Apostila sobre
Java Script

Java Script Básico

1. Introdução

1.1 O que é Java Script?

JavaScript é uma linguagem que permite injetar lógica em páginas escritas em HTML (HiperText Mark-up Language).
Os parágrafos de lógica do javaScript podem estar "soltos" ou atrelados a ocorrência de eventos.
Os parágrafos soltos são executados na sequência em que aparecem na página (documento) e os atrelados a eventos são executados apenas quando o evento ocorre.
Para inserir parágrafos de programação dentro do HTML é necessário identificar o início e o fim do set de JavaScript, da seguinte forma:

```
<SCRIPT>

    instruções ...

</SCRIPT>
```

Este procedimento pode ser adotado em qualquer local da página. Os comandos JavaScript são sensíveis ao tipo de letra (maiúsculas e minúsculas) em sua sintaxe.
Portanto, é necessário que seja obedecida a forma de escrever os comandos, de acordo com a forma apresentada ao longo deste manual. Caso seja cometido algum erro de sintaxe quando da escrita de um comando, o JavaScript interpretará, o que seria um comando, como sendo o nome de uma variável.

1.2 Qual é a diferença entre Java e JavaScript?

Ainda que os nomes sejam quase os mesmos, Java não é a mesma coisa que JavaScript! Essas são duas técnicas diferentes de programação na Internet. Java é uma linguagem de programação. JavaScript é uma linguagem de scripting (tal como diz o nome). A diferença é que se pode criar programas reais com Java. O mais das vezes, porém, você quer apenas criar um efeito chamativo, sem se importar com qualquer programa real. Assim, JavaScript foi pensado como algo fácil de se compreender e de se usar. Os autores de JavaScript não têm que se importar muito com programação. Nós poderíamos até dizer que JavaScript é muito mais uma extensão do HTML do que uma linguagem de computador separada. Naturalmente essa não é uma definição

"oficial" mas acho que ela torna mais compreensível a diferença entre Java e JavaScript.

1.3 Um pequeno exemplo do uso de scripts

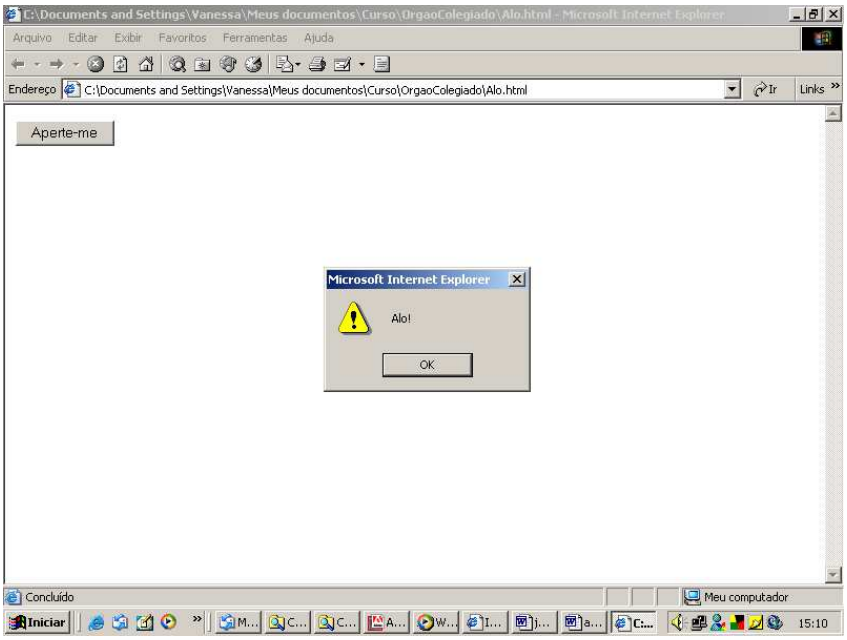
Exemplo 1:

Crie um arquivo HTML chamado Alo.html e coloque o código abaixo:

```
<html>
<body>
<form>
  <input type="button" name="Button1" value="Aperte-me"
onClick="mensagem()">
  </form>
</body>
</html>

  <script language="JavaScript">
     function mensagem() {
       alert("Alo!");
  }
 </script>
```

Abra o arquivo Alo.html com o seu navegador e clique no botão “Aperte-me”, o resultado será este:



Então, o que acontece neste script?

