WEBMASTER

JavaScript DHTML

JavaScript e HTML Dinâmico

JavaScript e HTML Dinâmico

Desenvolvimento, Aplicações e Referência

W3C - World Wide Web Consortium

HTTP://

Rafael Feitosa

Índice

PARTE I – Introdução ao JavaScript

1 – Introdução **5**

2 – Inserir JavaScript numa página da Web **5**
O elemento <script> **5**

3 – Comentários e blocos de código **6**
Comentários **6**
Blocos de código **7**

4 – Variáveis **7**
Declaração de Variáveis **7**
Os valores das Variáveis **8**
Conversões de Valores **8**

5 – Expressões Literais **9**
Representação de valores **9**
Números inteiros **10**
Números com vírgula flutuante **11**
Valores lógicos (booleanos) **11**
Expressões de texto **11**
Caracteres de escape **12**

6 – Cadeias de variáveis (Array) **13**

7 – Operadores **14**
Operadores de atribuição de valor **14**
Operadores de comparação **14**
Operadores aritméticos **15**
Operadores lógicos **15**

8 – Objetos **16**
Exemplos práticos com objetos **16**

9 – Definir uma Função **17**

10 – As instruções condicionais if...else **19**

11 – Executar código repetidamente **19**
Ciclos for **19**
Ciclos while **21**

PARTE II – Programação em JavaScript

1 – Revisão de matérias importantes **23**

2 – Os operadores da linguagem JavaScript **38**
Operadores aritméticos **38**
Operadores de atribuição (formas abreviadas) **38**
Operadores de comparação **38**
Operadores lógicos **39**

Adição de texto....44

3 – Instruções condicionais....45
As instruções if e if ... else....45
Atribuição condicional de valores....49
A instrução switch....50

4 – Execução repetida de código....54
Ciclos for....55
Ciclos while e ciclos do...while....57

5 – Código robusto: as instruções try ... catch....59

6 – Construir e usar uma função....63
Funções com um número fixo de argumentos....64
Funções com um número variável de argumentos....66

7 – Trabalhar com objetos....71
Criar um novo objeto....72
Propriedades de objetos....72
Métodos de objetos....72
Objetos definidos pelo padrão ECMAScript....73
As declarações this e with....73

8 – Objeto Array....76
Propriedades do objeto Array....76
Métodos do objeto Array....76
Coleções....77

9 - Objeto Date....81
Métodos do objeto Date....82
Métodos estáticos do objeto Date....84

10 – Objeto Math....89
Propriedades do objeto Math....89
Métodos do objeto Math....89

11 – Objeto String....94
Propriedades do objeto String....94
Métodos do objeto String....95
Métodos estáticos do objeto String....95

PARTE III – HTML Dinâmico

1 – Para que serve o HTML Dinâmico?....100

2 – O DHTML ainda é pouco aproveitado. Porquê?....108

3 – Que ferramentas vamos usar?....109
Os objetos do DOM....109
Objetos principais usados em DHTML....109

4 – O objeto window....111
Propriedades....111
Coleções....112

Métodos....112

5 – O objeto document....122
Propriedades....122
Coleções....123
Métodos....123

6 – O objeto event....132
Propriedades....132
Funções de compatibilidade para o objeto event....132

7 – O objeto navigator....144
Propriedades....144

8 – O objeto screen....146
Propriedades....146

9 – O objeto location....147
Propriedades....148
Métodos....148

10 – O objeto history....150
Propriedades....150
Métodos....150

PARTE IV– Controle do Elementos HTML

11 – Objetos que representam elementos do HTML....151
anchor....151
applet....153
embed....154
frame....154
frameset....155
form....155
iframe....158
image....160
input....163
object....168
option....169
select....172
table....178
tablecell....186
tablerow....189
textarea....193

12 – Mais controle sobre os elementos do HTML....196
Propriedades intrínsecas dos elementos do HTML....197
A propriedade style....197
A propriedade innerHTML....198
A propriedade id e o método getElementById()....199
Posicionamento e medição de elementos em DHTML....201

PARTE I: Introdução ao JavaScript

1. Introdução

O JavaScript é uma linguagem de programação simples criada para dar mais interatividade e maior funcionalidade às páginas da Web. Tendo sido inicialmente desenvolvida pela Netscape, a linguagem JavaScript acabou por dar origem à especificação técnica ECMAScript, que é um padrão oficial reconhecido pela indústria. Apesar desta linguagem ser mais conhecida pelo nome de JavaScript, e de a versão produzida pela Microsoft ter recebido o nome de JScript, a verdade é que se tratam de implementações que sendo fiéis à norma ECMAScript lhe acrescentaram novas funcionalidades úteis, mas respeitando sempre as especificações oficiais.

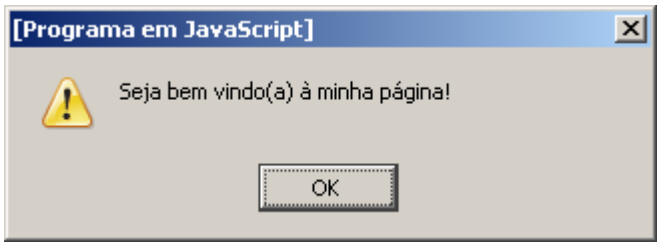
O código escrito em JavaScript destina-se a ser executado pelo web browser quando a página HTML que o contém é visualizada. Ele é uma parte integrante da página e permite que o browser seja capaz de tomar decisões quanto ao modo com que o conteúdo é apresentado ao usuário e como pode ser manipulado.

2. Inserir JavaScript numa página da Web

2.1 O elemento <script>

Os browsers capazes de executar código escrito em JavaScript reconhecem o elemento <script>. É dentro desse elemento que se coloca todo o código, como ilustra o exemplo seguinte:

```
<html>
<head>
<title>A Minha Página com JavaScript</title>
<script type="text/javascript">
   alert("Seja bem vindo(a) à minha página!");
</script>
</head>
<body>
   Aqui colocamos o conteúdo da página em HTML
</body>
</html>
```

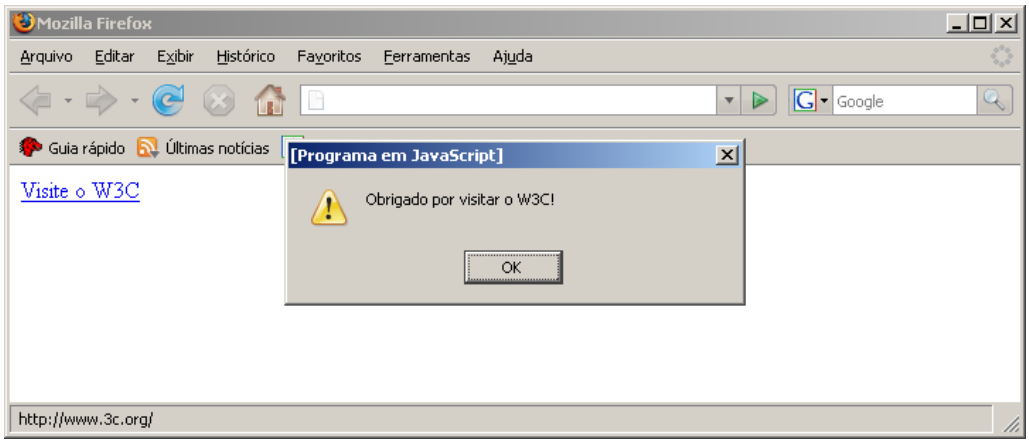


Repare que no final da linha de código colocamos o caractere ; o qual dá ao interpretador de JavaScript a indicação de que a instrução termina nesse local. O JavaScript não nos obriga a terminar as instruções deste modo, bastando que mudemos de linha para que ele perceba que a instrução chegou ao fim. No entanto isso torna mais difícil a localização dos erros e pode também contribuir para gerar mais erros. É conveniente que os principiantes terminem todas as instruções com o caractere ; e, se preferirem, só deixem de o fazer quando se sentirem completamente à vontade com a linguagem.

Graças ao JavaScript podemos fazer com que os objetos gráficos apresentados na página (como por exemplo uma imagem, um botão ou uma ligação de hipertexto) respondam dinamicamente às ações do usuário. Para que isso aconteça basta adicionar um novo atributo ao elemento responsável pela apresentação desse objeto e escrever o código que ao ser executado dará origem ao

comportamento esperado. O exemplo seguinte executa uma caixa de diálogo com um agradecimento sempre que o link for clicado:

```
<html>
<body>
   <a href="http://www.w3c.org/" target="_blank"
   onclick="alert('Obrigado por visitar o W3C!')">Visite o W3C</a>
</body>
</html>
```



Certamente já conhece bem o atributo href="...", que serve para especificar o URL da página a que a ligação de hipertexto conduz, mas note que o atributo onclick="..." é bem diferente porque o seu conteúdo é constituído por código JavaScript, que neste caso faz aparecer a caixa de diálogo com a mensagem de agradecimento. (Se não conseguir compreender o modo como o texto contido no atributo onclick consegue fazer isto não se preocupe, esta técnica, entre outras, serão explicadas durante este curso).

3. Comentários e blocos de código

3.1 Comentários

Os comentários permitem-nos descrever o código JavaScript que produzimos tornando-o mais legível e mais fácil de manter. Se comentar adequadamente o código que produz, quando mais tarde precisar de o melhorar ou fazer alterações será mais fácil e rápido perceber o que fez antes. Se você desenvolver um código para partilhar com outras pessoas então os comentários são ainda mais importantes para que os outros percebam aquilo que você escreveu.

Em JavaScript podemos usar comentários com uma única linha e comentários com várias linhas. Os comentários com uma única linha começam com os caracteres //. Isto dá ao interpretador de JavaScript a indicação de que o resto da linha é um comentário, deste modo este ignora o resto da linha, continuando a interpretar o código na linha seguinte.

Um comentário que se estende por várias linhas começa com a seqüência de caracteres /* e continua até ser encontrada a seqüência de caracteres */, que marcam o fim do comentário. Ao encontrar a seqüência /* o interpretador de JavaScript procura imediatamente a seqüência de finalização */, continuando aí a interpretação do código e ignorando o que está no meio.

Abaixo estão alguns exemplos de comentários em JavaScript.

```
// Este é um comentário com uma única linha

/* Este comentário ocupa uma só linha mas podia ocupar mais */

/*
Este comentário ocupa várias linhas. Tudo o que for
escrito aqui dentro será ignorado pelo interpretador
de JavaScript
*/
```

3.2 Blocos de código

Quando temos que executar funcionalidades não triviais é quase sempre necessário executar seqüências de instruções compostas por várias linhas. Se essas seqüências tiverem de ser executadas condicionalmente (veja por exemplo a descrição da instrução if mais à frente), ou se formarem uma função, então elas constituem um bloco e têm de ser agrupadas. Isso se consegue colocando-as entre chaves ({ }.)

```
{       // isto é um bloco de código
        var i = 0;
        var j = i * 3;
}
```

4. Variáveis

O que são as Variáveis?

As variáveis são objetos que servem para guardar informação. Elas permitem-nos dar nomes a cada um dos fragmentos de informação com que temos que lidar. Se esses nomes forem bem escolhidos fica fácil saber onde é que se deve guardar um determinado pedaço de informação e onde é que se pode ir buscar a informação que se guardou antes. Para evitar erros e aumentar a produtividade é importante escolher nomes que descrevam aquilo que cada variável guarda. Assim, se escrevermos um programa que divide dois números é recomendado chamar dividendo, divisor e quociente os números envolvidos na operação. Escolhas como por exemplo n1, n2 e n3, apesar de funcionarem, provocam confusão e dão origem a erros difíceis de detectar porque tornam o código mais difícil de ler.

É importante que saibamos quais as regras que temos de respeitar quando escolhemos um nome para uma variável:

- Todos os nomes têm que começar com uma letra ou com o caractere _.
- Os restantes caracteres que compõem o nome podem igualmente conter números. Nunca se esqueça que para o JavaScript letra maiúscula e letra minúscula são coisas diferentes e que, por exemplo, as variáveis variavel1, Variavel1 e vaRiavel1 são três objetos distintos.

4.1 Declaração de Variáveis

Ao ato de criar uma variável se dá o nome de declaração. As variáveis que são declaradas fora de qualquer função (mais à frente iremos ver exemplos de declarações de variáveis e o que são funções) são designadas por variáveis globais. Aqui o termo global significa que a variável em causa pode ser utilizada em qualquer parte do script; ela está permanentemente acessível. Quando uma variável é declarada dentro de uma função ela será uma variável local porque só pode ser utilizada dentro dessa função.

Se tentarmos acessar uma variável local fora da função em que ela foi declarada será gerado um erro porque a variável só existe no universo da função em que foi declarada; ela não faz parte do mundo exterior a essa função e como tal não pode ser utilizada.

A seguir temos alguns exemplos de declaração de variáveis:

```
dividendo = 12;
divisor = 3;
sabor = "Doce";
pi = 3.14159;
```

Nestes exemplos todas as variáveis declaradas serão variáveis globais. Se quisermos declarar variáveis cuja existência se limite a uma pequena seção do código teremos de usar a declaração var, assim: var dividendo = 12;

Se usarmos esta declaração fora de qualquer função então, porque a variável é declarada na base da estrutura de código, ela será global.

Temos assim que a declaração var serve para limitar o contexto em que a variável existe e que:

- As variáveis declaradas sem a declaração var são variáveis globais;
- As variáveis declaradas usando a declaração var existem apenas no contexto em que foram definidas.

Antes de começar a escrever código em JavaScript é importante planejar o modo como este será organizado. Deve-se começar a identificar os dados que vão ser utilizados. A seguir escolhem-se os nomes das variáveis que vão guardar esses dados e só depois é que se começa a escrever o código propriamente dito.

4.2 Os valores das Variáveis

A linguagem JavaScript é capaz de reconhecer três tipos de dados:

- Números, como por exemplo 12 ou 3.14159
- Texto (variáveis de tipo String), como por exemplo: "Seja Bem Vindo(a)!"
- Valores lógicos (true ou false)
- null, que é uma palavra especial que significa que a variável em causa não guarda qualquer valor, está vazia.

Como iremos ter oportunidade de aprender neste e nos cursos seguintes, usando apenas estes tipos de dados podemos executar muitas ações úteis nas nossas páginas da Web.

4.3 Conversões de Valores

A linguagem JavaScript exige pouco trabalho ao programador para definir o tipo de dados que uma variável deve guardar. É o próprio interpretador de JavaScript que em função dos dados que recebe decide se estes representam um número, texto (string), um valor lógico, ou nada (null). Assim, se escrever:

```
var resposta = 42;
```

o interpretador decidirá guardar internamente a variável resposta como um número inteiro, mas se escrevermos:

```
var resposta = 42;
resposta = "JavaScript é muito fácil de aprender.";
```

ao chegar à segunda linha de código o interpretador mudará de idéia e a variável resposta deixará de ser guardada internamente como um número inteiro para passar a ser guardada como uma String (texto). Esta conversão no tipo da variável acontece de forma automática e o programador não precisa fazer nada para que ela aconteça.

Esta liberdade que nos é dada pelo JavaScript destina-se apenas a simplificar a escrita do código. Quando é mal utilizada ela pode dar origem a código difícil de ler e a erros. As regras de boa programação dizem que ao definir uma variável o programador deve decidir qual o tipo de dados (número, texto ou valor lógico) que esta irá conter e não deverá escrever código que provoque uma conversão no tipo de dados que a variável guarda. Sempre que uma tal conversão for necessária deverá ser definida uma nova variável para guardar o resultado da conversão, mantendo inalterados os tipos das variáveis antigas. Na prática esta recomendação raramente é seguida.

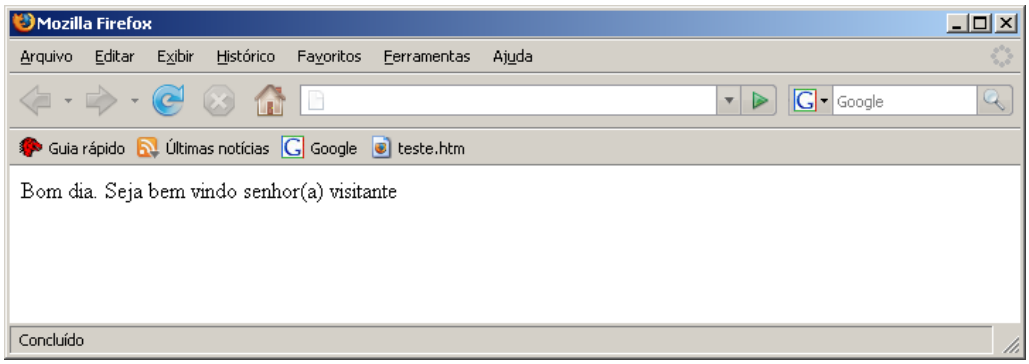
5. Expressões Literais

5.1 Representação de valores

As expressões literais representam valores fixos. Elas são escritas diretamente pelo programador ao produzir o script. Exemplos de expressões literais podem ser: 123 ou "Isto é uma expressão literal".

As expressões literais podem ser usadas de diversas maneiras, como ilustra o exemplo de código apresentado a seguir (o exemplo seguinte usa as instruções if/else que só são estudadas mais à frente):

```
<html>
<body>
<script type="text/javascript">
<!--
   var nome = "visitante";
   var hora = 11;

   if(hora < 12)
      document.write("Bom dia. Seja bem vindo senhor(a) " + nome);
   else {
      if(hora >= 13)
         document.write("Boa tarde. Seja bem vindo senhor(a) " + nome);
      else
         document.write("Seja bem vindo! Almoça conosco?");
}
-->
</script>
</body>
</html>
```



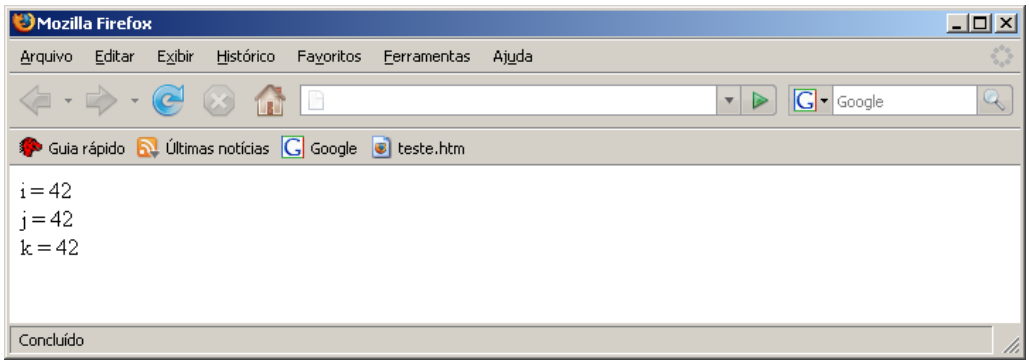
Na primeira linha usamos a expressão literal "visitante" para dar um valor inicial à variável nome. Na segunda linha usamos uma expressão literal numérica para dar um valor à variável hora. O resto do código usa as expressões literais 12 e 13 para determinar a parte do dia (manhã, tarde ou hora de almoço) e cumprimentar usando o texto (expressão literal) mais adequado.

5.2 Números inteiros

Os números inteiros podem ser expressos na forma decimal (base 10), hexadecimal (base 16) ou octadecimal (base 8). Um número decimal consiste numa seqüência de dígitos que nunca deve começar por 0 (zero). Se escrevermos um número com um zero no início isso significa que se trata de um número escrito na forma octadecimal. Por outro lado, se no início escrevermos os caracteres 0x (ou 0X) isso significa que o número está escrito na forma hexadecimal. Os números escritos na forma decimal podem conter os dígitos (0-9), a forma octadecimal aceita apenas dígitos de (0-7) e a forma hexadecimal aceita os dígitos (0-9) mais as letras a-f e A-F.

Exemplos de números inteiros são: 42, 052, 0X2A, que representam todos o valor decimal 42. No exemplo seguinte as variáveis i, j, k possuem todas o mesmo valor, apesar de serem usadas bases diferentes para as inicializar:

```
<html>
<body>
<script type="text/javascript">
<!--
   var i = 42;       // decimal
   var j = 052;      // octal
   var k = 0X2A;     // hexadecimal

   // quando executar este código repare que as variáveis
   // têm todas o mesmo valor

   document.write("i = " + i);
   document.write("<br>");
   document.write("j = " + j);
   document.write("<br>");
   document.write("k = " + k);
-->
</script>
</body>
</html>
```



5.3 Números com vírgula flutuante

Uma expressão literal com vírgula flutuante representa um número que não é inteiro mas que contém uma parte inteira e uma parte fracionária. Os números 21.37 e -0.0764 são exemplos disto. A representação que a máquina constrói para estes números baseia-se na notação científica. Por exemplo, o número -7645.4532 é igual a -7.64532 multiplicado por 10 elevado a 3, e escreve-se como -7.6454532E3, em que E3 representa 10 elevado a 3. Um outro exemplo é o número 0.00045431, que é representado na forma 4.5431E-4, ou seja 4.5431 multiplicado por 10 elevado a -4. Esta representação é construída automaticamente pela máquina, o programador pode escrever o número na forma que mais lhe agradar.

5.4 Valores lógicos (booleanos)

Estas expressões podem assumir apenas dois valores: true (verdadeiro) e false (falso).

5.5 Expressões de texto

Uma expressão de texto é composta zero ou mais caracteres colocados entre aspas ("), como por exemplo "esta é uma expressão de texto", ou entre apóstrofos ('), como por exemplo 'esta é outra expressão de texto'. Se começarmos a expressão com aspas temos a obrigação de usar aspas para a terminar, e se a iniciarmos com um apóstrofo temos de usar outro apóstrofo para a terminar.

Além dos caracteres normais, as expressões de texto podem conter os caracteres especiais apresentados na lista seguinte:

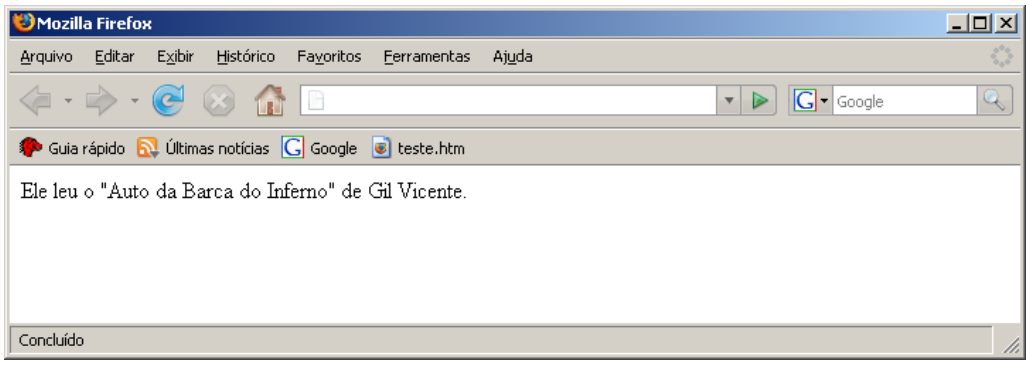
Caractere	Significado
\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	tab
\\	backslash

Cada um destes caracteres produz o mesmo resultado que se obtém acionando a tecla indicada na segunda coluna. Assim o caractere \b equivale a acionar a tecla backspace (apagar o caractere à esquerda). O caractere \n provoca uma mudança de linha tal como a tecla "enter". O caractere \ é usado como prefixo dos outros caracteres especiais, o que faz também dele um caractere especial. Por isso, para obtermos este caractere temos de temos escrevê-lo duas vezes (\\). Se o escrevermos uma única vez ao invés de o obtermos estaremos tentando introduzir um outro caractere especial e o resultado será diferente do que pretendemos.

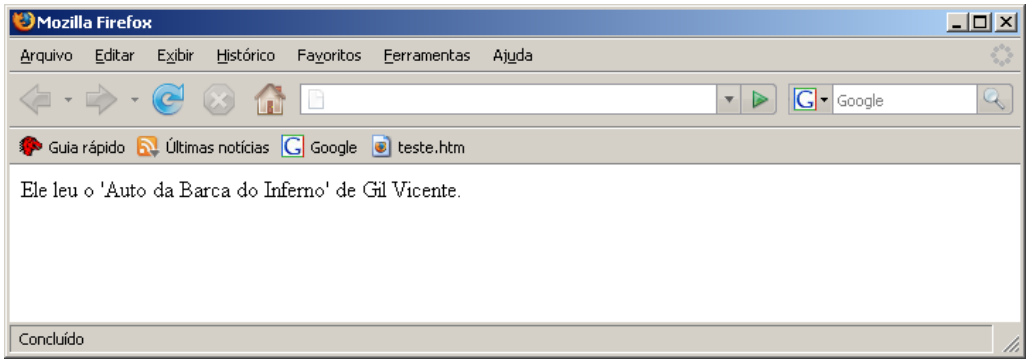
5.6 Caracteres de escape

Se o caractere que vem a seguir a \ não pertencer à lista anterior o seu efeito será nulo, mas há duas exceções: as aspas (") e o apóstrofo ('). Se pretendemos escrever aspas dentro de uma expressão de texto temos de colocar o caractere \ antes delas, como mostra o exemplo seguinte:

```
<html>
<body>
<script type="text/javascript">
<!--
   var texto = "Ele leu o \"Auto da Barca do Inferno\" de Gil Vicente.";
   document.write(texto);
-->
</script>
</body>
</html>
```



Se ao invés de usar aspas usarmos apenas apóstrofos teremos:



Porém, a melhor solução para este problema não é nenhuma das anteriores. Se usarmos apóstrofos como caracteres delimitadores de uma string então passamos a poder usar as aspas como parte do conteúdo sem qualquer problema, como se mostra a seguir:

```
var texto = 'Ele leu o "Auto da Barca do Inferno" de Gil Vicente.';
document.write(texto);
```

Mas se quisermos colocar apóstrofos no conteúdo a melhor forma de evitarmos os problemas consiste em usar aspas como caracteres delimitadores da string, como se mostra a seguir:

```
var texto = "Ele leu o 'Auto da Barca do Inferno' de Gil Vicente.";
document.write(texto);
```

6. Cadeias de variáveis (Array)

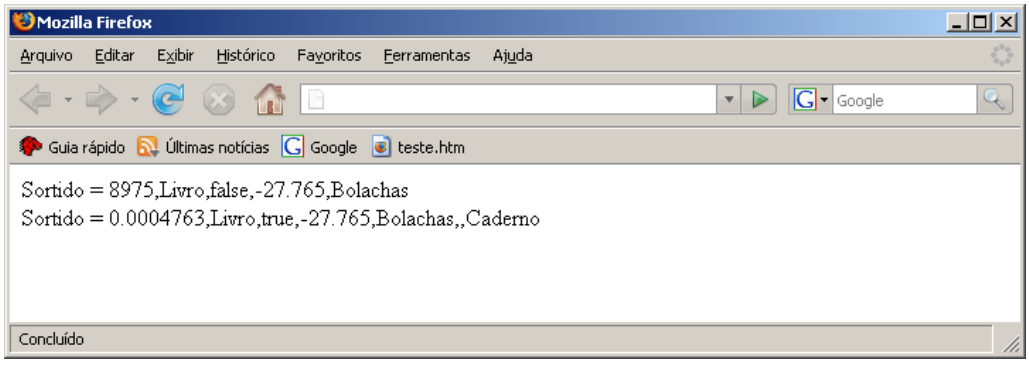
Uma cadeia de variáveis (objeto Array) é um objeto capaz de guardar muitos valores, tantos quanto a memória disponível na máquina permitir. Cada uma das variáveis que compõem o array possui um índice. Ilustremos isto com um exemplo:

```
var frutas_tropicais = new Array("Goiaba", "Manga", "Maracujá");
var frutas_nacionais = new Array(3);
frutas_nacionais[0] = "Maçã";
frutas_nacionais[1] = "Cereja";
frutas_nacionais[2] = "Laranja";
```

Ao declararmos a variável frutas_tropicais nós declaramos o Array e atribuímos-lhe os valores numa única operação. Já no segundo caso primeiro declaramos o Array e só depois definimos os valores que ele deve conter. Neste caso temos que a variável frutas_tropicais[2] possui o valor "Maracujá" e a variável frutas_nacionais[0] possui o valor "Maçã".

Em JavaScript as variáveis não têm um tipo definido, por isso um array pode conter valores de tipos diferentes que podemos alterar sempre que necessário, como se mostra a seguir:

```
<html>
<body>
<script type="text/javascript">
<!--
   var sortido = new Array(8975, "Livro", false, -27.765, "Bolachas");
   document.write("Sortido = " + sortido);

   sortido[0] = 0.0004763;
   sortido[2] = true;
   sortido[6] = "Caderno";

   document.write("<br>");
   document.write("Sortido = " + sortido);
-->
</script>
</body>
</html>
```



Se atribuirmos um valor a um elemento do array com um índice mais alto do que o seu comprimento, o sistema JavaScript resolve o problema aumentando o tamanho do array até chegar ao índice pretendido. É isso que acontece no exemplo anterior quando se chega à linha que tem sortido[6] = "Caderno"; Os arrays são objetos, e entre as suas propriedades conta-se a propriedade length, que nos dá o número de elementos (variáveis) que ele contém num determinado momento. Assim, se ao exemplo anterior juntarmos uma linha com o seguinte código:

```
var numeroDeElementos = sortido.length;
```

a variável numeroDeElementos ficará com o valor 7 (repare que inserimos um elemento adicional com o índice 6, o que fez crescer o array). De forma análoga se usarmos frutas_nacionais.length iremos obter 3.

7. Operadores

A linguagem JavaScript possui muitos operadores de diversos tipos. Aqui iremos abordar apenas os aspectos mais básicos dos operadores disponíveis.

7.1 Operadores de atribuição de valor

Uma das coisas que os operadores podem fazer é fornecer um valor àquilo que estiver à sua esquerda. Se o que está à esquerda for uma variável então o valor dela passará a ser aquilo que o operador forneceu, se for outro operador o valor fornecido será usado como operando.

Os operadores mais conhecidos são as quatro operações aritméticas básicas (adição, subtração, multiplicação e divisão.) Para estes a linguagem JavaScript define as seguintes versões curtas:

Versão curta	Significado
x+=y	x=x+y
x-=y	x=x-y
x*=y	x=x*y
x/=y	x=x/y

Repare que aqui o sinal = não representa a igualdade matemática. Ele serve apenas para indicar que a variável que está à sua esquerda deve passar a ter um valor igual ao valor da expressão que está à sua direita. Se tivermos x=5 e y=7 a expressão x=x+y não representa uma igualdade matemática mas sim a indicação que o valor de x deve passar a ser igual à soma do valor que tem atualmente com o valor de y. Neste caso x passaria a valer 12.

7.2 Operadores de comparação

Um operador de comparação compara os valores que lhe são fornecidos (que designamos por operandos) e retorna um valor lógico que indica se o resultado da comparação é verdadeiro ou falso. Os valores que recebe para analisar podem ser números ou variáveis de texto (string). Quando atuam sobre variáveis de texto, as comparações baseiam-se na forma como os caracteres estão ordenados seqüencialmente. Esta ordenação baseia-se na ordem alfabética. A lista seguinte apresenta estes operadores.

Operador	Descrição	Exemplo
Igualdade (==)	Verifica se os dois operandos são iguais	x==y dá true se x igualar y
Desigualdade (!=)	Verifica se os operandos são desiguais	x!=y dá true se x não for igual a y
Maior do que (>)	Verifica se o operando da esquerda é maior do que o da direita	x>y dá true se x for maior do que y
Maior ou igual (>=)	Verifica se o operando da esquerda é maior ou igual ao da direita	x>=y dá true se x for maior ou igual a y
Menor do que	Verifica se o operando da esquerda é	x<y dá true se x for menor

(<)	menor do que o da direita	do que y
Menor ou igual (<=)	verifica se o operando da esquerda é menor ou igual ao da direita	x<=y dá true se x for menor ou igual a y

7.3 Operadores aritméticos

Um operador aritmético recebe valores numéricos (tanto variáveis como expressões literais) e produz um valor numérico como resultado. Os operadores numéricos mais importantes são a adição (+), a subtração (-), a multiplicação (*), a divisão (/) e o resto da divisão (%). O funcionamento destes operadores em JavaScript respeita todas as regras da álgebra.

Devido a muitas vezes ser necessário adicionar ou subtrair uma unidade a uma variável, a linguagem JavaScript define dois operadores especiais com esta finalidade. Assim, para adicionarmos uma unidade à variável variavel1 podemos escrever variavel1++, e para subtrairmos uma unidade à variavel2 escrevemos variavel2--. Por ação destes operadores no final do exemplo seguinte a variável variavel1 terá o valor 4 e a variável variavel2 terá o valor 6.

```
var variavel1 = 3;
variavel1++;
var variavel2 = 7;
variavel2--;
```

7.4 Operadores lógicos

Os operadores lógicos aceitam os valores lógicos true e false (verdadeiro e falso) como operandos e retornam valores lógicos como resultado. Os operadores lógicos base encontram-se listados a seguir (os restantes definem-se com base nestes três).

Operador	Utilização	Descrição
e (&&)	b && c	Dá true se b for true e c for true.
ou (||)	b || c	Dá false se b for false e c for false. Dá true nos casos restantes.
negação (!)	!b	Dá true se b for false e dá false se b for true.

Os casos mais úteis e interessantes de uso destes operadores utilizam dois ou os três operadores ao mesmo tempo, como mostra a seguir:

Se tivermos x = 4 e y = 7 a operação seguinte dá false:

```
((x + y + 2) == 13) && (((x + y) / 2) == 2)
```

Se tivermos x = 4 e y = 7 a operação seguinte dá true:

```
((y - x + 9) == 12) || ((x * y) == 2)
```

Se tivermos x = 4 e y = 7 a operação seguinte dá false:

```
!((x/2 + y) == 9) || ((x * (y/2)) == 2)
```

8. Objetos

O objetivo da coleção de documentos de estudo de que este curso faz parte é ensinar as tecnologias padrão definidas para criar páginas e aplicações para a Web. A

utilização dos objetos da linguagem JavaScript é aqui tratada de forma rápida. O estudo aprofundado deste tópico será feito no Curso de Programação em JavaScript.

Objetos definidos no padrão ECMAScript

A linguagem JavaScript é uma implementação do padrão ECMAScript. Esse padrão define as regras de sintaxe que estamos estudando e um conjunto mínimo de objetos que fazem do ECMAScript uma verdadeira linguagem de programação, mas não define os objetos que permitem manipular e interagir tanto com o browser como com as páginas da Web. Para ser verdadeiramente útil o JavaScrit tem de complementar o ECMAScript com objetos adicionais:

Document Object Model (DOM)

O W3C (World Wide Web Consortium) definiu o padrão DOM para padronizar a forma como os browsers e as aplicações da Web manipulam e interagem com as páginas da Web. Todos os browsers modernos implementam estes padrões. Apesar de essas implementações serem geralmente incompletas, elas são suficientes para que possamos programar quase tudo numa forma que funciona em todos os browsers dominantes (MSIE 6 e superior, Mozilla/Netscape e Opera.)

Outros objetos úteis

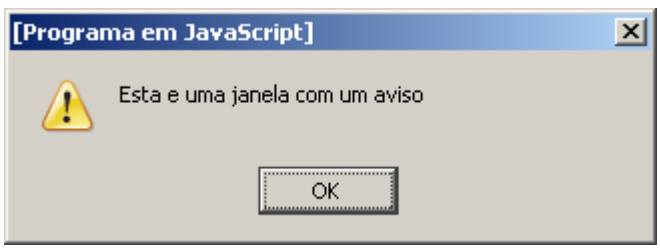
Quando a linguagem JavaScript surgiu, os seus criadores definiram aqueles objetos que lhe pareceram importantes. Dentre eles alguns foram incorporados pelo padrão ECMAScript, outros foram de alguma forma incorporados pelo DOM (geralmente com modificações), e outros não estão presentes em qualquer padrão oficial mas são suportados universalmente, o que faz deles padrões de fato.

8.1 Exemplos práticos com objetos

Dois dos objetos que ficam imediatamente disponíveis quando carrega um documento no browser são: o objeto document, que nos permite manipular e interagir com a página da Web, e o objeto window, que nos permite controlar a janela do browser que contém a página.

O objeto window possui vários métodos. Entre eles temos os métodos close(), alert(), confirm() e prompt(), com os quais podemos fechar a janela do browser, apresentar avisos ao usuário e pedir-lhe para nos dar uma resposta ou escrever alguma coisa. O código:

```
<html>
<body>
<script type="text/javascript">
<!--
   window.alert("Esta &eacute; uma janela com um aviso");
-->
</script>
</body>
</html>
```



faz aparecer uma janela com um aviso para o usuário. A notação por pontos significa que estamos chamando o método alert() pertencente ao objeto window. Neste caso podíamos ter escrito apenas alert(mensagem) e omitido a parte window (o browser já sabe que o método alert pertence ao objeto window).

O objeto document contém uma representação da página HTML. Cada um dos elementos que compõem a página (formulários, parágrafos, imagens, links, etc) podem ser lidos e manipulados utilizando este objeto. Depois de uma página estar carregada, o código seguinte:

```
alert("A segunda imagem desta página foi carregada a partir de: " +
document.images[1].src);
```

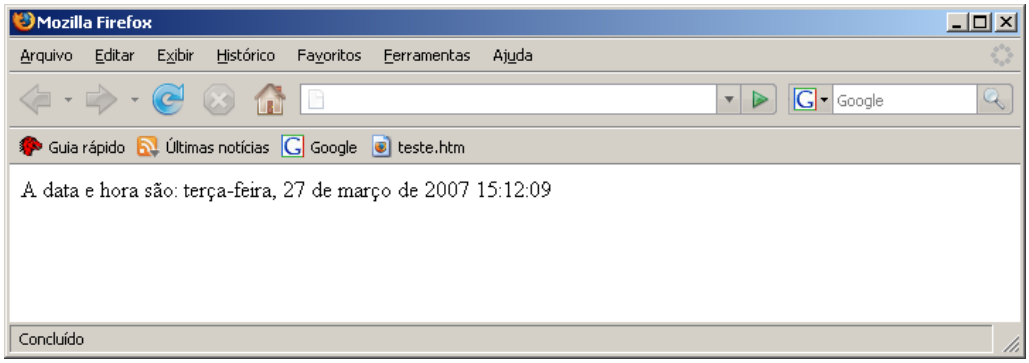
mostra a origem (src) de uma imagem. Repare que com o objeto document temos de usar sempre a notação por pontos, não sendo aceitas abreviações.

9. Definir uma Função

As funções permitem-nos agrupar várias linhas de código que realizam um determinado trabalho, dar-lhe um nome e executá-las chamando-as por esse nome.

O exemplo seguinte define uma função:

```
<html>
<body>
<script type="text/javascript">
<!--
   function dataAtual()
   {
      /*
      Cria um objeto com a data e hora atuais e mostra
      o seu valor na janela recorrendo ao método
      toLocaleString() do objeto Date
      */
      var d = new Date();
      document.write("A data e hora s&atilde;o: " + d.toLocaleString());
   }

   dataAtual(); // esta linha faz executar a função
-->
</script>
</body>
</html>
```

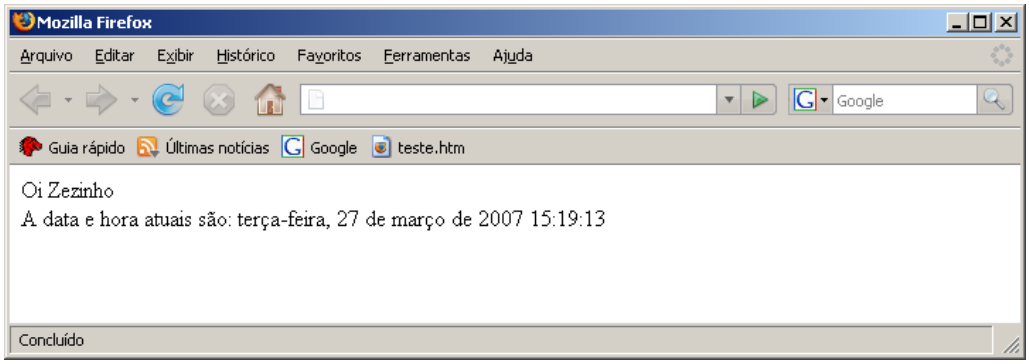


Nunca esqueça que em JavaScript as letras maiúsculas não são equivalentes às letras minúsculas, por isso tenha sempre muita atenção ao fato de que o nome que se usa para chamar uma função tem de ser rigorosamente igual ao nome dando durante sua definição.

No exemplo anterior usamos os caracteres { e } para delimitar um bloco de código. Tudo o que está dentro destes delimitadores faz parte da função e será executado sempre que esta for invocada escrevendo dataAtual() no seu código. Como resultado será escrita na página a data e hora do momento em que a função foi chamada.

Também podemos passar argumentos para a função, como mostra a seguir:

```
<html>
<body>
<script type="text/javascript">
<!--
   function cumprimentar(nome)
   {
      var d = new Date();
      document.write("Oi " + nome + "<br>A data e hora atuais s&atilde;o: "
                                  + d.toLocaleString());
   }

   cumprimentar('Zezinho'); // esta linha faz executar a função
-->
</script>
</body>
</html>
```



como teremos oportunidade de ver quando aprofundarmos o nosso estudo, as funções têm uma importância fundamental na programação de scripts complexos.

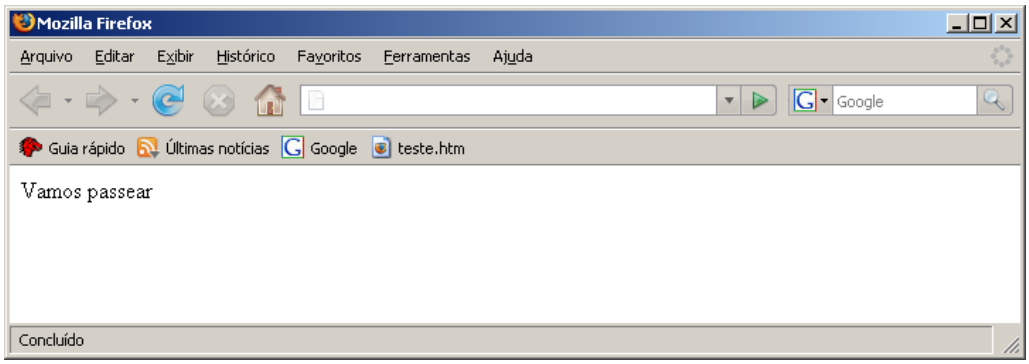
10. As instruções condicionais if...else

Uma instrução if permite-nos executar uma porção de código apenas se for verdadeira uma determinada condição. Se essa condição não for verdadeira essa porção de código não será executada, podendo ser ou não executado outro código alternativo, que será especificado através da palavra else.

A idéia principal que está na base das instruções if/else pode ser resumida numa frase: "Se chegarmos antes da hora de almoço vamos dar um passeio, caso contrário vamos para a mesa". O exemplo seguinte ilustra esta idéia:

```
<html>
<body>
```

```
<script type="text/javascript">
<!--
   var hora = 10;
   if(hora < 12)
      document.write("Vamos passear");
   else
      document.write("Vamos para a mesa");
-->
</script>
</body>
</html>
```



Neste exemplo a hora é de antes do almoço e será apresentada uma janela que tem escrito “Vamos passear”. Se a hora fosse 12 ou mais seria mostrado o texto “Vamos para a mesa”.

Uma instrução if não precisa de ter associada a si uma instrução else. Quando isso acontece se a condição não se verificar não será executado qualquer código alternativo.

11. Executar código repetidamente

Um dos recursos mais poderosos no arsenal de qualquer linguagem de programação é a capacidade para repetir a realização de tarefas de uma forma simples. Para isso definem-se ciclos de repetição dentro dos quais se coloca o código que se pretende executar repetidamente.

11.1 Ciclos for

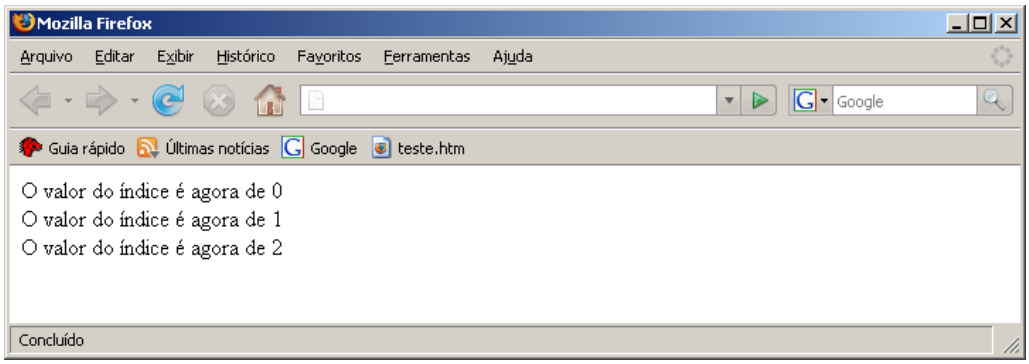
Um ciclo for consiste num conjunto de três expressões contidas entre parêntesis, separadas pelo caractere ; (ponto e vírgula) e pelo código a executar em cada um dos ciclos. Normalmente esse código estará contido entre chaves para formar um bloco, mas se consistir numa única linha não é preciso usar as chaves.

A primeira das expressões do ciclo for declara a variável a usar como índice (funciona apenas como contador) e a inicia. A segunda expressão declara uma condição que deve ser testada sempre que se inicia um novo ciclo. Se essa condição der false como resultado o ciclo pára e o código definido abaixo não voltará a ser executado. A terceira expressão serve para atualizar o valor do índice no final de cada ciclo.

Ilustremos isto com um exemplo simples:

```
<html>
<body>
<script type="text/javascript">
<!--
```

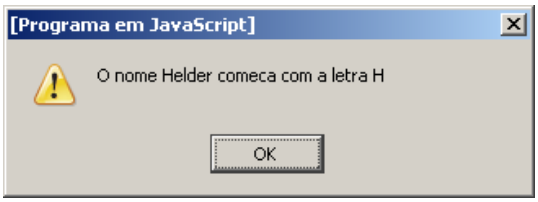
```
   soma = 0;
   for(var i = 0; i < 3; i++)
   {
      soma += i;
      document.write("O valor do &iacute;ndice &eacute; agora de " + i
      + "<br>");
   }
-->
</script>
</body>
</html>
```

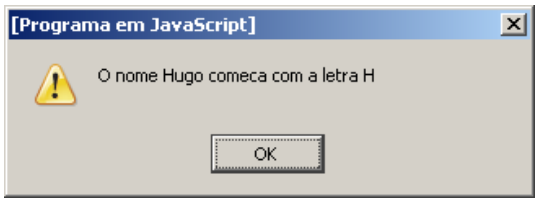


Este pedaço de código começa por definir uma variável (global) chamada soma atribuindo-lhe o valor zero. O ciclo for define uma variável de índice (var i = 0) e verifica se a condição i < 3 é cumprida. Se o resultado da verificação for true será executado o código que se encontra entre as chaves mais abaixo, o qual adiciona i à variável soma e apresenta uma mensagem informando sobre o valor atual da variável i. Depois é executada a terceira instrução do ciclo (i++), a qual soma uma unidade ao valor do índice i e dá-se início a um novo ciclo. Este começa testando de novo o respeito pela condição i < 3. Se o resultado for true volta a executar o código que está entre as chaves com o valor atualizado de i. Isto se repete até que i < 3 dê false, o que termina a execução do ciclo for.

O exemplo seguinte é mais elaborado e executa um ciclo que percorre todos os elementos de um array de nomes e destaca aqueles que começam com a letra H.

```
<html>
<body>
<script type="text/javascript">
<!--
   var nomes = new Array("Manuel", "Rita", "Joana", "Francisco", "Luís",
   "Bernardo", "Helder", "Patrícia", "Hugo", "Antônio", "Fabrício");
   for(var i=0; i < nomes.length;i++)
   {
      var nome = nomes[i]
      if(nome.charAt(0) == "H")
         alert("O nome " + nome + " comeca com a letra H");
   }
-->
</script>
</body>
</html>
```





Neste exemplo usamos o método charAt() do objeto String para verificar se o caractere inicial do nome (aquele está na posição zero) é igual à letra H.

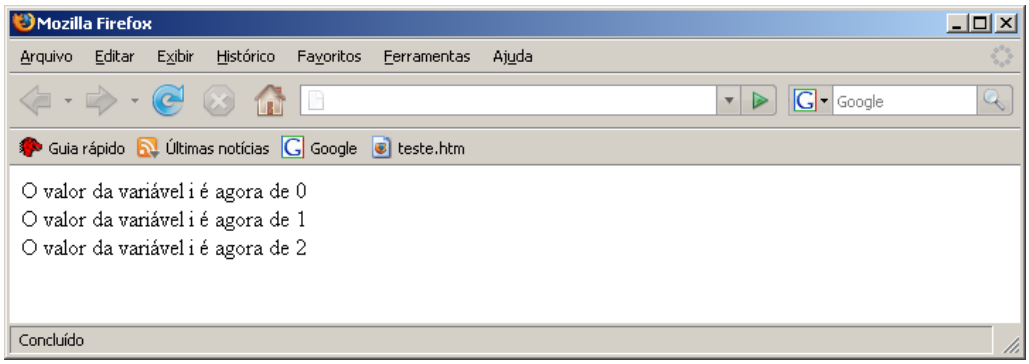
11.2 Ciclos while

O ciclo while é muito parecido com o ciclo for. De fato tudo o que um faz pode ser feito com o outro, embora por questões de legibilidade (e de elegância do código) cada um deles possa ter áreas de aplicação que para as quais é mais indicado do que o outro.

O ciclo while avalia uma condição e se ela der true executa o bloco de código que vem imediatamente a seguir. Se der false salta para frente do bloco de código que vem a seguir sem o executar.

Este exemplo usa um ciclo while para produzir o mesmo efeito que o exemplo anterior feito com o ciclo for:

```
<html>
<body>
<script type="text/javascript">
<!--
   soma = 0;
   i = 0;
   while(i < 3)
   {
      soma += i;
      document.write("O valor da vari&aacute;vel i &eacute; agora de " + i
      + "<br>");
      i++;
   }
-->
</script>
</body>
</html>
```



PARTE II: Programação em JavaScript

1. Revisão de matérias importantes

Na 1ª Parte deste curso em "Introdução ao JavaScript" nós abordamos os aspectos básicos desta linguagem de programação. Apesar disso, voltamos a chamar a atenção do leitor para algumas coisas que sendo simples podem causar problemas a quem está dando os primeiros passos.

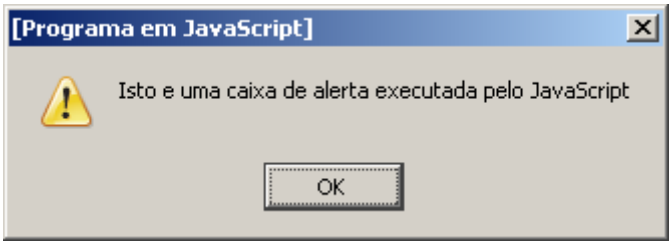
Questões básicas

Para começar certifique-se de que irá compreender bem os exercícios práticos que serão apresentados a seguir. Eles ilustram alguns aspectos importantes para o estudo que se segue. Os exercícios apresentados neste tutorial contêm explicações e devem ser estudados para compreender as matérias.

Exemplos de Aplicação

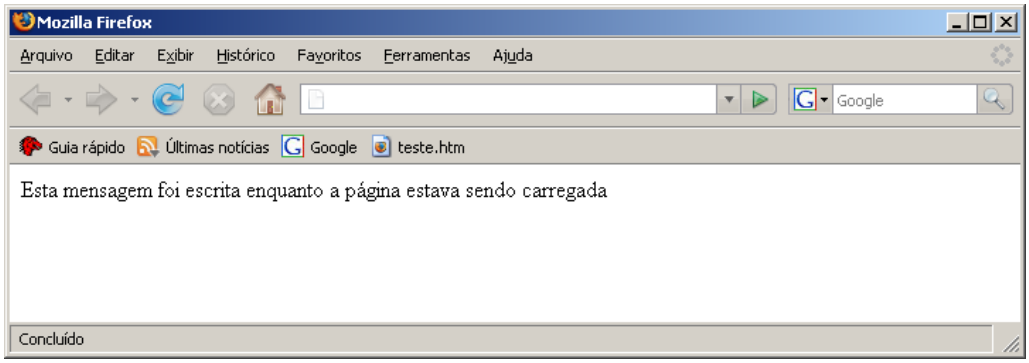
JavaScript na seção head

```
<html>
<head>
<script type="text/javascript">
<!--
   alert("Isto e uma caixa de alerta executada pelo JavaScript")
// -->
</script>
<title></title>
</head>
<body>
</body>
</html>
```



JavaScript no corpo do documento

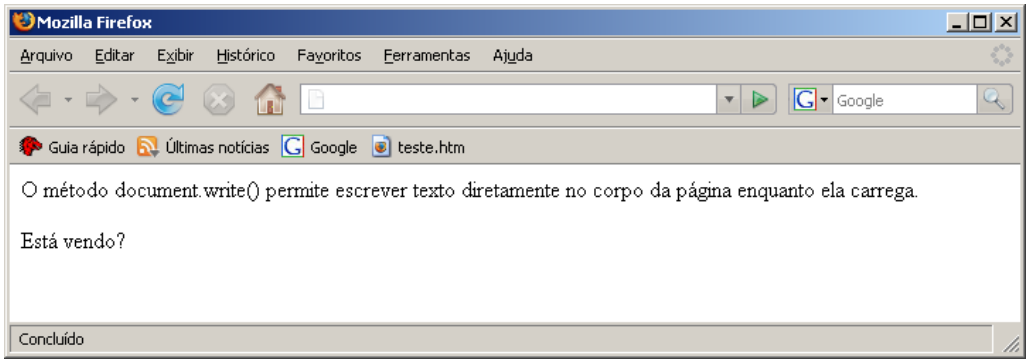
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
    document.write("Esta mensagem foi escrita enquanto a p&aacute;gina estava
sendo carregada")
// -->
</script>
</body>
</html>
```



Escrever texto na página com JavaScript

```
<html>
<head>
<title></title>
</head>
<body>
```

```
   <p>
      O método document.write() permite escrever texto diretamente no corpo da
      página enquanto ela carrega.
   </p>
<script type="text/javascript">
<!--
   document.write("Está vendo?")
// -->
</script>
</body>
</html>
```

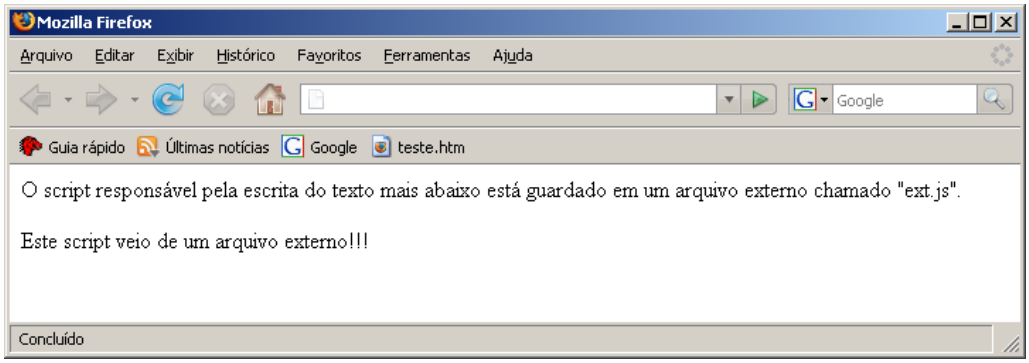


Um arquivo externo com código JavaScript

ext.js

```
document.write("Este script veio de um arquivo externo!!!")
```

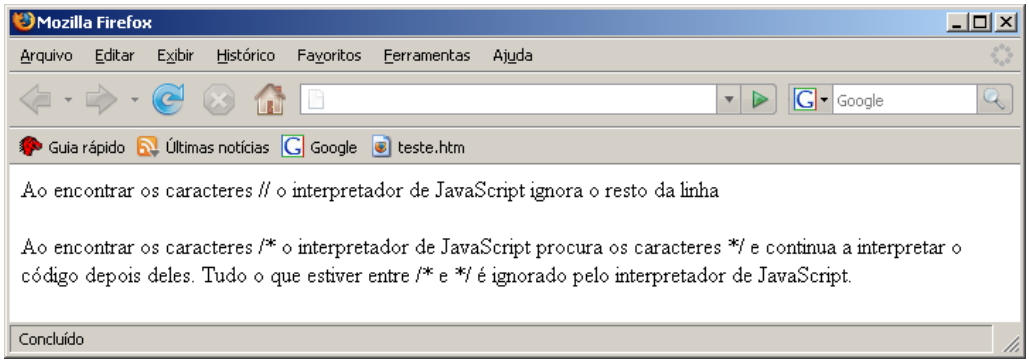
```
<html>
<head>
<title></title>
</head>
<body>
   <p>
      O script responsável pela escrita do texto mais abaixo está guardado em
      um arquivo externo chamado "ext.js".
   </p>
   <p>
      <script type="text/javascript" src="ext.js"></script>
   </p>
</body>
</html>
```



Comentários no código JavaScript

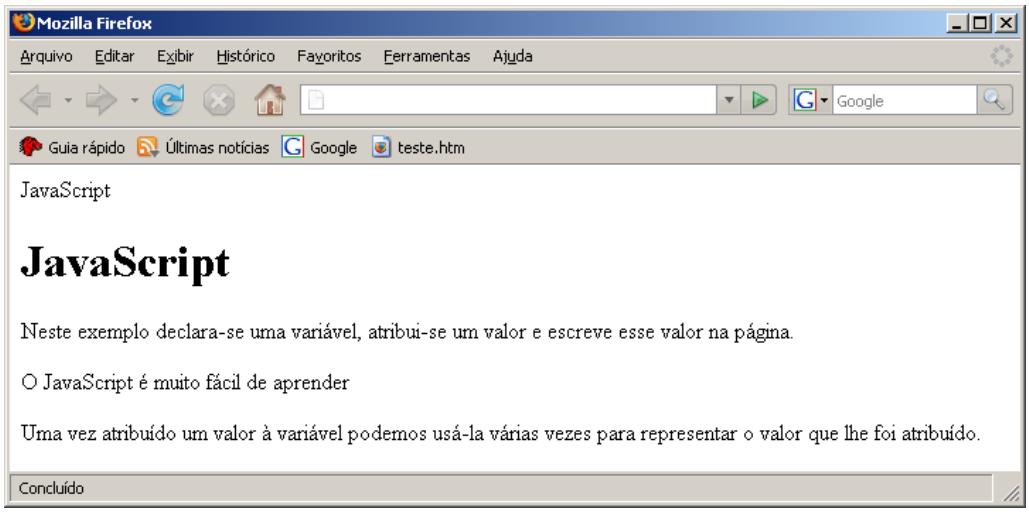
```
<html>
<head>
<title></title>
```

```
</head>
<body>
<script type="text/javascript">
<!--
   // Aqui temos um  comentário com uma única linha
   document.write("Ao encontrar os caracteres \/\/ o interpretador de JavaScript
ignora o resto da linha")
   document.write("<br><br>")
   /*
   Aqui temos um comentário com várias linhas. O interpretador de JavaScript
   ignora tudo o que estiver dentro do comentário.
*/
   document.write("Ao encontrar os caracteres /* o interpretador de ")
    document.write("JavaScript procura os caracteres */ e continua a interpretar
o código depois deles. ")
      document.write("Tudo  o  que  estiver  entre  /*  e  */  é  ignorado  pelo
interpretador de JavaScript.")
// -->
</script>
</body>
</html>
```



Declarar uma variável, atribuir-lhe um valor e apresentá-la na página

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var linguagem = "JavaScript"
   document.write(linguagem)
   document.write("<h1>"+linguagem+"<\/h1>")
// -->
</script>
   <p>
      Neste exemplo declara-se uma variável, atribui-se um valor e escreve
      esse valor na página.
   </p>
<script type="text/javascript">
<!--
   var linguagem = "JavaScript"
   document.write("<p>O "+linguagem+" é muito fácil de aprender<\/p>")
// -->
</script>
   <p>
      Uma vez atribuído um valor à variável podemos usá-la várias vezes para
      representar o valor que lhe foi atribuído.
   </p>
</body>
</html>
```



Terminar ou não as linhas com ponto e vírgula

Na 1ª Parte deste curso em "Introdução ao JavaScript", em todos os exemplos apresentados, terminamos cada uma das linhas de código com o caractere ; (ponto e vírgula.) Essa prática visa evitar alguns erros e facilita a sua detecção, por isso ela é recomendada a todos aqueles são iniciantes no estudo do JavaScript.

Porém, ao contrário do que acontece nas linguagens Java e C++, em que o uso do ponto e vírgula é obrigatório, em JavaScript ele é facultativo. A maioria dos programadores de JavaScript não gosta de ver as linhas terminadas com ponto e vírgula e por isso não o usam.

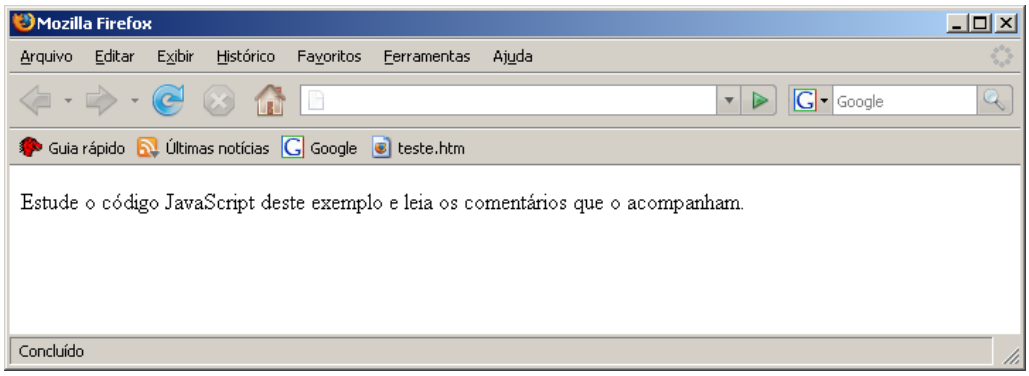
Por isso, os exemplos e exercícios de aplicação que você encontrará daqui pra frente não irão seguir a recomendação do uso do ; (ponto e vírgula), porém sempre que puder use tal recomendação. Lembre-se que você ainda é um iniciante em JavaScript.

