

## Manual JavaScript

### Operadores Matemáticos

São operadores a serem utilizados em cálculos, referências de indexadores e manuseio de strings. Ao longo do manual estes operadores serão largamente utilizados, dando, assim, uma noção mais precisa do seu potencial.

- + adição de valor e concatenação de strings
- subtração de valores
- \* multiplicação de valores
- / divisão de valores
- % obtém o resto de uma divisão:

Ex: 150 % 13 retornará 7

7 % 3 retornará 1

**+=** concatena /adiciona ao string/valor já existente. Ou seja:

$x += y$  é o mesmo que  $x = x + y$

da mesma forma podem ser utilizados: **-=** , **\*=** , **/=** ou **%=**

Um contador pode ser simplificado utilizando-se : **X++** ou **X--** o que equivale as expressões:

$X = X + 1$  ou  $X = X - 1$  respectivamente.

Para inverter sinal: **X = -X** **negativo** para **positivo** ou **positivo** para **negativo**.

### Operadores Lógicos

São operadores a serem utilizados em comandos condicionais, tais como: **IF** , **FOR** e **WHILE**. Os comandos condicionais serão vistos mais a frente.

- ==** Igual
- !=** Diferente
- >** Maior
- >=** Maior ou Igual
- <** Menor
- <=** Menor ou Igual
- &&** E
- ||** Ou

## *Controles Especiais*

**\b** - backspace

**\f** - form feed

**\n** - new line characters

**\r** - carriage return

**\t** - tab characters

// - Linha de comentário

**/\*...\*/** - Delimitadores para inserir um texto com mais de uma linha como comentário.

Os delimitadores naturais para uma string são " ou ' . Caso seja necessário a utilização destes caracteres como parte da string, utilize \ precedendo " ou ' .

Ex. alert ("Cuidado com o uso de \" ou \' em uma string")

---

## *Comandos Condicionais*

São comandos que condicionam a execução de uma certa tarefa à veracidade ou não de uma determinada condição, ou enquanto determinada condição for verdadeira.

São eles:

### **Comando IF**

**if** (condição)

{ ação para condição satisfeita }

[ **else**

{ ação para condição não satisfeita } ]

Ex.

if (Idade < 18)

{ Categoria = "Menor" }

else

{ Categoria = "Maior" }

### **Comando FOR**

**for** ( [inicialização/criação de variável de controle ;]

[condição ;]

```
[incremento da variável de controle] )
{ ação }
```

Ex.

```
for (x = 0 ; x <= 10 ; x++)
  { alert ("X igual a " + x) }
```

## Comando WHILE

Executa uma ação enquanto determinada condição for verdadeira.

```
while (condição)
  { ação }
```

Ex.

```
var contador = 10
  while (contador > 1)
    { contador-- }
```

## Move condicional

```
receptor = ( (condição) ? verdadeiro : falso)
```

Ex.

```
NomeSexo = ((VarSexo == "M") ? "Masculino" : "Feminino")
```

OBS:

Nos comandos FOR e WHILE a diretiva "**break**" pode ser utilizada para interromper a condição principal e sair do loop. Da mesma forma, a diretiva "**continue**" interrompe uma ação (se determinada condição ocorrer) mas volta para o loop.

Diretivas/condições entre [ ] significam que são opcionais.

## *Eventos*

São fatos que ocorrem durante a execução do sistema, a partir dos quais o programador pode definir ações a serem realizadas pelo programa.

Abaixo apresentamos a lista dos eventos possíveis, indicando os momentos em que os mesmos podem ocorrer, bem como, os objetos passíveis de sua ocorrência.

- onload** - Ocorre na carga do documento. Ou seja, só ocorre no BODY do documento.
- onunload** - Ocorre na descarga (saída) do documento. Também só ocorre no BODY.
- onchange** - Ocorre quando o objeto perde o focus e houve mudança de conteúdo.  
válido para os objetos Text, Select e Textarea.
- onblur** - Ocorre quando o objeto perde o focus, independente de ter havido mudança.  
válido para os objetos Text, Select e Textarea.
- onfocus** - Ocorre quando o objeto recebe o focus.  
válido para os objetos Text, Select e Textarea.
- onclick** - Ocorre quando o objeto recebe um Click do Mouse.  
válido para os objetos Buton, Checkbox, Radio, Link, Reset e Submit.
- onmouseover** - Ocorre quando o ponteiro do mouse passa por sobre o objeto.  
válido apenas para Link.
- onselect** - Ocorre quando o objeto é selecionado.  
válido para os objetos Text e Textarea.
- onsubmit** - Ocorre quando um botão tipo Submit recebe um click do mouse.  
válido apenas para o Form.
- 

## *Criando Variáveis*

A variável é criada automaticamente, pela simples associação de valores a mesma.

Ex. NovaVariavel = "Jose"

Foi criada a variável de nome **NovaVariavel** que, passou a conter a string **Jose**.

As variáveis podem ser **Locais** ou **Globais**. As variáveis que são criadas dentro de uma função são Locais e referenciáveis apenas dentro da função. As variáveis criadas fora de funções são Globais, podendo serem referenciadas em qualquer parte do documento.

Desta forma, variáveis que precisam ser referenciadas por várias funções ou em outra parte do documento, precisam ser definidas como globais.

Embora não seja recomendável, em uma função, pode ser definida uma variável local com o mesmo

nome de uma variável global. Para isso utiliza-se o método de definição **var**.

Ex. Variável Global : MinhaVariavel = ""  
 Variável Local : var MinhaVariavel = ""

---

## *Escrevendo no Documento*

O JavaScript permite que o programador escreva linhas dentro de uma página (documento), através do método **write**. As linhas escritas desta forma, podem conter textos, expressões JavaScript e comandos Html. As linhas escritas através deste método aparecerão no ponto da tela onde o comando for inserido.

