

# **Projeto e Criação de Páginas de Web**

**Javascript**

Prof.: João Sérgio dos Santos Assis

e-mail: [joao@nce.ufrj.br](mailto:joao@nce.ufrj.br)

Tel.: 598-3214

# Projeto e Criação de Páginas de Web

## Índice

1. Bibliografia auxiliar	1
2. O que é JavaScript ?	2
3. Elementos da linguagem	5
4. Desvio condicional	12
5. Laços	13
6. Funções	15
7. Peculiaridades do Netscape	17
8. Objetos	20
9. Hierarquia de Objetos do browser	26
10. Scripts dependentes do navegador	31
11. Eventos	32
12. Janelas e Frames	35
13. Formulários	39
Apêndice A – Hierarquia dos objetos	48

## 1. Bibliografia auxiliar

### **Livros:**

- JavaScript para Netscape  
Peter Kent e John Kent, Makron Books
- JavaScript Sourcebook  
G. McComb, Makron Books
- Aprenda em 24 horas JavaScript 1.3  
Editora Campus

### **Internet:**

- Documentação da Netscape  
<http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm>
- Documentação da Microsoft  
<http://www.microsoft.com/jscript>

## 2. O que é JavaScript ?

É uma linguagem de programação interpretada, com características rudimentares de Orientação a Objetos e programação dirigida por eventos. Foi desenvolvida num convenio entre a Netscape e a Sun. Inicialmente se chamava LiveScript mas mudou de nome por questões de marketing.

Esta linguagem permite que código executável possa ser incluído em páginas da Web. O Interpretador fica embutido no próprio navegador.

Apesar da expectativa criada pelo nome a linguagem tem pouco a ver com Java. A principal relação entre as duas linguagens vem de sua origem comum: as linguagens C e C++.

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

### O que é possível fazer em Javascript

- Interagir com o usuário
- Alterar características do documento
- Controlar o navegador
  - carregar um novo documento
  - retornar para a página anteriormente visitada
  - abrir/fechar janelas
  - exibir páginas diferentes de acordo com o browser do usuário
  - Controlar o conteúdo de formulários HTML
- Manipular imagens embutidas
- Ler/escrever o estado do cliente em Cookies

## O que não é possível fazer

- Não é possível escrever ou ler arquivos
- Não possui capacidade gráfica
- Não suporta estabelecer conexões em rede
- Não oferece multithreading

## Versões do JavaScript

O Javascript pode ser executado no lado do servidor ou no lado do cliente. No nosso curso abordaremos apenas a versão executada no cliente (versão client-side). A versão que roda no servidor (server-side) é utilizada nos servidores Netscape como uma alternativa aos scripts CGI. Nesta versão, diferente da client-side, é possível ler e escrever arquivos e bancos de dados no servidor.

Como em HTML, há diferenças na linguagem Javascript em função do tipo de navegador onde o programa será interpretado.

No Netscape		No Explorer	
Versão Browser	Versão Javascript	Versão Browser	Versão Jscript
Netscape 2.0	Javascript 1.0		
Netscape 3.0	Javascript 1.1	Explorer 3.0	JScript 1.0
Netscape 4.0	Javascript 1.2	Explorer 4.0	JScript 3.0
Netscape 4.5	Javascript 1.3	Explorer 5.0	JScript 5.0

## Execução de Programas JavaScript

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

Há três formas de incluir um código Javascript em uma página. A primeira delas é através da tag <SCRIPT> ... </SCRIPT>:

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

Esta tag pode aparecer em qualquer lugar da página, inclusive na área de cabeçalho (dentro da tag <HEAD> ... </HEAD>). Quando incluída na área de cabeçalho o programa é executado antes que a página seja totalmente carregada. É possível utilizar a tag <NOSCRIPT> ... </NOSCRIPT> para exibir uma alternativa ao código javascript em navegadores que não saibam executá-lo. A inclusão do código entre comentários (tag <!-- //-->) é recomendável para que os navegadores que não conheçam javascript tentem exibir o código.

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

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

As outras duas formas são:

- Através da seleção de um link

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

- Associada a um evento

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

### 3. Elementos da linguagem

Em javascript, diferente de HTML, as letras minúsculas são diferenciadas das maiúsculas. Os espaços, tabs e newlines entre os comandos são ignorados. Utiliza-se o caracter ponto e vírgula (;) após cada comando.

É possível incluir comentários em meio as instruções. Há duas formas de fazer isso:

- // - As barras duplas em qualquer ponto da linha fazem com que o interpretador ignore todos os demais caracteres até o fim da linha.
- /\* ... \*/ - O caracter barra seguido do asterisco faz com que o interpretador ignore todos os caracteres (inclusive o de mudança de linha) até encontrar o caracter asterisco seguido da barra.

#### Constantes

A tabela a seguir mostra a forma de representar dados nos tipos básicos da linguagem Javascript:

Tipo	Constante
string	"string", "Sra O'Neil"
	'string', 'o filme "silêncio dos inocentes" é ótimo'
inteiro na base 10	8, -12
inteiro octal	056, -0123
inteiro hexadecimal	0x7A, -0x10
real	3.14, -10.28
booleano	true e false

É possível criar strings entre aspas ou entre plicas. A utilização de um ou de outro caracter como delimitador da string é uma questão de conveniência do programador. Por exemplo, caso se deseje incluir uma plica em uma string deve-se utilizar aspas como delimitador. O caracter \ (barra invertida) tem um significado especial, permitindo a representação de caracteres que não tem um

símbolo gráfico associado, como tabulações (\t) e de mudança de linha (\n). É possível utilizar este carácter também para indicar que uma plica ou uma aspa não está sendo utilizada como fim de string (por exemplo, “assim pode-se inserir um carácter \" numa string delimitadas por aspas”).

## Variáveis

Os nomes de variáveis podem conter letras, dígitos (desde que não seja o primeiro carácter do nome), o carácter '\_' e o carácter '\$'. Algumas palavras não podem ser usadas como nome de variável pois são reservadas para comandos e expressões da linguagem, como por exemplo: true, var, if, while, etc. Diferente de uma linguagem como Pascal, não é preciso declarar previamente uma variável nem definir explicitamente o seu tipo.

### Exemplos válidos:

```
var i;          // Cria a variável i
i = 10;         // Guarda o valor 10 dentro de i
var i = 2;      // Faz as duas coisas ao mesmo tempo
i = 11;         // A palavra var é opcional, a princípio
i = "onze";     // O tipo da variável passou a ser string.
```

As variáveis são criadas no momento em que são utilizadas pela primeira vez e o seu tipo é definido pelo tipo do valor que recebe. Se uma variável é utilizada sem que antes tenha um valor atribuído, o seu valor é null. Embora não seja muito recomendável, é possível modificar o tipo de uma variável que já tenha sido criada simplesmente atribuindo um valor de outro tipo. A mudança do tipo de uma variável é uma fonte potencial de erros de execução.

