

# Programação em Javascript II

<criarweb.com>

Manual para imprimir

## Autores do manual

Este manual foi criado pelos seguintes colaboradores de Criarweb.com:

**Miguel Angel Alvarez -**

**Tradução de JML**

(40 capítulos)

## Introdução ao manual II de Javascript

Nesta [segunda parte do manual de Javascript](#) vamos tratar de explicar todos os recursos com os que conta um programador de Javascript e com os que pode criar todo tipo de efeitos e aplicações.

Para ler e entender bem o que vem nos seguintes capítulos é necessário ter lido antes a [primeira parte deste manual](#): Programação em Javascript I, onde se explicam as bases sobre as que temos que assentar os seguintes conhecimentos. Na primeira parte deste manual conhecemos a origem e as aplicações de Javascript, mas, sobretudo ressaltamos sua sintaxe que é muito importante para entender os scripts que faremos nos seguintes capítulos.

Os objetivos dos seguintes capítulos cobrirão aspectos diversos de Javascript como:

- Funções incorporadas na linguagem Javascript
- Os objetos em Javascript
- Hierarquia de objetos do navegador
- Trabalho com formulários
- Controle de janelas secundárias e frames
- Eventos

Como se pode ver, todos os temas têm um forte caráter prático e cobrem vários aspectos com os quais formamos a nível avançado em Javascript. Esperamos que sirvam para iluminar uma área tão ampla do desenvolvimento de páginas web como é o scripting do lado do cliente.

Vamos sem mais pausa com esta [segunda parte do manual](#), que será muito mais interessante e prática que [a primeira](#).

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Biblioteca de funções Javascript

Em todas as linguagens de programação existem bibliotecas de funções que servem para fazer diversas coisas e muito competitiva na hora de programar. As bibliotecas das linguagens de programação economizam a tarefa de escrever as funções comuns que em geral os programadores podem necessitar. Uma linguagem de programação bem desenvolvida terá uma boa quantidade delas. Em algumas ocasiões é mais complicado conhecer todas as bibliotecas do que aprender a programar a linguagem.

Javascript contém uma boa quantidade de funções em suas bibliotecas. Como se trata de uma linguagem que trabalha com objetos, muitas das bibliotecas se implementam através de objetos. Por exemplo, as funções matemáticas ou as de manejo de strings se implementam mediante os objetos Math e String. Entretanto, existem algumas funções que não estão associadas a nenhum objeto e são as que veremos neste capítulo, já que ainda não conhecemos os objetos e não os necessitaremos para estudá-las.

Estas são as funções que Javascript coloca à disposição dos programadores.

### **eval(string)**

Esta função recebe uma cadeia de caracteres e a executa como se fosse uma sentença de Javascript.

**parseInt(cadeia,base)**

Recebe uma cadeia e uma base. Devolve um valor numérico resultante de converter a cadeia em um número na base indicada.

**parseFloat(cadeia)**

Converte a cadeia em um número e o devolve.

**escape(caractere)**

Devolve um caractere que recebe por parâmetro em uma codificação ISO Latin 1.

**unescape(caractere)**

Faz exatamente o oposto da função escape.

**isNaN(número)**

Devolve um booleano dependendo do que recebe por parâmetro. Se não é um número devolve um true, se é um número devolve false.

As bibliotecas que se implementam mediante objetos e as do manejo do explorador, que também se manejam com objetos, serão vistas mais adiante.

Vamos ver algum exemplo com as funções mais importantes desta lista.

*Artigo por Miguel Angel Alvarez - Tradução de JML*

## Exemplos das funções da biblioteca Javascript

Agora podemos ver vários exemplos de utilização de funções da biblioteca que proporciona Javascript.

**Função eval**

Esta função é muito importante, tanto que existem algumas aplicações de Javascript que não poderiam se realizar se não a utilizamos. A utilização desta função é muito simples, mas pode ser que seja mais complexo entender em que caso utiliza-la porque às vezes é um pouco sutil sua aplicação.

Com os conhecimentos atuais não podemos dar um exemplo muito complicado, mas pelo menos podemos ver em funcionamento a função. Vamos utilizá-la em uma sentença um pouco rara e que não serve muito, mas se conseguimos entendê-la, conseguiremos entender também a função eval.

```
var meuTexto = "3 + 5"  
eval("document.write(" + meuTexto + ")")
```

Primeiro, criamos uma variável com um texto, na seguinte linha utilizamos a função eval e como parâmetro lhe passamos uma instrução javascript para escrever na tela. Se concatenamos os strings que existem dentro dos parênteses da função eval ficaria assim.

```
document.write(3 + 5)
```

A função eval executa a instrução que for passada por parâmetro, portanto executará esta sentença, o que dará como resultado que se escreva um 8 na página web. Primeiro se resolve a soma que há entre parênteses, com o qual obtemos o 8 e logo se executa a instrução de escrever na tela.

## Função parseInt

Esta função recebe um número, escrito como uma cadeia de caracteres, e um número que indica uma base. Na verdade pode receber outros tipos de variáveis, dado que as variáveis não têm tipo em Javascript, mas costuma-se utilizar passando um string para converter a variável de texto em um número.

As distintas bases que pode receber a função são 2, 8, 10 e 16. Se não passamos nenhum valor como base a função interpreta que a base é decimal. O valor que devolve a função sempre tem base 10, de modo que se a base não é 10 converte o número a essa base antes de devolvê-lo.

Vejamos uma série de chamadas à função parseInt para ver o que devolve e entender um pouco mais a função.

```
document.write (parseInt("34"))
```

Devolve o número 34

```
document.write (parseInt("101011",2))
```

Devolve o número 43

```
document.write (parseInt("34",8))
```

Devolve o número 28

```
document.write (parseInt("3F",16))
```

Devolve o número 63

Esta função se utiliza na prática para um monte de coisas distintas no manejo com números, por exemplo, obter a parte inteira de um decimal.

```
document.write (parseInt("3.38"))
```

Devolve o número 3

Também é muito habitual seu uso para saber se uma variável é numérica, pois se passamos um texto à função que não seja numérico nos devolverá NaN (Not a Number) o que quer dizer que Não é um Número.

```
document.write (parseInt("criarweb.com"))
```

Devolve o número NaN

Este mesmo exemplo é interessante com uma modificação, pois se passamos uma combinação de letras e números, daria o seguinte.

```
document.write (parseInt("16XX3U"))
```

Devolve o número 16

```
document.write (parseInt("TG45"))
```

Devolve o número NaN

Como se pode ver, a função tenta converter o string em número e se não pode, devolve NaN.

Todos estes exemplos, meio desconexos, sobre como trabalha parseInt serão revisados mais adiante em exemplos mais práticos quando tratarmos o trabalho com formulários.

## Função isNaN

Esta função devolve um booleano dependendo se o que se recebe é um número ou não. O único que pode receber é um número ou a expressão NaN. Se recebe um NaN devolve true e se recebe um número devolve false. É uma função muito simples de entender e de utilizar.

A função costuma trabalhar em combinação com a função parseInt ou parseFloat, para saber se o que devolvem estas duas funções é um número ou não.

```
meuInteger = parseInt("A3.6")  
isNaN(meuInteger)
```

Na primeira linha atribuímos a variável meuInteger o resultado de tentar converter a inteiro o texto A3.6. Como este texto não se pode converter a número, a função parseInt devolve NaN. A segunda linha comprova se a variável anterior é NaN e sendo assim devolve um true.

```
meuFloat = parseFloat("4.7")  
isNaN(meuFloat)
```

Neste exemplo convertemos um texto a número com decimais. O texto se converte perfeitamente porque corresponde com um número. Ao receber um número a função isNaN devolve um false.

**Referência:** [Validar inteiro em campo de formulário](#)

Temos um Workshop de Javascript muito interessante que foi realizado para aprofundar os conhecimentos destes capítulos. Trata-se de um script para validar um campo de formulário de maneira que saibamos com certeza que dentro do campo há sempre um número inteiro. Pode ser muito interessante lê-lo agora, já que utilizamos as funções isNaN() e parseInt(). [Ver o Workshop](#)

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Objetos em Javascript

Vamos entrar em um tema muito importante de Javascript como são os objetos. É um tema que ainda não vimos e sobre o qual adiante vamos tratá-lo constantemente, pois todas as coisas em Javascript, inclusive as mais simples, vamos realizar através do manejo de objetos. De fato, nos exemplos realizados até agora fizemos um grande esforço para não utilizar objetos e ainda assim, o utilizamos em alguma ocasião, pois é muito difícil encontrar exemplos em Javascript que, embora sendo simples, não façam uso deles.

A programação orientada a objetos representa uma nova maneira de pensar na hora de fazer um programa. Javascript não é uma linguagem de programação orientada a objetos, embora os utiliza em muitas ocasiões: podemos criar novos objetos e utilizar muitos que estão criados desde um princípio. Entretanto, a maneira de programar não vai mudar muito e o que vimos até aqui relativo à sintaxe, funções, etc., pode ser utilizado da mesma forma que foi indicado. Somente vamos aprender uma espécie de estrutura nova.

Para começar a aprofundarmos na programação orientada a objetos é imprescindível que se leia um [pequeno artigo publicado em CriarWeb sobre este tema](#). Depois de sua leitura você pode continuar com estas linhas e se já conhece a programação orientada a objetos continue lendo sem pausa.

### Como instanciar objetos

Instanciar um objeto é a ação de criar um exemplar de uma classe para poder trabalhar com ele logo. Lembramos que um objeto se cria a partir de uma classe e a classe é a definição das características e funcionalidades de um objeto. Com as classes não se trabalha, estas somente são definições, para trabalhar com uma classe devemos ter um objeto instanciado dessa classe.

Em javascript para criar um objeto a partir de uma classe se utiliza a instrução new, desta maneira.

```
var meuObjeto = new minhaClasse()
```

Em uma variável que chamamos meuObjeto atribuo um novo (new) exemplar da classe minhaClasse. Os parênteses se preenchem com os dados que a classe necessita para iniciar o objeto, se não há que colocar nenhum parâmetro, coloca-se os parênteses vazios. Na verdade o que se faz quando se cria um objeto é chamar ao construtor dessa classe e o construtor é o encarregado de cria-lo e e inicia-lo. Falaremos sobre isso mais adiante.

### Como acessar às propriedades e métodos dos objetos

Em Javascript podemos acessar às propriedades e aos métodos de objetos de forma similar a como se faz em outras linguagens de programação, com o operador ponto (".").

As propriedades se acessam colocando o nome do objeto seguido de um ponto e o nome da propriedade que se deseja acessar. Desta maneira:

```
meuObjeto.minhaPropriedade
```

Para chamar aos métodos utilizamos uma sintaxe similar, mas colocando ao final entre parênteses os parâmetros que passamos aos métodos. Do seguinte modo:

```
meuObjeto.meuMetodo(parametro1,parametro2)
```

Se o método não recebe parâmetros colocamos os parênteses também, mas sem nada dentro.

```
meuObjeto.meuMetodo()
```

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Objetos incorporados em Javascript

Sabendo já o que é a programação orientada a objetos vamos introduzir o manejo de objetos em Javascript e para isso vamos começar com o estudo das classes pré-definidas que implementam as livrarias para o trabalho com strings, matemáticas, datas etc. As classes que vamos ver a seguir são as seguintes:

- **String**, para o trabalho com cadeias de caracteres.
- **Date**, para o trabalho com datas.
- **Math**, para realizar funções matemáticas.
- **Number**, para realizar algumas coisas com números
- **Boolean**, trabalho com booleanos.

**Nota: As classes se escrevem com a primeira letra em maiúsculas.** Tem que ficar claro que uma classe é uma espécie de "declaração de características e funcionalidades" dos objetos. Com outras palavras, as classes são descrições da forma e do funcionamento dos objetos. Com as classes geralmente não se realiza nenhum trabalho, mas sim com os objetos, que criamos a partir das classes. Uma vez compreendida a diferença entre objetos e classes, cabe assinalar que String é uma classe, o mesmo que Date. Se quisermos trabalhar com cadeias de caracteres ou datas necessitamos criar objetos das classes String e Date respectivamente. Como sabemos, Javascript é uma linguagem sensível às letras maiúsculas e minúsculas, e neste caso, **as classes, por convenção, sempre se escrevem com a primeira letra em maiúscula e também a primeira letra das palavras seguintes**, se é que o nome da classe está composto por várias palavras. Por exemplo, se tivéssemos a classe de "Alunos universitarios" se escreveria com a -A- de alunos e a -U- de universitarios em maiúsculo. AlunosUniversitarios seria o nome de nossa suposta classe.

Há outros que pertencem a este mesmo conjunto, os enumeramos aqui para que fique a constância deles, porém não os abordaremos nos capítulos seguintes.

- **Array**, já vimos em [capítulos correspondentes ao primeiro manual de Javascript](#).
- Também a classe **Function**, foi vista parcialmente ao [estudar as funções](#).
- Existe outra classe **RegExp** que serve para construir padrões que veremos talvez junto com Function quando tratarmos de temas ainda mais avançados.

**Nota: Uso de maiúsculas e minúsculas.** Já que paramos anteriormente para falar sobre o uso de maiúsculas em certas letras dos nomes das classes, vamos terminar de explicar a convenção que se realiza em Javascript para nomear os elementos.

Para começar, qualquer variável costuma-se escrever em minúsculas, embora se este nome de variável se compõe de várias palavras, costuma-se escrever em maiúscula a primeira letra das seguintes palavras. Isto se faz em qualquer tipo de variável ou nome nos nomes das classes, onde se escreve também em maiúscula o primeiro caractere da primeira palavra.

Exemplos:

**Number**, que é uma classe se escreve com a primeira em maiúscula.

**RegExp**, que é o nome de outra classe composto por duas palavras, tem a primeira letra das duas palavras em maiúscula.

**minhaCadeia**, que suponhamos que é um objeto da classe String, se escreve com a primeira letra em minúscula e a primeira letra da segunda palavra em maiúscula.

**data**, que suponhamos que é um objeto da classe Date, se escreve em minúsculas por ser um objeto.

**minhaFunção()**, que é uma função definida por nós, costuma-se escrever com minúscula a primeira.

Como dissemos, esta é a norma geral para dar nomes às variáveis, classes, objetos, funções,, etc. em Javascript. Se a seguimos assim, não teremos problemas na hora de saber se um nome em Javascript tem ou não alguma maiúscula e ademais todo nosso trabalho será mais claro e fácil de seguir por outros programadores ou por nós mesmos no caso de retomar um código uma vez passado algum tempo.

Artigo por **Miguel Angel Alvarez - Tradução de JML**

## Classe string em Javascript

Em javascript as variáveis de tipo texto são objetos da classe String. Isto quer dizer que cada uma das variáveis que criamos de tipo texto tem uma série de propriedades e métodos. Lembramos que as propriedades são as características, como, por exemplo, longitude em caracteres do string e os métodos são funcionalidades, como podem ser extrair um substring ou colocar o texto em maiúsculas.

Para criar um objeto da classe String a única coisa que há que fazer é atribuir um texto a uma variável. O texto vai entre aspas, como já vimos nos capítulos de sintaxe. Também se pode criar um objeto string com o operador new, que veremos mais adiante. A única diferença é que

em versões de Javascript 1.0 não funcionará new para criar os Strings.

## Propriedades de String

### Length

A classe String tem somente uma propriedade: length, que salva o número de caracteres do String.

