

Netscape  Netcenter™
Brasil

JavaScript – Aplicações Interativas para a Web



ADRIANO GOMES LIMA

**BELO HORIZONTE
2006**

INTRODUÇÃO A LINGUAGEM JAVASCRIPT	9
JAVA E JAVASCRIPT	9
VBSCRIPT E JAVASCRIPT	10
AS VERSÕES DO JAVASCRIPT	10
COMPATIBILIDADE ENTRE BROWSER'S	11
GUIA DE REFERÊNCIA NA INTERNET	11
ORIENTAÇÃO A OBJETOS	11
MANIPULAÇÃO DE OBJETO	13
PROPRIEDADES DE OBJETOS	14
MÉTODOS DE OBJETOS	14
EVENTOS.....	15
MANIPULADORES DE EVENTOS UTILIZADOS.....	16
VARIÁVEIS	19
NOMES DE VARIÁVEIS	19
LITERAIS	22
INTEIROS (INTEGER)	23
PONTO FLUTUANTE.....	23
BOOLEANOS	23
LITERAIS STRING	23
CARACTERES ESPECIAIS	24
EXPRESSÕES	24
OPERADORES.....	25
OPERADORES DE INCREMENTO E DECREMENTO	26
OPERADORES RELACIONAIS	28
OPERADORES RELACIONAIS	28
OPERADORES LÓGICOS.....	28
OPERADOR DE CONCATENAÇÃO DE STRING	29
DECLARAÇÕES	30
OPERADOR NEW	30
PALAVRA-CHAVE THIS.....	30
BREAK.....	30

UTILIZAÇÃO DE COMENTÁRIOS.....	31
VAR	32
DESENVOLVIMENTO DE SCRIPTS.....	33
DESENVOLVENDO SCRIPTS COM O TAG <SCRIPT>	33
DESENVOLVENDO SCRIPTS ATRAVÉS DE UM ARQUIVO EXTERNO.....	34
NOTIFICAÇÃO DE ERROS	36
INSTRUÇÕES BÁSICAS	38
MÉTODO DOCUMENT.WRITE().....	38
MÉTODO ALERT()	39
MÉTODO CONFIRM().....	39
COMANDOS CONDICIONAIS E REPETIÇÃO	41
INSTRUÇÃO WHILE	41
INSTRUÇÃO FOR	42
INSTRUÇÃO FOR...IN	43
IF ... ELSE	44
RETURN.....	47
SWITCH	48
INSTRUÇÃO WITH	49
OBJETO ARGUMENTS	54
UTILIZANDO EVENTOS	56
EVENTO ONBLUR.....	56
EVENTO ONCHANGE	57
EVENTO ONCLICK.....	57
EVENTO ONFOCUS.....	57
EVENTO ONLOAD.....	58
EVENTO ONUNLOAD	58
EVENTO ONMOUSEOVER	58
EVENTO ONMOUSEOUT	59
EVENTO ONMOUSEDOWN	60
EVENTO ONMOUSEUP	60
EVENTO ONKEYPRESS.....	60

EVENTO ONKEYDOWN	60
EVENTO ONKEYUP	60
EVENTO ONSELECT	61
EVENTO ONSUBMIT	61
FUNÇÕES DA LINGUAGEM JAVASCRIPT	63
FUNÇÃO EVAL	63
FUNÇÃO ISNAN	64
FUNÇÃO PARSEFLOAT	65
FUNÇÃO PARSEINT	66
FUNÇÕES PRÉ-PROGRAMADAS	68
IMPRESSÃO DA PÁGINA	68
ADICIONAR AO FAVORITOS	68
JANELA EM MOVIMENTO	69
TEXTO NA BARRA DE STATUS EM MOVIMENTO	70
TABELA DE CORES.....	72
TEXTO EM MOVIMENTO EM UM CAMPO DE FORMULÁRIO	73
OBJETOS PRÉ-CONSTRUÍDOS	76
OBJETO DATE	76
MÉTODOS DO OBJETO DATE	77
OBJETO STRING	78
PROPRIEDADES.....	78
PROPRIEDADES DO OBJETO STRING.....	78
MÉTODOS DO OBJETO STRING.....	78
MÉTODO ANCHOR.....	79
MÉTODO BIG	79
MÉTODO SMALL	80
MÉTODO BOLD	80
MÉTODO ITALICS	81
MÉTODO FIXED	81
MÉTODO STRIKE	82
MÉTODO FONTCOLOR	82

MÉTODO FONTSIZE	83
MÉTODO SUB.....	83
MÉTODO SUP	83
MÉTODO charAT	84
MÉTODO INDEXOF	84
MÉTODO LASTINDEXOF	85
MÉTODO LINK	86
MÉTODO REPLACE	86
MÉTODO SUBSTRING	87
MÉTODO TOLOWERCASE	88
MÉTODO TOUPPERCASE	88
OBJETO IMAGE	89
MÉTODOS DE INTERFACE COM O USUÁRIO.....	92
MÉTODO ALERT	92
MÉTODO CONFIRM	93
MÉTODO PROMPT	94
OBJETO WINDOW	96
PROPRIEDADES DO OBJETO WINDOW/FRAME.....	96
WINDOW.STATUS E DEFAULTSTATUS	97
MÉTODO OPEN	97
MÉTODO CLOSE	98
MÉTODO SETTIMEOUT	98
MÉTODO CLEARTIMEOUT.....	100
TRABALHANDO COM JANELAS	101
ABRINDO PÁGINAS EM FULLSCREEN (Tela Cheia).....	110
O OBJETO MATH	111
PROPRIEDADES DE CÁLCULO DO OBJETO MATH.....	111
MÉTODOS DO OBJETO MATH	112
ABS	112
ACOS	112
ASIN	113

CEIL.....	113
COS	114
EXP.....	114
FLOOR.....	114
LOG	115
MAX.....	115
POW (base,expoente)	116
RANDOM	116
ROUND.....	117
SIN	118
SQRT	118
TAN	118
OBJETO DATE	119
MÉTODOS GET DO OBJETO DATE	119
MÉTODO PARSE E UTC	121
MÉTODOS SET DO OBJETO DATE.....	122
MÉTODO TOGMTSCRING	123
MÉTODO TOLOCALESTRING	123
EXERCÍCIOS	126
OBJETO DOCUMENT	128
PROPRIEDADES DO OBJETO DOCUMENT	128
MÉTODOS DO OBJETO DOCUMENT	132
MÉTODO CLEAR.....	132
MÉTODO CLOSE	133
MÉTODO WRITE E WRITELN.....	134
EXERCÍCIOS	136
OBJETO LINK.....	148
PROPRIEDADES DO OBJETO LINKS	148
UTILIZANDO ARRAYS	149
ARRAY ANCHORS[]	153
ARRAY ELEMENTS[]	154

EXERCÍCIOS:	157
MANIPULANDO FRAMES	161
HIERARQUIA FRAMESET WINDOW	163
OBJETO FORM.....	170
PROPRIEDADES DO OBJETO FORMS.....	170
MÉTODOS DO OBJETO FORM	172
ELEMENTOS DE UM FORMULÁRIO	172
OBJETO TEXT	173
MANIPULADORES DE EVENTO PARA FORMULÁRIOS	173
OBJETO PASSWORD.....	176
OBJETO TEXTAREA	176
OBJETO BUTTON	177
OBJETO SUBMIT	178
OBJETO RESET	179
OBJETO CHECKBOX (Caixa de Verificação).....	179
MANIPULADORES DE EVENTO	181
OBJETO RADIO.....	182
EVITANDO O USO DA TECLA ENTER.....	187
OBJETO LOCATION	189
PROPRIEDADES DO OBJETO LOCATION.....	190
EXERCÍCIOS	192
UTILIZANDO O OBJETO HISTORY	203
PROPRIEDADE.....	203
MÉTODOS BACK E FORWARD	203
UTILIZANDO O OBJETO NAVIGATOR	205
UTILIZANDO O OBJETO NAVIGATOR	205
PROPRIEDADES DO OBJETO NAVIGATOR.....	205
ACESSANDO CÓDIGO-FONTE A PARTIR DE UM LINK.....	207
UTILIZANDO COOKIES	209
Criando Cookies	210
DEPURAÇÃO DE CÓDIGO	219

ISOLAMENTO DE PROBLEMAS	219
ERROS EM TEMPO DE CARREGAMENTO (Load-Time)	220
ERROS EM TEMPO DE EXECUÇÃO (Run-Time)	221
ERROS DE LÓGICA (Logic Errors)	221
ERROS COMUNS EXISTENTES	222
ANALISANDO A ORIGEM DOS ERROS	223
OUTROS ERROS COMUNS	224
RESUMO GERAL DE OBJETOS JAVASCRIPT	225
RESUMO GERAL DE MÉTODOS JAVASCRIPT	228
MÉTODOS DO OBJETO DOCUMENT	228
MÉTODOS DO OBJETO FORM	228
MÉTODOS DO OBJETO DATE	229
MÉTODOS DO OBJETO HISTORY	231
MÉTODOS DO OBJETO MATH	231
MÉTODOS DO OBJETO STRING.....	232
MÉTODOS DE INTERFACE COM O USUÁRIO	234
MÉTODOS DO OBJETO WINDOW	234

INTRODUÇÃO A LINGUAGEM JAVASCRIPT

Desenvolvida pela NETSCAPE, a linguagem JavaScript foi criada para trabalhar com aplicações interativas nas páginas HTML. Esta linguagem teve sua primeira versão desenvolvida para o browser **Netscape Navigator 2.0** e em seguida, atribuído também ao **Internet Explorer 3.0**. A princípio, chamado de **LiveScript**, a Netscape após o sucesso inicial desta linguagem, recebe uma colaboração considerável da **Sun Microsystems**, empresa que há longo tempo vem se dedicando ao desenvolvimento de aplicações para a Internet, como talvez a linguagem mais poderosa da rede, o **Java**, uma linguagem que requer um profundo conhecimento de programação e de seu kit de desenvolvimento, bem diferente do **JavaScript** que não necessita de tanto. Após esta colaboração, podemos dizer que o **JavaScript** é uma linguagem compatível com a linguagem **Java**, por esta razão, a semelhança dos nomes "**JavaScript**".

Conhecida também como uma extensão da linguagem HTML (Linguagem de Marcação de Hipertexto), os comandos JavaScript são embutidos nas páginas HTML e interpretados pelo Browser, ou seja, o JavaScript não possui nenhum procedimento de compilação.

JAVA E JAVASCRIPT

Mesmo sendo uma extensão da linguagem HTML, o JavaScript é uma linguagem baseada na linguagem Java. Com isto, o JavaScript suporta a maior parte das sintaxes e comandos da linguagem Java.

A linguagem Java é usada na criação de objetos e os chamados Applets (aplicativos que são executados em uma página da Internet). Já a linguagem JavaScript, é usada normalmente pelos programadores que fazem uso da

linguagem HTML para controlar dinamicamente o comportamento de objetos nas páginas.

À única limitação da linguagem JavaScript é que ela suporta poucos tipos de dados, e implementa apenas alguns conceitos de orientação a objetos, ao contrário da linguagem Java.

VBSRIPT E JAVASCRIPT

Para não ficar com uma tecnologia terceirizada, a MICROSOFT desenvolveu uma linguagem de scripts similar ao JavaScript denominada VBScript. Uma extensão da conhecida linguagem Visual Basic. A NETSCAPE por sua vez, não implementou esta linguagem em seu Browser, impedindo-o qualquer script que seja desenvolvido na linguagem VBScript de ser executado em seu Browser.

AS VERSÕES DO JAVASCRIPT

Atualmente a versão utilizada do JavaScript é a 1.5 que é suportada pelo Netscape 6.0 e Internet Explorer 5.5, que contém todos os comandos da linguagem JavaScript.

Observe pela tabela a seguir, a relação das versões existentes do JavaScript e a sua aceitação pelos navegadores mais utilizados:

Versão do JAVASCRIPT:	SUPORTADA PELO:
1.0	Netscape 2.0 / Explorer 3.0
1.1	Netscape 3.0 / Explorer 4.0
1.2	Netscape 4.0 e 4.5 / Explorer 4.0
1.3	Netscape 4.6 e 4.7 / Explorer 5.0
1.4	Internet Explorer 5
1.5	Netscape 6.0 / Explorer 5.5

A linguagem JavaScript assim como a linguagem HTML é submetida à uma norma internacional, o ECMA que originou a especificação ECMA-262, que determina o padrão para a linguagem JavaScript, também conhecida como ECMAScript.

COMPATIBILIDADE ENTRE BROWSER'S

É importante que o usuário evite usar comandos JavaScript que foram inseridos nas últimas versões, a não ser que o usuário saiba anteriormente qual o browser são executados. É claro que existem maneiras que garantem que um determinado comando do JavaScript só seja executado em determinado browser, facilitando ainda mais que suas páginas sejam compatíveis com diversas versões de browsers.

Os comandos mais utilizados dentro da linguagem JavaScript são os que fazem parte da sua primeira versão, já aqueles que fazem o tratamento de objetos irão variar de acordo com sua versão.

GUIA DE REFERÊNCIA NA INTERNET

A NETSCAPE, possui um enorme guia para o JavaScript na Internet. Para ter acesso a este guia basta acessar o seguinte endereço:

<http://developer.netscape.com/> → (em inglês)

ORIENTAÇÃO A OBJETOS

Diferente da Linguagem HTML, a linguagem JavaScript corresponde a programação orientada a objetos, isto significa que todos os elementos de uma página da Web são tratados como objetos. Estes objetos são agrupados de acordo com seu tipo ou finalidade. Dentro da linguagem JavaScript, são criados automaticamente objetos que permitem que o usuário possa criar novos objetos de acordo com sua conveniência. Ao ser carregada uma página da Web, é criado um determinado número de objetos JavaScript, com

propriedades e valores próprios que são ajustados pelo conteúdo da própria página. Todos eles seguem uma hierarquia que reflete toda a estrutura de uma página HTML. A linguagem JavaScript pode ser utilizada para a criação de scripts tanto do lado cliente como do lado servidor. Seguindo a hierarquia de objetos da linguagem JavaScript, são criados os seguintes objetos ao ser carregada uma página:

window: O objecto mais acima na hierarquia, contém propriedades que se aplicam a toda a janela. Há também um objecto desta classe para todas as "sub-janelas" de um documento com *frames*

location: Contém as propriedades da URL actual.

history: Contém as propriedades das URLs visitadas anteriormente.

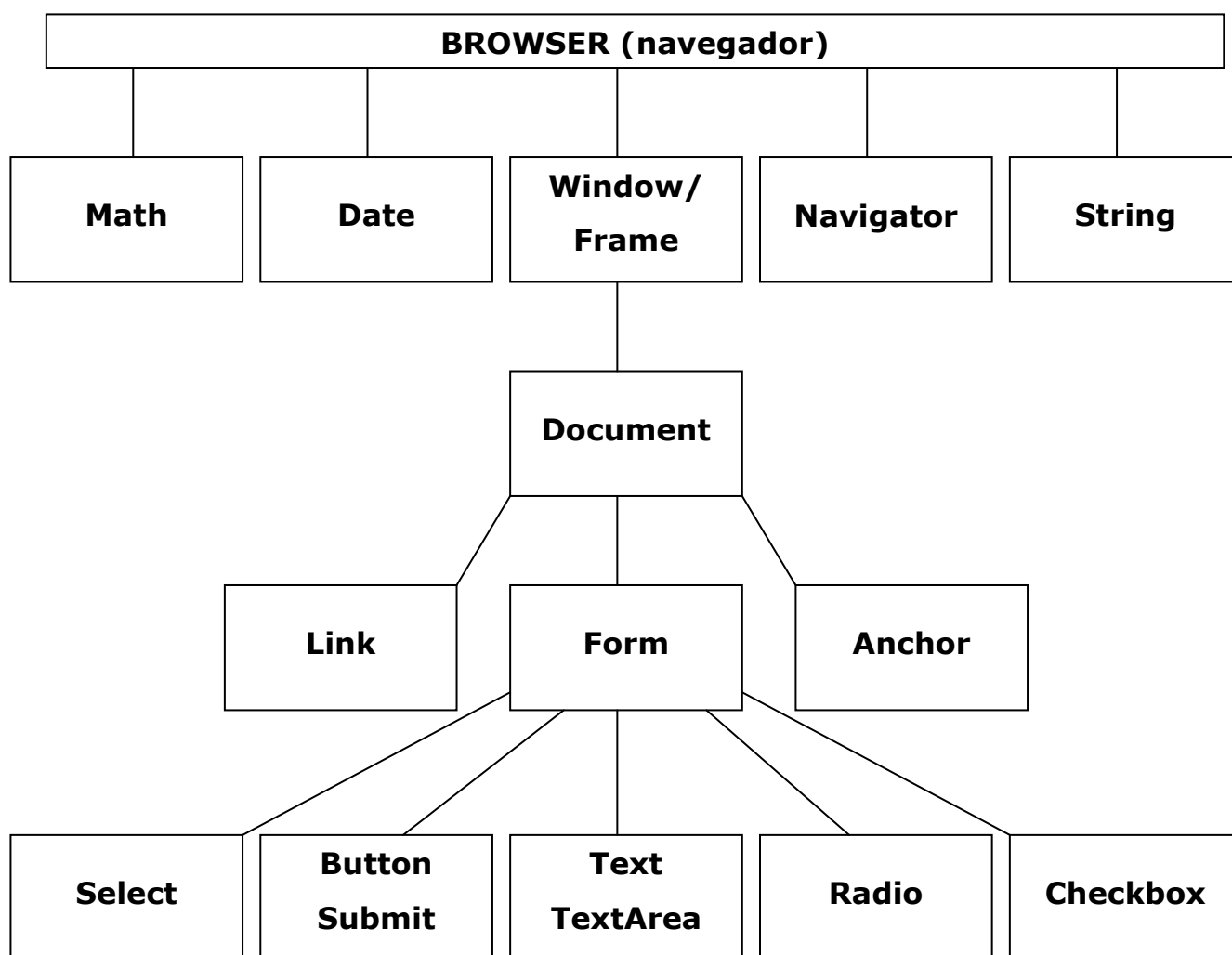
document: Contém as propriedades do documento contido na janela, tais como o seu conteúdo, título, cores, etc

ANOTAÇÕES:

MANIPULAÇÃO DE OBJETO

A linguagem JavaScript manipula vários tipos de objetos através do uso de suas propriedades e métodos. Estes objetos são representados por uma hierarquia, fazendo com que alguns objetos se tornem propriedades de outros, observe pelo exemplo da figura a seguir esta hierarquia formada:

Hierarquia dos Objetos do JavaScript



Conforme visto no organograma apresentado, observe que existem vários objetos e muitos deles pertencem à outros, sendo chamados então de propriedades. Veja pelo exemplo do objeto **FORM** que possui diversas propriedades, sendo este objeto também uma propriedade do objeto **DOCUMENT**.

PROPRIEDADES DE OBJETOS

Cada objeto existente na manipulação do JavaScript possuem propriedades (características). Exemplo, sabemos que um documento HTML possuem título e corpo, estas características do documento podemos chamar de propriedades que existem neste documento.

Estas propriedades existem de dois tipos, algumas são os objetos propriamente ditos enquanto outras não. Um exemplo disto, é o objeto **form** (formulário) que é uma propriedade do objeto **document** (documento), conforme mostrado no organograma apresentado anteriormente. Já a propriedade de título da página (title), é pertencente ao objeto **document** não havendo nenhuma propriedade sobre ela. Concluindo, podemos dizer que a propriedade **form** do objeto **document** é um objeto-filho e o objeto **document** é o objeto-pai. Em geral, as propriedades podem conter valores (string, números, entre outros tipos). A utilização de propriedades se dá acompanhada de seu objeto sendo separados por um ponto apenas. Veja abaixo a sintaxe de utilização de propriedades:

```
nomeObjeto.propriedade
```

MÉTODOS DE OBJETOS

Além das propriedades, os objetos podem conter métodos que são funções pré-definidas pela linguagem JavaScript que irão executar determinada operação. Por exemplo dentro de um documento o usuário poderá utilizar o método de escrever neste documento para exibir um texto qualquer. Os métodos estarão sempre associados à algum objeto presente no documento e cada método faz parte de um objeto específico. Não tente usar métodos em objetos que não o utilizam, isto faz com que a linguagem JavaScript cause erro na execução do script. Na maioria das vezes os métodos são usados para

alterar o valor de uma propriedade ou executar uma tarefa específica. Veja a sintaxe de utilização dos métodos:

```
nomeObjeto.método(argumento)
```

Na sintaxe apresentada, **nomeObjeto** faz referência ao objeto a ser utilizado e o qual sofrerá uma ação do método, já **método** é o nome de identificação do método usado e entre parênteses **(argumento)** é a expressão ou valor opcional que será usada para alterar sobre o objeto.

EVENTOS

Em linguagens orientadas a objetos é comum a manipulação de eventos que é qualquer reação ou ação que executará determinado procedimento, normalmente ocorre por ato executado pelo usuário, como clicar em um botão, selecionar algum objeto e até mesmo pressionar alguma tecla. Resumindo **EVENTOS** são quaisquer ações iniciadas por parte do usuário.

Sua utilização se dá como atributos da linguagem HTML, ou seja dentro dos próprios Tag's HTML. Sua sintaxe tem a seguinte formação:

```
<TAG nomeEvento="Instruções JavaScript">
```

Onde é apresentado **TAG** é uma instrução da linguagem HTML.

Onde é **evento** é o nome do evento gerado da linguagem JavaScript.

Onde "**Instruções JavaScript**" serão as instruções JavaScript à serem executadas. Elas estarão sempre entre aspas.

Caso haja mais de um comando JavaScript a ser executado para o mesmo evento estes deverão estar separados por ponto e vírgula (;), conforme mostrado no exemplo a seguir:

```
<TAG nomeEvento="JavaScript1;JavaScript2;JavaScript3">
```

MANIPULADORES DE EVENTOS UTILIZADOS

EVENTO	MANIPULADOR	DESCRIÇÃO
blur	onBlur	Ocorre quando o usuário retira o foco de um objeto de formulário.
change	onChange	Ocorre quando o usuário muda o valor de um objeto de formulário.
click	onClick	Ocorre quando o usuário clica sobre o objeto.
focus	onFocus	Ocorre quando o usuário focaliza o objeto.
load	onLoad	Ocorre quando o usuário carrega a página.
unload	onUnload	Ocorre quando o usuário abandona a página.
mouseover	onMouseOver	Ocorre quando o ponteiro do mouse passa sobre um link ou âncora. Válidos apenas para hiperlinks.
select	onSelect	Ocorre quando o usuário seleciona um elemento de um formulário.

EVENTO	MANIPULADOR	DESCRIÇÃO
submit	onSubmit	Ocorre quando o usuário envia um formulário.
mouseDown	onMouseDown	Ocorre quando o botão do mouse é pressionado.
mousemove	onMouseMove	Ocorre quando o ponteiro do mouse se movimenta sobre o objeto.
mouseout	onMouseOut	Ocorre quando o ponteiro do mouse afasta de um objeto. Válidos apenas para hiperlinks.
mouseUp	onMouseUp	Ocorre quando o botão do mouse é solto.
keyDown	onKeyDown	Ocorre quando uma tecla é segurada.
keyPress	onKeyPress	Ocorre quando uma tecla é pressionada.
keyUp	onKeyUp	Ocorre quando uma tecla é solta.

Vejamos a utilização dos eventos dentro de alguns TAG's HTML, sem a necessidade de criarmos rotinas separadas para os mesmos. Vejamos o exemplo a seguir:

```
<HTML>
  <HEAD>
    <TITLE>Manipuladores de Eventos</TITLE>
  </HEAD>
  <BODY onLoad="defaultStatus=('Seja Bem Vindo!!!')">
```

No exemplo apresentado anteriormente, foi usado o evento **onLoad** que ocorre quando a página é carregada. Neste evento foi usada a instrução **defaultStatus** que exibe a mensagem **SEJA BEM VINDO!!!** na barra de status do navegador.

Outro exemplo que pode ser aplicado através de um evento, é utilizar o evento **onUnload** que executará alguma ação quando o usuário sair de sua página, baseado no exemplo anterior, inclua no corpo de sua página **<BODY>** a seguinte linha abaixo:

```
<BODY onLoad="defaultStatus=('Seja Bem Vindo!!!')"  
onUnload="alert('Obrigado pela Visita')">
```

Neste exemplo, o evento **onUnload**, faz com que se o usuário abandonar esta página seja entrando em outra, acessando hiperlinks ou até mesmo fechando o browser, é execute a instrução **alert()** que tem a função de exibir uma caixa de diálogo do Windows com a mensagem definida, permitindo ao usuário, pressionar o botão de **OK** para encerra-la.

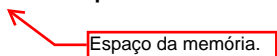
ANOTAÇÕES:

ELEMENTOS DA LINGUAGEM

O JavaScript pode ser diferente em alguns aspectos de outras linguagens, mas nem por isso não deixa de ser uma linguagem de programação, com isto veja os elementos existentes dentro da linguagem.

VARIÁVEIS

Assim como as propriedades que armazenam dados sobre os objetos, é possível com JavaScript a utilização das variáveis que têm a finalidade de armazenar temporariamente informações como textos, valores, datas, entre outros.



Espaço da memória.

O conteúdo de uma variável pode ser simplesmente atribuído ou vir de um resultado de uma ação dada de uma expressão ou função. Veja alguns exemplos.

```
nome="ADRIANO LIMA"
```

```
idade="25"
```

```
Soma=2002-25
```

```
tempo=Date( )
```

No JavaScript temos 3 tipos de variáveis:

- let;
- const; (Valor NÃO pode ser alterado)
- var; (DESCONTINUADO / NÃO USAR)

NOMES DE VARIÁVEIS

O nome de uma variável poderá iniciar-se com uma letra ou através do caractere "underscore" seguido de letras ou números. Outra semelhança do JavaScript com outras linguagens é a diferenciação de de letras minúsculas e maiúsculas. Veja alguns nomes válidos para variáveis:

nome

_senac

escola

Na linguagem JavaScript existem dois tipos de variáveis que são:

GLOBAIS → usadas em qualquer parte de uma aplicação.

LOCAIS → usadas somente na rotina que foi desenvolvida.

Para criar variáveis locais, é necessário que o usuário utilize a palavra-chave **var**. Veja a declaração de uma variável local:

```
var nome="ADRIANO LIMA"  
var soma=2002-25
```

As variáveis definidas fora de uma função sempre estão disponíveis para todas as funções dentro do script que estão na mesma página. Estas variáveis são referenciadas como variáveis **globais**. As variáveis que são definidas dentro de função, também são globais, desde que não seja utilizado a instrução **var** em sua declaração.

Caso o usuário declare uma variável dentro de uma função através da instrução **var**, esta variável passa a ser apenas local, ou seja, são utilizadas apenas para aquela função onde foi declarada.

É bom saber que, as variáveis globais ficam na memória mesmo após a execução do script, estas variáveis somente são liberadas da memória quando o documento é descarregado.

As variáveis podem ser declaradas também separadas por vírgula, da seguinte maneira:

```
var nome, endereco, telefone;
```

ou

```
var nome;  
var endereco;  
var telefone;
```

Outro exemplo prático de atribuição, é atribuir um mesmo valor a mais de uma variável, da seguinte maneira:

```
var campo1 = campo2 = campo3 = 5
```

No exemplo anterior, foi atribuído o número 5 nas variáveis **campo1**, **campo2** e **campo3**.

Veja pelo exemplo do código abaixo como manipular variáveis através da linguagem JavaScript:

```
<HTML>  
<HEAD>  
  <TITLE>CÁLCULOS</TITLE>  
</HEAD>  
<BODY>  
  <script>  
    valor=30  
    document.write("Resultado do cálculo ",(10*2)+valor)  
  </script>
```

Neste exemplo foi definida a variável **valor** que armazena o valor 30 em seu conteúdo, em seguida, através do objeto **document** foi usado o método **write** que escreverá no corpo da página o texto **Resultado do cálculo** e em seguida o resultado da expressão **(10*2)+valor** que resultará em 50.

Caso tenha que executar outro cálculo abaixo do primeiro, utilize o tag HTML **
** após o cálculo, separando-o com vírgula e entre aspas. Veja o exemplo abaixo:

```
document.write("Resultado do cálculo ",(10*2)+valor,"<BR>")
document.write("A soma de 5+2 é: ",5+2)
```

O resultado iria apresentar os valores dos cálculos um abaixo do outro, veja agora o mesmo exemplo colocando o resultado em negrito através do tag HTML ****.

```
document.write("A soma de 5+2 é: ", "<b>", 5+2, "</b>")
```

lembre-se que estas instruções deverão estar entre as tag's HTML **<SCRIPT>** e **</SCRIPT>**. No caso de querer utilizar alguma instrução HTML, atribua-as entre aspas como propriedade do método conforme exemplo mostrado anteriormente.

LITERAIS

São representações de números ou strings, estas informações são fixas, bem diferente das variáveis, não podem ser alteradas. As variáveis são criadas na execução do programa, já os literais fazem parte do código-fonte. Veja abaixo alguns exemplos de literais:

52	➔	Número inteiro.
2.1518	➔	Número de ponto flutuante.
"Adriano Gomes Lima"	➔	Texto.

Existem vários tipos de literais, eis os existentes:

INTEIROS (INTEGER)

Representam números positivos, negativos ou fracionários. Exemplo:

A=500

B=0.52

C=-32

PONTO FLUTUANTE

Este literal também chamado de **notação científica** é representado da seguinte maneira:

2.34e4

O número **2.34** é multiplicado por dez à quarta potência, ou **2.34*10000**.

BOOLEANOS

Este tipo de literal representa valores lógicos que podem ser:

Com letra minúscula

TRUE ou **1**

FALSE ou **0**

LITERAIS STRING

Este literal representa qualquer cadeia de caracteres envolvida por aspas ou apóstrofo. Veja abaixo alguns exemplos:

"Adriano Lima"

`CFP-INFORMÁTICA`

" "

"500"

Template Literals ou Template String
O uso da crase permite escrever texto de forma a "pular" de linha sem a necessidade de declarar uma sintaxe para isso. Ex:

```
`Começo nessa linha e vou para outra  
sem a necessidade de uma sintaxe específica`
```

Para declarar uma variável dentro do template basta colocar a variável dentro da chaves seguida do cifrão dólar. Ex:

```
const myAge = 30  
  
const myString = `Minha idade é ${myAge}`  
  
console.log(myString)
```

Mesmo sendo número, as aspas fazem com que o literal seja uma string.

CARACTERES ESPECIAIS

Estes caracteres são especificados dentro de uma string. Veja na tabela abaixo estes caracteres e sua descrição:

Caractere	Descrição
\n	Insere uma quebra de linha.
\t	Insere uma tabulação.
\r	Insere um retorno.
\f	Insere um caractere de barra.
\t	Tabulação.
\'	Apóstrofo.
\"	Aspas.
\\	Barra Invertida.
\XXX	Caractere representado pela codificação Latin-1 . Exemplo \251 representa o caractere de copyright ©.

OBS: As letras dos operadores devem apresentar-se em letras minúsculas.

EXPRESSÕES

Uma expressão é normalmente uma combinação de variáveis, literais, métodos, funções e operadores que retornam um valor qualquer. Usada para atribuir valores em variáveis ou até mesmo para testá-la e atribuir uma ação específica com base do seu resultado. Veja o exemplo da criação de uma variável numérica:

```
numero=5
```


Neste exemplo fora atribuído o valor número **5** à variável chamada **numero**. Esta atribuição de valor pode ser considerada uma expressão. Veja outro exemplo de expressão:

```
numero2=5*2
```

Neste exemplo foi atribuído o resultado da expressão **5*2** à variável chamada **numero2** que neste caso é **10**. Vejamos outro exemplo em outra situação:

```
If numero+numero2 > 10
```

Já neste exemplo foi usado a instrução condicional **if** que testa o resultado da expressão **numero+numero2** e em seguida o compara com o número 10. Se o resultado da expressão for superior à 10, a mesma retornará o valor booleano **TRUE**, em caso contrário o valor passa a ser **FALSE**.