Ex:  

```
<script>
valor = 30
document.write ("Minha primeira linha")
document.write ("Nesta linha aparecerá o resultado de : " + (10 * 10 + valor))
</script>
```

A idéia do exemplo acima é escrever duas linhas. Entretanto o método **write** não insere mudança de linha, o que provocará o aparecimento de apenas uma linha com os dois textos emendados. Para evitar este tipo de ocorrência, existe o método **writeln** que escreve uma linha e spaceja para a seguinte. Entretanto, em nossos testes, este comando não surtiu efeito, obtendo-se o mesmo resultado do método **write**. A solução encontrada para esta situação foi a utilização do comando de mudança de parágrafo da linguagem Html.

Ex:  

```
<script>
valor = 30
document.write ("<p>Minha primeira linha</p>")
document.write ("<p>Nesta linha aparecerá o resultado de : " + (10 * 10 + valor) + "</p>")
</script>
```

Isto resolve a questão da mudança de linha, porém, vai gerar uma linha em branco, entre cada linha, por se tratar de mudança de parágrafo. Caso não seja desejado a existência da linha em branco, a alternativa é utilizar o comando Html **<br>** que apenas muda de linha.

Ex:  

```
<script>
valor = 30
```

```
document.write("<br>Minha primeira linha")
document.write("<br>Nesta linha aparecerá o resultado de : " + (10 * 10 + valor) )
</script>
```

---

## *Mensagens*

Existem três formas de comunicação com o usuário através de mensagens.

### **Apenas Observação:**

**alert** ( mensagem )

Ex.  
alert ("Certifique-se de que as informações estão corretas")

### **Mensagem que retorna confirmação de OK ou CANCELAR:**

**confirm** (mensagem)

Ex.  
if (confirm ("Algo está errado...devo continuar??"))  
 { alert("Continuando") }  
else  
 { alert("Parando") }

### **Recebe mensagem via caixa de texto Input:**

Receptor = **prompt** ("Minha mensagem", "Meu texto")

Onde:  
**Receptor** é o campo que vai receber a informação digitada pelo usuário  
**Minha mensagem** é a mensagem que vai aparecer como **Label** da caixa de input  
**Meu texto** é um texto, opcional, que aparecerá na linha de digitação do usuário

Ex.  
Entrada = prompt("Informe uma expressão matemática", "")  
Resultado = eval(Entrada)  
document.write("O resultado é = " + Resultado)

---

## *Criando Funções*

Uma função é um set de instruções, que só devem ser executadas quando a função for acionada.

A sintaxe geral é a seguinte:

```
function NomeFunção (Parâmetros)
    { Ação }
```

Suponha uma função que tenha como objetivo informar se uma pessoa é maior ou menor de idade, recebendo como parâmetro a sua idade.

```
function Idade (Anos) {
    if (Anos > 17)
        { alert ("Maior de Idade") }
    else
        { alert ("menor de Idade") }
}
```

Para acionar esta função, suponha uma caixa de texto, em um formulário, na qual seja informada a idade e, a cada informação, a função seja acionada.

```
<form> <input type=text size=2 maxlength=2 name="Tempo"      onchange="Idade(Tempo.value)"> </form>
```

Observe-se que o parâmetro passado (quando ocorre o evento "onchange") foi o conteúdo da caixa de texto "Tempo" (propriedade "value") e que, na função, chamamos de "Anos". Ou seja, não existe co-relação entre o nome da variável passada e a variável de recepção na função. Apenas o conteúdo é passado.

---

## *Funções Intrínsecas*

São funções embutidas na própria linguagem. A sintaxe geral é a seguinte:

Result = função (informação a ser processada)

- **eval** = Calcula o conteúdo da string
- **parseInt** - Transforma string em inteiro
- **parseFloat** - Transforma string em número com ponto flutuante
- **date()** - Retorna a data e a hora (veja o capítulo manipulando datas)

ex1: Result = eval ( " (10 \* 20) + 2 - 8" )

ex2: Result = eval (string)

No primeiro exemplo Result seria igual a 194. No segundo, depende do conteúdo da string, que também pode ser o conteúdo (value) de uma caixa de texto.

### - Funções tipicamente Matemáticas:

**Math.abs**(número) - retorna o valor absoluto do número (ponto flutuante)

**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, arredondado, do número

**Math.pow**(base, expoente) - retorna o cálculo do exponencial

**Math.max**(número1, número2) - retorna o maior número dos dois fornecidos

**Math.min**(número1, número2) - retorna o menor número dos dois fornecidos

**Math.sqrt**(número) - retorna a raiz quadrada do número

**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.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.pi** retorna o valor de PI (aproximadamente 3.14159)

**Math.log**(número) - retorna o logarítmo de um número

**Math.E** - retorna a base dos logarítmos naturais (aproximadamente 2.718)

**Math.LN2** - retorna o valor do logarítmo de 2 (aproximadamente 0.693)

**Math.LOG2E** - retorna a base do logarítmo de 2 (aproximadamente 1.442)

**Math.LN10** retorna o valor do logarítmo de 10 (aproximadamente 2.302)

**Math.LOG10E** - retorna a base do logarítmo de 10 (aproximadamente 0.434)

### Observação:

Em todas as funções, quando apresentamos a expressão "(número)", na verdade queremos nos referir a um argumento que será processado pela função e que poderá ser: um número, uma variável ou o conteúdo de um objeto (propriedade **value**).

## *Criando Novas Instâncias*

Através do operador **new** podem ser criadas novas instâncias a objetos já existentes, mudando o seu conteúdo, porém, mantendo suas propriedades.

