JavaScript avançado

1) Introdução

O *JavaScript* incorpora, uma série de objetos que permitem realizar as tarefas mais importantes que se esperam realizar com esta linguagem. Relativamente aos objetos que podem ser utilizados em *scripts JavaScript*, dividem-se em quatro tipos:

a) Objetos para operações com *Strings*

Permitem efectuar as mas variadas operações com texto e sequências de caracteres.

b) Operações matemáticas

Implementam as operações matemáticas correntes.

c) Data e Hora

Esta família de objetos permite realizar operações com datas e tempo.

d) Objetos relacionados com o *browser*

Estes objetos são os mais importantes, porque são os que permitem várias operações e manipulações ao nível do *browser* Web.

No *JavaScript* há **funções que não estão associadas a nenhum objeto** e que, portanto, não podem ser consideradas métodos. Estas funções são as seguintes:

escape(String)

unEscape(String)

eval(String)

parseFloat(String)

parseInt(String,base)

As funções *escape()* e *unEscape()* têm uma finalidade muito específica. Uma vez que certos caracteres (por exemplo, espaços em branco) não podem ser incluídos nos endereços Web (URL), é necessário fazer uma codificação, de forma a utilizá-los dessa forma. Assim, a função *escape()* gera como resultado o código correspondente a uma determinada entrada em texto simples, que pode ser incorporado num URL, enquanto o *unEscape()* realiza a função inversa. Por exemplo:

```
<script language="JavaScript">
        var x = escape(“ “);
        var y = eval(“(12/4)*5“);
        document.write("x = ",x);
        document.write("y = ",y);
</script>
```

Devolve o resultado:
X = %20
Y = 15

Estas funções irão ser úteis apenas para os leitores que utilizarem *cookies*, uma vez que poderá ser necessário colocar espaços e outros caracteres fora do comum, nos URL's.

A função *eval()*, procura avaliar o conteúdo que lhe é passado corno parâmetro e gera um resultado numérico.

A função *eval()* é das mais úteis deste conjunto de funções e é, normalmente, muito utilizada nos programas para avaliação de expressões numéricas.

As funções *parselnt()* e *parseFloat()* têm características semelhantes. O objetivo é converter o valor passado corno parâmetro (uma *String* de texto) num valor numérico, inteiro no caso do *parselnt()*, ou de vírgula flutuante no caso do *parseFloat()*.

Vejamos dois exemplos de aplicação destas duas funções: caso do

```
var x = parseInt("FF",16);
var y = parseFloat("23.34 litros ");

Obtém-se o seguinte resultado:
x = 255
y = 23.34
```

O *parselnt()* converte a *String* que lhe é passada num número inteiro na base indicada ("FF" na base 16 é igual a 255 em decimal), O *parseFloat()* converte a *String* que lhe é passada num número de vírgula flutuante (23.34, ignorando o restante texto).

a) STRINGS

O objeto String é utilizado no JavaScript para todas as operações que se possam realizar envolvendo strings. Este objeto pode ser utilizado sem sequer necessitarmos de criar com a palavra reservada "*new*". No entanto, em termos formais, estaremos a chamar uma função JavaScript e não a instanciar um objeto (o comportamento do código deverá ser exatamente igual).

O objeto **String** é composto por **uma propriedade** e por um **conjunto de métodos**, que seguidamente se enunciam:

Propriedades:

length
Comprimento da *string* (número de caracteres)

Métodos:

anchor(name)
Devolve uma *String* com o texto HTML correspondente ao elemento de *link (Anchor* – <a name= "name">, com o parâmetro name preenchido.

big()
Formatação HTML com o elemento <big> (texto grande).

blink()
Formatação HTML com o elemento <blink> (texto a piscar).

bold()
Formatação HTML com a marca <b> *(Bold).*

charAt(posição)
Devolve o carácter localizado na posição indicada da *String* de texto.

charCodeAt(posição)
Devolve o código do carácter localizado na posição indicada da *String* de texto.

concat(valor1, ...)
Concatenar valores (1 a n) a uma *string.*

fixed()
Devolve uma *String* com a formatação HTML correspondente ao elemento <fixed>.

fontcolor(cor)
Devolve uma *String* com código HTML, em que é aplicado um elemento <font>, com um atributo *color* definido com o valor colocado em "cor", atributo esse definido na escala RGB, da forma que é aplicado no HTML.

fontSize(tamanho)
Devolve formatação HTML baseada na marca <fontsize>, com o atributo *size* igual a "tamanho".

fromCharcode(c1,c2,...)
(static) cria uma string a partir de códigos de carácter passados como parâmetro.

indexOf(string,loc)
Devolve a posição em que se encontra a primeira ocorrência da "string", a partir da posição indicada por "loc".

