Índice

Índice

Índice

1 Introdução ao Javascript

A Internet constituiu−se no maior e um dos mais importantes meios de comunicação e difusão de informações da humanidade. Sua estrutura, prega a liberdade de viajar num imenso caminho sem fim, onde cada informação obtida, leva a uma outra informação nova, de forma que, ao final desta viagem, o ponto de partida passa a não existir mais. Somente existe o destino, aliás, muitos destinos.

É comum você se perder nas milhões de páginas, com seus detalhes, imagens, textos e sons.Às vezes, nem se lembra mais o motivo de sua visita. Mas não foi sempre assim. Digo, a atratividade da Internet nem sempre existiu. Houve um tempo, em que textos mortos e informações brutas, existiam únicos na grande rede.

No início, quando a Internet ainda era um projeto militar e restrito a algumas poucas universidades, as informações que transitavam por ali tinham o simples objetivo de coexistir em lugares diferentes, para que, em uma eventual guerra, pudessem preservar tais importantes documentos. Não havia preocupação com a estética das páginas ou intenções de atrair visitantes.

1.1 HTML e Javascript

Surgiu então o HTML − HyperText Markup Language. O HTML tornou−se a "linguagem" padrão da Internet, que hoje não existe sem ela. Mas o que é o HTML? É uma linguagem de marcação de hipertexto, ou seja, nada mais faz do que formatar e diagramar as informações que vemos no navegador. Imaginem um editor de textos. Tudo que fazemos nele, é trabalhar nosso texto, de forma a torná−lo mais interessante e bem apresentável aos nossos olhos. O HTML nada mas faz do que isso. Torna as páginas bonitas e organizadas. Dessa forma conseguem atrair a atenção para elas.

Será que só o HTML faz tudo isso? Consegue sozinho sustentar toda a Web?

A resposta é ***não***. A internet é muito mais do que páginas. É uma grande mistura de linguagens, protocolos, convenções e outras tantas coisas. Tudo isso é gerenciado por um órgão, chamado W3C. Ele regulamenta e disponibiliza os padrões a serem utilizados na Web de forma a termos um mínimo de organização.

Mas estamos aqui para falar de Javascript. Onde ele entra nesta história? Como dito anteriormente, o HTML apenas formata e diagrama as informações. Dissemos também, que a Web é um grande mix de várias outras coisas. Essas outras "coisas" surgiram para dar vida a Internet. As páginas que continham apenas HTML não podiam interagir com o usuário. Eram ***estáticas***. Não podiam enviar mensagens, ou permitir que você participasse de um chat. Imaginem agora a Internet sem enquetes, janelas popups − parece brincadeira, mas as vezes elas são importantes, botões para serem clicados, webmails, informações novas a cada instante. Imagino que sem tudo isso, não estaria escrevendo este livro agora.

Surgiram então as linguagens de script. Agora sim, tudo isso que falamos acima tornou−se possível e não preciso mais pensar em arrumar um outro emprego.

Entre essas novas linguagens, surgiu uma de nome ***Livescript***, de uma empresa chamada Netscape. Em uma associação de marketing, entre a Netscape e a Sun Microsystems, esta linguagem passou a se chamar ***Javascript***. Muita atenção: ***Javascript não é Java***. O Java é uma linguagem muito poderosa e independente, totalmente orientada a objetos. Difere−se principalmente do Javascript, por produzir aplicações independentes de um navegador. O Java pode produzir ainda, pequenas aplicações chamadas Applets que, apenas neste caso, precisariam do auxílio do navegador para serem visualizadas.

O Javascript somente é executado, ou melhor, interpretado por um navegador. Isso indica que é uma linguagem que chamamos de Client Side − acontece na máguina do usuário que está acessando a página. Por isso, tem−se um processamento muito rápido e uma interação com o computador cliente, tornando as páginas ***dinâmicas***. Podemos dizer também, que trata−se de uma linguagem baseada em objetos, ou seja, a princípio,

todos os elementos HTML são tratados como objetos.

O Javascript está hoje, na sua versão 1.5. É importante ressaltar que a maioria dos navegadores oferecem suporte para a execução de scripts, embora nem sempre estejam atualizados com as novidades da linguagem. Significa que, dependendo da nossa ferramenta de navegação, poderemos enfrentar alguns problemas na execução de nossos códigos, porém não chegam a interromper nosso trabalho.

Muito bem! Acho que conversamos bastante. Para este nosso aprendizado, vamos precisar de três coisas básicas:

- um editor de textos de sua preferência
- um navegador de sua preferência
- muita criatividade

No próximo capítulo, veremos alguns elementos do Javascript, seus comandos básicos e como este poderá interagir com o HTML. Então, mãos a obra!!

1.2 Resumo do capítulo

- A Internet é constituída de vários elementos: linguagens, protocolos, padrões.
- O HTML é a linguagem básica da Web. Sem HTML, não há Web.
- O HTML é uma linguagem de formatação e diagramação das informações dispostas na Internet.
- Javascript é uma linguagem de script Client Side, baseada em objetos e interpretada por um navegador.
- Javascript não é Java.

2 Elementos da Linguagem

Neste capítulo, iremos nos ambientar com o Javascript, conhecer um pouco suas características básicas e estruturas. É importante saber, quais os recursos de programação básica a linguagem poderá nos oferecer. Assim podemos expandir nossos horizontes de programação e conseqüentemente obter melhores resultados nos nossos projetos. Veremos a seguir como utilizar variáveis, estruturas de controle, operadores, laços de repetição e estruturas condicionais.

Por enquanto, vamos tentar manter o escopo de aprendizado associado à lógica de programação, objetivando somente conhecimento da sintaxe e algoritmos. Nos próximos capítulos praticaremos o que aprendermos agora.

2.1 Variáveis

O uso de variáveis em programas de computador é necessário. A medida que o código for se desenvolvendo, uma hora ou outra precisaremos de um auxílio desse recurso. As variáveis em Javascript não necessitam de declaração, mas recomendo que se faça a indicação de seu uso.

Declarando as variáveis, conseguimos evitar que algumas diferenças de versão impeçam a execução correta de nossos códigos.
A declaração implica em uma melhor legibilidade das linhas de código escritas.

Para declararmos uma variável em Javascript utilizamos a palavra reservada ***var***. Devemos observar alguns detalhes nesta declaração:

- os nomes de variáveis deverão iniciar sempre por caracteres alfabéticos ou pelo sinal de sublinhado "_";
- na seqüência do nome, números e "_" são permitidos;
- as variáveis não podem conter espaços;
- em Javascript, as variáveis são ***casesensitive***, ou seja, há diferenciação de maiúsculas e minúsculas;

Vejamos alguns exemplos:

```
var nome          = "Sistemas Abertos";
var _curso        = "Javascript";
var aula1         = "Introdução à Linguagem";
var 1aula         = "Introdução à Linguagem";//errado
var nome completo = "José da Silva";//errado

/*
Observem o uso dos comentários. Comentar e documentar o código é muito importante para bom
entendimento e manutenção do sistema, além de ser uma ótima prática de boa programação.

Comentários de apenas uma linha podem ser feitos através da inclusão de "//" no início da
linha.

Comentários de múltiplas linhas são feitos como este texto que você está lendo, através da
inclusão dos limitadores no início e fim do bloco.
*/
```

O uso de ";" no final de cada linha de comando, também é muito útil na organização do código, embora seja dispensado da obrigatoriedade.

2.2 Tipos de dados

As variáveis em Javascript podem assumir alguns tipos de dados. São eles:

- ***Strings***: utilizado para representar seqüências de caracteres. Devem estar sempre envoltas por "".

```
var nome  = "José";
var cpf   = "123456789−00";
var idade = "25";
```

- ***Inteiros***: é a representação dos números inteiros positivos e negativos.

```
var ano   = 2004;
var zero  = 0;
var erro  = −1;
```

- ***Double*** ou ***ponto flutuante***: são os números que apresentam uma segunda precisão, com indicação de maior exatidão.

```
var peso    = 58.9;

//Podem ser representados também por notação científica.
var grande  = 350E12;//equivale a 350*10 elevado a 12ª potência
```

Temos ainda dois outros tipos de dados em Javascript: ***arrays*** e ***objetos***. Iremos falar mais detalhadamente nos capítulos seguintes.

Embora tenhamos toda essa definição de tipos, torna−se pouco comum a utilização de variáveis baseada em suas características. Em programação para Internet é normal todas as variáves assumirem diferentes tipos a cada instante, basta que, para isso, sejam−lhes atribuídos novos valores, de diferentes tipos dos quais já foram alocados.

2.3 Operadores

- Atribuição

  Os operadores de atribuição são utilizados para destinar valores às variáveis. Seu operador principal é representado pelo sinal de igual(=). É possível também, atribuir valores às variáveis e, ao mesmo tempo, realizar operações matemáticas com os operandos. Vejamos alguns exemplos:

```
var nome = "José";//lê−se: nome recebe a string José
var num1 = 5;//lê−se: num1 recebe o inteiro 5
num1    += 3;//equivalente a: num1 = num1 + 3
num1    −= 3;//num1 = num1 − 3;
num1    *= 3;//num1 = num1 * 3;
num1    /= 3;//num1 = num1 / 3;
num1    %= 3;//num1 = num1 % 3;
```

- Matemáticos

Os operadores matemáticos possuem representação semelhante ao que comumente utilizamos na álgebra. Os sinais +, −, * e /, são respectivamente adição, subtração, multiplicação e divisão. Tem−se ainda um operador de módulo, que produz o resultado do resto de uma divisão não inteira. Este operador tem a simbologia da porcentagem(%). Observem:

```
var total = 2 + 3;//produz 5
total     = 6 − 1;//5
total     = 2 * 3;//6
total     = 6 / 3;//2
total     = 5 % 2;//1 − resto da divisão
```

2.4 Comparação

A comparação entre valores pode ser feita de várias formas. A mais comum é a comparação de igualdade. Junto com ela, observamos comparações de superioridade, inferioridade e diferença, além da combinação permitida dos conjuntos citados acima. As comparações sempre retornarão um resultado booleano, indicando se houve ou não sucesso na operação.

```
var num1 = 5;
var num2 = 4;

/*
utilizarei a linguagem de algoritmos para exemplificar, enquanto ainda não aprendemos as
estruturas de controle
*/
se( num1 == num2 ) ...//lê−se: se num1 é igual a num2 − retorna false
se( num1 >  num2 ) ...//lê−se: se num1 é maior do que num2 − retorna true
se( num1 <  num2 ) ...//lê−se: se num1 é menor do que num2 − retorna false
se( num1 >= num2 ) ...//lê−se: se num1 é maior que ou igual a num2 − retorna true
se( num1 <= num2 ) ...//lê−se: se num1 é menor que ou igual a num2 − retorna false
se( num1 <> num2 ) ...//lê−se: se num1 é diferente de num2 − retorna true
se( num1 = num2 ) ...//lê−se: se num1 é diferente de num2 − retorna true
```

- Incremento e Decremento

Estes operadores realizam um adição(++) ou subtração(−−) de apenas uma unidade, na variável a que for aplicado. Atenção para a ordem de utilização destes operadores. Sua seqüência em relação a variável altera o resultado da operação. Veja o exemplo abaixo:

```
var num1 = 5;

//novamente utilizaremos a linguagem de algoritmos
imprima( num1++ );//imprime 5; logo após a impressão, num1 passará a ter valor 6
imprima( ++num1 );//imprime 7; neste caso, há o incremento de num1 antes da impressão
imprima( num1−− );//imprime 7; a impressão ocorre antes que haja o decremento
imprima( −−num1 );//imprime 5; na expressão anterior, num1 havia obtido o valor 6
```

- Lógicos

Os operadores lógicos são muito úteis quando há a necessidade de se fazer múltiplos testes entre vários elementos. Basicamente temos três operadores lógicos: ***e lógico(&&)***, ***ou lógico(||)*** e ***negação(!)***.

- ***e lógico(&&)***: para que o resultado seja verdadeiro, todas as expressões testadas devem ser verdadeiras:

```
var nome1 = "José";
var nome2 = "José";

se( nome1 == "José" && nome1 == nome2 ) ...//lê-se: se nome1 é igual a José e nome1 é igua
/*
Neste caso, o resultado de toda a expressão é verdadeiro, porque ambas as expressões inter
foram satisfeitas. Se nome2 tivesse o valor "Maria", o resultado final seria falso, porque
segunda expressão não foi satisfeita.
*/
```

- ***ou lógico(||)***: neste caso, se uma das expressões internas for verdadeira, já basta para que toda a expressão também o seja:

```
var nome1 = "José";
var nome2 = "Maria";

se( nome1 == "José" || nome1 == nome2 ) ...//lê-se: se nome1 é igual a José ou nome1 é igu
/*
Observem que a segunda expressão não é verdadeira. Neste caso não importa, porque a expres
1 já satisfez a condição. Somente teríamos um resultado falso nesta expressão se ambas as
internas não satisfizessem as condições.
*/
```

- ***negação(!)***: este operador nada mais faz do que negar toda um expressão. Pode ser

utilizado também na comparação de variáveis, substituíndo o operador de diferença(<>) por (=). Vejam:

```
se ( !x ) ...//lê-se: se x não existir ou se x não estiver setado ou se x não for vazio
se ( a = b ) ...//lê-se: se a for diferente de b ou se a não for igual a b
```

- Concatenação

Este operador, representado aqui pelo sinal(+) concatena variáveis, sejam elas numéricas ou simplesmente texto. Pergunta: mas se utilizarmos o sinal de (+) relacionando dois números? Não seria uma adição?

