

URL original:

<http://www.truquesedicas.com/tutoriais/javascript/index.htm>



TUTORIAL DE JAVASCRIPT

Índice

[O que é o Javascript?](#)

Uma breve noção sobre o Javascript desde da sua origem até a sua funcionalidade.

[Javascript não é Java!](#)

É importante saber que o Javascript é completamente diferente do Java.

[Um pouco de teoria objecto](#)

Uma simples explicação ilustrada sobre os objecto do javascript.

[Ferramentas para o Javascript](#)

O pouco que é necessário para trabalhar com o Javascript

[O Javascript simples](#)

O começo com esta linguagem.

[Inserir texto](#)

Todas os métodos de inserir texto em Javascript

[Variáveis](#)

As variáveis contém dados que podem ser modificados durante a execução de um programa...

[Operadores](#)

Vejam os diferentes operadores que estão a nossa disposição no Javascript.

[Funções](#)

Uma função é um grupo de linha(s) de código de programação...

[Eventos](#)

Com os eventos e sobretudo sua gestão, abordamos o lado "mágico" do Javascript

[Condições](#)

Num momento ou outro da programação, teremos necessidade de testar uma condição.

[Formulários](#)

Com o Javascript, os formulários Html tomem outra dimensão.

[Um pouco de tudo](#)

Alguns métodos e funções mais usados.

[As mensagens de erros](#)

Existem 3 grandes categorias de erros na utilização de um programa Javascript.

[Mini FAQ](#)

Respostas para as dúvidas mais frequentes

[Objecto Window](#)

Tudo sobre o objecto Window.

[Objecto String](#)

Tudo sobre o objecto String.

[Objecto Math](#)

Tudo sobre o objecto Math.

[Objecto Date](#)

Tudo sobre o objecto Date.

[Objecto Array](#)

Tudo sobre o objecto Array.

[Frames](#)

Em Javascript, interessa a capacidade das frames à interagir. Ou seja a capacidade de trocar informações entre elas.

O que é o Javascript?

O Javascript é uma linguagem de script que incorporado nos tag's Html, permite incrementar a apresentação e interactividade das páginas Web.

Javascript é então uma extensão do código Html das páginas Web. Os scripts, que se inserem nos tag's Html, podem ser comparados aos macros de uma formatação de texto. Estes scripts vão ser gerados e executados pelo próprio browser sem fazer apelo aos recursos de um servidor. Estas instruções serão assim executadas directamente e sobretudo sem atrasos.

Javascript foi desenvolvido inicialmente pela Netscape e na altura intitulava-se LiveScript. Adoptado no fim do ano de 1995, pela firma Sun (que também desenvolveu o Java), ele tomou assim o seu nome actual Javascript.

Javascript não é próprio do browser Netscape. Microsoft também adoptou o Javascript desde do seu Internet Explorer 3. E melhorou significativamente no Explorer 4.

As versões do Javascript sucederam-se com as diferentes versões do Netscape: Javascript para Netscape 2, Javascript 1.1 para Netscape 3 e Javascript 1.2 para Netscape 4. Apesar de existir alguns problemas de compatibilidade, segundo o browser utilizado nas página que contém codificação Javascript.

O futuro do Javascript está entre as mãos dos dois grandes browsers da Web e em parte ligada a guerra entre a Microsoft e a Netscape. Podemos mesmo assim prever um futuro promissor a esta linguagem sobretudo pela sua independência em relação ao servidor.

Javascript não é Java!

É importante saber que o Javascript é completamente diferente do Java. Mesmo que os dois sejam utilizados para criar páginas Web evoluídas. Mesmo que os dois usam o mesmo termo Java (café em americano), temos aí duas ferramentas informáticas bem diferentes.

JAVASCRIPT	JAVA
Código integrado na página Html	Modulo (applet) distinto da página Html
Código interpretado pelo browser no momento da execução	O código é compilado antes da sua execução
Códigos de programação simples mas para aplicações limitadas	Linguagem de programação muito mais complexa mas mais poderosa
Permite de aceder aos objectos do browser	Não tem acesso aos objectos do browser
Confidencialidade do código é nulo (Código é visível)	Segurança (Código compilado)

Javascript é mais simples realizar visto que o código que se insere directamente nas páginas Html com por exemplo um simples editor de texto como o Notepad. O Java necessita de uma compilação prévia do código.

O campo de aplicação do Javascript é um tanto limitado enquanto que com o Java pode-se fazer tudo em princípio.

Como o código Javascript é incluído nas páginas Html, este é visível e pode ser copiado por todos. Pelo contrário o Java, o código é compilado o que o torna indecifrável.

Os códigos de Javascript não atrasem o carregamento da página enquanto que um applet Java pode demorar minutos a carregar.

Um pouco de teoria objecto...

Os objectos e as suas hierarquias

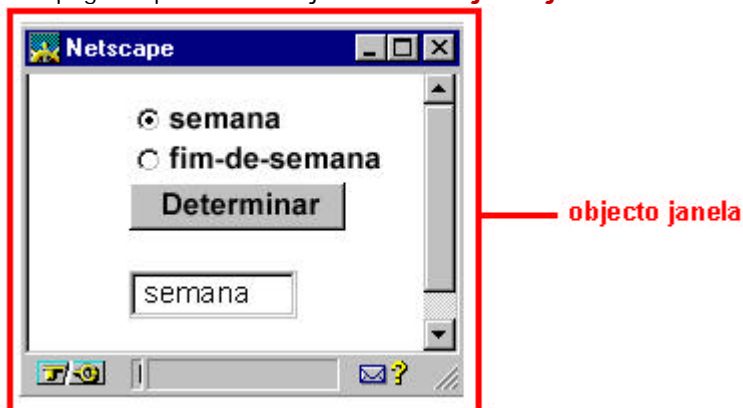
Para bons internautas, vêem neste écran uma página Web. Javascript vai dividir esta página em objectos e vai permitir aceder e manipular este objectos sem retirar informações.

Vejamos em primeiro uma ilustração dos diferentes objectos que uma página pode conter.

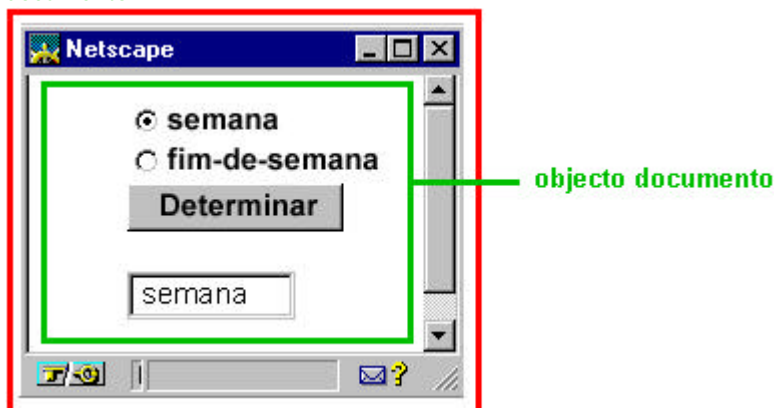
Carregou-se a página seguinte:



Esta página aparece numa janela. É o **objecto janela**.



Nesta janela, há um documento Html, é o **objecto documento**. Quer isto dizer (e é aqui que vemos aparecer a noção de hierarquia dos objectos Javascript), o **objecto janela** contém o **objecto documento**.

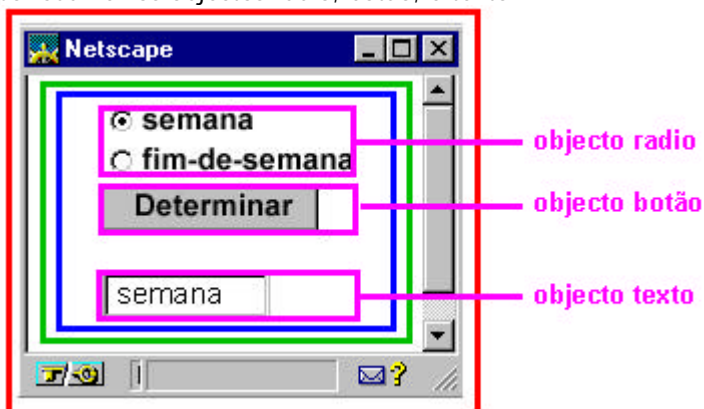


Um pouco de teoria objecto... (continuação)

Neste documento, temos um formulário. É o **objecto formulário**. Dito de outra maneira, o **objecto janela** contém um **objecto documento** que contém por sua vez um **objecto formulário**.



Neste documento encontramos 3 objectos. Os botões radio, um botão clássico e uma zona de texto. Que são respectivamente o **objecto radio**, o **objecto botão**, e o **objecto texto**. Por outras palavras o **objecto janela** contém o **objecto documento** que contém o **objecto formulário** que contém por sua vez os objectos radio, botão, e texto.



A hierarquia deste exemplo é então:

```

janela > documento > formulário > radio
                                     > botão
                                     > texto
    
```

Para aceder a um objecto, é necessário dar o caminho completo do objecto começando pelo objecto mais externo até ao objecto de referência.

Seja por exemplo para o botão radio "semana": `(window).document.form.radio[0]`.

Insira-se o objecto window entre parenteses porque ele ocupa a primeira posição na hierarquia.

E enfim para os puristas, Javascript não é propriamente uma linguagem orientada para os objectos tal o como C++ ou Java. Dizemos antes que Javascript é uma linguagem baseada nos objectos.

As propriedades dos objectos

Uma propriedade é um atributo, uma característica, uma descrição do objecto. Por exemplo, o objecto volante de um carro tem como propriedades que ele pode ser de madeira ou em couro. O objecto livro tem como propriedades seu autor, sua editora, seu título, etc.

Ainda os objectos Javascript tem propriedades personalizadas. No caso dos botões radio, uma das suas propriedades é, por exemplo, sua selecção ou sua não-selecção (checked em inglês).

Em Javascript, para aceder as propriedades, utiliza-se a sintaxe:

nome_do_objecto.nome_da_propriedade

No caso do botão radio "semana", para testar a propriedade de selecção, escreva-se:

document.form.radio[0].checked

Ferramentas para o Javascript

Para aprender e descobrir o Javascript, é necessário:

1. Um browser compatível com o Javascript.
2. Um bom conhecimento de Html.
3. Um simples editor de texto.

Um browser compatível com o Javascript

Unicamente a Netscape e a Microsoft dispõem de browser Javascript "enabled". Para a Microsoft a partir do IExplorer 3.0 e a Netscape a partir do Netscape Navigator 2.0.

De toda a maneira deveremos estar atento as versões de Javascript exploradas por este browsers.

Netscape 2.0	Javascript (baptizado posteriormente por 1.0)
Netscape 3.0	Javascript 1.1
Netscape 4.0	Javascript 1.2
I Explorer 3.0	Alguma codificação de Javascript 1.0
I Explorer 4.0	Javascript 1.2

Um bom conhecimento de Html

Como o código do Javascript vem juntar-se ao código da linguagem Html, um conhecimento aprofundado dos tags Html é praticamente indispensável. Assim os utilizadores dos editores Html "whsiwyg" ou outros "publishers" Html risquem de serem obrigados a voltar aos seus estudos.

Posso apenas recomendar a consulta do "**Tutorial de Html**"

Um simples editor de texto

Uma página Html é simplesmente texto. O código Javascript é também ele que só texto. Nada mais simples do que editor de texto como o Notepad do Windows para inserir o vosso Javascript na vossa página Html. Cada vez mais editores Html whsiwyg propõem uma janela para o Javascript. Atenção ! Se alguns parecem bem feitos como o Dreamweaver e o WebExpert 2 com outros, o código Javascript introduzido seja alterado pelo editor como FrontPage ou Netscape Gold...

O Javascript simples

O tag <SCRIPT>

Como já foi visto o Javascript insere-se numa página Web.

A linguagem Html utiliza os tags para "dizer" ao browser para inserir uma porção de texto em negrito, em itálico, etc.

Na lógica da linguagem Html, é então necessário assinalar ao browser através de um tag que o texto que segue é script e que é Javascript (et não VBScript). É o tag:

<SCRIPT LANGUAGE="Javascript">

Também é necessário informar o browser do fim do script.
É o tag:

```
</SCRIPT>.
```

Comentários

Será por vez útil incluir comentários pessoais no vosso código Javascript.
É mesmo vivamente recomendado como para todas as linguagens de programação. Javascript utiliza as convenções utilizadas em C e C++ seja:

```
// comentário
```

Tudo o que está escrito entre o // e o fim da linha será ignorado.

Também é possível incluir comentários em diversas linhas com o código:

```
/* comentário em  
diversas linhas */
```

Não confundir os comentários Javascript e os comentários Html (<!-- ...-->).

Esconder o script para os antigos browsers

Os browsers que não compreendem o Javascript (e ainda existem) ignoram o tag <script> e vão inserir o código do script na página sem poder o executar. Para evitar o aparecimento pouco estético das inscrições do código, utiliza-se os tags de comentário da linguagem Html <!-- ... -->. O Javascript irá parecer assim:

```
<SCRIPT LANGUAGE="javascript">  
<!--Esconder o script para os antigos browsers  
...  
programa Javascript  
...  
// Cessem de esconder o script -->  
</SCRIPT>
```

Onde incluir o código em Javascript ?

O princípio é simples. Basta respeitar este dois princípios seguintes :

- em qualquer sítio
- mas onde é necessário

O browser trata a página Html de cima para baixo.
Por consequência, todas as instruções só poderão ser executadas se o browser possuir neste preciso momento todos os elementos necessários para sua execução. Estes elementos devem ser declarados antes ou a quando a instrução.

