

**Bibliografia
auxiliar para o
segundo módulo:**

Goodman, Danny
JavaScript, a Bíblia,
Rio de Janeiro:
Editora Campus,
2001.

Moncur, Michael,
*Aprenda em 24
horas JavaScript 1.3*,
Rio de Janeiro:
Editora Campus,
1999.

Flanagan, David,
*JavaScript the
definitive guide*,
Cambridge:
O'Reilly, 1988.

**Negrino, T. e
Smith, D.,** *Java-
Script para a WWW*,
Rio de Janeiro:
Editora
Campus, 2000.

Osmar Sila, *Java
Script, Guia Prático*,
Editora Erica, 2000.

Documentação da
Netscape:
<http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm> .

Documentação da
Microsoft:
<http://msdn.microsoft.com/scripting/JScript/doc/Jstoc.htm>.

Aula 9: Elementos da Linguagem JavaScript

Bem-vindo ao segundo módulo do nosso curso! Nele você aprenderá como incluir programas, ou *scripts*, em suas páginas Web. Este módulo será todo dedicado à linguagem JavaScript. E nessa aula você aprenderá o que é JavaScript, o que pode fazer com ela, e como inclui-la nas suas páginas HTML.

Objetivos:

Nesta aula você aprenderá os fundamentos de JavaScript e verá:

- os tipos e as versões do JavaScript;
- como incluir código na página HTML (tag `script`);
- a estrutura da linguagem;
- suas constantes e variáveis;
- a concatenação de strings.

Pré-requisitos:

As aulas 1 a 3 e a aula 7 do módulo anterior.

1. Introdução à JavaScript

JavaScript, que aprenderemos neste módulo, como HTML, é uma linguagem interpretada, mas, diferentemente de HTML, é uma linguagem de **programação**.

JavaScript inclusive tem características rudimentares de **Orientação a Objetos (O.O.)** e **programação dirigida por eventos** (no decorrer do módulo, você entenderá melhor o que estas características acrescentam à linguagem).

Esta linguagem foi desenvolvida (por **Bredan Eich da Netscape**) num **convênio entre a Netscape e a Sun**. Inicialmente, chamava-se **LiveScript**, mas mudou de nome às vésperas do lançamento por questões de marketing.

O código fonte fica **embutido** na página HTML e é **interpretado** pelo navegador.

Apesar da expectativa criada pelo nome, esta linguagem tem muito pouco a ver com Java. A principal relação entre as duas linguagens vem de sua origem comum: as linguagens C e C++. Além de alguma semelhança sintática e do fato de que, tanto Java quanto JavaScript, podem fornecer um conteúdo executável pelos navegadores Web.

Na linguagem Java, o código fonte é compilado para um código intermediário, que pode ser executado em qualquer máquina que possua um interpretador Java (não necessariamente o interpretador embutido no navegador). Além desta diferença, há muitas outras diferenças em termos de sintaxe, abrangências e objetivos das duas linguagens.

Com JavaScript é possível manipular imagens embutidas na página; ler e escrever o estado do cliente em **Cookies**; interagir com o usuário; alterar características do documento; e controlar o comportamento do navegador fazendo este:

- carregar um novo documento;
- retornar para a página anteriormente visitada;
- abrir/fechar janelas;
- exibir páginas diferentes de acordo com o browser do usuário;
- controlar o conteúdo de formulários HTML.

JavaScript, no entanto, não possibilita **escrever ou ler arquivos**; não possui capacidade de desenho, não suporta estabelecer conexões em rede, nem oferece possibilidades **de processamento em segundo plano ou multitarefa** (*multithreading*).

2. Tipos e Versões de JavaScript

Existem variantes da linguagem dependendo do contexto onde se embute o JavaScript. Algumas destas são as chamadas *Client-Side JavaScript* ou "no lado do cliente" e *Server-Side JavaScript* ou "no lado do servidor".

“**Cookie**” é o termo usado (inicialmente pelo Netscape) para descrever uma pequena **quantidade de dados armazenados** pelo cliente de forma **temporária ou permanente**.