Teoricamente sim, mas no Javascript, veremos que, em determinados casos, ao "somarmos" valores numéricos advindos de campos textos, estes não serão adicionados e sim concatenados. Em uma operação de inteiros teríamos uma soma. Mas se um dos operandos for uma string, teremos uma concatenação. Observem os exemplos abaixo:

```
var a = "Curso de ";
var b = "Javascript";

imprima( a + b );//imprime: Curso de Javascript

var c = 5;
var d = "8";

imprima( c + d );//imprime: 58; c é um inteiro, mas d uma string, por isso não há a adição
```

```
var e = 5;
var f = 8;

imprima( e + f );//agora sim irá imprimir 13, pois ambas as variáveis são inteiras
```

2.5 Estruturas de Controle e Laços de Repetição

- if, else

Utilize esta estrutura para realizar testes nos seus códigos. Caso a condição testada seja verdadeira, um código é executado, caso contrário, outro é. Sinais de chave({}) são delimitadores de código. Em caso de haver apenas uma linha de comando, as chaves são dispensadas. Observem:

```
var a = 2;

if( a == 2 )
{
        //os comandos vêm aqui
        //somente serão executados se a condição acima for satisfeita
}
else
{
        //os comandos vêm aqui
        //se a condição proposta for falsa, o bloco else é executado
}
```

- function

Funções em Javascript são estruturas muito utilizadas, por serem excelentes concentradores de códigos e permitirem objetivos específicos na obtenção dos resultados. Podem ainda trabalhar como métodos de objetos, desde que sejam declaradas para isso. Veremos com mais detalhes no capítulo de objetos.

```
function cubo( a ) {
        /*
        Função que calcula o cubo de um número
        Recebe parâmetro numérico
        Imprime o resultado
        */
        imprima( a * a * a );//utilizando mais uma vez a linguagem de algoritmos
}

cubo( 5 );//funções só executam quando são chamadas
//deverá imprimir o número 125
```

- return

Na lógica de programação, aprendemos que funções retornam valores e procedimentos executam blocos de códigos. Em Javascript, uma ***function*** realiza as duas coisas. No exemplo acima, a função cubo imprimiu o resultado, não retornando nenhum valor para que a chamou. Neste próximo exemplo, vamos modificar o exercício de forma que a função retorne o resultado para o comando de impressão. Para isso, utilizamos o comando ***return***. Observem:

```
function cubo( a ) {
        /*
        Função que calcula o cubo de um número
        Recebe parâmetro numérico
        Imprime o resultado
        */
        return a * a * a;
}

imprima( cubo( 5 ) );//a impressão será feita com o resultado retornado pela função
//deverá imprimir o número 125
```

- for

Utilizamos esta estrutura, quando queremos que um bloco de código seja executado repetidas vezes, enquanto uma determinada condição se mantiver verdadeira. Sua sintaxe é:

for(valor inicial do contador; condição de execução; incremento do contador) { //bloco de código }

Vejam um exemplo:

```
for(i = 0; i < 3; i++) {
        /*
        Laço que será executado enquanto o contador i for menor do 3; O contador tem
        seu valor inicial igual a 0 e a cada impressão, seu valor se torna 1(uma) unidade
        maior, chegando ao ponto de tornar a condição falsa e interromper o laço.
        */
        imprima( i + " " );
}

/*
Resultado:
0 1 2
*/
```

- while

O laço while é idêntico ao laço for no quesito finalidade. Ambos servem para executar blocos de código um determinado número de vezes. A diferença básica, é que no laço while, a condição é controlada pelo programador. Aqui não existe um contador explícito, o programador define a hora em que a execução deve ser interrompida.

while(condição de execução) { //bloco de código }

O exemplo anterior utilizando um laço while:

```
var contador = 0;//esta variável fará o controle da execução do laço

while( contador < 3 ) {
        /*
        Este bloco de código será executado até que contador atinja o valor 3.
        Observem que a variável contador deve ter seu valor alterado a cada volta
        do laço, para que, em algum momento torne a condição falsa, evitando assim
        um laço infinito, o que poderia travar o seu navegador.
        */
        imprima( i + " " );
```

```
        contador++;
}

/*
Resultado:
0 1 2
*/
```

- break

O comando break é utilizado para interromper a execução de um laço, quando uma segunda condição for estabelecida, e, claro, desde que seja interesse do código. No momento em que um break acontece, todo o código entre o break e o delimitador } deixa de ser executado. O programa seguirá sua execução normal a partir da linha subseqüente ao delimitador }. Este comando é geralmente utilizado em laços de repetição, como o for e o while.

```
/*
Utilizando o exemplo anterior, criaremos um laço que será interrompido, caso a variável
de controle contador, assuma o valor 2. Vamos aumentar a limitação desta variável para o
valor 5
*/
var contador = 0;//esta variável fará o controle da execução do laço

while( contador < 5 ) {
        if( contador == 2 ) {
                break;
        }

        imprima( i + " " );
        contador++;
}
/*
Quando a linha de código executar o comando break, a execução do programa continuará
desta linha e os comandos imprima( i + " ") e contador++ não serão executados.

O resultado:
0 1
*/
```

- continue

O comando continue é bastante parecido com o break, exceto pelo fato dele não interromper totalmente a execução do laço. No momento em que o programa encontra o continue, todo o código entre o comando e o delimitador } deixa de ser executado, porém, a execução volta a testar novamente a condição. Observe o exemplo abaixo:

```
/*
Substituiremos o comando break do exercício anterior, pelo continue. Veremos em que isso
influenciará no resultado final.
*/
var contador = 0;//esta variável fará o controle da execução do laço

while( contador < 5 ) {
        if( contador == 2 ) {
                continue;
        }

        imprima( i + " " );
```

```
        contador++;
}
/*
No momento da execução do continue, os comandos imprima( i + " " ) e contador++,
deixam de ser executados. Mas, o programa volta a testar a condição e a execução do
código terá sua normalidade restabelecida.

O resultado final será:
0 1 3 4
*/
```

- switch

  O switch é uma estrutura bastante útil, porque seu uso permite uma considerável economia de código, legibilidade e alguns "ifs" a menos. Seu funcionamento é bastante simples. Uma variável inicial é testada e de acordo com seu valor, diferentes ações são executadas. Observem:

```
var dia_da_semana = 4;//suponhamos que este valor seja informado pelo usuário

switch( dia_da_semana ) {
        case 0 : imprima "Domingo"; break;
        case 1 : imprima "Segunda"; break;
        case 2 : imprima "Terça";   break;
        case 3 : imprima "Quarta";  break;
        case 4 : imprima "Quinta";  break;
        case 5 : imprima "Sexta";   break;
        case 6 : imprima "Sábado";  break;
        default: imprima "Dia inexistente";
}
/*
Este código testa o valor de dia_da_semana e executa uma ação de acordo com o seu valor.
É muito importante não esquecer o comando break. Uma vez que o caso já foi descoberto,
não há a necessidade de se continuar executando o código do switch. Mas, se você esquecer,
verá que ele continuará executando todas as ações até que um break ou o delimitador de
fechamento } sejam encontrados. Se nenhum caso foi satisfeito, a ação executada será a
contida na linha de comando default.
*/
```

Em Javascript, possuímos ainda dois outros tipos de estrutras de controle: ***for...in*** e ***with***. Estas duas estruturas são utilizadas quando manipulamos objetos. Como ainda não conhecemos objetos, é preferível visualizarmos estas estruturas em seus momentos adequados. Guardem esses nomes para que possamos lembrar deles no capítulo de objetos.

2.6 Resumo do capítulo

- As variáveis são úteis quando queremos preservar valores na memória. Sua nomenclatura deve ser iniciada com qualquer caracter alfabético ou pelo sinal "_", permitindo em seguida números ou "_".
- As variáveis em Javascript são casesensitive.
- As variáveis em Javascript não precisam ser declaradas, embora seja aconselhado a informação de seu uso.
- O ponto−e−vírgula não é necessário para a finalização das linhas de comando, mas é interessante que seja utilizado, por auxiliar na legibilidade do código.
- Utilize comentários para documentar e informar seu código.
- Temos cinco tipos de dados em Javascript: strings, inteiros, double, objetos e arrays.
- Os operadores de atribuição podem ser: , +, −=, *=, /= e %=.

- Para operações matemáticas básicas utilize: +, −, *, / e %.
- Para comparar: `=, >, , >=, <> e =.`
- Os operadores de incremento e decremento: var++, ++var, var−− e −−var.
- Para comparações lógicas utilize: &&, || e !.
- Utilize o sinal + para concatenar valores.
- Em Javascript temos estruturas que nos auxiliam na codificação: if/else, function, return, for, while, break, continue, switch, for...in e with.

2.7 Exercícios de fixação

1. Declare as seguintes variáveis e atribua valores correspondentes aos seus tipos: nome(string), idade(inteiro), peso(double).
2. Utilize uma estrutura de controle para descobrir se o valor de sua idade é par ou ímpar.
3. Faça um laço para imprimir todos os anos, do ano atual até o ano de seu nascimento.

3 Javascript Básico

Agora que já aprendemos um pouco da estrutura básica da linguagem Javascript, podemos enfim, nos aventurar pelas várias possibilidades de programação web client que ela nos oferece. Nesta nova etapa, vamos fazer nosso primeiro programa utilizando esta linguagem. Para isso, será necessário aprender como utilizar o Javascript em conjunto com o HTML, como inserir os códigos nas páginas, utilizar bibliotecas externas e alguns novos elementos.

Vamos ter também, nosso primeiro contato com os objetos. Veremos o quão importante eles são e o espaço que ocupam no âmbito do Javascript.

3.1 Inserindo código Javascript

A partir de agora, será necessária a utilização de um editor de textos e um navegador. Fique a vontade para escolherem o de sua preferência.
Salvem sempre seus arquivos sem nenhum tipo de formatação. Dessa forma teremos certeza que o navegador conseguirá interpretar corretamente o código.

A utilização de Javascript nas páginas de Internet é feita principalmente através da tag "script". Normalmente, o código Javascript principal, é inserido dentro da tag "head" de um documento HTML. É comum concentrar neste local, o conjunto de todas as funções e declarações a serem utilizadas em toda a página. Vejam como ficaria um documento utilizando Javascript:

```
.<html>
.    <head>
.        <script language="Javascript">
.            //declarações
.        </script>
.    </head>
.    <body>
.        <!-- código html -->
.    </body>
.</html>
```

Temos acima, um documento HTML com uma inserção de código Javascript. A tag ***script*** possui a propriedade ***language***, que permite especificar qual a linguagem e respectiva versão que está sendo utilizada no documento. Poderíamos ter uma outra definição de language:

```
.<script language="Javascript1.5">
```

Neste livro adotaremos a primeira forma.

Observem que o código acima foi escrito de forma a facilitar o entendimento do mesmo. Identem o código, pois isso é uma boa prática de programação.
Notem que em alguns momentos do texto, mencionamos a palavra ***documento***. No Javascript, as páginas são chamadas de documentos ou ***document***.

Para que possamos enfim, criarmos nossa primeira aplicação em Javascript, é necessário entender mais algumas particularidades da Linguagem.

Escrevam todos os comandos minúsculos. O Javascript faz distinção de maiúsculas e minúsculas de seus comandos. Embora essa não seja a regra principal, evitaremos menos erros na escrita. Veremos que, ao mencionarmos os métodos e propriedades dos objetos, veremos que seus nomes sempre começarão minúsculos, e a partir da segunda palavra, todas as primeiras letras são maiúsculas. Mas não se preocupem agora. Procurem ambientar−se com a linguagem e as regras iremos dominando aos poucos.

Nossa primeira aplicação, consistirá em escrevermos na tela o texto "Alô Mundo". Não é muito original, mas temos que começar de alguma forma. Vejamos o código a seguir e as considerações logo após o seu desfecho. Lembrem−se de salvar seus documentos sem nenhum tipo de formatação textual.

AloMundo.htm

```
01. <html>
02.     <head>
03.         <title>Aplicação Alô Mundo!!!</title>
04.         <script language="Javascript">
05.             document.write( "Alô Mundo!!!" );
06.         </script>
07.     </head>
08.     <body>
09.     </body>
10. </html>
```

Abra o arquivo no navegador de sua preferência e visualize o resultado. Neste exemplo, conhecemos nosso primeiro "comando" Javascript. Não entraremos no mérito dos objetos agora. Basta saber que o ***método write***, que pertence ao *objeto document*(página) é utilizado para escrever na tela. No próximo capítulo conheceremos um pouco mais a respeito dessa sintaxe.

Vamos aperfeiçoar nosso exercício. Vamos trabalhar a impressão para que tenha uma formatação mais apresentável. Sabemos que a linguagem HTML é responsável por formatar e diagramar o conteúdo de uma página. Mas como fazer isso, se quem está imprimindo o texto é o Javascript?

Isso é bastante simples. Todo elemento HTML que precisar ser utilizado dentro de código Javascript, deverá ser impresso na tela, exatamente como o texto "Alô Mundo!!!". Veja o exemplo alterado:

AloMundo2.htm

```
01. <html>
02.     <head>
03.         <title>Aplicação Alô Mundo!!!</title>
04.         <script language="Javascript">
05.             document.write( "<center><font color=\"#FF0000\">Alô Mundo!!!</font></center>" );
06.         </script>
07.     </head>
08.     <body>
09.     </body>
10. </html>
```

3.2 Utilizando um arquivo de script externo

Vejamos agora, como utilizar um arquivo externo que poderá ser uma fonte de rotinas úteis a todas as páginas do nosso site. Isso é importante pois implica em melhor manutenção e maior agilidade na escrita do código, porque não há a necessidade de se escrever tudo novamente para cada página.

Recortem a linha 05 e a colem em um arquivo, que aqui vamos chamar de biblioteca.js. Depois modifiquem o arquivo AloMundo2.htm da seguinte forma:

biblioteca.js

```
document.write( "<center><font color=\"#FF0000\">Alô Mundo!!!</font></center>" );
```

AloMundo2.htm

```
01. <html>
02.     <head>
03.         <title>Aplicação Alô Mundo!!!</title>
04.         <script language="Javascript" src="biblioteca.js"></script>
05.     </head>
06.     <body>
07.     </body>
08. </html>
```

Agora carreguem novamente a página. Ela foi exibida da mesma forma que as anteriores. Isso aconteceu porque, através da propriedade ***src*** da tag ***script***, informamos ao navegador que ele deve procurar o código Javascript no arquivo ***biblioteca.js***. Este arquivo poderá ser utilizado por qualquer página, desde que se faça a referência de seu uso.

3.3 O que fazer quando não acontece nada?

Algumas vezes, ao testar o script, nada acontece. Na verdade um erro ocorreu, mas o seu navegador ou alguma configuração que ele esteja usando, não mostrará nenhum tipo de informação de que aconteceu alguma coisa inesperada.

Para descobrirmos o que aconteceu, precisamos entender como o navegador está programado para tratar disso. No Mozilla e Netscape, precisamos digitar na caixa de endereços a seguinte url: ***javascript:***. Dessa forma, o navegador exibirá o erro, ou uma lista de erros que ocorreram. A partir daí, é possível fazer a depuração do código.

No Internet Explorer, é comum aparecer um ponto de exclamação no canto inferior direito da janela. Clicando duas vezes neste ícone, uma janela informando o erro será aberta. Há ainda neste navegador, uma opção para que os erros sejam mostrados na tela na hora em que ocorrerem. Vá no menu Ferramentas, submenu Opções da Internet. Na guia Avançadas, procure o item Navegação. Marque a checkbox com o texto "Exibir notificação sobre cada erro de script". Assim, os erros poderão ser tratados com maior facilidade.

Em ambos os navegadores, os erros são exibidos com uma mensagem pertinente ao que aconteceu, bem como a linha que gerou o erro. Nem sempre esta linha está correta, mas indica com certeza, a proximidade do código causador do erro.

3.4 Resumo do capítulo

- Para inserir código Javascript em um documento HTML, utilizamos a tag ***script***.
- Código HTML dentro de código Javascript deve ser impresso.
- Identar o código é uma boa prática de programação.
- Páginas são chamadas de ***document*** em Javascript.
- Arquivos externos podem ser referenciados através da propriedade ***src*** da tag ***script***.
- A princípio, todos os comandos Javascript são escritos em minúsculas. Atenção para a escrita de métodos e propriedades dos objetos, em que, apenas as primeiras letras, a partir da segunda palavra que compõe o comando, são maiúsculas.
- Mensagens de erros são compostas de explicação do erro e número da linha que gerou o mesmo.

