Apostila de JavaScript

(Curso Básico)

Versão 0.01

09 de abril de 2005

Prof. Luís Rodrigo de O. Gonçalves

E-mail:luisrodrigoog@yahoo.com.br

Site: http://www.lrodrigo.cjb.net

OBS: esta apostila foi montada com trechos retirados de vários sites da internet.

Índice

1.) Introdução

JavaScript é uma linguagem para auxilio na criação de Home-Pages, as funções escritas em JavaScript podem ser embutidas dentro de seu documento HTML, possibilitando o incremento das funcionalidades do seu documento HTML com elementos interessantes. Sendo possivel: responder facilmente a eventos iniciados pelo usuário, incluir efeitos que tornem sua página dinâmica. Logo, podemos criar sofisticadas páginas com a ajuda desta linguagem.

Apesar dos nomes bem parecidos, Java não é o mesmo que JavaScript. Estas são duas técnicas diferentes de programação na Internet. Java é uma linguagem de programação. JavaScript é uma linguagem de hiper-texto. A diferença é que você realmente pode criar programas em Java. Mas muitas vezes você precisa apenas criar um efeito bonito sem ter que se incomodar com programação. A solução então é JavaScript pois é fácil de entender e usar.

Podemos dizer que JavaScript é mais uma extensão do HTML do que uma linguagem de programação propriamente dita. O primeiro browser a suportar JavaScript foi o Netscape Navigator 2.0, mas a versão para o "Mac" parece apresentar muitos bugs.

1.1) Considerações iniciais

Em documentos HTML, a utilização da linguagem JavaScript, se dá sob a forma de funções (applets), as quais são chamadas em determinadas situações ou em resposta a determinados eventos, estas funções podem estar localizadas em qualquer parte do código HTML, a única restrição é que devem começar com a declaração <SCRIPT> e termina com o respectivo </SCRIPT>, por convenção costuma-se colocar todas as funções no início do documento (estre as TAGs <HEAD> e </HEAD>, isso para garantir que o código JavaScript seja carregado antes que o usuário interaja com a Home Page), ou seja, antes do <BODY>.

Exemplo:

```
<HTML>
  <HEAD>
    <TITLE>Exemplo</TITLE>
          <!--
            Se houvesse alguma função seria bom declará-la aqui!!!
          -->
    </HEAD>
    <BODY>
           Esta linha está escrita em HTML
          <SCRIPT>
```

```
                document.write("Aqui já é JavaScript");
            </SCRIPT>
              Voltamos para o HTML
        </BODY>
</HTML>
```

É importante ressaltar que todas as linhas devem ser terminadas com; (ponto e virgula) a menos que a próxima instrução seja um “else” e se você precisar escrever mais de uma linha para executar uma condição seja ela em uma estrutura “for”, “if” ou “while”, este bloco de instruções deve estar entre “{ }” (chaves). Inclusive a definição de funções segue este modelo, ou seja, todo o código da função deve estar limitado por { (no início) e } (no final).

Um browser que não suporta JavaScript, ele não conhece a TAG <SCRIPT>. Ele ignora a TAG e logicamente tudo todo o código que estiver sendo limitado por ela, mostrando todo o código na tela como se fosse um simples texto HTML. Deste modo o usuário veria o código JavaScript do seu programa dentro do documento HTML e como certamente essa não deve ser sua intenção, existe um meio de esconder o código JavaScript dos browsers que não conhecem esta linguagem, basta utilizar os comentários HTML ***“<!- -”*** e ***“-->”***. O código do nosso exemplo anterior ficaria assim:

```
<HTML>
  <HEAD>
      <TITLE> Exemplo </TITLE>
          <!--
              ...
              Se houvesse alguma função seria bom declará-la aqui!!!
              ...
           -->
   </HEAD>
   <BODY>
       Esta linha está escrita em HTML
      <SCRIPT>
         <!-- Esconde o código JavaScript dos browsers mais antigos
            document.write("Aqui já é JavaScript");
         // -->
      </SCRIPT>
      Voltamos para o HTML
    </BODY>
</HTML>
```

Se o browser não suportar JavaScript e não inserirmos o comentário HTML, o que apareceria na tela seria:

```
Esta é uma linha escrita em HTML
document.write("Aqui já é JavaScript");
Voltamos para o HTML
```

Note que esse artifício não esconde completamente o código JavaScript, o que ele faz é prevenir que o código seja mostrado por browsers mais antigos, porém o usuário tem acesso a todas as informações do código fonte de sua Home Page (tanto HTML, quanto JavaScript).

1.2) Variáveis

Em JavaScript, variáveis dinâmicas podem ser criadas e inicializadas sem declarações formais. Existem dois tipos de abrangência para as variáveis:

- ***"Global"*** - Declaradas/criadas fora de uma função. As variáveis globais podem ser acessadas em qualquer parte do programa.
- ***"Local"*** - Declaradas/criadas dentro de uma função. Só podem ser utilizadas dentro da função onde foram criadas e precisa ser definida com a instrução Var.

Com relação à nomenclatura, as variáveis devem começar por uma letra ou pelo caractere sublinhado **"_"**, o restante da definição do nome pode conter qualquer letra ou número.

É importante ressaltar que a variável ***"Código"*** é diferente da variável ***"código"***, que por sua vez é diferente de ***"CODIGO"***, sendo assim, muito cuidado quando for definir o nome das variáveis, utilize sempre um mesmo padrão.

Existem três tipos de variáveis: ***Numéricas***, ***Booleanas*** e ***Strings***.

Como já era de se esperar, as variáveis numéricas são assim chamadas, pois armazenam números, as Booleanas valores lógicos (True/False) e as Strings, seqüência de caracteres. As strings podem ser delimitadas por aspas simples ou duplas, a única restrição é que se a delimitação começar com as aspas simples, deve terminar com aspas simples, da mesma forma para as aspas duplas. Podem ser incluídos dentro de uma string alguns caracteres especiais, a saber:

\t - posiciona o texto a seguir, na próxima tabulação;

\n - passa para outra linha;

\f - form feed;

\b - back space;

\r - carrige return.

O JavaScript reconhece ainda um outro tipo de contudo em variáveis, que é o ***NULL***. Na prática isso é utilizado para a manipulação de variáveis não inicializadas sem que ocorra um erro no seu programa. Quando uma variável possui o valor ***NULL***, significa dizer que ela possui um valor desconhecido ou nulo. A representação literal para ***NULL*** é a string ***'null'*** sem os delimitadores. Quando referenciado por uma função ou comando de tela, será assim que ***NULL*** será representado. Observe que ***NULL*** é uma palavra reservada.

Você pode trabalhar ainda com Arrays, mas para isso será necessário algum conhecimento sobre Programação Orientada a Objetos.

1.3) Operadores

Junto com funções e variáveis, operadores são blocos de construção de expressões. Um operador é semelhante a uma função no sentido de que executa uma operação específica e retorna um valor.

1.3.1) Operadores unários e binários

Todos os operadores em JavaScript requerem um ou dois argumentos, chamados operandos. Aqueles que requerem um operando apenas são denominados operadores ulnários, e os que requerem dois operandos são chamados de operadores binários.

1.3.2) Operador de String

Operador de concatenação (**+**)

Exemplo:

```
Nome1="José"
Nome2="Silva"
Nome = Nome1+" da "+Nome2 // O resultado é: "José da Silva"
```

1.3.3) Operadores Matemáticos

Adição (+)

Exemplo:

```
V01=5
V02=2
V=V01+V02 // resulta em: 7
```

Subtração (-)

Exemplo:

```
V01=5
V02=2
V=V01-V02 // resulta em: 3
```

Multiplicação (*)

Exemplo:

```
V01=5
V02=2
V=V01*V02 // resulta em: 10
```

Divisão (/)

Exemplo:

```
V01=5
V02=2
V=V01/V02 // resulta em: 2.5
```

Módulo da divisão ou resto (%)

Exemplo:

```
V01=5
V02=2
V=V01%V02 // resulta em: 1
```

Incremento (++)

Pode acontecer de duas formas:

++Variável ou

Variável++

Exemplo:

```
V01 = 5
V02 = ++V01 // Resulta em 6
V03 = V01 // Resulta em 6
```

Exemplo:

```
V01 = 5
V02 = V01++ // Resulta em 5
V03 = V01 // Resulta em 6
```

Decremento (--):

Pode acontecer de duas formas:

--Variável ou

Variável--

Exemplo:

```
V01 = 5
V02 = --V01 // Resulta em 4
V03 = V01 // Resulta em 4
```

Exemplo:

```
V01 = 5
V02 = V01-- // Resulta em 5
V03 = V01 // Resulta em 4
```

1.3.4) Operadores relacionais

< Menor que

> Maior que

= = Igual

!= Diferente

>= Maior ou igual

<= Menor ou igual

1.3.5) Operadores lógicos

&& E lógico

|| Ou lógico

1.3.6) Operadores de atribuição

= Atribuir

+= Soma ou concatenação e atribuição: x+=5 // é o mesmo que: x=x+5

-= Subtração e atribuição. x-=5 // é o mesmo que: x=x-5

= Multiplicação e atribuição. x=5 // é o mesmo que: x=x*5

/= Divisão e atribuição. x/=5 // é o mesmo que: x=x/5

%= Módulo e atribuição. x%=5 // é o mesmo que: x=x%5

2) Objetos

Quando compramos um televisor, recebemos um manual, que por mais simples que possa ser, traz sempre algumas especificações técnicas do aparelho. Por exemplo: Polegadas da tela, voltagem de trabalho, entre outras. Essas especificações técnicas transferido para o vocabulário da OOP são as propriedades do objeto (televisor). Em JavaScript essas propriedades nada mais são do que variáveis internas do objeto.