Exemplos de Aplicação

Usar ou não usar ponto e vírgula

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   /* As linhas seguintes terminam com ponto e vírgula, que indica o fim da
   instrução. */
   var s = "Kábong";
   var z = "El ";
   var ss = z+s; var u="Babalou";
   // Repare que na linha anterior colocamos duas instruções numa só linha.
   /* Isto foi possível porque as instruções estão separadas por ponto e
   vírgula */
   /* Nas linhas seguintes não se usa ponto e vírgula. O fim da instrução é
   indicado pelo fim da linha. */
   var s1 = "Pibe"
   var z1 = "El "
   var ss1 = z1+s1
   var u1="Ganda nóia"
   /* Quando não usamos ponto e vírgula só podemos escrever uma única
   instrução em cada linha */
```

```
// -->
</script>
   <p>
      Estude o código JavaScript deste exemplo e leia os comentários que
      o acompanham.
   </p>
</body>
</html>
```



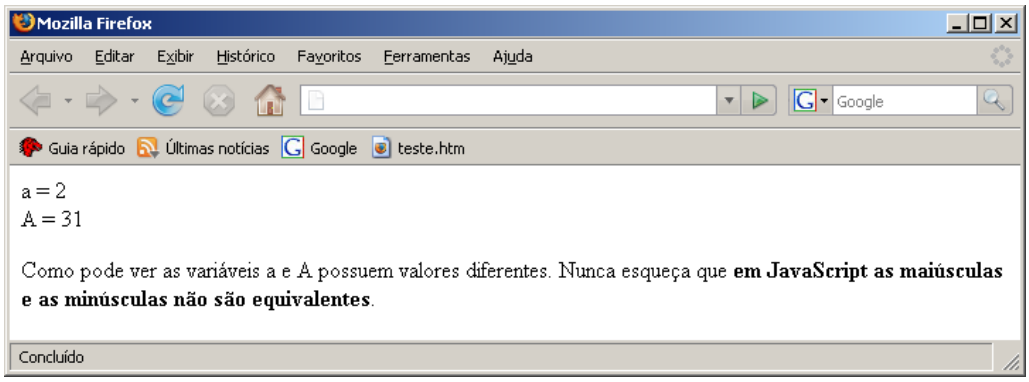
Letra maiúscula não é equivalente à letra minúscula

Uma outra coisa que devemos levar em conta é que em JavaScript não podemos trocar letras maiúsculas por minúsculas nem minúsculas por maiúsculas. Esta regra aplica-se a nomes de instruções, nomes de variáveis, nomes de funções, nomes de objetos e a tudo o que possa existir num script.

Exemplos de Aplicação

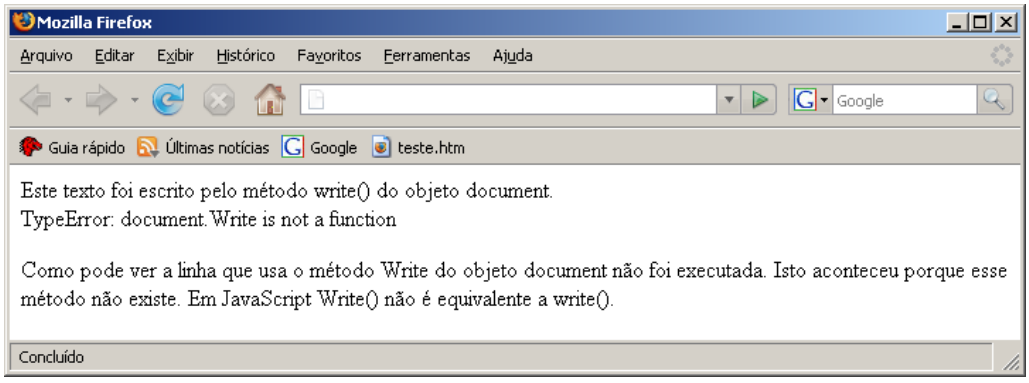
Diferenças entre letra maiúscula e letra minúscula (nomes de variáveis)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var a, A
   a=2
   A=31

   document.write("a = " + a)
   document.write("<br>")
   document.write("A = " + A)
// -->
</script>
   <p>
      Como pode ver as variáveis a e A possuem valores diferentes.
      Nunca esqueça que <b>em JavaScript as maiúsculas e as
      minúsculas não são equivalentes</b>.
</p>
</body>
</html>
```



Diferenças entre letra maiúscula e letra minúscula

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   // Estude as instruções try ... catch no capítulo 5
   try
      {
       /* Colocamos o código dentro de um bloco try ... catch porque quando
       tentarmos invocar o método Write (inexistente) do objeto document
       será gerado um erro de script*/

       document.write("Este texto foi escrito pelo método write() do objeto
document.")
       document.write("<br>")

       /* Agora no nome do método write() trocamos o w minúsculo por um W
       maiúsculo */
       document.Write("Este texto foi escrito pelo método Write() do objeto
document.")
      }
catch(e)
   {
      document.write(e)
   }
// -->
</script>
   <p>
      Como pode ver a linha que usa o método Write do objeto document não foi
      executada. Isto aconteceu porque esse método não existe. Em JavaScript
      Write() não é equivalente a write().
   </p>
</body>
</html>
```



Cuidados para se ter com as variáveis

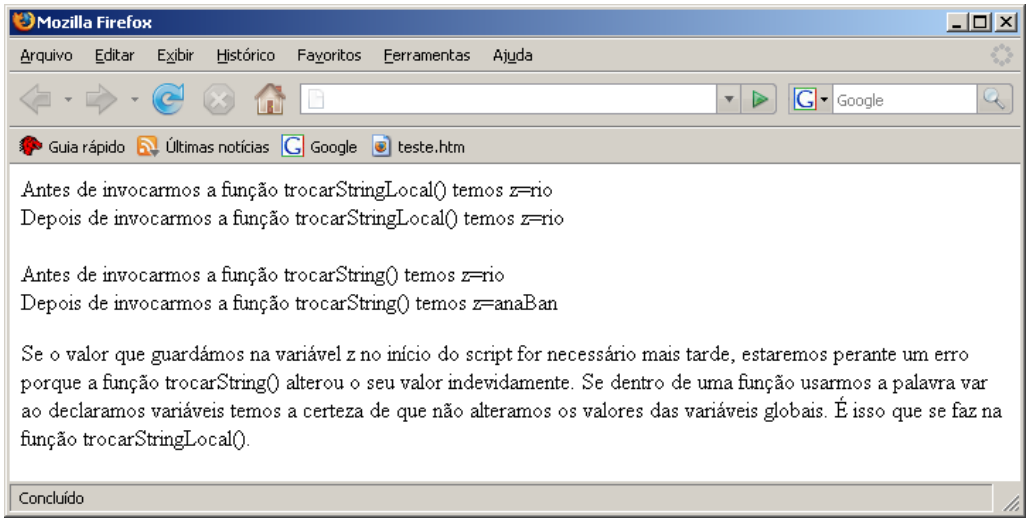
Quando os scripts que escrevemos são curtos, é fácil desenvolver de modo a que cada parte do código não perturbe o funcionamento das outras restantes. No entanto, à medida que os conhecimentos avançam, o número de funções que criamos irá aumentar. Quando isso acontece precisamos ter algum cuidado ao declararmos as variáveis porque podemos fazer com que uma função acabe alterando dados globais, o que pode provocar erros. O primeiro exemplo da lista seguinte ilustra uma situação destas:

Exemplos de Aplicação

Variáveis globais

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   var s="Canário"
   var z=s.substring(4,7) // z contém a string "rio"

   /* As duas funções seguintes fazem a mesma coisa, mas a primeira altera a
   variável global z enquanto que a segunda declara uma variável local com o
   mesmo nome deixando o valor da global inalterado */

   function trocarString(c)
   {
      var l=c.length
      // repare que não se usa a palavra var no início da próxima linha
      z=c.substring(l/2,l)+c.substring(0,l/2)
      return z
   }

   function trocarStringLocal(c)
   {
      var l=c.length
      // repare que se usa a palavra var no início da próxima linha
      var z=c.substring(l/2,l)+c.substring(0,l/2)
      return z
   }
// -->
</script>
</head>
<body>
<script type="text/javascript">
<!--
   document.write("Antes de invocarmos a função trocarStringLocal() temos
   z="+z)
   trocarStringLocal("Cereja")
   document.write("<br>")
   document.write("Depois de invocarmos a função trocarStringLocal() temos
   z="+z)
   document.write("<br><br>")
   document.write("Antes de invocarmos a função trocarString() temos z="+z)
   trocarString("Banana")
   document.write("<br>")
   document.write("Depois de invocarmos a função trocarString() temos z="+z)
// -->
</script>
   <p>
      Se o valor que guardamos na variável z no início do script for
      necessário mais tarde, estaremos perante um erro porque a função
      trocarString() alterou o seu valor indevidamente. Se dentro de uma
```

```
      função usarmos a palavra var ao declaramos variáveis temos a certeza de
      que não alteramos os valores das variáveis globais. É isso que se faz na
      função trocarStringLocal().
   </p>
</body>
</html>
```



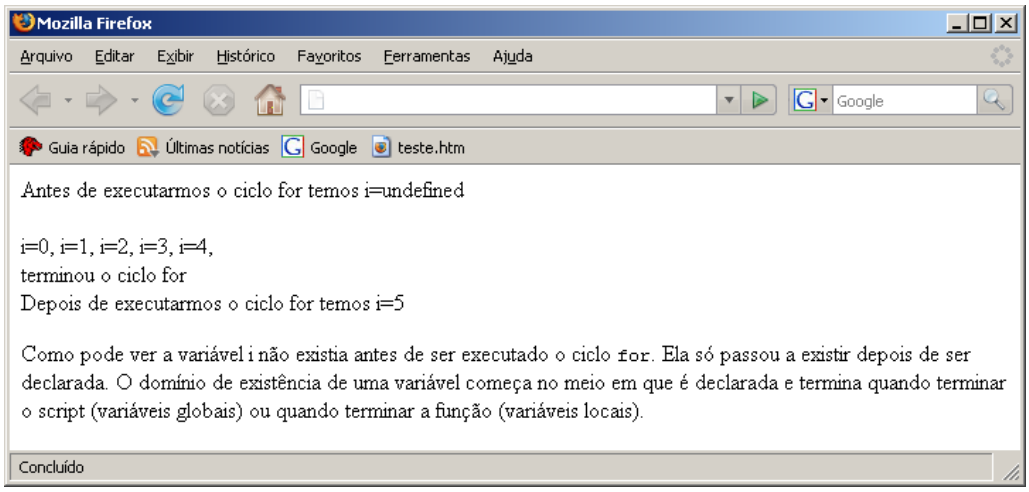
Variáveis locais

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function trocarString(c)
   {
      var l=c.length
      // repare que se usa a palavra var no início da próxima linha
      var z=c.substring(l/2,l)+c.substring(0,l/2)
      return z
   }
// -->
</script>
</head>
<body>
<script type="text/javascript">
<!--
   // As 7 linhas seguintes são necessárias para evitarmos um erro de script
   // Estude o capítulo 5 para saber mais sobre as instruções try ... catch
   var s
   try
   {
      s=z   //a variável z existe
   }
   catch(e)
   {
      s='indefinido'   //a variável z não existe
   }

   document.write("Antes de invocarmos a função trocarString temos z="+s)
   trocarString("Cereja")
   document.write("<br>")

   // As 6 linhas seguintes são necessárias para evitarmos um erro de script
   try
   {
      s=z   //a variável z existe
   }
```

```
   catch(e)
   {
      s='indefinido'//     a variável z não existe
   }

   document.write("Depois de invocarmos a função trocarString() temos z="+s)
// -->
</script>
   <p>
      Como pode ver a variável z só existe dentro da função em que foi
      declarada. Fora dessa função ela nunca chega a existir. Isto acontece
      porque ela é uma variável local
   </p>
</body>
</html>
```

Domínio de existência de uma variável

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   // As 7 linhas seguintes são necessárias para evitarmos um erro de script
   // Estude o capítulo 5 para saber mais sobre as instruções try ... catch
   var s
   try
   {
     s=i   // a variável i existe
   }
   catch(e)
   {
      s='indefinido'   //a variável i não existe
   }

   document.write("Antes de executarmos o ciclo for temos i="+s)
   document.write("<br><br>")

   for(var i=0;i<5;++i)
      document.write("i="+i+", ")

   document.write("<br>terminou o ciclo for<br>")
   s=i
   document.write("Depois de executarmos o ciclo for temos i="+s)
// -->
</script>
   <p>
      Como pode ver a variável i não existia antes de ser executado o
      ciclo <code>for</code>. Ela só passou a existir depois de ser declarada.
      O domínio de existência de uma variável começa no meio em que é
      declarada e termina quando terminar o script (variáveis globais) ou
      quando terminar a função (variáveis locais).
   </p>
</body>
</html>
```



Caracteres especiais

Em JavaScript, as strings (variáveis que contêm texto) definem-se colocando texto entre aspas ou entre apóstrofos (caractere '). Por isso dizemos que as aspas e os apóstrofos são caracteres delimitadores de uma string: a string começa na primeira ocorrência de um caractere delimitador e termina na segunda ocorrência mas sempre na mesma linha. O exemplo seguinte ilustra este ponto:

```
var s = "Isto é uma string"
var z = 'Isto é outra string'
```

Imaginemos agora que queremos colocar os próprios caracteres delimitadores como parte do conteúdo da string. Não podemos escrever o que se mostra a seguir porque está errado:

```
var s = "O meu nome é "Zezinho""   // isto está errado
var z = 'O meu nome é 'Luisão''    // isto está errado
```

Mas podemos escrever

```
var s = "O meu nome é \"Zezinho\""
var z = 'O meu nome é \'Luisão\''
```

Ao colocarmos o caractere \ (barra para trás) antes das aspas ou dos apóstrofos estamos dizendo ao interpretador de JavaScript que aquela aspa ou apóstrofo não é um delimitador mas sim um caractere normal que deve fazer parte do conteúdo da string.

Podemos obter o mesmo resultado do exemplo anterior escrevendo na forma seguinte (que é bem mais legível e deve ser preferida):

```
var s = 'O meu nome é "Zezinho"'
var z = "O meu nome é 'Luisão'"
```

Há caracteres que só podem ser obtidos usando a barra para trás. O mais importante deles é o caractere newline (nova linha), que provoca uma mudança de linha e não pode ser obtido de nenhuma outra forma. Sempre que quisermos forçar uma mudança de linha numa string temos de escrever \n (caractere newline), assim:

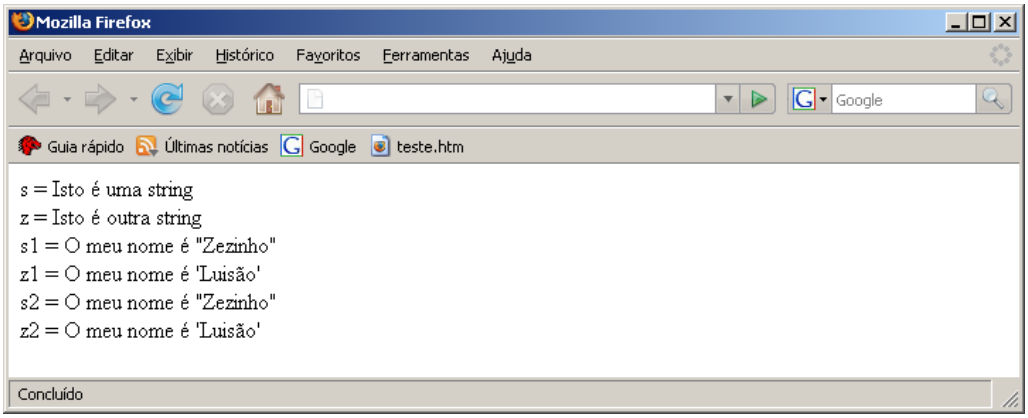
```
var s = 'O meu nome é "Zezinho".\nTodos têm respeitinho por mim!'
```

Também é conveniente (mas não absolutamente necessário) que ao escrevermos o caractere / num script o façamos na forma \/ (colocando uma barra para trás antes da barra para frente). Isto é recomendável porque os comentários com uma só linha começam com os caracteres //. Se usarmos a barra para trás temos a certeza de que o interpretador não irá confundir o caractere / com o início de um comentário.

Exemplos de Aplicação

Caracteres especiais: delimitadores de strings

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var s = "Isto é uma string"
   var z = 'Isto é outra string'

   var s1 = "O meu nome é \"Zezinho\""
   var z1 = 'O meu nome é \'Luisão\''

   var s2 = 'O meu nome é "Zezinho"'
   var z2 = "O meu nome é 'Luisão'"

   document.write('s = '+s)
   document.write('<br>')
   document.write('z = '+z)
   document.write('<br>')
   document.write('s1 = '+s1)
   document.write('<br>')
   document.write('z1 = '+z1)
   document.write('<br>')
   document.write('s2 = '+s2)
   document.write('<br>')
   document.write('z2 = '+z2)
// -->
</script>
</body>
</html>
```



Caracteres especiais: newline

```
<html>
<head>
<title></title>
<script type="text/javascript">
```

```
<!--
   // Cada ocorrência dos caracteres  \n  provoca uma mudança de linha
    var s = "Linha 1: Esta é uma string com várias linhas.\n Linha 2\n Linha
3\n\nLinha 5"
   alert(s)
// -->
</script>
</head>
<body>
</body>
</html>
```

Utilizar caracteres especiais do HTML em um script

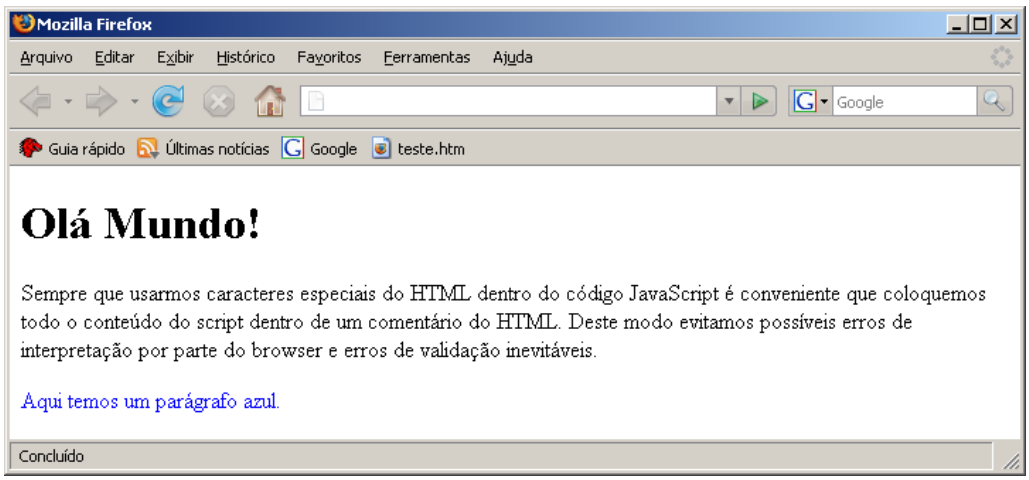
A linguagem HTML e XHTML trata alguns caracteres de forma especial. Entre eles temos: &, < e >. Estes caracteres são utilizados na linguagem JavaScript para formar operadores lógicos. Se os usarmos num script (o que é inevitável, porque eles são indispensáveis) as nossas páginas já não serão capazes de passar num teste de validação de HTML ou de XHTML e até podem ser interpretadas incorretamente por alguns browsers.

Colocar os scripts dentro de comentários do HTML

Para evitarmos que isso aconteça devemos fazer o seguinte: sempre que dentro de um script colocarmos um dos caracteres &, < ou > devemos colocar todo o código do script dentro de um comentário do HTML. Isto serve para evitar que o browser tente interpretar o conteúdo do script como se fosse HTML, o que seria errado. Por outro lado, um validador de HTML ou de XHTML que vá analisar a página fica sabendo que o script não deve ser validado porque está escrito numa linguagem diferente. O exemplo seguinte ilustra este ponto:

```
<html>
<head><title></title></head>
<body>
<script type="text/javascript">
<!--
   document.write("<h1>Olá Mundo!</h1>")
-->
</script>
   <p>
      Sempre que usarmos caracteres especiais do HTML dentro do código
      JavaScript é conveniente que coloquemos todo o conteúdo do
      script dentro de um comentário do HTML. Deste modo evitamos
      possíveis erros de interpretação por parte do browser e erros de
      validação inevitáveis.
   </p>
<script type="text/javascript">
<!--
   var s='<p style="color: blue">Aqui temos um parágrafo azul.</p>'
   document.write(s)
-->
</script>
```

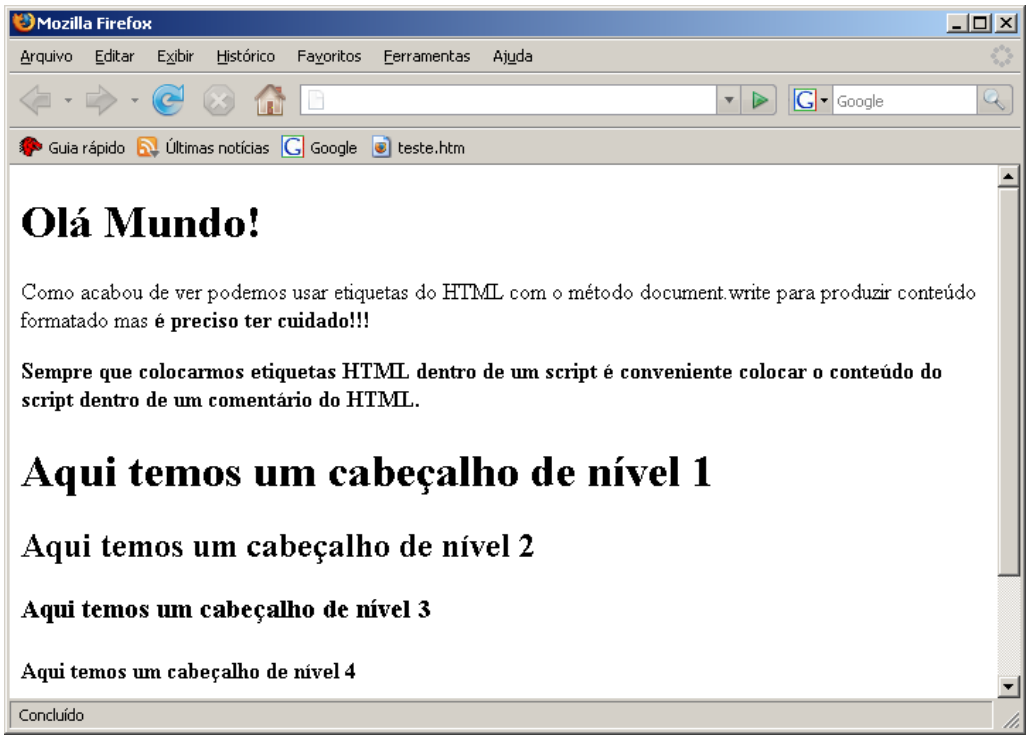
```
</body>
</html>
```



Exemplos de Aplicação

Formatar texto com etiquetas HTML

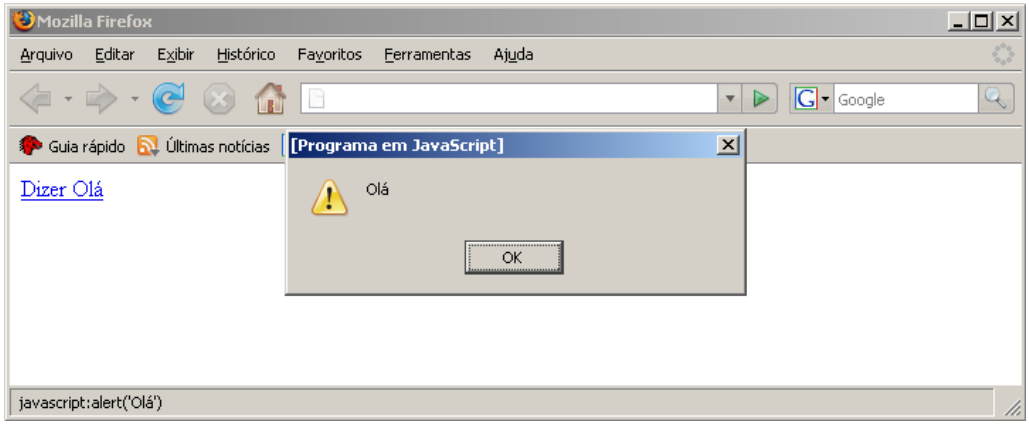
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   document.write("<h1>Olá Mundo!<\/h1>")
// -->
</script>
   <p>
      Como acabou de ver podemos usar etiquetas do HTML
      com o método document.write para produzir conteúdo
      formatado mas <b>é preciso ter cuidado!!!</p>
   </p>
   <p>
      <b>Sempre que colocarmos etiquetas HTML dentro de um
      script é conveniente colocar o conteúdo do script dentro de
      um comentário do HTML.</b>
   </p>
<script type="text/javascript">
<!--
  for(var i=1;i<=6;++i)
     document.write('<h'+i+'>Aqui temos um cabeçalho de nível '+i+'</h'+i+'>')
// -->
</script>
</body>
</html>
```



Chamar uma função a partir de uma ligação de hipertexto

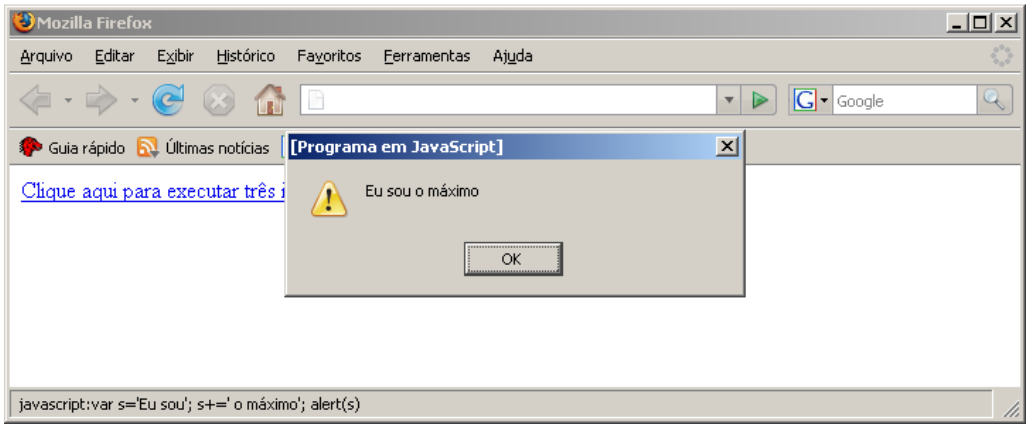
Por diversas vezes, em um desenvolvimento, será útil ter ligações de hipertexto que ao invés de transportarem imediatamente o usuário para outra página executem uma instrução ou uma função escritas em JavaScript. A forma mais simples de conseguirmos isso consiste em escrever o nome da função no atributo href do elemento <a> precedido pela string javascript:, assim:

```
<html>
<body>
   <a href="javascript:alert('Olá')">Dizer Olá</a>
</body>
</html>
```



Quando encontra uma ligação cujo atributo href começa por javascript: o browser passa a interpretar o resto do valor do atributo href como um conjunto de instruções escritas em JavaScript e não carrega uma nova página. Este método permite executar uma ou mais instruções que devem obrigatoriamente ser separadas por ponto e vírgula, como se mostra a seguir:

```
<html>
<body>
   <a href="javascript:var s='Eu sou'; s+=' o máximo'; alert(s)">
   Clique aqui para executar três instruções</a>
</body>
</html>
```



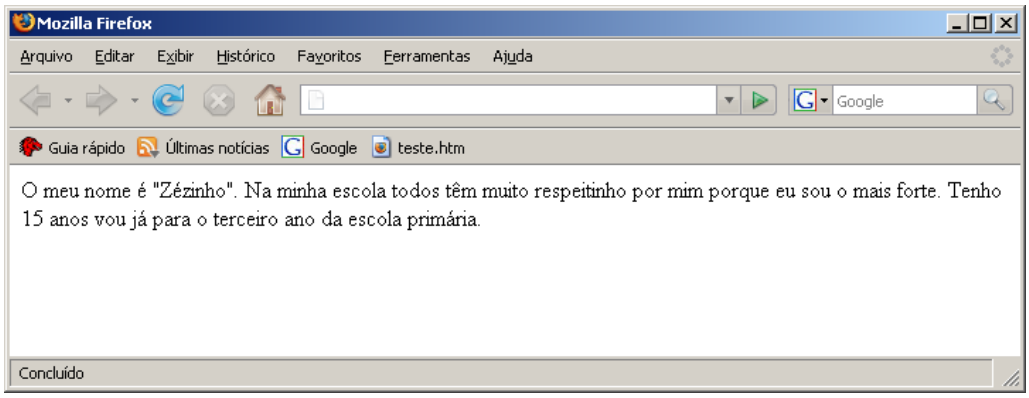
Esta forma de executar JavaScript a partir de uma ligação tem algumas aplicações úteis, mas existe uma outra forma mais poderosa que se baseia no evento onclick. Essa forma permite continuar a usar o atributo href da maneira habitual e executar JavaScript.

Continuar na linha seguinte uma linha extensa

O JavaScript não impõe limites ao comprimento das linhas de código. Elas podem se entender pelo comprimento que for necessário.

No entanto, quando escrevemos strings (variáveis de texto) extensas, a utilização de linhas muito longas dificulta muito a escrita e a legibilidade do código. Nestes casos é preciso limitar o comprimento das linhas de código. Por isso o JavaScript permite-nos interromper a escrita de strings e continuar a escrever na linha seguinte, assim:

```
<html>
<body>
<script type="text/javascript">
<!--
   var s = 'O meu nome é "Zézinho". Na minha escola todos têm muito \
   respeitinho por mim porque eu sou o mais forte. \
   Tenho 15 anos vou já para o terceiro ano da escola primária.'

   document.write(s)
-->
</script>
</body>
</html>
```



Ao terminarmos uma linha com o caractere \ (barra para trás) estamos dizendo ao interpretador de JavaScript que a string continua na linha seguinte.

Isto se aplica apenas a strings e não ao código normal.

2. Os operadores da linguagem JavaScript

Os operadores servem para realizar operações com valores, dando como resultado novos valores.

2.1 Operadores aritméticos

Operador	Descrição	Exemplo	Resultado
+	Adição	2+2	4
-	Subtração	5-2	3
*	Multiplicação	4*5	20
/	Divisão	15/5 5/2	3 2.5
%	Resto da divisão	5%2 10%8 10%2	1 2 0
++	Incrementar (aumentar uma unidade)	x=5 x++	 x=6
--	Decrementar (diminuir uma unidade)	x=5 x--	 x=4

2.2 Operadores de atribuição (formas abreviadas)

Operador	Exemplo	É O Mesmo Que
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

2.3 Operadores de comparação

Operador	Descrição	Exemplo	Resultado
==	é igual a	5==8	false
!=	não é igual a	5!=8	true
>	é maior do que	5>8	false

	é menor do que	5<8	true
>=	é maior ou igual a	5>=8	false
<=	é menor ou igual a	5<=8	true

2.4 Operadores lógicos

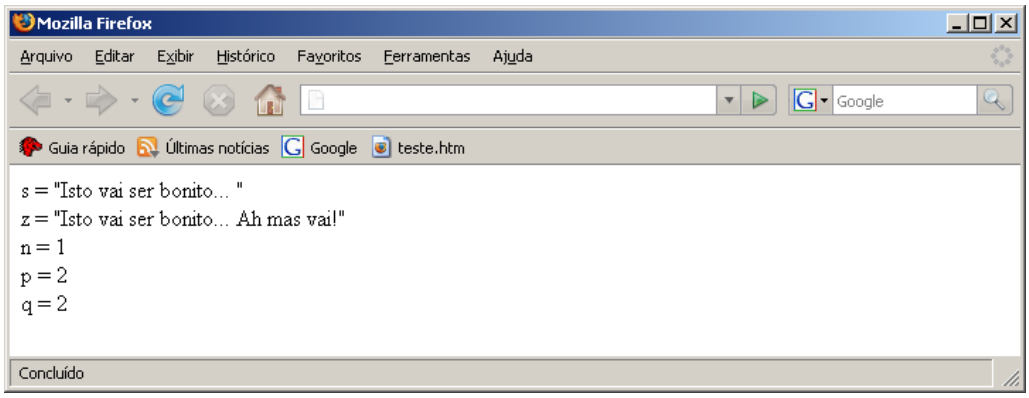
Operador	Descrição	Exemplo
&&	e (and)	x=6 y=3 (x < 10 && y > 1) dá true
||	ou (or)	x=6 y=3 (x==4 || y==4) dá false
!	negação (not)	x=6 y=3 !(x==y) dá true

Neste estudo não abordamos os operadores que operam bit a bit porque eles não são fáceis de compreender e não são importantes para os objetivos que temos em vista. O seu estudo poderá ser feito em um curso de JavaScript avançado.

Exemplos de Aplicação

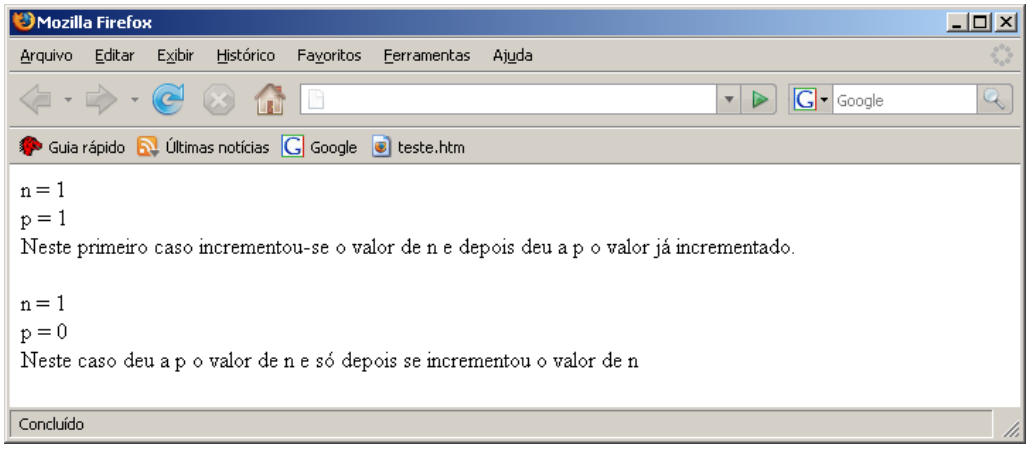
Atribuição simples de valores a variáveis

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   //Exemplos de atribuição simples de valores a variáveis
   var s = "Isto vai ser bonito... " // uma string
   var z = s + "Ah mas vai!"
   var n = 1   // um número inteiro
   var p = q = 2   // dois números inteiros

   document.write("s = \""+s+"\"")
   document.write("<br>")
   document.write("z = \""+z+"\"")
   document.write("<br>")
   document.write("n = "+n)
   document.write("<br>")
   document.write("p = "+p)
   document.write("<br>")
   document.write("q = "+q)
// -->
</script>
</body>
</html>
```



Incrementar valores (aumentar uma unidade)

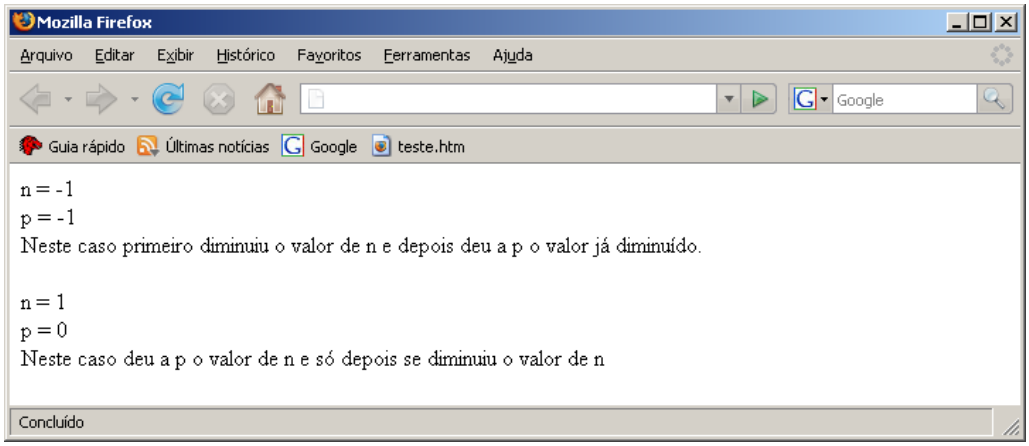
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var p,n = 0
   p = ++n
   document.write("n = " + n)
   document.write("<br>")
   document.write("p = " + p)
   document.write("<br>Neste primeiro caso incrementou-se o valor de n ")
   document.write(" e depois deu a p o valor já incrementado.<br><br>")
   n=0
   p = n++
   document.write("n = " + n)
   document.write("<br>")
   document.write("p = " + p)
   document.write("<br>Neste caso deu a p o valor de n ")
   document.write(" e só depois se incrementou o valor de n")
// -->
</script>
</body>
</html>
```



Decrementar valores (diminuir uma unidade)

```
<html>
<head>
<title></title>
</head>
<body>
```

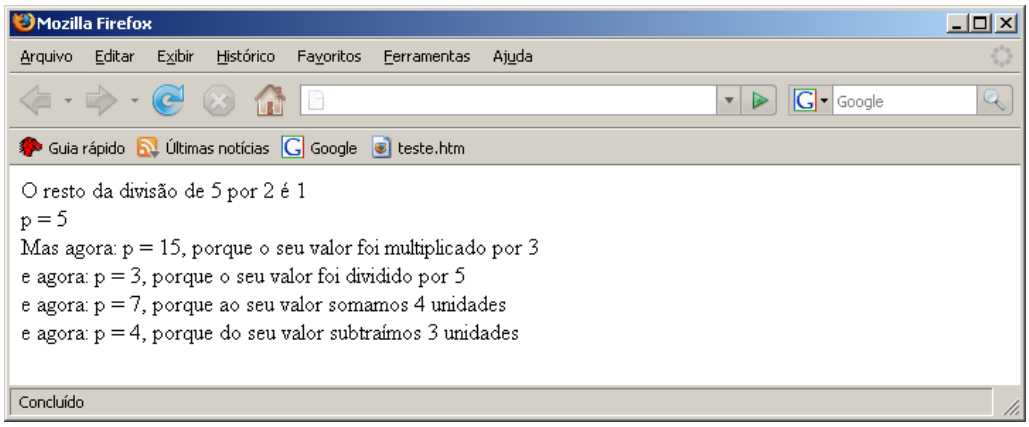
```
<script type="text/javascript">
<!--
   var p,n = 0
   p = --n
   document.write("n = " + n)
   document.write("<br>")
   document.write("p = " + p)
   document.write("<br>Neste caso primeiro diminuiu o valor de n ")
   document.write(" e depois deu a p o valor já diminuído.<br><br>")
   n=0
   p = n++
   document.write("n = " + n)
   document.write("<br>")
   document.write("p = " + p)
   document.write("<br>Neste caso deu a p o valor de n ")
   document.write(" e só depois se diminuiu o valor de n")
// -->
</script>
</body>
</html>
```



Operações aritméticas

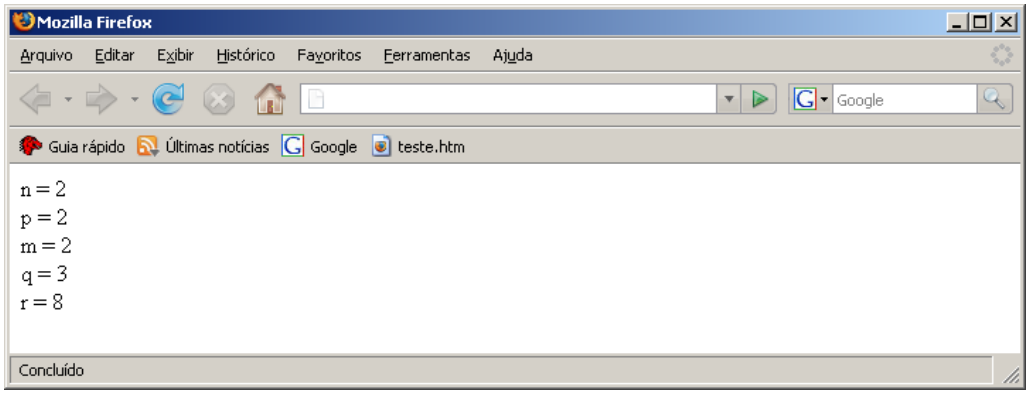
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var p = 5, n = 2
   var r = p % n
   document.write("O resto da divisão de 5 por 2 é " + r)
   document.write("<br>")
   document.write("p = " + p)
   p *= 3   // equivale a  p = p*3
     document.write("<br>Mas agora: p = " + p + ", porque o seu valor foi
multiplicado por 3<br>")
   p /= 5   // equivale a  p = p/5
    document.write("e agora: p = " + p + ", porque o seu valor foi dividido por
5<br>")
   p += 4   // equivale a  p = p+4
     document.write("e agora: p = " + p + ", porque ao seu valor somamos 4
unidades<br>")
   p -= 3   // equivale a  p = p-3
    document.write("e agora: p = " + p + ", porque do seu valor subtraímos 3
unidades")
// -->
</script>
</body>
```

```
</html>
```



Atribuição mais elaborada de valores a variáveis

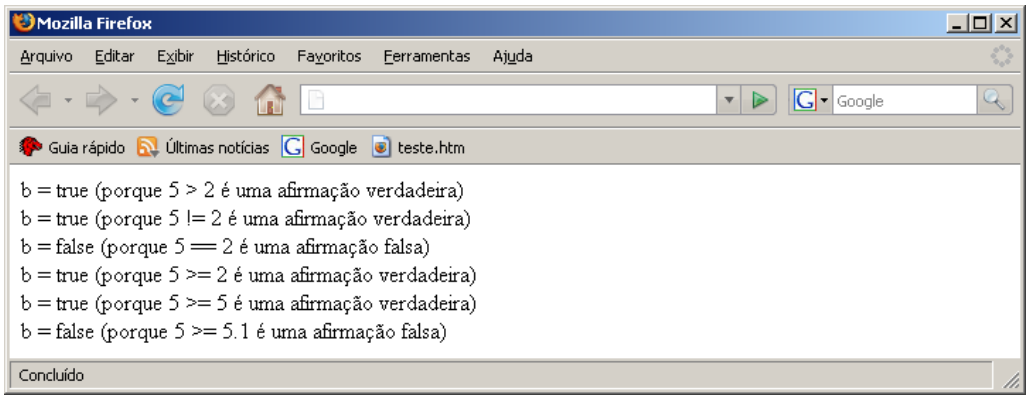
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   // Exemplos mais elaborados de atribuição de valores a variáveis
   var p=2,q=3,n=1   // três números inteiros declarados de uma só vez
   var m = ++n   // número inteiro com incremento antes da atribuição
   var r = (q * p) + m   // valor obtido após a realização de várias operações

   document.write("n = "+n)
   document.write("<br>")
   document.write("p = "+p)
   document.write("<br>")
   document.write("m = "+m)
   document.write("<br>")
   document.writeln("q = "+q)
   document.write("<br>")
   document.write("r = "+r)
// -->
</script>
</body>
</html>
```



Comparações

```
<html>
<head>
<title></title>
```

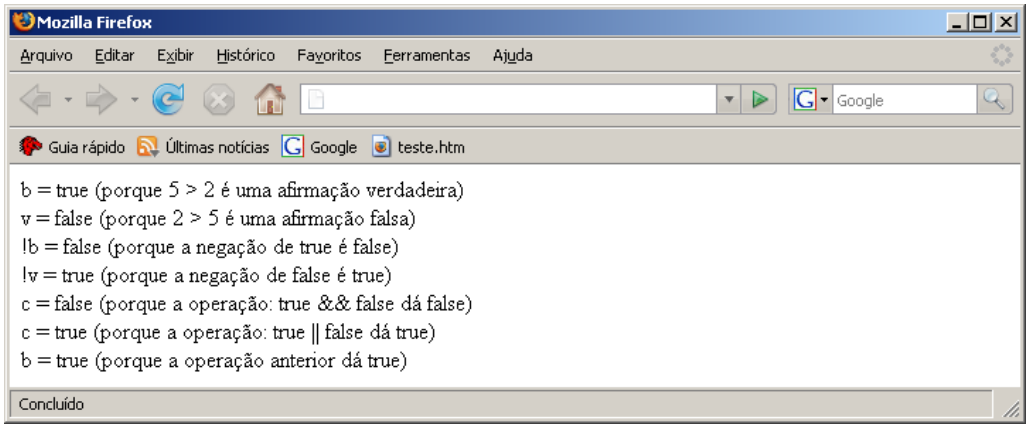
```
</head>
<body>
<script type="text/javascript">
<!--
   var b = (5 > 2)
   document.write("b = " + b + " (porque 5 > 2 é uma afirmação verdadeira)")
   document.write("<br>")
   b = (5 != 2)
   document.write("b = " + b + " (porque 5 != 2 é uma afirmação verdadeira)")
   document.write("<br>")
   b = (5 == 2)
   document.write("b = " + b + " (porque 5 == 2 é uma afirmação falsa)")
   document.write("<br>")
   b = (5 >= 2)
   document.write("b = " + b + " (porque 5 >= 2 é uma afirmação verdadeira)")
   document.write("<br>")
   b = (5 >= 5)
   document.write("b = " + b + " (porque 5 >= 5 é uma afirmação verdadeira)")
   document.write("<br>")
   b = (5 >= 5.1)
   document.write("b = " + b + " (porque 5 >= 5.1 é uma afirmação falsa)")
// -->
</script>
</body>
</html>
```



Operações lógicas

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var b = (5 > 2)
   var v = (2 > 5)
   document.write("b = " + b + " (porque 5 > 2 é uma afirmação verdadeira)")
   document.write("<br>")
   document.write("v = " + v + " (porque 2 > 5 é uma afirmação falsa)")
   document.write("<br>")
   document.write("!b = " + !b + " (porque a negação de true é false)")
   document.write("<br>")
   document.write("!v = " + !v + " (porque a negação de false é  true)")
   document.write("<br>")
   var c = b && v
   document.write("c = " + c + " (porque a operação: true && false dá false)")
   document.write("<br>")
   c = b || v
   document.write("c = " + c + " (porque a operação: true || false dá true)")
   document.write("<br>")
   b = (3 < 10 && 2 > 1) || !c
```

```
   document.write("b = " + b + " (porque a operação anterior dá true)")
// -->
</script>
</body>
</html>
```



2.5 Adição de texto

O JavaScript permite produzir facilmente uma nova variável de texto (String) cujo valor é igual à justaposição dos valores de outras variáveis. Isso se consegue usando o operador + (adição), assim:

```
txt1="Ai minha mãezinha... "
txt2="Isto vai ser um massacre!"
txt3=" Estamos fritos!"
txt4=txt1+txt2+txt3
```

O resultado é igual a:

```
txt4="Ai minha mãezinha... Isto vai ser um massacre! Estamos fritos!"
```

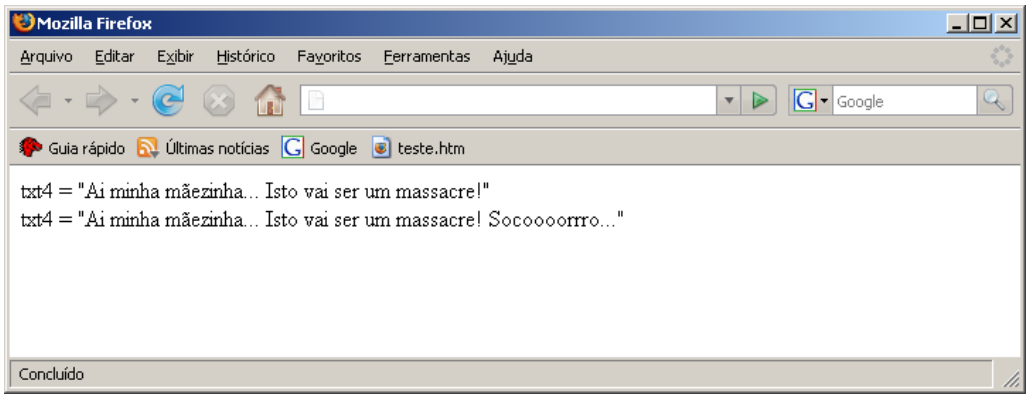
Se adicionar uma variável de texto a um valor que não seja texto, o sistema JavaScript faz a adição na mesma. Para isso ele converte para forma textual o valor que não é texto e faz a adição. O resultado é uma string (texto.) Nos exercícios listados mais abaixo pode ver melhor a forma como isso é feito.

Exemplos de Aplicação

Somar texto com texto

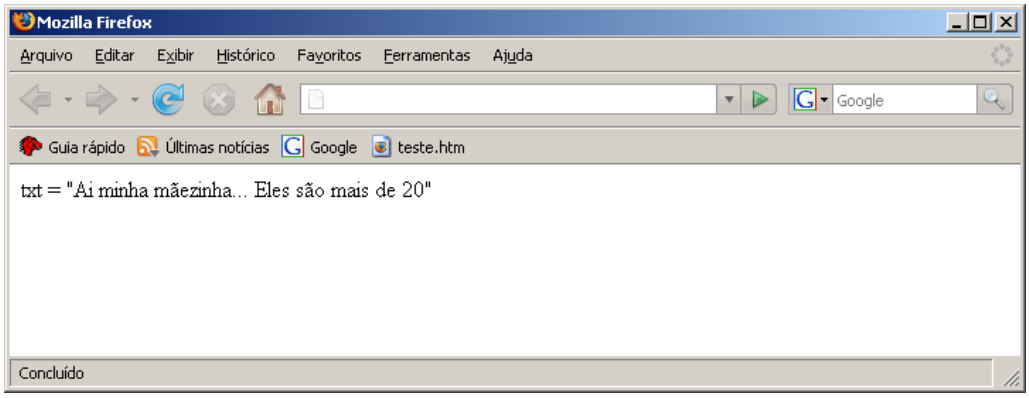
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var txt1="Ai minha mãezinha... "
   var txt2="Isto vai ser um massacre!"
   var txt4=txt1+txt2
   document.write('txt4 = "' + txt4 +'"')
   document.write('<br>')
   txt4 += " Socoooorrro..."
   document.write('txt4 = "' + txt4 +'"')
// -->
</script>
</body>
```

```
</html>
```



Somar texto com outros valores

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var txt="Ai minha mãezinha... "
   var n= 20
   txt += "Eles são mais de " + n
   /* quando somamos um número a uma string o sistema
   javascript constrói automaticamente uma versão de
   texto do número e soma esse texto à string */
   document.write('txt = "' + txt +'"')
// -->
</script>
</body>
</html>
```



3. Instruções condicionais

As instruções condicionais testam uma condição e com base no resultado do teste decidem se uma parte do código deve ou não ser executada. Elas nos permitem executar código diferente em situações diferentes.

3.1 As instruções if e if ... else

A instrução if

A instrução if se usa para testar uma condição e executar um bloco de código apenas quando ela é satisfeita. A sua sintaxe é a seguinte:

```
if (condição)
{
   código a executar se a condição for verdadeira
}
```

A seguir temos exemplos com esta instrução:

```
var i = 10
var s
if (i < 10)
   s = "O número i é menor do que 10"
// o código a ser executado só ocupa uma linha não sendo
// preciso colocá-lo entre chaves

if (i >= 10)
{
   s = "O número i é maior ou igual a 10"
   i = 0
}
// o código a ser executado só ocupa uma linha não sendo
// preciso colocá-lo entre chaves
```

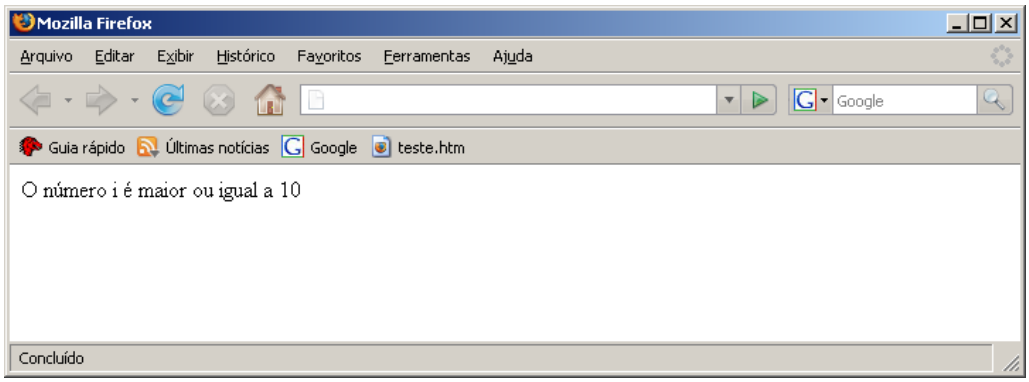
A instrução if ... else

A instrução if ... else usa-se para testar uma condição. Se a condição for satisfeita será executado um bloco de código e se não for satisfeita será executado um outro bloco alternativo. A sua sintaxe é a seguinte:

```
if (condição)
{
   código a executar se a condição for verdadeira
}
else
{
   código a executar se a condição for falsa
}
```

A seguir temos um exemplo com esta instrução:

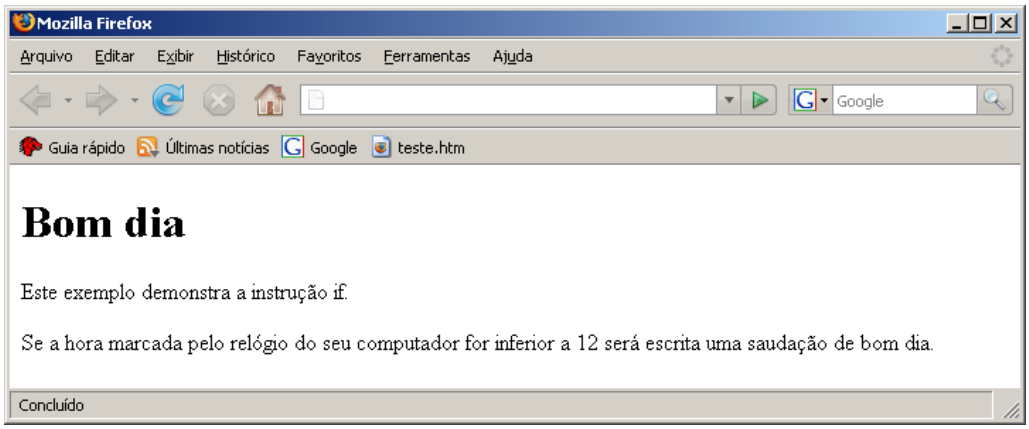
```
<html>
<body>
<script type="text/javascript">
<!--
   var i = 10
   var s
   if (i < 10)
      s = "O número i é menor do que 10"
   else
   {
      s = "O número i é maior ou igual a 10"
      i = 0
   }
   document.write(s)
-->
</script>
</body>
</html>
```



Exemplos de Aplicação

A instrução if

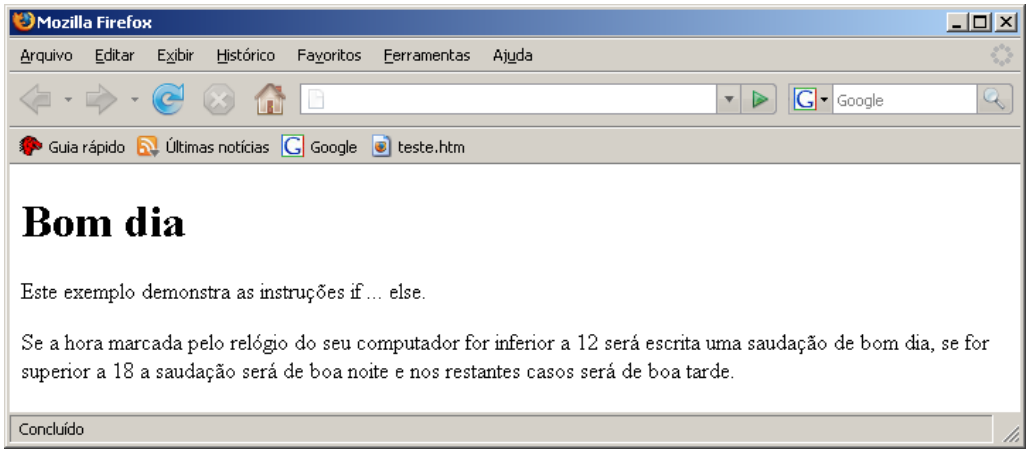
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var d = new Date()
   var time = d.getHours()
   if (time < 12)
      document.write("<h1>Bom dia</h1>")
// -->
</script>
   <p>
      Este exemplo demonstra a instrução if.
   </p>
   <p>
      Se a hora marcada pelo relógio do seu computador
      for inferior a 12 será escrita uma saudação de
      bom dia.
   </p>
</body>
</html>
```



Instrução if...else

```
<html>
<head>
<title></title>
</head>
<body>
```

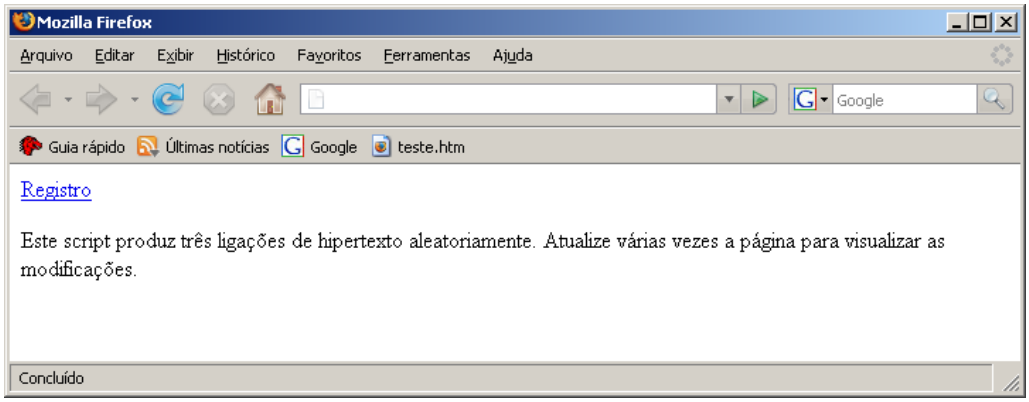
```
<script type="text/javascript">
<!--
   var d = new Date()
   var hora = d.getHours()
   if (hora < 12)
      document.write("<h1>Bom dia</h1>")
   else
   {
      if(hora > 18)
         document.write("<h1>Boa noite</h1>")
      else
         document.write("<h1>Boa tarde</h1>")
   }
// -->
</script>
   <p>
      Este exemplo demonstra as instruções if ... else.
   </p>
   <p>
      Se a hora marcada pelo relógio do seu computador for
      inferior a 12 será escrita uma saudação de bom dia,
      se for superior a 18 a saudação será de boa noite
      e nos restantes casos será de boa tarde.
   </p>
</body>
</html>
```



Produzir uma ligação aleatória

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var r=Math.random()
   if (r>0.666)
      document.write('<a href="http://www.w3.org" target="_blank">W3C</a>')
   else
   {
      if(r<0.333)
         document.write('<a href=http://www.google.com.br target="_blank">
         Google</a>')
      else
         document.write('<a href=http://www.registro.br
         target="_blank">Registro</a>')
   }
// -->
</script>
```

```
   <p>
      Este script produz três ligações de hipertexto aleatoriamente.
      Atualize várias vezes a página para visualizar as modificações.
   </p>
</body>
</html>
```



3.2 Atribuição condicional de valores

A linguagem JavaScript possui um operador que nos permite escolher o valor a atribuir a uma variável consoante o resultado do teste que é feito a uma condição.

Esse operador é um operador condicional de atribuição de valor e é composto por diversas partes: uma condição, um ponto de interrogação e dois valores que podem ser atribuídos à variável separados pelo caractere : (dois pontos). Ele tem a vantagem de permitir escrever código compacto que é mais fácil de ler pelos programadores experientes. A sua sintaxe é a seguinte:

```
variável = (condição) ? valor1 : valor2
```

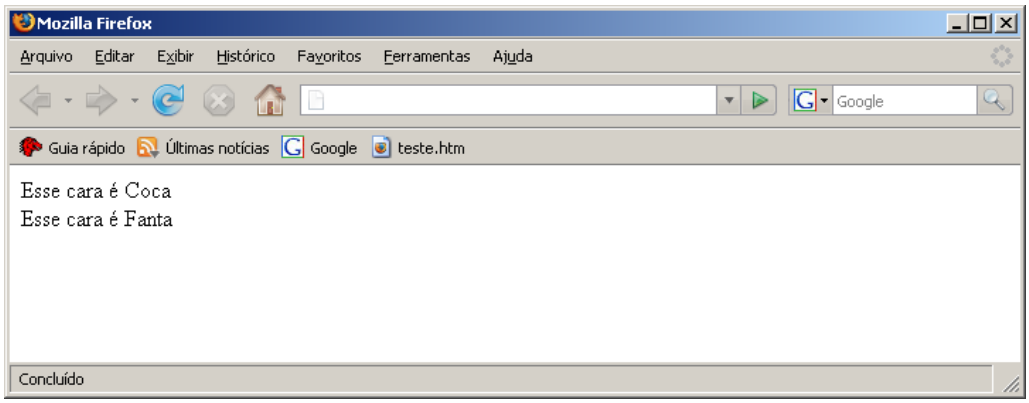
Este operador atua do seguinte modo: se a condição for verdadeira a variável passará a ter o valor1; se a condição não for satisfeita será atribuído o valor2 à variável.

Exemplos de Aplicação

Atribuição condicional de valores

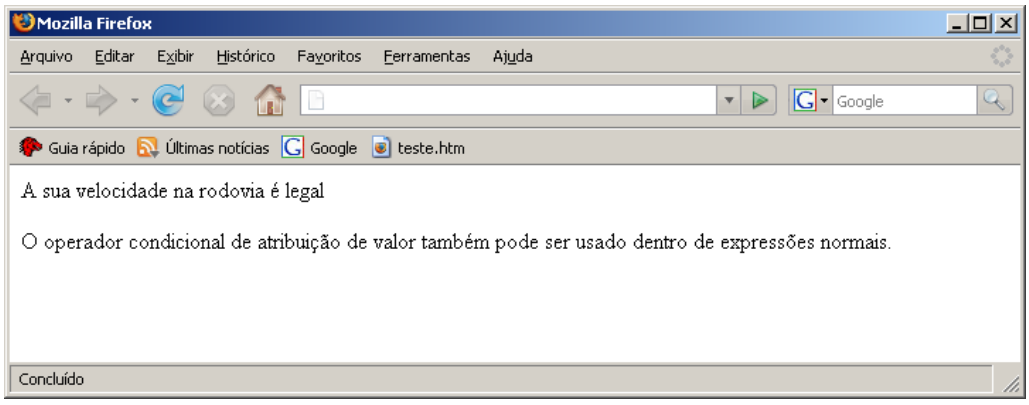
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var sexo = "masculino"
   var tipo = (sexo=="masculino") ? "Coca" : "Fanta"

   document.write("Esse cara é " + tipo)
   document.write("<br>")

   var sexo = "feminino"
   var tipo = (sexo=="masculino") ? "Coca" : "Fanta"

   document.write("Esse cara é " + tipo)
// -->
</script>
</body>
```

```
</html>
```



Atribuição condicional de valores (versão 2)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var velocidade = 100
   var s = (velocidade > 100) ? "excessiva" : "legal"

   document.write("A sua velocidade na rodovia é " + s)
// -->
</script>
   <p>
      O operador condicional de atribuição de valor também pode ser usado
      dentro de expressões normais.
   </p>
</body>
</html>
```



3.3 A instrução switch

Nota: Apesar de esta instrução não fazer parte do padrão ECMAScript, ela é suportada por todos os browsers importantes.

A instrução switch usa-se para comparar o valor do seu argumento (uma variável ou uma expressão) com vários valores. Para cada caso em que houver uma igualdade será executada uma determinada porção de código. A sintaxe desta instrução é a seguinte:

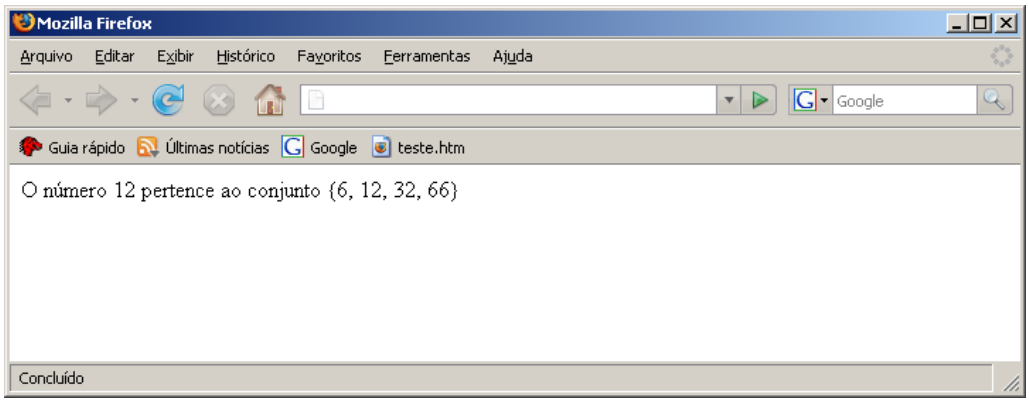
```
switch (expressão)
```

```
{
   case label1:
      código a executar se expressão = label1
      break
   case label2:
      código a executar se expressão = label2
      break
   default:
      código a executar se a expressão não for igual
      a nenhuma das alternativas apresentadas antes
}
```

Esta instrução funciona do seguinte modo: Primeiro calcula-se o valor da expressão que é argumento da instrução switch. A seguir compara-se o resultado da expressão com um conjunto de alternativas que são fornecidas a seguir à palavra "case" e terminadas pelo símbolo : (dois pontos). Sempre que a comparação detectar uma igualdade será executada a porção de código que está associada a esse caso. A execução do código prossegue pelas linhas seguintes até ser encontrada a instrução break ou até que termine o bloco switch.

O exemplo seguinte mostra como se usa esta instrução.

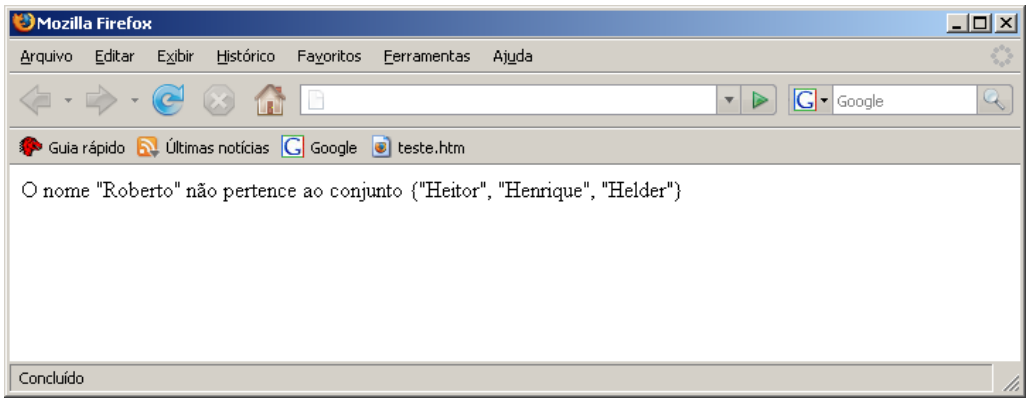
```
<html>
<body>
<script type="text/javascript">
<!--
   // este exemplo usa a instrução switch para dizer se um número i
   // pertence ou não ao conjunto {6, 12, 32, 66}
   var i = 12
   var s = "O número " + i

   switch(i)
   {
      case 6:
         s += " pertence "
         break
      case 12:
         s += " pertence "
         break
      case 32:
         s += " pertence "
         break
      case 66:
         s += " pertence "
         break
      default:
         s += " não pertence "
   }

   s += "ao conjunto {6, 12, 32, 66}"
   document.write(s)
-->
</script>
</body>
</html>
```



Uma vez detectado um acerto, inicia-se a execução da porção de código que lhe corresponde. Se no fim dessa porção não colocarmos uma instrução break, todas as instruções do bloco switch que estão mais abaixo serão executadas até que o bloco switch termine ou até que seja encontrada uma instrução break.

Este comportamento pode ser explorado para escrever código mais compacto. O exemplo seguinte mostra como isso se faz:

```
<html>
<body>
<script type="text/javascript">
<!--
   // este exemplo usa a instrução switch para dizer se um nome
   // pertence ou não ao conjunto {"Heitor", "Henrique", "Helder"}

   var nome = "Roberto"
   var s = 'O nome "' + nome

   switch(nome)
   {
      case "Heitor":
      case "Henrique":
      case "Helder":
         s += '" pertence '
         break
      default:
         s += '" não pertence '
   }
   s += 'ao conjunto {"Heitor", "Henrique", "Helder"}'
   document.write(s)
-->
</script>
</body>
</html>
```

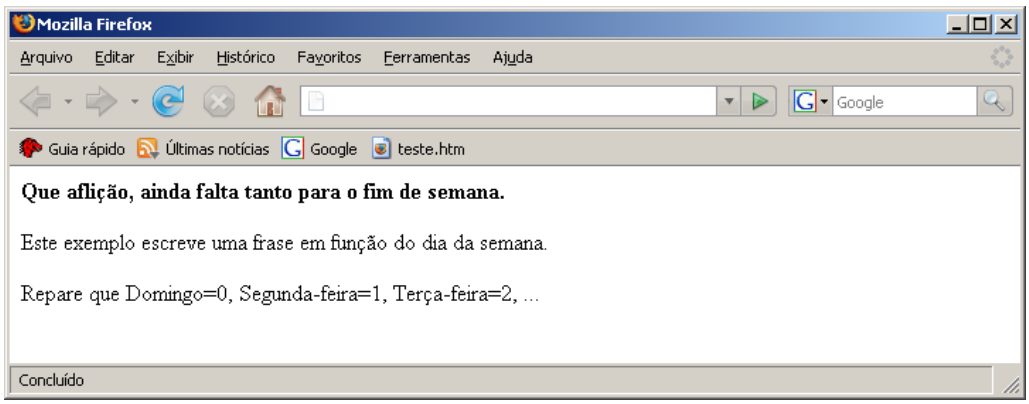


Como pode ver, para acrescentarmos um elemento ao conjunto de nomes basta acrescentar uma linha com a palavra "case".

Exemplos de Aplicação

A instrução switch

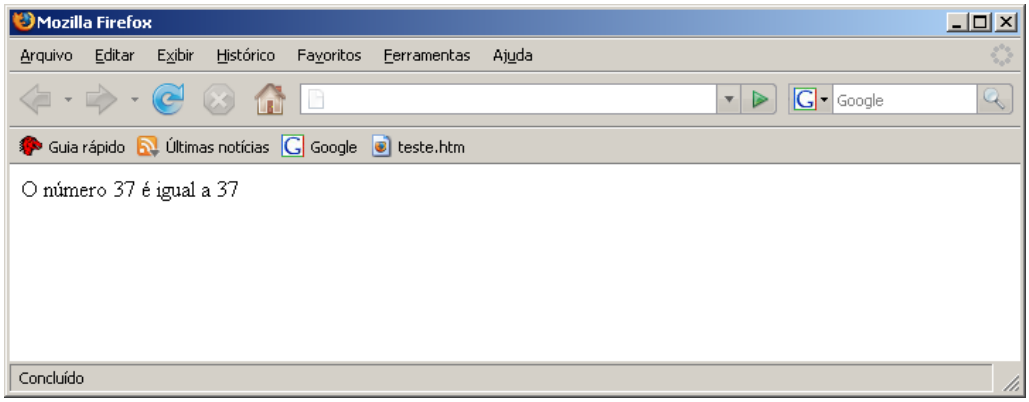
```
<html>
<head>
<title></title>
</head>
<body>
<b><script type="text/javascript">
<!--
   var d = new Date()
   var dia = d.getDay()
   switch (dia)
   {
     case 5:
       document.write("Finalmente é Sexta-feira!")
       break
     case 6:
       document.write("Hoje é Sábado. Ihuuuu!")
       break
     case 0:
       document.write("Hoje é domingo. Já falta pouco para Segunda-feira :-(")
       break
     default:
       document.write("Que aflição, ainda falta tanto para o fim de semana.")
   }
// -->
</script></b>
   <p>
      Este exemplo escreve uma frase em função do dia da semana.
   </p>
   <p>
      Repare que Domingo=0, Segunda-feira=1, Terça-feira=2, ...
   </p>
</body>
</html>
```



A instrução switch (versão 2)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   // este exemplo usa a instrução switch para dizer se um número i
```

```
   // pertence ao conjunto {6, 12, 66}, ao conjunto {16, 22, 76}, se
   // é igual a 37, ou se não verifica nenhuma destas condições

   var i = 37
   var s = "O número " + i

   switch(i)
   {
      case 6:
      case 12:
      case 66:
         s += " pertence ao conjunto {6, 12, 66}"
         break
      case 16:
      case 22:
      case 76:
         s += " pertence ao conjunto {16, 22, 76}"
         break
      case 37:
         s +=" é igual a 37"
         break
      default:
         s += " não verifica nenhuma das condições"
   }
   document.write(s)
// -->
</script>
</body>
</html>
```



4. Execução repetida de código

Muitas vezes ao escrevermos código em JavaScript precisamos que um bloco de código ou uma instrução sejam executados repetidamente. Os ciclos de repetição nos fornecem meios para conseguirmos isso.

