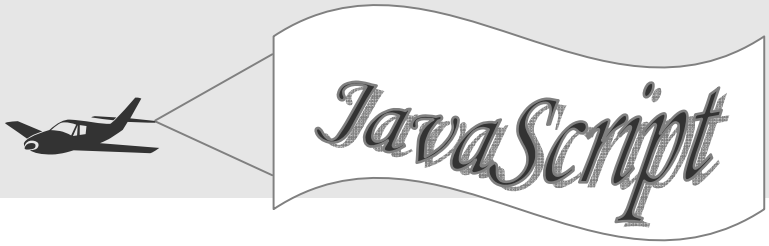
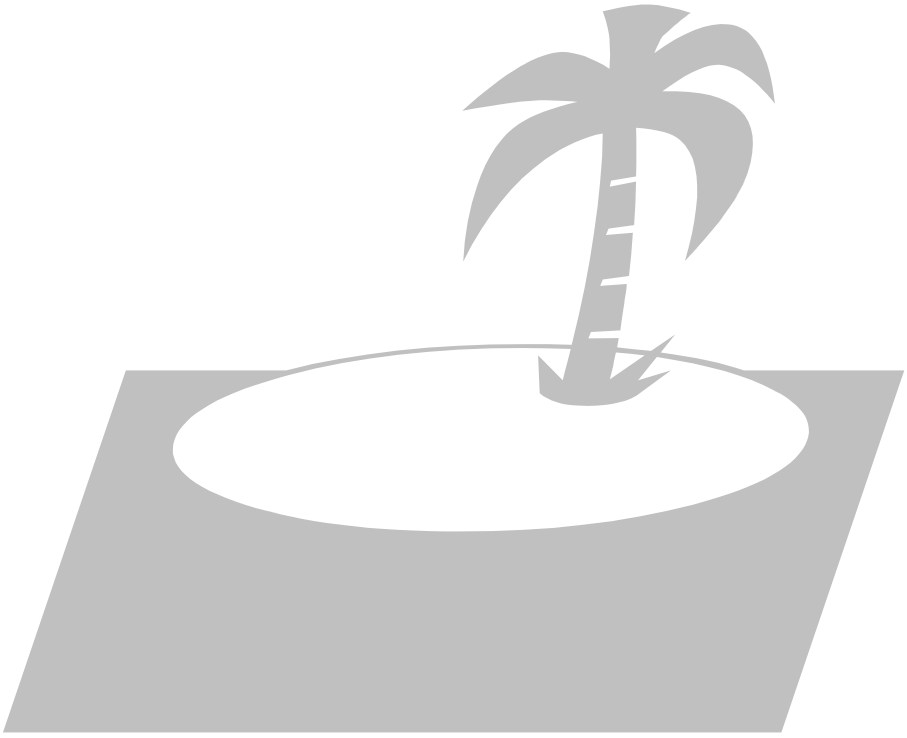




Apostila

Prof.a Elisabete da Silva Santos

Departamento de Tecnologia da Informação

FATEC-SP

JavaScript

JavaScript

Índice

1. Introdução

1.1. Histórico

JavaScript foi desenvolvido por Brendan Eich, da Netscape Communications Corporation. Originalmente chamado *LiveScript* e foi introduzido no Netscape Navigator 2.0 em 1995. Mais tarde, foi batizado como *JavaScript* para indicar seu relacionamento com Java, termo da *Web* muito atraente no momento. *JavaScript* foi a primeira linguagem de script da Web e é de longe a mais popular. Quando a Microsoft percebeu o quanto podia ser útil uma linguagem de script para a *Web*, lançou sua própria variante chamada JScript 1.0 com o navegador Internet Explorer 3.0.

O *JScript* 1.0 era compatível com o *JavaScript* 1.0 do Netscape, o que significava que um *JavaScript* escrito para um dos navegadores tinha uma boa chance de funcionar como esperado em um outro navegador. Após alguns *upgrades* de versões tanto da Netscape quanto, principalmente, da Microsoft, gerou-se o início de incompatibilidades de navegadores.

Foi criado então um padrão chamado de linguagem ECMAScript em 1997, chamado também ECMA-262, criado pelo grupo suíço European Computer Manufacturing Association. A partir daí, grande parte do caos ocorrido foi dando lugar a uma melhor concordância entre ambas as linguagens e os padrões. O *JavaScript1*.5 e *JScript* 5.5 atualmente estão sendo 100% concordantes com o ECMAScript Edição 3, pelo menos na teoria, se todas as pessoas estiverem usando a última versão do navegador...

1.2. O quê é JavaScript?

JavaScript é uma linguagem de script que lhe possibilita adicionar um novo nível de interatividade e função às páginas Web.
Um *script* é uma seqüência de instruções (ou, até mesmo um programa) que são interpretados ou executados por um outro programa e não pelo processador do computador. Eles são mais fáceis e mais rápidos de se escrever do que os programas e necessitam no mínimo de um editor de texto simples, que pode ser gratuito!
Diferentemente de HTML, se houver erros nos scripts o programa interromperá a execução e poderá até travar o computador. Normalmente você poderá visualizar a mensagem de erro gerado em seu navegador.

1.3. Java, C++ e JavaScript são a mesma coisa?

JavaScript é uma linguagem de terceira geração, o que significa que ela é prima do C, Pascal e BASIC. Existem semelhanças, mas existem também diferenças fundamentais:

- *JavaScript* é independente de formatação.
- *JavaScript* é uma linguagem interpretada.
- *JavaScript* é altamente portátil e independente de hardware.
- *JavaScript* se encaixa facilmente em outro software, como os navegadores.

Se você tiver alguma experiência em C, C++, Pascal, BASIC, Java, FORTRAN ou Perl, você pode achar familiares alguns dos aspectos do *JavaScript*. Mas não se iluda, para escrever *JavaScript*, você precisa usar Java*Script!*

1.4. O quê posso fazer com JavaScript?

- Você pode adicionar mensagens que rolam na tela ou alterar as mensagens da linha de status do navegador.
- Validar conteúdo de um formulário.
- Fazer cálculos;
- Trabalhar com datas, horas e fuso horário;
- Exibir mensagens para o usuário tanto como parte de uma página da *Web* como em caixas de alertas;
- Fazer animações de imagens ou imagens que mudam quando você move o cursor sobre elas;
- Detectar o navegador em utilização e exibir conteúdo diferente para navegadores diferentes;
- Detectar *plug*-ins instalados e notificar o usuário se um *plug*-in foi exigido;
- E muito mais, é só usar a criatividade!

1.5. O quê preciso para programar em JavaScript?

- Um editor de texto (não utilize formatação) ou um bloco de notas.
- Um navegador. (Netscape, Internet Explorer, Opera...)
- Conhecimentos básicos em HTML, onde o *script* será incluído.

***Curiosidade**: Java é uma ilha densamente povoada na Indonésia, produtora de café. Seu nome foi utilizado para batizar a linguagem Java quando seus desenvolvedores estavam tomando um cafezinho... Já pensou se o cafezinho fosse brasileiro?*

2. JavaScript em uma Página HTML

Para incluir Java*S*cript em uma página HTML utilizamos as tags **<script>** *conteúdo* **</script>**.
Podemos também indicar a linguagem de script a ser utilizada e a versão:

<script language="javascript"> ... **</script>** // *JavaScript é a linguagem padrão do navegador.*

<script language="javascript1.3"> ... **</script>** // 1.3 é *a versão JavaScript utilizada.*

Atenção:

- O *script* é sensível a letras maiúsculas e minúsculas (*sensitive case*), portanto, digite em minúsculo o que estiver em minúsculo e em maiúsculo o que estiver em maiúsculo!
- Não utilize caracteres especiais (acentos, asteriscos, pontos...), exceto como conteúdo de variáveis!

Podemos incluir as *tags* de *Javascript* em vários lugares do código HTML:

- Em um Arquivo Externo
- No Cabeçalho do Documento
- Dentro de *Tags*, utilizando Tratadores de Eventos
- No Corpo do Documento

Exemplos:

```
1. <html>
     <head>
       <title> Javascript1 </title>
         <script language="javascript">
            alert("Seja bem-vindo(a)!");
           // JavaScript no Cabeçalho da página HTML.
         </script>
     </head> ...

2.  <html>
     <head>
       <title> Javascript2 </title>
         <script language="javascript" src="arquivo_externo.js">
          // Chamando um Arquivo Externo contendo JavaScript.
         </script>
     </head> ...

3.  <html>
     <head>
       <title> Javascript3 </title>
     </head>
     <body>
       <input type="button" value="Mensagem" onclick="alert('Ola!');">
       <!-- JavaScript em uma "tag" HTML através de um Tratador de Eventos.
       -->
     </body> …

4.   <html>
     <head>
       <title> Javascript4 </title>
     </head>
     <body>
       <script language="javascript">
         document.write("Iniciando em JavaScript");
         // JavaScript no Corpo de uma página HTML.
       </script>
     </body> ...
```

2.1. Tags de Comentário:

Nós podemos escrever anotações dentro do código fonte sem que elas sejam executadas como comandos do *Javascript*, para isto, precisamos indicar que as linhas são apenas de comentários.

Exemplos:

```
//  Comentando uma linha.

/*  Comentando
    várias linhas... */
```

2.2. Ocultando JavaScript de Navegadores mais Antigos

Navegadores mais antigos, ou que tenham o *JavaScript* desabilitado, não reconhecem a *tag* `<script>`. Para que o código em *JavaScript* não seja exibido no meio da página, comente-o com *tags* de HTML:

```
<!-- Comentário em HTML -->
```

O navegador que suportar *JavaScript* ignorará os comentários em HTML e executará o *script*.

Exemplo:

```
<script language="javascript">
  <!--
    document.write("Seu navegador suporta JavaScript")
    //  -->
  </script>
```

Utilize o comentário do JavaScript (//) para comentar o final do comentário de Html!

2.1. Escrevendo em uma Página – document.write( )

Você deve ter percebido que utilizamos várias vezes nos exemplos o comando **`document.write( )`**. Em *JavaScritp* nós o chamamos de Método, porque ele executa uma função (ação), ou seja, com ele você pode escrever no documento, que é o corpo da página em HTML.

Exemplo:

```
<script language="javascript">
   document.write("Escrevendo no documento");
</script>
```

O conteúdo dos parênteses deve vir entre aspas ou apóstrofes, a menos que seja uma variável.
Todos os comandos em JavaScript terminam com ponto e vírgula (;).

3. Caixas Pop-up

As caixas Pop-up também são métodos em *JavaScript*. Com elas podemos interagir com o usuário exibindo mensagens, recebendo dados através da caixa de diálogo e confirmações ou não de informações.

3.1. Método alert( )

Exibe uma caixa de mensagem e aguarda um clique do usuário no botão <OK> para fechá-la.

Exemplo:

```
<script language="javascript">
   alert("Seja muito bem-vindo(a)!");
</script>
```

3.2. Método prompt( )

Abre uma caixa de diálogo para que o usuário possa entrar com dados.

```
1. <script language="javascript">
     prompt("Digite seu nome");
   </script>
```

Neste exemplo, o texto entre parênteses será exibido na parte superior da caixa pop-up e o campo de resposta aparecerá com a palavra "undefined" selecionado.

```
2. <script language="javascript">
      origem = prompt("Informe o seu país de origem","Brasil");
   </script>
```

Neste outro exemplo, o texto após a vírgula e dentro dos parênteses, será exibido no campo de resposta como resposta padrão, se preferir deixar a resposta em branco, não inclua conteúdo nem espaços entre as aspas.
A resposta poderá ser armazenada em uma variável para ser tratada posteriormente pelo programa.

3.3. Método confirm( )

Exibe uma caixa de confirmação contendo dois botões: *<ok>* (sim) e *<cancel>* (não).

Exemplo:

```
1. <script language="javascript">
     resp = confirm("Será que vai chover?");
   </script>
```

A resposta também poderá ser armazenada em uma variável para ser tratada posteriormente pelo programa.

4. Caracteres de Escape

Os caracteres de escape são usados em *Strings* para:

- Evitar que certos caracteres causem erros dentro do programa;
- Adicionar formatação (como retorno de carro) a *Strings*.

Caracter	Descrição
\b	*Backspace*
\t	*Tabulação Horizontal*
\f	*Avanço de Formulário (form feed)*
\n	*Nova Linha (line feed)*
\r	*Retorno de Carro*
\"	*Aspas Duplas*
\'	*Apóstrofe*
\\	*Barra invertida*

Exemplos:

```
1. <script language="javascript">
     alert("Um retorno de carro está\rbem no meio desta linha!");
     alert("\"Isto não saiu como deveria!\" disse ela");
     alert("Esta linha tem uma tabulação\taqui.");
   </script>
```

5. Variáveis

5.1 Globais e Locais

Criar variáveis em *JavaScript* é muito fácil, é dar um nome e atribuir um valor a ela e estará criada!

Exemplo:

```
Nome = "Maria";
```

Esta variável será global, ou seja, você terá acesso a ela a partir de qualquer *script* relacionado a este documento.

Você poderá criar variáveis desta outra forma também, utilizando a palavra chave *var*:

```
var Nome1 = "José";
```

A variável `Nome1` será global se estiver fora de uma função, caso contrário será local, ou seja, você só poderá acessá-la dentro da função onde ela foi criada.

- *Variáveis globais têm o script inteiro como escopo.*
- *Variáveis locais têm uma única função como escopo.*

Exemplos:

```
<script language="javascript">
   num = 1234;             // Criando e atribuindo um valor à uma variável.

   var msg1;               // Criando uma variável por linha.
   var msg2;

   var num1, nome1, msg1;  // Criando todas as variáveis em uma única linha.

   var nome2 = "João", num2 = "12", msg2 = "telefone do João: 1234-5678";
   // Criando todas as variáveis e atribuindo valores, em uma única linha.

   function cliente(){
     var nome3="João";     // Criando uma variável local dentro de uma função
   }
</script>
```

5.2. Nomes de Variáveis

- Nomes de variáveis só podem conter letras maiúsculas e/ou minúsculas, números e sublinhado, portanto, não podem conter espaços, hífens ou outros caracteres especiais;
- O primeiro caractere do nome da variável deve ser uma letra ou o sublinhado;
- *Javascript* é "*case sensitive*", portanto, letras maiúsculas são diferentes de minúsculas. Atenção!!!
- Não há limite oficial no comprimento de nomes de variáveis, portanto, vale o bom senso.
- Utilize sempre nomes mnemônicos, ou seja, que traduzam o conteúdo da variável.

Exemplos de nomes válidos:

```
Total_geral
nome3
_num1
i (que é diferente de I)
```

6. Tipos de Dados em JavaScript

Em *JavaScript* você não precisa definir uma variável especificando o tipo de dados que ela armazenará (exceto em casos raros), ela se adaptará ao tipo de dados do conteúdo a ela atribuído.

Tipos de Dados do JavaScript:

6.1. Básicos

- `Number` – Números Inteiros ou de Ponto Flutuante.
  Ex.: `0; 1; -3; 42; 3,14159265358979323846264338327795 ou 3.76e2.`
- `String` – Trata-se de uma cadeia de caracteres (entre aspas).
  Ex.: "Meu nome é Guilherme".
- `Boolean` – Booleanos ou Lógicos; possuem dois valores: Verdadeiro ou Falso.
  Ex.: `true; false.`

6.2. Especias

Valores numéricos especiais:

- `Positive infinite` – Infinito positivo.
- `Negative infinite` – Infinito negativo.
- `0` – zero positivo e negativo.
- `NaN` - *Not a Number* (Não é um Número). Usualmente é gerado como resultado de uma operação matemática que não faz sentido, por exemplo, divisão de qualquer número por zero.

Não há possibilidade de digitar nenhum dos valores acima. Os valores relacionados com infinito resultam quando se ultrapassa o limite de 10^308 ou 10^ -308. Exemplo, multiplicando 1.0e300 por si mesmo. Eles não são muito úteis para nós.

