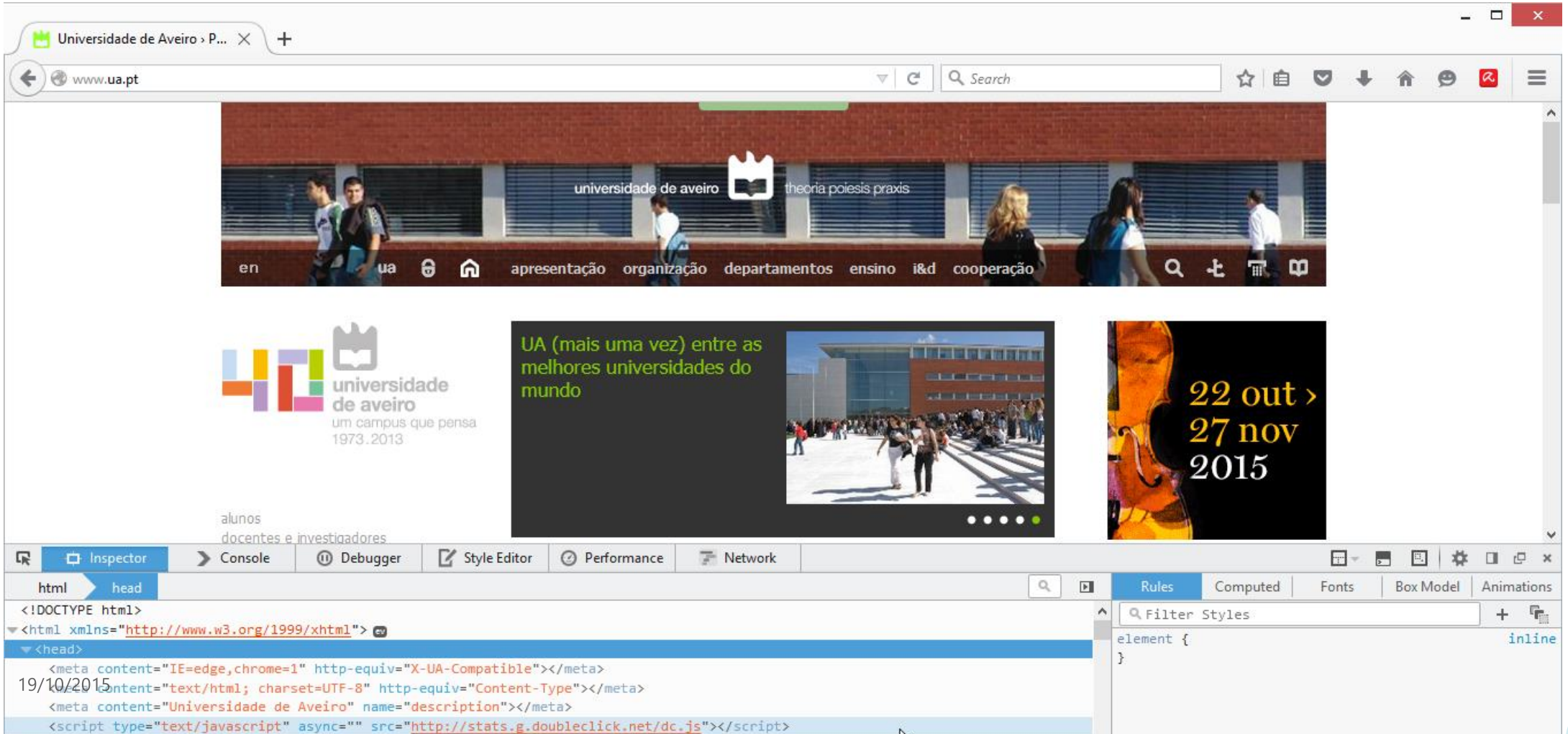
A janela de depuração
Chrome



A janela de depuração
Internet Explorer

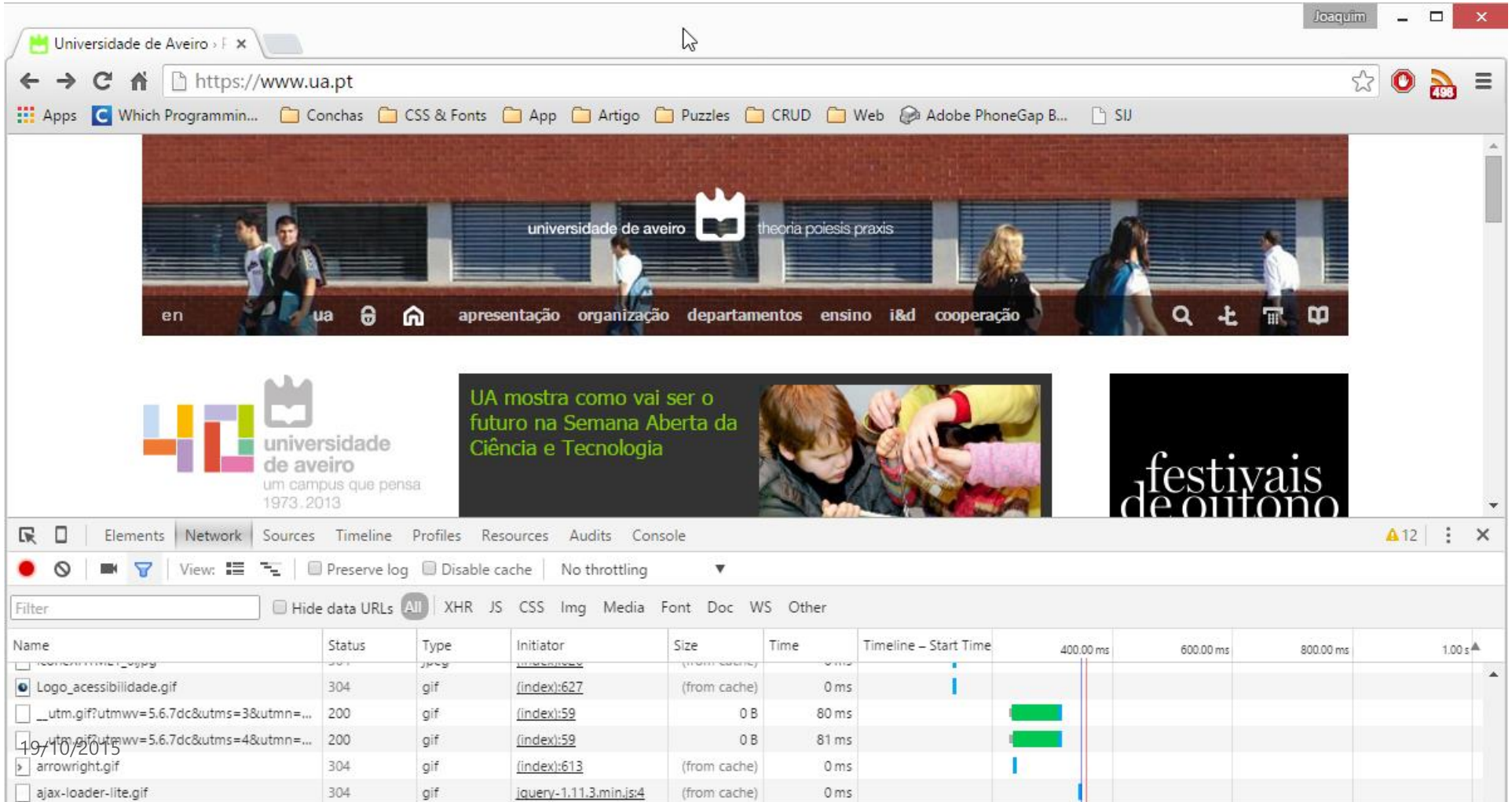


A janela de depuração
Firefox



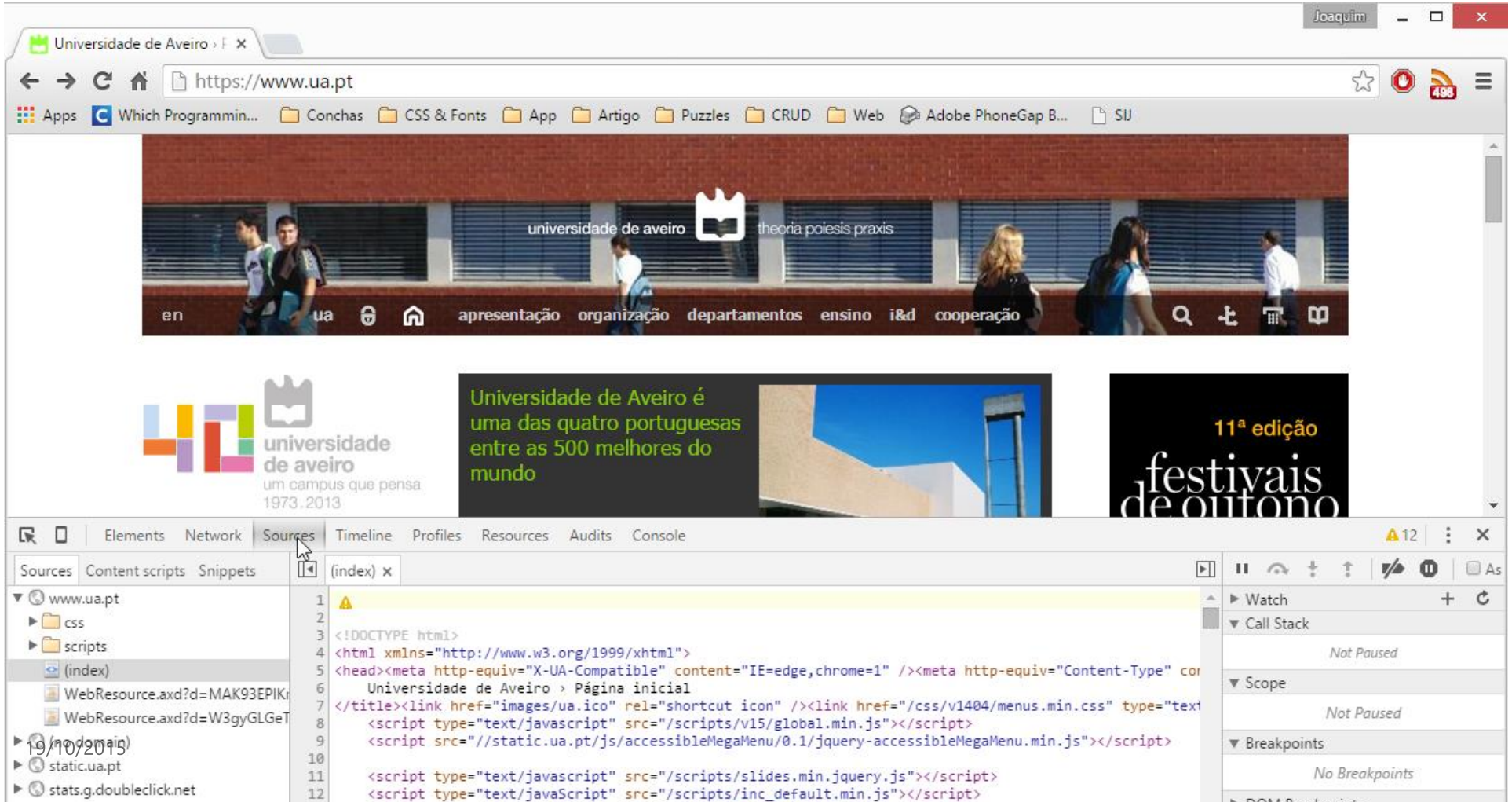
A janela de depuração