Para assegurar que o programa script é carregado na página e pronta a funcionar a todas as intervenções do utilizador, aqui tomaremos por hábito de declarar sistematicamente (sempre que será possível) um máximo de elementos nos tags de cabeçalho seja entre <HEAD> e </HEAD> e antes do tag <BODY>. Será o caso por exemplo para as funções.

Nada proíba de inserir vários scripts na mesma página Html.

É preciso notar que a utilização do tag script não é sempre obrigatório. Será o caso dos eventos Javascript (por exemplo "onClick") onde se tem simplesmente que inserir o código no interior do comando Html como um atributo desta. O evento irá chamar a função Javascript quando o comando Html será activado. Pode-se dizer que o Javascript funciona como uma extensão da linguagem Html.

Uma primeira instrução Javascript

Sem entrar verdadeiramente com detalhes, vejamos uma primeira instrução Javascript (um método do objecto window) seja a instrução **alert()**.

```
alert("vosso texto");
```

Teste

Esta instrução insere uma mensagem numa caixa de diálogo munido com um botão OK. Para continuar na página o utilizador deverá clicar neste botão.

Pode reparar os pontos-e-virgulos no fim de cada instrução Javascript, o Javascript é mais flexível do que as outras linguagens e geralmente não dá mensagem de erro caso que faltam. Podemos considerar que o ponto-e-virgula é opcional e só é obrigatório quando escreve-se várias instruções na mesma linha.

O Javascript simples (continuação)

Primeira página Html com Javascript

<code><HTML></code>	Html normal
<code><HEAD></code>	...
<code><TITLE>Meu primeiro Javascript</TITLE></code>	...
<code></HEAD></code>	...
<code><BODY></code>	...
Bla-bla em Html	...
<code><SCRIPT LANGUAGE="Javascript"></code>	Início do script
<code><!--</code>	Esconder o script
<code>alert("vosso texto");</code>	Script
<code>//--></code>	Fim de esconder
<code></SCRIPT></code>	Fim do script
Continuação bla-bla em Html	Html normal
<code></BODY></code>	...
<code></HTML></code>	...

Notação

Javascript é **case sensitive**. Assim será necessário escrever `alert()` e não `Alert()`. Para a escrita das instruções Javascript, utilizaremos o alfabeto ASCII clássico (a 128 caracteres) como em Html. Os caracteres acentuados como "é" ou "à" só podem ser utilizados nos textos tipo "vosso texto" do nosso exemplo.

As aspas " e o apóstrofo ' fazem parte da linguagem do Javascript. Pode-se utilizar uma ou outra forma na condição de não as misturar. Assim `alert('...')` dará uma mensagem de erro.

Para browser que suportem o Javascript 1.0

Versões da linguagem Javascript

Com as diferentes versões existente (Javascript 1.0, Javascript 1.1 et Javascript 1.2), podemos imaginar scripts adaptados as diferentes versões mas sobretudo aos diferentes browsers;

```
<SCRIPT LANGUAGE="Javascript">
// programa para Netscape 2 et Explorer 3
var version="1.0";
</SCRIPT>

<SCRIPT LANGUAGE="Javascript1.1">
```

```
// programa para Netscape 3 et Explorer 4
var version=1.1;
</SCRIPT>

<SCRIPT LANGUAGE="Javascript1.2">
// programa para Netscape 4
var version=1.2;
</SCRIPT>
```

Extensão .js para scripts externos

É possível utilizar ficheiros externos para os programas Javascript. Podemos assim guardar os scripts dentro de ficheiros distintos (com a extensão **.js**) e as chamar a partir de um ficheiro Html. O programador pode desta maneira constituir-se uma biblioteca de script e as chamar na maneira dos #include do C ou C++.

Alert() ...vermelha

Aquela simples janelinha em Javascript que criamos é utilizada para chamar atenção do utilizador para coisas mais importantes. O Javascript põe a vossa disposição a possibilidade de criar novas janelas com a dimensão desejada que aparecem um pouco como as Popup dos ficheiros de ajuda. Que iremos estudar mais a frente em o objecto Window.

Alert() é um método do objecto Window. Para se conformar à notação clássica nome_do_objecto.nome_da_propriedade, poderíamos ter escrito **window.alert()**. Sendo window sempre o primeiro objecto Javascript, este é por defeito sempre interpretado pelo browser assim ele torna-se facultativo.

Para que o texto da janela alert() aparece em várias linhas, será necessário utilizar o caractere especial **\n** para criar uma nova linha.

Inserir texto

Método do objecto documento

Como já foi explicado anteriormente, o que aparece no écran pode ser dividido em objectos e que o Javascript dá possibilidade de aceder a este objectos (**um pouco de teoria do objecto**).

A página Html que aparece na janela do browser é um objecto tipo **document**.

A cada objecto Javascript, o programador da linguagem previu um conjunto de métodos (ou funções dedicados a este objecto). Para **document** o Javascript dedicou o método "escrever no documento" , é o método **write()**.

O chamamento do método se faz segundo a notação:

```
nome_do_objecto.nome_do_método
```

Para chamar o método write() do documento, escreva-se:

```
document.write();
```

O método write()

A sintaxe é bastante simples

```
write("vosso texto");
```

Pode-se também escrever uma variável, seja a variável resultado,
write(resultado);

Para associar texto (cadeia de caracteres) e variáveis, utiliza-se a instrução **write("O resultado é" + resultado);**

Pode-se utilizar os tags Html para incrementar o texto **write("O resultado é" + resultado);** ou **write ("" + "O resultado é" + "" + resultado)**

Exemplo (clássico !)

Agora vamos escrever texto em Html e em Javascript.

```
<HTML>
<BODY>
<H1>Isto é Html</H1>
<SCRIPT LANGUAGE="Javascript">
<!--
document.write("<H1>E isto é Javascript</H1>");
//-->
</SCRIPT>
</BODY>
</HTML>
```

E o resultado é:



Formatação dos caracteres em Javascript

variável.big();

O uso de **.big()** vai inserir a variável como se ela estivesse compreendida entre os tags Html **<BIG></BIG>**.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto"; (str é uma variável)

document.write("<BIG>" + str + "</BIG>");
document.write('<BIG>Texto</BIG>');
document.write(str.big());
document.write("Texto".big());
```

variável.small();

O uso de **.small()** vai inserir a variável como se ela estivesse compreendida entre os tags Html **<SMALL></SMALL>**.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto";

document.write("<SMALL>" + str + "</SMALL>");
```

```
document.write("<SMALL>Texto" + "</SMALL>");  
document.write(str.small());  
document.write("Texto".small());
```

variável.blink();

O uso de **.blink()** vai inserir a variável como se ela estivesse compreendida entre os tags Html **<BLINK></BLINK>**.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto";  
  
document.write('<BLINK>'+str+'</BLINK>');  
document.write("<BLINK>Texto</BLINK>");  
document.write(str.blink());  
document.write("Texto".blink());
```

variável.bold();

O uso de **.bold()** vai inserir a variável como se ela estivesse compreendida entre os tags Html ****.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto";  
  
document.write("<B>" + str + "</B>");  
document.write("<B>Texto</B>");  
document.write(str.bold());  
document.write("Texto".bold());
```

variável.fixed();

O uso de **.fixed()** vai inserir a variável como se ela estivesse compreendida entre os tags Html **<TT></TT>**.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto";  
  
document.write("<TT>" + str + "</TT>");  
document.write("<TT>Texto</TT>");  
document.write(str.fixed());  
document.write("Texto".fixed());
```

Inserir texto (continuação)

variavel.italics();

O uso de **.italics()** vai inserir a variável como se ela estivesse compreendida entre os tags Html **<I></I>**.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto"; (str é uma variável)  
  
document.write("<I>" + str + "</I>");  
document.write("<I>Texto</I>");  
document.write(str.italics());  
document.write("Texto".italics());
```

variavel.fontcolor(color);

O uso de **.fontcolor(color)** vai inserir a variável como se ela estivesse compreendida entre os tags Html ****.

As quatro instruções Javascript seguintes são equivalentes :

```
str1="Texto";
str2="red"

document.write("<FONT COLOR='red'>"+str1+"</FONT>");
document.write("<FONT COLOR='red'>"+"Texto"</FONT>");
document.write(str1.fontcolor(str2));
document.write("Texto".fontcolor("red"));
```

variavel.fontSize(x);

O uso de **.fontSize(x)** vai inserir a variável como se ela estivesse compreendida entre os tags Html ****.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto";
x=3

document.write('<FONT SIZE=3>'+str+'</FONT>');
document.write("<FONT SIZE=3>"+"Texto"</FONT>");
document.write(str.fontSize(x));
document.write(str.fontSize(3));
```

variavel.strike();

O uso de **.strike()** vai inserir a variável como se ela estivesse compreendida entre os tags Html **<STRIKE></STRIKE>**.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto";

document.write("<STRIKE>"+str+"</STRIKE>");
document.write("<STRIKE>Texto"</STRIKE>");
document.write(str.strike());
document.write("Texto".strike());
```

variavel.sub();

O uso de **.sub()** vai inserir a variável como se ela estivesse compreendida entre os tags Html ****.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto";

document.write("<SUB>"+str+"</SUB>");
document.write("<SUB>Texto"</SUB>");
document.write(str.sub());
document.write("Texto".sub());
```

variavel.sup();

O uso de **.sup()** vai inserir a variável como se ela estivesse compreendida entre os tags Html ****.

As quatro instruções Javascript seguintes são equivalentes :

```
str="Texto";

document.write("<SUP>"+str+"</SUP>");
document.write("<SUP>Texto"</SUP>");
document.write(str.sup());
document.write("Texto".sup());
```

Instruções de formatação do *document*

Para já queria lembrar que o que se segue é opcional e que se pode utilizar a instrução **document.write()** de maneira clássica.

Seja **document.write("<BODY BGCOLOR="#FFFFFF");**

document.bgColor

Esta instrução permita de especificar a cor do fundo de um objecto **document**. pode-se utilizar o nome da cor ou o seu valor RGB.

```
document.bgColor="white";  
document.bgColor="#FFFFFF";
```

document.fgColor

Esta instrução permita de especificar a cor do primeiro plano (texto) de um objecto **document**. pode-se utilizar o nome da cor ou o seu valor RGB.

```
document.fgColor="black";  
document.fgColor="#000000";
```

document.alinkColor

Esta instrução permita de especificar a cor de um link activo (depois do clique do rato mas antes de sair do link) de um objecto **document**. pode-se utilizar o nome da cor ou o seu valor RGB.

```
document.alinkColor="red";  
document.alinkColor="#FF0000";
```

document.linkColor

Esta instrução permita de especificar a cor de um link de um objecto **document**. pode-se utilizar o nome da cor ou o seu valor RGB.

```
document.linkColor="blue";  
document.linkColor="#0000FF";
```

document.vlinkColor

Esta instrução permita de especificar a cor de um link já visitados (depois do clique do rato mas antes de sair do link) de um objecto **document**. pode-se utilizar o nome da cor ou o seu valor RGB.

```
document.vlinkColor="red";  
document.vlinkColor="#FF0000";
```

Variáveis

As variáveis em Javascript

As variáveis contêm dados que podem ser modificados durante a execução de um programa. Faz-se referência pelo nome desta variável.

Um nome de variável deve começar por uma letra (alfabeto ASCII) ou o símbolo `_` e ser composto de letras, de números e de caracteres `_` e `$`. O número de caracteres não está definido. Para chamar funções o Javascript é case sensitive. Atenção então as maiúsculas e minúsculas!

Declarações de variáveis

As variáveis podem declararem-se de duas maneira:

1. seja de maneira explícita. Diz-se em Javascript que isto é uma variável.

O comando que permite declarar uma variável é a palavra **var**. Por exemplo:

```
var Numero = 1  
var Nome = "Sérgio"
```

2. seja de maneira implícita. Escreva-se directamente o nome da variável seguido do valor de atribuição. Por exemplo:

```
Numero = 1  
Nome = "Sérgio"
```

Atenção! Apesar desta aparência facilidade, a maneira que se declare a variável terá uma grande importância com a **visibilidade** da variável no programa Javascript.

Ver para este assunto, a distinção entre variável local e variável global no Javascript avançado deste capítulo.

Para tornar o vosso script mais legível, aconselhamos de utilizar cada vez palavra var par declarar as variáveis.

Os dados sob Javascript

O Javascript utiliza 4 tipos de dados:

TIPO	DESCRIÇÃO
Números	Todo número inteiro ou com virgula tal que 22 ou 3.1416
Cadeias de caracteres	Toda cadeia de caracteres inseridas entre aspas tal que "cadeia de caracteres"
Booleanos	as palavras true para verdadeiro e false para falso
Palavra nulo	Palavra especial que representa <i>sem valor</i>

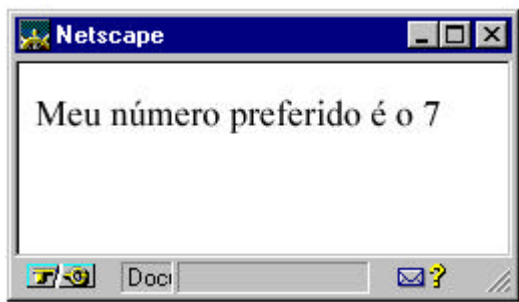
Nota-se também que ao contrário da linguagem C ou C++, Não se declara o tipo de dados de uma variável. Sendo assim não temos necessidade de inserir o tipo (int, float, double, char e outros) em Javascript.

Exercício

Vamos utilizar o método **write()** para visualizar as variáveis. Defina-se uma variável chamada "texto" que contém uma cadeia de caracteres "Meu número preferido é o " e uma outra chamada "variavel" inicializada em 7.