Cookies são transmitidos **de e para** o servidor e permitem que a **página ou o site “lembre”** de coisas sobre o cliente, como por exemplo: que o usuário já visitou antes o site, **já se registrou** ou expressou anteriormente alguma preferência sobre o *layout* da página.

JavaScript permite ler e escrever valores nos **cookies**, podendo gerar HTML baseado nos valores descritos neles.

Em nosso curso abordaremos apenas a forma executada no cliente (*client-side*). Esta forma, é a variante mais comum; quando a maioria das pessoas se referem à JavaScript está se referindo a esta forma, nela o interpretador se encontra embutido no navegador Web. A forma no lado do servidor (*server-side*) é utilizada nos servidores Netscape como uma alternativa aos scripts CGI. Nesta forma, diferente da *client-side*, é possível ler e escrever arquivos e bancos de dados no servidor.

Como em HTML, há diferenças na linguagem JavaScript em função do tipo de navegador onde o programa será interpretado. O nome JavaScript é propriedade da Netscape, a implementação da Microsoft da linguagem é oficialmente denominada JScript. Versões de JScript são mais ou menos compatíveis com as versões equivalentes do JavaScript. A tabela 9.1 mostra essas versões.

No final do capítulo, em um apêndice, você encontra um histórico das versões destas duas formas da Linguagem. A tabela 9.4 considera o Jscript (ou seja o JavaScript da Microsoft) com suas versões, datas e comparação com o JavaScript. A tabela 9.5 as versões do JavaScript suas datas de lançamento e as versões dos navegadores que as suportam.

Tabela 9.1 - Versões de JavaScript e navegadores

Versões no Netscape		Versões no Explorer	
Browser	Javascript	Browser	JScript
Netscape 2.0	Javascript 1.0		
Netscape 3.0	Javascript 1.1	Explorer 3.0	JScript 1.0
Netscape 4.0	Javascript 1.2	Explorer 4.0	JScript 3.0
Netscape 4.5	Javascript 1.3	Explorer 5.0	JScript 5.0
Netscape 6.0	Javascript 1.5	Explorer 6.0	JScript 5.6
Firefox 3.0	Javascript 1.6	Explorer 7.0	JScript 5.7

3. Execução de Programas JavaScript

Os programas JavaScript são executados na ordem em que aparecem na página e não é possível fazer referência a elementos HTML que ainda não foram definidos.

Há três formas de incluir um código JavaScript em uma página:

- através da tag `<SCRIPT> . . . </SCRIPT>`.
- através da seleção de um [link](#).
- através da associação a um [evento](#).

Na primeira forma, a tag `<script>` pode aparecer em qualquer lugar da página, inclusive na área de cabeçalho (dentro da tag `<HEAD> ... </HEAD>`). Quando incluída na área de cabeçalho, o programa é executado antes que a página seja totalmente carregada. As linhas abaixo exemplificam esta forma:

```
<script language="Javascript">
<!--
...//codigo Javascript ...
-->
</script>
```

Quando esta forma de JavaScript for usada é possível utilizar a tag `<NOSCRIPT> ... </NOSCRIPT>` para exibir uma alternativa ao código JavaScript em navegadores que não saibam executá-lo.

A estrutura léxica de uma linguagem de programação é o conjunto de regras elementares que especificam como você escreve programas com ela.

Definindo coisas como a aparência dos nomes, as formas usadas para comentários, e como os comandos são separados uns dos outros.

A inclusão do código entre comentários (tag `<!-- ... -->`) é recomendável para que os navegadores que não conheçam JavaScript não tentem exibir o código.

É possível também não escrever o código diretamente na página, colocando-o em um arquivo separado. Neste caso se usa o atributo SRC para indicar o nome do arquivo de onde virá o código. Por exemplo:

```
<script language="Javascript" src="arquivo.js">
</script>
```

Na segunda forma, o link ao ser selecionado promove a execução do programa:

```
<a href="javascript: ..." > ... </a>
```

Quando associada a um **evento** a sintaxe usada é:

```
<tag-html onEvento=" ... ">
```

No exemplo atividade do final da aula, você terá a oportunidade de ver com mais detalhes estas formas.