Chrome - network

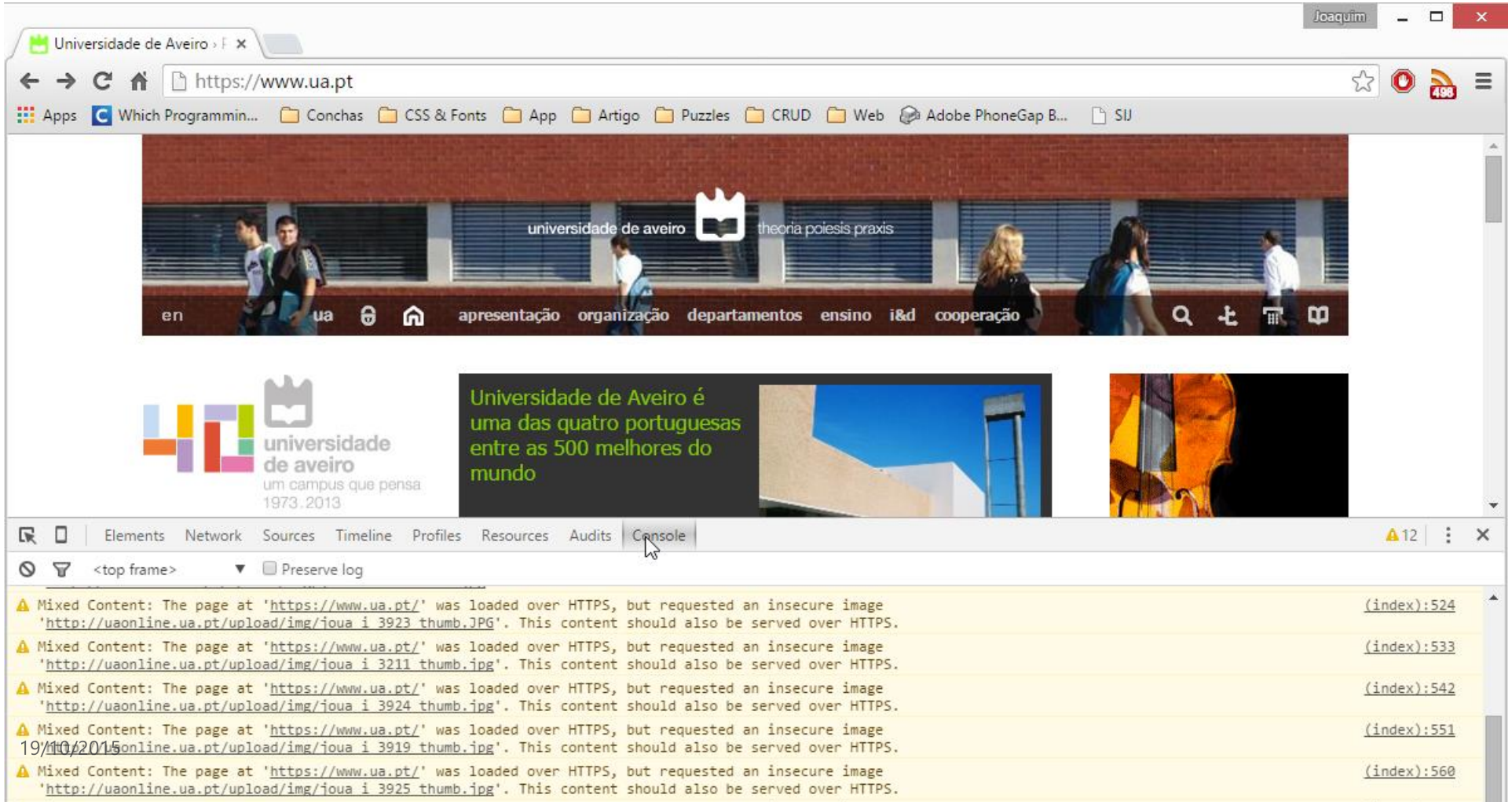


A janela de depuração

Chrome – source files



A janela de depuração
Chrome – console



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 instrução, podendo estas instruções ser terminadas 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

O exemplo seguinte 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 s = "3";
        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 faze-se de modo convencional:

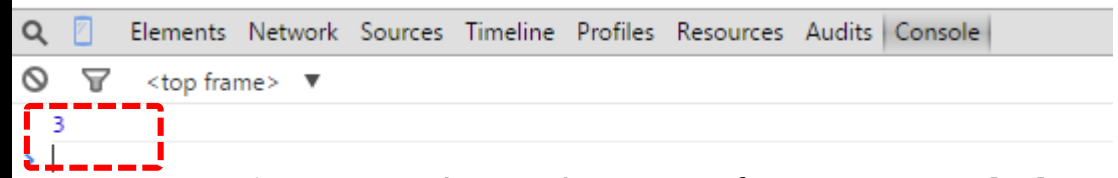
`<nome_da_variável> <operação> <valor>`

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 s ="3";
        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 signicado desta operação irá variar com o tipo de variável (que depende do seu conteúdo atual).