Em JavaScript temos ao nosso dispor as seguintes instruções para produzir ciclos de repetição:

- for - executam um bloco de código enquanto uma condição for satisfeita.
- while - repetem a execução de um bloco de código enquanto uma condição for satisfeita.
- do...while - repetem a execução de um bloco de código enquanto uma condição for satisfeita mas executam-no pelo menos uma vez, mesmo que a condição nunca seja satisfeita.

4.1 Ciclos for

Os ciclos for são implementados através da instrução for. Esta é uma instrução complexa que aceita vários argumentos separados pelo caractere ; (ponto e vírgula). A sua sintaxe é a seguinte:

```
for (inicialização; condição; atualização)
{
   bloco de código a executar
}
```

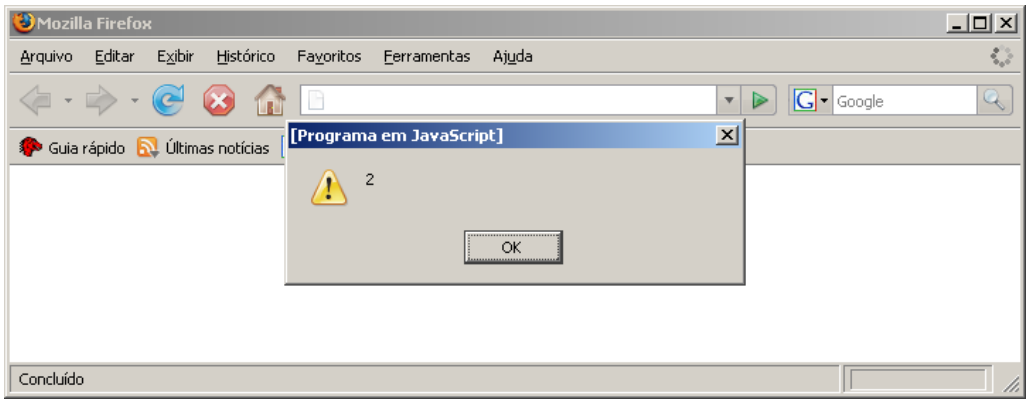
O primeiro argumento (inicialização) é composto por uma ou mais instruções (separadas por vírgulas). Essas instruções são executadas antes de se iniciar o ciclo.

O segundo argumento (condição) é composto por uma ou mais condições (separadas por vírgulas) que são testadas antes de se executar o bloco de código associado ao ciclo. Se uma dessas condições não for verdadeira o ciclo termina.

O terceiro argumento (atualização) é composto por uma ou mais instruções (separadas por vírgulas) que são executadas sempre que se completa uma execução do bloco de código associado ao ciclo. Normalmente essas instruções usam-se para incrementar uma variável que funciona como contador, mas podem ser usadas para outros fins.

O uso mais comum que é dado aos ciclos for é a execução de um bloco de código um número determinado de vezes. É precisamente isso que se ilustra a seguir:

```
<html>
<body>
<script type="text/javascript">
<!--
   for (var i = 1; i <= 3; ++i)
      alert(i)
-->
</script>
</body>
</html>
```



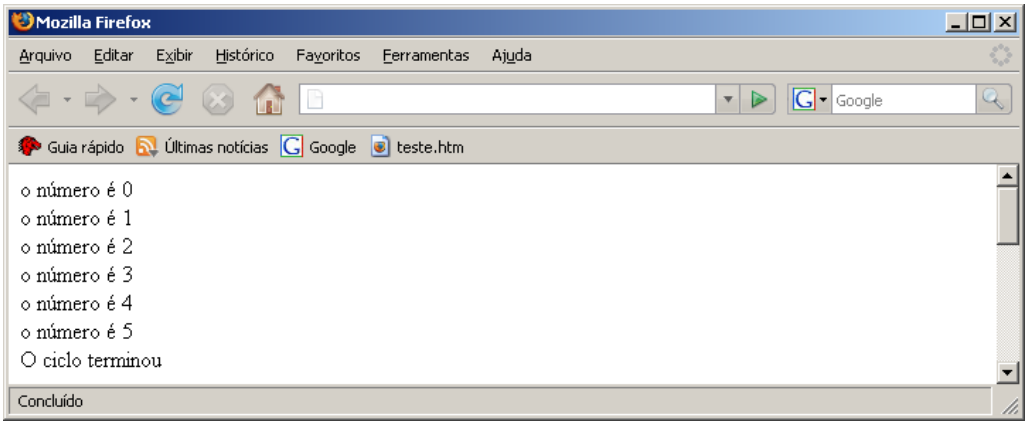
Neste exemplo a inicialização é var i = 1, a condição é i <= 3 e a atualização é ++i. O ciclo funciona do seguinte modo:

1. Declara-se uma variável i e atribui o valor 1 (um).
2. Verifica-se a condição i <= 3. Se for verdadeira salta-se para o passo seguinte, mas se for falsa pára-se a execução do ciclo.
3. Executa-se o bloco de código associado ao ciclo for (mostrar uma caixa de alerta).
4. Executa a parte ++i, que aumenta em uma unidade o valor da variável i.
5. Salta para o passo 2 (teste da condição).

Exemplos de Aplicação

Ciclos (de repetição) for

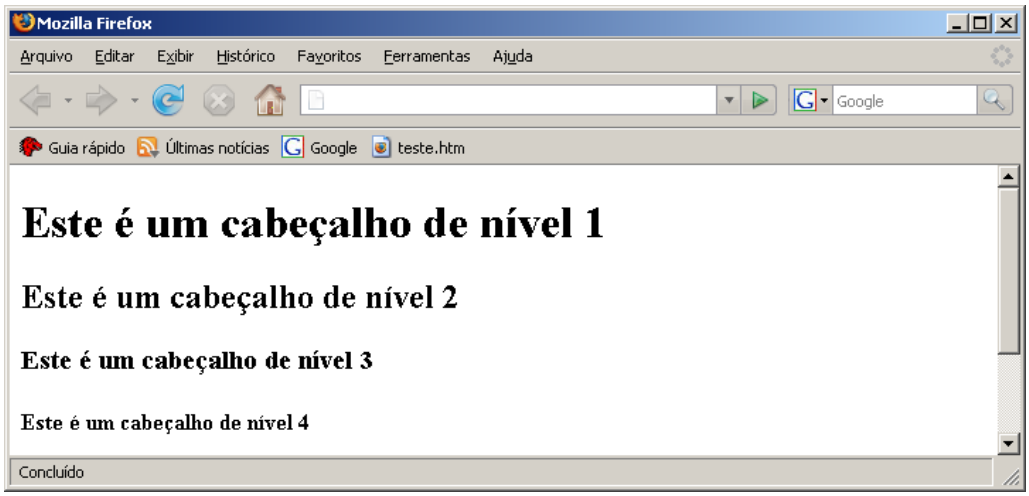
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   for (var i = 0; i <= 5; i++)
   {
      document.write("o número é " + i)
      document.write("<br>")
   }
   document.write("O ciclo terminou<br>")
// -->
</script>
   <h3>Explicação</h3>
   <p>
      1) O ciclo <code>for</code> começa dando
      à variável <code>i</code> o valor 0.
   </p>
   <p>
      2) O bloco de código que está entre chaves
      ({ ... }) logo a seguir à instrução <code>for</code>
      é executado uma vez.
   </p>
   <p>
      3) Depois de terminada a execução do bloco de código
      é executada a terceira instrução do ciclo <code>for</code>,
      que é <code>i++</code>. Esta terceira instrução aumenta em
      uma unidade o valor da variável <code>i</code>.
   </p>
   <p>
      4) A seguir executa-se a segunda instrução do ciclo, que é
      <code>i <= 5</code>. Se o resultado for
      verdadeiro (true) salta-se para o passo 2 (ou seja, executa mais
      uma vez o bloco de código e repetem-se as instruções que se seguem).
      Se o resultado for falso abandona-se o ciclo for e passa-se para a
      Instrução que vem imediatamente a seguir ao bloco de código.
   </p>
</body>
</html>
```



Gerar todos os cabeçalhos no corpo de um documento

```
<html>
<head>
```

```
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   for (var i = 1; i <= 6; i++)
   {
      document.write("<h" + i + ">Este é um cabeçalho de nível " + i)
      document.write("</h" + i + ">")
   }
// -->
</script>
</body>
</html>
```



4.2 Ciclos while e ciclos do...while

A instrução while repete a execução de um bloco de código enquanto uma condição for satisfeita. A sua sintaxe é a seguinte:

```
while (condição)
{
   código a executar
}
```

A instrução do...while repete a execução de um bloco de código enquanto uma condição for satisfeita mas executa-o pelo menos uma vez, mesmo que a condição nunca seja satisfeita. A sua sintaxe é a seguinte:

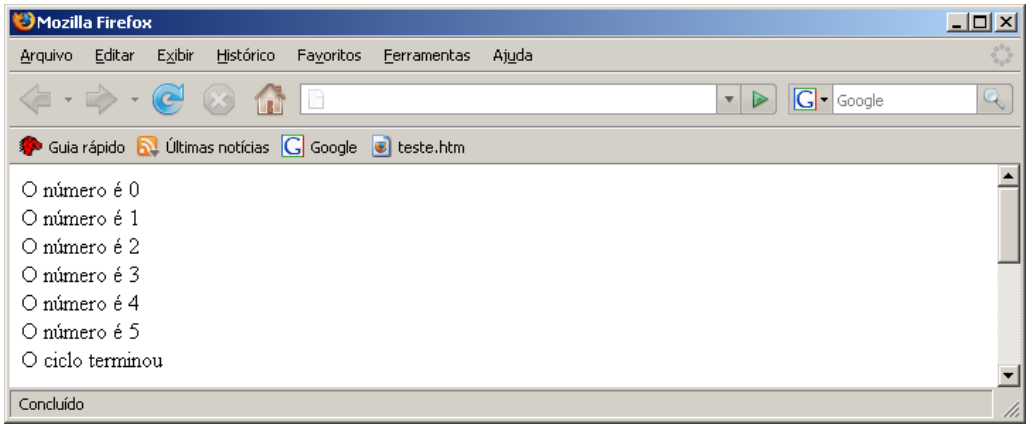
```
do
{
   código a executar
}
while (condição)
```

Exemplos de Aplicação

Ciclos (de repetição) while

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
```

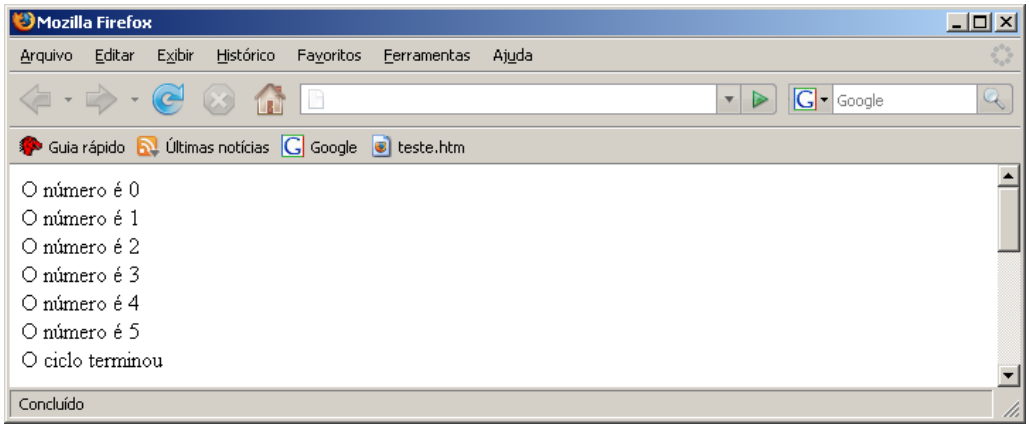
```
   var i = 0
   while (i <= 5)
   {
      document.write("O número é " + i)
      document.write("<br>")
      i++
   }
   document.write("O ciclo terminou<br>")
// -->
</script>
<h3>Explicação</h3>
   <p>
      1) O ciclo <code>while</code> verifica
      a condição <code>i <= 5</code>.
      Se o resultado for verdadeiro (true) executa
      o bloco de código que está entre chaves ({ ... })
      logo a seguir à instrução <code>while</code>.
      Se o resultado for falso abandona-se o ciclo for e passa para a
      instrução que vem imediatamente a seguir ao bloco de código.
   </p>
   <p>
      2) Repete-se o passo 1. Repare que o ciclo acaba parando porque cada
      vez que o bloco de código é executado o valor da variável
      <code>i</code> é aumentado em uma unidade. Se não fosse assim o ciclo
      nunca terminaria.
   </p>
</body>
</html>
```



Ciclos (de repetição) do ... while

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   i = 0
   do
   {
      document.write("O número é " + i)
      document.write("<br>")
      i++
   }
   while (i <= 5)
   document.write("O ciclo terminou<br>")
// -->
</script>
<h3>Explicação</h3>
   <p>
```

```
      1) O bloco de código que está entre chaves ({ ... })
      logo a seguir à instrução <code>do</code> é sempre
      executado sem que seja necessário verificar qualquer condição.
</p>
<p>
      2) Verifica-se a condição que está dentro da instrução
      <code>while</code>, que neste caso é <code>i <= 5</code>.
      Se o resultado for verdadeiro (true) executa-se mais uma vez
      o bloco de código que está entre chaves ({ ... }).
      Se o resultado for falso abandona-se o ciclo e passa-se para a
      instrução vem imediatamente a seguir ao bloco de código.
</p>
<p>
      Repare que o ciclo é sempre executado pelo menos uma vez.
      Ele acaba parando porque a cada execução o bloco de código
      aumenta o valor da variável <code>i</code> em uma unidade.
      Se não fosse assim o ciclo nunca terminaria.
</p>
</body>
</html>
```



5. Construir código robusto: as instruções try ... catch

Nota: Apesar destas instruções não fazerem parte do padrão ECMAScript, elas são suportadas por todos os browsers importantes e são de grande utilidade para a construção de um código robusto.

Diversas vezes surgem situações em que uma parte de um script é executada em condições desfavoráveis. Isso acontece sempre que um script tenta usar um objeto que não está definido ou tenta realizar uma operação que não pode ser realizada.

Sempre que aparece uma situação deste tipo surge uma exceção na execução do código. O comportamento normal do interpretador de JavaScript é parar a execução do script porque não sabe o que deve fazer a seguir.

Porém, há situações adversas das quais o script pode recuperar em boas condições desde que o programador lhe dê instruções para tal. Isso se consegue fazendo a captura da exceção gerada pelo erro e executando código capaz de fazer a recuperação.

Para conseguirmos isso devemos usar as instruções try...catch, que têm a sintaxe seguinte:

```
try
{
   bloco com código normal mas que pode gerar erros
}
catch(exceção)
{
```

```
   bloco com código capaz de fazer a recuperação dos erros
}
```

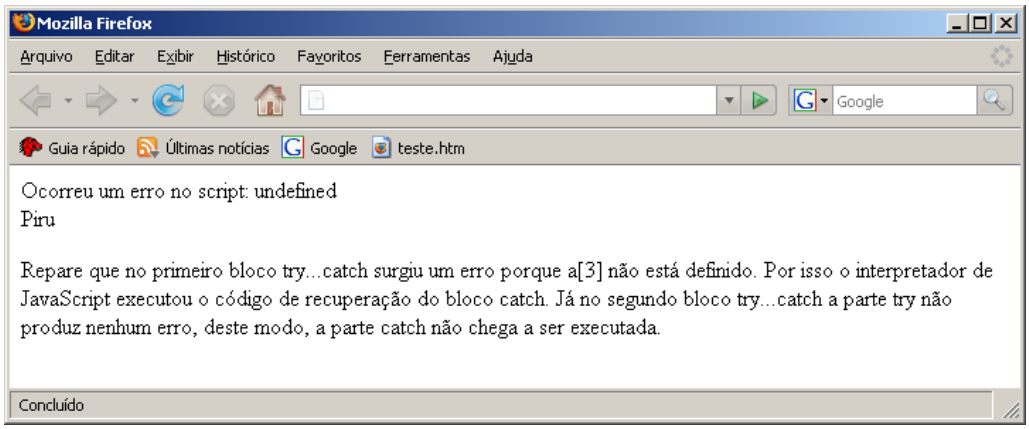
A seção try contém o código normal logo a seguir à instrução try. Ao chegar a esta instrução o interpretador de JavaScript tenta executar esse bloco da forma habitual. Se não ocorrer nenhum erro o interpretador ignora o código que está na seção catch e continua a executar o resto do script. Se ocorrer um erro no bloco try esse erro é capturado pelo bloco catch. É aí que o interpretador de JavaScript continua a executar o código imediatamente após o erro. No bloco catch devemos colocar o código de recuperação.

Os exemplos apresentados na lista seguinte mostram como isto funciona:

Exemplos de Aplicação

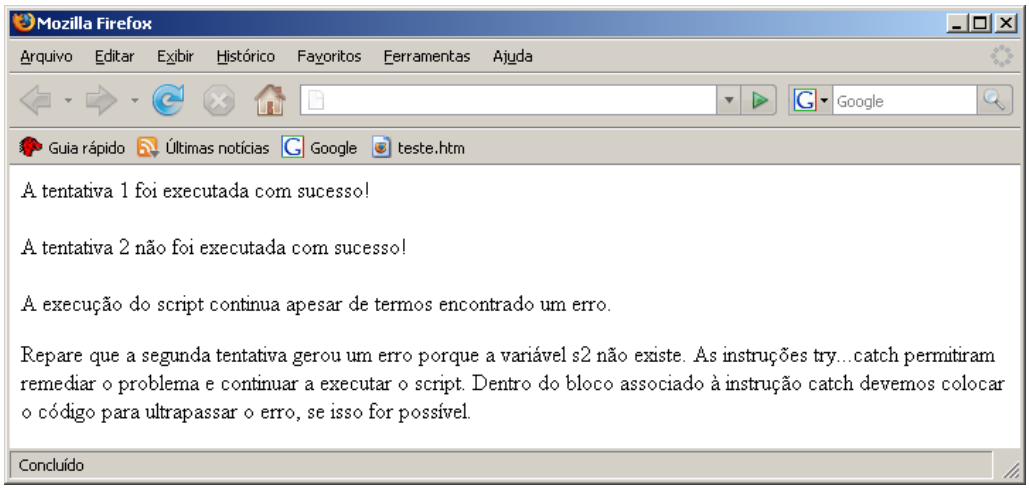
As instruções try ... catch

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var s
   var a = new Array("Balinhas", "Pirulitos")

   try
   {
      s = a[3].substring(0,4)
   }
   catch(e)
   {
      s = "Ocorreu um erro no script: "+e.description
   }
   document.write(s)
   document.write('<br>')

   try
   {
      s = a[1].substring(0,4)
   }
   catch(e)
   {
      s = "Ocorreu um erro no script: "+e
   }
   document.write(s)
// -->
</script>
   <p>
      Repare que no primeiro bloco try...catch surgiu um
      erro porque a[3] não está definido. Por isso o
      interpretador de JavaScript executou o código de recuperação
      do bloco catch. Já no segundo bloco try...catch a parte
      try não produz nenhum erro, deste modo, a parte catch não
      chega a ser executada.
   </p>
</body>
</html>
```



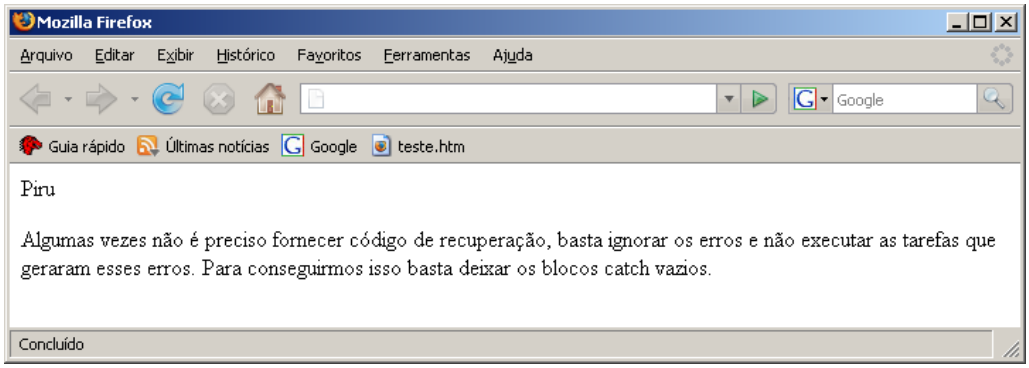
As instruções try ... catch (versão 2)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var s1 = "A tentativa 1 foi executada com sucesso!<br><br>"
   // TENTATIVA 1
   try
   {
      // Vamos tentar executar código que não tem erros
      document.write(s1)
   }
   catch(e)
   {
      document.write("A tentativa 1 não foi executada com sucesso!<br><br>")
   }

   // TENTATIVA 2
   try
   {
      // Vamos tentar executar código que TEM ERROS (s2 não existe)
      document.write(s2)
   }
   catch(e)
   {
      document.write("A tentativa 2 não foi executada com sucesso!<br><br>")
   }

   document.write("A execução do script continua apesar de termos encontrado um
erro.")
// -->
</script>
   <p>
      Repare que a segunda tentativa gerou um erro porque a variável
      s2 não existe. As instruções try...catch permitiram
      remediar o problema e continuar a executar o script.
      Dentro do bloco associado à instrução catch devemos colocar
      o código para ultrapassar o erro, se isso for possível.
   </p>
</body>
</html>
```



As instruções try ... catch (versão 3)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var a = new Array("Balinhas", "Pirulitos")
   try
   {
      document.write(a[3].substring(0,4))
      document.write('<br>')
   }
   catch(e){}

   try
   {
      document.write(a[1].substring(0,4))
      document.write('<br>')
   }
   catch(e){}
// -->
</script>
   <p>
      Algumas vezes não é preciso fornecer código de recuperação, basta
      ignorar os erros e não executar as tarefas que geraram esses erros.
      Para conseguirmos isso basta deixar os blocos catch vazios.
   </p>
</body>
</html>
```



6. Construir e usar uma função

Designamos por função um bloco de código a que damos um nome para que possa ser chamado várias vezes em locais diferentes. Para fazermos com que esse código seja executado nós invocamos chamando-o pelo seu nome (nome da função).

Vejamos um exemplo de uma função simples que não faz nada:

```
function fazerNada()
{
}
```

No exemplo seguinte a função não executa qualquer operação, mas devolve um valor de texto constante:

```
function fazerNada_2()
{
   return "Eu não faço nada!"
}
```

Sempre que uma função precisa devolver o resultado do seu trabalho deve usar a instrução "return". Esta instrução especifica o valor que é devolvido, o qual será inserido no local em que a função foi chamada.

```
function fazerNada_2()
{
   return "Eu não faço nada!"
}
```

Para usarmos esta função basta escrevermos:

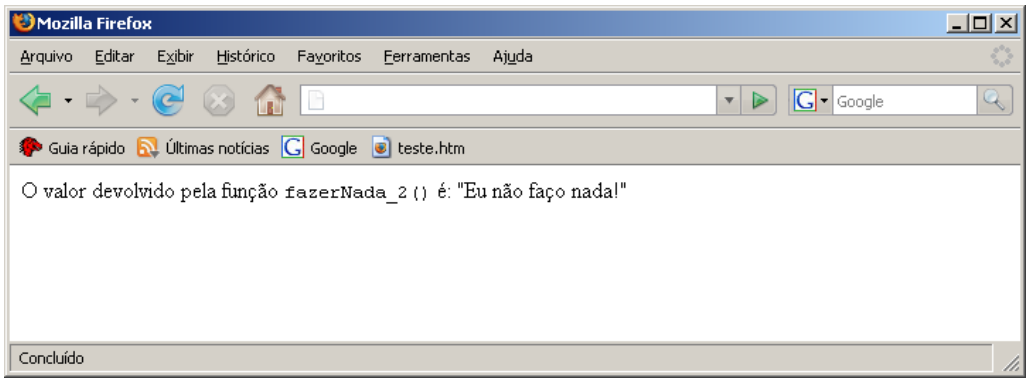
```
var s = fazerNada_2()
```

e a variável s passará a guardar o valor "Eu não faço nada!"

Exemplos de Aplicação

Uma função que não recebe argumentos

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function fazerNada_2()
   {
      return "Eu não faço nada!"
   }
// -->
</script>
</head>
<body>
   <p>
      O valor devolvido pela função <code>fazerNada_2()</code> é:
      <script type="text/javascript">
      <!--
         document.write('"'+fazerNada_2()+'"')
      // -->
      </script>
</body>
</html>
```



6.1 Funções com um número fixo de argumentos

A seguir temos uma função que recebe um número como argumento, soma duas unidades a esse número e devolve o valor da soma:

```
function somar(n)
{
   var soma = n + 2
   return soma
}
```

Para usarmos esta função basta que escrevamos:

```
var r = somar(5)
```

e a variável r passará a ter o valor 7.

Se precisarmos usar mais argumentos basta especificá-los ao declarar a função:

```
function multiplicar(p, q)
{
   var m = p*q
   return m
}
```

Para usarmos esta função basta que escrevamos:

```
var produto = multiplicar(5, 4)
```

e a variável produto passará a ter o valor 20, que é igual a 5 vezes 4.

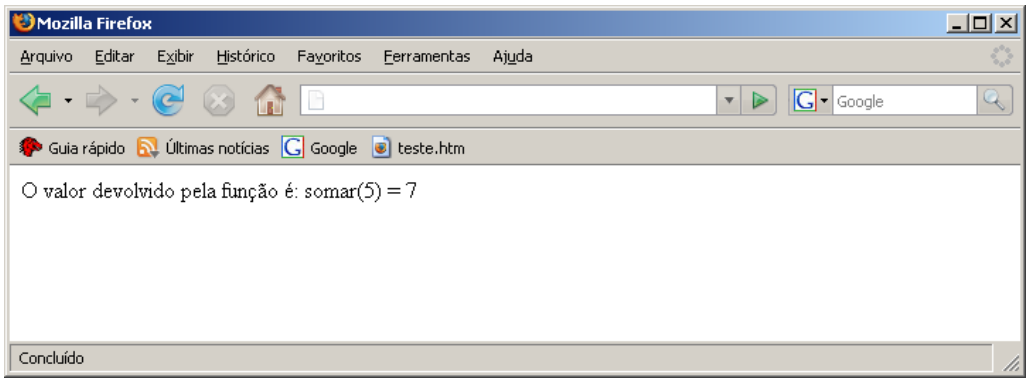
A linguagem JavaScript nos permite utilizar tantos argumentos quanto precisarmos.

Exemplos de Aplicação

Uma função que aceita um argumento

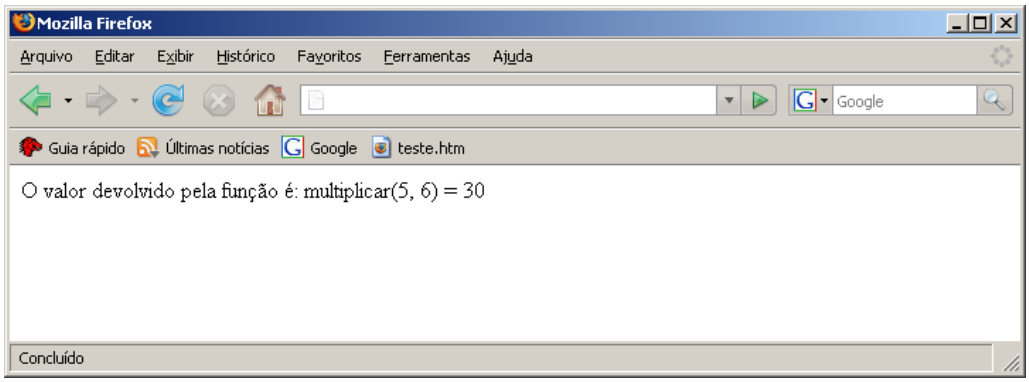
```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function somar(n)
   {
      var soma = n + 2
      return soma
```

```
   }
// -->
</script>
</head>
<body>
   <p>
      O valor devolvido pela função é:
      <script type="text/javascript">
      <!--
         document.write('somar(5) = '+somar(5))
      // -->
      </script>
   </p>
</body>
</html>
```



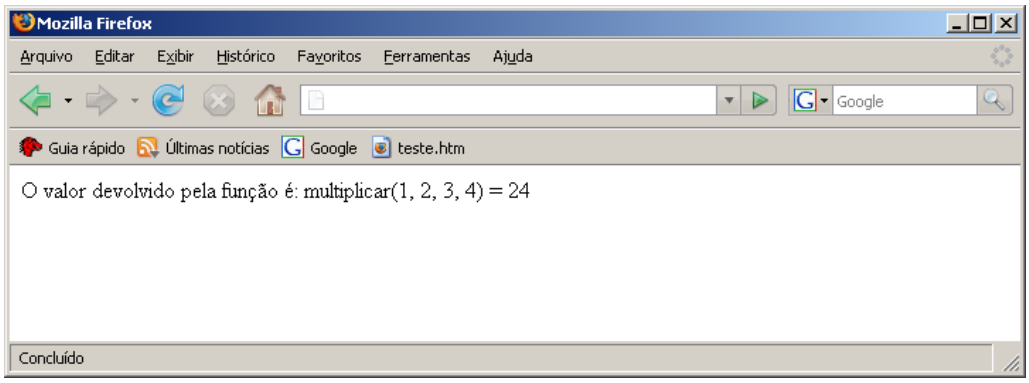
Uma função que aceita dois argumentos

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function multiplicar(p, q)
   {
      var m = p*q
     return m
   }
// -->
</script>
</head>
<body>
   <p>
      O valor devolvido pela função é:
      <script type="text/javascript">
      <!--
         document.write('multiplicar(5, 6) = '+multiplicar(5, 6))
      // -->
      </script>
   </p>
</body>
</html>
```



Uma função que aceita quatro argumentos

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function multiplicar(m1, m2, m3, m4)
   {
      var m = m1*m2*m3*m4
      return m
   }
// -->
</script>
</head>
<body>
   <p>
      O valor devolvido pela função é:
      <script type="text/javascript">
      <!--
         document.write('multiplicar(1, 2, 3, 4) = '+multiplicar(1, 2, 3, 4))
      // -->
      </script>
   </p>
</body>
</html>
```



6.2 Funções que aceitam um número variável de argumentos

Há casos em que o número de argumentos que são passados a uma função pode variar. Numa situação dessas pode ser preciso atribuir valores por omissão aos argumentos que possam estar em falta. Para fazermos isso precisamos fazer uma distinção entre duas situações diferentes: 1) Todos os argumentos são declarados ao definir a função; 2) Nenhum argumento é declarado ao definir a função.

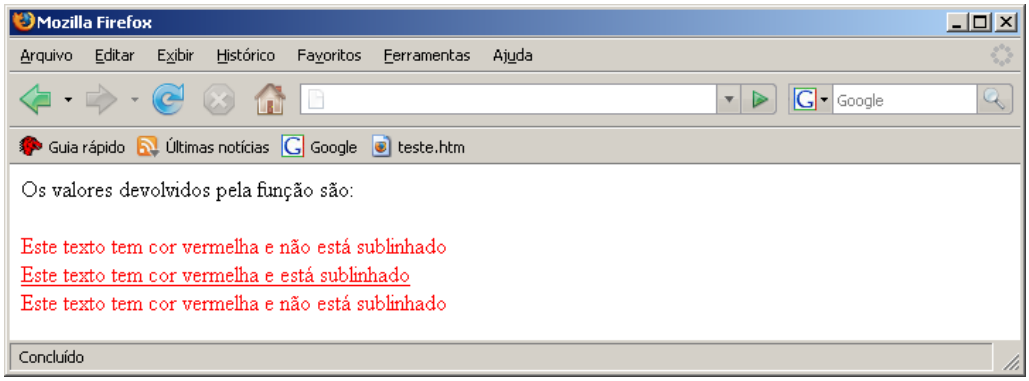
Todos os argumentos são declarados ao definir a função

Esta situação ocorre quando os argumentos têm naturezas diferentes. Alguns argumentos são obrigatórios mas outros são opcionais. Quando um argumento opcional não é fornecido a função deve dar-lhe um valor por omissão.

Exemplos de Aplicação

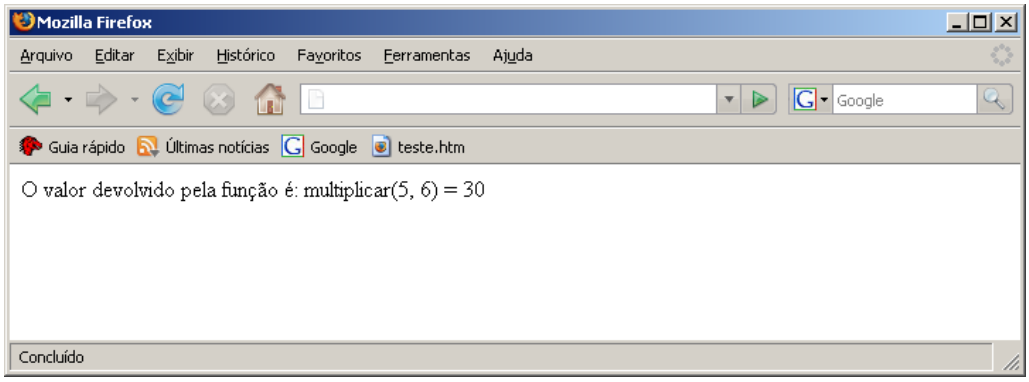
Uma função em que o segundo argumento é opcional

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function colorir(txt, sublinhar)
   {
      /*Se o argumento sublinhar não tiver sido
      fornecido o seu valor será undefined.
      Nesse caso damos um valor por omissão,
      que será false.*/
      if((''+sublinhar)=='undefined')
         var sublinhar = false
      var s='<span style="color: red;'
      if(sublinhar)
         s+='text-decoration: underline;'
      s+='">'+txt+'</span>'
      return s
   }
// -->
</script>
</head>
<body>
   <p>
      Os valores devolvidos pela função são:<br><br>
      <script type="text/javascript">
      <!--
         var s=colorir("Este texto tem cor vermelha e não está sublinhado",
         false)
         document.write(s)
         document.write('<br>')
         s=colorir("Este texto tem cor vermelha e está sublinhado", true)
         document.write(s+'<br>')
         // agora omitimos o segundo argumento
         s=colorir("Este texto tem cor vermelha e não está sublinhado")
         document.write(s)
      // -->
      </script>
</p>
</body>
</html>
```



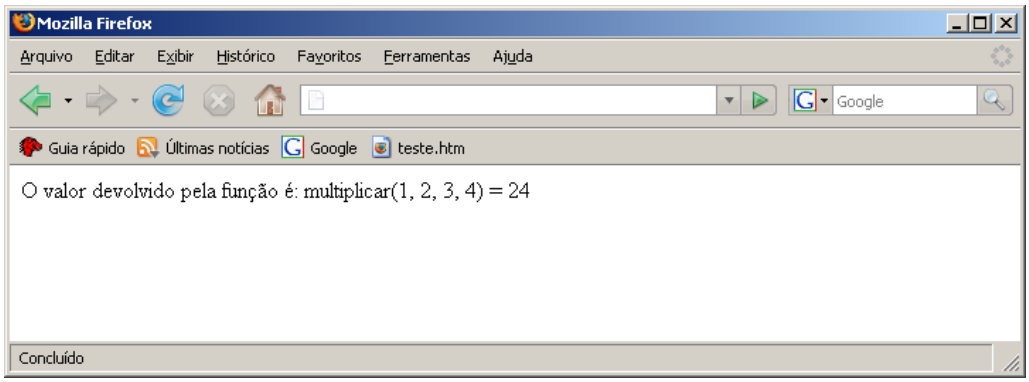
Uma função que aceita dois argumentos

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function multiplicar(p, q)
   {
      var m = p*q
      return m
   }
// -->
</script>
</head>
<body>
   <p>
      O valor devolvido pela função é:
      <script type="text/javascript">
      <!--
         document.write('multiplicar(5, 6) = '+multiplicar(5, 6))
      // -->
      </script>
   </p>
</body>
</html>
```



Uma função que aceita quatro argumentos

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function multiplicar(m1, m2, m3, m4)
   {
      var m = m1*m2*m3*m4
      return m
   }
// -->
</script>
</head>
<body>
   <p>
      O valor devolvido pela função é:
      <script type="text/javascript">
      <!--
         document.write('multiplicar(1, 2, 3, 4) = '+multiplicar(1, 2, 3, 4))
      // -->
      </script>
</body>
</html>
```



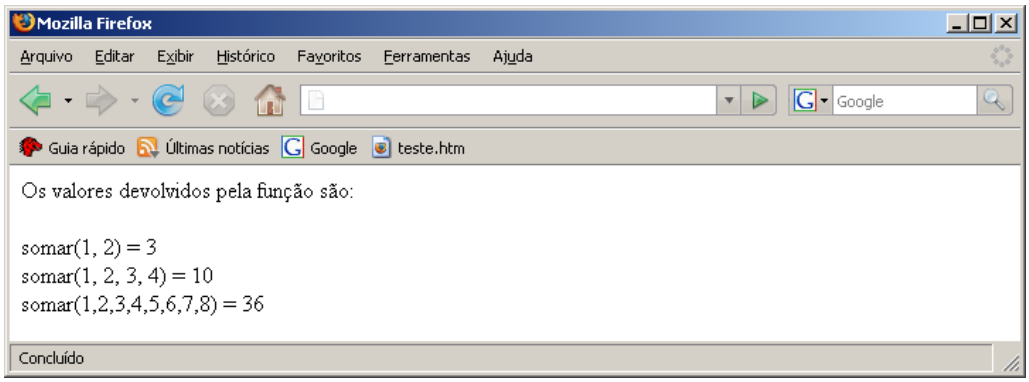
Não são declarados argumentos ao definir a função

Esta situação ocorre normalmente quando os argumentos são todos do mesmo tipo (são todos números ou são todos strings).

Exemplos de Aplicação

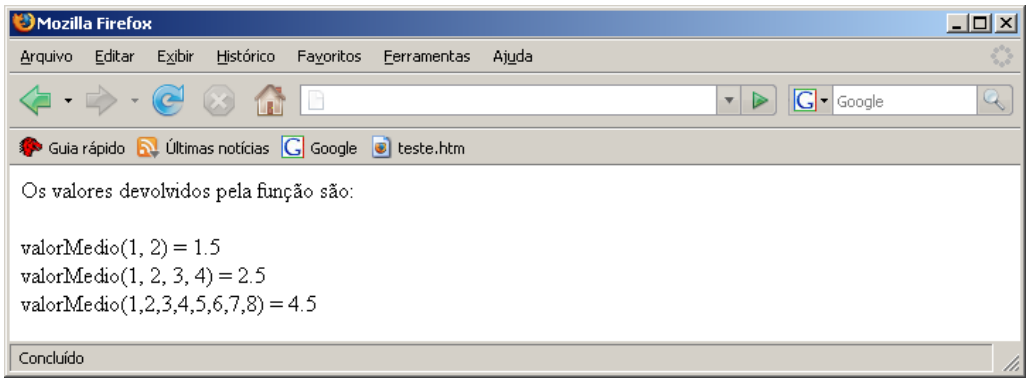
Uma função que devolve a soma dos seus argumentos (números)

```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function somar()
   {
      var soma=0
      for(var i=0;i<somar.arguments.length;++i)
         soma += somar.arguments[i]
      return soma
   }
// -->
</script>
</head>
<body>
   <p>
      Os valores devolvidos pela função são:<br><br>
      <script type="text/javascript">
      <!--
         document.write('somar(1, 2) = ' + somar(1, 2))
         document.write('<br>')
         document.write('somar(1, 2, 3, 4) = ' + somar(1, 2, 3, 4))
         document.write('<br>')
         document.write('somar(1,2,3,4,5,6,7,8) = ' + somar(1,2,3,4,5,6,7,8))
      // -->
      </script>
   </p>
</body>
</html>
```



Uma função que devolve o valor médio dos seus argumentos

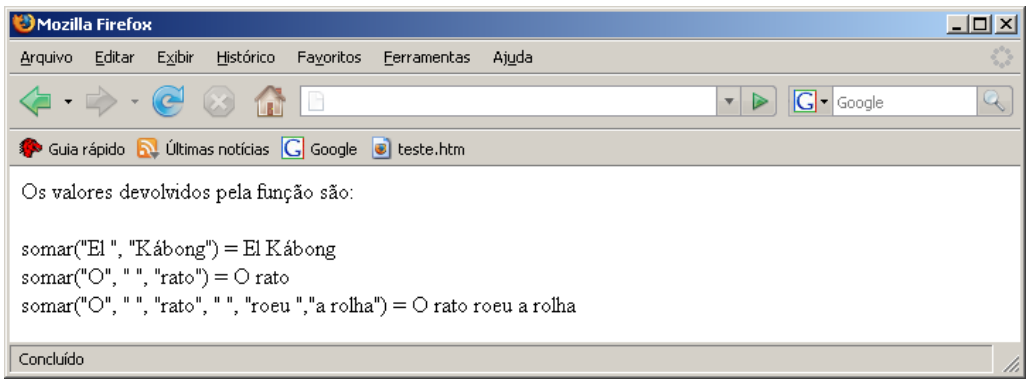
```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function valorMedio()
   {
      var soma=0
      for(var i=0;i<valorMedio.arguments.length;++i)
         soma += valorMedio.arguments[i]
      return soma/valorMedio.arguments.length
   }
// -->
</script>
</head>
<body>
   <p>
      Os valores devolvidos pela função são:<br><br>
      <script type="text/javascript">
      <!--
         document.write('valorMedio(1, 2) = ' + valorMedio(1, 2))
         document.write('<br>')
         document.write('valorMedio(1, 2, 3, 4) = ' + valorMedio(1, 2, 3, 4))
         document.write('<br>')
         document.write('valorMedio(1,2,3,4,5,6,7,8) = ' + valorMedio(1,2,3,4,
         5,6,7,8))
      // -->
      </script>
   </p>
</body>
</html>
```



Uma função que devolve a soma dos seus argumentos (strings)

```
<html>
<head>
```

```
<title></title>
<script type="text/javascript">
<!--
   function somar()
   {
      var soma=''
      for(var i=0;i<somar.arguments.length;++i)
         soma += somar.arguments[i]
      return soma
   }
// -->
</script>
</head>
<body>
   <p>
      Os valores devolvidos pela função são:<br><br>
      <script type="text/javascript">
      <!--
        document.write('somar("El ", "Kábong") = ' + somar("El ", "Kábong"))
        document.write('<br>')
        document.write('somar("O", " ", "rato") = ' + somar("O", " ", "rato"))
        document.write('<br>')
        document.write('somar("O", " ", "rato", " ", "roeu ","a rolha") = ' +
        somar("O", " ", "rato", " ", "roeu ","a rolha"))
// -->
</script>
</body>
</html>
```



7. Trabalhar com objetos

A linguagem JavaScript é uma linguagem orientada a objetos. Isto significa que em JavaScript tudo são objetos. No entanto, o estilo de programação adaptado em JavaScript torna esse fato menos evidente do que em outras linguagens, particularmente as linguagens compiladas como o Java, o C++ ou o C#. Para programar corretamente em JavaScript o programador não precisa dominar o modelo de programação orientada para objetos, bastando saber usar os objetos que são oferecidos pela linguagem e pelo web browser que corre o programa.

Na prática, o programador de JavaScript segue quase sempre o modelo de programação orientada para procedimentos (programação estruturada), que é menos exigente, e recorre sempre que necessário aos objetos pré-definidos que o browser coloca ao seu dispor. Esta estratégia funciona bem desde que as aplicações que se escreve se limitem ao controle e manipulação de páginas da Web. O JavaScript não serve para criar aplicações de classe empresarial como aquelas que se escrevem em Java, C++ ou C#.

Esta forma de programar, aliada à não necessidade de declarar os tipos das variáveis e ao fato de não ser preciso compilar os programas, faz com que seja possível produzir e testar código JavaScript com grande rapidez e com muito menos esforço do que aquele que é exigido por outras linguagens. A natureza das tarefas que são executadas em JavaScript faz com que só em situações excepcionais surja a necessidade de definir novos objetos.

7.1 Criar um novo objeto

O web browser cria automaticamente muitos dos objetos que coloca a dispor do programador de JavaScript. Esses objetos estão prontos para ser usados sem que o programador tenha de criá-los.

Para poder trabalhar com os objetos que não são criados pelo browser o programador é obrigado a criá-los antes de os usar. Um bom exemplo é o objeto Array, que pode ser criado de diversas formas:

```
// um Array com espaço reservado para 3 elementos
// e cujo conteúdo será definido mais tarde
var a = new Array(3)

// um Array com três elementos
var b = new Array("Papagaio", "Ave do paraíso", "Canário")

// um Array cujas propriedades e conteúdo serão definidos mais tarde
var c = new Array()
```

Depois de criarmos um objeto podemos utilizar as propriedades e os métodos que ele coloca à nossa disposição.

7.2 Propriedades dos objetos

As propriedades de um objeto são variáveis que pertencem a esse objeto. Nós podemos ler os valores dessas variáveis (propriedades) e alterar os valores daquelas que aceitem ser escritas (algumas só podem ser lidas.) O objeto Array, que já vimos antes, oferece uma única propriedade, que é a propriedade length. O valor dessa propriedade é igual ao número de elementos que estão guardados no Array.

Para acessarmos uma propriedade de um objeto escrevemos o nome da variável que representa o objeto, um ponto e o nome da propriedade. O exemplo seguinte mostra como se acessa a propriedade length do objeto Array:

```
var a = new Array("Papagaio", "Ave do paraíso", "Canário")

// Para acessarmos a propriedade escrevemos o nome da variável que
// representa o objeto (que é a), um ponto e o nome da propriedade.

var l = a.length // l passou a valer 3 (número de elementos do Array)
```

7.3 Métodos de objetos

Os métodos de um objeto são funções que pertencem a esse objeto. Repare que enquanto que as propriedades correspondem a variáveis, os métodos correspondem a funções. Os métodos realizam transformações no objeto a que pertencem e devolvem o resultado da sua atuação. O método reverse() devolve um novo Array com os mesmos elemento dos Array a que pertence mas com a ordem invertida:

```
var a = new Array("Papagaio", "Ave do paraíso", "Canário")

// Para acessarmos o método escrevemos o nome da variável que
// representa o objeto (o seu nome é a), um ponto e o nome do método.

// b será igual a ("Canário", "Ave do paraíso", "Papagaio")
var b = a.reverse()
```

Métodos estáticos

Alguns objetos oferecem métodos estáticos. Esses métodos diferem dos métodos normais pelo fato de não pertencerem a um objeto criado com a instrução new. Os métodos estáticos estão sempre disponíveis para serem usados, não é preciso criar um objeto para usá-los.

Um exemplo de um método estático é o método String.fromCharCode(). Por ser um método estático do objeto String ele deve ser sempre invocado na forma String.fromCharCode () e antes de .fromCharCode () nunca se deve colocar o nome de uma variável. Quando utilizado deve começar sempre por String.

7.4 Objetos definidos pelo padrão ECMAScript

O padrão ECMAScript define as regras da linguagem, as instruções que a compõem e um conjunto reduzido de objetos. Este padrão serve de base a diversas linguagens de script. Entre elas temos: o JavaScript criado pela Netscape, o JScript da Microsoft e a versão 2 da linguagem ActionScript da Macromedia (implementada nas versões do Flash Player).

Apesar de todas estas linguagens se basearem no padrão ECMAScript, elas não são iguais porque ao padrão acrescentaram os seus próprios objetos. Isto é necessário porque o padrão ECMAScript constitui apenas o núcleo dessas linguagens. Ele não define objetos que são específicos de cada uma das áreas em que vai ser aplicado. Essa é uma tarefa que fica ao cuidado dos criadores das implementações de ECMAScript.

Podemos dizer que o padrão ECMAScript oferece uma infra-estrutura sólida sobre a qual se constroem linguagens de script destinadas a diversos fins: JavaScript para o HTML dinâmico, ActionScript para controlar aplicações baseadas no Flash Player da Macromedia, ou outras que podem servir para outros fins.

Os objetos definidos pelo padrão ECMAScript são apenas aqueles que são necessários para criar uma linguagem robusta e funcional. Nos capítulos seguintes deste curso vamos estudar esses objetos, que são: Array, Date, Math e String. Estes objetos oferecem funcionalidades de importância fundamental e por isso estão presentes em todas as linguagens compatíveis com o padrão ECMAScript.

7.5 As declarações this e with

A declaração this

A declaração this representa o próprio objeto em que é usada. Esse objeto pode ser uma função, uma ligação de hipertexto, uma imagem, etc. Esta declaração é bastante útil em HTML Dinâmico porque nos ajuda muito a construir funções que respondem a eventos sabendo sempre qual foi o objeto que originou o evento e funcionam corretamente em todos os browsers.

A declaração with

Esta declaração estabelece o objeto ao qual se aplica uma série de instruções. Os dois exemplos seguintes usam o objeto Math para ilustrar o uso desta declaração e são totalmente equivalentes (se ainda não conhece o objeto Math estude-o mais à frente).

Versão 1:

```
x = Math.cos(3 * Math.PI) + Math.sin(Math.LN10)
y = Math.tan(14 * Math.E)
```

Versão 2, equivalente à anterior. Repare que não foi preciso colocar a parte Math antes dos nomes dos métodos cos(), sin() ou tan().

```
with (Math)
{
   x = cos(3 * PI) + sin (LN10)
   y = tan(14 * E)
}
```

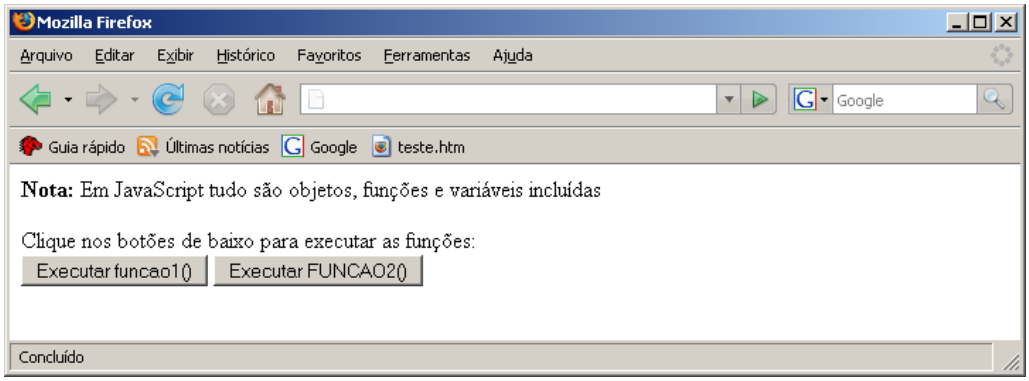
Quando o número de instruções é pequeno não vale a pena usar a declaração with, mas se esse número crescer muito o código será mais compacto e fácil de ler se usarmos esta declaração.

Exemplos de Aplicação

As funções como objetos: como se usa a declaração this

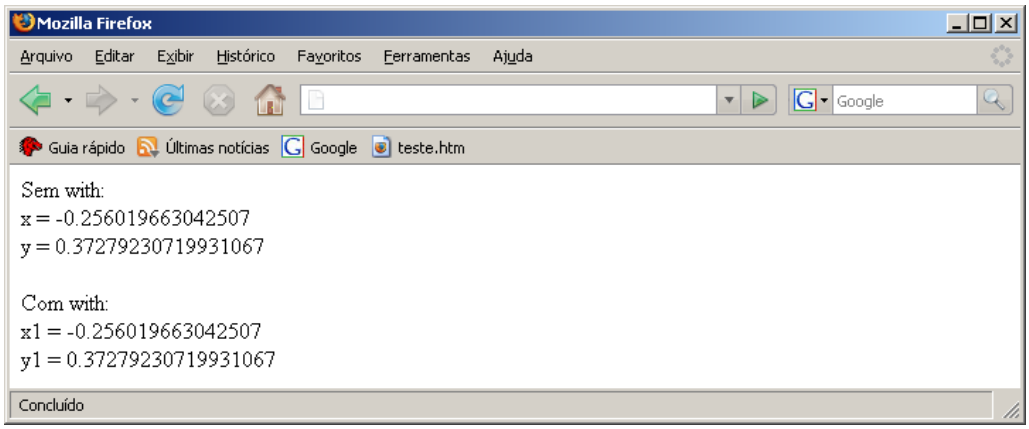
```
<html>
<head>
<title></title>
<script type="text/javascript">
<!--
   function mostrarStringS(o)
   {
      // O argumento o representa um objeto (que neste caso é uma função)
      var z="O valor da string s pertencente ao objeto \
      que nos foi passado como argumento é:\n\n"
      alert(z+o.s)
   }

   function funcao1()
   {
      // Declaramos uma variável s que pertence a este (this) objeto
      this.s="Eu sou a variável s da funcao1() e sou uma string."

      // A seguir invocamos a função mostrarStringS() dando como
      // argumento este (this) objeto
      mostrarStringS(this)
   }

   function FUNCAO2()
   {
      // Declaramos uma variável s que pertence a este (this) objeto
      this.s="Eu pertenço à FUNCAO2() e sou uma string. O meu nome é s"

      // A seguir invocamos a função mostrarStringS() dando como
      // argumento este (this) objeto
      mostrarStringS(this)
   }

// -->
</script>
</head>
```

```
<body>
   <p>
      <b>Nota:</b>
      Em JavaScript tudo são objetos, funções e variáveis incluídas
   </p>
   <form action="javascript:;">
      Clique nos botões de baixo para executar as funções:<br>
      <input type="button" onclick="funcao1()" value="Executar funcao1()">
      <input type="button" onclick="FUNCAO2()" value="Executar FUNCAO2()">
   </form>
</body>
</html>
```



A declaração with

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var x, y, x1, y1

   x = Math.cos(3 * Math.PI) + Math.sin(Math.LN10)
   y = Math.tan(14 * Math.E)

   document.write("Sem with:<br>")
   document.write("x = " + x)
   document.write("<br>")
   document.write("y = " + y)
   document.write("<br><br>")

   with (Math)
   {
      x1 = cos(3 * PI) + sin (LN10)
      y1 = tan(14 * E)
   }
   document.write("Com with:<br>")
   document.write("x1 = " + x1)
   document.write("<br>")
   document.write("y1 = " + y1)
// -->
</script>
</body>
</html>
```



8. Objeto Array

O objeto Array serve para definir um tipo de variável que é capaz de guardar sob o mesmo nome uma quantidade de valores numéricos, de texto ou de objetos. Este objeto pode ser considerado como uma lista, ou cadeia de itens, em que cada item é uma variável ou um objeto. O acesso a cada um dos itens da lista faz-se recorrendo ao seu índice, que é o número de ordem do item na lista.

Criação de um novo Array

```
// um Array cujo tamanho e conteúdo serão definidos mais tarde
var c = new Array()

// um Array com espaço reservado para N elementos
// e cujo conteúdo será definido mais tarde
var a = new Array(N)

// um Array com N+1 elementos definidos logo de início
var b = new Array(elemento_0, elemento_1, ..., elemento_N)
```

8.1 Propriedades do objeto Array

Propriedade	Descrição
length	Fornece o número de elementos que estão no Array. O valor desta propriedade é estabelecido quando o Array é criado, mas pode crescer se formos juntando mais elementos ao Array.

8.2 Métodos do objeto Array

Método	Descrição
join()	Devolve uma String (variável de texto) que representa o conteúdo do Array
reverse()	Devolve um novo Array em que a ordem dos elementos está invertida (em particular temos que o elemento que tinha o índice zero passa a ter o índice mais alto e vice versa)
sort()	Devolve um novo Array em que os elementos estão ordenados (geralmente por ordem crescente)

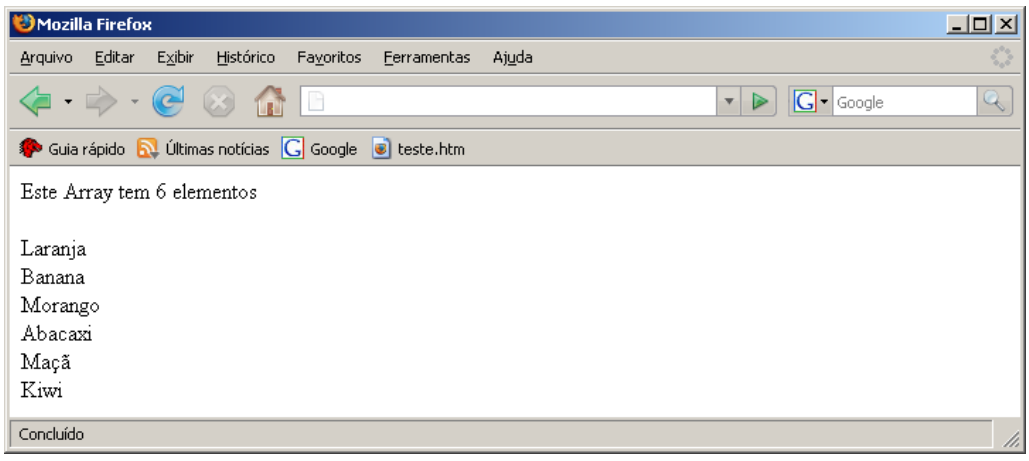
8.3 Coleções

O termo coleção usa-se normalmente para designar um Array cujos itens são objetos todos do mesmo tipo. O conceito de coleção é muito usado em HTML Dinâmico para designar grupos de objetos do mesmo tipo. Um exemplo disto é a coleção images, que é um Array cujos itens são os objetos que representam as imagens que estão no documento (cada imagem corresponde um objeto Image.)

Exemplos de Aplicação

Construir um Array para guardar nomes

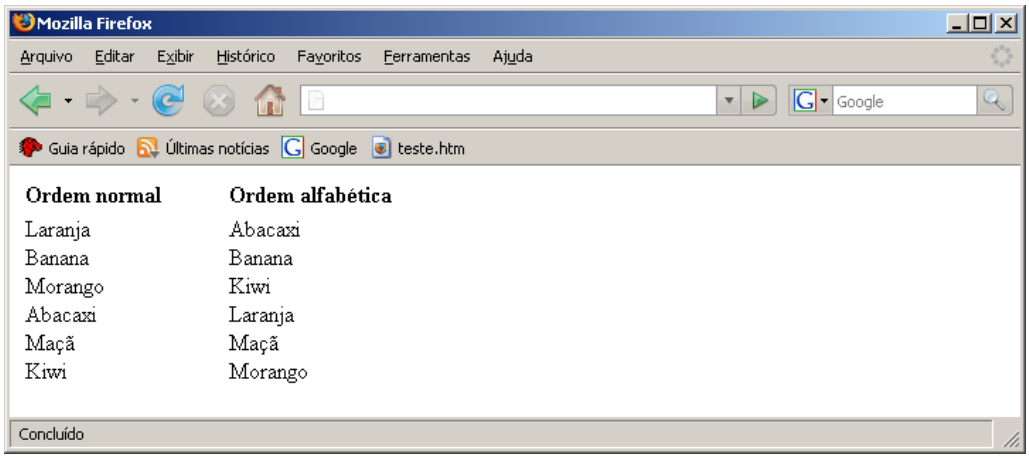
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var fruta = new Array(6)
   fruta[0] = "Laranja"
   fruta[1] = "Banana"
   fruta[2] = "Morango"
   fruta[3] = "Abacaxi"
   fruta[4] = "Maçã"
   fruta[5] = "Kiwi"

   document.write("Este Array tem "+fruta.length+" elementos<br><br>")
   for (var i=0; i<fruta.length; i++)
   {
      document.write(fruta[i] + "<br>")
   }
// -->
</script>
</body>
</html>
```



Ordenar os elementos de um Array (strings)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var fruta = new Array("Laranja","Banana","Morango","Abacaxi","Maçã","Kiwi")
// -->
</script>
   <table>
```

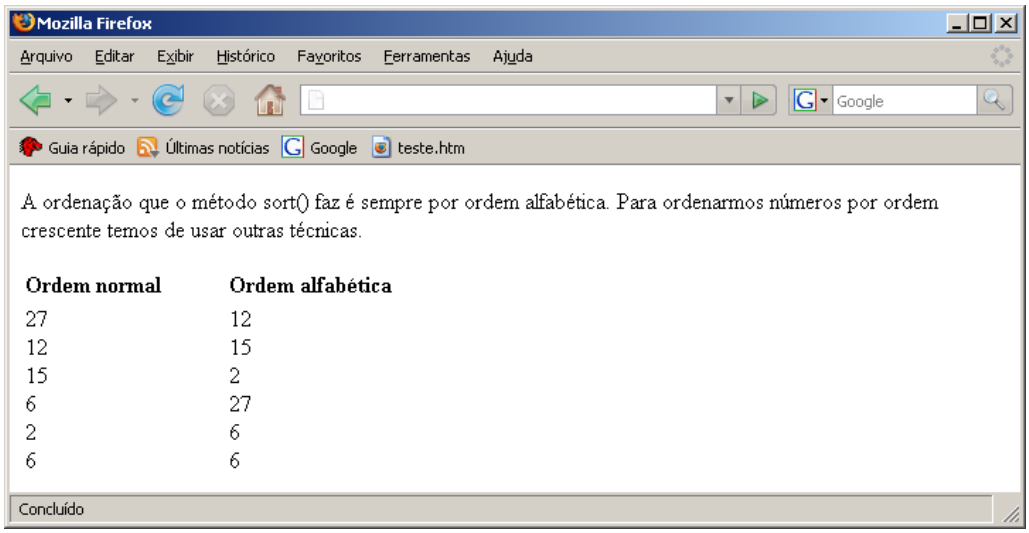
```
      <tr>
         <td width="140"><b>Ordem normal</b></td>
         <td><b>Ordem alfabética</b></td>
      </tr>
         <td>
            <script type="text/javascript">
            <!--
               for (var i=0; i<6; i++)
                  document.write(fruta[i] + "<br>")
            // -->
         </script>
         </td>
         <td>
            <script type="text/javascript">
            <!--
               // Agora ordenamos o Array por ordem alfabética invocando o
               // método sort()
               fruta.sort()
               for (var i=0; i<6; i++)
                  document.write(fruta[i] + "<br>")
            // -->
            </script>
         </td>
      </tr>
   </table>
</body>
</html>
```



Ordenar os elementos de um Array (números)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var numeros = new Array(27, 12, 15, 6, 2, 6)
// -->
</script>
   <p>
      A ordenação que o método sort() faz é sempre por ordem alfabética.
      Para ordenarmos números por ordem crescente temos de usar outras
      técnicas.
   </p>
   <table>
      <tr>
         <td width="140"><b>Ordem normal</b></td>
         <td><b>Ordem alfabética</b></td>
      </tr>
```