```
<HTML>  
<BODY>  
<SCRIPT LANGUAGE="Javascript">  
<!--  
var texto = "Meu número preferido é o "  
var variavel = 7  
document.write(texto + variavel);  
//-->  
</SCRIPT>  
</BODY>  
</HTML>
```

E obtêm-se o seguinte resultado:



Os nomes reservados

as palavras da lista abaixo só podem ser utilizadas para nomes de funções ou de variáveis. Algumas destas palavras são palavras-chaves Javascript, outras foram reservadas pela Netscape para um eventual uso futuro.

- A abstract
- B boolean break byte
- C case catch char class const continue
- D default do double
- E else extends
- F false final finally float for function
- G goto
- I if implements import in instanceof int interface
- L long
- N native new null
- P package private protected public
- R return
- S short static super switch synchronized
- T this throw throws transient true try
- V var void
- W while with

Variáveis globais e variáveis locais

As variáveis declaradas no início do script, fora e antes de todas as funções (ver mais a frente), serão globais, mesmo que esta seja declarada com `var` ou de maneira contextual. Pode-se assim chamar em qualquer sítio no script.

Numa função, uma variável declarada pela palavra-chave `var` e só pode ser invocada dentro da sua função. Sendo assim não se pode invocar a variável fora da função. Daí seu nome local. No entanto, sempre numa função, se a variável é declarada contextualmente (sem utilização da palavra `var`), sua invocação será global.

Voltaremos a falar deste tema nos estudos das funções.

Operadores

As variáveis são úteis mas é necessário saber as manipular-las, Vejamos os diferentes operadores que estão à nossa disposição no Javascript.

Nos exemplos, O valor inicial da variável x será sempre igual 11

operadores de cálculos

SÍMBOLO	NOME	SIGNIFICADO	EX.	RESUL.
+	mais	adição	$x + 3$	14
-	menos	subtracção	$x - 3$	8
*	multiplicar por	multiplicação	$x * 2$	22
/	dividir por	divisão	$x / 2$	5.5
%	resto da divisão por	resto	$x \% 5$	1
=	toma valor de	valorização	$x = 5$	5

operadores de comparação

SÍMBOLO	NOME	EX.	RESUL.
==	igual	$x == 11$	true
<	inferior	$x < 11$	false
<=	inferior ou igual	$x <= 11$	true
>	superior	$x > 11$	false
>=	superior ou igual	$x >= 11$	true
!=	diferente	$x != 11$	false

Importante. Confunda-se muitas vezes o = e o == (dois símbolos =). O = é um operador de atribuição de valor enquanto que o == é um operador de comparação. Este confusão é uma grande fonte de erros de programação.

operadores de associação

Chama-se assim aos operadores que realizem um cálculo no qual uma variável intervém nos dois lados do símbolo = (são de alguma maneira operadores de atribuição).

No exemplo que segue o x tem sempre o valor 11 e y terá como valor 5

SÍMBOLO	DESCRIÇÃO	EX.	SIGNIFICADO	RESUL.
+=	mais igual	$x += y$	$x = x + y$	16
-=	menos igual	$x -= y$	$x = x - y$	6
*=	multiplicar igual	$x *= y$	$x = x * y$	55
/=	dividir igual	$x /= y$	$x = x / y$	2.2

operadores lógicos

Também chamados operadores booleanos, esses operadores servem a verificar duas ou várias condições.

SÍMBOLO	NOME	EX.	SIGNIFICADO
&&	e	(condição1) && (condição2)	condição1 <u>e</u> condição2
	ou	(condição1) (condição2)	condição1 <u>ou</u> condição2

operadores de incrementação

Estes operadores vão aumentar ou diminuir o valor da variável de uma unidade. Esses operador são muito usados e úteis, por exemplo para executar um ciclo.

Nos exemplo x vale 3.

SÍMBOLO	DESCRIÇÃO	EX.	SIGNIFICADO	RESUL.
x++	incrementação (x++ é o mesmo que x=x+1)	y = x++	3 depois mais 1	4
x--	decrementação (x-- é o mesmo que x=x-1)	y = x--	3 depois mais 1	2

operadores de incrementação

O operadores executam-se na ordem seguinte de prioridade (do grau de prioridade o mais fraco ou grau de prioridade o mais elevado).

No caso dos operadores iguais, da esquerda para a direita.

OPERADOR	OPERAÇÃO
,	vírgula ou separador de lista
= += -= *= /= %=	valorização
? :	operador condicional
	"ou" lógico
&&	"e" lógico
== !=	igualdades
< <= >= >	relações
+ -	adição subtração
* /	multiplicar dividir
! - ++ --	incrementos
()	parêntese

Funções

definição

Uma função é um grupo de linha(s) de código de programação destinado uma tarefa bem específica e que podemos de necessário, utilizar várias vezes. A utilização de funções melhora bastante a leitura do script.

Em Javascript, existem dois tipos de funções:

- As funções próprias do Javascript. Que chamamos de "métodos". Elas são associadas a um objecto bem particular como o caso do método Alert() com o objecto window.
- As funções escritas por vós para executar o vosso script. É a estas que nos vamos interessar agora.

declaração de funções

Para declarar ou definir uma função, utiliza-se a palavra **function**.

A sintaxe de uma declaração de função é a seguinte:

```
function nome_da_função(argumentos) {
... código de instruções ...
}
```


O nome da função segue as mesmas regras do que as das variáveis (número de caracteres indefinido, começado por uma letra pode incluir números...). Volto a lembrar que o Javascript é **case sensitive**. Assim função() não será igual a Função(). Todos os nomes de funções num um script devem ser únicos.

A menção dos argumentos é facultativo mas no caso dos parêntese devem ficar. É alias graças aos parêntese que o interpretador Javascript distinga as variáveis das funções. Voltaremos a falar mais em pormenor sobre os argumentos e outros parâmetros mais a frente.

O facto de definir uma função não vai executar os comandos que ela contém. ~só é executada quando chamamos a função.

invocação de uma função

A invocação de uma função se faz o mas simples possível, pelo nome da função com parêntese).

Seja por exemplo **nome_da_função()**;

Convém verificar (porque o browser lê o script de cima a baixo) que a vossa função deve estar bem definida antes de a chamar.

As funções dentro <HEAD>...</HEAD>

É assim aconselhado de inserir todas as declarações de funções no cabeçalho da página , isto é entre os tags <HEAD>...</HEAD>. assim terão a certeza que as funções já estarão interpretadas antes de serem invocadas no <BODY>.

exemplos

Neste exemplo, definimos dentro dos tags HEAD, uma função chamada mensagem() que insere um texto "Bem vindo a minha página". Esta função será chamada no carregamento da página ver **onLoad=....** no tag <BODY>.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<--
function mensagem() {
document.write("Bem vindo a minha página");
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad="mensagem()" >
</BODY>
</HTML>
```

passar um valor a uma função

Pode-se passar valores ou parâmetros as funções Javascript. Assim as funções podem utilizar valores. Para passar um parâmetro a uma função, fornece-se um nome de uma variável dentro da declaração da função.

Um exemplo simples para compreender. Está escrito uma função que insere uma caixa de aviso em que o texto pode ser alterado.

Na declaração da função, escreve-se:

```
function Exemplo(Texto) {
alert(texto);
}
```

O nome da variável é Texto e é definida como um parâmetro da função.

Na invocação da função, fornece-se o texto:

```
Exemplo("Bom dia a todos");
```

passar vários valores a uma função

Pode-se passar vários parâmetros a uma função. Como é frequente o caso em Javascript, separa-se os parâmetros por vírgulas.

```
function nome_da_função(arg1, arg2, arg3) {  
  ... código de instrução ...  
}
```

Declaração de função:

```
function Exemplobis(Texto1, Texto2){...}
```

Invocação da função:

```
Exemplobis("Bem vindo a minha página", "Bom dia a todos")
```

voltar a um valor

O princípio é simples (a prática por vez não é tão simples). Para reenviar um resultado, basta escrever a palavra chave **return** seguido da expressão a reenviar. Notar que cercar a expressão de parênteses. Por exemplo:

```
function cubo(numero) {  
  var cubo = numero*numero*numero  
  return cubo;  
}
```

A instrução return é facultativa e podemos encontrar vários return na mesma função.

Para explorar este valor da variável reenviada pela função, utiliza-se uma formulação do tipo document.write(**cubo(5)**).

variáveis locais e variáveis globais

Com as funções, o bom uso de variáveis locais e globais tem toda a sua importância.

Uma variável declarada dentro uma função pela palavra chave **var** terá uma invocação limitada a esta própria função. Não se pode assim chama-la fora do script. Chamamos assim variável local.

```
function cubo(numero) {  
  var cubo = numero*numero*numero  
}
```

Assim a variável cubo neste exemplo é uma variável local. Se fazemos referência a ela fora do script, esta variável será desconhecido pelo interpretador Javascript (mensagem de erro).

Se a variável é declarada contextualmente (sem utilizar a palavra var), a sua invocação será global.

```
function cubo(numero) {  
  cubo = numero*numero*numero  
}
```

A variável cubo declarada será aqui uma variável global.

As variáveis declaradas logo no início do script, fora e antes de todas as funções, serão sempre globais, que ela seja declarada com var ou de maneira contextual.

```
<SCRIPT LANGUAGE="javascript">
```

```
var cubo=1
function cubo(numero) {
var cubo = numero*numero*numero
}
</SCRIPT>
```

A variável cubo será global.

Para facilitar a gestão das variáveis, posso aconselhar de as declarar logo no início do script (como a maior parte das linguagens de programação). Este hábito pode prevenir algumas complicações.

Eventos

Geral

Com os eventos e sobretudo sua gestão, abordamos o lado "**mágico**" do Javascript. Em Html clássico, há um evento que conhecem bem. É o clique do rato sobre um link que vai abrir outra página Web. Infelizmente, é praticamente o único. Felizmente, o Javascript vai acrescentar uma boa dezena.

Os eventos Javascript, associados as funções, aos métodos e aos formulários, abrem uma grande porta para uma verdadeira **interactividade** das páginas.

Eventos

Vamos ver os diferentes eventos implementados em Javascript.

EVENTOS	DESCRIÇÃO
Clik	Quando o utilizador clica sobre um botão, um link ou outro elementos.
Load	Quando a página é carregada pelo browser.
Unload	Quando o utilizador saia da página.
MouseOver	Quando o utilizador coloca o ponteiro do rato sobre um link ou outro elemento.
MouseOut	Quando o ponteiro do rato não está sobre um link ou outro elemento.
Focus	Quando um elemento de formulário tem o focus, istoé, que está activo.
Blur	Quando um elemento de formulário perde o focus, isto é, quando o deixa de estar activo.
Change	Quando o valor de um campo de formulário é modificado.
Select	Quando o utilizador selecciona um campo dentro de elemento de formulário.
Submit	Quando o utilizador clica sobre o botão <i>Submit</i> para enviar um formulário.

Gestão de eventos

Para ser eficaz, é necessário que a estes eventos sejam associados as acções previstas por ti. É o papel de gestão de eventos. A sintaxe é:

```
onevento="function()"
```

Por exemplo, `onClick="alert('Bem vindo ao Truques e Dicas')"`.

De maneira literária, no clicar do utilizador, abrir uma caixa de aviso com a mensagem indicada.

onClick

Evento mais clássico em informática, o clique do rato.



O código deste botão é:

```
<FORM>
<INPUT TYPE="button" VALUE="Clicar" onClick="alert('Acabas-te de clicar no botão')">
</FORM>
```

Mais pormenores sobre os formulários no próximo capítulo.

onLoad e onUnload

O evento **Load** aparece quando a página acaba de se carregar. O inverso, **Unload** aparece quando o utilizador saia da página.