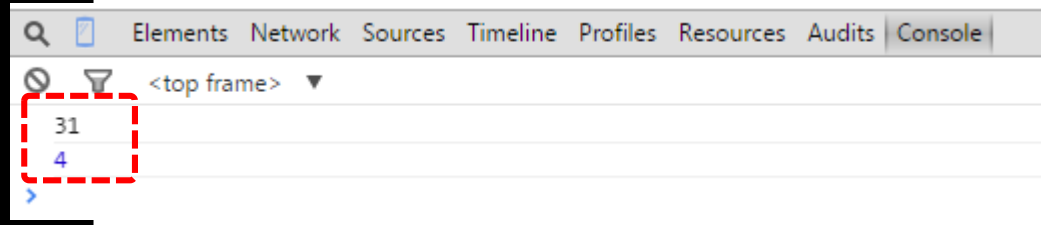
Um bom exemplo é o operador +, que no caso de números irá calcular a soma, mas no caso de sequências de caracteres irá concatená-los.

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 */
        document.write("uma-string" - 2);
    </script>
</head>
<body>

</body>
</html>
```

Neste caso, ao invés do resultado ser apresentado na consola da janela de depuração, o resultado +e "escrito" no documento `document.write(…)`

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 se subtrair um inteiro a uma *String*.

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.

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.

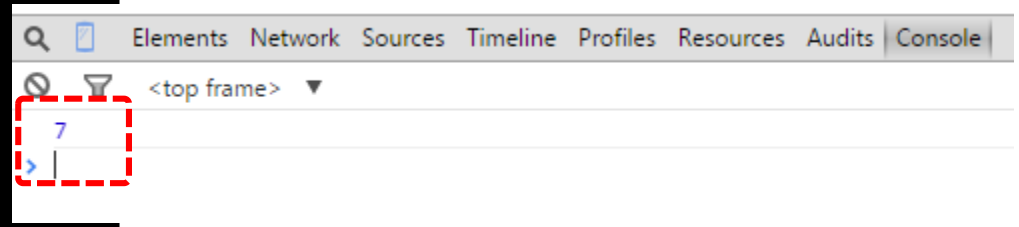
Comparando com a linguagem Java, verifica-se que, no Javascript não é necessário declarar qual o tipo de retorno da função, nem os tipos dos parâmetros.

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: "<", ">", ">=", "==", etc...

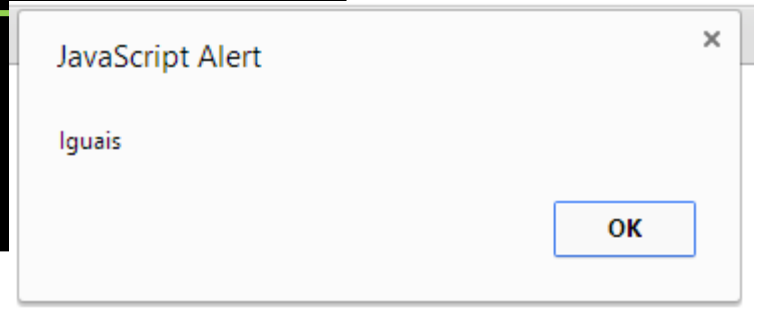
Sintaxe da linguagem Javascript

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.

No entanto, por vezes, pretende-se efetuar uma comparação do valor e do tipo. 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, a não é igual a 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`.

```
var a ="abc";
switch(a){
    case "abc":
        alert("string abc");
        break;
    case 3:
        alert("inteiro 3");
        break;
    default:
        alert("outro");
}
```

- 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{
    /* operação */
} while(condição);
```

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

```
for (inicio; comparação; incremento){
    /* operação */
}
```

Diferenças entre os diversos tipos de ciclos:

- `do-while` – o ciclo é executado pelo menos uma vez porque a instrução 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` – o ciclo é executado 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, é a 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 recarregar a página.