Um objeto está sujeito a determinados métodos. Um método geralmente é uma função que gera alguma informação referente ao objeto. Por exemplo ao mudar de canal, nós estamos executando uma função do televisor, o mesmo ocorre quando aumentamos ou diminuímos o volume.

Seguindo nosso exemplo, quando a tensão da rede sai da faixa de trabalho no caso de uma queda de tensão ou uma sobrecarga, o sistema de segurança da Tv, não permite que ocorram danos no aparelho, quando muito, queima o fusível da fonte de alimentação. Em aparelhos mais modernos, quando uma emissora sai do ar, a tela fica azul, sem aquele chiado irritante. Sendo assim podemos concluir que nosso objeto está sujeito a algumas situações, estas situações podem ocorrer a qualquer momento, e são chamadas de eventos.

2.1) Criando Objetos

Trabalhar com objetos é a única forma de manipular os arrays, vejamos como: Digamos que queremos implementar uma lista de clientes, nosso objeto poderia ser definido assim:

```
Function CriaClientes(nome,endereco,telefone,renda)
{
        this.nome=nome;
        this.endereco=endereco;
        this.telefone=telefone;
            this.renda=renda;
}
```

A propriedade ***"this"*** especifica o objeto atual como sendo fonte dos valores passados a função. Agora, basta criar o nosso objeto:

```
        Maria = New CriaClientes('Maria Aparecida','Rua Guilhotina dos Patos, S/N','332-1148',1300)
```

Para acessar as propriedades do objeto Maria, basta usar a seguinte sintaxe:

```
                Maria.nome - retorna 'Maria Aparecida'
                Maria.endereco - retorna 'Rua Guilhotina dos Patos, S/N'
                Maria.telefone - retorna '332-1148'
```

```
Maria.renda - retorna 1300
```

Uma outra forma de referenciar as propriedades do objeto Maria, é:

```
Maria[0], Maria[1], Maria[2], Maria[3]
```

Uma forma mais prática de criar arrays poderia ser a seguinte:

```
Function Matriz(n)
{
        this.length=n
        for (var contador=1 ; contador <=n ; conatdor=contador+1)
        {
                this[contador]=""
        }
}
```

Para criar nossa matriz usaríamos o seguinte comando:

```
Mes=Matriz(12)
```

O próximo passo seria apenas incluir os dados:

```
Mes[1] = 'Janeiro'
Mes[2]='Fevereiro'
...
Mes[12]='Dezembro'
```

Podemos também utilizar os dois métodos ao mesmo tempo!

```
Clientes=New Matriz(3)
        Clientes[1]=New CriaClientes("Charlene","Rua A, 51","331-0376",1150)
        Clientes[2]=New CriaClientes("José","Rua das Avencas, S/N","332-2781",950)
        Clientes[3]=New CriaClientes("Joaquim Manoel", "Rua Amancio Pinto, 171", ,1000)
```

Teríamos agora:

```
Clientes[1].nome = "Charlene";
Clientes[2].telefone="332-2781"
Clientes[3].telefone=null
```

3) Comandos

Além das estruturas de controle, o JavaScript oferece alguns poucos comandos:

- Break
- Continue
- Var
- With
- Function
- Return
- Comment

3.1) VAR

Em JavaScript, nem sempre é necessário definir uma variável antes de utilizá-la, é o que ocorre com variáveis globais, porém, é importante ressaltar que a utilização da instrução ***"var"***, a nível de documentação é muito bem-vinda. Já nas definições de variáveis locais, é obrigatório a utilização da instrução var.

Você pode armazenar um valor na própria linha de definição da variável, se não o fizer, para o JavaScript, esta variável possui um valor desconhecido ou nulo.

Não é obrigatória a utilização de uma instrução ***"var"*** para cada variável declarada, na medida do possível, você pode declarar várias variáveis em uma só instrução var.

Forma geral:

```
Var NomeDaVariável [ = <valor> ];
```

Exemplo:

```
Var Contador = 0;
Var Inic="",Tolls=0,Name="TWR";
Var Teste; // Neste caso, Teste possui valor null
```

3.2) with

Quando você precisa manipular várias propriedades de um mesmo objeto, provavelmente prefere não ser obrigado a repetir todas as vezes a digitação do nome do objeto. A instrução ***"with"***, permite fazer isso eliminando a necessidade de digitar o nome do objeto todas as vezes.

Forma geral:

```
with (<objeto>)
{
        ... Instruções
}
```

Por exemplo vamos supor que será necessário executar uma série de operações matemáticas:

```
with (Math)
{
        a=PI;
        b=Abs(x);
        c=E;
}
```

3.3) Break

Pode ser executado somente dentro de loops ***"for"***... ou ***"while"***... e tem por objetivo o cancelamento da execução do loop sem que haja verificação na condição de saída do loop, passando a execução a linha imediatamente posterior ao término do loop.

Forma geral:

```
Break
```

Exemplo:

Neste exemplo, quando a variável x atinge o valor 5 o loop é cancelado, e é impresso o número 5 na tela.

```
For (var x=1 ; x < 10 ; x++)
{
        If (x = = 5)
        {
                Break
        }
}
```

```
document.write(x) // resulta: 5
```

3.4) Continue

Pode ser executado somente dentro de loops ***"for"***... ou ***"while"*** ... e tem por objetivo o cancelamento da execução do bloco de comandos passando para o início do loop.

Forma geral:

```
Continue
```

Exemplo:

Neste exemplo, serão impressos os números de 1 a 10, com exceção do número 5, ou seja, quando a variável ***"x"*** atinge o valor 5 a execução do bloco de comandos é interrompida e o controle retorna ao início do loop, onde a variável ***"x"*** será incrementada.

```
For (var x=1 ; x < 10 ; x++)
{
        If (x = = 5)
        {
                continue
        }
        document.write(x)
}
```

3.5) Funções

As funções podem ser definidas como um conjunto de instruções, agrupadas para executar uma determinada tarefa. Dentro de uma função pode existir uma chamada a outra função. Para as funções podem ser passadas informações, as quais são chamadas de parâmetros.

As funções podem ou não retornar alguma informação, o que é feito com o comando ***Return***.

A definição de uma função é feita da seguinte forma:

```
Function NomeDaFunção([ parametro1, parametro2, ..., parametroN ])
{
        ...
        [Return(ValorDeRetorno)]
}
```

A chamada de funções é feita da seguite forma:

```
NomeDaFunção([parâmetros])
```

Funções são melhor declaradas entre as *tags* ***<head>*** de sua página *HTML*. Funções são frequentemente chamadas por eventos acionados pelo usuário. Assim parece razoável colocar as funções entre as *tags* <head>. Elas são carregadas antes que o usuário faça alguma coisa que possa chamar uma função.

```
<html>
    <head>
        <script language="LiveScript">
            Function hello(){
                alert("Alô mundo!!!");
            }
        </script>
    </head>
    <body>
        ...
        <script>hello();</script>
        ...
    </body>
</html>
```

Nota: *Em JavaScript, não é possível utilizar-se da recursividade, ou seja, uma função não pode chamar ela mesma.*

3.6) Comentários

Comentários podem ser formulados de varias maneiras, dependendo do efeito que você precisa. Para comentários longos de várias linhas, ou blocos de comentários, use:

/* O barra-asterisco inicia um bloco de comentário que pode conter quantas linhas você precisar todo o texto após o barra asterisco é ignorado, até que asterisco-barra seja encontrado, terminando assim o bloco de comentário ***/**

Para comentários de uma linha, use barra dupla (**//**) para introduzir o comentário. Todo o texto seguindo este simbolo até o próximo *carrige-return* será considerado um comentário e ignorado para fins de processamento. Exemplo:

```
// este texto será tratado como comentário
```

Os códigos JavaScript podem ser colocados em campos de comentário de modo que browsers antigos não mostrem o próprio código JavaScript, como vemos a seguir:

```
<html>
    <head>
        <script language="LiveScript">
        <!-- hide script from old browsers
            Function hello(){
                alert("Alô mundo!!!");
            }
```

```
                    // end hiding contents -->
                    </script>
                </head>

                <body>
                    ...
                    <script>hello();</script>
                    ...
                </body>
            </html>
```

3.7) Estruturas de Controle

Existem algumas estruturas de controle que lhe permitem modificar o fluxo de execução de um programa. Estas estruturas permitem executar o código baseado em condições lógicas ou um número determinado de vezes.