Os eventos onLoad e onUnload são utilizados sob forma de atributos do tags <BODY> ou <FRAMESET>. Pode-se assim escrever um script para desejar as boas vindas na abertura de uma página e uma pequena palavra de adeus ao sair desta.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE='Javascript'>
function bemvindo() {
alert("Bem vindo a esta página");
}
function adeus() {
alert("Adeus");
}
</SCRIPT>
</HEAD>
<BODY onLoad='bemvindo()' onUnload='adeus()''>
Html normal
</BODY>
</HTML>
```

onmouseover e onmouseout

O evento **onmouseover** executa-se quando o cursor passa por cima (sem clicar) de um link ou de uma imagem. Este evento é bastante prático, por exemplo, para inserir explicações na barra de estado ou mesmo com uma pequena janela tipo info-objecto.

Passar o cursor do rato sobre a palavra exemplo (sem clicar no link).

Exemplo

O evento **onmouseout**, geralmente associado um onmouseover, executa-se quando o cursor saia da zona sensível (link ou imagem).

Falaremos com mais pormenor sobre o onmouseover e o onmouseout na página seguinte.

Eventos (continuação)

onFocus

O evento **onFocus** é quando um objecto se torna activo, isto é, sempre que activamos uma janela, ou uma textfiel que fica pronta para receber texto. Os objectos podem tornar-se activos com o clique do rato ou com o uso da tecla "Tab".

Se clicar na zona do texto, irá se efectuar-se um focus.

onBlur

O evento **onBlur** executa-se quando por exemplo uma text field de um formulário perde o focus. Isto aconteça quando o utilizador desactiva a text field clicando fora dela ou utilizando a tecla "Tab".

Se depois de clicar e/ou escrever na zona do texto, clica-se fora do documento, produza-se o evento Blur.

O código é:

```
<FORM>  
<INPUT TYPE=text onBlur="alert('Isto é um Blur!)" >  
</FORM>
```

Gestão de eventos

Para ser eficaz, é necessário que a estes eventos sejam associados as acções previstas por ti. É o papel de gestão de eventos. A sintaxe é:

```
onEvento="function()"
```

Por exemplo, **onClick="alert('Bem vindo ao Truques e Dicas')"**.

De maneira literária, no clicar do utilizador, abrir uma caixa de aviso com a mensagem indicada.

onchange

Este evento é bastante semelhante ao do onBlur mas com uma pequena diferença. Não só a área de texto deve ter perdido o focus mas também o seu conteúdo deve ter sido alterado pelo utilizador.

onSelect

Este evento executa-se quando o utilizador seleccionou toda ou parte de um texto numa text field.

Gestões de eventos disponíveis em Javascript

Agora apresento uma lista de objectos que correspondem a gestões de evento bem determinado.

OBJECTO	GESTÕES DE EVENTOS DISPONÍVEIS
Janela	onLoad, onUnload
Link hipertexto	onClick, onMouseOver, onMouseOut
Elemento de texto	onBlur, onChange, onFocus, onSelect
Elemento de zona de texto	onBlur, onChange, onFocus, onSelect
Elemento botão	onClick
Botão Radio	onClick
Lista de selecção	onBlur, onChange, onFocus
Botão Submit	onClick
Botão Reset	onClick

Sintaxe do onMouseOver

O código de gestão de evento **onMouseOver** acrescenta-se nos tags do de link :

```
<A HREF="#" onMouseOver="acao()" >link</A>
```

Assim, quando o utilizador passa com o rato sobre o link, a função action() é invocada. O atributo HREF é indispensável. Ele pode conter o endereço de uma página Web se assim o desejares ou simplesmente o cardinal "#" para que o link fica activo mas não abra nenhuma página.

Exemplo: Ao sobrevoar o link "[mensagem importante](#)", uma janela de aviso abra-se.

O código é :

```
<BODY>
...
<A HREF="#" onMouseOver="alert('Truques e Dicas')">mensagem
importante</A>
...
<BODY>
```

Ou se preferes usar os tags <HEAD>

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
function mensagem(){
alert("Truques e Dicas")
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="#" onMouseOver="mensagem()">mensagem importante</A>
</BODY>
</HTML>
```

Sintaxe do onMouseOver

Completamente similar ao onMouseOver, só que o evento se executa quando o cursor do do rato saía do link ou da zona sensível.

Pode-se imaginar o código seguinte:

```
<A HREF="#" onMouseOver="alert('Truques e Dicas')" onMouseOut="alert('Obrigado pela visita')">mensagem importante</A>
```

Exemplo: Ao sobrevoar o link "[mensagem importante](#)", uma janela de aviso abra-se.

Troca de imagens

Com a gestão de eventos `onmouseover`, podemos prever que depois de sobrevoar uma imagem pelo o utilizador, uma outra aparece (*ambas as imagem devem ter o mesmo tamanho*). O código é relativamente simples.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript1.1">
function lightUp() {
document.images["homeButton"].src="button_hot.gif"
}
function dimDown() {
document.images["homeButton"].src="button_dim.gif"
}
</SCRIPT>
</HEAD>
<BODY>
<A HREF="#" onMouseOver="lightUp();" onMouseOut="dimDown();">
<IMG SRC="button_dim.gif" name="homeButton" width=100 height=50
border=0> </A>
</BODY>
</HTML>
```

Completem sempre em Javascript os atributos `width=x height=y` das tuas imagens.

Exemplo:



Condições

Expressão if

Num momento ou outro da programação, teremos necessidade de testar uma condição. O que vai permitir executar ou não uma série de instruções.

Na sua formulação a mais simples, a expressão **if** apresenta-se assim:

```
if (condição verdadeira) {
uma ou várias instruções;
}
```

Assim, se a condição é verdadeira, as instruções executam-se. Se ela não é, as instruções não se

executam-se e o programa passa para o comando seguinte.

De maneira um pouco mais evoluídos, tem-se a expressão **if...else**

```
if (condição verdadeira) {  
  instrução1;  
}  
else {  
  instrução2;  
}
```

Se a for verdadeira (true), o bloco de instruções 1 se executa. Se ela não for (false), o bloco de instruções 2 se executa.

Graça aos operadores lógicos "e" e "ou", a expressão de teste pode testar uma associação de condições. Assim **if ((condição1) && (condição2))**, testará se a condição 1 e a condição 2 é realizada. E **if ((condição1) || (condição2))**, testará se uma ao menos das condições é verdadeira.

Para ser mais completo (e para aqueles que gostam da escrita concisa), também há:

```
(expressão) ? instrução a : instrução b
```

Se a expressão entre parêntese é verdadeira, a instrução **a** é executada. Se a expressão entre parêntese volta falsa, é a instrução **b** que se executa-se.

Expressão for

A expressão **for** permite de executar um bloco de instruções um certo número de vez em função da realização de um certo critério. Sua sintaxe é:

```
for (valor inicial ; condição ; progressão) {  
  instruções;  
}
```

Pegamos um exemplo concreto

```
for (i=1, i<10, i++) {  
  document.write(i + "<BR>")  
}
```

Na primeira passagem, a variável **i**, é inicializada a 1. Sendo a variável inferior a 10. Ela é então incrementada de uma unidade pelo operador de incrementação **i++**

(i vale então 2) e as instruções executam-se.

No fim da execução das instruções, voltamos ao contador. A variável **i** (que vale 2) é ainda inferior a 10. Ela é aumentada de 1 e as instruções prosseguem, até que **i** vale 10. A variável **i** não satisfaz mais a condição **i<10**. O ciclo interrompa-se e o programa continua depois da chaveta fechada

While

A instrução **while** permite de executar uma instrução (ou um grupo de instruções) um certo número de vezes.

```
while (condição verdadeira){  
  continuar a fazer alguma coisa  
}
```

Enquanto que a condição é verdadeira, o Javascript continua executar as instruções entre as chavetas. Uma vez que a condição não mais verdadeira, o ciclo interrompa-se e continua-se o script.

Pegamos neste exemplo:


```

cont=1;
while (cont<5) {
document.write ("linha : " + cont + "<BR>");
cont++;
}
document.write("fim do ciclo");

```

Vejamos como funciona este exemplo. Primeiro a variável que irá servir de contador **cont** é inicializada em 1. o ciclo **while** arranca então com o valor 1 que é inferior a 5. A condição é verdadeira. Executa-se as instruções das chavetas.

Primeiro, "linha : 1" é inserido e depois o contador é incrementado de 1 e toma assim o valor 2. A condição ainda é verdadeira. as instruções entre as chavetas são executadas. E isso até inserir a linha 4. Aí, o contador depois da incrementação vale 5. A condição já não sendo verdadeiro, continuamos no script e é então o fim do ciclo.



Atenção ! Com o sistema de ciclo, o risco existe (se a condição é sempre verdadeira), de fazer o ciclo indefinidamente a 'instrução. O que pode fazer *crachar* o browser !

Break

A instrução **break** permite interromper prematuramente um ciclo **for** ou **while**.

Para ilustrar isto, retomamos o nosso exemplo:

```

cont=1;
while (cont<5) {
if (cont == 4)
break;
document.write ("linha : " + cont + "<BR>");
cont++;
}
document.write("fim do ciclo");

```

O funcionamento é semelhante ao exemplo precedente só que quando o contador vale 4. Nesse momento, o **break** faz-nos sair do ciclo e "fim do ciclo" é inserido.

O resultado no écran será:

```

linha : 1
linha : 2
linha : 3
fim do ciclo

```

Continue

A instrução **continue** permite de saltar uma instrução num ciclo **for** ou **while** e de continuar de seguida com o ciclo (sem sair deste como faz o **break**).

Retomamos o nosso exemplo ;

```

cont=1;
while (cont<5) {
if (cont == 3){
cont++
continue;}
document.write ("linha : " + cont + "<BR>");
cont++;
}
document.write("fim do ciclo");

```

Aqui, o ciclo começa, quando o contador vale 3, devido a instrução **continue**, salta-se a instrução **document.write** (linha : 3 não é afixada) e continua-se com o ciclo. Nota-se que tivemos de inserir **cont++** antes **continue**; para evitar um ciclo infinito e crachar o browser (cont fica em 3).

O resultado no écran será:

```

linha : 1
linha : 2
linha : 4
fim do ciclo

```

Formulários

Em geral

Com o Javascript, os formulários Html tomem outra dimensão. Não esquecer que em Javascript, podemos aceder a cada elemento de um formulário para, por exemplo, ler ou escrever um valor, uma escolha a qual poderá associar uma gestão de evento... Todos esses elementos irão reforçar as capacidades interactivas da página.

Um formulário é um elemento Html declarado pelo tags **<FORM></FORM>**. Um formulário contém um ou vários elementos que chamamos os controladores (widgets). Esses controladores são escrito por exemplo pelo tags **<INPUT TYPE= ...>**.

Declaração de um formulário

A declaração de um formulário faz-se pelos tags **<FORM>** e **</FORM>**. Pode-se notar que em Javascript, o atributo **NAME="nome_do_formulário"** tem toda a sua importância para designar o caminho completo dos elementos. Em suma, os atributos **ACTION** e **METHOD** são facultativos pelo menos até não fazer apelo ao servidor.

Um erro clássico e em Javascript é de esquecer de declarar o fim do formulário **</FORM>** depois de ter inserir um controlador.

Controlador de linha de texto

A zona de texto é o elemento de entrada/saída por excelência do Javascript. A sintaxe Html é **<INPUT TYPE="text" NAME="nome" SIZE=x MAXLENGTH=y>** para uma área de texto de uma só linha, de comprimento x e de comprimento máximo y.

O objecto texto tem 3 propriedades:

PROPRIEDADES	DESCRIÇÃO
name	indica o nome do controlador pelo que podemos aceder.
defaultValue	indica o valor por defeito que será afixada na zona de texto.

value indica que o valor da zona de texto. Seja ele escrito pelo utilizador ou já ter um valor por defeito.

Ler um valor numa zona de texto

Um exemplo que iremos pormenorizar:

Insira um valor e carregue no botão para controla-lo.

O script completo é assim:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript">
function controle(form1) {
var test = document.form1.input.value;
alert("Escreveste: " + test);
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="form1">
<INPUT TYPE="text" NAME="input" VALUE=""><BR>
<INPUT TYPE="button" NAME="botao" VALUE="Controlar"
onClick="controle(form1)" >
</FORM>
</BODY>
</HTML>
```

Quando clica-se no botão "controlar", o Javascript chama a função **controle()** no qual passamos o formulário em que o nome é **form1** como argumento.

Esta função **controle()** definida nos tags **<HEAD>** Toma sob a variável **test**, o valor da zona de texto. Para aceder a este valor, escreve-se o caminho completo deste (ver o capítulo "[Um pouco de teoria objecto](#)"). No documento presente, tem o objecto formulário chamado form1 que contém o controlador de texto nomeado input e que tem como propriedade o elemento de valor **value**. O que dá **document.form1.input.value**.

Escrever um valor na zona de texto

Inserir um valor qualquer na zona do texto de entrada. Carregar sobre o botão para inserir este valor na zona de texto saída.

Zona de texto de entrada

Zona de texto de saída

Código:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="javascript">
function afixar(form2) {
var testin =document. form2.input.value;
document.form2.output.value=testin
}
</SCRIPT>
```

```

</HEAD>
<BODY>
<FORM NAME="form2">
<INPUT TYPE="text" NAME="input" VALUE=""> Zona de texto de entrada<BR>
<INPUT TYPE="button" NAME="botao" VALUE="Afixar" onClick="afixar(form2)"><BR>
<INPUT TYPE="text" NAME="output" VALUE=""> Zona de texto de saída
</FORM>
</BODY>
</HTML>

```

Ao clicar no botão "Afixar", o Javascript chama a função a função **afixar()** na qual passa o formulário em que o nome é **form2** desta vez como argumento.

Esta função **afixar()** definida nos tags <HEAD> toma sob a variável **testin**, o valor da zona de texto de entrada. À instrução seguinte, dizemos ao Javascript que o valor da zona de texto **output** contida no formulário nomeado **form2** é a da variável **testin**. Utiliza-se o caminho completo para chega a propriedade valor do objecto desejado, seja em Javascript **document.form2.output.value**.

Formulários (continuação)

Os botões radio

Os botões radio são utilizados para escolher uma escolha, e só uma, entre um conjunto.

PROPRIEDADES	DESCRIÇÃO
name	indica o nome do controlador. Todos os botões tem o mesmo nome.
index	o index ou o conjunto de botões radio começando por 0.
checked	indica o estado em curso do elemento radio
defaultchecked	indica o estado do botão seleccionado por defeito.
value	indica o valor do elemento radio.

Exemplo :