4. Estrutura Léxica da Linguagem

JavaScript é uma linguagem *case sensitivity* (como C e diferente de HTML): as letras minúsculas (*lowercase*) são diferenciadas das maiúsculas (*uppercase*). Isso significa que as palavras-chave da linguagem, as variáveis, os nomes de funções ou qualquer outro identificador deve sempre ser escrito da mesma forma. A palavra-chave `break`, por exemplo, se for escrita `Break` ou `BREAK` representará outra coisa e não mais a palavra-chave.

Algumas exceções a esta regra são, entretanto, encontradas na forma executada no cliente (*client-side*). No Explorer 3, por exemplo, todos os objetos e propriedades executados no cliente são *case-insensitive*. Outro exemplo são as folhas de estilo JavaScript executadas no cliente no Navigator 4, que permitem a especificação de estilos por tag HTML (projetadas com o propósito de serem *case-insensitive*).

Os **espaços em branco**, as **tabulações** (*tabs*) e as **mudanças de linha** (*newlines*) entre os comandos são ignorados. Você pode assim usar os espaços e as mudanças de linha livremente ou de uma maneira consistente em seu programa de modo a torná-lo fácil de entender.

O caracter **ponto e vírgula** (;) deve ser usado como limitadores, após cada comando. Esta é a mesma forma de limitar comandos de C, C++ e Java. Mas em JavaScript você pode omiti-los se os comandos se encontrarem em linhas diferentes. Por exemplo, nas linhas abaixo você, poderia tê-los omitido.

```
a=3;  
b=5;
```

Mas o primeiro ";" não poderia ser omitido se você escrevesse ambas as atribuições na mesma linhas, como:

```
a=3;b=5;
```

Não é um bom hábito deixar por conta da linguagem, onde os comandos devem ser separados. O melhor é você se acostumar a sempre usá-los. Saber que a linguagem JavaScript permite que dois comandos sejam separados por linhas, automaticamente inserindo ponto e vírgula por você, evita alguns mal-entendidos.

A linguagem JavaScript possui duas formas de incluir **comentários** em meio às instruções (herdadas das linguagens C e C++):

- //
- o conjunto: /* ... */

As barras duplas, //, em qualquer ponto da linha fazem com que o interpretador ignore todos os demais caracteres até o **fim da linha**.

A seqüência /* (barra seguida do asterisco) faz com que o interpretador ignore **todos os caracteres** (inclusive o de mudança de linha) até encontrar a seqüência */ (asterisco seguido da barra). Este tipo, no entanto, não pode ser “aninhado”, isto é, não se pode ter outro comentário do mesmo estilo no interior. As linhas que seguem mostram isso.

```
// essa linha e comentario
/* esta tambem
   /* e esta
   por enquanto tambem */
   mas o final da linha anterior
   confundiu tudo e as tres ultimas
   nao sao mais comentarios */
```

Além do estilo C e C++, a forma *client-side* reconhece a seqüência de abertura de comentários de HTML: <!--. Mas esta forma é tratada como se fosse um comentário de linha única, // pois JavaScript não reconhece a seqüência de fechamento HTML: -->. Há uma razão para isso: em programas JavaScript, se a primeira linha começar com <!-- e a última terminar com //-->, todo o programa é contido dentro de um comentário HTML e é ignorado pelo browser que não suportar JavaScript. Desde que a primeira linha comece com <!-- e a última linha com //, JavaScript ignora ambas, mas não as linhas entre elas. Desta maneira é possível “esconder” o código de um navegador que não pode entendê-lo e “não esconder” se ele puder ser entendido pelo browser. Devido a este propósito especial do comentário <!--, você deve usá-lo apenas na primeira linha de seus *scripts*, já que outra forma poderia causar confusão.

5. Tipos de Dados

Os tipos de dados utilizáveis por uma linguagem é um dos seus aspectos mais fundamentais. JavaScript tem 3 tipos primitivos de dados: números, strings e booleanos.

Um **literal** é um dado que aparece diretamente no programa.