## Operadores

A linguagem Javascript oferece uma série de operadores para manipular variáveis e constantes. É possível misturar operandos de tipos diferentes que a linguagem se encarrega de fazer a conversão dos tipos. A tabela a seguir mostra os operadores disponíveis em Javascript em comparação com os operadores oferecidos em Pascal:



Javascript	Pascal	Significado
<b>Operadores Aritméticos</b>		
+ - * /	+ - * /	Soma, Subtração, Multiplicação e Divisão
%	mod	Resto da divisão inteira
i++	i = i + 1	Incremento
i--	i = i - 1	Decremento
<b>Operadores de Comparação</b>		
> >=	> >=	Maior que, Maior e igual a
< <=	< <=	Menor que, Menor e igual a
==	=	Igual a
!=	<>	Diferente de
<b>Operadores Lógicos</b>		
!	not	Negação
&&	and	E
	or	OU
<b>Operador de Atribuição</b>		
=	:=	Atribuição de valor a uma variável

## Concatenação de Strings

O operador + também é utilizado em Javascript para fazer a concatenação de strings. É possível misturar valores numéricos e strings numa operação envolvendo o operador +. Neste caso os valores numéricos são

convertidos para string. No exemplo abaixo a variável data recebe o valor “15 de agosto” e a variável som o valor “55510”:

```
dia = 15;
data = dia + " de agosto";
x = "555";
som = x + 10; // resulta 55510
```

Obs: A conversão do valor numérico para string só é válida no caso do operador +, nos demais casos a string será convertida para um valor numérico quando possível (quando não for possível será convertida para NaN - não é numérico). No exemplo abaixo a variável sub recebe o valor 45:

```
x = "555";
sub = x - 10; // resulta 45
```

### Atribuição composta

A linguagem Javascript, a exemplo de sua antecessora C, permite a escrita simplificada de expressões do tipo “a = a + b”, onde uma variável recebe o valor de uma expressão em que ela própria aparece. Esta expressão pode ser escrita como: “a += b”. A tabela abaixo mostra algumas das combinações que podem ser realizadas:

Expressão	Simplificação
a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b

## Precedência de operadores

A tabela a seguir mostra a ordem em que são avaliadas as expressões. Para alterar a precedência é necessário usar parênteses

Ordem de avaliação das expressões	
1º	! - ++ --
2º	* / %
3º	+ -
4º	< <= > >=
5º	== !=
6º	&&
7º	
8º	? :
9º	= += -= *= /= %=

## Algumas Funções para entrada e saída

Antes de poder fazer um programa em Javascript é conveniente conhecer algumas funções que permitem realizar entrada e saída de dados:

- **prompt**

**Sintaxe:** prompt(mensagem);

**Descrição:** Abre uma janela para entrada de uma linha de texto, exibindo a mensagem passada como parâmetro. A função retorna o texto digitado pelo usuário, que deve ser atribuído a uma variável.

**Exemplo:** nome = prompt("Digite o seu nome");

- **alert**

**Sintaxe:** alert(aviso);

**Descrição:** Abre uma janela para exibir um aviso.

**Exemplo:** alert("Você digitou um caracter inválido!");

- **confirm**

**Sintaxe:** confirm(mensagem);

**Descrição:** Abre uma janela para para exibir uma pergunta para o usuário. A função retorna verdadeiro ou false de acordo com a resposta.

**Exemplo:** if (confirm("Você quer mesmo sair da página ?"))

- **document.write**

**Sintaxe:** document.write(string);

**Descrição:** Escreve uma string na página sendo exibida pelo navegador.

**Exemplo:** document.write("<H1>Esta é minha página</H1>");

## Algumas outras Funções Predefinidas

- **eval**

**Sintaxe:** eval (str);

**Descrição:** Efetua uma expressão contida numa string.

**Exemplo:** expr= "x\*2+5"; result=eval(expr);

- **isNaN**

**Sintaxe:** isNaN (valor);

**Descrição:** Retorna "true" se o valor não for numérico.

**Exemplo:** x=prompt("Entre um numero:","");

if (isNaN(x)) { ... }

- **parseInt**

**Sintaxe:** parseInt (str) ou parseInt (str,base);

**Descrição:** Converte a string str para um número inteiro. Opcionalmente pode-se indicar a base em que deve ser interpretado o número contido na string. Se o parâmetro base não for especificado, assume-se a base 10.

**Exemplo:** num = "3A";

x = parseInt(num);

y = parseInt(num,16);

- **parseFloat**

**Sintaxe:** parseFloat (str);

**Descrição:** Converte uma string num número real.

**Exemplo:** num = "3.4";

x = parseFloat (num);

## 4. Desvio Condicional

### Comando if

Um desvio condicional permite escolher qual comando (ou conjunto de comandos) será ou não executado de acordo com uma condição. O if do Javascript funciona da mesma maneira que o if do Pascal porém sua sintaxe é um pouco diferente:

Em Javascript	Em Pascal
<pre>if (nota &gt;= 7) {     aprovado = true; } else {     aprovado = false; }</pre>	<pre>if nota &gt;= 7 then begin     aprovado := true; end else begin     aprovado := false; end;</pre>

### Operador condicional ternário

Um tipo especial de desvio condicional pode ser escrito de uma forma bastante compacta em Javascript. Quando se utiliza um desvio condicional para determinar qual valor será atribuído a uma variável, é possível substituir o desvio pela utilização do operador condicional ternário (? :). O exemplo a seguir mostra dois trechos de código equivalentes em Javascript, o da esquerda utilizando o desvio condicional e o da direita o operador condicional ternário:

Desvio condicional	Operador condicional
<pre>if (a &gt; b) {     maior = a; } else {     maior = b; }</pre>	<pre>maior = (a &gt; b) ? a : b;</pre>

## 5. Laços

### Comando While

Permite repetir um bloco de comandos enquanto uma condição for verdadeira. É semelhante ao while do Pascal, apenas com uma sintaxe diferente:

Em Javascript	Em Pascal
<pre>i = 0; while (i &lt; 20) {     i++; }</pre>	<pre>i := 0; while i &lt; 20 do begin     i := i + 1; end;</pre>

### Comando for

Permite repetir um bloco de comandos enquanto uma condição for verdadeira. É mais poderoso que o for do Pascal pois :

Em Javascript	Em Pascal
<pre>for (i = 1; i &lt;= 20; i++) {     soma += i; }</pre>	<pre>for i := 0 to 20 do begin     soma := soma + i; end;</pre>

A sintaxe do for da linguagem Javascript foi definida a partir da seguinte constatação: o controle do número de vezes que um laço é executado, normalmente, envolve a inicialização de uma variável, o teste desta variável e a mudança de seu valor, o que permitirá determinar se o laço será interrompido ou não. A sintaxe do for, portanto, é a seguinte:

```
for (atribuição; condição; modificador)
{
    comandos;
}
```

A atribuição é executada apenas uma vez, antes do início da execução do laço. A seguir, é feito o teste da condição. Se esta for falsa, o laço não é executado. Se for verdadeira os comandos do interior do laço são executados, seguido da execução do modificador e volta-se ao teste da condição.

## Comandos Break e Continue

O comando break permite a interrupção de um laço antes que a condição seja satisfeita. Este comando é utilizado após um desvio condicional que testa uma segunda condição para o fim do laço. No exemplo a seguir o laço é executado 9 vezes, a menos que o usuário selecione cancelar na janela aberta pela função confirm:

```
for (i = 1; i < 10; i++)
{
    if ( !confirm("i = " + i +
        "\nDê um clique em Cancelar") )
    {
        break;
    }
}
alert("O laço terminou\n i = " + i);
```

O comando continue interrompe a iteração atual do laço, passando imediatamente a próxima iteração. Este comando também é utilizado após um desvio condicional. No exemplo a seguir o número de vezes que o write será executado é determinado pelo número de vezes que o usuário selecionar cancelar na janela aberta pela função confirm. Se o usuário cancelar 4 vezes, o write será executado apenas 5 vezes:

```
var i = 1;
while (i < 10)
{
    i++;
    if (!confirm("i = " + i +
        "\n Clique cancelar para executar um continue"))
    {
        continue;
    }
    document.write("iteração " + i);
}
alert("O laço terminou\ni = " + i);
```



## 6. Funções

Uma função é uma sequência de comandos, realizando uma tarefa específica, a qual se atribui um nome. Normalmente os comandos de uma função estão profundamente relacionados entre si. O nome da função permite incluir no programa uma chamada a função, que significa fazer um desvio para executar os comandos da função e depois retornar para a instrução seguinte ao nome da função. Uma chamada a uma função pode ser incluída numa expressão e neste caso a função deve retornar um valor que será utilizado no cálculo da expressão. Uma função pode receber parâmetros que modificam a sua execução.