- For...
- For...In
- If...Else...
- While...

3.7.1) For...

Repete uma seção do código um determinado número de vezes. Consiste de uma declaração que define as condições da estrutura e marca seu início. Esta declaração é seguida por uma ou mais declarações executáveis, que representam o corpo da estrutura.

Estabelece um contador inicializando uma variável com um valor numérico. O contador é manipulado através da ***<ação>*** especificada no comando toda a vez que o *loop* alcança seu fim, permanecendo nesse *loop* até que a ***<condição>*** seja satisfeita ou a instrução ***Break*** seja executada.

Forma geral:

```
For (<inicialização> ; <condição> ; <ação>)
{ Corpo da Estrutura }
```

No exemplo abaixo, o bloco de instruções será executado 10 vezes, pois a variável ***contador*** é inicializada com o valor 1 e o bloco de instruções será executado enquanto ***contador*** for menor que 11. A cada execução do bloco de instruções ***contador*** é incrementado.

```
For (var contador = 1; contador < 11; contador++)
{ document.write(Contador); }
```

3.7.2) For...In

Este comando tem por objetivo, procurar a ocorrência de uma variável, dentro das propriedades de um objeto, ao encontrar a referida variável, um bloco de comandos pode ser executado.

Forma geral:

```
For (variavel In objeto)
{
        bloco de comandos
}
```

Exemplo: Esta função procura por uma propriedade do ***Objeto***, cujo o nome esteja especificado pela variável ***Procura***, onde ***Nome*** é uma string correspondendo ao nome do objeto.

```
Function SearchIn(Procura,Objeto,Nome)
{
        Var ResultadoDaBusca = ""
        For (Procura In Objeto)
        {
                document.write(Nome+"."+Procura+"="+Objeto[Procura]+"<BR>");
        }
}
```

3.7.3) If...Else...

A estrutura ***If***... executa uma porção de código se a condição especificada for verdadeira. A estrutura pode também ser especificada com código alternativo para ser executado se a condição for falsa.

```
Function VerificaIdade(anos)
{
        If anos >= 16
        {
                Return ('Já pode votar!')
        }
        else
        {
                Return (' Ainda é muito cedo para votar...')
        }
}
```

Uma alternativa para economizar ***If***'s seria a utilização de uma expressão condicional, que funciona para situações mais simples, o exemplo acima ficaria da seguinte forma:

```
VariavelDeRetorno= (anos>=16) ? 'Já pode votar!' : 'Ainda é muito cedo para votar...'
```

3.7.5) While

Outro tipo de ***loop*** é aquele baseado numa condição ao invés de no número de repetições. Por exemplo, suponha que você necessita que um determinado processo seja repetido até que um determinado teste dê um resultado verdadeiro ou seja executada a instrução ***Break***.

Forma geral:

```
while (<condição>)
{ Corpo da Estrutura }
```

No exemplo abaixo, o bloco de instruções será executado 10 vezes, pois a variável ***Contador*** é inicializada com o valor 1 e o bloco de instruções será executado enquanto ***Contador*** for menor que 11. A cada execução do bloco de instruções ***Contador*** é incrementado.

```
Var Contador=1;
While ( Contador < 11 )
{ document.write(Contador++) ;}
```

4) Funções internas

A linguagem JavaScript além dos recursos descritos anteriormente, ainda possui algumas funçòes internas, que não estão ligadas diretamente a nenhum objeto, porém isso não impede que essas funções recebam objetos como parâmetros. A seguir estas funções serão vistas detalhadamente:

alert - Mostra uma caixa de alerta, seguido de um sinal sonoro e o botão de OK.

Ex:

```
alert('Esta é uma janela de alerta!')
```

confirm - Mostra uma caixa de diálogo, seguida de um sinal sonoro e os botão de OK e Cancel. Retorna um valor verdadeiro se o usuário escolher OK.

```
Ex: retorno=confirm('Deseja prosseguir?')
```

escape - Obtém o código ASCII de um caracter que não seja alfa-numérico.

```
Ex: document.write(escape("@"))
```

eval - Avalia uma expressão numérica, retornando um resultado também numérico.

```
Ex: document.write(eval(10*9*8*7*6*5*4*3*2))
```

parseFloat - Converte uma *string* que representa um número, para um número com ponto flutuante. Caso a *string* não possa ser avaliada, a função retorna 0.

```
Ex: document.write(parseFloat("-32.465e12")
```

parseInt - Converte uma *string*, que representa um número em uma base predefinida para base 10. Caso a *string* possua um caracter que não possa ser convertido, a operação para, retornando o valor antes do erro.

```
Ex: paseInt("string",base)
        parseInt("FF",15) // retorna 256
        parseInt("3A",10) // retorna 3
        parseInt("10",2) // retorna 2
```

prompt - Monta uma caixa de entrada de dados, de forma simplificada comparando-se com o objeto text.

```
Ex: prompt(label [,valor])
```

onde:

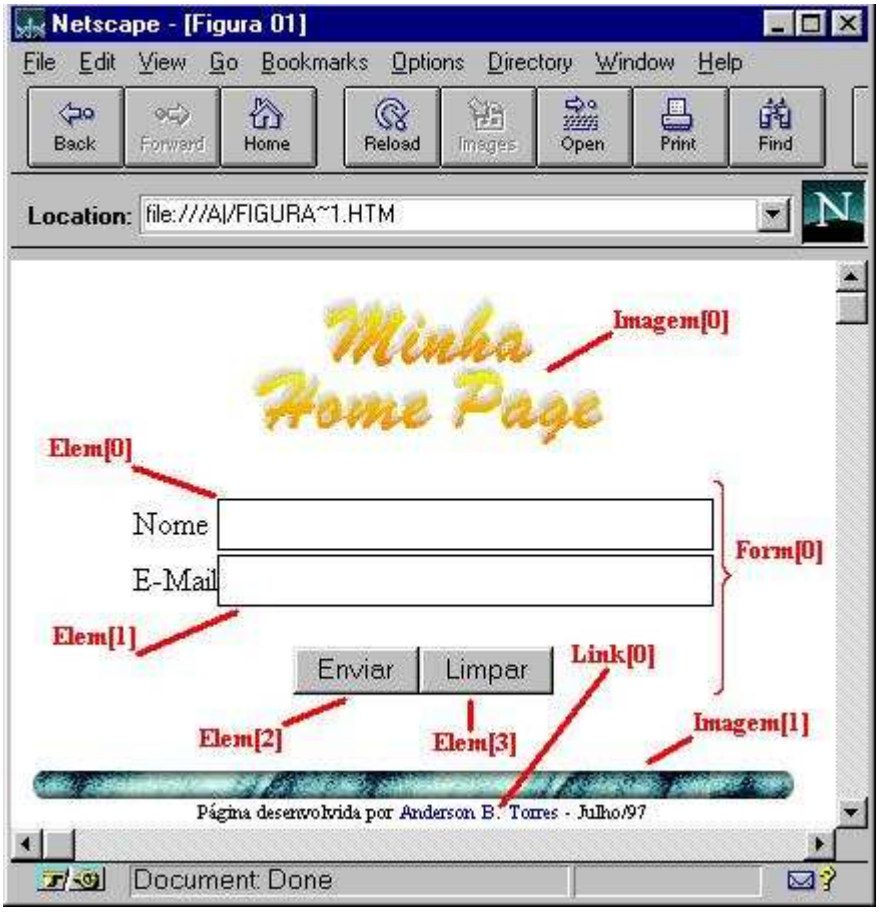
label - texto que aparece ao lado da caixa.

valor - é o conteúdo inicial da caixa.

5) Objetos JavaScript

JavaScript organiza todos os elementos de uma Home Page dentro de uma hierarquia. Cada elemento é visto como um objeto. Os objetos podem ter propriedades, métodos e responder a certos eventos. Por isso é muito importante entender a hierarquia dos objetos HTML.