A sintaxe geral é a seguinte:

```
NovoObjeto = new ObjetoExistente (parâmetros)
```

Ex1.

```
MinhaData = new Date ()
```

**MinhaData** passou a ser um objeto tipo **Date**, com o mesmo conteúdo existente em Date (data e hora atual)

Ex2:

```
MinhaData = new Date(1996, 05, 27)
```

**MinhaData** passou a ser um objeto tipo **Date**, porém, com o conteúdo de uma nova data.

Ex3:

Suponha a existência do seguinte objeto chamado **Empresas**

```
function Empresas (Emp, Nfunc, Prod)
    { this.Emp = Emp
      this.Nfunc = Nfunc
      this.Prod = Prod }
```

Podemos criar novas instâncias, usando a mesma estrutura, da seguinte forma:

```
Elogica = new Empresas("Elogica", "120", "Serviços")
Pitaco = new Empresas("Pitaco", "35", "Software")
Corisco = new Empresas("Corisco", "42", "Conectividade")
```

Assim, a variável **Elogica.Nfunc** terá o seu conteúdo igual a **120**

---

## *Manipulando Arrays*

O JavaScript não tem um tipo de dado ou objeto para manipular arrays. Por isso, para trabalhar com arrays é necessário a criação de um objeto com a propriedade de criação de um array.

No exemplo abaixo, criaremos um objeto tipo array de tamanho variável e com a função de "**limpar**" o conteúdo das variáveis cada vez que uma nova instância seja criada a partir dele.

```
function CriaArray (n) {
    this.length = n
    for (var i = 1 ; i <= n ; i++)
        { this[i] = "" } }
```

Agora podemos criar novas instâncias do objeto "CriaArray" e alimentá-los com os dados necessários.

```
NomeDia = new CriaArray(7)
NomeDia[0] = "Domingo"
NomeDia[1] = "Segunda"
NomeDia [2] = "Terça"
NomeDia[3] = "Quarta"
NomeDia[4] = "Quinta"
NomeDia[5] = "Sexta"
NomeDia[6] = "Sábado"
```

```
Atividade = new CriaArray(5)
Atividade[0] = "Analista"
Atividade[1] = "Programador"
Atividade[2] = "Operador"
Atividade[3] = "Conferente"
Atividade[4] = "Digitador"
```

Agora poderemos obter os dados diretamente dos arrays.

```
DiaSemana = NomeDia[4]
Ocupação = Atividade[1]
```

DiaSemana passaria a conter **Quinta** e Ocupação conteria **Programador**.

Outra forma de se trabalhar com arrays é criar novas instâncias dentro do próprio objeto do array, o que proporciona o mesmo efeito de se trabalhar com matriz. Isso pode ser feito da seguinte forma:

```
function Empresas (Emp, Nfunc, Prod) {
    this.Emp = Emp
    this.Nfunc = Nfunc
    this.Prod = Prod }
```

```

TabEmp = new Empresas(3)
TabEmp[1] = new Empresas("Elogica", "120", "Serviços")
TabEmp[2] = new Empresas("Pitaco", "35", "Software")
TabEmp[3] = new Empresas("Corisco", "42", "Conectividade")

```

Assim, poderemos obter a atividade da empresa número 3, cuja resposta seria **Conectividade**, da seguinte forma:

```
Atividade = TabEmp[3].Prod
```

Obs:

É importante lembrar que, embora os exemplos estejam com indexadores fixos, os indexadores podem ser referências ao conteúdo de variáveis.

## *Manipulando String's*

O JavaScript é bastante poderoso no manuseio de String's, fornecendo ao programador uma total flexibilidade em seu manuseio.

Abaixo apresentamos os métodos disponíveis para manuseio de string's.

**string.length** - retorna o tamanho da string (quantidade de bytes)

**string.charAt(posição)** - retorna o caracter da posição especificada (inicia em 0)

**string.indexOf("string")** - retorna o número da posição onde começa a primeira "string"

**string.lastIndexOf("string")** - retorna o número da posição onde começa a última "string"

**string.substring(index1, index2)** - retorna o conteúdo da string que corresponde ao intervalo especificado. Começando no caracter posicionado em index1 e terminando no caracter imediatamente anterior ao valor especificado em index2.

Ex.

```
Todo = "Elogica"
```

```
Parte = Todo.substring(1, 4)
```

(A variável **Parte** receberá a palavra **log**)

**string.toUpperCase()** - Transforma o conteúdo da string para maiúsculo (Caixa Alta)

**string.toLowerCase()** - Transforma o conteúdo da string para minúsculo (Caixa Baixa)

**escape** ("string") - retorna o valor ASCII da string (vem precedido de %)

**unescape**("string") - retorna o caracter a partir de um valor ASCII (precedido de %)

---

## *Manipulando Datas*

Existe apenas uma função para que se possa obter a data e a hora. É a função **Date()**. Esta função devolve data e hora no formato: Dia da semana, Nome do mês, Dia do mês, Hora:Minuto:Segundo e Ano

Ex.

**Fri May 24 16:58:02 1996**

Para se obter os dados separadamente, existem os seguintes métodos:

**getDate()** - Obtém o dia do mês (numérico de 1 a 31)

**getDay()** - Obtém o dia da semana (0 a 6)

**getMonth()** - Obtém o mês (numérico de 0 a 11)

**getFullYear()** - Obtém o ano

**getHours()** - Obtém a hora (numérico de 0 a 23)

**getMinutes()** - Obtém os minutos (numérico de 0 a 59)

**getSeconds()** - Obtém os segundos (numérico de 0 a 59)

No exemplo abaixo obteremos o dia da semana. Para tal, utilizaremos a variável **DataToda** para armazenar data/hora e a variável **DiaHoje** para armazenar o número do dia da semana.

```
DataToda = new Date()
```

```
DiaHoje = DataToda.getDay()
```

Para obter o dia da semana **alfa**, teremos que construir uma tabela com os dias da semana e utilizar a variável **DiaHoje** como indexador.

```
function CriaTab (n) {
  this.length = n
  for (var x = 1 ; x<= n ; x++)
    { this[x] = "" } }
```

```
NomeDia = new CriaTab(7)
```

```
NomeDia[0] = "Domingo"  
NomeDia[1] = "Segunda"  
NomeDia [2] = "Terça"  
NomeDia[3] = "Quarta"  
NomeDia[4] = "Quinta"  
NomeDia[5] = "Sexta"  
NomeDia[6] = "Sábado"
```

```
DiaSemana = NomeDia[DiaHoje]
```

Para criar uma variável tipo `Date` com o conteúdo informado pela aplicação, existe o método **set**. Assim, temos os seguintes métodos: **setDate**, **setDay**, **setMonth**, **setYear**, **setHours**, **setMinutes** e **setSeconds**.