A tabela a seguir mostra formas de representar literais na linguagem JavaScript:

Tabela 9.2 - Exemplos de literais

Tipo	Constante
string	"Lanchonete Bob's", "Pi = 3.14", "A"
	'vi o filme "Central do Brasil"', 'Pi=3.14', 'A'
inteiro na base 10	8, -12, -9007199254740992, 9007199254740992
inteiro octal	056, -0123, 0377,
inteiro hexadecimal	0x7A, -0x10, 0xCAFE911, 0xff, 0377
real	3.14, -10.28, .3333333, 6.02e+23, 1.4738E-32
booleano	true , false

Uma **variável** é um nome associado a um dado ou valor, diz-se que uma variável "armazena" ou "contém" um valor. Elas permitem que se armazene e manipule dados no programa, assumindo ou recebendo valores diferentes.

Por exemplo, na linha abaixo se atribui à variável denominada `inicio` o valor 2, e depois adiciona-se a `inicio` o valor 3, e atribui-se o resultado a uma nova variável. `soma`:

```
inicio = 2;
soma = inicio + 3;
```

Os nomes de variáveis podem conter letras, dígitos (desde que não seja o primeiro caracter do nome), o caracter `'_'` e o caracter `'$'`. Algumas palavras não podem ser usadas como nome de variável, pois são **reservadas** para comandos e expressões da linguagem, como por exemplo: `true`, `var`, `if`, `while` etc.

Diferente de uma linguagem como Pascal, não é preciso declarar previamente uma variável nem definir explicitamente o seu tipo. Para declarar variável, usa-se a palavra-chave: `var`. Assim, são exemplos de declarações e usos de variáveis:

```
var i;          // Cria a variavel i
i = 10;        // Guarda o valor 10 em i
var i = 2;     // Cria e armazena 2
i = 2;        /*idem a linha anterior: a palavra "var" e
                opcional, a principio */
i = "onze";   /* agora o tipo da variavel passou a ser
                string. */
```

Se uma variável é criada no momento em que é utilizada pela primeira vez, o seu tipo é definido pelo tipo do valor que ela recebe. Se uma variável é utilizada sem que antes tenha um valor atribuído, o seu valor é **undefined**.

Embora não seja muito recomendável, é possível modificar o tipo de uma variável que já tenha sido criada simplesmente atribuindo um valor de outro tipo. Isso ocorre porque JavaScript, diferente de C, Java, Pascal e de muitas outras é uma linguagem **não-tipada** (*untyped*). Ela automaticamente converte valores de um tipo para outro se necessário. Nós veremos mais sobre isso quando falarmos de **Conversão de tipos de dados**, na próxima aula. A mudança do tipo de uma variável é, entretanto, uma fonte potencial de erros de execução e deve ser no mínimo usada com consciência.

5.1 - Números

Número é o tipo mais básico possível e não requer explicação, a não ser pela observação de que (diferente de C e Java) JavaScript não faz distinção entre valores inteiros e reais (pontos flutuantes). Todos os números em JavaScript são representados como valores de ponto flutuante em 8 bytes.

Quando um número aparece diretamente em um programa JavaScript, nós o chamamos **literal numérico**. Os formatos de literais numéricos suportados pela linguagem são:

- Inteiros na base 10.
- Inteiros octais.
- Inteiros hexadecimais.
- Pontos flutuantes.

Um número de ponto flutuante de dupla precisão (64-bit) ocupa até 53 bits para os dígitos significativos com uma parte exponencial de 11 bits (com 1 bit para o sinal).

Inteiros na base 10 são representados como seqüência de dígitos que não começam com o dígito zero, exceção feita ao próprio zero.

Como todos os números são representados como valores de ponto flutuante é possível especificar literais inteiros extremamente grandes. Todos os inteiros entre -2^{53} (= -9007199254740992) e 2^{53} (= 9007199254740992), inclusive estes, podem ser representados exatamente.

Um valor literal **octal** (número na base 8) começa com o dígito zero, seguido por uma seqüência de números entre 0 e 7.

Como em C e C++ , um literal hexadecimal começa com 0x ou)X, seguido de dígitos de 0 a 9 ou letras a (ou A) a f (ou F), as quais representam os números 10 a 15.