OPERADORES

Os operadores são utilizados em expressões para comparar seus conteúdos. O operador mais utilizado em uma linguagem de programação é o de atribuição conhecido como sinal de igualdade (**=**). Veja abaixo alguns exemplos de sua utilização:

```
X=50
```

```
X=30*5/2
```

```
X=Y
```

Além deste caractere de atribuição, é possível a utilização de outros operadores como mostrado à seguir:

x += y

x -= y **** : Exponencial**

x *= y

x /= y

x %=y **Resto**

o operador typeof informa qual o tipo do elemento. Ex.:

```
const conta = 2344  
console.log(typeof conta) //informa que é do tipo number
```

o operador delete permite deletar um parâmetro específico. Ex.:

```
const myObject = {  
  name: "João",  
  age: 25,  
  address: "Rua Doze, 10"}  
  
delete myObject.address //Deleta somente o endereço.
```

Analisando os operadores apresentados, podemos defini-los de outra maneira, conforme mostrado abaixo:

x = x + y

x = x - y

x = x * y

x = x / y

x = x % y

Veja a relação dos operadores que são utilizados na linguagem JavaScript:

ARITMÉTICOS

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo

OBS: O operador **Módulo** retorna o resto da divisão do operandos um e dois.

OPERADORES DE INCREMENTO E DECREMENTO

Além dos operadores apresentados anteriormente, existe outro tipo de operador que têm a tarefa de aumentar e/ou diminuir o valor do operando. O

operador incremental é representado pelo duplo sinal de adição “++”, já o operador decremental é representado pelo duplo sinal de subtração “--”. Veja a seguir alguns exemplos:

```
variável++ ou ++variável  
variável-- ou --variável
```

Sempre que o operador for colocado antes do operando, é incrementado ou decrementado o operando e o valor é atualizado. Em caso contrário, será retornado o valor do operando para depois ocorrer o incremento ou decremento. Observe um exemplo:

```
x = 10  
A = x++
```

Neste exemplo, foi atribuído à variável **x** o valor numérico 10, e em seguida é atribuído à variável **A** o valor de **x** incrementado, neste caso a variável **A** recebe o valor numérico **11**. analisando o caso contrário:

```
x = 10  
A = ++x
```

Já neste caso **x** é incrementado e o novo valor é atribuído em **A**. O mesmo ocorre para o operador de decremento.

NULL & UNDEFINED

- O undefined é um erro de declaração, onde o valor de uma variável não é definido ocasionando erro.
- O null é um valor vazio dado a uma variável em certos casos e evitam erro.

ARRAY

É um vetor que pode armazenar diferentes tipos de variáveis. E a posição dos elementos no array começa em ZERO. Ex:

```
const myArray = [20, 30, "Olá JS", {name: "Zequinha", age: 26}]
```

```
const users = [  
  {name: "Zequinha", age: 26, id: null },  
  {name: "Joana", age: 45, id: 458779}]
```

```
/*Consultando um elemento na array*/  
console.log(users[0])
```

```
/*Alterando um elemento na array*/  
array = [23, 45, 458]  
array[2] = 687  
console.log(array)  
users[1].name = "Thais"  
console.log(users)
```

OPERADORES RELACIONAIS

Estes operadores comparam o conteúdo dos operandos e retornam um valor booleano **TRUE** ou **FALSE**, baseado no resultado da comparação. Veja a relação destes operadores.

Operador	Descrição
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
=	Atribuição
==	Igualdade
===	Igual e mesmo tipo
!=	Diferente

!== Totalmente diferente, valor e tipo

valor

Somente será retornado TRUE se a comparação satisfizer a condição.

OPERADORES LÓGICOS

Para estes operadores, são exigidos valores booleanos, como operandos, e será retornado um valor lógico.

Operador	Descrição
&& ou AND	E
ou OR	OU
! ou NOT	NÃO

O operador "&&" retorna TRUE somente se todas as expressões forem verdadeiras.

O operador "||" retorna TRUE se uma das expressões forem verdadeiras. Se as duas forem falsas, será retornado FALSE.

O operador "!" nega uma expressão. Se for verdadeira, será retornado FALSE. Se for falsa, será retornado o valor TRUE.

OPERADOR DE CONCATENAÇÃO DE STRING

Para concatenar duas ou mais string's, basta utilizar o sinal de adição, veja um exemplo:

```
A = "ADRIANO"
```

```
B = "LIMA"
```

```
C=A+B
```

```
D="Senac"+"Minas"
```

OBJECT (SEMELHANTE AO DICIONÁRIO NO PYTHON)

OBS: no exemplo abaixo temos um objeto dentro de outro objeto.

```
const user = {  
  name: "José",  
  age: 45,  
  address: {  
    street: "Rua Hum",  
    number: 100,  
    city: "São Paulo"  
  }  
}
```

```
/*Visualizando a informação só do nome*/  
console.log(user.name)
```

```
/*Visualizando a informação só da city*/  
console.log(user.address.city)
```

```
/*Alterando o número do endereço*/  
user.address.number = 332
```

OBS: APESAR DO OBJETO SER DO TIPO "CONST", É POSSÍVEL FAZER ALTERAÇÕES PONTUAIS (CONFORME O EXEMPLO ACIMA).

DECLARAÇÕES

Vejamos agora uma relação das declarações existentes na linguagem JavaScript que são utilizadas na criação da estrutura lógica de um programa. Normalmente estas declarações são atribuídas às tomadas de decisões, laços repetitivos e funções.

OPERADOR NEW

Este operador irá permitir que o usuário crie uma nova instância de um objeto definido. Veja sua sintaxe:

```
NomeObjeto=new Tipo(parâmetros)
```

PALAVRA-CHAVE THIS

Esta palavra-chave é utilizado para fazer referência ao objeto corrente. Veja sua sintaxe:

```
this.propriedade
```

BREAK

Esta instrução desvia o JavaScript de uma estrutura controlada e continua sua execução na primeira linha após o bloco da instrução onde foi encontrado. Esta instrução pode ser utilizada em estruturas baseadas nas seguintes intruções:

for

for...in

while

UTILIZAÇÃO DE COMENTÁRIOS

Assim como qualquer outra linguagem de programação, a linguagem JavaScript faz o uso de comentários que irão permitir ao programador inserir anotações referentes ao seu desenvolvimento ou explicar determinada operação de seu script. Estes comentários na execução do script, são ignorados pelo interpretador (browser). Veja a sintaxe do uso de comentários na linguagem JavaScript:

```
// Comentário de uma linha de texto.
```

```
/* Comentário de várias linhas de texto,  
continuação do comentário de várias linhas */
```

Conforme visto no exemplo anterior, quando o comentário for um pequeno texto que irá ocupar uma linha o usuário fará o uso da instrução `“//”` caso o mesmo irá compor mais linhas de texto no início do comentário utiliza-se a instrução `“/*”`, e após a última linha de texto encerra-se com a instrução `“*/”`.

Além destes comentários é recomendável que utilize antes de iniciar um script o Tag de comentário da Linguagem HTML, que irá permitir que navegadores já ultrapassados no sentido de não reconhecer as instruções JavaScript, possam ignorar estas instruções evitando erros futuros. A sintaxe de utilização do Tag de comentário em um script é formada da seguinte forma:

```
<!--Início do JavaScript  
Instruções  
//Término do JavaScript -->
```

Observe que no final do script, foi definido um comentário de uma linha de texto no JavaScript, encerrando-se com o Tag de Fechamento da Linguagem

HTML. O comentário do JavaScript somente foi necessário em razão de haver um texto de comentário, caso contrário, bastaria o Tag de Comentário do HTML.

VAR

A palavra-chave **var** declara o nome de uma variável e caso queira o usuário poderá atribuir um valor à mesma. O conteúdo da variável poderá ser visualizado por uma função ou por outras variáveis, declaradas fora da função na qual foi criada. Veja alguns exemplos:

```
var nome  
var endereço="R. Tupinambás 1038"
```

ANOTAÇÕES:

DESENVOLVIMENTO DE SCRIPTS

As instruções da linguagem JavaScript podem ser escritas em qualquer editor ASCII, como por exemplo, o Bloco de Notas do Windows e até mesmo o Edit do MS-DOS, sendo que seu arquivo deverá ser salvo com a extensão HTML ou .JS. Para visualizar a execução deste script, basta acessá-lo através do browser.

Quando se desenvolve scripts em uma página HTML, é necessário que o usuário os delimite através do Tag **<SCRIPT>** ou utilize-os como manipuladores de eventos através de alguns Tag's HTML. Outra maneira é criar um arquivo externo para ser chamado à partir de uma página HTML. Este arquivo separado deverá possuir a extensão **.JS**.

DESENVOLVENDO SCRIPTS COM O TAG <SCRIPT>

Com o Tag <SCRIPT> é possível ao usuário incorporar seus scripts dentro de uma página HTML. Veja a sintaxe de utilização deste Tag:

```
<SCRIPT>
instruções do JavaScript...
</SCRIPT>
```

Em alguns casos é possível observar o tag SCRIPT com o seguinte atributo:

```
<SCRIPT LANGUAGE="JAVASCRIPT">
instruções do JavaScript...
</SCRIPT>
```

O atributo **LANGUAGE** é de uso opcional, este atributo irá especificar a versão da linguagem JavaScript utilizado. A sua omissão assume qualquer instrução do JavaScript independente da sua versão.

Se for especificada à versão conforme exemplo abaixo, apenas os browsers que sejam compatíveis com a versão específica poderão executar este script:

```
<SCRIPT LANGUAGE="JAVASCRIPT1.3">
instruções do JavaScript...
</SCRIPT>
```

DESENVOLVENDO SCRIPTS ATRAVÉS DE UM ARQUIVO EXTERNO

As instruções da linguagem JavaScript podem ser executadas de um arquivo externo. Com isto, o usuário não precisará repetir instruções várias vezes, isto, facilita a manutenção do código desenvolvido e a reutilização do mesmo.

Para isto, o usuário deverá criar o código em qualquer editor ASCII da mesma forma que se cria uma página HTML, e ao salvá-lo, o usuário deverá atribuir ao seu nome a extensão **.JS**.

Neste arquivo o usuário não precisará utilizar o Tag HTML para delimitar suas instruções.

Para que uma página HTML possa processar as instruções desenvolvidas no arquivo externo, basta utilizar o seguinte parâmetro na página HTML:

```
<SCRIPT LANGUAGE="JAVASCRIPT" SRC="NomeArquivo.js"></SCRIPT>
```

pode ser colocado no head ou no body no código do HTML.

Veja pela figura abaixo o ícone que representa um arquivo externo com instruções da linguagem JavaScript:

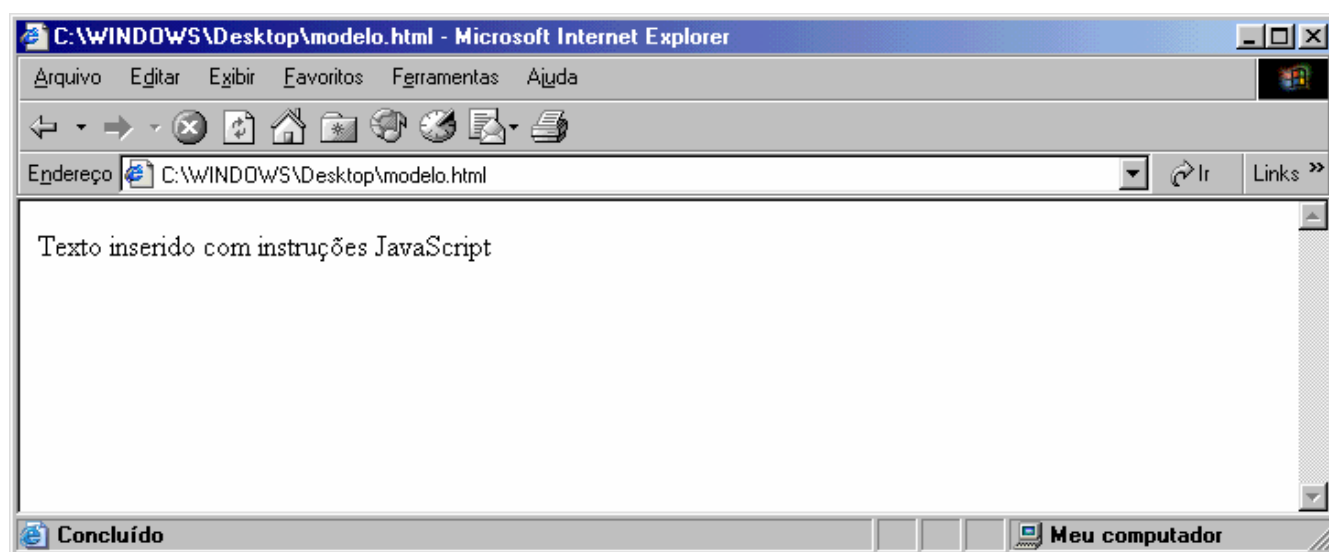


Quando colocar no "head", usa-se o "defer" para dar tempo de carregar a página.
<script src="/scripts.js" defer></script>

Conforme dito anteriormente, a linguagem JavaScript é interpretada pelo browser e que seu código é embutido dentro do código HTML entre os tag's **<SCRIPT>** e **</SCRIPT>** ou através de um arquivo externo que possua a extensão **.JS**. Observe o uso de algumas ações que o JavaScript pode desenvolver através da figura a seguir:



Caixa de diálogo criada por uma instrução da Linguagem JavaScript.

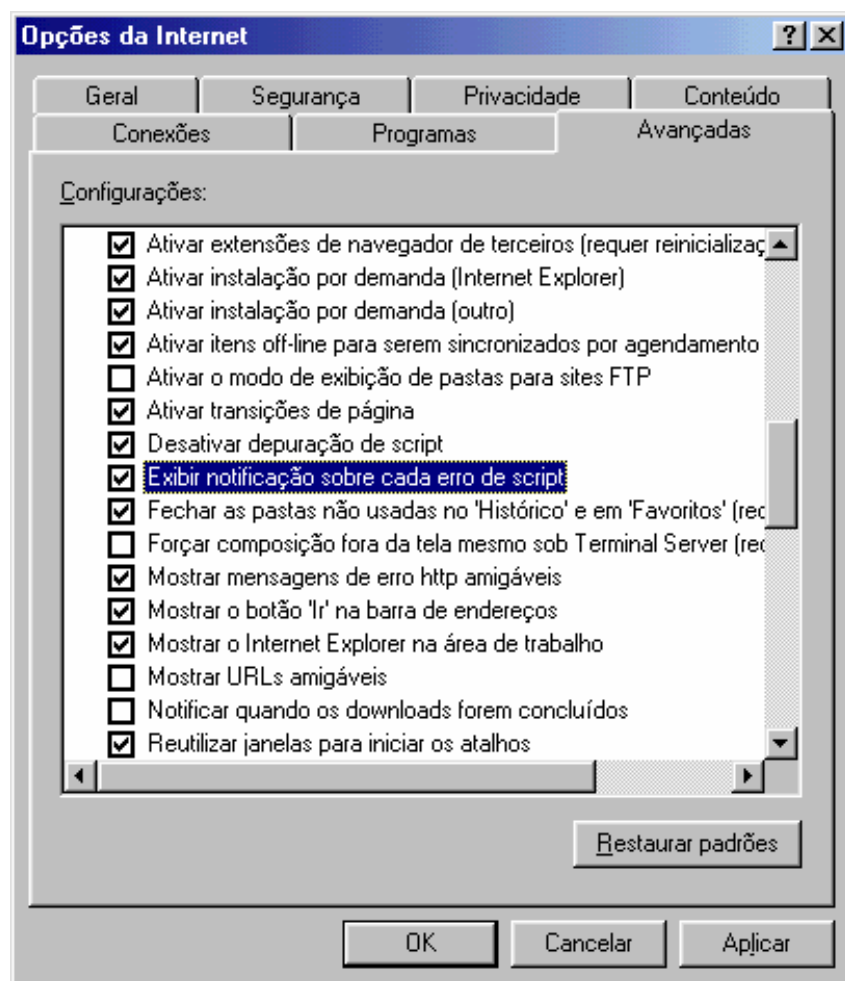


Texto inserido no corpo de uma página através de instruções da Linguagem JavaScript.

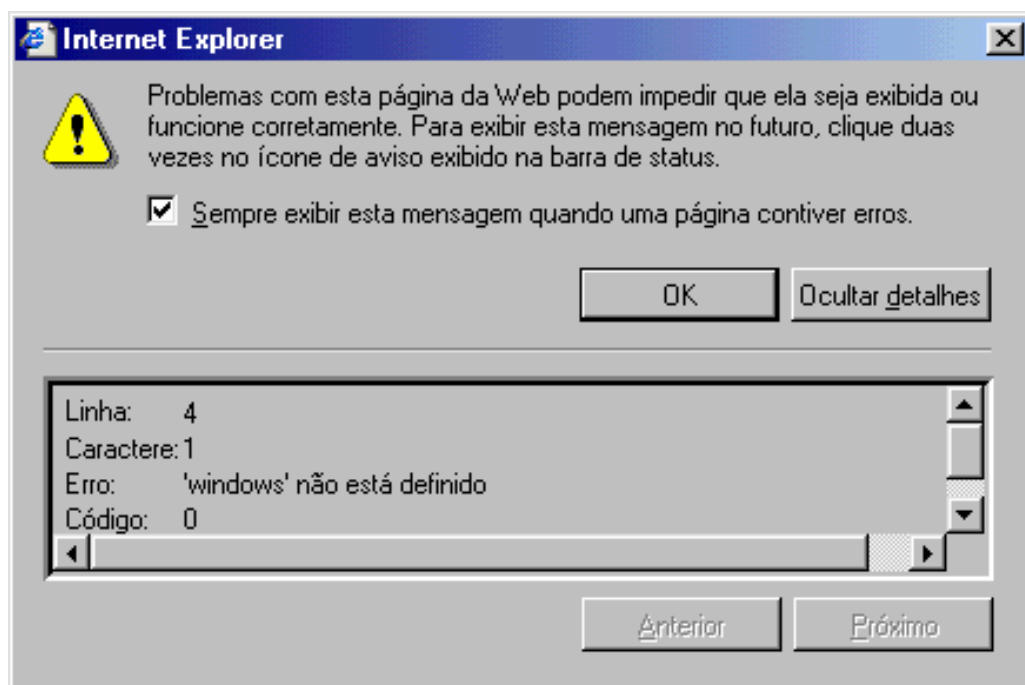
NOTIFICAÇÃO DE ERROS

Além dos comentários, que irão evitar que os navegadores mais antigos exibam algum código JavaScript que não é reconhecido, durante o desenvolvimento e execução do código o programador precisará saber a origem de qualquer erro existente no seu programa. Para isto, é possível configurar o browser para exibir uma notificação de erro de script durante seus testes.

Utilizando o Internet Explorer o usuário poderá acessar o menu **Ferramentas** e em seguida, **Opções da Internet** e logo mais acessar a guia **Avançada** e selecionar a opção **Exibir Notificação sobre cada erro de script** conforme mostrado na figura a seguir:



Feito isto, qualquer erro existente em seu programa será notificado pelo browser de acordo com a figura abaixo:



ANOTAÇÕES:

INSTRUÇÕES BÁSICAS

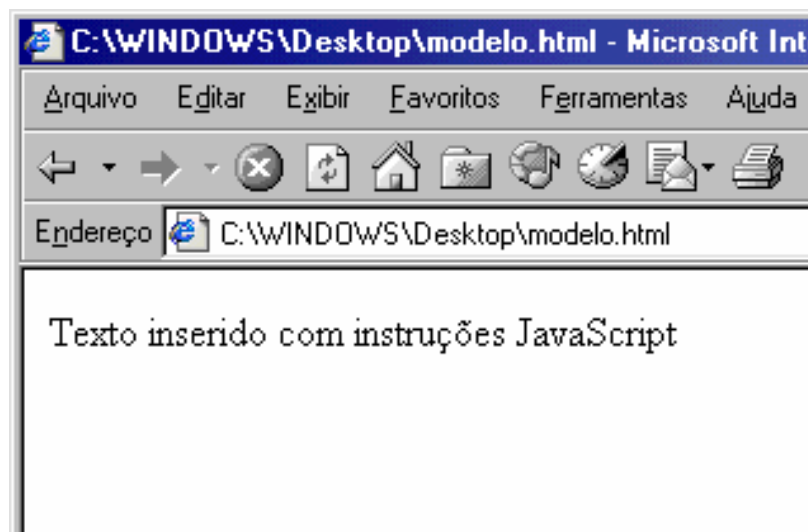
Neste ponto do treinamento o usuário irá conhecer algumas instruções que irão facilitar o entendimento e a construção de um programa em JavaScript. Serão apresentados comandos que permitirão a manipulação e inserção de objetos em documento HTML.

MÉTODO DOCUMENT.WRITE()

Esta instrução na realidade segue a sintaxe de ponto da linguagem JavaScript, uma das maneiras de seguir a hierarquia dos objetos presentes na linguagem. Nesta linha de comando temos o método **write()** que é pertencente ao objeto **document** que retrata o documento como um todo. Vejamos um exemplo de sua utilização através do código apresentado a seguir:

```
document.write("Texto inserido com instruções JavaScript");
```

Através do exemplo apresentado anteriormente foi dado como argumento do método **write** a string apresentada, determinando-o a se apresentar no corpo do documento, observe pelo exemplo da figura a seguir:



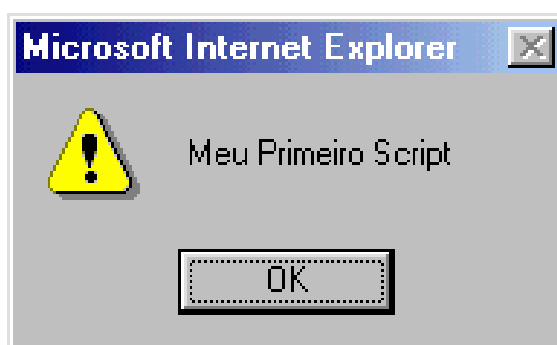
MÉTODO ALERT()

A finalidade deste método é emitir uma caixa de diálogo do windows conforme mostrado no exemplo passado com uma mensagem e um botão de OK. Este método é pertencente ao objeto **window** do JavaScript, porém seu uso com a sintaxe de ponto é opcional, assim sendo observe a sintaxe de seu funcionamento e o exemplo da próxima figura:

```
window.alert("Meu Primeiro Script");
```

ou

```
alert("Meu Primeiro Script");
```



MÉTODO CONFIRM()

Uma outra alternativa além do método **alert()** está no método **confirm()** que exibe uma caixa de diálogo e os botões de OK e CANCELAR. Caso seja pressionado o botão **OK**, o método retornará o valor booleano **TRUE** e pressionado o botão **CANCELAR**, é retornado o valor **FALSE**.

Com isto, o usuário poderá determinar uma tomada de decisão dentro de seu script. Assim como o método **alert()**, o método **confirm** é pertencente ao objeto **window**, sendo seu uso opcional, observe sua sintaxe abaixo e veja o exemplo da caixa de diálogo presente na figura a seguir:

```
window.confirm("Tem Certeza??");
```

ou

```
confirm("Tem Certeza??");
```



O método GET permite obter informações de elementos do HTML. E o documento HTML é chamado no JS por "document". Com os seguintes métodos:

- getElementById -> Trás UM elemento pelo ID.
`document.getElementById("nomeIDdoElemento")`
- getElementsByClassName -> Trás TODOS os elementos com essa classe.
`document.getElementsByClassName("nomeDaClasse")`
- getElementsByTagName -> Trás TODOS os elementos com essa TAG (p, h1, h2...)
`document.getElementsByTagName("nomeDaTag")`
- getElementByName -> Trás TODOS os elementos com esse NAME.
`document.getElementsByName("nomeDoElemento")`
- querySelector -> Trás UM elemento (pode ser TAG, Classe, ID, Name), o PRIMEIRO que encontrar. No exemplo abaixo vamos obter o primeiro parágrafo.
`document.querySelector("p")`
Agora queremos pegar um botão da classe abacate:
`document.querySelector("button.abacate")`
Se fosse só a classe:
`document.querySelector(".abacate")`
Se fosse um ID:
`document.querySelector("#main-put")`
- querySelectorAll -> Trás TODOS os elementos que encontrar.
`document.querySelectorAll("p")`

Alterando textos & HTML e CSS(estilos) no JavaScript

```
const element = document.querySelector(".paragraph-js")

/*ALTERANDO O TEXTO, pode ser usado qualquer um dos exemplos abaixo:
element.textContent = "O novo <b>texto</b> é esse"
OU
element.innerHTML = "O novo <b>texto</b> é esse"

console.log(element.textContent) // SÓ HTML
console.log(element.innerText) // LEVA EM CONTA O CSS. Ex.: Se implementarmos o comando no CSS p{visibility: hidden;}. Não vamos visualizar o texto do parágrafo.
console.log(element.innerHTML) // TRÁS TUDO INCLUSIVE OS ELEMENTO DO HTML. Ex.: TAG's <b></b>

/*ALTERANDO O ESTILO CSS
const button = document.querySelector(".main-button")
button.style.backgroundColor = "#852394" Nesse caso, o estilo no CSS é background-color, para a sintaxe no JS devemos TIRAR o "-" e colocar a letra em Maiúsculo.
```


COMANDOS CONDICIONAIS E REPETIÇÃO

INSTRUÇÃO WHILE

A instrução **while** realiza uma ação enquanto determinada condição for satisfeita. Sua sintaxe básica é:

```
while (expressão) {  
    comandos  
}
```

```
Do While:  
Primeiro faz uma ação pra depois verificar a condição:  
let i = 0  
  
do {  
    i++  
    console.log(i)  
} while (i < 10)
```

Veja no exemplo seguinte a utilização do laço **while** que é repetido por total de 10 vezes:

```
num=0;  
while(num<10){  
    document.write("Número: "+num+"<br>");  
    num++;  
}
```

Neste exemplo, foi definido a variável **num** com o valor zero, em seguida este valor é comparado na condição **while** que é **num<10**, que caso seja verdadeira a variável **num** é incrementada com mais 1 e exibido na tela, quando ele atinge o valor limite que é 10, o código é desviado para a primeira condição após o laço. Veja agora um exemplo prático de sua utilização:

```
<script>  
function condicao(){  
while(form1.nome.value==""){  
    alert("Favor Preencher o campo");  
    form1.nome.value=prompt("Digite seu nome agora","");  
}
```

```
}
alert("Obrigado, "+form1.nome.value);
}
</script>
<pre>
<form name="form1">
Nome:
<input type="text" name="nome" onBlur="condicao(this.value)">
</script>
```

INSTRUÇÃO FOR

A instrução **for** realiza uma ação até que determinada condição seja satisfeita. Sua sintaxe básica é:

```
for (início;condição;incremento) {
    comandos
}
```

O **início** determina o valor inicial do laço **for**. Normalmente é 0 ou 1, porém poderá ser especificado qualquer outro valor. O valor especificado é atribuído em uma variável, por exemplo **i=0**, **count=1**.

A **condição** determina a expressão que irá controlar o número de repetições do laço. Enquanto esta expressão for verdadeira, o laço continuará, caso o laço seja falso, o laço terminará. Por exemplo: **i<20**. Enquanto o valor de **i** for menor que 20, a condição é verdadeira.

O **incremento** determina como o laço irá contar, de 1 em 1, 2 em 2, 5 em 5, 10 em 10, enfim. Exemplo: **i++**. Será aumentado o valor da variável **i** a cada repetição. Diferente de todas as outras linguagens, em JavaScript, a instrução **for**, utiliza ponto e vírgula para separar os argumentos ao invés de vírgula.

Vejamos um exemplo prático de utilização do laço **for** que conta valores de 1 até 10, acrescentando um valor de cada vez:

```
<script>
for (i=1 ; i<=10 ; i++){
document.write("número: " + i + "<br>");
}
</script>
```

for of: é utilizado para iterar nos elementos de uma String.

```
const myName = "Antonio"

for (const letter of myName) {
  console.log(letter)
}
```

i é a variável utilizada para armazenar o contador do laço **for**. Logo mais, o laço inicia com 1 e continua até 10. A expressão condicional é **i<10** (i é menor que 10).

Enquanto a expressão condicional for verdadeira, o laço for continua. No incremento, a cada repetição do laço será adicionado a variável **i** mais 1. Veja outro exemplo:

```
<script>
for (i=1 ; i<=10 ; i=i+2){
document.write("iteração: "+i+"<br>");
}
</script>
```

INSTRUÇÃO FOR...IN

Podemos dizer que o laço **for...in** é uma versão especial da instrução **for**, que é usada para saber os nomes de propriedades e conteúdos das propriedades de objetos no JavaScript. Esta instrução é muito usada na depuração de códigos. Por exemplo, caso uma parte do código JavaScript não esteja funcionando corretamente e existe uma suspeita de que existe uma falha do objeto JavaScript, o usuário poderá usar **for...in** para examinar as propriedades do objeto usado. Sua sintaxe é formada da seguinte maneira:

Usado para obter a CHAVE de um OBJECT, mas também é possível usar a sua estrutura e obter o VALOR de um OBJECT:

```
const users = {name: 'Rodolfo', age: 30, street: 'Manga chupada'}
```

```
for (const key in users) {
  console.log(key)
}
```

```
for (const key in users) {
  console.log(`${key} : ${users[key]}`)
}
```

```
for (variável in objeto){  
    instruções;  
}
```

Veja pelo exemplo a seguir, o uso do laço **for...in** para determinar as propriedades do objeto **navigator** que contém informações sobre o browser. Ao listar todas as propriedades do objeto, o laço automaticamente se encerrará:

```
for (teste in document){  
    alert(teste);  
}
```

Neste laço, foi criado uma variável chamada **teste** que foi atribuído o conteúdo do objeto **document**. Dentro do laço foi executado a instrução **alert** que exibe o conteúdo da variável **teste**.

IF ... ELSE

Realiza determinada ação de acordo com uma condição. Sua sintaxe básica é:

```
if (condição) {  
    comandos  
} else {  
    comandos  
}
```

Caso haja mais de uma condição a ser avaliada pode-se fazer o uso da instrução **ELSE IF**. Observe sua sintaxe:

```

if (condição) {
    comandos
} else if (condição2) {
    comandos
} else {
    comandos
}

```

Veja um exemplo da utilização da instrução **if**:

```

<script>
function condicao(){
if(form1.nome.value==""){
    alert("Favor Preencher o campo");
    return form1.nome.focus();
}
}
</script>
<pre>
<form name="form1">
Nome:
<input type="text" name="nome" onBlur="condicao(this.value)">

```

Neste exemplo foi criada uma função que testará o conteúdo da variável **nome**, que ocorrerá assim que o usuário retirar o foco do campo, caso o valor da variável esteja vazio, será exibida uma mensagem de alerta informando para o preenchimento e em seguida o foco será retornado para o campo **nome**. Em caso contrário, o script continuará seu processamento normal até o fim. Criaremos agora uma condição alternativa para quando o campo não estiver vazio, observe:

```

<script>
function condicao(){
if(form1.nome.value==""){
    alert("Favor Preencher o campo");
    return form1.nome.focus();
}else{
    alert("Obrigado, "+form1.nome.value);
    return form1.nome.select();
}
}
</script>

<pre>
<form name="form1">
Nome:
<input type="text" name="nome" onBlur="condicao(this.value)">

```

Já neste exemplo, foi definido a instrução **else** que determinará o que deve ocorrer caso a condição seja falsa. No exemplo anterior, caso o campo não esteja vazio será exibida outra mensagem de alerta e em seguida foi definido que como retorno o texto do campo será selecionado.