3.5 Exercícios de fixação

1. Crie um código que imprima duas variáveis: nome e sobrenome. Utilize HTML para melhorar a atratividade visual.
2. Transporte esse código para um arquivo externo e faça sua página chamar este arquivo.
3. Experimente escrever o método ***write*** com a letra maiúscula. Visualize o erro e entenda a mensagem que foi gerada. A página foi executada corretamente?
4. Como carregar um arquivo de biblioteca externo?

```
a <script language="JavaScript">arquivo.js</script>
b <script language="JavaScript">document.include(arquivo.js)</script>
c <script language="JavaScript">document.write("arquivo.js")</script>
d <script language="JavaScript" src="arquivo.js"></script>
e <script language="JavaScript"><include "arquivo.js"></script>
```

4 Objetos I − Entendendo a Hierarquia de Objetos

No Capítulo I, aprendemos que a linguagem Javascript é baseada em objetos. Isto nada mais significa, do que, interpretar todos os elementos HTML como objetos. Além destes, temos outros, tais como a janela do navegador, o próprio navegador, objetos core do Javascript e, nossos próprios objetos.

Veremos cada um deles em detalhes. Utilizaremos seus métodos e suas propriedades em favor de nossa aplicação. Mas antes é necessário compreender como estes objetos estão organizados. Observem o diagrama abaixo:

```
window(objeto pai)
--->document
    --->forms(text, textarea, radio, button, checkbox, select, select multiple, file, subm
    --->links
    --->images
    --->applets
    --->embeds
    --->plugins
    --->anchors
--->frames(coleção de objetos document)
--->navigator
--->location
--->history
--->event
--->screen
```

Acima, estão listados alguns dos objetos mais importantes do Javascript. Falamos em hierarquia, porque existe uma ordem de acesso aos objetos e suas propriedades e métodos, ordem essa que deve ser obedecida.

Falou−se muito em métodos e propriedades e objetos. Mas o que é isso tudo?

Quando falamos a respeito de objetos, queremos dizer que, eles são a representação de algum elemento do mundo real. Pessoas, carros, computadores... tudo isso pode ser representado através de objetos. E da mesma forma que esses elementos possuem características que os identificam, os objetos também as tem. As propriedades dos objetos podem ser acessadas e manipuladas através da seguinte sintaxe:

```
objeto.propriedade
```

Supondo que temos um ***Fusca 1977 branco***, vamos tentar fazer uma analogia ao mundo dos objetos. Observem que caracterizamos nosso carro da forma como o vemos e conhecemos. No mundo dos objetos, a essas características, damos o nome de ***propriedades***. Vamos transformar nosso Fusca em um objeto genérico chamado ***carro***. A partir daí, queremos acessar suas propriedades. Como já conhecemos a sintaxe necessária, vamos então visualizá−la:

```
//imprimindo as propriedades do objeto carro
document.write(carro.modelo, '<br>');
document.write(carro.ano, '<br>');
document.write(carro.cor, '<br>');

/*
Resultado:
.    Fusca
.    1977
.    branca
*/
```

O código acima exemplifica somente como teríamos acesso às propriedades de um objeto. No entanto, ainda não está pronto para ser utilizado. Precisaríamos criar nosso objeto para termos aquele acesso. Faremos isso nos capítulos seguintes. Até lá, vamos trabalhar com objetos prontos do Javascript.

Ainda de posse de nosso exemplo, sabemos que um carro pode buzinar, andar, quebrar. Essas ***ações*** que o carro pode fazer, são chamadas de ***métodos***. É muito simples identificar um método. Vejam que todos as nossas ações são verbos no infinitivo. Outra forma de identificar um método, são os parênteses que o acompanham. Como executam alguma coisa, é normal que o façam de acordo com a nossa vontade. E podemos expressar nossa vontade, enviando parâmetros de execução para um método. Vejam a sintaxe:

```
objeto.metodo()
```

É idêntico ao acesso de uma propriedade, exceto pelos parênteses. Agora, fazendo a mesma analogia com o nosso carro:

```
//executando os métodos do objeto carro
carro.buzinar('vaca');
carro.andar();
carro.quebrar();

/*
.    Imaginem uma dessas buzinas onde escolhemos qual som deve ser emitido quando a
.    acionarmos. Os parâmetros dos métodos existem para que possamos dizer, como
.    queremos que ele atue. Para cada valor passado, o método buzinar() terá uma forma
.    de agir.
*/
```

Agora já estamos aptos a entender o comando ***document.write()***, que vimos no capítulo anterior. Na verdade, trata−se do método ***write*** do objeto ***document***, ou seja, representa a capacidade que o objeto document tem de escrever na tela.

Vejam no exercício abaixo, algumas características e métodos do principal objeto com o qual iremos trabalhar: ***document***. É o principal porque é a representação visual da sua página. A manipulação que você fizer neste objeto, será imediatamente refletida na visualização da página. Podemos dizer que ele é a tag ***body*** do HTML representada por um objeto.

```
01. <html>
02.     <head>
03.         <title>Visualizando propriedades e métodos</title>
04.         <script language="Javascript">
05.             //imprime o texto na tela com a formatação de titulo <h3>
06.             document.write("<h3>Conhecendo propriedades e métodos!!!</h3>");
07.
08.             //alterando a cor de fundo do documento
09.             document.bgColor    = '#EAEAEA';
10.
11.             //alterando o título da página
12.             document.title      = 'O título foi alterado.';
13.
14.             //alterando a cor do link
15.             document.linkColor  = '#00FF00';
16.
17.             //alterando a cor do link quando ativo
18.             document.alinkColor = '#00FF00';
19.
20.             //alterando a cor do link que foi visitado
21.             document.vlinkColor = '#00FF00';
22.
23.             /* imprimindo a data e hora da última atualização da página.
```

```
24.                Observem que um texto foi concatenado com a propriedade
25.                lastModified do objeto document.
26.             */
27.             document.write( 'Atualizado em: ' + document.lastModified);
28.         </script>
29.     </head>
30.     <body>
31.         <hr>
32.         <!-- Esta tag href foi colocada apenas para visualizarmos as propriedades dos
33.              que alteramos com o Javascript acima.
34.         -->
35.         <a href="http://www.sistemasabertos.com.br">Sistemas Abertos</a>
36.     </body>
37. </html>
```

⚠ Atenção para a diferenciação de maiúsculas e minúsculas dos comandos. Lembrem−se que, a partir da segunda palavra de um comando, a primeira letra é maiúscula.

Acima estão algumas das propriedades e métodos do objeto document. É claro que existem um pouco mais, inclusive, objetos que pertencem ao document e que necessitam ser referenciados através da hierarquia, tal como o objeto forms. Na seqüência dos próximos capítulos vamos nos familiarizar melhor com isso.

Um outro objeto muito importante do Javascript, e que nos permite trabalhar com a janela e seus objetos internos é o ***window***. Com este objeto, conseguimos interagir com o usuário e algumas características da máquina cliente, como resolução de tela e definição de cores. A seguir, veremos alguns exemplos da utilização deste objeto.

4.1 Exibindo mensagens de alerta

Muitas vezes nos deparamos com alguma mensagem de alerta nos sites em que navegamos. Algumas informam erros, outras cumprimentos e boasvindas. Sua sintaxe é simples e seu uso pode ser bastante útil. Observem o exemplo:

```
01. <html>
02.     <head>
03.         <title>O método alert</title>
04.         <script language="Javascript">
05.             alert('Seja bemvindo ao Curso de Javascript');
06.         </script>
07.     </head>
08.     <body>
09.         <center><h3>Página do Curso de Javascript</h3></center>
10.     </body>
11. </html>
```

💡 Métodos e propriedades do objeto window não precisam ser acessadas pela sintaxe: objeto.metodo() ou objeto.propriedade. Podem ser utilizadas diretamente pelo seu nome. Isso acontece porque o ***window*** é o objeto de maior hierarquia, sendo facultativa a sua menção.

No exemplo acima, utilizamos o método ***alert()*** que pertence ao objeto ***window*** para enviar uma mensagem de alerta para o usuário. O parâmetro utilizado neste método é a própria mensagem de interação.

4.2 Exibindo caixa de diálogo de confirmação

Agora iremos interagir com o usuário, solicitando deste, uma confirmação para que seja executada ou não uma determinada ação. Para isso vamos utilizar o método ***confirm()*** do mesmo objeto ***window***.

```
01. <html>
02.     <head>
03.         <title>O método confirm</title>
04.         <script language="Javascript">
05.             if( confirm( 'Esta é uma página proibida! Deseja mesmo acessá−la?' ))
06.                 document.write( '<h4>Você concordou em acessar a página.</h4>');
07.             else
08.                 document.write( '<h4>Você não concordou em acessar a página.</h4>');
09.         </script>
10.     </head>
11.     <body>
12.         <center><h3>Página Proibida!!!</h3></center>
13.     </body>
14. </html>
```

O método ***confirm()*** retorna um valor booleano. True se o usuário clicar em "OK" e false se o botão clicado for "Cancel". Dessa forma, o programa entrará no if se a condição for satisfeita(clique no botão "OK"), ou no else, se o retorno do teste for falso(clique no botão "Cancel").

4.3 Exibindo caixa de diálogo de entrada de dados

Neste exemplo, vamos solicitar que o usuário informe um número. Então o cubo desse número será calculado. Vamos utilizar a função cubo(a) que exercitamos nas primeiras aulas.

```
01. <html>
02.     <head>
03.         <title>O método prompt</title>
04.         <script language="Javascript">
05.             function cubo( a )
06.             {
        07.                 return a * a * a;
        08.             }
09.
10.             var numero = prompt( 'Informe o número a ser calculado:', '' );
11.
12.             document.write( 'O cubo de ' + numero + ' é ' + cubo( numero ) );
13.         </script>
14.     </head>
15.     <body>
16.         <center><h3>Calculando o cubo de um número!!!</h3></center>
17.     </body>
18. </html>
```

Nas linhas 5, 6 e 7, declaramos a função cubo(), que recebe um parâmetro ***a*** e retorna este valor multiplicado por ele mesmo por três vezes(cubo matemático).

Na linha 9, criamos uma variável ***numero*** quer irá receber o resultado do método ***prompt***. Este método possui dois parâmetros, sendo que o segundo é opcional. O primeiro é a mensagem de interação com o usuário. O segundo, é o valor inicial da caixa de texto que será aberta na janela de diálogo.

Embora seja opcional, é interessante que se coloque pelo menos uma string vazia("), como fizemos. Assim

evitamos que apareça o texto ***undefined*** quando a caixa de diálogo for carregada.

Na linha 11, imprimimos o resultado, que é composto por um texto concatenado, com os valores originais de ***numero*** e o resultado da função ***cubo()***.

Nas próximas páginas, aprenderemos um pouco mais de propriedades e métodos destes dois importantes objetos: ***document*** e ***window***. A seguir, vamos conhecer um assunto até então não falado: os ***eventos***. Veremos que são a principal fonte de interação com os usuários e passaremos boas horas no estudo de suas possibilidades.

4.4 Resumo do Capítulo

- Em Javascript, os objetos estão dispostos em uma certa hierarquia, que deve ser observada.
- O objeto que está no topo da hierarquia, é o objeto ***window***.
- Objetos possuem propriedades(características) e métodos(funções).
- Para acessar uma propriedade de um objeto utilizamos: objeto.propriedade.
- Para executarmos um método de um objeto utilizamos: objeto.metodo().
- O Javascript faz diferenciação de maiúsculas e minúsculas de seus comandos.
- Os métodos ***alert()***, ***confirm()*** e ***prompt()*** pertencem ao objeto window, e por isso são acessados sem sua referência(métodos estáticos).

4.5 Exercícios de Fixação

1. Qual é a maneira correta de acessar a propriedade ***peso*** do objeto ***elefante***?
   - ♦ elefante[peso]
   - ♦ elefante.peso()
   - ♦ peso.elefante
   - ♦ elefante.peso
   - ♦ peso()
2. Crie uma rotina para perguntar ao usuário se ele deseja trocar a cor de fundo da página. Se ele clicar em 'Ok', peça para ele escolher entre três cores: blue, red, green. A cor digitada por ele deverá ser a cor de fundo da página.
3. Faça um script para solicitar o nome do usuário. Exiba uma mensagem de boas vindas com o nome informado.
4. Escreva uma função para receber do usuário 3(três) números e informar a sua média.
5. O que o código abaixo faz:

```
Alert('Olá Usuário');
```

- ♦ Escreve na página a mensagem "Olá Usuário".
- ♦ Exibe uma mensagem de alerta com o texto informado.
- ♦ Nada, porque não foi informado o objeto que contém este método.
- ♦ Acontece um erro.
- ♦ Pergunta ao usuário seu nome.

5 Formulários HTML

Neste momento, entramos no estudo dos formulários e formas de interação com o usuário. Aprenderemos que os objetos dos formulários, são sub−objetos do objeto ***forms***. Veremos as características individuais destes elementos e acessá−los via Javascript.

5.1 Um pouco de HTML

Formulários HTML é uma das principais formas de comunicação com o usuário. Com eles podemos "conversar" com as pessoas que acessam nosso site. Caixas de textos e botões são comumente usados para obter e mostrar informações. Observem os exemplos a sua volta: bancos online, webmails, chats. Tudo isso faz uso dos objetos dos formulários. Podemos perceber que sem eles, continuaríamos com a mesma monotonia das páginas estáticas.

Vamos então relembrar a porção da linguagem HTML que trata de formulários. Faremos um breve resumo dos seus recursos e características. Começaremos pela declaração de um formulário dentro de uma página:

```
<form name=f1 action="calcular.php" method="get" enctype="text/plain">
.
.
.
</form>
```

Mas o que significa tudo isso?

A declaração de um formulário se dá pela tag ***form***. Esta tag possui algumas propriedades. A propriedade ***name***, armazena o nome do meu formulário. Com este nome, podemos acessar qualquer propriedade ou sub−objeto do nosso formulário. Nas linguagens de script server−side, como o PHP, ASP ou JSP, o ***name*** será utilizado como nome da variável.

⚠ Se a propriedade ***name*** for esquecida, o acesso a este objeto poderá ser comprometido, embora haja outras formas de tratar isso.

Outra propriedade essencial à tag ***form***, é a ***action***. Esta propriedade informa qual script servidor, irá tratar as informações que foram enviadas. No exemplo acima, citamos um valor imaginário, mas com um nome sugestivo. Podemos entender que as informações remetidas, passarão por um processo de cálculo.

💡 Procure nomear seus objetos e arquivos com nomes sugestivos, que identificam previamente do que se trata tal elemento. Isso auxilia na manutenção e entendimento do código.

As propriedades ***method*** e ***enctype***, tratam da forma que as informações serão transmitidas e que tipo de informação estará transitando no momento do envio. A propriedade ***method*** pode assumir dois valores: ***get*** e ***post***. O primeiro diz ao servidor que as informações estão sendo passadas via url. O segundo informa que as informações estão indo ***como um anexo***, de forma que o usuário não tenha acesso as mesmas.

Finalmente, a propriedade ***enctype*** diz o tipo de dado que será enviado. O valor que está declarado no exemplo, ***text/plain***, indica que será apenas texto puro. Em casos de páginas que permitem upload de arquivos, dizemos que o ***enctype*** a ser utilizado é o ***multpart/form−data***, ou seja, arquivos estarão transitando no envio.