## Métodos de String

Os objetos da classe String têm uma boa quantidade de métodos para realizar muitas coisas interessantes. Primeiro, vamos ver uma lista dos métodos mais interessantes e logo, veremos outra lista de métodos menos úteis.

### charAt(índice)

Devolve o caractere que há na posição indicada como índice. As posições de um string começam em 0.

### indexOf(caractere,desde)

Devolve a posição da primeira vez que aparece o caractere indicado por parâmetro em um string. Se não encontra o caractere no string devolve -1. O segundo parâmetro é opcional e serve para indicar a partir de que posição se deseja que comece a busca.

### lastIndexOf(caractere,desde)

Busca a posição de um caractere exatamente igual a como faz a função indexOf, mas desde o final no lugar do princípio. O segundo parâmetro indica o número de caracteres desde onde se busca, igual que em indexOf.

### replace(substring\_a\_buscar,novoStr)

Implementado em Javascript 1.2, serve para substituir porções do texto de um string por outro texto, por exemplo, poderíamos utilizá-lo para substituir todas as aparições do substring "xxx" por "yyy". O método não substitui no string, e sim, devolve uma resultante de fazer essa substituição. Aceita expressões regulares como substring a buscar.

### split(separador)

Este método só é compatível com javascript 1.1 em diante. Serve para criar um vetor a partir de um String no que cada elemento é a do String que está separada pelo separador indicado por parâmetro.

### substring(início,fim)

Devolve o substring que começa no caractere de início e termina no caractere de fim. Se intercambiarmos os parâmetros de início e fim também funciona. Simplesmente nos dá o substring que há entre o caractere menor e o maior.

### toLowerCase()

Coloca todos os caracteres de um string em minúsculas.

### toUpperCase()

Coloca todos os caracteres de um string em maiúsculas.

### toString()

Este método tem todos os objetos e se usa para convertê-los em cadeias.



Até aqui vimos os métodos que nos ajudará a respeito das cadeias. Agora vamos ver outros métodos que são menos úteis, mas que há que indicá-los para que fique a constância de todos. Todos servem para aplicar estilos a um texto e é como se utilizássemos etiquetas HTML. Vejamos como:

**anchor(name)**

Converte em uma âncora (lugar a onde encaminhar um link) uma cadeia de caracteres usando como o atributo name da etiqueta <A> o que recebe por parâmetro.

**big()**

Aumenta o tamanho de letra do string. É como se colocássemos em um string ao princípio a etiqueta <BIG> e ao final </BIG>.

**blink()**

Para que pisque o texto do string, é como utilizar a etiqueta <BLINK>. Vale somente para Netscape.

**bold()**

Como se utilizássemos a etiqueta <B>.

**fixed()**

Para utilizar uma fonte com um espaçamento único, etiqueta <TT>.

**fontColor(color)**

Coloca a fonte a essa cor. Como utilizar a etiqueta <FONT color=a\_cor\_indicada>.

**fontSize(tamanho)**

Coloca a fonte no tamanho indicado. Como se utilizássemos a etiqueta <FONT> com o atributo size.

**italics()**

Coloca a fonte em cursiva. Etiqueta <I>.

**link(url)**

Coloca o texto como um link à URL indicada. É como se utilizássemos a etiqueta <A> com o atributo href indicado como parâmetro.

**small()**

É como utilizar a etiqueta <SMALL>

**strike()**

É Como utilizar a etiqueta <STRIKE>, que serve para que o texto apareça riscado.

**sub()**

Atualiza o texto como se estivesse utilizando a etiqueta <SUB>, de subíndice.

**sup()**

É Como se utilizássemos a etiqueta <SUP>, de superíndice.

*Artigo por Miguel Angel Alvarez - Tradução de JML*

## Exemplos de funcionamento da classe String

Agora veremos uns exemplos sobre como se utilizam os métodos e propriedades do objeto String.

### Exemplo de strings 1

Vamos escrever o conteúdo de um string com um caractere separador ("-") entre cada um dos caracteres do string.

```
var meuString = "Ola Amigos"
var result = ""

for (i=0;i<meuString.length-1;i++) {
    result += meuString.charAt(i)
    result += "-"
}
result += meuString.charAt(meuString.length - 1)

document.write(result)
```

Primeiro, criamos duas variáveis, uma com o string a percorrer e outra com um string vazio, onde salvaremos o resultado. Logo, fazemos um loop que percorre desde o primeiro até o penúltimo caractere do string -utilizamos a propriedade length para conhecer o número de caracteres do string- e em cada iteração colocamos um caractere do string seguido de um caractere separador "-". Como ainda nos resta o último caractere para colocar, o colocamos na seguinte linha depois do loop. Utilizamos a função charAt para acessar as posições do string. Finalmente, imprimimos na página o resultado.

Podemos [ver o exemplo em funcionamento em uma página a parte](#).

### Exemplo de strings 2

Vamos fazer um script que rompa um string em duas metades e as imprima por tela. As metades serão iguais, sempre que o string tenha um número de caracteres par. No caso de que o número de caracteres seja ímpar não poderá ser feito a metade exata, mas partiremos o string o mais aproximado à metade.

```
var meuString = "0123456789"
var metade1,metade2

posicao_metade = meuString.length / 2

metade1 = meuString.substring(0,posicao_metade)
metade2 = meuString.substring(posicao_metade,meuString.length)

document.write(metade1 + "<br>" + metade2)
```

As duas primeiras linhas servem para declarar as variáveis que vamos utilizar e iniciar o string para partir. Na seguinte linha, encontramos a posição da metade do string.

Nas seguintes linhas é onde realizamos o trabalho de colocar em uma variável a primeira metade do string e na outra a segunda. Para isso, utilizamos o método substring passando como início e fim no primeiro caso desde 0 até a metade e no segundo desde a metade até o final. Para finalizar imprimimos as duas metades com uma quebra de linhas entre elas.

Podemos [ver o exemplo em funcionamento em uma página a parte](#).

*Artigo por Miguel Angel Alvarez - Tradução de JML*

## Classe Date em Javascript

Sobre este objeto recai todo o trabalho com datas em Javascript, como obter uma data, o dia, a hora e outras coisas. Para trabalhar com datas precisamos instanciar um objeto da classe Date e com ele já podemos realizar as operações que necessitamos.

Um objeto da classe Date pode ser criado de duas maneiras distintas. Por um lado podemos criar o objeto com o dia e hora atuais e por outro podemos cria-lo com um dia e hora distintos aos atuais. Isto depende dos parâmetros que passemos ao construir os objetos.

Para criar um objeto fecha com o dia e hora atuais, colocamos os parênteses vazios ao chamar ao construtor da classe Date.

```
minhaData = new Date()
```

Para criar um objeto data com um dia e hora diferentes dos atuais temos que indicar entre parênteses o momento para iniciar o objeto. Existem várias maneiras de expressar um dia e hora válida, por isso podemos construir uma data nos guiando por vários esquemas. Estes são dois deles, suficientes para criar todo tipo de datas e horas.

```
minhaData = new Date(ano,mês,dia,hora,minutos,segundos)
minhaData = new Date(ano,mês,dia)
```

Os valores que deve receber o construtor são sempre numéricos. Um detalhe, o mês começa por 0, ou seja, janeiro é o mês 0. Se não indicamos a hora, o objeto data se cria com hora 00:00:00.

Os objetos da classe Date não têm propriedades, mas existem um montão de métodos que têm, vamos vê-los agora.

### **getDate()**

Devolve o dia do mês.

### **getDay()**

Devolve o dia da semana.

### **getHours()**

Retorna a hora.

**getMinutes()**

Devolve os minutos.

**getMonth()**

Devolve o mês (atenção ao mês que começa por 0).

**getSeconds()**

Devolve os segundos.

**getTime()**

Devolve os segundos transcorridos entre o dia 1 de janeiro de 1970 e a data correspondente ao objeto ao que se passa a mensagem.

**getYear()**

Retorna o ano, ao que se restou 1900. Por exemplo, para o 1995 retorna 95, para o 2005 retorna 105. Este método está obsoleto em Netscape a partir da versão 1.3 de Javascript e agora se utiliza getFullYear().

**getFullYear()**

Retorna o ano com todos os dígitos. Usar este método para estar certos de que funcionará todo bem em datas posteriores ao ano 2000.

**setDate()**

Atualiza o dia do mês.

**setHours()**

Atualiza a hora.

**setMinutes()**

Muda os minutos.

**setMonth()**

Muda o mês (atenção ao mês que começa por 0).

**setSeconds()**

Muda os segundos.

**setTime()**

Atualiza a data completa. Recebe um número de segundos desde 1 de janeiro de 1970.

**setYear()**

Muda o ano recebe um número, ao que lhe soma 1900 antes de coloca-lo como ano da data. Por exemplo, se recebe 95 colocará o ano 1995. Este método está obsoleto a partir de Javascript 1.3 em Netscape. Agora se utiliza setFullYear(), indicando o ano com todos os dígitos.

**setFullYear()**

Muda o ano da data ao número que recebe por parâmetro. O número se indica completo ex: 2005 ou 1995. Utilizar este método para estar certo de que tudo funciona para datas posteriores a 2000.

Como foi possível apreciar, há algum método obsoleto por questões relativas ao ano 2000: `setYear()` e `getYear()`. Estes métodos se comportam bem na Internet Explorer e não nos dará nenhum problema utilizá-los. Entretanto, não funcionarão corretamente em Netscape, portanto é interessante utilizarmos em seu lugar os métodos `getFullYear()` e `setFullYear()`, que funcionam bem em Netscape e Internet Explorer.

*Artigo por Miguel Angel Alvarez - Tradução de JML*

## Exemplo de funcionamento de Date

Neste exemplo vamos criar duas datas, uma com o instante atual e outra com data do passado. Em seguida, imprimiremos as duas e extrairemos seu ano para imprimi-lo também. Logo, atualizaremos o ano de uma das datas e voltaremos a escreve-la com um formato mais legível.

```
//nestas linhas criamos as datas
minhaDataaAtual = new Date()
minhaDataPassada = new Date(1998,4,23)

//nestas linhas imprimimos as datas.
document.write (minhaDataAtual)
document.write ("<br>")
document.write (minhaDtaPassada)

//extraímos o ano das duas datas
anoAtual = minhaDataAtual.getFullYear()
anoPassado = minhaDataPassada.getFullYear()

//Escrevemos em ano na página
document.write("<br>O ano atual é: " + anoAtual)
document.write("<br>O ano passado é: " + anoPassado)

//mudamos o ano na data atual
minhaDataAtual.setFullYear(2005)

//extraímos o dia, mês e ano
dia = minhaDataAtual.getDate()
mes = parseInt(minhaDataAtual.getMonth()) + 1
ano = minhaDataAtual.getFullYear()

//escrevemos a data em um formato legível
document.write ("<br>")
document.write (dia + "/" + mes + "/" + ano)
```

Há que destacar um detalhe antes de terminar, é que o número do mês pode começar desde 0. Pelo menos no Netscape com o qual realizamos as provas começava o mês em 0. Por esta razão somamos um ao mês que devolve o método `getMonth`.

Existem mais detalhes para destacar, pois é que no Netscape o método `getFullYear()` devolve os anos transcorridos desde 1900, com o qual ao obter o ano de uma data de, por exemplo,

2005, indica que é o ano 105. Para obter o ano completo temos a nossa disposição o método `getFullYear()` que devolveria 2005 da mesma forma que em Netscape e Internet Explorer.

Muita atenção no trabalho com datas em distintas plataformas, visto que poderia ser problemático o fato de oferecerem distintas saídas aos métodos de manejo de datas, dependendo sempre da marca e versão de nosso navegador.

*Artigo por Miguel Angel Alvarez - Tradução de JML*

## Classe Math em Javascript

A classe Math proporciona os mecanismos para realizar operações matemáticas em Javascript. Algumas operações se resolvem rapidamente com os operadores aritméticos que já conhecemos, como a multiplicação ou soma, mas existe uma série de operações matemáticas adicionais que têm que se realizar usando a classe Math como podem ser calcular um seno ou fazer uma raiz quadrada.

De modo que para qualquer cálculo matemático complexo utilizaremos a classe Math, com uma particularidade. Até agora, cada vez que quiséssemos fazer algo com uma classe deveríamos instanciar um objeto dessa classe e trabalhar com o objeto e no caso da classe Math se trabalha diretamente com a classe. Isto se permite porque as propriedades e métodos da classe Math são o que se chama propriedades e métodos de classe e para utiliza-los se opera através da classe no lugar dos objetos. Com outras palavras, para trabalhar com a classe Math não deveremos utilizar a instrução `new` e utilizaremos o nome da classe para acessar a suas propriedades e métodos.

### Propriedades de Math

As propriedades salvam valores que provavelmente necessitaremos em algum momento se estamos fazendo cálculos matemáticos. É provável que estas propriedades sejam um pouco raras para as pessoas que desconhecem as matemáticas avançadas, mas os que as conhecem saberão de sua utilidade.

#### **E**

Número E ou constante de Euler, a base dos logaritmos neperianos.

#### **LN2**

Logaritmo neperiano de 2.

#### **LN10**

Logaritmo neperiano de 10.

#### **LOG2E**

Logaritmo em base 2 de E.

#### **LOG10E**

Logaritmo em base 10 de E.

#### **PI**

Conhecido número para cálculo com círculos.

**SQRT1\_2**

Raiz quadrada de um meio.

**SQRT2**

Raiz quadrada de 2.

**Métodos de Math**

Ainda assim, temos uma série de métodos para realizar operações matemáticas típicas, embora um pouco complexas. Todos os que conheçam as matemáticas a um bom nível conhecerão o significado destas operações.

**abs()**

Devolve o valor absoluto de um número. O valor depois de tirar o signo.

**acos()**

Devolve o arco co-seno de um número em radianos.

**asin()**

Devolve o arco co-seno de um número em radianos.

**atan()**

Devolve um arco tangente de um número.

**ceil()**

Devolve o inteiro igual ou imediatamente seguinte de um número. Por exemplo, `ceil(3)` vale 3, `ceil(3.4)` é 4.

**cos()**

Retorna o co-seno de um número.

**exp()**

Retorna o resultado de elevar o número E por um número.

**floor()**

O contrário de `ceil()`, pois devolve um número igual ou imediatamente inferior.

**log()**

Devolve o logaritmo neperiano de um número.

**max()**

Retorna o maior de 2 números.

**min()**

Retorna o menor de 2 números.

**pow()**

Recebe dois números como parâmetros e devolve o primeiro número elevado ao segundo número.

**random()**

Devolve um número aleatório entre 0 e 1. Método criado a partir de Javascript 1.1.

**round()**

Arredonda ao inteiro mais próximo.

**sin()**

Devolve o seno de um número com um ângulo em radianos.

**sqrt()**

Retorna a raiz quadrada de um número.

**tan()**

Calcula e devolve a tangente de um número em radianos.