Você entenderá rapidamente como isto funciona com a ajuda de um exemplo. O exemplo seguinte é uma página HTML simples:



No exemplo acima, nós temos, um *link*, duas imagens, e um formulário com dois campos texto e dois botões. Do ponto de vista do JavaScript a janela do *browser* é um objeto window*, que* contém certos elementos, como a barra de *status*.

Dentro da janela, nós podemos carregar uma página *HTML*. Esta página é um **objeto document**. Desta forma o objeto *document* representa o documento HTML (que está carregado no momento). O objeto *document* é muito importante, em JavaScript você sempre o usará muito. As propriedades e métodos do objeto *document* serão vistas detalhadamente, mais adiante.

Mas o que é mais importante é que todos os objetos *HTML* são propriedades do objeto *document*. Um objeto *HTML* pode ser por exemplo um *link* ou um formulário.

Nós podemos obter informações de diversos objetos e então manipulá-los. Para isso nós devemos saber como acessar os diferentes objetos. Você primeiro verifica o nome do objeto no diagrama de hierarquia. Se você então precisar saber como referenciar a primeira imagem na página *HTML*, basta seguir o caminho hierárquico. Você deve percorrer o diagrama de cima para baixo, o primeiro objeto é chamado **document**, a primeira imagem é representada por *Imagem[0]*. Desta forma nós podemos acessar este objeto em JavaScript, da seguinte forma: *document.Imagem[0]*.

Se você quiser saber o que o usuário digitou dentro do primeiro elemento do formulário, você primeiro precisa pensar em como acessar esse objeto. Novamente nós seguiremos o diagrama de hierarquia, de cima para baixo. Siga o caminho que leva até *Elem[0]*. Todos os nomes de objeto por onde você passou tem que constar na referência ao primeiro elemento do formulário. Desta forma você pode acessar o primeiro elemento texto assim: *document.Form[0].Elem[0]*

Mas como obteremos agora, o texto digitado? Este elemento texto possui uma propriedade chamada *value* - não se preocupe agora, com propriedades, métodos ou eventos, eles serão vistos detalhadamente mais adiante - esta propriedade armazena o conteúdo do objeto, ou seja, o texto digitado. A seguinte linha de código obtém o texto digitado:

```
nome = document.forms[0].elements[0].value;
```

A *string* é armazenada na variável *name*. Nós podemos agora trabalhar com esta variável. Por exemplo, nós podemos criar uma janela *popup* com ***alert("Oi "+name);***. Se a entrada for *"Anderson"* o comando ***alert("Oi "+name)*** abrirá uma janela popup com o texto *"Oi Anderson"*.

Se você estiver trabalhando com páginas muito grandes pode ficar um pouco confuso referenciar objetos diretamente pelo endereçamento do diagrama de hierarquia, você pode ter referências do tipo: *document.forms[3].elements[15]* ou *document.forms[2].elevent[21]*. Para evitar esse problema você pode dar nomes diferentes aos objetos, vejamos o seguinte fragmento de um documento *HTML*:

```
<form NAME="clientes">
    Nome: <input TYPE="text" NAME="empresa" value=" ">
    ...
```

Dessa forma, em vez de usarmos, por exemplo:

```
document.forms[0].elements[0].value;
```

Podemos usar:

```
document.clientes.empresa.value;
```

Isto traz muitas facilidades, especialmente com páginas grandes e com muitos objetos. Observe que a caixa das letras faz diferença. Muitas propriedades dos objetos JavaScript não são apenas para leitura, você pode atribuir novos valores a algumas propriedades. Observe o exemplo:

```
Location
```

Este objeto contém informações sobre a URL da página atual.

Forma geral:

```
location.propriedade
location.metodo()
```

Propriedades:

hash - Esta propriedade funciona de forma semelhante ao famigerado ***"go to"*** de algumas linguagens de programação. Normalmente é usado em *links*, que acessam a mesma página.

Ex: O exemplo abaixo demonstra a utilização da propriedade *hash*, para criar um *link* para outro trecho da mesma página.

```
<HTML>
    ...
    <A HREF = "location.hash='2'">Item 1</A>
    ...
    <A NAME = "1"> </A>Item 1
    ...
    <A NAME = "2"> </A>Item 2
    ...
</HTML>
```

host - Armazena uma *string* com o formato "hostname:port" da página HTML atual.

```
Ex: alert('Demostração da propriedade host: '+location.host)
```

hostname - Armazena uma string, com o IP da página HTML atual.

```
Ex: alert('Demostração da propriedade hostname: '+location.hostname)
```

href - String identica a mostrada na barra "location" do browser.

```
Ex: alert('A URL desta página é: '+ location.href)
```

pathname - Contém uma string com o path da página HTML atual.

```
Ex: alert('O path da URL desta página é: '+ location.pathname)
```

port - Armazena a porta por onde está sendo feita a conexão com o servidor.

```
Ex: alert('A porta usada para conexão com o servidor é: '+ location.port)
```

protocol - String que armazena o protocolo de acesso a um determinado endereço. ("http:","ftp:","file:").

```
Ex: alert('O protocolo de acesso para esta página é: '+ location.protocol)
```

Métodos:

toString - Converte o conteúdo do objeto location para uma string.

```
Ex: alert('location.toString() = '+location.toString) // Este valor é o mesmo que location.href.
```

5.1) Select

Cria uma listBox, no mesmo padrão que no Windows. Onde o usuário pode selecionar uma ou mais opções disponíveis, depende da configuração desejada pelo programador.

Forma geral:

```
<SELECT
        NAME = "selectName"
        [SIZE = tamanho]
        [MULTIPLE]
        [onBlur = "ação"]
        [onChange = "ação"]
        [onFocus = "ação"] >

                <OPTION
                VALUE = "optionValue"
                [SELECTED] >
                        Texto
                        . . .
                <OPTION...>
        </SELECT>
```

onde:

selectName - Nome dado pelo programador, para o objeto select

tamanho - Número de linhas, da caixa select.

MULTIPLE - Se definido, permite que várias opções sejam selecionadas.

ação - Define o que fazer quando algum evento ocorrer.

optionValue - Valor que é enviado as servidor, quando o formulário é submetido.

SELECTED - Se definido, informa a opção que será inicialmente selecionada.

Propriedades:

lenght - Informa o número de opções disponíveis.

Ex: selectName.lenght

name - Informa o nome que o programador definiu para o objeto select.

Ex: selectName.name

options - Vetor com todas as opções existentes no menu select.

Ex: selectName.options[0..selectName.lenght-1]

selectedIndex - Informa o índice do ítem que está selecionado.

Ex: selectName.selectedIndex

defaultSelected - Informa o ítem que detém a seleção inicial. Pode-se alterar este valor, dede que o formulário ainda não tenha sido exibido.

Ex: selectName.options[Indice].defaultSelected

index - Obtém o número do índice de uma opção em um menu select.

Ex: selectName.options[Indice].index

selected - Valor lógico referente a opção em questão. Se a opção estiver selecionada, retorna "1", caso contrário, retorna "0".

Ex: selectName.options[indice].selected

text - Armazena o texto que aparece como opção do menu select. Este texto é definido após a TAG <OPTION>.

Ex: selectName.options[indice].text

value - Armazena o campo VALUE, que é enviado ao servidor quando o formulário é submetido (enviado).

Eventos:

onBlur - Ocorre quando o objeto perde o foco.

onChange - Ocorre quando o objeto perde o foco e seu conteúdo foi alterado.

onFocus - Ocorre quando o objeto recebe o foco.

5.2) Button

Este objeto mostra um botão 3-D (no mesmo padrão do Windows). ao ser pressionado, ele produz um efeito de profundidade e geralmente chama uma função.

Pode ser utilizado para inúmeras aplicações, dependendo apenas de sua imaginação, a única precaução é defini-lo dentro de um formulário.

Forma geral:

```
<Input Type="button" Name="NomeDoBotão" Value="Rótulo" onClick="RespostaAoEvento">
```

Onde:

```
Type - define o objeto
```

Name - define o nome do objeto para nossa aplicação. É por este nome que referenciamos alguma propriedade deste objeto

Value - define o que será escrito na face do botão

onClick - É o único evento possível para este objeto, normalmente define uma função a ser executada quando clicamos no botão.