Seguindo nas nossas lembranças, listaremos abaixo uma lista de objetos dos formulários e algumas de suas propriedades:

```
.    <!−−
.        Este objeto é uma caixa de texto, com name igual a nome e valor "Marcello".
```

```
.        A propriedade maxlength indica quantos caracteres podem ser digitados na caixa.
.    -->
.    <input type="text" name="nome" value="Marcello" maxlength="20">
.
.    <!--
.        O objeto password, é uma caixa de texto onde os caracteres inseridos são mascarados,
.        já que se trata de um objeto onde são digitados senhas ou textos ocultos. Possui as
.        mesmas propriedades da caixa de texto simples.
.    -->
.    <input type="password" name="senha" maxlength="8">
.
.    <!--
.        A textarea é também uma caixa de texto, porém possui características que lhe permitem
.        ter várias linhas escritas como conteúdo. As propriedades cols e rows definem respectiva
.        número de caracteres por linha e número de linhas disponíveis até que se crie uma barra
.        de rolagem do texto.
.    -->
.    <textarea cols="50" rows="5" name="comentarios">Valor inicial(opcional)</textarea>
.
.    <!--
.        Se quisermos dar opções de escolha única, ao usuário, utilizamos o objeto radio. Observe
.        que os dois botões de rádio possuem o mesmo nome. Isso impede que o usuário escolha mais
.        A propriedade checked do primeiro botão indica que o seu valor default é Masculino.
.    -->
.    <input type="radio" name="sexo" value="Masculino" checked>
.    <input type="radio" name="sexo" value="Feminino">
.
.    <!--
.         Se queremos dar uma opção de escolha, podemos utilizar o objeto checkbox. Este objeto
.         oferece ao usuário uma opção que é marcada de acordo com o seu interesse.
.    -->
.    <input type="checkbox" name="newsletter" value="sim" checked>
.
.    <!--
.         Ainda no assunto opções de escolha, temos o objeto select. Muito parecido com os botões
.         rádio no quesito funcionalidade. Dá ao usuário o direito de escolha de somente uma opçã
.    -->
.    <select name="escolaridade">
.        <option value="Superior">Superior</option>
.        <option value="2º Grau">2º Grau</option>
.        <option value="1º Grau">1º Grau</option>
.    </select>
.
.    <!--
.         Caso se deseje que o usuário possa escolher mais de uma opção, acrescentamos a
.         palavra chave multiple à tag select. Observem:
.    --->
.    <select name="cursos" multiple>
.        <option value="HTML">HTML</option>
.        <option value="PHP">PHP</option>
.        <option value="Javascript">Javascript</option>
.    </select>
.
.    <!--
.        Se queremos que o usuário envie um arquivo para o servidor, precisamos dar a ele acesso
.        para que escolha em sua máquina, o arquivo desejado. Utilizamos o objeto file da seguint
.    -->
.    <input type="file" name="arquivo">
.
.    <!--
.        Se a intenção é fazer com que uma imagem atue como um botão, podemos usar o objeto image
.        A propriedade src informa qual é o endereço da imagem. Width e height, a largura e a alt
.        respectivamente.
.    -->
.    <input type="image" src="botao.gif" width="100" height="40">
.
.    <!--
```

```
.        O botão submit é um button que tem a ação pré-programada de enviar os dados do form para
.        o arquivo especificado na propriedade action da tag form. O valor informado na proprieda
.        value, será o label do botão.
.    -->
.    <input type="submit" name="enviar" value="Enviar">
.
.    <!--
.        O botão reset é outro button com ação definida: limpar o conteúdo de todos os campos do
.        formulário.
.    -->
.    <input type="reset" name="limpar" value="Limpar">
.
.    <!--
.        Finalmente, temos um botão sem ação nenhuma. Utilizamos este botão, quando queremos
.        que ele execute alguma rotina por nós definida. Neste caso, utilizamos a própria linguag
.        Javascript para programar este botão.
.    -->
.    <input type="button" name="calcular" value="Calcular">
```

Listamos acima os objetos que um formulário pode conter. Vejamos agora, como trabalhar com eles dentro do Javascript, como declarar e obter informações dos mesmos.

5.2 O array forms

De acordo com a hierarquia de objetos, o objeto ***window*** é o objeto ***pai*** de todos os outros. Sendo assim, podemos dizer que o objeto ***forms*** pertence a ele. Esta informação é, em parte, correta. Na verdade, o objeto ***forms*** pertence imediatamente ao objeto ***document***. Este sim, por sua vez, está diretamente ligado ao objeto ***window***. Então, baseado nestas informações, a forma de se trabalhar com este objeto seria:

window.document.forms ou document.forms

Lembrem-se de que, sendo o ***window***, objeto principal, não há a necessidade de mencioná-lo. Dessa forma, adotaremos neste material a segunda forma de acesso.

Estamos falando de um objeto ou um array? Um array em Javascript também é um objeto. Quando dizemos que o Javascript interpreta todos os elementos HTML como objetos, é comum termos vários objetos de um mesmo tipo. Isso implica que, implicitamente, temos um array de vários objetos distintos, como por exemplo, um array de formulários − ***forms***. Veremos que, existem muitos outros arrays de elementos, que trataremos com mais detalhes nas páginas seguintes.

Imaginem a seguinte página. Ela é composta de dois formulários, ou melhor, possui duas tags ***form*** no seu conteúdo.

```
01. <html>
02.     <head>
03.         <title>Formulários</title>
04.     </head>
05.     <body>
06.         <form name="f1" method="get">
07.             <h3>Formulário 1 - Login</h3><hr>
08.             Usuário:<br>
09.             <input type="text" name="user"><br>
10.             Senha:<br>
11.             <input type="password" name="senha" maxlength="10"><br>
12.             <input type="submit" name="enviar" value="Login">
13.         </form>
14.         <br>
15.         <form name="f2" method="get">
16.             <h3>Formulário 2 - Esqueceu sua senha?</h3><hr>
```

```
17.             Preencha o campo abaixo e a senha será enviada para você:<br>
18.             <input type="text" name="email" value="Digite aqui seu email"><br>
19.             <input type="submit" name="enviar" value="Enviar">
20.         </form>
21.     </body>
22. </html>
```

A página deverá se parecer com a imagem abaixo:

Agora vamos entender os elementos Javascript e sua relação com o HTML. A princípio temos dois formulários em uma mesma página. Poderíamos ter quantos quiser, por isso a necessidade de se utilizar arrays para manipular estes objetos. Como faríamos para acessar estes objetos, se por ventura, esquecêssemos de atribuir algum valor a propriedade ***name*** destes objetos? Através dos índices dos arrays. Observem:

```
//Primeiro utilizaremos os nomes dos formulários para acessá-los.
document.f1 //primeiro formulário
document.f2 //segundo formulário

//Agora, usando os índices do array de formulários.
document.forms[0] //primeiro formulário
document.forms[1] //segundo formulário

//Número de formulários da minha página.
document.forms.length //Propriedade "length" do objeto forms

/*
Algumas considerações:
1) A hierarquia dos objetos deve ser respeitada.
2) Arrays em Javascript tem índice inicial zero.
*/
```

Muito bem. Já sabemos como podemos trabalhar com formulários em Javascript. No entanto, é necessário saber também, como interagir com os seus elementos. As caixas de texto e botões não pertencem ao array ***forms***. Para estes objetos, utilizamos um outro array: o ***elements***. O array ***elements*** é um objeto que pertence ao formulário. Veja como podemos acessar os elementos deos formulários acima:

```
//Primeiro utilizaremos os nomes dos formulários e seus elementos.
document.f1.user //campo usuário do primeiro formulário
document.f1.senha //campo senha do primeiro formulário
document.f1.enviar //botão Login do primeiro formulário

document.f2.email //campo email do segundo formulário
document.f2.enviar //botão Enviar do segundo formulário

//Agora, usando os índices do array de formulários e array de elementos.
document.forms[0].elements[0] //campo usuário do primeiro formulário
document.forms[0].elements[1] //campo senha do primeiro formulário
document.forms[0].elements[2] //botão Login do primeiro formulário

document.forms[1].elements[0] //campo email do segundo formulário
document.forms[1].elements[1] //botão Enviar do segundo formulário

//Número de elementos dos formulários
document.forms[0].elements.length //Quantidade de elementos do primeiro formulário
document.forms[1].elements.length //Quantidade de elementos do segundo formulário
```

Cada uma das propriedades dos elementos HTML, tais como name, value, src e tantas outras, serão acessadas da mesma forma proposta no início do capítulo de objetos: objeto.propriedade. No momento em que temos ***document.f1.user***, basta que informemos qual propriedade ou método, queremos utilizar neste objeto. Esta

string ***document.f1.user*** é a representação codificada da minha caixa de texto ***Usuário*** do meu primeiro formulário. Traduzindo: o campo ***user*** pertence ao formulário ***f1***, que por sua vez, pertence à minha página, ***document***. Reconstruindo o exemplo, queremos obter alguns valores desta página:

```
01. <html>
02.     <head>
03.         <title>Formulários</title>
04.     </head>
05.     <body>
06.         <form name="f1" method="get">
07.             <h3>Formulário 1 − Login</h3><hr>
08.             Usuário:<br>
09.             <input type="text" name="user"><br>
10.             Senha:<br>
11.             <input type="password" name="senha" maxlength="10"><br>
12.             <input type="submit" name="enviar" value="Login">
13.         </form>
14.         <br>
15.         <form name="f2" method="get">
16.             <h3>Formulário 2 − Esqueceu sua senha?</h3><hr>
17.             Preencha o campo abaixo e a senha será enviada para você:<br>
18.             <input type="text" name="email" value="Digite aqui seu email"><br>
19.             <input type="submit" name="enviar" value="Enviar">
20.         </form>
21.         <!−−
22.             Neste ponto, vamos adicionar o código Javascript. Faremos isso aqui, porque,
23.             na tag <head> ainda não teríamos o código dos formulários executado. Assim,
24.             o Javascript não conseguiria devolver as informações solicitadas.
25.         −−>
26.         <script language="Javascript">
27.             document.write("Número de formulários: ", document.forms.length, "<br>");
28.             document.write("Nome do 1º formulário: ", document.forms[0].name, "<br>");
29.             document.write("Valor do 1º campo do 2º formulário: ", document.forms[1].elements
30.         </script>
31.     </body>
32. </html>
```

O método write aceita várias seqüências de impressões ao mesmo tempo. Separe por vírgulas, o conteúdo de cada impressão desejada.
Procure acessar os objetos pelo nome. Isto facilitará a manutenção e a legibilidade do código.
O código Javascript pode ser inserido em qualquer parte da página, embora seja recomendado centralizá−lo na tag ***head***, todas as funções e códigos comuns à página.

Embora tenhamos conseguido obter as propriedades destes objetos dos formulários, ainda não conseguimos interagir com eles. Por exemplo: não conseguimos verificar a consistência dos dados de uma caixa de texto, pelo menos neste momento de aprendizado. Toda esta interação, pode ser feita com o auxílio de mais um benefício da linguagem Javascript: ***os eventos***. É sobre este assunto que vamos falar nas próximas páginas. Com certeza, destinaremos grande parte do nosso tempo a estes artifícios.

5.3 Resumo do capítulo

- Formulários são os principais meios de comunicação entre a aplicação e o usuário.
- Para utilizá−los, é necessário especificar basicamente, o nome do script de destino das informações(action), forma como as informações serão enviadas(method) e tipo de dados que estarão transitando(enctype).
- A propriedade ***name*** é muito importante e não deve ser esquecida. Com ela obtemos fácil acesso aos objetos.
- O objeto Javascript que representa os fomulários é o ***forms***. Trata−se de uma coleção de objetos do tipo formulário.

- Arrays em Javascript, tem como primeiro índice, o "zero".
- O objeto ***forms*** pertence ao objeto ***document***.
- Elementos como caixas de texto, botões, radio e checkbox pertencem ao objeto ***elements***, que é uma coleção de elementos de formulários.
- O objeto ***elements*** pertence ao objeto ***forms***.

5.4 Exercícios de fixação

1. Utilize o exemplo proposto para exibir os nomes de cada elemento dos formulários, bem como, valores declarados, quando houver. Mostre quantos elementos cada formulário possui.
2. Escreva uma linha de comando para atribuir o valor "João" ao campo Usuário, utilizando o mesmo exemplo do exercício 1.
3. O que a instrução abaixo faz:

```
document.write( document.forms[1].elements[2].value );

a (   ) Imprime o nome do objeto elements
b (   ) Imprime o valor do segundo elemento do primeiro formulário
c (   ) Imprime o nome do segundo elemento do primeiro formulário
d (   ) Imprime o valor do terceiro elemento do segundo formulário
e (   ) Imprime quantos elementos há no primeiro formulário
```

6 Eventos

Os eventos constituem−se de ações que alguns objetos sofrem, ou interações provocadas pelo usuário. Um clique de mouse, abrir e fechar janelas, são ***ações*** que podemos monitorar e principalmente, reagir da melhor forma a satisfazer as necessidades da nossa aplicação.

Imaginem que, nossa aplicação possa executar, a cada clique do mouse em um determinado botão, uma rotina escrita especialmente para este fim. Ou, quando o usuário tentar enviar os seus dados através de um formulário, possamos verificar a consistência dos valores digitados por ele. Tudo isso pode acontecer se identificarmos as ações que estão acontecendo.

Iremos trabalhar com eventos em nossas aplicações. A lista abaixo, informa alguns dos eventos disponíveis para Javascript, como acontecem e sobre quais objetos eles têm influência. Sua nomenclatura segue os padrões utilizados pelo Javascript: a partir da segunda palavra, as iniciais de cada palavra serão ***maiúsculas***.

6.1 Alguns eventos

- ***onAbort***: quando o usuário abortar o carregamento da página e seus elementos, estará gerando este evento.
- ***onBlur***: este evento ocorre quando um objeto perde o foco. Afetam objetos do tipo texto, botões, páginas e frames.
- ***onChange***: este evento acontecerá, sempre que o valor de um campo for alterado. Pode ser aplicado em objetos do tipo text, textarea, select.
- ***onClick***: é o evento do clique do mouse. Sempre que este for clicado, o evento onClick é disparado. É possível utilizá−lo em toda a página e seus elementos.
- ***onDragDrop***: acontece quando um objeto é arrastado pelo usuário para dentro da janela do navegador.
- ***onError***: se a aplicação gerar algum erro, é este evento que estará acontendo.
- ***onFocus***: é o oposto do evento onBlur. Sempre que um objeto receber o foco da aplicação, este evento é disparado. Atinge os mesmos elementos do onBlur.
- ***onKeyPress***: acontece quando uma tecla é pressionada.
- ***onKeyDown***: acontece enquanto a tecla continuar pressionada.
- ***onKeyUp***: acontece quando a tecla é liberada.
- ***onLoad***: sempre que uma página for carregada, este evento é executado.
- ***onMouseDown***: é disparado quando o botão do mouse é pressionado.
- ***onMouseUp***: é o evento da liberação do botão do mouse, após ele ter sido pressionado.
- ***onMouseMove***: acontece quando o mouse está se movimentando.

- ***onMouseOver***: acontece quando o mouse está sobre um determinado objeto.
- ***onMouseOut***: acontece quando o mouse deixa de estar sobre um determinado objeto.
- ***onMove***: é quando a página está sendo movimentada pela tela.
- ***onReset***: é disparado quando o botão ***reset*** é clicado.
- ***onResize***: acontece quando a janela está sendo redimensionada.
- ***onSelect***: quando um texto, ou parte dele, for selecionada, este evendo é gerado.
- ***onSubmit***: como o próprio nome está dizendo, este evento será disparado quando o botão ***submit*** for clicado, e conseqüêntemente, o formulário estiver sendo enviado. É um evento exclusivo da tag ***form***.
- ***onUnload***: quando uma nova página é acessada, a página corrente é descarregada. Neste momento o evento onUnload foi acionado. Acontece também quando a página é atualizada ou quando a janela do navegador é fechada.