OPERADOR TERNÁRIO

Além dos métodos condicionais apresentados, a linguagem JavaScript suporta um método alternativo para criar expressões condicionais. Este método já suportado em outras linguagens de programação permite ao usuário construir um exemplo prático e simples para sua utilização. Sua sintaxe básica tem a seguinte formação:

Se quiser tirar o else do ternário
(condição) && Valor verdadeiro

if ... else if ... else

(condição) ? Valor verdadeiro : Valor falso;

(condição) ? Valor de resposta : (2ªcondição) ? Resp. 2ª : Resposta do ELSE

Onde é **condição**, é a expressão à ser avaliada.

Onde é **Valor verdadeiro**, especifica a ação que ocorrerá se a condição for verdadeira.

Onde é **Valor falso**, especifica a ação que ocorrerá caso a condição seja falsa.

Veja abaixo um exemplo de utilização deste tipo de expressão condicional:

```
exemplo=prompt("Digite seu nome ou clique em Cancelar.", "");  
(exemplo==null) ? alert("O usuário Cancelou!") : alert("O usuário  
digitou: "+exemplo);
```

Assim como na maioria das linguagens de programação a instrução **if** não está limitada a utilização apenas do sinal de igualdade (**==**). O usuário poderá fazer diferentes tipos de testes lógicos como se os valores são diferentes, maior que ou menor que, entre outros.

RETURN

Esta instrução tem como finalidade marcar o final de uma função. A linguagem JavaScript ao encontrar esta instrução ele irá retornar para o ponto imediatamente após a chamada da função. Ela ainda pode ser definida com um valor de retorno ou não.

Quando um valor é incluído com esta instrução, a função irá retornar este valor definido pela instrução. Quando um valor não é incluído com a instrução **return**, então a função retorna um valor nulo.

Por padrão, esta instrução jamais é usada fora de uma função. Quando isto acontece, a linguagem irá retornar um erro quando a mesma estiver definida fora de uma função. Os parênteses apresentados no exemplo abaixo não são obrigatórios.

Vejamos alguns exemplos de scripts usando a instrução **return**.

```
<script>
<!--
function teste(){
    var valor=5;
    if (valor>=5){
        return (true);
    } else {
        return (false);
    }
}
-->
</script>
```

SWITCH

Esta instrução é bem semelhante com uma estrutura **IF**, porém é mais eficiente em razão de ser mais simples sua utilização e seu entendimento. Veja a sintaxe utilizada para o uso de instruções **SWITCH**:

```
switch (expressão){
    case CONSTANTE:
        comandos;
        break;

    case CONSTANTE2:
        comandos;
        break;

    case default:
        comandos;
        break;
}
```


INSTRUÇÃO WITH

Esta instrução faz com que um objeto se torne default para uma série de opções existentes. Normalmente esta instrução é utilizada com o objeto **Math**, uma vez que ele exige que o usuário especifique o nome do objeto quando acessar qualquer uma de suas propriedades. Veja sua sintaxe:

```
with (objeto){  
    instruções  
}
```

Vejamos alguns exemplos de sua utilização:

```
<script>  
alert(Math.PI);  
alert(Math.round(1234.5678));  
</script>
```

Utilizando a instrução **with** o usuário irá determinar os valores que deseja economizando tempo na aplicação. Observe como ficaria estas instruções aplicadas com a instrução **with**:

```
<script>  
with (Math){  
    alert(PI);  
    alert(round(1234.5678));  
}  
</script>
```

Veja pelo exemplo anterior, que o usuário não necessitou utilizar o objeto **Math** várias vezes.

Outra questão, é que a instrução with não é utilizada somente com o objeto Math. Ela poderá ser usada com a maioria dos outros objetos da linguagem JavaScript.

FOR EACH

ForEach(item, index, array) ---> É utilizado para iterar em array. OBS: Ele não contém o BREAK.

```
const users = [
  { name: 'Rodolfo', age:33, contact: '(19) 94343-3434' },
  { name: 'Paulo', age:21, contact: '(12) 93443-3434' },
  { name: 'Aline', age:40, contact: '(13) 94566-3434' },
  { name: 'Maria', age:12, contact: '(14) 94343-3476' }
]

users.forEach(item => {
  console.log(item)
});
/*
users.forEach( function (item, index) {
  console.log( `${item} ${index}` )
});*/
//OR
users.forEach( (item, index, array) => {
  console.log( `${index} \n${item} \n${array}` )
});
```

FUNÇÕES

Funções em JavaScript nada mais são que uma rotina JavaScript que possui um conjunto de instruções à serem executadas. Sua sintaxe compõem-se dos seguintes parâmetros:

```
function nomeFunção(argumentos) {  
    Comandos  
}
```

Uma Função sem o RETURN é uma função vazia (VOID), pois não retorna nada!
Para retornar alguma coisa temos que implementar o RETURN.

Se a função possuir argumentos, estes deverão estar colocados entre parênteses após o nome da função. O corpo da função deverá ser colocado entre chaves que indicarão o início do bloco de instruções e o fim do bloco de instruções.

Normalmente, as funções são definidas dentro do cabeçalho da página HTML representados pelo tag **<HEAD>**. Com isto, todas as funções são carregadas assim que a página é carregada, bem antes que o usuário pense em executar alguma ação.

Vejamos um exemplo simples de uma função que exibe uma mensagem na tela:

```
function primeiraFuncao(){  
    alert("Isto é uma função JavaScript");  
}
```

Com isto, o usuário apenas definiu a função, para que ela seja executada, é necessário fazer a sua chamada. Para chamar a função basta incluir seu nome no local do código que deseja que ela seja executada. No exemplo a seguir,

note que após a função foi feita sua chamada, bastando para tanto redigir seu nome, observe:

```
function primeiraFuncao(){  
    alert("Isto é uma função JavaScript");  
}  
primeiraFuncao();
```

É padrão da linguagem JavaScript que ao encontrar a chamada de uma função, ele desvia para as instruções respectivas desta função e ao terminá-la, volta para o primeiro código após a chamada da função.

Uma função pode ser chamada de diversas formas, dentro da área do código JavaScript e até mesmo através de um evento dentro de um tag HTML, como um botão de formulário ou hiperlink. Veja um exemplo de uso da chamada de uma função através da ação do usuário ao clicar em um botão de formulário:

```
<HTML>  
<HEAD>  
<TITLE>UTILIZANDO FUNÇÕES</TITLE>  
</HEAD>  
<BODY>  
<SCRIPT>  
function primeiraFuncao(){  
    alert("Isto é uma função JavaScript");  
}  
</SCRIPT>  
<INPUT TYPE="BUTTON" VALUE="Clique aqui!" onClick="primeiraFuncao()">  
</BODY>  
</HTML>
```

Neste exemplo, foi definido a chamada da função através do evento **onClick** que é processado assim que o usuário dá um clique sobre o botão que executará a função.

O usuário poderá através do uso de funções passar valores a mesma, com isto a função usará os valores no processamento. Vejamos no exemplo abaixo que foi definido como argumento da função **exemplo** a variável **texto**, esta variável é usada como o texto que será exibido pela instrução **alert**. Ao chamar a função, basta o usuário definir o texto que deseja ser apresentado como argumento da função:

```
<script>
function exemplo(texto){
    alert(texto);
}
exemplo("Curso de JavaScript");
</script>
```

Em algumas situações o usuário talvez queira retornar um valor de uma função. Para isto, basta fazer o uso da instrução **return** mais o valor que queira retornar. Vejamos um exemplo típico do uso de uma função que irá retornar um determinado valor. Observe:

```
<script>
var ret=prompt("digite o nome","");
var shor=mostranome(ret);
alert(shor);

function mostranome(nome){
    if (nome==" " || nome==null)
        return ("erro");
    else
```

```
        return (nome);  
    }  
}
```

OBJETO ARGUMENTS

Normalmente as funções são declaradas para aceitar um determinado número de parâmetros, vejamos um exemplo que usa uma função que é declarada para usar dois argumentos e usados para exibir os mesmos em uma mensagem de alerta:

```
<script>  
mensagem("SENAC","Minas Gerais");  
  
function mensagem(mensagem1,mensagem2){  
    alert(mensagem1);  
    alert(mensagem2);  
}  
</script>
```

É claro que se houvesse vários argumentos à serem exibidos, isto seria uma maneira penosa de programar. Através da linguagem JavaScript, o usuário poderá fazer uso do objeto **arguments** que é criado dentro de uma função. Este objeto é um array que poderá receber todos os argumentos necessários para passar a função quando a mesma for chamada. Veja no exemplo a seguir sua utilização:

```
<script>  
mensagem("SENAC","Minas Gerais");  
mensagem("CFP","Informática");  
  
function mensagem(){  
    for (i=0;i<mensagem.arguments.length;i++){
```

```
        alert(mensagem.arguments[i]);
    }
}
</script>
```

Arrow Function é uma sintaxe que resume a forma de declarar uma função:

```
função padrão -> function() {}
arrow function -> () => {}
```

Exemplo:

// Função normal

```
function sayMyName(name){
    return `Seu nome é ${name}`
}
```

// Arrow function

```
const sayMyName = (name) => `Seu nome é ${name}`
```

// Outra maneira

```
const sayMyName = (name) => {
    return `Seu nome é ${name}`
}
```

```
sayMyName("Ana")
```

UTILIZANDO EVENTOS

EVENTO ONBLUR

Com o evento **onBlur** o usuário irá controlar o conteúdo de um elemento em um formulário select, text ou textarea quando o mesmo remove o foco. Veja pelo exemplo a seguir de uma caixa de texto exibir uma mensagem na tela assim que o campo perde o foco:

```
<pre>
<form name="form1">
Digite seu Nome:
<input type="text" name="nome" onBlur="alert('Favor preencher')">
</form>
</pre>
```

Todo evento começa com ON

função é chamada

Para pegar o valor do INPUT, podemos implementar um função da seguinte forma:

```
function digiteNoInput(){console.log(input.value)}
```

Veja agora outro exemplo, criando uma função para que caso o usuário deixe o campo em branco, seja exibida a mensagem de alerta:

```
<script>
<!--
function teste(){
  if (form1.campo1.value==""){
    alert("FAVOR PREENCHER");
  }
}
-->
</script>
<pre>
<form name="form1">
Digite seu Nome:
```



```
<input type="text" name="campo1" onBlur="teste()">
</form>
```

EVENTO ONCHANGE

Com o evento **onChange** o usuário poderá acionar alguma função sempre que for alterado o conteúdo dos objetos **textarea**, **text** ou **select**. Este evento é bem similar com o evento **onBlur**, porém ele verifica se o conteúdo do elemento foi alterado. Vejamos um exemplo de sua utilização:

Digite seu Endereço:

```
<input type="text" name="campo2" onChange="alert('testando')">
```

EVENTO ONCLICK

O evento **onClick** é utilizado em links, botões, radio, checkbox, reset. Vejamos um exemplo de sua utilização em um botão de formulário:

```
<input type="button" name="botao" value="Envia"
onClick="alert('Obrigado pelos Dados')">
```

EVENTO ONFOCUS

Com o manipulador **onFocus** o usuário poderá criar uma ação sempre que os elementos **select**, **text** ou **textarea** receberem o foco. Ao contrário do evento **onBlur** que executa ações sempre que o elemento perde o foco. Veja um exemplo de sua utilização:

Digite seu Nome:

```
<input type="text" name="campo1" onBlur="teste()"
onFocus= "alert ('Digite seu nome completo')">
```

Digite seu Endereço:

```
<input type="text" name="campo2" onChange="alert('testando')"
onFocus="this.value='R. Castelo da Beira'">
```

EVENTO ONLOAD

Conforme exemplo mostrando anteriormente, com o evento **onLoad** executa alguma ação assim que o documento é carregado no browser. Este objeto é aplicado diretamente ao objeto **window**.

Veja abaixo um exemplo de exibição de uma mensagem de alerta assim que a página é carregada:

```
<body onLoad="alert('Seja Bem Vindo')">
```

EVENTO ONUNLOAD

Com o evento **onUnLoad** o usuário pode determinar uma ação assim que o usuário sai da página, seja por meio de um hiperlink ou até mesmo fechando o navegador. Com base no exemplo anterior foi criado uma mensagem de alerta assim que o usuário deixa a página:

```
<body onLoad="alert('Seja Bem Vindo')"
onUnLoad="alert('Obrigado pela visita!')">
```

EVENTO ONMOUSEOVER

Com o evento **onMouseOver** o usuário criará uma ação que será acionada sempre que o ponteiro do mouse se mover sobre um link. Veja um exemplo de sua utilização:

```
<a href="http://www.mg.senac.br"
onMouseOver="defaultStatus='Olhe para a barra de
Status'">SENAC</A><BR>
```

Será exibida uma mensagem na barra de status, assim que o mouse sair de cima do link. Para evitar este problema, atribua para este evento a instrução **return true** que executará o comando é executado sem problemas. Veja pelo exemplo a seguir:

```
<a href="http://www.mg.senac.br"
onMouseOver="defaultStatus='Olhe para a barra de Status';
return true">SENAC</A><BR>
```

Lembre-se que quando quiser executar duas ações para o mesmo evento, basta separá-las com um ponto e vírgula.

EVENTO ONMOUSEOUT

Com este evento o usuário poderá determinar uma ação assim que o mouse sair de cima de um hiperlink. Pelo exemplo do evento **onMouseMove** o usuário consegue atribuir uma mensagem na barra de status, porém a mensagem permanece, utilizando o evento **onMouseOut**, o usuário poderá informar o que deve acontecer quando o ponteiro do mouse sair do objeto.

Avaliando o exemplo anterior, vamos determinar que ao ponteiro do mouse sair do hiperlink, a mensagem da barra de status será omitida. Veja pelo exemplo a seguir:

```
<a href="http://www.mg.senac.br"
onMouseOver="defaultStatus='Olhe para a barra de Status';
return true" onMouseOut=defaultStatus="">SENAC</A><BR>
```

EVENTO ONMOUSEDOWN

O evento **onMouseDown** ocorre sempre que é pressionado o botão do mouse. Veja pelo exemplo apresentado abaixo:

```
<a href="http://www.mg.senac.br" onMouseOver="defaultStatus='Olhe
para a barra de Status';return true" onMouseOut=defaultStatus=""
onMouseDown="alert('testando')">SENAC</A>
```

EVENTO ONMOUSEUP

O evento **onMouseUp** ocorre sempre que o botão do mouse é solto. Este evento segue os mesmos padrões do evento apresentado anteriormente.

EVENTO ONKEYPRESS

O evento **onKeyPress** ocorre sempre que uma tecla é pressionada. Este evento segue os mesmos padrões do evento apresentado anteriormente.

EVENTO ONKEYDOWN

O evento **onKeyDown** ocorre sempre que uma tecla é abaixada. Este evento segue os mesmos padrões do evento apresentado anteriormente.

EVENTO ONKEYUP

O evento **onKeyUp** ocorre sempre que uma tecla é solta. Este evento segue os mesmos padrões do evento apresentado anteriormente.

EVENTO ONSELECT

O evento **onSelect** ocorre sempre quando o usuário seleciona um texto dos elementos de formulário **text** ou **textarea**.

Vejamos um exemplo de sua utilização:

```
<form name="form1">
Digite seu Nome:
<input type="text" name="campo1"
onSelect="alert('O usuário selecionou '+this.value)">
```

EVENTO ONSUBMIT

O evento **onSubmit** ocorre sempre que o usuário envia o formulário. Com este evento o usuário poderá determinar ou não o envio do formulário. Vejamos um exemplo para sua utilização:

```
<form name="form1" onSubmit="alert('Formulário Enviado')">
```

Com este evento o usuário poderá verificar a validação de dados, como por exemplo se todos os campos do formulário estão preenchidos.

Veja agora um exemplo desta utilização, caso o campo específico esteja em branco o usuário receberá uma mensagem informando que o campo não foi preenchido:

```
<script>
<!--
```

```

function teste(){
    if (form1.campo1.value==""){
        alert("FAVOR PREENCHER");
        return(false);
    } else {
        return(true);
    }
}
-->
</script>
<pre>
<form name="form1" onSubmit="teste()">
Digite seu Nome:
<input type="text" name="campo1">

```

AddEventListener: Verifica a ocorrência de eventos.

```

const input = document.querySelector("#main-input")
const select = document.querySelector("select")
const button = document.querySelector(".main-button")

```

/*Implementando uma função anônima dentro do AddEventListener:

```

select.addEventListener("change", function(){
    console.log("troquei de valor")
})

```

OBS: o "change" é uma função do método, existem vários!

FUNÇÕES DA LINGUAGEM JAVASCRIPT

As funções utilizadas em JavaScript são embutidas no núcleo da linguagem. Estas funções não pertencem à nenhum objeto, elas funcionam com variáveis número e as que não são objetos. Com estas funções não é necessário a declaração de um objeto-pai para usá-las, sendo bem diferentes dos métodos como por exemplo **document.write**. Write é o método pertencente ao objeto **document**.

FUNÇÃO EVAL

Esta função avalia uma expressão que poderá ser em formato string, o que se torna prático quando o usuário deseja estabelecer expressões no momento em que for preciso. Sua sintaxe é formada da seguinte maneira:

```
eval(expressão);
```

Criaremos um exemplo que irá avaliar uma expressão que contém números, operadores e strings. Neste exemplo formaremos um cálculo que será representado como string. Com o uso da função **eval**, será testado todos os argumentos da função fazendo a compreensão de todos os elementos presentes:

```
<script>
valor=5
alert(eval("2*valor"));
</script>
```

FUNÇÃO ISNAN

A função **isNaN** tem a finalidade de testar um número para determinar se é ou não um número. Normalmente esta função é usada para comparar valores do tipo número ou string. Com o emprego desta função o usuário poderá determinar se um valor contém realmente um valor numérico. Esta função pode ser utilizada em conjunto com as funções **parseInt** e **parseFloat** em razão de retornarem a string **NaN** quando o conteúdo da variável não é um número. Sua sintaxe tem a seguinte formação:

```
isNaN(valor);
```

No exemplo a seguir, foi criado um script bem simples que testa se o valor digitado no campo de formulário é um número.


```
<script>
function condicao(valor){
    if(isNaN(valor)){
        alert("Não é um número!!!");
    }
}
</script>
<form name="form1">
Nome:
<input type="text" name="nome" onBlur="condicao(this.value)">
```

FUNÇÃO PARSEFLOAT

Com a função **parseFloat**, é feita a conversão de um objeto string, retornando um valor de ponto flutuante. Com ela é convertido uma string em um valor numérico equivalente. Se a função encontrar um caractere diferente de um sinal (+ ou -), números (0 à 9), ponto decimal ou expoente, será retornado o valor até aquele ponto e ignorado o restante. Caso o primeiro caractere não puder ser convertido para um número, **parseFloat** irá retornar os valores **0** para a plataforma Windows e **NaN** para as outras plataformas. Sua sintaxe tem a seguinte formação:

```
parseFloat(string);
```

Veja a seguir um exemplo da utilização da função **parseFloat**.

```
<script>
valor=parseFloat("123,456");
alert(valor)+1;
</script>
```

FUNÇÃO PARSEINT

Com a função **parseInt**, o usuário poderá converter valores de string em valores numéricos equivalentes. É possível a conversão de números de bases como hexadecimal ou octal em valores decimais. Caso a função encontra um caractere diferente de um sinal (+ ou -), números (0 à 9), ponto decimal ou expoente, será retornado o valor até aquele ponto e ignorado o restante. Caso o primeiro caractere não puder ser convertido para um número, a função **parseInt** irá retornar o valor 0 para Windows e NaN para outros sistemas. Sua sintaxe tem a seguinte formação:

```
parseInt(string,radix);
```

Onde é apresentado **radix**, é um inteiro que representa a base do valor de retorno. No exemplo a seguir é convertido um valor string em seu valor numérico equivalente:

```
<script>
valor=parseInt("123.456");
alert(valor);
</script>
```

através do parâmetro **radix**, é possível a conversão de um número de uma base para decimal, já no caso contrário isto não é possível. Veja um exemplo de sua utilização:

```
valor=parseInt("ff",16);      //Conversão hexadecimal, retorna 255
valor=parseInt("0xff",16);    //Conversão hexadecimal, retorna 255
valor=parseInt("1111",2);     //Conversão binário, retorna 15
valor=parseInt("765",8);      //Conversão octal, retorna 501
```

Vejamos os valores de cada base existente:

- 2 ➔ Binário.**
- 8 ➔ Octal.**
- 10 ➔ Decimal.**
- 16 ➔ Hexadecimal.**

A omissão do parâmetro **radix**, a linguagem JavaScript assume que o valor definido está no formato decimal.

FUNÇÕES PRÉ-PROGRAMADAS

As funções pré-programadas do JavaScript, permite ao usuário executar operações simples como configurar sua página como inicial, adicionar uma URL ao favoritos, imprimir o documento, aumentar sua lógica de raciocínio facilitando a criação de novos scripts, entre outras operações. Vejamos alguns exemplos.

IMPRESSÃO DA PÁGINA

Através da função **print()**, o usuário poderá executar a função de imprimir evitando caminhos longos para isto ou facilitar ao usuário iniciante em informática, abaixo segue um exemplo simples:

```
<P onMouseDown='self.print()'">Imprimir Documento</p>
```

Neste exemplo foi usado o evento **onMouseDown** que avisa ao navegador para imprimir o documento atual quando o usuário clicar sobre o texto de parágrafo.

ADICIONAR AO FAVORITOS

Segue a seguir, um exemplo de inserção de uma URL à pasta dos Favoritos. Veja sua utilização e estude sua lógica, observe que é bem simples:

```

<script>
<!--
var url="http://www.mg.senac.br"
var titulo="Portal Senac Minas"
function Favoritos(){
if (document.all)
window.external.AddFavorite(url,titulo)
}
// -->
</script>
<input type="button" value="Favoritos" onClick=Favoritos(>

```

JANELA EM MOVIMENTO

Outro Exemplo interessante é a abertura de uma página que abre uma janela em movimento. Veja o código abaixo e faça um teste:

```

function expandingWindow(website) {
var heightspeed=2;//velocidade vertical (valor maior = mais lento)
var widthspeed=7;//velocidade horizontal(valor maior = mais lento)
var leftdist = 0; // distância do canto esquerdo da janela
var topdist = 0; // distância do canto superior da janela
if (document.all) {
var winwidth = window.screen.availWidth - leftdist;
var winheight = window.screen.availHeight - topdist;
var sizer = window.open("", "", "left=" + leftdist + ",top=" +
topdist + ",width=1,height=1,scrollbars=yes");
for (sizeheight = 1; sizeheight < winheight; sizeheight +=
heightspeed) {
sizer.resizeTo("1", sizeheight);
}
for (sizewidth = 1; sizewidth < winwidth; sizewidth += widthspeed)
{

```

```

sizer.resizeTo(sizewidth, sizeheight);
}
sizer.location = website;
}
else
window.location = website;
}
// End -->
</script>
<a href="http://javascript.internet.com/"
onClick="expandingWindow('http://www.aglima.ezdir.net/');return
false;">Open JavaScriptSource.com!</a>

```

TEXTO NA BARRA DE STATUS EM MOVIMENTO

```

<html>
<head>
<script LANGUAGE="JavaScript">
<!--
var speed = 10
var pause = 1500
var timerID = null
var bannerRunning = false
var ar = new Array()
ar[0] = "Adriano... "
ar[1] = "Gomes"
ar[2] = "Lima"
ar[3] = "Informática:"
ar[4] = "à cidade de Santos."
var message = 0
var state = ""
clearState()
function stopBanner() {
if (bannerRunning)

```

```

clearTimeout(timerID)
timerRunning = false
}
function startBanner() {
stopBanner()
showBanner()
}
function clearState() {
state = ""
for (var i = 0; i < ar[message].length; ++i) {
state += "0"
}
}
function showBanner() {
if (getString()) {
message++
if (ar.length <= message)
message = 0
clearState()
timerID = setTimeout("showBanner()", pause)
} else {
var str = ""
for (var j = 0; j < state.length; ++j) {
str += (state.charAt(j) == "1") ? ar[message].charAt(j) : " "
}
window.status = str
timerID = setTimeout("showBanner()", speed)
}
}
function getString() {
var full = true
for (var j = 0; j < state.length; ++j) {
if (state.charAt(j) == 0)

```

```

full = false
}
if (full) return true
while (1) {
var num = getRandom(ar[message].length)
if (state.charAt(num) == "0")
break
}
state = state.substring(0, num) + "1" + state.substring(num + 1,
state.length)
return false
}
function getRandom(max) {
var now = new Date()
var num = now.getTime() * now.getSeconds() * Math.random()
return num % max
}
// --></script>
</head>
<body onLoad=showBanner(>

```

TABELA DE CORES

```

<SCRIPT>

function geraTabela() {

    document.write('<table border=1 width="100%">');

    var valores = "00336699CCFF";

    var r, g, b;

    var cor;

    for (r=0; r<6; r++) {
        for (g=0; g<6; g++) {

```



```

        if (g%2==0) document.write("<tr>");
        for (b=0; b<6; b++) {
            cor = valores.substr(2*r,2)
                + valores.substr(2*g,2)
                + valores.substr(2*b,2);
            document.write('<td align="center" bgcolor="#'+cor+'">');
            if (g<3) document.write("<font size=-2 color=white>");
            else document.write("<font size=-2 color=black>");
            document.write(cor+"</font></td>");
        }
        if (g%2==1) document.write("</tr>");
    }
}
document.write("</table>");
}
</SCRIPT>
<body onLoad=geraTabela(>

```

TEXTO EM MOVIMENTO EM UM CAMPO DE FORMULÁRIO

```

<script language="javascript">
<!--
var mensagem = "Curso de JavaScript do Senac Minas!!!";
var texto;
var ligado = false;
var timeoutID = null;
var posicao = 0;
var tamanho;
var janela;

function ligarMarquee(){
    if (ligado) pararMarquee();

```

```
    texto = document.form1.marquee.value + mensagem +  
document.form1.marquee.value;  
    tamanho = texto.length;  
    janela = document.form1.marquee.size;  
    atualizarMarquee();  
    ligado = true;  
}
```

```
function pararMarquee(){  
    if (ligado) clearTimeout(timeoutID);  
    ligado = false;  
}
```

```
function atualizarMarquee(){  
    document.form1.marquee.value=texto.substring(posicao++%tamanho,  
posicao+janela*tamanho);  
    timeoutID = setTimeout("atualizarMarquee()", 100);  
}  
// -->  
</script>  
<form name="form1">  
    <p><input type="text" name="marquee"  
value="                                " size="20"><br>  
    <input type="button" name="ligar" value="Ligar"  
onClick="ligarMarquee();"> <input  
    type="button" name="parar" value="Parar"  
onClick="pararMarquee();"> </p>  
</form>
```


OBJETOS PRÉ-CONSTRUÍDOS

A linguagem JavaScript possui objetos padronizados para uso nos scripts. Dentre eles, temos:

DATE → Manipula datas e horas.

STRING → Manipula string's.

MATH → Realiza operações matemáticas.

OBJETO DATE

O objeto **DATE** permite que o usuário possa trabalhar com datas e horas. Para determinar um novo objeto de data, temos as seguintes sintaxes:

```
NomeObjeto=new Date()
```

```
NomeObjeto=new Date("mês dia,ano horas:minutos:segundos")
```

```
NomeObjeto=new Date(ano,mês,dia)
```

```
NomeObjeto=new Date(ano,mês,dia,horas,minutos,segundos)
```

Veja exemplos conforme sintaxe apresentada anteriormente:

Data=new Date() → atribui a variável data, a data e hora correntes.

Data=new Date(1975,11,23) → atribui a variável data, a data **23 de novembro de 1975**.

MÉTODOS DO OBJETO DATE

Se o usuário desejar utilizar os métodos do objeto de data, deve-se seguir a seguinte sintaxe:

`NomeObjeto.método(parâmetros)`

Veja a relação dos métodos utilizados no objeto **DATE**:

OBJETO STRING

PROPRIEDADES

Os objetos string são de nível superior.

SINTAXE

Variável="valor"

S1="SENAC"

PROPRIEDADES DO OBJETO STRING

Veja na tabela a seguir a relação das propriedades do objeto **String**:

PROPRIEDADES	DESCRIÇÃO
length	Comprimento de uma string.

MÉTODOS DO OBJETO STRING

Os métodos do objeto **string** permitem a manipulação do objeto. O usuário poderá utilizar string literal ou de variáveis. Vejamos sua sintaxe abaixo:

"String literal".método()

TextString="string de variável"

```
TextString.método()
```

MÉTODO ANCHOR

Este método tem a função de criar uma âncora a partir de uma string. Este método é similar à criação de uma âncora utilizando o tag HTML ****, o mesmo ocorreria se definir **string.anchor(valor)**. Vejamos a sintaxe de utilização do método **anchor**:

```
String.anchor(nome)
```

Veja um exemplo de utilização deste método:

```
<script>
Ancora="Início do Documento";
valor=Ancora.anchor("inicio");
document.write(valor);
</script>
```

Este script poderia ser utilizado pela linguagem HTML através do seguinte código:

```
<A NAME="inicio">Início do Documento</a>
```

MÉTODO BIG

Este método substitui o tag HTML **<BIG>**, que tem a função de aumentar a fonte e atribuir o estilo de negrito. Para utilizá-lo, siga a seguinte sintaxe:

```
string.big();
```

Veja o exemplo de utilização deste método:

```
<script>
texto="SENAC-MG";
document.write(texto.big());
</script>
```

MÉTODO SMALL

Este método substitui o tag HTML **<SMALL>** que tem a função de reduzir o tamanho da fonte. Para utilizá-lo, siga a seguinte sintaxe:

```
String.small();
```

Veja o exemplo de utilização deste método:

```
<script>
texto="SENAC-MG";
document.write(texto.small());
</script>
```

MÉTODO BOLD

Referente ao tag HTML **** que tem a função de atribuir o estilo de negrito sobre o texto. Sua sintaxe segue o seguinte padrão:

```
String.bold();
```

Veja o exemplo de utilização deste método:

```
<script>
texto="SENAC-MG";
document.write(texto.bold());
</script>
```


MÉTODO ITALICS

Este método é referente ao tag HTML **<I>** que atribui o estilo de itálico em um texto. Sua sintaxe segue o mesmo padrão do método **bold**. Veja abaixo um exemplo da utilização do método **italics**

```
<script>
texto="SENAC-MG";
document.write(texto.italics());
</script>
```

MÉTODO FIXED

Com o método **fixed**, é possível formatar o qualquer texto em fonte fixa, ou como conhecido em HTML, em fonte monoespaço definido pelo tag **<TT>**. Sua sintaxe segue a seguinte composição:

```
String.fixed();
```

Exemplo de utilização do método **fixed**:

```
<script>
texto="SENAC-MG";
document.write(texto.fixed());
texto2="ADRIANO LIMA".fixed();
document.write("<BR>",texto2);
</script>
```

MÉTODO STRIKE

Este método tem a função de criar um texto tachado que exibe uma linha no meio do texto exibido. Este método tem a mesma função do tag HTML **<STRIKE>**. Sua sintaxe básica segue o seguinte padrão:

```
<script>
texto="SENAC-MG";
document.write(texto.strike());
</script>
```

MÉTODO FONTCOLOR

Determina a cor da fonte em um texto de acordo com o tag HTML ****.

SINTAXE

```
String.fontcolor(cor);
```

Exemplo de utilização do método **fontcolor**:

```
<script>
texto="SENAC-MG";
document.write(texto.fontcolor("red"));
document.write("Informática".fontcolor("blue"));
</script>
```

O método **fontcolor** aceita nomes de cores sólidas, assim como, os valores hexadecimais da cor referente.