Propriedades:

NAME: Informa o nome do objeto button em forma de string, da mesma forma como definido no campo Name que foi utilizado na definição do botão. É importante não confundir o campo Name com a propriedade NAME, veja a diferença:

```
<input type="button" name="OK" value="Confirma" onClick="ConfirmaInformacoes()">
OK.Name // equivale a "OK"
```

VALUE: Informa o label do botão em forma de string da mesma forma como foi definido no campo Value que foi utilizado na definição do botão.

```
OK.Value // equivale a "Confirma"
```

Métodos:

c**lick:** Este método simula um clique do mouse no objeto button, ou seja, executa um procedimento associado a um botão como se o botão tivesse sido pressionado mas sem que o usuário tenha realmente clicado.

```
OK.click() // executaria a função ConfirmaInformacoes
```

Eventos associados:

onClick: Define o que fazer quando clicamos no objeto button

Exemplo:

```
<FORM>
        <INPUT TYPE="button" VALUE="Clique aqui!!!" NAME="botao1" onClick="alert('A propriedade NAME deste botão é:'+botao1.name+'\nA propriedade VALUE deste botão é:'+botao1.value)")>
</FORM>
```

5.3) Navigator

Neste objeto ficam armazenadas as informações sobre o browser que está sendo utilizado.

Forma geral:

```
Navigator.propriedade
```

Propriedades:

appCodeName - Armazena o codnome do browser.

```
Ex: Navigator.appCodeName
```

appName - Armazena o nome do browser.

```
Ex: Navigator.appName
```

appVersion - Armazena a versão do browser.

```
Ex: Navigator.appVersion
```

userAgent - Armazena o cabeçalho (user-agent) que é enviado para o servidor, no protocolo HTTP, isto serve para que o servidor identifique o software que está sendo usado pelo cliente.

```
Ex: Navigator.userAgent
```

Exemplo:

```
<HTML>
<HEAD>
        <TITLE>JavaScript </TITLE>
        <SCRIPT>
        <!--
        function getBrowserName() {
```

```
            document.forms[0].elements[0].value =navigator.appName;
            }

            function getBrowserVersion() {
            document.forms[0].elements[0].value = navigator.appVersion;
            }

            function getBrowserCodeName() {
            document.forms[0].elements[0].value = navigator.appCodeName;
            }

            function getBrowserUserAgent() {
            document.forms[0].elements[0].value = navigator.userAgent;
            }

            function getBrowserNameVersion() {
            document.forms[0].elements[0].value = navigator.appName + " " + navigator.appVersion;
            }
            // -->
            </SCRIPT>
</HEAD>
<BODY >
            <CENTER>
            <FORM NAME="detect">
            <INPUT TYPE="text" NAME="browser" SIZE=50 MAXLENGTH=50 VALUE=" Seus dados serão mostrados nesta janela ! ">
            <BR><BR><BR>
            <INPUT TYPE="button" VALUE="Nome do Navegador" onClick="getBrowserName()">
            <INPUT TYPE="button" VALUE="Versão do Navegador" onClick="getBrowserVersion()">
            <INPUT TYPE="button" VALUE="E-mail" onClick="getBrowserCodeName()">
            <BR><BR>
            <INPUT TYPE="button" VALUE="E-mail e versão" onClick="getBrowserUserAgent()">
            <BR> <BR>
            <INPUT TYPE="button" VALUE="Nome e Versão" onClick="getBrowserNameVersion()">
            </FORM>
</BODY>
</HTML>
```

5.4) Form

Os formulários tem muitas utilidades, uma das principais seria a transferência de dados dentro da própria página HTML, sem que para isso seja necessária a intervenção de qualquer outro meio externo.

Ao se criar um formulário na página HTML, automaticamente é criada uma referência para este formulário, que fica guardada na propriedade form do objeto document. Como você deve ter visto na página

que trata do objeto document, a propriedade form é um vetor, e a cada formulário criado também é criado um novo elemento para este vetor, o índice inicial deste vetor é 0 (zero) e varia até o número de formulários do documento -1.

Como você pode notar, cada formulário criado em uma página HTML, é considerado um objeto distinto, tendo suas próprias referências, métodos, propriedades e eventos associados. A forma de acessarmos diferenciadamente esses formulários dentro da página HTML, é utilizar a propriedade form do objeto document.

Forma geral:

```
<FORM NAME="Nome"]
        [ACTION="URL"]
        [METHOD="GET | POST"]
        [onSubmit="evento"]>
```

Onde:

Nome = Nome do formulário, para futuras referências dentro da página HTML.
URL = Especifica o URL do servidor ao qual sera enviado o formulario.
GET | **POST** = metodos de transferencia de dados do browser para o servidor

Propriedades:

action - Especifica o URL do servidor ao qual sera enviado o formulario.

```
Ex: document.NomeDoFormulario.action
```

```
documet.GuestBook.action = "esaex@canudos.ufba.br"
```

elements - Vetor que armazena todos os objetos que são definidos dentro de um formulário (caixas de texto, botões, caixas de entrada de dados, checkboxes, botões de rádio). O número de elementos deste vetor varia de 0 (zero) até o número de objetos dentro do formulário -1.

```
Ex: document.NomeDoFormulario.elements[indice]
```

method - Seleciona um método para acessar o URL de ação. Os métodos são: get e post. Ambos os métodos transferem dados do browser para o servidor, com a seguinte diferença:

method=get - os dados de entrada são acrescentados ao endereço (URL) associado, como se fossem uma query (pesquisa a banco de dados) comum;

method=post - os dados não são acrescentados ao URL, mas fazem parte do corpo da mensagem enviada ao servidor.

*Obs.: O método mais usual é o **post**:*

Esta propriedade pode ser alterada, porém só surtirá efeito antes que o formulário seja mostrado na tela.

```
Ex: document.NomeDoFormulario.method = "post" ( ou "get")
```

Métodos:

submit - Transfere os dados do formulário para o endereço especificado em action, para serem processados. Funcionado de maneira análoga ao botão submit em HTML.

```
Ex: document.NomeDoFormulario.submit( )
```

Eventos:

onSubmit - Ocorre quando um botão do tipo ***submit*** recebe o clique do mouse, transferindo os dados de um formulário para o servidor especificado em action.

Os dados só são enviados se o evento receber um valor verdadeiro (true), valor este que pode ser conseguido como resultado a chamada de uma função que valida as informações do formulário.

```
Ex: document.NomeDoFormulario.onSubmit ="return Valida_Informacoes(NomeDoFormulario)"
```

Exemplo:

```
<HTML>
<HEAD>
<TITLE>Exemplo - Objeto Form</TITLE>
</HEAD>
<BODY>
<FORM action="mailto:esaex@canudos.ufba.br" method="POST">
<P><TT><B><H1>Exemplo:</H1></B></TT>
<P>Este exemplo demonstra a funcionalidade de um
formul&aacute;rio, para improvisar um "Guest Book"
<P> 
Nome,Nascimento: <BR>
<INPUT TYPE="text" NAME="nomidade" VALUE=" " SIZE=70><BR>
Endere&ccedil;o: <BR>
<INPUT TYPE="text" NAME="endereco" VALUE=" " SIZE=70><BR>
E-Mail: <BR>
<INPUT TYPE="text" NAME="email" VALUE=" "
SIZE=70><BR>
```

```
Sua Home-Page: <BR>
<INPUT TYPE="text" NAME="hp" VALUE=" " SIZE=70><BR>
IRC: <BR>
<INPUT TYPE="text" NAME="irc"
VALUE=" " SIZE=70><BR>
Sugest&otilde;es, etc.: <BR>
<TEXTAREA NAME="sujestao" ROWS=7 COLS=70></TEXTAREA>
<P><CENTER><INPUT TYPE="submit" NAME="Submit"
VALUE="Enviar"> <INPUT TYPE="reset" VALUE="Limpar"></CENTER>
</FORM>
<CENTER>
<FORM>
<INPUT TYPE="button" VALUE="Página Anterior" onClick="history.go(-1)">
<IMG SRC="S177.gif" WIDTH=540 HEIGHT=46 ALIGN=bottom><BR>
<FONT SIZE="-2">P&aacute;gina desenvolvida por </FONT><FONT
SIZE="-2"><A HREF="mailto:esaex@canudos.ufba.br">Anderson Barros
Torres</A></FONT><FONT SIZE="-2">. Julho/97</FONT>
</FORM>
</CENTER>
</BODY>
</HTML>
```

5.5) CheckBox

Este objeto como o próprio nome sugere, exibe uma caixa de checagem igual às que encontramos no Windows, o funcionamento também é o mesmo: a condição de selecionada ou não, é alternada quando clicamos o mouse sobre o objeto, ou seja, se clicarmos sobre um objeto checkbox já marcado ele será automaticamente desmarcado, ao passo que se clicarmos em um objeto checkbox desmarcado ele será automaticamente marcado.