Tipos especiais:

- `Null` – Valor nulo, quando não há nada de útil disponível.
- `Undefined` – Indefinido. Na maioria das vezes é o mesmo que *null*. Sua presença indica que algo saiu errado em seu *JavaScript*.

7. Conversão entre Tipos de Dados

O *JavaScript* trata conversões entre tipos de dados para você sempre que ele pode. Exemplo:

```
total= 40;
document.write("O total é " + total);
```

A instrução imprimirá a mensagem: "O total é 40".

A função de `document.write()` trabalha com *Strings*, portanto, o interpretador de *JavaScript* automaticamente converte quaisquer não-*strings* na expressão em *Strings* antes de desempenhar a função.

A caixa *prompt*() também trabalha com *Strings*, portanto, antes de executarmos quaisquer cálculos com valores fornecidos através do *prompt*, devemos converter o tipo de *String* para números.

Às vezes, uma *String* pode conter um número e precisamos convertê-la em uma variável numérica regular. O *JavaScript* possui duas funções embutidas para transformar *Strings* em números:

7.1. Funções parseInt( ) e parseFloat( )

- `parseInt(string)` *//Converte uma string em um número inteiro.*
- `ParseFloat(string)` *//Converte uma string em um número com ponto flutuante.*

A duas funções efetuarão a leitura da *string* desde o seu início e retornarão uma versão numérica.

Exemplos:

1.
```
num_b = parseFloat(prompt("Digite um número",""));  // num_b será do tipo number.
alert(num_b + "é do tipo " + typeof(num_b));
```

2.
```
string_a = "45 anos de experiência!";
num_a = parseInt(string_a);
alert(num_a);      // A variável num_a armazenará o valor 45, a parte não numérica é ignorada.
```

O número que será capturado deverá estar no início da String.

- Essas funções são utilizadas para converte conteúdo das caixas de texto.

7.2. Outros métodos de conversão de tipos de dados - Number(), String(), Boolean()

- `String` *Converte o tipo de uma variável em String.*
- `Number` *Converte o tipo de uma variável em numérico.*
- `Boolean` *Converte o tipo de uma variável em booleano.*

Exemplo:

```
1. a = 2;
   b = String(a);      //Converte a variável numérica "a" em String.
   alert(typeof(b));
```

7.3. Fixando e Arredondado o número de casas "decimais" - toFixed( )

Para fixar e arredondar a quantidade de números após o valor inteiro, utilizamos o método *toFixed()*.

Exemplo:

```
1. a = 10/3;                    // A variável "a" recebe o resultado de 10 dividido por 3.
   document.write(a);
   a = a.toFixed(2);            // O argumento "2" define o número de casas "decimais".
   document.write("<br>"+ a);   // O valor de "a" será: 3.33
```

8. Operadores

Os operadores precisam agir sobre alguma coisa para que possam operar. Eles podem funcionar com um só dado, que os tornam operadores unários; com dois, binários ou com três, ternário.

Operadores mais comuns:

8.1. Aritméticos

Esses são os operadores matemáticos familiares (binários):

- **Adição (+)**

```
a = 2 ,  b = 3;              //Lê-se: a recebe 1 e b recebe 3
c = a + b;
document.write("<br>"+c);    // o valor de "c" será 5
```

- **Substração (-)**

```
a = 3 ,  b = 1;
c = a - b;
document.write("<br>"+c);    // o valor de "c" será 2
```

- **Divisão (/)**

```
a = 8 ,  b = 2;
c = a / b;
document.write("<br>"+c);    // o valor de "c" será 4
```

- **Multiplicação (*)**

```
a = 2 ,  b = 3;
c = a * b;
document.write("<br>"+c);    // o valor de "c" será 6
```

- **Módulo ou Resto da Divisão(%)**

```
a = 5 , b = 2;
c = a % b;
document.write("<br>"+c);        // o valor de "c" será 1
```

8.2. Relacionais

Os operadores relacionais são usados em comparações (binários):

- **Menor que (<)**

```
a = 5 , b = 6;
alert(a < b);
```

- **Menor ou igual a (<=)**

```
a = 3 , b = 5;
alert(a <= b);
```

- **Maior que (>)**

```
a = 4 , b = 6;
alert(b > a);
```

- **Maior ou igual a (>=)**

```
a = 5 , b = 7;
alert(b >= a);
```

- **Igual a (==)**

```
a = 3 , b = 3;
alert(a = = b);
```

- **Não igual a (!=)**

```
a = 3 , b = 4;
alert(a != b);
```

8.3. Lógicos

Eles são utilizados em instruções condicionais. (&& e || são binários e ! unário)

- **E (&&)**

```
a = 6 , b = 3, c = 1;
alert((a > b) && (b > c));       // (a é maior que b) e (b maior que c)
```

- **Ou (||)**

```
a = 6 , b = 3, c = 1;
alert((a > b) || (b == c));      // (a é maior que b) ou (b é igual a c)
```

- **Negação (!)**

```
a = 6 , b = 3, c = 1;
alert(a != b);                   // ( a não é igual a b)
```

8.4. Unários Diversos

- **Incremento de prefixo e sufixo (++)**

```
1. a = 5;
   alert(++a + 2);          // Adiciona 1 à variável "a" antes de executar a expressão.
   alert("a = " + a);       // O resultado do alert será 8
2. a = 5;
   alert(c++ + 2);          // Adiciona 1 à variável "a" após a execução da expressão.
   alert("c = " + c);       // O resultado do alert será  7
```

- **Decremento de prefixo e sufixo (--)**

```
1. a = 5
   b = --a + 2;    // Subtrai 1 da variável "a" antes da execução da expressão, "b" receberá 6.
2. a = 5
   b = a-- + 2;    //Subtrai 1 da variável "a" após a execução da expressão, "b" receberá 7.
```

- **Unário (-)**

```
1. a = 2;
   b = -a;          //Muda o sinal de positivo para negativo, portanto "b" receberá -2.
2. c = -3;
   d = -c;          //Muda o sinal de negativo para positivo, "d" receberá 3.
```

- **Unário (+)**

```
+a                  //Muda o operando para o tipo número (por exemplo, era uma string).
```

8.5. Operadores de Atribuição

- **Atribuição Plena (=)**

```
a = 3;              // A Variável "a" recebe o valor 3.
a = b = c = 5;      // As variáveis "a", "b" e "c" recebem o valor 5.
```

- **Compostos:**

```
x =+ 3;             // É o mesmo que x = x + 3;
x =- 3;             // É o mesmo que x = x - 3;
x =* 3;             // É o mesmo que x = x * 3;
x =/ 3;             // É o mesmo que x = x / 3;
```

8.6. Operadores Diversos

- **Condicional - Ternário (? :)**

```
1. x = 1, y = 2;
   (x > y)? alert("Sim, x é Maior que y"): alert("Não, x é Menor que y");
```

Se a expressão condicional `(x > y)` for verdadeira, executará a sentença após a interrogação (?), caso contrário, executará a sentença após os dois pontos (:).

```
2. resp = (x > y)? "Sim" : "Não";        //A variável resp receberá "Não".
```

A variável "`resp`" é opcional; podemos criá-las se precisarmos salvar dados.

- **Concatenador de string - Binário (+)**

```
a = "São" , b = "Paulo";
c = a + b;              // O conteúdo de "c" será "São Paulo".
```

- **Operador de tipo - Unário (typeof)**

```
1. x = 3;
   alert(typeof(x));   // O tipo da variável será exibido na caixa de alerta; no caso, number".

2, mens = "Olá!";
   alert(typeof(mens)); // O tipo "string" será exibido na caixa de alerta..
```

9. Funções

Funções são grupos de instruções em *JavaScript* que podem ser tratadas como uma unidade.
Elas são executadas apenas quando o programa as solicita, uma ou mais vezes.

Normalmente definimos as funções dentro do cabeçalho da HTML e podemos chamá-las em qualquer parte do *script*.

9.1. Formato

Uma função é definida a partir da palavra-chave *function,* seguida de seu nome e de parênteses. O conteúdo da função (sentenças) deve estar inserido entre chaves.

```
<html>
 <head> <title> Funções </title>
  <script language="javascript">
     function nome_da_função(){
          sentença1;      // Linhas de instruções da função.
          sentença2;
          ...
    }
  </script>
 </head>

 <body>
  Corpo da página...
   <script>
    nome_da_função();     //Chamada da função para que ela possa ser executada.
   </script>
 </body>
</html>
```

Uma função pode receber parâmetros e/ou retornar valores (dados).

9.2. Função sem Parâmetros

A função é executada sem receber parâmetros e/ou retornar valores.

Exemplo:

```
<head>
  <script language="javascript">
     function mens(){      // Definição da função no cabeçalho da página.
       alert("Ola!");
     }
  </script>

<script language="javascript">

   mens();                 // Chamada da função a partir do corpo da página.
   ...
   mens();                 // Chamando a mesma função novamente...

</script)
```

9.3. Função recebendo Parâmetros

Uma função pode receber parâmetros. Os dados que serão enviados à função deverão estar inseridos entre os parênteses da chamada da mesma.
A função receberá e armazenará os dados nas variáveis contidas entre os parênteses de sua definição, na respectiva ordem em que foram enviados.

Exemplo:

```
<script language="javascript">

  function mens(quem1 , quem2){      // Definição da função e das variáveis quem1 e quem2.
    alert("Ola " + quem1 + ",olá " + quem2);
  }
</script>
```

Chamando a função:

```
<script language="javascript">
   nome1 = "Pedro";
   nome2 = "Paulo";
   mens(nome1,nome2);     //Chamada da função mens() passando os parâmetros nome1 e nome2.
</script)
```

A função `mens()` será chamada e enviará o conteúdo das variáveis `nome1(Pedro)` e `nome2` (Paulo) para serem recebidos pelas variáveis `quem1` e `quem2`.

A função será executada e exibirá na caixa de alerta uma saudação para "Pedro" e "Paulo".

9.4. Função retornando Valores

As funções podem retornar valores para o ponto de sua chamada.

Exemplo:

```
<script language="javascript">
  function media(a,b,c){        // Função recebendo parâmetros em a, b e c.
       calculo =(a+b+c)/3;      // Cálculo dos valores armazenados nas variáveis.
       return calculo;          // Função retornando o resultado do cálculo.
  }
</script>
<script language="javascript">
  result = media(3,5,8);   // Chamada da função media() passando os valores (3, 5, e 8).
  alert(result);           // Exibindo o retorno do cálculo que foi atribuído a variável result.
</script)
```

O valor contido na variável calculo foi enviado de volta para a chamada da função. Na chamada da função fez-se a atribuição do valor retornado sobre ela para a variável result.

10. Strings

JavaScript armazena *Strings* como *objetos do tipo String*. Os objetos possuem métodos e propriedades, genericamente, tudo o que o objeto pode fazer (ação) e tudo o que o objeto possui (características).

10.1. Objeto Strings

Podemos criar objetos *String* de duas formas (vide exemplos a seguir): da primeira forma nós já criamos, para criarmos de acordo com a segunda forma, vamos utilizar a sintaxe oficial de objetos:

```
1. teste = "Isto é um teste";

2. teste = new String("Isto é um teste");
```

Na segunda forma utilizamos a palavra-chave `new`, solicitando ao navegador que crie um novo objeto do tipo *String* contendo o texto "`Isto é um teste`" e atribua-o à variável `teste`.

Podemos utilizar o operador de concatenação para combinar valores de duas *Strings*:

```
teste1 = "Isto é um teste. ";
teste2 = "Somente um teste";
teste3 = teste1 + teste2;
alert(teste3);    // O resultado exibido será: "Isto é um teste. Somente um teste"
```

Podemos utilizar também o operador `=+` para adicionar conteúdo em uma string:

```
teste3 =+ "!";   // Esta instrução adiciona uma exclamação ao conteúdo atual da string teste3.
```

10.2. Comprimento de String - length

Para saber quantos caracteres uma variável de *String* armazena podemos utilizar a propriedade `length` do objeto *String*. Digitamos o nome do objeto *String* seguido por `.length`:

```
frase = "Este é o meu Brasil.";
document.write(frase.length);      //A String frase possui 20 caracteres.
```

Os espaços também são incluídos na contagem de caracteres.

10.3. Conversão de String – toUpperCase( ) e toLowerCase( )

Existem dois métodos que convertem o texto de caixa alta para caixa baixa. São eles:
toUpperCase( ) - Converte todos os caracteres em letras maiúsculas.
toLowerCase( ) - Converte todos os caracteres em letras minúsculas.

Exemplos:

```
1. frase1 = "Ouvindo você esquece. Lendo você aprende. Fazendo, você sabe!";
   alert(frase1.toUpperCase( ));
```

Exibe: OUVINDO VOCÊ ESQUECE. LENDO VOCÊ APRENDE. FAZENDO, VOCÊ SABE!

```
2.  alert(frase1.toLowerCase( ));
```

Exibe: ouvindo você esquece. lendo você aprende. fazendo, você sabe!

10.4. Substrings – substring( )

O *JavaScript* nos permite trabalhar com *substrings* ou seja, partes de uma *String*.
O método `substring( )` retorna uma parte da *string* original, formando uma outra *String*.
A *substring* é localizada a partir de dois índices, de início e fim, informados entre os parênteses:

Formato: `objeto.substring(ii,if+1);` sendo: `ii` – índice inicial e `if` – índice final (mais 1).

Exemplo:

```
frase2 = "Os seus  olhos";
          0 1 2 3 4 5 6 7  8 9 10 11 12
document.write(frase2.substring(3,7);    //Imprime a substring: "seus".
```

- Cada caractere possui um índice.
- Os índices são seqüenciais e iniciam-se em zero (0).
- Espaços em branco também possuem índices.
- O primeiro índice do método indica o início da *substring*;
- O segundo índice indica o término da *substring* e é exclusive.
- Os índices podem aparecer em qualquer ordem, o menor será assumido como o inicial.
- Sempre que os dois índices forem iguais o resultado será uma *string* vazia.

10.5. Obtendo um Único Caractere - charAt( )

O método charAt( ) obtém um único caractere de uma string:

```
palavra = "Crepúsculo";
document.write(palavra.charAt(3);          //Imprime a letra "p".
```

- 0 (zero) é o índice inicial da *String*.
- O método charAt() obtém o caractere correspondente ao índice indicado entre parênteses.

10.6. Localizando uma Substring - indexOf( )

Uma boa utilização de *subtrings* é localizar uma *string* dentro de outra *String*.
Podemos fazer isto através do método indexOf( ). Exemplo:

```
frase3 = "Brasil, meu Brasil brasileiro...";
ind = frase3.indexOf("Brasil");
document.write(ind);                    //Imprime o índice inicial da substing: 0.
```

- Entre os parênteses indicamos a *substring* que queremos localizar no texto.
- Atenção com as letras maiúsculas, minúsculas e acentos!!
- O método retornará o índice inicial da *substring* pertencente ao texto.
- Se a *substring* não for localizada, retornará o valor negativo -1.

Você pode especificar um segundo parâmetro (opcional) para indicar o índice inicial da procura, a partir do qual começará a pesquisa. Exemplo:

```
frase3 = "Brasil, meu Brasil brasileiro...";
document.write(frase3.indexOf("Brasil",1));     //Imprime índice:12.
```

10.7. Localizando a Última Ocorrência de uma Substring – lastIndexOf( )

O método *lastIndexOf()* pesquisa uma *substring* iniciando a busca a partir do último caractere da *string:*

```
frase3 = "Brasil, meu Brasil brasileiro...";
document.write(frase3.lastIndexOf("Brasil"));     //Imprime índice:12.
```

Neste caso, também podemos especificar um segundo parâmetro (opcional) para indicar o índice inicial da próxima procura. Exemplo:

```
frase3 = "Brasil, meu Brasil brasileiro...";
document.write(frase3.lastIndexOf("Brasil",11));  //Imprime índice:0.
```

Podemos salvar os índices em variáveis para utilizarmos como parâmetros na localização das próximas ocorrências da substring, adicionando 1(um) para avançar na varredura da string ou subtraindo 1(um) para retroceder.