Italics()
Devolve uma *String* com a formatação HTML correspondente ao itálico (elemento <i>).

lastIndexOf(string,loc)
Idêntico a "indexOf()", mas devolve a última ocorrência de "*string*", em vez da primeira.

link(href)
Devolve uma *String* com o código HTML correspondente à colocação de um *link* (<a href = "href">), com o parâmetro "href" preenchido.

match(exp)
Procura uma expressão (exp) numa string e devolve a string, bem como outros dados sobre a parte encontrada.

replace(exp, sub)
Idem, mas subtituindo a expressão encontrada (exp) pela de substituição (sub).

search(exp)

Procura numa *string* uma determinada substring (exp), devolvendo a posição encontrada.

slice(inicio,fim)

Define um pedaço de uma string entre as posições início e fim.

small()

Formatação HTML com o elemento <small>.

split(delimitador)

Parte uma string num array de strings, delimitada pelo delimitador.

strike()

Formatação HTML com o elemento <strike>.

sub()

Formatação HTML com o elemento <sub>.

substring(loc1 ,loc2)

Devolve uma *String* contida na *String* mãe, localizada entre as posições "loc1" e "loc2".

substr(inicio,comp)

Define uma substring, a partir de início e com o tamanho “comp".

sup()

Forrnatação HTML com o elemento <sup>.

toLowerCase()

Devolve uma *String* com todos os caracteres convertidos para minúsculas.

toUpperCase()

Idem, mas convertendo todos os caracteres para maiúsculas.

```
<Script language = "JavaScript">
        var s="Esta e uma string de texto";
        document.write("Esta e uma string de texto", "</p>");
        document.write("Tamanho da String = ",s.length, " caracteres");
        document.write(“ <br/> ");

        // Definicao de uma Substring
        document.write("uma substring de texto", "</p>");
        var s1 = s.substring(4,10);
        document.write(s1);
        document.write(" <br/> ");

        // Passagem para maiusculas
```

```
        var s2 = s1.toUpperCase();
        document.write(s2);

        // Algumas formatações HTML
        document.write("<br/> Italico: ",s2.italics());
        document.write("<br/> Bold: ",s2.bold());
        document.write("<br/> Piscar: ",s2.blink());
</script>
```

Neste exemplo, pode constatar que foi definida uma *string* logo no início e colocada na variável "s". Depois, foi calculado o tamanho da *string* e enviado para o ecrã. Foi depois definida urna *substring* ("s1") que é composta pelos caracteres localizados entre as posições 4 e 10 da *string* "s". Depois de enviada para o ecrã , foi convertida para maiúsculas (com o correspondente método **toUpperCase()** que coloca o resultado numa outra *string* "s2").

Finalmente, aplicaram-se algumas formatações HTML sobre esta *string* final, o que inclui o itálico, o **Bold** e o **Blink**.

b) OPERAÇÕES MATEMÁTICAS

Outro tipo de necessidades que existem em termos de programação é a realização de operações de natureza matemática. Estas são asseguradas por meio de um objeto denominado *"Math"*. À semelhança do objeto anterior *(String)*, com este objeto é também possível tratar este tipo de elementos como funções, não utilizando o "new" para efectuar a sua criação. Este objeto *"Math"* tem como característica fundamental o facto de as suas propriedades serem todas estáticas, isto é, não mudarem no decurso da utilização do objeto.