A definição de uma função em Javascript segue a seguinte sintaxe:

```
function nome()  
{  
    ... comandos ...  
}
```

Na linguagem Pascal existe uma distinção entre funções que retornam e que não retornam valores (functions e procedures). Em Javascript não há esta distinção e os dois tipos de funções são declaradas da mesma maneira. A forma de fazer uma função retornar um valor é utilizar o comando `return` seguido de um valor. Ao encontrar o comando `return`, a execução da função é interrompida e o programa volta para o ponto onde foi chamado, utilizando o valor retornado na expressão onde foi incluída a chamada a função. A tabela abaixo mostra a declaração da função `quadrado` e sua utilização no cálculo da expressão  $x = 2^2 + 3 * 5^2$ :

Declaração da função	Utilização da função
<pre>function quadrado(x) {     var q = x * x;     return q; }</pre>	<pre>... x = quadrado(2) + 3 * quadrado(5); ...</pre>

Quando uma variável é criada em uma função numa declaração precedida da palavra *var* (como na variável *q* da função *quadrado* do exemplo anterior), ela só existe dentro desta função, enquanto ela estiver sendo executada (são variáveis locais).

## 7. Peculiaridades do Netscape

### Visualizando o código da página

Os navegadores costumam ter uma opção de menu que permite visualizar o código fonte da página que está sendo exibida (**view > Page Source** no caso do Netscape e **exibir > Código Fonte** no caso do Explorer).

No caso de páginas que são modificadas por um `document.write`, a seleção desta opção do menu no Netscape vai mostrar a página final, após a execução do javascript. Neste caso, se o arquivo original teste.html for:

```
<HTML>
<BODY>
Hello,
<SCRIPT>document.write(" there.")</SCRIPT>
</BODY>
</HTML>
```

O Netscape exibira após a seleção da função **view > Page Source** o seguinte resultado:

```
<HTML>
<BODY>
Hello,
  there.
</BODY>
</HTML>
```

Para ter acesso ao código original da página é necessário acrescentar `view-source:` antes da url da página na barra de endereços. Se a página do exemplo tiver como url `file:///C:/tmp/teste.html`, deve-se colocar na barra de endereços `view-source:file:///C:/tmp/teste.html`.

### Lidando com erros de sintaxe

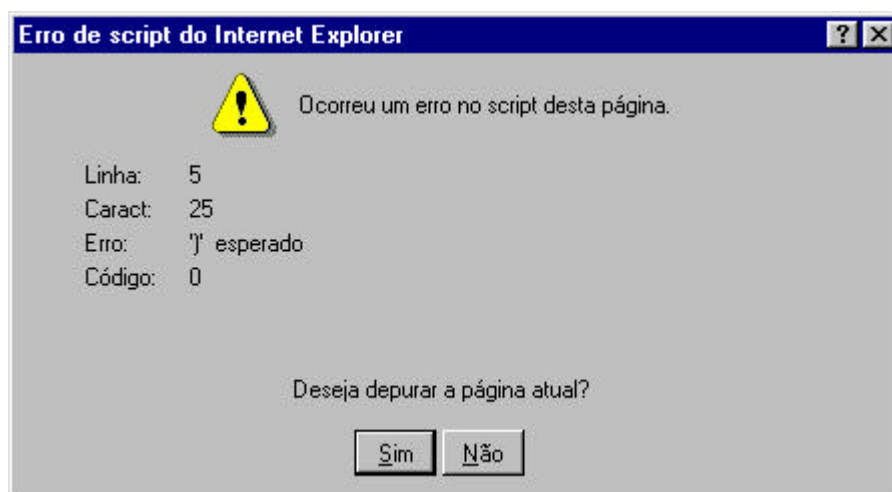
Em qualquer linguagem de programação, por mais cuidadoso que seja o programador é muito difícil não cometer erros de sintaxe. Quando isto ocorre numa linguagem compilada isto não é muito problemático pois para poder

executar qualquer pedaço do código é necessário que o programa tenha passado pelo compilador e, conseqüentemente, não contenha mais nenhum erro de sintaxe. Numa linguagem interpretada (como no caso do javascript), o interpretador vai descobrindo os erros a medida que executa. Um erro existente num trecho que é executado após um desvio condicional pode eventualmente jamais ser descoberto se o valor da condição nunca levar o código errado a ser executado. Quando descobre um erro de sintaxe, o interpretador não pode prosseguir com a execução do programa. A forma como os navegadores informam que há um erro no código javascript difere um pouco entre o Netscape e o Explorer.

No Explorer, ao encontrar um erro de sintaxe o navegador interrompe a execução do javascript e abre uma janela indicando o erro e em qual linha ocorreu. Dado o seguinte código fonte:

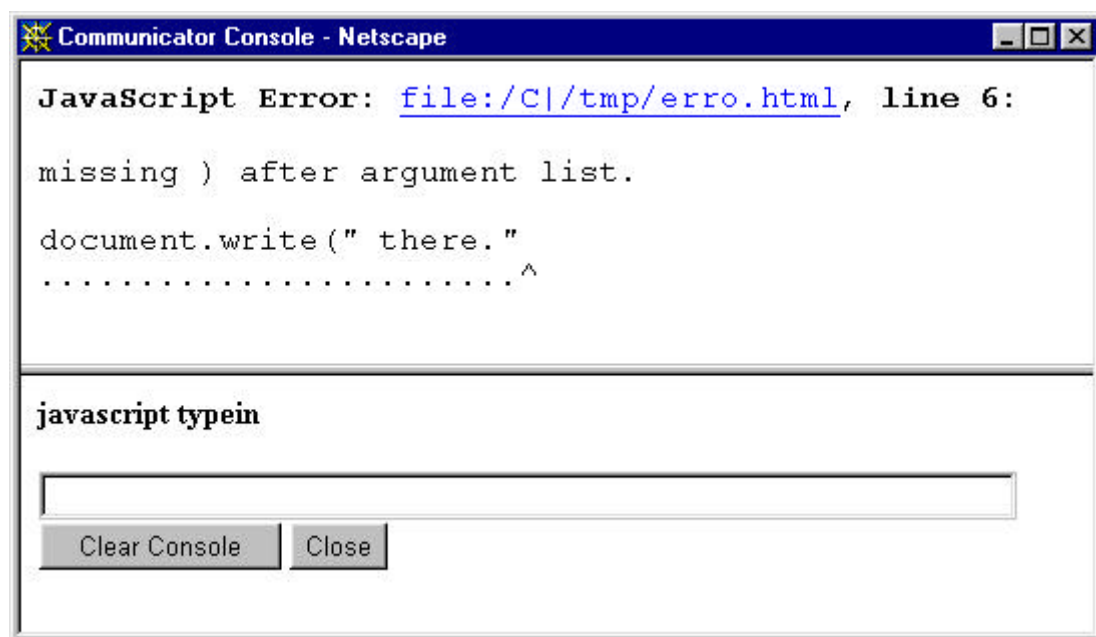
```
<HTML>
<BODY>
Hello,
<SCRIPT language="javascript">
document.write(" there."
</SCRIPT>
</BODY>
</HTML>
```

Se tentarmos exibi-lo no Explorer, a falta do caracter ) no write fará com que o navegador abra a seguinte janela:



Com as informações desta janela, basta editar o arquivo, ir na linha e coluna indicada e corrigir o erro.

No caso do Netscape, o navegador não dá nenhum sinal que ocorreu o erro. Só é possível descobrir que alguma coisa está errada por que não vai acontecer o que esperávamos. Para fazer com que o navegador mostre o erro é necessário digitar javascript: na barra de endereços e teclar <enter>, o que causará a exibição da seguinte janela:



Com as informações desta janela, basta editar o arquivo, ir na linha e coluna indicada e corrigir o erro.