MÉTODO FONTSIZE

Este método, determina o tamanho da fonte seguindo os padrões do tag HTML **** que possui tamanhos que vão de 1 a 7, assim como a possibilidade de valores relativos através dos sinais de + e -. Sua sintaxe básica segue o seguinte padrão:

```
<script>
texto="SENAC-MG";
document.write(texto.fontSize(7));
</script>
```

MÉTODO SUB

Este método cria um texto subscrito tendo o mesmo efeito do tag HTML **<SUB>**. Sua sintaxe básica tem a seguinte formação:

```
String.sub();
```

Veja um exemplo para sua utilização:

```
<script>
texto="SENAC-MG";
document.write(texto.sub());
</script>
```

MÉTODO SUP

Este método cria um texto sobrescrito tendo o mesmo efeito do tag HTML **<SUP>**. Sua sintaxe básica tem a seguinte formação:

```
String.sup();
```

Veja um exemplo para sua utilização:

```
<script>
texto="SENAC-MG";
document.write(texto.sup());
</script>
```

MÉTODO charAT

Com este método o usuário poderá retornar o caractere em uma determinada posição em uma string. Por exemplo, temos a string **SENAC** e a posição de referência é 3, com base nisto o caractere de retorno é **"A"**. Estas posições são contadas à partir de 0 da esquerda para a direita.

SINTAXE:

```
String.charAt(valorRetorno);
```

Veja o exemplo de utilização do método **charAt**:

```
<script>
texto="SENAC-MG";
document.write(texto.charAt(3));
</script>
```

MÉTODO INDEXOF

Com o método **indexOf** o usuário pode retornar a posição de um caractere dentro de uma string. Um exemplo claro do método **indexOf**, é a maneira de saber se determinada string possui algum caractere específico. Caso a string não contiver o caractere específico, o método irá retornar o valor **-1**, caso haja

a ocorrência do caractere procurado, será retornado o valor **0** ou superior, sendo que **0** é a posição do primeiro caractere da string, **1** a posição do segundo caractere e assim por diante. Caso exista duplicidade do caractere específico, o método irá retornar a posição do primeiro caractere encontrado. Sua sintaxe segue o seguinte padrão:

```
string.indexOf(caractere)
```

Veja pelo exemplo a utilização do método **indexOf**:

```
<script>
texto="SENAC-MG";
document.write(texto.indexOf("A"));
</script>
```

Valor retornado: **A**

Uma das práticas utilizações deste método, é determinar se determinado valor de uma string é um número ou uma letra.

MÉTODO LASTINDEXOF

Com o método **lastIndexOf** o usuário poderá retornar a última posição de um determinado caractere em uma string. Um exemplo de utilização deste método é a de retornar a posição de um caractere barra (/) em uma string, para por exemplo utilizar com URL's. Sua sintaxe básica, segue o seguinte exemplo:

```
String.lastIndexOf(caractere,offset);
```

Onde **caractere**, é o caractere que deseja procurar dentro da string.

Onde **offset**, é a posição na string a partir de onde o usuário deseja começar a pesquisa. Veja abaixo um exemplo que localiza a última ocorrência da letra **"N"** na string **SENAC-MG** utilizada como exemplo.

```
<script>
texto="SENAC-MG";
document.write(texto.lastIndexOf("N"));
</script>
```

O resultado será **2**. É bom lembrar que as strings sempre se baseiam em 0).

MÉTODO LINK

Este método é similar ao tag HTML **<A HREF>** que tem a função de criar hiperlinks em uma página. Sua sintaxe básica tem a seguinte formação:

```
String.link(href);
```

Onde é **href** é a referência de vínculo do hiperlink.

Vejamos um exemplo:

```
<script>
texto="SENAC-MG";
document.write(texto.link("http://www.mg.senac.br"));
</script>
```

MÉTODO REPLACE

Este método tem a função de trocar valores dentro de uma string. Sua sintaxe básica tem a seguinte formação:

```
String.replace(s1,s2);
```

Onde **s1** é o caractere procurado dentro de uma string.

Onde **s2** é o novo caractere que será trocado por **s1**.

Vejamos um exemplo simples que ao ser digitado um nome com acento agudo na letra **A**, ao clicar sobre o um botão é trocado a letra sem acento.

```
function troca(){
texto=document.form1.nome2.value;
document.form1.nome2.value=texto.replace("á","a");
}
```

Logo a seguir o código do botão que chama a função **troca()**.

```
<input type="button" onClick="troca()" value="troca">
```

MÉTODO SUBSTRING

Este método retorna uma parte de uma string. O usuário poderá especificar o início e o final da parte que deseja extrair indicando a posição inicial como 0, já a posição final é determinada com a instrução **string.length-1**, isto é, um a menos do que o comprimento da string. Sua sintaxe básica tem a seguinte formação:

```
string.substring(início,fim);
```

Vejamos um exemplo da utilização do método **substring**:

```
<script>
texto="SENAC-MG";
document.write(texto.substring(0,4));
</script>
```

Valor retornado: **SENA**.

MÉTODO TOLOWERCASE

Com o método **toLowerCase** o usuário poderá converter uma string em letras minúsculas. Sua sintaxe básica segue o seguinte padrão:

```
<script>
texto="SENAC-MG";
document.write(texto.toLowerCase());
</script>
```

Veja que o conteúdo da variável **texto** está em letras maiúsculas, com o uso do método **toLowerCase**, este texto será apresentado no documento em letras minúsculas.

MÉTODO TOUPPERCASE

Com o método **toUpperCase**, o usuário poderá converter uma string em letras maiúsculas. Sua sintaxe básica segue o seguinte padrão:

```
<script>
texto="senac-mg";
document.write(texto.toUpperCase());
</script>
```


OBJETO IMAGE

Na linguagem JavaScript as imagens que são inseridas através da linguagem HTML são consideradas cada uma um objeto do tipo **IMAGE**. Com isto, podemos concluir que as imagens possuem propriedades e métodos assim como os outros objetos já existentes. Através deste objeto é possível que o usuário possa interagir melhor e dinamicamente as imagens utilizadas em suas páginas.

Vejamos pelo exemplo a seguir a instrução HTML que insere uma imagem em uma página.

```
<html>
<body>

```

Até aqui tudo bem, mas note que fora atribuído uma variável nesta imagem através do atributo **name**. Esta variável serve para fazer referência à imagem atualmente inserida na página no código JavaScript que será desenvolvido. Vamos agora inserir um botão de formulário que será responsável pelo evento que iremos desenvolver, logo nosso código ficará da seguinte forma:

```
<html>
<body>

<br>
<input type="button" value="Muda Figura">
```

No código a seguir, foi utilizado o evento **onClick** que determinará a ação para o botão. Esta ação foi designada para trocar a imagem atualmente inserida por

outra imagem. Note que foi feita uma referência para inserir à nova imagem no local da imagem atual.

```
<html>
<body>

<br>
<input type="button" value="Muda Figura"
onClick="document.senac.src='iso9001.gif'">
```

Em análise sobre este código simples, foi determinado que no documento atual, especificamente no objeto chamado **senac** que trata a figura atualmente inserida, será originada outra imagem **"iso9001.gif"** em seu local atual. Resultando na troca da imagem. Veja agora o mesmo código fazendo alternância entre as duas imagens de acordo com a opção escolhida, observe:

```
<html>
<body>

<br>
<input type="button" value="Muda Figura"
onClick="document.senac.src='iso9001.gif'">
<br>
<input type="button" value="Volta Figura"
onClick="document.senac.src='logo_senac.jpg'">
```

observe agora a criação de uma função que fará com que quando o usuário mover o mouse sobre a imagem a mesma será ampliada e ao movimentar para fora da imagem, a mesma retornará ao seu tamanho original:

```
<script>
    function figura(valor){
        document.senac.width=valor;
    }
</script>

```

ANOTAÇÕES:

MÉTODOS DE INTERFACE COM O USUÁRIO

Com este tipo de método, o usuário poderá criar objetos especiais que criam diferentes tipos de caixas de diálogo. Estes métodos fazem parte do objeto **window**. Com base nisto é possível por exemplo utilizar as seguintes instruções que resultam no mesmo efeito:

```
alert("Seja Bem Vindo!!!");
```

ou

```
window.alert("Obrigado pela Visita");
```

MÉTODO ALERT

Com o método **alert**, o usuário poderá sempre que desejar, exibir uma mensagem na tela exibindo uma caixa de diálogo como mostrado na figura



abaixo:

Este método segue a seguinte sintaxe:

```
alert(valor);
```

Veja pelo exemplo do script abaixo, a apresentação de uma mensagem através do método **alert**:

```
<script>
alert("Seja Bem Vindo!!!");
</script>
```

Com o método **alert** é possível exibir vários tipos de valores como numéricos, strings, booleanos, entre outros. Veja outras maneiras de utilização do método **alert**:

```
<script>
texto="SENAC-MG";
alert("Seja Bem Vindo!!!"); // Exibe a string.
alert(12)                   // Exibe o valor numérico 12.
alert(texto)                // Exibe o conteúdo da variável TEXTO.
alert(texto+" Informática") // Exibe a variável mais a string.
alert(true)                 // Exibe o valor true.
</script>
```

Para que o texto da janela alert() apareça em várias linhas, será necessário utilizar o caractere especial **/n** para criar uma nova linha.

MÉTODO CONFIRM

Com o método **confirm** o usuário irá exibir uma caixa de diálogo com os botões **OK** e **CANCELAR**. De acordo com o botão escolhido a linguagem JavaScript irá retornar um determinado valor. Quando o usuário pressiona o

botão **OK**, é assumido o valor 1, caso seja pressionado o botão **CANCELAR**, será assumido o valor 0. Sua sintaxe básica é formada da seguinte maneira:

```
confirm(valor);
```

vejam os um exemplo da utilização do método **confirm**:

```
<script>
teste=confirm("Tem certeza que deseja fechar?");
if (teste==1){
alert("Arquivos fechados");
}else{
alert("Operação Cancelada");
}
</script>
```

MÉTODO PROMPT

Com o método **prompt** é possível a criação de uma caixa de diálogo onde o usuário poderá entrar com alguma informação, além de poderem optar no uso dos botões **OK** e **CANCELAR**. Quando o usuário cancela, automaticamente é assumido ao método **prompt** um valor nulo. Sua sintaxe é formada da seguinte maneira:

```
prompt(valor,default);
```

onde **default**, é o valor padrão que será exibido na caixa de texto ao usuário.

Veja um exemplo da utilização do método **prompt**:

```
<script>
teste=prompt("Digite seu Nome:","");
```

```
alert(teste+" seja Bem Vindo!");  
</script>
```

Os possíveis valores a serem retornados pelo método **prompt** são:
String, quando o usuário digita um texto e pressiona o botão **OK**.

String vazia, quando o usuário não digita nada e pressiona **OK**.

NULO (null), quando o usuário pressiona o botão **CANCELAR**.

ANOTAÇÕES:

OBJETO WINDOW

A manipulação de janelas se dá através do objeto **window** da linguagem JavaScript. Sempre que se abre o browser automaticamente é criado este objeto que representa exatamente a janela que fora aberta. Isto é feito automaticamente sem a necessidade de usar os comandos da linguagem HTML.

Este objeto permite que o usuário crie e abra novas janelas de formas diferentes. Tudo isto é possível através das propriedades e métodos do objeto **window**. Para utilizá-los, basta seguir a seguinte sintaxe:

`window.propriedade`

`window.método`

PROPRIEDADES DO OBJETO WINDOW/FRAME

As propriedades deste objeto modificam os aspectos em relação da janela do navegador e de seus frames caso existam.

PROPRIEDADES	DESCRIÇÃO
frames[]	Array de frames em uma janela.
length	Quantidade de frames existentes em uma janela.
name	Nome do objeto window.
parent	Representa a janela ou frame-pai.
self	Representa a janela atual de um frame.
top	Representa a janela superior do browser.
window	Representa a janela ou frame-pai.

status	Define uma string que será apresentada na barra de status
defaultStatus	Define uma mensagem padrão que será apresentada na barra de status.

WINDOW.STATUS E DEFAULTSTATUS

Tanto **status** como **defaultStatus** são utilizadas para atribuir uma string na barra de status, com a diferença que a propriedade **defaultStatus** pode ser utilizada como um simples objeto apesar de ser ainda uma propriedade do objeto **window**, mas além disto a propriedade **defaultStatus** permite armazenar a mensagem padrão que será exibida, ou seja, aquela que voltará a ser exibida após modificações temporárias provocadas por mensagens colocadas na propriedade status. A sintaxe básica das duas propriedades seguem o seguinte parâmetro:

```
window.status("mensagem");
window.defaultStatus = "Esta é a mensagem padrão.";
defaultStatus = "Esta é a mensagem padrão";
```

Veja o funcionamento disto acionando os botões abaixo:

MÉTODO OPEN

Este método tem como objetivo abrir uma nova janela do browser. Com este método o usuário poderá abrir uma nova janela em branco ou uma janela que contém um documento específico. Sua sintaxe tem a seguinte formação:

```
NomeJanela=window.open(URL,alvo,opções);
```

Onde **NomeJanela** é uma variável que será uma referência a nova janela.

Onde **URL** é o endereço da janela a ser aberta.

Onde **alvo** é o nome da janela para ser usado no atributo **target** dos tag's **<FORM>** ou **<A>**.

Onde **opções** é o parâmetro que controla o comportamento da janela.

MÉTODO CLOSE

O método **close** do objeto **window** tem a finalidade de fechar uma janela. Normalmente utiliza-se este método atribuído à um botão de ação criado com os formulários. Sua sintaxe básica tem a seguinte formação:

```
window.close()
```

No exemplo abaixo temos uma página com um botão chamado **Fechar**, onde quando o usuário clicar sobre o mesmo é acionado este evento.

```
<input type="button" value="Fechar" onClick="window.close()">
```

Neste caso, foi utilizado o evento **onClick** que executa a instrução **window.close()** assim que o usuário clica sobre o botão.

MÉTODO SETTIMEOUT

Através do método **setTimeout** o usuário poderá criar um contador de tempo que executa alguma ação após um determinado intervalo de tempo. Sua sintaxe segue o seguinte padrão:

```
variável=setTimeout("função",intervalo);
```

Executa uma vez

setInterval -> Executa infinitas vezes

Onde é **função** é alguma instrução que o usuário quer que execute após o intervalo especificado em **milissegundos (milésimos de segundos)**. Na maioria das vezes, função é uma chamada de uma função criada anteriormente.

Onde é **intervalo** é o tempo até que a função seja executada.

Um dos exemplos mais comuns de uso do método **setTimeout** é o da criação de contadores em uma página e textos rolantes na barra de status. Veja pelo exemplo do script a seguir, o uso de texto piscante na barra de status:

ANOTAÇÕES:

3 maneiras de usar o setTimeout:

```
setTimeout(() => {  
  // executa o que estiver aqui  
}, timeout);
```

```
setTimeout(function() => {  
  // executa o que estiver aqui  
}, timeout);
```

```
function myFuncion(){  
  // executa o que estiver aqui  
}
```

```
setTimeout(myFuncion, timeout);
```

```
<script>
```

```
<!--
```

```
texto="SENAC-MG";
```

```
velocidade=150;
```

```
controle=1;
```

```
function flash(){
```

```
  if (controle == 1){
```

```
    window.status=texto;
```

```
    controle=0;
```

```
  }else{
```

```
    window.status=" ";
```

```
    controle=1;
```

```
    }  
    setTimeout("flash();", velocidade);  
  
}  
// -->  
</script>
```

MÉTODO CLEARTIMEOUT

Através do método **clearTimeout** o usuário poderá cancelar um marcador de tempo que foi criado pelo método **setTimeout**. Sua sintaxe segue o seguinte padrão:

```
clearTimeout(tempo);
```

Onde é **tempo** é o manipulador de identificação do timer criado pelo método **setTimeout**.

clearInterval serve para pausar o setInterval. Exemplo de contador:

```
let number = 0  
let cron  
let h1 = document.querySelector('h1')  
  
function start() {  
    cron = setInterval(function () {  
        number++  
        h1.innerHTML = number  
    }, 1000)  
}  
  
function stop() {  
    clearInterval(cron)  
}
```

TRABALHANDO COM JANELAS

Qualquer usuário que costuma navegar na Internet, sabe que é possível manipular a janela aberta de um site através de comandos do próprio browser como por exemplo o comando **Tela Cheia** encontrado no menu **Exibir** do navegador, entre outras opções. Mas através da linguagem JavaScript é possível realizar as mesmas operações através de sua programação.

Se o usuário desejar abrir uma nova janela à partir de uma janela já aberta, basta utilizar o método **open** em sua estrutura. Veja sua sintaxe:

```
window.open( "URL" );  
janela=window.open( "URL" );
```

Onde:

janela: é referente ao nome dado para a janela a ser aberta para fins das instruções de programação.

window.open: OPEN é o método do objeto window para abrir uma nova janela.

URL: é o endereço da página que será aberta na nova janela. Caso o usuário omitir este endereço, será aberta uma nova janela vazia.

A omissão de uma URL, será aberta uma nova janela em branco.

Veja no exemplo abaixo, a criação de uma nova janela chamada **SENAC** onde será aberto o site do SENAC.

```
Senac=window.open("http://www.mg.senac.br")
```

É possível através de outros argumentos, definir o comportamento de como a nova janela deverá ser apresentada. Veja os argumentos existentes para o método **open**:

TOOLBAR → permite a exibição da barra de ferramentas do navegador.

LOCATION → permite a exibição da barra de endereço do navegador.

DIRECTORIES → permite a exibição da barra de links do navegador.

STATUS → permite a exibição da barra de status do navegador.

MENUBAR → permite a exibição da barra de menu do navegador.

SCROLLBARS → permite a exibição das barras de rolagem do navegador.

RESIZABLE → permite ao usuário redimensionar a nova janela do navegador.

WIDTH → especifica a largura da nova janela do navegador em pixels.

HEIGHT → especifica a altura da nova janela do navegador em pixels.

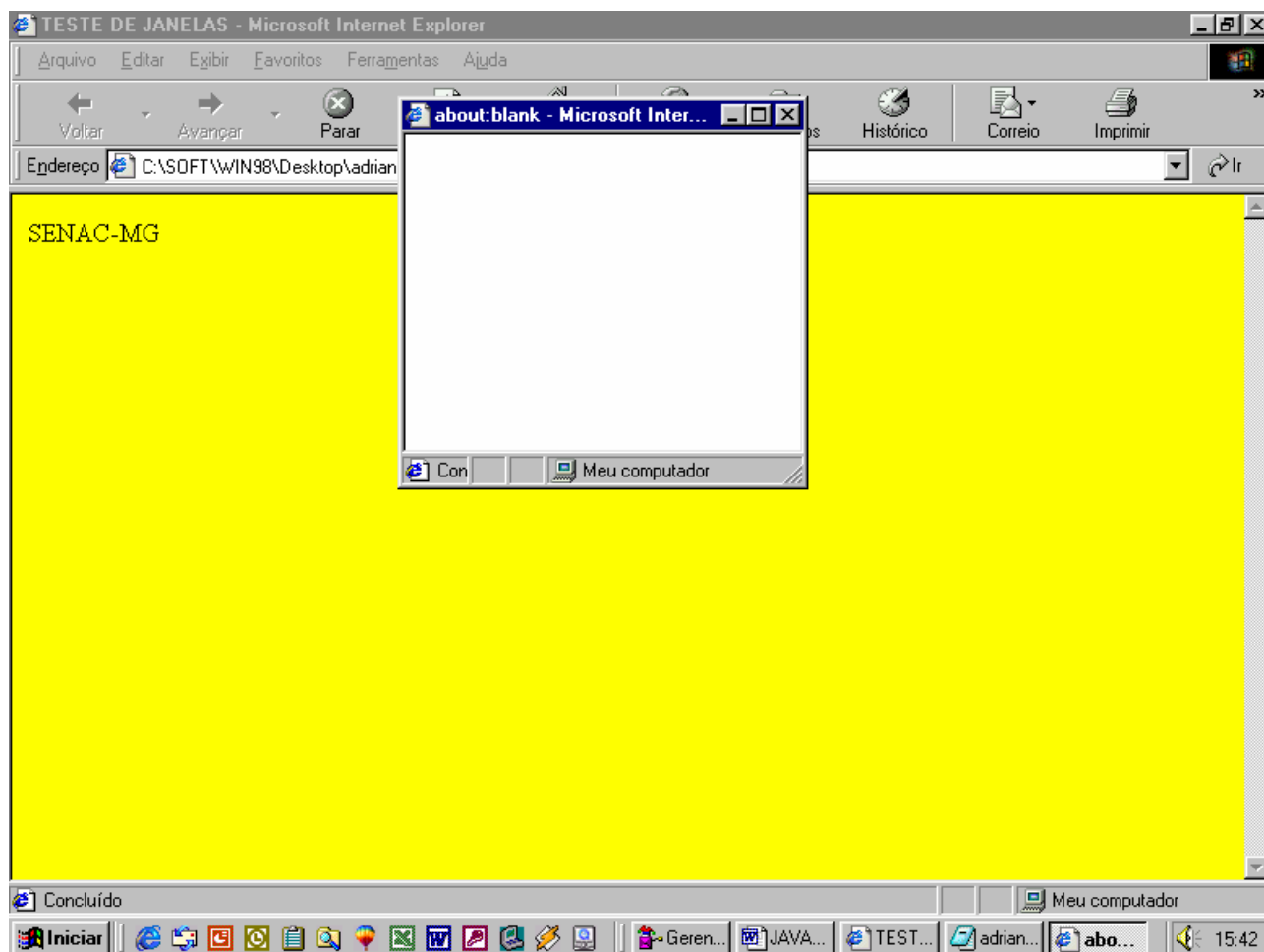
Quaisquer uns destes argumentos possuem valores booleanos (**yes/no,1/0**). Se utilizar mais de um argumento, os mesmos deverão estar separados por vírgula. Aquelas opções não definidas irão assumir valores falsos.

No exemplo apresentado a seguir, é mostrada a abertura de uma segunda página apenas com a barra de status e com tamanho de 250 x 200 pixels:

```
<HTML>
<HEAD>
  <title>TESTE DE JANELAS</TITLE>
  <script>
    janela2=window.open("", "", "status=yes,width=250,height=200")
  </script>
</head>
<body bgcolor=yellow>
  SENAC-MG

</body>
</html>
```

Veja o efeito deste código apresentado na figura a seguir:



Se o usuário desejar controlar o código HTML da janela gerada, basta determinar no código JavaScript da janela principal os tag's específicos para a segunda janela. Veja pelo exemplo a seguir:

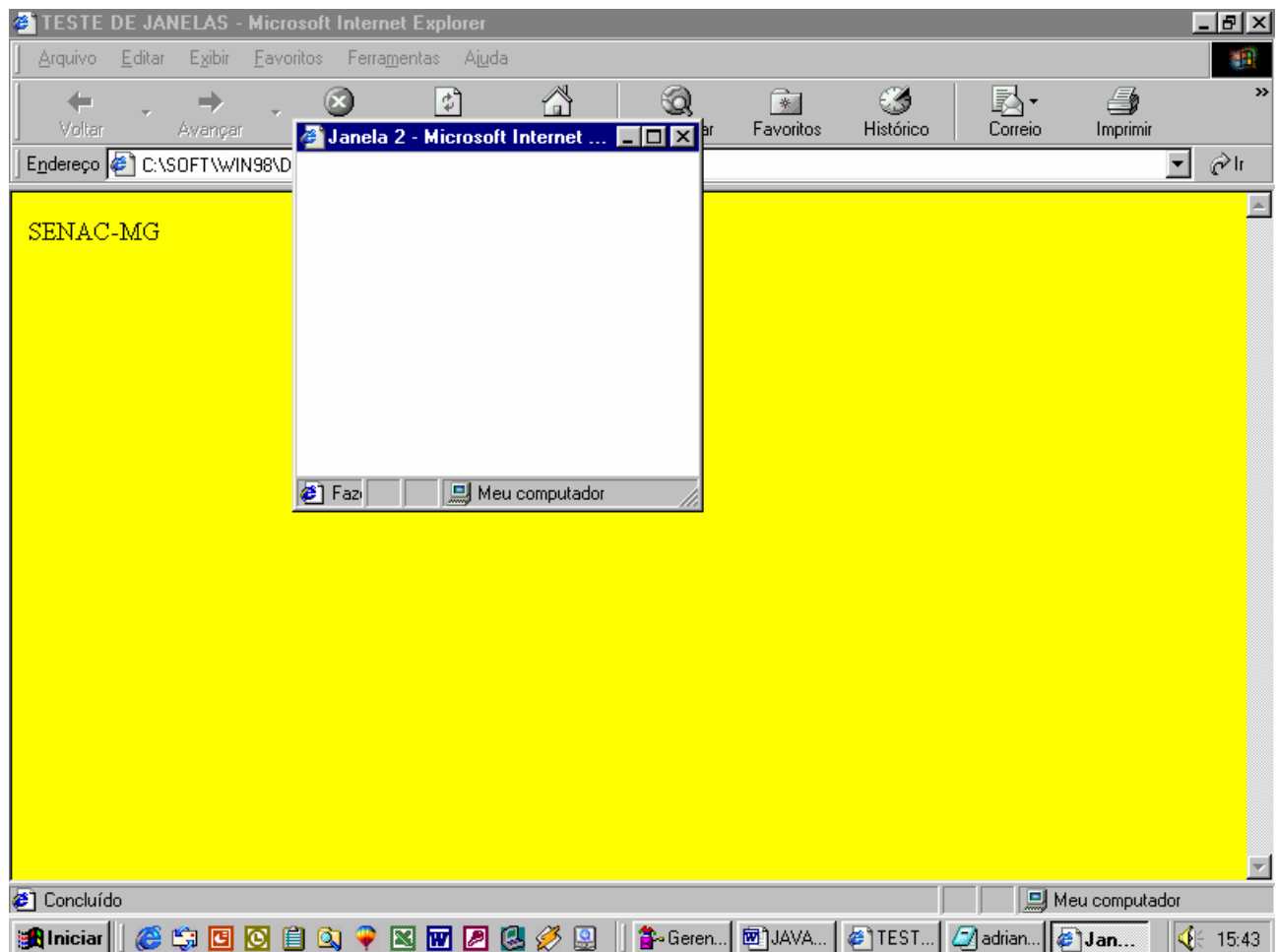

```
<HTML>
<HEAD>
    <title>TESTE DE JANELAS</TITLE>
    <script>
        janela2=window.open("", "", "status=yes,width=250,height=200")
        janela2.document.write("<HEAD><TITLE>Janela 2</TITLE></HEAD>")
    </script>
</head>
<body bgcolor=yellow>
    SENAC-MG

</body>
</html>
```

Neste código foi usado o objeto document atribuído a variável que representa a janela secundária **"JANELA2"** para especificar os tag's de cabeçalho e título para esta nova janela.

```
janela2.document.write("<HEAD><TITLE>Janela 2</TITLE></HEAD>")
```

Veja pelo exemplo da próxima figura que a nova janela será apresentada com o título **JANELA 2** em sua barra de título:

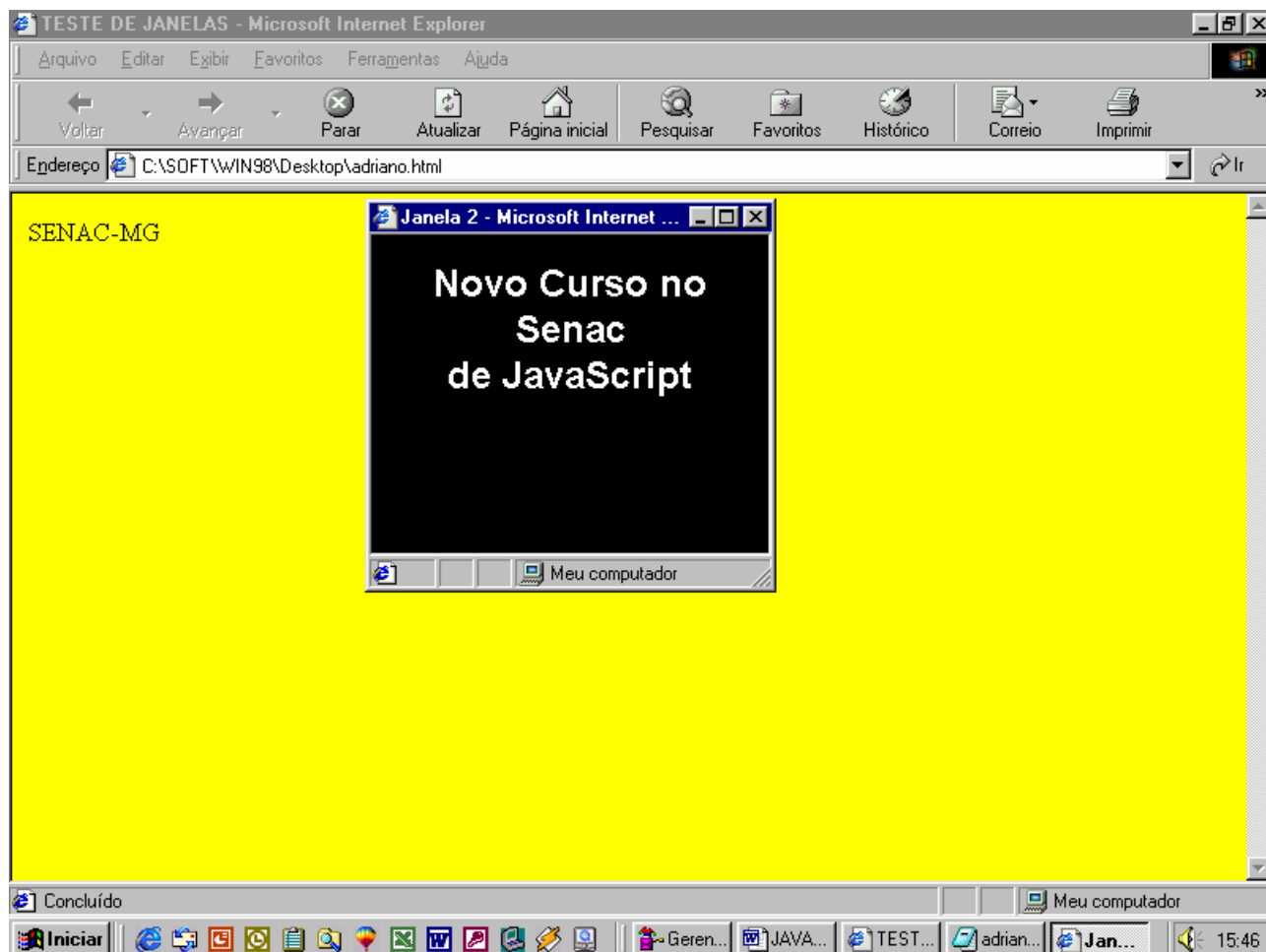


Vamos agora incrementar nosso código, controlando o corpo da nova janela com cores e até com um texto presente. Veja no código a seguir o uso do objeto **document** e seu método **write** que irá permitir o controle sobre a segunda janela aberta a partir da primeira:

Vamos adicionar ao nosso script existente a seguinte linha além da já existente:

```
janela2.document.write("<HEAD><TITLE>Janela 2</TITLE></HEAD>")
janela2.document.write("<body bgcolor=black>")
janela2.document.write("<CENTER><H2><FONT                                FACE=arial
color=white>Novo                Curso                no                Senac<BR>de
JavaScript</H2></CENTER></FONT>")
```

teremos o seguinte efeito conforme mostrado na figura a seguir:



Aproveitando a criação desta nova janela, iremos criar um botão de formulário onde atribuiremos uma ação de evento que fará o fechamento automático desta janela.