O objeto Math é composto por propriedades e por um conjunto de métodos, que seguidamente se enunciam:

Propriedades: (constantes matemáticas)

E	Número de *neper*.
LN10	Logaritmo natural / neperiano de 10.
LN2	Logaritmo natural / neperiano de 2.
PI	Número PI.
SQRT1_2	Raiz quadrada de 1/2.
SQRT2	Raiz quadrada de 2.
LOG10E	Logaritmo base 10 de e.
LOG2E	Logaritmo base 2 de e.

```
<script language = "JavaScript">

        document.write("E = ",Math.E);
        document.write("<br/>LN10 = ",Math.LN10);
        document.write("<br/>LN2 = ",Math.LN2);
        document.write("<br/>PI = ",Math.PI);
        document.write("<br/>SQRTC2 = ",Math.SQRT1_2);
```

```
        document.write("<br/>SQRT2 = ",Math.SQRT2);
        document.write("<br/>LOG10E = ",Math.LOG10E);
        document.write("<br/>LOG2E = ",Math.LOG2E);

</script>
```

Todos os números, sendo reais e de cálculo infinito, aparecem com a precisão máxima que é possível ser conseguida:

E= 2,1718281828459045
LN10 = 2,302585092994046
LN2 = 0,6931471805599453
PI= 3,141592653589793
SQRT1_2 = 0,7071067811865476
SQRT2 = 1,4142135623730951
LOG10E = 0,4342944819032518
LOG2E = 1,4426950408889634

Métodos:

Observação: Todos os métodos de natureza trigonométrica funcionam com valores em radianos.

abs(número)
Valor absoluto do "número", ou seja, sempre o valor independentemente do sinal (positivo ou negativo).

acos(número)
Arco-coseno de "número".

asin(número)
Arco-seno de "número".

atan(número)
Arco tangente de "número".

atan2(x,y)
Ângulo entre o eixo dos x e um ponto (identificado por x y).

ceil(núrnero)
Devolve o próximo inteiro maior que "número".

cos(núrnero)
Coseno de "número".

exp(número)
Devolve "**e**" levantado ao "número" ($e^{número}$).

floor(número)
Devolve o inteiro anterior menor que "número".

log(número)
Devolve o logaritmo de "número".

max(num1,num2)
Devolve o número maior dos números "num1" e "num2",

min(num1,num2)
Devolve o número menor dos números "num1" e "num2" .

pow(num,expoente)
Devolve "num" elevado a "expoente".

random()
Devolve um número aleatório entre 0 e 1.

round(número)
Arredonda "número" para o inteiro mais próximo.

sin(número)
Seno de "número".

sqrt(número)
Raiz quadrada de "número".

tan(número)
Tangente de "número".

Vejamos um conjunto de alguns exemplos de utilização de métodos do objeto *"Math":*

```
<script language = "JavaScript">

        // Coseno de PI (-1)
        document.write("<br/>Coseno de PI = ",Math.cos(Math.PI));

        // Raiz quadrada de 9 (3)
        document.write("<br/>Raiz Quadrada de 9 = ", Math.sqrt(9));

        // Arredondar um número (-5)
        var x = -4.56;
        document.write("<br/>Arredond. de -4.56 = ", Math.round(x));

        //Gerar um número aleatório inteiro entre 0 e 10
        var y = 10*(Math.random());                    //gerar um número
        var z = Math.round(y));                        // arrendondar
        document.write("<br/>z = ", z);

</script>
```

c) DATAS E HORAS

O trabalho com datas e tempo em *JavaScript* faz-se utilizando um objeto específico, "*Date*". Este objeto é criado e utilizado como qualquer outro objeto, sendo necessário efetuar a criação de um objeto concreto para o poder utilizar.

A criação de um objeto de tipo "*Date*" faz-se com a seguinte sintaxe:

nome_objeto = new Date(parâmetro);

O "parâmetro" é bastante importante para a utilização que se fizer do novo objeto criado. Se não for preenchido, a data que vai ficar contida no objeto é a data e hora correntes. Caso contrário, poderá ser colocada uma data e hora específicas, utilizando para tal a notação "ano,mês, dia,horas,minutos,segundos" ou, em alternativa, o tempo em milissegundos (representação interna).

O objeto *Date* é composto por um largo conjunto métodos.