Existe uma coisa nova no tag de <input>. Lá você pode ver **'onclick'**. Isto diz ao browser que função ele tem que chamar quando o botão é pressionado (é claro que só quando o browser suporta JavaScript). A função 'mensagem()' é declarada na página. Quando o botão é pressionado a função é executada. Tem uma coisa nova neste script - o método 'alerta'. Este método já é declarado no JavaScript - desse modo você só tem que chamá-lo. Existem muitos métodos diferentes que você pode chamar.

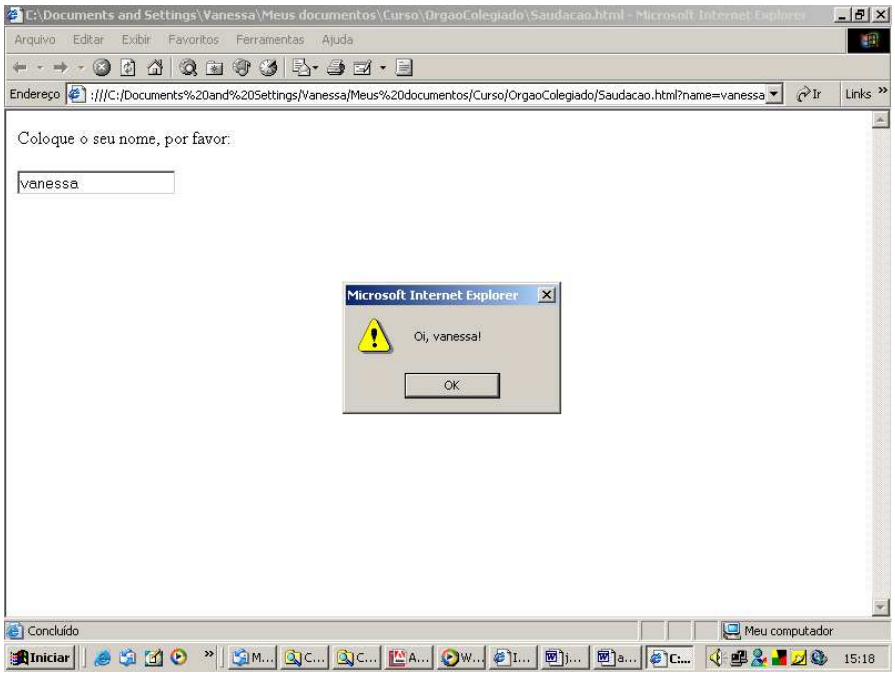
Exemplo 2:

Crie um arquivo HTML chamado Saudacao.html e coloque o código abaixo:

```
<html>
<head>
<script language="JavaScript">
<!--  hide script from old browsers
  function getname(str) {
    alert("Oi, "+ str+"!");
  }
// end hiding contents -->
</script>
</head>
```

```
<body>
Coloque o seu nome, por favor:
<form>
  <input type="text" name="name"
onBlur="getname(this.value)" value="">
</form>
</body>
</html>
```

Abra o arquivo Saudacao.html com o seu navegador, digite seu nome na caixa de texto e clique na parte em branco da página, o resultado será este:



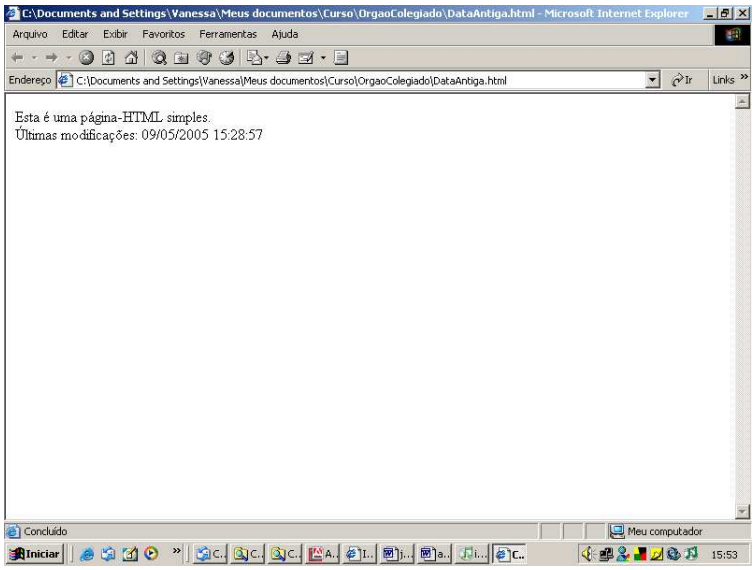
Nós temos alguns elementos novos implementados neste script novamente. Em primeiro lugar, você certamente notou o comentário dentro do script. Dessa maneira você pode esconder o script dos browsers antigos que não podem rodar scripts. Você tem que manter a ordem daquilo que é mostrado! O início do comentário deve estar logo depois do primeiro tag de <script>. O comentário termina logo antes do tag de </script>. Neste documento-HTML você tem um elemento de formulário onde o usuário pode colocar o seu nome. O **'onBlur'** no tag de <input> avisa ao cliente que função que ele tem que chamar quando alguma coisa é colocada dentro do formulário. A função 'getname(str)' será chamada quando você 'deixa' este elemento de formulário, ou pressiona o 'enter' depois de haver colocado alguma coisa. A função pegará a seqüência que você colocou através do comando 'getname(this.value)'. 'This.value' significa o valor que você colocou neste elemento de formulário.

Exemplo 3:

Vamos implementar uma função de data dentro do nosso script. Assim, se você criou uma página-HTML, você pode fazer o cliente imprimir a última modificação no documento. Você não precisa, entretanto, escrever a data no documento. Você simplesmente escreve um pequeno programa de script. Quando você fizer pequenas modificações no futuro, a data se modificará automaticamente.

Crie um arquivo HTML chamado DataAntiga.html e coloque o código abaixo:

```
<html>
<body>
Esta é uma página-HTML simples.
<br>
Últimas modificações:
  <script language="JavaScript">
  <!--  hide script from old browsers
    document.write(document.lastModified)
  // end hiding contents -->
  </script>
</body>
</html>
```



2. Operadores e Controles Especiais

2.1 Operadores Matemáticos

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

+	Adição de valor e concatenação de strings
-	Subtração de valores
*	Multiplicação de valores
/	Divisão de valores
%	Obtém o resto de uma divisão:

Ex: 150 % 13 retornará 7.
7 % 3 retornará 1.

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

x += y é o mesmo que x = x + y

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

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

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

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

2.2 Operadores Lógicos

São operadores a serem utilizados em comandos condicionais, tais como: IF, FOR e WHILE.

Os comandos condicionais serão vistos mais à frente.

==	Igual
!=	Diferente
>	Maior
>=	Maior ou Igual

<	Menor
<=	Menor ou Igual
&&	E
||	Ou

2.3 Caracteres Especiais

\b	backspace
\f	form feed
\n	new line caracters
\r	carriage return
\t	tab characters
//	Linha de comentário
/*....*/	Delimitadores para inserir um texto com mais de uma linha como comentário

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

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

3 Comandos Condicionais

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

São eles:

Comando IF

if (condição) {
ação para condição satisfeita
}
[else {
ação para condição não satisfeita }]

Ex.:

```
if (Idade < 18) {
     Categoria = "Menor"
} else {
     Categoria = "Maior"
}
```

Comando FOR

for ([inicialização/criação de variável de controle ;]
[condição ;]
[incremento da variável de controle])
{ ação }

Ex.:

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

Comando WHILE

Executa uma ação enquanto determinada condição for verdadeira.

while (condição)
{ ação }

Ex.:

```
var contador = 10

while (contador > 1) {
     contador-
}
```

Move condicional

receptor = ((condição) ? verdadeiro : falso)

Ex.:

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

OBS:

Nos comandos **FOR** e **WHILE** a diretiva **"break"** pode ser utilizada para interromper a condição principal e sair do loop. Da mesma forma, a diretiva **"continue"** interrompe uma ação (se determinada condição ocorrer), mas volta para o loop.
Diretivas/condições entre [ ] significam que são opcionais.