Forma geral:

```
<FORM>
<INPUT TYPE="checkbox" NAME="NomeDoObjeto" [CHECKED] VALUE="Label" onClick="Ação">
</FORM>
```

Onde:

Type - Nome do objeto;

Name - Nome dado pelo programador, para futuras referenciações ao objeto;

CHECKED - Se especificado a CheckBox já vai aparecer selecionada;

Value - Define um rótulo para a CheckBox.

onClick - Define o que fazer quando dá-se um clique na CheckBox, fazendo com que o objeto CheckBox funcione como um objeto Button.

Propriedades:

name - Nome do objeto CheckBox em forma de string, da mesma forma como definido no campo Name utilizado na criação da CheckBox.

```
NomeDoObjeto.name // equivale a string "NomeDoObjeto"
```

value - Armazena o conteúdo do campo VALUE, definido na TAG.

```
NomeDoObjeto.value
```

checked - Retorna um valor lógico que depende do estado do objeto CheckBox

```
NomeDoObjeto.checked
// equivale a True se o objeto  selecionado e False caso contrário
```

defaultChecked - Informa/Altera o estado default de um objeto CheckBox. Com relação a alteração, somente os objetos CheckBox ainda não exibidos podem ter seu estado default alterado.

```
NomeDoObjeto.defaultChecked
// equivalerá a True, se a cláusula CHECKED estiver presente e a False caso contrário
```

Métodos:

click: este método simula um clique do mouse no objeto *CheckBox*, ou seja, executa um procedimento associado a uma *CheckBox* como se estivéssemos clicado na *CheckBox* mas sem que o usuário tenha realmente clicado.

```
Select01.click() // executaria uma função
```

Eventos associados:

onClick: Define o que fazer quando clicamos no objeto *CheckBox*

Exemplo:

```
<HTML>
    <HEAD>
        <TITLE>Exemplo CheckBox</TITLE>
        <SCRIPT>
            function exemplo(p1,p2,p3,p4){
                alert('Veja os conteúdos das propriedades:
                \nName='+p1+
                '\nValue='+p2+
                '\nChecked='+p3+
                '\ndefaultChecked='+p4);
            }
        </SCRIPT>
```

```
</HEAD>
<BODY>
    <CENTER>
        <H3>Exemplo do objeto CheckBox</H3>
        <HR>
        <FORM>
            <INPUT TYPE="checkbox" NAME="chb" VALUE="QQ COISA" CHECKED
                    onClick="exemplo(chb.name,chb.value,chb.checked,chb.defaultChecked)")>
            Tecle aqui...
        </FORM>
        <BR><HR><BR>
        Tecle no CheckBox para observar o funcionamento!!!
        Para retornar clique o mouse <A HREF="history.go(-1)">AQUI</A>
    </CENTER>
</BODY>
</HTML>
```

5.6) Document

Este objeto armazena todas as características da página HTML, como por exemplo: cor das letras, cor de fundo, figura que aparecerá como papel de parede, etc. Sempre que incluímos alguma declaração no <*body*> do documento, estamos alterando o objeto *Document*.

Forma geral:

```
<body [background="imagem"]
        [bgcolor="#cordefundo"]
        [fgcolor="#cordotexto"]
        [link="#cordoslinks"]
        [alink="#cordolinkativado"]
        [vlink="#cordolinkvisitado"]
        [onload="função"]
        [onunload="funcao"]>
```

Onde:

Imagem = figura no formato GIF, que servirá como papel de parede para a Home Page;

#Cor... = número (hexadecimal), com seis dígitos, que corresponde a cor no formato RGB, o "#" é obrigatório. Os dois primeiros dígitos correspondem a R (red), os dois do meio a G (green) e os dois últimos a B (blue). A combinação dos três, forma a cor no formato RGB.

função = Nome de uma função pré-definida, que será chamada quando o evento ocorrer.

Propriedades:

alinkColor - Determina a cor do *link* enquanto o botão do o mouse estiver pressionado sobre ele.

```
Ex: document.alinkColor="#FFFFFF"
```

anchors - Vetor que armazena as âncoras definidas em uma página *HTML* com o comando *<A NAME="ancora">*. Esta propriedade é somente para leitura, não pode ser alterada.

```
Ex: document.anchors[índice]
```

bgColor - Determina a cor de fundo da página HTML.

```
Ex: document.bgColor="#000000"
```

cookie - Os *cookies* são pequenos arquivos que alguns sites da Web gravam no computador dos visitantes. A idéia é identificar o usuário, anotar quais caminhos ele já percorreu dentro do site e permitir um controle mais eficaz dos espectadores.

fgColor - Determina a cor das letras em uma página *HTM*L. Esta propriedade não altera o que já está impresso na página HTML.

```
Ex: document.fgColor="#0000FF"
```

forms - Vetor que armazena as referências aos formulários existentes na página HTML. Esta propriedade é somente para leitura, não pode ser alterada.

```
Ex: document.forms[índice]
```

lastModified - Obtém a data da última atualização da página HTML. Esta propriedade é somente para leitura, não pode ser alterada.

```
Ex: document.lastModified
```

linkColor - Determina a cor dos *links* que ainda não foram visitados pelo usuário.

```
Ex: document.linkColor = "#00FF00"
```

links - Vetor que armazena os *links* definidos em uma página *HTML*. Esta propriedade é somente para leitura, não pode ser alterada.

```
Ex: document.links[índice]
```

location - Armazena o endereço *(URL)* atual em forma de *string*. Esta propriedade é somente para leitura, não pode ser alterada.

referrer - Armazena o endereço *(URL)* de quem chamou a página HTML atual. Com essa propriedade você pode saber como usuário chegou à sua página. Esta propriedade é somente para leitura, não pode ser alterada.

```
Ex: document.referrer
```

title - Armazena uma *string* com o título da página *HTML* atual. Esta propriedade é somente para leitura, não pode ser alterada.

```
Ex: document.title
```

vlinkColor - Determina a cor que o link aparecerá após ser visitado.

```
Ex: document.vlinkColor = "#80FF80"
```

Métodos:

clear - limpa a tela da janela atual.

```
Ex: document.clear( )
```

open - Abre um documento e envia (mas não exibe) a saída dos métodos write/writeln. Os dados enviados são exibidos, quando é encontrado o método close.

```
Ex: document.open( )
```

close - Termina uma seqüência iniciada com o método open, exibindo o que tinha sido apenas enviado.

```
Ex: document.close( )
```

write - Imprime informações na página HTML.

```
Ex: document.write("Qualquer coisa" [,variável] [,..., expressão])
```

writeln - Imprime informações na página HTML e passa para a próxima linha. Em meus testes, esse método não apresentou diferença com relação ao método write.

```
Ex: document.writeln("Qualquer coisa" [,variável] [,..., expressão])
```

Eventos:

onLoad - Ocorre assim que um browser carrega uma página *HTML* ou *frame*.

```
Ex: <BODY ... onLoad='alert("Oi!!!")'>
```

onUnload - Ocorre quando se abandona uma página HTML ou frame.

```
Ex: <BODY ... onUnload='alert("Tchau!!!")'>
```

5.7) Date

Objeto muito útil que retorna a data e hora do sistema no seguinte formato: *Dia da semana, Nome do mês, Dia do mês, Hora:Minuto:Segundo e Ano*. Como todo objeto, podem ser criadas novas instâncias para este objeto, essa prática possibilita a utilização de quantos objetos data você precisar.

Forma geral:

```
NovoObjeto = NEW date( )
```

Onde:

NovoObjeto = Objeto definido pelo usuário, para manipular datas.