11. Teste e Comparação de Valores

Agora você poderá utilizar melhor as variáveis comparando, testando e avaliando seus valores com as ferramentas fornecidas pelo *JavaScript*.

11.1. Instrução if/else

O `if` é a instrução condicional principal do *JavaScript*. Esta instrução tem o mesmo significado da palavra em inglês: "se".

Exemplo:

```
a = 5;
if (a > 2) alert("a é maior que 2");    //Exibe a mensagem "a é maior que 2".
```

Esta instrução inclui uma condição entre os parênteses `(a > 2)`. Se a condição for verdadeira, a sentença que se segue: `alert("a é maior que 2")` será executada; caso contrário, não faz nada e o *JavaScript* pula para o próximo comando após o ponto e vírgula.
Múltiplas instruções também podem ser escritas se incluídas entre chaves ({ }).

```
if (a > 2) {
    alert("a é maior que 2");
    b=a;
}
```

O `else` (senão) trata o caso contrário, ou seja, se condição for falsa:

```
if (a > 2) {
   alert("a é maior que 2");
   a = 0;
}
else
   alert("a é menor ou igual a 2");   //Será executada se "a" não for maior que 2.
```

No caso de várias ações no `else`, inclua as sentenças entre chaves:

```
if (a > 2) {
   alert("a é maior que 2");
   a = 0;
}
else {
     alert("a é menor ou igual a 2");
     a=5;
}
```

Outros operadores podem ser utilizados em comparações, como os lógicos **`&&`** (e) e o **`||`** (ou):

```
1.  x = 2, y = 3;
    if (x == 2 && y == 3) z = x + y;      //O valor da variável z será 5.
```

Executará a sentença se as duas condições forem verdadeiras.

```
2.  x = 2, y = 4;
    if (x == 2 || y == 4) w = x + y;  //O valor da variável w será 5.
```

Executará a sentença se pelo menos uma das condições for verdadeira.

11.2. Expressão Condicional

Esta é uma expressão utilizada para se tomar decisões rápidas, também é encontrada em outras linguagens como C.

Formato:

*Variável = **(**condição**) ?** sentença_verdade **:** sentença_falso;*

Se a condição for verdadeira, a expressão executará a sentença_verdade e, opcionalmente, salvará o resultado na variável indicada; caso contrário, executará a sentença_falso e, opcionalmente, salvará o resultado na mesma variável.

11.3. Instrução switch( )

O *JavaScript* possui a instrução `switch` que permite combinar vários testes da mesma variável ou expressão em um único bloco de instruções.

```
dia_semana = 5;
switch (dia_semana) {
       case 0 : document.write("Domingo");
                   break;
       case 1 : document.write("Segunda-feira");
                   break;
       case 2 : document.write("Terça-feira");
                   break;
       case 3 : document.write("Quarta-feira");
                   break;
       case 4 : document.write("Quinta-feira");
                   break;
       case 5 : document.write("Sexta-feira"); //Será impresso "Sexta-feira".
                   Break;
       case 6 : document.write("Sábado");
                   break;
       default : alert(" Dia da semana inválido!");
}
```

- A variável que terá o seu valor testado deverá estar entre os parentes da instrução `switch`;
- As sentenças do `switch` deverão estar contidas entre as chaves ({ }).
- Cada instrução `case` possui um valor específico que será comparado com o valor da variável. Se o valor do `case` corresponder ao da variável, as instruções após os dois pontos (:) serão executadas; caso contrário, o próximo `case` será testado.
- A instrução `break` é colocada em cada `case`. Se um dos casos for satisfeito, então o `switch` poderá ser finalizado.
- O `default` é opcional. Se nenhuma das instruções `case` for satisfeita, as instruções do `default` serão executadas.

12. Array

Uma Matriz ou *Array* permite armazenar vários dados separadamente dentro de uma única variável, formando um conjunto. Usualmente, todos esses dados possuem um esquema de conexão.

Os *Arrays* simplificam o seu código porque diminui o número de variáveis que você poderia criar, com nomes similares. (Ex.: dias da semana).

Domingo	Segunda-feira	Terça-feira	Quarta-feira	Quinta-feira	Sexta-feira	Sábado
0	1	2	3	4	5	6

Cada elemento de uma célula é chamado de elemento. Por exemplo, o *Array* dos dias da semana possui 7 (sete) elementos. Cada elemento é identificado através de um índice conforme a sua posição.

Podemos acessar os valores de cada elemento através dos índices. O valor do índice inicial é 0 (zero). *Array* é um Objeto do *JavaScript,* todo novo objeto do tipo *Array* deverá ser criado a partir deste Objeto modelo, adquirindo assim a sua estrutura.

12.1. Criando um Array Numérico

Os *Arrays* podem conter *strings*, números, objetos ou outros tipos de dados.

Exemplo:

```
notas = new Array(5);
notas[0] = 8.5;
notas[1] = 5.0;
notas[2] = 10.0;
notas[3] = 9.0;
notas[4] = 4.5;
```

- A palavra-chave `new` define a variável `notas` como um novo objeto do tipo *Array*, `notas` agora possuirá todas as propriedades e métodos do objeto *Array*;
- O número entre parênteses indica a quantidade de elementos do *Array*, no caso 5;
- O índice do primeiro elemento é 0 (zero);
- Para atribuirmos valores aos elementos, utilizarmos o nome do *Array* seguido do índice correspondente ao elemento, inserido entre colchetes ([ ]):

```
notas[0] = 8.5;
notas[1] = 5.0;
...
```

12.2. Acessando os Elementos do Array

- Para ler o conteúdo de um *Array* é só utilizar a mesma notação que se utilizou na atribuição de valores.

  A seguinte instrução exibe os valores dos primeiros três elementos do *Array* `notas`:

```
document.write("Notas: " + notas[0] + ", " + notas[1] + " e " + notas[2]);
```

12.3. Criando Arrays de Strings

Criamos *Arrays* de *Strings* da mesma forma que o *Array* numérico, apenas atribuindo valores do tipo *String*.

Exemplo:

```
dia_semana = new Array(7);
dia_semana[0] = "Domingo";
dia_semana[1] = "Segunda-feira";
dia_semana[2] = "Terça-feira";
dia_semana[3] = "Quarta-feira";
dia_semana[4] = "Quinta-feira";
dia_semana[5] = "Sexta-feira";
dia_semana[6] = "Sábado";
```

Esses elementos de *Array* podem ser utilizados em qualquer lugar que se utilizaria uma *string*. Podendo até utilizar os métodos e propriedade do objeto *String* introduzidos anteriormente.

Exemplo:

```
nomes = new Array(10);
nomes[0] = "Ana Cristina";
nomes[1] = "Pedro José";
document.write(nomes[1].substring(6,10));      //Imprime José.
```

12.4. Criando Arrays Rapidamente

Eis uma forma muito rápida de criar exatamente o mesmo *Array* do exemplo anterior:

```
dia_semana = new Array("Domingo","Segunda-feira","Terça-feira",
             "Quarta-feira","Quinta-feira","Sexta-feira","Sábado");
```

- O *Array* foi criado e invés de indicarmos a quantidade de elementos que ele deverá possuir, informamos o conteúdo de cada elemento na sua respectiva ordem de índice.
- O acesso aos elementos é igual ao do *Array* anterior.

Obs. Importante: Devemos digitar o conteúdo dos parênteses sem quebra de linha!

12.5. Verificando a Quantidade de Elementos de um Array

Assim como o objeto *String*, *Array* também possui a propriedade *length*, só que ela informa o número de elementos pertencentes ao *Array* (o que é útil no caso de você criar um *Array* sem informar o tamanho e solicitar ao usuário que entre com a quantidade de dados).

Exemplo:

```
pontos = new Array(20);
document.write(pontos.length);      //Será impresso 20.
```

12.6. Dividindo uma String e Armazenando-a em um Array – split( )

O método *split( )* divide uma *String* a partir de um caractere especificado entre os parênteses, gerando outras *Strings* que serão armazenadas em um *Array*.

Exemplo:

```
nome = "Guilherme Henrique Santos";
partes = nome.split(" ");
```

- O método *split( )* dividiu a *String* `nome` em *substrings* a partir dos espaços encontrados.
- A variável `partes` se tornou um *Array* contendo as *substrings* em seus elementos:

```
document.write(partes[0]);        // Imprime Guilherme
document.write(partes[1]);        // Imprime Henrique
document.write(partes[2]);        // Imprime Santos
```

Você pode utilizar a propriedade `length` neste caso para verificar a quantidade de elementos do Array `partes`.

Exemplo:

```
alert(partes.length);             // Exibirá 3.
```

Resumindo: O método split() transforma Strings em Arrays.

12.7. Remontando um Array em uma String – join( )

O método *join( )* remonta um *Array* gerando uma *String*, a partir da junção de seus elementos.

- As *substrings* contidas nos elementos do *Array* `partes` serão separadas (dentro da *String gerada*) pelo caractere contido entre os parênteses do *join*().
- Caso o caractere de separação não seja informado, as vírgulas serão utilizadas.

A instrução abaixo remonta o *Array* `partes` atribuindo o resultado gerado a *String* `inteiro`.

```
inteiro = partes.join(" ");
alert(inteiro);                   //Exibirá: Guilherme Henrique Santos
```

Resumindo: O método join() transforma Arrays em Strings.

12.8. Classificando Elementos de um Array

O método *sort( )* retorna uma versão classificada do *Array* (alfabética ou numérica).

Exemplo:

```
nomes = new Array("Rafael", "Joaquim", "João", "Ana");
nomes_classif = nome.sort( );
document.write(nomes_classif);     // Imprimirá: Ana, Joaquim, João, Rafael
```

Os nomes poderão ser acessados separadamente através de seus índices em `nomes_classif`:

```
alert(nomes_classif[2]);          // Exibirá João
```

13. Loops

O *JavaScript* possui recursos que fazem o computador desempenhar tarefas repetitivas para você.

13.1. Loop for

O *loop for* é o primeiro que utilizaremos para criar *loops* (laços, voltas ou repetições).

Exemplo:

```
for (i = 1; i < 5; i++){
    document.write("Esta é a linha ",i, "<br>");
}
```

Resultado: Esta é a linha 1
Esta é a linha 2
Esta é a linha 3
Esta é a linha 4

Tags de HTML podem ser inseridas na instrução document.write() (entre aspas) como também podem ser concatenadas com dados em JavaScript.
Vírgulas (,) podem ser utilizadas para concatenar no lugar do caractere mais (+).

Analisando o Exemplo anterior:

- A primeira parte da instrução (`i = 1`) é chamada de *expressão inicial*, porque estabelece o estado inicial do *loop* especificando uma variável e atribuindo um valor inicial a ela.

  A expressão é executada apenas uma única vez, no início do *loop*.
- O segundo parâmetro (`i < 5`) é uma condição que deve permanecer verdadeira para manter o *loop* executando (lê-se: enquanto `i` for menor que `1`).

  Esta instrução é chamada de *condição do loop*. Se a condição for falsa, o loop é encerrado.
- O terceiro parâmetro (`i++`) é uma instrução que é executada no final de cada iteração (volta) do *loop*, após a execução do bloco de instruções.

  Esta instrução é chamada de *expressão de incremento,* porque normalmente é utilizada para incrementar o contador.
- Após a especificação dos três parâmetros, um conjunto de sentenças envoltas por chaves é escrito para ser executado a cada iteração do *loop*, no caso da condição for verdadeira.

Comentando o exemplo: inicialmente `i` recebe o valor `1`; a condição é testada: se verdade que `i` é menor que `5`, a instrução `document.write("Esta é a linha ",i,"<br>") é executada`; é adicionado `1` em `i`; (fim da primeira iteração).
Verifica-se novamente a condição, enquanto `i` for menor que 5 o *loop* continua, caso contrário, o *loop* é encerrado.

13.2. Loop while

Diferente do *loop* for, o *loop while* não necessita obrigatoriamente de um contador para controlá-lo, em vez disso, ele executa enquanto (*while*) uma condição for verdadeira. Se a condição iniciar como falsa, as instruções não serão executadas.

Exemplo:

```
valor = new Array(3,2,4,1,5);
i = 0;
total = 0;
while (total < 10){
      total += valor[i];
      i++;
}
document.write("Total = " + total);      //Resultado: Total = 10
```

- A condição da instrução *while* deve estar entre parênteses, ela é testada no início do *loop*;
- Bloco de instruções fica entre chaves.

Comentando o exemplo: Inicialmente a condição é testada; Enquanto o conteúdo da variável `total` for menor que `10`, `i` será incrementado em `1` e servirá de índice para o *Array* `valor`, que terá seu conteúdo acumulado em `total`. Caso contrário, se `total` for maior ou igual a `10`, o *loop* será encerrado.

Podemos utilizar contadores para controlar o loop, ele deve ser declarado antes do loop e ser incrementado no bloco de instruções.

13.3. Loop do...while

O terceiro *loop* é o *do...while* (faça...enquanto). A diferença em relação ao *loop while* é que no *do...while*, a condição é testada no final do *loop*, portanto, as instruções inseridas no *do* são executadas pelo menos uma vez.

Exemplo:

```
valor = new Array(3,2,4,1,5);
cont = 0;
total = 20;
do {
   total += valor[cont];
   cont++;
}
while (total < 10)

document.write("Total = " + total);      //Resultado: Total = 23
```

- O bloco de instruções deve estar inserido entre chaves após o comando *do* (faça);
- A condição, inserida entre parênteses, é testada no final do *loop*.

13.4. Loop for...in

Este *loop* é especificamente projetado para desempenhar uma operação em cada propriedade de um objeto. O *for...in* também é muito útil para trabalhar com *Arrays*.

Exemplo:

```
nomes = new Array("Jonas", "Ana", "Ruth", "Tiago", "Marcos");
document.write("<ol>");
for (i in nomes) {
       document.write("<li>"+nomes[i]+ "<br>");
}
document.write("</ol>");
```

Resultado:
1. *Jonas*
2. *Ana*
3. *Ruth*
4. *Tiago*
5. *Marcos*

- `i` é uma variável de índice iniciada em zero e incrementada de 1 em 1 automaticamente;
- `nomes` é o *Array* que terá seus elementos acessados através do índice `i`;
- O *loop* é encerrado quando o último elemento do *Array* for acessado.

Comentando o exemplo: No início do exemplo foi criado um *Array* de nomes e iniciada uma lista ordenada em HTML; no *loop*, a instrução for (`i in nomes`) utilizou o índice `i` para acessar os elementos do *Array* `nomes`; a instrução `document.write("<li>"+nomes[i]+"<br>")`, entre as chaves, é executada a cada iteração do *loop*, imprimindo um elemento do *Array* como um item da lista ordenada; Após o termino do *loop*, a lista HTML foi finalizada.

13.5. Criando um Loop Infinito

Os *loops for* e *while* permitem bastante controle sobre o *loop*. Em alguns casos, isso pode causar problemas se você não tiver cuidado.

Exemplo:

```
j=0, n=0;
while (j < 10) {
      n++;
      document.write(" n = " + n);
}
```

Há um equívoco neste exemplo. A condição do *loop while* refere-se à variável `j`, mas essa variável não se altera durante o *loop*, isso cria um *loop infinito*! O *loop* continua sendo executado até que seja interrompido pelo usuário, que gere algum tipo de erro ou até mesmo que provoque uma pane no sistema.

Loops infinitos não são identificados pelo JavaScript, certifique-se de que há uma saída para o seu loop.

13.6. Escapando de um Loop Infinito

Há uma maneira de escapar de um *loop* infinito, saindo imediatamente dele e continuando a execução do *script* a partir da primeira instrução após o *loop*. Você pode usar a instrução *break* associada a uma condição incluída nas sentenças do *loop*.