Os literais de **ponto flutuante** podem ter um ponto decimal, uma parte inteira do número e uma parte fracionária (depois do ponto decimal). Também podem ser representados na notação exponencial: onde a letra e (ou E) seguida de um sinal + ou - introduz expoentes inteiros que representam potências de 10. De modo que $100.10=1.001e+2$, ou $0.01=1E-2$.

A linguagem tem alguns **valores numéricos especiais**. Quando um ponto flutuante se torna maior que o valor máximo representável, o resultado é um valor especial que JavaScript imprime como **Infinity**. De maneira semelhante quando um valor negativo se torna mais negativo que o maior negativo possível de ser representado, o resultado impresso é **-Infinity**.

Outro valor numérico especial aparece quando uma operação matemática resulta um valor indefinido (por exemplo em uma divisão de zero por zero). Neste caso, o resultado é um valor especial denominado "not-a-number" que é impresso como **NaN**. Este valor tem uma propriedade especial: em uma operação de comparação (veremos estas operações na próxima aula) não é igual a nenhum número nem a ele próprio. Por este motivo existe uma função especial **isNaN()** para testar este valor.

Existem constantes definidas para cada um dos valores numéricos especiais. Estas são listadas na tabela 9.3. O exemplo atividade no final da aula, mostra como elas são vistas na página gerada pelo script.

Tabela 9.3 - Constantes numéricas especiais

Constante	Significado
Number.MAX_VALUE	o maior valor possível
Number.MIN_VALUE	o menor (mais próximo a zero) possível
Number.POSITIVE_INFINITY	representação especial de infinito
Number.NEGATIVE_INFINITY	representação especial de infinito negativo
Number.NaN	o valor especial not-a-number

5.2. Strings

Strings é uma seqüência de letras, dígitos, caracteres de pontuação e outros, que são representados pela linguagem como texto. Strings literais podem ser incluídas nos programas delimitando algo por pares de '...' ou "...". JavaScript não tem um tipo de dado especial para um caracter isolado, como o char de C ou Java. Para representar

um único caracter, você simplesmente o representa como uma string de tamanho 1.

Você percebeu por que é chamado de *escape sequence* (o que seria ao pé da letra: seqüência de escape)? Porque é uma forma de escapar da interpretação usual do caracter!

Assim é possível criar strings entre **aspas** ou entre **plicas**. A utilização de um ou de outro caracter como delimitador da *string* é uma questão de conveniência do programador. Por exemplo, caso se deseje incluir uma plica em uma *string*, deve-se utilizar aspas como delimitador. Ou seja, uma " pode estar contida dentro de strings delimitada por ' , e ' pode aparecer dentro de strings delimitada por " .

O caracter \ (barra invertida) tem um significado especial, combinado com o caracter que o segue, ele pode representar um caracter que de outra maneira não poderia ser representado dentro da string. Como em C e C++, a barra invertida e o caracter seguinte são chamados "**escape sequence**".

Estas seqüências têm diversos usos. Permitem a representação de caracteres que não têm um símbolo gráfico associado, como tabulações (\t) e de mudança de linha (\n).

Diversas seqüências podem representar símbolos especiais como \251 que permite imprimir o símbolo de *copyright*.

É possível utilizar este caractere também para indicar que uma aspa (ou plica) não está sendo utilizada como fim de *string* (por exemplo, "assim pode-se inserir um caracter \" numa *string* delimitada por aspas").

Finalmente, se \ precede um caracter que não tem significado especial, como \a , ela é simplesmente ignorada, e tem o mesmo sentido de apenas a.

Como JavaScript é usada para produzir páginas Web, você provavelmente usará muitas strings literais em seus scripts.

Uma das propriedades embutidas na linguagem é a possibilidade de **concatenar** strings.

Se você usar o símbolo + (mais) com números, terá como resultado uma operação de adição. Mas se você usar este símbolo com strings, você as juntará simplesmente acrescentando a segunda à primeira, o que é chamado de **concatenação**.