4 Eventos

São fatos que ocorrem durante a execução do sistema, a partir dos quais o programador pode definir ações a serem realizadas pelo programa. Abaixo apresentamos a lista dos eventos possíveis, indicando os momentos em que os mesmos podem ocorrer, bem como, os objetos passíveis de sua ocorrência.

onload - Ocorre na carga do documento. Ou seja, só ocorre no BODY do documento.

onunload - Ocorre na descarga (saída) do documento. Também só ocorre no BODY.

onchange - Ocorre quando o objeto perde o foco e houve mudança de conteúdo.
Válido para os objetos Text, Select e Textarea.

onblur - Ocorre quando o objeto perde o foco, independente de ter havido mudança.
Válido para os objetos Text, Select e Textarea.

onfocus - Ocorre quando o objeto recebe o foco.
Válido para os objetos Text, Select e Textarea.

onclick - Ocorre quando o objeto recebe um Click do Mouse.
Válido para os objetos Buton, Checkbox, Radio, Link, Reset e Submit.

onmouseover - Ocorre quando o ponteiro do mouse passa por sobre o objeto.
Válido apenas para Link.

onselect - Ocorre quando o objeto é selecionado.
Válido para os objetos Text e Textarea.

onsubmit - Ocorre quando um botão tipo Submit recebe um click do mouse.
Válido apenas para o Form.

Exemplo:

Crie uma página com o nome Eventos.html e insira este código nela:

```
<html>
<body>
<a href="ColegiadoOrgaoManutencao.html"
onMouseOver="window.status='Curso de Java Script'; return
true"> Cadastro do Conselho </a>

</body>
</html>
```

Abra esta página com o seu navegador e veja o resultado. Posicione o mouse sob o link e olhe para a barra de status na parte de baixo do seu browser.

A única coisa que você tem que fazer é acrescentar a propriedade onMouseOver ao seu tag de link <a>. O 'window.status' lhe permitirá escrever coisas na barra de status do seu browser. Como se pode ver, você tem que alterar as aspas. Você não vai poder usar " todo o tempo, porque se não o JavaScript não será capaz de identificar a seqüência que você quer imprimir na barra de status. Depois da seqüência você tem que escrever *;return true*.

5 Criando variáveis

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

Exemplo:

```
novaVariavel = "Jose"
```

Foi criada a variável de nome novaVariavel que, passou a conter a string Jose.
As variáveis podem ser **Locais** ou **Globais**. As variáveis que são criadas dentro de uma função são Locais e referenciáveis apenas dentro da função. As variáveis criadas fora de funções são Globais, podendo ser referenciadas em qualquer parte do documento.
Desta forma, variáveis que precisam ser referenciadas por várias funções ou em outra parte do documento, precisam ser definidas como globais.
Embora não seja recomendável, em uma função, pode ser definida uma variável local com o mesmo nome de uma variável global. Para isso utiliza-se o método de definição var.

Exemplo:

```
Variável Global → MinhaVariavel = ""
Variável Local → var MinhaVariavel = ""
```

6 Escrevendo no documento

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

Vamos falar sobre o objeto **document.write()**.

O objeto

Método do objeto documento:

O arquivo HTML que aparece na janela do browser é um objeto tipo **document.**

A cada objeto Javascript, o programador da linguagem previu um conjunto de métodos - ou funções dedicadas a este objeto. Para `document`, o Javascript dedicou o método ***'escrever no documento'***, conhecido como o método ***write()***.