Exemplo de utilização da classe Math

Vamos ver um simples exemplo sobre como utilizar métodos e propriedades da classe Math para calcular o seno e o co-seno de 2 PI radianos (uma volta completa). Como alguns de vocês sabem, o co-seno de 2 PI radianos deve dar como resultado 1 e o seno 0.

```
document.write (Math.cos(2 * Math.PI))
```

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

```
document.write (Math.sin(2 * Math.PI))
```

2 PI radianos é o resultado de multiplicar 2 pelo número PI. Esse resultado é o que recebe o método cos, e dá como resultado 1. No caso do seno, o resultado não é exatamente 0 mas está aproximado com uma exatidão demais de um milésimo de fração. Escrevem-se o seno e co-seno com uma quebra de linha no meio para que fique mais claro.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Classe Number em Javascript

A classe Number modela o tipo de dados numéricos. Foi acrescentado na versão 1.1 de Javascript (com Netscape Navigator 3). Serve para criar objetos que têm dados numéricos como valor. É muito provável que você não chegue a utilizar em nenhuma ocasião. Pelo menos em todos os scripts que servem para fazer as coisas mais distintas e úteis.

Nota: Conhecemos o [tipo de dados numéricos no primeiro manual de javascript](#). Este nos servia para salvar valores numéricos sem mais. Este objeto modela este tipo de dados e a classe em si, oferece algum método que pode ser útil. Para os cálculos matemáticos e o uso de números em geral vamos utilizar sempre as variáveis numéricas vistas anteriormente.

O valor do objeto Number que se cria depende do que receba o construtor da classe Number. Com estas regras:

- Se o construtor recebe um número, então inicia o objeto com o número que recebe. Se receber um número entre aspas, o converte a valor numérico, devolvendo também tal número.



- Devolve 0 em caso de que não receba nada.
- No caso de que receba um valor não numérico devolve NaN, que significa "Not a Number" (Não é um número)
- Se receber false se inicia a 0 e se receber true se inicia a 1.

Seu funcionamento pode ser resumido nestes exemplos.

```
var n1 = new Number()  
document.write(n1 + "<br>")  
//mostra um 0
```

```
var n2 = new Number("oi")  
document.write(n2 + "<br>")  
//mostra NaN
```

```
var n3 = new Number("123")  
document.write(n3 + "<br>")  
//mostra 123
```

```
var n4 = new Number("123asdfQWERTY")  
document.write(n4 + "<br>")  
//mostra NaN
```

```
var n5 = new Number(123456)  
document.write(n5 + "<br>")  
//mostra 123456
```

```
var n6 = new Number(false)  
document.write(n6 + "<br>")  
//mostra 0
```

```
var n7 = new Number(true)  
document.write(n7 + "<br>")  
//mostra 1
```

### Propriedades da classe Number

Esta classe também nos oferece várias propriedades que contém os seguintes valores:

#### NaN

Como vimos, significa Not a Number, ou em português, não é um número.

#### MAX\_VALUE e MIN\_VALUE

Salvam o valor do máximo e do mínimo valor que se pode representar em Javascript

#### NEGATIVE\_INFINITY e POSITIVE\_INFINITY

Representam os valores, negativos e positivos respectivamente, a partir dos quais há transbordamento.

Estas propriedades são de classe, portanto acessaremos-las a partir do nome da classe, tal como podemos ver neste exemplo no qual se mostra cada um dos seus valores.

```
document.write("Propriedade NaN: " + Number.NaN)
document.write("<br>")
document.write("Propriedade MAX_VALUE: " + Number.MAX_VALUE)
document.write("<br>")
document.write("Propriedade MIN_VALUE: " + Number.MIN_VALUE)
document.write("<br>")
document.write("Propriedade NEGATIVE_INFINITY: " + Number.NEGATIVE_INFINITY)
document.write("<br>")
document.write("Propriedade POSITIVE_INFINITY: " + Number.POSITIVE_INFINITY)
```

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Classe Boolean em Javascript

Esta classe serve para criar valores booleanos. Foi acrescentado na versão 1.1 de Javascript (com Netscape Navigator 3). Uma de suas possíveis é a de conseguir valores booleanos a partir de dados de qualquer outro tipo.

**Nota:** Conhecemos o [tipo de dados boolean no primeiro manual de Javascript](#). Este nos servia para salvar um valor verdadeiro (true) ou falso (false). Esta classe modela esse tipo de dados para criar objetos booleanos.

Dependendo do que receba o construtor da classe Boolean o valor do objeto booleano que se cria será verdadeiro ou falso, da seguinte maneira:

- **Inicia-se a false** quando você não passa nenhum valor ao construtor, ou se passa uma cadeia vazia, o número 0 ou a palavra false sem aspas.
- **Inicia-se a true** quando recebe qualquer valor entre aspas ou qualquer número distinto de 0.

Pode-se compreender o funcionamento deste objeto facilmente se examinarmos alguns exemplos.

```
var b1 = new Boolean()
document.write(b1 + "<br>")
//mostra false

var b2 = new Boolean("")
document.write(b2 + "<br>")
//mostra false

var b25 = new Boolean(false)
document.write(b25 + "<br>")
//mostra false

var b3 = new Boolean(0)
```

```
document.write(b3 + "<br>")
//mostra false

var b35 = new Boolean("0")
document.write(b35 + "<br>")
//mostra true

var b4 = new Boolean(3)
document.write(b4 + "<br>")
//mostra true

var b5 = new Boolean("Olá")
document.write(b5 + "<br>")
//mostra true
```

Pode-se [ver em funcionamento o exemplo em uma página a parte](#).

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Criação de classes em Javascript

Agora que já conhecemos um pouco os objetos e aprendemos a manejá-los podemos passar a um tema mais avançado, como é o de construir nossos próprios objetos, que pode ser útil em certas ocasiões para temas avançados.

Então vamos a ver como podemos definir nossos próprios objetos, com suas propriedades e métodos, de maneira que aprendamos o mecanismo, mas sem entrar muito em aspectos práticos que deixaremos para exemplos no futuro.

Para criar nossos próprios objetos devemos criar uma classe, que lembramos que é algo assim como a definição de um objeto com suas propriedades e métodos. Para criar a classe em Javascript devemos escrever uma função especial, que se encarregará de construir o objeto na memória e inicia-lo. Esta função chama-se construtor na terminologia da programação orientada a objetos.

```
function MinhaClasse (valor_iniciacao){
    //Inicio as propriedades e métodos
    this.minhaPropriedade = valor_iniciacao
    this.meuMetodo = nome_de_uma_funcao_definida
}
```

Isso era um construtor. Utiliza a palavra this para declarar as propriedades e métodos do objeto que se está construindo. This faz referência ao objeto que se está construindo, pois lembremos que a esta função a chamaremos para construir um objeto. A esse objeto que se está construindo lhe vamos atribuindo valores em suas propriedades e também lhe vamos atribuindo nomes de funções definidas para seus métodos. Ao construir um objeto tecnicamente é igual que declarar uma propriedade ou um método, somente difere em que a uma propriedade lhe atribuímos um valor e a um método lhe atribuímos uma função.

## A classe AlunoUniversitário

Veremos tudo mais detalhadamente se fazemos um exemplo. Neste exemplo, vamos criar um objeto estudante universitário. Como estudante terá umas características como o nome, a idade ou o número de matrícula. Ademais poderá ter algum método como, por exemplo, matricular ao aluno.

### Construtor: Colocamos propriedades

Vejamos como definir o construtor da classe AlunoUniversitário, mas somente vamos colocar por agora as propriedades da classe.

```
function AlunoUniversitario(nome, idade){  
    this.nome = nome  
    this.idade = idade  
    this.numMatricula = null  
}
```

O construtor recebe os valores de iniciação como parâmetros, neste caso é só o nome e a idade, porque o número de matrícula o aluno não recebe até que esteja matriculado. É por isso que atribuímos a null a propriedade numMatricula.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Criação de classes em Javascript II

Para construir um método devemos criar uma função. Uma função que se constrói com a intenção de que seja um método para uma classe pode utilizar também a variável this, que faz referência ao objeto sobre o que invocamos o método. Portanto, devemos recordar que para chamar a um método devemos ter um objeto e this faz referência a esse objeto.

```
function matriculese(num_matricula){  
    this.numMatricula = num_matricula  
}
```

A função matricular recebe um número de matrícula por parâmetro e o atribui à propriedade numMatricula do objeto que recebe este método. Assim, preenchemos o da propriedade que nos faltava.

Vamos construir outro método que imprime os dados do aluno.

```
function imprimir(){  
    document.write("Nome: " + this.nome)  
    document.write("<br>Idade: " + this.idade)  
    document.write("<br>Número de matrícula: " + this.numMatricula)  
}
```

Esta função vai imprimindo todas as propriedades do objeto que recebe o método.

## Construtor: Colocamos métodos

Para colocar um método em uma classe devemos atribuir a função que queremos que seja o método ao objeto que está sendo criado. Vejamos como ficaria o construtor da classe AlunoUniversitario com o método matricular.

```
function AlunoUniversitario(nome, idade){  
    this.nome = nome  
    this.idade = idade  
    this.numMatricula = null  
    this.matriculese = matriculese  
    this.imprimir = imprimir  
}
```

Vemos que nas últimas linhas atribuímos aos métodos os nomes das funções que contém seu código.

## Para instanciar um objeto

Para instanciar objetos da classe AlunoUniversitario utilizamos a sentença new, que já tivemos a oportunidade de ver em outras ocasiões.

```
meuAluno = new AlunoUniversitario("José Dias",23)
```

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Criação de classes em Javascript III

Para ilustrar o trabalho com objetos e terminar com o exemplo do AlunoUniversitário, vamos ver todo este processo em um só script no qual definiremos a classe e logo a utilizaremos um pouquinho.

```
//definimos o método matriculese para a classe AlunoUniversitario  
function matriculese(num_matricula){  
    this.numMatricula = num_matricula  
}  
  
//definimos o método imprimir para a classe AlunoUniversitario  
function imprimir(){  
    document.write("<br>Nome: " + this.nome)  
    document.write("<br>Idade: " + this.idade)  
    document.write("<br>Número de matrícula: " + this.numMatricula)  
}  
  
//definimos o construtor para a classe  
function AlunoUniversitario(nome, idade){  
    this.nome = nome  
    this.idade = idade  
    this.numMatricula = null  
    this.matriculese = matriculese
```

```
this.imprimir = imprimir
}

//criamos um aluno
meuAluno = new AlunoUniversitario("José Dias",23)

//pedimos que se imprima
meuAluno.imprimir()

//pedimos que se matricule
meuAluno.matriculese(305)

//pedimos que se imprima de novo (com o número de matrícula preenchido)
meuAluno.imprimir()
```

Se desejarmos, podemos [ver o script em funcionamento em uma página a parte](#).

No momento não vamos mais falar sobre como criar e utilizar nossos próprios objetos, mas no futuro trataremos este tema com mais profundidade e faremos algum exemplo adicional.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Hierarquia de objetos do navegador

Chegamos ao tema mais importante para aprender a manejar Javascript com toda sua potência, o tema no qual aprenderemos a controlar ao navegador e os distintos elementos da página.

Sem dúvida, este tema dará muita vida aos nossos exemplos, já que até agora não tinham muito caráter prático porque não trabalhavam com o navegador e com as páginas, que é realmente para o que está feito Javascript. De modo que esperamos a partir de agora que o manual seja mais entretido para todos, porque cobrirá os aspectos mais práticos.

Quando se carrega uma página, o navegador cria uma hierarquia de objetos na memória que servem para controlar os distintos elementos de tal página. Com Javascript e com a nomenclatura de objetos que aprendemos, podemos trabalhar com essa hierarquia de objetos, acessar às suas propriedades e invocar seus métodos.

Qualquer elemento da página pode ser controlado de uma maneira ou de outra acessando a essa hierarquia. É crucial conhecê-la bem para poder controlar perfeitamente as páginas web com Javascript ou qualquer outra linguagem de programação do lado do cliente.

### Exemplo de acesso à hierarquia

Antes de começar a ver rigorosamente a hierarquia de objetos do navegador, vamos ver o exemplo típico de acesso a uma propriedade desta hierarquia para mudar o aspecto da página. Trata-se de uma propriedade da página que armazena a cor de fundo da página web: a propriedade `bgColor` do objeto `document`.

```
document.bgColor = "red"
```

Se executarmos esta sentença em qualquer momento, mudamos a cor de fundo da página para vermelho. Há que observar que a propriedade bgColor tem a "C" em maiúscula. É um erro típico se equivocar com as maiúsculas e minúsculas na hierarquia. Se não o escrevemos bem não funcionará e em alguns casos nem sequer dará uma mensagem de erro.

Nesta página definida com a cor de fundo branca mudamos essa propriedade com Javascript, com o que sairá com a cor de fundo vermelha.

```
<HTML>
<HEAD>
  <TITLE>Prova bgColor</TITLE>
</HEAD>
<BODY bgcolor=white>

<script>
  document.bgColor = "red"
</script>
</BODY>
</HTML>
```

Podemos [ver esta página em funcionamento](#) em uma janela a parte.

Nos exemplos que vimos até agora também fizemos uso dos objetos da hierarquia do navegador. Teoricamente utilizamos muito o método write() do objeto document para escrever um texto na página.

```
document.write("O texto a escrever")
```

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Trabalhando com a hierarquia em Javascript

Vamos ver agora como está composta esta hierarquia. Os objetos que fazem parte dela estão representados no seguinte gráfico.



### Hierarquia de objetos do navegador em Javascript 1.2.

Poderia faltar por percorrer algum objeto, mas serve perfeitamente para ter uma idéia de como se organizam os objetos na hierarquia.

Como se pode observar, todos os objetos começam em um objeto que se chama window. Este objeto oferece uma série de métodos e propriedades para controlar a janela do navegador.



Com eles podemos controlar o aspecto da janela, a barra de estado, abrir janelas secundárias e outras coisas que veremos mais adiante quando explicarmos com detalhe o objeto.

Além de oferecer controle, o objeto window dá acesso a outros objetos como o documento (a página web que se está visualizando), o histórico de páginas visitadas ou os distintos frames da janela. De modo que para acessar a qualquer outro objeto da hierarquia deveríamos começar pelo objeto window. Tanto é que javascript entende perfeitamente que a hierarquia começa em window embora no o atribuímos.

Nos exemplos incluídos no capítulo anterior podíamos ter escrito também as sentenças de acesso à hierarquia começando pelo objeto window, desta maneira.

```
window.document.bgColor = "red"  
window.document.write("O texto a escrever")
```

Não o fizemos para que o código ficasse mais claro e economizar algo de texto, mas agora já sabemos que toda a hierarquia começa no objeto window.

### **As propriedades de um objeto podem ser por sua vez outros objetos**

Muitas das propriedades do objeto window são por sua vez outros objetos. É o caso de objetos como o histórico de páginas web ou o objeto documento, que têm por sua vez outras propriedades e métodos.

Entre eles se destaca o objeto document, que contem todas as propriedades e métodos necessários para controlar muitos aspectos da página. Já vimos alguma propriedade como bgColor, mas tem muitas outras como o título da página, as imagens que existem incluídas, os formulários, etc. Muitas propriedades deste objeto são por sua vez outros objetos, como os formulários. Veremos tudo isto quando tratarmos cada um dos objetos separadamente. Ademais, o objeto document tem métodos para escrever na página web e para manejar eventos da página.

### **Navegação através da hierarquia**

O objetivo deste capítulo sobre a hierarquia de objetos é aprender a navegar por ela para acessar a qualquer elemento da página. Esta não é uma tarefa difícil, mas pode haver algum caso especial em que acessar aos elementos da página se faça de uma maneira que ainda não comentamos.