Seguindo o exemplo acima, para mudar o mês para novembro, teríamos:

```
DataToda.setMonth(10)
```

Exemplos adicionais serão encontrados no capítulo "**Usando Timer e Date**".

---

## *Interagindo com o Usuário*

A interação com o usuário se dá através de objetos para entrada de dados (textos), marcação de opções (radio, checkbox e combo), botões e link's para outras páginas.

Conceitualmente, os objetos são divididos em: **Input**, **Textarea** e **Select**.

O objeto **Input** divide-se (propriedade `Type`) em:

- Password
- Text
- Hidden
- Checkbox
- Radio
- Button
- Reset
- Submit

A construção destes objetos é feita pela linguagem HTML (HiperText Mark-up Language). Portanto, é aconselhável que sejam criados utilizando-se ferramentas de geração de páginas HTML, como o HotDog ou, mais recomendado, FrontPage.

## ***Objeto Input TEXT***

É o principal objeto para entrada de dados.

Suas principais propriedades são: type, size, maxlength, name e value.

**type=text** : Especifica um campo para entrada de dados normal

**size** : Especifica o tamanho do campo na tela.

**maxlength** : Especifica a quantidade máxima de caracteres permitidos.

**name** : Especifica o nome do objeto

**value** : Armazena o conteúdo do campo.

Os eventos associados a este objeto são: onchange, onblur, onfocus e onselect.

Ex:

```
<form name="TText">  
<p>Entrada de Texto <input type=text size=20 maxlength=30 name="CxTexto"  
value="" onchange="alert ('Voce digitou ' + CxTexto.value)">  
</p>  
</form>
```

---

## ***Objeto Input PASSWORD***

É o objeto para entrada de Senhas de acesso (password). Os dados digitados neste objeto são criptografados e, só são interpretados (vistos) pelo "server", por razões de segurança.

Suas principais propriedades são: type, size, maxlength, name e value.

**type=password** : Especifica um campo para entrada de senha. Os dados digitados são substituídos (na tela) por "\*".

**size** : Especifica o tamanho do campo na tela.

**maxlength** : Especifica a quantidade máxima de caracteres permitidos.

**name** : Especifica o nome do objeto

**value** : Armazena o conteúdo digitado no campo.

Os eventos associados a este objeto são: onchange, onblur, onfocus e onselect.

Ex:

```
<form name="TPassword">  
<p>Entrada de Senha<input type=password size=10 maxlength=10 name="Senha" value="">  
</p>  
</form>
```

---

## ***Objeto Input HIDDEN***

É um objeto semelhante ao input text, porém, invisível para o usuário. Este objeto deve ser utilizado para passar informações ao "server" (quando o formulário for submetido) sem que o usuário tome conhecimento. Suas propriedades são: name e value.

**name** : Especifica o nome do objeto.

**value** : Armazena o conteúdo do objeto

Ex:

```
<form name="THidden">  
<input type=hidden size=20 maxlength=30 name="HdTexto" value="" >  
</form>  
</p>
```

---

## ***Objeto Input CHECKBOX***

São objetos que permitem ao usuário ligar ou desligar uma determinada opção.

Suas principais propriedades são: name, value e checked.

**name** : Especifica o nome do objeto

**value** : Especifica o valor que será enviado ao "server" se o objeto estiver ligado (checked).

Caso seja omitido, será enviado o valor default "on" .

Esta propriedade também serve para ativar comandos lógicos, testando-se a condição de "checked".

**checked** : Especifica que o objeto inicialmente estará ligado

O único evento associado a este objeto é onclick.

Ex:

No exemplo abaixo, criaremos um objeto input.text e três objetos checkbox. O primeiro checkbox,

quando ativado, transformará o texto em caracteres minúsculos. O segundo checkbox, quando ativado, transformará o texto em caracteres maiúsculos. O terceiro checkbox, quando ativado, dará um aviso do conteúdo que será recebido pelo "server" caso o formulário seja submetido para este.

```
<SCRIPT>
function AltMaiusc () {
    document.TCheck.Muda.value = document.TCheck.Muda.value.toUpperCase()    document.TCheck.
Opt1.checked = false
}
function AltMinusc () {
    document.TCheck.Muda.value = document.TCheck.Muda.value.toLowerCase()    document.TCheck.
Opt2.checked = false
}
</SCRIPT>
<p>
<form name="TCheck">
Muda Case <input type=text size=20 maxlength=20 name="Muda"> </p>
<p>
Minusc<input type=checkbox name="Opt1" value="1" checked
    onclick="if (this.checked)
        { AltMinusc() } ">
Maiusc<input type=checkbox name="Opt2" value="2"
    onclick="if (this.checked)
        { AltMaiusc() } ">
Demo valor<input type=checkbox name="Opt3"
    onclick="if (Opt3.checked)
        { alert ('Server recebera = ' + Opt3.value) } ">
</p>
</form>
```

Existe ainda uma outra forma de manipular este objeto, em forma de array, que é a seguinte: **form.elements[index].propriedade**. Esta não é uma boa forma porque o index é único dentro de um formulário, exigindo muito cuidado quando se acrescenta ou se deleta um objeto, pois, neste caso, haverá um natural deslocamento do index, podendo comprometer a lógica.

## *Objeto Input RADIO*

São objetos que permitem ao usuário a escolha de apenas uma alternativa, diante de uma série de opções.

Suas principais propriedades são: name, value e checked.

**name** : Especifica o nome do objeto. Para caracterizar uma mesma série de opções, todos os objetos desta série têm que ter o mesmo "name".

**value** : Especifica o valor que será enviado ao "server" se o objeto estiver ligado (checked). Caso seja omitido, será enviado o valor default "on" . Esta propriedade também serve para ativar comandos lógicos, testando-se a condição de "checked".