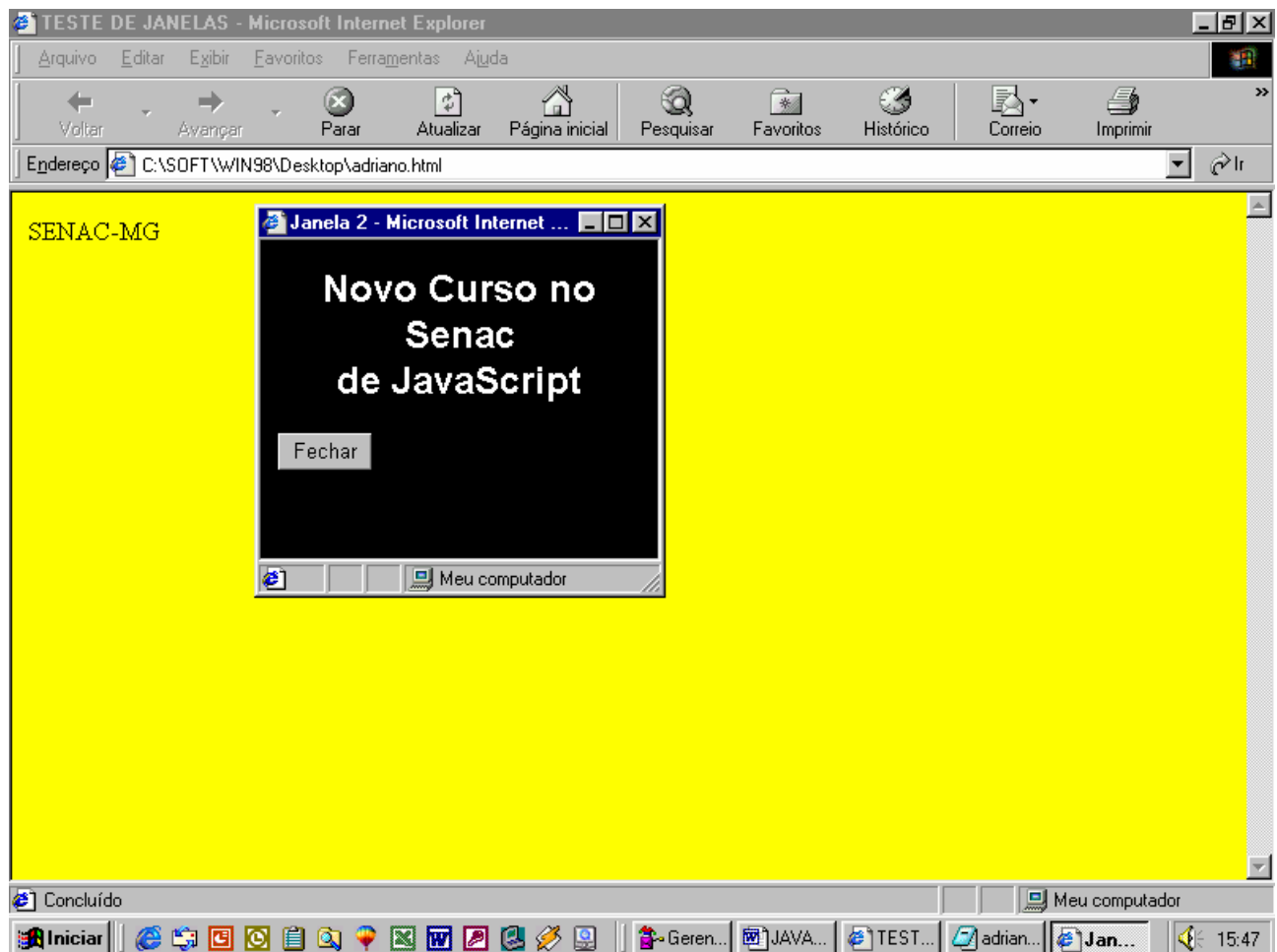
Para isto, basta utilizar o método **close** para o objeto **window**. Veja pelo exemplo do código a seguir:

```

<HTML>
<HEAD>
    <title>TESTE DE JANELAS</TITLE>
    <script>
        janela2=window.open( "", "", "status=yes,width=250,height=200")
        janela2.document.write("<HEAD><TITLE>Janela 2</TITLE></HEAD>")
        janela2.document.write("<body bgcolor=black>")
        janela2.document.write("<CENTER><H2><FONT          FACE=white>Novo
Curso no Senac<BR>de JavaScript</H2></CENTER></FONT>")
        janela2.document.write("<INPUT          TYPE=button          NAME=fecha
VALUE=Fechar Janela onClick=window.close()>")
    </script>
</head>
<body bgcolor=yellow>
SENAC-MG
</body>
</html>

```

o método **onClick** foi utilizado para acionar o objeto **window** assim que o usuário clicar sobre este botão. Com isto, ele executará o método **close** que tem a função de fechar a janela onde ele se encontra. Veja pelo exemplo da figura a seguir:



Criaremos agora na janela principal um novo botão com a finalidade de abrir uma nova janela, para isto deve-se definir o botão no corpo da janela principal conforme mostrado no código a seguir:

```
<body bgcolor=yellow>
SENAC-MG<br>
<form name="abre">
<input type="button" name="botao1" value="Abrir Janela" onClick=
janela2=window.open("", "", "status=yes,width=250,height=200")
```

Para que se abra a mesma janela que foi fechada, é necessário que se crie uma função para aproveitar o código já utilizado.

ABRINDO PÁGINAS EM FULLSCREEN (Tela Cheia)

Através do argumento **fullscreen** o usuário poderá abrir a janela do browser em tela cheia, fazendo-o ocupar toda a área do monitor. O código a seguir permite ao usuário abrir sua página em tela cheia

```
<script>
<!--
function Remote() {
var remote = null
remote
window.open('', 'vRemote', 'toolbar=no,location=no,directories=no,status=no,menubar=no,scrollbars=yes,resizable=no,fullscreen=yes')
if (remote != null) {
if (remote.opener == null) {
remote.opener = self
}
remote.location.href = 'http://www.aglima.ezdir.net'
}
}
Remote();
history.go(-1);
// -->
</script>
```

O OBJETO MATH

Este objeto é utilizado para realizar operações matemáticas. Estas operações podem ser aritméticas, funções trigonométricas, funções de arredondamento e comparação. A sintaxe de utilização dos métodos deste objeto seguem a seguinte sintaxe:

```
Math.método(valor)
```

Ou

```
with (Math){  
  
método(valor)  
  
}
```

PROPRIEDADES DE CÁLCULO DO OBJETO MATH

Veja na tabela abaixo a relação das propriedades do objeto **Math**:

PROPRIEDADES	DESCRIÇÃO
E	Constante de Euler e a base dos logaritmos naturais (próximo de 2,118).
LN2	Logaritmo natural de 2.
LN10	Logaritmo natural de 10.
LOG2E	Logaritmo na base 2 de E.
LOG10E	Logaritmo na base 10 de E.
PI	Equivalente numérico de PI, arredondado para 3,14.

SQRT1_2	Raiz quadrada de um meio.
SQRT2	Raiz quadrada de 2.

No exemplo que foi utilizado a estrutura **with**, permite ao usuário utilizar uma série de métodos **math** sem a necessidade de acrescentar varios **Math.Objeto**, facilitando todo um trabalho.

MÉTODOS DO OBJETO MATH

ABS

Este método tem a função de retornar o valor absoluto de um número. Isto significa que o valor será sempre positivo. Caso seja utilizado um valor negativo à este método. Ele será retornado como positivo. Por exemplo, caso seja definido o valor **-123**, ele será convertido para **123**. Veja o exemplo abaixo:

```
<script>
valor=Math.abs(-123);
alert(valor);
</script>
```

Neste exemplo foi definido à variável **valor** o método **abs** do objeto **Math** que possui o valor negativo **-123**, em seguida foi solicitado através de uma caixa de alerta a exibição do conteúdo da variável valor que foi convertido em número positivo.

ACOS

Este método irá retornar o arco co-seno (em radianos) de um número. Vejamos o exemplo a seguir:


```
<script>
valor=Math.acos(0.12);
alert(valor);
</script>
```

O script acima irá retornar o resultado: **1.4505064444001085**

ASIN

O método **asin** retorna o arco seno (em radianos) de um valor. Veja o exemplo a seguir:

```
<script>
valor=Math.asin(0.12);
document.write(valor);
</script>
```

O script acima irá retornar o resultado: **0.12028988239478806**

CEIL

Este método retorna um inteiro maior que ou igual a um número. O resultado deste método é equivalente ao arredondamento de um número. Claro que a lógica do arredondamento de um número é que se um número é um valor positivo como por exemplo 12,6 – o resultado é 13, quando o número for um valor negativo, por exemplo –12,6 – o resultado é –12. Vejamos o exemplo do uso do método **ceil**.

```
<script>
```

```
valor=Math.ceil(12.6);  
valor2=Math.ceil(-12.6);  
alert(valor);  
alert(valor2);  
</script>
```

Os resultados retornados serão: **13** e **-12**.

COS

Este método irá retornar o co-seno (em radianos) de um número. Vejamos o exemplo a seguir:

```
<script>  
valor=Math.cos(0.12);  
alert(valor);  
</script>
```

O resultado obtido será: **0.9928086358538662**

EXP

Este método irá retornar o logaritmo do número na base E. Vejamos um exemplo:

```
<script>  
valor=Math.exp(0.0009);  
alert(valor);  
</script>
```

O resultado obtido será: **1.0009004051215273**

FLOOR

Retorna o maior inteiro menor ou igual a um número. Vejamos o exemplo:

```
<script>
valor=Math.floor(101.25);
valor2=Math.floor(-101.25);
alert(valor);
alert(valor2);
</script>
```

Com isto teremos o seguinte resultado: **101** e **-102**.

LOG

Retorna o logaritmo natural de um número (base E). Vejamos o exemplo a seguir:

```
<script>
Valor=Math.log(1.1);
alert(valor);
</script>
```

RESULTADO: **0.09531017980432493**

MAX

Retorna o maior valor entre dois números. Exemplo:

```
<script>
Valor=Math.max(5,10);
alert(valor);
</script>
```

RESULTADO: **10.**

MIN

Retorna o menor valor entre dois números. Exemplo:

```
<script>
Valor=Math.min(5,10);
alert(valor);
</script>
```

RESULTADO: **5.**

POW (base,expoente)

Retorna a base elevada à potência do expoente, por exemplo, 2 elevado à décima potência é 1024. Com o método **pow** apresenta-se os argumentos de base e de expoente. Vejamos este exemplo o seu resultado:

```
<script>
Valor=Math.pow(1024,2);
alert(valor);
</script>
```

RESULTADO: **1.048.576.**

RANDOM

Retorna um número aleatório entre 0 e 1 com até 15 dígitos. Este número aleatório é definido através do relógio do computador. Veja pelo script a seguir a apresentação de um número aleatório:

```
<script>
```

Link para a documentação:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random

```
alert(Math.random());  
</script>
```

ROUND

Com o método **round** é possível arredondar um valor. O arredondamento segue a regra de arredondamento mostrada anteriormente. Vejamos o exemplo:

```
<script>  
valor=Math.round(125.6);  
alert(valor);  
</script>
```

Existe uma função que permite colocar o número de casas decimais. Ex:
const num = 0.5
num.toFixed(2)// vai ficar 0.50

SIN

Este método retorna o seno de um número. Veja o exemplo a seguir:

```
<script>
valor=Math.sin(1.6);
alert(valor);
</script>
```

RESULTADO: **0.9995736030415051.**

SQRT

Retorna a raiz quadrada de um número.

```
<script>
valor=Math.sqrt(49);
alert(valor);
</script>
```

RESULTADO: **7.**

TAN

Retorna a tangente de um número.

```
<script>
valor=Math.tan(1.5);
alert(valor);
</script>
```

RESULTADO: **14.101419947171718.**

ANOTAÇÕES:

OBJETO DATE

Através do objeto **Date** o usuário poderá manipular datas e horas. Observe pela sintaxe adiante a criação de um novo objeto de data.

PROPRIEDADE

Este objeto é de nível superior.

SINTAXE

```
Variável=new Date();
```

MÉTODOS GET DO OBJETO DATE

Os métodos **getDate**, **getDay** entre outros têm a finalidade de recuperar a data e hora de um objeto **date**. Ele poderá conter a data e hora atuais ou específicas. Após a especificação do objeto **date**, basta especificar qual o método veja pelos exemplos apresentados abaixo:

```
Data=new Date();  
alert(Data.getDay()) // Retorna o dia atual.  
alert(Data.getMonth()) // Retorna o mês atual.  
alert(Data.getYear()) // Retorna o ano atual.
```

MÉTODOS	DESCRIÇÃO
getDate	Dia da semana (0=Domingo).
getDay	Dia do mês.
getHours	Horas (0 a 23).
getMinutes	Minutos (0 a 59).
getMonth	Mês do ano (0=janeiro).
getSeconds	Segundos (0 a 59).
getTime	Milissegundos desde 1 de janeiro de 1990 (00:00:00).
getTimezoneOffset	Diferença de fuso horário em minutos para o GMT.
getYear	2 dígitos do ano até 1999. Após 2000, 4 dígitos.

Este objeto tem a função de armazenar a data e hora atuais no formato **mm/dd/aa hh:mm:ss**. Os valores do mês são contados de 0 até 11 e os dias da semana de 0 a 6 da seguinte forma:

0 → Janeiro	0 → Domingo	As horas são determinadas de 00:00 às 23:00
1 → Fevereiro	1 → Segunda	
2 → Março	2 → Terça	
3 → Abril	3 → Quarta	
4 → Maio	4 → Quinta	
5 → Junho	5 → Sexta	
6 → Julho	6 → Sábado	
7 → Agosto		
8 → Setembro		
9 → Outubro		
10 → Novembro		
11 → Dezembro		

O objeto **date** pode definir data e hora a partir de 1 de janeiro de 1970. Após a criação do objeto **date**, o mesmo pode ser usado com qualquer método apresentado anteriormente.

MÉTODO PARSE E UTC

O método **parse** retorna o valor de milissegundos a partir de 1 de janeiro de 1970, 00:00:00, já o método UTC faz a mesma coisa, porém no fuso horário GMT. Vejamos um exemplo da apresentação da quantidade milissegundos contados até o dia 23 de Novembro de 1975.

```
alert(Date.parse("Nov 23, 1975 GMT"));
```

Teste e veja se o resultado será **185932800000** milissegundos contados desde 1 de janeiro de 1970.

Veja outro exemplo de script apresentando na tela a hora atual, dia atual e as horas e minutos atuais.

```
<HTML>
  <HEAD>
    <TITLE>OBJETO DATE</TITLE>
  <SCRIPT>
hoje=new Date();
alert("A hora atual é "+hoje.getHours());
alert("A dia atual é "+hoje.getDay());
alert("Agora são: "+hoje.getHours()+":"+hoje.getMinutes());
</SCRIPT>
  </HEAD>
```

MÉTODOS SET DO OBJETO DATE

Os métodos **setDate**, **SetDay** entre outros, permitem ao usuário definir a data e a hora de um objeto **date**. Estes métodos são utilizados da mesma forma dos métodos **get**. Vejamos a relação destes métodos:

MÉTODOS	DESCRIÇÃO
setDate	Dia da semana (0 para Domingo).
setHours	Horas (0 a 23).
setMinutes	Minutos (0 a 59).
setMonth	Mês do ano.
setSeconds	Segundos (0 a 59).
setTime	Milissegundos de 1 de janeiro de 1990.
setYear	Ano.

MÉTODO TOGMTSTRING

A definição de GMT é **Greenwich Mean Time**, que define o fuso horário internacional padrão para configuração de relógios. Este método faz a conversão de um objeto date para uma string usando convenções GMT.

Veja pelo exemplo a seguir, a conversão da hora atual em uma string no formato GMT. Certifique-se que o computador esteja com a definição de fuso-horário correta.

```
alert(data.toGMTString());
```

O resultado, será a criação de uma string no formato:

Fri, 21 Sep 2001 20:53:44 UTC

MÉTODO TOLOCALESTRING

O método **toLocaleString** tem a função de formatar a data e a hora usando as convenções de string no computador local. Por exemplo: USA, Reino Unido, França, Alemanha, entre outros. A idéia principal deste método é apresentar ao usuário a data e hora de forma que o mesmo possa interpretar de maneira simples na página, mesmo estando fora de sua localidade. Veja o exemplo de utilização deste método:

```
alert(data.toLocaleString());
```

O resultado esperado, será similar ao formato: **09/21/2001 17:58:03**

Vejamos agora um exemplo que irá apresentar um relógio digital na barra de status que fará a hora se atualizar de um em um segundo. Analise o código apresentado abaixo:

```
<html>
  <head>
    <script>
      function relógio(){
        tempo=new Date();
        hora=tempo.getHours();
        min=tempo.getMinutes();
        sec=tempo.getSeconds();
        if(sec<10){
          sec="0"+sec;
        }
        defaultStatus=hora+":"+min+": "+sec;
        setTimeout("relógio()", "1000");
      }
    </script>
    <body onLoad="relógio()">
```

ANOTAÇÕES:

EXERCÍCIOS

Crie um script que apresente a data atual do computador na tela e em seguida exiba as horas, minutos e segundos atuais.

Crie um script que exiba o número do dia da semana, assim como, o número do mês atual.

Crie um script que exiba na tela a data e hora atuais no seguinte formato:

Agora são: hh:mm:ss do dia dd-mm-yy

Crie um script que apresente a data e hora no formato do fuso horário default.

Altere este script fazendo a data e hora apresentarem-se no formato atual do fuso horário local.

Crie um script que apresente na página a seguinte condição:

Se horas for menor que 12h, exiba "BOM DIA";

Se horas estiver entre 12h e 18h, exiba: "BOA TARDE";

Se horas estiver entre 00h e 05h, exiba: "BOA MADRUGADA".

Crie um script que apresente na barra de status a data e as horas sendo atualizado de 1 em 1 segundo. Veja o formato a ser apresentado na barra de status:

Segunda-feira, 23 de Novembro de 2001. Agora são: hh:mm:ss

OBJETO DOCUMENT

O objeto **document** é responsável por conter diversas informações sobre o documento corrente. Veja suas propriedades e eventos utilizados:

PROPRIEDADE

Este objeto é uma propriedade do objeto **window**.

SINTAXE

```
document.propriedade  
document.método( )
```

Veja na tabela abaixo a lista de propriedades do objeto **document**.

PROPRIEDADES DO OBJETO DOCUMENT

PROPRIEDADES	DESCRIÇÃO
alinkColor	Especifica o valor do atributo ALINK, que determina a cor do link acionado.
anchors[]	Lista todas as âncoras em um documento.
bgColor	Especifica o valor do atributo BGCOLOR, que determina a cor de fundo da página.
cookie	Especifica uma string contendo uma parte da informação que é armazenado no micro do usuário.

defaultStatus	Especifica um texto para a barra de status do navegador.
fgColor	Especifica o valor do atributo TEXT, que apresenta a cor dos textos presentes no documento.
forms[]	Array que contém todos os formulários do documento.
lastModified	Apresenta a data da última modificação feita no documento.
linkColor	Especifica o valor do atributo LINK, que determina a cor dos links não visitados.
links[]	Array que contém os hiperlinks do documento.
location	Especifica a URL completa do documento corrente.
referrer	Especifica a URL que originou o documento corrente.
status	Especifica o texto atual da barra de status do navegador.
title	Especifica o valor do atributo TITLE do documento.
vlinkColor	Especifica o valor do atributo VLINK, que determina a cor dos links visitados.

Assim como ao abrir uma janela do browser, é criado um objeto chamado **window**, com ele é também criado um objeto denominado **document**. Grande parte dos objetos em uma página são diretamente propriedades do objeto **document**, um exemplo claro disto é o objeto **form** que possui várias propriedades, porém ele mesmo é uma propriedade do objeto **document**. Veja a seguir a utilização de algumas das propriedades apresentadas:

Neste exemplo é apretnada uma mensagem de alerta que exibe o código HEXADECIMAL da cor de fundo definida para a página.

```
<script>
alert(document.bgColor);
</script>
```

No exemplo a seguir foi usada a propriedade **bgColor** no evento **onMouseover** que mudará a cor de fundo do documento assim que o ponteiro do mouse passar sobre o nome de uma cor

```
<html>
<body>
<script>
<!--
function mudaCor(cor){
document.bgColor=cor;
}
</script>
<body>
<pre>
<a href onmouseover="mudaCor('blue')">Azul</a>
<a href onmouseover="mudaCor('azure')">Azul Fraco</a>
<a href onmouseover="mudaCor('lightblue')">Azul Claro</a>
<a href onmouseover="mudaCor('red')">Vermelho</a>
<a href onmouseover="mudaCor('green')">Verde</a>
<a href onmouseover="mudaCor('lightgreen')">Verde Claro</a>
<a href onmouseover="mudaCor('pink')">Rosa</a>
<a href onmouseover="mudaCor('silver')">Cinza</a>
<a href onmouseover="mudaCor('purple')">Púrpura</a>
<a href onmouseover="mudaCor('orange')">Laranja</a>
<a href onmouseover="mudaCor('magenta')">Magenta</a>
<a href onmouseover="mudaCor('yellow')">Amarelo</a>
<a href onmouseover="mudaCor('black')">Preto</a>
</pre>
<body>
</html>
```

Já neste outro exemplo foi criado quatro botões de radio que ao clicar sobre um dos botões, é mudada a cor de fundo da página.

```
<html>
<body>
```

```
<PRE>
<input type="radio" name=grupo
onClick="document.bgColor='blue'">Azul
<input type="radio" name=grupo
onClick="document.bgColor='red'">Vermelho
<input type="radio" name=grupo
onClick="document.bgColor='yellow'">Amarelo
<input type="radio" name=grupo
onClick="document.bgColor='silver'">Cinza
</pre>
</body>
</html>
```

MÉTODOS DO OBJETO DOCUMENT

Veja a relação dos métodos utilizados no objeto **document**.

MÉTODO	DESCRIÇÃO
clear	Limpa a janela (window).
close	Fecha o documento atual.
write	Permite a inclusão de texto no corpo do documento.
writeln	Permite a inclusão de texto no corpo do documento com um caractere de nova linha anexado.

MÉTODO CLEAR

Com o método **clear()** do objeto **document** é possível ao usuário limpar o conteúdo de uma janela.



IMPORTANTE: É bom saber que este método não funciona no Internet Explorer, sendo compatível apenas para o Netscape a partir de sua versão 2. Veja um exemplo de utilização deste método:

```
<html>  
<head><title>MÉTODO CLEAR</title></head>
```

```
<body>
```

Texto incluso no corpo de um documento HTML.

```
<input type=button name="teste" value="Limpar Página!"  
onClick="document.clear()"
```

```
</body>
```

```
</html>
```

MÉTODO CLOSE

O método **close()** do objeto **document** diferente do método **clear()** tem como finalidade fechar o documento, sendo utilizado hoje em dia pelo objeto **window**, seu uso também é restrito apenas para algumas versões do Netscape.



IMPORTANTE: É bom saber que este método não funciona no Internet Explorer, sendo compatível apenas para o Netscape a partir de sua versão 2. Veja um exemplo de utilização deste método:

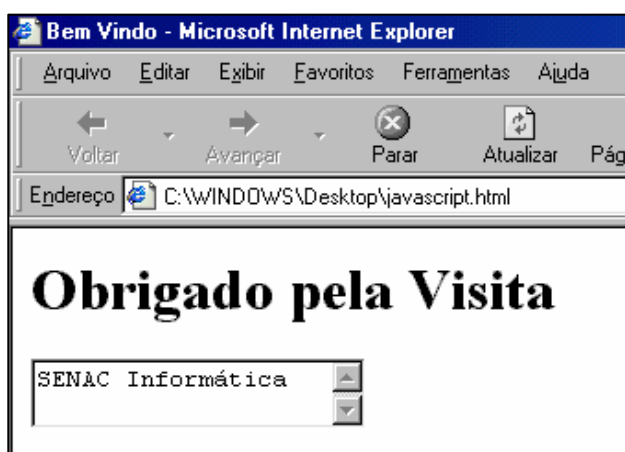
MÉTODO WRITE E WRITELN

Estes métodos permitem a inserção de texto em documento. Estes métodos são bem similares, diferenciando-se que o método **writeln** acrescenta um caractere de nova linha ao final de uma string. Normalmente este caractere é ignorado pela linguagem HTML, com exceção dos Tag's <PRE> e <TEXTAREA>. É certo que o método mais utilizado em JavaScript é o método **write** do objeto **document**. Sua sintaxe básica tem a seguinte composição:

```
document.write(texto);  
document.writeln(texto);
```

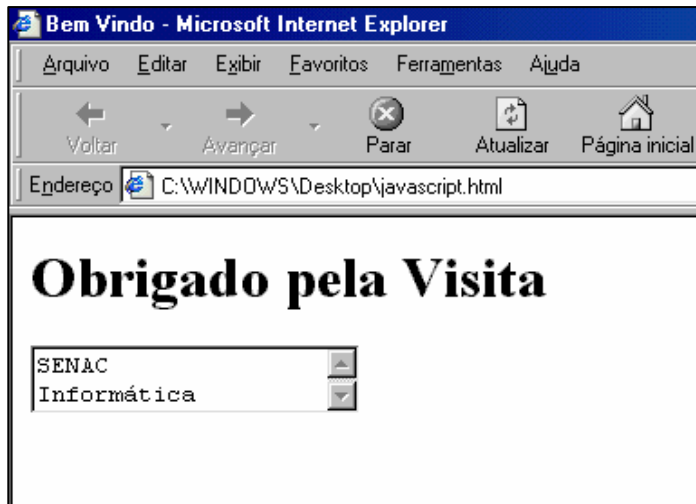
Caso o texto seja uma string, o mesmo deverá estar entre aspas. Com estes métodos o usuário poderá criar uma página inteira utilizando o JavaScript. Mas caso seja da pretensão do usuário incluir nestes textos tag's HTML, basta utilizá-los envolvendo o texto em questão veja pelo exemplo a seguir:

```
document.write("<TITLE>Bem Vindo</TITLE>");  
document.write("<H1>Obrigado pela Visita</H1>");  
document.write("<TEXTAREA>SENAC");  
document.write("Informática</TEXTAREA>");
```



Observe agora o uso do método **writeln** que permitira ao texto criado no tag <TEXTAREA> a quebra de linha entre eles:

```
document.writeln("<TEXTAREA>SENAC");  
document.writeln("Informática</TEXTAREA>");
```



Veja outro exemplo a seguir da apresentação de um cálculo sendo exibido através do método **write**.

```
<SCRIPT>  
document.write("O resultado de 5+2 é: ",5+2);  
</script>
```

Veja pelo exemplo anterior que o cálculo foi separado da string com uma vírgula, com isto, o browser entende que deverá efetuar o mesmo e apresentar o seu resultado no documento.

EXERCÍCIOS

Crie em uma página HTML um script que mostre em uma caixa de alerta a cor de fundo da página.

Crie um script que exiba uma caixa de alerta que mostre a data da última modificação.

Crie um script que exiba uma caixa de alerta que mostre a URL completa da página.

Crie um script que exiba uma caixa de alerta que mostre o título presente na barra de título.

Crie um arquivo externo com a extensão **.JS** e desenvolva o seguinte script:

Crie um script que apresente no documento corrente, a data da última modificação feita no mesmo. Crie uma página HTML e faça uma chamada para o arquivo **.JS** que contém o arquivo. Observe em qual dos dois arquivos é feita a atualização da data após a sua alteração.

Através da linguagem HTML, atribua uma cor sólida qualquer como fundo da página. Feito isto, faça um script que apresente escrito na página o código HEXADECIMAL da cor de fundo definida.

Crie um script sobre esta página que quando o usuário sair da mesma, seja exibida uma mensagem de alerta para o usuário.

Crie um script que apresente no corpo do documento o seu nome completo e no mesmo script apresente Tag's HTML em negrito, fonte e cor.

Crie um script que seja apresentado no documento a Data e Hora atuais no seguinte formato:

Seja Bem Vindo à Minha Home Page!
Agora são: hh:mm, do dia dd/mm/aa

Crie um script que quando a página for carregada execute uma função chamada **TESTE()** que possui uma rotina que exibe uma caixa de alerta com o texto: **"Obrigado pelo Sua Visita"**.

Crie um SCRIPT que ao abrir a página, seja solicitado a digitação do nome do usuário, dentro desta função, crie uma condição que enquanto o nome não for digitado, seja solicitado continuamente.

E nesta mesma função, crie uma rotina que quando o tamanho do nome em caracteres for superior à 10, seja exibida uma caixa de alerta informando à quantidade de caracteres que o nome possui. Em seguida, outro alerta avisando que somente é aceitável nomes até 10 caracteres. Com isto, faça-o retornar ao início da função criada.

Crie uma página HTML com qualquer texto sendo feito sem o código JavaScript

Nesta mesma página crie quatro botões de formulário cada um com o nome de uma cor qualquer.

Faça com que ao clicar sobre um dos botões, seja alterado a cor de fundo da página.

Crie nesta mesma página mais quatro botões cada um com o nome das cores já utilizadas nos primeiros quatro botões.

Para estes botões, faça com que ao clicar sobre um deles seja alterada a cor do texto da página.

Crie uma função que ao abrir a página, seja solicitado à entrada de um número de 1 até 20. Em seguida a entrada de outro número de 1 até 20. Caso seja digitado um valor superior à 20. Seja exibido um alerta informando que o número "x" é superior à 20.

Continuando a questão anterior, faça com que no corpo da página, seja exibido o texto: **"A multiplicação do número: "x" com o número "y" resulta em: "x*y"**. Faça com que o resultado seja apresentado em vermelho.

Crie uma página com dois hiperlinks com os textos **"SENAC-MG (www.mg.senac.br)"** e **"SENAC-BRASIL (www.senac.br)"**.

Faça com que ao movimentar o ponteiro do mouse sobre um dos hiperlinks criados. Seja exibida uma segunda janela apenas com barra de status apenas e com dimensões 300x300 pixels.

Faça com que estas janelas apresentem os seguintes textos:

Janela do SENAC-MG → Portal de Serviços do SENAC Minas.

Janela do SENAC-BRASIL → Portal de Informações do Senac Brasileiro.

Determine que quando o ponteiro estiver fora do hiperlink, as janelas respectivas sejam fechadas.

Criar um script que ao carregar a página, seja solicitado as informações:
NOME, PESO, ALTURA, COR DOS OLHOS, COR DOS CABELOS, SEXO.

Para o campo **Cor dos Olhos**, o usuário poderá digitar apenas as opções:
AZUIS, CASTANHOS, VERDES, OUTRA.

Para o campo **Cor dos Cabelos**, o usuário poderá digitar apenas as opções:
LOIROS, CASTANHOS, RUIVOS, OUTRA.

Para o campo **Sexo**, o usuário poderá digitar apenas os dados **MASCULINO** ou **FEMININO.**

Faça com que os valores digitados sejam apresentados no corpo da página, sendo que os dados alfanuméricos sejam apresentados em letras maiúsculas.

Crie uma página HTML sem conteúdo.

Crie um arquivo externo com um script que solicite a digitação do nome do usuário, e em seguida, solicite o nome de uma cor.

Crie dentro deste script uma rotina condicional que se a cor for igual à azul, a página HTML, deverá possuir a cor azul como fundo, caso a cor seja igual à verde, a página HTML deverá assumir a cor verde, caso a cor seja igual à vermelho, a página HTML deverá assumir a cor vermelha e caso a cor seja igual à amarelo, a página HTML deverá assumir a cor amarela no fundo.

Feito isto, faça com que na página HTML, seja mostrado o código HEXADECIMAL da cor assumida.

OBJETO LINK

PROPRIEDADES DO OBJETO LINKS

PROPRIEDADES	DESCRIÇÃO
hash	Especifica o texto seguido da simbologia “#” em um URL.
host	Contém o hostname:port de um URL.
hostname	Especifica o host e o domínio (endereço IP) de um URL.
href	Especifica o URL inteiro.
length	Determina o número de âncoras de um documento.
pathname	O path atual do URL.
port	Especifica a porta lógica de comunicação obtida da URL.
protocol	Especifica o protocolo da URL.
search	Especifica a porção search da URL.
target	Determina o link de destino.

UTILIZANDO ARRAYS

Primeiramente, saiba que um **ARRAY** é um grupo de itens que são tratados como uma única unidade. Um exemplo disto, é o grupo de meses do ano estarem dentro de um array chamado meses. Os elementos de um array podem ser strings, números, objetos ou outros tipos de dados.