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>... é um `objecto`; um elemento ... também é um `objeto`; ...

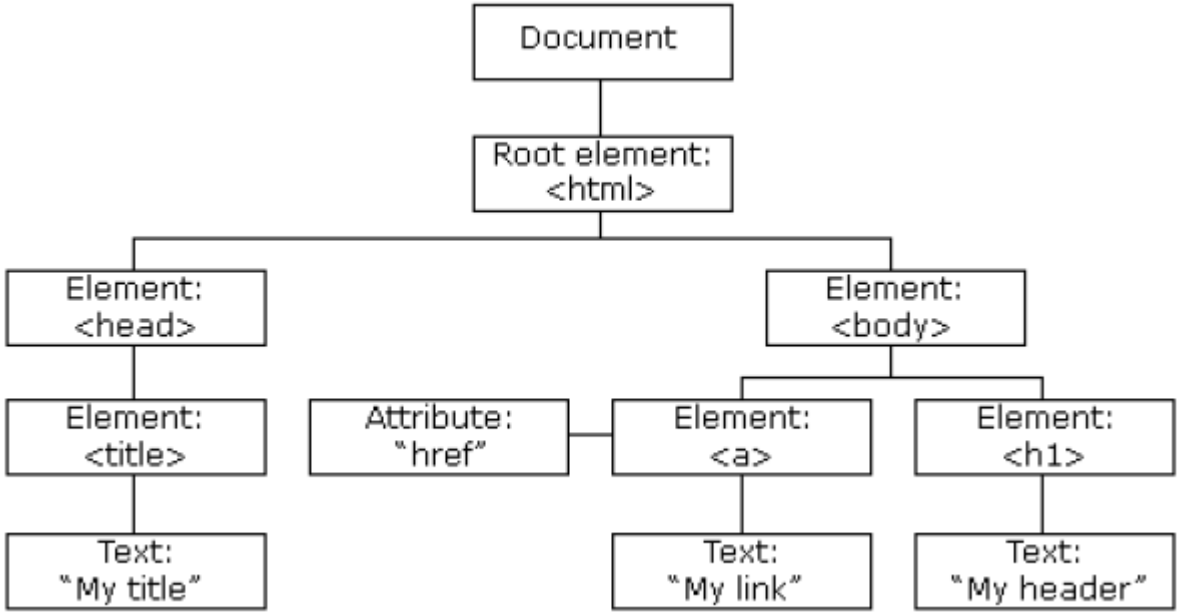
Exemplo:

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

Através de Javascript é possível consultar valor de `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>
<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 necessário que os elementos HTML já existam na DOM quando o código JavaScript 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 cujo ID seja inexistente, o valor devolvido pelo método getElementById será null, o que pode ser verificado usando uma condição (if) :

```
var x = document.getElementById("nao-existe");
if(x == null)
    alert("Elemento não encontrado");
else
    alert(x.value);
```

Sintaxe da linguagem Javascript
Eventos

Até agora o código JS tem sido executado de forma automática aquando do carregamento da página – daí ser colocado no fim do documento...

Só o código que se encontra fora de funções é que é automaticamente executado. Este pode depois invocar as diversas funções disponíveis.

```
<script>
      function soma(x,y){
            return x+y;
      }
      var resultado = soma(3,4);
      console.log(resultado);
</script>
```

Por vezes este não é o comportamento desejado, podendo o programador querer que nenhum código seja executado automaticamente, mas somente após alguma situação especial acontecer (um evento!).

Sintaxe da linguagem Javascript
Eventos – window.onload

O exemplo seguinte melhora o código anterior, através da utilização do evento `window.onload`.