Todos os eventos possuem o prefixo ***on***.
Os eventos sempre existirão dentro do objeto que estiver sendo atingido.

Acima estão listados os eventos mais comuns. Como podemos ver, são vários, e às vezes, muito seletores. Nas páginas seguintes, vamos trabalhar com exemplos práticos do dia−a−dia, atividades que freqüentemente necessitam da utilização dos eventos.

6.2 onBlur

Este evento acontece quando um objeto perde o foco. Parece bastante interessante, que, uma validação de campo possa ser feita neste momento. Imaginem que, se o campo não estiver de acordo com o objetivo da aplicação, uma mensagem possa ser enviada para o usuário, alertando−o da inconsistência. Bom, nosso exemplo fará basicamente o que foi falado acima, porém, o fará no exato momento que o objeto a ser tratado, perder o foco.

O exemplo abaixo solicita para o usuario preencher os campos ***usuário*** e ***senha***. A aplicação determina que a senha não poderá ter menos do que 6 caracteres. Se o usuário tentar deixar o campo senha, com menos do que 6 caracteres preenchidos, uma mensagem deverá alertá−lo, e o foco do campo, ser reestabelecido para a correção. Observem:

```
01. <html>
02.     <head>
03.         <title>onBlur</title>
04.         <script language="Javascript">
05.             function checaTamanho() {
        06.                 if( document.f1.senha.value.length < 6 ) {
                07.                     alert('*** Senha deve possuir no mínimo 6 caracteres.');
                08.                     document.f1.senha.focus();//remete o foco para o campo se
                09.                 }
        10.             }
11.         </script>
12.     </head>
13.     <body>
14.         <form name="f1" method="get">
15.             <h3>Formulário de Login</h3><hr>
16.             Usuário:<br>
17.             <input type="text" name="user"><br>
```

```
18.             Senha:<br>
19.             <input type="password" name="senha" maxlength="10" onBlur="checaTamanho();">
20.             (mínimo de 6 caracteres)<br>
21.             <input type="submit" name="enviar" value="Login">
22.         </form>
23.     </body>
24. </html>
```

O exemplo acima possui um formulário com dois campos: usuário e senha. Se queremos verificar o tamanho de ***senha***, concluímos que o evento ***onBlur*** deve estar contido dentro desse objeto. Sendo assim, quando acontecer a perda de foco, seja ela provocada por um ***tab*** ou ***clique fora do campo***, a função ***checaTamanho()*** será chamada.

Esta função, testa se, o tamanho(***propriedade length***) do valor(***value***) da caixa de texto ***senha***, que pertence ao formulário ***f1***, que está declarado na página(***document***), for menor do que 6, então a mensagem será enviada através do método ***alert()***. Logo após a mensagem, o foco é retornado para o campo através do evento ***focus()***, do objeto de destino.

 Percebam a hierarquia dos objetos sendo respeitada no exemplo.

Objeto this

Existe uma forma de se evitar, em certos casos, a referência a nomes muito longos de objetos no código. Estamos falando do objeto ***this***. Quando utilizado, ele substitui o objeto que o contiver. Vejam o exemplo acima modificado:

```
01. <html>
02.     <head>
03.         <title>onBlur</title>
04.         <script language="Javascript">
05.             function checaTamanho( valor ) {
        06.                 if( valor.length < 6 ) {
                07.                     alert('*** Senha deve possuir no mínimo 6 caracteres.');
                08.                     document.f1.senha.focus();//remete o foco para o campo se
                09.                 }
        10.             }
11.         </script>
12.     </head>
13.     <body>
14.         <form name="f1" method="get">
15.             <h3>Formulário de Login</h3><hr>
16.             Usuário:<br>
17.             <input type="text" name="user"><br>
18.             Senha:<br>
19.             <input type="password" name="senha" maxlength="10" onBlur="checaTamanho( this.val
20.             (mínimo de 6 caracteres)<br>
21.             <input type="submit" name="enviar" value="Login">
22.         </form>
23.     </body>
24. </html>
```

Observem que, ao chamarmos a função ***checaTamanho()***, estamos passando o parâmetro ***this.value***. Ao fazermos isso, queremos dizer que, estamos enviando para a função, o valor(***value***) de ***this***. Sabemos que ***this*** substitui o objeto que o contiver. Sendo assim, estamos dizendo que passamos para a função o valor do campo senha.

Por sua vez, a função ***checaTamanho()*** foi alterada para receber o valor do campo senha, através da variável de função ***valor***. Novamente, verifica−se o tamanho de ***valor*** com a propriedade ***length***, e segue−se o

procedimento normal de acordo com o resultado.

Usar ***this*** pode facilitar a escrita e a legibilidade do código.

6.3 onChange

O evento ***onChange*** será disparado sempre que houver uma alteração de valor no objeto em que ele for declarado. Nosso próximo exemplo, irá monitorar uma caixa de seleção, e de acordo com o valor escolhido, a página terá sua cor de fundo alterada.

```
01. <html>
02.     <head>
03.         <title>onChange</title>
04.     </head>
05.     <body>
06.         <h3>O evento <b>onChange</b></h3><hr>
07.         Escolha uma cor de fundo para sua página:<br>
08.         <select name="cores" onChange="document.bgColor = this.value;">
09.             <option selected value="white">Escolha sua cor</option>
10.             <option value="red">Vermelha</option>
11.             <option value="green">Verde</option>
12.             <option value="blue">Azul</option>
13.             <option value="yellow">Amarela</option>
14.         </select>
15.     </body>
16. </html>
```

Observem que o evento ***onChange*** está dentro da tag ***select***. Cada vez que uma cor é escolhida, o valor da ***select*** é mudado. Neste momento, a propriedade ***bgColor*** do objeto ***document***, recebe o valor escolhido e executa a operação.

6.4 onClick

O evento onClick é bem intuitivo, e irá acontecer sempre que um objeto receber o clique do mouse. Basicamente, todos os objetos estão sujeitos a ação deste evento. Imagens, botões e o próprio ***document*** são exemplos comuns. A seguir, vamos reproduzir o exemplo visto acima, no qual, utilizamos o evento ***onChange***. Vamos modificá−lo para trabalhar da mesma forma, mas agora, operando com o evento ***onClick***.

```
01. <html>
02.     <head>
03.         <title>onClick</title>
04.     </head>
05.     <body>
06.         <h3>O evento <b>onClick</b></h3><hr>
07.         Escolha uma cor de fundo para sua página:<br>
08.         <input type="button" name="cor1" value="Vermelha" onClick="document.bgColor = '#FF000
09.         <input type="button" name="cor2" value="Verde" onClick="document.bgColor = '#00FF00';
10.         <input type="button" name="cor3" value="Azul" onClick="document.bgColor = '#0000FF';"
11.         <input type="button" name="cor4" value="Preta" onClick="document.bgColor = '#000000';
12.     </body>
13. </html>
```

Este exemplo, utilizou objetos do tipo ***button*** para disparar os eventos. A cada clique em um dos botões, a cor correspondente é atribuída à página.
Botões do tipo ***button*** pode ser programados através de Javascript. Os botões ***submit*** e ***reset*** já possuem tarefas específicas.

6.5 onFocus

Este evento é o oposto do evento ***onBlur***, ou seja, acontecerá toda vez que um objeto receber o foco da aplicação. Para exemplificar este evento, o código abaixo fará uma operação matemática, definida pelo usuário. Teremos 3 caixas de texto, onde serão digitados os operandos e operdor da aplicação. A última caixa de texto será o resultado.

Para testar, certifique−se de que esteja inserindo apenas números e sinais matemáticos válidos. Ainda não aprendemos como tratar númericos e alfabéticos.

```
01. <html>
02.     <head>
03.         <title>onFocus</title>
04.         <script language="Javascript">
05.             function operacao() {
        06.                 var expressao = document.f1.num1.value +
        07.                                 document.f1.operador.value +
        08.                                 document.f1.num2.value;
        09.                 return expressao;
        10.             }
11.
12.             function mostraResultado() {
        13.                 /*
        14.                     Uma vez que temos caixas de texto recebendo os dados, estes serão
        15.                     considerados variáveis de texto − Strings. Para que a operação ma
        16.                     tenha sucesso, será preciso converter os dados. O método eval() a
        17.                     a expressão contida em seus parâmetros, realizando todas as conve
        18.                 */
        19.                 document.f1.resultado.value = eval( operacao() );
        20.             }
21.         </script>
22.     </head>
23.     <body>
24.         <h3>O evento <b>onFocus</b></h3></hr>
25.           Digite apenas números e sinais matemáticos válidos:
26.             <form name="f1">
27.                 − Informe o primeiro número<br>
28.                 <input type="text" name="num1" size="10" onFocus="document.f1.reset();"><br>
29.                 − Informe o operador matemático<br>
30.                 <input type="text" name="operador" size="4"><br>
31.                 − Informe o segundo número<br>
32.                 <input type="text" name="num2" size="10"><br>
33.                 − Resultado<br>
34.                 <input type="text" name="resultado" size="10" onFocus="mostraResultado();">
35.             </form>
36.     </body>
37. </html>
```

O exemplo acima, mostra o uso do evento ***onFocus*** em duas ocasiões. Na primeira caixa de texto, temos o evento ***onFocus*** executando a instrução: ***document.f1.reset()***. Significa que, a caixa de texto ***num1***, ao receber o foco, limpará todos os campos do formulário ***f1***, tal qual um botão ***reset***.

Logo em seguida, na caixa de texto ***resultado***, o evento ***onFocus*** chama a função ***mostraResultado()***. Esta função existe apenas para avaliar os valores digitados e produzir o resultado. Na linha 19, a propriedade ***value*** do objeto resultado(document.f1.resultado), recebe o valor obtido do método ***eval***. Este método, faz uma conversão implícita dos dados e executa a operação que recebeu como parâmetro.

Este parâmetro, por sua vez, é o retorno obtido da função ***operacao()***. Esta função nada mais faz, do que concatenar todos os valores, formando uma expressão válida. Ex. a+b. Esta expressão é retornada para o método ***eval()***, que a chamou, e o processo então é finalizado.

O exercício poderia ser simplificado, utilizando apenas a função ***mostraResultado()***. O uso das duas funções, se justificou apenas para fins didáticos. Observem que, temos um método, recebendo como parâmetro uma função. E tudo isso está dentro de outra função.

6.6 Objeto event

Este objeto é responsável por conter informações sobre os eventos gerados pelo sistema ou usuário. Por exemplo, se uma tecla é pressionada, o objeto ***event*** sabe qual tecla é. Se um botão do mouse é clicado, o ***event*** sabe qual foi.

Este mecanismo pode ser muito diferente entre browsers de famílias diferentes. O Internet Explorer usará este objeto com o nome ***event***. Já o Netscape, utilizará o nome ***eventObj***. Neste material, vamos convencionar o uso da palavra chave ***event***.

Nos exemplos abaixo usaremos o obejto ***event*** para trabalhar em conjunto com os eventos de teclado e mouse.

6.7 onKeyPress

Este evento é disparado todas as vezes que uma tecla é pressionada. É excelente para validar formatações e criar máscaras para campos. Criaremos um exemplo que não permitirá ao usuário, digitar valores não numéricos. Se ele tentar fazer isso, a tecla pressionada será ignorada e não vai aparecer na caixa de texto.

```
01. <html>
02.     <head>
03.         <title>onKeyPress</title>
04.         <script language="Javascript">
05.             function numerico() {
        06.
        07.                 /* Vamos utilizar o objeto event para identificar quais teclas estão
        08.                 if( !(event.keyCode >= 48 && event.keyCode <= 57 ) ) {
                09.
                10.                     /* A propriedade keyCode revela o código ASCII da tecla p
                11.                        os números de 0 a 9 estão compreendidos entre os valor
                12.                        Então, caso os valores não estejam(!) neste intervalo,
                13.                        para quem a chamou. */
                14.                     return false;
                15.                 }
        16.             }
17.         </script>
18.     </head>
19.     <body>
20.         <h3>O evento <b>onKeyPress</b></h3><hr>
21.         CPF:<input type="text" name="cpf" maxlength="11" onKeyPress="return numerico();">(Som
22.     </body>
23. </html>
```

Vimos que, o evento ***onKeyPress*** se localiza dentro da caixa de texto, pois é lá que os valores serão digitados. Apesar da informação ***somente números*** estar presente no formulário, existem usuários distraídos que tentarão digitar algum caracter alfabético. Então ele verá que, ao fazer isso, nada acontecerá. Fizemos esta monitoração com a propriedade ***keyCode*** do objeto ***event***. Ela armazena o código ASCII das teclas pressionadas.

6.8 onKeyUp

O evento ***onKeyUp*** é responsável por responder cada vez que uma tecla pressionada é liberada. O código de exemplo abaixo mostra uma máscara de data muito utilizada. Ele irá monitorar o pressionamento de teclas, e

quando o tamanho do texto for igual a 2 e a 5, barras de separação "/" serão automaticamente inseridas.

Tomemos a data 26/07/2004 como exemplo. Observem que, ao digitarmos 26, teremos o tamanho do texto igual a 2(número de caracteres). Neste momento concatenaremos com o valor existente, a barra separadora. Logo em seguida, digitamos os números 0 e 7. Teremos agora 5 caracteres. Novamente uma barra separadora é necessária. Finalmente, o ano 2004 é inserido.

Vamos acrescentar a função ***numerico()*** desenvolvida no exemplo sobre ***onKeyPress*** para evitar que letras sejam digitadas.

```
01. <html>
02.     <head>
03.         <title>onKeyUp</title>
04.         <script language="Javascript">
05.             function numerico() {
        06.                 if( !(event.keyCode >= 48 && event.keyCode <= 57 ) )
        07.                     return false;
        08.             }
09.
10.             function formataData() {
        11.                 //Criaremos esta variável para evitar a digitação de nomes longos
        12.                 var data = document.getElementById('data');
        13.
        14.                 if( data.value.length == 2 || data.value.length == 5 ) {
                15.                     /* Aqui vemos uma outra forma de se fazer acesso ao objet
                16.                        Procurem pela propriedade id na declaração da tag, e a
                17.                        Traduzindo: o valor do objeto, cujo id é igual a 'data
                18.                        ele mesmo +(concatenando) a barra de separação '/'
                19.                     */
                20.                     document.getElementById('data').value += "/";
                21.                 }
        22.             }
23.         </script>
24.     </head>
25.     <body>
26.         <h3>O evento <b>onKeyUp</b></h3><hr>
27.         Data: <input type="text" name="data" id="data" maxlength="10" onKeyUp="formataData();
28.                     onKeyPress="return numerico();">
29.     </body>
30. </html>
```

Se queremos formatar a data digitada, o evento ***onKeyUp*** deve estar declarado dentro do objeto text. Cada vez que uma tecla é pressionada, a função ***formataData()*** é chamada. Nesta função, resolvemos atribuir o objeto a uma variável, de nome ***data***. Ao fazermos isso evitamos digitar todo o nome do objeto ***document.getElementById('data').value.length***. Sabemos que a variável já contém a referência para este objeto. Basta então, que acessemos sua propriedade length.