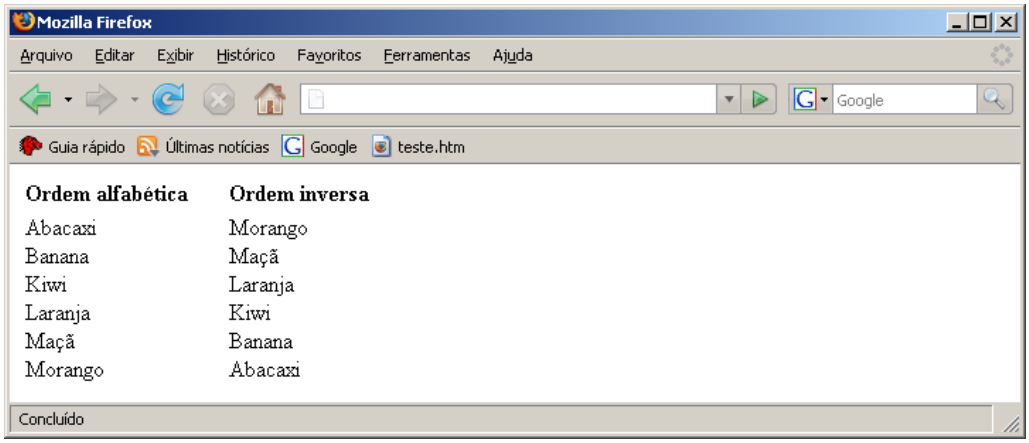
```
         <td>
            <script type="text/javascript">
            <!--
               for (var i=0; i<6; i++)
                  document.write(numeros[i] + "<br>")
            // -->
            </script>
         </td>
         <td>
            <script type="text/javascript">
            <!--
               // Agora ordenamos o Array por ordem alfabética invocando o
               //método sort()
               numeros.sort()
               for (var i=0; i<6; i++)
                  document.write(numeros[i] + "<br>")
            // -->
            </script>
         </td>
      </tr>
   </table>
</body>
</html>
```



Inverter a ordem dos elementos de um Array

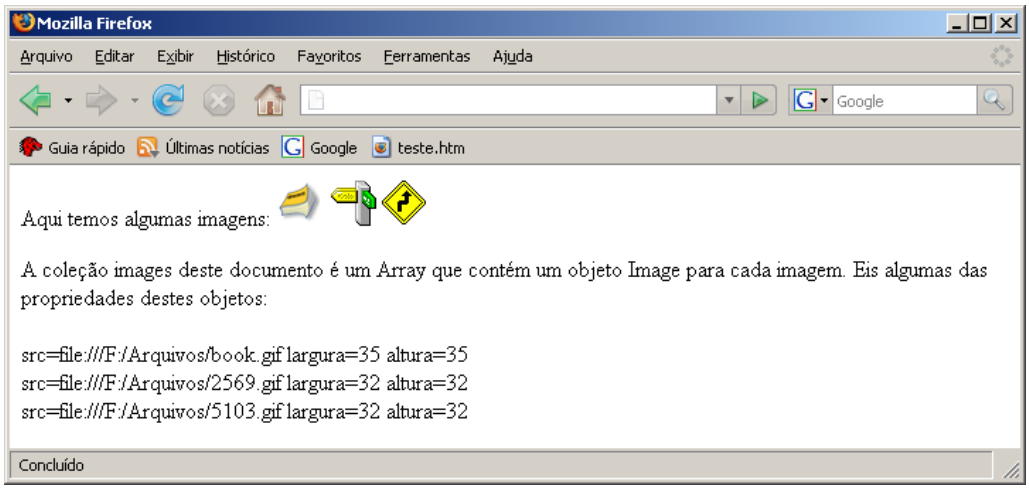
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var fruta = new Array("Laranja","Banana","Morango","Abacaxi","Maçã","Kiwi")
   // Agora ordenamos o Array por ordem alfabética invocando o método sort()
   fruta.sort()
// -->
</script>
   <table>
      <tr>
         <td width="140"><b>Ordem alfabética</b></td>
         <td><b>Ordem inversa</b></td>
      </tr>
         <td>
            <script type="text/javascript">
            <!--
               for (var i=0; i<6; i++)
```

```
                document.write(fruta[i] + "<br>")
            // -->
            </script>
         </td>
         <td>
            <script type="text/javascript">
            <!--
               // Agora invertemos a ordem do Array invocando o método
               //reverse()
               fruta.reverse()
               for (var i=0; i<6; i++)
                  document.write(fruta[i] + "<br>")
            // -->
            </script>
         </td>
      </tr>
   </table>
</body>
</html>
```



A coleção Images

```
<html>
<head>
<title></title>
</head>
<body>
   <p>
      Aqui temos algumas imagens:
      <img src="book.gif" alt="Um livro">
      <img src="2569.gif" alt="Direções">
      <img src="5103.gif" alt="Curvas">
   </p>
   <p>
      A coleção images deste documento é um Array que contém um objeto Image
      para cada imagem. Eis algumas das propriedades destes
      objetos:<br><br>

      <script type="text/javascript">
      <!--
         var imgs=document.images
         for(var i=0;i<imgs.length;++i)
            document.write("src="+imgs[i].src+" largura="+imgs[i].width+"
            altura="+imgs[i].height+"<br>")
      // -->
      </script>
   </p>
</body>
</html>
```



9. Objeto Date

O objeto Date permite-nos ler, construir e realizar operações com datas e horas.

Datas numéricas e datas de texto

Em JavaScript nós podemos definir um conjunto data/hora por extenso ou através de um valor numérico. A definição por extenso pode ser feita de várias formas, como ilustra a seguir:

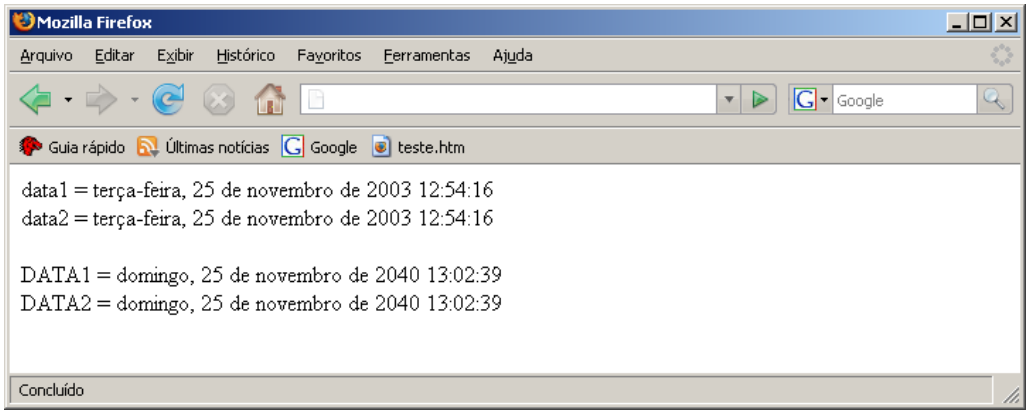
```
Fri, 21 Nov 2003 10:43:34 UTC
Tue Nov 25 14:45:42 UTC 2003
Tue, 25 Nov 2003 14:48:21 GMT
Tue Nov 25 2003 14:46:37 GMT+0000
```

A outra forma que podemos usar para definir um conjunto data/hora usa como ponto de referência as zero horas do dia 1 de Janeiro de 1970. Para definirmos uma data e uma hora usando esta forma nós indicamos o número de milisegundos que decorreram entre as zero horas de 1 de Janeiro de 1970 e a data que queremos definir. Se usarmos um valor negativo estaremos indicando uma data anterior ao ponto de referência.

A seguir temos exemplos desta forma de definir datas:

```
// 1069772056710  equivale a  Tue, 25 Nov 2003 14:54:16 UTC
// 2237468559000  equivale a  Sun Nov 25 2040 15:02:39 GMT+0000

<html>
<body>
<script type="text/javascript">
<!--
   // 1069772056710  equivale a  Tue, 25 Nov 2003 14:54:16 UTC
   // 2237468559000  equivale a  Sun Nov 25 2040 15:02:39 GMT+0000

   var data1=new Date(1069772056710)
   var data2=new Date("Tue, 25 Nov 2003 14:54:16 UTC")
   document.write("data1 = "+data1.toLocaleString())
   document.write("<br>")
   document.write("data2 = "+data2.toLocaleString())

   document.write("<br><br>")

   var DATA1=new Date(2237468559000)
```

```
   var DATA2=new Date("Sun Nov 25 2040 15:02:39 GMT+0000")
   document.write("DATA1 = "+DATA1.toLocaleString())
   document.write("<br>")
   document.write("DATA2 = "+DATA2.toLocaleString())
-->
</script>
</body>
</html>
```



Criação de um novo objeto Date

```
// Cria um objeto com a data e a hora atuais
oData1 = new Date()

// Cria objeto com a data e hora indicadas por valorData (texto ou numérica)
oData2 = new Date(valorData)

// Cria um objeto com a data indicada. É obrigatório indicar o ano, o mês e o
// dia. As horas, os minutos, os segundos e os milisegundos são facultativos.
oData3 = new Date(ano, mês, dia[, horas[, minutos[,segundos[,milisegundos]]]])
```

9.1 Métodos do objeto Date

Método	Descrição
getDate()	Devolve o dia do mês (de 1 a 31) que está guardado no objeto Date
getDay()	Devolve o dia da semana (de 0=Domingo até 6=Sábado) guardado no objeto Date
getMonth()	Devolve o mês (de 0=Janeiro até 11=Dezembro) guardado no objeto Date
getFullYear()	Devolve o valor completo do ano (quatro dígitos) guardado no objeto Date
getYear()	Devolve o valor incompleto do ano (apenas dois dígitos) guardado no objeto Date. Não use este método, use getFullYear em seu lugar.
getHours()	Devolve o valor da hora (de 0 a 23) guardada no objeto Date
getMinutes()	Devolve o valor dos minutos (de 0 a 59) guardados no objeto Date
getSeconds()	Devolve o valor dos segundos (de 0 a 59) guardados no objeto Date
getMilliseconds()	Devolve o valor dos milisegundos (de 0 a 999) guardados no objeto Date
getTime()	Devolve a quantidade de milisegundos decorridos desde as zero horas do dia 1 de Janeiro de 1970

	até à data que está guardada no objeto Date
getTimezoneOffset()	Devolve a diferença de tempo entre o fuso horário do computador que está a ser usado e o Tempo Médio de Greenwich ("Greenwich Mean Time", ou GMT)
getUTCDate()	Devolve o dia do mês (de 1 a 31) guardado no objeto Date medido no padrão de tempo universal (UTC)
getUTCDay()	Devolve o dia da semana (de 0 a 6) guardado no objeto Date medido no padrão de tempo universal (UTC)
getUTCMonth()	Devolve o valor do mês (de 0 a 11) guardado no objeto Date medido no padrão de tempo universal (UTC)
getUTCFullYear()	Devolve o valor completo do ano (com quatro dígitos) guardado no objeto Date medido no padrão de tempo universal (UTC)
getUTCHours()	Devolve a hora (de 0 a 23) guardada no objeto Date medida no padrão de tempo universal (UTC)
getUTCMinutes()	Devolve o valor dos minutos (de 0 a 59) guardados no objeto Date medidos no padrão de tempo universal (UTC)
getUTCSeconds()	Devolve o valor dos segundos (de 0 a 59) guardados no objeto Date medidos no padrão de tempo universal (UTC)
getUTCMilliseconds()	Devolve o valor dos milisegundos (de 0 a 999) guardados no objeto Date medidos no padrão de tempo universal (UTC)
setDate(dia_mês)	Acerta o valor do dia (de 1 a 31) do mês guardado no objeto Date
setFullYear(ano)	Acerta o valor do ano (com quatro dígitos) guardado no objeto Date
setHours(horas)	Acerta o valor da hora (de 0 a 23) guardada no objeto Date
setMilliseconds(milisegundos)	Acerta o valor dos milisegundos (de 0 a 999) guardados no objeto Date
setMinutes(minutos)	Acerta o valor dos minutos (de 0 a 59) guardados no objeto Date
setMonth(mês)	Acerta o valor do mês (de 0=Janeiro a 11=Dezembro) guardado no objeto Date
setSeconds(segundos)	Acerta o valor dos segundos (de 0 a 59) guardados no objeto Date
setTime(data_numérica)	Acerta a data e a hora guardadas no objeto Date para o valor correspondente às zero horas do dia 1 de Janeiro de 1970 mais o número de milisegundos que é fornecido como argumento do método
setYear(ano)	Acerta o valor do ano guardado no objeto Date. Não use este método, use antes o método setFullYear.
setUTCDate(dia_mês)	Acerta o valor do dia (de 1 a 31) do mês guardado no objeto Date usando o padrão de tempo universal (UTC)
setUTCDay(dia_semana)	Acerta o valor do dia da semana (de 0=Domingo a 6=Sábado) guardado no objeto Date usando o

	padrão de tempo universal (UTC)
setUTCMonth(mês)	Acerta o valor do mês (de 0=Janeiro a 11=Dezembro) guardado no objeto Date usando o padrão de tempo universal (UTC)
setUTCFullYear(ano)	Acerta o valor do ano (com quatro dígitos) guardado no objeto Date
setUTCHour(horas)	Acerta o valor da hora (de 0 a 23) guardada no objeto Date usando o padrão de tempo universal (UTC)
setUTCMinutes(minutos)	Acerta o valor dos minutos (de 0 a 59) guardados no objeto Date usando o padrão de tempo universal (UTC)
setUTCSeconds(segundos)	Acerta o valor dos segundos (de 0 a 59) guardados no objeto Date usando o padrão de tempo universal (UTC)
setUTCMilliseconds(milisegundos)	Acerta o valor dos milisegundos (de 0 a 999) guardados no objeto Date usando o padrão de tempo universal (UTC)
toGMTString()	Devolve uma representação textual (a data e a hora escritas por extenso) usando como referência o fuso horário do Tempo Médio de Greenwich (GMT)
toLocaleString()	Devolve uma representação textual (a data e a hora escritas por extenso) no fuso horário do computador local.
toUTCString()	Fornece uma representação textual (a data e a hora escritas por extenso) da data contida no objeto Date medida no padrão UTC.

9.2 Métodos estáticos do objeto Date

Método	Descrição
Date.parse(data_texto)	Devolve o número de milisegundos decorridos entre as zero horas do dia 1 de Janeiro de 1970 e a data indicada por *data_texto* (formato de texto)
Date.UTC(ano, mês, dia, horas, minutos, segundos, milisegundos)	Devolve o número de milisegundos que separam a data fornecida como argumento das 0 horas do dia 1 de Janeiro de 1970. É obrigatório indicar o ano, o mês e o dia. As horas, os minutos, os segundos e os milisegundos são facultativos.

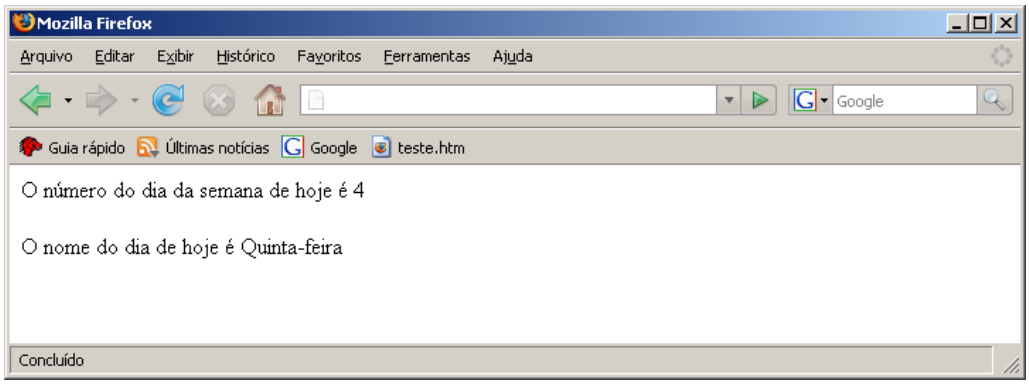
Nota: quando usar os métodos estáticos lembre-se que por serem estáticos eles se escrevem sempre na forma Date.parse() e Date.UTC(). Não tente colocar nomes de variáveis antes deles.

Exemplos de Aplicação

Mostrar o dia da semana

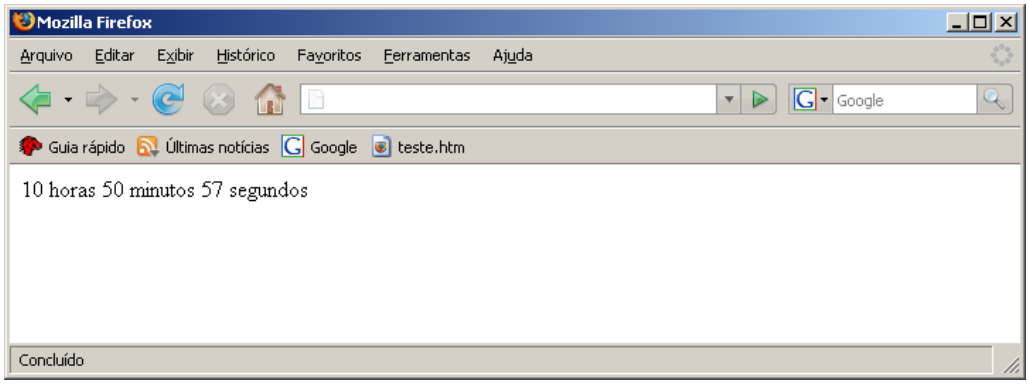
```
<html>
<head>
<title></title>
</head>
<body>
```

```
<script type="text/javascript">
<!--
   var d = new Date()
   var dia = d.getDay()
   document.write("O número do dia da semana de hoje é "+dia)

   // Para obtermos o nome do dia criamos um Array em que Domingo ocupa a
   // posição 0, segunda ocupa a posição 1, ...
   var nomesDias=new Array("Domingo","Segunda-feira","Terça-feira","Quarta-
   feira","Quinta-feira","Sexta-feira","Sábado")

   document.write("<br><br>O nome do dia de hoje é "+nomesDias[dia])
// -->
</script>
</body>
</html>
```



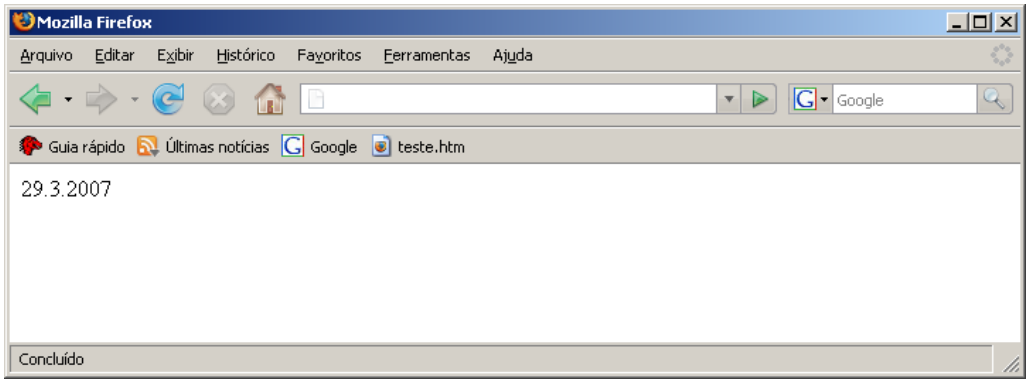
Obter a hora marcada pelo relógio do seu computador

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var d = new Date()
   document.write(d.getHours()+" horas ")
   document.write(d.getMinutes()+" minutos ")
   document.write(d.getSeconds()+" segundos")
// -->
</script>
</body>
</html>
```



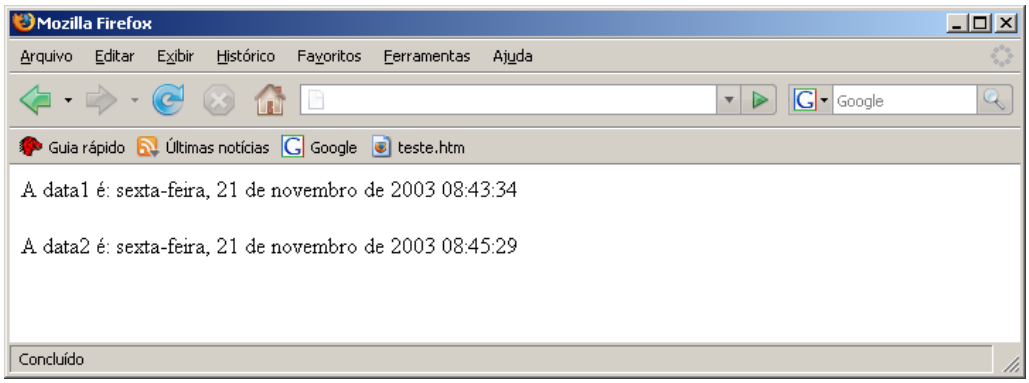
Obter o dia e o mês em que estamos

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var d = new Date()
   document.write(d.getDate())
   document.write(".")
   document.write(d.getMonth() + 1)
   document.write(".")
   document.write(d.getFullYear())
// -->
</script>
</body>
</html>
```



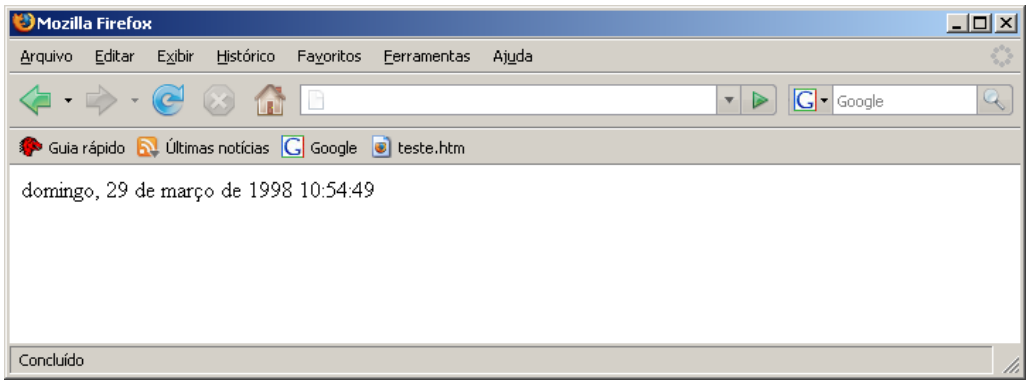
Criar uma data com um valor pré-definido

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var data1 = new Date("Fri, 21 Nov 2003 10:43:34 UTC")
   document.write("A data1 é: "+data1.toLocaleString())

   document.write("<br><br>")
   var data2 = new Date(1069411529550)
   document.write("A data2 é: "+data2.toLocaleString())
// -->
</script>
</body>
</html>
```



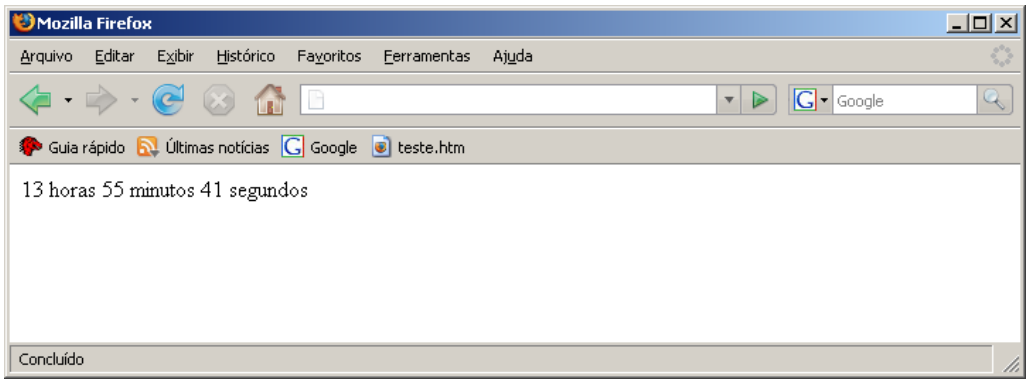
Estabelecer o ano para uma data

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var d = new Date()
   d.setFullYear("1998")
   document.write(d.toLocaleString())
// -->
</script>
</body>
</html>
```



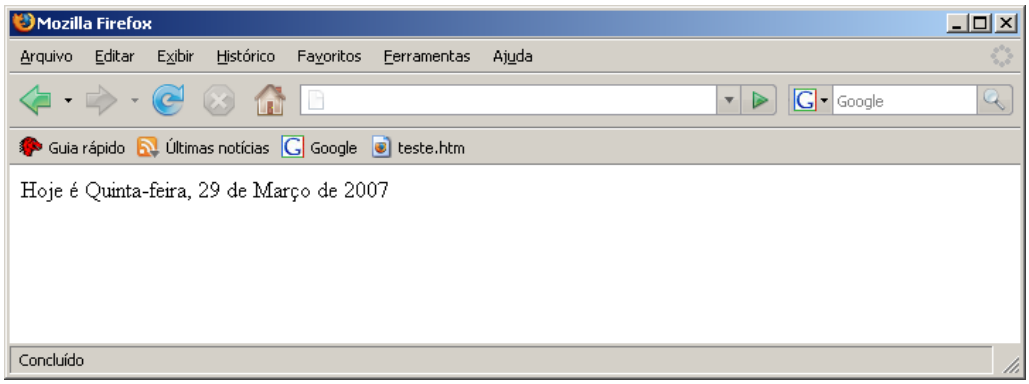
O padrão de tempo universal (UTC)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var d = new Date()
   document.write(d.getUTCHours()+" horas ")
   document.write(d.getUTCMinutes()+" minutos ")
   document.write(d.getUTCSeconds()+" segundos")
// -->
</script>
</body>
</html>
```



Escrever a data completa com os nomes do dia e do mês

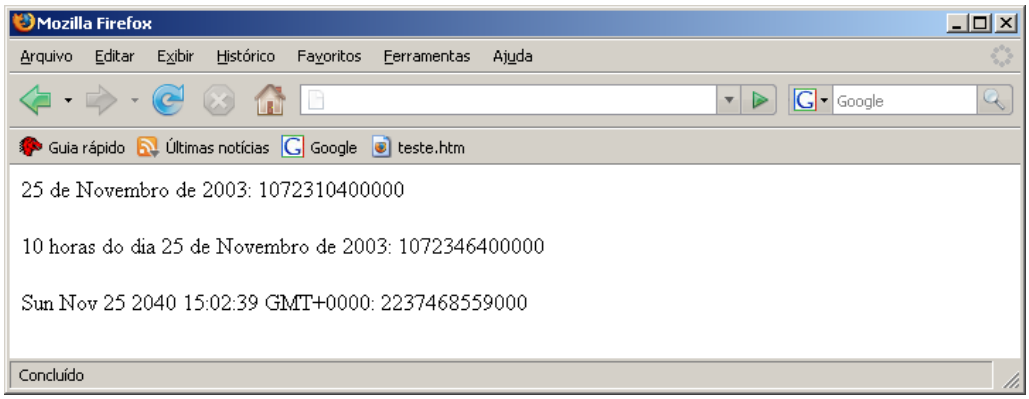
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var d = new Date()
   var dia = d.getDay()
   var mes = d.getMonth()
   var ano = d.getFullYear()

   // Para obtermos o nome do dia criamos um Array em que Domingo ocupa a
   // posição 0, segunda ocupa a posição 1, ...
   var nomesDias=new Array("Domingo","Segunda-feira","Terça-feira","Quarta-
   feira","Quinta-feira","Sexta-feira","Sábado")
   var nomesMeses=new Array("Janeiro","Fevereiro","Março","Abril","Maio",
   "Junho","Julho","Agosto","Setembro","Outubro","Novembro","Dezembro")

   var s="Hoje é "+nomesDias[dia]+", "+d.getDate()
   s+=" de "+nomesMeses[mes]+" de "+ano
   document.write(s)
// -->
</script>
</body>
</html>
```



Os métodos estáticos

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
```

```
<!--
   var s = "25 de Novembro de 2003: " + Date.UTC(2003,11,25)
   document.write(s+"<br><br>")

   var s = " 10 horas do dia 25 de Novembro de 2003: " + Date.UTC(2003,11,
   25,10)
   document.write(s+"<br><br>")

   var s = "Sun Nov 25 2040 15:02:39 GMT+0000: "
   s += Date.parse("Sun Nov 25 2040 15:02:39 GMT+0000")
   document.write(s+"<br><br>")
// -->
</script>
</body>
</html>
```



10. Objeto Math

O objeto Math é um objeto intrínseco do sistema JavaScript. Ele não deve ser criado explicitamente pelo programador porque é o próprio sistema que o cria automaticamente ao arrancar.

10.1 Propriedades do objeto Math

Propriedade	Descrição
E	Contém a base dos logaritmos naturais (número de Euler)
LN2	Contém o logaritmo natural de 2 (base E)
LN10	Contém o logaritmo natural de 10 (base E)
LOG2E	Contém o logaritmo de E na base 2
LOG10E	Contém o logaritmo de E na base 10
PI	Contém o número PI (3.1415927...)
SQRT1_2	Contém 1 a dividir pela raiz quadrada de 2
SQRT2	Contém a raiz quadrada de 2

10.2 Métodos do objeto Math

Todos os métodos do objeto Math aceitam um, dois ou nenhum números como argumentos e devolvem um número como resultado.

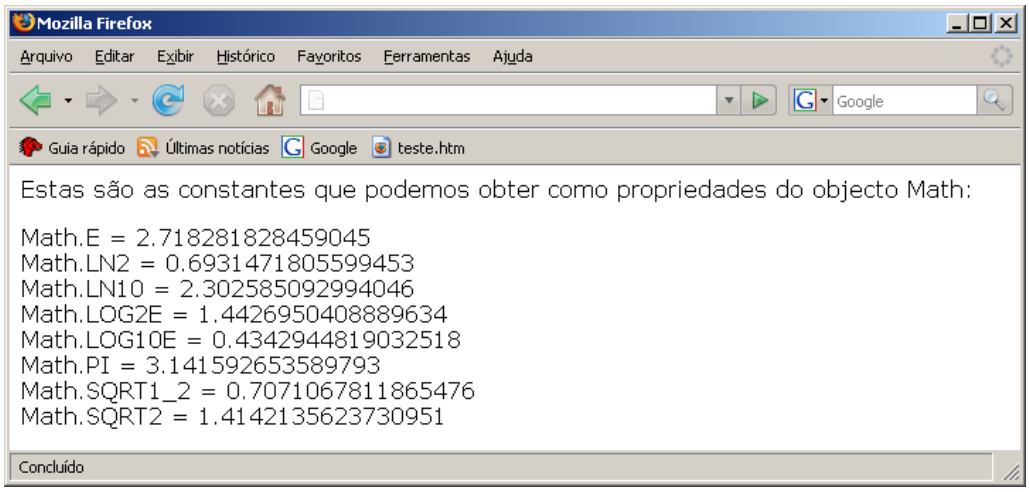
Método	Descrição
abs(x)	Devolve o valor absoluto de x
acos(x)	Devolve o valor do arco (radianos) cujo cosseno é x
asin(x)	Devolve o valor do arco (radianos) cujo seno é x
atan(x)	Devolve o valor do arco (radianos) cuja tangente é x
atan2(x, y)	Devolve o valor do ângulo formado pelo eixo dos xx com a linha

	que une a origem dos eixos ao ponto de coordenadas (x, y)
ceil(x)	Devolve o número inteiro mais próximo de x e não inferior a x
cos(x)	Devolve o cosseno de x
exp(x)	Devolve o valor da exponencial de x (E elevado à potência x)
floor(x)	Devolve o número inteiro mais próximo de x e não superior a x
log(x)	Devolve o logaritmo natural de x
max(x,y)	Devolve o maior dos números (x, y)
min(x,y)	Devolve o menor dos números (x, y)
pow(x,y)	Devolve o valor x elevado à potência y
random()	Devolve um número aleatório situado entre 0 e 1 (não aceita argumentos)
round(x)	Devolve o número inteiro mais próximo de x
sin(x)	Devolve o seno de x
sqrt(x)	Devolve a raiz quadrada de x
tan(x)	Devolve a tangente de x

Exemplos de Aplicação

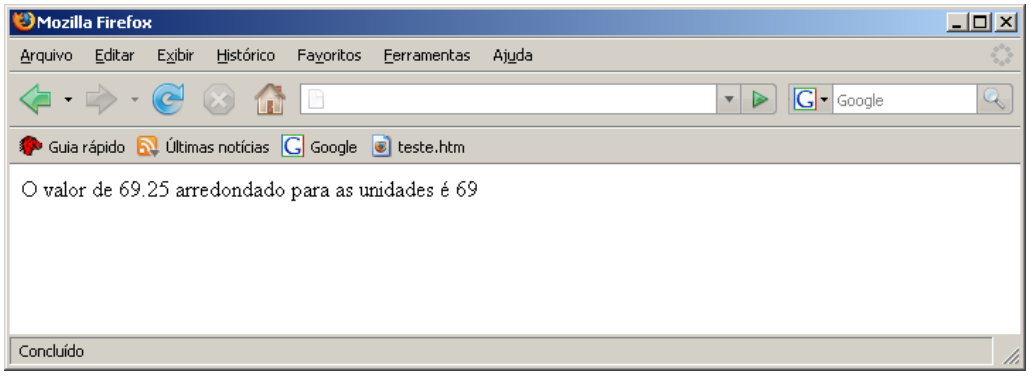
Ler os valores das constantes pré-definidas (propriedades)

```
<html>
<head>
<title></title>
</head>
<body style="font-family:verdana">
   <p>
      Estas são as constantes que podemos obter como propriedades do
      objeto Math:
   </p>
<script type="text/javascript">
<!--
   document.write("Math.E = "+Math.E+"<br>")
   document.write("Math.LN2 = "+Math.LN2+"<br>")
   document.write("Math.LN10 = "+Math.LN10+"<br>")
   document.write("Math.LOG2E = "+Math.LOG2E+"<br>")
   document.write("Math.LOG10E = "+Math.LOG10E+"<br>")
   document.write("Math.PI = "+Math.PI+"<br>")
   document.write("Math.SQRT1_2 = "+Math.SQRT1_2+"<br>")
   document.write("Math.SQRT2 = "+Math.SQRT2+"<br>")
// -->
</script>
</body>
</html>
```



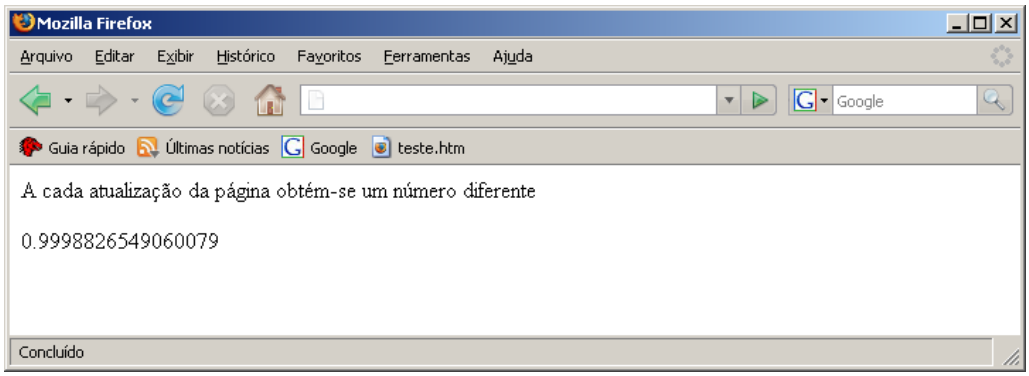
Arredondar um número para o valor inteiro mais próximo (método round())

```
<html>
<head>
<title></title>
</head>
<body>
   O valor de 69.25 arredondado para as unidades é
   <script type="text/javascript">
   <!--
      document.write(Math.round(69.25))
   // -->
   </script>
</body>
</html>
```



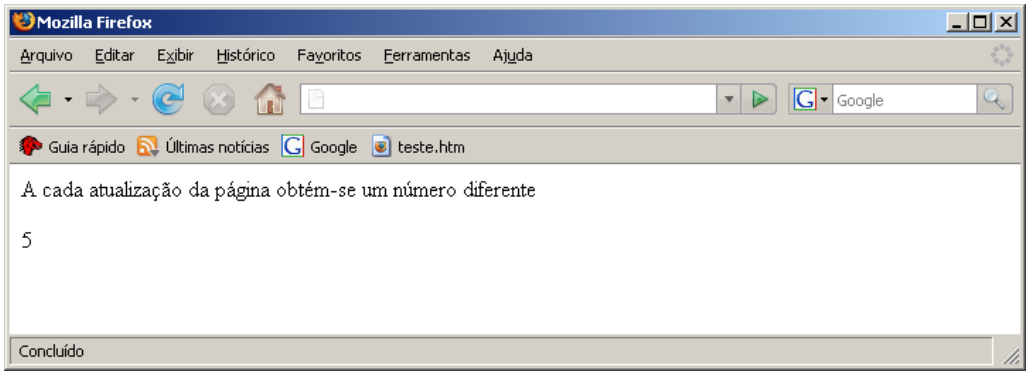
Números aleatórios distribuídos entre 0 e 1(método random())

```
<html>
<head>
<title></title>
</head>
<body>
   <p>
      A cada atualização da página obtém-se um número diferente
   </p>
   <script type="text/javascript">
   <!--
      document.write(Math.random())
   // -->
   </script>
</body>
</html>
```



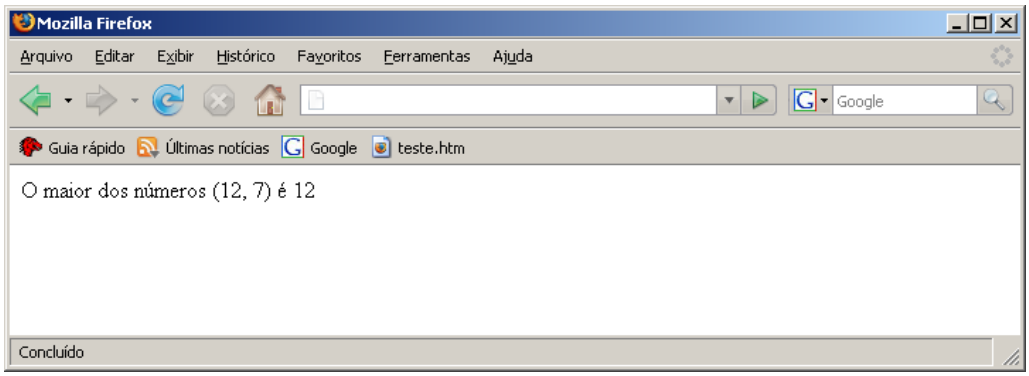
Gerar números aleatórios inteiros distribuídos entre 0 e 10

```
<html>
<head>
<title></title>
</head>
<body>
   <p>
      A cada atualização da página obtém-se um número diferente
   </p>
   <script type="text/javascript">
   <!--
      no=Math.random()*10
      document.write(Math.floor(no))
   // -->
   </script>
</body>
</html>
```



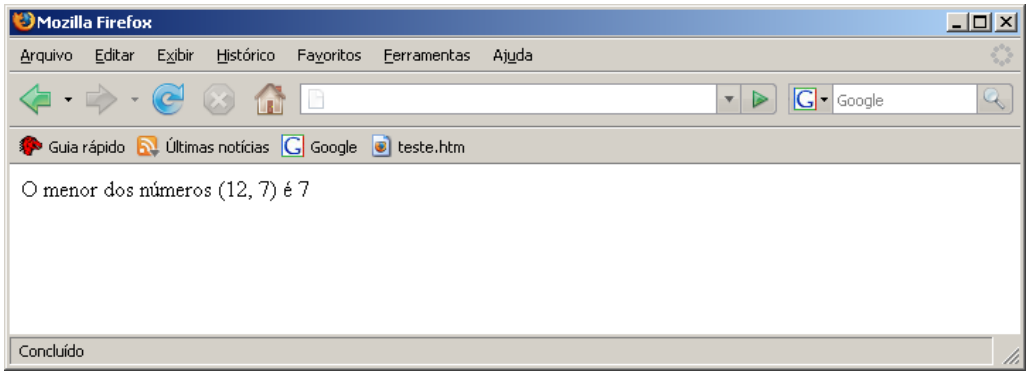
Escolher o maior de dois números (método max())

```
<html>
<head>
<title></title>
</head>
<body>
   O maior dos números (12, 7) é
   <script type="text/javascript">
   <!--
      document.write(Math.max(12,7))
   // -->
   </script>
</body>
</html>
```



Escolher o menor de dois números (método min())

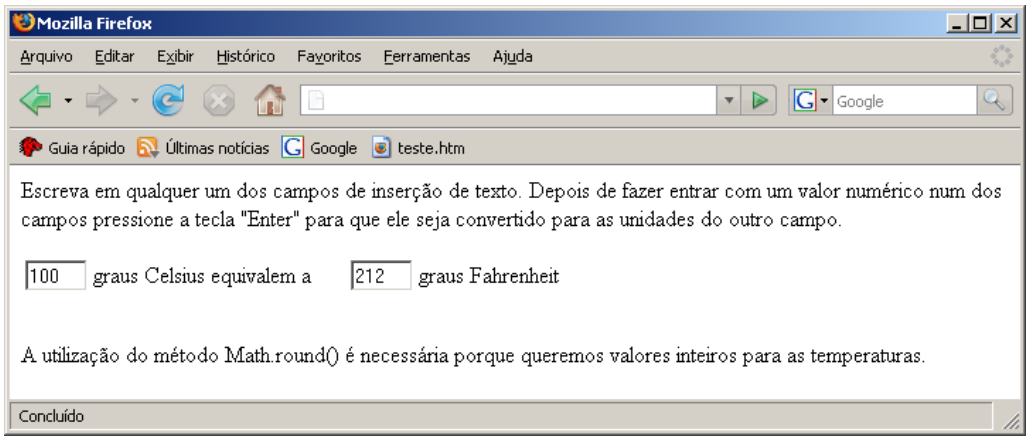
```
<html>
<head>
<title></title>
</head>
<body>
   O menor dos números (12, 7) é
   <script type="text/javascript">
   <!--
      document.write(Math.min(12,7))
   // -->
   </script>
</body>
</html>
```



Converter temperaturas de graus Celsius para Fahrenheit e vice versa

```
<html>
<head>
<script type="text/javascript">
<!--
   function celsius_para_Fahrenheit()
   {
      var F=parseInt(document.getElementById("celsius").value)*9/5+32
      document.getElementById("fahrenheit").value=Math.round(F)
   }

   function fahrenheit_para_Celsius()
   {
      var C=(parseInt(document.getElementById("fahrenheit").value) - 32)*5/9
       document.getElementById("celsius").value=Math.round(C)
   }
// -->
</script>
<title></title>
</head>
<body>
```

```
   <p>
      Escreva em qualquer um dos campos de inserção de texto.
      Depois de fazer entrar com um valor numérico num dos campos
      pressione a tecla "Enter" para que ele seja convertido
      para as unidades do outro campo.
   </p>
   <table>
      <tr>
         <td>
            <form action="javascript:celsius_para_Fahrenheit()">
               <input id="celsius" size="3"> graus Celsius equivalem a
            </form>
         </td>
         <td width="20"> </td>
         <td>
            <form action="javascript:fahrenheit_para_Celsius()">
               <input id="fahrenheit" size="3"> graus Fahrenheit
            </form>
         </td>
      </tr>
   </table>
   <p>
      A utilização do método Math.round() é necessária
      porque queremos valores inteiros para as temperaturas.
   </p>
</body>
</html>
```



11. Objeto String

Criação de um objeto String

```
// Uma string cujo conteúdo será definido mais tarde
var s = new String()

// Uma string cujo conteúdo é definido logo à partida
var s = new String("conteúdo da string")
```

O interpretador de JavaScript transforma automaticamente em objetos String todas as variáveis que tenham texto como conteúdo. Se uma variável contém texto ela é automaticamente um objeto String, mesmo que não tenha sido criada como tal.

11.1 Propriedades do objeto String

Propriedade	Descrição
length	Contém o número de caracteres que compõem a string

11.2 Métodos do objeto String

Método	Descrição
charAt(índice)	Devolve o caractere que ocupa a posição *índice* na string
charCodeAt(índice)	Devolve o código (conjunto Unicode) do caractere que ocupa a posição *índice* na string
indexOf(string_busca, índice_opcional)	Devolve a posição em que se encontra a primeira ocorrência de *string_busca* ou -1 se essa ocorrência não existir. Se não fornecermos um *índice_opcional* a busca inicia-se na posição zero, mas se o fornecermos é nesse ponto que se inicia a busca.
lastIndexOf(string_busca, índice_opcional)	Devolve a posição em que se encontra a última ocorrência de *string_busca* ou -1 se essa ocorrência não existir. A busca faz-se a partir do fim e caminha progressivamente para o início. Se não fornecermos um *índice_opcional* a busca inicia-se no fim, mas se o fornecermos é nesse ponto que se inicia a busca.
split(string_separador, limite_opcional)	Divide uma string em partes usando as ocorrências de *string_separador* como pontos de divisão. Devolve um array com todas as divisões (substrings) encontradas. Cada elemento do array é uma substring da string original. O *limite_opcional* indica o número máximo de partes a incluir no array que é devolvido. A *string_separador* é excluída das divisões e o objeto String sobre o qual foi invocado este método não sofre alterações.
substring(início, fim)	Devolve uma seção da string composta pelos caracteres que ocupam as posições com índices entre *início* (incluída) e *fim* (excluída).
toLowerCase()	Devolve uma versão da string com todos os caracteres convertidos para letra pequena
toUpperCase()	Devolve uma versão da string com todos os caracteres convertidos para letra grande

11.3 Métodos estáticos do objeto String

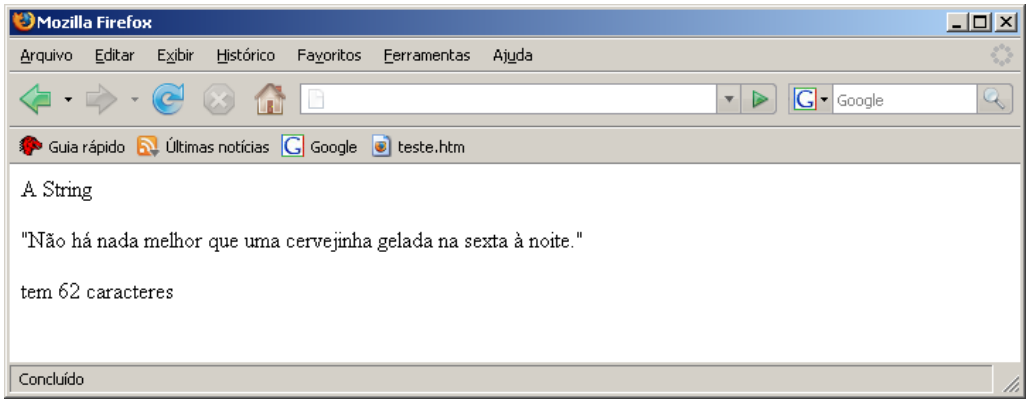
Método	Descrição
String.fromCharCode()	Devolve o caractere correspondente ao código Unicode fornecido

Este objeto é muito importante em diversas tarefas que podemos realizar em JavaScript: validar formulários, trabalhar com cookies, etc.

Exemplos de Aplicação

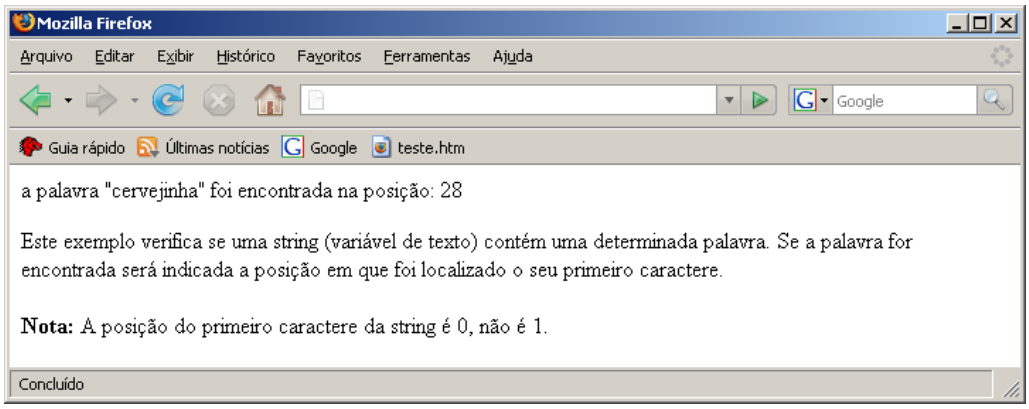
Contar o número de caracteres de uma String (propriedade length)

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var str="Não há nada melhor que uma cervejinha gelada na sexta à noite."
   document.write('A String <p> "'+ str + '"</p> tem ')
   document.write(str.length+" caracteres")
// -->
</script>
</body>
</html>
```



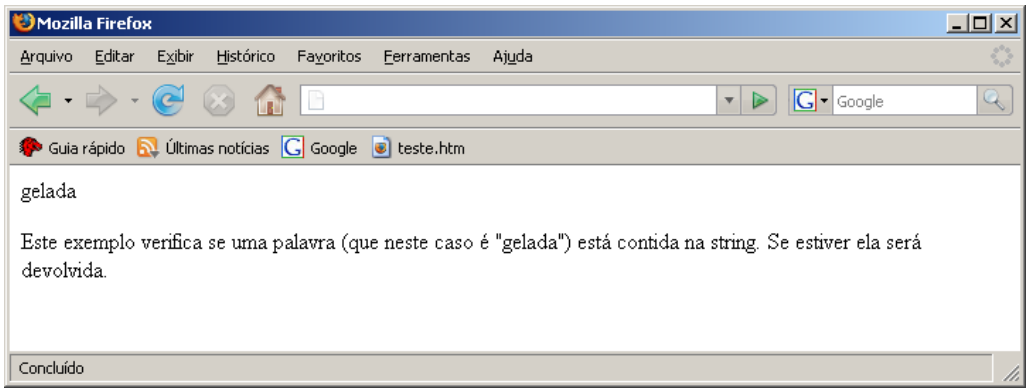
Localizar a ocorrência de uma seqüência de caracteres

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var str="Não há nada melhor que uma cervejinha gelada na sexta à noite."
   var pos=str.indexOf("cervejinha ")
   if (pos>=0)
   {
      document.write('a palavra "cervejinha" foi encontrada na posição: ')
      document.write(pos + "<br>")
   }
   else
   {
      document.write('a palavra "cervejinha" não foi encontrada!')
   }
// -->
</script>
   <p>Este exemplo verifica se uma string (variável de texto)
   contém uma determinada palavra. Se a palavra for
   encontrada será indicada a posição em que foi localizado o seu
   primeiro caractere.<br><br>
   <b>Nota:</b> A posição do primeiro caractere da string é 0, não é 1.
   </p>
</body>
</html>
```



Verificar se uma string contém um determinado caractere (método match())

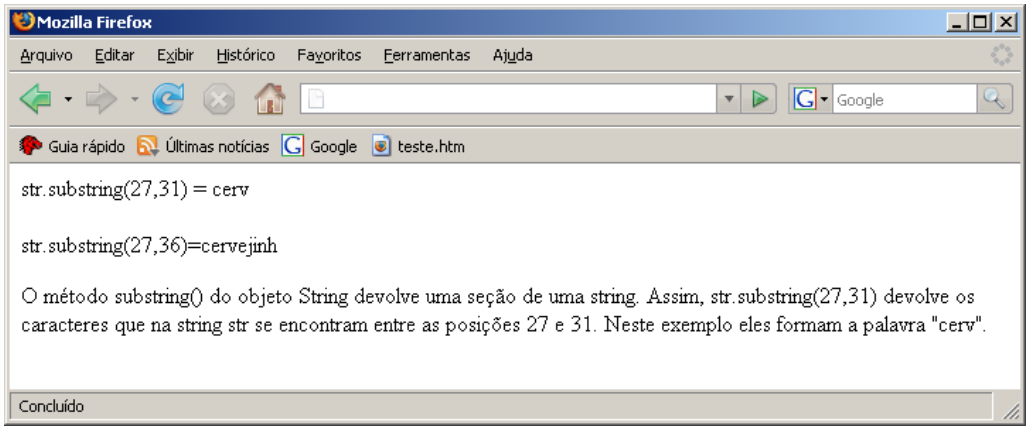
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var str="Não há nada melhor que uma cervejinha gelada na sexta à noite."
   document.write(str.match("gelada"))
// -->
</script>
   <p>Este exemplo verifica se uma palavra (que neste
   caso é "gelada") está contida na string. Se estiver
   ela será devolvida.</p>
</body>
</html>
```



Devolver uma seção de uma string (método substring())

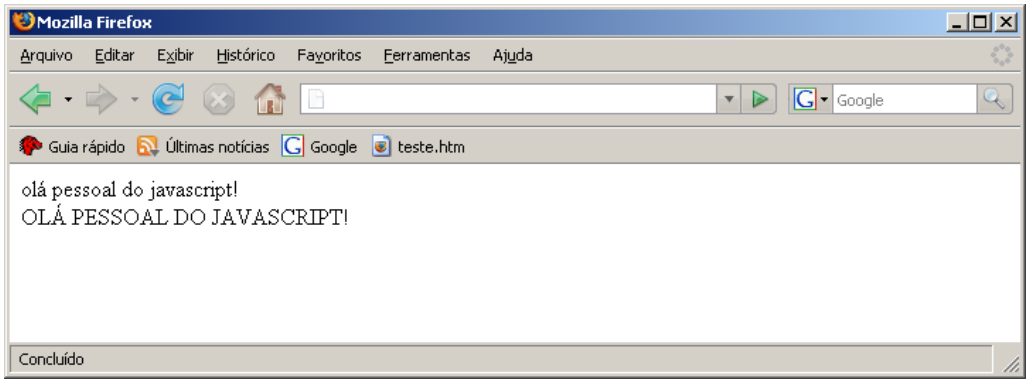
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var str="Não há nada melhor que uma cervejinha gelada na sexta à noite."
   document.write("str.substring(27,31) = "+str.substring(27,31))
   document.write("<br><br>")
   document.write("str.substring(27,36)="+str.substring(27,36))
// -->
</script>
   <p>O método substring() do objeto String devolve uma
   seção de uma string. Assim, str.substring(27,31) devolve
```

```
   os caracteres que na string str se encontram entre
   as posições 27 e 31. Neste exemplo eles formam
   a palavra "cerv".</p>
</body>
</html>
```



Converter em maiúsculas ou em minúsculas (métodos toLowerCase() e toUpperCase())

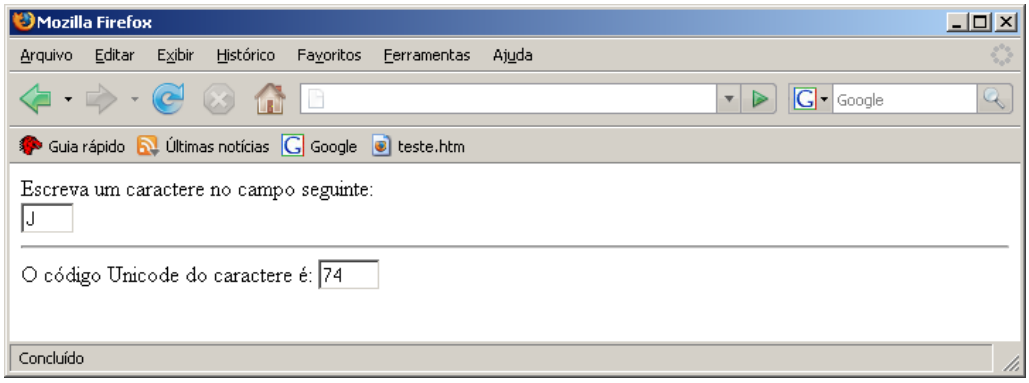
```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
<!--
   var str="Olá Pessoal do JavaScript!"
   document.write(str.toLowerCase())
   document.write("<br>")
   document.write(str.toUpperCase())
// -->
</script>
</body>
</html>
```



Obter o código unicode de um caractere

```
<html>
<head>
<script type="text/javascript">
<!--
   function toUnicode()
   {
      var str=document.getElementById("myInput").value
      if (str!="")
      {
         var unicode=str.charCodeAt(0)
```

```
            document.getElementById("unicode").value=unicode
         }
      }
// -->
</script>
<title></title>
</head>
<body>
   <form action="javascript:;">
      Escreva um caractere no campo seguinte:<br>
      <input maxlength="1" onkeyup='toUnicode()' type="text" size="2"
      id="myInput">
      <hr>
      O código Unicode do caractere é:
      <input size="3" type="text" id="unicode">
   </form>
</body>
</html>
```



PARTE III: HTML Dinâmico

Se já domina bem o HTML 4.01 (ou o XHTML 1), o JavaScript e os estilos CSS, então chegou o momento juntar o DOM ("Document Object Model") a estes ingredientes para enriquecer as suas páginas da Web. Esta combinação de tecnologias costuma ser designada por HTML Dinâmico, ou DHTML, e vai ajudá-lo a proporcionar experiências mais ricas aos usuários do seu website, tornando a sua tarefa de webmaster muito mais emocionante.

1. Para que serve o HTML Dinâmico?

O DHTML pode ser usado para criar menus avançados, painéis de texto dinâmicos, movimentar os elementos sobre a página, etc.

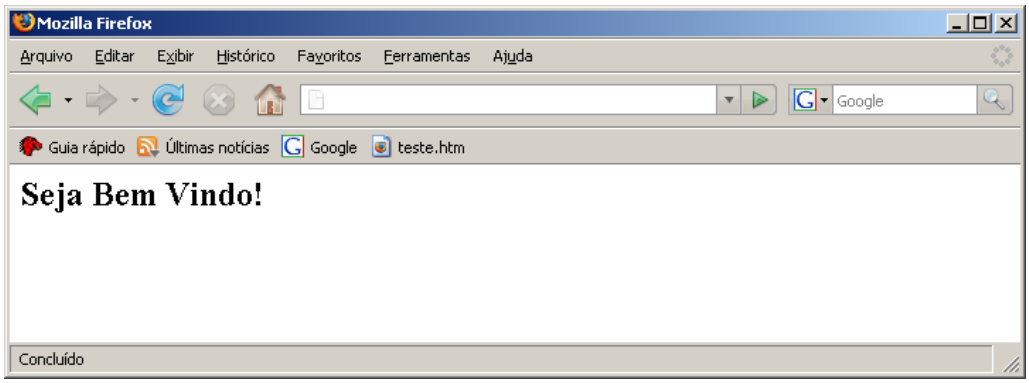
As aplicações mais úteis são aquelas que contribuem para dar mais eficiência à forma como um website transmite a informação que contém. Muitas delas são bastante elaboradas e não podem ser ilustradas de forma simples.

A lista seguinte apresenta alguns exemplos muito simples. Se dominarmos bem as técnicas básicas usadas nestes exemplos e se soubermos combiná-las umas com as outras seremos capazes de produzir efeitos bastante sofisticados e de melhorar a forma com os usuários interagem com os nossos websites.