**checked** : Especifica que o objeto inicialmente estará ligado

Para utilização deste objeto é importante o conhecimento de outras propriedades associadas:

Objeto.**length** : Retorna a quantidade de opções existentes na lista

Objeto.[index].value : retorna o texto (value) associado a cada opção

Objeto.[index].checked : retorna verdadeiro ou falso

O único evento associado a este objeto é onclick.

Ex. No exemplo abaixo temos dois set's de objetos radio. O primeiro tem o objetivo de mudar a cor de fundo do documento atual. O segundo tem o objetivo levar informações ao "server".

<p>Radio</p>

```
<p>
  <input type=radio name="Rad" value="1"
    onclick="document.bgColor='green'"> Fundo Verde
  <input type=radio name="Rad" value="2"
    onclick="document.bgColor='blueviolet'"> Fundo Violeta
  <input type=radio name="Rad" value="3"
    onclick="document.bgColor='#FFFF00'"> Fundo Amarelo
```

</p>

```
<p>
  <input type=radio name="Rad2" value="1"> Solteiro
  <input type=radio name="Rad2" value="2"> Casado
  <input type=radio checked name="Rad2" value="3"> Tico Tico
```

</p>

---

## ***Objeto Input BUTTON***

Este objeto tem por finalidade criar um botão ao qual se possa atrelar operações lógicas, a serem executadas quando o mesmo receber um click.

Suas propriedades são: name e value.

**name** : Especifica o nome do objeto.

**value** : Especifica o nome que aparecerá sobre o botão

O único evento associado a este objeto é onclick.

Ex.

```
<p>
<form method="POST" name="TstButton">
Digite um Texto <input type=text size=30 maxlength=30 name="Teste" value="">
</p><p>
Click no Botao <input type=button name="Bteste" value="Botao de teste"
onclick="alert ('Voce digitou: ' + TstButton.Teste.value)">
</p>
</form>
```

---

## ***Objeto Input RESET***

Este objeto é um botão que tem por finalidade, única, limpar os campos digitados pelo usuário, restaurando o conteúdo do formulário para os valores iniciais.

É recomendável a utilização deste objeto, para facilitar o usuário a limpar suas informações, uma vez que a utilização da opção "reload" do "browser" (que seria uma forma) não perde as informações digitadas.

Suas propriedades são: name e value.

**name** : Especifica o nome do objeto.

**value** : Especifica o nome que aparecerá sobre o botão

O único evento associado a este objeto é onclick.

Ex.

```
<p>
<form method="POST" name="TesteRes">
Digite um Texto<input type=text size=10 maxlength=20 name="Teste" value="">
Apague o Texto <input type=reset name="Bres" value="Reset">
</form>
</p>
```

---

## Objeto *Input SUBMIT*

Este objeto é um botão que tem por finalidade submeter (enviar) o conteúdo dos objetos do formulário ao "server". O formulário será submetido à URL especificada na propriedade "action" do formulário. Suas propriedades são: name e value.

**name** : Especifica o nome do objeto.

**value** : Especifica o nome que aparecerá sobre o botão

O único evento associado a este objeto é onclick. Embora se possa atrelar lógica a este evento, não se pode evitar que o formulário seja submetido, portanto, não é aconselhável o seu uso. Mais seguro e mais útil é a utilização da propriedade onSubmit do formulário. Este permite que se atrele lógica e decida-se pela submissão ou não.

Ex.

```
<script>
function TestaVal() {
if (document.TesteSub.Teste.value == "") {
    alert ("Campo nao Preenchido...Form nao Submetido")
    return false }
else {
    alert ("Tudo Ok....Form Submetido")
    return true } }
</script>
<p>
<form method="POST" name="TesteSub"
    onSubmit="return TestaVal()"
    action="http://10.0.5.2/scripts/isapielo.dll/vbloja.loja.action">
Digite um Texto <input type=text size=10 maxlength=10 name="Teste" value="">
Botao Submit <input type=submit name="Bsub" value="Manda p/Server">
</p>
</form>
```

No exemplo acima, o formulário está sendo submetido a URL "10.0.5.2" (que é o endereço IP de um "Server"). Este servidor está rodando o "Microsoft Internet Information Server". Estamos enviando os dados a um "OLE", que está no subdiretório "scripts", chamado "isapielo.dll", que tem por objetivo fazer a conexão com aplicações escritas em VB. A aplicação VB que está sendo chamada, é um OLE de nome "vbloja" no qual estamos acionando a classe "loja" e o método "action".

A aplicação VB, deste exemplo, fará apenas a devolução dos dados recebidos pelo Server.

---

## ***Objeto TEXTAREA***

É um objeto para entrada de dados em um campo de múltiplas linhas. Suas principais propriedades são: name, rows e cols.

**name** : Especifica o nome do objeto

**rows** : Especifica a quantidade de linhas que aparecerão na tela

**cols** : Especifica a quantidade de caracteres que aparecerão em cada linha

**value** : Acessa o conteúdo do campo via programação.

Os eventos associados a este objeto são: onchange, onblur, onfocus e onselect.

Ex:

```
<form name="TesteTextarea">
```

```
<p>
```

```
Texto de Múltiplas Linhas <textarea name="MultText" rows=2 cols=40>
```

```
Primeira linha do texto inicial
```

```
segunda linha do texto inicial
```

```
</textarea>
```

```
</p>
```

---

## ***Objeto SELECT***

É um objeto para entrada de opções, onde o usuário, a partir de uma lista de alternativas, seleciona uma ou mais opções.

Suas principais propriedades são: name, size, value e multiple.

**name** : Especifica o nome do objeto

**size** : Especifica a quantidade de opções que aparecerão na tela simultaneamente

**value** : Associa um valor ou string para cada opção (opcional)

**multiple** : Especifica a condição de múltipla escolha (usando-se a tecla Ctrl)

Para utilização deste objeto é importante o conhecimento de outras propriedades associadas:

**Objeto.length** : Retorna a quantidade de opções existentes na lista

**Objeto.selectedindex** : Retorna o "index" do objeto selecionado (primeiro = 0) **Objeto.options[index].text** : retorna o texto externo associado a cada opção **Objeto.options[index].value** : retorna o texto interno (value) associado a cada opção **Objeto.options[index].selected** : retorna verdadeiro ou falso

Os eventos associados a este objeto são: onchange, onblur e onfocus.