Por exemplo:

```
"2 dividido por " + "0" + " = " +  
"matematicamente indefinido !"
```

junta todo o texto em uma string única:

```
"2 dividido por 0 = matematicamente indefinido  
!"
```

5.3. Booleanos

O tipo booleano de dados pode ter apenas 2 valores: **true** (verdadeiro) ou **false** (falso). Valores booleanos são geralmente o resultado de comparações feitas no programa.

Por exemplo, o código abaixo testa se a variável *a* é 4:

```
a==4
```

Se *a* for 4, o resultado desta comparação será **true**, se não for o resultado é **false**.

Valores booleanos são tipicamente usados nas **estruturas de controle** (como veremos nas próximas aulas) do fluxo dos programas.

A melhor maneira de aprender é fazendo! Por isso estamos propondo um exemplo atividade que além de resumir o que vimos nesta aula, servirá para você exercitar o conteúdo.

Exemplo Atividade:

Neste exemplo, as linhas de código mostram um programa JavaScript, ou *script*, embutido em uma página HTML.

```
<HTML>  
<HEAD>  
<TITLE>Curso de Constru&ccedil;&atilde;o de  
P&aacute;ginas WEB- Mod. 2  
</TITLE>  
</HEAD>  
<BODY>  
<H1 align=center>  
Primeiro Exemplo de JavaScript  
</H1>  
<H2>
```

```
Usando os tipos de dados para calcular potências
de 2.
</H2>
<script language="Javascript">
for (i=1,pot=2;i<=10;i++,pot*=2)
{
document.write("2 elevado a "+ i + " = "+ pot);
document.write("<br>");
}
document.write("<br><h2>Tipos especiais:</h2>");
document.write(" 2 dividido por 0 = " + 2/0 );
//Infinity
document.write("<br>");
document.write("-2 dividido por "+ 0 +" = "+ -2/0);
//-Infinity
document.write("<br>");
document.write("0 dividido por "+ 0 + " = "+ 0/0 );
//NaN
</script>
</BODY>
</HTML>
```

Bata estas linhas e carregue-as em uma página Web. Quando você carregar esta página em um navegador que interpreta JavaScript verá a tela mostrada a seguir, na figura 9.1.

Neste exemplo, você usou a tag <script> para embutir o código JavaScript dentro da página.

Você certamente ficou sem entender alguns detalhes. Sublinhe no código acima as linhas em que não entendeu alguma coisa.

Você merece parabéns se a primeira linha que você não entendeu foi:
document.write("2 elevado a "+ i + " = "+ pot);

Ela exemplifica o método `document.write();`

Este método é usado para escrever textos, dinamicamente, na página HTML, e nós o usaremos muito ao longo do curso. Falaremos sobre ele detalhadamente nas próximas aulas. Mas agora, olhando para o resultado obtido na página, e, em todas as vezes que este método foi usado, tente entender todas as outras linhas em que ele aparece.

Antes de passar aos exercícios, tente também adivinhar o que o comando

`for (...; ...; ...)` deve estar fazendo no *script*.

Exercícios:

1. Utilize, no exemplo atividade, uma variável em substituição ao texto:

"2 elevado a "

de modo que a página gerada não se altere. Além disso, faça a tabela iniciar com a potência de 1 e terminar com potência de 20, e esconda o texto que segue a tabela. Visualize o novo aspecto que o exemplo terá agora!