Exemplos simples que pode modificar

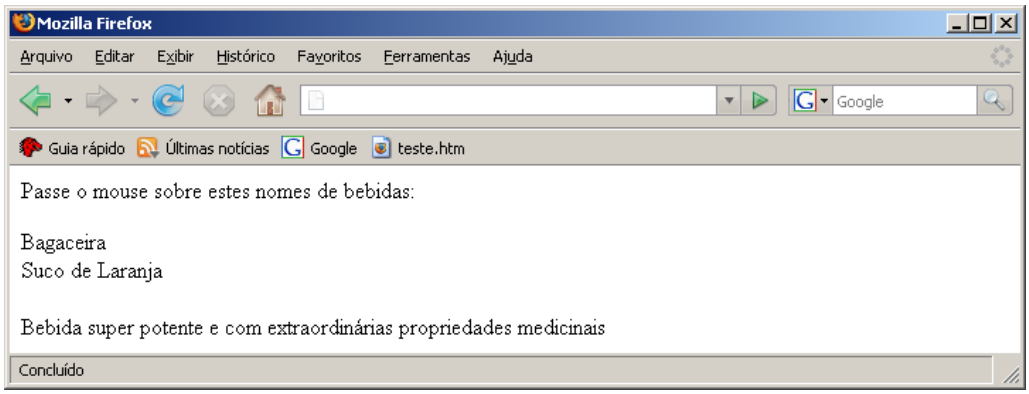
Modificar o conteúdo de um elemento

```
<html>
<head>
<script type="text/javascript">
   function nameon()
   {
      document.getElementById("textoh2").innerHTML="Seja Bem Vindo!"
   }
   function nameout()
   {
      document.getElementById("textoh2").innerHTML="Como você está?"
   }
</script>
<title></title>
</head>
<body>
   <h2 id="textoh2" onmouseover="nameon()" onmouseout="nameout()">Passe o
   mouse sobre este texto!</h2>
</body>
</html>
```



Apresentar uma dica

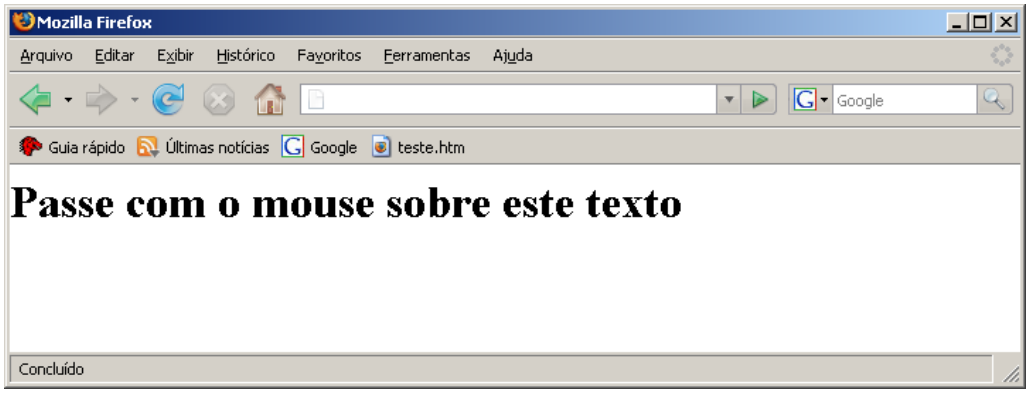
```
<html>
<head>
<script type="text/javascript">
   function gettip(txt)
   {
      document.getElementById("tip").innerHTML=txt
   }

   function reset()
   {
      document.getElementById("tip").innerHTML=""
   }
</script>
<title></title>
</head>
<body>
   <p>Passe o mouse sobre estes nomes de bebidas:</p>
   <span onmouseover="gettip('Bebida super potente e com extraordinárias
   propriedades medicinais')" onmouseout="reset()">Bagaceira</span><br>

   <span onmouseover="gettip('Bebida excelente quando muito diluída em
   Vodka')" onmouseout="reset()">Suco de Laranja</span><br><br>

   <div id="tip">
</body>
</html>
```



Alterar a posição de um elemento

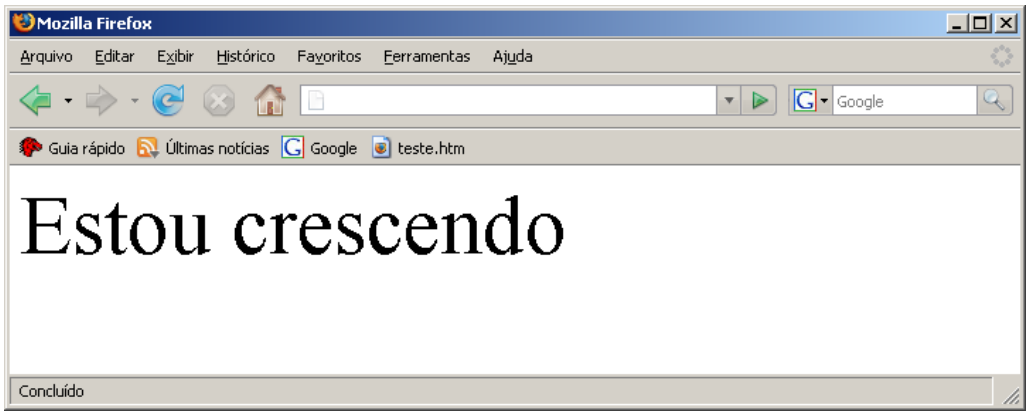
```
<html>
<head>
<script type="text/javascript">
   function moveleft()
   {
      document.getElementById("header").style.position="absolute"
```

```
      document.getElementById("header").style.left="0"
   }

   function moveback()
   {
      document.getElementById("header").style.position="relative"
   }
</script>
<title></title>
</head>
<body>
   <h1 id="header" onmouseover="moveleft()" onmouseout="moveback()">Passe com
   o mouse sobre este texto</h1>
</body>
</html>
```



Texto crescendo

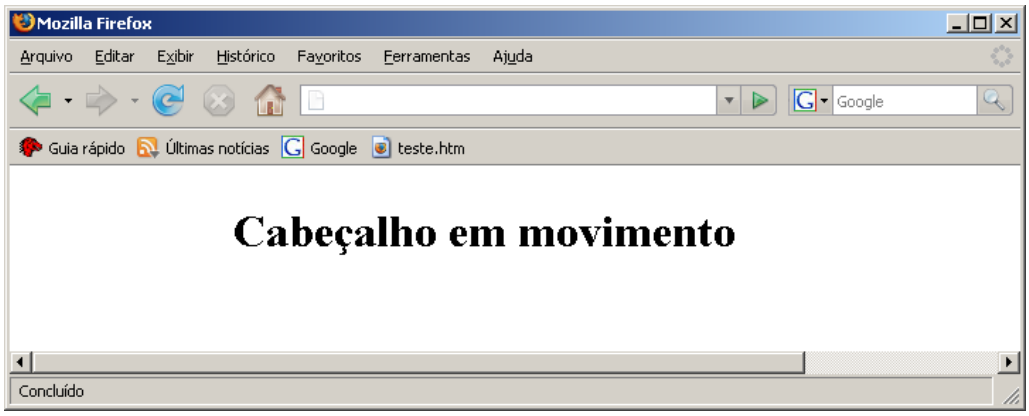
```
<html>
<head>
<script type="text/javascript">
   msgfield=null
   txtsize=0
   maxsize=60

   function writemsg()
   {
      if(msgfield==null)
         msgfield=document.getElementById("msg")
      if (txtsize<maxsize)
      {
         msgfield.style.fontSize=txtsize
         txtsize++
         timer=setTimeout("writemsg()",10)
      }
   }

   function stoptimer()
   {
      clearTimeout(timer)
   }
</script>
<title></title>
</head>
<body onload="writemsg()" onunload="stoptimer()">
   <p id="msg" style="FONT-SIZE: 2px">Estou crescendo</p>
</body>
</html>
```



Um efeito dinâmico

```
<html>
<head>
<script type="text/javascript">
<!--
   move=150
   var i=0
   var j=0
   var timer
   var header=null
   function moveheader()
   {
      if(header==null)
         header=document.getElementById("h1")
      header.style.position="relative"
      if (i<=move)
      {
         header.style.left=i
         i++
      }
      else
      {
         if (j<=move)
         {
            header.style.top=j
            j++
         }
         else
         {
            if (move>=0)
            {
               header.style.left=move
               header.style.top=move
               move-=1
            }
         }
      }
      timer=setTimeout("moveheader()",20)
   }

   function stoptimer()
   {
      clearTimeout(timer)
   }
-->
</script>
<title></title>
</head>
<body onload="moveheader()" onunload="stoptimer()">
   <h1 id="h1" style="LEFT:5px; POSITION:relative">Cabeçalho em movimento</h1>
</body>
```

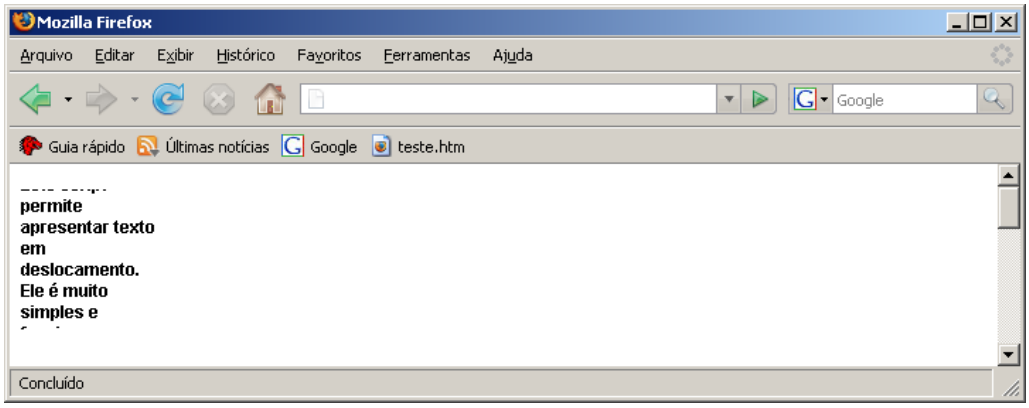
```
</html>
```



Quadro com texto em deslocamento

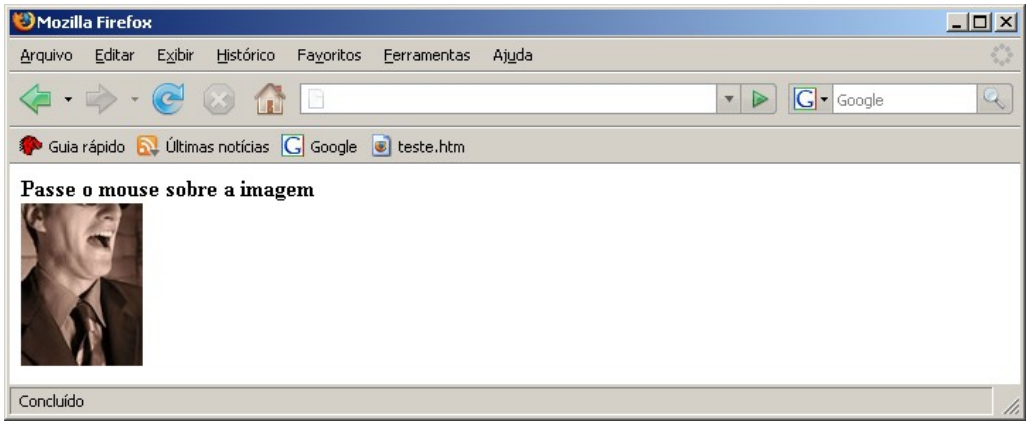
```
<html>
<head>
<style type="text/css">
span
      {
        font: 12px arial;
        position: absolute;
        width: 100px;
        height: 500px;
        top: 100px;
        clip:rect(0 100 100 0);
      }
</style>
<script type="text/javascript">
   var display=null
   var interval
   startPosition=0
   topPosition=17
   endPosition=100
   speed=60

   function scrollit()
   {
      if(display==null)
         display=document.getElementById("quadro")
      if (startPosition!=200)
      {
         startPosition=startPosition+1
         topPosition=topPosition-1
               display.style.clip="rect(" + (startPosition + 1) + " 100 " +
(startPosition + endPosition) + " 0)"
         display.style.top=topPosition
         interval=setTimeout("scrollit()",speed)
      }
      else
      {
         startPosition=0
         topPosition=100
         endPosition=100
         interval=setTimeout("scrollit()",speed)
      }
   }

   function stopinterval()
   {
      clearTimeout(interval)
   }
</script>
```

```
<title></title>
</head>
<body onload="scrollit()" onunload="stopinterval()">
      <span id="quadro" style="CLIP: rect(4px 100px 103px 0px); TOP: 17px">
      <br><br>
      <b>Este script permite apresentar texto em deslocamento.
      Ele é muito simples e funciona em todos os browsers relevantes.</b>
      </span>
</body>
</html>
```



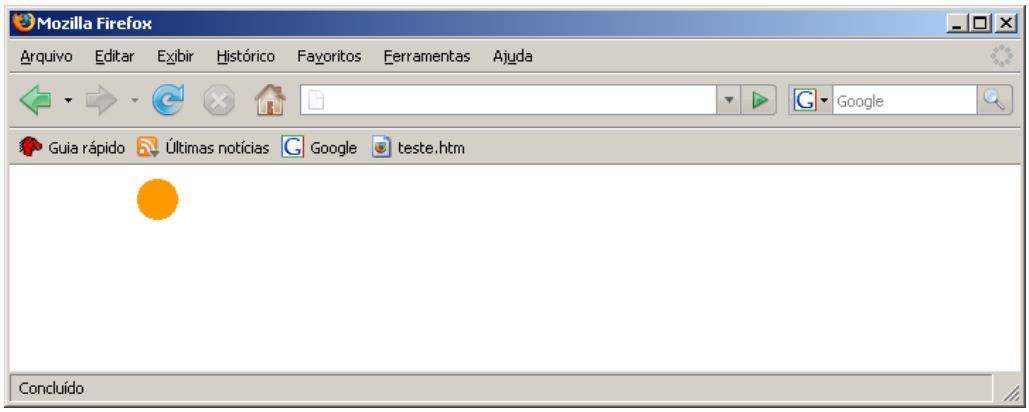
Trocar uma imagem por outra

```
<html>
<head>
<script type="text/javascript">
   var myImage
   function moveover()
   {
      foto=document.getElementById("imagem")
      foto.src="aberta.jpg"
   }

   function moveback()
   {
      foto.src="fechada.jpg"
   }
</script>
<title></title>
</head>
<body>
      <b>Passe o mouse sobre a imagem</b><br>
      <img id="imagem" onmouseover="moveover()" onmouseout="moveback()"
      height="115" src="fechada.jpg" width="86">
</body>
</html>
```



Dar movimento a uma imagem

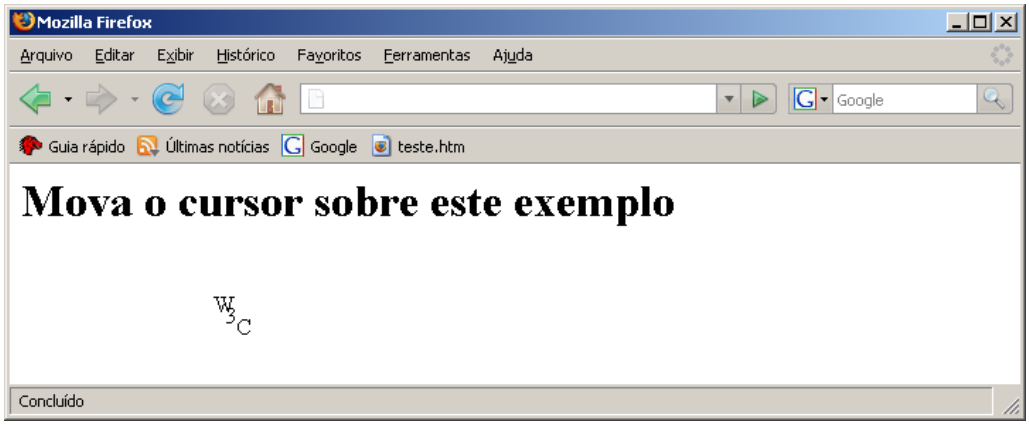
```
<html>
<head>
<script type="text/javascript">
   var i=1
   var myImage
   function starttimer()
   {
      myImage=document.getElementById("bola")
      nextMove()
   }

   function nextMove()
   {
      myImage.style.position="relative"
      myImage.style.left=+i+"px"
      i++
      timer=setTimeout("nextMove()",100)
   }

   function stoptimer()
   {
      clearTimeout(timer)
   }
</script>
<title></title>
</head>
<body onload="starttimer()" onunload="stoptimer()">
   <img id="bola" style="LEFT: 4px; POSITION: relative" height="32"
   src="bola.gif" width="32">
</body>
</html>
```



Um cursor que deixa rasto

```
<html>
<head>
<script type="text/javascript">
   var i=0
   var txt=new Array("rastoA","rastoB","rastoC")

   function cursor(intervalo, event)
   {
      pos=i*8+5
      if (intervalo=='primeiro')
         i=0
      if(i==0)
      {
         xpos=event.clientX
         ypos=event.clientY
```

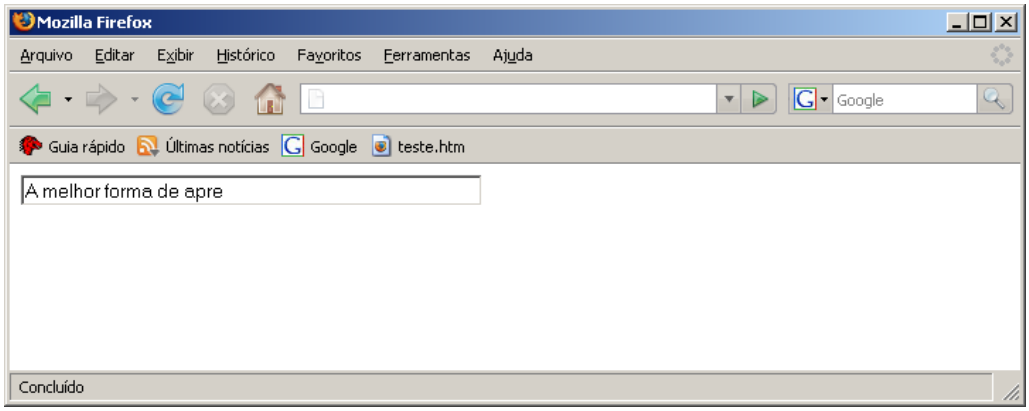
```
         document.getElementById(txt[i]).style.visibility="visible"
         document.getElementById(txt[i]).style.position="absolute"
         document.getElementById(txt[i]).style.left=xpos+5
         document.getElementById(txt[i]).style.top=ypos+5
      }
      else
      {
         document.getElementById(txt[i]).style.visibility="visible"
         document.getElementById(txt[i]).style.position="absolute"
         document.getElementById(txt[i]).style.left=xpos+pos
         document.getElementById(txt[i]).style.top=ypos+pos
      }
      i++
      if (i==txt.length)
         i-=1
      setTimeout("cursor('seguinte')",10)
   }
</script>
<title></title>
</head>
<body onmousemove="cursor('primeiro', event)">
      <h1>Mova o cursor sobre este exemplo</h1>
      <span id="rastoA" style="VISIBILITY: hidden">W</span>
      <span id="rastoB" style="VISIBILITY: hidden">3</span>
      <span id="rastoC" style="VISIBILITY: hidden">C</span>
      <br><br><br><br>
</body>
</html>
```



Apresentar mensagem tipo máquina de escrever

```
<html>
<head>
<script type="text/javascript">
   message="A melhor forma de aprender é com exemplos"
   pos=0
   maxlength=message.length+1

   function writemsg()
   {
      if (pos<maxlength)
      {
         txt=message.substring(pos,0)
         document.forms[0].msgfield.value=txt
         pos++
         timer=setTimeout("writemsg()", 100)
      }
   }
   function stoptimer()
   {
      clearTimeout(timer)
```

```
   }
</script>
<title></title>
</head>
<body onload="writemsg()" onunload="stoptimer()">
   <form>
      <input id="msgfield" size="50" value="T">
   </form>
</body>
</html>
```



O presente curso vai prepará-lo para aprender muito bem todos esses scripts que vamos construir brevemente. Esses novos exemplos vão aplicar estes conhecimentos para realizar tarefas muito específicas. Uma das primeiras será a criação fácil de menus de abrir eficazes e muito fáceis de preparar.

Outro exemplo daquilo que o DHTML lhe pode oferecer são as funcionalidades disponíveis neste curso. É graças ao DHTML que é possível incluir um número tão elevado de exemplos interativos e produzir as funcionalidades avançadas que os fazem funcionar da forma como se vê.

2. O DHTML ainda é pouco aproveitado. Porquê?

O DOM permite manipular todo o conteúdo de uma página recorrendo a scripts: alterar o conteúdo de um elemento, inserir novos elementos, remover elementos, alterar estilos CSS, etc. Os manipuladores de eventos do HTML permitem-nos escrever scripts que reagem às ações do usuário e respondem fazendo alterações ao conteúdo da página. Se soubermos usar bem estes recursos seremos capazes de criar páginas verdadeiramente dinâmicas que rapidamente estabelecem uma comunicação muito mais rica com os usuários.

Mas apesar deste grande potencial, a verdade é que até hoje a utilização do DHTML ainda está a um nível muito inferior ao que seria de esperar. A capacidade para facilitar e enriquecer a interação dos usuários com um website não foi suficiente para convencer os criadores de páginas a adotar o DHTML com entusiasmo. As razões principais para isto são três:

- Muitos criadores de páginas ainda não abandonaram completamente as práticas desaconselhadas a que foram obrigados a aderir no passado. Essas práticas eram usadas para compensar as fortes imperfeições dos web browsers.
- Apesar da qualidade dos web browsers atuais ser muito aceitável, ainda existem incompatibilidades entre eles, que apesar de serem fáceis de superar ainda assustam muita gente.

- Alguns usuários (muito poucos) ainda insistem na utilização de web browsers antigos que desrespeitam os padrões técnicos (HTML 4.01, CSS, DOM) e as práticas recomendadas.

3. Que ferramentas vamos usar?

Um bom web browser é tudo o que precisamos para fazer este curso. Pode ser usado o MSIE 6.0 ou superior, o Opera 7.2+, ou o Mozilla e seus derivados. Estes web browsers já contêm todos os objetos de que precisamos.

3.1 Os objetos do DOM

Os browsers modernos colocam ao dispor dos criadores de páginas um conjunto muito rico de objetos que permitem ler e manipular todo o conteúdo de uma página da Web.

As especificações técnicas recomendadas pelo W3C para este conjunto de objetos (designadas por DOM) definem formas para manipular documentos HTML através de scripts e especificam padrões adequados para escrever programas de software sofisticados capazes de manipular documentos XML com eficiência.

Porém, estas especificações são extremamente extensas e difíceis de implementar, deste modo, os criadores de web browsers decidiram implementar apenas aquelas que são mais relevantes para a interação dos scripts com as páginas da web. Se juntarmos a isto o fato de os browsers oferecerm outros objetos úteis que não pertencem às especificações do DOM notamos que, no momento atual, não podemos trabalhar com todos os objetos definidos no DOM e devemos usar outros que não pertencem ao DOM.

Temos assim que para o criador de páginas da Web a situação é a seguinte: nem todos os objetos definidos no DOM do W3C são relevantes para o DHTML, e há objetos que são úteis e estão disponíveis mas não pertencem ao DOM do W3C.

Por isso, as ferramentas que vamos usar no DHTML são compostas por objetos definidos no DOM do W3C e por objetos adicionais que são oferecidos pelos browsers. Esses objetos adicionais, que são poucos, apesar de não estarem definidos em padrões oficiais são suportados pelos browsers dominantes, o que faz deles padrões de fato. Recorde que aqui consideramos como dominantes os seguintes browsers: MSIE 6.0 e superior, Opera 7.2+, Mozilla e seus derivados.

Para que tudo fique muito claro aqui fica uma reafirmação do critério usado neste curso:

Vamos usar todos os recursos disponíveis que nos ofereçam garantias de bom funcionamento em todos os browsers importantes. Se existir uma recomendação do W3C aplicável ao caso que estamos tratando e suportada pelos browsers então essa será a nossa escolha. Se não existir recomendação mas houver uma alternativa segura então optaremos por essa alternativa.

3.2 Objetos principais usados em DHTML

Objeto	Contém	Descrição
window	métodos propriedades	O objeto window ocupa a posição de topo no DHTML. Ele contém outros objetos e informação acerca da janela do browser e do seu conteúdo.

document	métodos propriedades coleções	O objeto document pertence ao objeto window e representa o conteúdo da página HTML. Ele nos dá acesso aos elementos que definem a página e permite-nos controlar tudo o que está na página.
event	propriedades	O objeto event contém informação acerca dos acontecimentos que ocorrem numa página HTML, os quais resultam de ações do usuário.
history	propriedades métodos	O objeto history contém a história da navegação de uma janela do browser e permite regressar a páginas que já foram visitadas antes
location	propriedades métodos	O objeto location contém informação acerca da procedência de uma página
navigator	propriedades	O objeto navigator contém informação acerca do browser que está apresentando a página.
screen	propriedades	O objeto screen guarda informação acerca da tela em que a página está sendo visualizada

Os capítulos seguintes descrevem estes e outros objetos importantes para o HTML Dinâmico e apresentam muitos exemplos de aplicação.

Notas importantes sobre os objetos

Alguns dos métodos destes objetos aceitam argumentos que não podem ser descritos de forma simples, deste modo, se optou por não incluir os argumentos nestas descrições.

A melhor forma de compreender a forma como se usam os argumentos dos métodos mais complexos consiste em estudar e executar os numerosos exercícios de aplicação que são oferecidos.

A maioria das propriedades aqui descritas podem ser lidas e modificadas para alterar a forma como uma página é apresentada. Quando uma propriedade pode ser lida mas não pode ser modificada, ou pode ser modificada mas não pode ser lida, essa situação será indicada de forma explícita. Sempre que não for dada qualquer indicação assume-se que a propriedade pode ser lida e pode ser modificada.

Muitos elementos da linguagem HTML possuem atributos de apresentação (como por exemplo vspace e hspace nos elementos <img> e <applet>) que foram rejeitados nas versões modernas do HTML. Apesar do DOM continuar a suportar esses atributos fornecendo as propriedades correspondentes, o seu uso deve ser desencorajado, usando-se estilos CSS em seu lugar. Por esse motivo algumas dessas propriedades não são aqui mencionadas.

Resolução de incompatibilidades

Alguns métodos e propriedades dos objetos que estudaremos neste curso, apesar de não serem comuns a todos os browsers, oferecem-nos funcionalidades e informação úteis que não podemos obter com segurança de nenhum outro modo.

A sua utilização deve-se ao fato de sermos capazes de resolver facilmente as incompatibilidades recorrendo a funções de compatibilidade muito simples.

Nos casos em que o método/propriedade é de pouca utilidade, ou em que não é possível escrever uma função de compatibilidade simples que tire partido da sua funcionalidade, optamos por ignorar esse método ou propriedade. Num caso destes a

sua utilização faria com que o nosso código não funcionasse do mesmo modo em todos os browsers.

As funções de compatibilidade permitem-nos escrever scripts que funcionam corretamente em todos os browsers importantes. Com elas tudo acontece como se as diferenças não existissem.

Neste tutorial usaremos duas bibliotecas com funções de compatibilidade:

compat_window.js

```
function documentScrollTop(obj)
{
   var o=obj.document.body.scrollTop
   if(''+o)=='undefined')
      return o
   return obj.window.pageYOffset
}
```

compat_event.js

```
function evento_FromTo(event)
{
   /*
   Esta função devolve uma lista com dois itens em que o primeiro
   item é o elemento que acabou de ser abandonado pelo mouse e o
   segundo item é o elemento para o qual o mouse se dirigiu.

   Esta função pode ser usada com os eventos
   onmouseover e onmouseout
   */
   if(''+event.fromElement)=='undefined')
   {
      if(''+event.type)=='mouseover')
         return new Array(event.relatedTarget,event.target)
      return new Array(event.target,event.relatedTarget)
   }
   return new Array(event.fromElement,event.toElement)
}
```

4. O objeto window

O objeto window é o objeto de topo no DOM do DHTML. Ele contém informação acerca da janela do browser e do seu conteúdo: tamanho, posição, documento, molduras (frames).

As tabelas seguintes mostram listas com as propriedades e os métodos mais importantes deste objeto. Mais abaixo você encontrará diversos exemplos que ilustram a forma como alguns destes métodos e propriedades são usados na prática.

Propriedade	Descrição
closed	Indica (true ou false) se a janela está fechada ou não
defaultStatus	Lê ou define a mensagem que por omissão aparece escrita na barra de status do browser
document	Devolve o objeto document, que representa o documento que está sendo apresentado na janela.
event	Devolve o objeto event, que descreve o último acontecimento que ocorreu na janela.
history	Objeto que contém uma lista com os endereços (URLs) das páginas já visitadas na presente sessão e permite regressar

	a elas
length	Lê a quantidade de molduras (frames) que estão na janela
location	Contém informação acerca do endereço (URL) da página que está sendo apresentada e permite alterá-lo
name	Lê ou define o nome da janela
navigator	Devolve o objeto navigator
opener	Devolve (se existir) o objeto que abriu a janela (só em janelas popup)
outerHeight	(Só Mozilla/Netscape e Opera) Contém a altura exterior da janela do browser
outerWidth	(Só Mozilla/Netscape e Opera) Contém a largura exterior da janela do browser
pageXOffset	(Só Mozilla/Netscape e Opera) Contém a coordenada x do canto superior esquerdo da janela
pageYOffset	(Só Mozilla/Netscape e Opera) Contém a coordenada y do canto superior esquerdo da janela
parent	Caso a página esteja numa moldura (frame ou iframe) devolve o objeto window que a contém
returnValue	Lê o valor devolvido por uma caixa de diálogo
screen	Devolve um objeto que contém informação acerca da tela utilizada para ver a página
screenLeft	(Só MSIE) Contém a coordenada x do canto superior esquerdo da área da janela que contém a página (difere de pageXOffset)
screenTop	(Só MSIE) Contém a coordenada y do canto superior esquerdo da área da janela que contém a página (difere de pageYOffset)
self	Devolve o objeto window em que a página se encontra
status	Lê ou define a mensagem que aparece escrita na barra de estado
top	Devolve o objeto window que ocupa a posição mais elevada na hierarquia de molduras

Nota: Algumas das propriedades do objeto window são próprios objetos.

Coleções do objeto window

Coleção	Descrição
frames	Contém uma lista com todos os objetos que representam molduras que foram criadas usando os elementos <frame> ou <iframe>. Os objetos desta lista são próprios objetos window (a cada moldura corresponde uma janela.)

Métodos do objeto window

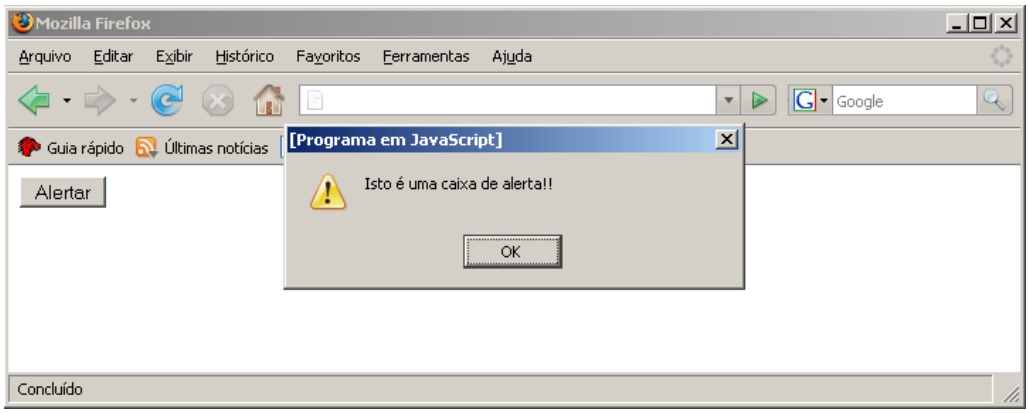
Método	Descrição
alert()	Mostra uma caixa com uma mensagem de texto e um botão OK.
blur()	A janela perde o foco.
clearInterval()	Anula uma ação do método setInterval().
clearTimeout()	Anula uma ação do método setTimeout().
close()	Fecha a janela.
confirm()	Mostra uma caixa com uma mensagem de texto, um botão para Cancelar e um botão para aceitar (OK).

escape()	Codifica uma string (texto) para que ela possa ser enviada como parte do URL ou guardada num cookie
focus()	Dá o foco a uma janela
moveBy()	Desloca a janela relativamente à sua posição atual.
moveTo()	Move a janela para a posição especificada.
open()	Abre uma nova janela (popup).
print()	Imprime o documento que se encontra na janela.
prompt()	Mostra uma caixa com uma pergunta e um campo para inserir uma resposta.
resizeBy()	Altera o tamanho da janela somando aos valores atuais os valores x e y especificados.
resizeTo()	Altera o tamanho da janela para os valores x e y especificados.
scrollBy()	Desloca o início da parte visível da janela somando os valores x, y às coordenadas da parte que está atualmente no início da parte visível.
scrollTo()	Faz com que a parte visível do conteúdo da janela se inicie na parte da página que tem as coordenadas (x, y) especificadas.
setInterval()	Executa a função indicada a intervalos regulares (medidos em milisegundos)
setTimeout()	Executa a função indicada uma só vez depois de passado algum tempo (medido em milisegundos)
unescape()	Descodifica uma string (texto) que estava preparada para ser enviada como parte do URL colocando-a na forma normal. Efetua a operação inversa do método escape().

Exemplos de Aplicação

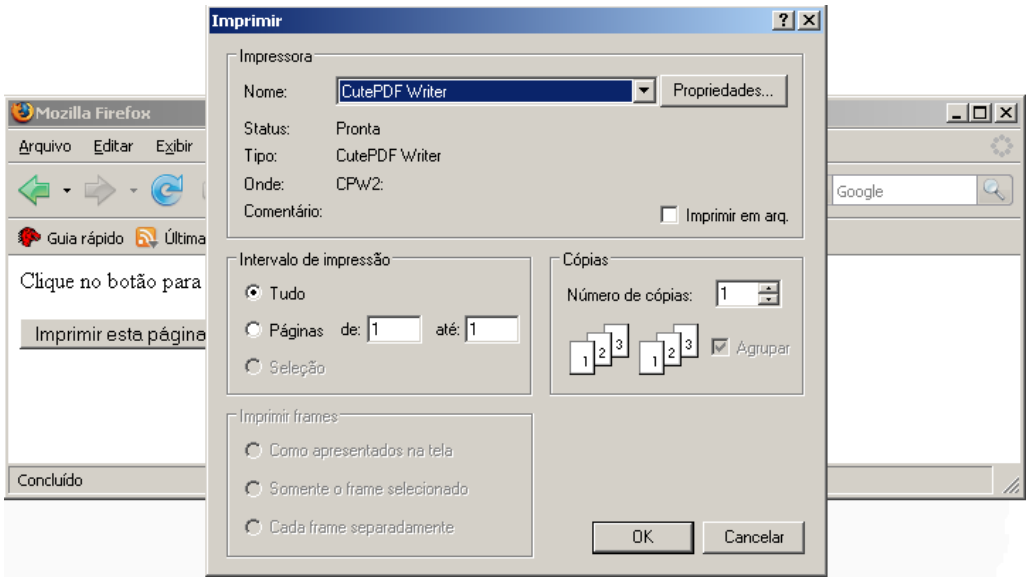
Apresentar uma caixa de alerta

```
<html>
<head>
<script type="text/javascript">
   function salert()
   {
      alert("Isto é uma caixa de alerta!!")
   }
</script>
<title></title>
</head>
<body>
   <form>
      <input onclick="salert()" type="button" value="Alertar">
   </form>
</body>
</html>
```



Imprimir uma página

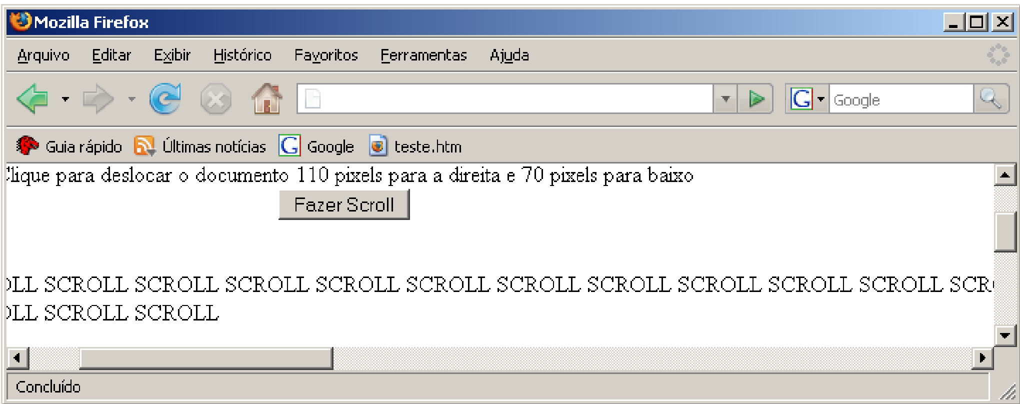
```
<html>
<head>
<script type="text/javascript">
   function printPage()
   {
      window.print()
   }
</script>
<title></title>
</head>
<body>
   <p>Clique no botão para imprimir esta belíssima página.</p>
   <form action="javascript:;">
      <input onclick="printPage()" type="button" value="Imprimir esta
      página">
   </form>
</body>
</html>
```



Deslocar (scroll) o documento dentro da janela relativamente à posição atual

```
<html>
<head>
<script type="text/javascript">
   function scrollWindow(x,y)
   {
      window.scrollBy(x,y)
```

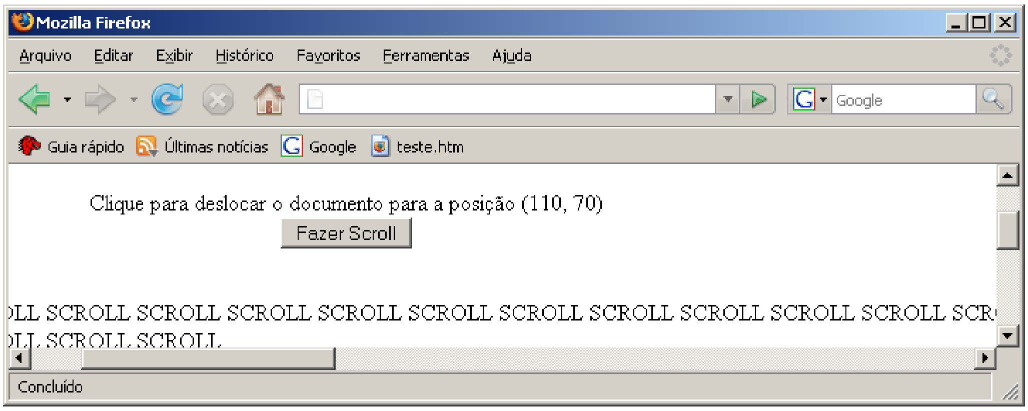
```
   }
</script>
    <title></title>
</head>
<body>
   <div style="width: 2200px">
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   </div><br>
   <form style="text-align:center" action="javascript:;">
      Clique para deslocar o documento 110 pixels para a direita e 70 pixels
      para baixo<br>
      <input onclick="scrollWindow(110,70)" type="button" value="Fazer
      Scroll">
   </form><br>
   <div style="width: 2200px">
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   </div><br><br>
   <div style="width: 2200px">
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   </div><br><br>
   <div style="width: 2200px">
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   </div><br><br>
</body>
</html>
```



Deslocar (scroll) o documento dentro da janela para uma posição absoluta

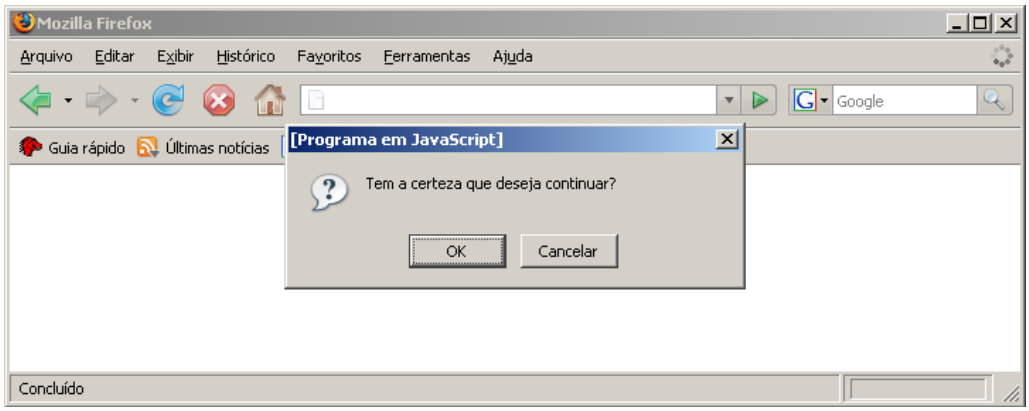
```
<html>
<head>
<script type="text/javascript">
   function scrollWindow(x,y)
   {
      window.scrollTo(x,y)
   }
</script>
title></title>
</head>
<body>
   <div style="width: 2200px">
```

```
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   </div><br>
   <form style="text-align:center" action="javascript:;">
      Clique para deslocar o documento para a posição (110, 70)<br>
      <input onclick="scrollWindow(110,70)" type="button" value="Fazer
      Scroll">
   </form><br>
   <div style="width: 2200px">
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   </div><br><br>
   <div style="width: 2200px">
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   </div><br><br>
   <div style="width: 2200px">
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL
   </div><br><br>
</body>
</html>
```



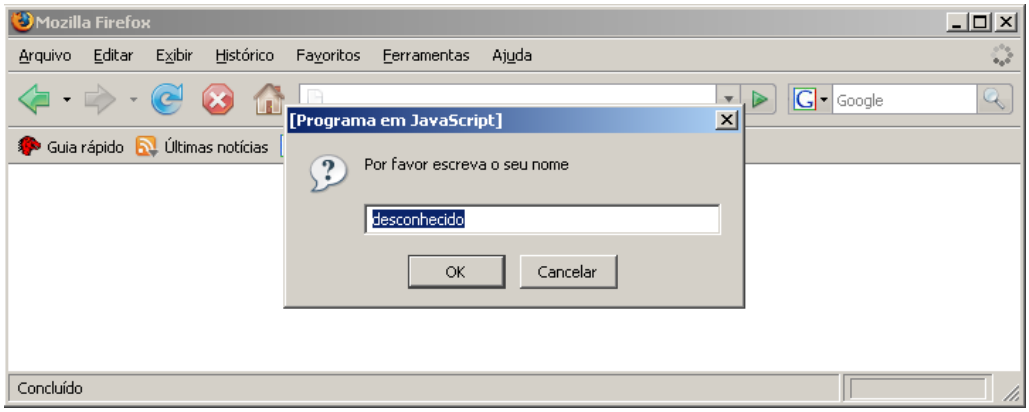
Pedir uma confirmação

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
   resposta=confirm("Tem a certeza que deseja continuar?")
   if(resposta==true)
      document.write("Você respondeu OK (quer continuar.)")
   else
      document.write("Você cancelou (não quer continuar.)")
</script>
</body>
</html>
```



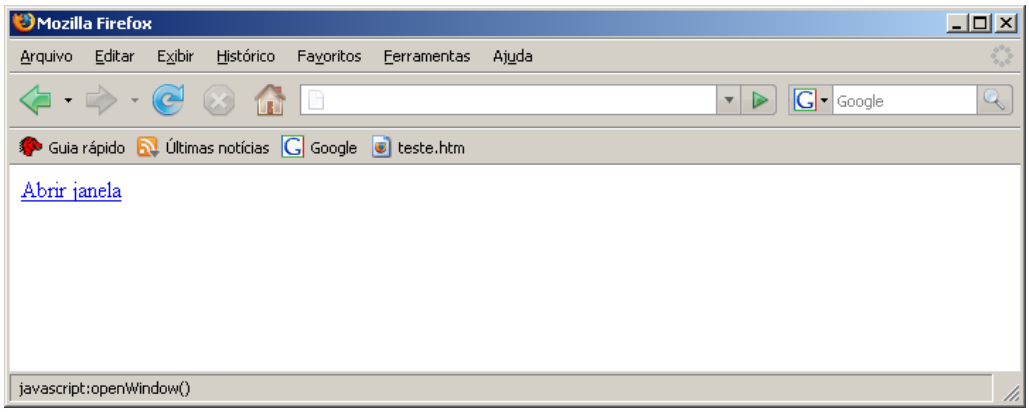
Pedir informação ao usuário

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
   var nome = prompt("Por favor escreva o seu nome","desconhecido")
   if(nome == null || nome == "")
      nome="sem nome"
   document.write("Olá " + nome)
</script>
</body>
</html>
```



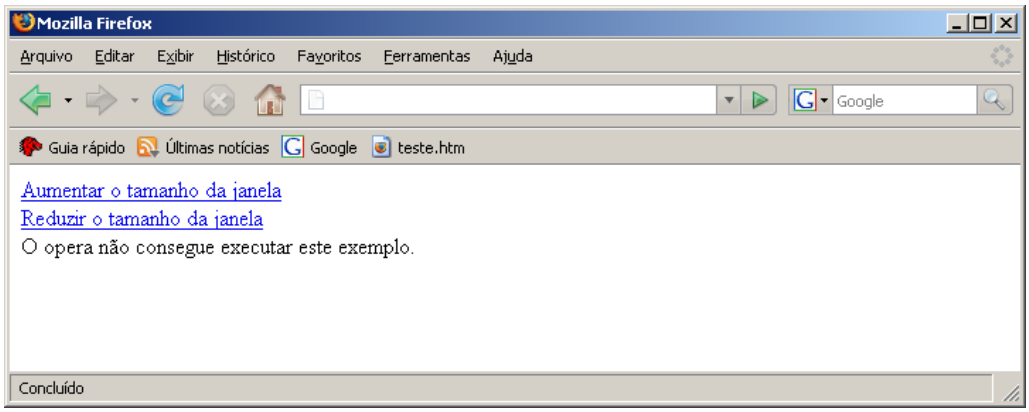
Abrir uma janela

```
<html>
<head>
<script type="text/javascript">
   function openWindow()
   {
      window.open("pagina.html")
   }
</script>
<title></title>
</head>
<body>
   <a href="javascript:openWindow()">Abrir janela</a>
</body>
</html>
```



Aumentar ou reduzir as dimensões de uma janela

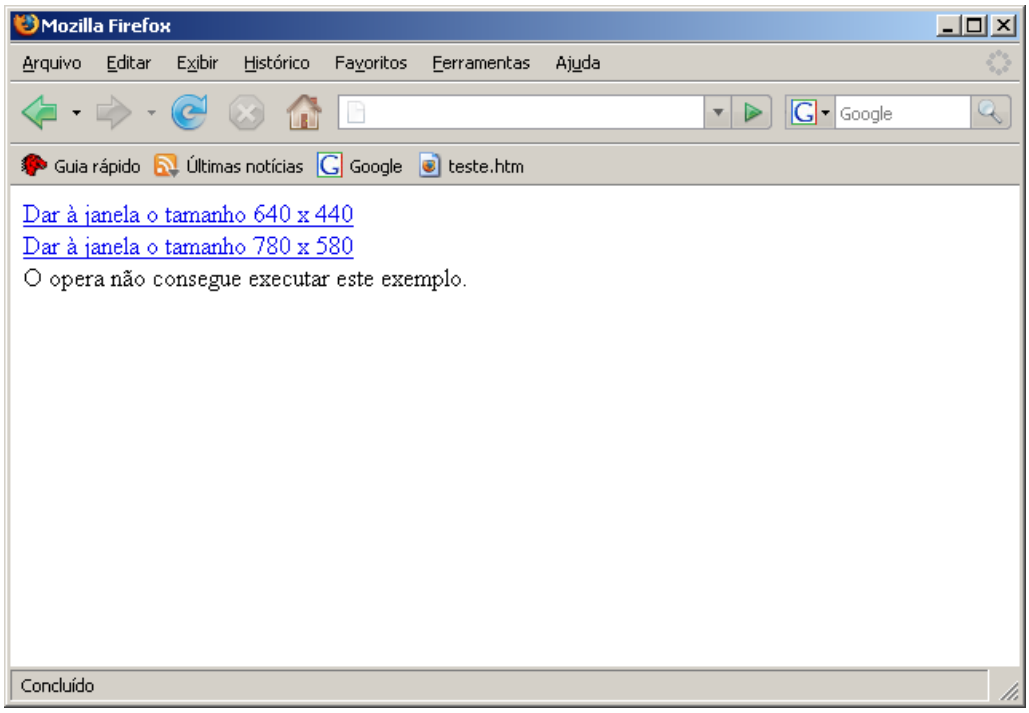
```
<html>
<head>
<script type="text/javascript">
   function resizeWindow(x,y)
   {
      window.resizeBy(x,y)
   }
</script>
<title></title>
</head>
<body>
   <a href="javascript: resizeWindow(40,20)">
   Aumentar o tamanho da janela</a><br>
   <a href="javascript: resizeWindow(-40,-20)">
   Reduzir o tamanho da janela</a><br>
   O opera não consegue executar este exemplo.
</body>
</html>
```



Dar um determinado tamanho a uma janela

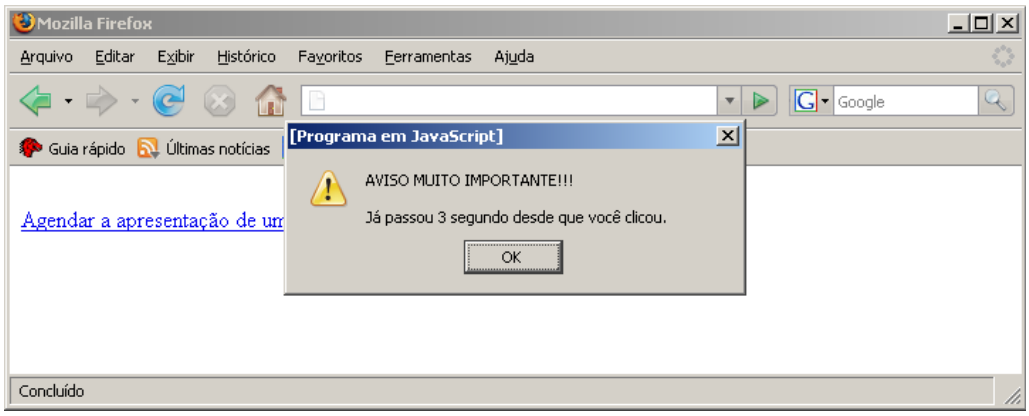
```
<html>
<head>
<script type="text/javascript">
   function resizeWindow(x,y)
   {
       window.resizeTo(x,y)
   }
</script>
<title></title>
</head>
<body>
   <a href="javascript: resizeWindow(640,440)">
```

```
   Dar à janela o tamanho 640 x 440</a><br>
   <a href="javascript: resizeWindow(780,580)">
   Dar à janela o tamanho 780 x 580</a><br>
   O opera não consegue executar este exemplo.
</body>
</html>
```



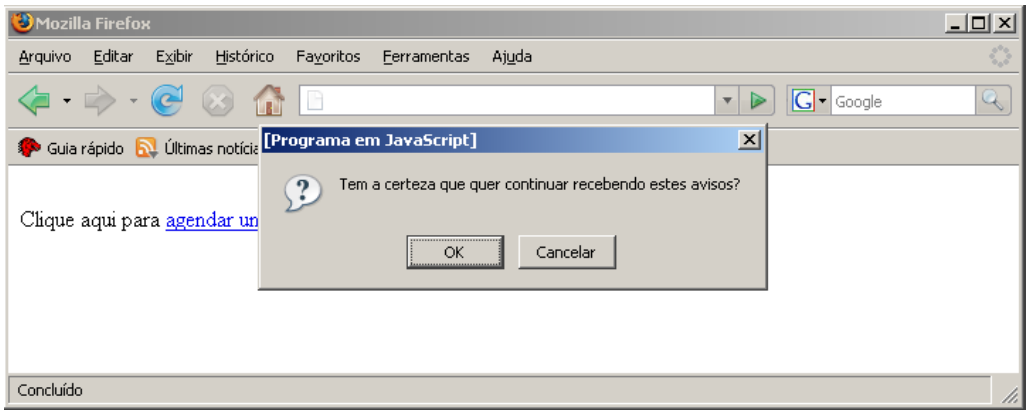
Agendar a execução de uma função

```
<html>
<head>
<script type="text/javascript">
   function avisar()
   {
      alert('AVISO MUITO IMPORTANTE!!!\n\nJá passou 3 segundo desde que você
      clicou.')
   }
   function agendar()
   {
      setTimeout("avisar()", 3000)
   }
</script>
<title></title>
</head>
<body>
   <br><a href="javascript:agendar()">Agendar a apresentação de um aviso</a>
</body>
</html>
```



Agendar a execução repetida de uma função

```
<html>
<head>
<script type="text/javascript">
   var interval
   function avisar()
   {
      var resposta=confirm("Tem a certeza que quer continuar recebendo estes
      avisos?")
      if(resposta==false)
         clearInterval(interval)
   }
   function agendar()
   {
      interval=setInterval("avisar()",2000)
   }
</script>
<title></title>
</head>
<body>
   <br>Clique aqui para <a href="javascript:agendar()">
   agendar uma apresentação repetida de um aviso</a>
</body>
</html>
```



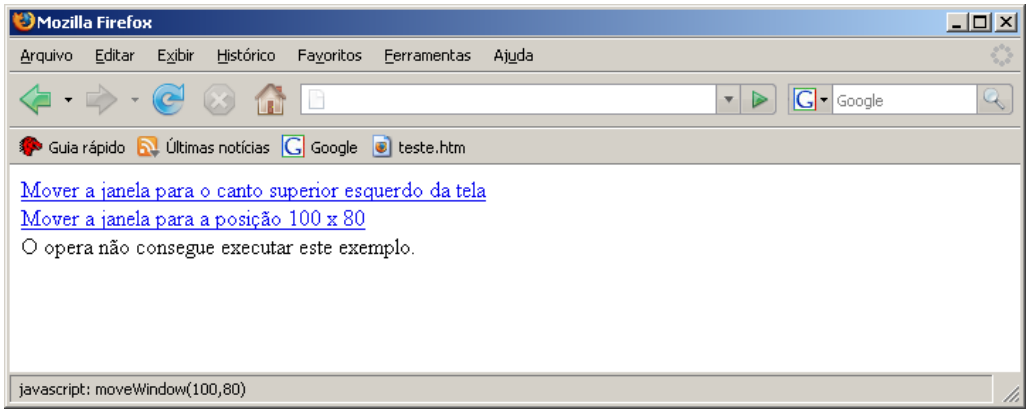
Deslocar uma janela relativamente à posição que ocupa

```
<html>
<head>
<script type="text/javascript">
   function moveWindow(x,y)
   {
      top.window.moveBy(x,y)
   }
```

```
</script>
<title></title>
</head>
<body>
   <a href="javascript: moveWindow(20,10)">
   Deslocar a janela 20 pixels para a direita e 10 pixels para baixo</a><br>
   <a href="javascript: moveWindow(-20,-10)">
   Deslocar a janela 20 pixels para a esquerda e 10 pixels para cima</a><br>
   O opera não consegue executar este exemplo.
</body>
</html>
```

Mover uma janela para uma posição absoluta

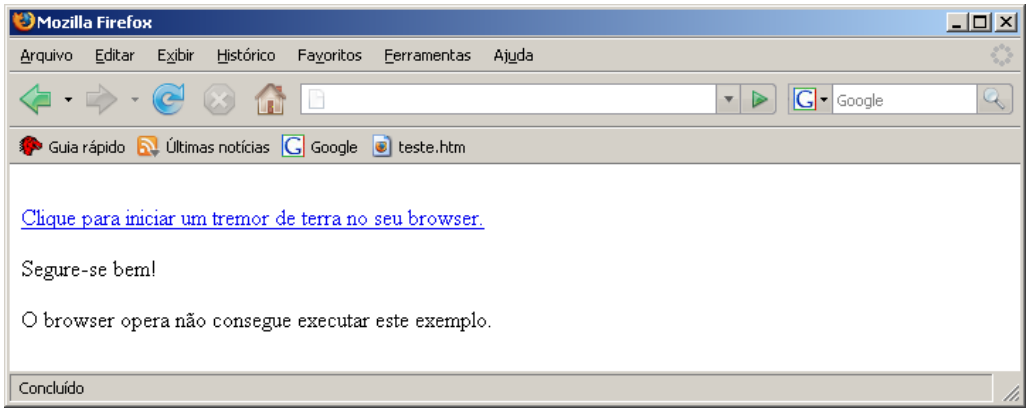
```
<html>
<head>
<script type="text/javascript">
   function moveWindow(x,y)
   {
      top.window.moveTo(x,y)
   }
</script>
<title></title>
</head>
<body>
   <a href="javascript: moveWindow(0,0)">
   Mover a janela para o canto superior esquerdo da tela</a><br>
   <a href="javascript: moveWindow(100,80)">
   Mover a janela para a posição 100 x 80</a><br>
   O opera não consegue executar este exemplo.
</body>
</html>
```



Faça um tremor de terra no o seu browser

```
<html>
<head>
<script type="text/javascript">
   var xa=ya=0,x,y
   function tremer(i)
   {
      x=Math.floor((Math.random()<0.5) ? -i:i)
      y=Math.floor((Math.random()<0.5) ? -i:i)
      top.window.moveBy(x-xa,y-ya)
      i*=0.95
      if(x!=0||y!=0||xa!=0||ya!=0)
         setTimeout('tremer('+i+')',20)
      else
         top.window.moveBy(-10,-10)
      xa=x
      ya=y
```

```
   }

   function sismo(event)
   {
      top.window.moveBy(10,10)
      setTimeout('tremer(6)',20)
      return false
   }
</script>
<title></title>
</head>
<body>
   <br><a href="#" onclick="return sismo(event)">
   Clique para iniciar um tremor de terra no seu browser.</a><br>
   <p>Segure-se bem!</p>
   <p>O browser opera não consegue executar este exemplo.</p>
</body>
</html>
```



5. O objeto document

O objeto document faz parte do objeto window, sendo obtido como uma propriedade deste, na forma window.document. Este objeto contém uma representação de um documento escrito em HTML.

Se escrevermos apenas document, omitindo o objeto window, obteremos o objeto que representa o documento da página que estamos usando. Se especificarmos um objeto window correspondente a outra janela ou a outra moldura então iremos obter o objeto document correspondente à página que está nessa janela ou moldura.

Propriedades do objeto document

Propriedade	Descrição
body	Devolve o elemento <body> (se o documento estiver escrito em HTML sem molduras) ou o elemento <frameset> (se o documento estiver escrito em HTML Frameset.)
cookie	Devolve os cookies associados ao documento
domain	Lê o nome de domínio do servidor que enviou o documento
referrer	Lê o endereço (URL) da página em que está a ligação que conduziu até ao documento atual
title	Lê ou define o título do documento
URL	Lê o endereço (URL) do documento

Nota: Algumas das propriedades do objeto document são próprios objetos.

Coleções do objeto document

Coleção	Descrição
anchors	Contém uma lista com todos os elementos <a> (âncoras) existentes no documento
applets	Contém uma lista com todos os elementos <applet> (miniaplicações escritas em Java)
embeds	Contém uma lista com todos os objetos que foram embutidos na página usando os elementos <embed> ou <object>
forms	Contém uma lista com todos os elementos <form> existentes no documento
images	Contém uma lista com todas as imagens existentes no documento
links	Contém uma lista com todos os elementos <a> existentes no documento que especifiquem um valor para o atributo href (é uma subcoleção de anchors.)
plugins	Contém a mesma lista que a coleção embeds

As coleções pertencentes ao objeto document contêm objetos.

Métodos do objeto document

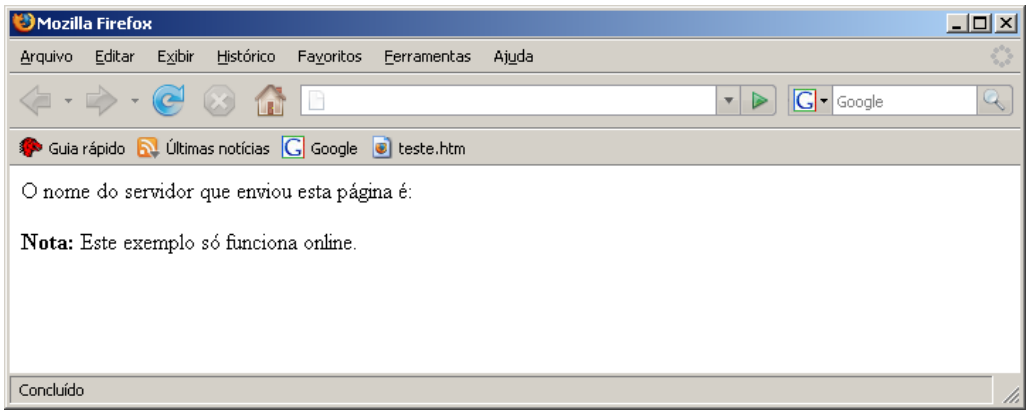
Método	Descrição
close()	Fecha um documento que tenha sido aberto com o método document.open() e escrito com document.write(). Não confundir com o método open() do objeto window, que serve para abrir uma nova janela.
getElementById()	Devolve o elemento que possui o valor especificado no atributo id
getElementsByTagName()	Contém uma lista com todos os elementos que têm um determinado nome (dado por uma etiqueta do HTML)
open()	Abre um documento para nele escrever conteúdo novo
write()	Escreve texto num documento
writeln()	Escreve uma linha de texto num documento

Exemplos de Aplicação

Ler o nome do domínio do servidor que enviou o documento

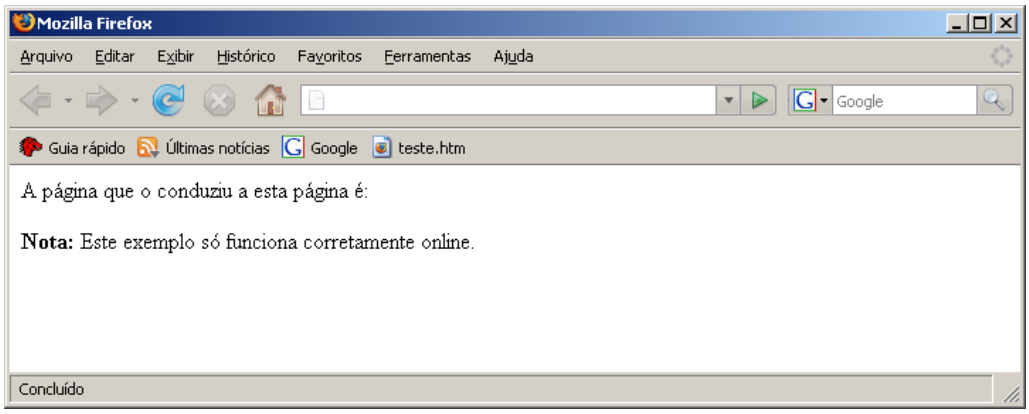
```
<html>
<head>
<title></title>
</head>
<body>
   <p>
   O nome do servidor que enviou esta página é:
   <script type="text/javascript">
      document.write(top.document.domain)
   </script>
   </p>
   <p>
      <b>Nota:</b> Este exemplo só funciona online.
   </p>
</body>
```

```
</html>
```



Ler o URL da página em que estava a ligação que conduziu até esta página

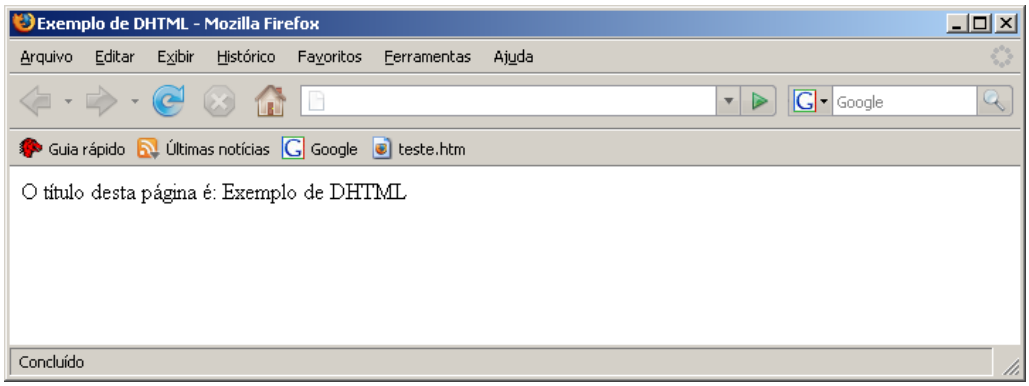
```
<html>
<head>
<title></title>
</head>
<body>
   <p>
   A página que o conduziu a esta página é:
   <script type="text/javascript">
      document.write(top.document.referrer)
   </script>
   </p>
   <p>
   <b>Nota:</b> Este exemplo só funciona corretamente online.
   </p>
</body>
</html>
```



Ler o título do documento

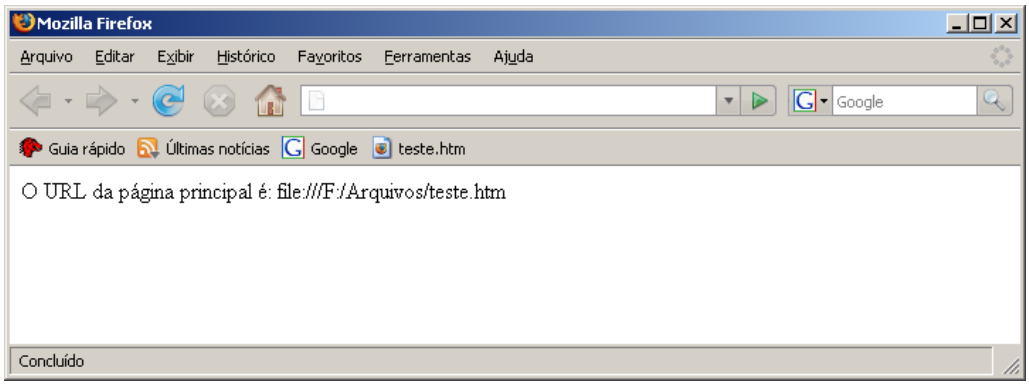
```
<html>
<head>
<title>Exemplo de DHTML</title>
</head>
<body>
   <p>
   O título desta página é:
   <script type="text/javascript">
      document.write(document.title)
   </script>
</p>
</body>
```

```
</html>
```



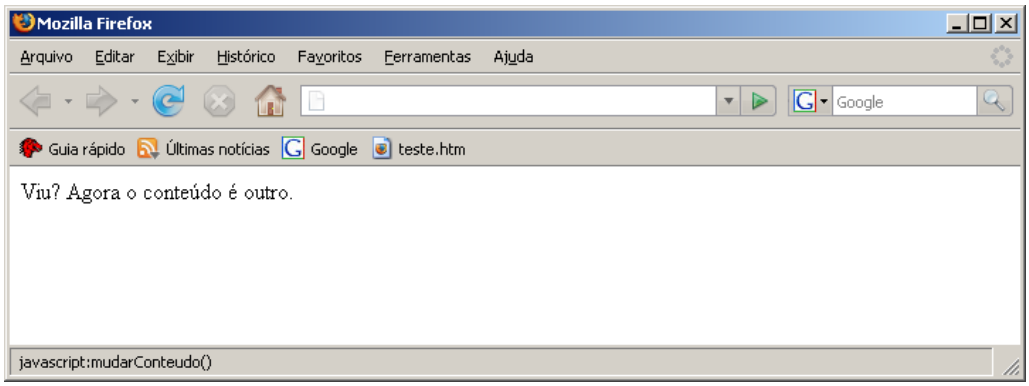
Ler o URL do documento principal

```
<html>
<head>
<title></title>
</head>
<body>
   <p>
   O URL da página principal é:
   <script type="text/javascript">
      document.write(top.document.URL)
   </script>
</p>
</body>
</html>
```



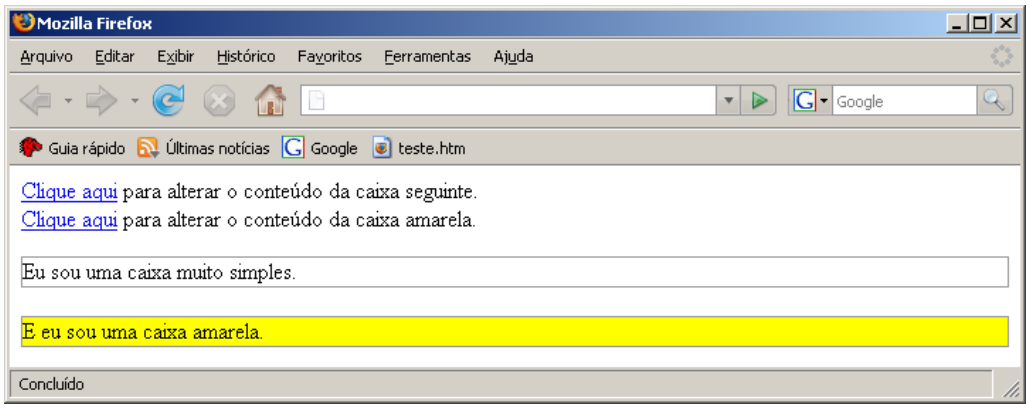
Trabalhar com o objeto que guarda todo o conteúdo da página

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function mudarConteudo()
   {
      document.body.innerHTML='<p>Viu? Agora o conteúdo é outro.</p>'
   }
</script>
</head>
<body>
   <p>
      <a href="javascript:mudarConteudo()">
      Clique aqui</a> para alterar o conteúdo desta página.
   </p>
</body>
</html>
```



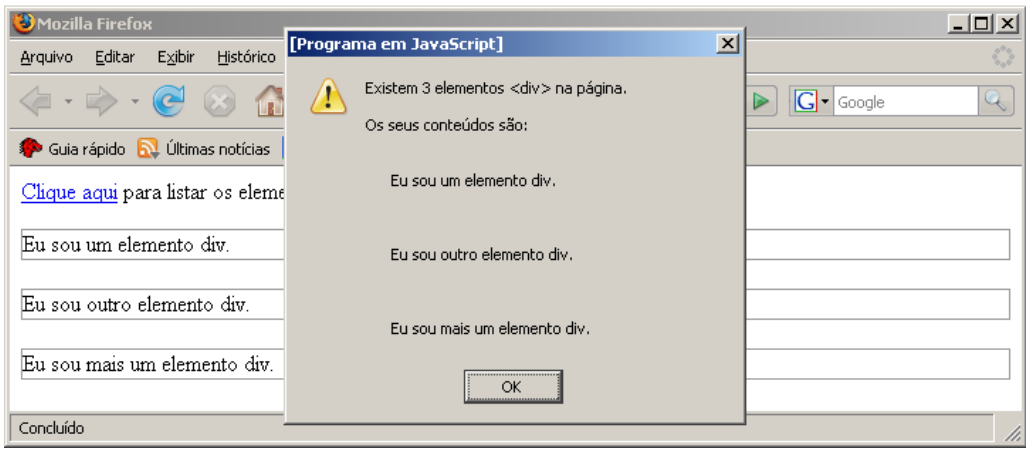
Selecionar um elemento específico e manipulá-lo

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function mudarConteudo()
   {
      var o=document.getElementById("caixa")
      o.innerHTML='Viu? Agora estou diferente.'
   }

   function mudarConteudo2()
   {
      var o=document.getElementById("caixa2")
      o.style.backgroundColor="red"
      o.style.color="white"
      o.innerHTML='E agora sou uma caixa vermelha com letras brancas.'
   }
</script>
</head>
<body>
   <p>
      <a href="javascript:mudarConteudo()">Clique aqui</a> para
      alterar o conteúdo da caixa seguinte.<br>
      <a href="javascript:mudarConteudo2()">Clique aqui</a> para
      alterar o conteúdo da caixa amarela.
   </p>
   <div id="caixa" style="border: solid 1px #999999">
      Eu sou uma caixa muito simples.
   </div><br>
   <div id="caixa2" style="border: solid 1px #999999; background-color:
   yellow">
      E eu sou uma caixa amarela.
   </div>
</body>
</html>
```



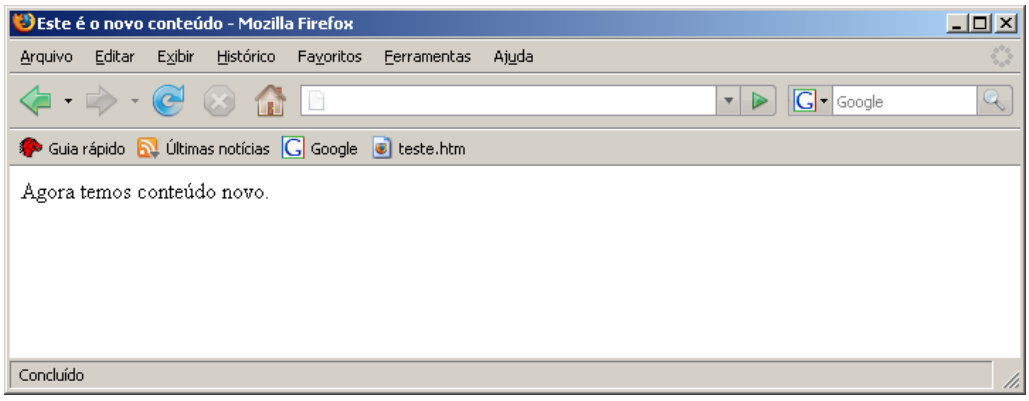
Listar todos os elementos de um determinado tipo