Métodos:

getMonth - Obtém o número do mês. Retornando um valor entre 0 e 11. (janeiro=0)

```
Ex: Mes=NovoObjeto.getMonth( )
```

setMonth - Estabelece um novo valor para o mês. O valor deve estar entre 0..11

```
Ex: NovoObjeto.setMonth(NumeroDoMes)
```

getDate - Obtém o número do dia, considerando-se o mês. Retornando um valor numérico entre 1..31.

```
Ex: dia = NovoObjeto.getDate( )
```

setDate - Estabelece um novo valor para o dia do mês. Este valor deve estar entre 1..31

```
Ex: NovoObjeto.setDate(NumeroDoDia)
```

getDay - Obtém o número do dia, considerando-se a semana. Retornando um valor numérico entre 0..6. Lembre-se de que a semana começa no domingo, logo 0, corresponde ao domingo.

```
Ex: DiaDaSemana = NovoObjeto.getDay( )
```

getHours - Obtém um número correspondente a hora. Retornando um valor numérico entre 0..23

```
Ex: Hora = NovoObjeto.getHours( )
```

setHours - Estabelece um novo valor para a hora. O valor deve estar entre 0..23

```
Ex: NovoObjeto.setHours(NovaHora)
```

getMinutes - Obtém um número correspondente aos minutos. Retornando um valor numérico entre 0..59

```
Ex: Minutos = NovoObjeto.getMinutes( )
```

setMinutes - Estabelece um novo valor para os minutos. O valor deve estar entre 0..59

```
Ex: NovoObjeto.setMinutes(Minutos)
```

getSeconds - Obtém um número correspondente aos segundos. Retornando um valor numérico entre 0..59

```
Ex: Segundos = NovoObjeto.getSeconds( )
```

setSeconds - Estabelece um novo valor para os segundos. O valor deve estar entre 0..59

```
Ex: NovoObjeto.setSeconds(Segundos)
```

getTime - Obtém o tempo decorrido desde 01/01/70 até o presente momento. O único inconveniente é que esta data é dada em miliSsegundos.

```
Ex: TempoDecorrido=NovoObjeto.getTime( )
```

setTime - Estabelece uma nova data.

```
Ex: DataDeNascimento=New Date("August 2, 1970")
```

uma outra forma para definir a data seria:

```
OutraForma = New Date( )
OutraForma.setTime(DataDeNascimento.getTime( ))
```

getTimezoneOffset - Obtém a diferença entre o horário local e o horário do meridiano central (Greenwich). Este tempo é dado em minutos, logo, para saber o fuso-horário, deve-se dividir o resultado obtido por esta função por 60.

```
Ex: FusoHorário=NovoObjeto.getTimezoneOffset( ) / 60
```

getYear - Obtém um valor numérico correspondente ao ano.

```
Ex: Ano=NovoObjeto.getYear ( )
```

setYear - Estabelece um novo valor ao ano. O valor deve ser maior ou igual a 1900.

```
Ex: NovoObjeto.setYear(1997)
```

toGMTstring - Converte um objeto data para uma string seguindo o padrão Internet GMT.

```
Ex: NovoObjeto.toGMTstring( )
```

toLocaleString - Converte uma data para uma string seguindo o padrão local.

```
Ex: NovoObjeto.toLocalString( )
```

Exemplo:

```
<HTML>
    <HEAD>
        <TITLE>Exemplo - Objeto Date</TITLE>
        <SCRIPT>
        <!--
            var timerID = null;
            var timerRunning = false;
            function startclock ()
            {
              stopclock();
              time();
            }

            function stopclock ()
            {
              if(timerRunning)
              clearTimeout(timerID);
             timerRunning = false;
            }

            function time ()
            {
                var now = new Date();
            var yr = now.getYear();
            var mName = now.getMonth() + 1;
            var dName = now.getDay() + 1;
            var dayNr = ((now.getDate()<10) ? "0" : "")+ now.getDate();
            var ampm = (now.getHours() >= 12) ? " P.M." : " A.M."
            var hours = now.getHours();
            hours = ((hours > 12) ? hours - 12 : hours);
            var minutes = ((now.getMinutes() < 10) ? ":0" : ":") + now.getMinutes();
            var seconds = ((now.getSeconds() < 10) ? ":0" : ":") + now.getSeconds();
            if(dName==1) Day = "Domingo";
            if(dName==2) Day = "Segunda";
            if(dName==3) Day = "Terça";
            if(dName==4) Day = "Quarta";
            if(dName==5) Day = "Quinta";
            if(dName==6) Day = "Sexta";
```

```
            if(dName==7) Day = "Sabado";

            if(mName==1) Month="Janeiro";
             if(mName==2) Month="Fevereiro";
            if(mName==3) Month="Março";
            if(mName==4) Month="Abril";
            if(mName==5) Month="Maio";
            if(mName==6) Month="Junho";
            if(mName==7) Month="Julho";
            if(mName==8) Month="Augosto";
             if(mName==9) Month="Setembro";
            if(mName==10) Month="Outubro";
            if(mName==11) Month="Novembro";
             if(mName==12) Month="Dezembro";

            var DayDateTime=(" "
                        + Day
                        + ", "
                        + dayNr
                        + " de "
                        + Month
                        + " de "
                        + ""
                        + "19"
                        + yr
                        + ". Agora são:"
                        + hours
                        + minutes
                        + seconds
                        + " "
                        + ampm
                        );

            window.status=DayDateTime;
             timerID = setTimeout("time()",1000);
            timerRunning = true;
      }

      function clearStatus()
      {
            if(timerRunning)
            clearTimeout(timerID);
            timerRunning = false;
            window.status=" ";
      }
//-->
</SCRIPT>
```

```
</head>
 <BODY BACKGROUND="b190.gif" onLoad="startclock ()">
    <H1>Exemplo:</H1>
     Demonstração do objeto Date, conforme visto na página anterior. Funcionamento: a data e hora ficam sendo mostradas no rodapé do browser.
    <FORM>
        <CENTER>
            <BR>
            <INPUT TYPE="button" VALUE="Página Anterior" onClick="history.go(-1)">
        </CENTER>
    </FORM>
    <CENTER>
        <IMG SRC="S177.GIF"><BR>
        <H6>Página desenvolvida por <A HREF="mailto:webmaster.@areainicial.zzn.com"> Fernando Dercoli </A>. Julho/97</H6>
    </CENTER>
</BODY>
</HTML>
```

5.8) History

Este objeto armazena todas as *URL* das páginas *HTML* por onde o usuário passou durante a sessão atual do browser. É uma cópia das informações armazenadas na opção *Go* da barra de menu do *Navigator*.

Forma geral:

```
history.propriedade
history.método
```

Propriedades:

lenght - Informa a quantidade de páginas visitadas.

```
Ex: history.lenght
```

Métodos:

back - Retorna à página anterior, de acordo com a relação de páginas do objeto *history*. Equivale a clicar o botão *back* do *browser*.

```
Ex: history.back()
```

forward - Passa para a próxima página, de acordo com a relação de páginas do objeto *history*. Equivale a clicar o botão *forward* do browser.

```
Ex: history.forward()
```

go - Permite que qualquer URL que esteja presente na relação de páginas visitadas do objeto history, seja carregada.

```
Ex: history.go(parâmetro)
```

Existem duas possibilidades para "parâmetro":

1 - parâmetro é um número: Ao definir um número, este deve ser inteiro. Se for positivo, a página alvo está "parâmetro"páginas à frente. Ao passo que se for negativo, a página alvo está "parâmetro" páginas para traz.

2 - parâmetro é uma string: Neste caso, o alvo é a URL que mais se assemelhe ao valor da string definida por "parâmetro".

5.9) Window

É o objeto que ocupa o topo do esquema hierárquico em *JavaScript*.

Propriedades:

defaultStatus - Determina o conteúdo padrão da barra de status do browser, quando nada de importante estiver acontecendo.

```
Ex: widow.defaultStatus='Qualquer coisa'
```

frames - Vetor que armazena as referências para as *frames* da janela atual.

```
Ex: parent.frames.lenght
// obtém o número de frames da janela principal, assumindo que estamos em uma frame.
```

parent - Refere-se a janela pai da *frame* atual.

self - Refere-se a janela atual.

```
Ex: self.defaultStatus='Qualquer coisa'
```

status - Define uma mensagem que irá aparecer no rodapé do browser, em substituição por exemplo, a *URL* de um *link*, quando estivermos com o mouse sobre o *link*.

```
Ex: window.status="qualquer texto"
```

top - Refere-se a janela de nível mais alto do esquema hierárquico do JavaScript.

```
Ex: top.close() // fecha a janela principal do browser
```

window - Refere-se a janela atual. Funciona de modo análogo a self.

```
Ex: window.status='Qualquer coisa'
```

Métodos:

alert - Mostra uma caixa de alerta, seguido de um sinal sonoro e o botão de *OK*.

```
Ex: alert('Esta é uma janela de alerta!')
```

close - Termina a sessão atual do browser.

```
Ex: top.close()
```

confirm - Mostra uma caixa de diálogo, seguida de um sinal sonoro e os botão de *OK* e *Cancel*. Retorna um valor verdadeiro se o usuário escolher *OK*.

```
Ex: retorno=confirm('Deseja prosseguir?')
```

open - Abre uma nova sessão do *browser*, como se o usuário pressionasse *<CTRL>+N*

```
Ex: window.open("URL", "Nome" [,"características"])
```

Onde:

- **URL** : endereço selecionado inicialmente quando da abertura da nova janela.
- **Nome** : nome da nova janela, definido pelo programador;
- **Características** - série de opções de configuração da nova janela, se especificados devem estar na mesma *string*, separados por vírgulas e sem conter espaços:

  - toolbar=0 ou 1
  - location=0 ou 1
  - directories=0 ou 1
  - status=0 ou 1
  - menubar=0 ou 1
  - scrollbars=0 ou 1
  - resizable=0 ou 1
  - width=valor inteiro positivo
  - height=valor inteiro positivo

```
Ex:window.open("http://www.geocities.com/CapitolHill/6126/javainde.htm",
                "NovaJanela","toolbar=1,location=1,directories=0,status=1,menubar=1,
                scrollbars=1,resizable=0,width=320,height=240")
```

prompt - Monta uma caixa de entrada de dados, de forma simplificada comparando-se com o objeto *text*.

```
Ex: prompt(label [,valor])
```

onde:

- **label** : texto que aparece ao lado da caixa;
- **valor** : é o conteúdo inicial da caixa;\
- **setTimeout** - Faz com que uma expressão seja avaliada após um determinado tempo (em milissegundos).

```
Ex: ID=setTimeOut(alert('você chegoua aqui, a 10 segundos'),10000)
```

ID - identificador utilizado para o cancelamento de *setTimeOut*

clearTimeout - Cancela *setTimeOut.*

```
Ex: clearTimeOut(ID)
```

Eventos:

- **onLoad** : Ocorre assim que a página *HTML* termina de ser carregada.
- **onUnload** : Ocorre assim que o usuário sai da página atual.