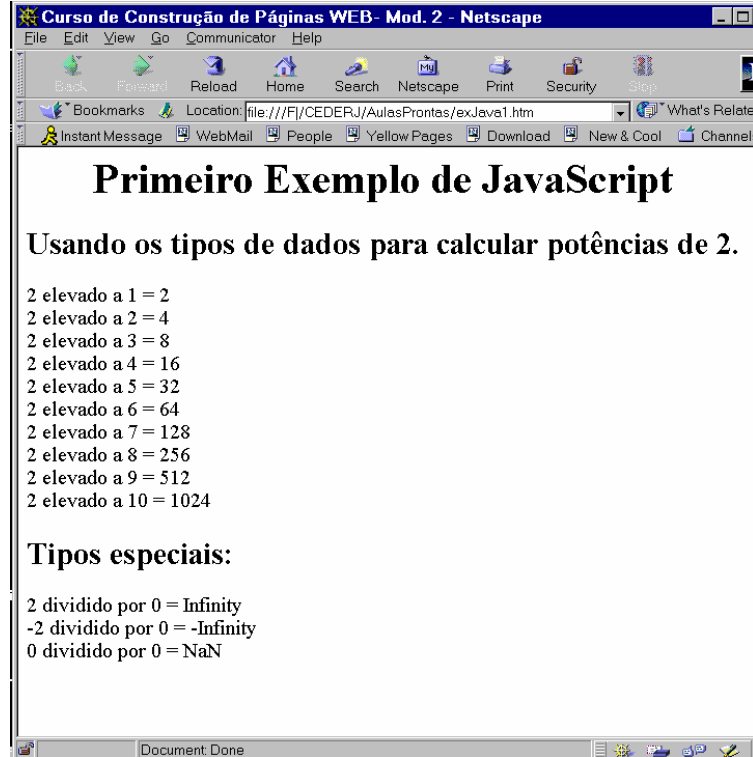


Figura 9.1 - Página gerada pelo código do exemplo atividade

2. Mude o programa de modo a atribuir títulos às linhas da tabela, utilizando `<H3>` `</H3>`. Compare como fica a página agora.

3. No exercício 2, ao final do arquivo, antes da tag `</body>` inclua as linhas de formulário que seguem:

```
<form>
  <input type="button" value="Aperte aqui"
  onClick="alert (' clicaram-me! ' ) ">
</form>
```

Veja o resultado e depois de reler a seção 3 desta aula responda: qual a forma de incluir JavaScript usada agora?

Resumo:

Nesta aula, você aprendeu a incluir programas nas páginas HTML. O que é e como é a estrutura básica da linguagem JavaScript. Fez uso desta ferramenta escrevendo páginas com “programas embutidos” e testou seus novos conhecimentos nos exercícios.

Auto-avaliação:

Você concluiu com sucesso os exercícios? Se não entendeu bem apenas os métodos que apareceram no exemplo atividade e nos exercícios (isto é: `document.write(); for(..), onClick="alert(' clicaram-me!')"`) não se preocupe, estes pontos são para estimular sua curiosidade e lhe mostrar as potencialidades de JavaScript, e ficarão bem claros nos assuntos das próximas aulas.

Mas se teve outras dúvidas, talvez seja melhor ler novamente a aula, antes de passar para os **Comandos** que serão apresentados na próxima.

Apêndice:

Tabela 9.4 - Versões Jscript (o JavaScript da Microsoft) x versões do de JavaScript , datas de lançamento e navegadores que os suportam

Versão	Data	Introduzido pelo	Baseado no(a)	~ JavaScript
1.0	Agosto de 1996	Explorer 3.0	JavaScript do Netscape	1.0
2.0	Janeiro de 1997	IIS 3.0	JavaScript do Netscape	1.1
3.0	Outubro de 1997	Explorer 4.0	1ª ed. do ECMA-262	1.3
4.0		Visual Studio 6.0	1ª ed. do ECMA-262	1.3
5.0	Março de 1999	Explorer 5.0	2ª ed. do ECMA-262	1.4
5.1		Explorer 5.01	2ª ed. do ECMA-262	1.4
5.5	Julho de 2000	Explorer 5.5	3ª ed. do ECMA-262	1.5
5.6	Outubro de 2001	Explorer 6.0	3ª ed. do ECMA-262	1.5
5.7	Novembro de 2006	Explorer 7.0	3ª ed. do ECMA-262	1.5

Tabela 9.5 - Versões do JavaScript suas datas de lançamento e as versões dos navegadores que as suportam.

JavaScript Versão	Lançamento	Versão Explorer	Versão Netscape
1.0	Dezembro/95	3.0	2.0
1.1	Novembro/96	4.0	3.0
1.2	Dezembro/97	4.0	4.0
1.3	Mai/98	5.0	4.06
1.4	Outubro/98		4.07
1.5	Setembro/2000		6.0