Exemplo:

```
    valor = new Array(2,6,5,3,10,22,35);
     n=-1;
     while (true) {
        n++;
        if (valor[n]==10) break;
        document.write(valor[n]+"<br>");
}
```

Comentando o exemplo: A instrução *while* define o *loop* como infinito, por causa do `true (verdade)` especificado na condição do *while*, portanto, enquanto for verdade, o *loop* continua sendo executado.
A instrução condicional `if` verifica se algum dos valores dos elementos do *Array* é igual a `10`, se for, o *loop* é encerrado.

13.7. Desprezando uma Iteração de um Loop

Uma outra instrução disponível para controlar a execução de um *loop* é o `continue`. Ele despreza as sentenças que vierem após ele, continuando a execução do *loop* a partir da próxima iteração.

Exemplo:

```
j=0;
pontos = new Array(5,0,4,2,0,7,0,8,1,6);
for (i = 0; i < 10; i++) {
      j=i;
      if (pontos[i] == 0) continue;
      document.write("Atleta número ",++j," - pontos: ",pontos[i],"<br>");
}
```

Comentando o exemplo: Neste exemplo criamos um *Array* com os valores dos elementos já atribuídos. No *loop*, a variável `i` é iniciada com o valor `0` (zero); a condição estabelece que o *loop* deve ser executado enquanto `i` for menor que `10`. A cada iteração, as instruções entre as chaves são executadas; para os elementos do *Array* cujos pontos forem iguais a `0` (zero), a instrução `continue` faz com que a instrução `document`.write`(...)` seja desprezada (não executada) e o *loop* continue a partir da próxima volta; para os elementos cujos valores forem diferentes de zero, uma linha com o número do atleta e os pontos obtidos por ele é impressa através do `document.write()`; ao final de cada iteração `i` é acrescido em 1.

14. Objetos Embutidos

Objetos embutidos são aqueles que existem automaticamente em qualquer programa *JavaScript*. Possuem propriedades e métodos (funções intrínsecas, ou embutidas).
A sintaxe geral para utilização dessas funções é:

*resultado = **função(**informação a ser processada**);***

Exemplificaremos utilizando a função *eval()* do objeto `Math`, que calcula o conteúdo de uma *String*.

Exemplo:

```
1. resultado = eval("(10 * 20) + 2 - 8");   // O valor de resultado será 194.

2. calculo1 = ("3*6+2");
   resultado = eval (calculo1);             //O valor de resultado será 20.
```

A string `calculo1` também pode ser o conteúdo de uma caixa de texto.

parseInt() e *parseFloat()*, assunto que já abordamos anteriormente, também fazem parte das funções intrínsecas (convertem *Srings* em números e em números com ponto flutuante, respectivamente).

14.1. Objeto Math

O objeto *Math* traz para o *JavaScript* toda a funcionalidade e constantes matemáticas básicas que você pode utilizar através de suas propriedades e métodos embutidos.

14.1.1. Propriedades e Funções Matemáticas

São aquelas tipicamente matemáticas:

Propriedades	Descrição
`Math.`**`E`**	*Retorna a base dos logaritmos naturais (aproximadamente 2.718).*
`Math.`**`LN2`**	*Retorna o valor do logaritmo de 2 (aproximadamente 0.693).*
`Math.`**`LOG2E`**	*Retorna a base do logaritmo de 2 (aproximadamente 1.442).*
`Math.`**`LN10`**	*Retorna o valor do logaritmo de 10 (aproximadamente 2.302).*
`Math.`**`LOG10E`**	*Retorna a base do logaritmo de 10 (aproximadamente 0.434).*
`Math.`**`SQRT2`**	*Retorna a raiz quadrada de 2 (aproximadamente 1.414).*
`Math.`**`SQRT_2`**	*Retorna a raiz quadrada de 1/2 (aproximadamente 0.707).*
`Math.`**`PI`**	*retorna o valor de PI (aproximadamente 3.14159).*

Exemplo:

```
val_pi = Math.PI;
alert(val_pi);                    // Exibe 3.141592653589793
```

Métodos	Descrição
Math.**abs(**número**)**	*Retorna o valor absoluto do número (ponto flutuante).*
Math.**pow(**base, expoente**)**	*Retorna o cálculo do exponencial.*
Math.**max(**número1, número2**)**	*Retorna o maior número entre os fornecidos.*
Math.**min(**número1, número2**)**	*Retorna o menor número entre dos dois fornecidos.*
Math.**sqrt(**número**)**	*Retorna a raiz quadrada do número.*
Math.**sin(**número**)**	*Retorna o seno de um número (anglo em radianos).*
Math.**asin(**número**)**	*Retorna o arco seno de um número (em radianos).*
Math.**cos(**número**)**	*Retorna o cosseno de um número (anglo em radianos).*
Math.**acos(**número**)**	*Retorna o arco cosseno de um número (em radianos).*
Math.**tan(**número**)**	*Retorna a tangente de um número (anglo em radianos).*
Math.**atan(**número**)**	*Retorna o arco tangente de um número (em radianos).*
Math.**log(**número**)**	*Retorna o logarítmo de um número.*
Obs.: Em todos os métodos, a expressão "(número)" refere-se a um argumento que será processado pela função e que poderá ser um número, uma variável ou o conteúdo de um objeto (propriedade *value*).	

Exemplo:

```
base = 3;
expoente = 2;
resultado = Math.pow(base,expoente);
document.write(resultado);          //Imprime 9.
```

14.1.2. Arredondando e Truncando Valores

Métodos	Descrição
Math.**ceil(**número**)**	*Retorna o próximo valor inteiro maior que o número.*
Math.**floor(**número**)**	*Retorna o próximo valor inteiro menor que o número.*
Math.**round(**número**)**	*Retorna o valor inteiro do número, arredondado.*

Exemplo:

```
x = 3.46;
document.write(Math.ceil(x)+"<br>");        // Imprime 4.
document.write(Math.floor(x)+"<br>");       // Imprime 3.
document.write(Math.round(x)+"<br>");       // Imprime 3.
```

14.1.3. Criando Números Pseudo-Aleatórios

Métodos	Descrição
Math.**random()**	*Retorna um número decimal entre 0 e 1 aleatório (não exige nenhum parâmetro).*

Exemplos:

```
1. alert(Math.random());          // Vai exibir um número aleatório entre 0 e 1,
                                  // Exemplo: 0.7149896088624416
2. num=5;
   valor= Math.floor(Math.random() * num) + 1;
   document.write(valor);         // Imprime um número aleatório entre 1 e 5.
```

Comentando o exemplo 2: Esta função multiplica um número aleatório (`Math.random()`) pelo valor que você passa para ela (`num = 5`) e depois o converte em um inteiro entre 1 (`+1`) e o número indicado, utilizando o método `Math.floor()`.

14.2. Trabalhando com Números

14.2.1. Criando um objeto Number

Construtor	Descrição
new Number(n**)**	*Construtor de objetos Number*
Number(n**)**	*Converte um valor em número*

Exemplos:

```
1. num = new Number(23);          //A variável num recebe 23.
   document.write("Tipo de objeto: "+ typeof(num)+" = "+num);

2. num = new Number("23");        //Apesar das aspas, num é numérico.
   document.write("Tipo de objeto: "+ typeof(num)+" = "+num);
```

14.2.2. Verificando se o conteúdo de uma variável é numérico

Métodos	Descrição
isNaN(variável**)**	*Verifica se **não** é um número (Not a Number).*

```
num = "a23.45";
(isNaN(num))?alert(num+"-True, não é número"):alert(num +"-False, é número");
```

//Exibe: a23.45 não é numérico

14.2.3. Fixando o Número de Algarismos após a Casa Decimal

Métodos	Descrição
`Number.toFixed(algarismos)`	*Fixa (e arredonda) o número de algarismos a serem exibidos após a casa decimal.*

Exemplo:

```
x = 3.4656;
document.write(x.toFixed(2));          // Arredonda e imprime com 2 casas: 3.47.
```

14.2.4. Convertendo um número em String

Métodos	Descrição
`Number.toString(base)`	*Converte um número em String, utilizando uma base (opcional) entre 2 e 36.*

Exemplos:

```
a=240;

1. document.write(typeof(a.toString()));     //Imprime: string

2. document.write(a.toString(16));           //Imprime: f0

3. document.write(a.toString(2));            //Imprime: 11110000
```

14.3. Objeto Date

Date é um objeto embutido do *JavaScript* que trabalha convenientemente com datas e horas. O objeto *Date* não possui propriedades, só métodos.

As datas são armazenadas em milésimos de segundos desde a meia-noite de 1º de Janeiro de 1970.

14.2.1. Criando um Objeto Date

Um objeto *Date* é criado a partir da palavra-chave *new*. Opcionalmente podemos especificar a data que queremos armazenar no objeto quando o criamos. Podemos utilizar qualquer um dos seguintes formatos:

```
Hoje  = new Date( );                                  //Armazena a data corrente
natal = new Date("December 25, 2020 00:00:00"); //mm, dd, aa, hh, mm e ss
natal = new Date(12, 25, 2020);                       //mm, dd e aa
natal = new Date(12, 25, 2020, 0, 0, 0);              //mm, dd, aa, hh, mm e ss
```

Se parâmetros não forem informados entre os parênteses, como no primeiro exemplo, a data atual obtida a partir da data do sistema operacional do computador do usuário é armazenada no objeto.

14.3.2. Alterando Valores de Datas

Uma variedade de métodos .*set* permite configurar componentes de um objeto *Date*:

Método	Descrição
.setDate( )	*Estabelece dia do Mês.*
.setMonth( )	*Estabelece o Mês. (Valores de 0 à 11; Janeiro = 0).*
.SetDay( )	*Estabelece o dia da Semana (Valores de 0 à 6; Domingo = 0).*
.setYear( )	*Estabelece o Ano. (2 dígitos)*
.setFullYear( )	*Estabelece o Ano. (4 dígitos)*
.setTime( )	*Estabelece a Hora (e a data) em milésimos de segundos.*
.setHours( )	*Estabelece a Hora.*
.setMinutes( )	*Estabelece os Minutos.*
.SetSeconds( )	*Estabelece os Segundos.*

Exemplo:

```
hoje = new Date( );              //Acessa a data do sistema operacional
hoje.setFullYear(2020);          //Altera o ano para 2020
alert(hoje);
```

A caixa de alerta exibirá o resultado no formato *String, (e em inglês)*, exibindo o dia da semana, mes, dia, horas, fuso horário e ano. Se o `alert()` fosse executado no momento da configuração da data, o resultado seria em milésimos de segundos.

Exemplo:

```
alert(hoje.setYear(2020));       //Resultado: 1587767807656
```

14.3.3. Obtendo Valores de Datas

O método .*get* obtém valores de um objeto *Date*.

Método	Descrição
.getDate( )	*Obtém o dia do Mês.*
.getMonth( )	*Obtém o Mês. (Valores de 0 à 11; Janeiro = 0).*
.getDay( )	*Obtém o dia da Semana (Valores de 0 à 6; Domingo = 0).*
.getYear( )	*Obtém o Ano. (2 dígitos)*
.getFullYear( )	*Obtém o Ano. (4 dígitos)*
.getTime( )	*Obtém a Hora (e a data) em milissegundos.*
.getHours( )	*Obtém a Hora.*
.getMinutes( )	*Obtém os Minutos.*
.getSeconds( )	*Obtém os Segundos.*

Exemplo:

```
hoje = new Date("Apr 24 2020");
dia = hoje.getDate( );
alert(dia);                      //Exibirá: 24.
```

14.3.4. Trabalhando com Fuso Horário

Algumas funções estão disponíveis para ajudar os objetos *Date* a trabalhar com valores de hora local e fuso horário:

Método	Descrição
.getTimezoneOffset( )	*Fornece a diferença entre o fuso horário local e o GMT (Greenwich Mean Time ou UTC), em minutos.*
.toGMTString( )	*Converte o valor de hora do objeto Date em texto, utilizando o GMT.*
.toLocaleString( )	*Converte o valor de hora do objeto Date em texto, utilizando a hora local.*

Exemplo:

```
hoje = new Date("Apr 24 2020 00:00:00");
alert(hoje.getTimezoneOffset( )); //Exibirá: 180. (*)
alert(hoje.toGMTString( )); //Exibirá: Fri, 24 Apr 2020 03:00:00 UTC
alert(hoje.toLocaleString( )); //Exibirá: Sext-feira, 24 de abril de 2020 00:00:00
```

* - No horário de verão, este valor sofre alteração (120 em São Paulo)

14.3.5. Convertendo Formatos de Data

Método	Descrição
Date.parse(string**)**	*Converte uma string de data em um objeto Date. (número de milésimos de segundos desde 01/01/1970).*
Date.UTC(valor**)**	*Converte um valor de objeto Date (número de milésimos de segundos) em uma hora UTC (GMT).*

Exemplo:

```
document.write(Date.parse("Apr 20, 1996")); //Imprime: 829969200000
```

15. Instrução with

A instrução *with*, permite criar uma instrução para um objeto, reduzindo assim a digitação.

Exemplo sem o with:

```
n1 = prompt(" Entre com um número","");
n2 = prompt(" Entre com um número","");
n3 = prompt(" Entre com um número","");
alert("O maior número digitado foi: " + Math.max(n1, n2, n3));
alert("O menor número digitado foi: "+ Math.min(n1, n2, n3));
```

Exemplo o *with*:

```
n1 = prompt("Entre com 0 número 1","");
n2 = prompt("Entre com 0 número 2","");
n3 = prompt("Entre com 0 número 3","");
with (Math){
   alert("O maior número digitado foi o: " + max(n1, n2, n3));
   alert("O menor número digitado foi o: " + min(n1, n2, n3));
}
```

Com a instrução `with` não precisamos digitar o nome do objeto (`Math`) para utilizar seus métodos.

16. Tratadores de Eventos

Neste capítulo você aprenderá a utilizar uma ampla variedade de *handlers* (tratadores) de eventos suportados pelo *JavaScript*. Em vez de executar na ordem da codificação, os *scripts* que utilizam *handlers* de evento podem ser executados a partir da ação do usuário. Esses eventos são aqueles que o usuário pode gerar através do mouse, teclado e outros eventos especializados. Você adiciona um atributo de *handler* de evento em uma *tag* de HTML (botão, link, janela, imagem etc.) e insere o *script* em *Javascript* entre aspas, os *scripts* são executados da ação do evento.

Por convenção, destacamos em maiúsculo as letras iniciais referentes aos eventos, mas não fazem parte da sintaxe, portanto, os tratadores de eventos podem ser todos digitados em minúsculo.

16.1. Respondendo a Eventos

A seguir, você conhecerá uma lista de tratadores de eventos que poderão ser utilizados com **moderação**, ou seja, somente **se** necessário, visto que muitos tratadores de eventos atrapalham a navegação, irritando os usuários. Cuidado com o tipo de aspas utilizado (` /")!!!

16.1.1. onClick

O evento ocorre quando o usuário clica no botão esquerdo do *mouse* sobre algum elemento do documento:

```
<body>
   <input type="button" value="Não clique!" onclick="alert('Você clicou!!')">
</body>
```

16.1.2. onDblClick

O evento ocorre quando o usuário dá um clique duplo sobre algum elemento do documento.

```
<body>
   <b ondblclick="alert('Você clicou duas vezes!');">
       Dê um duplo clique aqui!
   </b>
</body>
```

O tratamento que será dado como resposta ao evento deve ser digitado entre aspas. Neste exemplo utilizamos uma caixa de alerta, o seu conteúdo deve vir entre apóstrofes e não aspas para não finalizar incorretamente o tratador de eventos.