Como já dissemos, toda a hierarquia começa no objeto window, embora não era necessário fazer referência a window para acessar a qualquer objeto da hierarquia. Logo, em importância está o objeto document, onde podemos encontrar alguma propriedade especial que vale a pena comentar separadamente, porque seu acesso é um pouco diferente. Trata-se de propriedades que são arrays, por exemplo, a propriedade imagens é um array com todas as imagens da página web. Também encontramos arrays para salvar os links da página, os applets, os formulários e as âncoras.

Quando uma página se carrega, o navegador constrói na memória a hierarquia de objetos. De maneira adicional, constrói também estes arrays de objetos. Por exemplo, no caso das imagens, vai criando o array colocando na posição 0 a primeira imagem, na posição 1 a segunda imagem e assim até que introduzimos todas. Vamos ver um loop que percorre todas as imagens da página e imprime sua propriedade src, que contem a URL onde está situada a

imagem.

```
for (i=0;i<document.images.length;i++){  
    document.write(document.images[i].src)  
    document.write("<br>")  
}
```

Utilizamos a propriedade length do array imagens para limitar o número de interações do loop. Logo, utilizamos o método write() do objeto document passando o valor de cada uma das propriedades src de cada imagem.

Podemos ver uma [página com varias imágenes donde se accede a sus propiedades com o loop anterior](#).

Agora vamos ver o uso de outro array de objetos. Neste caso vamos acessar um pouco mais dentro da hierarquia para chegar à matriz elements dos objetos formulário, que contem cada um dos elementos que compõem o formulário. Para isso, teremos que acessar a um formulário da página, ao que poderemos acessar pelo array de formulários, e dentro dele à propriedade elements, que é outro array de objetos. Para cada elemento do formulário vamos escrever sua propriedade value, que corresponde com a propriedade value que colocamos em HTML.

```
for (i=0;i<document.forms[0].elements.length;i++){  
    document.write(document. forms[0].elements[i].value)  
    document.write("<br>")  
}
```

é um loop muito parecido ao que tínhamos para percorrer as imagens, com a diferença que agora percorremos o vector de elements, ao que acessamos pela hierarquia de objetos passando pelo array de formulários em sua posição 0, que corresponde como primeiro formulário da página.

Para ver este exemplo em funcionamento, temos uma [página com um formulário onde executa este percorrido a seus elementos](#).

Com isto, aprendemos a nos mover pela hierarquia de objetos, com o que poderemos acessar a qualquer elemento do navegador ou a página. Mais adiante conheceremos com detalhe cada um dos objetos da hierarquia, começando pelo objeto window e baixando pela hierarquia até ver todos.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Objeto Window de Javascript

É o objeto principal na hierarquia e contém as propriedades e métodos para controlar a janela do navegador. Dele dependem todos os demais objetos da hierarquia. Vamos ver a lista de suas propriedades e métodos.

### Propriedades do objeto window

A seguir podemos ver as propriedades do objeto window. Existem algumas muito úteis e

outras menos.

**closed**

Indica a possibilidade de que tenha fechado a janela. (Javascript 1.1)

**defaultStatus**

Texto que se escreve por padrão na barra de estado do navegador.

**document**

Objeto que contem na página web que está sendo mostrada.

**Frame**

Um objeto frame de uma página web. Acessa-se pelo nome dele.

**frames array**

O vetor que contem todos os frames da página. Acessa-se pelo índice a partir de 0.

**history**

Objeto histórico de páginas visitadas.

**innerHeight**

Tamanho em pixels do espaço onde se visualiza a página, na vertical. (Javascript 1.2)

**innerWidth**

Tamanho em pixels do espaço onde se visualiza a página, na horizontal. (Javascript 1.2)

**length**

Número de frames da janela.

**location**

A URL do documento que está sendo visualizando. Podemos mudar o valor desta propriedade para irmos a outra página. Ver também a propriedade location do objeto document.

**locationbar**

Objeto barra de endereços da janela. (Javascript 1.2)

**menubar**

Objeto barra de menus da janela. (Javascript 1.2)

**name**

Nome da janela. Atribuímos quando abrimos uma nova janela.

**opener**

Faz referência à janela do navegador que abriu a janela onde estamos trabalhando. Será vista com mais detalhes no tratamento de janelas com Javascript.

**outerHeight**

Tamanho em pixels do espaço de toda a janela, na vertical. Isto inclui as barras de deslocamento, botões, etc. (Javascript 1.2)

**outerWidth**

Tamanho em pixels do espaço de toda a janela, na horizontal. Isto inclui as barras de

deslocamento. (Javascript 1.2)

**parent**

Faz referência à janela onde está situado o frame que estamos trabalhando. Veremos detalhadamente ao estudar o controle de frames com Javascript.

**personalbar**

Objeto barra pessoal do navegador. (Javascript 1.2)

**self**

Janela ou frame atual.

**scrollbars**

Objeto das barras de deslocamento da janela.

**status**

Texto da barra de estado.

**statusbar**

Objeto barra de estado do navegador. (Javascript 1.2)

**toolbar**

Objeto barra de ferramentas. (Javascript 1.2)

**top**

Faz referência à janela onde está situado o frame que estamos trabalhando. Como a propriedade parent.

**window**

Faz referência à janela atual, assim como a propriedade self.

Vamos ver um exemplo de utilização da propriedade status do objeto window. Esta propriedade serve para escrever um texto na barra de estado do navegador (a barra debaixo da janela). Neste exemplo tivemos que nos adiantar um pouco no andamento do manual, pois utilizamos um manipulador de eventos e que ainda não vimos o que são. Em concreto, utilizamos o manipulador de eventos onclick, que serve para executar sentenças Javascript quando o usuário clica um elemento da página.

Os manipuladores de eventos são colocados em etiquetas HTML, em nosso caso, colocamos em um botão de formulário. As sentenças Javascript associadas ao evento onclick do botão serão executadas quando clicarmos o botão.

Vejamoss já o código que faz com que mude o texto da barra de estado quando clicarmos um botão.

```
<form>  
<input type="Button" value="Clique-me!" onclick="window.status='Olá a todo mundo!'">  
</form>
```

Simplesmente atribuímos um texto à propriedade status do objeto window. O texto que colocamos na barra de estado está escrito entre aspas simples porque estamos escrevendo dentro de umas aspas duplas.

Podemos ver uma [página a parte com este exemplo](#).

*Artigo por Miguel Angel Alvarez - Tradução de JML*

## Métodos de window em Javascript

Vamos ver agora os distintos métodos que tem o objeto window. Muitos destes métodos terão que ser vistos separadamente porque são muito úteis e ainda não o utilizamos, agora vamos lista-los e já veremos alguns exemplos.

### **alert(texto)**

Apresenta uma janela de alerta onde se pode ler o texto que recebe por parâmetro.

### **back()**

Ir uma página atrás no histórico de páginas visitadas. Funciona como o botão de voltar da barra de ferramentas. (Javascript 1.2)

### **blur()**

Tirar o foco da janela atual. (Javascript 1.1)

### **captureEvents(eventos)**

Captura os eventos que se indiquem por parâmetro (Javascript 1.2).

### **clearInterval()**

Elimina a execução de sentenças associadas a um intervalo indicadas com o método setInterval().(Javascript 1.2)

### **clearTimeout()**

Elimina a execução de sentenças associadas a um tempo de espera indicadas com o método setTimeout().

### **close()**

Fecha a janela. (Javascript 1.1)

### **confirm(texto)**

Mostra uma janela de confirmação e permite aceitar ou rejeitar.

### **find()**

Mostra uma janelinha de busca. (Javascript 1.2 para Netscape)

### **focus()**

Coloca o foco da aplicação na janela. (Javascript 1.1)

### **forward()**

Ir uma página adiante no histórico de páginas visitadas. Como se clicássemos o botão de adiante do navegador. (Javascript 1.2)

### **home()**

Ir à página de início o explorador que tenha configurado. (Javascript 1.2)

**moveBy(pixelsX, pixelsY)**

Move a janela do navegador os pixels que se indicam por parâmetro para a direita e para baixo. (Javascript 1.2)

**moveTo(pixelsX, pixelsY)**

Move a janela do navegador à posição indicada nas coordenadas que recebe por parâmetro. (Javascript 1.2)

**open()**

Abre uma janela secundária do navegador.

**print()**

Como se clicássemos o botão de imprimir do navegador. (Javascript 1.2)

**prompt(pergunta, inicio\_da\_resposta)**

Mostra uma caixa de diálogo para pedir um dado. Devolve o dado que se escreveu.

**releaseEvents(eventos)**

Deixa de capturar eventos do tipo que se indique por parâmetro. (Javascript 1.2)

**resizeBy(pixelslargo, pixelsAlto)**

Redimensiona o tamanho da janela, acrescentando ao seu tamanho atual os valores indicados nos parâmetros. O primeiro para a altura e o segundo para a largura. Admite valores negativos se se deseja reduzir a janela. (Javascript 1.2)

**resizeTo(pixelslargo, pixelsAlto)**

Redimensiona a janela do navegador para que ocupe o espaço em pixels que se indica por parâmetro (Javascript 1.2)

**routeEvent()**

Encaminha um evento pela hierarquia de eventos. (Javascript 1.2)

**scroll(pixelsX, pixelsY)**

Faz um scroll da janela para a coordenada indicada por parâmetro. Este método está desaconselhado, pois agora se debería utilizar scrollTo()(Javascript 1.1)

**scrollBy(pixelsX, pixelsY)**

Faz um scroll do conteúdo da janela relativo à posição atual. (Javascript 1.2)

**scrollTo(pixelsX, pixelsY)**

Faz um scroll da janela à posição indicada pelo parâmetro. Este método tem que ser utilizado ao invés do scroll. (Javascript 1.2)

**setInterval()**

Define um script para que seja executado indefinidamente em cada intervalo de tempo. (Javascript 1.2)

**setTimeout(sentença, segundos)**

Define um script para que seja executado uma vez depois de um tempo de espera determinado.

**stop()**

Como clicar o botão de stop da janela do navegador. (Javascript 1.2)

Para ilustrar um pouco melhor o funcionamento de algum destes métodos -os mais estranhos-, criamos uma página web que os utiliza. O código da página mostra-se a seguir e também podemos [ver a página em funcionamento](#).

```
<form>
<input type="button" value="Janela de busca (Somente Netscape)" onClick="window.find()">
<br>
<br>
<input type="button" value="Mover a janela 10 direita,10 abaixo" onClick="moveBy(10, 10)">
<br>
<br>
<input type="button" value="Mover a janela ao ponto (100,10)" onClick="moveTo(100, 10)">
<br>
<br>
<input type="button" value="Imprimir esta página" onClick="print()">
<br>
<br>
<input type="button" value="Aumentar a janela 10 largura,10 comprimento" onClick="resizeBy(10, 10)">
<br>
<br>
<input type="button" value="Fixar o tamanho da janela em 400 x 200" onClick="resizeTo(400, 200)">
<br>
<br>
<input type="button" value="Scroll acima de tudo" onClick="scroll(0,0)">
<br>
<br>
<input type="button" value="Scroll acima 10 pixels" onClick="scrollBy(0,-10)">
</form>
```

Estes exemplos são muito simples, embora pouco práticos. Unicamente trata-se de uma série de botões que, ao clicá-los, chamam a outros tantos métodos do objeto window. No atributo onclick da etiqueta do botão se indicam as sentenças Javascript que queremos que se executem quando se clica sobre tal botão.

No capítulo seguinte veremos outros exemplos realizados com métodos do objeto window de Javascript, um pouco mais detalhados.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Exemplos de métodos de Window

Agora vamos realizar algum exemplo de utilização dos métodos da janela. Vamos nos centrar nos exemplos que servem para por caixas de diálogo, que são muito úteis.

### Caixa de alerta

Para por um texto em uma janelinha com um botão de aceitar. Recebe o texto por parâmetro.

```
window.alert("Este é o texto que sai")
```

Como o objeto window se subentende podemos escrevê-lo assim.

```
alert("Este é o texto que sai")
```

Coloca-se uma janela como a que [pode se ver nesta página](#).

### Caixa de confirmação

Mostra uma janela com um texto indicado por parâmetro e um botão de aceitar e outro de rejeitar. Dependendo do botão que se clica devolve um true (se clicar aceitar) ou um false (se clicar rejeitar).

```
<script>
var resposta = confirm("Aceite-me ou rejeite-me")
alert ("Você clicou: " + resposta)
</script>
```

Este script mostra uma caixa de diálogo confirm e logo mostra em outra caixa de diálogo alert o conteúdo da variável que devolveu a caixa de diálogo. Novamente, [podemos ver o funcionamento deste script se acessamos a esta página a parte](#).

### Caixa de introdução de um dado

Mostra uma caixa de diálogo onde se formula uma pergunta e existe um campo de texto para escrever uma resposta. O campo de texto aparece preenchido com o que se escreva no segundo parâmetro do método. Também existe um botão de aceitar e outro de cancelar. Neste caso de clicar aceitar, o método devolve o texto que foi escrito. Se foi clicado cancelar, devolve null.

O exemplo seguinte serve para pedir o nome da pessoa que está visitando a página e logo mostrar na página uma saudação personalizada. Utiliza um loop para repetir a tomada de dados sempre que o nome da pessoa seja null (porque clicou o botão de cancelar), ou seja, um string vazio (porque não escreveu nada).

```
<script>
nome = null
while (nome == null || nome == ""){
    nome = prompt("Diga-me seu nome:", "")
}
document.write("<h1>Olá " + nome + "</h1>")
</script>
```

Se observarmos na caixa prompt, veremos que recebe dois parâmetros. O segundo era o texto por padrão que sai na caixa como resposta. Deixamos como um string vazio para que não sai nada como texto por padrão.

Podemos [ver este último script em funcionamento em uma página a parte](#).

Até aqui os exemplos dos métodos do objeto window. De qualquer forma, no resto do manual teremos ocasião de ver como trabalhar com muitas propriedades e métodos deste objeto.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*



## Objeto document em Javascript

Com o objeto document controla-se a página web e todos os elementos que contem. O objeto document é a página atual que está sendo visualizada nesse momento. Depende do objeto window, mas também pode depender do objeto frame no caso de que a página esteja sendo mostrada em um frame.

### Propriedades do objeto document

Vejamos uma lista das propriedades do objeto document e logo veremos algum exemplo.

#### **alinkColor**

Cor dos links ativos

#### **Anchor**

Uma âncora da página. Consegue-se com a etiqueta <A name="nome\_da\_ancora">. Acessa-se pelo seu nome.

#### **anchors array**

Um array das âncoras do documento.

#### **Applet**

Um applet da página. Acessa-se pelo seu nome. (Javascript 1.1)

#### **applets array**

Um array com todos os applets da página. (Javascript 1.1)

#### **Area**

Uma etiqueta <AREA>, das que estão vinculadas aos mapas de imagens (Etiqueta ). Acessa-se pelo seu nome. (Javascript 1.1)

#### **bgColor**

A cor de fundo do documento.

#### **classes**

As classes definidas na declaração de estilos CSS. (Javascript 1.2)

#### **cookie**

Uma cookie

#### **domain**

Nome do domínio do servidor da página.

#### **Embed**

Um elemento da página incrustado com a etiqueta <EMBED>. Acessa-se pelo seu nome. (Javascript 1.1)

#### **embeds array**

Todos os elementos da página incrustados com <EMBED>. (Javascript 1.1)

**fgColor**

A cor do texto. Para ver as mudanças há que reescrever a página.