Métodos:

getDate()
Devolve o dia do mês (inteiro entre 1 e 31).

getDay()
Devolve o dia da semana (0 = Domingo, 1 = Segunda-Feira, ... ,6 = Sábado).

getFullYear()
Devolve o ano correspondente à data, com quatro dígitos.

getHours()
Devolve a hora (inteiro entre 0 e 23).

getMilliseconds()
Devolve o campo milissegundos.

getMinutes()
Devolve os minutos (inteiro entre 0 e 59).

getMonth()
Devolve o mês (inteiro entre 0 = Janeiro e 11 = Dezembro).

getSeconds()
Devolve os segundos (inteiro entre 0 e 59).

getTime()
Devolve um inteiro com o número de milissegundos que se passaram desde 1 de Janeiro de 1970 às 0:00:00.

getTimezoneOffset()
Devolve o número de minutos de diferença para a hora de Greenwich (GMT).

getUTCDay()
Devolve o dia da semana, estando a data expressa em tempo universal (UTC).

getUTCFullYear()
Devolve o ano (tempo UTC).

getUTCHours()
Devolve a hora (UTC).

getUTCMilliseconds()
Devolve o campo milissegundos (UTC).

getUTCMinutes()
Devolve minutos (UTC).

getUTCMonth()
Devolve o mês (UTC).

getUTCSeconds()
Devolve segundos (UTC).

getYear()
Devolve o ano correspondente à data num formato de dois dígitos.

parse(data)
Devolve o número de milissegundos ocorridos entre 1 de Janeiro de 1970 e a data.

setDate(valor)
Coloca o Mês do objeto igual ao valor inteiro entre 1 e 31 passado em "valor".

setFullYear(valor)
Idem para o ano (ano completo 4 dígitos).

setHours(valor)
Idem para as horas ("valor" entre 0 e 23).

setMilliseconds(valor)
Idem para o campo milissegundos.

setMinutes(valor)
Idem para os minutos ("valor" entre 0 e 59).

setMonth(valor)
Idem para o mês ("valor" entre 0 e 11).

setSeconds(valor)
Idem para os segundos ("valor" entre 0 e 59).

setTime(valor)

Fixa a data, sendo o "valor" o número de milissegundos desde 1 de Janeiro de 1970 às 0:00:00.

setUTCDate(dia_mes)
Fixa o dia do mês (tempo UTC).

setUTCFullYear(valor)
Fixa o ano (UTC).

setUTCHours(valor)
Fixa a hora (UTC).

setUTCMilliseconds()
Fixa o campo milissegundos (UTC).

setUTCMinutes()
Fixa os minutos (UTC).

setUTCMonth()
Fixa o mês (UTC).

setUTCSeconds()
Fixa os segundos (UTC).

setYear(valor)
Fixa o ano.

toString()

toGMTString()

toLocaleString()

Diversas formas de obter diferentes *Strings* com a data representada no objeto em causa. Se usar o primeiro, obterá uma representação corrente.

toUTCString()
Converte uma data para uma *string* (UTC)

valueOf()
Converte uma data para um número.

UTC (ano,mes,dia,hora,minuto,segundo) - Devolve o número de milissegundos decorridos entre o dia 1 de Janeiro de 1970, às 0:00:00 GMT e a hora especificada.

No exemplo de *script* que se segue, vemos como podemos utilizar estes métodos para colocar no documento HTML exatamente aquilo que pretendemos, em termos de data e hora:

```
<Script language = "JavaScript">
        var hoje = new Date();
        var dia = hoje.getDate();
        var hora = hoje.getHours();
        var minuto = hoje.getMinutes();
        var ontem = new Date();

        ontem.setYear(2000)

        document.write("Hoje é dia ", dia);
        document.write(" e são ",hora," horas e ");
        document.write(minuto," minutos", "</p> ");    //Hoje é dia dd e são hh horas e
yy minutos

        document.write("Data: ", hoje.toLocaleString());     //Data: day, month dd, aaaa
hh:yy:mm
        document.write("Diferença entre hoje e ontem = ");   //aaaa – 2000
        document.write(hoje.getYear() – ontem.getYear(),"anos. ");    //Diferença de
datas:
</script>
```

d) OBJETOS DO *BROWSER*

Neste domínio é onde residem os maiores problemas de incompatibilidades entre *browsers*. Não é aconselhado utilizar-se elementos que sejam específicos de um determinado *browser*.

→ *history*

Este objeto permite trabalhar sobre a lista de acessos a locais WWW existente no *browser*. As propriedades e métodos disponíveis são os seguintes:

Propriedades:

current
Uma *string* que representa o URL do documento actual.

length
Representa o número de elementos da lista de acessos.

next
Representa o URL do elemento após o corrente, na lista.

previous
Idem para o anterior.