Para que se possa declarar um array, use a seguinte sintaxe:

NomeArray = new Array(numElementos)

Veja como declarar um array chamado meses e seus elementos.

Meses = new Array(12)

Outra maneira, é declarar os valores para o novo array criado. Observe a sintaxe abaixo:

Meses = new Array("janeiro","fevereiro","março","abril","maio", "...)

Quando atribuído o número de elementos no array, é necessário declarar os elementos que farão parte do mesmo. Utilize a seguinte sintaxe:

NomeArray[numElemento]

Meses[0]=janeiro

Meses[1]=Fevereiro

Meses[2]=março

E assim por diante...

ANOTAÇÕES:

Veja pelo exemplo do script abaixo a apresentação da data atual presente no navegador:

```
<script>
// Array com os dias da semana

hoje=new Date();
semana=new
Array("Domingo","Segunda","Terça","Quarta","Quinta","Sexta")

// Array com os meses do ano

meses=new
Array("Janeiro","Fevereiro","Março","Abril","Maio","Junho","Julho"
,"Agosto","Setembro","Outubro","Novembro","Dezembro");
```

```
document.write("Hoje                                     é:  
",semana[hoje.getDay()],",",meses[hoje.getMonth()],",      de      ",  
hoje.getYear())  
</SCRIPT>
```

Neste exemplo foi declarado uma variável chamada **hoje** que define seu conteúdo como valores de **data** e criados dois arrays, o primeiro chamado **semana** e o outro chamado **meses**. Cada um dos arrays possui como conteúdo os dias da semana e os meses do ano.

Finalizando, foi utilizado o objeto **document.write** que apresentará a variável **hoje** com o array correspondente da variável **semana** de acordo com seu método **getDay()** que apresenta o valor especificado do dia da semana. Ocorrendo o mesmo com a variável **meses** para os meses do ano.

Veja agora outro exemplo da utilização dos arrays fazendo com que seja criado vários campos de formulário de acordo com a quantidade definida pelo usuário. Neste exemplo quando o usuário carrega a página é solicitado a quantidade de campos que deseja criar, para isto foi definido o seguinte código:

```
<form name="form1">  
<script>  
nome=prompt("digite a quantidade","");
```

Em seguida foi criado um laço **for** que caso o valor da variável **i** for menor que a quantidade referenciada na variável **nome**, será incrementado o valor de **nome** dentro da variável **i**. analise o código a seguir:

```
for(i=0;i<nome;i++){  
    document.write("<br>Nome",[i],":<input type=text  
name=campo",[i],">");
```

Para a execução do laço foi definido que será criado no documento atual um campo de formulário do tipo texto e a variável de cada campo criado que aqui chamada de **campo**, receberá cada uma o valor de **i** cada vez que o laço se repete. Com isto serão criados os campos cada um nomeado da seguinte forma:

Se o usuário informar 5 campos, serão criados cinco campos cada um chamado de: **campo0, campo1, campo2, campo3, campo4**. Lembre-se que um array sempre inicia-se a partir de 0. faça um teste e veja o resultado obtido.

Criaremos agora fora do script um botão de formulário que ao clicar sobre ele, será exibido em um caixa de alerta o valor que o usuário digitou em um determinado campo. Analise o código a seguir:

```
<input type="button" value="ok"  
onClick="alert(form1.campo3.value)">
```

Veja a seguir o código completo:

```
<html>  
<body>  
<form name="form1">  
<br>  
<script>  
nome=prompt("digite a quantidade","");  
for(i=0;i<nome;i++){
```



```
document.write("<br>Nome",[i],":<input type=text  
name=campo"+[i], ">");  
}  
</script>  
<br>  
<input type="button" value="ok"  
onClick="alert(form1.campo3.value)">
```

Após análise do código anterior, crie uma rotina que faça com que quando o usuário deixar a variável **nome** vazia ou nula, seja solicitado novamente a digitação do valor e que as variáveis criadas apresentem valores à partir de 1 e não de 0.

ARRAY ANCHORS

Este array lista todas as âncoras existentes em um documento. Este objeto possui a propriedade **length** e é uma propriedade do objeto **document**.

SINTAXE:

```
document.anchors.length
```

Veja um exemplo de um script que informará a quantidade de âncoras existentes no documento.

```
<html>  
  <head>  
    <title>ARRAY ANCHORS</title>  
  </head>  
  <body>  
    <A NAME=1>primeira âncora</a>  
    <A NAME=2>segunda âncora</a>
```

```
<A NAME=3>terceira âncora</a>
<A NAME=4>quarta âncora</a>
<script>
<!--
ancoras=document.anchors.length;
alert("Esta página possui "+ancoras);
// -->
</script>
```

No script apresentado, foi definido na página quatro âncoras a partir do tag HTML **<A NAME>** e em seguida já no script, foi criada a variável **ancoras** que contará a quantidade de âncoras existentes na página através da propriedade **length** e logo depois, é executado a instrução **alert** que tem a função de exibir uma mensagem na tela informando o conteúdo da variável **ancoras**.

ARRAY ELEMENTS[]

O array **elements[]** tem a finalidade de listar todos os controles de um formulário. Sua sintaxe tem a seguinte formação:

```
document.NomeForm.elements[x].propriedade;
```

```
document.NomeForm.elements.length;
```

x é o número de elementos presentes dentro do formulário também iniciado com zero.

No exemplo a seguir, foi criado um código que tem a função de selecionar uma lista de caixas de verificação de um formulário quando é acionado um botão. Observe o código:

```
<script>
```

```

function seleciona(){
    itens=document.form1.elements;
    for(i=0;i<itens.length;i++){
        document.form1.elements[i].checked=true;
    }
}
function tira(){
    itens=document.form1.elements;
    for(i=0;i<itens.length;i++){
        document.form1.elements[i].checked=false;
    }
}
</script>

```

Neste exemplo, foi criado uma função chamada **seleciona()** que cria uma variável que receberá os elementos do formulário **form1**. Em seguida, foi criado um laço **for** que somará a variável **i** a quantidade de elementos presentes no formulário, onde cada elemento deverá receber para sua propriedade **checked** o valor verdadeiro, ou seja, selecionar a caixa de verificação.

Logo mais, foi criada uma função chamada **tira()** que tem a função contrária da função **seleciona()**. Faça um teste e veja o que acontece.

No script abaixo é apresentado uma função que exibe o dia da semana mais as horas sendo atualizadas de um em um segundo:

```

<html>
<head>
<script>
    function relogio(){
        tempo=new Date();

```

```

dia=new Array( "Domingo", "Segunda-feira", "Terça-feira", "Quarta-
feira", "Quinta-feira", "Sexta-feira", "Sábado" );
hora=tempo.getHours();
min=tempo.getMinutes();
sec=tempo.getSeconds();
if(sec<10){
    sec="0"+sec;
}

defaultStatus=dia[tempo.getDay()]+", "+hora+": "+min+": "+sec;

setTimeout("relogio()", "1000");

```

Map: O método MAP serve para mapear um array. Mapear ?

Sim, com o map, você pode percorrer posição por posição de um array e criar um novo array, alterando tudo que você quiser do array original.

/* O Map aceita até 3 parâmetros:const novoArray = arrayOriginal.map((valorAtual, indice, arrayOriginal) => xxx)

1) O Map vai passar por todos os itens do array, e o primeiro parâmetro é cada item, um por vez do array que estamos mapeando. Item OBRIGATÓRIO.

2) O índice é a posição atual que estamos mapeando. Item OPCIONAL.

3) Uma cópia do array original. Item OPCIONAL.*/

```
const numbers = [1, 2, 3, 4];
```

```
const double = numbers.map((num) => num * 2);
```

```
// double ficou assim... [2, 4, 6, 8];
```

```
// numbers continua... [1, 2, 3, 4];
```

Reduce: O método REDUCE serve para 'reduzir' um array a apenas um item. Como assim ? Ele vai passar item por item no array e no final vai restar apenas um valor.

/* O Reduce aceita até 4 parâmetros:const novoArray = arrayOriginal.reduce((acumulador, valorAtual, índice, arrayOriginal) => { return xxxxx }, valorInicial);

1) O acumulador, na primeira iteração terá o valor inicial que daremos a ele.

Já nas demais iterações, ele terá o valor que iremos acumular nele. Item OBRIGATÓRIO.

2) O valor do atual elemento sendo iterado. Item OBRIGATÓRIO.

3) O índice do elemento atual. Item OPCIONAL.

4) O array original. Item OPCIONAL.*/

```
const numbers = [1, 2, 3, 4, 5];
```

```
const total = numbers.reduce((acumulador, atual) => { return acumulador = acumulador + numero;}, 0) // repare nesse 0. Ele é o valor inicial que o acumulador receberá*/
```

O que aconteceu aqui ? Definimos um valor inicial para o acumulador. Então ele começou a somar número a número dentro do array.

Na primeira iteração foi assim: acumulador = acumulador(0) + atual(1) -> Agora acumulador valerá 1

Na segunda iteração, agora acumulador vale 1.

```
acumulador = acumulador(1) + atual(2) -> Agora acumulador valerá 3*/
```

```
// total é igual a 15;
```

```
// numbers continua... [1, 2, 3, 4];
```

Filter: O método FILTER serve para filtrar um array. Ele passará por todos os valores do array e você decidirá quais valores vão para seu novo array, e quais vão ser descartados.

/* const novaArray = arrayOriginal.filter((valorAtual, indice, arrayOriginal) => { seu código aqui});

1) O filter vai passar por todos os itens do array, e o primeiro parâmetro é cada item, um por vez do array que estamos filtrando. Item OBRIGATÓRIO. 2

) O índice é a posição atual que estamos filtrando. Item OPCIONAL.

3) Uma cópia do array original. Item OPCIONAL.

A cada item, fazemos uma 'pergunta' ao código. Se a resposta for verdadeira naquele item, ele guardará o valorAtual no novo array. Caso seja falso, o valor será descartado */

```
const numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
const pares = numeros.filter(valorAtual => valorAtual % 2 === 0 );
```

```
// pares ficou assim... [2, 4, 6, 8, 10];
```

```
// numbers continua... [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
const list = [20, 3, 234, 12, 17, 541, 6, 87, 275, 1000]
```

```
const newList = list.filter( number => {
    return number > 100 ? true : false
})
```

```
// OU const newList = list.filter( number => number > 100)
```

```
console.log(newList)
```

Podemos utilizar o 3 métodos numa mesma array

```
const companies = [
  { name: 'Samsung', marketValue: 50, CEO: 'Kim Hyum Suk', foundedOn: 1938 },
  { name: 'Microsoft', marketValue: 415, CEO: 'Satya Nadella', foundedOn: 1975 },
  { name: 'Intel', marketValue: 117, CEO: 'Brian Krzanich', foundedOn: 1968 },
  { name: 'Facebook', marketValue: 383, CEO: 'Mark Zuckerberg', foundedOn: 2004 },
  { name: 'Spotify', marketValue: 30, CEO: 'Daniel Ek', foundedOn: 2006 },
  { name: 'Apple', marketValue: 845, CEO: 'Tim Cook', foundedOn: 1976 }
]

// Adicionar 10% de valor de mercado a todas as companhias -> MAP
// Filtrar somente companhias fundadas abaixo de 1990 -> Filter
// Somar o valor de mercado das restantes -> Reduce

const marketValueOldCompanies = companies.map(company => {
  company.marketValue *= 1.1

  return company
}).filter(company => company.foundedOn < 1990).reduce((acc, company) => company.marketValue + acc, 0)

console.log(marketValueOldCompanies)

/* #####Outra forma de aninhar#####
const add10Percent = company => {
  company.marketValue *= 1.1

  return company
}

const filterCompanies = company => company.foundedOn < 1990

const calculateTotalMarketValue = (acc, company) => acc + company.marketValue

const marketValueOldCompanies = companies
  .map(add10Percent)
  .filter(filterCompanies)
  .reduce(calculateTotalMarketValue, 0)
*/
```

Crie um link HTML nesta página que vá para outra página.

Na nova página, crie um botão de formulário que tenha a mesma função de voltar do navegador.

Desenvolva uma página simples com cinco links como os apresentados abaixo:

GLOBO → www.globo.com

UOL → www.uol.com.br

AOL → www.americaonline.com.br

SENAC-BRASIL → www.senac.br

FACULDADES SENAC → www.sp.senac.br/faculdades

EDITORA SENAC → www.senac.br/editora

Crie um evento sobre cada hiperlink que faça com que sempre que o usuário movimentar o ponteiro do mouse sobre o link, seja exibida uma mensagem na barra de status.

Crie outro evento sobre cada hiperlink que faça com que ao usuário movimentar o ponteiro do mouse para fora do hiperlink, seja exibida outra mensagem.

Crie um evento no corpo desta página para que quando o usuário deixar a página, seja exibida uma caixa de alerta com a mensagem **"Prazer em Conhecê-lo."**

Utilize o evento **onClick** para exibir uma mensagem quando o usuário clicar com o botão do mouse sobre cada um dos hiperlink's.

Defina uma segunda ação para o evento **onClick** de cada hiperlink, fazendo-os exibir outra mensagem de alerta com o seguinte texto:

“Aguarde o Carregamento...”

ANOTAÇÕES:

MANIPULANDO FRAMES

Como já conhecido na linguagem HTML, os frames são divisões entre páginas que são visualizadas pelo navegador, cada frame é também chamado de **quadro** que podem ser em linhas ou colunas e possuir tamanhos variados. Este recurso é muito útil para fazer uma página de índice onde o usuário escolhe um determinado link no menu e visualiza seu conteúdo em outro quadro presente no navegador, sem a necessidade de sair do menu.

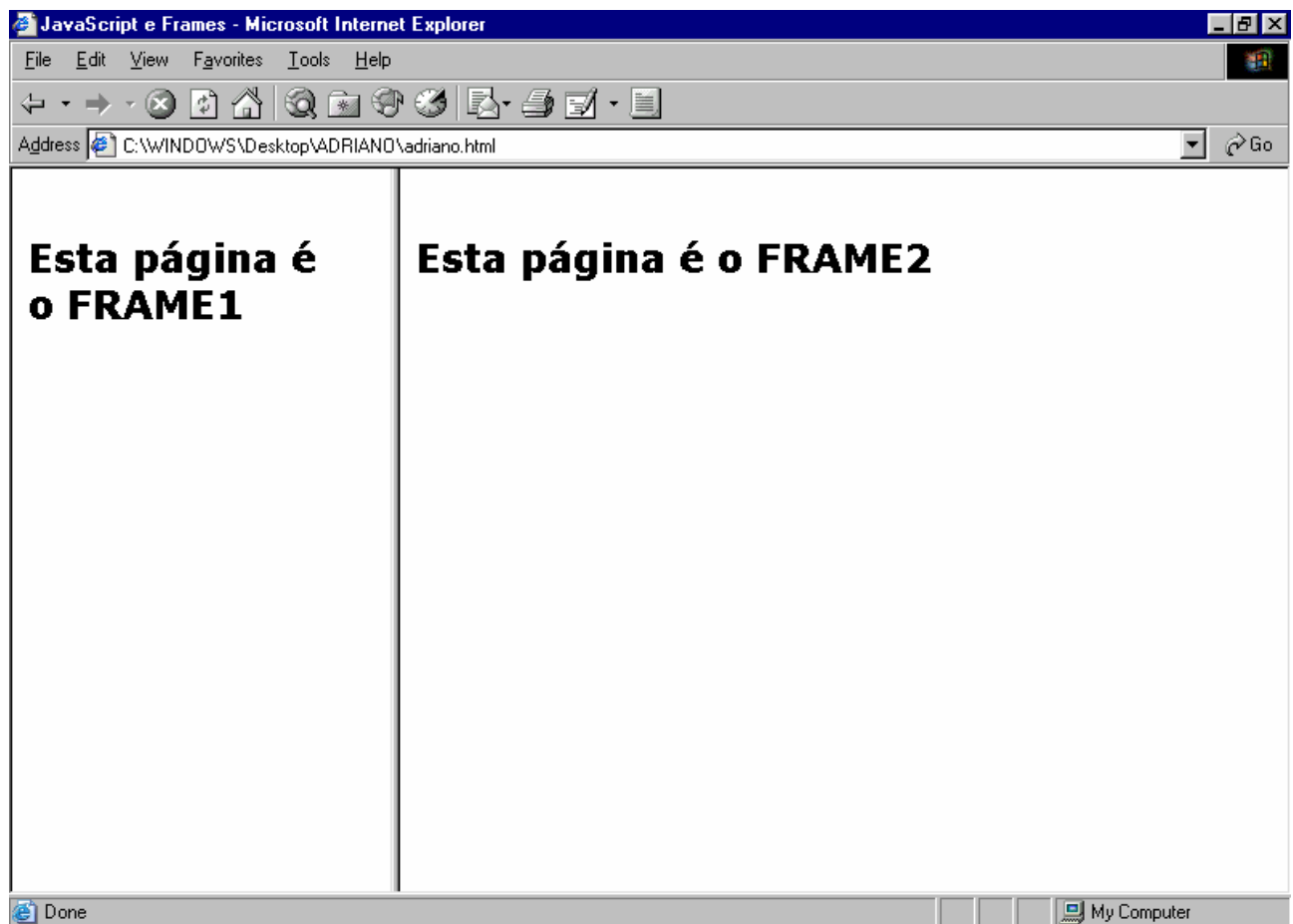
Teoricamente, os frames visualmente parecem simples de criar, porém são bastante complexos que podem fazer com que o usuário se confunda dependendo do projeto que esteja desenvolvendo. Espero que você tenha aprendido isto de forma clara e praticado bastante este recurso, porque inicialmente parece bem confuso, por isso, pratique mesmo a sua utilização para que seu entendimento seja claro.

Pois bem, como já sabido, os frames possuem uma estrutura única que não mostra nada na sua página, sua função básica é dividir a tela da sua maneira e exibir em cada quadro uma página específica, portanto, esta página não possui corpo. O tag de corpo de uma página de frame é substituído pelo tag de configuração de frames chamado **<FRAMESET>**. Vejamos a seguir um bem simples de utilização de frames:

```
<html>
<head>
<title>JavaScript e Frames</title>
</head>
<frameset cols="30%,*">
    <frame src="frame1.html" name="frame1">
    <frame src="frame2.html" name="frame2">
</frameset>
```

</html>

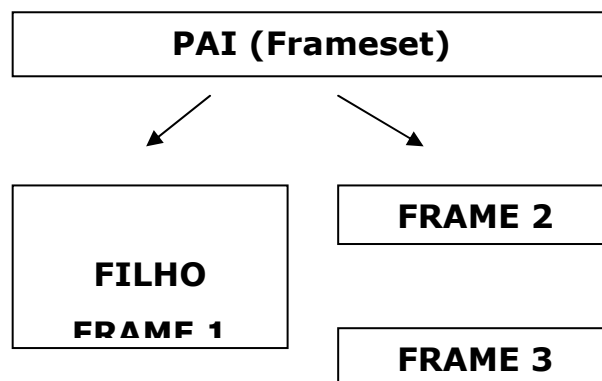
Neste exemplo foi criado um frame de duas colunas onde a primeira possui 30% de largura do navegador e a segunda o restante. Em seguida, definiu-se através do tag **<FRAME SRC=>** os arquivos que iriam aparecer em cada frame assim que a página fosse carregada, dando a seguinte apresentação:



Um detalhe importante na utilização de scripts em páginas com frames, é a nomeação dos frames através do atributo **NAME**, no exemplo anterior os dois frames criados foram chamados de **frame1** e **frame2**.

HIERARQUIA FRAMESET WINDOW

É bom saber que, cada frame é considerado uma janela separada para o navegador, em razão de cada uma possuir seu próprio código HTML. Estas janelas de frame estarão sempre subordinadas pela janela principal, aquela que contém o tag **<FRAMESET>**. Como dito anteriormente, esta janela não possui corpo sua função é dividir a tela e mostrar suas páginas filho. Entenda então que a página que contém o tag **<FRAMESET>** é a página PAI, enquanto que as que estão subordinadas à ela são as páginas FILHO.



Caso o usuário faça diversos frames aninhados, serão apresentadas janelas netas que são subordinadas as janelas do frame-filho, com isto temos a seguinte sintaxe no JavaScript:

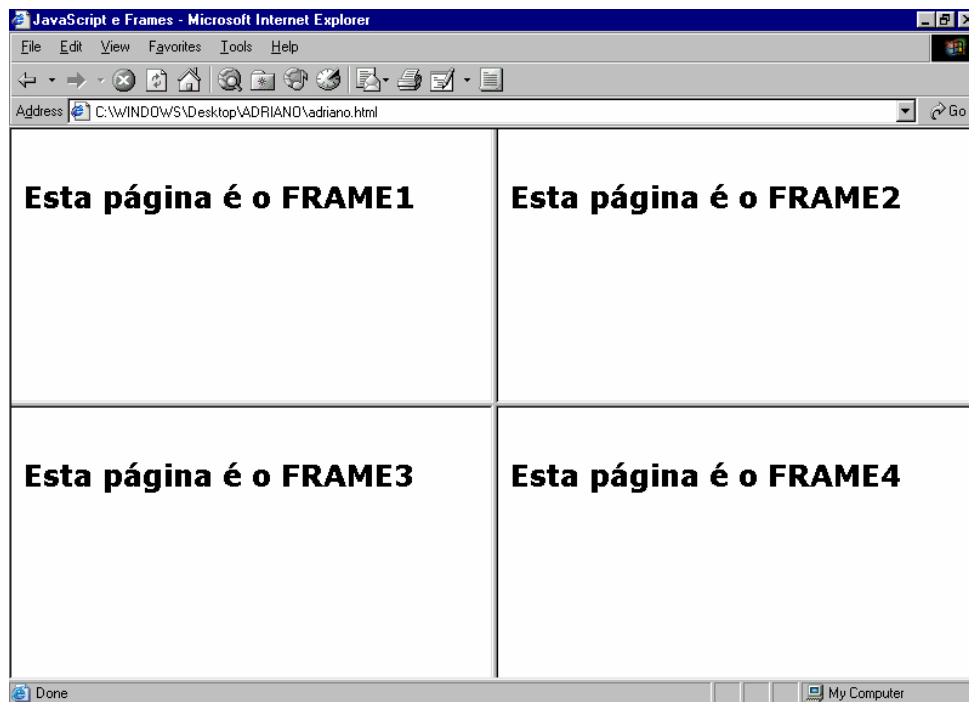
```
parent.filho.neto
```

Concluindo, a página filho é nome da janela que está subordinada a janela principal, a do frameset (pai). Já a página neto é o nome de uma janela que está subordinada sob a janela de frame-filho.

Vejamos a utilização de código JavaScript para o gerenciamento dos frames dividindo o navegador em 4 frames nomeados da seguinte forma: frame1, frame2, frame3 e frame4 conforme exemplo mostrado na figura a seguir:

```
<html>
<head>
<title>JavaScript e Frames</title>
</head>
<frameset rows="50%,*">
<frameset cols="50%,*">
    <frame src="controle.html" name="controle">
    <frame src="frame2.html" name="segundo">
</frameset>
<frameset cols="50%,*">
    <frame src="frame3.html" name="terceiro">
    <frame src="frame4.html" name="quarto">
</frameset>
</html>
```

Com este código teremos a seguinte estrutura visualizada no navegador:



Criaremos agora um arquivo de controle para nosso frame que possuirá algumas instruções JavaScript. No exemplo do código à seguir, trocaremos o frame superior esquerdo por esta página desenvolvida, veja o resultado na próxima figura:

```
<HTML>
<HEAD>
<SCRIPT>
function checaFrame(frameNum){
    if (frameNum==1){
        alert(parent.controle.name);
    }else if(frameNum==2){
        alert(parent.segundo.name);
    }else if(frameNum==3){
        alert(parent.terceiro.name);
    }else if(frameNum==4){
        alert(parent.quarto.name);
    }
}
```

```

function escreveFrame(frameNum){
    if(frameNum==2){
        parent.segundo.document.write("<br>Segundo Frame");
    }else if(frameNum==3){
        parent.terceiro.document.write("<br>Terceiro Frame");
    }else if(frameNum==4){
        parent.quarto.document.write("<br>Quarto Frame");
    }
}

function limpaFrame(){
    for (i=1;i<4;i++){
        parent.frames[i].document.close();
        parent.frames[i].document.open();
    }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<table>
<tr valign="top"><td>
<INPUT TYPE="BUTTON" VALUE="FRAME1" onClick="checaFrame(1)"><br>
<INPUT TYPE="BUTTON" VALUE="FRAME2" onClick="checaFrame(2)"><br>
<INPUT TYPE="BUTTON" VALUE="FRAME3" onClick="checaFrame(3)"><br>
<INPUT TYPE="BUTTON" VALUE="FRAME4" onClick="checaFrame(4)"><br>
</td><td>
<INPUT          TYPE="BUTTON"          VALUE="ESCREVE          FRAME2 "
onClick="escreveFrame(2)"><br>
<INPUT          TYPE="BUTTON"          VALUE="ESCREVE          FRAME3 "
onClick="escreveFrame(3)"><br>
<INPUT          TYPE="BUTTON"          VALUE="ESCREVE          FRAME4 "
onClick="escreveFrame(4)"><br>

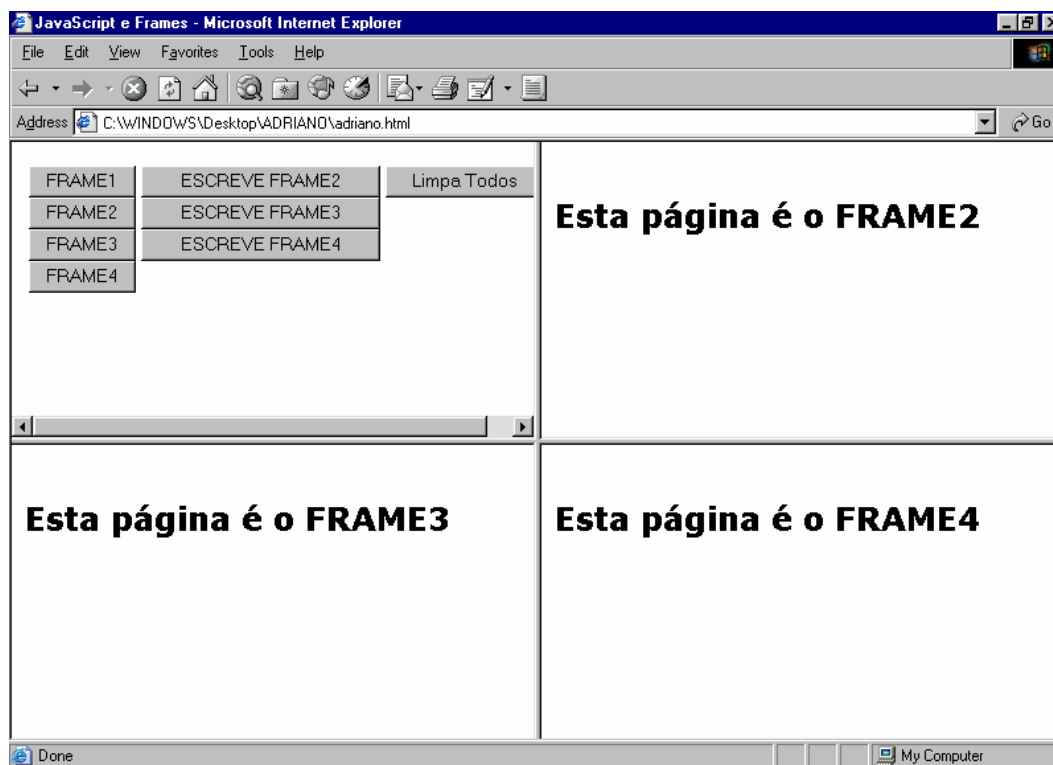
```

```

</td><td>
<INPUT                TYPE="BUTTON"                VALUE="Limpa                Todos "
onClick="limpaFrame( )"><br>
</td></tr>
</table>
</FORM>
</BODY>
</HTML>

```

Abrindo novamente o arquivo principal dos frames, esta nova página chamada **controle.html**, será aberta no primeiro frame, observe pelo exemplo da figura a seguir:



Analisando nosso controle, foi criado três funções, cada qual com suas rotinas a serem executadas, vejamos os detalhes destas funções:

Função **checaFrame()**

Nesta função foi solicitado que caso o usuário clique em um botão que está chamando a função, será aberta uma mensagem de alerta com o nome do frame atual definido no arquivo do frameset. Isto foi possível pela linha de código:

```
alert(parent.nomeFrame.name);
```

O objeto **parent**, especifica o documento que possui o **frameset**.

A propriedade **nomeFrame**, especifica o frame que está sendo referenciado.

A propriedade **name** define o nome para esta janela.

Função escreveFrame()

Esta função que também é chamada com o clique sobre um botão de formulário irá permitir que o usuário possa escreve algum texto em um frame específico, isto é possível através do código:

```
parent.nomeFrame.document.write("<br>Este é um Frame");
```

O objeto **parent**, especifica o documento que possui o **frameset**.

A propriedade **nomeFrame**, especifica o frame que está sendo referenciado.

A propriedade **document** define a janela especificada.

O método **write** foi usado para escrever no documento especificado.

Função limpaFrame()

Já esta função permite que o usuário ao clicar em um botão limpe o conteúdo de todos os frames presentes. O código que permite esta ação é:

```
for (i=1;i<4;i++){
```



```
parent.frames[i].document.close();  
parent.frames[i].document.open();  
parent.frames[i].document.writeln(" ");  
}
```

Neste código foi manipulado todos os frames fazendo o uso do array **frames[]**. Analisando este código:

O objeto **parent**, especifica o documento que possui o **frameset**.

frames[] é um array.

document define a janela especificada.

close, open são métodos para o documento da janela de frames.

Cada elemento do array **frames[]** contém um frame. É feita a contagem em 0 para o primeiro frame, 1 para o segundo, 2 para o terceiro. Para que não seja omitido o frame que possui o controle, veja que a contagem foi iniciada em um, em razão do frame de controle ser o primeiro ele assume zero e conta até 3. Cada frame é limpo em cada laço da instrução **for**.

OBJETO FORM

Através do objeto **form** da linguagem JavaScript o usuário poderá interagir melhor com seus dados inseridos pelos recursos de formulários existentes na linguagem HTML, entre eles temos os campos **checkbox**, **radio** e **listas de seleção**. O formulário e seus objetos podem ser facilmente manipulados através de scripts. Formulários também podem ser usados com um programa CGI em um servidor ou para validação de dados.

PROPRIEDADE

Este objeto é uma propriedade do objeto **document**.

Veja na tabela abaixo a relação das propriedades do objeto **form**.

PROPRIEDADES DO OBJETO FORMS

PROPRIEDADES	DESCRIÇÃO
Action	Especifica a URL do servidor onde as variáveis do formulário são enviadas.
defaultChecked	Estado de seleção de uma caixa de verificação de um botão de opção.
defaultSelected	Seleção atual de lista de opções.
defaultValue	Valor padrão da caixa de texto ou área de texto em um formulário.
checked	Estado padrão de um checkbox ou botões do tipo radio em um formulário.
elements[]	Lista os elementos existentes do formulário.

encoding	Formato de código MIME para o formulário.
form	Objeto de formulário.
index	Especifica uma opção de uma lista de seleção (select) em um formulário.
length	Especifica o número de itens de uma lista.
method	Método que determina como as informações do formulário serão processadas através dos valores GET ou POST.
Name	Nomeia um objeto do formulário.
options[]	Lista de opções de uma lista (select) dentro do formulário.
selected	Estado atual de uma caixa de verificação ou um botão de opção (radio).
selectedIndex	Determina a opção selecionada de uma lista de seleção (select) dentro do formulário.
target	Especifica um alvo.
text	Especifica o texto de uma opção (option) de uma lista de seleção (select) do formulário.
value	Nome dado ao texto de uma caixa de texto ou área de texto (text e areatext).