O método ***getElementById()***, localiza o objeto pelo valor contido na propriedade ***id*** declarada na tag do objeto. Isso é útil quando não temos a tag ***form*** no código. Então, compara−se o tamanho do texto, e quando este tiver tamanho 2 ou 5, as barras são colocadas.

6.9 onLoad

Este evento é muito utilizado, pena que, muitas das vezes, com más intenções. Sempre que a página for carregada, o evento ***onLoad*** acontece. Muitos sites, atribuem a tarefa de ficar abrindo janelas popups à ocorrência deste evento. Isso às vezes, é bastante inconveniente.

Para demonstrar este evento, faremos uma simples aplicação. O usuário, ao carregar a página, receberá uma

mensagem de boas vindas. Observem:

```
01. <html>
02.     <head>
03.         <title>onLoad</title>
04.     </head>
05.     <body onLoad="alert('Caro usuário, seja bemvindo!');">
06.         <h3>O evento <b>onLoad</b></h3><hr>
07.     </body>
08. </html>
```

Como o evento ***onLoad*** é disparado quando a página é carregada, ele deve ser declarado dentro da tag ***body***. É esta tag que representa visualmente a página, e também a conhecemos como ***document***.

6.10 onMouseDown

O evento ***onMouseDown*** reage quando ocorre um clique do mouse. É comum o seu uso juntamente com o objeto ***event***, assim é possível identificar qual botão foi pressionado e tratá−lo conforme a necessidade. Nosso próximo exemplo, irá bloquear o menu popup que geralmente é originado do clique no botão direito de um mouse. Assim, ele estará impossibilitado de usar o botão direito.

```
01. <html>
02.     <head>
03.         <title>onMouseDown</title>
04.         <script language="Javascript">
05.             function botao() {
        06.
        07.                 /* A propriedade button do objeto event, reconhece qual botão do mous
        08.                    a seguinte lista:
        09.                        0 − nenhum botão
        10.                        1 − botão esquerdo
        11.                        2 − botão direito
        12.                        4 − botão do meio, caso seja um mouse de 3 botões
        13.                 */
        14.                 if( event.button == 2 )
        15.                     alert( '*** Esta página não permite o clique do botão direito do
        16.             }
17.
18.             /* Como o clique é dado no corpo do documento, a função deve ser atribuída à prop
19.                onmousedown do objeto document. */
20.             document.onmousedown = botao;
21.         </script>
22.     </head>
23.     <body>
24.         <h3>O evento <b>onMouseDown</b></h3><hr>
25.         Para testar este exemplo, procure clicar com o botão direito do mouse.
26.     </body>
27. </html>
```

6.11 onMouseOver e onMouseOut

Continuando com os eventos de mouse, temos pela frente o ***onMouseOver*** e o ***onMouseOut***. São eventos que acontecem quando o mouse passa sobre um objeto e quando ele sai de cima desse objeto, respectivamente. Vamos aplicar os dois ao mesmo tempo, e no mesmo objeto. Sim, utilizar eventos diferentes dentro de um mesmo objeto é possível e muito útil.

Neste exemplo, teremos duas imagens(gyn.gif e bsb.gif − clique com o botão direito do mouse e depois em ***salvar destino como***). Faremos uma aplicação de troca de banners. Quando o mouse passar sobre a imagem, outra irá aparecer. Quando o mouse sair de cima dela, a primeira retorna.

```
01. <html>
02.     <head>
03.         <title>onMouseOver e onMouseOut</title>
04.     </head>
05.     <body>
06.         <h3>Os eventos <b>onMouseOver e onMouseOut</b></h3><hr>
07.         Passe o mouse sobre a imagem:<br>
08.         <img src="gyn.gif" onMouseOver="this.src = 'bsb.gif';" onMouseOut="this.src = 'gyn.gi
07.     </body>
08. </html>
```

Quando o mouse estiver sobre a imagem, o evento ***onMouseOver*** é disparado. Então a propriedade ***src*** da imagem(representada pelo objeto ***this***) é alterada para exibir outro arquivo. Quando o mouse sair de cima da imagem, a propriedade ***src*** novamente é alterada, agora para exibir o arquivo original.

6.12 onSelect

Este evento irá acontecer, sempre que um texto ou parte dele, for selecionado. Obviamente, os objetos em que isso é possível são as caixas de texto, senha ou textarea. Vamos exemplificar isso de duas formas: a primeira será o usuário selecionar diretamente na caixa de texto. Neste momento, a cor da fonte de todo o texto será alterada. A segunda maneira, caberá a um botão selecionar todo o texto contido no objeto.

```
01. <html>
02.     <head>
03.         <title>onSelect</title>
04.     </head>
05.     <body>
06.         <h3>O evento <b>onSelect</b></h3><hr>
07.         Exemplo 1 − Digite e selecione um texto.<br>
08.         <textarea rows="5" cols="30" name="testeSelect" onSelect="this.style.color = '#FF0000
09.         <br><br>
10.         Exemplo 2 − Clique no botão abaixo para selecionar todo o texto.<br>
11.         <textarea rows="5" cols="30" name="testeSelect2" id="sel">
12.             Este texto será selecionado quando o botão "Selecionar" for clicado.
13.         </textarea>
14.         <br>
15.         <input type="button" name="selecionar" value="Selecionar" onClick="document.getElemen
16.     </body>
17. </html>
```

Algumas novidades foram inseridas no exemplo acima. A primeira, é que podemos manipular os estilos da página, utilizando Javascript, conforme a linha 08. Dissemos que, quando o texto for selecionado, a propriedade ***color*** do objeto ***style***, daquele objeto(***textarea***), deveria ser alterada.

Na segunda ***textarea*** utilizamos o evento ***onClick*** para disparar o evento ***onSelect***. Quando o botão for clicado, o objeto cujo id é 'sel', terá seu texto totalmente selecionado.

6.13 onSubmit

Uma das formas mais eficientes de se validar campos de um formulário, é utilizando o evento ***onSubmit***. Este evento é exclusivo do formulário e será gerado quando este for submetido. Neste momento, é que fazemos as validações e testes necessários, antes de enviar os dados. Caso os dados não estejam satisfatórios, o formulário

não é enviado. Observem o exemplo abaixo:

```
01. <html>
02.     <head>
03.         <title>onSubmit</title>
04.         <script language="Javascript">
05.             function checaCampos() {
        06.
        07.                 //criamos uma referência para o formulário
        08.                 obj = document.f1;
        09.
        10.                 //testamos cada um dos campos
        11.                 if( obj.nome.value == '' || obj.endereco.value == '' || obj.fone.valu
                12.                     alert( '*** Preencha todos os campos!' );
                13.                     return false;
                14.                 }
        15.             }
16.         </script>
17.     </head>
18.     <body>
19.         <h3>O evento <b>onSubmit</b></h3><hr>
20.         <form name="f1" method="get" onSubmit="return checaCampos();">
21.             Nome:<br>
22.             <input type="text" name="nome"><br>
23.             Endereço:<br>
24.             <input type="text" name="endereco"><br>
25.             Fone:<br>
26.             <input type="text" name="fone"><br><br>
27.             <input type="submit" name="enviar" value="Enviar">
28.         </form>
29.     </body>
30. </html>
```

Se algum dos três campos existentes, não for preenchido, já é condição suficiente para que o formulário não seja enviado. Observem que o evento ***onSubmit*** está dentro da tag ***form***.

6.14 onUnload

Este evento é muito útil quando queremos executar alguma rotina, antes do usuário deixar a página ou fechar a janela. Além das duas situações mencionadas, acontecerá também quando a página for recarregada. Para exemplificar, deixaremos uma mensagem de adeus, quando o usuário sair de nosso site.

```
01. <html>
02.     <head>
03.         <title>onUnload</title>
04.     </head>
05.     <body onUnload="alert( 'Volte sempre!' );">
06.         <h3>O evento <b>onUnload</b></h3><hr>
07.     </body>
08. </html>
```

⚠ A rotina de encerramento é executada antes da página ser descarregada.

Como vimos durante este capítulo, os eventos são muito importantes para que tenhamos um bom desempenho na nossa programação cliente. As possibilidades são inúmeras, só dependerá da sua criatividade.

6.15 Resumo do capítulo

- Eventos são reações da aplicação quando esta interage com o usuário ou o sistema.
- Existem eventos para tratar ações do mouse, teclado, submissão de dados, entre outros.
- Todos os eventos possuem o prefixo "on".
- Eventos devem estar contidos dentro do objeto em que são gerados.
- O objeto ***this*** é utilizado para referenciar o objeto que o contiver.
- A propriedade ***length*** representa o número de caracteres do objeto.
- O método ***eval*** avalia e executa uma expressão matemática, fazendo implicitamente as conversões de tipos necessárias.
- O objeto ***event*** contém informações geradas a partir de um evento, tais como tecla pressionada e botão do mouse que foi clicado, entre outros.
- ***event.keyCode*** armazena o código ASCII de uma tecla que foi pressionada.
- O método ***getElementById()***, obtém o objeto pelo seu ***id***, que deve estar contido dentro da declaração da tag.
- ***event.button*** armazena qual botão do mouse foi clicado.
- O método ***onUnload*** executa a rotina de encerramento antes de descarregar a página.

6.16 Exercícios de Fixação

1. Qual é o evento mais adequado para tratar as informações de um formulário, no momento de sua submissão?

- (a) onClick
- (b) onChange
- (c) onBlur
- (d) onSubmit
- (e) onReset

2. Construa uma rotina para avaliar o formulário abaixo:

```
01. <html>
02.     <head>
03.         <title>Formulário</title>
04.     </head>
05.     <body>
06.         <form name="f1">
07.             <h3>Formulário de Inscrição:</h3><hr>
08.             Nome:<br>
09.             <input type="text" name="nome"><br>
10.             CPF:<br>
11.             <input type="text" name="cpf">(somente números)<br>
12.             Data de Nascimento:<br>
13.             <input type="text" name="datanascimento">(dd-mm/yyyy)<br><br>
14.             <input type="submit" name="enviar" value="Enviar">
15.         </form>
16.     </body>
17. </html>
```

- ♦ Todos os campos devem ser preenchidos.
- ♦ O campo ***CPF*** deve permitir a entrada somente de números.
- ♦ O campo ***Data de Nascimento*** deve possuir a máscara ***dd-mm-yyyy***.

3. O que o código abaixo faz:

```
...
<input type="button" value="Clique Aqui!" onClick="this.value = 'Fui clicado!!!';">
...
```

- (a) Produz a mensagem Fui clicado!!! quando o botão for pressionado.
- (b) Chama a função Fui clicado!!! quando o botão for pressionado.
- (c) Altera o texto de exibição do botão para "Fui clicado!!!" quando o mouse passar sobre o botão.
- (d) Altera o texto de exibição do botão para "Fui clicado!!!" quando o botão for pressionado.
- (e) Nenhuma das anteriores.

4. Crie uma máscara para um campo de CPF, seguindo o padrão: ***000.000.000−00***.

5. Geralmente, o evento "onUnload" é utilizado para qual das seguintes opções?

- (a) Executar ações após o documento ser descarregado
- (b) Fechar janelas ou objetos frames
- (c) Executar ações antes do documento ser descarregado
- (d) Descarregar um documento HTML
- (e) Remover um documento do cache do browser

7 Objetos II − String, Math e Date

O Javascript possui objetos que não estão relacionados com os elementos HTML que ele utiliza. São objetos prontos e independentes, que auxiliam na construção de aplicações mais completas, com maiores recursos. Estes objetos são chamados de ***Objetos Core***. Neste capítulo, vamos estudar os objetos String, Math e Date.

Estes objetos irão nos auxiliar na manipulação de variáveis e também de outros objetos, fornecendo métodos para trabalhar com strings, métodos matemáticos e para utilização de datas.

7.1 Objeto String

O objeto ***String***, é constituído de uma seqüência de 0 ou mais caracteres, e vai nos fornecer métodos, para que possamos utilizar da forma mais adequada, os valores assumidos. Além de vários métodos de formatação HTML, veremos alguns muito úteis, tais como: separação de strings por padrão, código ASCII, conversões e substrings. Compreender e saber trabalhar com ***String***, é fundamental para um bom desempenho da aplicação.

Existem duas formas de se declarar o uso de uma string:

```
var str = "Isto é uma string";//forma mais simples
var str = new String("Isto é uma string");//utilizando um construtor

/**
A partir desta declaração, a variável str, torna−se uma referência em memória para um objeto do
tipo string. Todos os métodos e propriedades do objeto String estarão disponiveis para str.
**/
```

- Alguns métodos de formatação HTML

1. 1. big() − retorna uma cópia da string formatada com a tag ***big***
   2. blink() − retorna uma string com a formatação da tag ***blink***
   3. bold() − retorna uma string envolta pela tag ***b***
   4. fixed() − utiliza a tag ***tt*** para formatar a string
   5. fontColor() − utiliza o atributo ***color*** da tag ***font***
   6. fontSize() − utiliza o atributo ***size*** da tag ***font***
   7. italics() − transforma a string em itálico, equivalente à tag ***i***
   8. small() − retorna uma string formatada com a tag ***small***
   9. sub() − formata a string utilizando a tag ***sub***
   10. sup() − formata a string utilizando a tag ***sup***

Vejam o exemplo abaixo. Criaremos uma string e vamos tentar aplicar alguns dos métodos de formatação vistos acima.

```
01. <html>
02.     <head>
03.         <title>Trabalhando com strings</title>
04.         <script language="Javascript">
05.             var str = new String("Javascript");
06.             var num = '2';
07.
08.             /* Vamos imprimir na tela várias nuances obtidas através dos métodos de
09.                formatação que aprendemos.
10.             */
11.             document.write( str.big(), '<br>' );
12.             document.write( str.bold(), '<br>' );
13.             document.write( str.fontcolor('#FF0000'), '<br>' );// informe a cor desejada
```

```
14.             document.write( str.fontsize(16), '<br>' );// informe o tamanho da fonte
15.             document.write( str.italics(), '<br>' );
16.             document.write( 'Nota de rodapé' + num.sub(), '<br>' );
17.             document.write( '5' + num.sup() + ' = 25' );
18.         </script>
19.     </head>
20.     <body>
21.     </body>
22. </html>
```

Vários parâmetros são permitidos dentro do método ***write()***, desde que sejam separados por ",".