## 8. Objetos

### Programação Orientada a Objetos

Constantemente, vem sendo criadas técnicas para facilitar o desenvolvimento e a manutenção dos programas. Estas técnicas consistem principalmente em regras que, uma vez seguidas, agilizam e facilitam o processo de desenvolvimento. Mais que uma técnica, a programação orientada a objetos busca modificar a forma como o programador vê o problema a ser solucionado, criando uma abstração mais próxima do mundo real do que nas linguagens de programação mais antigas.

A programação orientada a objetos vê um problema como um conjunto de entidades (objetos) que interagem. Cada entidade tem suas características próprias (atributos ou propriedades) e faz a interação com outros objetos por meio de uma interface (métodos).

Na perspectiva tradicional de resolução de problema, primeiro se decide quais as operações (funções) serão efetuadas, depois se pensa em quais os dados estarão envolvidos. Numa perspectiva orientada a objetos, primeiro se identificam as entidades envolvidas para depois pensar na interação entre elas.

Infelizmente, fora a capacidade de criar e utilizar objetos Javascript não contem características que a permitam definir como uma linguagem orientada a objetos.

### Entendendo objetos

Como foi dito anteriormente, um objeto tem variáveis que descrevem o seu estado (propriedades) e interagem com os demais objetos por meio de funções (métodos). Um método é uma função ligada diretamente a uma classe de objetos e escrita para manipular suas propriedades. A forma mais fácil de compreender um objeto é através de um exemplo: é possível definir qualquer círculo a partir de sua posição (par de coordenadas x e y) e o seu raio; dado

um círculo é possível desenhá-lo ou verificar se um ponto (definido pelas coordenadas px e py) está em seu interior ou não. Sendo assim, para o objeto círculo:

Propriedades	Métodos
x	desenha()
y	dentro(px, py)
raio	

Para ter acesso aos métodos e propriedades de um objetos utiliza-se o operador “.” (ponto). Assim, por exemplo, para o objeto círculo teríamos:

```
circulo.x = 10;  
circulo.y = 20;  
circulo.raio = 50;  
circulo.desenha();
```

Em javascript existem 3 tipos de objetos:

- Objetos embutidos ou predefinidos (Date, Math, Array, String)
- Objetos do browser (Window, Document, Navigator)
- Objetos criados pelo programador

Durante este curso trataremos apenas dos dois primeiros tipos de objetos. Os objetos do browser (e o objeto Math) são previamente criados e já os encontramos a nossa disposição quando o programa começa a ser executado. Nos exemplos anteriores, inclusive, já foi utilizado o objeto document e invocado um de seus métodos (write). Os objetos embutidos Date e Array funcionam como um tipo de dado e é possível criar variáveis para armazená-los. Diferente porém dos outros tipos, é necessário utilizar o operador new para fazer a inicialização do objeto antes de poder utilizá-lo:

```
data = new Date;  
vetor = new Array(10);
```

A seguir são descritos os objetos embutidos, suas propriedades e seus métodos.

## Strings

Em capítulos anteriores já se falou do objeto string. Este objeto embutido, diferente de outro, não necessita ser inicializado através do operador new. Viu-se também que é possível utilizar o operador + com o sentido de concatenação, como no exemplo:

```
dados = nome + ":" + telefone;
```

Propriedades	
length	Tamanho da string
Métodos	
charAt (i)	Retorna o i-esimo caracter da string (começa por 0).
indexOf (s, i)	Retorna a posição da string s na string, começando a busca da i-esima posição.
lastIndexOf (s, i)	Retorna a posição da string s na string, começando a busca da i-esima posição para o início.
substring (i, f)	Retorna a string entre o i-esimo e o f-esimo caracter.
toLowerCase()	Converte um string para imprimí-la em minúsculas (não modifica a variável).
toUpperCase()	Converte um string para imprimí-la em maiúsculas (não modifica a variável).
split(s)	Retorna um Array com cada palavra da string usando a string s como separador.

## Arrays

O objeto embutido Array permite criar variáveis indexadas capazes de armazenar um conjunto de valores. Antes de utilizar um Array é preciso criá-lo



utilizando o operador new, indicando o seu tamanho (número de elementos que pode guardar). Este tamanho do Array pode ser posteriormente modificado. Para acessar cada elemento de um Array utiliza-se um índice entre colchetes após o nome da variável e a primeira posição é sempre 0. O exemplo a seguir mostra a criação de um Array de 4 posições e a atribuição de valores a estas posições:

```
valores = new Array(4);
valores[0] = 39;
valores[1] = 40;
valores[2] = 100;
valores[3] = 49;
```

Propriedades	
length	Tamanho do Array
Métodos	
join(s)	Retorna uma string através da junção de cada elemento do Array usando a string s como separador de palavras.
sort()	Ordena um Array.

## Math

Este objeto é utilizado para armazenar constantes úteis a operações matemáticas e funções para efetuar diversos tipos de cálculo.

Propriedades	
E	Constante de Euler (2.718).
PI	Valor de PI (3.1415...)
Métodos	
abs(n)	Valor absoluto de n.
sin(a), cos(a), tan(a)	Seno, cosseno e tangente do ângulo a (em radianos).
exp(n), log(n)	$E^n$ e logaritmo de n.

ceil(n)	Arredonda um número n para cima.
floor(n)	Arredonda um número n para baixo.
round(n)	Arredonda um número n para o valor mais próximo.
max(a,b), min(a,b)	Máximo e mínimo entre dois números.
pow(n, e)	Retorna n <sup>e</sup> .
sqrt(n)	Raiz quadrada de um número.
random()	Gerador de números aleatórios.

### Alguns exemplos:

```
radianos = graus * Math.PI / 180;
numero = Math.ceil (Math.random() * 100) -1;
x= -b + Math.sqrt(Math.pow(b,2) - 4*a*c);
```

## Date

O objeto Date permite trabalhar com datas e horários, como a data atual ou a diferença entre duas datas. Este é um dos objetos que precisa ser criado através do operador new. Há 4 formas de criar uma data:

- Data atual do sistema

```
dataHoje = new Date() ;
```

- Utilizando uma string com a data

```
data = new Date("1 15, 1997 12:02:00");
```

- Através do ano, dia e mês

```
data = new Date(1997, 1, 15);
```

- Através do ano, dia, mês, hora, minuto, segundo

```
data = new Date(1997, 1, 15, 14, 02, 12);
```

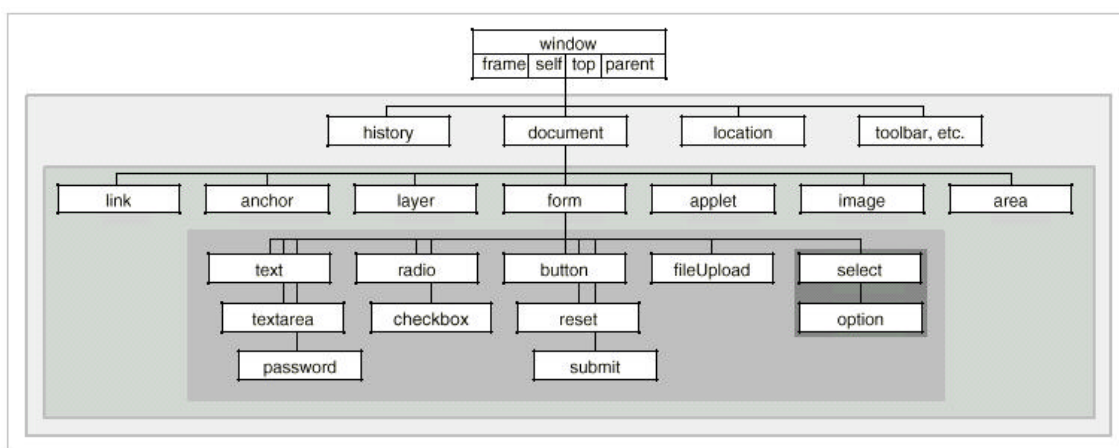
O objeto Date tem uma série de métodos que permitem a manipulação de cada parte de uma data em separado.