SINTAXE

`document.NomeFormulário.propriedade`

MÉTODOS DO OBJETO FORM

MÉTODO	DESCRIÇÃO
blur()	Quando remove o foco de um campo do tipo text , textarea e campos de senha password .
click()	Quando é dado um clique sobre um elemento de botão em formulário ou campos do tipo radio e checkbox .
focus()	Quando é dado o foco sobre um campo do tipo text , textarea e campos de senha password .
select()	Quando é selecionado o conteúdo de um campo do tipo text , textarea ou password .
submit()	Quando o formulário é enviado ao servidor.

ELEMENTOS DE UM FORMULÁRIO

Já sabemos que em um formulário temos diferentes componentes que auxiliam a entrada de dados do usuário. Destacamos:

Button

Checkbox

Hidden

Password

Radio

Reset

Select

Submit

Text

TextArea

Todos estes componentes apresentados, já estamos familiarizados no uso da linguagem HTML.

OBJETO TEXT

Sabemos que se pode criar campos de preenchimento de textos com o uso do formulário, e através da linguagem JavaScript é possível a manipulação dos dados digitados para este campo com o uso do objeto **TEXT**. Sua sintaxe geral é:

```
document.nomeForm.nomeText.propriedade
```

Veja abaixo a relação das propriedades existentes para o objeto **TEXT**:

PROPRIEDADES	DESCRIÇÃO
DefaultValue	Determina o valor padrão para a caixa de texto.
Name	Determina o nome do objeto para a caixa de texto.
Value	Determina o valor para a caixa de texto.

MANIPULADORES DE EVENTO PARA FORMULÁRIOS

MÉTODO	EVENTO	DESCRIÇÃO
focus()	onFocus	Executa uma instrução quando o é dado o foco sobre o campo de texto.
blur()	onBlur	Executa uma instrução quando é retirado o foco sobre o campo de texto.
select()	onSelect	Executa uma instrução quando o conteúdo da caixa de texto é selecionado.

Vejamos um exemplo JavaScript que quando o usuário retirar o foco da caixa de texto, será exibida uma mensagem informando para não deixar o campo em branco e em outro situação após o preenchimento do campo em letras minúsculas seu conteúdo será convertido para letras maiúsculas:

```

<HTML>
  <HEAD>
    <TITLE>CHECAGEM DE DADOS</TITLE>
  <script>
    function requer(texto){
      if (texto==""){
        alert("Favor preencher o campo!")
        document.form1.nome.focus( )
      }
    }
  </script>
</head>
<body>
  <form name="form1">
    <pre>

```

Digite seu Nome:

```
<input type="text" name="nome"
onChange="this.value=this.value.toUpperCase()"
onFocus="this.select()" onBlur="requer(this.value)">
```

Digite seu Sobrenome:

```
<input type="text" name="sobrenome" onFocus="this.select()">
</form>
</body></html>
```

No script apresentado, fora criado uma função que determina se o usuário deixar o campo de preenchimento em branco será exibida uma mensagem avisando-o. Já no corpo, temos dois campos de texto que utiliza o evento **onChange** que irá colocar o conteúdo do campo texto em letras maiúsculas e o evento **onBlur** que executa a função **requer** criada no início do documento. Temos também o evento **onFocus** que seleciona o conteúdo do campo quando o usuário utiliza a tecla TAB.

Veja o resultado na figura à seguir, quando o usuário deixa o campo em



branco:

Vejamos um exemplo prático que permite que quando o usuário deixa um campo vazio é exibida uma mensagem de alerta e o cursor continua presente dentro do campo de texto:

```
function verifica(texto){
```

```
if(texto==""){  
alert("Não deixe em branco");  
return(form1.nome.focus());  
}
```

Esta função irá testar o valor da variável **texto** que caso esteja vazio é exibido um alerta e o foco retorna para dentro do campo de texto. Veja o código que chama a função no campo de formulário:

```
<input type="text" name="nome" onBlur="verifica(this.value)">
```

A chamada da função é feita quando o usuário retira o foco do campo que executa a função **verifica** que armazena o valor atual do campo. Havendo vários campos que são de preenchimento obrigatório, esta função poderá ser reutilizada.

OBJETO PASSWORD

Este objeto permite ao usuário controlar campos de preenchimento de SENHA. Sua sintaxe é:

document.nomeForm.campoSenha.propriedade

As propriedades e métodos deste objeto, são os mesmos do objeto **TEXT** apresentados anteriormente.

OBJETO TEXTAREA

Este objeto tem como objetivo a criação de áreas de texto composta por várias linhas. Sua sintaxe é:

document.nomeForm.campoTextArea.propriedade

Suas propriedades, métodos e eventos equivalem as mesmas do objeto **TEXT**.

OBJETO BUTTON

Já utilizado, sabemos que este objeto representa os botões criados em um formulário onde atribuímos ações específicas. Sua sintaxe tem a seguinte formação:

document.nomeForm.nomeButton.propriedade

MÉTODO	EVENTO	DESCRIÇÃO
click()	onClick	Executa uma instrução quando o é dado um clique sobre o botão.

Veja pelo exemplo do script a seguir a exibição de uma mensagem de alerta informando o que foi digitado na caixa de texto assim que o usuário pressiona um botão:

```
<form name="form1">
<pre>
Digite seu Nome:
<input type="text" name="campo1">
<input type="button" value="Clique Aqui" onClick="alert('Você
digitou '+form1.campo1.value) ">
</form>
```

OBJETO SUBMIT

Como já sabido, este objeto controla o botão responsável pelo envio dos dados de um formulário. Suas propriedades, métodos e eventos são equivalentes as do objeto BUTTON. Sua sintaxe básica tem a seguinte formação:

document.nomeForm.ButtonSubmit.propriedade

Veja no exemplo a seguir um script que verificará se determinado campo foi preenchido, caso não tenha sido preenchido o JavaScript não enviara o formulário:

```
<html>
<head>
<script>
function envia(){
    if (form1.campo1.value==""){
        alert("Campo em Branco");
        return(false);
    } else {
        return(true);
    }
}
</script>
</head>
<body bgcolor="yellow">
<form name="form1">
<pre>
Digite seu Nome:
<input type="text" name="campo1">
<input type="submit" value="Clique Aqui" onClick="envia()">
```

</form>

OBJETO RESET

Responsável pelo botão que retorna qualquer elemento de um formulário para seus valores default. Suas propriedades, métodos e eventos são equivalentes as do objeto BUTTON. Sua sintaxe é:

document.nomeForm.ButtonReset.propriedade

OBJETO CHECKBOX (Caixa de Verificação)

Este objeto cria uma lista onde o usuário poderá marcar várias opções. Sua sintaxe é equivalente as dos objetos de formulário anteriormente apresentadas.

PROPRIEDADE

Este objeto é uma propriedade do objeto **form**.

Veja abaixo a relação das propriedades existentes para o objeto **CHECKBOX**:

PROPRIEDADES	DESCRIÇÃO
name	Contém o conteúdo do atributo name .
value	Contém o valor "on" ou "off" que determina o estado da caixa.
status	Valor booleano que determina o estado da caixa, selecionado (True) ou não selecionado (False).

defaultStatus

Valor booleano que indica se o estado padrão do botão definido pelo atributo checked.

MANIPULADORES DE EVENTO

MÉTODO	EVENTO	DESCRIÇÃO
click()	onClick	Executa uma instrução assim que o usuário clica sobre o elemento.

Os botões de formulário do tipo **CHECKBOX** são botões que o usuário pode ativar e desativar. O atributo **checked** determina o estado default da caixa de verificação. Esta propriedade assume valores booleanos, quando ativada assume o valor **true** e desativada o valor **false**. Vejamos um exemplo:

```
<HTML>
  <HEAD>
    <TITLE>OBJETO CHECKBOX</TITLE>
  </HEAD>
<BODY>
<SCRIPT>
function exemplo(form){
  teste=form.opcao.checked;
  alert("A caixa de verificação está "+teste);
}
</SCRIPT>
<FORM>
<INPUT TYPE="checkbox" NAME="opcao">Primeira Opção
<INPUT TYPE="checkbox" NAME="opcao2">Segunda Opção
<HR>
<INPUT TYPE="button" NAME="acao" value="Execute"
onClick="exemplo(this.form)">
```

Veja um outro exemplo de utilização do objeto **checkbox**:

```

function teste(){
    if (form1.caixa1.checked){
        form1.campo1.value="caixa1"
    } else if (form1.caixa2.checked){
        form1.campo1.value="caixa2"
    } else if (form1.caixa3.checked){
        form1.campo1.value="caixa3"
    }
}
</script>
</head>
<body bgcolor="yellow">
<form name="form1">
<pre>
Digite seu Nome:
<input type="text" name="campo1">
<input type="submit" value="Clique Aqui" onClick="envia()">
<input type="checkbox" name="caixa1" onClick="teste()">caixa1
<input type="checkbox" name="caixa2" onClick="teste()">caixa2
<input type="checkbox" name="caixa3" onClick="teste()">caixa3
</form>

```

OBJETO RADIO

Similar ao objeto **CHECKBOX**, este objeto cria uma lista de opções, onde o usuário poderá escolher apenas uma única opção. Sua sintaxe, segue os mesmos parâmetros dos objetos anteriores. Vejamos a relação de suas propriedades, métodos e eventos:

Veja agora a utilização de um script no uso do objeto **radio** do formulário:

```

function acessa(){
if (form1.senacmg.checked){
    window.location.href("http://www.mg.senac.br");
}else if (form1.senacbr.checked){
    window.location.href("http://www.senac.br");
}
}
</script>
<form name="form1">
<pre>
<input type="radio" name="senacmg" onClick="acessa()">SENAC-MG
<input type="radio" name="senacbr" onClick="acessa()">SENAC BRASIL

```

Vejamos agora outro exemplo que apresenta a soma de valores de acordo com o que o usuário seleciona. Faça um teste com o código abaixo:

```

<FORM NAME=fm1>
<INPUT TYPE="RADIO" NAME="massa1" onClick="fina()">Massa fina
(10,00 reais)
<INPUT TYPE="RADIO" NAME="massa1" onClick="grossa()">Massa grossa
(10,00 reais)

<INPUT TYPE="CHECKBOX" NAME="queijo">Mussarela (+ 1,30 reais)
<INPUT TYPE="CHECKBOX" NAME="calab">Calabresa (+ 2,50 reais)
<INPUT TYPE="CHECKBOX" NAME="ovoceb">Ovo e cebola (+ 0,70 reais)

<INPUT TYPE="BUTTON" VALUE="Calcular" total"
onClick="precoTotal()">
<INPUT TYPE="TEXT" NAME="total" SIZE=50>
</FORM>
<SCRIPT LANGUAGE="JavaScript">
var massa;
var ticado = false;

```

```

function fina(){
    massa="massa fina";
    ticado=true;
}
function grossa(){
    massa= "massa grossa";
    ticado=true;
}
function precoTotal(){
    if(!ticado){
        alert("SELECIONE UM TIPO DE MASSA!");
    }
    else{
        var tot=10.00;
        if(document.fm1.queijo.checked){
            tot=tot+1.30;
        }
        if(document.fm1.calab.checked){
            tot=tot+2.50;
        }
        if(document.fm1.ovoceb.checked){
            tot=tot+0.70;
        }
        ts= new String(tot);
        tss=ts.replace(".",",");
        if(tss.lastIndexOf(",")>0){
            document.fm1.total.value="Sua pizza de "+massa+" custa:
"+tss+"0 reais";
        }
        else{
            document.fm1.total.value="Sua pizza de "+massa+" custa: "+tss+"
reais";
        }
    }
}

```



```
}  
}  
</SCRIPT>
```

OBJETO SELECT

Muito comum, este objeto representa uma lista de opções que o usuário poderá selecionar. Com o objeto **SELECT**, o usuário poderá determinar uma seleção única ou múltipla.

Este objeto irá permitir ao usuário controlar os itens de uma lista de opções criada com o tag HTML **<SELECT>**. Veja um exemplo de utilização do objeto **select**.

```
function itens(){
alert(form1.lista.selectedIndex);
}
</script>
<form name="form1">
<pre>
<select name="lista">
<option>Item 0
<option>Item 1
<option>Item 2
<option>Item 3
<option>Item 4
</select>
<input type="button" onClick="itens()" value="veja">
```

A propriedade **selectedIndex** informa qual dos itens da lista de seleção foi selecionado pelo usuário. Veja abaixo outro exemplo que ao usuário selecionar um dos itens o valor da opção redireciona para uma determinada URL:

```
function acessa(texto){
window.location.href=texto;
}
```

Foi criada uma função que possui uma variável como argumento que armazenará o valor de uma opção da lista de seleção.

```
<select name="lista"
onChange="acessa(lista.options[selectedIndex].value)">
<option value="http://www.mg.senac.br">Senac
<option value="http://www.sp.senac.br">SenacSp
</select>
```

Já na página, foi criada uma lista de seleção que ao alterar o valor da variável lista, a função acessa irá armazenar na sua variável **(texto)** o valor da opção selecionada da lista. Veja que foi definido como valor de cada opção na lista de seleção um endereço específico que será enviado para a variável da função que será usado como hiperlink na janela do browser.

EVITANDO O USO DA TECLA ENTER

Normalmente na maioria dos formulários, caso o usuário estando em algum dos campos presentes, ao pressionar a tecla ENTER, o mesmo é enviado imediatamente, muitos usuários se perdem na navegação quando este problema ocorre dependendo da situação. Através do JavaScript é possível ao usuário evitar que o mesmo seja enviado quando se pressiona a tecla ENTER.

É claro que este recurso irá cancelar o envio do formulário mesmo se o usuário clicar sobre o botão de submissão, bastando ao usuário a criação de uma função que envie o formulário. Vejamos o código a seguir para este recurso:

```
<script>
function envia(form){
    form.submit();
}
```

```
</script>
<form name="form1" action="mailto:adrianolima@mg.senac.br"
onSubmit="return false">
Digite seu Nome:
<input type="text" name="nome">
<input type="submit" value="ok" onClick="envia(this.form)">
```

Observe que foi usado o evento **onSubmit** que determina a ação a ser tomada ao enviar o formulário, foi definido o valor **return false** que evita que o formulário seja enviado, mesmo clicando em um botão.

Já para que o formulário seja enviado, foi criado um botão simples de formulário que ao ser acionado, será executada a função **envia(this.form)** para este formulário. Na função foi definido a instrução **form.submit()** que enviará o formulário.

OBJETO LOCATION

Este objeto é bem interessante por conter informações sobre a URL da página. Cada propriedade do objeto LOCATION representa uma parte diferente do endereço. A sintaxe utilizada para este objeto possui a seguinte composição:

Protocolo:hostname:port path seach hash

Vejamos o significado da representação mostrada acima.

PROPRIEDADE	DESCRIÇÃO
Hash	Determina nome de âncora.
Host	Determina parte hostname:port URL
Href	Determina uma URL
Pathname	Determina caminho.
Port	Determina a porta de comunicação que é utilizado no servidor para comunicação.
Protocol	Início de uma URL.
Search	Determina uma pesquisa.

Vejamos agora a relação dos tipos de URL conhecidas:

TIPO	URL
Web	http://www.dominio.com.br/
Local	File:///disco:/diretório/página.html
FTP:	ftp://ftp.local.com.br/diretório
Mailto:	mailto:nome@dominio.com.br
UseNet	News://news.servidor.com.br
Gopher	Gopher.servidor.com.br

PROPRIEDADES DO OBJETO LOCATION

Para definir propriedades do objeto **Location**, siga a seguinte sintaxe:

```
window.location
```

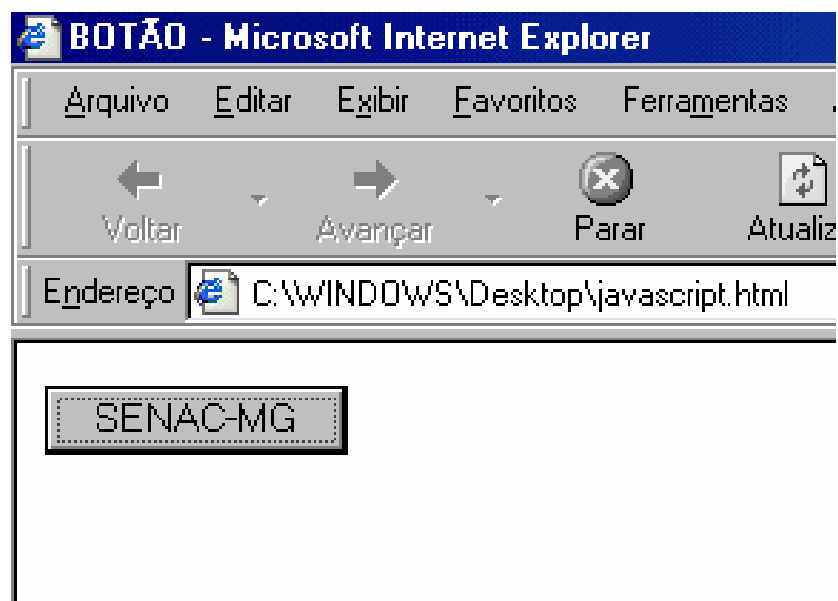
Caso o objeto **location** esteja fazendo referência a janela corrente, não é necessário utilizar o objeto **window**. Caso o usuário deseja retornar o URL da janela corrente, basta utilizar o seguinte comando:

```
location.href; // URL da janela corrente.
```

```
location.host; // Parte da URL.
```

Veja um exemplo de um hiperlink usado em um botão de formulário:

```
<HTML>
  <HEAD>
    <TITLE>BOTÃO</TITLE>
  </HEAD>
  <FORM NAME="form1">
    <input type="button" value="SENAC-MG" onClick="window.location.
href='http://www.mg.senac.br/'">
  </form>
</body>
</html>
```



EXERCÍCIOS

Crie um script que possua um campo de formulário do tipo **Texto** que solicite a digitação do nome do usuário e, quando usuário retirar o foco do campo de texto, seja exibida uma caixa de alerta exibindo o texto:

"Como vai, <NomeUsuário>

Crie dentro de um script uma função chamada **exibe** e como argumento para esta função, defina a variável **(texto)**. Desenvolva para esta rotina, uma caixa de alerta que apresente o seguinte texto: **"Olá visitante texto".**

No corpo da página, crie um campo de formulário do tipo texto, que execute a função atribuindo à variável definida o valor digitado pelo usuário assim que o mesmo retirar o foco do campo de texto.

Crie um formulário contendo os campos **Nome, Endereço, Telefone, CEP, CIDADE, ESTADO.**

Crie uma função que determine para os campos **NOME, TELEFONE e CEP** preenchimento obrigatório.

Crie uma função que determine para o campo telefone o preenchimento de valores numéricos exibindo uma caixa de alerta quando o preenchimento estiver incompatível.

Crie um script que possua dois campos de formulário do tipo texto e determine que quando o primeiro campo seja preenchido com algum valor, o segundo campo possa refletir o que a no primeiro campo.

Crie uma rotina neste exemplo que se for digitado um texto em minúsculo no primeiro campo o segundo campo apresente o texto em maiúsculo.

Crie um script que possua dois campos de formulário do tipo texto que irá armazenar valores numéricos.

Crie uma função para este script que multiplique os valores dos dois campos de texto e apresente seu resultado em um terceiro campo quando o mesmo receber um foco pelo usuário.

Desenvolva um script que possua três campos: **NOME**, **ENDEREÇO** e **SEXO**.

Faça com que quando o usuário retirar o foco do campo, seja exibida uma mensagem de alerta informando-o para não os deixarem em branco.

Crie uma rotina para o campo **SEXO** para que caso seja digitado valores diferentes de **MASCULINO** ou **FEMININO**, seja exibida uma mensagem de alerta que o valor atual não é válido. EX: "**valor atual não é um sexo válido!**". Esta rotina deverá ocorrer quando o usuário retirar o foco do campo.

Crie dois campos de formulário, o primeiro para o preenchimento de um **LOGIN** do tipo **TEXT** e o outro do tipo **PASSWORD** chamado **SENHA** e um botão de ação.

Crie uma variável chamada **AUTENTICA()** que ao clicar sobre o botão caso o login e senha sejam diferentes de aluno e 5245, seja exibida uma caixa de alerta informando que os dados preenchidos são inválidos.

Crie um formulário com um campo de formulário do tipo texto. Crie um evento que ao carregar a página, seja exibida uma mensagem na barra de status e que o cursor apareça posicionado dentro do campo de formulário.

Crie dois campos de formulário do tipo texto. Faça com que ao selecionar o texto digitado dentro do campo 1, seja mostrado seu conteúdo no campo 2.

Crie dois campos de formulário, sendo o primeiro do tipo password e o segundo do tipo texto. Crie um script que faça com que ao digitar uma senha no primeiro campo seja digitado o que a pessoa digitou. Crie um evento que ao carregar a página, o campo senha venha com o cursor posicionado.

Crie um script que seja exibida uma caixa de entrada para o usuário conforme mostrado na figura a seguir:



Faça com que esta caixa seja acionada quando o usuário clicar sobre um botão de formulário. Defina como texto padrão a string: **http://**.

Faça com que quando o usuário clicar sobre o botão OK, o navegador redirecione para o endereço digitado na caixa.

Crie em uma página HTML quatros campos de formulário do tipo **radio** e logo mais atribua um evento que faça com que ao escolher um dos botões presentes seja alterado a cor de fundo da página, como mostrado no exemplo a seguir:

- ☒ Azul
- ☐ Vermelho
- ☐ Amarelo
- ☐ Cinza

UTILIZANDO O OBJETO HISTORY

PROPRIEDADES	MÉTODOS	EVENTOS
length	back	Nenhum
	Forward	
	go	

PROPRIEDADE

Este objeto é uma propriedade do objeto **window**.

Este objeto é como um histórico que contém informações sobre as URL's que o usuário esteve. Estes endereços ficam armazenados e podem ser acessados à partir dos botões **VOLTAR** e **AVANÇAR** do Browser. Ao acessar páginas na Internet, o browser armazena as páginas anteriores em um histórico próprio. Com o objeto **HISTORY** é possível ao usuário manipular este histórico. Sua sintaxe é formada da seguinte forma:

```
history.propriedade
```

MÉTODOS BACK E FORWARD

Os métodos **back** e **forward** representam a função dos botões de **Avançar** e **Voltar** presentes no browser. Veja o exemplo de utilização destes métodos:

```
history.back() // Retorna a URL anterior.  
history.forward() // Retorna a URL posterior.
```

MÉTODO GO

O método **go** permite a especificação de uma URL do histórico, basta fornecer o número da URL que deseja ir. Veja pelo exemplo a seguir o deslocamento para a quarta URL anterior:

```
history.go(-4) // Vai para a quarta URL anterior.  
History.go(5)  // Vai para a quinta URL a frente.
```

Dentre os métodos utilizados o método **go** é mais versátil por ser utilizado para controlar o histórico amplamente.

ANOTAÇÕES:

UTILIZANDO O OBJETO NAVIGATOR

Este objeto possui informações sobre o navegador utilizado pelo usuário como, versão e nome do mesmo. Para utilizar suas propriedades, deve-se seguir a seguinte sintaxe:

navigator.propriedade

Vejamos a relação de suas propriedades:

PROPRIEDADES DO OBJETO NAVIGATOR

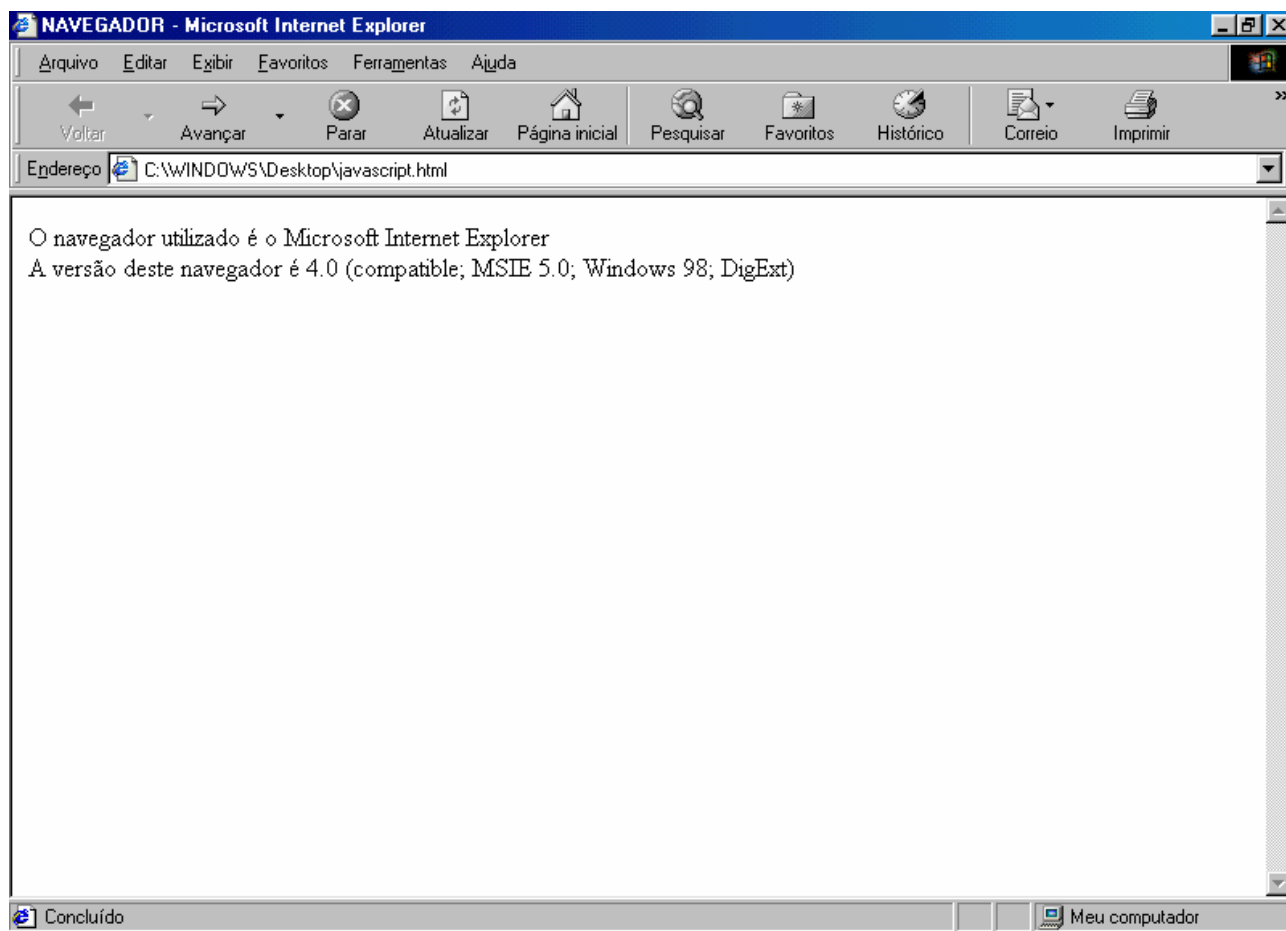
PROPRIEDADE	DESCRIÇÃO
appName	Especifica o nome do navegador utilizado.
appVersion	Especifica a versão do navegador utilizado.
userAgent	Especifica uma string do vendedor do navegador.

Vejamos agora um exemplo que mostrará informações sobre o navegador utilizado.

```
<HTML>
  <HEAD>
    <TITLE>NAVEGADOR</TITLE>
  </HEAD>
  <script>
```

```
document.write("O navegador utilizado é o  
"+navigator.appName+"<br>")  
document.write("A versão deste navegador é "+navigator.appVersion)  
</script>  
  
</body>  
</html>
```

veja o resultado deste script pela figura a seguir:



ACESSANDO CÓDIGO-FONTE A PARTIR DE UM LINK

Alguns usuários para facilitar a visualização do código-fonte de sua página, criam hiperlinks que fazem com que seus visitantes tenham uma maneira mais simples de visualizar seu código. Veja o exemplo a seguir:

```
<script>
function openWin(){
    location="view-source:"+window.location;
}
</script>
<body>
<a href="#" onClick="openWin()">Veja o código desta página</a>
```


UTILIZANDO COOKIES

Quem nunca ouviu falar em “**cookies**”? Um termo muito falado para os usuários que trabalham diretamente com internet, porém muito pouco entendido.

Os Cookies são pequenos textos (de geralmente 1Kb), colocados em seu disco rígido por alguns sites que você visitou.

Eles contêm informações que o próprio internauta forneceu ao site como e-mail, preferências, o que você comprou, seu nome, etc...Mas apenas o que o internauta forneceu. Se ele apenas entrou no site e não digitou informação nenhuma, então o cookie não conterá nenhuma informação.

Alguns sites de comércio eletrônico colocam estes cookies no disco rígido do usuário com o objetivo de personalizar os próximos atendimentos. Por exemplo, o usuário entrou em uma loja virtual e comprou o livro “E o vento Levou “. Pagou com cartão de crédito e forneceu seu nome e mais alguns dados para que a compra pudesse ser raizada.

Em seu próximo acesso a este mesmo site, este usuário receberá uma mensagem em sua tela dizendo: “Bom dia Fulano de Tal, que tal conhecer “E o vento Levou 2 ? “. Ou seja, o atendimento foi personalizado para este usuário. Ele foi reconhecido e um livro que provavelmente será de seu agrado lhe foi oferecido.

Assim o cliente pode ser atendido de acordo com seu perfil e suas preferências, e o site terá uma maior probabilidade de vender outro livro. Este tipo de operação envolvendo cookies e personalizando o atendimento visa criar um vínculo com o cliente com o objetivo que este volte outras vezes ao site.

Nos sites de comércio eletrônico, os cookies também são utilizados para criar os carrinhos de compras. Digamos que o usuário esteja num site fazendo

compras e de repente, por algum motivo, cai sua conexão... Acontece que ele já encheu seu carrinho com um monte de coisas. Será que o site vai perder esta venda? Pois, mesmo se o cliente voltar, será que ele terá paciência para comprar tudo outra vez?

Graças aos cookies está tudo bem. Se o cliente retornar ao site e quiser continuar de onde parou, os cookies “lembrarão” o que tinha dentro do carrinho e o cliente não precisará começar tudo de novo.

Apenas como esclarecimento, os cookies não transmitem vírus e podem ser lidos apenas por aqueles que o colocaram no hard disk do usuário, evitando o tráfego aberto de informações pela rede.

Outra utilidade dos cookies é fornecer informação sobre número, frequência e preferência dos usuários para que se possa ajustar a página de acordo com o gosto do internauta.

Criando Cookies

O *cookie* é uma propriedade do objeto **window.document** e possui uma restrição numérica de 300 cookies no total e no máximo 20 cookies por site, além de um tamanho máximo de 4 KB, embora estes números possam variar conforme a versão do browser.

Quando gravamos um *cookie*, portanto, apenas inserimos uma variável *string* que contém os valores desejados em um arquivo *cookie* que é associado ao nosso documento. O exemplo simplificado abaixo nos mostra como os *cookies* operam:

```

<script language="JavaScript">
<!--
function DefineCookie(nome, valor, form){
    document.cookie = nome+"="+valor+";";
    form.Nome.value = "";
    form.Valor.value= "";
}

function ExibeCookie(form) {
    form.Resultado.value = document.cookie;
}
// -->
</script>

```

Note que usamos apenas 3 funções, com os objetivos de inserir valores dentro de um *cookie*, mostrá-lo ou reiniciá-lo. Note que a propriedade **document.cookie** é, de fato, uma string, com a convenção de que cada *cookie* seja separado do outro por um ";" como consta na função **DefineCookie()**.