- Outros métodos para HTML

1. 1. anchor()
   2. link()

O exemplo abaixo mostra o uso destes dois métodos. O ***anchor()*** cria uma âncora para um link, e o ***link()*** cria um link, propriamente dito.

```
01. <html>
02.     <head>
03.         <title>Trabalhando com strings</title>
04.         <script language="Javascript">
05.             var str = "Fim da Página";
06.             document.write( str.link( '#FIM' ) );// Cria link para âncora FIM
07.
08.             // As quebras de linha existirão apenas para fazer a página crescer verticalmente
09.             document.write('<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><
10.                             <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><
11.                             <br><br><br><br><br><br><br><br><br><br><br>');
12.             document.write( str.anchor( 'FIM' ) );// Cria a âncora FIM
13.         </script>
14.     </head>
15.     <body>
16.     </body>
17. </html>
```

- Maiúsculas e minúsculas

1. 1. toUpperCase()
   2. toLowerCase()

Os métodos acima transformam a string em maiúsculas(***toUpperCase()***) ou minúsculas(***toLowerCase()***):

```
01. <html>
02.     <head>
03.         <title>Trabalhando com strings</title>
04.         <script language="Javascript">
05.             var str = "Curso de Javascript";
06.             document.write( str.toUpperCase(), '<br>' );// maiúsculas
07.             document.write( str.toLowerCase() );// minúsculas
08.         </script>
09.     </head>
10.     <body>
11.     </body>
12. </html>
```

- Substrings

1. 1. charAt()
   2. charCodeAt()
   3. fromCharCode()
   4. indexOf()
   5. lastIndexOf()
   6. slice()
   7. split()
   8. substr()
   9. substring()

Estes métodos são responsáveis por tratar a string, ou pedaços dela, de várias formas. É possível obter sua codificação, procurar pedaços de uma string dentro de outra e até recriar uma string, através de seu código ASCII. Vejamos alguns exemplos de uso desses métodos, que podem ser considerados os mais importantes deste tópico:

- Exemplo 1 − charAt(), charCodeAt() e fromCharCode()

```
01. <html>
02.     <head>
03.         <title>Trabalhando com strings</title>
04.         <script language="Javascript">
05.             var str = "Curso de Javascript";
06.
07.             document.write( str, '<br>' );
08.
09.             // charAt() retorna o caracter que ocupa a posição informada como parâmetro.
10.             document.write( str.charAt(9), '<br>' );// irá retornar "J"
11.
12.             // charCodeAt() retorna o código ASCII do caracter que ocupa a posição informada
13.             document.write( str.charCodeAt(9), '<br>' );// retorna 74
14.
15.             // fromCharCode() constrói uma string a partir do código ASCII informado
16.             document.write( String.fromCharCode( 74, 97, 118, 97, 115, 99, 114, 105, 112, 116
17.
18.             /* Como String é um objeto, podemos utilizar diretamente seu método estático from
19.                para originar uma nova string, conforme a linha 16. O resultado impresso será
20.                palavra "Javascript".
21.             */
22.         </script>
23.     </head>
24.     <body>
25.     </body>
26. </html>
```

Considere que uma string é um array de caracteres. Sendo assim, a posição inicial do array é a posição "0".

- Exemplo 2 − indexOf() e lastIndexOf()

```
01. <html>
02.     <head>
03.         <title>Trabalhando com strings</title>
04.         <script language="Javascript">
05.             var str = "O rato roeu a roupa do rei de Roma";
06.
07.             document.write( str, '<br>' );
08.
```

```
09.             /* indexOf() procura pela primeira ocorrência do caracter ou string informada. Se
10.                retorna −1. */
11.             document.write( str.indexOf('rei'), '<br>' );// retorna 23, posição onde ocorre "
12.
13.             /* Procura pela última ocorrência do caracter ou string informada. Retorna −1
14.                caso não encontre. */
15.             document.write( str.lastIndexOf('o') ); // retorna a posição 31
16.         </script>
17.     </head>
18.     <body>
19.     </body>
20. </html>
```

⚠ Os métodos ***indexOf()*** e ***lastIndexOf()*** reconhecem maiúsculas e minúsculas.

- Exemplo 3 − slice(), substr() e substring()

```
01. <html>
02.     <head>
03.         <title>Trabalhando com strings</title>
04.         <script language="Javascript">
05.             var str = "Curso de Javascript";
06.
07.             document.write( str, '<br>' );
08.
09.             /* slice() retorna uma substring delimitada pelas posições de início e fim inform
10.                Obs: A posição final não é incluída no retorno. */
11.             document.write( str.slice( 9, 13 ), '<br>' );// retorna "Java".
12.
13.             /* substr() retorna uma substring delimitada pela posição inicial constante no 1º
14.                comprimento, informado no segundo parâmetro. */
15.             document.write( str.substr( 9, 4 ), '<br>' );// retorna "Java".
16.
17.             /* substring() é idêntico ao método slice(). Informe as posições inicial e final,
18.                a posição final não está incluída no retorno. */
19.             document.write( str.substring( 9, 13 ) );// retorna "Java".
20.         </script>
21.     </head>
22.     <body>
23.     </body>
24. </html>
```

- Exemplo 4 − split()

```
01. <html>
02.     <head>
03.         <title>Trabalhando com strings</title>
04.         <script language="Javascript">
05.             var data = "03/08/2004";
06.
07.             document.write( data, '<br>' );
08.
09.             var arrayData = data.split('/');// divide a string onde houver "/" e atribui os p
10.
11.             //impressão dos valores obtidos
12.             document.write( 'Dia: ' + arrayData[0], '<br>' );
13.             document.write( 'Mês: ' + arrayData[1], '<br>' );
14.             document.write( 'Ano: ' + arrayData[2], '<br>' );
15.         </script>
16.     </head>
17.     <body>
```

```
18.     </body>
19. </html>
```

- Outros métodos de String

1. 1. concat()
   2. length − é uma propriedade

```
01. <html>
02.     <head>
03.         <title>Trabalhando com strings</title>
04.         <script language="Javascript">
05.             var str1 = "Sistemas";
06.             var str2 = "Abertos";
07.
08.             // concat() é utilizada para concatenar uma ou mais strings
09.             var str3 = str1.concat(str2);
10.             document.write( str3, '<br>' );
11.
12.             // length indica o número de caracteres que compõem a string
13.             document.write( 'Tamanho de str3: ' + str3.length );
14.         </script>
15.     </head>
16.     <body>
17.     </body>
18. </html>
```

7.2 Objeto Math

Este objeto, é responsável por manter métodos que irão auxiliar o uso de operações matemáticas dentro da nossa aplicação. Possui inúmeras funções pré−definidas para facilitar a programação exata. Não há construtor para este objeto, de forma que, para usar os métodos, basta invocar o objeto ***Math*** diretamente.

A seguir, visualizaremos um script que irá aplicar alguns desses métodos. Acompanhe a lista completa de métodos no capítulo de referência de métodos.

```
01. <html>
02.     <head>
03.         <title>Objeto Math</title>
04.         <script language="Javascript">
05.             var num1 = 2;
06.             var num2 = −6;
07.             var num3 = 9;
08.             var num4 = 8.4;
09.
10.             // retorna o valor absoluto de um número
11.             document.write( Math.abs( num2 ) );
12.
13.             // arredonda para cima
14.             document.write( Math.ceil( num4 ) );
15.
16.             // arredonda para baixo
17.             document.write( Math.floor( num4 ) );
18.
19.             // retorna o maior dos parâmetros informados
20.             document.write( Math.max( num1, num2, num3, num4 ) );
21.
22.             // retorna o menor dos parâmetros informados
```

```
23.             document.write( Math.min( num1, num2, num3, num4 ) );
24.
25.             // retorna o valor de PI(constante);
26.             document.write( Math.PI );
27.
28.             // eleva o primeiro valor informado, à potência representada pelo segundo valor
29.             document.write( Math.pow( num3, num1 ) );
30.
31.             /* arredonda para cima se o valor passado é x.5 ou maior, caso contrário,
32.                arredonda para baixo
33.             */
34.             document.write( Math.round( num4 ) );
35.
36.             // raiz quadrada
37.             document.write( Math.sqrt( num3 ) );
38.         </script>
39.     </head>
40.     <body>
41.     </body>
42. </html>
```

⚠ Observem que o objeto ***Math*** deve ser mencionado com a primeira letra maiúscula.

Vimos acima uma pequena lista de métodos. Outros como, seno, coseno, tangente e logaritmos estão disponíveis também.

7.3 Objeto Date

A utilização de datas em Javascript é feita através do objeto ***Date***. Este objeto instancia datas, faz conversões de horários e permite obter valores individualizados de um determinado momento.

Datas são instanciadas através do construtor ***Date()*** e, basicamente possui métodos ***get*** e ***set*** para obter ou definir valores de datas. Há ainda, alguns métodos de conversão e fuso−horários. Maiores detalhes, veja capítulo de referência de métodos.

- Instanciando datas

```
.    var hoje  = new Date();
.    var natal = new Date( 2004, 11, 25 );// os meses começam com zero, sendo Dezembro o 11º mês
```

⚠ Quando referenciar um objeto ***Date***, lembre−se de manter a 1ª letra maiúscula.

Um inconveniente quando utilizamos datas, em linguagens de programação, é o fato do retorno sempre ser em inglês ou no idioma de origem da linguagem. O exercício abaixo pode ser um exemplo de solução para este problema. A finalidade do exemplo é reproduzir o cabeçalho "Goiânia, 09 de agosto de 2004" e que ele possa ser usado permanentemente.

```
01. <html>
02.     <head>
03.         <title>Trabalhando com datas</title>
04.         <script language="Javascript">
05.             function transformaMes( mes ) {
        06.                 switch( mes ) {
                07.                     case  0: return "Janeiro"; break;
                08.                     case  1: return "Fevereiro"; break;
                09.                     case  2: return "Março"; break;
                10.                     case  3: return "Abril"; break;
                11.                     case  4: return "Maio"; break;
                12.                     case  5: return "Junho"; break;
```

```
                13.                     case  6: return "Julho"; break;
                14.                     case  7: return "Agosto"; break;
                15.                     case  8: return "Setembro"; break;
                16.                     case  9: return "Outubro"; break;
                17.                     case 10: return "Novembro"; break;
                18.                     case 11: return "Dezembro"; break;
                19.                 }
        20.             }
21.
22.             var hoje = new Date()// retorna a data atual
23.
24.             /* Para o exemplo, precisamos de 3 valores: dia, mês e ano. Dia e
25.                ano podem ser utilizados exatamente como os métodos getDate() e getFullYear
26.                retornarem. Como o mês é nominal, deverá sofrer um tratamento antes de compor
27.                a string de cabeçalho, já que o retorno da função getMonth é numérico.
28.             */
29.             var dia = hoje.getDate();// retorna o dia atual
30.             var mes = hoje.getMonth();// retorna o mês atual
31.             var ano = hoje.getFullYear();// retorna o ano atual com 4 dígitos
32.
33.             /* A função transformaMes() receberá como parâmetro, o inteiro retornado pelo
34.                método getMonth(), que será representado pela variável mês, e retornará o
35.                mês, da forma textual que conhecemos.
36.             */
37.             mes = transformaMes( mes );
38.
39.             // Produzindo a saída
40.             document.write( 'Goiânia, ' + dia + ' de ' + mes + ' de ' + ano );
41.         </script>
42.     </head>
43.     <body>
44.     </body>
45. </html>
```

O próximo exemplo calcula quantos dias se passaram desde o seu nascimento.

```
01. <html>
02.     <head>
03.         <title>Trabalhando com datas</title>
04.         <script language="Javascript">
05.             // Data atual
06.             var hoje       = new Date();
07.
08.             // Data de Nascimento
09.             var nascimento = new Date( 1977, 4, 11 );
10.
11.             // Descobrindo o número de milissegundos de um dia
12.             var miliDia    = 24 * 60 * 60 * 1000;
13.
14.             // Calculando a diferença de dias
15.             var diff       = ( hoje − nascimento ) / miliDia;
16.             diff           = Math.round( diff );
17.
18.             document.write( 'Você já viveu ' + diff + ' dias!' );
19.         </script>
20.     </head>
21.     <body>
22.     </body>
23. </html>
```

Como os valores retornados pelo objeto ***Date*** estão em milissegundos, torna−se necessário que haja uma conversão para obtermos os valores na unidade desejada. No exemplo, esta unidade foi o ***dia***. Calculamos

quantos milissegundos tem um dia. Se quiséssemos um ***mês***, multiplicaríamos o valor de um ***dia*** por ***30*** (mês genérico).
Utilizamos o método ***round()*** para truncar o número de dias, uma vez que provavelmente estará com casas decimais.

Neste próximo exemplo, queremos criar um script que nos informe quantos dias faltam para o Natal, e queremos que ele funcione corretamente pelos anos seguintes. Observem que a data desejada tem o ano corrente, mas o script irá atualizá−la para que todos os anos sejam contemplados, claro, no seu devido tempo.

```
01. <html>
02.     <head>
03.         <title>Trabalhando com datas</title>
04.         <script language="Javascript">
05.            var hoje  = new Date();
06.            var natal = new Date( 2004, 11, 25 );
07.
08.            // define o ano para obter sempre o ano corrente
09.            natal.setFullYear( hoje.getFullYear() );
10.
11.            // calcula−se o número de milissegundos de um dia
12.            miliDia   = 24 * 60 * 60 * 1000;
13.
14.            // calculando a diferença de dias
15.            diff      = ( natal − hoje ) / miliDia;
16.            diff      = Math.round( diff );
17.
18.            // imprimindo o valor final
19.            document.write( 'Faltam ' + diff + ' dias para o natal!' );
19.         </script>
20.     </head>
21.     <body>
22.     </body>
23. </html>
```

Ao utilizarmos o método ***setFullYear()*** , estamos definindo que a data ***natal*** terá seu valor ***ano*** alterado para o valor ***ano*** da data ***hoje***. Como hoje é sempre a data atual, então teremos o script funcionando para todos os anos.

7.4 Resumo do capítulo

- O Javascript possui objetos que auxiliam na elaboração do código. Funções de datas, matemáticas e string estão prontas para serem utilizadas.
- ***String*** é um objeto que representa uma seqüência de 0 ou mais caracteres. Pode ser criadas através do construtor ***String()*** ou pela simples atribuição de valores contidos entre aspas.
- ***String*** possui métodos para formatação textual, criação de HTML e manipulação de caracteres.
- O objeto ***Math*** possui métodos de operações matemáticas. Todos os métodos de ***Math*** são estáticos, devendo ser acessados diretamente dele.
- ***Date*** é o objeto de manipulação de datas. Possui em sua maioria métodos de obtenção e definição de valores de data, além de algumas conversões de tempo e tipo.
- Pode ser instanciado através do construtor ***Date()***.

7.5 Exercícios de fixação

1. Retire da string abaixo, todos os nomes de animais e imprima um valor(cada animal) por linha. Utilize os métodos de ***String*** para conseguir os resultados.

```
01.     var str = "Cachorro, gato, rato, leão, girafa, veado";
```

2. Construa a string "CURSO DE JAVASCRIPT", com partes extraídas de outras strings, conforme a lista abaixo:

```
01.     var str1 = "CEF abre concurso para Advogados.";
02.     var str2 = "Abel pede demissão do Flamengo.";
03.     var str3 = "Bancos utilizam Java em operações financeiras online.";
04.     var str4 = "Este script deve ser concluído em 5 minutos!!!";
```

3. Qual o resultado da expressão abaixo:

```
.       Math.round( Math.ceil( 3.8 ) * Math.abs( ( math.PI − 10 ) ) );
```

- (a) −27
- (b) 7.2826
- (c) Acontecerá um erro
- (d) −7.28
- (e) 27

4. Faça um script para calcular quantos meses se passaram desde o dia em que você nasceu.

8 Objetos III − Location, Navigator, Window e History

O Javascript, pelo fato de ser interpretado no navegador da máquina cliente, permite que algumas propriedades da máquina e do ambiente de execução da aplicação, sejam acessadas livremente, ou que, alguns métodos possam agir como funções deste ambiente. Nada que interfira na performance e tão pouco na segurança, embora cookies possam ser enviados.