A chamada do método faz-se segundo a notação:

nome_do_objeto.nome_do_método

Para chamar o método ***write()*** do documento, escreva-se:

document.write();

Método write()

A sintaxe é bastante simples

write("seu texto");

Pode-se também escrever uma variável, seja a variável resultado,
`write(resultado);`

Para associar texto (cadeia de caracteres) e variáveis, utiliza-se à instrução:

write("O resultado é" + resultado);

Podem-se utilizar os tags Html para incrementar o texto:

write("<B>O resultado é</B>" + resultado); ou
write ("<B>" + "O resultado é" + "</B>" + resultado)

Exemplos:

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

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

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

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

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

O problema

O objeto ***'document.write()'*** limpa todo o documento antes de imprimir algo na tela.
Para solucionar este problema utilizaremos o objeto **‘*window’*.** Com esse simples parâmetro adicional, ***'window-janela'*** no objeto, ele se torna muito útil, além de não mais limpar a tela para imprimir.

Exemplo:

```
<html>
<head>

<script language="Java Script">
     window.document.write("Sem limpar...");
</script>

</head>
<body>
o documento...
</body>
</html>
```

7 Mensagens

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

Apenas Observação.

alert (mensagem)

Exemplo:

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

Mensagem que retorna confirmação de OK ou CANCELAR

confirm (mensagem)

Exemplo:

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

Recebe mensagem via caixa de texto Input

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

Onde:

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

Ex.

```
Entrada = prompt ("Informe uma expressão matemática", "")
Resultado = eval (Entrada)

document.write("O resultado é = " + Resultado)
```

8 Funções

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

A sintaxe geral é a seguinte:

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

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

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

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

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

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

8.1 Funções intrínsecas

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

Result	função (informação a ser processada)
eval	Calcula o conteúdo da string
parseInt	Transforma string em inteiro
parseFloat	Transforma string em número com ponto flutuante

Date()	- date() - Retorna a data e a hora ex1: Result = eval (" (10 * 20) + 2 - 8") ex2: Result = eval (string) No primeiro exemplo Result seria igual a 194. No segundo, depende do conteúdo da string, que também pode ser o conteúdo (value) de uma caixa de texto.

Funções tipicamente Matemáticas:

Math.abs(número)	Retorna o valor absoluto do número (ponto flutuante)
Math.ceil(número)	Retorna o próximo valor inteiro maior que o número
Math.floor(número)	Retorna o próximo valor inteiro menor que o número
Math.round(número)	Retorna o valor inteiro, arredondado, do número.
Math.pow(base, expoente)	Retorna o cálculo do exponencial
Math.max(número1, número2)	Retorna o maior número dos dois fornecidos
Math.min(número1, número2)	Retorna o menor número dos dois fornecidos
Math.sqrt(número)	Retorna a raiz quadrada do número
Math.SQRT2	Retorna a raiz quadrada de 2 (aproximadamente 1.414)
Math.SQRT_2	Retorna a raiz quadrada de 1/2 (aproximadamente 0.707)
Math.sin(número)	Retorna o seno de um número (anglo em radianos)
Math.asin(número)	Retorna o arco seno de um número (em radianos)
Math.cos(número)	Retorna o cosseno de um número (anglo em radianos)
Math.acos(número)	Retorna o arco cosseno de um número (em radianos)
Math.tan(número)	Retorna a tangente de um número (anglo em radianos)
Math.atan(número)	Retorna o arco tangente de um número (em radianos)
Math.pi	Retorna o valor de PI (aproximadamente 3.14159)
Math.log(número)	Retorna o logarítmo de um número
Math.E	Retorna a base dos logarítmos naturais (aproximadamente 2.718)
Math.LN2	Retorna o valor do logarítmo de 2 (aproximadamente 0.693)
Math.LOG2E	Retorna a base do logarítmo de 2 (aproximadamente 1.442)
Math.LN10	Retorna o valor do logarítmo de 10 (aproximadamente 2.302)
Math.LOG10E	Retorna a base do logarítmo de 10 (aproximadamente 0.434)

9 Manipulando String's e Datas

String's

O JavaScript é bastante poderoso no manuseio de String´s, fornecendo ao programador uma total flexibilidade em seu manuseio.
Abaixo apresentamos os métodos disponíveis para manuseio de string´s:

string.length	Retorna o tamanho da string (quantidade de bytes)
string.charAt(posição)	Retorna o caracter da posição especificada (inicia em 0)
string.indexOf("string")	Retorna o número da posição onde começa a primeira "string"
string.lastindexOf("string")	Retorna o número da posição onde começa a última "string"
string.substring(index1, index2)	Retorna o conteúdo da string que corresponde ao intervalo especificado. Começando no caracter posicionado em index1 e terminando no caracter imediatamente anterior ao valor especificado em index2.