Ex1:

Neste exemplo é importante observar os seguintes aspectos:

- A lista permite apenas uma seleção
- A quarta opção aparecerá inicialmente selecionada (propriedade "selected")
- Não utilizamos a propriedade "value". Assim, a propriedade "text" e a propriedade "value" passam a ter o mesmo valor, ou seja, o valor externo que aparece na tela.

```
.
<script>
function Verselect(Campo) {
Icombo = Campo.selectedIndex
alert ("Voce escolheu " + Campo.options[Icombo].text) }
</script>
<p>
Objeto Select <select name="Combo1" size=1 onchange="Verselect(Combo1)">
<option>Opcao 1
<option>Opcao 2
<option>Opcao 3
<option selected>Opcao 4 (recomendada)
<option>Opcao 5
<option>Opcao 6
</select>
</p>
```

Ex2:

Neste exemplo é importante observar os seguintes aspectos:

- A lista permite múltiplas seleções
- Utilizamos a propriedade "value". Assim as propriedades "text" e "value" têm valores diferentes: text retornará Escolha 1 a Escolha 4 e value retornará List1 a List4.
- O parâmetro passado, quando da ocorrência do evento onblur, foi **this**. Esta diretiva significa que estamos passando este objeto.

```
.
<script>
function Vermult(Lista) {
var opcoes = ""
for (i = 0 ; i < Lista.length ; i++) {
if (Lista.options[i].selected) {
```

```

opcoes += (Lista.options[i].value + ", ") }
}
alert ("As opcoes escolhidas foram : " + opcoes) }
</script>
<p>
Objeto Select2 <select name="Combo2" size=4 multiple onblur="Vermult(this)">
<option value="List1">Escolha 1 </option>
<option value="List2">Escolha 2 </option>
<option value="List3">Escolha 3 </option>
<option value="List4">Escolha 4 </option>
</select>
</p>

```

---

## ***Focando um Objeto***

Este método permite que o cursor seja ativado em um determinado objeto (focado). Isso pode ser feito na carga do documento, a partir da ocorrência de um evento ou mesmo dentro de uma função. Observe que até agora o usuário tinha que dar um "Click" para focar o objeto desejado.

De forma semelhante existe o método "Select". Este método marca o conteúdo do objeto com uma tarja roxa, permitindo ao usuário, em caso de substituição do conteúdo do campo, não ter que deletar o conteúdo anterior, pois, com este método, a deleção se dá de forma automática quando da digitação do novo conteúdo.

Os métodos "Focus" e "Select" podem ser utilizados nos seguintes objetos:

### **password, select, text e textarea**

No exemplo abaixo, utilizaremos o evento **onload** para setar o **focus** para o primeiro objeto do formulário e os métodos **focus** e **select** para, na rotina de crítica dos dados, focar o objeto que contiver erro de preenchimento.

```

Ex.
<body onload="document.TstFocus.Nome.focus()">
<script>
DdosOk = true
function Criticar() {
  DadosOk = false
  DataAtual = new Date()
  MesAtual = DataAtual.getMonth() + 1

```

```

AnoAtual = DataAtual.getYear() + 1900
Nome = document.TstFocus.Nome.value
Mes = parseInt(document.TstFocus.Mes.value)
Ano = parseInt (document.TstFocus.Ano.value)
//
if (Ano < 1900)
  {Ano = Ano + 1900 }
if (Nome == "")
  { alert ("Informe o seu Nome, Não deixe em branco")
  document.TstFocus.Nome.focus()
  return }
if (Mes < 1 || Mes > 12)
  { alert ("O Mês informado não é válido, informe corretamente")    document.TstFocus.Mes.focus()
  document.TstFocus.Mes.select()
  return }
if (Ano == AnoAtual && Mes > MesAtual)
  { alert ("O período informado é superior a data atual")
  document.TstFocus.Mes.focus()
  document.TstFocus.Mes.select()
  return }
if (Ano < 1996 || Ano > AnoAtual)
  { alert ("O Ano informado não é válido, informe corretamente")    document.TstFocus.Ano.focus()
  document.TstFocus.Ano.select()
  return }
DadosOk = true
}
</script>

```

```

<form name="TstFocus" method="POST">
<p>
Informe o seu Nome <input type="text" size=30 maxlength=30 name="Nome">
</p>
<p> Informe o mês desejado <input type="text" size=2 maxlength=2 name="Mes">
</p>
<p> Informe o ano desejado <input type="text" size=4 maxlength=4 name="Ano" >
</p>
<p> <input type="button" name="Testa" value="Testar Validade"
      onclick="Criticar()
              if (DadosOk)
                {alert ('Todos os Dados estão Corretos') } ">
</p>
</form>
</body>

```

## *Usando Timer e Date*

É um método que permite a programação para que uma determinada ação só ocorra após o transcurso de um determinado tempo.

Variável = **setTimeout** ("ação", tempo)

Onde:

**Variável** é uma variável apenas para controle do timer

**ação** é a ação que se quer realizar.

**tempo** é o tempo de espera para que a ação ocorra, em milisegundos.

Obs:

É importante observar que a ação só ocorrerá uma vez. Para que a ação volte a ocorrer, será necessário repetir o comando dentro da ação, obtendo-se, assim, um LOOP.