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function verElementos()
   {
      var o=document.getElementsByTagName("div")
      var s='Existem '+o.length+' elementos <div> na página.\n\n'
      s+='Os seus conteúdos são:\n\n'
      for(var i=0;i<o.length;++i)
         s+=o[i].innerHTML+'\n\n'
      alert(s)
   }
</script>
</head>
<body>
   <p>
      <a href="javascript:verElementos()">Clique aqui</a> para
      listar os elementos &lt;div&gt; desta página.
   </p>
   <div style="border: solid 1px #999999">
      Eu sou um elemento div.
   </div><br>
   <div style="border: solid 1px #999999">
      Eu sou outro elemento div.
   </div><br>
   <div style="border: solid 1px #999999">
      Eu sou mais um elemento div.
   </div>
</body>
</html>
```



Escrever um documento de novo

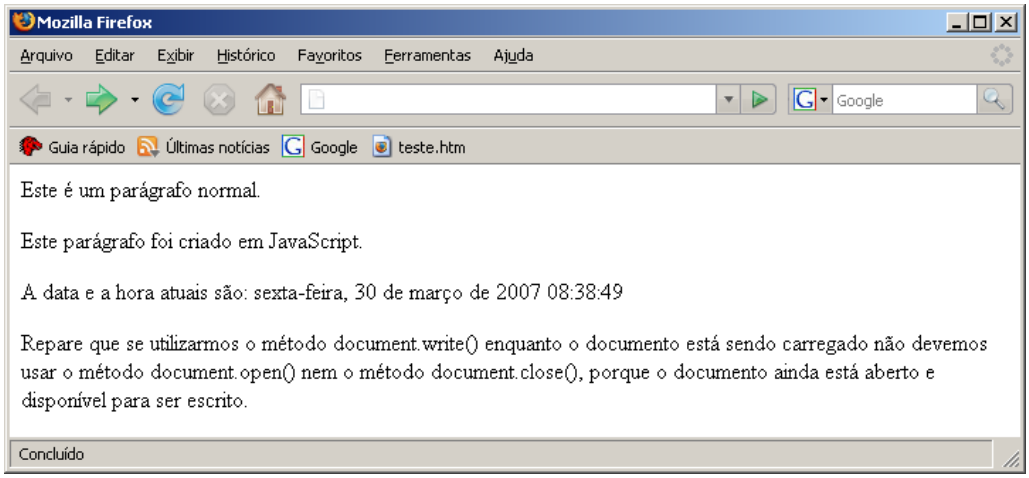
```
<html>
<head>
<title></title>
<script type="text/javascript">
   function escreverNovoConteudo()
   {
      var s='<html>'
      s+='<head><title>Este é o novo conteúdo</title></head>'
      s+='<body>'
      s+='<p>Agora temos conteúdo novo.</p>'
      s+='</body></html>'
      document.open()
      document.write(s)
      document.close()
      /*
      Repare que devemos utilizar os métodos document.open() e
      document.close() porque o documento já estava completamente
      carregado. Como ele já estava fechado é necessário abri-lo,
      escrever nele e fechá-lo de novo.
      */
}
</script>
</head>
<body>
   <p>
      <a href="javascript:escreverNovoConteudo()">Clique aqui</a> para
      substituir o contéudo desta página por novo conteúdo.
   </p>
</body>
</html>
```



Acrescentar conteúdo a um documento

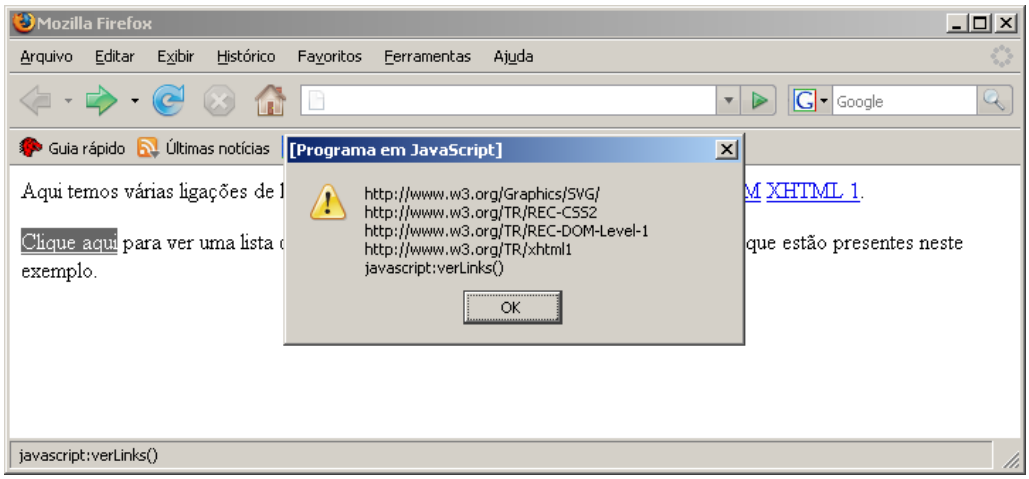
```
<html>
<head>
<title></title>
</head>
<body>
   <p>Este é um parágrafo normal.</p>
   <script type="text/javascript">
   <!--
      var s='<p>Este parágrafo foi criado em JavaScript.</p>'
      var d=new Date()
      s+='<p>A data e a hora atuais são: '+d.toLocaleString()+'</p>'
      document.write(s)
   -->
   </script>
   <p>
   Repare que se utilizarmos o método document.write() enquanto o documento
   está sendo carregado não devemos usar o método document.open() nem o
```

```
   método document.close(), porque o documento ainda está aberto e
   disponível para ser escrito.
   </p>
</body>
</html>
```



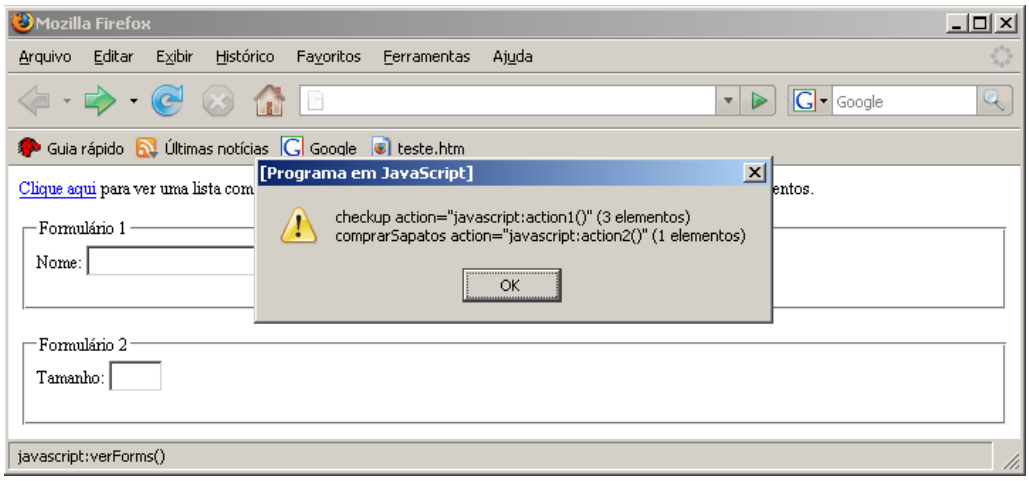
Listar todas as ligações de hipertexto existentes no documento

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function verLinks()
   {
      var s=''
      var o=document.links
      for(var i=0;i<o.length;++i)
         s+=o[i].href+'\n'
      alert(s)
   }
</script>
</head>
<body>
   <p>
   Aqui temos várias ligações de hipertexto para o website do W3C:
   <a href="http://www.w3.org/Graphics/SVG/" target="_blank">SVG</a>,
   <a href="http://www.w3.org/TR/REC-CSS2" target="_blank">CSS2</a>,
   <a href="http://www.w3.org/TR/REC-DOM-Level-1" target="_blank">DOM</a>
   <a href="http://www.w3.org/TR/xhtml1" target="_blank">XHTML 1</a>.
   <p>
   <a href="javascript:verLinks()"
   style="background-color:#666666;color:white">Clique aqui</a>
   para ver uma lista com os valores dos atributos href de todas as ligações
   que estão presentes neste exemplo.
   </p>
</body>
</html>
```



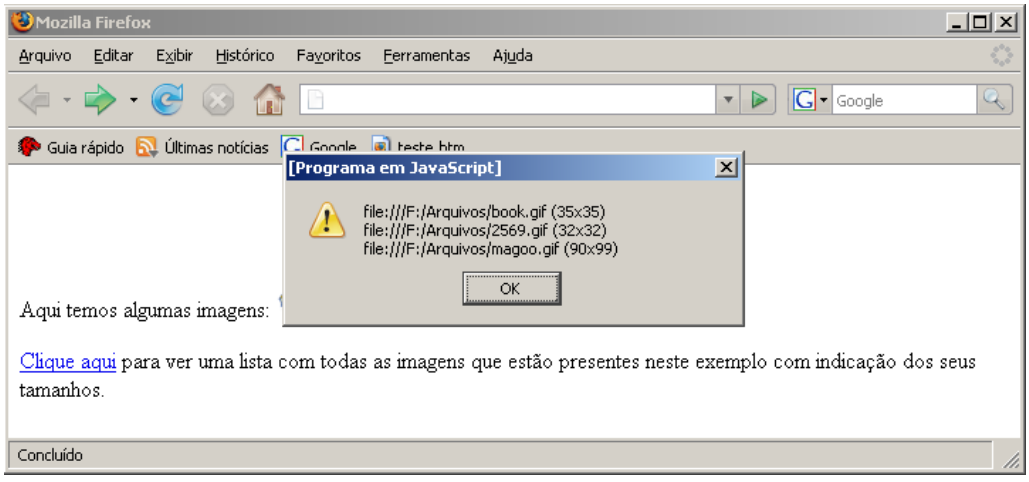
Listar todos os formulários que estão no documento

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function verForms()
   {
      var s=''
      var o=document.forms
      for(var i=0;i<o.length;++i)
         s+=o[i].name+' action="'+o[i].action+'"
         ('+o[i].length+' elementos)\n'
      alert(s)
   }
</script>
</head>
<body style="font-size: 12px">
   <p>
      <a href="javascript:verForms()">Clique aqui</a> para
      ver uma lista com todos os formulários deste exemplo
      com indicação do número dos seus elementos.
   </p>
   <fieldset><legend>Formulário 1</legend>
      <form action="javascript:action1()" name="checkup">
         Nome: <input type="text">
         Peso: <input type="text" size="4">
         Altura: <input type="text" size="4">
      </form>
   </fieldset><br>
   <fieldset><legend>Formulário 2</legend>
      <form action="javascript:action2()" name="comprarSapatos">
         Tamanho: <input type="text" size="2">
      </form>
   </fieldset>
</body>
</html>
```



Listar todas as imagens que estão no documento

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function verImagens()
   {
      var s=''
      var o=document.images
      for(var i=0;i<o.length;++i)
         s+=o[i].src+' ('+o[i].width+'x'+o[i].height+')\n'
      alert(s)
   }
</script>
</head>
<body>
   <p>
   Aqui temos algumas imagens:
   <img src="book.gif" alt="Um livro"> <img src="2569.gif" alt="Direções">
   <img src="magoo.gif" alt="O Senhor Magoo">
   </p>
   <p>
   <a href="javascript:verImagens()">Clique aqui</a> para
   ver uma lista com todas as imagens que estão presentes neste exemplo
   com indicação dos seus tamanhos.
   </p>
</body>
</html>
```



6. O objeto event

O objeto event tem propriedades que descrevem os acontecimentos que ocorrem numa página HTML.

Propriedades do objeto event

Propriedade	Descrição
altKey	Indica (true ou false) se a tecla Alt está pressionada ou não
button	Diz qual dos botões do mouse está pressionado
cancelBubble	Permite controlar e saber se o evento atual se propaga (bubble up) ou não aos objetos que estão cima na hierarquia do DOM
clientX	Lê a coordenada x do cursor do mouse no momento em que ocorreu o evento
clientY	Lê a coordenada y do cursor do mouse no momento em que ocorreu o evento
ctrlKey	Indica (true ou false) se a tecla Ctrl está pressionada ou não
dataFld	Devolve a coluna de dados afetada pelo evento oncellchange
fromElement	Devolve o elemento que o cursor do mouse acaba de abandonar
keyCode	Lê o valor do código Unicode correspondente à tecla que está sendo pressionada
offsetX	Lê a coordenada horizontal do cursor
offsetY	Lê a coordenada vertical do cursor
propertyName	Lê o nome da propriedade qua acabou de mudar de valor
repeat	Indica se o evento se repete ou não
returnValue	Lê ou define o valor que será devolvido pelo evento
screenX	Lê a coordenada x do cursor do mouse relativamente à origem da tela
screenY	Lê a coordenada y do cursor do mouse relativamente à origem da tela
shiftKey	Indica (true ou false) se a tecla Shift está pressionada ou não
srcElement	Devolve o elemento no qual o evento teve origem (MSIE)
toElement	Devolve o objeto para o qual o cursor acabou de entrar (MSIE)
type	Lê o tipo do evento
x	Lê a coordenada x do cursor
y	Lê a coordenada y do cursor

6.1 Funções de compatibilidade para o objeto event

Os eventos são objetos em que ainda existem algumas incompatibilidades entre os browsers dominantes. Para conseguirmos que as nossas páginas que usam eventos funcionem sem modificações em todos os browsers nós somos obrigados a usar código de compatibilidade.

Neste curso usamos duas pequenas bibliotecas com funções de compatibilidade:

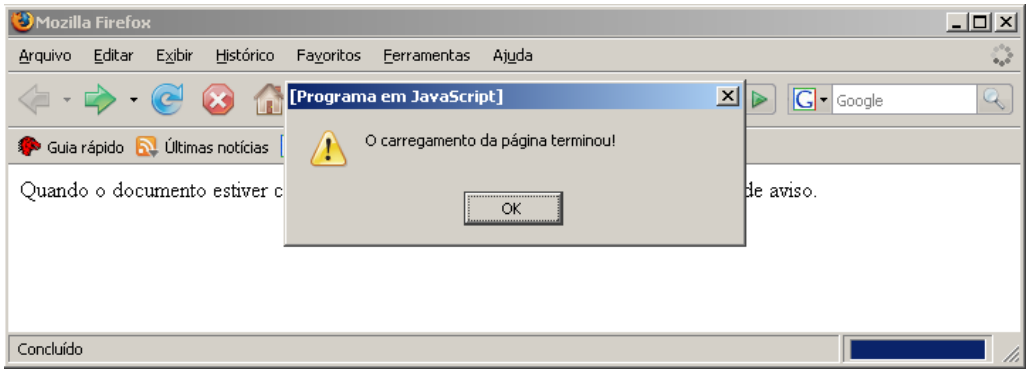
- compat_window.js
- compat_event.js

Alguns dos exemplos que se encontram mais abaixo mostram como se usam na prática estas funções de compatibilidade.

Exemplos de Aplicação

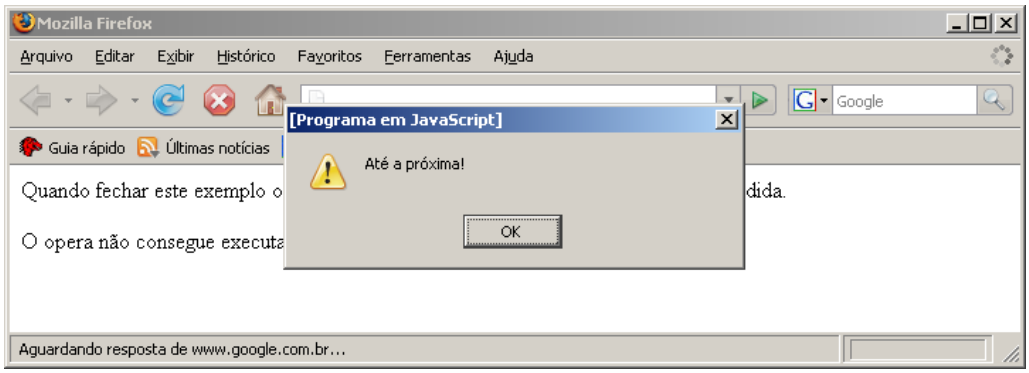
Executar uma ação logo que o documento esteja carregado

```
<html>
<head>
<title></title>
<script type="text/javascript">
  function avisar()
   {
      alert("O carregamento da página terminou!")
   }
</script>
</head>
<body onload="avisar()">
   <p>
      Quando o documento estiver completamente carregado será
      apresentada uma caixa de aviso.
   </p>
</body>
</html>
```



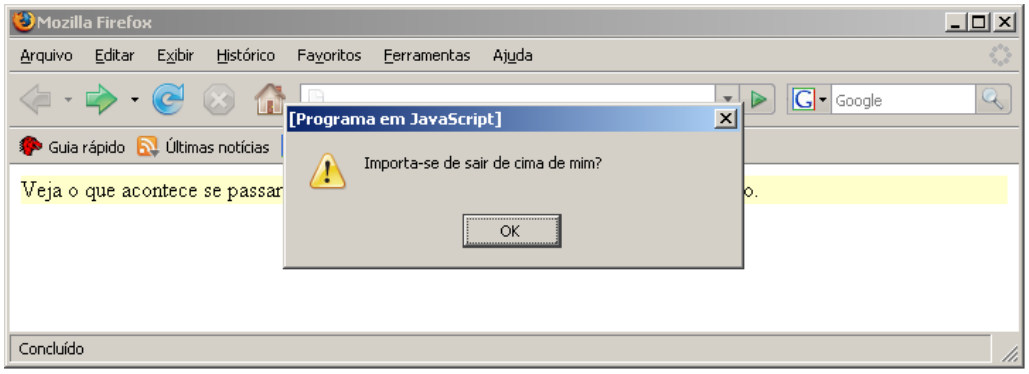
Executar uma ação quando o documento for abandonado

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function tchau()
   {
      alert("Até a próxima!")
   }
</script>
</head>
<body onunload="tchau()">
   <p>
      Quando fechar este exemplo ou carregar outro será
      apresentada uma caixa de despedida.
   </p>
   <p>O opera não consegue executar este exemplo corretamente.</p>
</body>
</html>
```



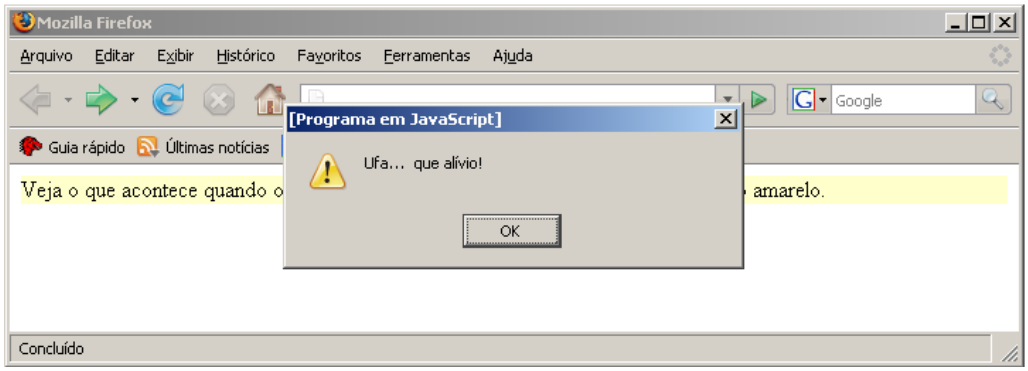
Reagir quando o mouse passar sobre um elemento

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function over()
   {
      alert("Importa-se de sair de cima de mim?")
   }
</script>
</head>
<body>
   <p onmouseover="over()" style="background-color: #ffffcc">
      Veja o que acontece se passar com o mouse sobre este
      parágrafo com fundo amarelo.
   </p>
</body>
</html>
```



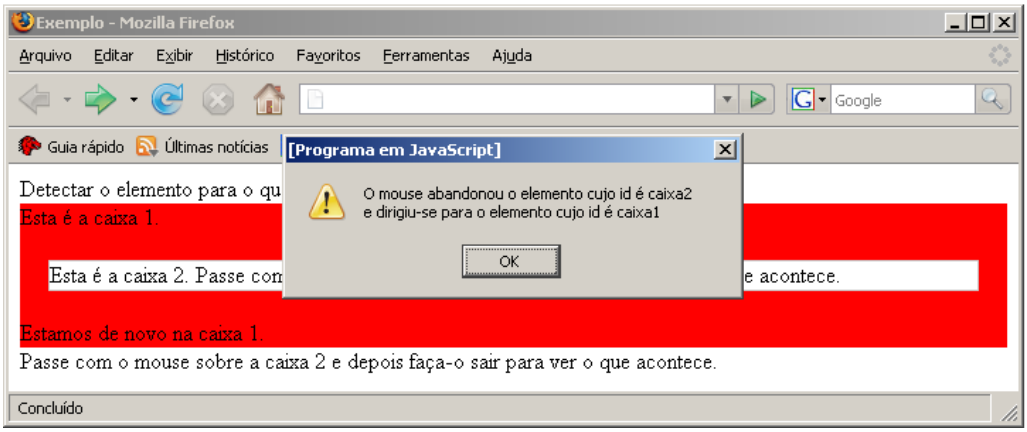
Reagir quando o mouse deixar de estar sobre um elemento

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function out()
   {
      alert("Ufa...  que alívio!")
   }
</script>
</head>
<body>
   <p onmouseout="out()" style="background-color: #ffffcc">
      Veja o que acontece quando o mouse deixa de estar
      sobre este parágrafo com fundo amarelo.
   </p>
</body>
</html>
```



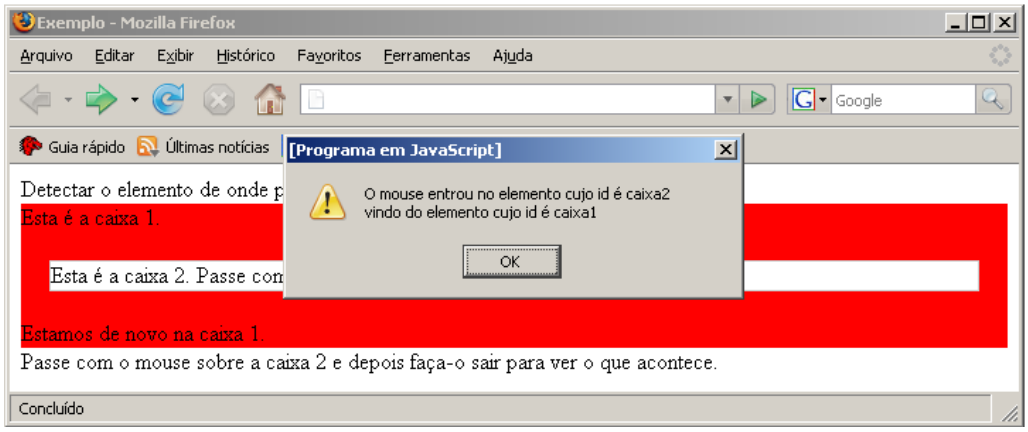
Detectar o elemento para o qual se deslocou o mouse

```
<html>
<head>
<title>Exemplo</title>
</head>
<script type="text/javascript" src="compat_event.js"></script>
   <!--
   A linha anterior faz o carregamento da biblioteca de compatibilidade
   para o objeto event.
   -->
<script type="text/javascript">
   function saiu(event)
   {
      var a=evento_FromTo(event)
      var objDestino=a[1]
      var objPartida=a[0]
      var s="O mouse abandonou o elemento cujo id é "+objPartida.id
      s+="\ne dirigiu-se para o elemento cujo id é "+objDestino.id
      alert(s)
      /*
      As funções eventFromElement() e eventToElement()
      são definidas na biblioteca de compatibilidade
      */
   }
</script>
<body>
   Detectar o elemento para o qual se deslocou o mouse
   <div id="caixa1" style="background-color: red">
      Esta é a caixa 1.
      <div id="caixa2" onmouseout="saiu(event)"
      style="border: solid 1px #cccccc; background-color:white; margin:20px">
         Esta é a caixa 2. Passe com o mouse sobre ela e depois
         faça-o sair para ver o que acontece.
      </div>
      Estamos de novo na caixa 1.
   </div>
   Passe com o mouse sobre a caixa 2 e depois faça-o sair para ver o que
   acontece.
</body>
</html>
```



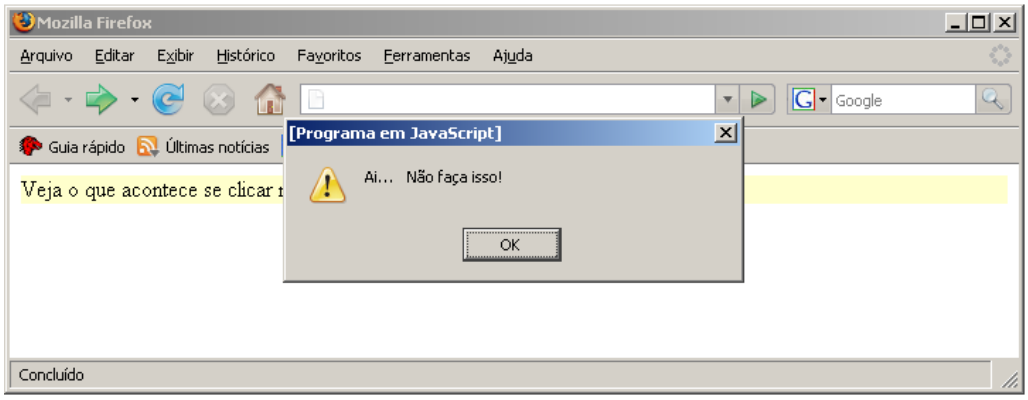
Detectar o elemento de onde vem o mouse

```
<html>
<head>
<title>Exemplo</title>
</head>
<script type="text/javascript" src="compat_event.js"></script>
   <!--
   A linha anterior faz o carregamento da biblioteca de compatibilidade
   para o objeto event.
   -->
<script type="text/javascript">
   function entrou(event)
   {
      var a=evento_FromTo(event)
      var objDestino=a[1]
      var objPartida=a[0]
      var s="O mouse entrou no elemento cujo id é "+objDestino.id
      s+="\nvindo do elemento cujo id é "+objPartida.id
      alert(s)
      /*
      As funções eventFromElement() e eventToElement()
      são definidas na biblioteca de compatibilidade
      */
   }
</script>
<body>
   Detectar o elemento de onde partiu o mouse ao deslocar-se para a caixa 2
   <div id="caixa1" style="background-color: red">
      Esta é a caixa 1.
      <div id="caixa2" onmouseover="entrou(event)"
      style="border: solid 1px #cccccc; background-color:white; margin:20px">
         Esta é a caixa 2. Passe com o mouse sobre ela para
         descobrir de onde ele vem.
      </div>
      Estamos de novo na caixa 1.
   </div>
   Passe com o mouse sobre a caixa 2 e depois faça-o sair para ver o que
   acontece.
</body>
</html>
```



Reagir a um clique do mouse

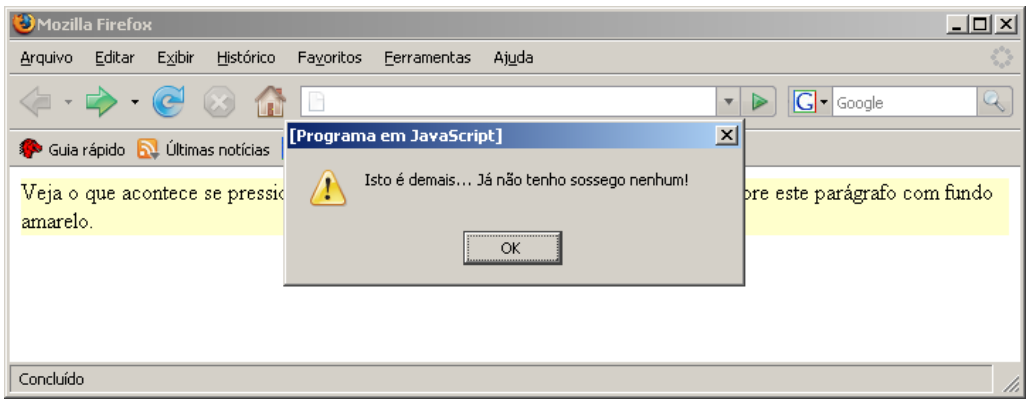
```
<html>
<head>
<title></title>
<script type="text/javascript">
   function clicou()
   {
      alert("Ai...   Não faça isso!")
   }
</script>
</head>
<body>
   <p onclick="clicou()" style="background-color: #ffffcc">
      Veja o que acontece se clicar neste parágrafo com fundo amarelo.
   </p>
</body>
</html>
```



Reagir quando o botão do mouse é pressionado

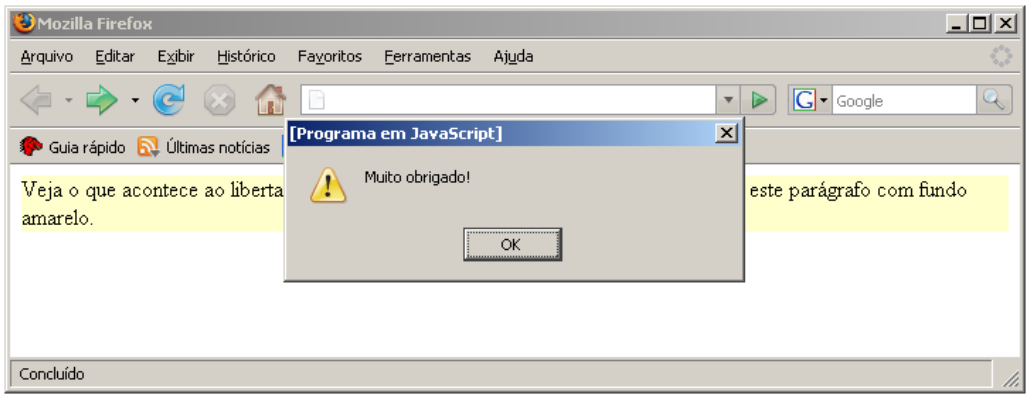
```
<html>
<head>
<title></title>
<script type="text/javascript">
   function mousedown()
   {
      alert("Isto é demais... Já não tenho sossego nenhum!")
   }
</script>
</head>
<body>
   <p onmousedown="mousedown()" style="background-color: #ffffcc">
      Veja o que acontece se pressionar o botão esquerdo do
      mouse enquanto ele está sobre este parágrafo com fundo amarelo.
```

```
   </p>
</body>
</html>
```



Reagir quando o botão do mouse é libertado

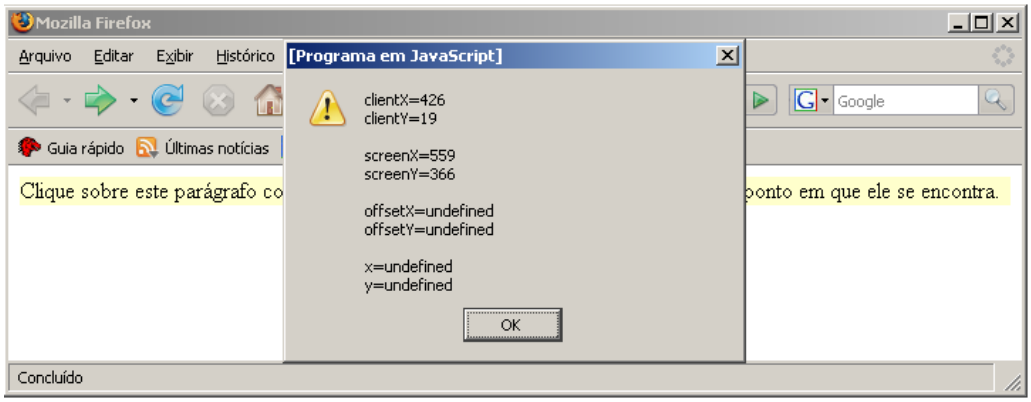
```
<html>
<head>
<title></title>
<script type="text/javascript">
   function mouseup()
   {
      alert("Muito obrigado!")
   }
</script>
</head>
<body>
   <p onmouseup="mouseup()" style="background-color: #ffffcc">
      Veja o que acontece ao libertar o botão esquerdo do
      mouse enquanto ele está sobre este parágrafo com fundo amarelo.
   </p>
</body>
</html>
```



Onde estava o mouse quando aconteceu o evento?

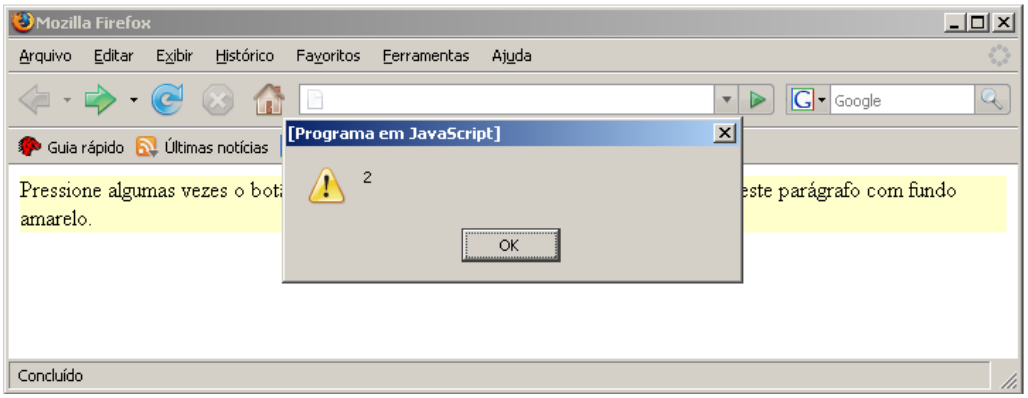
```
<html>
<head>
<title></title>
<script type="text/javascript">
   function pressionou(event)
   {
      var s="clientX="+event.clientX+"\nclientY="+event.clientY
      s+="\n\nscreenX="+event.screenX+"\nscreenY="+event.screenY
      s+="\n\noffsetX="+event.offsetX+"\noffsetY="+event.offsetY
      s+="\n\nx="+event.x+"\ny="+event.y
```

```
      alert(s)
   }
</script>
</head>
<body>
   <p onmousedown="pressionou(event)" style="background-color: #ffffcc">
      Clique sobre este parágrafo com fundo amarelo e
      fique sabendo as coordenadas do ponto em que ele se encontra.
   </p>
</body>
</html>
```



Detectar qual dos botões do mouse foi pressionado

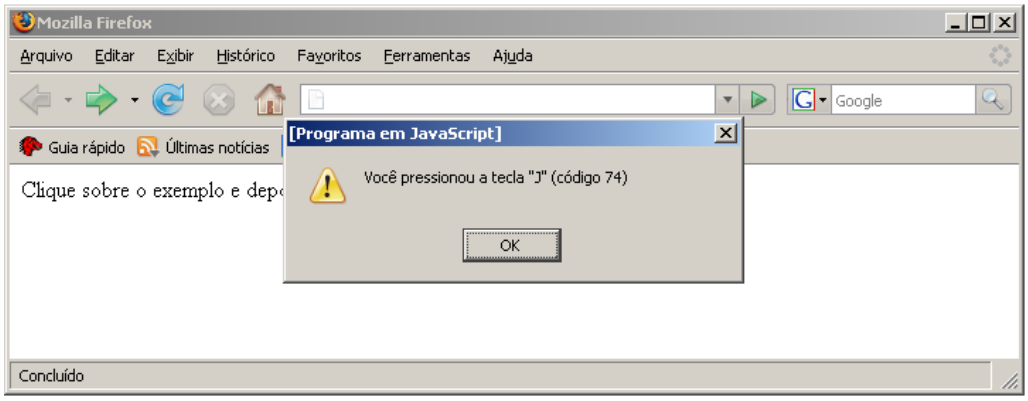
```
<html>
<head>
<title></title>
<script type="text/javascript">

   function pressionou(event)
   {
      alert(event.button)
   }
</script>
</head>
<body>
   <p onmousedown="pressionou(event)" style="background-color: #ffffcc">
      Pressione algumas vezes o botão esquerdo do mouse e outras o
      botão direito sobre este parágrafo com fundo amarelo.
   </p>
</body>
</html>
```



Reagir quando uma tecla é pressionada

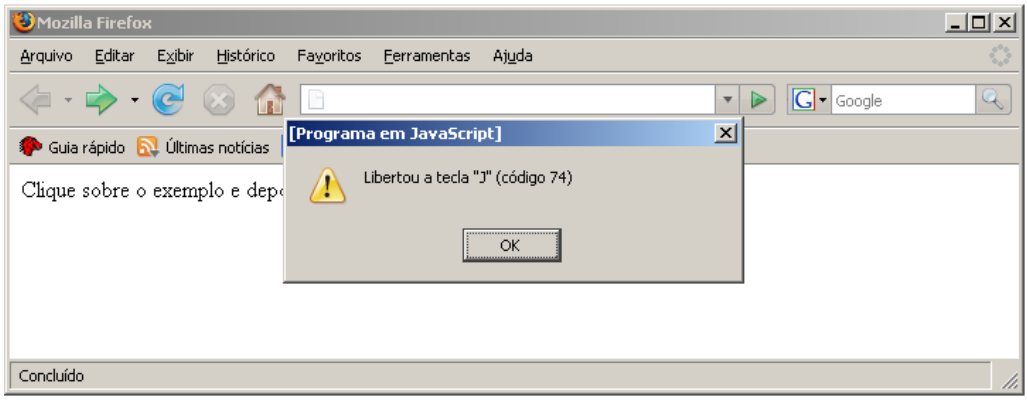
```
<html>
<head>
```

```
<title></title>
<script type="text/javascript">
   function pressionou(event)
   {
      var codigo=event.keyCode
      var tecla=String.fromCharCode(codigo)
      alert('Você pressionou a tecla "'+tecla+'" (código '+codigo+')')
      return false
   }
</script>
</head>
<body onkeydown="pressionou(event)">
   <p>
      Clique sobre o exemplo e depois pressione algumas teclas.
   </p>
</body>
</html>
```



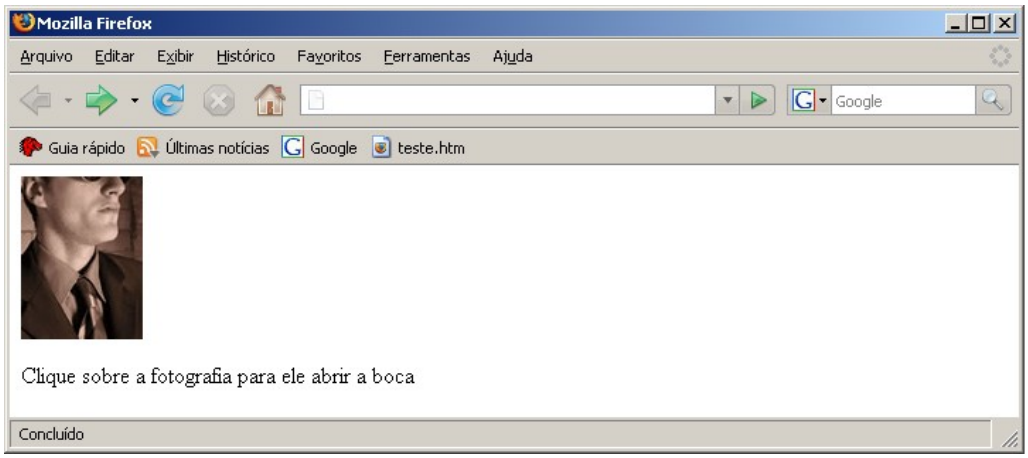
Reagir quando uma tecla é libertada

```
<html>
<head>
<title></title>
<script type="text/javascript">
   function libertou(event)
   {
      var codigo=event.keyCode
      var tecla=String.fromCharCode(codigo)
      alert('Libertou a tecla "'+tecla+'" (código '+codigo+')')
      return false
   }
</script>
</head>
<body onkeyup="libertou(event)">
   <p>Clique sobre o exemplo e depois pressione algumas teclas.</p>
</body>
</html>
```



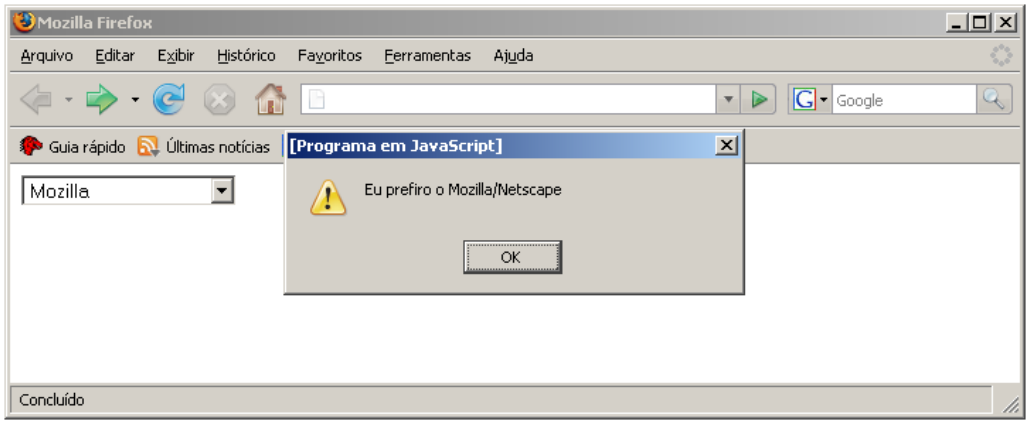
Usar evento onmousedown para trocar imagens

```
<html>
<head>
<script type="text/javascript">
   function abreBoca()
   {
      document.getElementById("imagem").src="aberta.jpg"
   }

   function fechaBoca()
   {
      document.getElementById("imagem").src="fechada.jpg"
   }
</script>
<title></title>
</head>
<body>
   <img onmouseup="fechaBoca()" onmousedown="abreBoca()" id="imagem"
   height="115" src="fechada.jpg" width="86">
   <p>Clique sobre a fotografia para ele abrir a boca</p>
</body>
</html>
```



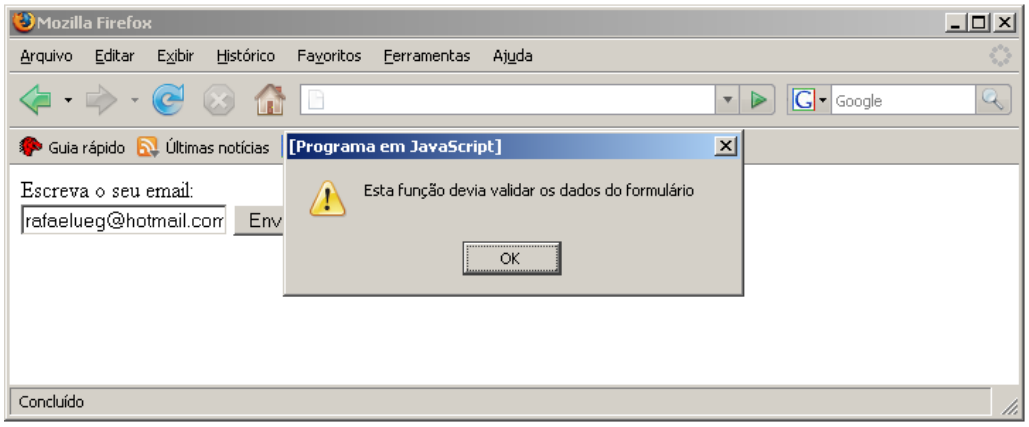
Onchange

```
<html>
<head>
<script type="text/javascript">
   function browserPreferido()
   {
      prefere=document.forms[0].browsers.value
      alert("Eu prefiro o " + prefere)
   }
</script>
<title></title>
</head>
<body>
   <form>
      <select id="browsers" onchange="browserPreferido()">
        <option selected="selected">Escolha um Browser</option>
        <option value="Mozilla/Netscape">Mozilla</option>
        <option value="Microsoft Internet Explorer">Internet Explorer</option>
        <option value="Opera">Opera</option>
      </select>
   </form>
</body>
</html>
```



Onsubmit

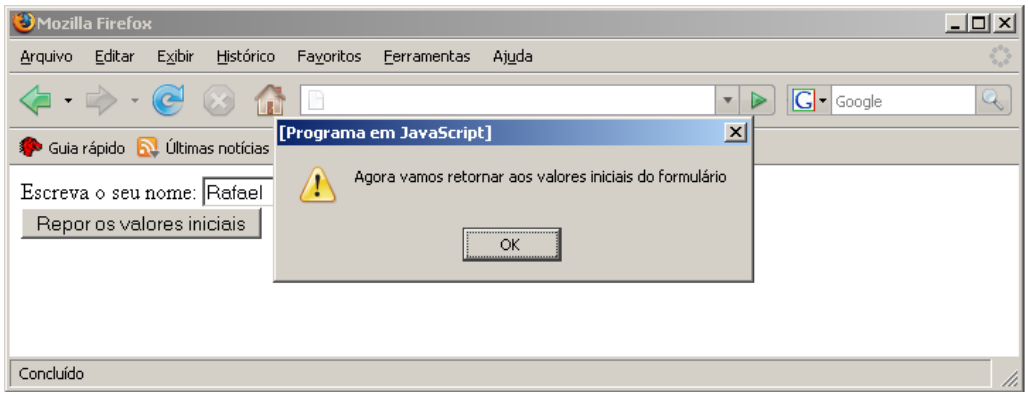
```
<html>
<head>
<script type="text/javascript">
   function validar()
   {
      alert("Esta função devia validar os dados do formulário")
      return false
   }
</script>
<title></title>
</head>
<body>
   <form onsubmit="validar()">
      Escreva o seu email:<br>
      <input type="text" name="mail">
      <input type="submit" value="Enviar dados">
   </form>
</body>
</html>
```



Onreset

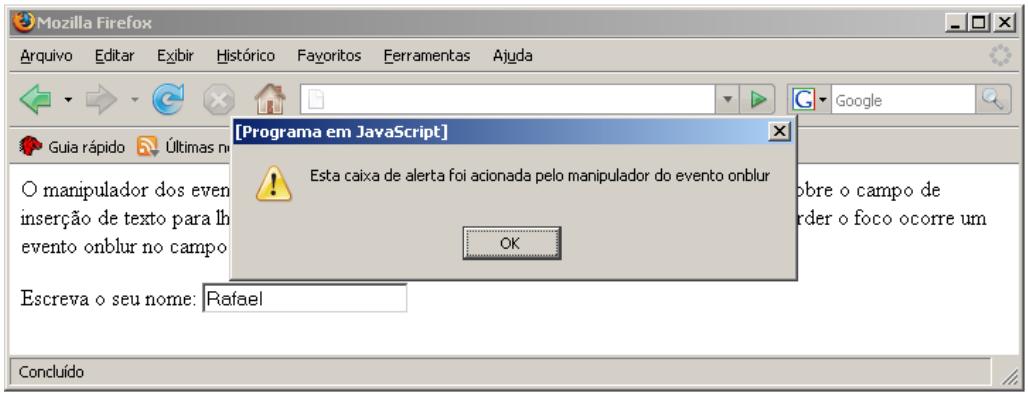
```
<html>
<head>
<script type="text/javascript">
   function mensagem()
   {
      alert("Agora vamos retornar aos valores iniciais do formulário")
   }
</script>
<title></title>
```

```
</head>
<body>
   <form onreset="mensagem()">
      Escreva o seu nome: <input value="desconhecido"><br>
      <input type="reset" value="Repor os valores iniciais">
   </form>
</body>
</html>
```



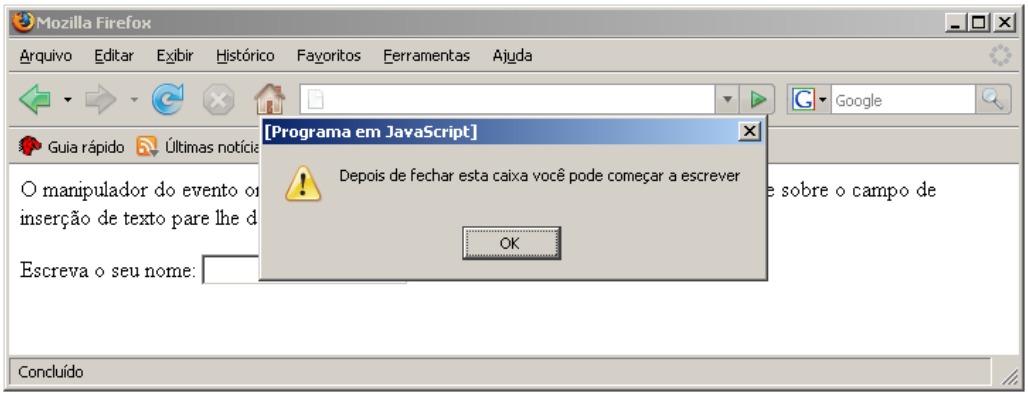
Onblur

```
<html>
<head>
<script type="text/javascript">
   function mensagem()
   {
      alert("Esta caixa de alerta foi acionada pelo manipulador do evento
      onblur")
   }
</script>
<title></title>
</head>
<body>
   <p>
      O manipulador dos eventos onblur é chamado quando um
      elemento perde o foco. Clique sobre o campo de
      inserção de texto para lhe dar o foco. A seguir clique
      em outra parte do documento. Ao perder o foco ocorre um evento
      onblur no campo de inserção de dados e o manipulador de
      eventos é invocado.
   </p>
   <form>
      Escreva o seu nome:
      <input onblur="mensagem()">
   </form>
</body>
</html>
```



Onfocus

```
<html>
<head>
<script type="text/javascript">
   function mensagem()
   {
      alert("Depois de fechar esta caixa você pode começar a escrever")
   }
</script>
<title></title>
</head>
<body>
   <p>
      O manipulador do evento onfocus é chamado quando um
      elemento ganha o foco. Clique sobre o campo de
      inserção de texto pare lhe dar o foco.
   </p>
   <form>
       Escreva o seu nome:
       <input onfocus="mensagem()">
   </form>
</body>
</html>
```



7. O objeto navigator

O objeto navigator contém propriedades que guardam informação acerca do browser que está apresentando a página.

Propriedades do objeto navigator

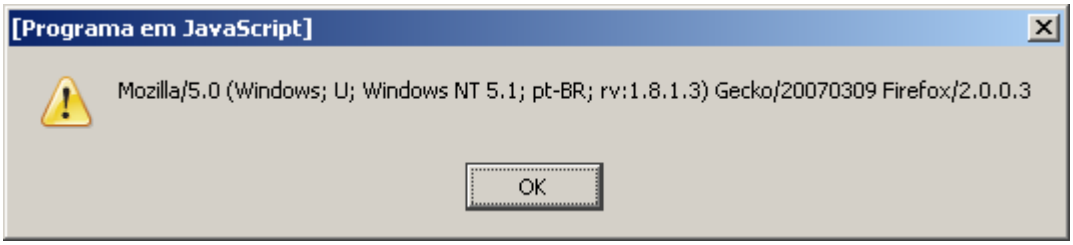
Propriedade	Descrição
appCodeName	Lê o nome de código do browser
appMinorVersion	(Só MSIE) Lê a parte baixa do número de versão do browser
appName	Lê o nome do browser
appVersion	Lê o número de versão do browser e a plataforma em que está sendo executado
browserLanguage	(Só MSIE) Indica a língua principal usada pelo usuário do browser
language	(Só Mozilla/Netscape e Opera) Indica a língua desta versão do browser
cookieEnabled	Indica (true ou false) se o browser aceita cookies
cpuClass	(Só MSIE) Indica a classe do CPU usado no sistema em que o browser está sendo executado

oscpu	(Só Mozilla/Netscape e Opera) Indica o sistema operacional em que o browser está sendo executado
onLine	(Só MSIE) Indica (true ou false) se o sistema está online ou não
platform	Lê o nome da plataforma em que o browser está sendo executado
systemLanguage	(Só MSIE) Indica a língua principal do sistema em que o browser está sendo executado
userAgent	Lê o texto com que o browser se identifica perante os servidores da Web
userLanguage	Indica a língua principal usada pelo usuário do browser

Exemplos de Aplicação

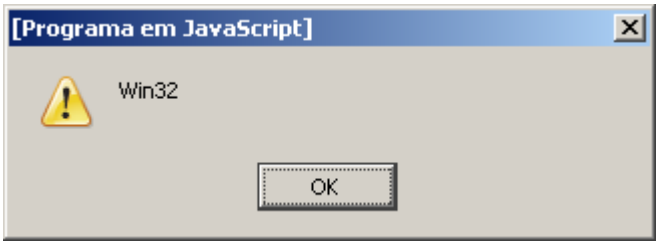
Ler o nome completo do browser

```
<html>
<head>
<title>Exemplo</title>
</head>
<script type="text/javascript">
   alert(navigator.userAgent)
</script>
<body>
</body>
</html>
```



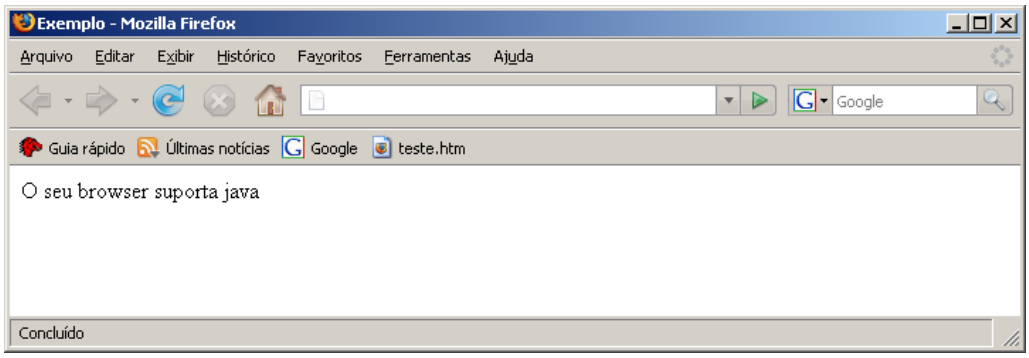
Ler o nome da plataforma que está a ser usada

```
<html>
<head>
<title>Exemplo</title>
</head>
<script type="text/javascript">
   alert(navigator.platform)
</script>
<body>
</body>
</html>
```



Saber se este browser suporta Java

```
<html>
<head>
<title>Exemplo</title>
</head>
<body>
<script type="text/javascript">
   if(navigator.javaEnabled()==true)
      document.write("O seu browser suporta java")
   else
      document.write("O seu browser não suporta java")
</script>
</body>
</html>
```



8. O objeto screen

Alguns browsers definem propriedades para este objeto que não são suportadas pelos outros. No entanto isso não coloca qualquer problema porque as propriedades que são comuns a todos os browsers são suficientes para satisfazer as necessidades.

Propriedades do objeto screen

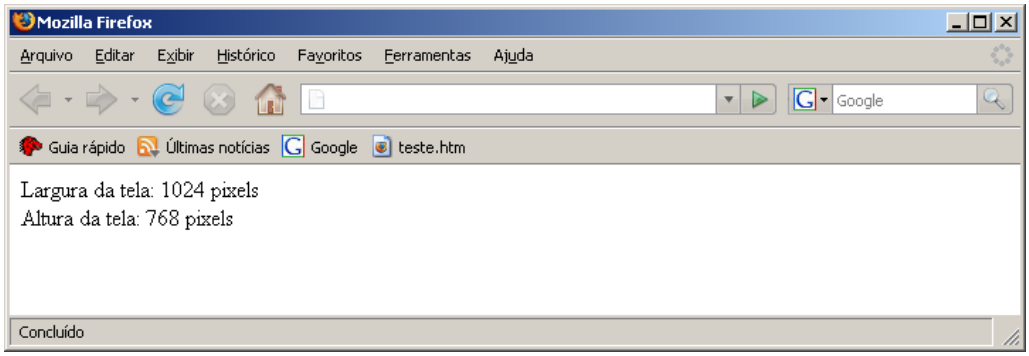
Propriedade	Descrição
width	Lê o valor da largura total da tela (em pixels)
height	Lê o valor da altura total da tela (em pixels)
availWidth	Valor da largura máxima de tela disponível para apresentar a página. Este valor é igual à largura total menos a largura ocupada pelos elementos permanentes ou semi-permanentes que o sistema operacional coloca na tela (como é o caso da barra de tarefas do Windows).
availHeight	Valor da altura máxima de tela disponível para apresentar a página. Este valor é igual à altura total menos a altura ocupada pelos elementos permanentes ou semi-permanentes que o sistema operacional coloca na tela (como é o caso da barra de tarefas do Windows).
colorDepth	Lê o número de bits correspondente à paleta de cores que está sendo usada para ver a página

Exemplos de Aplicação

Ler o tamanho da tela em que está sendo mostrada a página

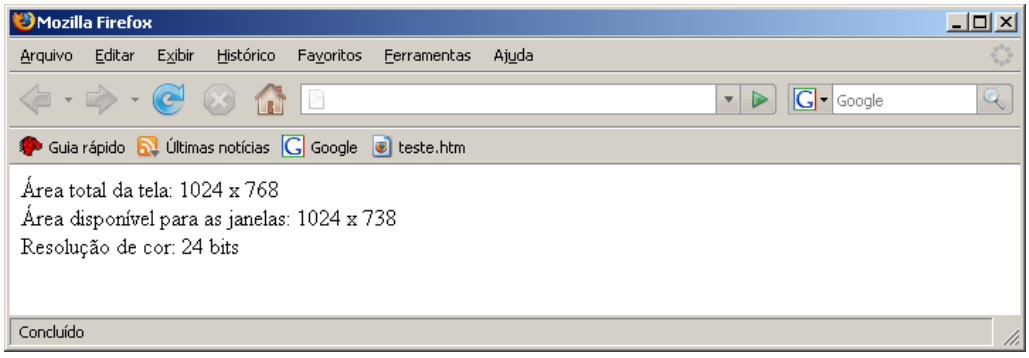
```
<html>
<head>
```

```
<title></title>
</head>
<body>
<script type="text/javascript">
   s="Largura da tela: "+screen.width+" pixels<br>"
   s+="Altura da tela: "+screen.height+" pixels"
   document.write(s)
</script>
</body>
</html>
```



Ler as propriedades da janela e da tela usada para ver a página

```
<html>
<head>
<title></title>
</head>
<body>
<script type="text/javascript">
   s="Área total da tela: "+screen.width+" x "+screen.height
   s+="<br>"
   s+="Área disponível para as janelas: "+screen.availWidth+" x
   "+screen.availHeight
   s+="<br>"
   s+="Resolução de cor: "+screen.colorDepth + " bits"
   document.write(s)
</script>
</body>
</html>
```



9. O objeto location

O objeto location faz parte do objeto window, sendo obtido como uma propriedade deste, na forma window.location. Este objeto contém o URL completo da página que está carregada numa determinada janela.

Se escrevermos apenas location, omitindo o objeto window, obteremos o objeto correspondente à página que estamos usando. Se especificarmos um objeto window então iremos obter o objeto location correspondente à página que está na janela que indicamos.

Todas as propriedades deste objeto possuem valores formados por texto.

Propriedades do objeto location

Propriedade	Descrição
hash	A propriedade hash é composta pela parte do endereço da página (URL) que começa pelo símbolo #.
host	Esta propriedade contém apenas a parte inicial do endereço. É composta pelo protocolo, pelo nome do servidor e eventualmente pelo número da porta usada para comunicar com o servidor.
hostname	Esta propriedade contém apenas o nome do servidor
href	Contém o endereço completo da página. Podemos dar como valor o endereço de uma nova página para que ela seja carregada em substituição da atual. As restantes propriedades são extraídas do valor desta propriedade.
pathname	A porção do URL que vem a seguir ao valor da propriedade host. Esta propriedade contém o local em que a página se encontra dentro do servidor a que pertence.
port	Número da porta usada para comunicar com o servidor
protocol	Parte inicial do URL (normalmente http://)
search	Parte do URL que começa com o símbolo ? (ponto de interrogação) e termina no fim do URL ou logo que seja encontrado o símbolo # (se ele estiver presente)

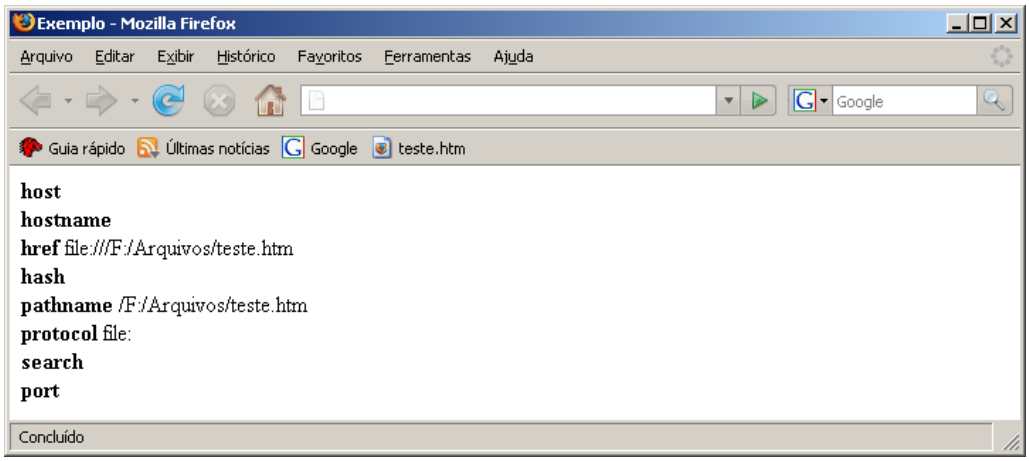
Métodos do objeto location

Método	Descrição
reload()	Refresca a página fazendo um novo pedido de envio ao servidor
replace()	Substitui a página atual por uma outra sem adicionar o endereço da página atual à história da navegação

Exemplos de Aplicação

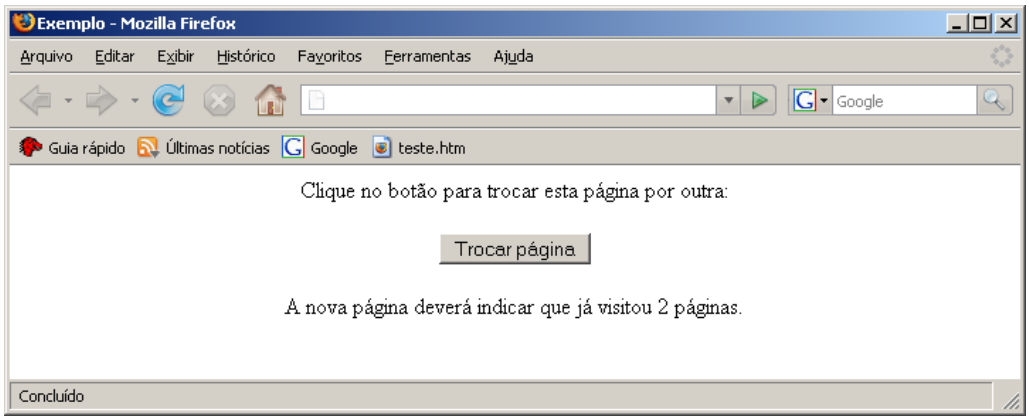
Propriedades do objeto location para a página que executa o exercício

```
<html>
<head>
<title>Exemplo</title>
</head>
<body>
<script type="text/javascript">
   var s="<b>host</b> "+location.host+"<br>"
   s+="<b>hostname</b> "+location.hostname+"<br>"
   s+="<b>href</b> "+location.href+"<br>"
   s+="<b>hash</b> "+location.hash+"<br>"
   s+="<b>pathname</b> "+location.pathname+"<br>"
   s+="<b>protocol</b> "+location.protocol+"<br>"
   s+="<b>search</b> "+location.search+"<br>"
   s+="<b>port</b> "+location.port
   document.write(s)
</script>
</body>
</html>
```



Navegar para outra página