<b>Métodos</b>	
getDate() / setDate()	Obtém / define o dia do mês
getDay() / setDay()	Obtém / define o dia da semana (de 0 a 6).
getHours() / setHours()	Obtém / define a hora.
getMinutes() / setMinutes()	Obtém / define o minuto.
getMonth() / setMonth()	Obtém / define o mês (de 0 a 11).
getSeconds() / setSeconds()	Obtém / define os segundos
getTime() / setTime()	Obtém / define o nº de miliseg. desde 01/01/70
getFullYear() / setFullYear()	Obtém / define o ano.
getFullYear() / setFullYear()	Obtém / define o ano com 4 dígitos.

## 9. Hierarquia de Objetos do browser

Os objetos do browser são criados pelo próprio browser de forma a refletir características do navegador e da página sendo exibida. Há dois objetos básicos: navigator e window. O objeto navigator representa o próprio navegador e através dele é possível controlar o browser e obter informações sobre suas características. O objeto window representa uma janela do browser e contém nele todos os elementos da janela.

A figura abaixo mostra a hierarquia de objetos dentro de window, onde cada objeto ligado a um objeto acima representa uma relação de “contido em”:



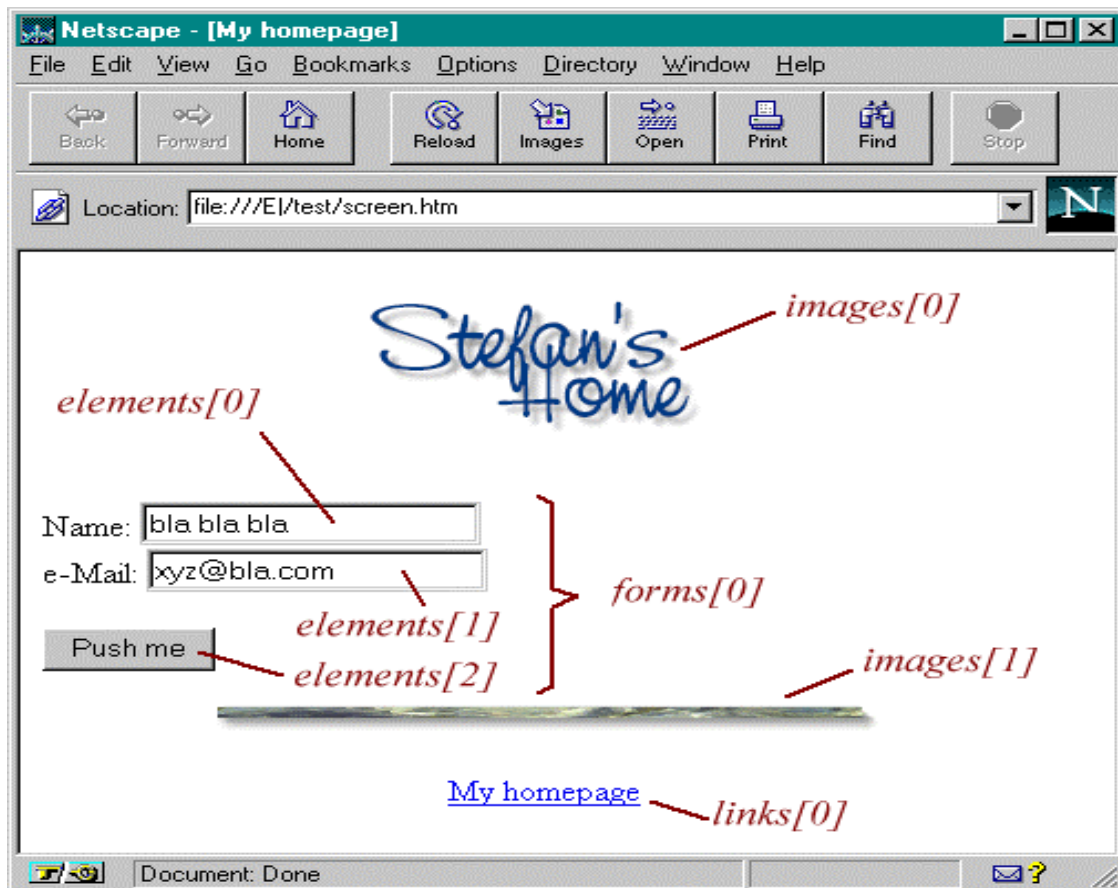
A hierarquia acima significa que há um objeto window e dentro deste objeto, na forma de propriedades, encontramos os objetos history, location, document e toolbar. Para acessar a qualquer um deles é necessário fazer:

```
window.history.back();
window.location = "http://www.nce.ufrj.br";
window.document.write("<H1>Minha Home Page</H1>");
```

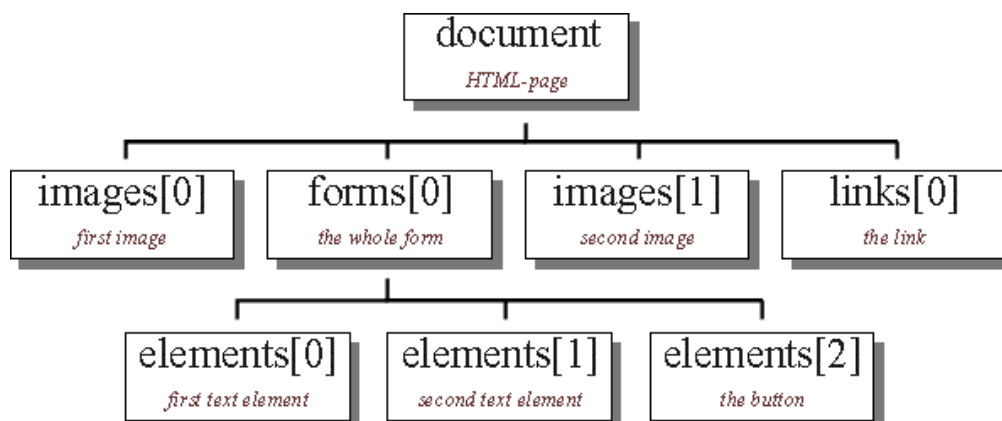
Por uma questão de simplificação, o Javascript permite referenciar os objetos internos a window sem citar a própria window. Assim é possível fazer:

```
history.back();
location = "http://www.nce.ufrj.br";
document.write("<H1>Minha Home Page</H1>");
```

Para melhor compreender a hierarquia de objetos dentro de window é melhor observar um exemplo real:



Esta janela produziria uma window com os seguintes elementos:



Um document tem um Array de imagens, um de formulários e um de links. O número de elementos de cada um destes tipos contidos na janela indica o número de elementos de cada Array. Assim, a página do exemplo contém duas imagens e o Array images contém dois objetos, um para cada imagem

numeradas de acordo com a ordem que aparecem na página. Um formulário por sua vez tem um Array de elementos, onde cada elemento pode ser um dos elementos de interação possíveis de pertencer a um formulário.