Ficheiro 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;
```

O **evento** window.onload só executado "quando a janela (window) estiver completamente carregada. Nesse momento o DOM está completo e, portanto, é possível executar qualquer operação sem limitações.

Sintaxe da linguagem Javascript
Eventos – window.onload

Ficheiro 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;
```

Repare que para evitar que o código seja executado ao arranque – e só seja executado quando a window estiver completamente carregada – é necessário colocar dentro de uma função o código anteriormente executado. Neste caso a função tem o nome calculadora.

Sintaxe da linguagem Javascript
Eventos

Os eventos também se podem referir a ações do utilizador, nomeadamente:

- mover o apontador,
- pressionar teclas, ou
- simplesmente a modificação de algum elemento HTML.

No caso de uma calculadora pode-se considerar que será útil a existência de um botão que calcule o valor, e outro que indique a operação a efetuar.

Isto realiza-se através da inclusão de propriedades diretamente nas marcas HTML!

Para uma lista completa dos eventos possíveis, consulte http://www.w3schools.com/tags/ref_eventattributes.asp

Sintaxe da linguagem Javascript
Eventos - onclick

Considere o seguinte excerto de HTML:

```
<html>
<head>
    <script type="text/javascript" src="calculadora.js"></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="calcular()">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.

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:

```
<html>
<head>
    <script type="text/javascript" src="calculadora.js"></script>
</head>
<body>
    <input id="op1" value="2" />
    <span id="op-view" >+</span>
    <input id="op2" value="3" />
    <input id="res" value="" /><br/>
    <select onchange="operacao()">
        <option value="+"> Soma </option>
        <option value="-"> Subtração </option>
    </select>
    <button onclick="calcular()" >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 **operacao**() será chamada.

Sintaxe da linguagem Javascript
Eventos – event.target

Neste caso, o marcador `<select>` invocará a função `operacao()`. A função simplesmente irá preencher a variável `op` com a operação a realizar.

```
var op = "+"; //Deverá estar no topo do ficheiro.
function operacao(){
    var elemento = event.target;
    var elementoSeleccionado = elemento.options[elemento.selectedIndex];
    op = elementoSeleccionado.value;
    console.log("Operação: "+op);
}
```

A função **`event.target`** permite indicar à função qual o elemento que invocou o evento - ao invés de utilizar o nome do elemento.
Neste caso, permite aceder imediatamente ao elemento onde se clicou (o `<select>`), para saber qual a operação a realizar, evitando usar o **`document.getElementById()`**.
No entanto, este elemento não está disponível em todos os navegadores!