```

<HTML>
<HEAD>
<SCRIPT language="javascript">
function escolhaprop(form3) {
if (form3.escolha[0].checked) { alert("Escolheste a proposição " +
form3.escolha[0].value) };
if (form3.escolha[1].checked) { alert("Escolheste a proposição " +
form3.escolha[1].value) };
if (form3.escolha[2].checked) { alert("Escolheste a proposição " +
form3.escolha[2].value) };
}
</SCRIPT>
</HEAD>
<BODY>
Escolha :
<FORM NAME="form3">
<INPUT TYPE="radio" NAME="escolha" VALUE="1">Escolha número 1<BR>
<INPUT TYPE="radio" NAME="escolha" VALUE="2">Escolha número 2<BR>
<INPUT TYPE="radio" NAME="escolha" VALUE="3">Escolha número 3<BR>
<INPUT TYPE="button"NAME="but" VALUE="Qual é vossa escolha ?"
onClick="escolhaprop(form3)" >
</FORM>
</BODY>

```

</HTML>

Clica na tua escolha :

- Escolha número 1
- Escolha número 2
- Escolha número 3

No formulário nomeado **form3**, declara-se 3 botões radio. De notar que utiliza-se o mesmo nome para os três botões. De seguida temos um botão que invoca a função `escolhaprop()`.

Esta função teste qual o botão radio é checkado. Acede-se aos botões sob a forma de índice em relação ao nome dos botões radio que seja **escolha[0]**, **escolha[1]**, **escolha[2]**.

Teste-se a propriedade **checked** do botão por `if(form3.escolha[x].checked)`. Depois de escolheres e carregar no botão "Qual a vossa escolha", uma caixa de aviso aparece com a indicação do botão checkado. Esta mensagem toma o valor ligado a cada botão pelo caminho **form3.escolha[x].value**.

Checkbox

Os botões checkbox são úteis para escolher uma ou várias opções (relembro que os botões radio são utilizados para escolher só uma opção). Sua sintaxe e seu uso é bastante semelhante aos botões radio, excepto em relação ao atributo **name**.

PROPRIEDADES	DESCRIÇÃO
name	indica o nome do controlador. Todos os botões tem o mesmo nome.
checked	indica o estado em curso do elemento radio
defaultchecked	indica o estado do botão seleccionado por defeito.
value	indica o valor do elemento radio.

Check as vossas escolhas :
escolha os números 1,2 e 4 para obter uma boa resposta.

- Escolha número 1
- Escolha número 2
- Escolha número 3
- Escolha número 4

<HTML>

<HEAD>

<script language="javascript">

```
function resposta(form4) {
if ( (form4.check1.checked) == true && (form4.check2.checked) == true &&
(form4.check3.checked) == false && (form4.check4.checked) == true)
{ alert("É a boa respostas! ") }
else
{alert("Errado! continua a tentar.")}
}
</SCRIPT>
```

</HEAD>

<BODY>

Check as vossas escolhas:

<FORM NAME="form4">

<INPUT TYPE="CHECKBOX" NAME="check1" VALUE="1">Escolha número 1


```

<INPUT TYPE="CHECKBOX" NAME="check2" VALUE="2">Escolha número 2<BR>
<INPUT TYPE="CHECKBOX" NAME="check3" VALUE="3">Escolha número 3<BR>
<INPUT TYPE="CHECKBOX" NAME="check4" VALUE="4">Escolha número 4<BR>
<INPUT TYPE="button"NAME="but" VALUE="Corrigir" onClick="resposta(form4)" >
</FORM>
</BODY>
</HTML>

```

No formulário nomeado form4, declara-se 4 checkbox. De referir que utiliza-se um nome diferente para as quatro checkbox. De seguida temos um botão que irá invocar a função resposta(). Esta função teste quais botões que foram checkados. Para obter uma boa resposta, tem de se checkar o 1,2 e 4. Acede-se aos botões utilizando de cada vez os seus nomes. Teste-se a propriedade **checked** do botão por (**form4.nome_da_botão.checked**). Ao carregar no botão "Corrigir", no caso de positiva, uma caixa de aviso aparece que indica que está certo, em caso de negativo a caixa de aviso convida a tentar de novo.

Lista de selecção

OS botões da checkbox são utilizados para escolher uma ou várias opções. Sua syntax e seu uso é semelhante aos do botão radio excepto em relação ao atributo **name**.

PROPRIEDADES	DESCRIÇÃO
name	indica o nome da lista.
length	indica o número de elementos da lista.
selectedIndex	indica o lugar a partir do 0 do elemento da lista que foi seleccionado pelo utilizador.
defaultselected	indica o elemento da lista seleccionado por defeito; é aquele que aparece então na lista em primeiro.

Um pequeno exemplo :

escolha um elemento da lista:

```

<HTML>
<HEAD>
<script language="javascript">
function lista(form5) {
alert("O elemento " + (form5.list.selectedIndex + 1));
}
</SCRIPT>
</HEAD>
<BODY>
Escolha um elemento: : <FORM NAME="form5">
<SELECT NAME="list">
<OPTION VALUE="1">Elemento 1
<OPTION VALUE="2">Elemento 2
<OPTION VALUE="3">Elemento 3
</SELECT>
<INPUT TYPE="button"NAME="b" VALUE="Qual é o elemento escolhido?"
onClick="lista(form5)" > </FORM>
</BODY>
</HTML>

```

No formulário nomeado **form5**, declara-se uma lista de selecção pelos tags **<SELECT></SELECT>**. Entre estes dois tags, declara-se os diferentes elementos da lista pelo tag **<OPTION>**. Depois temos um botão que invoca a função **lista()**.

Esta função teste qual a opção foi seleccionada. o caminho completo do elemento seleccionado é **form5.nome_da_lista.selectedIndex**. Como o **index** comece em 0, acrescenta-se 1 para encontrar a linha elemento.

Formulários (continuação)

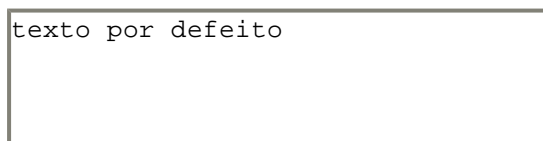
O controlador textarea

O objecto textarea é uma zona de texto de várias linhas.

A sintaxe Html é :

```
<FORM>  
<TEXTAREA NAME="nome" ROWS=x COLS=y>  
texto por defeito  
</TEXTAREA>  
</FORM>
```

onde **ROWS=x** representa o número de linhas e **COLS=y** representa o número de colunas.



O objecto textarea possui várias propriedades :

PROPRIEDADES	DESCRIÇÃO
name	indica o nome do controlador pelo qual pode-se aceder.
defaultValue	indica o valor por defeito que será afixado na zona do texto.
value	indica o valor a decorrer na zona de texto. Seja ela escrito pelo utilizador ou se este não escreveu nada, o valor por defeito.

A estas propriedades, deve-se acrescentar o `onFocus()`, `onBlur()`, `onSelect()` e `onChange()`.

Em Javascript, utiliza-se `\r\n` para quebra de linha.

Como por exemplo na expressão `document.Form.Text.value = 'Check\r\nthis\r\nout'`.

O controlador Reset

O controlador **Reset** permite de anular as modificações efectuadas aos controladores de um formulário e de restaurar os valores por defeito.

A sintaxe Html é:

```
<INPUT TYPE="reset" NAME="nome" VALUE="texto">
```

Onde **VALUE** dá o texto do botão.

Um único método associado ao controlador **Reset**, é o método **onClick()**. O que pode servir, por exemplo, para afixar um outro valor do que o por defeito.

O controlador Submit

O controlador tem a tarefa específica de transmitir todas as informações contidas no formulário ao URL designado no atributo **ACTION** do tag **<FORM>**.

A sintaxe Html é:

```
<INPUT TYPE="submit" NAME="nome" VALUE="texto">
```

Onde **VALUE** dá o texto do botão.

Um único método associado ao controlador **Submit**, é o método **onClick()**.

O controlador Hidden (escondido)

O controlador **Hidden** permite entrar no script dos elementos (geralmente dados) que não aparecem no écran. Estes elementos estão escondidos. Daí seu nome.

A sintaxe Html é:

```
<INPUT TYPE="hidden" NAME="nome" VALUE="os dados escondidos ">
```

Envio de um formulário por E-mail

Javascript, não permite escrever num ficheiros. Depois, o controlador **Submit** é sobretudo destinado aos CGI. A solução é o envio do formulário via correio electrónico.

A sintaxe é :

```
<FORM METHOD="post" ACTION="mailto:vosso_endereço_Email">  
<INPUT TYPE=text NAME="nome"> <br>  
<TEXTAREA NAME="mensagem" ROWS=2 COLS=35>  
</TEXTAREA> <br>  
<INPUT TYPE=submit VALUE="Submit">  
</FORM>
```

O que dá:



The image shows a rendered HTML form. It consists of a small rectangular text input field at the top left. Below it is a larger rectangular text area with a thin border. At the bottom left of the form is a button with the text "Submit" on it.

E recebemos no e-mail, algo assim:

nome=Sérgio+Brandão&mensagem=olá!+Bem+vindo+ao+Truques+e+Dicas.

Onde encontramos os campos **nome=** e **mensagem=** separados pelo símbolo **&**, os espaços são substituídos por + , e **17%OD%OA** corresponde a uma quebra de linha.

Um pouco de tudo

Caixas de Diálogo ou de Mensagem

Javascript coloca a vossa disposição 3 caixas de mensagens

- **alert()**
- **prompt()**
- **confirm()**

São as três métodos do objecto window.

O método alert()

O método **alert()** já deve ser-te familiar, visto que já o utilizamos frequentemente ao longo do tutorial.

O método **alert()** afixa uma caixa de diálogo na qual é reproduzido o valor (variável e/ou cadeia de caracteres) do argumento que lhe foi fornecido. Esta caixa bloqueia o programa até que o utilizador não terá clicado em "OK".

Sua sintaxe é:

```
alert(variável);
alert("cadeia de caracteres");
alert(variável + "cadeia de caracteres");
```



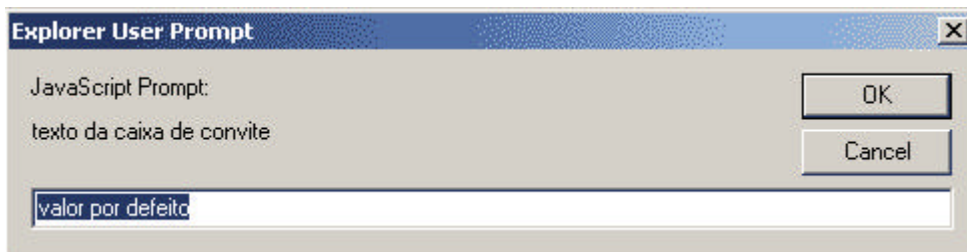
Para escrever em várias linhas, deves usar: `\n`.

O método prompt()

No mesmo estilo do que o método `alert()`, Javascript dispõe de uma outra caixa de diálogo, no caso presente é chamada caixa de convite, que é composto por um campo que contém uma entrada a completar pelo utilizador. Esta entrada contém um valor por defeito.

A sintaxe é:

```
prompt("texto da caixa de convite", "valor por defeito");
```



Ao clicar no OK, o método reenvia o valor escrito pelo utilizador ou a resposta proposta por defeito. Se utilizador clica em Cancel, o valor nulo é então reenviado.

`Prompt()` é porvez utilizado para incrementar dados fornecidos pelo utilizador.

O método confirm()

Este método afixa 2 botões o "OK" e "Cancel". Ao clicar no OK, `confirm()` reenvia o valor `true` e obviamente `false` caso clicar em Cancel. Esta método é utilizado sobretudo para confirmar uma opção.



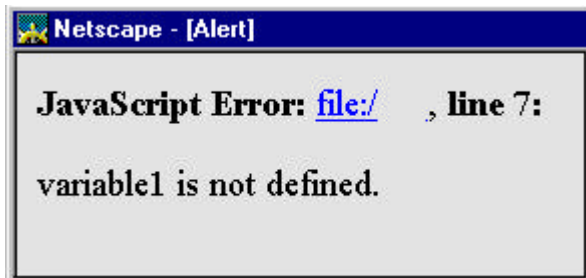
</FORM>

Em vez de usar `escolhaprop(form3)`, pode-se utilizar `escolhaprop(this.form)` e evitar assim toda confusão com os outros nomes dos formulários. Neste exemplo, **this.form** refere-se ao formulário `form3` completo. Enquanto que, **escolhaprop(this)** iria referir-se só ao elemento tipo botão do `form3`.

Para ser completo, `this` é utilizado também para criar uma ou várias propriedades de um objecto. Assim, para criar um objecto livro com as propriedades `autor`, `editor` e `preço` esta operação pode ser efectuada com a ajuda de uma função :