Para interromper um LOOP, provocado pela forma acima, deve-se utilizar o seguinte método:

**clearTimeout** (Variável)

Onde:

**Variável** é o nome da variável de controle do timer.

Abaixo encontra-se um exemplo de um formulário que apresenta a data e hora atual, atualizando os dados a cada um segundo, tendo dois botões de rádio que tem a função de ativar e desativar a atualização dos dados. Apresenta também, fora do formulário, a data contendo dia e mês por extenso.

```
.  
<script>  
function Hoje() {  
  ContrRelogio = setTimeout ("Hoje()", 1000)  
  Hr = new Date()  
  dd = Hr.getDate()  
  mm = Hr.getMonth() + 1  
  aa = Hr.getYear()  
  hh = Hr.getHours()  
  min = Hr.getMinutes()  
  seg = Hr.getSeconds()  
  DataAtual = ((dd < 10) ? "0" + dd + "/" : dd + "/")
```

```

DataAtual += ((mm < 10) ? "0" + mm + "/" + aa : mm + "/" + aa)
HoraAtual = ((hh < 10) ? "0" + hh + ":" : hh + ":")
HoraAtual += ((min < 10) ? "0" + min + ":" : min + ":")
HoraAtual += ((seg < 10) ? "0" + seg : seg)
document.DataHora.Data.value=DataAtual
document.DataHora.Hora.value=HoraAtual
}
//
function CriaArray (n) {
this.length = n }
//
NomeDia = new CriaArray(7)
NomeDia[0] = "Domingo"
NomeDia[1] = "Segunda"
NomeDia[2] = "Terça"
NomeDia[3] = "Quarta"
NomeDia[4] = "Quinta"
NomeDia[5] = "Sexta"
NomeDia[6] = "Sábado"
//
NomeMes = new CriaArray(12)
NomeMes[0] = "Janeiro"
NomeMes[1] = "Fevereiro"
NomeMes[2] = "Março"
NomeMes[3] = "Abril"
NomeMes[4] = "Maio"
NomeMes[5] = "Junho"
NomeMes[6] = "Julho"
NomeMes[7] = "Agosto"
NomeMes[8] = "Setembro"
NomeMes[9] = "Outubro"
NomeMes[10] = "Novembro"
NomeMes[11] = "Dezembro"
//
Data1 = new Date()
dia = Data1.getDate()
dias = Data1.getDay()
mes = Data1.getMonth()
ano = Data1.getYear()
document.write ("Recife, " + NomeDia[dias] + " " + dia + " de " +
NomeMes[mes] + " de " + (ano + 1900) )
</script>
<form name="DataHora">

```

```
Data : <input type=text size=10 maxlength=10 name="Data">
Hora : <input type=text size=10 maxlength=10 name="Hora">
      <input type=radio name="Botao" value="Para Relogio" checked
        onclick="clearTimeout(ContrRelogio)">Desativa
      <input type=radio name="Botao" value="Ativa Relogio"
        onclick="ContrRelogio = setTimeout('Hoje()', 1000)">Ativa
</form>
```

## *Escrevendo Textos Rolantes*

Observe o rodapé desta página.... existe uma mensagem que está sempre rolando. Na verdade, o texto não está rolando como dá a impressão. O que ocorre é a troca do conteúdo a cada um segundo. Para fazer isto escrevemos uma função que, a cada segundo, desloca o texto em um caracter. Ou seja: elimina o primeiro caracter, deslocando o texto, e insere um novo caracter no final do texto. Quando o novo texto é exibido obtemos o efeito apresentado.

No exemplo abaixo apresentamos o código necessário para criar textos rolantes. Os textos serão apresentados na área de status (rodapé) da página bem como em uma caixa de texto de um formulário. Inicialmente definimos as variáveis a serem utilizadas pela função e preenchemos com "brancos" a variável que conterà o texto a ser exibido, para que o texto comece a ser exibido a partir do canto direito da caixa.

No momento em que o documento é carregado, utilizamos o evento **onload** para ativar a função que fará a apresentação dos textos.

Ex:

```
<html>
<script>
/* ----- Apresenta uma caixa de texto rolante no rodapé da página
*/
var Texto = "Grupo Elógica - Manual Java Script - Principais Comandos....."
var Tamanho = Texto.length
var Trab = Texto.length
var Trab1=0
var Temp = ""
var Mensagem=""
var CtrTime=""
for (x=0 ; x <= 200 ; x++) {
    Mensagem += " " }
//
function Mostra() {
```

```

if (Trab <= Tamanho && Trab > 0) {
    Temp = Mensagem.substring(1, Mensagem.length)
    Mensagem = Temp + Texto.substring(Trab1, Trab1 + 1)
    Trab--
    Trab1++ }
else {
    Trab = Texto.length
    Trab1 = 0 }
window.status = Mensagem
document.Form1.Letreiro.value = Mensagem.substring(150, Mensagem.length)
CtrTime = setTimeout("Mostra()", 100)
}
</script>
<body onload="CtrTime = setTimeout('Mostra()', 100)">
<form method="POST" name=Form1>
<p><font color="#008000"><font size=4>
Apresentação de caixa de texto rolante</font></font> </p>
<p><input type=text size=50 maxlength=100 name="Letreiro"> </p>
</form>
</body>
</html>

```

## *Abrindo Novas Janelas*

Neste capítulo mostraremos como podem ser abertas novas janelas sobre uma janela contendo o documento principal.

É importante não confundir esta forma de abrir janelas com a divisão da tela em várias partes, ou mesmo com a chamada de outras páginas. Para que não existam dúvidas, explicaremos um pouco sobre estes dois outros métodos.

A divisão de uma tela em várias janelas contendo documentos diferentes é feita através do objeto **FRAME** do Html. Neste caso, a tela inteira é considerada como um **FrameSet** e cada parte em que ela for dividida é considerada como um **Frame**. Cada **Frame** é definido dentro do **FrameSet** através da especificação dos parâmetros: % da tela na vertical (cols), % da tela na horizontal (rows) e nome de cada frame. Uma vez criado o **FrameSet** poderemos abrir documentos distintos em cada **Frame**. Para fazer isto, acrescente ao **link** do documento a diretiva **target=nome do frame**.