16.1.3. onMouseDown

O evento ocorre quando o usuário pressiona o botão esquerdo do *mouse* sobre um objeto apropriado. O *handler* básico de evento é o *onClik*.

```
<body>
  <input type="button" value="Botão"
   onMouseDown="alert('Botão pressionado!');">
</body>
```

16.1.4. onMouseUp

O evento ocorre quando o usuário libera o botão do *mouse* que estava pressionado sobre um objeto.

```
<body>
   <input type="button" value="Botão"
    onMouseUp="alert('Botão liberado!');">
</body>
```

16.1.5. onMouseOver

O evento ocorre quando o usuário passa com o ponteiro do *mouse* sobre um *link*, imagem ou outro objeto que se encontra dentro no documento.

```
<body>
   <a href="#" onMouseOver="alert('Ponteiro sobre o link');"> link1 </a>
</body>
```

16.1.6. onMouseOut

O evento é o oposto do anterior, ocorre quando o ponteiro do mouse é movido para fora da borda do objeto. Geralmente utilizamos o *onMouseOut* associado ao *onMouseOver* (podemos criar com eles efeitos de animação utilizando duas imagens que ocupando o mesmo espaço, se alternam ao passarmos com o ponteiro do mouse sobre elas.

```
<body>
  <a href="#" onMouseOut="alert('Ponteiro fora do link');"> link2 </a>
</body>
```

16.1.7. onMouseMove

O evento ocorre quando o usuário move o mouse sobre o documento. Geralmente ele vem desabilitado por controlar o ponteiro do mouse do usuário o tempo todo. O resultado deste exemplo será exibido na linha de *status* do navegador (rodapé).

```
<head>
  <script>
   function moveu() {
    window.status = "Coordenadas do mouse: X = "+ event.x + "Y = " + event.y;
   }
  </script>
</head>
<body onMouseMove="moveu();">
</body>
```

16.1.8. onLoad

O evento ocorre quando todas as imagens da página corrente terminam de ser carregadas.

```
<body onLoad="alert('Que bom que você veio!');">
```

16.1.9. onUnLoad

O evento ocorre quando o usuário sai da página atual.

```
<body onUnLoad="alert('Não se vá!');">
```

16.1.10. onHelp

O evento ocorre quando o usuário pressiona a tecla F1 para ajuda. Você pode cancelar o evento padrão usando a propriedade *event.returnValue* e definindo-a como *false*.

```
<head>
  <script>
     function ajuda( ){
        alert("Eu também não sei!");
        event.returnValue = false;
     }
  </script>
</head>
<body onHelp = "ajuda( );">
```

16.1.11. onStop

O evento ocorre quando o usuário clica no botão *Stop* do Navegador.

```
<head>
   <script>
     function pare( ){
       alert("Porque parou? Parou porque?");
       event.returnValue = false;
       }
     </script>
   </head>
   <body onStop = "pare( );">
```

16.1.12. onContextMenu

O evento ocorre quando o usuário dá um clique no botão direito do mouse na área do documento para abrir o menu de contexto. Você pode cancelar o evento padrão usando a propriedade *event.returnValue* e definindo-a como *false*.

```
<head>
   <script>
      function menu( ){
        alert(“Pirataria não!”);
        event.returnValue = false;
      }
   </script>
</head>
<body onContextMenu = “menu( );”>
```

16.1.13. onAbort

O evento ocorre se o usuário abortar a página antes da imagem ser carregada.

```
<body>
   <img src="imagem.gif" onAbort="alert('Disse a imagem: –Não me aborte!');">
</body>
```

16.1.14. onError

O evento ocorre quando o arquivo de imagem não é encontrado ou está corrompido.

```
<body>
   <img src="imagem.gif" onError="alert('Imagine uma bela imagem aqui...');">
</body>
```

16.1.15. onKeyDown

O evento ocorre sempre que o usuário pressionar uma tecla.

```
<head>
   <script>
      function clique(){
        tecla = String.fromCharCode(event.keyCode);
        window.status="Você pressionou a tecla " + tecla;
      }
   </script>
</head>
<body onKeyDown="clique();">
```

Neste exemplo, quando a tecla é pressionada o caractere capturado é alterado em relação ao formato padrão Unicode (números usado para representar caracteres) para caracteres reais, usando String.fromCharCode(event.keyCode). Então os caracteres são mostrados na barra de status do navegador.

16.1.16. onKeyUp

Ao contrário do anterior, este evento ocorre quando o usuário solta a tecla.
(Utilizando o *script* anterior).

```
<body onKeyUp="clique();">
```

16.1.17. onKeyPress

O evento ocorre quando o usuário pressiona uma tecla alfanumérica.
(Utilizando o *script* anterior).

```
<body onKeyPress="clique();">
```

16.1.18. onResize

O evento ocorre quando o usuário redimensiona a página ou frames (quadros).

```
<body onResize="alert('Melhor não mexer!');">
     </body>
```

16.2. Objeto event

O *event* é um objeto especial que é enviado para um *handler* de evento à cada ocorrência. O *handler* de evento recebe esse objeto como um parâmetro. As propriedades do objeto *event* oferecem mais informações sobre o evento que ocorreu. As propriedade disponíveis são:

- `type` Tipo de evento que ocorreu, como *mouseover*.
- `target` Objeto de destino para o evento (como o documento ou um *link*).
- `which` Valor numérico que especifica o botão do mouse que foi clicado para eventos de *mouse* ou a tecla que foi pressionada para eventos de teclado.
- `modifiers` Lista de chaves de modificador que foram pressionadas durante um evento de teclado ou de *mouse* (como Alt, Ctrl e Shift).
- `data` Lista de dados arrastados e soltos para eventos de arrastar e soltar.
- `x` e `y` Posição x e y do mouse quando ocorreu o evento, medida a partir do canto superior esquerdo da página.
- `screenX` Posição X do *mouse* , medida do canto superior esquerdo da tela.
  `screenY` Posição Y do *mouse* , medida do canto superior esquerdo da tela.
- `keyCode` Código ASCII da Tecla pressionada.

Exemplo:

```
<script>
   function coord( ){
       window.status="Coord. X = " + event.x + "Coord. Y = " + event.y;
       // Exibe as coordenadas x, y do mouse na linha de status do navegador.
</script>

<body onMouseMove="coord();">          <!-- Chamada da função -->
```

17. Objetos Personalizados

A linguagem *Javascript* também é Orientada a Objeto.

17.1. Conceitos

Objetos: São entidades concretas ou abstratas, simples ou complexas, que possuem atributos que os distinguem uns dos outros e relacionamentos que associam uns aos outros.
Propriedades: São os atributos do Objeto.
Métodos: Comportamentos que os objetos são capazes de executar.
Instância: Um novo objeto criado a partir de uma estrutura de objeto já definida (classe).
Variável de objeto: Campo de recepção de dados.

Em outras palavras...

Objeto: Tudo o que é perceptível por qualquer dos sentidos (carro, pessoa, conta...).
Propriedade: Tudo o que o objeto possui (característica).
Método: Tudo o que o objeto pode fazer (ação).
Instância: "Clone" de um objeto.
Variável de objeto: Lugar onde os dados inseridos são recebidos.

Exemplo:

Objeto: Carro
Propriedades: Cor, marca, modelo, ano...
Métodos: Buzinar, acelerar, acender faróis...

Se você quiser criar um Cadastro de Clientes para armazenar as informações sobre eles (nome, telefone, e-mail...) e depois executar algumas ações (calcular, imprimir...) você pode utilizar variáveis ou *Arrays* para isto; mas, existe um modo muito mais prático e eficiente de se fazer a mesma coisa utilizando Objetos:

- Você cria uma estrutura modelo para este cadastro (objeto), contendo campos que armazenarão as informações sobre os clientes (propriedades) e desenvolve funções que executarão as ações pertinentes a eles (métodos).
- Após a criação desta estrutura, faz uma cópia deste modelo (instância) para cada cliente do cadastro, que posteriormente terão suas informações (dados) inseridas neste novo modelo (através das variáveis de objeto) e as ações executadas (métodos).
- Cada Cliente terá desta forma todos os dados e ações inseridas dentro de uma unidade própria (objeto).

17.2. Criando Objetos Personalizados

Os Objetos são criados a partir de funções Construtoras. Exemplo:

Objeto: Cadastro
Propriedades: Nome, Endereço e Telefone.
Método: Impressão dos dados dos clientes.

```
<head>...
 <script language="javascritp">
   function Cadastro(v_nome, v_ender,v_tel){ //Criando Objeto Cadastro e suas variáveis.
      this.nome= v_nome;
      this.ender=v_ender;                    //Criando as propriedades nome, ender e tel.
      this.tel=v_tel;
      this.impr=f_impr;                      // Criando o método impr.
   }
   function f_impr( ){
     linha1 = "<b>Nome: " + this.nome + "<br>";  //Definindo a função que será utilizada
     linha2 = "Endereço: " + this.ender + "<br>";   //como um método do objeto.
     linha3 = "Telefone: "+ this.tel  + "</b><hr>";
     document.write(linha1,linha2,linha3);
   }
 </script>
</head>
```

Comentando o exemplo:

- O nome da função construtora é o nome do Objeto: `Cadastro`.
- A palavra chave `this` antes das propriedades `nome`, `ender` e `tel` e do método `impr` do Objeto indica o objeto corrente, ou seja, `this` é um nome substituto do objeto que será instanciado posteriormente.
- As propriedades `nome`, `ender`, `tel` armazenarão o conteúdo que receberão das variáveis de objeto `v_nome`, `v_ender` e `v_tel`.
- `impr`: é um método. Todo método recebe uma função, neste caso, impr recebe a funç`f_impr`.

17.3. Criando Instâncias

A partir do modelo de Objeto `Cadastro` vamos criar as instâncias de objeto, ou seja, cadastrar os clientes e atribuir valores às suas propriedades.

```
<body>...
  <script>
     Maria = new Cadastro( );              //Criando a instância Maria.
     Maria.nome="Maria José dos Santos";   //Atribuindo valores às propriedades.
     Maria.ender="Rua Bela Vista, 100";
     Maria.tel="2222-0000";
     Maria.impr();                         //Utilizando o método.
  </script>
</body>
```

17.4. Criando Instâncias Atribuindo Valores

Neste exemplo, o conteúdo das propriedades é informado no momento da instanciação do objeto, sendo os dados inseridos na mesma ordem em que foram criados.

```
<body>...
  <script>
    Jose = new Cadastro("José Pedro dos Santos","Rua Alta, 33","3333-0000");
    Jose.impr();
    Marcos = new Cadastro("Marcos de Jesus","Rua Paraíso, 7","1111-0000");
    Marcos.impr();
  </script>
</body>
```

17.5. Relação de Métodos e Propriedades de Formatação

Métodos

Formatação de fonte:

`String.big()`	Aumenta o tamanho da fonte.
`String.small()`	Diminui o tamanho da fonte.
`String.blink()`	Altera a fonte para piscante.
`String.bold()`	Altera a fonte para negrito.
`String.fixed()`	Altera o tipo da fonte para mono-tipo.
`String.italics()`	Altera a fonte para itálico.
`String.fontcolor(cor)`	Altera a cor da fonte.
`String.fontsize(n)`	Altera o tamanho da fonte.
`String.strike()`	Altera a fonte para tachado.
`String.sub()`	Altera a fonte para sobrescrito.
`String.sup()`	Altera a fonte para subscrito.
`String.anchor(nome)`	Cria uma âncora no local.

Exemplo:

```
<script>
   nome = "José Maria";
   document.write(nome.bold());            // Imprime "José Maria" em negrito.
   document.write("Ana Paula".italics());  // Imprime "Ana Paula" em itálico.
</script>
```

Propriedades

Formatação de página:

`document.bgColor`	Altera a cor de segundo plano da página.
`document.fgColor`	Altera a cor padrão da fonte da página.
`document.linkColor`	Altera a cor padrão dos links da página.
`document.alinkColor`	Altera a cor padrão dos links ativos da página.
`document.vlinkColor`	Altera a cor padrão dos links visitados da página.

Exemplo:

```
<script>
   document.bgColor="green";          // Altera a cor de segundo plano da página para verde.
</script>
```

17.6. Protótipos de Objetos

Se você acha que um objeto não atende plenamente às suas necessidades, você pode estendê-lo adicionando uma nova propriedade ou método.
`Prototype` (protótipo) é outro nome para a definição de um objeto ou uma função construtora.

Vamos exemplificar adicionando um método que imprima um texto (*String*) no formato de título, utilizando a *tag* de títulos de HTML, o tamanho da fonte será passado ao método como parâmetro:

```
<head>...
  <script>
    function titulo(n){                    //Função recebendo parâmetros
       tag_inicio = "<h" + n + ">";        //Montagem da linha de impressão
       texto = this.toString();            //Conversão da frase que será inserida em String
       tag_final = "</h" + n + ">";        //Montagem da linha de impressão
       linha = tag_inicio + texto + tag_final;  //Montagem da linha de impressão
       return linha;                       //Retornando a linha de impressão
    }
    String.prototype.tit=titulo;

//prototype adiciona a função titulo() como um novo método do objeto String denomidado tit().
  </script>
</head>
<body>
  <script>
    frase=new String("É de batalhas que se vive a vida, tente outra vez...");
    document.write(frase.tit(2));          //Uso do novo método tit() pela String frase.
  </script>
</body>
```

18. Hierarquia do Objeto Browser

Uma vantagem que o *JavaScript* tem em relação a linguagens como Java, é que os *scripts* podem manipular o navegador da *Web*, como carregar uma nova página no navegador, trabalhar com partes da janela e do documento do navegador e até abrir novas janelas.

A fim de trabalhar com o navegador e os documentos, o *JavaScript* utiliza um variedade de objetos *browser*.

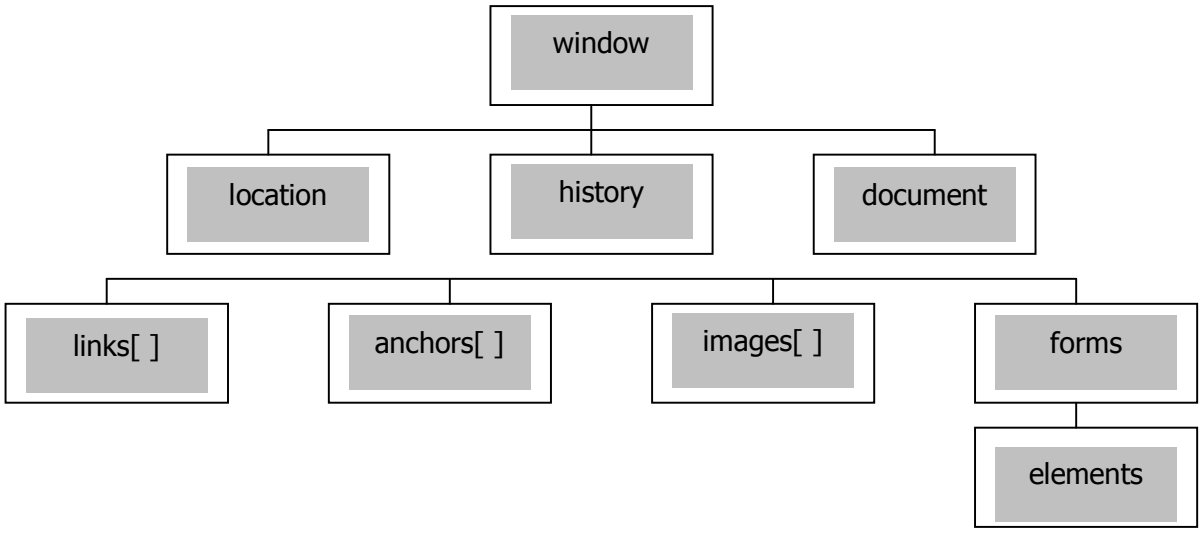
Os objetos *browser* estão organizados em uma hierarquia de objeto *pai* seguido pelo nome ou nomes do objeto *filho*, separado por pontos.