```
function livro(autor, editor, preço) {  
  this.auteur = autor;  
  this.editeur = editor;  
  this.prix = preço;  
}
```

As mensagens de erros



Pode-se dizer que as ferramentas do debug oferecidas pelo Javascript seja das mais elevadas.

Pode,os estar orgulhoso da precisão dada pelo interpretador. O famoso "**Javascript Error line x**".

Os tipos de erro

Existem 3 grandes categorias de erros na utilização de um programa Javascript :

- os erros no carregamento.
- os erros na execução.
- os erros de lógico.

Os erros no carregamento

No carregamento do script pelo browser, Javascript passa em revista os diferentes erros que podem o impedir o funcionamento deste.

Os erros no carregamento, são frequentes devido a erros de tecla e/ou erros de sintaxe. Para ajudar a determinar o erro, Javascript afixa caixa de mensagem de erro, que indica o erro e o texto com erro. É de notar que nem sempre o erro indicado pelo Javascript não é o erro verdadeiro, é sim o local onde o erro ocorre, o erro pode estar bem antes. Dos exemplos clássicos de erros no carregamento são os parênteses, chavetas e aspas não fechadas, etc.

Os erros na execução

Aqui o script carrega sem problema, mas a caixa de erro aparecem quando o script é invocado. Enquanto que os erros no carregamento aparecem devido a enganos na

sintaxe, os erros na execução aparecem devido ao mau uso dos comandos e de objectos Javascript.

Um exemplo de erros na execução é a invocação de uma variável ou uma função inexistente (isto acontece quando nos enganamos no nome da variável ou da função).

Os erros lógicos

Este são os mais viciados porque o "debugge" do Javascript não indica nenhum erro e o script funciona correctamente, Só que o resultado esperado não é o esperado.

A única solução é rever a construção lógica do script.

Numerosos erros lógicos são devido aos valores de variáveis incorrecto.

Alguns concelhos :

- No caso em que o utilizador deve inserir um valor, esta deve estar no bom formato. Pode-se criar um pequeno script que verifica o formato de entrada.
- Pode-se inserir ponto de controlo de valor de variável ou de passagem com a instrução `alert(variável)` ou `alert("Ponto de passagem1")`.

Os grande clássicos de erros

Pode-se afixar uma lista de erros que todos os iniciados fazem ou irão fazer fazer mais tarde.

- Deves ser vigilante ao nome das variáveis (case sensitive). Assim Variável e variável são duas variáveis distintas.
- O nome da função tem de ter a mesma ortografia na declaração e na invocação. O nome das funções tem de ser único no script
- Não esquecer as aspas ou as apóstrofes antes e depois de caracteres.
- Deve-se inserir vírgulas entre os diferentes parâmetros ou argumentos.
- Deve colocar as chavetas no sítio certo sem esquecer de os fechar.
- Assegura-te que os nomes dos objectos Javascript estão correctos. A armadilha é que os objectos Javascript comecem por uma maiúscula (Date, Math, Array...) mas as propriedades comecem por minúscula (alert).
- A confusão entre = operador de cálculo e == operador de comparação.

Mini FAQ

Nenhuma mudança após a modificação do script

Meu script não funciona numa tabela

Adaptar o script segundo o browser do leitor

Arredondar os números atrás da vírgula

Vê-se o código do meu Javascript!

Transmitir variáveis de uma página para outra

Os botões radio reenviem-me a ordem inverso

Nenhuma mudança após a modificação do script

Irá certamente acontecer após uma modificação (correção) do vosso script, e que nenhuma mudança seja visível no écran mesmo após de fazer fazer "Reload" ou "Actualizar".

- Verificar se gravaste bem as tuas modificações (isto acontece mesmo aos melhores).

- É necessário porvez recarregar mais profundamente a tua página, Shift + Reload em Netscape ou clicar na zona de localização do browser e fazer Enter.

Meu script não funciona numa tabela

Javascript nas tabelas, não nenhuma história de amor (bug?). Recomenda-se na literatura de não colocar tags <SCRIPT> nos tags <TD> mas sim de começar o tag <SCRIPT> antes do tag <TD> e de escrever o tag <TD> até o tag </TD> usando o objecto **document.write**. Assim fica:

```
<SCRIPT LANGUAGE="Javascript">
<!--
document.write("<TABLE BORDER=1>");
document.write("<TR>");
document.write("<TD>");
document.write("vosso texto");
document.write("</TD>");
document.write("<TD>");
document.write("vosso texto");
document.write("</TD>");
document.write("</TR>");
document.write("</TABLE>");
//-->
</SCRIPT>
```

Adaptar o script segundo o browser do leitor

Com os métodos e propriedades do objecto navegador (ver este capítulo). Há maneira de detectar o tipo e a versão do browser. O que é muito útil para adptar os scripts ao browser e a versão deste.

A compatibilidade das páginas Javascript com os diferentes tipos e versões em circulação coloca alguns problemas.

```
<SCRIPT LANGUAGE = "JavaScript">
<!--
var name = navigator.appName ;
if (name == 'Microsoft Internet Explorer') {
document.write('Atenção! estás a utilizar o Microsoft Explorer 3.0.') <BR>');
document.write('Com este browser, alguns scripts podem não corer
correctamente');
}
else { null }
//-->
</SCRIPT>
```

Arredondar os números atrás da vírgula

Pode acontecer que o Javascript afixa uma divisão do tipo 1.5999999999999999. O que não é agradável. Falaremos sobre este assunto mais aprofundamente no capítulo [Math](#). Entre os diferentes sistemas possíveis, propõe este:

```
variable= Math.round (variable*100)/100
```

Assim, 1.599999 é multiplicado por 100 o que faz 159.9999. O método Math.round (159.9999) dá 160, que divide por 100 que faz por sua vez 1.60. Com ...*100)/100, obtemos 2 números depois da vírgula.

Vê-se o código do meu Javascript!

Pois é, pelo "View Document Source", o leitor pode ver, estudar e copiar o código Javascript incluído nas vossas páginas html. Existe na net vários pequenos programas de codificação de scripts, que invoca ficheiros dissimulados, etc. Chegou-se a conclusão que nenhum sistema poderá garantir a 100% a confidencialidade dos scripts.

Transmitir variáveis de uma página para outra

As variáveis são definidas no script da entidade que constitui a página Web. Caso deseje-se continuar a utilizar estas variáveis numa outra página ou em todo o site. Como fazer? A solução é utilizar frames. O Javascript permita de passar variáveis de uma para objectos que pertencem a uma outra frame. E como todos os browsers Javascript admitem frames, porquê não as utilizares (ver um dos capítulos seguinte).

Os botões radio reenviem-me a ordem inverso

Este bug é próprio do Netscape 2, pode causar supresas desagradáveis, por exemplo num formulário que utiliza botões radio. Caso que o botão radio 1 de uma série de 3 é checked, é o valor 3 que é reenviado. Imagina, o cliente encomendou uma camisa larga apesar de ele desejar uma pequena!
Para corrigir este bug, basta acrescentar um getor de eventos vazio em cada controlo da série do botão radio.

```
<FORM NAME="radioTest">  
<INPUT TYPE="radio" NAME="test" VALUE="A" onClick="">A  
<INPUT TYPE="radio" NAME="test" VALUE="B" onClick="">B  
<INPUT TYPE="radio" NAME="test" VALUE="C" onClick="">C  
</FORM>
```

Objecto Window

Propriedades e métodos do objecto window

Algumas propriedades e métodos do objecto window não te são desconhecidas :
- as das caixas de diálogo. Seja **alert()**, **confirm()**, e **prompt()**,
- e as do temporizador. Seja **setTimeout()** e **clearTimeout()**.

Uma outra série relacionada com as frames :
- são **frames[]**, **length**, **parent**, **opener** e **top**.

Uma série em relação a barra de estado :
- são **status** e **defaultStatus**.

Uma série para a abertura e fecho de uma janela :
- são **open()** e **close()**.

E por fim **self** que reenvia a janela aberta.

Utilização da barra de estado

Com o Javascript, a barra de estado pode ser utilizado para afixar mensagens que desejas. Como eu sou míope como uma toupeira, não é a minha parte preferida do Javascript mas é uma opinião das mais subjectivas.

As propriedades utilizadas são:

PROPRIEDADES	DESCRIÇÃO
status	valor do texto afixado na barra de estado da janela.
defaultStatus	valor por defeito que aparece na barra de estado.

Geralmente, estes evento é accionado por um `onmouseover()` sobre um link.

Exemplo :

```
<HTML>
<BODY>
<A HREF="#" onmouseover="self.status='Vosso texto'; return true;"> Barra de estado</A>
</BODY>
</HTML>
```

Barra de estado
(sobrevoar)

É indispensável de acrescentar **return true;**

Abertura e fecho de uma janela (teoria)

Os métodos utilizados são:

MÉTODOS	DESCRIÇÃO
<code>open()</code>	abra uma nova janela.
<code>close()</code>	fecha a janela activa.

A sintaxe é:

```
[window.]open("URL","nome_da_janela","caracteristicas_da_janela")
```

Onde **URL** é o URL da página que desejamos abrir na nova janela.

Onde **caracteristicas_da_janela** é uma lista de alguma ou de todas as característica da janela seguinte, separadas por vírgulas e **sem espaços nem passagem de linha**.

CARACTERÍSTICA	DESCRIÇÃO
<code>toolbar=yes</code> ou <code>no</code>	Visualização da barra de ferramentas
<code>location=yes</code> ou <code>no</code>	Visualização de campo de endereço (ou de localização)
<code>directories=yes</code> ou <code>no</code>	Visualização dos botões de acesso rápido
<code>status=yes</code> ou <code>no</code>	Visualização da barra de estado
<code>menubar=yes</code> ou <code>no</code>	Visualização da barra de menus
<code>scrollbars=yes</code> ou <code>no</code>	Visualização das barras de desfilamento. (<code>scrollbars=no</code> funciona mal sob Explorer 3.0)
<code>resizable=yes</code> ou <code>no</code>	Possibilidade de modificar as dimensões da janela
<code>width=x</code> em pixels	Largura da janela em pixeis
<code>height=y</code> em pixels	Altura da janela em pixeis

Pode-se utilizar **1** e **0** em vez de **yes** e **no**.

Nota:

Esta nova janela vai aparecer em qualquer sitio no vosso écran. Mas pode-se decidir o local exacto onde a janela vai aparecer.

O uso de novas janelas é bastante simples com o Javascript para afixar informação suplementar sem sobrecarregar a página inicial.

Abertura com um botão (usando onClick)

Vamos abrir uma pequena janela que irá afixar o ficheiro test.htm com um botão na página.

Ficheiro test.htm :

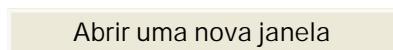
```
<HTML>
<BODY>
<H1>Isto é um teste</H1>
<FORM>
<INPUT TYPE="button" value= " Continuar " onClick="self.close()" >
</FORM>
</BODY>
</HTML>
```

onde **self.close()** fecha a janela corrente, isto é, a nova janela.

Na página inicial:

```
<FORM>
<INPUT TYPE ="button" value="Abrir une nova janela"
onClick="open('test.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')">
(sem espaços nem passagem de linha)
</FORM>
```

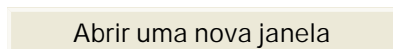
xxxxx

**Abertura com um botão** (chamando uma função)

Na página inicial:

```
<SCRIPT LANGUAGE="javascript">
<!--
function new_window() {
xyz="open('test.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')"
// sem espaços nem passagem de linha
}
// -->
</SCRIPT>

<FORM>
<INPUT TYPE ="button" value="Abrir uma nova janela"
onClick="new_window()" >
</FORM>
```

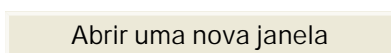
**Objecto Window (continuação)****Fecho da janela após alguns segundos**

Com este script, sem intervenção do utilizador, a nova janela se fecha automaticamente após 4 segundos. Ao clicar no botão, o utilizador interrompe a contagem decrescente que ia fechar a janela e terá de a fechar ele mesmo. Com este sistema, temos a certeza que a nova janela será fechada.


```
<HTML>
<BODY onLoad='compt=setTimeout("self.close()";,4000)'>
<H1>Isto é um teste</H1>
<FORM>
<INPUT TYPE="button" value=" Continuar "
onClick="clearTimeout(compt);self.close();" >
</FORM>
</BODY>
</HTML>
```

Na página inicial :

```
<FORM>
<INPUT TYPE = "button" value = "Abrir uma nova janela"
onClick = "open('testc.htm','new','width=300,height=150,toolbar=no,
location=no,directories=no,
status=no,menubar=no,scrollbars=no,resizable=no')">
(sem espaços nem passagem de linha)
</FORM>
```



Abertura clicando num link ou numa imagem

Acrescenta-se simplesmente "onClick=open..." ao tag <A> do link ou da imagem.

Na página inicial, temos:

```
<A HREF="#" onClick="open('testc.htm', '',
'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=
no,scrollbar=no,resizable=no')">link de teste</A> (sem espaços nem passagem de linha)
```

link de teste

(clicar)

Abertura sobrevoando um link ou uma imagem

Acrescenta-se simplesmente "onMouseOver=open..." ao tag <A> do link ou da imagem.

Na página inicial, temos:

```
<A HREF="#" onMouseOver="open('testc.htm', '',
'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=
no,scrollbar=no,resizable=no')">link de teste</A> (sem espaços nem passagem de linha)
```

link de teste

(sobrevoar)

Abertura sobrevoando o link e fecho saíndo do link

aqui utiliza-se **onmouseover** e **onmouseout**. De relembrar, que o onmouseover é do Javascript 1.1 e não funciona então com o Explorer 3.0.

Na página inicial, temos:

```
<A HREF="#" onMouseOver="open('testc.htm', '',
'width=300,height=150,toolbar=no,location=no,directories=no,status=no,menubar=
no,scrollbar=no,resizable=no')" onmouseout="self.close()">link de teste</A>
```

(sem espaços nem passagem de linha)

link de teste

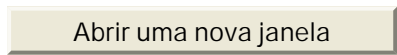
(sobrevoar)

Abertura com um botão (chamando uma função)

Na página inicial:

```
<SCRIPT LANGUAGE="javascript">
<!--
function new_window() {
xyz="open('test.htm', 'new', 'width=300,height=150,toolbar=no,location=no,
directories=no,status=no,menubar=no,scrollbars=no,resizable=no')"
// sem espaços nem passagem de linha
}
// -->
</SCRIPT>

<FORM>
<INPUT TYPE="button" value="Abrir uma nova janela"
onClick="new_window()" >
</FORM>
```



Objecto String

Em geral

Voltemos ao objecto String para nos interessar a manipulação dos caracteres tão úteis para o aspecto da programação do Javascript.

INSTRUÇÃO	DESCRIÇÃO
length	É um inteiro que indica o comprimento da cadeia de caracteres.
charAt()	Método que permita de aceder a um caractere isolado de uma cadeia.
indexOf()	Método que reenvia a posição de uma cadeia parcial a partir de uma posição determinada. (começando do início da cadeia principal seja na posição 0).
lastIndexOf()	Método que reenvia a posição de uma cadeia parcial a partir de uma posição determinada. (começando do FIM da cadeia principal seja na posição 1).
substring(x,y)	Método que reenvia uma String parcial situada entre a posição x e a posição y-1.
toLowerCase()	Transforma todas as letras em minúsculas.
toUpperCase()	Transforma todas as letras em Maiúsculas.

A propriedade length

A propriedade length devolva um inteiro que indica o número de elementos numa cadeia de caracteres. Se a cadeia está vazia (" "), o número é zero.

A sintaxe é simples :

```
x=variable.length;  
x=("cadeia de caracteres").length;
```

A propriedade length só serve para as Strings, mas também para conhecer o comprimento e número de elementos :

- de formulários. Quantos formulários diferentes existem?
- de botões radio. Quantos botões radio existem num grupo?
- de checkbox. Quantas checkbox existem num grupo?
- de opções. Quantas opções existem num Select?
- de frames. Quantas frames existem?
- de âncoras, de links, etc.

O método charAt()

Temos para já de notar os caracteres são contados da esquerda para a direita e que a posição do primeiro caractere é 0. A posição do último caractere é então o comprimento (length) da cadeia de caractere menos 1;

```
cadeia :      Javascript (comprimento = 10)  
           | | | | | | | | | |  
posição :    0123456789 (comprimento - 1)
```

Se a posição que se indica é inferior à zero ou maior que o comprimentos menos 1, Javascript devolva uma cadeia vazia.

A sintaxe de charAt() é:

```
cadeia_reposta = cadeia_partida.charAt(x);
```

Onde x é um inteiro compreendido entre 0 e o comprimento da cadeia a analisar menos 1.

Notar os seguintes exemplos:

```
var str="Javascript";           A resposta é "J".  
var chr=str.charAt(0);  
var chr="Javascript".charAt(0);  
ou var chr=charAt(str,0);  
ou var chr=charAt("Javascript",0);
```

```
var str="Javascript";           A resposta é "t".  
var chr=str.charAt(9);  
var chr=charAt(str,9);
```

```
var str="Javascript";           A resposta é ""  
var chr=charAt(str,13);         seja vazia.
```

O método indexOf

Este método reenvia à posição, seja x, de uma String parcial (letra única, grupo de letras ou palavra) numa cadeia de caracteres começando na posição indicado por y. Isto permita, por exemplo, de ver se uma letra, um grupo de letras ou uma palavra existe numa frase.