Métodos:

back()

Retorna ao documento anteriormente visitado (como o botão *Back* do browser) na lista de acessos.

forward()

Avança para o documento seguinte na lista de acessos,

go(posição)

Avança para o documento WWW identificado por "posição". Se este argumento for um inteiro, a página identificada por este número na lista de acessos é escolhida, se pelo contrário for uma *String,* esta conterá um URL ou parte dele na lista de acessos.

toString()

Devolve uma tabela HTML contendo o histórico de acessos do *browser*.

```
history.go(-1);
history.back();
```

São equivalentes a pressionar o botão *Back* do *browser*.

→ document

Este objeto é criado quando uma determinada página WWW é carregada e contém informação diversa acerca desse documento. Para alem disso, permite a sua manipulação com um conjunto e métodos muito simples.
Vejamos uma lista das propriedades e dos métodos associados. Pela primeira vez vamos ver algumas propriedades que podem ser modificadas e não só lidas.
No entanto, nem todas as propriedades que se seguem o permitem:

Propriedades:

alinkColor

Representa a cor que um *link*, ou ponteiro de tipo *Anchor* do HTML, toma quando é pressionado com o rato, mas antes de o botão deste ser largado.

anchors[]

Array de elementos de tipo *Anchor* do documento HTML.

applets[]

Array de objetos Java, um objeto por cada *applet* Java.

bgColor

A cor do fundo do documento.

cookie

Uma *string* com parâmetros *"cookie"* do documento.

domain

String que designa o domínio *Internet* a que o documento pertence.

ernbeds[]
Array com objetos inseridos com <embed>.

fgColor
A cor do texto.

forms[]
Array de objetos *Form* existentes na página HTML.

lmages[]
Array de objetos imagem do documento (<img>).

lastModified
Uma *String* com a data da última alteração do documento.

linkColor
A cor com que os *links* aparecem no documento.

links[]
Um *array* contendo todos os *links* do documento.

location
Uma *String* com o URL do corrente documento.

plugins[]
Idêntico ao embeds[].

referrer
Uma *String* contendo o URL do documento a Partir do qual foi chamado o corrente.

title
O título do documento HTML.

URL
URL do documento.

vlinkColor
Cor que tomam os *links* já visitados.

Propriedades específicas *Netscape:*

classes
Array que contém objetos de estilo.

height
Altura em *pixels* do documento.

ids

Array de objetos de estilo (<id>).

layers[]
Array de objetos *layer*.

tags
Array de objetos composto pelo conjunto de elementos HTML do documento.

width
Largura em *pixels* do documento.

Propriedades específicas *Internet Explorer:*

activeElement
Define o elemento <input> que tem o foco.

alI[]
Identifica todos os elementos HTML de um documento.

charset
Página de caracteres em uso.

children[]
Elementos HTML que compõem o documento, pela ordem pela qual aparecem no documento.

defaultCharset
Página de caracteres predefinida para um documento.

expando
Se colocada a *false*, esta propriedade impede que um objeto seja expandido.

parentWindow
Define a janela que contém o documento.

readyState
Indica o estado de carregamento de um document *(uninitialized* - não começou o carregamento, *loading* documento em carregamento, *interactive* - em carregamento, mas já possível interacção com utilizador ou *complete* - página carregada).

Atenção: nas propriedades, as cores são expressas na forma de texto ou em RGB com dados hexadecimais, por exemplo:

```
document.bgColor = "Red";
```

ou em alternativa:

```
document.bgColor = "FF0000";
```

Coloca o fundo de um documento a vermelho.

Métodos:

clear()
Limpa o conteúdo de uma janela.

close()
Termina a importação de um documento e dá carregamento como terminado *("Done").*

open(tipo)
Abre um documento para recepção de dados (por exemplo vindos de um write()). O "tipo" é um parâmetro opcional, em que se pode colocar um outro tipo de dados (por exemplo, suportado por um *plug-in* - produto externo ao *browser).*

write()
Escreve texto (HTML) no documento.

writeln()
Escreve texto (HTML) no documento e coloca no final um carácter de fim de linha (só tem efeito se for utilizada formatação HTML <PRE>).