**Form**

Um formulário da página. Acessa-se pelo seu nome.

**forms array**

Um array com todos os formulários da página.

**ids**

Para acessar a estilos CSS. (Javascript 1.2)

**Image**

Uma imagem da página web. Acessa-se pelo seu nome. (Javascript 1.1)

**images array**

Cada uma das imagens da página introduzidas em um array. (Javascript 1.1)

**lastModified**

A data de última modificação do documento.

**linkColor**

A cor dos links.

**Link**

Um link da página. Acessa-se pelo seu nome.

**links array**

Um array com cada um dos links da página.

**location**

A URL do documento que se está visualizando. É somente de leitura.

**referrer**

A página da qual vem o usuário.

**tags**

Estilos definidos às etiquetas de HTML na página web. (Javascript 1.2)

**title**

O título da página.

**URL**

O mesmo que location, mas é aconselhável utilizar location já que URL não existe em todos os navegadores.

**vlinkColor**

A cor dos links visitados.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Exemplos de propriedades de document

Depois de estudar as [propriedades do objeto document](#), veremos algum exemplo para ilustrar o modo de acesso e utilização das mesmas.

### Exemplo com a propriedade bgColor

Vamos utilizar o evento onclick para colocar três botões na página que ao clicá-los mudará a cor de fundo da página.

```
<script>
function mudaCor(colorin){
    document.bgColor = colorin
}

</script>
<form>
<input type="Button" value="Vermelho" onclick="mudaCor('ff0000')">
<input type="Button" value="Verde" onclick="mudaCor('00ff00')">
<input type="Button" value="Azul" onclick="mudaCor('0000ff')">
</form>
```

Primeiro, definimos uma função que será a encarregada de mudar a cor e logo três botões que chamarão a função quando forem clicados passando a cor como parâmetro.

Podemos [ver o exemplo em funcionamento](#).

### Propriedade title

A propriedade title salva a cadeia que conforma o título de nossa página web. Se acessarmos a tal propriedade obteremos o título e se a mudamos, mudará o título da página web.

**Nota:** Lembramos que o título pode ser visto na barra que está acima de tudo da janela do navegador.

Vamos mostrar o título da página em uma caixa de alerta.

```
alert (document.title)
```

Agora vamos fazer uma função que modifica o título da página, atribuindo-lhe o texto que chegue por parâmetro.

```
function mudaTitulo(texto){
    document.title = texto
}
```

Como no exemplo anterior, vamos criar vários botões que chamem a função passando-lhe distintos textos, que se colocarão no título da página.

```
<form>
<input type="Button" value="Titulo = Olá a todos" onclick="mudaTitulo('Olá a todos')">
<input type="Button" value="Titulo = Bem-vindos a minha página web"
onclick="mudaTitulo('Bem-vindos a minha página web')">
<input type="Button" value="Titulo = Mais dias que trabalhamos" onclick="mudaTitulo('Mais
```

```
dias que trabalhamos')">
</form>
```

Podemos [ver em funcionamento este script](#) em outra página web.

*Artigo por Miguel Angel Alvarez - Tradução de JML*

## Métodos de document

O [objeto document](#), localizado abaixo do [objeto window](#) na [hierarquia de objetos de Javascript](#), também tem uma lista de métodos interessantes. Veremos a seguir:

### **captureEvents()**

Para capturar os eventos que acontecem na página web. Recebe como parâmetro o evento que se deseja capturar.

### **close()**

Fecha o fluxo do documento. (Será visto mais adiante neste manual um artigo sobre o fluxo do documento)

### **contextual()**

Oferece uma linha de controle dos estilos da página. No caso de desejarmos especificá-los com Javascript.

### **getSelection()**

Devolve um string que contem o texto que foi selecionado. Funciona somente em Netscape.

### **handleEvent()**

Invoca o manipulador de eventos do elemento especificado.

### **open()**

Abre o fluxo do documento.

### **releaseEvents()**

Libera os eventos capturados do tipo que se especifique, enviando-os aos objetos seguintes na hierarquia.

### **routeEvent()**

Passa um evento capturado através da hierarquia de eventos habitual.

### **write()**

Escreve dentro da página web. Podemos escrever etiquetas HTML e texto normal.

### **writeln()**

Escreve igual que o método write(), embora coloque uma quebra de linha no final.

Os eventos de document servem principalmente para controlar duas coisas. Um grupo nos oferece uma série de funções para o controle dos documentos, como escrever, abrir ou fechar. Veremos no próximo capítulo que fala sobre o controle do fluxo de escritura do documento. O outro grupo oferece ferramentas para o controle dos eventos no documento e o veremos mais

adiante quando tratarmos com detalhe o tema de eventos.

O método `getSelection()` fica para nós um pouco solto, pois só funciona nos navegadores de Netscape e portanto, não é muito útil para as aplicações que desejarmos que sejam compatíveis em todos os sistemas. Mesmo assim, faremos um exemplo sobre este método, já que os outros serão vistos nos próximos capítulos.

O exemplo consiste em uma página que tem um pouco de texto e um botão. Na página poderemos selecionar algo de texto e logo, apertar o botão que chama a uma função que mostra em uma caixa alert o texto que foi selecionado. O código é o seguinte:

```
<html>
<head>
<title>Resgatar o selecionado</title>
<script language="JavaScript">
function mostrarSelecionado(){
    alert("Foi selecionado:\n" + document.getSelection())
}
</script>
</head>

<body>
<h1>Resgatar o selecionado</h1>
<br>

<form>
<input type="button" value="clica-me!" onClick="mostrarSelecionado()">
</form>
```

Selecione qualquer texto da página e clique o botão.

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

Pode-se [ver em funcionamento o script em uma página a parte](#), embora somente funcionará em Netscape e Internet Explorer dará um error.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Fluxo de escritura do documento

Sobre o objeto `document` também é interessante falar um pouco sobre o fluxo de escritura do documento ou página web.

Quando o navegador lê uma página web vai interpretando e escrevendo em sua janela. O processo no qual o navegador está escrevendo a página lhe chamamos fluxo de escritura do documento. O fluxo começa quando a página começa a ser escrita e dura até que esta termine de ser escrita. Uma vez terminada a escritura da página, o fluxo de escritura do documento se fecha automaticamente. Com isto termina a recarga da página.

Uma vez fechado o fluxo, não se pode mais escrever na página web, nem texto nem imagens nem outros elementos.

Em javascript, temos que respeitar o fluxo de escritura do documento forçosamente. É por isso que só podemos executar sentenças `document.write()` (método `write()` do objeto `document`) quando está aberto o fluxo de escritura do documento, ou o que é igual, enquanto a página está sendo recarregada.

Lembramos as duas formas de executar um script em Javascript:

- Execução dos scripts enquanto a página está sendo recarregada. Aqui poderemos executar `document.write()` e fizemos habitualmente nos exemplos anteriores.
- Execução dos scripts quando a página foi recarregada, como resposta a um evento do usuário. Aqui, não podemos fazê-lo porque a página terminou de recarregar, de fato, não o fizemos nunca até agora.

Há o ponto em que não se pode escrever na página quando já está fechado o fluxo. Na verdade sim que pode, mas necessitamos abrir o fluxo outra vez para escrever na página, tanto é assim que embora nós não abramos explicitamente Javascript se encarregará disso. O que tem que ficar claro é que se fazemos um `document.write()`, o fluxo tem que estar aberto e se não estiver se abrirá. O problema de abrir o fluxo de escritura do documento, uma vez já está escrita a página, é que todo o conteúdo é apagado.

Para que fique claro vamos fazer um script para escrever na página uma vez esta tenha sido recarregada. Simplesmente necessitamos um botão e ao clicá-lo executar o método `write()` do objeto `document`.

```
<form>
<INPUT type=button value=escribir onclick="window.document.write('olá')">
</form>
```

Se observarmos, no manipulador de eventos `onclick`, colocamos a hierarquia de objetos desde o princípio, ou seja, começando pelo objeto `window`. Isto é porque existem alguns navegadores que não subentendem o objeto `window` nas mesmas sentenças escritas nos manipuladores de eventos.

Podemos [ver o exemplo em funcionamento](#).

O resultado da execução pode variar de um navegador a outro, mas de qualquer forma se apagará a página que se está executando.

### Métodos `open()` e `close()` de `document`

Os métodos `open()` e `close()` do objeto `document` servem para controlar o fluxo do documento desde Javascript. Permite-nos abrir e fechar o documento explicitamente.

O exemplo de escritura na página anterior deveria haver sido escrito com sua correspondente abertura e fechamento do documento e teria ficado algo parecido a isto.

```
<script>
function escreve(){
    document.open()
    window.document.write('Olá')
```

```
document.close()  
}  
</script>  
<form>  
<INPUT type=button value=escrever onclick="escreve()">  
</form>
```

Vemos que agora não escrevemos as sentenças dentro do manipulador, porque, quando há mais de uma sentença, fica mais claro colocar uma chamada a uma função e na função colocamos as sentenças que quisermos.

As sentenças do exemplo anterior, que cobrem a abertura, escritura e fechamento do documento. Podem [ser vistas aqui](#).

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Trabalho com formulários em Javascript

Para continuar vamos ver uma série de capítulos enfocados em aprender a trabalhar com os formulários, acessar aos seus elementos para controlar seus valores e atualizá-los.

O objeto form depende na hierarquia do objeto document. Em um objeto form podemos encontrar alguns métodos e propriedades, mas o mais destacado que poderemos encontrar é cada uno dos elementos do formulário. Ou seja, de um formulário dependem todos os elementos que existem dentro, como podem ser campos de texto, caixas de seleção, áreas de texto, botões de radio, etc.

Para acessar a um formulário pelo objeto document podemos fazer-lo de duas formas.

1. A partir de seu nome, atribuído com o atributo NAME de HTML.
2. Mediante a matriz de formulários do objeto document, com o índice do formulário ao que queremos acessar.

Para este formulário

```
<FORM name="f1">  
<INPUT type=text name=campo1>  
<INPUT type=text name=campo2>  
</FORM>
```

Poderemos acessar com seu nome desta maneira.

```
document.f1
```

Ou com seu índice, se supormos que é o primeiro da página.

```
document.forms[0]
```

De maneira similar acessamos aos elementos de um formulário, que dependem do objeto form.

1. A partir do nome do objeto atribuído com o atributo NAME de HTML.
2. Mediante a matriz de elementos do objeto form, com o índice do elemento ao que queremos acessar.

Poderíamos acessar ao campo 1 do anterior formulário de duas maneiras. Com seu nome faríamos assim.

```
document.f1.campo1
```

ou a partir do array de elementos.

`document.f1.elements[0]` (utilizamos o índice 0 porque é o primeiro elemento e em Javascript os arrays começam por 0.)

Se desejarmos acessar ao segundo campo do formulário, escreveríamos uma destas duas coisas:

```
document.f1.campo2  
document.f1.elements[1]
```

Lembramos que também podemos acessar a um formulário pelo array de forms, indicando o índice do formulário ao qual acessar. Deste modo, o acesso ao campo2 seria o seguinte:

```
document.forms[0].campo2  
document.forms[0].elements[1]
```

Nestes casos, supomos que este formulário é o primeiro que está escrito no código HTML, por isso lhe acessamos com o índice 0.

Esperamos que tenha ficado claro o acesso a formulários e a seus campos. Passaremos então, a um exemplo para praticar tudo isto.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Ex. de trabalho com formulários. Calculadora simples

Para ilustrar um pouco o trabalho com formulários, vamos realizar um exemplo prático. Pode ser que algumas coisas das quais vamos tratar, fiquem um pouco no ar por não terem sido explicadas detalhadamente antes, mas certamente nos serve para estarmos por dentro de como se trabalha com formulários e as possibilidades que temos.

### Exemplo de calculadora simples

Neste exemplo vamos construir uma calculadora, embora bastante simples, que permita realizar as operações básicas. Para fazer a calculadora vamos realizar um formulário no qual vamos colocar três campos de texto, os dois primeiros para as parcelas e um terceiro para o resultado. Ademais haverá uns botões para fazer as operações básicas.



## O formulário da calculadora pode ser visto aqui.

```
<form name="calc">
<input type="Text" name="parcela1" value="0" size="12">
<br>
<input type="Text" name="parcela2" value="0" size="12">
<br>
<input type="Button" name="" value=" + " onclick="calcula('+')">
<input type="Button" name="" value=" - " onclick="calcula('-')">
<input type="Button" name="" value=" X " onclick="calcula('*')">
<input type="Button" name="" value=" / " onclick="calcula('/')">
<br>
<input type="Text" name="resultado" value="0" size="12">
</form>
```

Mediante uma função vamos acessar aos campos do formulário para buscar as parcelas em duas variáveis. Os campos de texto têm uma propriedade chamada `value` que é onde podemos obter o que têm escrito nesse momento. Mais tarde recorreremos a [função `eval\(\)`](#) para realizar a operação. Colocaremos por último o resultado no campo de texto criado em terceiro lugar, utilizando também a propriedade `value` do campo de texto.

Chamamos à função que realiza o cálculo (que podemos ver a seguir) apertando os botões de cada uma das operações. Tais botões podem ser vistos no formulário e contém um atributo `onclick` que serve para especificar as sentenças Javascript que desejamos que se executem quando o usuário clicar sobre ele. Neste caso, a sentença a executar é uma chamada à função `calcula()` passando como parâmetro o símbolo ou a parcela da operação que desejamos realizar.

## O script com a função `calcula()`

```
<script>
function calcula(operacao){
    var parcela1 = document.calc.parcela1.value
    var parcela2 = document.calc.parcela2.value
    var result = eval(parcela1 + operacao + parcela2)
    document.calc.resultado.value = result
}
</script>
```

A [função `eval\(\)`](#), lembramos, que recebia um string e o executava como uma sentença Javascript. Neste caso, irá receber um número que concatenado a uma operação e outro número será sempre uma expressão aritmética que `eval()` solucionará perfeitamente.

Podemos [ver o exemplo da calculadora em funcionamento](#).

O acesso a outros elementos dos formulários se faz de maneira parecida a respeito da hierarquia de objetos, entretanto, como cada elemento tem suas particularidades as coisas que poderemos fazer com eles diferirão um pouco. Isto será visto mais adiante.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Propriedades e métodos do objeto form

Vamos ver agora o objeto form por si só, para destacar suas propriedades e métodos.

### Propriedades do objeto form

Têm umas propriedades para ajustar seus atributos mediante Javascript.

#### **action**

É a ação que queremos realizar quando se submete um formulário. Coloca-se geralmente um endereço de correio ou a URL a qual lhe mandaremos os dados. Corresponde com o atributo ACTION do formulário.

#### **elements array**

A matriz de elementos contém cada um dos campos do formulário.

#### **encoding**

O tipo de codificação do formulário

#### **length**

O número de campos do formulário.

#### **method**

O método pelo qual mandamos a informação. Corresponde com o atributo METHOD do formulário.

#### **name**

O nome do formulário, que corresponde com o atributo NAME do formulário.

#### **target**

A janela ou frame no qual está dirigido o formulário. Quando se submete se atualizará a janela ou frame indicado. Corresponde com o atributo target do formulário.

### Exemplos de trabalho com as propriedades

Com estas propriedades podemos mudar dinamicamente com Javascript os valores dos atributos do formulário para fazer com ele o que se deseje dependendo das exigências do momento.