```
<html>
<head>
<title>Exemplo</title>
<script type="text/javascript">
   function novaPagina()
   {
      location.href="pagina.html"
      return false
   }
</script>
</head>
<body>
   <form action="javascript:;" style="text-align:center">
      <script type="text/javascript">
         document.write("<p>Nesta sessão já visitou <b>"+history.length+"
         páginas.</b></p>")
      </script>
      Clique no botão para trocar esta página por outra:<br><br>
      <input type="button" onclick="novaPagina()" value="Trocar
      página"><br><br>
      A nova página deverá indicar que já visitou
      <script type="text/javascript">
         document.write(history.length+1)
      </script>
      páginas.
   </form>
</script>
</body>
</html>
```



10. O objeto history

Este objeto contém uma lista com os endereços de todas as páginas visitadas numa determinada janela do browser e métodos que nos permitem avançar e recuar para as páginas que se encontram nessa lista. Este objeto é um subobjeto de window.

Propriedades do objeto history

Propriedade	Descrição
length	Número de páginas visitadas por uma janela do browser

Métodos do objeto history

Método	Descrição
back()	Carregar a página que está na lista antes da atual
forward()	Carregar a página que sucede a atual na lista
go()	Saltar para qualquer página da lista. Por exemplo: history.go(-1) volta para a página anterior, history.go(-3) salta três páginas para trás, history.go(1) salta para a página seguinte e history.go(2) salta duas páginas para a frente.

PARTE IV: Controle dos Elementos HTML

11. Objetos que representam elementos do HTML

O objeto document é sem dúvida o objeto mais rico e mais importante para o HTML Dinâmico. A forma mais comum para manipular os seus elementos baseia-se nos estilos CSS e nos métodos genéricos do DOM.

Além destas formas genéricas de manipulação do documento, foram acrescentadas ao DOM algumas extensões HTML que nos oferecem funcionalidades acrescidas para controlar objetos que são especiais. Destes objetos há 16 que são mais importantes para o DHTML. Para estes objetos foram criadas propriedades e métodos específicos que são muito úteis para manipular o conteúdo de uma página da Web.

11.1 Objeto anchor

Este objeto representa de forma completa os elementos criados com a etiqueta <a> do HTML. A coleção anchors, pertencente ao objeto document, contém uma lista com todos os objetos deste tipo que estão na página.

Propriedades do objeto anchor

Propriedade	Descrição
accessKey	Define uma tecla para acessar rapidamente um elemento <a>
charset	Lê ou define o conjunto de caracteres usado na página a que a ligação conduz
coords	Lê ou define as coordenadas de um elemento <a>
href	Lê ou define o endereço (URL) da página a que a ligação conduz
hreflang	Lê ou define o código de língua usado na página a que a ligação conduz
name	Lê ou define o nome da âncora
rel	Lê ou define a relação existente entre o documento atual e a página para a qual aponta a ligação
rev	Lê ou define a relação existente entre a página para a qual aponta a ligação e o documento atual
shape	Lê ou define a forma geométrica da área afetada pela âncora
tabIndex	Lê ou define o índice (número de ordem) do elemento que corresponde a este objeto no acesso através da tecla tab
target	Lê ou define o nome da janela na qual se deve abrir a ligação
type	Lê ou define os tipos de conteúdos existentes na página (ou outro recurso da Web) a que a ligação conduz

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <a> em acordo com a linguagem HTML.

Métodos do objeto anchor

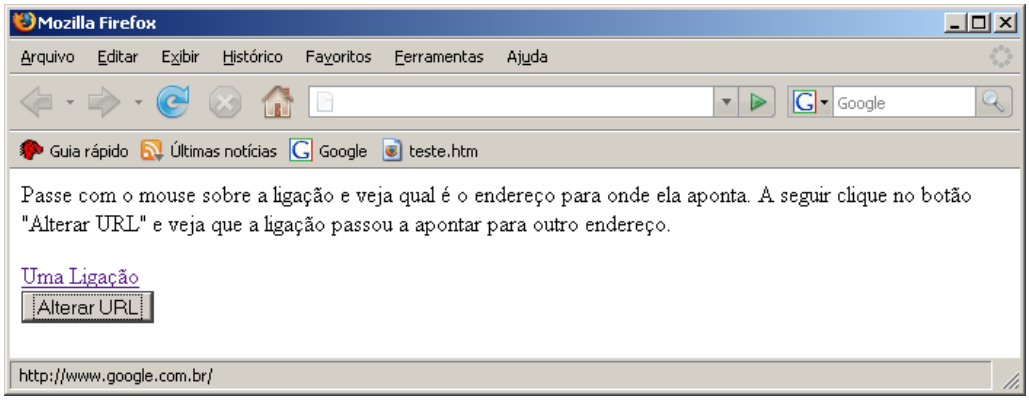
Método	Descrição

blur()	Faz o elemento perder o foco
focus()	Faz o elemento ganhar o foco

Exemplos de Aplicação

Modificar o valor do atributo href de uma ligação de hipertexto

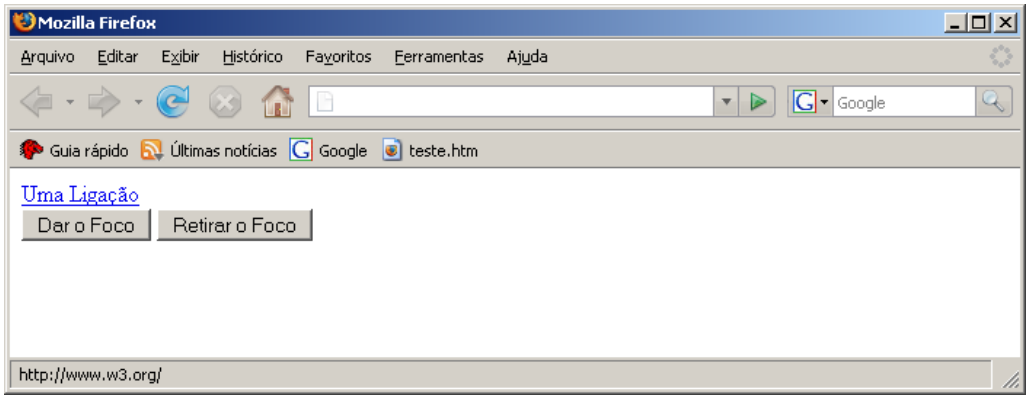
```
<html>
<head>
<script type="text/javascript">
   function myHref()
   {
      document.getElementById("myAnchor").href="http://www.google.com.br"
   }
</script>
<title></title>
</head>
<body>
   <p>
      Passe com o mouse sobre a ligação e veja qual é o endereço para onde
      ela aponta. A seguir clique no botão "Alterar URL" e veja que
      a ligação passou a apontar para outro endereço.
   </p>
   <a id="myAnchor" href="http://www.w3.org" target="_blank">Uma Ligação</a>
   <form>
      <input onclick="myHref()" type="button" value="Alterar URL">
   </form>
</body>
</html>
```



Dar e retirar o foco

```
<html>
<head>
<script type="text/javascript">
   function makeFocus()
   {
      document.getElementById("myAnchor").focus()
   }
   function makeBlur()
   {
      document.getElementById("myAnchor").blur()
   }
</script>
<title></title>
</head>
<body>
   <a id="myAnchor" href="http://www.w3.org" target="_blank">Uma Ligação</a>
   <form>
      <input onclick="makeFocus()" type="button" value="Dar o Foco">
```

```
      <input onclick="makeBlur()" type="button" value="Retirar o Foco">
   </form>
</body>
</html>
```



11.2 Objeto applet

Este objeto representa de forma completa os elementos criados com a etiqueta <applet> do HTML. A coleção applets, pertencente ao objeto document, contém uma lista com todos os objetos deste tipo que estão na página.

Propriedades

Propriedade	Descrição
align	Esta propriedade corresponde ao atributo align do elemento <applet> do HTML e funciona do mesmo modo que o atributo align do elemento <img>. Ela alinha o objeto (tanto na vertical como na horizontal) relativamente ao texto que o rodeia.
alt	Lê ou fornece o texto alternativo a apresentar pelo browser em vez do elemento caso não consiga apresentar o seu conteúdo.
archive	Nome do arquivo que contém o código que o applet (miniaplicação) deve executar. Podemos fornecer os nomes de vários arquivos separando-os com vírgulas.
code	Nome da classe Java na qual o applet deve iniciar a execução do código.
codeBase	Corresponde ao atributo codebase do elemento <applet> do HTML.
height	Lê ou redefine a altura do elemento
name	Nome da miniaplicação (applet). Corresponde ao atributo name.
object	Nome do arquivo que contém uma versão serializada da miniaplicação (applet)
width	Largura ocupada pelo applet

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <applet> em acordo com a linguagem HTML.

11.3 Objeto embed

Este objeto representa de forma completa os elementos criados com a etiqueta <embed> do HTML. A coleção embeds, pertencente ao objeto document, contém uma lista com todos os objetos deste tipo que estão na página.

Propriedades importantes

Propriedade	Descrição
accessKey	Tecla que permite acessar rapidamente o elemento representado por este objeto
align	Esta propriedade corresponde ao atributo align do elemento <embed> do HTML e funciona do mesmo modo que o atributo align do elemento <img>. Ela alinha o objeto (tanto na vertical como na horizontal) relativamente ao texto que o rodeia.
className	Lê ou define o nome da classe do objeto que é embutido na página através do elemento <embed>
height	Altura do objeto
src	Lê ou define o endereço (URL) de onde provém o objeto
width	Largura do objeto

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <embed> em acordo com a linguagem HTML.

Métodos importantes

Método	Descrição
blur()	Retira o foco ao objeto
click()	Desencadeia o mesmo comportamento que seria iniciado se fosse feito um clique sobre o objeto
focus()	Dá o foco ao objeto, e caso ele seja sensível ao evento onfocus inicia a reação correspondente
scrollIntoView()	Faz deslocar a página dentro da janela do browser (scroll) para um ponto em que o objeto fique visível na tela

11.4 Objeto frame

Num documento HTML com molduras este objeto representa de forma completa os elementos criados com a etiqueta <frame> do HTML Frameset. A coleção frames, pertencente ao objeto window contém uma lista com todos os objetos deste tipo que estão na página.

Propriedades

Propriedade	Descrição
frameBorder	Corresponde ao atributo frameborder do elemento <frame>
longDesc	Corresponde ao atributo longdesc
marginHeight	Corresponde ao atributo marginheight
marginWidth	Corresponde ao atributo marginwidth
name	Corresponde ao atributo name
noResize	Corresponde ao atributo noresize. Pode ter os valores true (a moldura não pode ser redimensionada) ou false (a moldura pode ser redimensionada.)
scrolling	Indica se a página que está na moldura deve ou não apresentar barras de deslocamento (scroll)
src	Corresponde ao atributo src

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <frame> em acordo com a linguagem HTML.

11.5 Objeto frameset

Este objeto representa de forma completa os elementos criados com a etiqueta <frameset> de um documento escrito em HTML Frameset.

Propriedades

Propriedade	Descrição
cols	Corresponde ao atributo cols (colunas) do elemento <frameset>
rows	Corresponde ao atributo rows (linhas) do elemento <frameset>

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <frameset> em acordo com a linguagem HTML.

11.6 Objeto form

Este objeto representa de forma completa os elementos criados com a etiqueta <form> do HTML. A coleção forms, pertencente ao objeto document, contém uma lista com todos os objetos deste tipo que estão na página.

Propriedades

Propriedade	Descrição
acceptCharset	Contém uma lista com os conjuntos de caracteres que o servidor pode aceitar
action	Lê ou define o valor do atributo action do formulário
elements	Contém uma lista com todos os campos (elementos) existentes no formulário
enctype	Lê o tipo de codificação usada para enviar os dados do formulário
length	Fornece a quantidade de elementos (campos) que compõem o formulário
method	Lê ou define o método HTTP usado para submeter os dados do formulário
name	Lê ou define o nome do formulário
target	Indica o nome da moldura ("frame") onde deve ser apresentada a resposta do servidor

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <form> em acordo com a linguagem HTML.

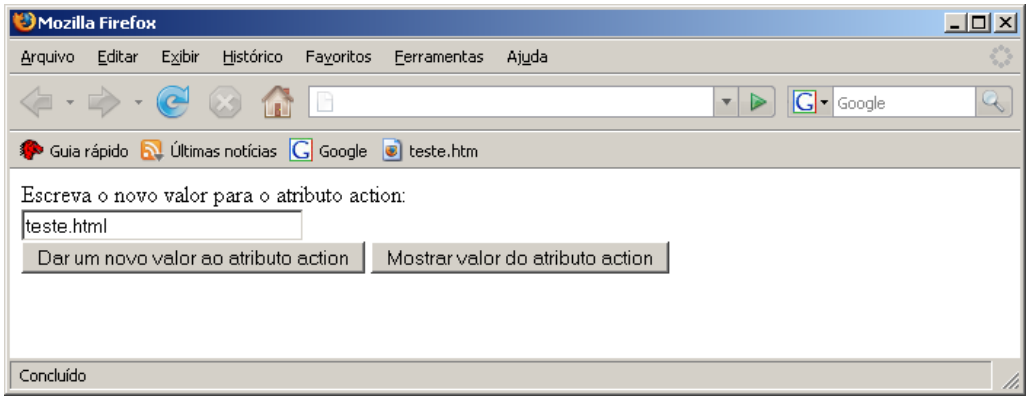
Métodos

Método	Descrição
reset()	Repõe os valores de todos os campos do formulário nos seus valores iniciais
submit()	Submete os dados do formulário para processamento

Exemplos de Aplicação

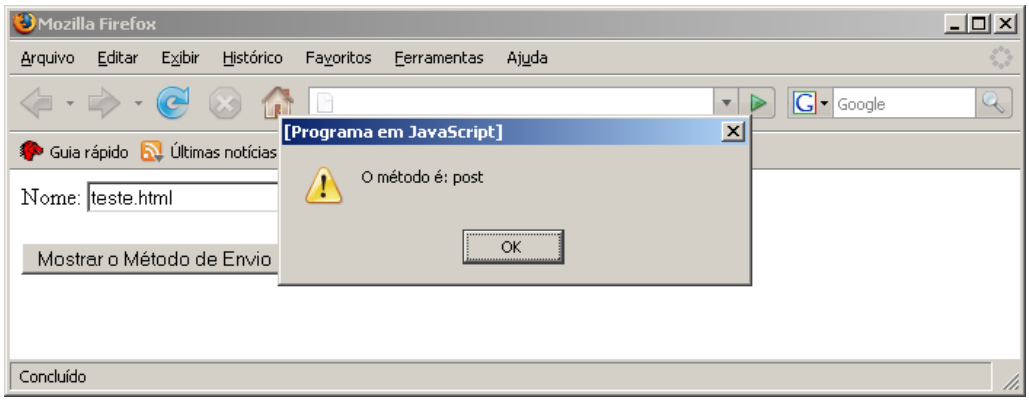
Alterar o valor do atributo action (URL do recurso que processará o formulário)

```
<html>
<head>
<script type="text/javascript">
   function formAction()
   {
      var f=document.getElementById("myForm")
      alert(f.action)
   }

   function novaAction()
   {
      var f=document.getElementById("myForm")
      f.action=document.getElementById("newAction").value
   }
</script>
<title></title>
</head>
<body>
   <form id="myForm" method="post" action="processar.html">
      Escreva o novo valor para o atributo action:<br>
      <input type="text" id="newAction" value="action" size="28"><br>
      <input onclick="novaAction()" type="button"
      value="Dar um novo valor ao atributo action">
      <input onclick="formAction()" type="button"
      value="Mostrar valor do atributo action"><br>
   </form>
</body>
</html>
```



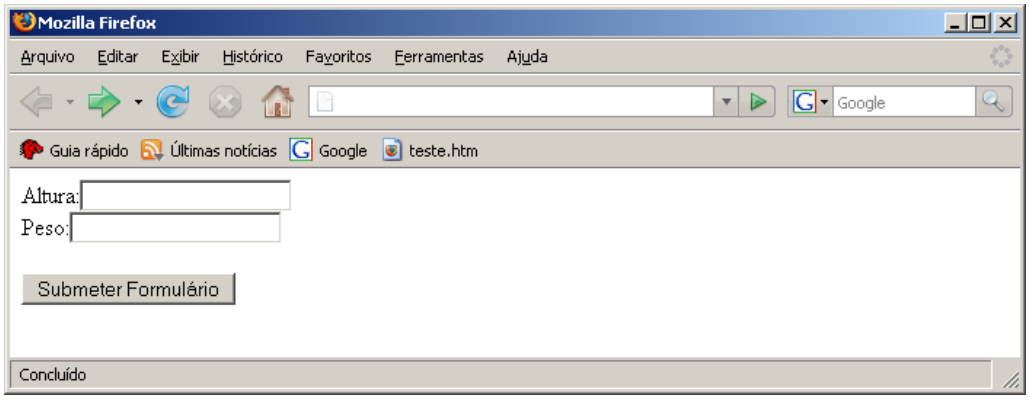
Mostrar o nome do método que será usado para enviar os dados do formulário

```
<html>
<head>
<script type="text/javascript">
   function formMethod()
   {
      var f=document.getElementById("myForm")
      alert("O método é: "+f.method)
   }
</script>
<title></title>
</head>
<body>
   <form id="myForm" method="post">
      Nome: <input type="text"><br><br>
      <input onclick="formMethod()" type="button"
      value="Mostrar o Método de Envio">
   </form>
</body>
</html>
```



Enviar os dados do formulário para serem processados

```
<html>
<head>
<script type="text/javascript">
   function submeter()
   {
      document.getElementById("myForm").submit()
   }
</script>
<title></title>
</head>
<body>
   <form id="myForm" action="processar.html" method="get">
      Altura:<input name="altura"><br>
      Peso:<input name="peso"><br><br>
      <input onclick="submeter()" type="button" value="Submeter Formulário">
   </form>
</body>
</html>
```



11.7 Objeto iframe

Este objeto representa de forma completa os elementos criados com a etiqueta <iframe> do HTML. A coleção frames, pertencente ao objeto window, contém uma lista com todos os objetos deste tipo que estão na página.

Propriedades

Propriedade	Descrição
align	Esta propriedade corresponde ao atributo align do elemento <iframe> do HTML e funciona do mesmo modo que o atributo

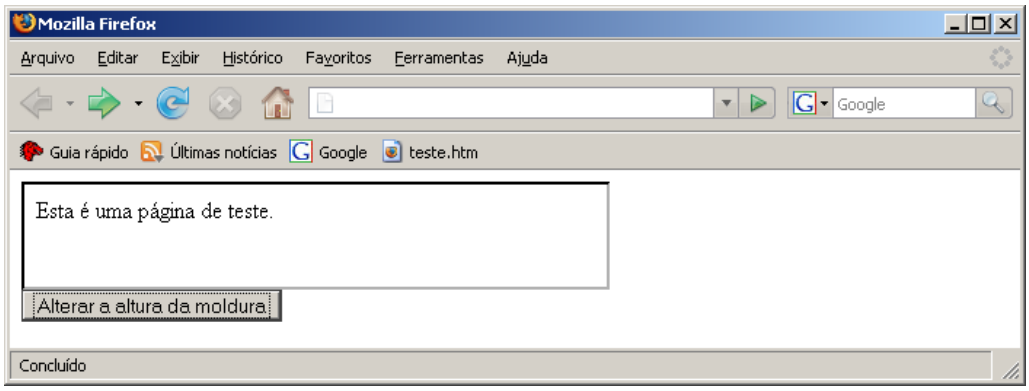
	align do elemento <img>. Ela alinha o objeto (tanto na vertical como na horizontal) relativamente ao texto que o rodeia.
frameBorder	Devolve o valor do contorno (border) do objeto iframe
height	Lê ou define o valor da altura do iframe
longDesc	Lê ou define o endereço da página que contém uma descrição extensa do conteúdo do iframe
marginHeight	Lê ou define a altura das margens do iframe
marginWidth	Lê ou define a largura das margens do iframe
name	Lê o valor do atributo name do iframe
scrolling	Lê ou define o valor do atributo scrolling, o qual controla o aparecimento ou a ocultação das barras de deslocamento do documento (scrollbars)
src	Lê ou define o endereço do documento que é mostrado dentro do iframe
width	Lê ou define o valor da largura do iframe

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <iframe> em acordo com a linguagem HTML.

Exemplos de Aplicação

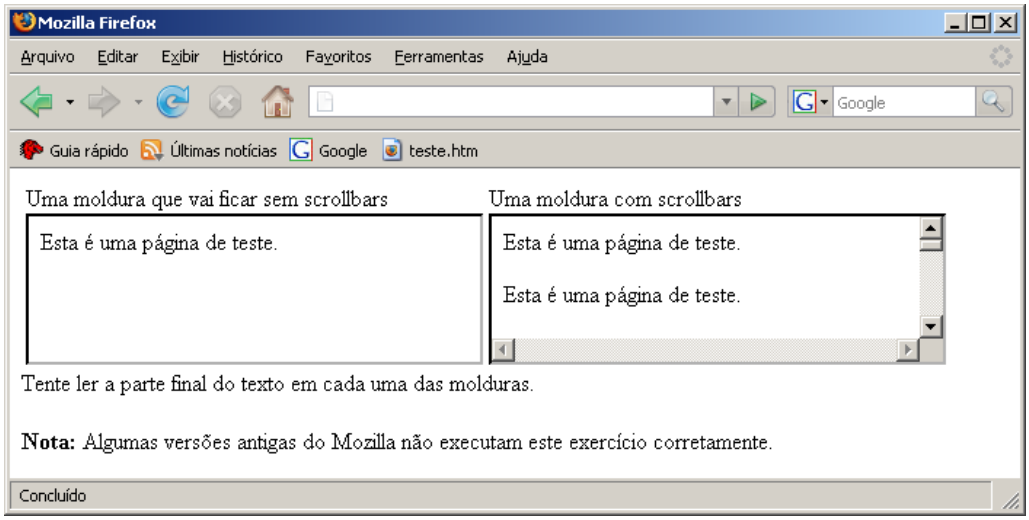
Alterar a altura do iframe

```
<html>
<head>
<script type="text/javascript">
   function changeHeight()
   {
      var frame=document.getElementById("myIframe")
      if(frame.height==70)
         frame.height=140
      else
         frame.height=70
   }
</script>
<title></title>
</head>
<body>
   <iframe id="myIframe" height="100" width="400" src="pagina.htm"></iframe>
   <form>
      <input onclick="changeHeight()"
      type="button" value="Alterar a altura da moldura">
   </form>
</body>
</html>
```



Scrollbars

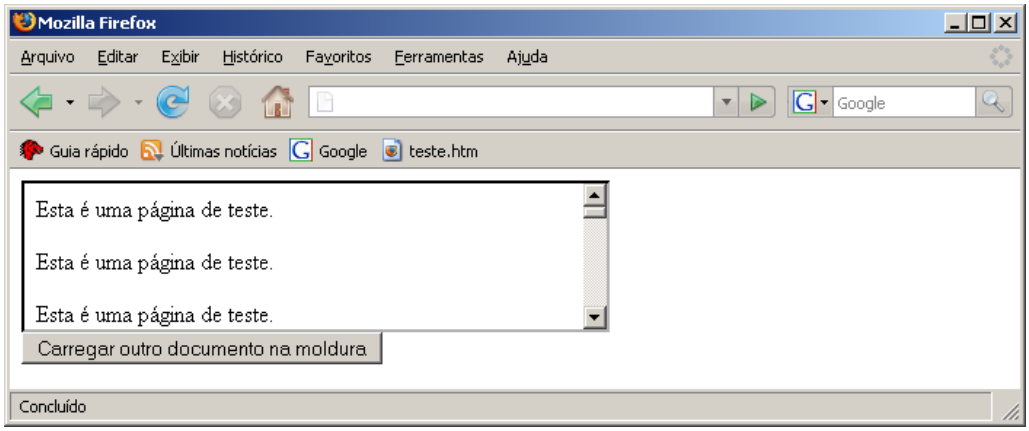
```
<html>
<head>
<title></title>
<script type="text/javascript">
   function retirarScrollbars()
   {
      // Agora eliminamos as scrollbars da primeira moldura.
      document.getElementById("myIframe1").scrolling="no"
   }
</script>
</head>
<body onload="retirarScrollbars()">
   <table width="620">
      <tr>
         <td>
            Uma moldura que vai ficar sem scrollbars
            <iframe id="myIframe1" height="100" width="310" scrolling="yes"
            src="pagina.html"></iframe>
         </td>
         <td>
            Uma moldura com scrollbars
            <iframe id="myIframe2" height="100" width="310" scrolling="yes"
            src="pagina2.html"></iframe>
         </td>
      </tr>
   </table>
   Tente ler a parte final do texto em cada uma das molduras.<br><br>
   <b>Nota:</b> Algumas versões antigas do Mozilla não executam
   este exercício corretamente.
</body>
</html>
```



Escolher outro documento para ser mostrado

```
<html>
<head>
<script type="text/javascript">
   function load()
   {
      var frame=document.getElementById("myIframe")
      frame.src="pagina2.html"
   }
</script>
<title></title>
</head>
<body>
   <iframe id="myIframe" height="100" width="400" src="pagina.html"></iframe>
   <form>
```

```
      <input onclick="load()"
      type="button" value="Carregar outro documento na moldura">
   </form>
</body>
</html>
```



11.8 Objeto Image

Este objeto representa de forma completa os elementos criados com a etiqueta <img> do HTML. A coleção images, pertencente ao objeto document, contém uma lista com todos os objetos deste tipo que estão na página.

Propriedades

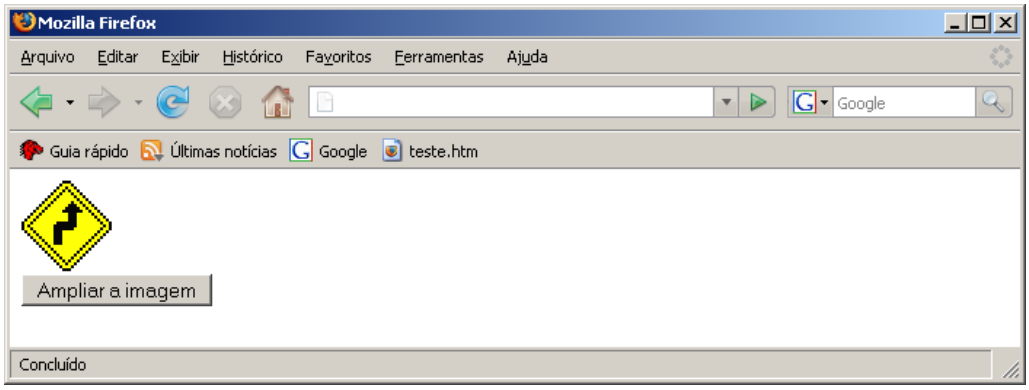
Propriedade	Descrição
align	Esta propriedade corresponde ao atributo align do elemento <img> do HTML. Ela alinha o objeto (tanto na vertical como na horizontal) relativamente ao texto que o rodeia.
alt	Lê ou define o valor do texto alternativo que será mostrado quando a imagem não puder ser carregada.
border	Atributo border. Espessura da linha de contorno que é desenhada ao redor da imagem.
height	Lê ou define a altura da imagem.
longDesc	Lê ou define o endereço de uma página que contém uma descrição extensa do conteúdo da imagem
lowSrc	Lê ou define o endereço de uma versão da imagem em baixa resolução
name	Lê ou define o o valor do atributo name.
src	Lê ou define o endereço (URL) de onde provém a imagem.
useMap	Se o elemento que corresponde a este objeto tiver sido dividido em partes usando o elemento <map>, então esta propriedade contém o valor true, no caso contrário contém false. Veja o atributo usemap do elemento que no HTML corresponde a este objeto.
width	Lê ou define a largura da imagem.

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <img> em acordo com a linguagem HTML.

Exemplos de Aplicação

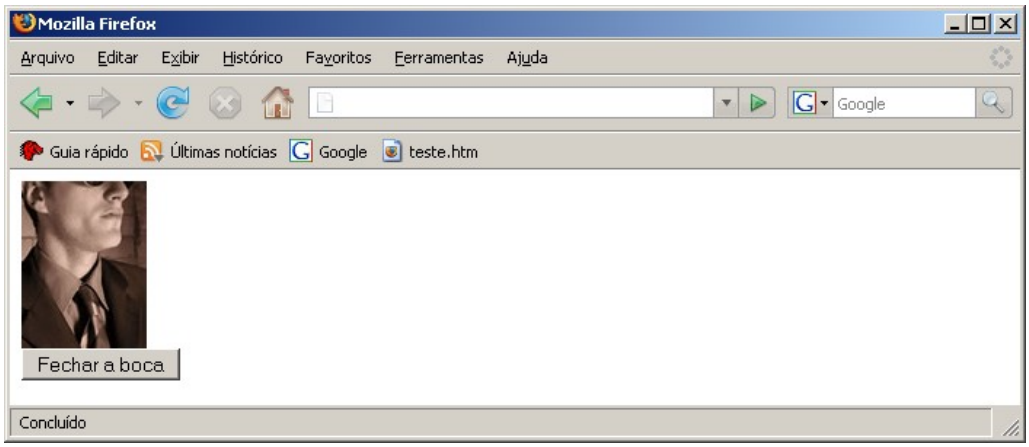
Alterar as dimensões de uma imagem

```
<html>
<head>
<script type="text/javascript">
   function setHeight()
   {
      var x=document.images
      x[0].height=64
      x[0].width=64
   }
</script>
<title></title>
</head>
<body>
   <img height="32" src="5103.gif" width="32">
   <form>
      <input onclick="setHeight()" type="button" value="Ampliar a imagem">
   </form>
</body>
</html>
```



Trocar uma imagem por outra

```
<html>
<head>
<script type="text/javascript">
   function setSrc()
   {
      var x=document.images
      x[0].src="fechada.jpg"
   }
</script>
<title></title>
</head>
<body>
   <img src="aberta.jpg">
   <form>
      <input onclick="setSrc()" type="button" value="Fechar a boca">
   </form>
</body>
</html>
```



Pré-carregamento de imagens em JavaScript

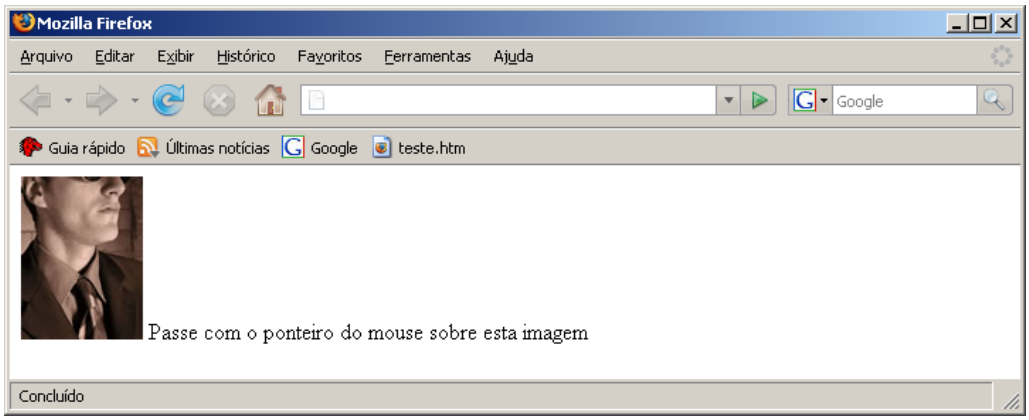
Uma das aplicações mais antigas e mais populares do objeto Image consiste em usá-lo para carregar na memória do computador uma imagem que só mais tarde será visualizada. Isto é útil em efeitos de roll-over, em que uma imagem é trocada por outra instantâneamente quando o ponteiro do mouse passa por cima de um elemento.

Para produzirmos esse efeito criamos um novo objeto Image e nele carregamos a imagem que queremos fazer aparecer instantâneamente, assim:

```
var img = new Image()
img.src="fechada.jpg"
```

Quando executar este código o browser carrega a imagem "fechada.jpg" em memória, que passa a estar pronta para ser mostrada rapidamente quando for necessário. O exemplo seguinte mostra como isto funciona:

```
<html>
<body>
<script type="text/javascript">
   var img = new Image()
   img.src="fechada.jpg"
   function fechar(o)
   {
      o.src=img.src
   }
</script>
   <img src="aberta.jpg" onmouseover="fechar(this)">
   Passe com o ponteiro do mouse sobre esta imagem
</body>
</html>
```



11.9 Objeto input

Este objeto representa de forma completa os elementos criados com a etiqueta <input> do HTML e pode assumir os seguintes tipos: button, checkbox, file, hidden, image, password, radio, reset, submit, ou text.

Propriedades

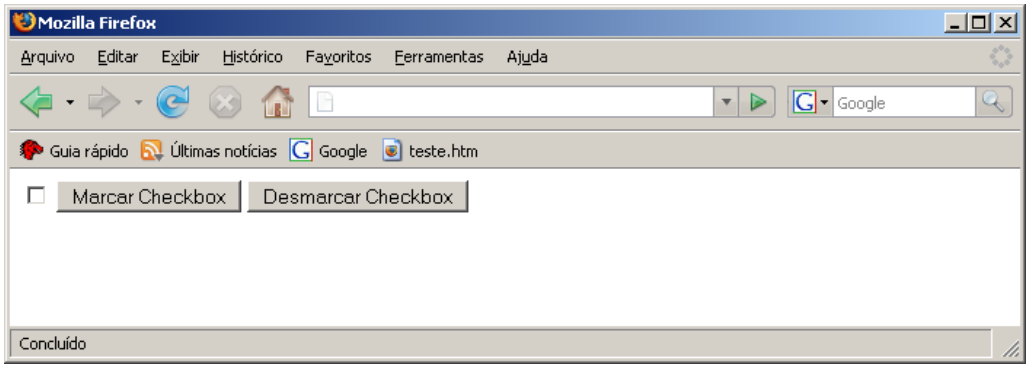
Propriedade	Descrição
accessKey	Define uma tecla para acessar rapidamente um elemento input
accept	Fornece uma lista com os tipos de conteúdo que o servidor encarregado de tratar o formulário consegue processar corretamente
align	Define o alinhamento do elemento.
alt	Lê ou define o valor do texto alternativo que será mostrado no caso de o browser não conseguir mostrar o elemento.
checked	Lê ou define um valor lógico (true ou false) que indica se a checkbox ou o radiobutton devem ficar no estado checked (validado) ou unchecked (não validado)
defaultValue	Lê ou define os dados iniciais (valor por omissão) contidos no elemento input
defaultChecked	Dá o valor true à propriedade checked (funciona com radiobuttons e checkboxes)
disabled	Lê ou define um valor lógico (true ou false) que indica se o elemento deve ser desativado ou se deve permanecer ativo
form	Devolve o objeto form em que este está contido
name	Lê ou define o nome do elemento
maxLength	Lê ou define o número máximo de caracteres que podem ser escritos no elemento
readOnly	Lê ou define um valor lógico (true ou false) que indica se o elemento pode ser apenas lido ou se também pode escrever-se o seu valor (funciona com type="text" e type="password")
size	Lê ou define o número de caracteres que podem ser vistos ao mesmo tempo
src	Quando se tem type="image" lê ou define o endereço (URL) da imagem
tabIndex	Lê ou define o índice (número de ordem) do elemento que corresponde a este objeto no acesso através da tecla tab
type	Lê o tipo (valor do atributo type) do elemento
value	Lê ou define o valor de um checkbox ou de um radiobutton

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <input> em acordo com a linguagem HTML.

Exemplos de Aplicação

Controlar o estado de um checkbox (caixa de validação)

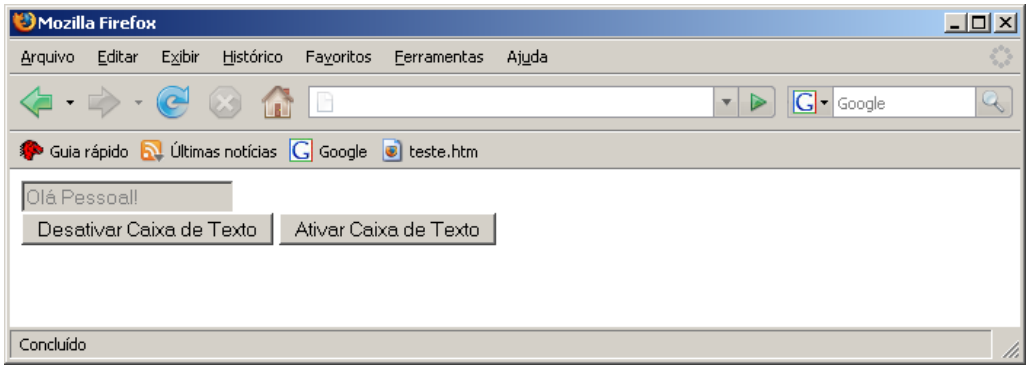
```
<html>
<head>
<script type="text/javascript">
   function check()
   {
      var x=document.getElementById("myInput")
      x.checked=true
   }

   function uncheck()
   {
      var x=document.getElementById("myInput")
      x.checked=false
   }
</script>
<title></title>
</head>
<body>
   <form>
      <input type="checkbox" value="on" id="myInput">
      <input onclick="check()" type="button" value="Marcar Checkbox">
      <input onclick="uncheck()" type="button" value="Desmarcar Checkbox">
   </form>
</body>
</html>
```



Desativar ou ativar um campo do formulário

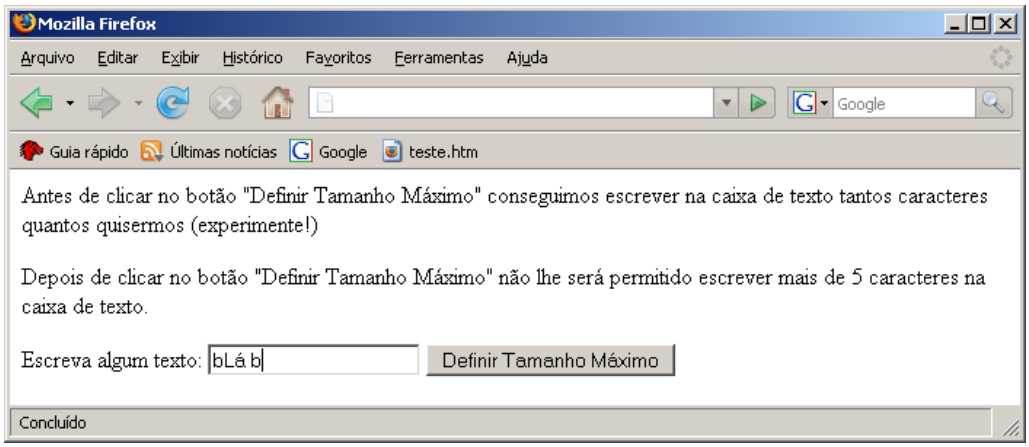
```
<html>
<head>
<script type="text/javascript">
   function makeDisable()
   {
      var x=document.getElementById("myInput")
      x.disabled=true
   }

   function makeEnable()
   {
      var x=document.getElementById("myInput")
      x.disabled=false
   }
</script>
<title></title>
</head>
```

```
<body>
   <form>
      <input value="Olá Pessoal!" id="myInput"><br>
      <input onclick="makeDisable()" type="button"
      value="Desativar Caixa de Texto">
      <input onclick="makeEnable()" type="button"
      value="Ativar Caixa de Texto">
   </form>
</body>
</html>
```



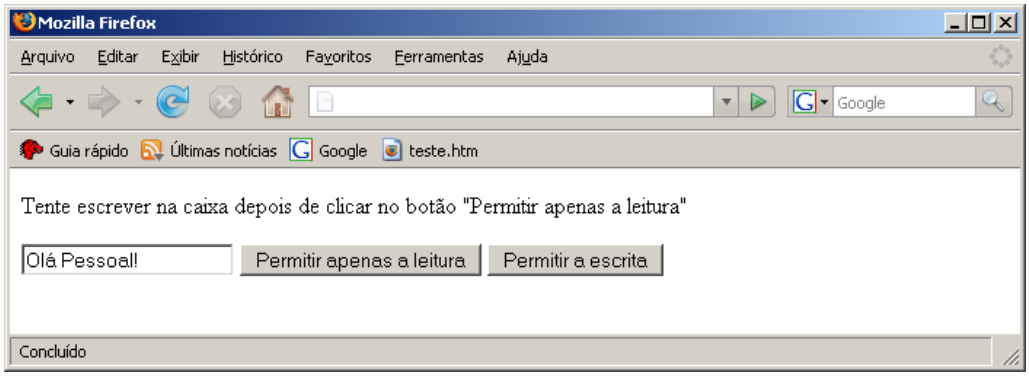
Definir o número máximo de caracteres que podem ser escritos num campo de texto

```
<html>
<head>
<script type="text/javascript">
   function maximum()
   {
      var x=document.getElementById("myInput")
      x.maxLength="5"
   }
</script>
<title></title>
</head>
<body>
   <p>
      Antes de clicar no botão "Definir Tamanho Máximo"
      conseguimos escrever na caixa de texto tantos
      caracteres quantos quisermos (experimente!)
    </p>
   <p>
      Depois de clicar no botão "Definir Tamanho Máximo"
      não lhe será permitido escrever mais de 5 caracteres
      na caixa de texto.
   </p>
   <form>
      Escreva algum texto: <input id="myInput">
      <input onclick="maximum()" type="button" value="Definir Tamanho Máximo">
    </form>
</body>
</html>
```



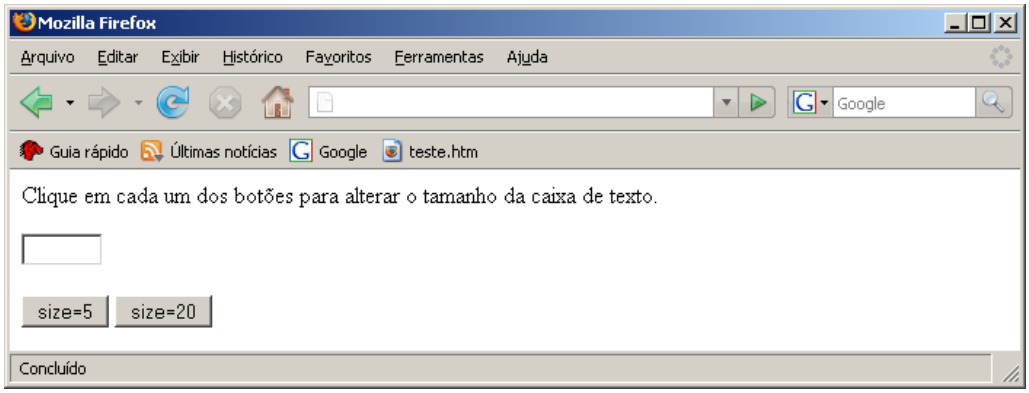
Fazer com que um campo possa apenas ser lido

```
<html>
<head>
<script type="text/javascript">
   function makeReadOnly()
   {
      var x=document.getElementById("myInput")
      x.readOnly=true
   }

   function delReadOnly()
   {
      var x=document.getElementById("myInput")
      x.readOnly=false
   }
</script>
<title></title>
</head>
<body>
   <form>
      <p>
         Tente escrever na caixa depois de clicar no botão
         "Permitir apenas a leitura"
      </p>
      <input value="Olá Pessoal!" id="myInput">
      <input onclick="makeReadOnly()" type="button"
      value="Permitir apenas a leitura">
      <input onclick="delReadOnly()" type="button" value="Permitir a escrita">
   </form>
</body>
</html>
```



Alterar o tamanho de um campo

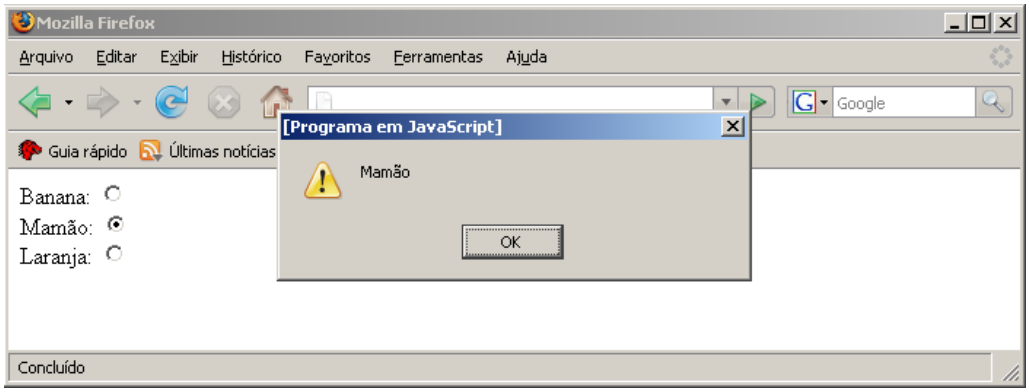
```
<html>
<head>
<script type="text/javascript">
   function sizeCinco()
   {
      var x=document.getElementById("myInput")
      x.size=5
   }

   function sizeVinte()
   {
      var x=document.getElementById("myInput")
      x.size=20
   }
</script>
<title></title>
</head>
<body>
   <p>
      Clique em cada um dos botões para alterar
      o tamanho da caixa de texto.
   </p>
   <form>
      <input id="myInput" type="text"><br><br>
      <input onclick="sizeCinco()" type="button" value="size=5">
      <input onclick="sizeVinte()" type="button" value="size=20">
   </form>
</body>
</html>
```



Ler e manipular o estado de um radiobutton

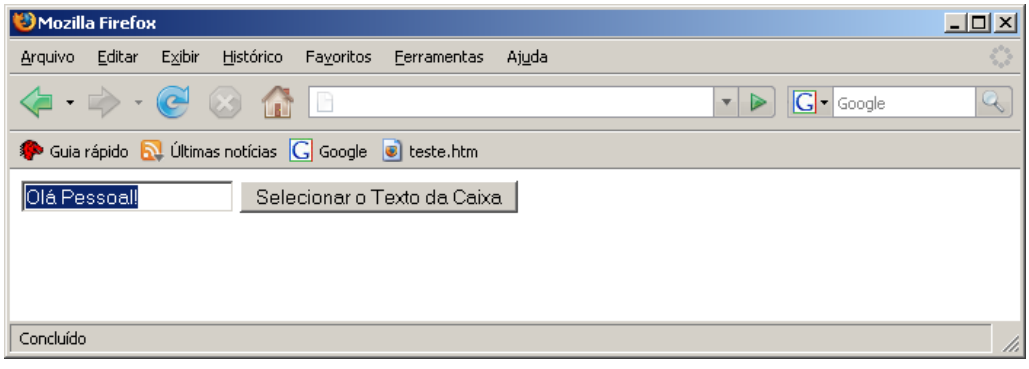
```
<html>
<head>
<script type="text/javascript">
   function fruit(index)
   {
      var x=document.getElementById("myForm")
      alert(x[index].value)
   }
</script>
<title></title>
</head>
<body>
   <form id="myForm">
      Banana:
      <input onclick="fruit(0)" type="radio" value="Banana" name="fruta"><br>
      Mamão:
      <input onclick="fruit(1)" type="radio" value="Mamão" name="fruta"><br>
      Laranja:
```

```
      <input onclick="fruit(2)" type="radio" value="Laranja" name="fruta">
   </form>
</body>
</html>
```



Selecionar o texto de um campo do formulário

```
<html>
<head>
<script type="text/javascript">
   function makeSelect()
   {
      var x=document.getElementById("myInput")
      x.select()
      x.focus()
   }
</script>
<title></title>
</head>
<body>
   <form>
      <input value="Olá Pessoal!" id="myInput">
      <input onclick="makeSelect()" type="button"
      value="Selecionar o Texto da Caixa">
   </form>
</body>
</html>
```



11.10 Objeto object

Este objeto representa de forma completa os elementos criados com a etiqueta <object> do HTML.

Propriedades

Propriedade	Descrição
form	Se o objeto estiver contido num elemento <form> devolve o

	objeto que corresponde a esse elemento <form>, caso contrário devolve null
code	Se o elemento <object> estiver sendo usado para inserir uma miniaplicação Java (em vez do elemento <applet>, que foi rejeitado nas versões modernas do HTML) este atributo contém o nome da classe Java na qual se deve iniciar a execução do código. Nos outros casos contém null.
align	Corresponde ao atributo align
archive	Corresponde ao atributo archive. Contém os nomes de um ou mais arquivos (separados por vírgulas) que guardam código a executar pelo objeto.
border	Corresponde ao atributo border
codeBase	Corresponde ao atributo codebase
codeType	Corresponde ao atributo codetype
data	Corresponde ao atributo data, que fornece o endereço de um recurso contendo dados que o objeto consome
declare	Propriedade ainda não utilizada mas que está reservada para uso no futuro
height	Corresponde ao atributo height
name	Corresponde ao atributo name
standby	Corresponde ao atributo standby
tabIndex	Lê ou define o índice (número de ordem) do elemento que corresponde a este objeto no acesso através da tecla tab
type	Corresponde ao atributo type
useMap	Se o elemento que corresponde a este objeto tiver sido dividido em partes usando o elemento <map> então esta propriedade contém o valor true, no caso contrário contém false. Veja o atributo usemap do elemento que no HTML corresponde a este objeto.
width	Corresponde ao atributo width

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <object> em acordo com a linguagem HTML.

11.11 Objeto option

Este objeto representa de forma completa os elementos criados com a etiqueta <option> do HTML.

Propriedades

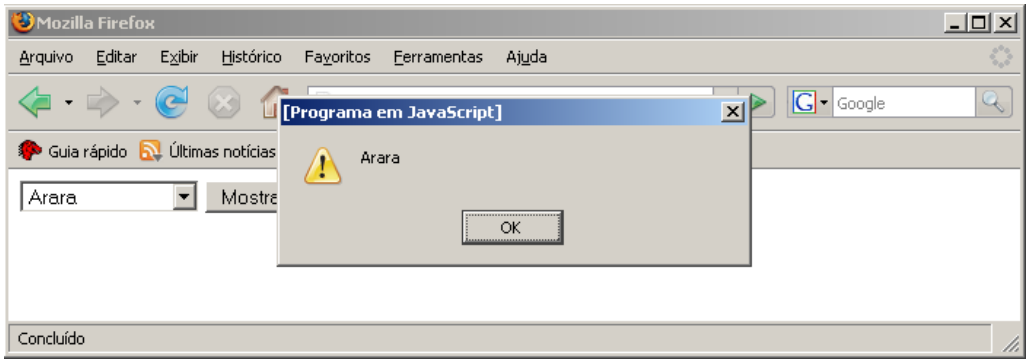
Propriedade	Descrição
defaultSelected	Indica que esta é a opção selecionada na partida
disabled	Lê ou define um valor lógico (true ou false) que indica se a opção deve ser desativada ou se deve permanecer ativa
form	Devolve o objeto form em que este está contido
index	Devolve o índice da opção relativamente ao elemento que a contém
label	Lê ou define o nome dado à opção
selected	Faz com que a opção passe ou não (valores true ou false) a ser a que está selecionada
text	Lê o texto correspondente à opção
value	Lê ou define o valor da opção (atributo value)

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <option> em acordo com a linguagem HTML.

Exemplos de Aplicação

Mostrar o texto correspondente à opção selecionada

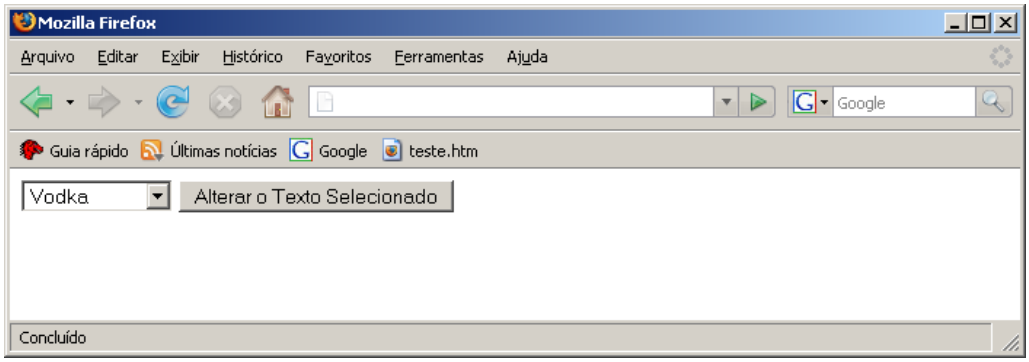
```
<html>
<head>
<script type="text/javascript">
   function formAction()
   {
      var x=document.forms.myForm.mySelect
      alert(x.options[x.selectedIndex].text)
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Papagaio</option>
         <option>Arara</option>
         <option>Ave do Paraíso</option>
      </select>
      <input onclick="formAction()"
      type="button" value="Mostrar o Texto da Opção Escolhida">
   </form>
</body>
</html>
```



Modificar o texto da opção

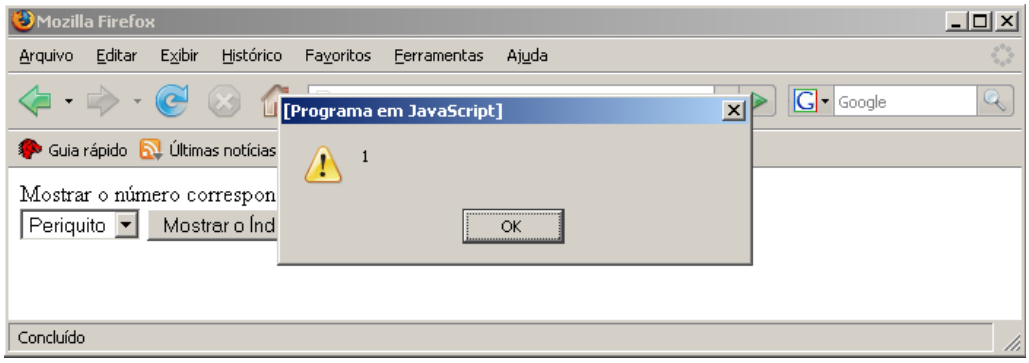
```
<html>
<head>
<script type="text/javascript">
   function optionText()
   {
      var x=document.forms.myForm.mySelect
      x.options[x.selectedIndex].text="Vodka"
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Água</option>
         <option>Refrigerante</option>
         <option>Leite</option>
      </select>
      <input onclick="optionText()"
```

```
      type="button" value="Alterar o Texto Selecionado">
   </form>
</body>
</html>
```



Mostrar o número correspondente à opção selecionada

```
<html>
<head>
<script type="text/javascript">
   function optionIndex()
   {
      var x=document.forms.myForm.mySelect
      alert(x.options[x.selectedIndex].index)
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Canário</option>
         <option>Periquito</option>
         <option>Araro</option>
      </select>
      <input onclick="optionIndex()" type="button" value="Mostrar o Índice">
   </form>
</body>
</html>
```



11.12 Objeto select

Este objeto representa de forma completa os elementos criados com a etiqueta <select> do HTML.

Propriedades

Propriedade	Descrição

disabled	Lê ou define o valor da propriedade disabled, que determina se a opção deve ser desativada ou se deve permanecer ativa
form	Devolve o elemento form que contém o elemento
length	Devolve o número de opções contidas dentro do elemento
multiple	Define e permite saber se é possível selecionar várias opções em simultâneo
name	Lê ou define o nome do elemento
selectedIndex	Lê o número correspondente à opção selecionada
size	Lê ou define o número de opções que podem ser vistas ao mesmo tempo
tabIndex	Lê ou define o índice (número de ordem) do elemento que corresponde a este objeto no acesso através da tecla tab
type	Lê o tipo (valor do atributo type) do elemento

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <select> em acordo com a linguagem HTML.

Coleções

Coleção	Descrição
options	Contém uma lista com todas as opções contidas dentro do elemento (elementos <option>)

Métodos

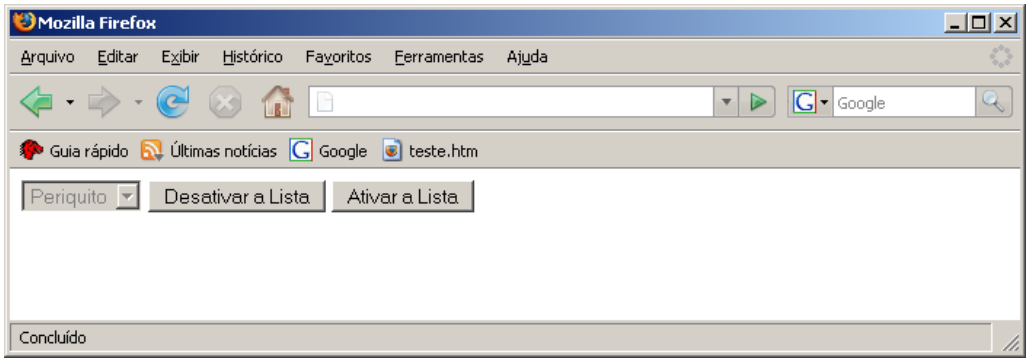
Método	Descrição
add()	Acrescenta uma nova opção à coleção de opções
remove()	Remove uma opção da coleção de opções
blur()	O elemento perde o foco
focus()	O elemento ganha o foco

Exemplos de Aplicação

Desativar e ativar um elemento select

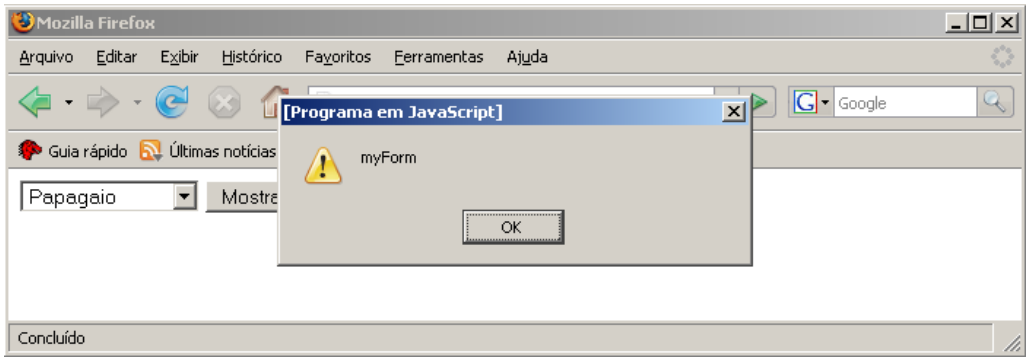
```
<html>
<head>
<script type="text/javascript">
   function makeDisable()
   {
      var x=document.forms.myForm.mySelect
      x.disabled=true
   }

   function makeEnable()
   {
      var x=document.forms.myForm.mySelect
      x.disabled=false
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Canário</option>
         <option>Periquito</option>
         <option>Arara</option>
      </select>
```

```
      <input onclick="makeDisable()" type="button" value="Desativar a Lista">
      <input onclick="makeEnable()" type="button" value="Ativar a Lista">
    </form>
</body>
</html>
```



Ler o nome do formulário que contém o elemento select

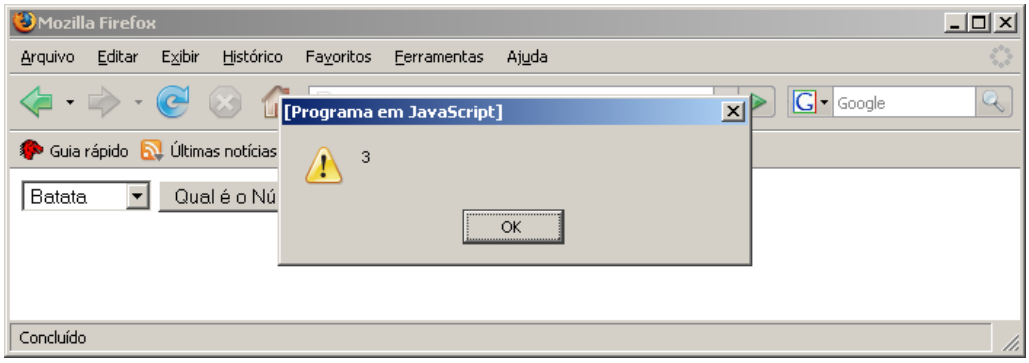
```
<html>
<head>
<script type="text/javascript">
   function formAction()
   {
      var x=document.forms.myForm.mySelect
      alert(x.form.name)
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Papagaio</option>
         <option>Arara</option>
         <option>Ave do Paraíso</option>
      </select>
      <input onclick="formAction()"
      type="button" value="Mostrar o Nome do Formulário">
    </form>
</body>
</html>
```



Mostrar o número de opções que estão no elemento select

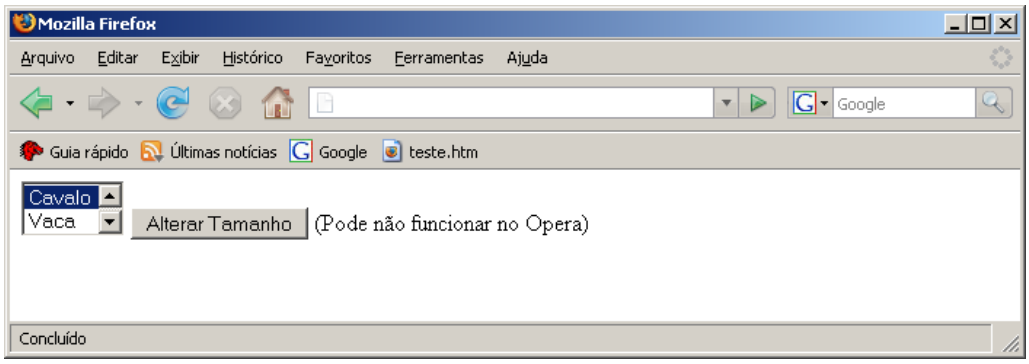
```
<html>
<head>
<script type="text/javascript">
   function formAction()
```

```
   {
      var x=document.forms.myForm.mySelect
      alert(x.length)
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Batata</option>
         <option>Feijão</option>
         <option>Mandioca</option>
      </select>
      <input onclick="formAction()"
      type="button" value="Qual é o Número de Opções?">
   </form>
</body>
</html>
```



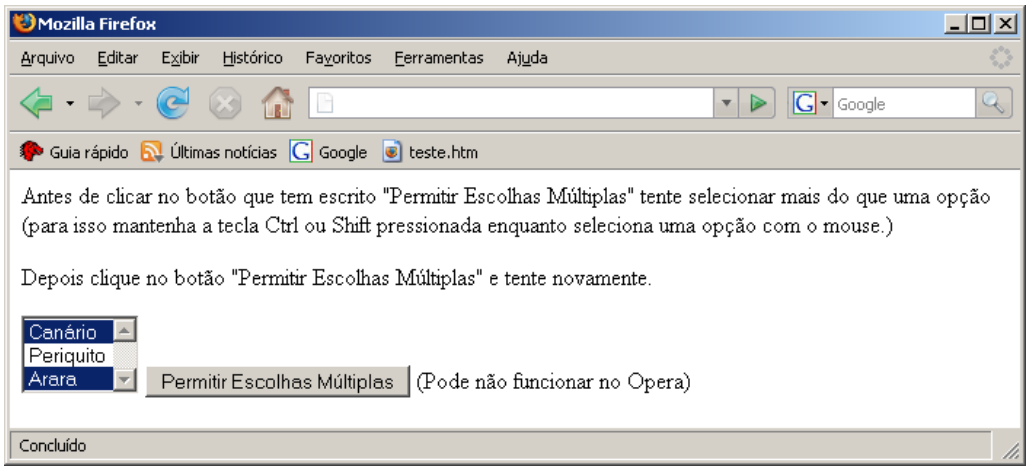
Especificar o número de opções que podem ser vistas ao mesmo tempo

```
<html>
<head>
<script type="text/javascript">
   function formAction()
   {
      var x=document.forms.myForm.mySelect
      x.size="2"
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Cavalo</option>
         <option>Vaca</option>
         <option>Galo</option>
      </select>
      <input onclick="formAction()" type="button"
      value="Alterar Tamanho">
      (Pode não funcionar no Opera)
   </form>
</body>
</html>
```



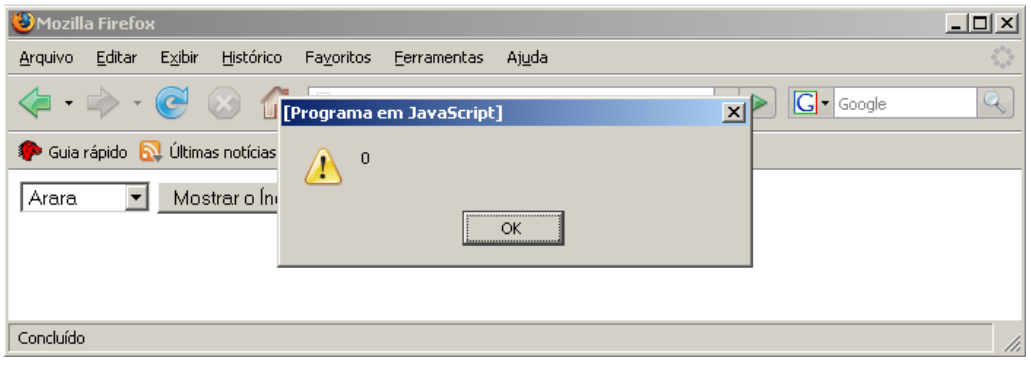
Selecionar várias opções ao mesmo tempo

```
<html>
<head>
<script type="text/javascript">
   function formAction()
   {
      var x=document.forms.myForm.mySelect
      x.multiple=true
   }
</script>
<title></title>
</head>
<body>
   <p>
      Antes de clicar no botão que tem escrito
      "Permitir Escolhas Múltiplas" tente selecionar
      mais do que uma opção (para isso mantenha a
      tecla Ctrl ou Shift pressionada enquanto seleciona
      uma opção com o mouse.)
   </p>
   <p> Depois clique no botão
   "Permitir Escolhas Múltiplas" e tente novamente.
   </p>
   <form name="myForm">
      <select size="3" name="mySelect">
         <option>Canário</option>
         <option>Periquito</option>
         <option>Arara</option>
      </select>
      <input onclick="formAction()" type="button"
      value="Permitir Escolhas Múltiplas">
      (Pode não funcionar no Opera)
    </form>
</body>
</html>
```



Mostrar o número correspondente à opção selecionada

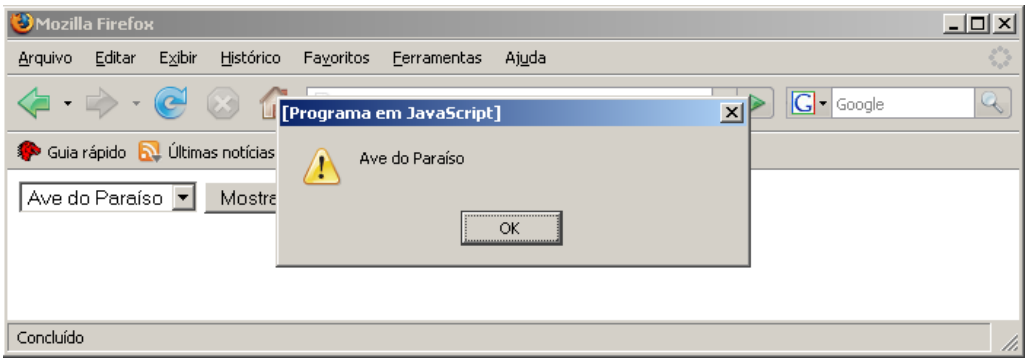
```
<html>
<head>
<script type="text/javascript">
   function formAction()
   {
      var x=document.forms.myForm.mySelect
      alert(x.selectedIndex)
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Arara</option>
         <option>Papagaio</option>
         <option>Canário</option>
      </select>
      <input onclick="formAction()" type="button" value="Mostrar o Índice">
   </form>
</body>
</html>
```



Mostrar o texto correspondente à opção selecionada

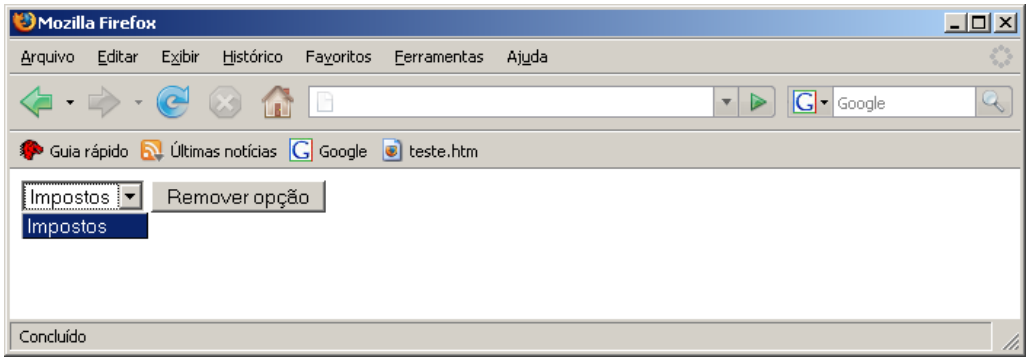
```
<html>
<head>
<script type="text/javascript">
   function formAction()
   {
      var x=document.forms.myForm.mySelect
      alert(x.options[x.selectedIndex].text)
   }
```

```
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Papagaio</option>
         <option>Arara</option>
         <option>Ave do Paraíso</option>
      </select>
      <input onclick="formAction()"
      type="button" value="Mostrar o Texto da Opção Escolhida">
   </form>
</body>
</html>
```



Remover uma opção