Métodos*:*

elementFromPoint(x,y)
Devolve o objeto HTML que estiver nas coordenadas em causa.

→ location

O objeto *location* providencia informação acerca do URL corrente. É composto somente por propriedades. Todas as propriedades sao *stings* que representam várias facetas distintas do URL:

Propriedades:

hash
Uma *string* com o nome da URL.

host
O nome da máquina e o porto no URL.

hostname
O nome da máquina.

href
Todo o URL.

pathname
Só a parte de caminho do URL.

port
Só o porto.

protocol
O protocolo usado (incluindo o carácter "**:**").

search
Informação eventualmente passada a um programa CGI (aparece a seguir a um carácter "?" no URL).

Métodos:

reload()
Recarrega o documento corrente.

replace()
Substitui o documento actual por outro, sem alterar a "história" do *browser*.

Como exemplo de utilização deste, vejamos um pequeno *script* de teste, que envia para o ecrã toda a informação relativa à localização em que o ficheiro HTML correspondente está:

```
<script language = "JavaScript">

        // Informação acerca do URL
        document.write("hash = ",location.hash);
        document.write("<br/>host = ",location.host);
        document.write("<br/>hostname = ",location.hostname);
        document.write("<br/>href = ",location.href);
        document.write("<br/>pathname = ",location.pathname);
        document.write("<br/>port = ",location.port);
        document.write("<br/>protocol = ",location.protocol);

</script>
```

→ *window*

O objeto *window* funciona corno o objeto mãe, que incorpora documentos ou outros objetos. Operando sobre o *window*, tem-se a oportunidade de controlar diretamente a janela do *browser* WWW utilizado.

Propriedades:

closed
Boleano que especifica se uma janela foi ou não fechada.

defaultStatus
Uma *string* com o valor contido na barra de *status*.

document
Referencia o documento contido na janela.

frames[]
Um *Array* com todas as *Frames* que integram a janela.

history
Referência ao objeto histórico na janela.

length
O número de *Frames* na janela.

location
Objeto *location* na janela.

Math
Referencia um objeto contendo funções matemáticas.

name
O nome da janela.

navigator
Objeto *navigator.*

offscreenBuffering
Define o tipo de *buffering* que o *browser* faz.

opener
Referencia uma janela que tenha invocado uma função open() para criar uma janela.

parent
O nome da janela principal que contém o conjunto de *Frames (Frameset).*

screen
Objeto screen.

self
O nome da janela corrente.

status
Valar a aparecer na barra de *status* (pode ser fixado atribuindo um valor a esta propriedade).

top
O nome da janela de topo.

window
O nome da janela corrente.

Propriedades *Netscape*:

crypto
Referencia um objeto que implementa criptografia (crypto).

innerHeight
Altura da área mostrável do documento.

innerWidth
Largura da mesma área.

Java
Referencia objeto global Java.

locationbar
Informa da presença, ou não, da barra de localização do *browser.*

menubar
Idem para a barra de menus.

netscape
Referencia a classe Java *netscape.*

outerHeight
Altura em *pixels* da janela.

outerWidth
Largura em *pixels* da janela.

Packages
Referencia *Packages* Java.

pageXOffset
Numero de *pixels* que o documento corrente foi deslocado para a direita (com a barra de deslocação).

pageYOffset
Idem na posição vertical (para baixo).

personalBar
Identifica a presença ou não da barra pessoal do *browser.*

screenX
Coordenada X do canto superior esquerdo da janela.

screenY
Coordenada Y do canto superior esquerdo da janela.

scrollbars
Visibilidade das barras de deslocação do *browser.*

statusbar
Visibilidade da barra de *status* do *browser*.

sun
Package Java da Sun.

toolbar
Visibilidade da barra *toolbar*.

Propriedades *Internet Explorer*:

clientInformation
Substituto da *Microsoft* para o objeto *navigator*.

event
Descreve o evento mais recente.