Exemplo:

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

(A variável Parte receberá a palavra log)

string.toUpperCase()	Transforma o conteúdo da string para maiúsculo (Caixa Alta)
string.toLowerCase()	Transforma o conteúdo da string para minúsculo (Caixa Baixa)
escape ("string")	Retorna o valor ASCII da string (vem precedido de %)
unscape("string")	Retorna o caracter a partir de um valor ASCII (precedido de %)

Datas

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

Exemplo:

Fri May 24 16:58:02 1996

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

getDate()	Obtém o dia do mês (numérico de 1 a 31)
getDay()	Obtém o dia da semana (0 a 6)
getMonth()	Obtém o mês (numérico de 0 a 11)
getYear()	Obtém o ano
getHours()	Obtém a hora (numérico de 0 a 23)
getMinutes()	Obtém os minutos (numérico de 0 a 59)
getSeconds()	Obtém os segundos (numérico de 0 a 59)

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

Exemplo:

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

10 Manipulando Arrays

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

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

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

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

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

Agora poderemos obter os dados diretamente dos arrays.

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

DiaSemana passaria a conter Quinta e Ocupação conteria Programador.
Outra forma de se trabalhar com arrays é criar novas instâncias dentro do próprio objeto do array, o que proporciona o mesmo efeito de se trabalhar com matriz. Isso pode ser feito da seguinte forma:

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

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

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

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

Obs:

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

11 Formulário e elementos do formulário

A maior parte da interatividade entre uma página da Web e o usuário ocorre dentro de um formulário. E aí que reside grande parte da HTML interativa para cada browser: campos de texto, botões, caixas de seleção, listas de opções e assim por diante. Todo formulário está contido em um documento, mesmo assim, o objeto **document** precisa fazer parte da referência ao formulário e seus elementos.

11.1 O Objeto FORM

Um objeto FORM pode ser referenciado por sua posição no array de formulário contidos por um documento ou pelo seu nome (se você atribuir um identificador ao atributo NAME dentro da tag <FORM>.) Se apenas um formulário aparecer no documento, ele ainda será um membro de um array (um array de um elemento), podendo ser referenciado desta forma:

```
document.forms[0]
```

Observe que a referência do array usa a versão no plural da palavra, seguida por um par de colchetes que contém o número de índice do elemento (zero é sempre o primeiro). Mas, se você atribuir um nome para o formulário, basta informar o nome do formulário na referência:

```
document.nomeFormulario
```

11.2 Objetos relacionados ao texto

Cada um dos quatro elementos do formulário HTML relacionados ao texto – texto, senha, oculto e TextArea – é um elemento na hierarquia de objetos do documento. Tudo, menos o objeto oculto, aparece na página, permitindo que os usuários insiram informações.
Para os objetos visíveis nessa categoria, os manipuladores de evento são disparados a partir de ações do usuário, como dar o foco a um campo (colocando o ponteiro de inserção do texto no campo) e alterando o texto (digitando novo texto e saindo do campo). A maior parte das suas ações de campo de texto é disparada pela mudança do texto (o manipulador de evento *onChange*).

a) Propriedade value

A propriedade mais usada de um elemento relacionado a texto é a propriedade *value*. Essa propriedade representa o conteúdo atual do

elemento de texto. Um script pode apanhar e definir seu conteúdo a qualquer momento. O conteúdo de uma propriedade *value* é sempre uma string. Poderá ser usado os métodos de manipulação de String's juntamente com a propriedade *value*, verificar capítulo 9.