Ex.

```
<href="Eventos.htm" target="Principal">
```

Isto fará com que o arquivo html **Eventos.htm** seja aberto dentro do frame de nome **Principal**

A simples chamada de outras telas (documentos) é feita através do **link** para o documento desejado.

Ex.

```
<href="Eventos.htm" >
```

Isto fará com que o arquivo html **Eventos.htm** seja aberto em substituição a tela existente.

Bem, voltemos ao nosso caso que é a abertura de janelas sobre um documento. Isto é feito através de comandos JavaScript, que permitem: Abrir uma janela, Abrir um documento dentro desta janela, Escrever o conteúdo da janela, Fechar a janela e Fechar o documento.

## Abrindo a Janela

A sintaxe geral deste método é a seguinte:

```
Variavel = window.open ("Url", "Nome da janela", "Opções")
```

Onde:

Variavel - Nome que será atribuído como propriedade da janela.

Url - Endereço **Internet** onde a janela será aberta. Normalmente voce estará utilizando a sua própria Url, neste caso, preencha com "".

Nome da Janela - É o nome que aparecerá no top da janela (Título)

Opções - São as opções que definem as características da janela, quais sejam:

- **toolbar** - Cria uma barra de ferramentas tipo "Back", "Forward", etc.
- **location** - Abre a barra de **location** do browse
- **directories** - Abre a barra de ferramentas tipo "What's New", "Handbook", etc.
- **status** - Abre uma barra de status no rodapé da janela
- **scrollbar** - Abre barras de rolamento vertical e horizontal
- **menubar** - Cria uma barra de menu tipo "File", "Edit", etc.
- **resizable** - Permite ao usuário redimensionar a janela
- **width** - Especifica a largura da janela, em pixels
- **height** - Especifica a altura da janela, em pixels

Todas as opções (exceto **width** e **height**) são booleanas e podem ser setadas de duas formas. Exemplo: "toolbar" ou "toolbar=1") são a mesma coisa. Se nada for especificado, entende-se que todas as opções estão ligadas; Caso seja especificada qualquer opção, será entendido que estão ligadas apenas as opções informadas.

As opções devem ser informadas separadas por vírgula, sem espaço entre elas.

## Abrindo um Documento

Para abrir um documento dentro da janela, deve ser utilizado o seguinte método:

## Variavel.**document.open()**

Onde "Variavel" é o nome da variável associada ao método **window.open**

### Escrevendo no Documento

Para escrever a tela no documento, deve ser utilizado o seguinte método:

Variavel.**document.write** ("Comandos html, Comandos JavaScript, Textos, etc.")

### Fechando a Janela

Para fechar a janela, utilize o seguinte método:

Variavel.**document.write** ("**window.close()**")

### Fechando o Documento

Para fechar o documento, utilize o seguinte método:

Variavel.**document.close** ()

A seguir, apresentamos um exemplo no qual estamos abrindo um documento onde o usuário escolherá uma opção (Elógica ou Recife) e dará um Click em um botão (Nova Janela). Neste momento será aberta uma nova janela que conterá a foto escolhida pelo usuário e um botão que, ao receber o Click, fechará a janela.

Normalmente, qualquer **href** ou **src** dentro de uma página, por padrão, acessa o arquivo ou a imagem no mesmo diretório onde está a página atual, a menos que seja especificado um novo caminho (Path).

No caso de abertura de uma nova janela, através do método **window.open**, as versões mais antigas dos browses não conseguem "ver" o **Path**, sendo necessária a completa informação do caminho (path) onde o arquivo ou imagem estão armazenados, em todas as chamadas dos comandos Html **href** ou **src**.

Observe que na função estamos utilizando dois novos métodos:

**navigator.appVersion** para verificarmos a versão do browse que esta sendo utilizado **document.location**. para obtermos o **Path** da localização do arquivo Html que está correntemente em uso.

No exemplo abaixo estamos, inicialmente, identificando a versão do browse. Caso seja antiga, para não escrevermos todo o caminho a cada chamada e ainda, considerando que os arquivos chamados estão no mesmo diretório da página atual, estamos obtendo o **Path** do arquivo atual e eliminando o nome do arquivo que está na última referencia do **Path**. Quando fizermos a chamada das imagens (comando **src**) só será necessário a concatenação do nome do arquivo chamado com a raiz do path que, no exemplo, armazenamos na variável de nome **Local**.

```
<script>
function Abrejanela(Opcao) {
```

```

Versao = navigator.appVersion
Versao = Versao.substring(0, 1)
Local = ""
if (Versao < 3) {
    Local = document.location
    UltLoc = Local.lastIndexOf("/")
    Local = Local.substring(0, UltLoc + 1)
}
//
NovaJanela = window.open ("", "OutraJanela", "width=300,height=400")    NovaJanela.document.
open()
NovaJanela.document.write ("<html><head><title>Nova Janela")
NovaJanela.document.write ("</title></head><body bgcolor='white'>")    NovaJanela.document.
write ("<form>")
if (Opcao == 1)
{ NovaJanela.document.write ("<br>Logomarca Elogica<hr><br>")
  NovaJanela.document.write
    ("") }
else
{ NovaJanela.document.write ("<br>Recife Alto Astral<hr><br>")
  NovaJanela.document.write
    ("") }
//
NovaJanela.document.write ("<br><hr><p></p></form>")
NovaJanela.document.write ("<form><input type='button' name='Fecha'" +
    "value='Fecha Janela'" + "onclick='window.close()'>")
NovaJanela.document.write ("</form></body></html>")
NovaJanela.document.close() }
</script>

<body>
<p></p>
<p>Escolha a foto a ser apresentada na nova janela:</p>
<form method="POST" name="Form1">
<p>
<input type=radio name="Opcao" value="1" checked>Elogica
<input type=radio name="Opcao" value="2">Recife
</p> <p>
<input type="button" name="Envia" value="Nova Janela"
    onclick="if (Form1.Opcao[0].checked == true)
        { Abrejanela(Form1.Opcao[0].value) }
    else
        { Abrejanela(Form1.Opcao[1].value) } ">

```

</p>

</form>

</body>