Métodos*:*

alert(mensagem)
Faz surgir uma janela de alerta com a mensagem passada como parâmetro. A janela capta a atenção do utilizador (não o deixa fazer mais nada no *browser)* e só desaparece quando é pressionado o botão.

blur()
Retira o foco do teclado da janela em causa, passando-o para a janela mãe.

clearInterval(id)
Pára uma execução periódica de código iniciada com um *setInterval().*

clearTimeout()
Cancela um *timeout* (execução diferida de código).

close()
Fecha a janela.

confirm(mensagem)
Faz surgir uma janela de confirmação com botões *"OK"* e *"Cancel"*. Conforme o botão pressionado, é devolvido o valor "verdadeiro" ou "falso" ao utilizador.

focus()
Dá foco de teclado a uma janela (proporciona a ocorrência de eventos de teclado).

moveBy(x,y)
Move uma janela *x pixels* para a direita e *Y* para baixo.

moveTo(x,y)
Move a janela para uma posição absoluta x,y.

open(URL, nome, param)

Abre uma janela e carrega o URL passado como parâmetro. No "param" podem ser configurados alguns aspectos relativos ao aspecto da janela,

prompt(msg,resp_defeito)

Abre uma janela de diálogo, que aceita uma entrada do utilizador, que é devolvida. "msg" contém o texto da pergunta e "resp_defeito" o valor inicial que aparece no campo a preencher.

resizeBy(a,l)

Altera o tamanho de uma janela de ***a*** pixels na altura e de ***I*** na largura.

resizeTo(a,l)

Altera as dimensões da janela para ***a*** por ***I*** pixels.

scroll(x,y)

Deslocação num documento para uma posição x,y.

scrollBy(x,y)

Deslocação no documento de x pixels para a direita e de y para baixo.

scrollTo(x,y)

Idêntico ao scroll(), que veio substituir.

setInterval(code,interv)

Executa o código *Javascript code* após um período de tempo de *interv* milisegundos e de periódica.

nome=setTimeout(exp,time)

Avalia a expressão "exp" quando passar o número de milissegundos definido por "time".

Métodos *Netscape*:

atob(str)

Descodifica uma *string* em base-64.

back()

Idêntico a pressionar o botão *back.*

btoa(dados)

Codifica os dados em base-64.

captureEvents(event-mask)

Especifica que eventos podem ser capturados.

disableExtemalCapture()

Impede a captura de eventos que ocorram num servidor diferente daquele em que corre o *script.*

enableExtemalCapture()

O oposto do anterior.

find()
Procurar texto no documento.

forward()
Equivalente a pressionar o botão *forward* do *browser.*

handleEvent(evento)
Passa um evento para o adequado gestor de eventos.

home()
Botão *home* do *browser.*

print()
Botão *print* do *browser.*

releaseEvents(mask)
Parar captura de eventos.

routeEvent(mask)
Passar evento para o próximo gestor de eventos.

setHotkeys(comando)
Activa ou desactiva a utilização de teclas para dar comandos ao *browser.*

setResizable(comando)
Permite ou impede o ajuste do tamanho da janela.

setZOptions()
Controla o comportamento das janelas quando há mais do que uma aberta em simultâneo.

sop()
Pára o carregamento do documento.

Métodos *Internet Explorer*:

navigate(url)
Carrega uma URL.

```
<script language = "JavaScript">
        window.open(http://www.google.com”);
</script>
```

Debugging

No *JavaScript* foi introduzido um par de funções muito útil para questões de *debugging* ou correção de erros em programas. Trata-se dos métodos:

watch()
unwatch()

Estes métodos aplicam-se a um determinado objeto e permitem que se force a execução de uma determinada função quando algo ocorre. O watch() aceita como argumentos o nome da propriedade a ser observada, bem corno o nome da função a ser executada quando a propriedade em causa mudar.

O unwatch() chama-se para desligar o efeito de debugging e cancelar o efeito do watch().

```
<html>
<head>
	<script language = "JavaScript">
		function var_mudou() {
			alert("Variável x foi modificada ");
		}
	</script>
</head>

<body>
	<script language = "JavaScript">
		var x=0;
		x=1;
		watch('x', var_mudou());
		unwatch();
	</script>
</body>
</html>
```

O watch() aplica-se neste caso à variável "x", definindo-se a função "var_mudou()" corno gestora do evento de mudança da variável.

Neste caso, a linha "x=1;" provoca a modificação da variável x e, portanto, a execução do código constante da função var_mudou(). .

O mesmo aconteceria mesmo que o valor da variável não tivesse sido modificado, mas desde que tivesse sido feita urna nova atribuição ao seu valor (por exemplo: x=0).