Exemplo:

```
window.document.image1
```

Neste exemplo, o objeto *image1* é filho do *document*, que por sua vez é filho do objeto *window*. O objeto *window* está na parte superior da hierarquia de objeto *browser*.

Diagrama contendo objetos básicos de um navegador.



18.1. Objeto window

Na parte superior da hierarquia de objeto *browser* está o objeto *window*, que representa uma janela de navegador. O objeto *window* sempre se refere à janela atual (aquela que contém o *script*). A palavra *self* também é sinônimo para a janela atual, isto é importante porque podemos trabalhar com várias janelas abertas ao mesmo tempo. Nós já utilizamos alguns métodos e propriedades deste objeto:

- **Propriedade:**

*18.1.1. window.**status***

Altera o conteúdo da linha de *status* do navegador, situada no "rodapé" da janela:

```
<body>
    <a href="pag1.html"
       onMouseOver="window.status='Descrição da Página 1';return true"
       onMouseOut="window.status='';return true"> Página 1
    </a>
</body>
```

A frase "Descrição da página 1" será mostrada quando o ponteiro do mouse pousar sobre o link e desaparecerá quando o ponteiro for retirado.

- **Métodos**:

*18.1.2. window.**alert()***

Exibe uma caixa de mensagem ao usuário:

```
<script>
   window.alert("Bom dia! ou Boa tarde! ou Boa noite?!!");
</script>
```

Clique no botão <OK> da caixa para fechá-la.

*18.1.3. window.**prompt()***

Exibe uma caixa de diálogo para que o usuário entre com informações:

```
<script>
   v_nome = window.prompt("Qual é o seu nome?","Digite o seu nome aqui.");
</script>
```

A frase "Qual é o seu nome?" será mostrada no cabeçalho da caixa de diálogo e "Digite o seu nome aqui.", no campo onde o usuário deverá digitar o seu nome. A resposta será atribuída a variável v_nome.

*18.1.4. window.**confirm()***

Exibe uma caixa de diálogo pedindo uma confirmação do usuário:

```
<script>
   v_resp=window.confirm("Quer ganhar um milhão agora?");
</script>
```

Uma caixa de diálogo se abrirá contendo dois botões: <OK> (true) e <Cancel> (false). O usuário confirmará ou cancelará a solicitação. A resposta será atribuída a variável v_resp.

*18.1.5. window.**setTimeout()***

Permite a execução de comandos com retardo de tempo (temporizador):

```
<script>
   window.setTimeout("alert('O tempo não pára!')",5000);
</script>
```

Uma caixa de diálogo será mostrada com a mensagem "O tempo não pára!" após 5 segundos.
A primeira parte do comando, antes da vírgula, indica a ação que deverá ser executada e a segunda, o tempo de espera em milésimos de segundos antes da execução.

18.1.6. window.clearTimeout()

Interrompe a execução de um temporizador antes do tempo marcado:

```
<script>
    n = 0;
    function atualiza(){
       n++;
       window.status = "contador = " + n;
       temp1 = window.setTimeout("atualiza()",1000);
    }
   atualiza();
</script>
<body>
   <a href="#" onClick="window.clearTimeout(temp1);"> Pára o contador </a>
</body>
```

No exemplo, foi atribuído um nome ao temporizador (temp1). O temporizador faz a chamada da função atualiza( ) a cada 1 segundo, gerando uma repetição recursiva. O método clearTimeout( ) interrompe a execução do temporizador temp1, especificado entre parênteses.

18.1.7. window.open()

Abre uma nova janela:

```
</script>
   janelinha=window.open("pag1.html","janela1","width=200,height=100");
</script>
```

No exemplo:

`janelinha` - é o nome dado a nova janela que será aberta;
`pag1.html` - é o "URL" da página que será carregada na nova janela. Pode-se abrir uma nova janela em branco omitindo o endereço da página, não deixando nenhum espaço entre as aspas.
`janela1` - é o nome da nova janela que será atribuído à propriedade *name* do objeto *window*.
`width` e `height` - fazem parte de um conjunto de recursos utilizados para configurar a nova janela.

Para habilitar ou desabilitar os recursos a seguir, utilize os valores "1" (yes) ou "0" (no):

toolbar	barra de ferramentas
location	barra de endereço
directories	diretórios
status	linha de status
menubar	barra de menu
scrollbars	barras de rolagem
resizable	redimensinamento
fullscreen	tela cheia

Para os demais recursos, especifique um valor:

height	altura
width	largura
top	disposição da janela a partir da margem superior da página
left	disposição da janela a partir da margem esquerda da página

18.1.8. window.***close()***

Fecha janelas:

```
<body>
 <input type="button" value="Fecha Janela Principal" onclick=window.close();">
   //window.close() fecha a janela principal do navegador, após a permissão do usuário!
 <input type="button" value="Fecha Nova Janela" onclick="janelinha.close();">
   //janelinha.close() fecha a nova janela aberta (janelinha) do exemplo anterior.
</body>
```

18.1.9. window.***print()***

Permite a impressão da página atual, que se encontra aberta.

```
<body>
   <input type="button" value="Imprime esta página" onclick="print();">
</body>
```

Clicando no botão "`Imprime esta página`", a página atual será impressa.

Não é necessário utilizar o nome do objeto (`window`) antes de propriedades ou métodos quando existir apenas uma janela aberta.

18.1.10. Objeto Frames

Frames são divisões da janela do navegador em múltiplos quadros ou painéis. Cada *frame* pode conter uma página diferente ou a saída de um *script*. Cada *frame* no *JavaScript* é representado por um objeto equivalente ao objeto *window,* com o qual trabalhamos. Seu nome é o mesmo do atributo *name* que você dá a ele na *tag <frame>*:

```
<html><head><title> Frames </title></head>
 <frameset cols="*,*">
   <frame name="ladoesquerdo" src="pagEsq.html">
   <frame name="ladodireito" src="pagDir.html">
 </frameset>
</html>
```

- **Array**

Cada objeto *frame* em uma janela é filho do objeto *window* pai.

Em vez de referenciar os *frames* de um documentos pelo nome, você pode utilizar o *Array frames[]*.
Os *frames* são indexados, iniciando com zero a partir da primeira *tag <frame>*.
`parent.frames[0]` é equivalente ao *frame name* "`ladoesquerdo" do exemplo.`
`parent.frames[1]` é equivalente ao *frame name* "`ladodireito" do exemplo.`

O conteúdo do atributo name é sensível às letras maiúsculas e minúsculas, não utilize caracteres especiais!

- **Hierarquia**

window e ***self*** referem-se à janela atual, onde se encontra o seu *script JavaScript*.
parent, refere-se à janela principal.

Se você utilizar *frames* aninhadas, muda um pouco:

window ou ***self*** ainda representam a janela atual, onde se encontra o *script*;(frame atual)
parent representa o *frameset* que contém o *frame* atual (pai deste *frame*);
top representa o *frameset* principal, que contém todos os outros *frames*. (pai de todos os frames).

Exemplo de *frames* aninhadas:

```
<html><head><title> Frames </title></head>
  <frameset rows="17%,*">
    <frame name="cabecalho" src="pagCab.html">
    <frameset cols="22%,*">
       <frame name="ladoesquerdo" src="pagEsq.html">
       <frame name="ladodireito" src="pagDir.html">
    </frameset>
  </frameset>
</html>
```

18.1.10.1. Frames de navegação

Através de `frames` de navegação, você pode controlar o documento em outro *frame*.
Vamos dividir a janela em três *frames* e criar uma página para abrir apenas no frame "`ladoesquerdo`", os demais ficarão em branco.

```
<html><head><title> Frames </title></head>
 <frameset cols="*,*,*">
   <frame name="ladoesquerdo" src="pagEsq.html">
   <frame name="meio"         src="about:blank">
   <frame name="ladodireito"  src="about:blank">
 </frameset>
</html>
```

No código fonte da página "`pagEsq.html`" vamos criar o seguinte *script*:

```
<html><head><title> Frames de Navegação </title></head>
 <body>
    <a href="parent.meio.location='pagMei.html';
            parent.ladodireito.location='pagDir.html';
            self.location='pagNova.html';"> Carrega páginas </a>
 </body> </html>
```

Atenção com as aspas e apóstrofes. Crie as páginas pagMei.html, pagDir.html e pagNova.html para o teste.

Ao clicarmos no link "Carrega páginas", que se encontra no quadro do lado esquerdo, serão carregadas três novas páginas, uma em cada quadro.

18.1.2. Objeto location

O objeto `location` armazena as informações referentes aos endereçamentos de URLs.

- **Propriedades**

*18.1.2.1. window.location.**href***

A propriedade *href* armazena o URL da página atual:

```
<script>
 document.write(window.location.href);
</script>
```

Este script mostra o endereço da página atual onde o script se encontra.

Você pode carregar uma outra página atribuindo um novo endereço:

```
<script>
 window.location.href="pag1.html";
</script>
```

A página "pag1.html" será carregada na janela atual.

*18.1.2.2. window.location.**protocol***

A propriedade `protocol` armazena o protocolo da página atual, basicamente `http`:

```
<script>
   document.write(window.location.protocol);
</script>
```

Será impresso o protocolo utilizado pela página atual.

- **Métodos:**

*18.1.2.3. window.location.**reload()***

O método `reload`() recarrega a página atual (atualiza):

```
<body>
   <a href="javascript:window.location.reload();"> Atualiza a Página </a>
</body>
```

Ao clicarmos no link "Atualiza a Página" a página atual será recarregada.

*18.1.2.4. window.location.**replace()***

O método `replace`() substitui a página atual por uma outra. O histórico de navegação não é atualizado! Portanto, não dá para retornar à página anterior através dos botões de navegação:

```
<body>
   <input type="button" value="Abre a Página 2"
            onclick="window.location.replace('pag2.html');">
</body>
```

Ao clicarmos no botão "Abre a Página 2", a "pag2.html" substituirá a página atual.

18.1.3. Objeto history

O objeto `history` armazena o histórico de navegação. *URLs* visitadas.

- **Propriedades**

*18.1.3.1. window.history.**length***

A propriedade `length` armazena o comprimento da lista de histórico de navegação, ou seja, a quantidade de localizações diferentes que o usuário visitou:

```
<script>
   document.write(history.length);
</script>
```

Será impresso um número correspondente ao comprimento da lista do histórico.

*18.1.3.2. window.history.**current***

A propriedade `current`, assim como a `window.location.href`, contém o endereço da página atual.

```
<script>
   document.write(history.current);
</script>
```

Será impresso o URL corrente.

*18.1.3.3. window.history.**next***

A propriedade `next` contém o endereço da próxima página, ou seja, aquela para onde o usuário foi e depois retornou, podendo recarregá-la novamente através do botão de navegação "*Avançar*":

```
<script>
   document.write(history.next);
</script>
```

Será impresso o próximo URL, que já foi anteriormente visitado.

*18.1.3.4. window.history.**previous***

A propriedade `previous` armazena o endereço da página anterior, ou seja, aquela de onde o usuário, podendo voltar novamente à ela através do botão de navegação "*Retornar*":

```
<script>
   document.write(history.previous);
</script>
```

Será impresso o URL anteriormente visitado.

- **Métodos**

18.1.3.5. window.history.go()

O método `go()` permite a navegação entre as páginas já vistadas pelo Internauta. Argumento com valor positivo avança para a próxima página já visitada. Equivale ao botão '"*Next*" do navegador:

```
<body>
   <a href="javascript:history.go(1);"> Avançar </a>
</body>
```

Ao clicarmos no link "Avançar", avançaremos para a próxima página (a qual já fomos anteriormente).

Argumento com valor negativo retrocede à página anterior. Equivale ao botão '*Back*' do navegador:

```
<body>
   <a href="javascript:history.go(-1);">Retroceder </a>
</body>
```

Ao clicarmos no link "Retroceder", retornaremos à página anterior.

Utilizando outros valores:

```
<body>
   <a href="javascript:history.go(-2);">Retroceder 2 Páginas </a>
</body>
```

O argumento "-2" faz retornar à antepenúltima página já visitada.

18.1.3.6. window.history.back()

O método `back`() retorna à página anterior. Equivale ao botão '*Back'* do navegador:

```
<body>
   <a href="javascript:history.back();">Retroceder</a>
</body>
```

Clicando no link "Retroceder" retrocederemos à página anterior. Equivale ao botão 'Back' do navegador.

18.1.3.7. window.history.forward()

O método `forward`() avança para apróxima página. Equivale ao botão '*Next'* do navegador:

```
<body>
   <a href="javascript:history.forward();">Avançar</a>
</body>
```

18.1.4. Objeto document

O objeto *document* armazena as informações referentes à página.

- **Propriedades**

18.1.4.1. window.document.URL

A propriedade `URL` assim como a `window.location.href` *e a* `window.history.current`, contém o endereço da página atua:

```
<script>
   document.write(document.URL);
</script>
```

18.1.4.2. window.document.title

A propriedade `title` armazena o título da página, que é exibido na barra de título do navegador:

```
<script>
   document.write(document.title);     // Exibe o título atual da página.
   document.title="Novo Título";       // Atribui um novo título à página.
</script>
```

18.1.4.3. window.document.referrer

A propriedade `referrer` armazena o endereço da página anterior, aquela que o usuário estava visualizando anteriormente, antes de clicar no *link* para carregar a página atual:

```
<script>
   document.write(document.referrer);
</script>
```

18.1.4.4. window.document.lastModified

A propriedade `lastModified` armazena a data da última atualização efetuada na página:

```
<script>
   document.write(document.lastModified);
</script>
```

Sempre que a página for carregada, será impresso a data da última atualização.

- **Métodos**

18.1.4.5. window.document.write()

O método `write`(), como já percebemos, imprime texto em um documento. Para imprimir um novo conteúdo, você deverá recarregar a página novamente.

```
<script>
   document.write("Imprimindo um texto...");
</script>
```

18.1.4.6. window.document.writeln()

O método `writeln`() também imprime texto, mas inclui um caractere de nova linha no final, permitindo que o seu texto seja a última coisa na linha (se funcionasse!).

```
<script>
   document.writeln("Imprimindo linha1...");
   document.writeln("Imprimindo linha2...");
</script>
```

18.1.4.7. window.document.open()

O método `open`() é utilizado para reescrever um documento primeiramente limpando o conteúdo anterior. É utilizado em novas janelas. Você abre um novo fluxo, escreve e depois fecha o fluxo.

```
<script>
  janela1=window.open("","janela1","width=200,height=100");
  function escreve(){
     janela1.document.open();
     janela1.document.write("Escrevendo na janela1 <br>");
     janela1.document.close();
     janela1.focus();
  }
</script>

<body>
   <input type="button" value="Escrever" onclick="escreve();">
</body>
```

Neste exemplo, sempre que pressionarmos o botão "*Escrever*", o conteúdo anterior da `janela1` será apagado e imprimirá novamente a frase "Escrevendo na janela1".

O método `.focus()` foi utilizado para que a `janela1` permanecesse em primeiro plano.

Comente (//) as linhas com os métodos `document.open()` e `document.close()` e você verificará que o conteúdo da `janela1` não será mais apagado e sim acumulado.

18.1.4.8. window.document.close()

O método `close`() fecha o novo fluxo aberto. Ele é utilizado com o método `document.open()`. (Exemplificado no exemplo anterior).

18.4.5. Objeto link

Os *links* (ligações) contidos em uma página são tratados como objetos no *Javascript*. Os *scripts* deverão ser chamados após o carregamento da página HTML, se forem chamados antes, os *links* não serão reconhecidos porque ainda não foram carregados. Se quisermos colocar os *scripts* no cabeçalho da página (*<head>*), devemos colocá-los dentro de funções e chamá-los utilizando tratadores de eventos (*onLoad*...) para serem executados após o carregamento completo da página.

- **Array**

O *Javascript* armazena os *links* do código HTML como elementos de um *Array*.