Sintaxe:

```
var  texto;
texto = document.nomeFormulario.nomeCampoTexto.value;
```

Exemplo usando o método de String **toUpperCase()**:

```
<script language="JavaScript">

function upperMe(){
        var campo = document.form_org.nomeAluno;
        var valorUpper = campo.value.toUpperCase();

        campo.value = valorUpper;
}
</script>
```

No exemplo acima a função upperMe() irá converter para letras maiúsculas o valor do campo texto "nomeAluno" , mas para que esta função funcione é necessário que um evento a invoque. Veja o exemplo abaixo da chamada da função com a utilização do evento *onChange()*:

```
<FORM >
      <input type="text" name="nomeAluno" value="exemplo"
     onChange = "upperMe()">
</FORM>
```

b) Método *blur* do objeto Text

Simula o evento de retirada do foco do objeto em questão.

Sintaxe:

```
document.nomeFormulario.nomeCampoTexto.blur();
```

c) Método focus do objeto Text

Simula o evento de focalização do objeto, ou seja, passa o foco para o objeto em questão.

Sintaxe:

```
document.nomeFormulario.nomeCampoTexto.focus();
```

11.3 O Objeto de botão

O botão é um dos objetos mais simples de se programar. Ele possui apenas algumas propriedades que raramente são acessadas ou modificadas nos scripts. O evento mais útil do objeto de botão é o evento *onClick*. Ele é disparado sempre que o usuário dá um clique no botão.

11.4 O Objeto de caixa de seleção

A propriedade *value* de uma caixa de seleção é qualquer outro texto que você deseja associar ao objeto. Esse texto não aparece na página de forma alguma. A propriedade *value* do objeto de caixa de seleção (*checkbox*) é a String que será enviada pelo formulário caso o *checkbox* em questão esteja checado.

a) Propriedade *checked*

É a propriedade principal de um objeto de caixa de seleção, esta propriedade verifica se a caixa está marcada ou não. A propriedade checked é um valor Booleano: true [verdadeiro] se a caixa estiver marcada, false [falso] se não estiver.
Toda a propriedade quando for booleano poderá ser usado em uma expressão de condição `if ou if...else`

Exemplo:

```
function verificaBox(){
     if (document.form_org.campoChecagem.checked){
          alert("Esta caixa de seleção está checada.");
     }else{
          alert("Esta caixa de seleção não está
          checada");
     }
}
```

b) Método *blur* do objeto Checkbox

Simula o evento de retirada do foco do checkbox em questão.

Sintaxe:

```
document.nomeFormulario.campoCheckbox.blur();
```

c) Método *focus* do objeto Checkbox

Simula o evento de focalização do objeto, ou seja, passa o foco para o objeto em questão.

Sintaxe:

```
document.nomeFormulario.campoCheckbox.focus();
```

d) Método *click* do objeto Checkbox

Simula o evento de clique do mouse.

Sintaxe:

```
document.nomeFormulario.campoCheckbox.click();
```

11.5 O objeto de botão de opção

Para deixar que o browser controle a marcação e a desmarcação de um grupo de botões relacionados, você precisa atribuir o mesmo nome a cada um dos botões no grupo. Você pode ter vários grupos dentro de um formulário, mas cada membro do mesmo grupo precisa ter o mesmo nome.
A atribuição do mesmo nome a um elemento do formulário força o browser a controlar os elementos de forma diferente do que se cada um tivesse um nome exclusivo. Em vez disso, o browser mantém uma lista dos objetos com o mesmo nome na forma de um array. O nome atribuído ao grupo torna-se o nome do array.

a) Propriedade *length*

Usado para descobrir quantos botões existem em um determinado grupo.

Sintaxe:

```
document.form_org.nomeGrupo.length;
```

b) Propriedade *checked*

Para descobrir se um botão específico está atualmente marcado terá que acessar o elemento de botão individualmente.

Sintaxe:

```
document.form_org.nomeGrupo[0].checked;
```

c) Método *blur* do objeto Radio

Simula o evento de retirada do foco do **Radio** em questão.

Sintaxe:

```
document.nomeFormulario.campoOption[indice].blur();
```

d) Método *focus* do objeto Radio

Simula o evento de focalização do objeto, ou seja, passa o foco para o objeto em questão.