Por exemplo, poderíamos mudar a URL que receberia a informação do formulário com a instrução.

```
document.meuFormulário.action = "minhaPágina.asp"
```

Ou mudar o target para submeter um formulário em uma possível janela secundária chamada minha\_janela.

```
document.meuFormulário.target = "minha_janela"
```

## Métodos do objeto form

Estes são os métodos que podemos invocar com um formulário.

### submit()

Para fazer com que o formulário se submeta, embora não se tenha clicado o botão de submit.

### reset()

Para reiniciar todos os campos do formulário, como se tivéssemos clicado o botão de reset.  
(Javascript 1.1)

## Exemplo de trabalho com os métodos

Vamos ver um exemplo muito simples sobre como validar um formulário para submete-lo no caso de que esteja preenchido. Para isso, vamos utilizar o método submit() do formulário.

O mecanismo é o seguinte: em vez de colocar um botão de submit colocamos um botão normal (<INPUT type="button">) e fazemos que ao clicar esse botão se chame a uma função que é a encarregada de validar o formulário e, no caso de que esteja correto, submete-lo.

O formulário ficaria assim:

```
<form name="meuFormulário" action="mailto:colabore@criarweb.com" enctype="text/plain">
<input type="Text" name="campo1" value="" size="12">
<input type="button" value="Enviar" onclick="validaSubmete()">
</form>
```

Observamos que não há um botão de submit, e sim, um botão normal com uma chamada a uma função que podemos ver a seguir.

```
function validaSubmete(){
    if (document.meuFormulário.campo1.value == "")
        alert("Deve preencher o formulário")
    else
        document.meuFormulário.submit()
}
```

Na função se comprova que se o que está escrito no formulário é um string vazio. Se for isso, mostra-se uma mensagem de alerta que informa que se deve preencher o formulário. No caso de haver algo no campo de texto submete o formulário utilizando o método submit do objeto form.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Controle de campos de texto com Javascript

Vamos ver agora os campos onde podemos salvar cadeias de texto, ou seja, os campos de texto, password e hidden. Existe outro campo relacionado com a escritura de texto, o campo TextArea, que veremos mais adiante.

## Campo Text

É o campo que obtemos ao escrever a etiqueta `<INPUT type="text">`. Utilizamos até este momento em vários exemplos, mas vamos parar um momento nele para descrever suas propriedades e métodos.

### Propriedades do campo text

Vemos a lista de propriedades destes tipos de campo.

#### **defaultValue**

É o valor por padrão que tem um campo. O que contém o atributo VALUE da etiqueta `<INPUT>`.

#### **form**

Faz referência ao formulário.

#### **name**

Contem o nome deste campo de formulário

#### **type**

Contem o tipo de campo de formulário que é.

#### **value**

O texto que está escrito no campo.

Vamos ver um exemplo sobre o que pode fazer a propriedade `defaultValue`. Neste exemplo, temos um formulário e um botão de reset. Se clicarmos o botão de reset, o campo de texto se esvazia porque seu `value` de HTML era um string vazio. Mas se clicarmos o botão seguinte, chamamos a função que muda o valor por padrão desse campo de texto, de modo que ao clicar o botão de reset mostrará o novo valor por padrão.

Este é o código da página completa.

```
<html>
<head>
  <title>Mudar o valor por padrao</title>
  <script>
    function mudaPadrao(){
      document.meuFormulario.campo1.defaultValue = "Olá!!"
    }
  </script>
</head>

<body>
<form name="meuFormulario" action="mailto:colabore@criarweb.com" enctype="text/plain">
<input type="Text" name="campo1" value="" size="12">
<input type="Reset">
<br>
<br>
<input type="button" value="Muda valor por padrao" onclick="mudaPadrao()">
</form>
</body>
</html>
```

## Métodos do objeto Text

Pode-se invocar os seguintes métodos sobre os objetos tipo Text.

**blur()**

Retira o foco da aplicação do campo de texto.

**focus()**

Coloca o foco da aplicação no campo de texto.

**select()**

Seleciona o texto do campo.

Como exemplo vamos mostrar uma função que seleciona o texto de um campo de texto de um formulário como o da página do exemplo anterior. Para fazê-lo utilizamos dois métodos, o primeiro para passar o foco da aplicação ao campo de texto e o segundo para selecionar o texto.

```
function selecionaFoco(){  
    document.meuFormulario.campo1.focus()  
    document.meuFormulario.campo1.select()  
}
```

Pode-se [ver em funcionamento nesta página](#).

**Campos Password**

Estes funcionam como os hidden, com a peculiaridade que o conteúdo do campo não pode se ver escrito no campo, por isso saem asteriscos no lugar do texto.

**Campos Hidden**

Os campos hidden são como campos de texto que estão ocultos para o usuário, ou seja, que não se vêem na página. São muito úteis no desenvolvimento de webs para passar variáveis nos formulários aos quais o usuário não deve ter acesso.

Colocam-se com HTML com a etiqueta <INPUT type=hidden> e se preenchem os dados com seu atributo value. Mais tarde poderemos mudar o dado que figura no campo acessando à propriedade value do campo de texto assim como fizemos antes.

```
document.meuFormulario.CampoHidden.value = "novo texto"
```

O campo hidden tem somente algumas das propriedades dos campos de texto. Teoricamente, tem a propriedade value e as propriedades que são comuns de todos os campos de formulário: name, from e type, que já foram descritos para os campos de texto.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Controle de checkbox em javascript

Os checkbox são umas caixas que permitem marca-las ou não para verificar alguma coisa em um formulário. Podemos ver uma caixa checkbox a seguir:



Os checkbox se conseguem com a etiqueta <INPUT type=checkbox>. Com o atributo NAME da etiqueta <INPUT> podemos lhe dar um nome para logo acessa-la com javascript. Com o atributo CHECKED indicamos que o campo deve aparecer checado por padrão.

Com Javascript, a partir da hierarquia de objetos do navegador, temos acesso ao checkbox, que depende do objeto form do formulário onde está incluído.

### Propriedades de um checkbox

As propriedades que têm um checkbox são as seguintes:

#### **checked**

Informa sobre o estado do checkbox. Pode ser true ou false.

#### **defaultChecked**

Se está checada por padrão ou não.

#### **value**

O valor atual do checkbox.

Também têm as propriedades form, name, type como qualquer outro elemento de formulário.

### Métodos do checkbox

Vejam os a lista dos métodos de um checkbox.

#### **click()**

É como se clicássemos sobre o checkbox, ou seja, muda o estado do checkbox.

#### **blur()**

Retira o foco da aplicação do checkbox.

#### **focus()**

Coloca o foco da aplicação no checkbox.

Para ilustrar o funcionamento das checkbox vamos ver uma página que tem um checkbox e três botões. Os dois primeiros para mostrar as propriedades checked e value e o terceiro para invocar a seu método click() com objetivo de simular um clique sobre o checkbox.

```
<html>
<head>
  <title>Exemplo Checkbox</title>
</head>
<script>
function alertaChecked(){
  alert(document.meuFormulario.meuCheck.checked)
}
function alertaValue(){
  alert(document.meuFormulario.meuCheck.value)
}
function metodoClick(){
  document.meuFormulario.meuCheck.click()
}
</script>
</html>
```

```
<body>
<form name="meuFormulario" action="mailto:colabore@criarweb.com" enctype="text/plain">
<input type="checkbox" name="meuCheck">
<br>
<br>
<input type="button" value="informa de sua propriedade checked" onclick="alertaChecked()">
<input type="button" value="informa de sua propriedade value" onclick="alertaValue()">
<br>
<br>
<input type="button" value="Simula um clique" onclick="metodoClick()">
</form>
</body>
</html>
```

Pode-se [ver a página em funcionamento aqui](#).

Artigo por **Miguel Angel Alvarez** - Tradução de JML

## Controle de botões de radio em Javascript

O botão de radio (ou radio button em inglês) é um elemento de formulário que permite selecionar uma opção e somente uma, sobre um conjunto de possibilidades. Pode-se ver a seguir:

- ☐ Branco
- ☐ Vermelho
- ☐ Verde

**Nota:** Na parte acima podemos ver três radio buttons ao invés de um só. Colocam-se três botões porque assim podemos examinar seu funcionamento ao formar parte de um grupo. Vejamos que ao selecionar uma opção se desmarca a opção que estiver marcada antes.

Consegue-se com a etiqueta `<INPUT type=radio>`. Com o atributo NAME da etiqueta `<INPUT>` damos um nome para agrupar os radio button e que somente se possa escolher uma opção entre várias. Com o atributo value indicamos o valor de cada um dos radio buttons. O atributo checked nos serve para indicar qual dos radio buttons tem que estar selecionado por padrão.

**Referencia:** Explicamos com detalhe a criação de botões de radio em nosso manual de HTML, no capítulo [Outros elementos de formulários](#).

Quando em uma página temos um conjunto de botões de radio cria-se um objeto radio por cada um dos botões. Os objetos radio dependem do formulário e pode-se acessá-los pelo array de elements, entretanto também cria-se um array com os botões de radio. Este array depende do formulário e tem o mesmo nome que os botões de radio.

### Propriedades do objeto radio

Vejamos uma lista das propriedades deste elemento.

#### **checked**

Indica se está checado ou não um botão de radio.

#### **defaultChecked**

Seu estado padrão.

**value**

O valor do campo de radio, atribuído pela propriedade value do radio.

**Length** (como propriedade do array de radios)

O número de botões de radio que formam parte no grupo. Acessível no vetor de radios.

**Métodos do objeto radio**

São os mesmos que tinha o [objeto checkbox](#).

**Exemplo de utilização**

Vejamos com um exemplo o método de trabalho com os radio buttons no qual vamos colocar vários deles e cada um terá associado uma cor. Também haverá um botão e ao clicá-lo mudaremos a cor de fundo da tela a cor que esteja selecionada no conjunto de botões de radio.

Vamos ver a página inteira e em seguida comentamos.

```
<html>
<head>
  <title>Exemplo Radio Button</title>
</script>
function mudaCor(){
  var i
  for (i=0;i<document.fcores.colorin.length;i++){
    if (document.fcores.colorin[i].checked)
      break;
  }
  document.bgColor = document.fcores.colorin[i].value
}
</script>
</head>

<body>
<form name=fcores>
<input type="Radio" name="colorin" value="ffffff" checked> Branco
<br>
<input type="Radio" name="colorin" value="ff0000"> Vermelho
<br>
<input type="Radio" name="colorin" value="00ff00"> Verde
<br>
<input type="Radio" name="colorin" value="0000ff"> Azul
<br>
<input type="Radio" name="colorin" value="ffff00"> Amarelo
<br>
<input type="Radio" name="colorin" value="ff00ff"> Lilás
<br>
<input type="Radio" name="colorin" value="000000"> Preto
<br>
<br>
<input type="Button" name="" value="Muda Cor" onclick="mudaCor()">
</form>
</body>
</html>
```

Primeiro, podemos observar no formulário e na lista de botões de radio. Todos se chamam "colorin", portanto estão associados em um mesmo grupo. Além disso, vemos que o atributo



value de cada botão muda. Também vemos um botão abaixo de tudo.

Com esta estrutura de formulário teremos um array de elements de 9 elementos, os 8 botões de radio e o botão debaixo.

Ademais, teremos um array de botões de radio que se chamará colorin e depende do formulário, acessível desta maneira:

```
document.form.colorin
```

Este array tem em cada posição um dos botões de radio. Assim, na posição 0 está o botão de cor branca, na posição 1 o de cor vermelha... Para acessar a estes botões de radio fazemos pelo seu índice.

```
document.fcores.colorin[0]
```

Se quisermos acessar por exemplo à propriedade value do último botão de radio escrevemos o seguinte:

```
document.fcores.colorin[7].value
```

A propriedade length do array de radios nos indica o número de botões de radio que fazem parte do grupo.

```
document.fcores.colorin.length
```

Neste caso, a propriedade length valerá 8.

Com estas notas poderemos entender mais ou menos bem a função que se encarrega de encontrar o radio button selecionado e mudar a cor de fundo da página.

Define-se uma variável na qual introduziremos o índice do radio button que temos selecionado. Para isso, vamos percorrendo o array de botões de radio até encontrarmos o que tem sua propriedade checked a true. Nesse momento, saímos do loop, com o qual a variável i armazena o índice do botão de radio selecionado. Na última linha mudamos a cor de fundo pelo que estiver selecionado no atributo value do radio button.

Podemos [ver esse exemplo em funcionamento](#).

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Controle de campos select com Javascript

O objeto select de um formulário é uma dessas listas dinâmicas que nos permitem selecionar um elemento. Desdobram-se apertando sobre uma seta, a seguir pode-se escolher um elemento e para acabar voltam a se dobrarem. Pode-se ver um elemento select de um formulário a seguir.

Um destes elementos pode ser obtido utilizando a etiqueta `<SELECT>` dentro de um formulário. Podemos acrescentar à etiqueta um atributo para lhe dar o nome, `NAME`, para logo acessar a esse campo mediante Javascript. Para expressar cada uma das possíveis opções do campo select utilizamos uma etiqueta `<OPTION>` a qual lhe introduzimos um atributo `VALUE` para expressar seu valor. O texto que colocamos depois da etiqueta `<OPTION>` serve como etiqueta dessa opção, é o texto que vê o usuário associado a essa opção.

### Propriedades do objeto select

Vamos ver uma lista das propriedades deste elemento de formulário.

#### **length**

Salva a quantidade de opções do campo select. Quantidade de etiquetas `<OPTION>`

#### **Option**

Faz referência a cada uma das suas opções. São por si mesmas objetos.

#### **options**

Um array com cada uma das opções do select.

#### **selectedIndex**

É o índice da opção que se encontra selecionada.

A parte das conhecidas propriedades comuns a todos os elementos de formulário: `form` e `name` e `type`.

### Métodos do objeto select

Os métodos são somente 2 e já conhecemos seu significado.

#### **blur()**

Para retirar o foco da aplicação desse elemento de formulário.

#### **focus()**

Para colocar o foco da aplicação.

### Objeto option

Temos que pararmos para ver também este objeto para entender melhor o campo select. Lembramos que as option são as distintas opções que tem um select, expressadas com a etiqueta `<OPTION>`.

### Propriedades de option

Estes objetos só têm propriedades, não têm métodos. Vamos vê-las.

#### **defaultSelected**

Indica com um `true` ou um `false` se essa opção é a padrão. A opção padrão se consegue com HTML colocando o atributo `selected` a um option.

#### **index**

O índice dessa opção dentro do select.

**selected**

Indica se essa opção se encontra selecionada ou não.

**text**

É o texto da opção. O que o usuário pode ver no select, que se escreve depois da etiqueta <OPTION>.

**value**

Indica o valor da opção, que se introduz com o atributo VALUE da etiqueta <OPTION>.

**Exemplo de acesso a um select**

Vamos ver um exemplo sobre como se acessa a um select com Javascript, como podemos acessar suas diferentes propriedades e à opção selecionada.