18.4.5.1. window.document.links[ ]

Cada *link*, por *definição,* faz parte do *Array links[]*. O endereço do primeiro *link* da página, criado com HTML, é armazenado no primeiro elemento do *Array links[]* de índice 0 (zero) e assim sucessivamente.

```
<body onload="document.write(document.links[0]);">
   <a href="pag1.html"> página um </a>
</body>
```

Refere-se ao primeiro link da página: file:///E:/pag1.html

- **Propriedades**

18.4.5.2. window.links[ ].href

A propriedade *href* armazena o endereço do *link* (idem ao item anterior):

```
<script>
  link1 = document.links[0].href;   //O endereço do primeiro link é salvo na variável link1
</script>
```

O endereço do primeiro link será armazenado na variável link1.

18.4.5.3. window.links.length

A propriedade *length* armazena o número *links* existentes na página:

```
<body onload="alert(document.links.length);">
  <a href="pag1.html"> Página Um </a>
  <a href="pag2.html"> Página Dois </a>
</body>
```

Imprime a quantidade de links existentes na página: 2.

18.4.6. Objeto anchor

As âncoras contidas em uma página também são tratadas como objetos no *Javascript*. Os *scripts* também deverão ser chamados após o carregamento da página HTML, se forem chamados antes, as âncoras não serão reconhecidas porque ainda não foram carregadas. Se quisermos utilizar os scripts no cabeçalho da página (*<head>*), devemos colocá-los dentro de funções e chamá-los utilizando tratadores de eventos (*onLoad*...).

- **Array**

O *Javascript* armazena as âncoras do código HTML como elementos de um *Array*.

18.4.6.1. window.document.anchors[ ]

Cada âncora, por definição, faz parte de um *Array* denominado *anchors[]*. A primeira âncora da página, criada com HTML, corresponde ao primeiro elemento do *Array anchors, de* índice 0 (zero) e assim sucessivamente:

```
document.anchors[0];
```

Refere-se à primeira âncora da página.

- **Propriedades**

18.4.6.2. window.anchors.name

A propriedade *name* armazena o nome da âncora contida na página:

```
<body>
   <a name="ancora1"> Texto... </a>
   <script>
      alert(document.anchors[0].name);
   </script>
</body>
```

Imprime o nome da primeira âncora da página: ancora1.

18.4.6.3. window.anchors.length

A propriedade *length* armazena o número de âncoras existentes na página:

```
<body>
   <a name="texto1"> Texto1 texto1 texto1... </a> <br>
   <a name="texto2"> Texto2 texto2 texto2... </a> <br>
   <script>
      alert(document.anchors.length);
   </script>
</body>
```

Imprime a quantidade de âncoras existentes na página: 2.

18.4.7. Objeto image

Como os demais elementos contidos em uma página HTML, as imagens também são tratadas como objetos no *Javascript*. Os *scripts* também deverão ser chamados após o carregamento da página HTML, se forem chamados antes, as imagens não serão reconhecidas porque ainda não foram carregadas. Se quisermos utilizar os *scripts* no cabeçalho da página (*<head>*), devemos colocá-los dentro de funções e chamá-los utilizando os tratadores de eventos (*onLoad, onClick*...).

- **Array**

O Javascript armazena as imagens do código HTML como elementos de um *Array*.

18.4.7.1. window.document.images[ ]

Cada imagem, por default, faz parte de um array denominado images. A primeira imagem da página, inserida com HTML, corresponde ao primeiro elemento da array images, de índice 0 (zero); a segunda imagem, índice 1 (um) e assim sucessivamente. Ex.:

```
document.images[0];
```

Refere-se à primeira imagem da página.

- **Propriedades**

Para falarmos sobre propriedades, vamos inserir uma imagem na página incluindo vários atributos.

```
<body>
   <img src="imagem1.jpg" name="img1" border="3" height="200" width="300"
        hspace="20" vspace="10" lowsrc="preimagem.gif">
</body>
```

Inclua os scripts para testes de propriedades no corpo da página após a inserção da imagem; ou condicione a execução dos scripts à tratadores de eventos.

18.4.7.2. window.document.images[ ].name

A propriedade *name* armazena o nome da imagem incluída na página:

```
<script>
  alert(document.images[0].name);        //Será impresso : img1
</script>
```

18.4.7.3. window.document.images[ ].border

A propriedade *border* armazena o valor da borda da imagem incluída na página:

```
<script>
  alert(document.images[0].border);      //Será impresso:3
</script>
```

18.4.7.4. window.document.images[ ].complete

A propriedade *complete* armazena os valores "*true*" ou "*false*", indicando se a imagem já foi carregada ou não na página:

```
<script>
  alert(document.images[0].complete);
</script>
```

//Se a primeira imagem da página foi carregada com sucesso será impresso "true", caso contrário, "false".

18.4.7.5. window.document.images[ ].height

A propriedade `height` armazena o valor da altura da imagem incluída na página:

```
<script>
  alert(document.images[0].height);   //Será impresso: 200.
</script>
```

18.4.7.6. window.document.images[ ].width

A propriedade *width* armazena o valor da largura da imagem incluída na página:

```
<script>
  alert(document.images[0].width);    //Será impresso: 300.
</script>
```

18.4.7.7. window.document.images[ ].hspace

A propriedade *hspace* armazena o valor do espaçamento horizontal da imagem incluída na página:

```
<script>
  alert(document.images[0].hspace);   //Será impresso: 20.
</script>
```

18.4.7.8. window.document.images[ ].vspace

A propriedade *vspace* armazena o valor do espaçamento vertical da imagem na página:

```
<script>
  alert(document.images[0].vspace);   //Será impresso: 10.
</script>
```

18.4.7.9. window.document.images[ ].lowsrc

A propriedade *lowsrc* armazena o endereço da pré-imagem, carregada antes do carregamento da imagem definitiva:

```
<script>
  alert(document.images[0].lowsrc);   //Será impresso: "preimagem.gif".
</script>
```

18.4.7.10. window.document.images[ ].src

A propriedade *src* armazena o endereço da imagem definitiva:

```
<script>
  alert(document.images[0].src);      //Será impresso: ""imagem1.jpg".
</script>
```

Podemos utilizar além do *Array* `images[ ]`, o nome do objeto definido através dos atributos "id" ou "name", incluídos na tag <img> da HTML, para nos referir ao objeto:

```
<body onload="alert(document.imag1.width); alert(imag1.height);">
   <img id="imag1" src="imagem1.gif" width="120" height="200">
</body>
```

//Será impresso: 120 (largura) e 200 (altura)

- **Tratadores de Eventos em Imagens:**

`onLoad` - Quando termina o carregamento das imagens na página.
`onAbort` - Quando o usuário interrompe o carregamento da página antes de carregar as imagens.
`onError` - Quando a imagem está corrompida ou não pode ser localizada (*pathname* incorreto).

Vide exemplos no capítulo 16 – "Respondendo a Eventos"

18.4.7.11. Rollovers

Utilizando o objeto *image associado a tratadores de eventos* podemos criar animações com imagens, um deles é o *Rollover*, onde uma imagem é substituída por outra quando o ponteiro do mouse passa sobre ela. Este efeito é muito utilizado em links:

```
<body>
  <a href="#" onmouseover="document.images[0].src='imagem2.jpg';"
              onmouseout="document.images[0].src='imagem1.jpg';">
     <img src="imagem1.jpg" width="250">
  </a>
</body>
```

Neste exemplo, a primeira imagem da página: `"imagem1.jpg"(incluída através da tag HTML)` é substituída pela `"imagem2.jpg"` quando o ponteiro do mouse passa sobre ela.
Quando o ponteiro do mouse se deslocada da imagem, a imagem `"imagem1.jpg"` retorna.
A largura das duas imagens é determinada pelo atributo `width` da tag `<img>`.
O *link* está acessando a própria página (#), mas poderíamos acessar um URL qualquer.

18.4.7.12. Pré-Carregamento de Imagens

As imagens são os últimos e mais demorados itens a serem carregados em uma página, principalmente se os arquivos forem "pesados". Para otimizarmos o carregamento delas, utilizamos o pré-carregamento de imagens, que consiste em carregar previamente as imagens no *cache* antes de sua exibição na página.

Para isto, vamos criar uma instância do objeto *Image*, que é um *Array*, e armazenar previamente todas as imagens que serão posteriormente exibidas:

```
<htmL>
 <head>
  <script>
    imgs = new Image( );
    imgs[0]= "imagem0.jpg";
    imgs[1]= "imagem1.jpg";
  </script>
 </head>

 <body>
  <img src="" name="imag1"> <br>
  <img src="" name="imag2">
  <script>
    document.images[0].src = imgs[0];
    document.imag2.src=imgs[1];
  </script>
 </body>
<htmL>
```

Comentando o exemplo: Pré-carregamos duas imagens no objeto `Image`, que é um *Array* por definição.
Utilizando HTML, inserimos duas *tags* de imagens sem os *pathnames* (endereços) no corpo do documento.
Através do *Javascript* carregamos no lugar da primeira imagem do documento, que corresponde ao elemento de índice 0 (zero) do *Array* `images[]`, a imagem armazenada no primeiro elemento do objeto `imgs[]`, que também é um *Array*;

Para o carregamento da segunda imagem do documento, utilizamos o nome definido através do atributo `name` de sua *tag* `<img>` e atribuímos o segundo elemento do objeto `imgs[]`.
Podemos inserir imagens no documento através destes dois modos, utilizando o nome da imagem inserido no atributo `name` ou utilizando o *Array* de imagens `images[]`.

18.4.7.13. Animação

O objeto *Image* associado ao temporizador *setTimeout()* permite a criação de animações através da exibição de uma seqüência de imagens (sobre um mesmo tema, com pequenas variações entre uma e outra), que exibidas sucessivamente no mesmo lugar e em um curto intervalo de tempo entre uma e outra, simulará um movimento. (Semelhante à técnica de desenho animado).

Exemplo 1: (Utilizando nove imagens, simulando um ratinho em movimento):

```
<html><head><title> Animação 1 </title>
<script language="javascript">
  img = new Image();                    // Pré-carregamento de imagens através de um loop.
  for(i=0;i<9;i++){
     img[i]="mouse"+i+".gif";
  }
  i=0;
  function mostra(){                    //Exibição das imagens utilizando recursividade.
    if (i>8) i=0;
    document.images[0].src=img[i];
    i++;
    ID=window.setTimeout("mostra()",150);
  }
</script>
<body bgcolor="brown">
<img src="mouse0.gif" border="0">
<br>
<input type="button" value="Start" onclick="mostra();">
    
<input type="button" value="Stop" Imagem" onclick="window.clearTimeout(ID);">
</body>
</html>
```

Utilizando propriedades do objeto `image` podemos alterar dinamicamente o tamanho de uma imagem.

Exemplo 2: (Simulação de um coração batendo)

```
<html>
<head>
<title> Animação 2 </title>
<script language="javascript">
  var vai=0;
  function carregaImagem(){
    imagem1.src="coração.jpg";
  }
  function bate(){
    if (vai==0){
       imagem1.width=imagem1.width+10;
       imagem1.height=imagem1.height+5;}
    else
       {imagem1.width=imagem1.width-10;
       imagem1.height=imagem1.height-5;}
    if (imagem1.width>480)
       vai=1;
    if (imagem1.width<400)
       vai=0;
    setTimeout("bate()", 50);}
</script>
</head>
```

```
<body onLoad="carregaImagem();">
<img name="imagem1" src="" align="center" width="400" height="250">
 <script language="javascript">
   bate();
 </script>
</body>
</html>
```

18.4.8. Objeto form

Para falarmos com detalhes sobre o objeto *form*, temos que primeiro criar um formulário em HTML. Tanto a *tag* form como os elementos criados através da HTML possuem propriedades, métodos, tratadores de eventos e *Arrays*. Descreveremos a seguir cada atributo destes elementos:

<form name="form1" action="mailto:email@prov.br" method="POST" enctype="text/plain"**>**
...
</form>

- **Propriedades**

`forms[ ].name`	Armazena o conteúdo do atributo *name* da tag form do formulário.
`forms[ ].action`	Armazena o conteúdo do atributo *action* da tag form do formulário.
`forms[ ].method`	Armazena o conteúdo do atributo *method* da tag form do formulário.
`forms[ ].target`	Armazena o conteúdo do atributo *target* da tag form do formulário.
`forms[ ].encoding`	Armazena o conteúdo do atributo enctype da tag form do formulário.
`forms[ ].length`	Armazena a quantidade de elementos (campos) existentes no formulário.

Para referenciar um formulário, utilizamos o Array forms[ ] ou o nome do objeto incluído no atributo "name" da tag <form> da HTML.

- **Métodos**

`forms[ ].submit( )`	Envia o formulário indicado.
`forms[ ].reset( )`	Limpa o formulário indicado.

- **Array**

`.forms[ ]`	Conjunto de formulários dentro de uma página.

- **Tratadores de Eventos**

`OnSubmit`	É acionado quando o usuário pressiona o botão *Submit* do formulário.
`OnReset`	É acionado quando o usuário pressiona o botão *Reset* do formulário.

Com o tratador de eventos `onSubmit`, podemos fazer o tratamento dos dados, como verificar se os campos foram corretamente preenchidos, antes do usuário enviar o formulário.

18.4.8.1. Objeto elements

- **Array**

`.elements[ ]`	Conjunto de elementos contidos em um formulário.

- **Propriedades**

`elements[ ].name`	Armazena o nome do elemento.
`elements[ ].length`	Armazena o comprimento do elemento.

O *Array* elements está subordinado a um formulário. Podemos utilizar o nome do objeto, definido através do atributo "name" incluído na tag de cada elemento do formulário (<textarea>,<button>...).

18.4.8.2. Objetos text e textarea

- **Propriedades**

`.name`	Armazena o nome do elemento.
`.defaultValue`	Armazena o valor inicial (padrão) do elemento.
`.value`	Armazena o valor atual do elemento (preenchido pelo usuário).

- **Métodos dos objetos text e textarea**

`.focus( )`	Posiciona o cursor sobre um elemento *text* ou *textarea*.
`.blur( )`	Retira o cursor do elemento.
`.select( )`	Seleciona o conteúdo inserido no elemento.

- **Tratadores de Eventos**

`onFocus`	É acionado quando o usuário posiciona o cursor no elemento.
`onBlur`	É acionado quando o usuário retira o cursor do elemento.
`onChange`	É acionado quando o usuário altera o valor do elemento e retira o cursor.
`onSelect`	É acionado quando o usuário seleciona um elemento.

18.4.8.3. Objeto checkbox

- **Propriedades**

`.name`	Armazena o nome do elemento.
`.value`	Armazena o valor do elemento que será enviado ao servidor.
`.checked`	Armazena o valor *true* ou *false*, de acordo com a situação inicial do campo.
`.defaultChecked`	Armazena o valor *true* ou *false,* de acordo com a manipulação do usuário.

- **Método**

`.click( )`	Dá um clique em um elemento.

- **Tratador de Evento**

`onClick`	É acionado quando o usuário dá um clique em um elemento.

18.4.8.4. Objeto radio

- **Propriedades de grupo do objeto radio**

`.name`	Armazena o nome do elemento.
`.length`	Armazena a quantidade de botões.
`.value`	Armazena o valor do elemento que será enviado ao servidor.

- **Propriedades dos botões do objeto radio**

`.defaultChecked`	Armazena o valor *true* ou *false* inicial do elemento (botão), que será enviado ao servidor.
`.checked`	Armazena o valor *true* ou *false* atual do elemento, que será enviado ao servidor.

- **Método**

`.click( )`	Dá um clique em um elemento.

- **Tratador de Evento**

`onClick`	É acionado quando o usuário dá um clique em um elemento.

- **Array**

`.Nome[]`	`Nome` deve ser o nome do elemento definido no atributo *name*, que será utilizado como o nome do *Array*.