```
variavel="cadeia_de_caracteres";  
var="string_parcial";  
x=variavel.indexOf(var,y);
```

Onde y é a posição à partir da qual a pesquisa (da esquerda para a direita) deve começar. Este pode ser qualquer inteiro compreendido entre 0 e o comprimento -1 da cadeia à analisar.

Se a posição não é especificada, a pesquisa começa por defeito da posição 0.

Se a string parcial não é encontrada na cadeia de caracteres analisada, o valor devolvido será igual a -1. exemplos :

Alguns exemplos

```
variavel="Javascript"           x vale 4
var="script"
x=variable.indexOf(var,0);

variavel="www.truquesedicas.com" x vale -1
var="@@"
x=variable.indexOf(var);
```

Objecto String (continuação)

O método lastIndexOf()

Este método é muito parecido com o indexOf() só que a pesquisa é feita da direita para a esquerda (começa pelo fim).

A sintaxe é idêntica só que o **y** representa uma posição em relação ao fim da cadeia de caracteres.

```
x=variavel.lastIndexOf(var,y);
```

Os seguintes exemplos mostrem a diferença entre indexOf() e lastIndexOf() :

```
variavel="Javascript"
var="a"
x=variavel.indexOf(var,0); aqui x vale 1 ou seja a posição do primeiro a.
x=variavel.lastIndexOf(var,9); ici x vale 3 ou seja a posição do segundo a.
```

De notar que mesmo quando começa-se a ler a partir do fim da cadeia, a posição devolvida é contada desde o início da cadeia começando por zero.

O método substring()

O método **substring()** é do tipo indexOf(), lastIndexOf() e charAt() que acabamos de estudar. Este método será particularmente útil, por exemplo, para tomar diferentes dados numa longa cadeia de caracteres.

```
variavel = "cadeia de caracteres"
resultado=variavel.substring(x,y)
```

Os **x** e **y** são inteiros compreendidos entre 0 e o comprimento menos 1 da cadeia de caracteres.

Se **x** é inferior ao **y**, o valor devolvido começa na posição **x** e acaba na posição **Y-1**.

Se **x** é superior ao **y**, o valor devolvido começa na posição **y** e acaba na posição **X-1**.

Isso, dá o mesmo resultado e é equivalente escrever por exemplo substring(3,6) ou substring(6,3).

Se **x** é igual ao **y**, substring() devolva uma cadeia vazia (lógico, não?)

Aqui estão alguns exemplos :

```
Javascript
| | | | | | | | | |
0123456789
str="Javascript";

str1=str.substring(0,4);
str2="Javascript".substring(0,4);
str3=str.substring(6,9);
```

Os resultados são :

```
str1="Java"; seja as posições 0,1,2 et 3.  
str2="Java"; seja as posições 0,1,2 et 3.  
str3="rip"; seja as posições 6,7 et 8
```

O método toLowerCase()

Este método rescreve uma cadeia toda em minúsculos.

```
variavel2="cadeia de caracteres";  
variavel1=variavel2.toLowerCase();
```

Exemplo :

```
str="JavaScript";  
str1=str.toLowerCase();  
str2="JavaScript".toLowerCase();
```

O resultado será :

```
str1="javascript";  
str2="javascript";
```

O método toUpperCase()

Este método rescreve uma cadeia toda em minúsculos.

```
variavel2="cadeia de caracteres";  
variavel1=variavel2.toUpperCase();
```

Exemplo :

```
str="JavaScript";  
str1=str.toUpperCase();  
str2="JavaScript".toUpperCase();
```

O resultado será :

```
str1="JAVASCRIPT";  
str2="JAVASCRIPT";
```

Utilidade do toLowerCase() e do toUpperCase()

A utilidade destes 2 métodos não salta a vista. Mas é importante, visto que o Javascript é **case sensitive**. Assim uma pesquisa sobre Euro irá dar o mesmo resultado do que EURO.

Pode-se assim aconselhar de converter as bases de dado em minúsculas (ou toda em maiúscula).

Objecto Math

Para manipular os numeros, aqui está o **objecto Math**.

O método abs()

x=Math.abs(y);

O método **abs()** reenvia o valor absoluto (valor positivo) de y. Isto é, este método elimina o símbolo negativo de um número.

```
y = 4;  
x = math.abs(y);  
x = Math.abs(4);  
x = math.abs(-4);
```

o resultado será sempre

x = 4

O método ceil()

x=Math.ceil(y);

O método **ceil()** reenvia o inteiro superior ou igual ao y.

Atenção ! Este método não arredonda o número.

Como pode-se ver neste exemplo, se y = 1.01, o valor de x será 2.

```
y=1.01;  
x=Math.ceil(y);
```

o resultado é

x=2.

O método floor()

x=Math.floor(y);

O método **floor()** reenvia o inteiro inferior ou igual ao y.

Atenção! este método não arredonda o número.

Como pode-se ver no exemplo, se y = 1.99, o valor de x será 1.

```
y=1.999;  
x=Math.floor(y);
```

o resultado será

x=1.

O método round(y)

x=Math.round(y);

O método **round()** arredonda o número ao inteiro mais próximo.

```
y=20.355;  
x=Math.round(y);  
o resultado será  
x=20;
```

Atenção! Alguns cálculos reclamam uma maior precisão. Para ter duas decimais após a vírgula, utiliza-se a seguinte fórmula:

```
x=(Math.round(y*100))/100;
```

e o resultado será neste caso

```
x=20.36;
```

O método `max()`

```
x=Math.max(y,z);
```

O método `max(y,z)` reenvia o maior de 2 números y e z.

```
y=20; z=10;  
x=Math.max(y,z);
```

o resultado será

```
x=20;
```

O método `min()`

```
x=Math.min(y,z);
```

O método `min(y,z)` reenvia o menor de 2 números y e z.

```
y=20; z=10;  
x=Math.min(y,z);
```

o resultado será

```
x=10;
```

Objecto Math (continuação)

O método `pow()`

```
x=Math.pow(y,z);
```

O método `pow()` calculo o valor de um número y elevado a z.

```
y=2; z=8  
x=Math.pow(y,z);
```

o resultado será

```
x=28 ou seja x=256
```

O método `random()`

```
x=Math.random();
```

O método `random()` reenvia o valor de um número aleatório escolhido entre 0 e 1.

Atenção! Este método só funciona em Unix.

Por isso pode-se utilizar a seguinte função que reenvia um número "aleatório" entre 0 e 9.

```
function srand(){ t=new Date(); r=t.getTime();  
p="a"+r; p=p.charAt((p.length-4)); x=p; }
```

O método `sqrt()`

```
x=Math.sqrt(y);
```

O método `sqrt()` reenvia a raiz quadrada de `y`.

```
y=25;  
x=Math.sqrt(y);
```

o resultado será

```
x=5;
```

O método `parseInt()`

```
x=parseInt(variável);
```

Devolva a parte inteira de um número com vírgula.

```
str='1.2345';  
x=parseInt(str);
```

o resultado será

```
x=1;
```

O método `eval()`

```
x=eval(variável);
```

esta função avalia uma cadeia de caracteres sob a forma de valor numérico. Pode-se inserir na cadeia operações numéricas, operações de comparação, instruções e mesmo funções.

```
str='5 + 10';  
x=eval(str);
```

o resultado será

```
x=15;
```

diz-se na literatura que esta função `eval()` é uma operação majora do Javascript. Que seu uso não é aconselhado para iniciados. Pior, que esta função `eval()` não é suportada por todas as plataformas, com todos as versões dos browsers. Verificar sempre o resultado devolvido pelo `eval()`.

As funções trigonométricas

Aqui (sem comentários) as diferentes funções trigonométricas:

```
x=Math.PI;  
x=Math.sin(y);  
x=Math.asin(y);  
x=Math.cos(y);  
x=Math.acos(y);  
x=Math.tan(y);  
x=Math.atan(y);
```


As funções logarítmicas

Para quem quer saber, as diferentes funções logarítmicas:

```
x=Math.exp(y);  
x=Math.log(y);  
x=Math.LN2;  
x=Math.LN10;  
x=Math.E;  
x=Math.LOG2E;  
x=Math.LOG10E;
```

Objecto Date

new Date()

Este método reenvia todas as informações "data e hora" do computador do utilizador.

```
variavel=new Date();  
(as variáveis não levem acentos, por isso "variavel" e não "variável")
```

Estas informações são gravadas pelo Javascript sob o formato :

```
"Fri Feb 17 09:23:30 2001"
```

Atenção! a data e a hora no Javascript começa em 1º Janeiro 1970. Toda referência a uma data anterior irá dar um resultado aleatório.

O método **new date()** sem argumentos reenvia a data e a hora corrente. Para introduzir uma data e uma hora determinada, será da seguinte maneira:
variavel=new date("Jan 1, 2000 00:00:00");

getFullYear()

```
variavel_data=new date();  
an=variavel_data.getFullYear();
```

Devolve os dois últimos números do ano na variavel_data. Ou seja aqui 01. Como só se tem dois números, pode-se colocar 20 como prefixo, assim:

```
an="20"+variavel_data.getFullYear();
```

getMonth()

```
variavel_data=new date();  
mes=variavel_data.getMonth();
```

Devolve o mês na variável_date sob a forma de um inteiro compreendido entre 0 e 11 (0 para janeiro, 1 para fevereiro, 2 para março, etc.). Ou seja 11 (o mês menos 1). Por isso para obter o mês certo pode-se fazer como este exemplo seguinte:

```
<SCRIPT LANGUAGE="Javascript">  
<!--  
variavel_data=new date();  
mes=variavel_data.getMonth();
```

```
document.write("Estamos no "+(mes+1)+"º mês do ano.");  
//-->  
</SCRIPT>
```

o resultado será:

Estamos no 2º mês do ano.

getDate()

```
variavel_data=new date();  
diam=variavel_data.getDate();
```

Devolva o dia do mês na variavel_data sob a forma de um inteiro compreendido entre 1 e 31. Pois é, aqui começa-se em 1 em vês de 0 (porquê???)
Não confundir com getDay() que Devolva o dia da semana.

getDay()

```
variavel_data=new date();  
dia=variavel_data.getDay();
```

Devolva o dia da semana na variavel_data sob a forma de um inteiro compreendido entre 0 e 6 (0 para Domingo, 1 para Segunda-feira, etc.).

getHours()

```
variavel_data=new date();  
horas=variavel_data.getHours();
```

Devolva a hora na variavel_data sob a forma de um inteiro compreendido entre 0 e 23.

getMinutes()

```
variavel_data=new date();  
minutos=variavel_data.getMinutes();
```

Devolva os minutos na variavel_data sob a forma de um inteiro compreendido entre 0 e 59.

getSeconds()

```
variavel_data=new date();  
segundos=variavel_data.getSeconds();
```

Devolva os segundos na variavel_data sob a forma de um inteiro compreendido entre 0 e 59.

Exemplo: Um script que dá a hora em que entrou na página.