## Comando with

Neste momento torna-se necessário introduzir um novo elemento que facilita a manipulação dos objetos, permitindo uma economia de digitação quando se deseja referenciar várias propriedades e métodos de um mesmo objeto. Ao indicar um objeto entre os parênteses, os métodos e propriedades dentro das chaves vão ser considerados como pertencentes ao objeto indicado (se for possível). Assim no exemplo abaixo é possível escrever:

```
with (document)
{
    fgColor = "#000000";
    bgColor = "#FFFFFF";
}
```

No lugar de:

```
document.fgColor = "#000000";
document.bgColor = "#FFFFFF";
```

## Comando for...in

O comando for tem uma outra sintaxe útil para simplificar a manipulação de propriedades de um objeto. Forma de percorrer as propriedades de um objeto. É possível através do for recuperar, um a um, os nomes das propriedades e posteriormente acessá-las como utilizando o objeto como um vetor. O laço a seguir imprime na página cada uma das propriedades do navegador:

```
for (prop in navigator)
    document.write(prop + " = " + navigator[prop] );
```

## Objeto location

O objeto location contém informações acerca da URL corrente. A propriedade mais óbvia deste objeto é a string da URL corrente: propriedade href. Por isso esta propriedade pode ser omitida como já vimos em exemplos anteriores. Cada pedaço da URL pode ser referenciado individualmente através das propriedades: protocol, host, pathname e search (dados de um formulário enviados pelo método get). Este objeto contém o método reload que permite recarregar a página corrente.

## Objeto History

O objeto history dá acesso a lista de URLs navegadas por uma janela ou frame. Este objeto contém as seguintes propriedades e métodos:

Propriedades	
length	Número de páginas no histórico.
current	URL da página atual.
next	URL da página posterior.
previous	URL da página anterior.
Métodos	
go(n)	Permite ir para a n-ésima página.
back()	Volta a página anterior.
forward()	Vai a página seguinte.

O exemplo a seguir ilustra a utilização do objeto history:

```
<form>
<input type=button value="<-- 2" onclick="history.go(-2)">
<input type=button value="Prev." onclick="history.go(-1)">
<input type=button value="Next" onclick="history.go(1)">
<input type=button value="2 -->" onclick="history.go(2)">
</form>
```

## Objeto Document

O objeto document serve para consultar e modificar características do documento atual.

Algumas Propriedades:

Propriedades	
bgColor	Cor de fundo (definida em <BODY>).
fgColor	Cor do texto (definida em <BODY>).
linkColor	Atributo LINK de <BODY>.
vlinkColor	Atributo VLINK de <BODY>.
alinkColor	Atributo ALINK de <BODY>.
lastModified	Data da última modificação.
referrer	URL do documento que chamou a página atual.
title	Interior da tag <TITLE>.
links[ ]	Array com os links (<a href=...>).
anchors[ ]	Array com as âncoras. (<a name=...>)
forms [ ]	Array com os formulários (<form> ... </form>).
images[ ]	Array com as imagens (<img src=...>).
Métodos	
write(v1, ... vn)	Imprime strings (v1... vn) no documento.
writeln(v1, ... vn)	Imprime strings (v1... vn) em uma linha do documento.
clear ()	Apaga tudo de um documento.
open ()	Cria um novo documento.
close ()	Termina de incluir coisas num documento.



## 10. Scripts dependentes do navegador

Devido as diferenças de implementação entre os principais tipos de navegadores, em muitas situações é necessário distinguir qual o navegador para saber que código utilizar. Uma forma de fazer isso é através do objeto navigator. Este objeto, como o próprio nome diz, informa uma série de informações acerca do browser que está sendo utilizado. A forma como este objeto está implementado difere do Explorer para o Netscape, mas pelo menos 3 propriedades são comuns e permitem a criação de documentos adaptáveis a cada tipo de navegador, são elas:

Propriedades	
appName	Nome do browser.
appVersion	Versão do browser.
platform	Tipo de máquina onde está sendo executado.

A técnica de criar código adaptável ao tipo do browser consiste em colocar o código dependente do navegador abaixo de um desvio condicional onde se testa o valor de appName. Como javascript é uma linguagem interpretada, o navegador só acusa o erro de não reconhecer um código quando o executa e o desvio condicional evita que isso ocorra. O exemplo abaixo mostra como incluir código dependente do navegador:

```
ns = (navigator.appName.indexOf("Netscape") != -1);
ie = (navigator.appName.indexOf("Explorer") != -1);
function click(ev)
{
    if (ns)
        qual = (ev.which == 1) ? "esquerdo" : "direito";
    else if (ie)
        qual = (event.which == 1) ? "esquerdo" : "direito";
    else
        return false;
    alert("Você clicou o botão " + qual);
    return false;
}
```

## 11. Eventos

Um evento é um acontecimento envolvendo alguma atitude do usuário (o movimentar do mouse, o pressionar de uma tecla, o envio de um formulário, etc) ou o próprio funcionamento do navegador (o carregamento de uma página para a exibição, não conseguir carregar uma imagem, etc). A linguagem Javascript permite associar trechos de código ou chamadas de uma função a alguns destes eventos. A versão 1.2 incorporou o objeto event que permitiu o tratamento de um grande número de novos eventos e uma flexibilidade maior para tratá-los.

Cada objeto possível de incluir em uma página HTML contém um determinado conjunto de eventos associados. Para definir o código de tratamento de um evento é necessário definir um atributo na tag relativa ao elemento. Para cada tipo de evento há um atributo específico. A seguir são mostrados alguns destes atributos:

Atributo	Descrição
onClick	Clique do mouse sobre o elemento.
onMouseOver	Passar o curso do mouse sobre o elemento
onMouseOut	Mouse não está mais sobre o elemento
onMouseDown	Botão do mouse pressionado
onChange	Modificação do conteúdo
onLoad	Página foi carregada
onError	Erro ao carregar imagem
onSubmit	Formulário enviado.
onKeyDown	Tecla pressionada

É importante notar que não é possível usar qualquer atributo com qualquer elemento. Há uma lista de atributos que faz sentido para cada tipo de elemento. É necessário consultar um guia de referência, como em

<ftp://ftp12.ba.best.com/pub/dgoodman/NS4Map.zip>, para saber qual evento é válido para qual elemento.

O exemplo a seguir ilustra como associar eventos a um elemento:

```
<A HREF="http://www.jsworkshop.com/"
  onMouseOver="window.alert('You moved over the link.');">
Click here</A>

<INPUT TYPE="text" NAME="identidade"
  onChange="validaIdentidade(this)">
```

## O objeto event

Este objeto está disponível a partir do Javascript 1.2 e permite que a função de tratamento do evento possa obter mais informações sobre ele. Por exemplo, ao detectar o pressionar de uma tecla, através do objeto event pode-se saber qual foi a tecla pressionada.

Propriedade	Descrição
type	Tipo do evento
target	Destino de um evento
which	Botão do mouse (1, 2, 3) ou código ASCII da tecla
modifiers	Shift, Control ou ALT pressionado
pageX, pageY	Posição dentro da página
screenX, screenY	Posição em relação ao vídeo.

O objeto event é criado automaticamente pelo navegador. No exemplo a seguir o objeto event é passado como parâmetro para a função elefante que é chamada quando o botão do mouse for pressionado:

```

```

No Netscape é possível forçar que todos eventos de um determinado tipo sejam direcionados apenas para uma função associada ao evento no objeto

document. Para isso utiliza-se a função `document.captureEvents`. Esta função não pode ser utilizada no explorer, causando erro na página pois o tipo de constante passada como parâmetro para o método não está definido.

## 12. Janelas e Frames

O principal objeto da hierarquia do Javascript é o objeto Window e dentro dele encontram-se todos os elementos de uma página HTML que podem ser manipulados através de Javascript. Suas principais propriedades são os arrays de formulários, links, âncoras e imagens da página. Além destes arrays de objetos, uma Window contém muitas outras propriedades, entre elas:

Propriedades	
defaultStatus	Mensagem mostrada na barra de status.
status	Mensagem temporária na barra de status.
name	Nome da janela.
opener	Janela onde foi aberta a janela atual.
self	A própria janela.
location	URL da página sendo exibida.
Métodos	
open(s1, s2, s3)	Abre uma janela carregando o documento s1, tendo como nome s2 e de acordo com as propriedades indicadas em s3.
close ( )	Fecha uma janela.
blur ( )	Retira o foco de uma janela.
focus ( )	Coloca o foco numa janela, movendo-a para frente de todas.
setTimeout (f, t)	Executa a função f após s segundos.
clearTimeout ( )	Desprograma o timer.