18.4.8.5. Objeto select

- **Propriedades de grupo do objeto select**

`.name`	Armazena o nome do elemento.
`.length`	Armazena a quantidade de opções.
`.selectedIndex`	Armazena o índice da opção selecionada.

- **Propriedades das opções do objeto select**

`.index`	Armazena o índice da opção.
`.defaultSelected`	Armazena o valor *true* ou *false* inicial do elemento.
`.selected`	Armazena o valor *true* ou *false* atual do elemento.
`.name`	Armazena o nome da opção.
`.text`	Armazena o texto contido no botão da opção.
`.value`	Armazena o valor da opção que será enviado ao servidor.

- **Métodos**

`.focus( )`	Posiciona o cursor sobre um elemento.
`.blur( )`	Retira o cursor do elemento.

- **Tratador de Evento**

`onFocus( )`	Ocorre quando o cursor é posicionado sobre um elemento do objeto *select*.
`onBlur( )`	Ocorre quando o cursor é retirado do elemento.
`onChange( )`	Ocorre quando o usuário seleciona uma opção e retira o curso do elemento.

- **Array**

`.options[]`	*Array* de opções do objeto *select*.

18.4.8.6. Exemplos de JavaScript utilizando campos de Formulários

18.8.6.1. Validação de Formulários

Exemplo 1: (Consistência de campos utilizando o tratador de eventos *onsubmit* para o envio de dados)

```
<html>
<head> <title>formulário</title>
<script language="Javascript">
function enviar(){
    if (document.form1.nome_usuario.value.length<3){
       alert("Nome Incompleto!");
       document.form1.nome_usuario.select();
       document.form1.nome_usuario.focus();
       return false;
    }
    if (document.form1.email_usuario.value.indexOf("@")<0){
       alert("Email Inválido!");
       document.form1.email_usuario.select();
       document.form1.email_usuario.focus();
       return false;
    }
    if (document.form1.comentario.value.length<3){
       alert("Faça só um comentariozinho!!!");
       document.form1.comentario.select();
       document.form1.comentario.focus();
       return false;
    }
    if (document.form1.pesquisa[0].checked==true)
       alert("Legal, pessoas inteligentes sempre gostam da minha página!!!");
    else {
       if (document.form1.pesquisa[1].checked==true)
           alert("Poxa, você não gostou da minha página???");
       else {
```

```
            alert("É para responder a pesquisaaa!");
            return false;
        }
    }
}
</script>
</head>
<body bgcolor="#666688" marginwidth="333" leftmargin="333">
<h1 align="center"><font color="#FFFFFF"> Formulário </font> </h1>
<hr>
<form name="form1" action="mailto:seuemail@prov.com.br" method="POST"
enctype="text/plain" onSubmit="return enviar();">
<i>Nome:</i><br>
<input type=text name="nome_usuario" size="52" title="Digite seu
nome"><BR><BR>
<i>E-mail:</i><br>
<input type=text name="email_usuario" size="52" title="Digite seu E-
mail"><BR><BR>
<i>Comentário:</i><br>
<textarea name="comentario" rows="05" cols="40" title="Digite seu
comentário"></textarea>
<br><br><center>
<h3> Pesquisa: </h3>
<i>Gostou da minha página???</i><br><br>
<input type="radio" name="pesquisa" value="s" title="Clique aqui!!!"> Sim
      
<center>
<input type="radio" name="pesquisa" value="n" title="Não clique aqui!">
Não<br><br><br>
<input type="submit" value="Enviar">       
<input type="reset"  value="Limpar">
</center>
</form>
</body> </html>
```

Exemplo 2: (Consistência de campos utilizando o método *submit()* para o envio de dados)

```
<html>
<head> <title>formulário</title>
<script language="Javascript">
function enviar(){
    if (document.form1.nome_usuario.value.length<3){
       alert("Nome Incompleto!");
       document.form1.nome_usuario.select();
       document.form1.nome_usuario.focus();
    }
    else
    if (document.form1.email_usuario.value.indexOf("@")<0){
       alert("Email Inválido!");
       document.form1.email_usuario.select();
       document.form1.email_usuario.focus();
    }
```

```
    else document.form1.submit();
}
</script>
</head>
<body bgcolor="#6666BB" marginwidth="333" leftmargin="333">
<h1 align="center"><font color="#FFFFFF"> Formulário </font> </h1>
<hr>
<form name="form1" action="mailto:seuemail@prov.com.br" method="POST"
enctype="text/plain">
<i>Nome:</i><br>
<input type=text name="nome_usuario" size="52" title="Digite seu
nome"><BR><BR>
<i>E-mail:</i><br>
<input type=text name="email_usuario" size="52" title="Digite seu E-
mail"><BR><BR>
<input type="button" value="Enviar" onclick="enviar();">
      
<input type="reset"  value="Limpar">
</center>
</form>
</body>
</html>
```

18.4.8.6.2. Menu de Navegação

Exemplo 1: (Menu utilizando o tratador de eventos *onChange* na caixa de seleção)

```
<html>
<head>
   <title>formulário</title>
<script language="Javascript">
function carrega_pagina(){
   i = document.form1.paginas.selectedIndex;
   pag = document.form1.paginas.options[i].value;
   window.location = pag;
}
</script>
</head>
<body bgcolor="#6666BB" marginwidth="333" leftmargin="333">
<h1 align="center"><font color="#FFFFFF"> Menu de Navegação </font> </h1>
<hr><center>
<form name="form1">
 <select name="paginas" onchange="carrega_pagina()">
  <option value="pagina1.html"> Página Um </option>
  <option value="pagina2.html"> Página Dois </option>
  <option value="pagina3.html"> Página Três </option>
  <option value="pagina4.html"> Página Quatro </option>
 </select>
</form>
</center>
</body>
</html>
```

Exemplo 2: (Menu utilizando o tratador de eventos *onClick* em um botão genérico)

```
<html>
<head>
   <title>formulário</title>
<script language="Javascript">
function carrega_pagina(){
   i = document.form1.paginas.selectedIndex;
   pag = document.form1.paginas.options[i].value;
   window.location = pag;
}
</script>
</head>
<body bgcolor="#6666BB" marginwidth="333" leftmargin="333">
<h1 align="center"><font color="#FFFFFF"> Menu de Navegação </font> </h1>
<hr><center>
<form name="form1">
 <select name="paginas">
  <option value="pagina1.html"> Página Um </option>
  <option value="pagina2.html"> Página Dois </option>
  <option value="pagina3.html"> Página Três </option>
  <option value="pagina4.html"> Página Quatro </option>
 </select>
<input type="button" value="OK" onclick="carrega_pagina();">
</form>
</center>
</body>
</html>
```

18.4.8.6.3. Relógio

```
<html>
  <script language="JavaScript">
    function relogio(){
      var hoje      = new Date();
      var horas     = hoje.getHours();
      var minutos   = hoje.getMinutes();
      var segundos  = hoje.getSeconds();
      var val_horas = ((horas < 10) ? "0":"") + horas;
      val_horas    += ((minutos < 10) ? ":0":":") + minutos;
      val_horas    += ((segundos < 10) ? ":0":":") + segundos;
      document.relog.visor.value = val_horas;
      window.setTimeout("relogio()",500);
    }
  </script>
  <head>
    <title>Java Script - Relógio</title>
  </head>

  <body onLoad="relogio();" bgcolor="FFFFCC">
    <center>
      <hr size="2">
```

```
      <font size=+3 face="Colonna MT,Brush Script MT">Relógio</font>
      <hr size="2">
      <br>
      Relógio:<br>
      <form name="relog" onSubmit="0">
        <input type="text" name="visor" size="05">
      </form>
    </center>
  </body>
</html>
```

18.4.8.6.4. Letreiro Digital

```
<html>
<head>
<title>Marquee comJavaScript</title>

<script>
i=0;
function marquee(){
mens=" Coloque a sua mensagem aqui...   ";
document.f1.texto.value = mens.substring(i,mens.length)+mens.substring(0, i-1);

if (i < mens.length){
  i++;
}
else {
  i=0;
}

window.setTimeout("marquee()",200)
}

</script>
</head>

<body onLoad="marquee()" bgcolor="#546365">
 <center>
  <form name="f1">
   <input type="text" name="texto" size="40">
  </form>
 </center>
</body>
</html>
```

19. Detecção e Diferenças entre Navegadores

O *Javascript* possui um objeto chamado *navigator* que armazena informações sobre o navegador do usuário. O *navigator* não é parte da hierarquia de objetos. Ele contém várias propriedades que servem para identificar o navegador do usuário.

19.1. Informações sobre o Navegador Atual

- **Propriedades:**

navigator**. appCodeName**	Nome interno do código do navegador, normalmente "*Mozilla*".
navigator**.appName**	Nome do navegador.
navigator**.appVersion**	Versão utilizada pelo navegador.
navigator**.userAgent**	Cabeçalho do usuário-agente, uma string que o navegador envia para o servidor Web quando solicita uma página da Web. Inclui informações completas da versão.
navigator**.systemLanguage**	Idioma. É armazenado como um código, como "*in*" para inglês, "*pt-br*" para português do Brasil...
navigator**.platform**	Plataforma do computador utilizado pelo navegador, como "*Win32*", "*MacPPC*"...

Todas as propriedades do navegador poderão ser impresssas através do *loop "for in"*:

Exemplo 1: (Imprime todas as propriedades do navegador utilizado)

```
<html>
<head> <title> Navegadores </title>
</head>
<body>
<h1 align="center"> Propriedades do Navegador Atual </h1>
<script language="javascript">
    for (i in navigator){
         document.write("propriedade: " + i + "<br>") ;
         document.write("conteúdo: " + navigator[i] + "<hr>");
    }
  </script>
</body>
</html>
```

Exemplo 2: (Identifica o Navegador Utilizado)

```
<html>
<head>
<script language="javascript">
if (navigator.appName.indexOf("Netscape") > -1)
    alert("Você esta utilizando o Netscape ou um Navegador Compatível");
else
if (navigator.appName.indexOf("Microsoft") > -1)
    alert("Você esta utilizando o I. Explorer ou um Navegador Compatível");
else
    alert("Seu navegador não é o Internet Explorer nem o Netscape");
</script>
</head>
```

Alguns navegadores possuem como nome interno o nome de outro navegador para indicar a compatibilidade. Ex.: O conteúdo da propriedade appName do Mozilla Firefox é Netscape.

20. Outros Scripts

20.1. Cookies

Um *cookie* é uma informação (uma seqüência de caracteres) que os sites enviam aos navegadores dos usuários e as mantêm na memória do computador.

Ao encerrar a sessão com o navegador, todos os cookies que ainda não expiraram são gravados em um arquivo (*cookie file*).

Nas visitas posteriores o navegador reenvia os dados para o servidor dono do *cookie* (*site*).

Os *sites* geralmente usam os *cookies* para distinguir usuários e memorizar preferências.

Muitas pessoas julgam que os cookies possam ser usados pelo servidor para obter informações a seu respeito ou invadir o seu disco rígido e obter dados a partir de lá, o que não é verdade. Todas as informações gravadas em um cookie são informações que você forneceu voluntariamente ao servidor (de uma forma ou de outra).

- **Propriedade:**

document**.cookie**

Exemplo:

Parte 1 - Enviando Dados para o Cookie

```
<html>
<head>
<title>Página armazenando um cookie</title>

<script language="javascript">
hoje= new Date();
function criaCookie(){
         linha1=document.formCookie1.seunome.value;
         linha2="*" + document.URL;
         linha3="*" + hoje.toGMTString();
         document.cookie=linha1+linha2+linha3;
}
</script>
</head>
<body bgcolor="beige">
<h2 align="center"> Carrega dados no Cookie </h2>
<form name="formCookie1">
 Seu nome:<br><input type="text" name="seunome" size="50"
onChange="criaCookie()">
</form>
<input type="button" value="Lê cookie em outra página"
onClick="window.location='cookieExibe.html';"><br><br>
<input type="button" value="Limpa Cookie" onClick="document.cookie='';>
</body>
</html>
```

Parte 2 – Exibindo os Dados da Página Anterior Armazenados no Cookie

```
<html>
<head>
<title> Página exibindo cookie </title>

<script language="javascript">
  function exibeCookie(){
    meuCookie=document.cookie;
    dados=meuCookie.split("*");
    document.formCookie2.usuario.value=dados[0];
    document.formCookie2.pagina.value=dados[1];
    document.formCookie2.dataCookie.value=dados[2];
  }
</script>

</head>
<body bgcolor="#6688BB" onLoad="exibeCookie()">

 <h2 align="center"> Exibe Cookie </h2>

  <form name="formCookie2">
    Nome:<br><input type="text" name="usuario" size="50"><br><br>
    Página anterior:<br><input type="text" name="pagina" size="50"><br><br>
    Data da criação do cookie:<br><input type="text" name="dataCookie"
    size="50"><br>
  </form>
</body>
</html>
```

20.2. Descendo a Barra de Rolagem Automaticamente

```
<html>
 <head>
   <title> Sobe Tela </title>
   <script language="javascript">
     contador=0;

     function sobetela(){
       if ((contador+=3)>600)
         contador=0;
         self.scroll(0,contador);
         setTimeout('sobetela()',200);
     }
   </script>
<body onLoad="sobetela()">
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
```

```
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
    Texto..<br> texto....<br> texto....<br>
</body>
</html>
```

20.3. Exibindo a Resolução da Tela

```
<html>
<head><title> screen </title>
</head>
  <script language="javascript">
    document.write("<b> Sua resolução:<b>");
    document.write(screen.width+"x"+screen.height);
  </script>
<body bgcolor="beige">
</body>
</html>
```

20.4. Movimentando uma Imagem através das Setas de Direção utilizando DHTML

(DHTML = HTML + CSS + JavaScript)

```
<html>
 <head>
  <title> Setas de direção</title>
   <script>
      function clique() {
         tecla = event.keyCode;
         window.status="Você pressionou a tecla " + tecla;

         if (tecla==37)                    // Seta à esquerda
             img1.style.pixelLeft-=5;      // Instrução para o I.E.
         else
         if (tecla==38)                    // Seta para cima
             img1.style.pixelTop-=5;       // Instrução para o I.E.
         else
         if (tecla==39)                    // Seta À direita
             img1.style.pixelLeft+=5;
         else
         if (tecla==40)                    // Seta para baixo
             img1.style.pixelTop+=5;
          else
              alert("tecla iválida!");
      }
// Instrução para o Netscape : document.img1.left e document.img1.top
   </script>
 </head>
```

```
 <body onKeyDown="clique();">
  <img id="img1" src="imagem1.jpg" width="150"
      style="position:absolute; top='100'; left='100';">
 </body>
</html>
```

20.5. Executando um arquivo de Som

```
<html>
 <head>
  <script>
    function tocar(som){          // som recebe o índice  0 (zero) que será utilizado no Array.
      document.embeds[som].play();// Instução para o Nestcape
     }
  </script>
 </head>
 <body onKeyDown="tocar(0);">
   <embed src="Sinfonia No 9 de Beethoven (scherzo).wma" hidden="true"
    autostart="false">
 </body>
</html>
```

// O I.E. utiliza o método .run().

embeds[]é um Array de de sons e vídeos. Podemos incluir vários arquivos de som na página e executá-los aleatoriamente,variando os índices randomicamente.

21. Bibliografia

- **Javascript – O Guia definitivo**
  David Flanagan – trad. Edson Furmankiewicz
  Bookman Companhia Editora
  2004

- **Iniciando em Javascript 1.5**
  Adrian Kingsley-Hughes e Kathie Kingsley-Hughes
  Editora Makron Books
  2001

- **Javascript 1.3. - Aprenda em 24 Horas**
  Michael Moncur
  Editora Campus
  1999

22. Sobre Sites...

Pesquise na Internet por **javascript** e encontrará vários sites com modelos e exemplos de scripts, como também tutoriais e apostilas. Não vou indicar nenhum porque os endereços podem mudar.
Vale a pena procurar. *Bom estudo!*