Deve ficar claro, portanto, que para armazenarmos vários pares <valor, informação> deveremos manipular a string do *cookie* para obtermos o valor desejado. Para isso, usamos a função abaixo, capaz de nos devolver o valor corrente de um *cookie*:

```

function GetCookie(nome) {
    var dc = document.cookie;
    var prefixo = nome + "=";
    var inicio = dc.indexOf(prefixo);
    if (inicio == -1) return null;

    var final = document.cookie.indexOf(";", inicio);
    if (final == -1) final = dc.length;
}

```

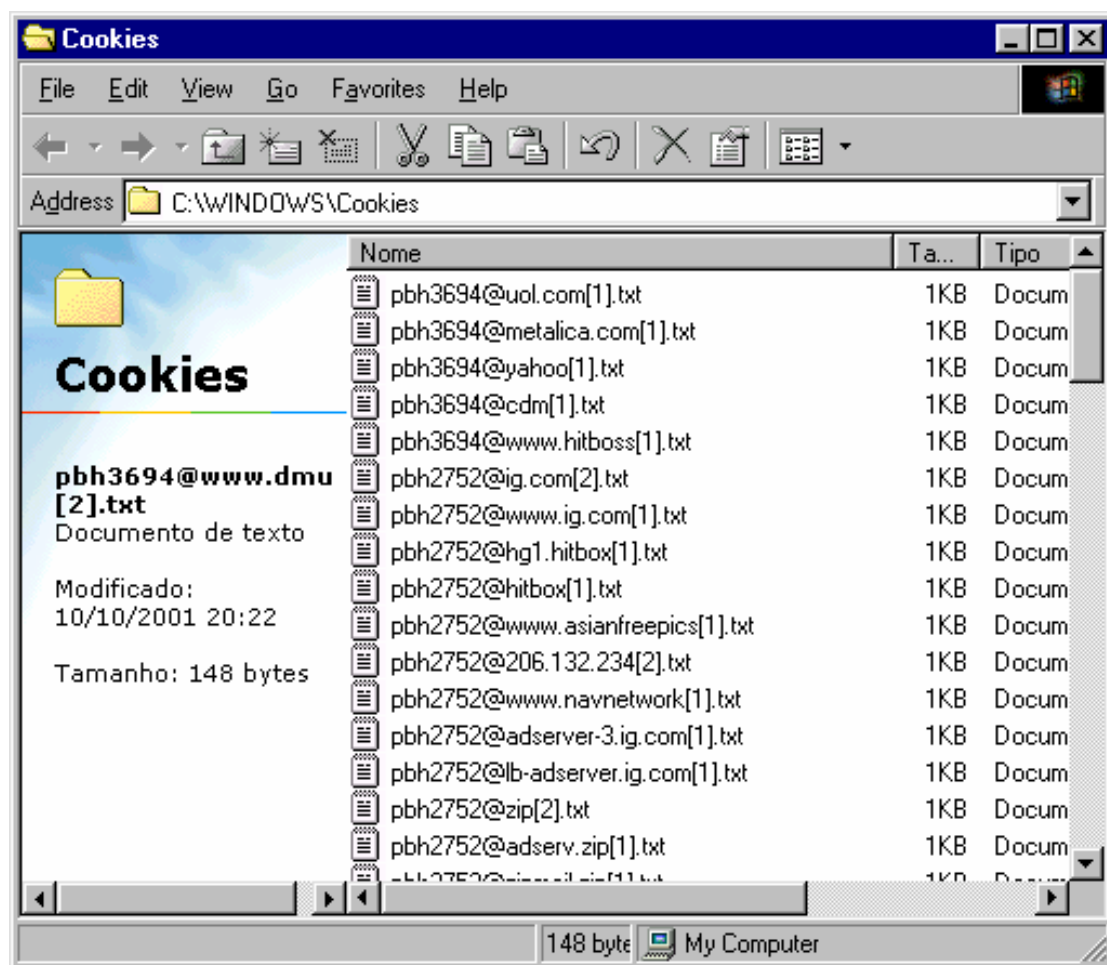
```
    return unescape(dc.substring(inicio+prefixo.length, final));  
}
```

Vamos agora fazer algo mais interessante, como contar o número de vezes que um determinado usuário visitou nossa página. O script abaixo mostra o uso de um botão que contará o número de vezes que o mesmo for clicado, o que certamente pode ser feito no momento da página ser carregada, exibindo uma mensagem do tipo "Olá, FULANO, que bom que você voltou! Já é a N-ésima vez que você nos visita!"

```
function ContarVisitas(form) {  
  
    visitas=GetCookie("Visitas");  
  
    if(!visitas) {  
  
        visitas = 1;  
  
        form.Contador.value="Esta é o seu primeiro click!"  
  
    } else {  
  
        visitas = parseInt(visitas) + 1;  
  
        form.Contador.value="Você já clicou " + visitas + " vezes";  
  
    }  
  
    document.cookie="Visitas="+visitas+";";  
}
```

Faça seus testes e veja como ocorre.

Veja uma relação de cookies armazenados na subpasta **Cookie** presente na pasta **Windows** conforme mostrado na figura a seguir:



Os nomes de cookies seguem um padrão do login da máquina acompanhado do local do domínio ou parte no restante do nome do arquivo. A sintaxe utilizada no conteúdo do arquivo do cookie, segue o seguinte padrão:

```
nome=valor;expires=data-no-formato-GMTString;path=/
nome1=valor1; nome2=valor2; nome3=valor3
```

Vejamos outro exemplo que irá criar dois campos de texto e um Botão. Será colocado o **nome** do cookie em um campo de texto e o **valor** em outro campo de texto. Clicando sobre o botão, será armazenado no computador do usuário um registro com nome/valor, num arquivo com a origem da página.

```
function gravaCookie(){  
    var dataexp = new Date ();  
    dataexp.setTime (dataexp.getTime() + (24 * 60 * 60 * 1000 * 1));  
    //vai valer por 1 dia  
    setCookie (document.fml.nome.value,    document.fml.valor.value,  
dataexp);  
}
```

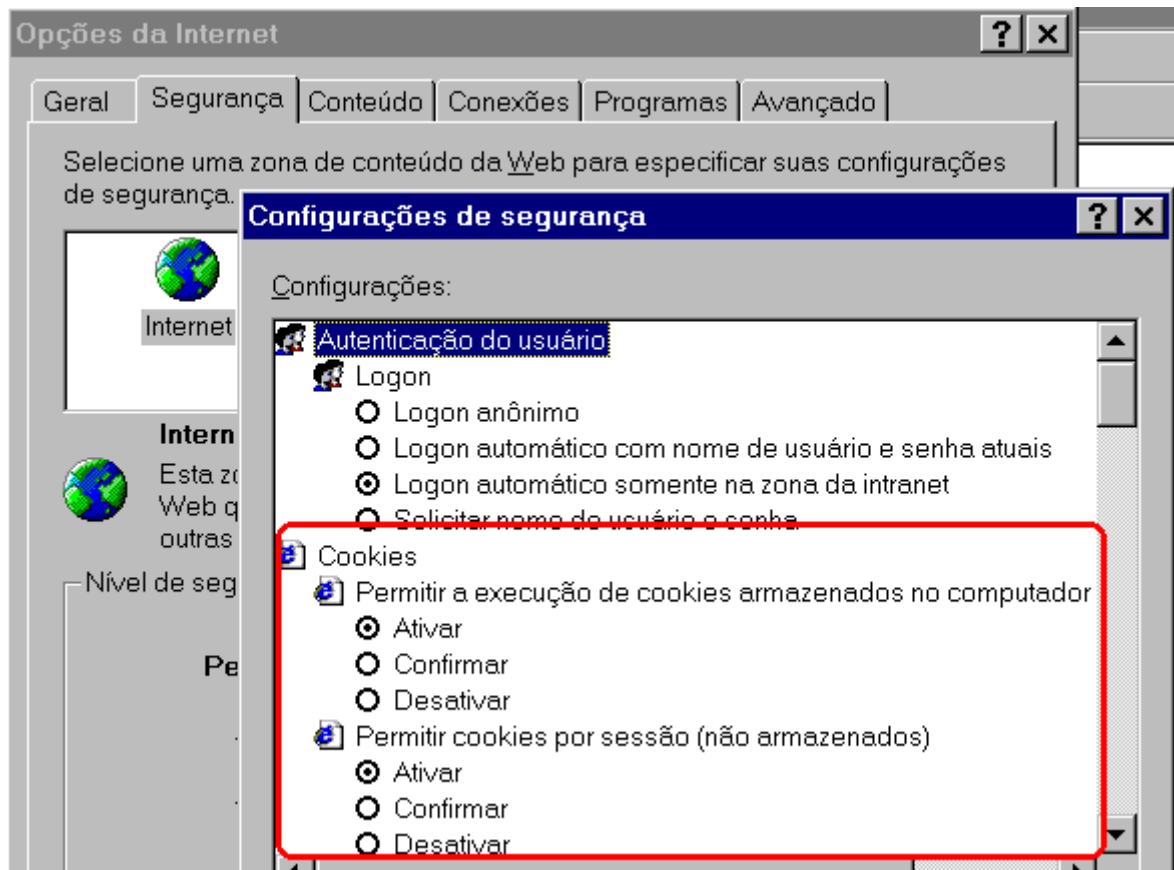
```
<form name="fml">  
Entre com um nome para o cookie:<input type = "text" name =  
"nome">  
<p>Entre com um valor para o cookie:<input type = "text" name=  
"valor">  
<p><input type = "button" value = "Gravar cookie" onClick=  
"gravaCookie()">  
</form>  
<p>Agora vá para outra página, clicando <a  
href="ck2.html">aqui</a>
```

Assim, através dos cookies, é possível manter uma continuação entre várias páginas de uma aplicação. Por exemplo: em páginas de uma aplicação de e-commerce com produtos que o usuário seleciona em determinada quantidade. Estes dados são gravados como cookies. Depois lidos numa outra página que tem um "carrinho de compras".

A lógica de programação destas continuidades pode ser um pouco complicada usando JavaScript. Nós, particularmente, achamos mais fácil trabalhar (desde que o browser aceite) com applets do Java e variáveis estáticas num frame adicional.

Uma outra razão pela qual não gostamos de usar cookies é que muitos crackers usam esta estrutura de acesso ao disco do usuário, para invadir

máquinas. E muita gente, com medo, desativa a aceitação de cookies (veja abaixo como se faz no IE), o que invalida toda sua aplicação.



É sempre bom então, se você vai usar cookies, alertar o usuário de que tem que aceitá-los para a aplicação funcionar.

ANOTAÇÕES:

FAQ SOBRE COOKIES

Cookies - são simplesmente bits de informação, pequenos arquivos texto (arquivos com terminacao .txt), geralmente com menos de 1Kb, que o seu browser capta em alguns sites e guarda em seu hard disk.

De onde surgem os *cookies*?

De você! A maioria das vezes os dados contidos nos arquivos texto vêm de informações que você forneceu. Essas informações podem ser o seu e-mail, o seu endereço ou qualquer informação que você tenha fornecido em um formulário online.

Para que eles são utilizados?

Em páginas personalizadas, nas quais a cada vez que você visita surgem opções que você selecionou previamente. Sem *cookies*, você teria de se registrar e resselecionar opções a cada vez que acessasse uma destas páginas. Com os *cookies*, o web site pode "lembrar" quem você é e quais as suas preferências.

Para que mais eles são utilizados?

Compras online e registro de acesso são outros motivos correntes de utilização. Quando você faz compras via Internet, cookies são utilizados para criar um "carriho de compras virtual", onde seus pedidos vão sendo registrados e calculados. Se você tiver de desconectar do site antes de terminar as compras, seus pedidos ficarão guardados até que você retorne ao site.

Webmasters e desenvolvedores de sites costumam utilizar os *cookies* para coleta de informações. Eles podem dizer ao webmaster quantas visitas o seu site recebeu, qual a freqüência com que os usuários retornam, que páginas eles visitam e de que eles gostam. Essas informações ajudam a gerar páginas mais eficientes, que se adaptem melhor as preferências dos visitantes.

Qual a utilização dos *cookies* na publicidade?

Algumas companhias já utilizaram, ou ainda utilizam cookies para coletar informações pessoais sobre os usuários e posteriormente enviar-lhes anúncios publicitários. O mais comum são os chamados *spam mails*, ou seja, mensagens não solicitadas que você recebe via e-mail, geralmente anunciando a venda de algum produto. Essa prática tem sido amplamente criticada e, atualmente, é considerado bastante anti-ético utilizar-se das informações providas por *cookies*, ou repassá-las para empresas interessadas em atingir um determinado público.

Se *cookies* são arquivos texto, quem pode ler esses arquivos?

Um *cookie* só pode ser lido pelo site que o criou. Webmasters não podem intrometer-se no diretório onde os cookies estão armazenados em seu computador para obter informações a seu respeito.

Um *cookie* pode trazer um vírus para o meu computador?

Não. Vírus somente são transportados por arquivos executáveis. Sendo *cookies* arquivos texto, não há perigo de carregarem nenhum vírus anexado à eles.

Então qual o problema? Por que algumas pessoas detestam *cookies*?

Alguns sentimentos anti-cookie são provocados apenas por desinformação. Cookies são simples arquivos texto, não podem entrar em seu hard disk e capturar nenhuma informação sobre você. Eles apenas guardam informações que você voluntariamente forneceu ao visitar um site. E os browsers revelam alguma informação sobre você de qualquer forma, mesmo sem a utilização dos cookies: o seu endereço de IP, seu sistema operacional, tipo de browser utilizado, etc.

Voce precisa aceitar *cookies*?

Não, não é preciso. A maioria dos browsers pode ser configurada para recusar cookies, embora não aceitando você vai perder muitos aspectos providos pela

rede. Você não terá que reconfigurar páginas personalizadas a cada vez que visitá-las, por exemplo.

Posso aceitar alguns cookies e rejeitar outros?

Sim, basta configurar seu browser para alertá-lo sempre antes de aceitar um cookie.

DEPURAÇÃO DE CÓDIGO

Qualquer programador que desenvolve programas com alguma linguagem sabe que erros de código são passíveis de ocorrer à qualquer momento na construção do programa. Isso não depende de experiência, seja novato ou veterano em programação. Contudo, as linguagens de programação ao encontrar um determinado erro em uma de suas linhas, interrompe a execução do programa e exibe uma mensagem informando a origem e qual foi o erro encontrado. Isto irá facilitar ao programador onde o mesmo deverá corrigir o erro.

Sabemos que, quando algo deixa de funcionar corretamente como foi designado pelo programador, isto é chamado de **BUG**. O processo de eliminação destes bug's é chamado de "depuração" ou como trata algumas linguagens "debugging". Caso o usuário encontre problemas na execução de seus scripts, basta entender um pouco como os erros ocorrem para saber o que fazer posteriormente. É preciso entender o que são erros, o que eles significam e principalmente, como corrigí-los. É bom também, saber quais são as mensagens de erro mais comuns que são exibidas quando o programador estiver escrevendo e testando seus scripts em JavaScript.

ISOLAMENTO DE PROBLEMAS

Normalmente os erros ocorrem durante o processo de edição dos scripts, portanto é bom sempre examinar todo o código ao primeiro sinal de problema. Existem três tipos de erros que ocorrem durante a execução de um programa em JavaScript:

Erros em tempo de carga (load-time)

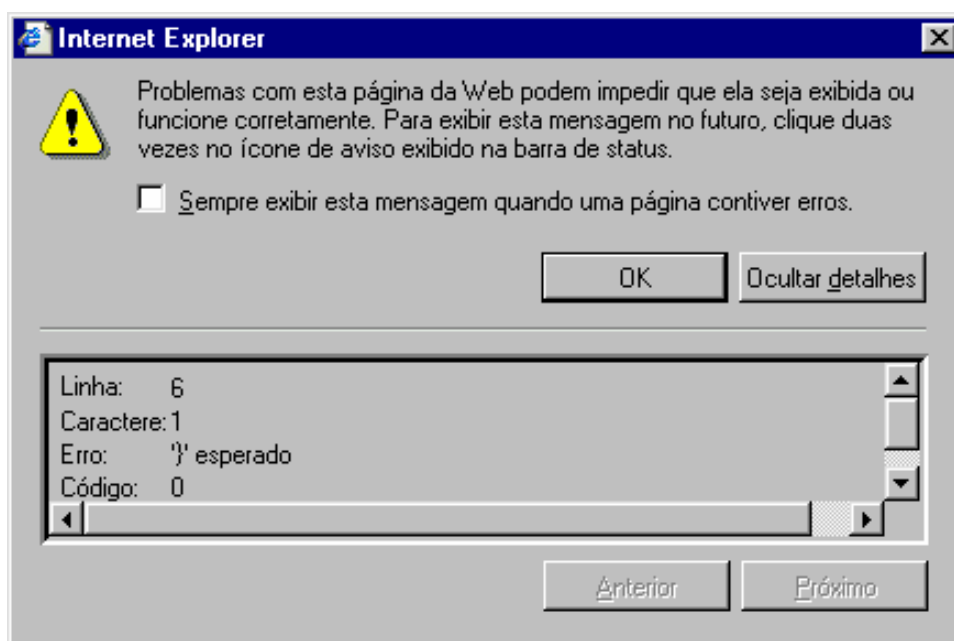
Erros em tempo de execução (run-time)

Erros de lógica

ERROS EM TEMPO DE CARREGAMENTO (Load-Time)

Este tipo de erro ocorre sempre que o browser carrega o script. Eles são os principais responsáveis pelo mal funcionamento de todo o script. É durante o processo de carregamento que é detectado todos os erros sérios que ocasionarão uma falha em todo o script. Com isto, o script não poderá ser carregado corretamente enquanto o erro existir.

Neste tipo de erro, a causa maior para ocorrer são normalmente os erros na sintaxe de alguma instrução. Um exemplo claro disto, é quando o usuário na criação de qualquer função esquece de delimitar o bloco de suas instruções com caractere de bloco que são as chaves. Para auxiliar a identificação do problema. O JavaScript exibe uma caixa de alerta informando que ocorreu um erro em tempo de carregamento.



Esta caixa de diálogo é apresentada quando o usuário dá um duplo clique sobre a mensagem que aparece na barra de status do Internet Explorer 6. Caso o usuário esteja utilizando uma versão anterior, esta mensagem é exibida imediatamente.

Observe pelo exemplo da figura anterior que ele destaca a linha onde existe o erro, o caractere que encontra-se errado e o que está faltando, neste caso é esperado o caractere de fechamento do bloco da função. É bom saber que nem sempre o erro poderá estar onde é informado, dependendo do erro, ele poderá estar localizado em outra parte do código ou da linha. Para que continue à execução do programa é necessário confirmar através do botão OK.

ERROS EM TEMPO DE EXECUÇÃO (Run-Time)

Neste tipo de erro, o problema ocorre quando o script está sendo executado, diferente dos erros em tempo de carregamento que aparecem no momento que o documento está sendo carregado. Normalmente são causados quando o usuário utiliza os códigos JavaScript de maneira incorreta. Um exemplo disto está quando o usuário faz uma referência a uma variável sem que ela tenha sido definida ou quando o usuário aplica incorretamente os objetos JavaScript. Vejamos um exemplo:

```
document(teste);
```

Sendo o correto:

```
document.write(teste);
```

ERROS DE LÓGICA (Logic Errors)

Já o erro de lógica ocorre sempre quando o script executa algo bem diferente do que foi programado a executar. Neste caso, não é devido aos parênteses ou chaves mal posicionados, mas sim, na construção do script. É necessário que o usuário proteja seu código contra erros de lógica sempre que puder. Por exemplo, caso o usuário defina dois campos de formulário do tipo texto e a

cada um defina um nome de variável, e em seguida, deseja que seja mostrado o conteúdo de uma das variáveis, só que especifica a outra variável, conseqüentemente os dados estarão trocados.

ERROS COMUNS EXISTENTES

Muitas das vezes o usuário notará que os erros encontrados são causados pelos simples erros existentes. Vejamos os mais comuns:

Posicionamento de chaves

É comum o usuário esquecer de delimitar o bloco de instruções presentes em uma função com os caracteres de bloco, que são as chaves. Seu uso é obrigatório para que a linguagem compreenda onde começa e termina sua função.

Strings entre aspas

Toda string possui aspas. Nunca esqueça delas para evitar problemas futuros.

Código de script correto

Sempre verifique o tipo de componente a ser usado. Saiba diferenciar instruções, métodos, propriedades, funções e objetos (quando estão em conjunto, são denominados entidades). Sempre analise seu script mais de uma vez e verifique se as entidades estão escritas de forma correta. É comum os programadores formatarem seus scripts com quebras de linha e tabulações (indentações) para criar uma espécie de hierarquia das entidades. Com certeza isso irá facilitar o acompanhamento e o relacionamento da ação de cada script. Vejamos um exemplo de script escrito de forma consistente:

```
<HTML>

    <HEAD>
    <TITLE>Página bem definida</TITLE>
    </HEAD>

<BODY>
    <H1>Formatando com HTML</H1>

    <SCRIPT>
        nome=prompt("Digite seu Nome:", "");
        if((nome=="") || (nome==null)){
            alert("Favor Preecher o campo!");
        }else{
            document.write("Seja Bem Vindo"+nome);
        }
    </SCRIPT>
</BODY>
<HTML>
```

Nomes de Objetos

Sempre verificar os nomes dos objetos se estão escritos corretamente, principalmente na diferenciação de letras maiúsculas e minúsculas. Um exemplo disto está no uso dos objetos **Date** e **Math** que possuem suas iniciais maiúsculas, já os outros começam com letras minúsculas.

ANALISANDO A ORIGEM DOS ERROS

É importante que o usuário na reparação de erros encontrados em seus scripts, isole suas funções e rotinas de modo que possa analisar etapa por

etapa de seu programa. Verificar minuciosamente cada trecho das linhas do script de maneira que encontre possíveis erros.

Muitos erros lógicos são causados por valores incorretos atribuídos as variáveis. Para facilitar a localização destes erros, basta que o usuário defina áreas de observação em vários locais de seu script. Estas áreas na maioria das vezes podem ser caixas de alerta para determinar se a execução do programa está chegando àquele ponto que se encontra. Logo após a depuração de todo o script, o usuário terá apenas que remover estas áreas de observação.

Outros programadores, preferem ter em mãos uma cópia de todo o seu código para que se possa fazer as devidas correções. Muitas das vezes, o usuário enxerga no papel o que ele não vê na tela.

OUTROS ERROS COMUNS

1. Deixar de utilizar aspas em strings.
2. Má utilização das aspas e apóstrofes (aspas simples).
3. Sinal de igualdade individual para expressões de comparação.
4. Utilizar o objeto de formulário pertencente ao Documento e não a Janela.
5. Utilização dos métodos com outros objetos que não os possuem.
6. Criação de laços infinitos.
7. Falta da instrução return em uma função.

RESUMO GERAL DE OBJETOS JAVASCRIPT

OBJETO	DESCRIÇÃO
anchor (âncora)	Link de hipertexto para a mesma página. As âncoras dependem do objeto document (documento).
anchors[] (âncoras)	Array que contém todas as âncoras no documento.
button	Cria um botão para um formulário. Este objeto pertence ao objeto form (formulário).
checkbox	Uma caixa de verificação de um formulário. Este objeto pertence ao objeto form .
Date	Define a Data/Hora atuais. Este é um objeto de nível superior.
document	Visualiza outros objetos no corpo de uma página. Este objeto pertence ao objeto window .
elements[] (elementos)	Array que apresenta todos os elementos de um formulário. Todos os elementos pertencem ao objeto form .
form (formulário)	Contém qualquer objeto criado em um formulário. Este objeto pertence ao objeto document .
forms[] (formulários)	Array que contém todos os formulários em documento.
frame	Determina as páginas divididas no navegador. Cada frame possui um

	objeto document . Pertencente ao objeto window .
frames[]	Array de frames do objeto window .
hidden (oculto)	Elemento de formulário que cria uma caixa de texto oculta. Pertencente ao objeto form .
history	Histórico de todas as páginas visitadas. Pertence ao objeto document .
link	Link de hipertexto. Pertence ao objeto document .
links[]	Array dos links de um documento.
location (localização)	URL (endereço) do documento atual. Pertence ao objeto document .
Math	Utilizado na execução de cálculos matemáticos. Objeto de nível superior.
navigator	Informações sobre o browser. Objeto de nível superior.
options[] (opções)	Array de itens de uma lista de seleção (select) em um formulário. Pertencente ao objeto form .
password (senha)	Elemento de um formulário que cria uma caixa de texto do tipo senha. Pertencente ao objeto form .
radio (botão de opção)	Elemento de um formulário que cria botões de opção. Pertencente ao objeto form .
reset	Elemento de um formulário que cria um botão que limpa os campos do formulário. Pertencente ao objeto form .

select (lista de opções)	Lista de seleção de um formulário. Pertencente ao objeto form .
string	Variáveis do tipo alfanumérico. Pertencente ao documento que se encontram.
submit	Elemento de um formulário que cria um botão de envio de dados em um formulário. Pertencente ao objeto form .
text (caixa de texto)	Elemento do formulário que cria um campo de texto. Pertencente ao objeto form .
textarea (área de texto)	Elemento do formulário que cria uma área de digitação de texto. Pertencente ao objeto form .
window (janela)	Representa a janela do browser. Objeto de nível superior.

RESUMO GERAL DE MÉTODOS JAVASCRIPT

MÉTODOS DO OBJETO DOCUMENT

MÉTODO	DESCRIÇÃO
clear	Limpa a janela (window).
close	Fecha o documento atual.
write	Permite a inclusão de texto no corpo do documento.
writeln	Permite a inclusão de texto no corpo do documento com um caractere de nova linha anexado.

MÉTODOS DO OBJETO FORM

MÉTODO	DESCRIÇÃO
blur()	Quando remove o foco de um campo do tipo text , textarea e campos de senha password .
click()	Quando é dado um clique sobre um elemento de botão em formulário ou campos do tipo radio e checkbox .
focus()	Quando é dado o foco sobre um campo do tipo text , textarea e campos de senha password .
select()	Quando é selecionado o conteúdo de um campo do tipo text , textarea ou password .
submit()	Quando o formulário é enviado ao servidor.

MÉTODOS DO OBJETO DATE

MÉTODO	DESCRIÇÃO
getDate	Retorna o dia do mês da data especificada a partir de um objeto date.
getDay	Retorna o dia da semana da data especificada. O valor retornado é um valor inteiro correspondente ao dia da semana, sendo 0 para Domingo, 1 para Segunda-feira e assim por diante
getFullYear	Retorna o ano composto de 4 dígitos.
getHours	Retorna a hora para a data especificada. O valor retornado corresponde a um número inteiro entre 0 e 23.
getMinutes	Retorna os minutos na hora especificada. O valor retornado corresponde a um número inteiro entre 0 e 59.
getMonth	Retorna o mês da data especificada. O valor retornado corresponde a um número inteiro entre 0 a 11, sendo 0 para janeiro, 1 para fevereiro e assim por diante
getSeconds	Retorna os segundos da data especificada. O valor retornado corresponde a um número inteiro entre 0 e 59.
getTime	Retorna o número de segundos entre 1 de janeiro de 1970 e uma data específica.
getTimezoneOffset	Retorna a diferença de fuso horário em

	minutos para a localidade atual.
getYear	Retorna o ano de uma data específica.
parse	Retorna o número de milissegundos de uma data a partir de 1 de janeiro de 1970, 00:00:00
setDate	Estabelece o dia do mês para uma data especificada sendo um inteiro entre 1 ou 31.
setHours	Estabelece a hora para uma data especificada, sendo um inteiro entre 0 e 23, representando a hora dia.
setMinutes	Estabelece os minutos para a data especificada, sendo um inteiro entre 0 e 59.
setMonth	Estabelece o mês para a data especificada, sendo um inteiro entre 0 e 11 para o mês.
setSeconds	Estabelece o número de segundos para data especificada, sendo um inteiro entre 0 e 59.
setTime	Estabelece o valor do objeto date, sendo um inteiro representando o número de milissegundos desde 1 de janeiro de 1970.
setYear	Define o ano de uma data específica
toGMTString	Converte a data para string usando as convenções da GMT.
toLocaleString	Converte uma data para string, usando as convenções locais.

toString	Retorna uma string representando a data especificada
UTC	Converte uma data delimitada por vírgulas para o número de segundos a partir de 1 de janeiro de 1970.

MÉTODOS DO OBJETO HISTORY

EVENTO	DESCRIÇÃO
Back	Representa a URL visitada anteriormente.
Forward	Representa a URL que estava antes de voltar.
Go	Representa uma URL que esteja na relação de URL's visitadas.

MÉTODOS DO OBJETO MATH

MÉTODOS	DESCRIÇÃO
Abs	Retorna o valor absoluto de um número.
Acos	Retorna o arco cosseno de um número, em radianos.
Asin	Retorna o arco seno de um número, radianos.
Atan	Retorna o arco tangente de um número, em radianos.
Ceil	Retorna o menor inteiro maior ou igual a um número.
Cos	Retorna o cosseno de um número.
Eval	Calcula o conteúdo de uma expressão. EVAL é uma função.

Exp1	Retorna o logarítmo do número na base E.
Floor	Retorna o maior inteiro menor ou igual ao número.
isNAN	Determina se um valor é um número ou não.
Log	Retorna o logarítmo natural de um número (base E).
Max(num1,num2)	Retorna o maior valor de dois números.
Min(num1,num2)	Retorna o menor valor de dois números.
Pow(base,expoente)	Retorna a base elevada ao expoente.
Random()	Retorna um número aleatório entre 0 e 1.
Round	Retorna o valor arredondado de um inteiro.
Sin	Retorna o seno de um número.
Sqrt	Retorna a raiz quadrada de um número.
Tan	Retorna a tangente de um número.

MÉTODOS DO OBJETO STRING

MÉTODOS	DESCRIÇÃO
Anchor	Cria uma âncora HTML que é utilizada como destino de um link de hipertexto.
Big	Mostra uma string em fonte maior. Similar ao tag <BIG>.
Blink	Apresenta uma string piscante. Similar ao tag <BLINK>.
Bold	Apresenta a string em negrito. Similar ao tag .

CharAt(índice)	Retorna o caractere no índice especificado.
Concat(s1,s2,s3...)	Concatena várias strings, retornando uma nova string.
Fixed	Apresenta a string em fonte monoespaço. Similar ao tag <TT>.
Fontcolor(cor)	Apresenta uma string com uma cor especificada. Similar ao tag .
FontSize(tamanho)	Apresenta a string com tamanho determinado. Similar ao tag .
IndexOf	Retorna a posição dentro da string onde aparece em primeiro o texto especificado. Ex.: string.indexOf(valor,[índice]).
Italics	Apresenta o texto em itálico da mesma forma que o tag <I>.
LastindexOf	Retorna a posição dentro da string da última ocorrência do texto procurado. Ex: string.lastIndexOf(valor,[índice]).
Link(href)	Cria um link HTML. Similar ao tag <A HREF>.
Slice (início,fim)	Extraí uma parte de uma string. Onde início é o caractere inicial da string e fim o caractere final.
Small	Apresenta a fonte em tamanho menor da mesma forma do tag <SMALL>.
sub	Apresenta a fonte em formato subscrito, da mesma forma do tag <SUB>.

sup	Apresenta a fonte em formato sobrescrito, da mesma forma do tag <SUP>.
substring(indexA,indexB)	Retorna um pedaço de um objeto string.
toLowerCase	Converte a string em caracteres minúsculos.
toUpperCase	Converte a string em caracteres maiúsculos.

MÉTODOS DE INTERFACE COM O USUÁRIO

MÉTODOS	DESCRIÇÃO
alert	Exibe uma caixa de diálogo com o botão OK.
confirm	Exibe uma caixa de diálogo com os botões OK e CANCELAR.
prompt	Exibe uma caixa de diálogo solicitando a entrada de informação ao usuário.

MÉTODOS DO OBJETO WINDOW

MÉTODOS	DESCRIÇÃO
clear	Limpa a janela.
clearTimeout	Limpa um contador de tempo definido anteriormente com o método setTimeout .
close	Fecha uma janela.
open	Abre uma janela
setTimeout	Define um contador de tempo.