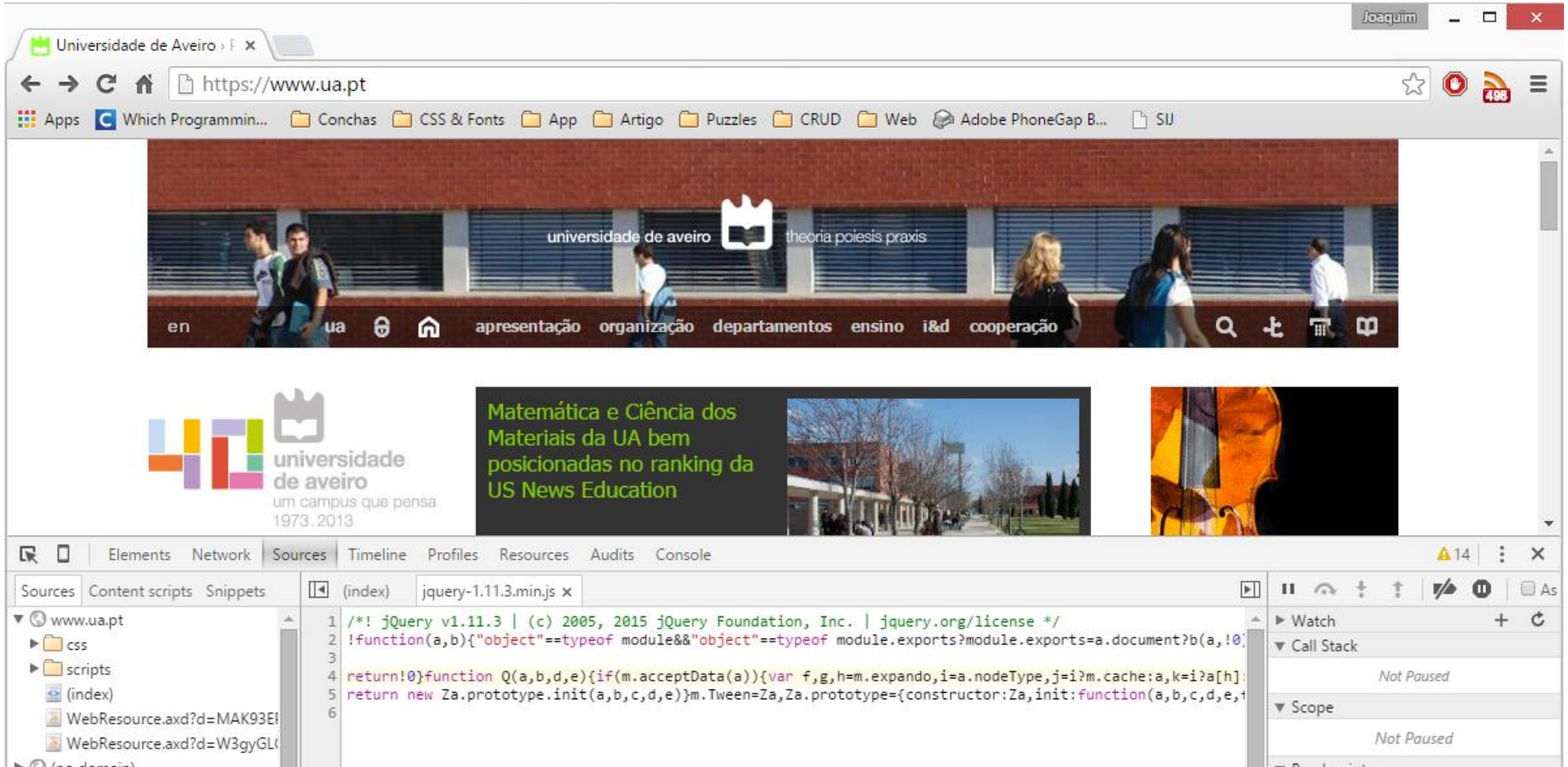
Prática

Exercício 1

1. Abra o web browser e escolha uma URL, como por exemplo, http://www.ua.pt
2. De seguida carregue na tecla F12 (que abrirá a janela de depuração/*debug*)
3. Selecione um dos scripts (nome_do_ficheiro.js) e veja o resultado

Exercício 1

4- Comprove que o script vem em linguagem textual (embora nem sempre muito legível)



Exercício 2

1. Construa uma página web , utilizando o exemplo apresentado no slide 17.

   Para voltar a executar o código JS basta atualizar a página do navegador, o que tipicamente se consegue através da tecla F5, ou da combinação CMD + R no caso do sistema OS X.

2. Coloque o código / script num ficheiro separado (myScript.js, por exemplo) e verique que o navegador obtém o ficheiro .

3. Experimente substituir a chamada *console.log* por *document.write* e *alert*, de forma a verificar formas simples como o JS pode apresentar mensagens aos utilizadores.

Exercício 3

1. Seguindo o exemplo do slide 20, replique-o no seu computador.
2. Aceda à consola do navegador e verique o valor impresso.
3. Experimente com outros valores, números reais e sequências de carateres.

Exercício 4

Repita o exercicio do Slide 25 e compare o resultado de uma condição utilizando os operadores == e ===.

Exercício 5

Verifique como pode utilizar a operação switch misturando tipos de variáveis diferentes. Tome como exemplo o apesentado no slide 26

Exercício 6

Implemente o exemplo dos slides 32 e 33, completando-o de forma a escrever no elemento <input id="res"...> o resultado da adição dos 2 valores.

Exercício 7

Complete o exercício do slide 39 implementando a função **`calcular`**`()` de forma a aplicar uma operação diferente, de acordo com o valor da variável op.

Adicione suporte para mais operações, tais como a multiplicação, divisão, resto da divisão inteira, fatorial, … tendo os devidos cuidados, como por exemplo divisão por zero ou factorial de números negativos.