Vamos começar vendo o formulário que tem o select com o qual vamos trabalhar. É um select que serve para valorizar o web que estamos visitando.

```
<form name="fomul">
Valoracao sobre este web:
<select name="minhaSelect">
<option value="10">Muito bom
<option value="5" selected>Regular
<option value="0">Muito ruim
</select>
<br>
<br>
<input type="button" value="Digame propriedades" onclick="digamePropriedades()">
</form>
```

Agora vamos ver uma função que percorre as propriedades mais significativas do campo select e as apresenta em uma caixa alert.

```
function digamePropriedades(){
    var texto
    texto = "O número de opções do select: " + document.formul.minhaSelect.length
    var indice = document.formul.minhaSelect.selectedIndex
    texto += "\nÍndice da opção escolhida: " + indice
    var valor = document.formul.minhaSelect.options[indice].value
    texto += "\nValor da opção escolhida: " + valor
    var textoEscolhido = document.formul.minhaSelect.options[indice].text
    texto += "\nTexto da opção escolhida: " + textoEscolhido
    alert(texto)
}
```

Esta função cria uma variável de texto onde vai introduzindo cada uma das propriedades do select. A primeira, contém o valor da propriedade length do select, a segunda, o índice da opção selecionada e as duas seguintes, contém o valor e o texto da opção selecionada. Para acessar à opção selecionada utilizamos o array options com o índice percorrido na segunda variável.

Podemos [ver o exemplo em funcionamento aqui](#).

## Propriedade value de um objeto select

Para acessar ao valor selecionado de um campo select podemos utilizar a propriedade value do campo select no lugar de acessar a partir do vetor de options.

Para o anterior formulário seria algo parecido a isto.

```
formul.minhaSelect.value
```

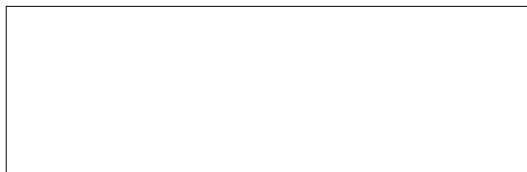
Entretanto, esta propriedade só está presente em navegadores Internet Explorer, portanto é recomendável acessar utilizando o vetor de options com o índice da posição selecionada se desejarmos que a página seja compatível com todos os navegadores. Quisemos acrescentar este ponto para que, se alguma vez utilizarmos ou virmos utilizar este método de acesso, sabemos seu problema e porque é melhor utilizar o vetor de options.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Controle de elementos Textarea em Javascript

Para acabar de descrever todos os elementos de formulários vamos ver o objeto textarea que é um elemento que apresenta um lugar para escrever texto, igual que os campos text, mas com a particularidade que podemos escrever várias linhas de uma só vez.

O campo textarea pode ser visto a seguir.



Um campo textarea se consegue com a etiqueta <TEXTAREA>. Com o atributo name podemos dar um nome para acessar ao campo textarea mediante Javascript. Outros atributos interessantes são cols e rows que servem para indicar a largura e a altura do campo textarea em caracteres, cols indica o número de colunas e rows o de filas. Embora não se possa acessá-los com Javascript. O valor padrão de um campo textarea se coloca entre as etiquetas <TEXTAREA> e seu correspondente fechamento.

### Propriedades de textarea

Pode-se ver uma lista das propriedades disponíveis em um textarea a seguir, que são os mesmos que um campo de texto.

#### defaultValue

Que contém o valor padrão do textarea.

#### value

Que contém o texto que está escrito no textarea.

Além disso, têm as conhecidas propriedades de elementos de formulário form, name e type.

## Métodos de textarea

Vejam os métodos, que são os mesmos que em um campo de texto.

### **blur()**

Para tirar o foco da aplicação do textarea.

### **focus()**

Para colocar o foco da aplicação no textarea.

### **select()**

Seleciona o texto do textarea.

Vamos ver um exemplo a seguir que apresenta um formulário que tem um textarea e um botão. Ao apertar o botão conta-se o número de caracteres e coloca-se em um campo de texto.

Para acessar ao número de caracteres fazemos a partir da propriedade value do objeto textarea. Como value contém um string podemos acessar à propriedade length que têm todos os strings para averiguar sua longitude.

O código da página pode ser visto aqui.

```
<html>
<head>
  <title>Exemplo textarea</title>
</script>
function conta(){
  numCaracteres = document.formul.texto.value.length
  document.formul.numCaracteres.value = numCaracteres
}
</script>
</head>
<body>
<form name="formul">
<textarea name=texto cols=40 rows=3>
Este é o texto padrão
</textarea>
<br>
<br>
Número de caracteres <input type="Text" name="numCaracteres" size="4">
<br>
<br>
<input type=button value="Conta caracteres" onclick="conta()">
</form>
</body>
</html>
```

O exemplo funcionando pode [ser visto em uma página independente](#).

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Os eventos em Javascript

Os eventos são a maneira que temos em Javascript de controlar as ações dos visitantes e definir um comportamento da página quando se produzam. Quando um usuário visita uma página web e interage com ela se produzem os eventos e com Javascript podemos definir o que queremos que ocorra quando se produzam.

Com Javascript podemos definir o que acontece quando se produz um evento como poderia ser que um usuário clique sobre um botão, edite um campo de texto ou abandone a página.

O manejo de eventos é o cavalo de batalha para fazer páginas interativas, porque com eles podemos responder às ações dos usuários. Até agora neste manual podemos ver muitos exemplos de manejo de um dos eventos de Javascript, o evento onclick, que se produz ao clicar um elemento da página. Até agora aplicamos sempre o evento a um botão, mas poderíamos aplica-lo a outros elementos da página.

### Como se define um evento

Para definir as ações que queremos realizar ao produzir um evento utilizamos os manipuladores de eventos. Existem muitos tipos de manipuladores de eventos, para muitos tipos de ações do usuário. O manipulador de eventos se coloca na etiqueta HTML do elemento da página que queremos que responda às ações do usuário.

Por exemplo, temos o manipulador de eventos onclick, que serve para descrever ações que queremos que se executem quando se clique. Se quisermos que ao clicar sobre um botão aconteça alguma coisa, escrevemos o manipulador onclick na etiqueta <INPUT type=button> desse botão. Algo parecido a isto.

```
<INPUT type=button value="clica-me" onclick="sentencas_javascript...">
```

Coloca-se um atributo novo na etiqueta que tem o mesmo nome que o evento, neste caso onclick. O atributo se iguala às sentenças Javascript que queremos que se executem ao se produzir o evento.

Cada elemento da página tem sua própria lista de eventos suportados, vamos ver outro exemplo de manejo de eventos, desta vez sobre um menu desdobrável, no qual definimos um comportamento quando mudamos o valor selecionado.

```
<SELECT onchange="window.alert('Mudou a seleção')">  
<OPTION value="opcao1">Opcao 1  
<OPTION value="opcao2">Opcao 2  
</SELECT>
```

Nets exemplo, cada vez que se muda de opção mostra uma caixa de alerta. Podemos [vê-lo em uma página a parte](#).

Dentro dos manipuladores de eventos podemos colocar tantas instruções quantas desejarmos, mas sempre separadas por ponto e vírgula. O habitual é colocar uma só instrução, e se se deseja colocar mais de uma, costuma-se criar uma função com todas as instruções e dentro do manipulador se coloca uma só instrução que é chamada à função.

Vamos ver como se colocariam em um manipulador várias instruções.

```
<input type=button value=Clicame  
onclick="x=30; window.alert(x); window.document.bgColor = 'red'">
```

São instruções muito simples como atribuir a x o valor 30, fazer uma janela de alerta com o valor de x e mudar a cor do fundo a vermelho. Podemos [ver o exemplo em uma página a parte](#).

Entretanto, tantas instruções postas em um manipulador ficam um pouco confusas, seria melhor criar uma função assim:

```
<script>  
function executaEventoOnClick(){  
    x = 30  
    window.alert(x)  
    window.document.bgColor = 'red'  
}  
</script>  
  
<FORM>  
<input type=button value=Pulsame onclick="executaEventoOnClick()">  
</FORM>
```

Agora utilizamos mais texto para fazer o mesmo, mas com certeza deve lhe parecer mais claro este segundo exemplo. Se se deseja, pode-se [ver esta última página em uma janela a parte](#)

### Hierarquia pelo objeto window

Nos manipuladores de eventos tem que se especificar a hierarquia inteira de objetos do navegador, começando sempre pelo objeto window. Isto é necessário porque existe algum browser antigo que não subentende o objeto window quando se escrevem sentenças Javascript vinculadas a manipuladores de eventos.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Os manipuladores de eventos em Javascript

Agora vamos ver uma lista dos manipuladores de eventos que existem disponíveis em Javascript, oferecendo uma pequena descrição de cada um.

**Nota:** Estes manipuladores são os mais comuns, presentes em Javascript 1.2 de Netscape Navigator. Existem outros manipuladores que também são muito interessantes e veremos mais adiante capítulos de temas avançados de eventos.

A lista de manipuladores de eventos contém o nome do manipulador em negrito, sua descrição e finalmente a versão de Javascript que incorporou tal manipulador.

### **onabort**

Este evento se produz quando um usuário detém a carga de uma imagem, seja porque detém a carga da página ou porque realiza uma ação que a detém, como por exemplo, sair da página.

Javascript 1.1

### **onblur**

Desata-se um evento onblur quando um elemento perde o foco da aplicação. O foco da

aplicação é o lugar onde está situado o cursor, por exemplo, pode estar situado sobre um campo de texto, uma página, um botão ou qualquer outro elemento.  
Javascript 1.0

**onchange**

Desata-se este evento quando muda o estado de um elemento de formulário, às vezes não se produz até que o usuário retire o foco da aplicação do elemento. Javascript 1.0  
Javascript 1.0

**onclick**

Produz-se quando se clica o botão do mouse sobre um elemento da página, geralmente um botão ou um link.  
Javascript 1.0

**ondragdrop**

Produz-se quando um usuário solta algo que havia arrastado sobre a página web.  
Javascript 1.2

**onerror**

Produz-se quando não se pode carregar um documento ou uma imagem e esta fica quebrada.  
Javascript 1.1

**onfocus**

O evento onfocus é o contrário de onblur. Produz-se quando um elemento da página ou a janela ganham o foco da aplicação.  
Javascript 1.0

**onkeydown**

Este evento é produzido no instante que um usuário pressiona uma tecla, independentemente que a solte ou não. É produzido no momento do clique.  
Javascript 1.2

**onkeypress**

Ocorre um evento onkeypress quando o usuário deixa uma tecla clicada por um tempo determinado. Antes deste evento se produz um onkeydown no momento que se clica a tecla..  
Javascript 1.2

**onkeyup**

Produz-se quando o usuário deixa de apertar uma tecla. É produzido no momento que se libera a tecla.  
Javascript 1.2

**onload**

Este evento se desata quando a página, ou em Javascript 1.1 as imagens, terminaram de se carregar.  
Javascript 1.0

**onmousedown**

Produz-se o evento onmousedown quando o usuário clica sobre um elemento da página. onmousedown se produz no momento de clicar o botão, soltando ou não.  
Javascript 1.2



**onmousemove**

Produz-se quando o mouse se move pela página.  
Javascript 1.2

**onmouseout**

Desata-se um evento onmuoseout quando a seta do mouse sai da área ocupada por um elemento da página.  
Javascript 1.1

**onmouseover**

Este evento desata-se quando a seta do mouse entra na área ocupada por um elemento da página.  
Javascript 1.0

**onmouseup**

Este evento se produz no momento que o usuário solta o botão do mouse, que previamente havia clicado.  
Javascript 1.2

**onmove**

Evento que se executa quando se move a janela do navegador, ou um frame.  
Javascript 1.2

**onresize**

Evento que se produz quando se redimensiona a janela do navegador, ou o frame, no caso de que a página os tenha.  
Javascript 1.2

**onreset**

Este evento está associado aos formulários e se desata no momento que um usuário clica no botão de reset de um formulário.  
Javascript 1.1

**onselect**

Executa-se quando um usuário realiza uma seleção de um elemento de um formulário.  
Javascript 1.0

**onsubmit**

Ocorre quando o visitante aperta sobre o botão de enviar o formulário. Executa-se antes do envio propriamente dito.  
Javascript 1.0

**onunload**

Ao abandonar uma página, seja porque se clique em um link que nos leva a outra página ou porque se fecha a janela do navegador, se executa o evento onunload.  
Javascript 1.0

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Exemplos de eventos em Javascript. Onabort

Ao longo dos [manuais I](#) e [II](#) de Javascript, mostramos muitos exemplos de utilização dos manipuladores de eventos. Aqui, veremos exemplos simples que nos ocorrem para utilizar outros manipuladores que ainda não vimos.

### Evento onabort

Vejamos um primeiro exemplo, neste caso sobre o evento onabort. Este evento se ativa ao se cancelar o carregamento de uma página, seja porque se clica o botão de cancelar ou porque o usuário sai da página por outro link.

Este exemplo contém uma imagem que tem o evento onabort atribuído para que se execute uma função no caso de que a imagem não chegue a se carregar. A função informa ao usuário de que a imagem não chegou a se carregar e lhe pergunta deseja carrega-la outra vez. Se o usuário responde que sim, então começa a baixar a imagem outra vez. Se responde que não, não faz nada. A pergunta é feita com uma caixa confirm de Javascript.

```
<html> <head>
  <title>Evento onabort</title>

  <script>
function perguntarSeguir(){
  resposta = confirm ("O carregamento da página foi detido e há uma imagem que você não está vendo.\nDeseja
carregar a imagem?")
  if (resposta)
    document.img1.src = "http://criarweb.com/artigos/imagens/desarrollogrande.gif"
}
</script>

</head>
<body>

<br>
Clique o botão de parar o carregamento da página e se colocará em funcionamento o evento onerror

</body>
</html>
```

Este exemplo estaria bem se sempre se detivesse o carregamento por clicar o botão de cancelar, mas o que acontece é que o usuário cancelou por ir a outra página através de um link, sairá a caixa de confirmação, mas não ocorrerá nada independentemente do que se responda e o navegador irá irremediavelmente à nova página.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Exemplo do evento onblur em Javascript

Onblur se ativa quando o usuário retira o foco da aplicação de um elemento da página. O foco da aplicação é o lugar onde está o cursor.

Se por exemplo, estamos situados em um campo de texto e saímos de tal campo, Seja porque clicamos com o mouse em outro campo do formulário ou em uma área vazia, ou seja porque o

usuário apertou o botão tabulador (Tab) que move o foco até o seguinte elemento da página.

Se eu desejo que, ao se situar fora de um campo de texto, se comprove que o valor introduzido é correto posso utilizar onblur e chamar a uma função que comprove se o dado é correto. Se não for correto posso obrigar ao foco da aplicação a se situar novamente sobre o campo de texto e avisar ao usuário para que mude tal valor.

Pode ser uma maneira interessante de assegurarmos que em um campo de texto encontra-se um valor válido. Embora tenha algum problema, como veremos mais adiante.