Sintaxe:

```
document.nomeFormulario.campoOption[indice].focus();
```

11.6 O objeto SELECT

O objeto Select é um objeto composto: um objeto que contém uma array de objetos Option. Sua lista pode ser instantânea, que só permite uma única seleção, ou então, uma lista rolável, que permite aceitar várias seleções pelo usuário.

a) Propriedade *selectedIndex*

Retorna o número de índice do item atualmente selecionado. O primeiro item (no topo da lista) possui um índice zero. O valor de *selectedIndex* é fundamental para permitir que você acesse propriedades da opção selecionada.

Sintaxe:

```
document.form_org.nomeSelecao.selectedIndex;
```

b) Propriedade *text*

A propriedade *text* é a String que aparece na tela, no objeto Select.

Sintaxe:

```
document.form_org.nomeSelecao.options[n].text;
```

c) Propriedade *value*

A propriedade *value* permite a leitura da string oculta de cada tag Option definida.

Sintaxe:

```
document.form_org.nomeSelecao.options[n].value;
```

Onde:

n = Índice da lista;

12 Passando dados e elementos do formulário a funções

Existem atalhos valiosos para a transferência de informações sobre formulários ou controle do formulário diretamente para a função, sem a necessidade de se lidar com aquelas referências normalmente longas que começam ao nível do objeto *window* ou *document*.
O JavaScript possui uma palavra-chave – *this* – que sempre se refere ao objeto que contém o script em que a palavra-chave é usada. Assim, em um manipulador de evento *onChange* para um campo de texto, você pode passar uma referência ao objeto de texto para a função inserindo a palavra-chave *this* como parâmetro da função:

```
<input type=”text” name=”nomeAluno” onChange=”upperMe(this)”>
```

No extremo receptor, a função define uma variável de parâmetro que transforma essa referência em uma variável, que pode ser usada pelo restante da função:

```
function upperMe(campo){
     instruções
}
```

13 Objetos da janela e do documento

13.1 Acessando as propriedades e métodos da janela

O objeto window representa uma janela do navegador ou um frame em um frameset. Toda janela aberta do navegador possui um objeto window, que se refere a ela própria – isso torna o objeto à base na hierarquia do modelo de objetos do navegador.

Sintaxe:

```
window.nomePropriedade;
window.nomeMétodo([parâmetros])
```

Um objeto `window` também possui um sinônimo quando o script realizando a referência aponta para a janela que abriga o documento. O sinônimo é `self`.

Sintaxe:

```
self.nomePropriedade;
self.nomeMétodo([parâmetros])
```

a) Método back

Restaura a última informação registrada no histórico sobre o conteúdo apresentado nos frames de uma janela do navegador. É o mesmo que pressionar o botão Voltar (Back) da barra de ferramentas do navegador.

Sintaxe:

```
window.back();
```

b) Método close()

Fecha a janela especificada – se ela foi aberta usando o método open ou se o seu histórico estiver vazio, nenhum aviso será gerado. Em todos os outros casos, será gerada uma mensagem de confirmação de seu fechamento.

Sintaxe:

```
window.close();
```

c) Método open()

Abre uma nova janela.
Este método contém até três parâmetros que definem a características da janela, como o URL do documento a ser carregado, seu nome para fins de referência do atributo TARGET em tags HTML e aparência física (tamanho e etc).

Sintaxe:

```
var varJanela = window.open(URL, nomeJanela
                [,propriedades] );
```

Exemplo:

```
var novaJanela = window.open("janelaNova.html",
     "Minha Janela", "height=200, width=300");
```

Propriedades do método open():

- height: Altura (em pixels) da nova janela.
- menubar: Se **yes**, é criada a barra de menus.
- resizable: Se **yes**, o usuário poderá redimensionar a janela.
- screenX: Distância (em pixels) do canto esquerdo da tela até o canto esquerdo da janela.
- screenY: Distância (em pixels) do topo da tela até o topo da janela.
- scrollbars: Se **yes**, permite o surgimento das barras de rolagem vertical e horizontal, quando necessárias.
- status: Se **yes**, é criada a barra de status na base da janela.
- toolbar: Se **yes**, é criada a barra de ferramentas da janela.
- width: Largura (em pixels) da nova janela.

d) Método print()

Imprime o conteúdo da janela em questão.

Sintaxe:

```
window.print();
```