Foram omitidos na tabela acima os métodos alert, prompt e confirm que já foram apresentados num dos primeiros capítulos. Assim como a propriedade location, estes métodos podem ser invocados sem a necessidade de se indicar a Window. As duas linhas abaixo são portanto equivalentes:

```
alert("mensagem ao usuário !");  
window.alert("mensagem ao usuário !");
```

## Janelas secundárias

Abrir um documento em uma janela diferente da atual é útil em diversas situações. Pode se usar este artifício para abrir documentos fora de nosso site e dessa forma sua página não é sobrescrita por um documento alheio. Outra utilidade é abrir uma janela com instruções para preencher um formulário, se contudo apagar o próprio formulário

A forma de abrir uma nova janela em Javascript é através do método `open` do objeto `window`. Este método recebe 3 parâmetros: o documento a ser carregado na nova janela (uma string vazia permite a criação de uma página em branco), um nome para ser atribuído a propriedade `name` (não pode conter espaços) e um terceiro parâmetro opcional com a lista recursos (separados por vírgula e sem espaços) que devem ser incluídos na janela. A tabela a seguir mostra alguns dos recursos que podem ser definidos:

Recursos	
height, width	Altura e largura da janela.
titlebar	Se a janela terá ou não área de título (yes ou no).
status	Se a janela terá ou não barra de status (yes ou no).
scrollbars	Se a janela terá ou não barras de rolagem (yes ou no).
resizable	Se a janela pode ou não ser redimensionada (yes ou no).
menubar	Se a janela terá ou não menu (yes ou no).
location	Se a janela terá ou não a barra de endereços (yes ou no).

O exemplo a seguir ilustra a criação de uma janela de dimensões 400x350, com `scrollbars` e sem `toolbar`, área de endereço e barra de status. Esta janela será posteriormente referenciada pela variável `jan`:

```
var jan = window.open("outraPag.html", "janelaNova",
    "toolbar=no,location=no,status=no,
    scrollbars=yes,width=400,height=350");
```

Como a janela principal, as janelas criadas através do open também tem um document e com o método write deste objeto é possível escrever nela. O método focus faz com que a janela seja exibida a frente das outras janelas (esmo daquelas que não tem nada a ver com o navegador). Para fechar a janela utiliza-se o método close.

## Limites de tempo

O método setTimeout permite programar a execução de uma função após uma determinada quantidade de milissegundos. Este método faz com que a função seja executada apenas uma vez. Para executar um número indeterminado de vezes basta incluir dentro da função uma nova chamada ao setTimeout. É possível que seja necessário interromper a seqüência de execuções após alguma condição, o que é feito através do método clearTimeout. As duas linhas de código abaixo mostram como deve ser utilizados estes métodos:

```
temporizador = window.setTimeout("instrucao",tempo);
```

```
window.clearTimeout (temporizador);
```

## Frames

Vimos anteriormente que é possível dividir uma janela em várias partes independentes, chamadas frames. Em Javascript, um frame se comporta exatamente como uma janela, tendo as mesmas propriedades e métodos. O objeto window tem um array com os frames definidos dentro da janela. O exemplo a seguir mostra um arquivo de layout que divide a janela em 3 frames:

```
<HTML><HEAD><TITLE>Exemplo de frames</TITLE></HEAD>
<FRAMESET COLS="75%,25%">
  <FRAME NAME="fra" SRC="pa.htm">
  <FRAMESET ROWS="33%,33%,*">
    <FRAME NAME="frb" SRC="pb.htm">
    <FRAME NAME="frc" SRC="pc.htm">
    <FRAME NAME="frd" SRC="pd.htm">
  </FRAMESET>
</FRAMESET></HTML>
```

Este arquivo de layout produziria a seguinte aparência:

fra	frb
	frc
	frd

Ao observar o arquivo de layout é possível notar que a divisão em frames é feita em dois níveis: no primeiro nível a janela é dividida em duas colunas (2 frames); no segundo nível, o segundo frame é subdividido em três linhas (3 frames). O objeto window tem portanto um array de dois elementos. O segundo elemento (window.frame[1]), por sua vez, tem um array de frames com três elementos. Cada frame pode ser acessado pelo array frame ou pelo nome definido no atributo name. As duas formas mostradas abaixo são válidas e tem o mesmo significado (escrever uma frase no frame de nome frb):

```
window.frames[1].frames[0].document.write("uma frase !");  
window.frames[1].frb.document.write("uma frase !");
```



## 13. Formulários

Uma das principais aplicações de Javascript é a possibilidade de criticar dados fornecidos pelo usuário através de formulários HTML. O conteúdo dos formulários pode ser acessado pelo script através do objeto FORM, guardados no array FORMS no objeto document. Um formulário também pode ser acessado pelo nome definido no atributo name.

```
<FORM NAME="meuform">  
</FORM>
```

Propriedades	
action	Atributo action do formulário.
method	Método de envio.
length	Numero de elementos do formulário.
Métodos	
submit( )	Envia o formulário ao servidor.
reset()	Limpa os campos do formulário.
Eventos	
onSubmit	Permite associar uma função ao submit do formulário.
onReset	Permite associar uma função ao reset do formulário.

É possível associar funções aos eventos de submit ou reset do formulário, que serão ativadas quando os respectivos botões forem selecionados. É possível, através da função associada ao submit fazer a verificação final nos campos do formulário, evitando o envio se ainda houver algum erro. Se alguma destas funções retornar FALSE a ação correspondente é cancelada. No exemplo a seguir o navegador pede uma confirmação se o usuário quer mesmo limpar um formulário:

```
onReset="return confirm('Quer mesmo apagar tudo?')"
```

## Elementos de um Formulário

O objeto FORM contém um array, onde são armazenados os elementos de interação do formulário. Um elemento pode ser acessado pelo array `elements` ou pelo nome definido no HTML. Por exemplo, dado o formulário abaixo:

```
<FORM NAME="meuform">  
<INPUT TYPE=TEXT NAME="endereco">  
</FORM>
```

As duas formas a seguir são equivalentes para referenciar a área de entrada de texto:

```
document.meuform.endereco  
document.forms[0].elements[0]
```

É possível determinar o número de elementos de um formulário através da propriedade `length` do elemento FORM. Como os elementos estão armazenados em um array (que também tem propriedade `length`), as duas formas abaixo são equivalentes:

```
document.forms[0].elements.length  
document.forms[0].length
```

Convém lembrar, porém, que FORMS também é um array e portanto se utilizarmos `length` sem indicar um índice estamos na verdade acessando o número de formulários do documento, como no exemplo abaixo:

```
document.forms.length
```

Cada posição do vetor `elements` pode conter qualquer um dos elementos de interação que vimos em HTML:

- Campos de texto: `text`, `password`, `hidden`
- Áreas de texto: `textarea`
- Botões: `button`, `reset`, `submit`
- Caixas de seleção: `checkbox`
- Botões de opção: `radio`
- Listas drop-down: `select`

## Campos de texto

Propriedades	
name	Nome associado ao elemento.
value	Valor digitado pelo usuário.
defaultValue	Valor a ser exibido no elemento após um reset.
Métodos	
focus( )	Coloca o elemento ativo (em foco).
blur()	Coloca o elemento inativo (remove o foco).
select()	Seleciona o texto no campo.
Eventos	
onFocus	Ocorre quando o campo recebe o foco.
onBlur	Ocorre quando o campo perde o foco.
onChange	Ocorre quando o valor do campo muda.
onSelect	Ocorre quando o usuário seleciona o texto do campo.