Vamos começar por um caso simples, no qual somente desejamos comprovar um campo de texto para assegurarmos que é um número inteiro.

```
<html>
<head>
  <title>Evento onblur</title>

<script>
function validarInteiro(valor){
  //tento converter a inteiro.
  //se era um inteiro não lhe afeta, se não era tenta converte-lo
  valor = parseInt(valor)

  //Comprovo se é um valor numérico
  if (isNaN(valor)) {
    //então (não é número) devolvo o valor cadeia vazia
    return ""
  }else{
    //Em caso contrário (Se era um número) devolvo o valor
    return valor
  }
}

function comprovaValidoInteiro(){
  inteiroValidado = validarInteiro(document.f1.numero.value)
  if (inteiroValidado == ""){
    //se era a cadeia vazia é que não era válido. Aviso
    alert ("Deve escrever um inteiro!")
    //seleciono o texto
    document.f1.numero.select()
    //coloco outra vez o foco
    document.f1.numero.focus()
  }else
    document.f1.numero.value = inteiroValidado
}
</script>
</head>
<body>
<form name=f1>
Escreva um número inteiro: <input type=text name=numero size=8 value="" onblur="comprovaValidoInteiro()">
</form>

</body>
</html>
```

Ao sair do campo de texto (onblur) se executa `comprovaValidoInteiro()`, que utiliza a função `validarInteiro`. Se o valor devolvido pela função não for o de um inteiro, neste caso se receberia uma cadeia vazia, mostra uma mensagem em uma caixa alert, seleciona o texto escrito na caixa e coloca o foco da aplicação na caixa de texto, para que o usuário coloque outro valor.

Até que o visitante não escreva um número inteiro no campo de texto o foco da aplicação permanecerá no campo e continuará recebendo mensagens de erro.

Podemos [ver este exemplo em funcionamento](#) em uma página a parte.

*Artigo por Miguel Angel Alvarez - Tradução de JML*

## Continuação do exemplo de onblur

Vimos o exemplo no [exemplo do método onblur relatado anteriormente](#) uma possível técnica para comprovar os dados de um campo de formulário. Agora vamos ver como evoluir esta técnica se tivermos mais de um campo a validar, dado que se pode complicar bastante o problema.

De fato, antes de ler nossa solução proposta, creio que seria um bom exercício para o leitor praticar esse mesmo exemplo para os dois campos e trabalhar um pouco com a página para ver se encontramos algum problema.

Muito provavelmente, encontraremos com um curioso loop infinito que nos vai dar mais quebra-cabeça para solucionar.

Na prática, o leitor pode tentar validar um número inteiro e um código postal. Para validar um código postal precisamos comprovar que é uma cadeia de texto composta geralmente por 5 caracteres e todos são inteiros.

Se alguém quiser tentar, a função para validar um código postal seria algo parecido com isto:

```
function ValidoCP(){
    CPValido=true
    //se não tem 5 caracteres não é válido
    if (document.f1.codigo.value.length != 5)
        CPValido=false
    else{
        for (i=0;i<5;i++){
            CAtual = document.f1.codigo.value.charAt(i)
            if (validarInteiro(CAtual)== ""){
                CPValido=false
                break;
            }
        }
    }
    return CPValido
}
```

Simplesmente se encarrega de comprovar que o campo de texto contém 5 caracteres e de fazer um percorrido por cada um dos caracteres para comprovar que todos são inteiros. A princípio se supõem que o código postal é correto, colocando a variável CPValido a true, e se alguma comprovação falha muda-se o estado correto a incorreto, passando tal variável a false.

Pode-se comprovar ao montar o exemplo com dois campos...nós agora vamos dar uma solução ao problema bastante complicadinha, pois incluímos instruções para evitar o efecto do loop

infinito. Não vamos ver o exemplo que daria o erro, deixamos para aqueles que desejarem tentar por si mesmo e recomendamos fazer isto porque assim, compreenderemos melhor o seguinte código.

```
<html>
<head>
  <title>Evento onblur</title>

<script>
avisado=false
function validarInteiro(valor){
  //tento converter a inteiro.
  //se era um inteiro nao lhe afeta, e se nao era tenta converte-lo
  valor = parseInt(valor)

  //Comprovo se é um valor numérico
  if (isNaN(valor)) {
    //entao (nao é numero) devolvo o valor cadeia vazia
    return ""
  }else{
    //No caso contrario (Se era um número) devolvo o valor
    return valor
  }
}

function comprovaValidoInteiro(){
  inteiroValidado = validarInteiro(document.f1.numero.value)
  if (interoValidado == ""){
    //se era a cadeia vazia é que nao era válido. Aviso
    if (!avisado){
      alert ("Deve escrever um inteiro!")
      //seleciono o texto
      document.f1.numero.select()
      //coloco outra vez o foco
      document.f1.numero.focus()
      avisado=true
      setTimeout('avisado=false',50)
    }
  }else
    document.f1.numero.value = inteiroValidado
}

function comprovaValidoCP(){
  CPValido=true
  //se nao tem 5 caracteres nao é válido
  if (document.f1.codigo.value.length != 5)
    CPValido=false
  else{
    for (i=0;i<5;i++){
      CAtual = document.f1.codigo.value.charAt(i)
      if (validarInteiro(CAtual)== ""){
        CPValido=false
        break;
      }
    }
  }
}
if (!CPValido){
  if (!avisado){
    //se nao é valido, aviso
    alert ("Deve escrever um código postal válido")
    //seleciono o texto
    document.f1.codigo.select()
    //coloco outra vez o foco
    //document.f1.codigo.focus()
  }
}
```

```
        avisado=true
        setTimeout('avisado=false',50)
    }
}
}
</script>

</head>
<body>

<form name=f1>
Escreva um número inteiro: <input type=text name=numero size=8 value="" onblur="comprovaValidoInteiro()">
<br>
Escreva um código postal: <input type=text name=codigo size=8 value="" onblur="comprovaValidoCP()"> *espera
uma cadeia com 5 caracteres numéricos

</form>

</body>
</html>
```

Este exemplo continua a guia do [primeiro exemplo de onblur](#) deste artigo, incluindo um novo campo a validar.

Para solucionar o tema do loop infinito, que vocês poderam investigar por vocês mesmos e no qual se mostrava uma caixa de alerta atrás da outra indefinidamente, utilizamos uma variável chamada avisado que contem um true se já foi avisado de que o campo estava mal e um false se ainda não foi avisado.

Quando se mostra a caixa de alerta, se comprova se foi avisado ou não ao usuário. Se já foi avisado não se faz nada, evitando que se mostrem mais caixas de alerta. Se ainda não foi avisado mostra-se a caixa de alerta e coloca-se o foco no campo que era incorreto.

Para restituir a variável avisado a false, de modo que a próxima vez que se escreva mal o valor se mostre a mensagem correspondente, se utiliza o método setTimeout, que executa a instrução com um atraso, neste caso de 50 milésimos de segundos. O suficiente para que não se meta em um loop infinito.

**Nota:** Depois de todos os apetrechos que tivemos que colocar para que este evento se comporte corretamente para cumprir com o objetivo desejado, é possível pensar que não vale a pena utiliza-lo neste objetivo. Podemos fazer uso do evento onchange, ou comprovar todos os campos de uma vez só quando o usuário decidiu envia-lo.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Elementos de formulário select associados

Vamos conhecer um dos truques mais solicitados de Javascript, que tem muita relação com o tema de formulários e onde se utiliza o evento onchange de Javascript. É um exemplo sobre como realizar uma página com dois selects onde, segundo o valor escolhido em um deles, mudem as opções possíveis do outro select.

O melhor para ver o que vamos fazer é ver uma [página web onde se mostra em funcionamento o script](#). Para ver seu funcionamento, devemos mudar a seleção do primeiro select e comprovaremos como as opções do segundo select mudam automaticamente.

O exemplo que ilustramos utiliza estados e países. Ao escolher no primeiro select um país, no segundo deve nos mostrar os estados desse país para que possamos escolher um estado, mas somente um que esteja no país selecionado no primeiro término.

## Conhecer o objeto select e os option

É importante conhecer os objetos de formulário select e os option. Os select correspondem com as caixas de seleção desdobráveis e os option com cada uma das opções da caixa desdobrável. Podemos [ver um artigo que fala sobre isso](#).

Teoricamente nos interessa fazer várias coisas que têm a ver com extrair o valor de um select em qualquer momento, observar seu número de opções e, para cada opção, colocar seu valor e seu texto associado. Tudo isto aprenderemos a fazer neste exemplo.

**Referência:** Para conhecer o trabalho e a hierarquia de objetos javascript (Tudo isso é a base do trabalho com os elementos das páginas em Javascript) devemos ler o [manual de Javascript II](#).

## Modo de solucionar o problema

Para começar, vamos utilizar um formulário com dois selects, um para o país e outro para o estado.

```
<form name="f1">
<select name=pais onchange="muda_estado()">
<option value="0" selected>Selecione...
<option value="1">Espanha
<option value="2">Brasil
<option value="3">Portugal
<option value="4">França
</select>

<select name=estado>
<option value="-">-
</select>
</form>
```

Observamos no select associado ao país deste formulário que, quando se muda a opção de país, deve-se chamar a função muda\_estado(). Veremos mais adiante esta função, agora é importante observar que está associada ao evento onchange que se ativa quando muda a opção no select.

Todo o resto será código Javascript. Começamos definindo um montão de arrays com os estados de cada país. Neste caso temos só 4 países, então necessitaremos 4 arrays. Em cada array tenho uma lista de estados de cada país, colocados em cada um dos elementos do array. Ademais, deixaremos o primeiro campo com um valor "-" que indica que não foi selecionado nenhum estado.

```
var estados_1=new Array("-", "Andalucía", "Asturias", "Balears", "Canarias", "Castilla y León", "Castilla-La Mancha", "...")
var estados_2=new Array("-", "Rio de Janeiro", "Bahia", "São Paulo", "Santa Catarina", "Minas Gerais", "...")
var estados_3=new Array("-", "Algarve", "Alentejo", "Norte", "Vale do Tejo", "...")
var estados_4=new Array("-", "Aisne", "Creuse", "Dordogne", "Essonne", "Gironde", "...")
```

Observemos que os índices do array de cada país se correspondem com os do select do país. Por exemplo, a opção Espanha, tem o valor associado 1 e o array com os estados de Espanha se chama estados\_1.

O script se completa com uma função que realiza o carregamento dos estados no segundo select. O mecanismo realiza basicamente estas ações:

- Detecto o país que foi selecionado
- Se o valor do país não for 0 (o valor 0 é quando não foi selecionado nenhum país)
  - Tomo o array de estados adequado, utilizando o índice do país.
  - Marco o número de opções que deve ter o select de estados
  - Para cada opção do select, coloco seu valor e texto associado, que faz corresponder com o indicado no array de estados.
- SE NÃO (O valor de país é 0, não foi selecionado país)
  - Coloco no select de estado um único option com o valor "-", que significava que não havia estado.
- Coloco a opção primeira do select de estado como o selecionado.

A função tem o seguinte código. Está comentado para que se entenda melhor.

```
function muda_estado(){
    //tomo o valor do select do pais escolhido
    var pais
    pais = document.f1.pais[document.f1.pais.selectedIndex].value
    //vejo se o pais está definido
    if (pais != 0) {
        //se estava definido, entao coloco as opcoes do estado correspondente.
        //seleciono o array de estado adequado
        meus_estados=eval("estados_" + pais)
        //calculo o numero de estados
        num_estados = meus_estados.length
        //marco o número de estados no select
        document.f1.estado.length = num_estados
        //para cada estado do array, o introduzo no select
        for(i=0;i<num_estados;i++){
            document.f1.estado.options[i].value=meus_estados[i]
            document.f1.provincia.options[i].text=meus_estados[i]
        }
    }else{
        //se não havia estado selecionado, elimino os estados do select
        document.f1.estado.length = 1
        //coloco um traço na única opção que deixei
        document.f1.estado.options[0].value = "-"
        document.f1.estado.options[0].text = "-"
    }
    //marco como selecionada a opção primeira de estado
    document.f1.estado.options[0].selected = true
}
```

Podemos [ver uma página com o exemplo em funcionamento](#).

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Evento onunload de Javascript

Vejamos um exemplo do evento onunload, que, lembramos, ativa-se quando o usuário abandona a página web. Portanto, onunload serve para executar uma ação quando o usuário sai da página, seja porque clica um link que leva fora da página ou porque fecha a janela do navegador.



O exemplo que desejamos mostrar serve para abrir uma página web em outra janela quando o usuário abandona a página. Deste modo, atuam muitos dos incômodos popups das páginas web, abrindo-se justo quando abandonamos o site que estávamos visitando.

```
<html>
<head>
  <title>Abre ao sair</title>
  <script>
    function abrejanela(){
      window.open("http://www.google.com.br","venda","")
    }
  </script>
</head>

<body onload="abrejanela()">

<a href="http://www.criarweb.com">CW!!</a>
</body>
</html>
```

O exemplo é tão simples que quase sobram as explicações. Simplesmente criamos uma função que abre uma janela secundária e a associamos com o evento onload, que se coloca na etiqueta <body>.

Pode-se [ver em funcionamento em uma página a parte](#).

**Referência:** Se não temos uma base de Javascript será bom acessar a nossa seção [Javascript a fundo](#).

Clique aqui se desejar [conhecer mais coisas sobre os eventos](#).

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*

## Evento onload de Javascript

O evento onload de Javascript se ativa quando termina de se carregar a página. Deve ser colocado na etiqueta <body>, embora em versões modernas de Javascript, também o aceitam outros elementos como as imagens.

Com o evento onload podemos executar ações justo quando terminem de se carregar todos os elementos da página. É um evento bastante utilizado, pois é muito habitual que seja desejado realizar ações nesse preciso instante. Em nosso exemplo, vamos ver como poderíamos fazer um script para motivar aos nossos visitantes que nos votem em um ranking qualquer de páginas web.

A idéia é que a página se carregue inteira e, uma vez estando carregada, apareça uma janela de Javascript onde se proponha ao visitante votar a página. É interessante esperar que termine de carregar a página inteira para que o visitante possa ver a web que se propõe votar, antes de realmente pedirmos seu voto.

O código seria o seguinte:

```
<html>
<head>
  <title>Vote-me!!</title>
```

```
<script language="JavaScript">
function pedirVoto(){
    if (confirm("Esta página está muito legal (como pode ser vista). Dê o seu voto!")){
        window.open("http://www.oquefor.com/votar.php?idvoto=12111","", "")
    }
}
</script>
</head>

<body onload="pedirVoto()">
<h1>Página SuperLegal</h1>
<br>
Esta página está muito bonita. Sou o autor e posso garantir que não há muitas páginas com tanta qualidade na
Internet
<br>
<br>
<a href="#">Entrar</a>

</body>
</html>
```

Observamos que na etiqueta <body> temos o evento onload="pedirVoto()". Ou seja, que quando se carregar a página se chamará a uma função chamada pedirVoto(), que definimos no bloco de script que temos no cabeçalho.

A função pedirVoto() utiliza uma caixa confirm para saber se realmente o usuário deseja nos votar. A função confirm() mostra uma caixa com um texto e dois botões, para aceitar ou cancelar. O texto é o que recebe por parâmetro. Dependendo do que se clique nos botões, a função confirm() devolverá um true, se se clicou o botão aceitar, ou um false, no caso de que se clicasse sobre cancelar.

A função confirm() está por sua vez colocada dentro de um bloco condicional if, de modo que, dependendo do que se clicou na caixa confirm, o if se avaliará como positivo ou negativo. Neste caso, somente realizam-se ações se se clicou sobre aceitar.

Para acessar à página onde se contabiliza nosso voto temos o método window.open(), que serve para abrir janelas secundárias ou popups. Mostramos a página onde se vota em uma janela secundária para que o visitante não saia de nosso web, já que acaba de entrar e não queremos que vá embora.

Com isto, fica mais ou menos ilustrado como fazer uso do evento onload. Com certeza haverá muitos mais casos onde utilizá-lo em suas criações.

*Artigo por **Miguel Angel Alvarez** - Tradução de JML*