```
<html>
<head>
<script type="text/javascript">
   function formAction()
   {
      var x=document.forms.myForm.mySelect
      x.remove(x.selectedIndex)
   }
</script>
<title></title>
</head>
<body>
   <form name="myForm">
      <select name="mySelect">
         <option selected="selected">Dívidas</option>
         <option>Despesas</option>
         <option>Impostos</option>
      </select>
      <input onclick="formAction()" type="button" value="Remover opção">
  </form>
</body>
</html>
```



11.13 Objeto table

Este objeto representa de forma completa os elementos criados com a etiqueta <table> do HTML.

Propriedades

Propriedade	Descrição
border	Lê ou define a espessura da linha de contorno da tabela
caption	Lê a legenda de uma tabela
cellPadding	Lê ou define a quantidade de espaço em branco que separa a fronteira da célula do seu conteúdo
cellSpacing	Lê ou define a quantidade de espaço em branco que separa as células umas das outras
frame	Indica quais as linhas de contorno que devem ser desenhadas (por cima, por baixo, ...)
rules	Especifica quais as linhas de contorno que devem ser desenhadas (nas linhas, nas colunas, ...)
summary	Lê ou define uma descrição sumária da tabela
tFoot	Devolve o elemento tfoot (rodapé) da tabela
tHead	Devolve o elemento thead (cabeçalho) da tabela
width	Lê ou define a largura da tabela

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <table> ou está intimamente relacionada com elementos que só podem existir como parte de uma tabela.

Coleções

Coleção	Descrição
rows	Contém uma lista com todas as linhas de uma tabela
tBodies	Contém uma lista com todos os elementos tbody da tabela

Métodos

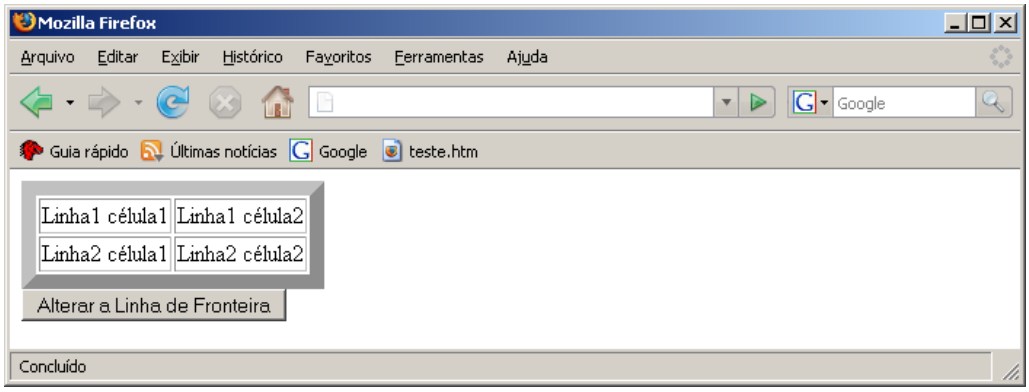
Método	Descrição
createCaption()	Cria uma legenda para a tabela ou lê a legenda que está definida
createTFoot()	Cria um rodapé para a tabela ou lê o rodapé que está definido
createTHead()	Cria um cabeçalho para a tabela ou lê o cabeçalho que está definido
deleteCaption()	Apaga a legenda da tabela
deleteTFoot()	Apaga o rodapé da tabela
deleteTHead()	Apaga o cabeçalho da tabela
deleteRow()	Apaga a linha especificada da tabela
insertRow()	Insere uma nova linha na tabela na posição especificada

Exemplos de Aplicação

Alterar o contorno da tabela

```
<html>
<head>
<script type="text/javascript">
   function setBorder()
   {
      document.getElementById("myTable").border="10"
   }
```

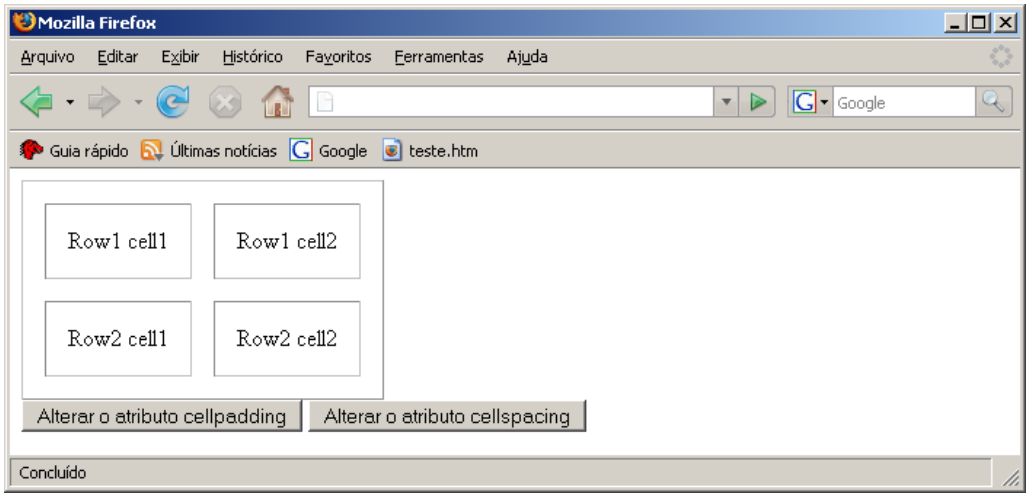
```
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="setBorder()" type="button"
      value="Alterar a Linha de Fronteira">
   </form>
</body>
</html>
```



Alterar os valores de cellPadding e cellSpacing

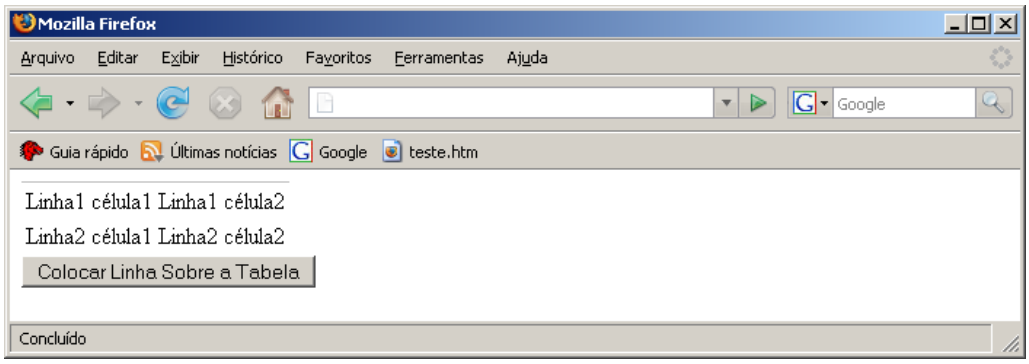
```
<html>
<head>
<script type="text/javascript">
   function padding()
   {
      document.getElementById("myTable").cellPadding="15"
   }

   function spacing()
   {
      document.getElementById("myTable").cellSpacing="15"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Row1 cell1</td>
            <td>Row1 cell2</td>
         </tr>
         <tr>
            <td>Row2 cell1</td>
            <td>Row2 cell2</td>
         </tr>
      </tbody>
```

```
   </table>
   <form>
      <input onclick="padding()" type="button"
      value="Alterar o atributo cellpadding">
      <input onclick="spacing()" type="button"
      value="Alterar o atributo cellspacing">
   </form>
</body>
</html>
```



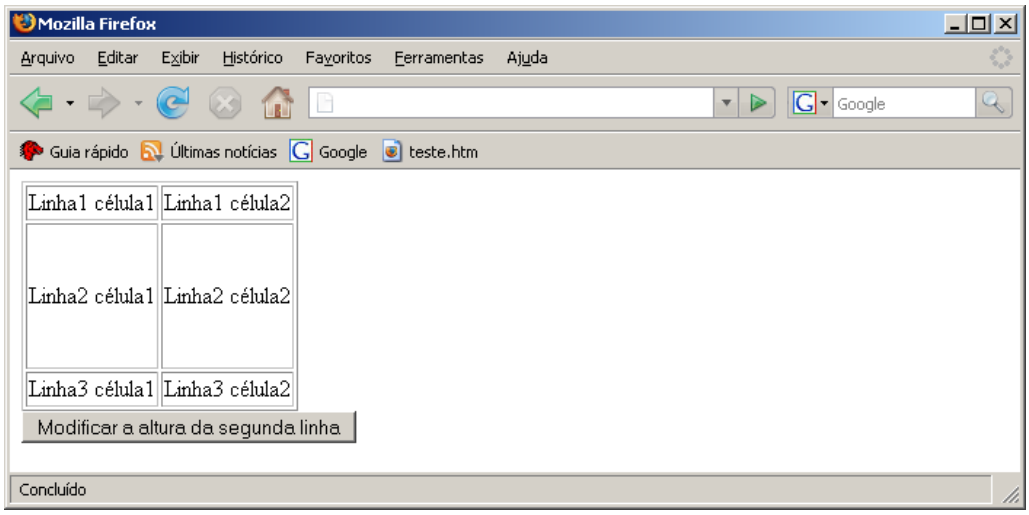
Especificar o contorno a aplicar

```
<html>
<head>
<script type="text/javascript">
   function setFrame()
   {
      document.getElementById("myTable").frame="above"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="setFrame()" type="button"
      value="Colocar Linha Sobre a Tabela">
   </form>
</body>
</html>
```



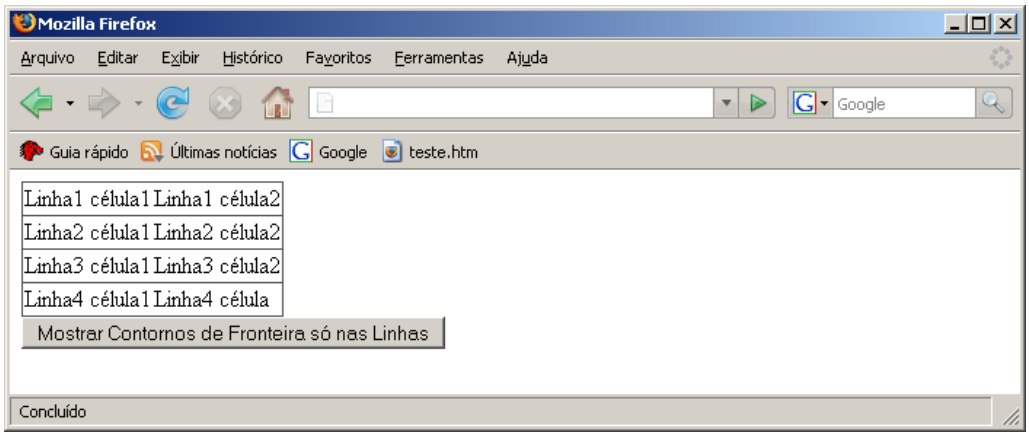
Alterar a altura da linha

```
<html>
<head>
<script type="text/javascript">
   function specifyRow()
   {
      var linhas=document.getElementById("myTable").rows
      linhas[1].style.height="100px"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
         <tr>
            <td>Linha3 célula1</td>
            <td>Linha3 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="specifyRow()" type="button"
      value="Modificar a altura da segunda linha">
   </form>
</body>
</html>
```



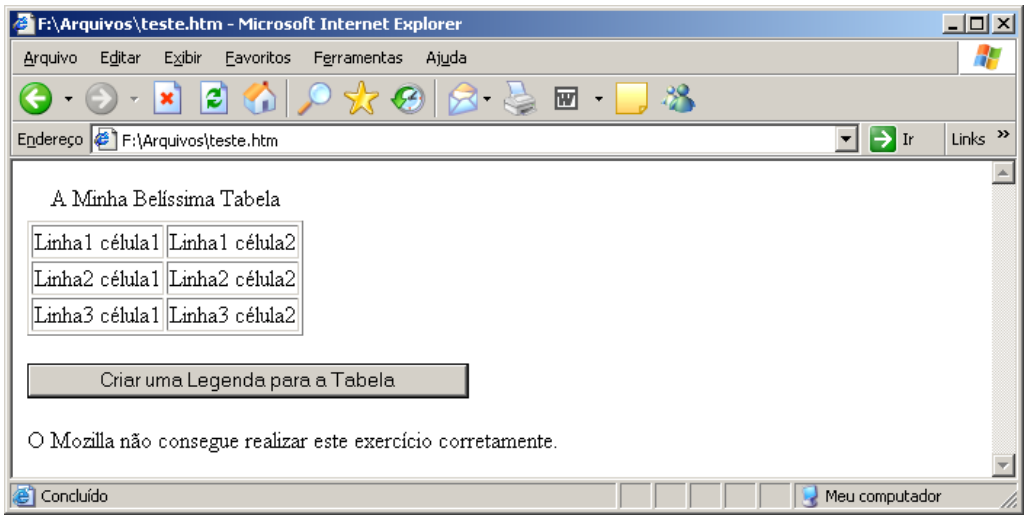
Especificar as linhas de contorno a desenhar

```
<html>
<head>
<script type="text/javascript">
   function setRules()
   {
      document.getElementById("myTable").rules="rows"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
         <tr>
            <td>Linha3 célula1</td>
            <td>Linha3 célula2</td>
         </tr>
         <tr>
            <td>Linha4 célula1</td>
            <td>Linha4 célula</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="setRules()" type="button"
      value="Mostrar Contornos de Fronteira só nas Linhas">
   </form>
</body>
</html>
```



Criar uma legenda para a tabela

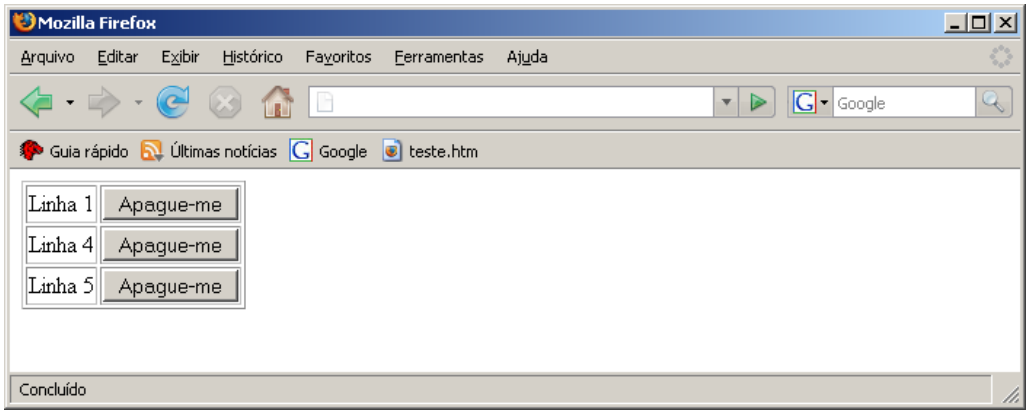
```
<html>
<head>
<script type="text/javascript">
   function caption()
   {
      var o=document.getElementById("myTable")
      var x=o.createCaption()
      x.innerText="A Minha Belíssima Tabela"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
         <tr>
            <td>Linha3 célula1</td>
            <td>Linha3 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="caption()" type="button"
      value="Criar uma Legenda para a Tabela">
   </form>
   <p>O Mozilla não consegue realizar este exercício corretamente.</p>
</body>
</html>
```



Apagar (remover) uma linha

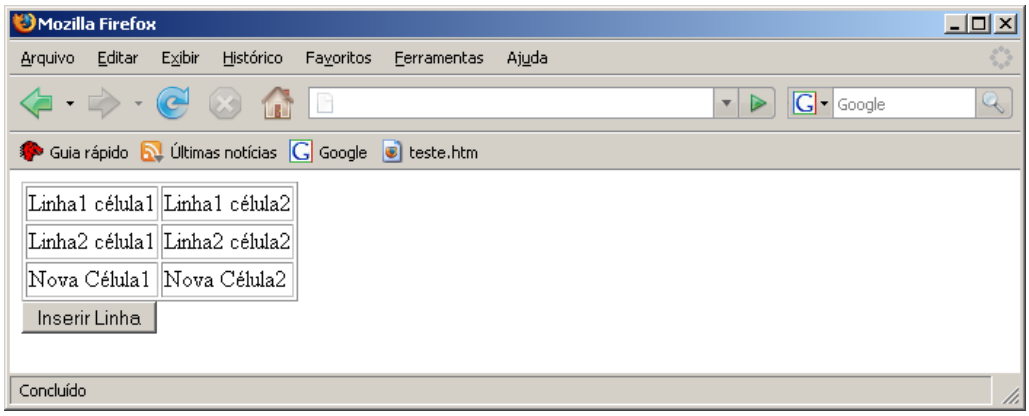
```
<html>
<head>
<script type="text/javascript">
   function deleteMe(rowIndex)
   {
      document.getElementById("myTable").deleteRow(rowIndex)
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Linha 1</td>
            <td>
               <input onclick="deleteMe(this.parentNode.parentNode.rowIndex)"
               type="button" value="Apague-me">
            </td>
         </tr>
         <tr>
            <td>Linha 2</td>
            <td>
               <input onclick="deleteMe(this.parentNode.parentNode.rowIndex)"
               type="button" value="Apague-me">
            </td>
         </tr>
         <tr>
            <td>Linha 3</td>
            <td>
               <input onclick="deleteMe(this.parentNode.parentNode.rowIndex)"
               type="button" value="Apague-me">
            </td>
         </tr>
         <tr>
            <td>Linha 4</td>
            <td>
               <input onclick="deleteMe(this.parentNode.parentNode.rowIndex)"
               type="button" value="Apague-me">
            </td>
         </tr>
         <tr>
            <td>Linha 5</td>
            <td>
               <input onclick="deleteMe(this.parentNode.parentNode.rowIndex)"
               type="button" value="Apague-me">
```

```
            </td>
         </tr>
      </tbody>
   </table>
</body>
</html>
```



Inserir uma linha

```
<html>
<head>
<script type="text/javascript">
   function insRow()
   {
      var o=document.getElementById("myTable")
      var x=o.insertRow(2)
      var y=x.insertCell(0)
      var z=x.insertCell(1)
      y.innerHTML="Nova Célula1"
      z.innerHTML="Nova Célula2"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="insRow()" type="button" value="Inserir Linha">
   </form>
</body>
</html>
```



11.14 Objeto tablecell

Este objeto representa elementos criados com as etiqueta <td> ou <th> do HTML. A coleção cells, pertencente ao objeto tablerow, contém uma lista com todos os objetos deste tipo existentes na linha de tabela que é representada pelo objeto tablerow.

Propriedades

Propriedade	Descrição
abbr	Lê ou define uma abreviação para os cabeçalhos das células
align	Define o alinhamento horizontal dos conteúdos das células
axis	Lê ou define o nome de um grupo de cabeçalhos
cellIndex	Lê o número de ordem (índice) de uma célula na linha a que pertence
colSpan	Lê ou define o valor do colspan aplicado a uma célula
rowSpan	Lê ou define o valor do rowspan aplicado a uma célula
vAlign	Define o alinhamento vertical dos conteúdos das células

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <td> em acordo com a linguagem HTML.

Coleções

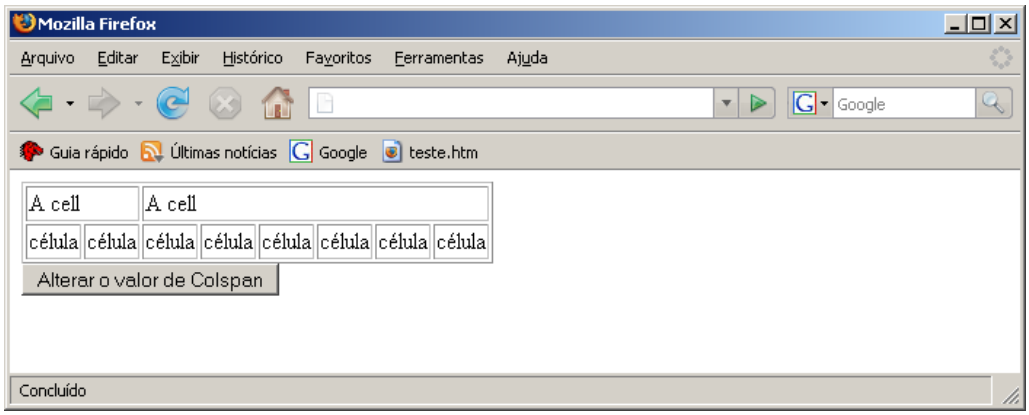
Coleção	Descrição
headers	Contém uma lista com os atributos id das células que têm cabeçalhos

Exemplos de Aplicação

Alterar o valor do atributo colspan

```
<html>
<head>
<script type="text/javascript">
   function setColSpan()
   {
      var x=document.getElementById("myTable").rows[0].cells
      x[0].colSpan="2"
      x[1].colSpan="6"
   }
</script>
<title></title>
</head>
```

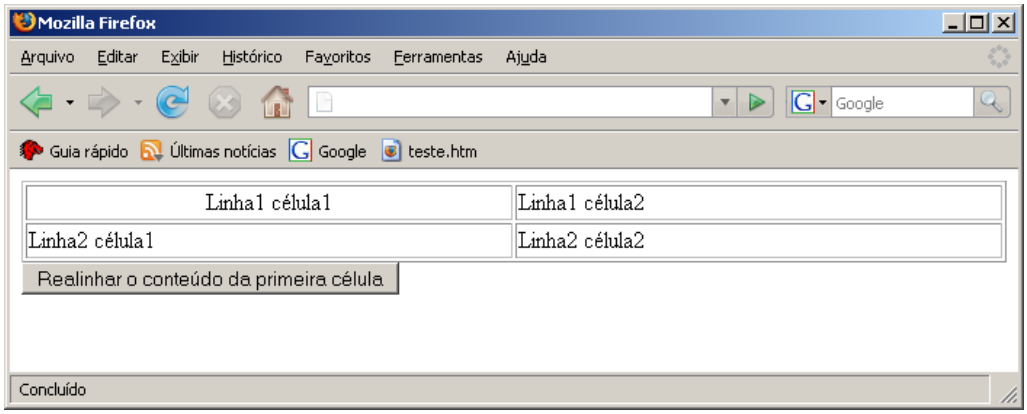
```
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
          <td colspan="4">A cell</td>
          <td colspan="4">A cell</td>
         </tr>
         <tr>
            <td>célula</td>
            <td>célula</td>
            <td>célula</td>
            <td>célula</td>
            <td>célula</td>
            <td>célula</td>
            <td>célula</td>
            <td>célula</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="setColSpan()" type="button"
      value="Alterar o valor de Colspan">
   </form>
</body>
</html>
```



Alinhar o conteúdo da célula

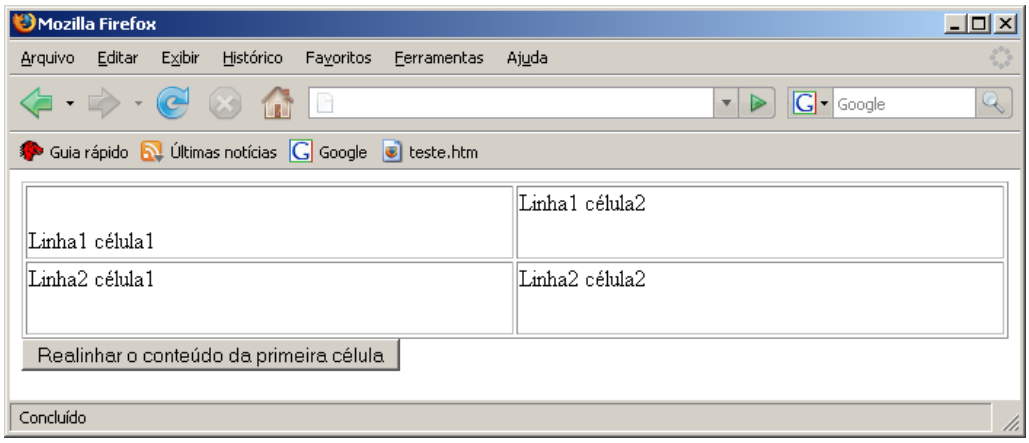
```
<html>
<head>
<script type="text/javascript">
   function alignCell()
   {
      var x=document.getElementById("myTable").rows[0].cells
      x[0].align="center"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" width="100%" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
      </tbody>
   </table>
```

```
   <form>
      <input onclick="alignCell()" type="button"
      value="Realinhar o conteúdo da primeira célula">
   </form>
</body>
</html>
```



Alinhar verticalmente o conteúdo da célula

```
<html>
<head>
<script type="text/javascript">
   function alignCell()
   {
      var x=document.getElementById("myTable").rows[0].cells
      x[0].vAlign="bottom"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" width="100%" border="1">
      <tbody>
         <tr height="50" valign="top">
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr height="50" valign="top">
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="alignCell()" type="button"
      value="Realinhar o conteúdo da primeira célula">
   </form>
</body>
</html>
```



11.15 Objeto tablerow

Este objeto representa de forma completa os elementos criados com a etiqueta <tr> do HTML. A coleção rows, pertencente ao objeto table, contém uma lista com todos os objetos deste tipo existentes na tabela que é representada por um objeto table.

Propriedades

Propriedade	Descrição
align	Define o alinhamento horizontal do conteúdo da célula
rowIndex	Lê o índice de uma linha medido relativamente ao início da tabela
sectionRowIndex	Lê o índice de uma linha medido relativamente ao início da seção atual ou dos elemento thead, tfoot ou tbody
vAlign	Define o alinhamento vertical dos conteúdos das células

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <tr> em acordo com a linguagem HTML.

Coleções

Coleção	Descrição
cells	Devolve todas as células de uma linha

Métodos

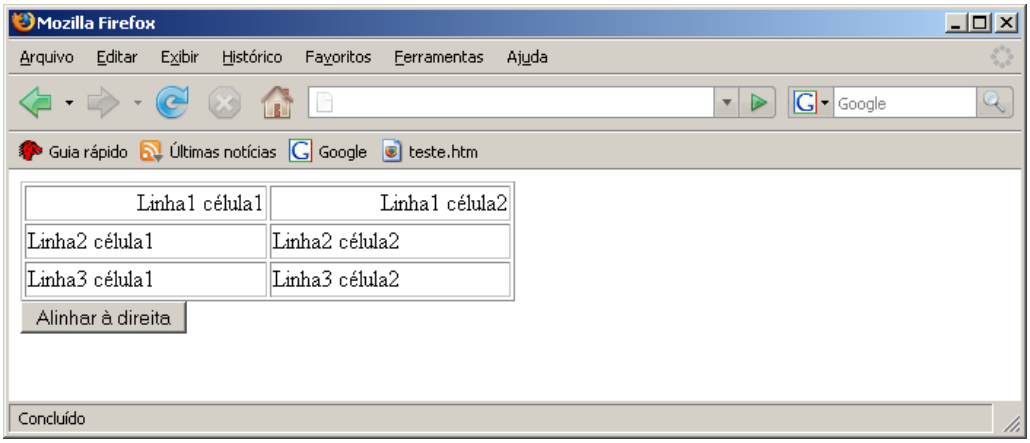
Método	Descrição
deleteCell()	Apaga (remove) a célula da tabela com o índice especificado
insertCell()	Insere uma célula na tabela com o índice especificado

Exemplos de Aplicação

Alinhar o conteúdo das células de uma linha

```
<html>
<head>
<script type="text/javascript">
   function alignRow()
   {
      var x=document.getElementById("myTable").rows
      x[0].align="right"
   }
```

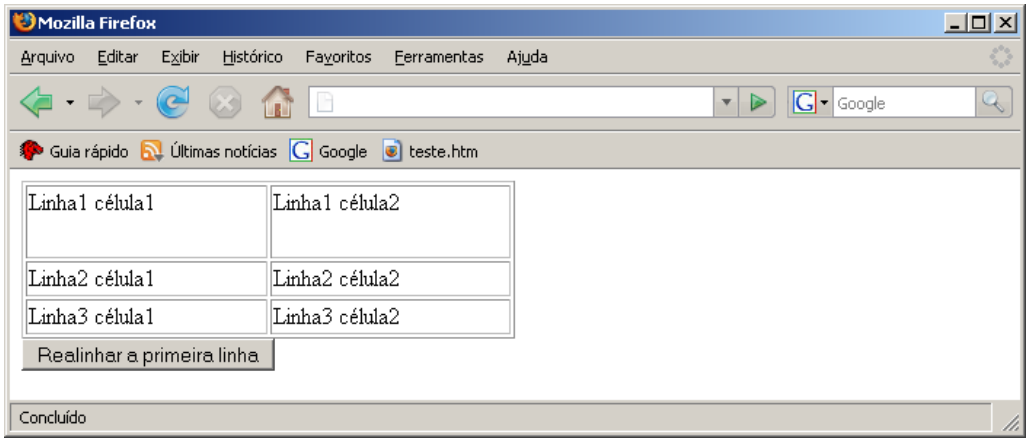
```
</script>
<title></title>
</head>
<body>
   <table id="myTable" width="50%" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
         <tr>
            <td>Linha3 célula1</td>
            <td>Linha3 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="alignRow()" type="button" value="Alinhar à direita">
   </form>
</body>
</html>
```



Alinhar verticalmente os conteúdos das células de uma linha

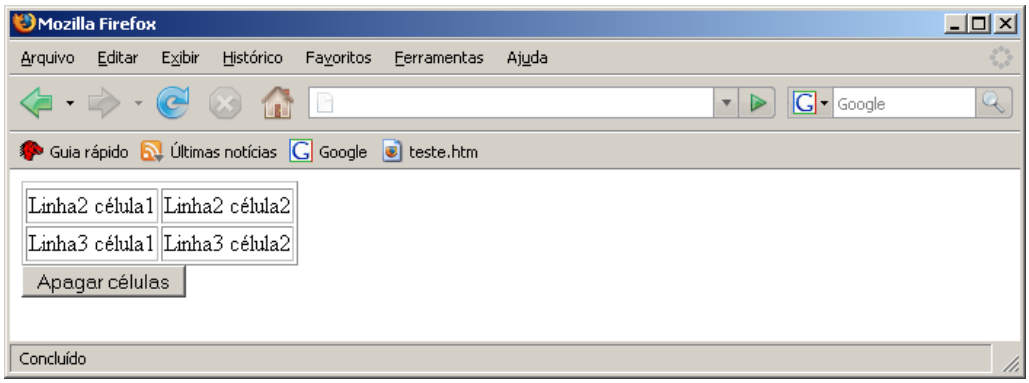
```
<html>
<head>
<script type="text/javascript">
   function alignRow()
   {
      var x=document.getElementById("myTable").rows
      x[0].vAlign="top"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" width="50%" border="1">
      <tbody>
         <tr height="50" valign="bottom">
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
         <tr>
```

```
            <td>Linha3 célula1</td>
            <td>Linha3 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="alignRow()" type="button"
      value="Realinhar a primeira linha">
   </form>
</body>
</html>
```



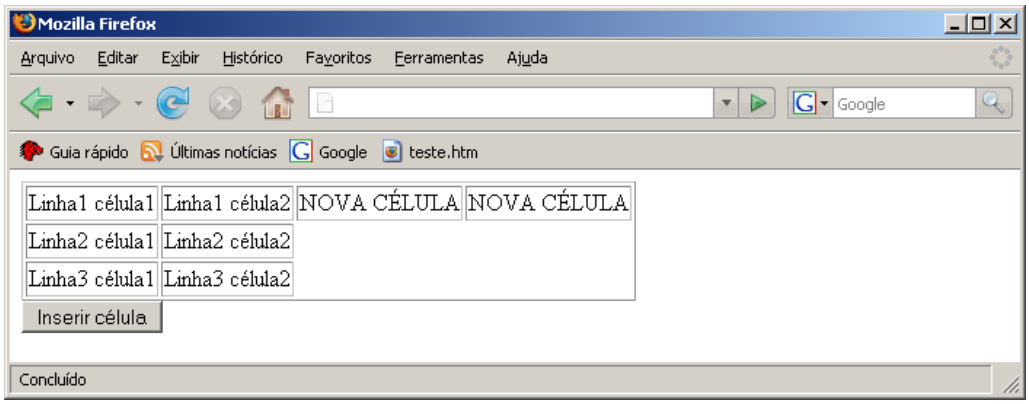
Apagar (remover) células

```
<html>
<head>
<script type="text/javascript">
   function delCell()
   {
      var x=document.getElementById("myTable").rows[0]
      var y=x.deleteCell(1)
      var z=x.deleteCell(0)
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
         <tr>
            <td>Linha3 célula1</td>
            <td>Linha3 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="delCell()" type="button" value="Apagar células">
   </form>
</body>
</html>
```



Inserir células

```
<html>
<head>
<script type="text/javascript">
   function insCell()
   {
      var x=document.getElementById("myTable").rows[0]
      var y=x.insertCell(2)
      y.innerHTML="NOVA CÉLULA"
   }
</script>
<title></title>
</head>
<body>
   <table id="myTable" border="1">
      <tbody>
         <tr>
            <td>Linha1 célula1</td>
            <td>Linha1 célula2</td>
         </tr>
         <tr>
            <td>Linha2 célula1</td>
            <td>Linha2 célula2</td>
         </tr>
         <tr>
            <td>Linha3 célula1</td>
            <td>Linha3 célula2</td>
         </tr>
      </tbody>
   </table>
   <form>
      <input onclick="insCell()" type="button" value="Inserir célula">
   </form>
</body>
</html>
```



11.16 Objeto textarea

Este objeto representa de forma completa os elementos criados com a etiqueta <textarea> do HTML.

Propriedades

Propriedade	Descrição
accessKey	Define uma tecla para acessar rapidamente a área de texto
cols	Lê ou define a largura da área de texto (medida em número de caracteres)
defaultValue	Lê ou define o texto inicial (valor por omissão) contido na área de texto
disabled	Lê ou define o valor da propriedade disabled
form	Devolve o elemento form que contém a área de texto
name	Lê ou define o nome da área de texto
readOnly	Lê e controla o valor do atributo que estabelece se a área de texto pode ser apenas lida ou se também se pode escrever nela
rows	Lê ou define o número de linhas da área de texto
tabIndex	Lê ou define o índice (número de ordem) do elemento que corresponde a este objeto no acesso através da tecla tab
type	Lê o tipo da área de texto
value	Lê ou define o texto contido na área de texto

Propriedades

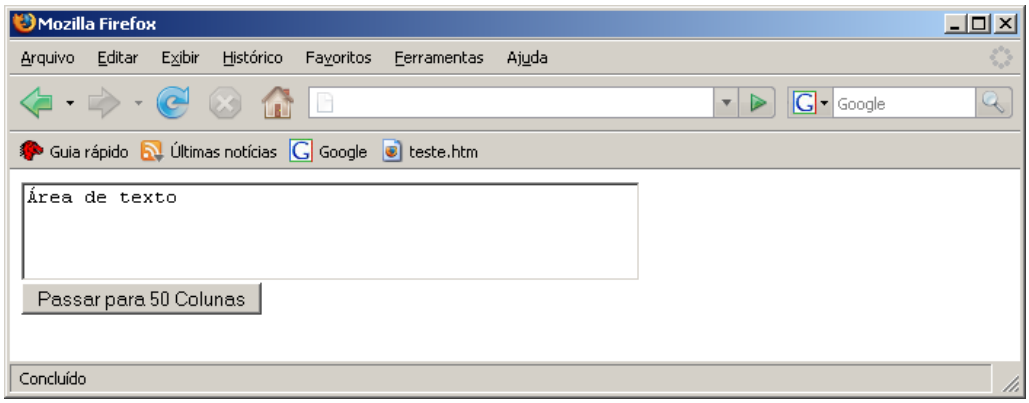
Propriedade	Descrição
blur()	A área de texto perde o foco
focus()	A área de texto ganha o foco
select()	Seleciona o conteúdo da área de texto

Cada uma das propriedades descritas mais acima corresponde a um atributo do elemento <textarea> em acordo com a linguagem HTML.

Exemplos de Aplicação

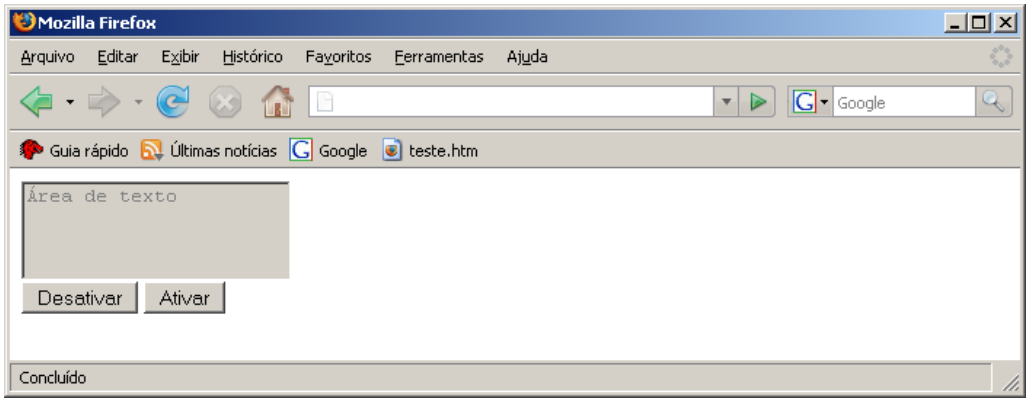
Alterar o número de colunas (atributo cols)

```
<html>
<head>
<script type="text/javascript">
   function changeCols()
   {
      document.getElementById("myTxtArea").cols=50
   }
</script>
<title></title>
</head>
<body>
   <form>
      <textarea id="myTxtArea" rows="3" cols="20">Área de texto</textarea><br>
      <input onclick="changeCols()" type="button"
      value="Passar para 50 Colunas">
   </form>
</body>
</html>
```



Desativar e ativar a área de texto

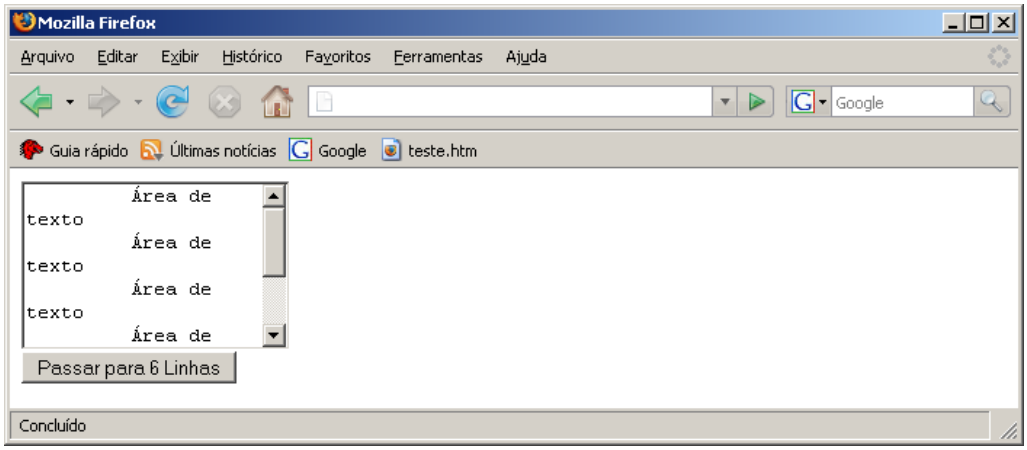
```
<html>
<head>
<script type="text/javascript">
   function makeDisable()
   {
      document.getElementById("myTxtArea").disabled=true
   }

   function makeEnable()
   {
      document.getElementById("myTxtArea").disabled=false
   }
</script>
<title></title>
</head>
<body>
   <form>
      <textarea id="myTxtArea" rows="3" cols="20">Área de texto</textarea><br>
      <input onclick="makeDisable()" type="button" value="Desativar">
      <input onclick="makeEnable()" type="button" value="Ativar">
   </form>
</body>
</html>
```



Alterar o número de linhas (atributo rows)

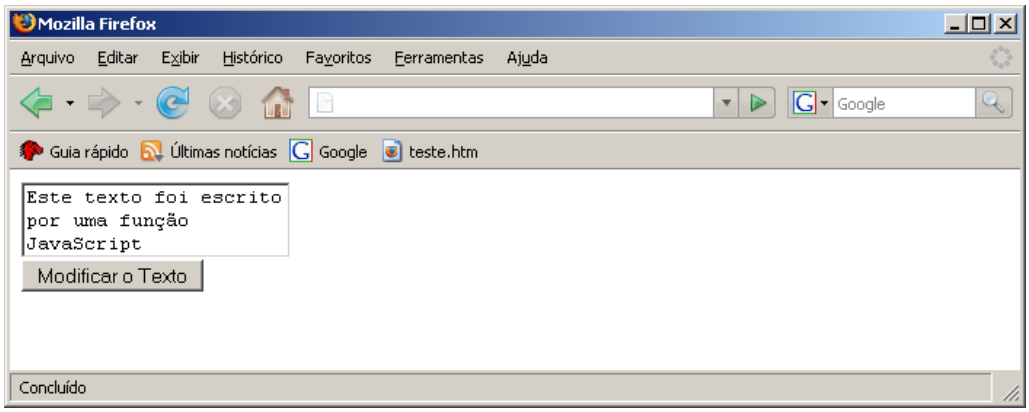
```
<html>
<head>
<script type="text/javascript">
   function changeRows()
   {
      document.getElementById("myTxtArea").rows=6
```

```
   }
</script>
<title></title>
</head>
<body>
   <form>
      <textarea id="myTxtArea" rows="3" cols="20">
         Área de texto
         Área de texto
         Área de texto
         Área de texto
         Área de texto
      </textarea><br>
      <input onclick="changeRows()" type="button"
      value="Passar para 6 Linhas">
   </form>
</body>
</html>
```



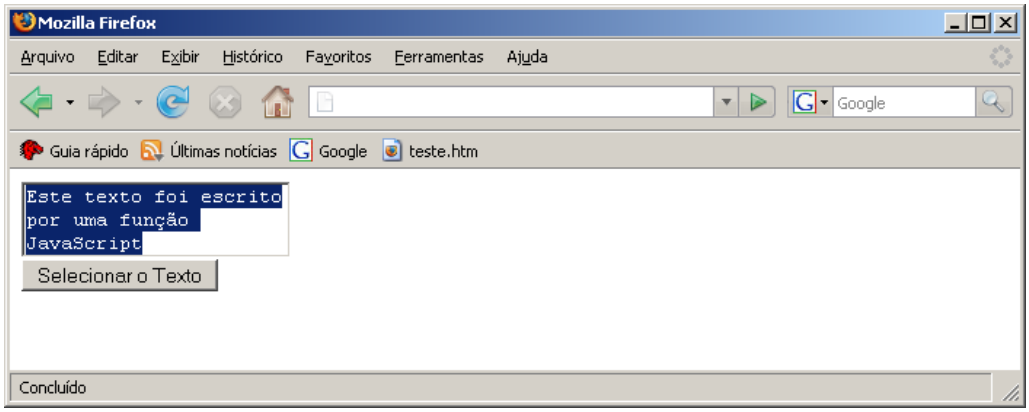
Modificar o conteúdo (texto)

```
<html>
<head>
<script type="text/javascript">
   function changeText()
   {
      document.getElementById("myTxtArea").value="Este texto foi escrito por
      uma função JavaScript"
   }
</script>
<title></title>
</head>
<body>
   <form>
      <textarea id="myTxtArea">Área de texto</textarea><br>
      <input onclick="changeText()" type="button" value="Modificar o Texto">
   </form>
</body>
</html>
```



Selecionar o conteúdo de uma área de texto

```
<html>
<head>
<script type="text/javascript">
   function makeSelect()
   {
      var x=document.getElementById("myTxtArea")
      x.select()
      x.focus()
   }
</script>
<title></title>
</head>
<body>
   <form>
      <textarea id="myTxtArea">Área de texto</textarea><br>
      <input onclick="makeSelect()" type="button" value="Selecionar o Texto">
   </form>
</body>
</html>
```



12. Mais controle sobre os elementos do HTML

No capítulo anterior estudmos objetos que nos permitem ler e manipular diversos elementos que encontramos como parte do conteúdo de uma página HTML. Cada um desses objetos possui métodos e propriedades talhados especificamente para um determinado elemento. Agora vamos aprender a usar técnicas que nos permitem ler e manipular quase todos os elementos que podemos encontrar numa página escrita em HTML, qualquer que seja o seu tipo.

12.1 Propriedades intrínsecas dos elementos do HTML e do XHTML

Com o DOM todos os elementos do HTML passam a pertencer à categoria de objetos. Como os diversos elementos podem ter características muito diferentes uns dos outros, os objetos que os representam irão ter propriedades e métodos que podem diferir bastante uns dos outros. Como tivemos oportunidade de ver, o objeto table tem um método chamado deleteRow(), que serve para apagar uma linha de células, e o objeto form não tem. Por outro lado o objeto form possui métodos que não fazem qualquer sentido numa tabela e por isso não estão presentes em um objeto table. Com as propriedades acontece o mesmo: o objeto table possui uma propriedade chamada cellPadding, que não existe em nenhum outro objeto.

Mas apesar de todas as diferenças existem propriedades intrínsecas que estão presentes em quase todos os objetos que representam elementos do HTML e outras que estão presentes em todos os objetos que representam elementos de uma determinada classe. Essas propriedades estão relacionadas com os atributos intrínsecos (ou nucleares) do HTML 4, que tivemos oportunidade de estudar no "Curso de HTML 4.01".

Como tivemos oportunidade de ver no curso e na referência de HTML, os atributos a que chamamos intrínsecos (ou nucleares) estão presentes em quase todos os elementos. Os mais importantes são os atributos id e style. Para cada um desses atributos o DOM reserva uma propriedade no objeto que representa o elemento.

A propriedade style

A propriedade style de um objeto corresponde ao atributo style do elemento por ele representado. Esta propriedade contém as definições de estilos CSS aplicadas ao elemento através do seu atributo style e pode ser modificada para alterar o aspecto visual do elemento. Para lermos ou modificarmos as propriedades CSS precisamos saber que algumas têm nomes em CSS que não são iguais aos nomes que têm num script.

Em CSS há propriedades que têm dois nomes separados pelo caractere "-" (sinal menos, ou hífen.) Um exemplo disto é a propriedade background-color, que define a cor de fundo de um elemento. O nome que essa propriedade tem em um script obtém-se a partir do nome CSS do seguinte modo: 1) passamos todas as letras do nome para minúsculas; 2) transformamos em maiúscula a primeira letra que vem a seguir ao caractere "-"; 3) eliminamos o caractere "-".

A lista seguinte contém alguns exemplos desta transformação:

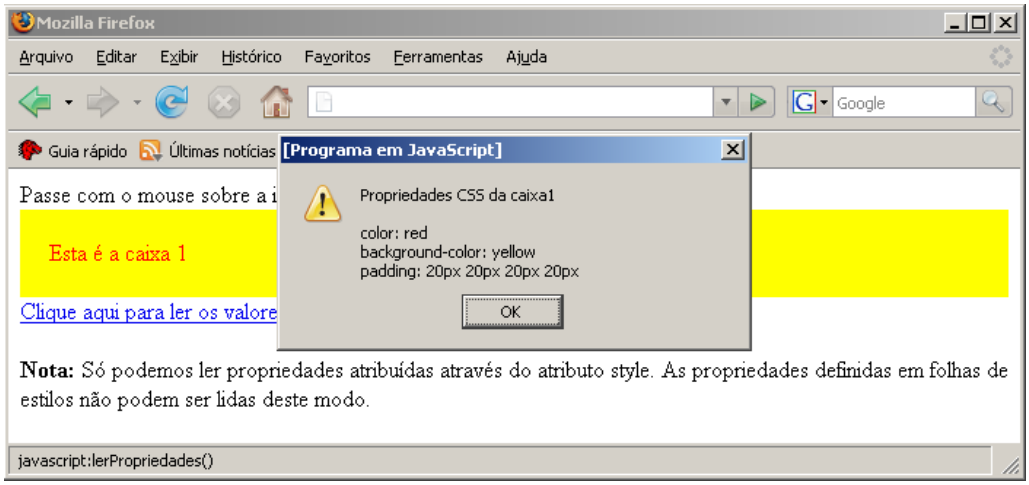
Propridade CSS	Nome da propriedade em um script
background-color	backgroundColor
padding-top	paddingTop
border-bottom	borderBottom
word-spacing	wordSpacing
text-decoration	textDecoration

Você já sabe que se precisar acessar por script uma propriedade CSS que tem mais do que um nome só precisa aplicar estas as regras que acabamos de aprender.

Exemplos de Aplicação

Ler os valores das propriedades CSS

```
<html>
<head>
<script type="text/javascript">
   function lerPropriedades()
   {
      var obj=document.getElementById("caixa1")
      var s="Propriedades CSS da caixa1\n\n"
      s+="color: "+obj.style.color
      s+="\nbackground-color: "+obj.style.backgroundColor
      s+="\npadding: "+obj.style.padding
      alert(s)
   }
</script>
<title></title>
</head>
<body>
   Passe com o mouse sobre a imagem
   <div id="caixa1" style="color:red; background-color:yellow; padding:20px">
      Esta é a caixa 1
   </div>
   <a href="javascript:lerPropriedades()">
      Clique aqui para ler os valores das propriedades CSS</a><br><br>
   <b>Nota:</b> Só podemos ler propriedades atribuídas através do atributo
   style. As propriedades definidas em folhas de estilos não podem ser lidas
   deste modo.
</body>
</html>
```



A propriedade innerHTML

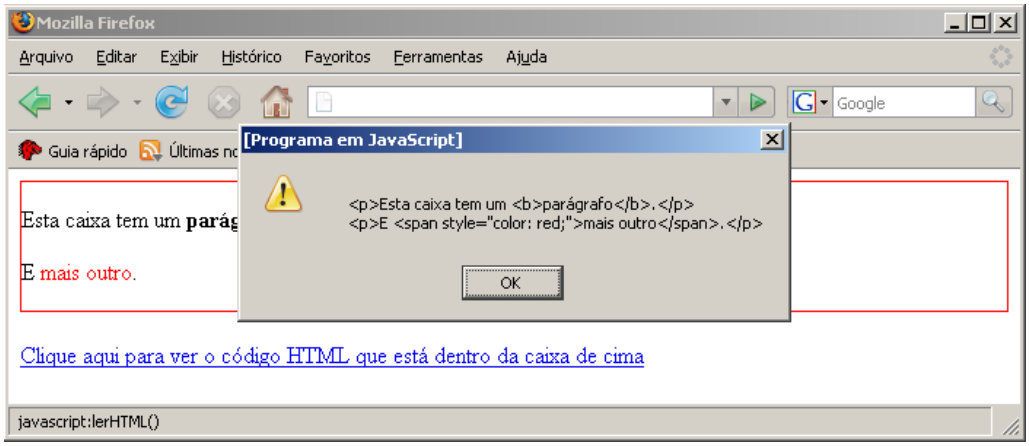
Outra propriedade muito útil para o DHTML é o innerHTML. Esta propriedade só existe em elementos que têm conteúdo (, , <td>, <p> são exemplos disto). Elementos como <img> ou <br> não têm conteúdo, deste modo, os objetos que os representam não possuem propriedade innerHTML.

O valor desta propriedade é o código HTML que está dentro do elemento representado pelo objeto. Se lermos o seu valor ficamos sabendo o conteúdo HTML do elemento, e se o modificarmos estaremos alterando o conteúdo do elemento.

Exemplos de Aplicação

Ler o código HTML que está dentro de um elemento

```
<html>
<head>
<script type="text/javascript">
   function lerHTML()
   {
      alert(document.getElementById("caixa1").innerHTML)
   }
</script>
<title></title>
</head>
<body>
   <div id="caixa1" style="border: solid 1px red">
      <p>Esta caixa tem um <b>parágrafo</b>.</p>
      <p>E <span style="color:red">mais outro</span>.</p>
   </div><br>
   <a href="javascript:lerHTML()">Clique aqui para ver o código HTML que
   está dentro da caixa de cima</a>
</body>
</html>
```



Para começar vamos aprender mais sobre o método getElementById() do objeto document. Este método serve para selecionar o elemento com o qual queremos trabalhar, independentemente do seu tipo (pode ser uma tabela, uma ligação, um parágrafo, etc.) Depois vamos usar as propriedades style e innerHTML para ler e manipular o objeto (elemento do HTML) que escolhemos.

A propriedade id e o método getElementById()

Nós já usamos o métdodo getElementById() (pertencente ao objeto document) por diversas vezes nos exercícios que efetuamos. Como você reparou, o modo como este método opera é fácil de perceber, deste modo, não vale a pena dar explicações muito detalhadas. Tudo o que precisamos saber a seu respeito é que se lhe dermos como argumento o valor do atributo id de um elemento da página ele nos fornece o objeto que representa esse elemento. Se dermos como argumento um valor de id que não existe na página então ele nos devolve o valor especial null, significando que o objeto que pedimos não existe na página em que o procuramos. Todos os objetos devolvidos pelo método getElementById() que não sejam null possuem propriedades úteis que podemos ler ou até manipular.

Todos os objetos que podem ser obtidos usando o método getElementById() possuem um valor no atributo id. Esse atributo está presente no objeto que representa o elemento sob a forma de uma propriedade que tem por nome id.

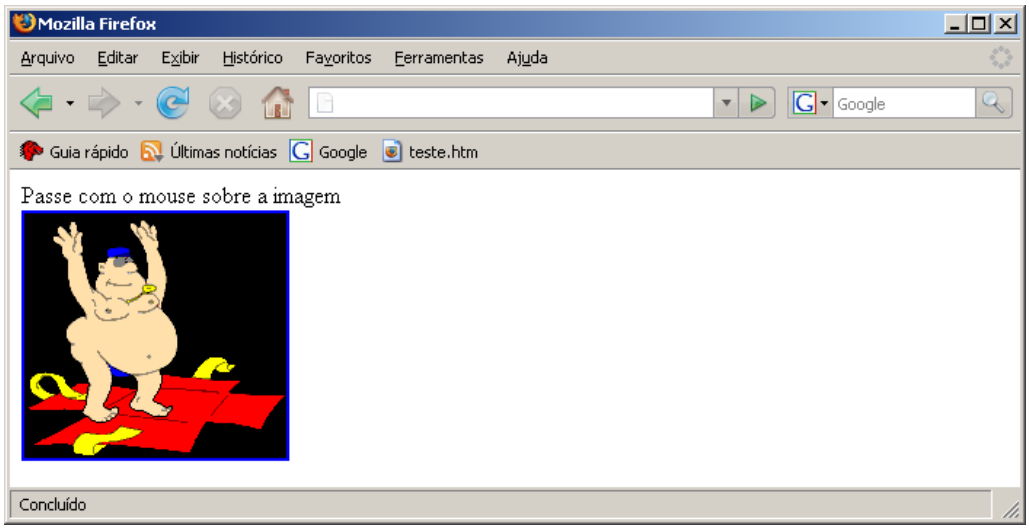
Do mesmo modo que podemos usar o método getElementById() para obter o objeto que representa um elemento, também podemos usar outras formas alternativas para obter esse mesmo objeto. Os dois exemplos seguintes são totalmente

equivalentes. No primeiro nós usamos o método getElementById() para obter o objeto que representa um elemento <img> que queremos controlar, e no segundo fornecemos o mesmo objeto como argumento da função.

Exemplos de Aplicação

Efeito de mouseover simples com CSS

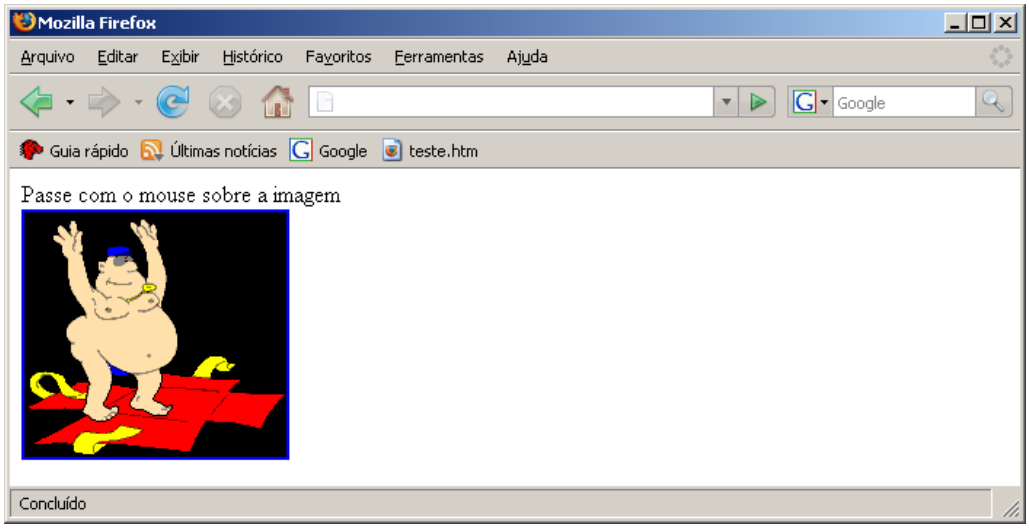
```
<html>
<head>
<script type="text/javascript">
   function mostrarContorno()
   {
      var obj=document.getElementById("sexy")
      obj.style.border="solid 2px blue"
   }

   function retirarContorno()
   {
      var obj=document.getElementById("sexy")
      obj.style.border="solid 2px white"
   }
</script>
<title></title>
</head>
<body>
   Passe com o mouse sobre a imagem<br>
   <img src="sexy.png" style="border:solid 2px white" id="sexy"
   onmouseover="mostrarContorno()" onmouseout="retirarContorno()">
</body>
</html>
```



Efeito de mouseover simples com CSS (versão 2)

```
<html>
<head>
<script type="text/javascript">
   function mostrarContorno(obj)
   {
      obj.style.border="solid 2px blue"
   }

   function retirarContorno(obj)
   {
      obj.style.border="solid 2px white"
   }
```

```
</script>
<title></title>
</head>
<body>
   Passe com o mouse sobre a imagem<br>
   <img src="sexy.png" style="border:solid 2px white" id="sexy"
   onmouseover="mostrarContorno(this)" onmouseout="retirarContorno(this)">
</body>
</html>
```



12.2 Posicionamento e medição de elementos em DHTML

O HTML Dinâmico oferece propriedades que nos permitem medir e posicionar os elementos nas páginas com todo o rigor. Como já sabemos, os estilos CSS são a melhor forma que temos para posicionar um elemento, mas quando precisamos saber o local exato em que ele se encontra ou o seu tamanho exato, então precisamos de algo mais. Para esclarecermos bem este ponto comecemos por apresentar quatro novas propriedades que o DOM nos oferece:

Propriedade	Descrição
offsetHeight	Largura do elemento medida em pixels. Esta propriedade só pode ser lida.
offsetLeft	Número de pixels que separam o início do elemento da extremidade esquerda da área onde é desenhada a página. Esta propriedade só pode ser lida.
offsetTop	Número de pixels que separam o início do elemento do topo da área onde é desenhada a página. Esta propriedade só pode ser lida.
offsetWidth	Altura do elemento medida em pixels. Esta propriedade só pode ser lida.

Nota: Estas propriedades estão presentes em objetos que representam elementos de bloco.

O fato de podermos manipular e ler os valores guardados pela propriedade style pode levar-nos a pensar que estas quatro propriedades adicionais não são necessárias, mas isso não é verdade. Os estilos CSS podem ser usados de muitas maneiras: eles permitem designar tamanhos em pixels, em polegadas, em pontos ou como percentagens. Do posicionamento por CSS podem resultar valores que variam de browser para browser, de máquina para máquina e de usuário para usuário. É a

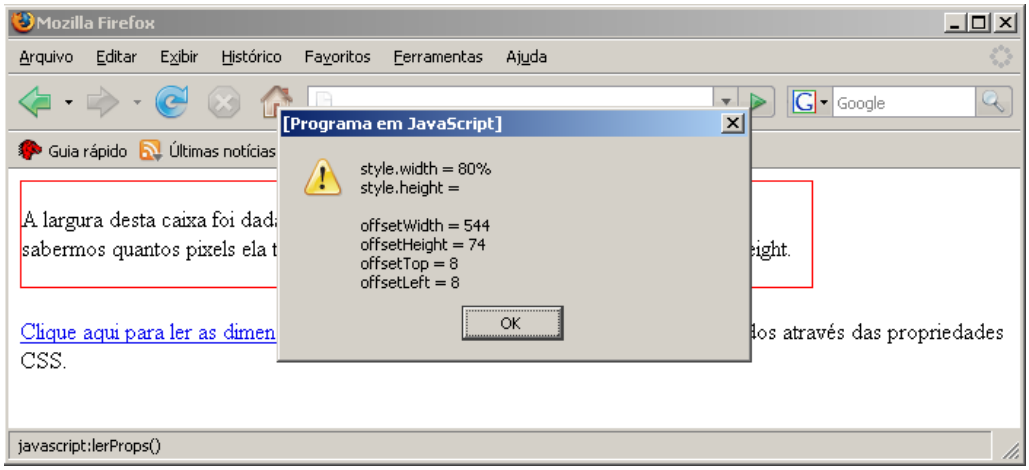
própria filosofia dos estilos CSS que nos impede de saber quais as dimensões exatas de um elemento.

É aqui que entram estas quatro propriedades. Aquilo que elas guardam são os tamanhos e as posições de cada elemento em cada instante. Quando redimensionamos a janela do browser os elementos podem mudar de posição na página mas os seus estilos CSS não sofrem alterações. Já as quatro propriedades que acabamos de ver são atualizadas sempre que ocorre uma alteração na página ou na janela do browser.

Exemplos de Aplicação

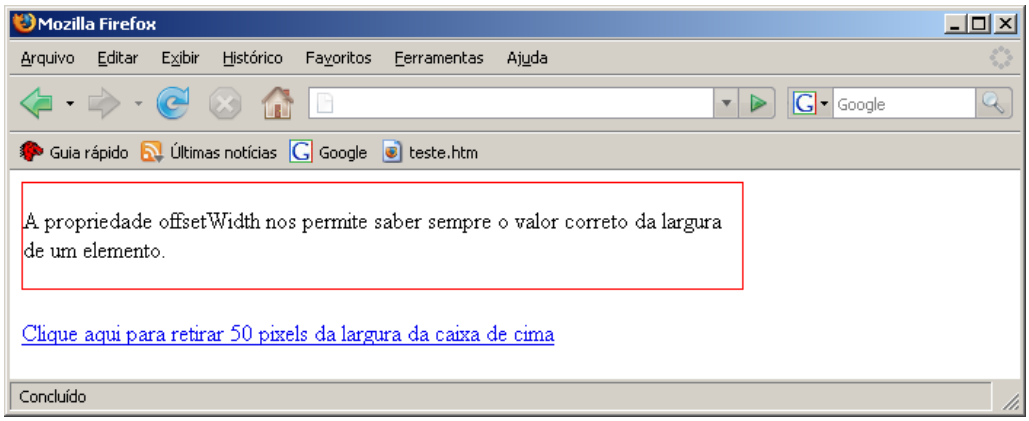
Ler as dimensões e a posição de um elemento

```
<html>
<head>
<script type="text/javascript">
   function lerProps()
   {
      var o=document.getElementById("caixa1")
      var s="style.width = "+o.style.width
      s+="\nstyle.height = "+o.style.height
      s+="\n\noffsetWidth = "+o.offsetWidth
      s+="\noffsetHeight = "+o.offsetHeight
      s+="\noffsetTop = "+o.offsetTop
      s+="\noffsetLeft = "+o.offsetLeft
      alert(s)
   }
</script>
<title></title>
</head>
<body>
   <div id="caixa1" style="border: solid 1px red; width: 80%">
      <p>
         A largura desta caixa foi dada em CSS de forma relativa.
         Por isso a única forma de sabermos quantos pixels ela
         tem de largura é lendo o valor da propriedade offsetHeight.
      </p>
   </div><br>
   <a href="javascript:lerProps()">Clique aqui para ler as dimensões da caixa
   de cima</a> e compare com os valores obtidos através das propriedades CSS.
</body>
</html>
```



Redimensionar um elemento

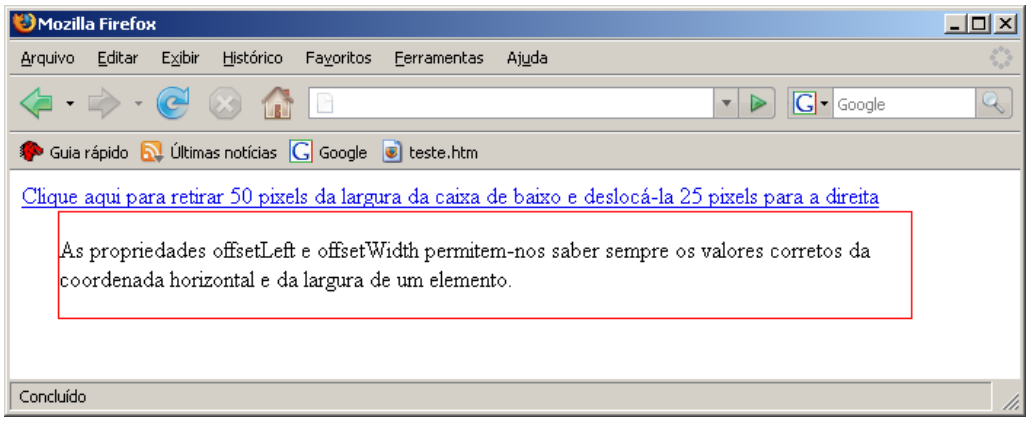
```
<html>
<head>
<script type="text/javascript">
   function reduzir()
   {
      var o=document.getElementById("caixa1")
      o.style.width=(o.offsetWidth-50)+"px"
   }
</script>
<title></title>
</head>
<body>
   <div id="caixa1" style="border: solid 1px red; width: 80%">
      <p>
         A propriedade offsetWidth nos permite saber sempre
         o valor correto da largura de um elemento.
      </p>
   </div><br>
   <a href="javascript:reduzir()">Clique aqui para retirar
   50 pixels da largura da caixa de cima</a>
</body>
</html>
```



Reposicionar um elemento

```
<html>
<head>
<script type="text/javascript">
   function reposicionar()
   {
      var o=document.getElementById("caixa1")
      o.style.width=(o.offsetWidth-50)+"px"
      o.style.left=(o.offsetLeft+25)+"px"
   }
</script>
<title></title>
</head>
<body>
   <a href="javascript:reposicionar()">Clique aqui para retirar
   50 pixels da largura da caixa de baixo e deslocá-la
   25 pixels para a direita</a><br>
   <div id="caixa1" style="border: solid 1px red; position: absolute">
      <p>
         As propriedades offsetLeft e offsetWidth permitem-nos
         saber sempre os valores corretos da coordenada horizontal
         e da largura de um elemento.
      </p>
   </div>
</body>
```

```
</html>
```


Mozilla Firefox
Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda
Google
Guia rápido Últimas notícias Google teste.htm
Clique aqui para retirar 50 pixels da largura da caixa de baixo e deslocá-la 25 pixels para a direita
As propriedades offsetLeft e offsetWidth permitem-nos saber sempre os valores corretos da coordenada horizontal e da largura de um elemento.
Concluído