## Botões

Propriedades	
name	Nome associado ao elemento.
value	Valor exibido no botão.
Eventos	
onFocus	Ocorre quando o botão recebe o foco.
onBlur	Ocorre quando o botão perde o foco.
onClick	Ocorre quando o botão é selecionado com o mouse.
onMouseDown	Ocorre quando o botão do mouse é pressionado.
onMouseUp	Ocorre quando o botão do mouse é levantado.

## Checkbox

Propriedades	
name	Nome associado ao elemento.
checked	Booleano que indica se a checkbox está selecionada.
defaultChecked	Se a checkbox estará selecionada após um reset.
Eventos	
onFocus	Ocorre quando o campo recebe o foco.
onBlur	Ocorre quando o campo perde o foco.
onClick	Ocorre quando a checkbox é selecionada com o mouse.

## Grupo de botões de Radio

Um grupo de botões de radio é formado por vários elementos criados com a tag `<INPUT>`, `TYPE=RADIO` e o mesmo valor para o atributo `NAME`. Estes elementos são tratados em conjunto pois o seu comportamento depende do grupo (apenas um dos botões do grupo pode estar selecionado em um determinado instante de tempo).

Como neste objeto é sempre obrigatório a definição do atributo `name`, o grupo de botões de radio é manipulado através de um array com este nome no objeto `FORM`. Assim, se no formulário `form1` tivermos um grupo de botões de radio chamado `radio1`, cada exemplo a seguir mostra a sintaxe correta para:

- Saber o número de botões agrupados  
`document.form1.radio1.length`
- Verificar se o primeiro botão está selecionado  
`if (document.form1.radio1[0].checked)`
- Testar o atributo `value` do segundo botão  
`if (document.form1.radio1[1].value == ...)`

As propriedades deste objeto são muito parecidas com as do objeto checkbox:

Propriedades	
name	Nome associado ao elemento.
checked	Booleano que indica se o botão de radio está selecionado.
defaultChecked	Se o botão estará selecionada após um reset.
Eventos	
onFocus	Ocorre quando o botão recebe o foco.
onBlur	Ocorre quando o botão perde o foco.
onClick	Ocorre quando o botão é selecionado com o mouse.

### Listas de seleção (select e option)

Este tipo de elemento de interação envolve dois tipos de objeto: o objeto select, que representa a lista, e o objeto option, cada uma das opções.

Propriedades de select	
name	Nome associado ao elemento.
length	Número de opções da lista.
options	Array com as opções.
selectedIndex	Índice da opção atualmente selecionada. Se for uma lista de seleção múltipla é o índice da primeira seleção.
propriedades de option	
index	Índice no array.
defaultSelected	Se a opção estará selecionada após um reset.
selected	Booleano que indica se a opção está selecionada.
text	Texto exibido na opção.
value	Valor associado a opção.

Eventos associados a select	
onFocus	Ocorre quando a lista recebe o foco.
onBlur	Ocorre quando a lista perde o foco.
onClick	Ocorre quando a lista é selecionada com o mouse.

A linguagem javascript permite que o select seja modificado após a página ter sido carregada. É possível:

- modificar o texto que está sendo exibido na opção

```
document.fl.sl.options[2].text = "novo valor";
```

- modificar o valor atribuído a opção

```
document.fl.sl.options[1].value = "v1";
```

- criar uma nova opção

```
document.fl.sl.options[8] = new option("texto",
                                       "valor");
```

- remover uma opção

```
document.fl.sl.options[4] = null;
```

Para criar uma nova opção é preciso criar um novo objeto option através do operador new. O construtor de option pode receber até 4 parâmetros (opcionais):

- texto que será exibido
- valor da opção
- defaultSelect
- Se está selecionada

A remoção de uma opção é feita atribuindo à posição respectiva do array options o valor null. Qualquer opção pode ser removida, inclusive aquela que está atualmente selecionada.

## Área de texto

Este é um elemento de interação bastante versátil e pode ser utilizado para exibição de mensagens geradas durante a execução do javascript. Escrever mensagens numa área de texto tem uma vantagem em relação ao document.write, pois, mesmo que a página já esteja completa, é possível acrescentar texto à página sem que se perca o conteúdo anterior (um write após o carregamento da página faz com que seja criado um novo documento, apagando o anterior).

Propriedades	
name	Nome associado ao elemento.
value	Valor digitado pelo usuário.
defaultValue	Valor a ser exibido no elemento após um reset.
Métodos	
focus( )	Coloca o elemento ativo (em foco).
blur()	Coloca o elemento inativo (remove o foco).
select()	Seleciona o texto no campo.
Eventos	
onFocus	Ocorre quando o campo recebe o foco.
onBlur	Ocorre quando o campo perde o foco.
onChange	Ocorre quando o valor do campo muda.
onSelect	Ocorre quando o usuário seleciona o texto do campo.
onKeyDown	Ocorre quando uma tecla é pressionada.
onKeyUp	Ocorre quando uma tecla é solta.
onKeyPress	Pressionar e soltar uma tecla.

## Crítica do formulário

Numa página com um formulário, a utilização de javascript permite auxiliar o usuário e evitar que ele cometa erros de preenchimento. Este procedimento, que é chamado de crítica do formulário, pode ser realizado após a modificação do valor de algum elemento de interação (para isto utiliza-se, normalmente, o evento `onChange`) ou após o usuário ter pedido ao navegador para enviar o formulário (para isso utiliza-se o evento `onSubmit` do formulário ou o evento `onClick` do botão de submit).

Normalmente, é necessário fazer os dois tipos de verificação: a crítica do campo permite detectar erros imediatamente após o usuário tê-los cometido e a crítica do formulário permite detectar erros resultantes da interdependência de dois ou mais campos.

O exemplo a seguir ilustra como é feita a crítica de um campo do formulário. A função do exemplo testa se um campo de texto (no caso um endereço) foi preenchido e informa ao usuário que o campo é obrigatório caso ele não tenha sido definido:

```
function critica(campo)
{
    if (campo.value.length == 0)
        alert ('Campo ' + campo.name + ' não preenchido.');
```

...

```
<form ...>
<input type=text name=endereco onChange="critica(this)">
...
</form>
```

O exemplo a seguir ilustra a crítica final de um formulário. Este exemplo mostra um formulário onde deve-se digitar duas datas, com campos para dia, mês e ano.

Antes de enviar o formulário é feita a verificação se a data de saída é posterior a data de entrada. Isso só pode ser feito quando o formulário já está pronto pois não se pode forçar a ordem do usuário definir o valor dos campos :



```

function criticar()
{
    with (document.forms[0])
    {
        var dataent = new Date( ano_ent.value,
                                mes_ent.value,
                                dia_ent.value);
        var datasai = new Date(ano_sai.value,
                                mes_sai.value,
                                dia_sai.value);

        if (dataent.getTime() >= datasai.getTime())
        {
            alert("Saída deve ser posterior a entrada !");
            return false;
        }
    }

    return true;
}
. . .
<form action="/cgi-bin/x" onSubmit="return criticar()">
. . .
</form>

```

Alguns tipos de crítica podem servir inclusive para modificar o valor de um campo de forma a ficar coerente com o valor de outro. O exemplo a seguir mostra como a seleção da nacionalidade como estrangeira causa a limpeza do campo naturalidade:

```

<form name="meuform" ...>
    <P>Nacionalidade:<br>
    <input type="radio" value="1" name="nacionalidade">
        Brasileiro<br>
    <input type="radio" value="2" name="nacionalidade"
        onClick="document.meuform.naturalidade.value = "">
        Estrangeiro
    <P>Naturalidade:
    <input type="text" name="naturalidade"
        onChange="validaNaturalidade(this)"> <br>
...
</form>

```