5.10) Reset

Este objeto restaura os campos de um formulário, para seus valores iniciais.

Forma geral:

```
<INPUT TYPE="reset" NAME="nome" VALUE="label" onClick="ação">
```

onde:

- **reset** : Tipo do objeto
- **nome** : Nome definido pelo programador, para futuras referenciações;
- **label :** *String* que será mostrada na face do botão.
- **ação** : Define o que fazer (além de sua função normal) quando clicamos no botão *reset*.

Propriedades:

name - Armazena o nome que foi definido pelo usuário, no campo *NAME*, para o objeto reset.

```
Ex: document.form[0].element[0].name
```

value - Armazena o texto que aparece na face do botão *reset*. Definido no campo *VALUE*.

```
Ex: document.form[0].element[0].value
```

Métodos:

click : simula um clique de mouse no botão *reset*, executando todas as funções a ele associadas, sem que no entanto o usuário tenha realmente clicado neste botão.

```
Ex: resetName.click()
```

Eventos:

onClick - Ocorre quando clicamos o mouse sobre o botão reset. Permite que associemos outra função ao botão reset.

Exemplo:

```
<HTML>
    <HEAD>
        <TITLE>Tutorial JavaScript - Exemplo: ResetButton</TITLE>
    </HEAD>
    <BODY>
        <P><CENTER> <FONT SIZE="+3" FACE="Britannic Bold" COLOR="#0000AF">J</FONT><FONT SIZE="+2" FACE="Britannic Bold" COLOR="#0000AF">ava</FONT><FONT SIZE="+3" FACE="Britannic Bold" COLOR="#0000AF">S</FONT><FONT SIZE="+2" FACE="Britannic Bold" COLOR="#0000AF">cript - </FONT><FONT SIZE="+3" FACE="Britannic Bold" COLOR="#0000AF">G</FONT><FONT SIZE="+2" FACE="Britannic Bold" COLOR="#0000AF">uia de </FONT><FONT SIZE="+3" FACE="Britannic Bold" COLOR="#0000AF">R</FONT><FONT SIZE="+2" FACE="Britannic Bold" COLOR="#0000AF">efer&ecirc;ncia</FONT><FONT SIZE="+2" FACE="Britannic Bold"><BR></FONT>

        <FONT SIZE="-1" FACE="Britannic Bold" COLOR="#000080">&copy; 1997 Anderson Barros Torres</FONT><BR><BR><BR>

        <B><I><FONT SIZE=+4 color=#000000 >E</FONT><FONT SIZE=+3 color=#000000 >xemplo</FONT></I></B>

    </CENTER>
    <BR><BR>
    <FORM action="" method="POST">
        <P>Digite o seu estilo musical:
        <P><INPUT TYPE="text" NAME="estilo" VALUE="" SIZE=30> <INPUT TYPE="RESET" NAME="apaga" VALUE="Limpa" >
    </FORM>
    <BR> <BR> <BR>
    <CENTER>
        <FORM>
            <INPUT TYPE="button" VALUE="Página Anterior" onClick="history.go(-1)">
        </FORM>
```

5.11) Link

HTML permite ligações de uma região de texto (ou imagem) à um outro documento. Nestas páginas, temos visto exemplos dessas ligações: o browser destaca essas regiões e imagens do texto, indicando que são ligações de hipertexto - também chamadas *hypertext links* ou *hiperlinks* ou simplesmente *links*.

Forma geral:

```
<A [ NAME="ancora" ]
    HREF="URL" [TARGET="janela"] [onClick="ação"] [onMouseOver="ação"]>
    Texto explicativo</A>
```

onde:

- **URL** : É o documento destino da ligação hipertexto;
- **âncora** : É o texto ou imagem que servirá de ligação hipertexto .
- **janela** : Nome da janela onde a página será carregada, para o caso de estarmos trabalhando com frames:
  - **"_top"** : Se estivermos trabalhando com frames, a página referenciada pelo link, substituirá a página que criou as frames, ou seja, a página atual, com todas as frames, dará lugar a nova página.
  - **"_self"** : A nova página é carregada na mesma janela do link.
  - **"_blank"** : Abre uma nova seção do *browse* para carregar a página.
- **ação** :Código de resposta ao evento.
- **Texto explicativo** : Texto definido pelo usuário, que aparece na tela de forma destacada.

Eventos:

onClick - Ocorre quando clicamos o mouse sobre o link.

```
Ex: <A HREF="URL qualquer" onClick="alert('você teclou no link!')">Texto</A>
```

onMouseOver - Ocorre quando o mouse passa por cima do link, sem ser clicado.

```
Ex: <A HREF="URL qualquer"  onMouseOver="self.status='Este texto aparecerá na barra de
status quando o mouse estiver posicionado sobre o link'">  Texto</A>
```

Caminhos para o documento destino:

1. **Caminho relativo** : O caminho relativo pode ser usado sempre que queremos fazer referência a um documento que esteja no mesmo servidor do documento atual. Para usar links com caminhos relativos é preciso, portanto, conhecer a estrutura do diretório do servidor no qual estamos trabalhando. Simplificando, é como acessarmos um arquivo que esteja no mesmo diretório, não sendo necessário acrescentar o path.

2. **Caminho absoluto** : Utilizamos caminho absoluto quando desejamos referenciar um documento que esteja em outro servidor. Com a mesma sintaxe, e possível escrever *links* para qualquer servidor *WWW* no mundo. Seria a aplicação do endereço completo da página em questão, como visto no ítem anterior.
3. **Ligações a trechos de documentos** - Além do atributo *href*, que indica um documento destino de uma ligação hipertexto, o elemento A possui um atributo *NAME* que permite indicar um trecho de documento como ponto de chegada de uma ligação hipertexto. Funciona tanto para documentos locais como para os armazenados em outro servidor. O trecho deve ser indicado com o símbolo "#".

```
Ex: JavaInde.htm
    <HTML>
        ...
        <A HREF = "parte1.htm#2">Diferença entre Java e JavaScript</A>
        ...
    </HTML>
```

```
Ex: parte1.htm
    <HTML>
        ...
        <A NAME="1"></A>O que é JavaScript
            JavaScript é...
        <A NAME="2"></A>Diferença entre Java e JavaScript
            A diferença é...
            ...
    </HTML>
```

No exemplo acima, o link de *JavaInde.htm* carrega a página "*parte1.htm*" e automaticamente posiciona o trecho "2" no topo da janela do *browser*.

6.) Palavras reservadas

Existem várias palavras que são reservadas para o *JavaScript* as quais são listadas abaixo. Essas palavras não podem ser utilizadas para identificar variáveis ou funções.

abstract	float	public
boolean	for	return
break	function	short
byte	goto	static
case	if	super
cath	implements	switch
char	import	sinchronized
class	in	this
const	instanceof	throw
continue	int	throws
default	interface	transient
do	long	true
double	native	try
else	new	var
extends	null	void
false	package	while
final	private	with
finally	protected	

7.) Tabela de cores

Ao invés de especificar códigos hexadecimais para utilizar cor em documentos *HTML*, você pode simplesmente utilizar um literal, que especifica o nome da cor para obter o mesmo resultado. A seguir serão mostrados os vários literais que você pode utilizar na especificação de cores:

Exemplo de utilização:

Definindo Background	<BODY BGCOLOR="literal">
Definindo a cor do texto	<BODY TEXT="literal">
Definindo a cor de trechos de textos	<FONT COLOR="literal"...</FONT>
Links, Followed Links, etc.	<BODY ALINK="literal"> etc.

8) Referências