```
25/5/19103 22:48:43  
14/5/19103 18:21:40
```

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
function getDt(){
dt=new date();
cal=""+ dt.getDate()+"/"+(dt.getMonth()+1)
+dt1.getYear();
hrs=dt.getHours();
min=dt.getMinutes();
sec=dt.getSeconds();
tm=" "+((hrs<10)?"0":"")+hrs+":";
tm+=((min<10)?"0":"")+min+":";
tm+=((sec<10)?"0":"")+sec+" ";
document.write(cal+tm);
}
// -->
</SCRIPT>
</HEAD>
<BODY >
<SCRIPT LANGUAGE="Javascript">
<!--
getDt();
// -->
</SCRIPT>
</BODY>
</HTML>

```

Exemplo: Um script com horas dinâmicas

Para quem quer que a hora altera-se todos os segundos. Recorda-se do temporizador `setTimeout` [[Um pouco de tudo...](#)]. Basta acrescentar ao script um `setTimeout` que insere a hora todos os segundos. A função que afixa a hora é `getDt()`, a instrução à acrescentar é `setTimeout("getDt()",1000)`; E o resultado é:

22:48:49

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
function getDt(){
dt=new date();
hrs=dt.getHours();
min=dt.getMinutes();
sec=dt.getSeconds();
Tm=" "+((hrs<10)?"0":"")+hrs+":";
Tm+=((min<10)?"0":"")+min+":";
Tm+=((sec<10)?"0":"")+sec+" ";
document.relogio.display.value=Tm;
setTimeout("getDt()",1000);
}
// -->
</SCRIPT>
</HEAD>
<BODY onLoad="getDt()">
<FORM name="relogio">
<INPUT TYPE="text" NAME="display" SIZE=15 VALUE="">
</FORM>
</BODY>
</HTML>

```

Outras propriedades (menos frequentes talvez)

getTime()

Devolva a hora corrente na `variavel_data` sob a forma de um inteiro representando o número de milissegundos que passaram desde 1 Janeiro 1970 00:00:00.

getTimezoneOffse()

Devolva a diferença entre a hora local e a hora GMT (Greenwich, UK Mean Time) sob a forma de um inteiro representando o número de minutos (e não em horas).

seMonth(x)

Devolva um mês ao actual valor da `variavel_data` sob a forma de um inteiro compreendido entre 0 e 11.

Exemplo : `variavel_data.seMonth(1);`

seDate(x)

Devolva um dia do mês ao actual valor da `variavel_data` sob a forma de um inteiro compreendido entre 1 e 31.

Exemplo : `variavel_data.seDate(1);`

seHours(x)

Devolva uma hora ao actual valor da `variavel_data` sob a forma de um inteiro compreendido entre 1 e 23.

Exemplo : `variavel_data.seHours(1);`

seMinutes(x)

Devolva os minutos ao actual valor de `variavel_data` sob a forma de um inteiro compreendido entre 1 e 59.

Exemplo : `variavel_data.seMinutes(1);`

seSeconds(x)

Devolva os segundos ao actual valor de `variavel_data` sob a forma de um inteiro compreendido entre 1 e 59.

Exemplo : `variavel_data.seSeconds(0);`

seTime(x)

Devolva a data desejada na `variavel_data` sob a forma de um inteiro representando o número de milissegundos que passaram desde 1 Janeiro 1970 00:00:00.

toGMTString()

Devolva o valor do actual valor da `variavel_data` em hora GMT (Greenwich Mean Time)

Objecto Array

Em geral

O objecto **Array** é uma lista de elementos indexados nos quais pode-se guardar (escrever) dados ou as invocar (ler).

Atenção! O objecto Array é do Javascript 1.1

Array de uma dimensão

Para fazer um array, procede-se a duas etapas:

- primeiro construir a estrutura do array. Nesta fase, os elementos do array estão vazios.
- depois inserir valores nos elementos definidos.

Começa-se por definir o array :

```
nome_do_array = new Array (x);
```

Onde **x** é o número de elementos do array.

De notar que, o número de elementos é limitado a 255.

Depois, depois alimenta-se a estrutura definida :

```
nome_do_array [i] = "elementos";
```

onde **i** é um número compreendido entre 0 e **x** menos 1.

Exemplo: um caderno de endereço com 3 pessoas

construção do array :

```
caderno = new Array(3);
```

inserir dados:

```
caderno[0]="Sérgio";
```

```
caderno[1]="Paulo";
```

```
caderno[2]="Angelo";
```

para aceder a um elemento, emprega-se:

```
document.write(caderno[2]);
```

Nota-se também que os dados são bem visíveis ao leitor (view source).

Nota:

- Muitas vezes é prático carregar o array com um ciclo. Admitindo que temos que carregar 50 imagens. Ou as carregamos manualmente (0.gif, 1.gif, 2.gif...), ou utiliza-se um ciclo do estilo:

```
function gifs() {
  gif = new Array(50);
  for (var=i;i<50;i++)
  {gif[i] =i+".gif";}
}
```

Propriedades e Métodos

ELEMENTOS DESCRIÇÃO

length Devolva o número de elementos do array.

- join() Junta todos os elementos do array numa única cadeia. Os diferentes elementos são separados por um caractere separador especificado no argumento. Por defeito, este separador é uma vírgula.
- reverse() Inversa a ordem dos elementos.
- sort() Devolva os elementos por ordem alfabético.

Usando os exemplo do quadro,

document.write(carnet.join()); dá como resultado : Sérgio,Paulo,Ângelo.
document.write(carnet.join("-")); dá como resultado : Sérgio-Paulo-Ângelo.
document.write(carnet.reverse().join("-")); dá como resultado : Ângelo-Paulo-Sérgio

Array de duas dimensões

Pode-se criar arrays de duas dimensões (e mais ainda).

Primeiro declara-se array de 1 dimensão da maneira clássica :

nome_do_array = new Array (x);

Depois, declara-se cada elemento do array como um array de 1 dimension :

nome_do_array[i] = new Array(y);

Para um array de 3 por 3 :

PREÇOS	T. SMALL	T. MEDIUM	T. LARGE
CAMISAS	1200	1250	1300
POLOS	800	850	900
T-SHIRT	500	520	540

```
nome = new Array(3);
nome[0] = new Array(3);
nome[1] = new Array(3);
nome[2] = new Array(3);
nome[0][0]="1200"; nome[0][1]="1250"; nome[0][2]="1300";
nome[1][0]="800"; nome[1][1]="850"; nome[1][2]="900";
nome[2][0]="500"; nome[2][1]="520"; nome[2][2]
```

Para explorar estes dados, aqui tem uma ilustração do que é possível:

Escolha do artigo :

Escolha do tamanho :

O formulário escreva-se:

```
<FORM name="form" >
<SELECT NAME="liste">
<OPTION>Camisas
<OPTION>Polos
<OPTION>T-shirts
</SELECT>
<SELECT NAME="tamanho">
<OPTION>T. Small
```

```

<OPTION>T. Medium
<OPTION>T. Large
</SELECT>
<INPUT TYPE="button" VALUE="Obter preço " onClick="affi(this.form)">
<INPUT TYPE="TEXT" NAME="txt">
</FORM>

```

Onde a função `affi()` formula-se assim:

```

function affi() {
  i = document.form.liste.selectedIndex;
  j = document.form.taille.selectedIndex;
  document.form.txt.value=nome[i][j];
}

```

Objecto Array (continuação)

Base de Dados

Aqui está um título bastante pesado! Si o Javascript pode guardar dados e apresentar estas segundos os gostos do cliente, mas estamos mesmo assim muito longe das ferramentas específicas para este tipo de trabalho.

As bases de dados em Javascript serão sempre do tipo **estático** e sobretudo com uma codificação trabalhosa.

Até agora, definiu-se alguns caracteres nos elementos dos array. Nas strings, pode-se inserir muitos e sobretudo com as ferramentas de manipulação das strings do Javascript, pode-se guardar coisas do tipo:

Sérgio*Brandão*rua da Passagem,444*4440*Valongo*Portugal*

Ou seja neste caso, 6 dados.

Codificação tipo fixo

Retomando o exmplo do caderno de endereço. De maneira clássica, deve-se prever os difrrentes campos e o número de posições consacradas a cada campo. Por exemplo:

Nome	20 posições
Apelido	10 posições
Endereço	20 posições
Código postal	10 posições
Cidade	10 posições
País	10 posições

Cada campo deve respeitar o número de posições predeterminados. As posições sobrenumerários devem ser completadas em branco. Com um risco de erro elevado a codificação e um tempo de carregamento da página sensivelmente aumentada para finalment transmitir que espaços vazios.

Nome	Apelido	Endereço	Postal	Cidade	País
20 p.	10 p.	20 p.	10 p.	10 p.	10 p.
Sérgio	Brandão	Rua da Passagem,444	4440	Valongo	Portugal

Para retomar os diferentes dados, utiliza-se a instrução **substring(x,y)**. Assim:

Nome	substring(0,19)
Apelido	substring(20,29)
Endereço	substring(30,49)
Código postal	substring(50,59)
Cidade	substring(60,69)
País	substring(70,79)

Codificação tipo variável

O princípio é de inserir um separador entre cada campo. E "pede-se" ao Javascript de retirar os caracteres compreendidos entre dois separadores.

Com o separador *, os dados tornam-se:

```
str="Sérgio*Brandão*Rua da passagem,444*4440*Valongo*Portugal*"
```

ou seja 60 posições em vés de 80 tem-se um ganho de 25 %.

Para ler este diferentes dados, proceda-se em varias etapas:

- Para **pos=str.indexOf("*")**, nota-se a posição na string do primeiro separador encontrado.
- Para **str.substring(0,pos)**, tem-se o primeiro dado compreendido entre o inicio da string (posição 0) e a posição menos 1 do separador ou seja Sérgio.
- Para **str=str.substring(pos+1,str.length)**, recria-se uma string correspondendo à string do início menos o dado parcial que acabou-se de extrair menos um separador. O que dá:
str="Brandão*Rua da passagem,444*4440*Valongo*Portugal*"
- E isto, por um ciclo, até haver separadores menos 1.

Para guardar os dados, a maneira mais prática para os guardar é o controlador do formulário Hidden (encondido) em vez da string.

Pourqué? Bem com as string funciona, mas há o risco de chegar a limitação do comprimento das strings em Javascript, 50 à 80 caracteres. Por isso usa-se o símbolo + e a concatenação dos dados para que ele seja aceito pelo compilador Javascript.

O controlador do formulário Hidden não está sujeito a esta limitação de caracteres. Pode-se guardar 30 K (e mais) de dados. Basta assim entrar no controlador do formulário Hidden e dar atributo value="os dados a guardar".

O script completo será algo do tipo:

Teste

```
<HTML>
<BODY>
<FORM name="form">
<INPUT type=hidden name="data" value="Sérgio*Brandão*Rua da
passagem,444*4440*Valongo*Portugal*">
</FORM>
<SCRIPT LANGUAGE="javascript">
str=document.form.data.value; // extriar o string do controlador hidden
nsep=6 // número do separador
for (var i=0;i<nsep;i++){
pos=str.indexOf("*");
document.write(str.substring(0,pos)+ "<BR>");
str=str.substring(pos+1,str.length);
}
</SCRIPT>
</BODY>
</HTML>
```

Frames

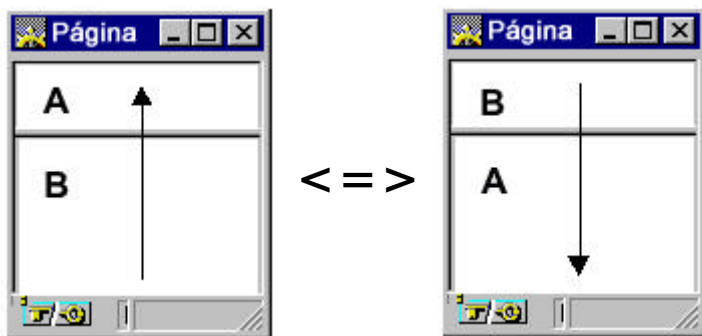
Em geral

Para já, vou apresentar as frames. Utilisar as frames permita a divisão da janela afixando a página HTML em várias partes independentes uns dos outros. Pode-se assim caregar diferentes páginas em cada parte. Para a sintaxe Html das frames, recomendo o tutorial de HTML - **As frames** -

Em Javascript, interessa a capacidade das frames à interagir. Ou seja a capacidade de trocar informações entre elas.

A filosofia do Html quer que cada página que compõe um site seja uma entidade independente.

Neste esquemas a página principal contem duas frames, em que podemos trocar informações entre as frames mantendo sempre a mesma página:



Propriedades

PROPRIEDADES	DESCRIÇÃO
length	Devolva o número de frames subordinadas na página principal (página que contem todas as frames).
parent	Sinónimo para a página principal.

Troca de informação entre frames

Com o exemplo seguinte, vamos transferir dados introduzidos pelo utilizador numa frame, para uma outra frame.

Teste do exemplo

A página principal das frames

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET ROWS="30%,70%">
<FRAME SRC="subordinada01.htm" name="subordinada01">
<FRAME SRC="subordinada02.htm" name="subordinada02">
</FRAMESET>
</HTML>
```

A página principal contém duas frames subordinadas "subordinada01" e "subordinada02".

A frame subordinada01.htm

```
<HTML>
<BODY>
<FORM name="form1">
<INPUT TYPE="TEXT" NAME="en" value=" ">
</FORM>
</BODY>
</HTML>
```

A frame subordinada02.htm

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="Javascript">
<!--
function affi(form) {
```

```
parent.subordinada01.document.form1.en.value=document.form2.out.value
}
// -->
</SCRIPT>
</HEAD>
<BODY>
Escreve um valor e clica em "Enviar".
<FORM NAME="form2" >
<INPUT TYPE="TEXT" NAME="out">
<INPUT TYPE="button" VALUE="Enviar" onClick="affi(this.form)">
</FORM>
</BODY>
</HTML>
```

Os dados introduzido pelo utilizador encontra-se pelo caminho **document.form2.out.value**. Transfere-se estes dados na zona de texto da outra frame. Para isso, temos de especificar o caminho completo. Primeiro a zona de texto encontra-se na frame subordinada chamada subordinada01. Então o caminho começa por **parent.subordinada01**. Nesta frame encontra-se um documento que contém um formulário (form1) que contém por sua vez uma zona de texto (en), que tem como propriedade value. o que faz, que o caminho seja **document.form1.en.value**. E a expressão completa será:

```
parent.subordinada01.document.form1.en.value=document.form2.out.value
```

Por: [Sérgio Brandão](#)