Discutiremos neste capítulo, propriedades da aplicação, como scripts que estão executando, servidor, urls. Veremos propriedades do navegador, podendo identificar suas características e funcionamento. Trabalharemos com o objeto ***window***: aprenderemos a abrir e fechar janelas, acessar objetos de janelas diferentes, descobrir características como a resolução da máquina cliente. Finalmente, poderemos acessar as páginas do histórico de navegação e ainda promover acesso direto a páginas que estiverem armazenadas neste objeto.

8.1 Objeto Location

Este objeto armazena informações sobre a ***URL***, servidor da página corrente, sub−diretórios, protocolos e porta de comunicação que está sendo utilizada. Como estas propriedades se referem a página que está carregada, deduzimos que o objeto ***Location*** pertence ao objeto ***Document***. Antes de prosseguirmos, vamos entender o que é uma ***URL***. Para isso observem o seguinte endereço:

http://www.sistemasabertos.com.br/cursos/inscricao.php?id=25&valor=500

As seguintes partes compõem ou podem compor uma URL:

- http:// − é o protocolo de transferência de hipertexto
- www.sistemasabertos.com.br – é o nome que representa o computador hospedeiro da página
- /cursos – sub−diretório do website
- /inscrição.php – página corrente
- ?id=25&valor=500 – parâmetros que a página inscrição.php irá receber. O símbolo "?" significa que parâmetro serão passados via URL. A string "id" representa o primeiro parâmetro e ele tem valor "=" a 25. O símbolo "&" representa que parâmetros adicionais serão enviados. A string "valor" é o segundo parâmetro e tem o valor "500". Utilize o "&" para cada parâmetro adicional.

Algumas das propriedades do objeto location retornam esses valores:

- hostname – retorna o nome do servidor da página corrente. Utilizando o exemplo acima, teríamos o valor: www.sistemasabertos.com.br.
- href – é o valor que compõe toda a URL. Retorna: http://www.sistemasabertos.com.br/cursos/inscricao.php?id=25&valor=500
- pathname – contém a parte da URL que está após o "hostname". Teríamos: /cursos/inscricao.php?id=25&valor=500.
- port – retorna a porta da URL, se houver uma.
- protocol – é o protocolo que está sendo utilizado no momento. No exemplo, temos: http://, mas poderíamos ter "file", "ftp" ou "mailto".
- hash – retorna a âncora de uma URL, se esta contiver uma. Tudo após o símbolo "#" é retornado.
- host – é a composição de "hostname" mais "port".

Digitem o exemplo abaixo e o carreguem no navegador. Nem todas os valores serão mostrados, porque alguns deles dependem do que e como a página foi carregada.

```
01. <html>
```

```
02.     <head>
03.         <title>Objeto Location</title>
04.         <script language="Javascript">
05.             document.write( document.location.hostname, '<br>' );
06.             document.write( document.location.href, '<br>' );
07.             document.write( document.location.pathname, '<br>' );
08.             document.write( document.location.port, '<br>' );
09.             document.write( document.location.protocol, '<br>' );
10.             document.write( document.location.hash, '<br>' );
11.             document.write( document.location.host, '<br>' );
12.         </script>
13.     </head>
14.     <body>
15.     </body>
16. </html>
```

8.2 Objeto Navigator

Objeto Navigator

O objeto ***Navigator*** guarda informações sobre o navegador que o usuário utiliza. Versões e qual navegador utilizado são as principais propriedades.

- appCodeName – retorna o nome de código do navegador. Ambos os navegadores Internet Explorer e Netscape retornarão o mesmo valor: Mozilla.
- appName – nome do navegador. Ex: Netscape, Microsoft Internet Explorer.
- appVersion – mostra a versão do navegador.
- browserLanguage – é a linguagem default do navegador.
- cookieEnabled – verifica se o navegador pode ler ou setar cookies na máquina cliente. Retorna true ou false.
- javaEnabled() – verifica se o navegador suporta ou não Java. Retorna true ou false.
- platform − retorna o nome do sistema operacional em que o navegador está rodando. Ex. win32.

O exemplo abaixo demonstra o objeto Navigator.

```
01. <html>
02.     <head>
03.         <title>Objeto Navigator</title>
04.         <script language="Javascript">
05.             document.write( navigator.appCodeName, '<br>' );
06.             document.write( navigator.appName, '<br>' );
07.             document.write( navigator.appVersion, '<br>' );
08.             document.write( navigator.browserLanguage, '<br>' );
09.             document.write( navigator.platform, '<br>' );
10.
11.             // verificando se o navegador aceita cookies
12.             if( navigator.cookieEnabled )
13.                 document.write( 'Cookies são permitidos.', '<br>' );
14.             else
15.                 document.write( 'Cookies não são permitidos.', '<br>' );
16.
17.             // verificando se o navegador suporta Java
18.             if( navigator.javaEnabled() )
19.                 document.write( 'Java é suportado.' );
20.             else
21.                 document.write( 'Java não é suportado.' );
22.         </script>
23.     </head>
24.     <body>
```

```
25.     </body>
26. </html>
```

8.3 Objeto Window

O objeto ***window*** é o objeto pai de todos os outros. Todos os outros objetos estão abaixo dele na hierarquia. Ele contém objetos que representam a janela do navegador, as definições de tela e cores, métodos de interação com o usuário.

Nos capítulos anteriores, pudemos observar alguns desses métodos: ***alert()***, ***prompt()*** e ***confirm()*** permitem interagir com o usuário. Agora conheceremos um pouco mais sobre este objeto.

Os tópicos seguintes mostrarão como identificar valores de cores e definição da máquina do usuário. E finalmente, como abrir, fechar janelas e acessar objetos de janelas diferentes.

Observem as seguintes propriedades. Elas pertencem a um sub−objeto de ***window***, o objeto ***screen***. O objeto ***screen*** contém os valores da resolução presente na máquina cliente, paleta de cores entre outras. Isso pode ser muito útil quando queremos nos certificar que, nossa página será carregada corretamente em qualquer resolução. Alguns programadores costumam produzir páginas em duas ou mais resoluções, e de acordo com os valores obtidos dessas propriedades, carregam a página mais adequada.

- ***width*** – é a área horizontal da tela em pixels.
- ***height*** – área vertical da tela em pixels.
- ***availWidth*** – é a área horizontal disponível para a página. Normalmente é igual ao valor de ***width***.
- ***availHeight*** – é a área vertical disponível para a página. Costuma ser menor que ***height***, pois se desconta o valor dos pixels ocupados pela barra de títulos e outras barras, como por exemplo, a barra de ferramentas do Windows.
- ***colorDepth*** – contém o número de cores configurado no computador cliente.

O exemplo abaixo imprime os valores descritos nas propriedades acima.

```
01. <html>
02.     <head>
03.         <title>Objeto Window</title>
04.         <script language=”Javascript”>
05.             // Resolução atual
06              document.write( 'Resolução atual: ' +
07.                 window.screen.width + 'x' +
08.                 window.screen.height, '<br>' );
09.
10.             // Resolução disponível
11.             document.write( 'Resolução disponível: ' +
12.                 window.screen.availWidth + 'x' +
13.                 window.screen.availHeight, '<br>' );
14.
15.             // Paleta de cores
16.             document.write( 'Paleta de cores: ' +
17.                 window.screen.colorDepth );
18.         </script>
19.     </head>
20.     <body>
21.     </body>
22. </html>
```

Agora vamos aprender a abrir e fechar janelas com os métodos ***open()*** e ***close()***. A sintaxe do método ***open()*** é a seguinte:

window.open(página a ser aberta, nome de acesso à página, características da janela)

- Características de uma janela:
  - ♦ toolbar − barra de ferramentas
  - ♦ location − barra de endereço
  - ♦ status − barra de status
  - ♦ menubar − menus
  - ♦ scroolbars − barras de rolagem
  - ♦ resizable − permite o redimensionamento da janela
  - ♦ width − largura da janela
  - ♦ height − altura da janela

E para fechar janelas utilizamos o método ***close()***, também podendo ser chamado como ***window.close()*** ou ***self.close()***, onde ***self*** representa a própria página.

Observem o exemplo abaixo. Criaremos um botão que ao ser clicado, abrirá uma janela popup. A janela aberta conterá um outro botão que terá a função de fechar a janela.

```
01. <!-- principal.htm -->
02. <html>
03.     <head>
04.         <title>Abrindo e fechando janelas</title>
05.         <script language="Javascript">
06.             function abreJanela() {
07.                 window.open( 'popup.htm', '', 'status=yes, menubar=no, resizab
08.                                 width=400, height=400' );
09.             }
10.         </script>
11.     </head>
12.     <body>
13.         <input type="button" name="abreJanela" value="Abrir janela"
14.             onClick="abreJanela();">
15.     </body>
16. </html>
```

Altere os parâmetros da janela e observe as mudanças nas características da mesma.

```
01. <!-- popup.htm -->
02. <html>
03.     <head>
04.         <title>Abrindo e fechando janelas</title>
05.     </head>
06.     <body>
07.         <input type="button" name="abreJanela" value="Fechar janela"
08.             onClick="close();">
09.     </body>
10. </html>
```

Agora, repetiremos o procedimento de lançamento de uma janela popup. Esta janela lançada conterá alguns campos. Escreveremos um script para manipular estes campos a partir da janela que originou a outra. Posteriormente, o procedimento inverso será desenvolvido por nós.

```
01. <!-- origem.htm -->
```

```
02. <html>
03.     <head>
04.         <title>Manipulando campos em outra janela</title>
05.         <script language="Javascript">
06.             function abreJanela() {
        07.                 janela = window.open( 'outrajanela.htm', '', 'status=yes, widt
        08.             }
09.         </script>
10.     </head>
11.     <body>
12.         <input type="button" name="abreJanela" value="Abrir janela"
13.             onClick="abreJanela();"><br>
14.         Após clicar no botão acima, utilize estes para trabalhar com os campos da jane
15.         <br>
16.         <input type="button" name="cor" value="Mudar a cor de fundo"
17.             onClick="janela.document.bgColor = '#0000FF';">
18.          
19.         <input type="button" name="valor" value="Obtendo valor do campo texto"
20.             onClick="alert('O valor do campo texto é: ' +
21.                 janela.document.f1.campotexto.value);">
22.          
23.         <input type="button" name="checar" value="Marcando o valor do campo checkbox"
24.             onClick="janela.document.f1.campocheckbox.checked = true;">
25.     </body>
26. </html>
```

```
01. <!-- outrajanela.htm -->
02. <html>
03.     <head>
04.         <title>Objeto window</title>
05.     </head>
06.     <body>
07.         <form name="f1">
08.             <input type="text" name="campotexto" value="Curso de Javascript">
09.             <br>
10.             <input type="checkbox" name="campocheckbox" value="Sim">Campo que será mar
11.             <br>
12.             <input type="button" name="fechar" value="Fechar janela" onClick="self.clo
13.         </form>
14.     </body>
15. </html>
```

No exemplo origem.htm, utilizamos três botões para manipular campos de uma janela que foi originada. Na linha 07, a variável ***janela*** recebe a referência de um objeto do tipo ***window***. Sendo assim, utilizamos ***janela*** como nome do nosso objeto. Dentro de ***janela***, temos uma página(***document***) e dentro dessa página, um formulário com alguns campos.

O primeiro botão, altera a cor de fundo(***bgColor***) da janela que foi aberta. O segundo botão, utiliza o método ***alert()*** para exibir o conteúdo do campo ***campotexto*** existente na janela popup. Finalmente, o terceiro botão muda o status de "não checado" do campo ***campocheckbox*** para ***checado***. Mudamos então o valor da propriedade ***checked*** do campo para ***true***.

A seguir, veremos dois métodos que têm a função de executar determinados procedimentos em intervalos regulares definidos pelo programador. A princípio, o procedimento é executado após o tempo definido, e apenas uma vez. Através de recursos de estruturas de programação como laços, a regularidade da execução pode ser conseguida. Estamos falando dos métodos ***setTimeout()*** e ***clearTimeout()***.

O método ***setTimeout()*** é utilizado para determinar um intervalo de tempo, onde um procedimento é executado. Sua sintaxe é a seguinte:

setTimeout(“expressão”, intervalo)

A expressão que faz parte do primeiro parâmetro, pode ser um comando ou uma função inteira e deve estar contido entre aspas. O intervalo é definido em milissegundos. Observem o exemplo a seguir:

```
01. <html>
02.     <head>
03.         <title>Método setTimeout</title>
04.         <script>
05.             setTimeout( ‘document.write( ‘Esta frase demorou 5
06.                          segundos para aparecer’, 5000 );
07.         </script>
08.     </head>
09.     <body>
10.     </body>
11. </html>
```

Quando executado, uma frase será escrita após 5 segundos(5000ms) serem passados. Neste caso, inserimos uma expressão diretamente no primeiro parâmetro do método. No próximo exemplo, uma função será o parâmetro de execução. Observem:

```
01. <html>
02.     <head>
03.         <title>Método setTimeout</title>
04.         <script>
05.             function redireciona() {
        06.                 document.location.href =
        07.                     'http://www.sistemasabertos.com.br';
        08.             }
09.
10.             setTimeout( "redireciona()", 7000 );
11.         </script>
12.     </head>
13.     <body>
14.         <center>
15.             Esta página será redirecionada dentro de 7 segundos.
16.         </center>
17.     </body>
18. </html>
```

A função redireciona() altera a propriedade href do objeto location para a nova URL informada. O método setTimeout() chama esta função após 7 segundos da página ter sido carregada. Então a página é redirecionada.

O método ***clearTimeout()*** tem a função de cancelar a execução de um procedimento iniciado pelo ***setTimeout()***. Para isso, deve−se atribuir a um identificador(variável), a referência de execução disparada pelo ***setTimeout()***. Sua sintaxe é a seguinte:

clearTimeout(identificador)

Observem o exemplo a seguir:

```
01. <html>
02.     <head>
```

```
03.         <title>Método clearTimeout</title>
04.         <style type="text/css">
05.             .botao {
06.                 font-family: Verdana;
07.                 font-size: 12px;
08.                 font-weight: bolder;
09.                 color: #FF0000;
10.                 background-color: #FFFFFF;
11.                 text-align: center;
12.                 height: 30px;
13.                 width: 70px;
14.                 border: 1px solid #000000;"
15.             }
16.         </style>
17.         <script language=”Javascript”>
18.             /*
19.                 Este exemplo mostra quanto segundos o usuário está
20.                 navegando na página. Iniciamos a variável segundos
21.                 com o valor 0 e criamos uma função que mostra o tempo
22.                 de permanência cada vez que o método setTimeout() a
23.                 chama. Observem que temos uma recursividade na
24.                 rotina, de forma que ela é executada a cada 1000
25.                 milissegundos, ou 1 segundo.
26.             */
27.             var segundos = 0;
28.
29.             function obtemSegundos()
30.             {
31.                 document.getElementById('tempo').value = segundos++ +
32.                     ' segundo(s)';
33.                 st = setTimeout('obtemSegundos()',1000);
34.             }
35.
36.             var st = setTimeout( 'obtemSegundos()', 1000 );
37.         </script>
38.     </head>
39.     <body>
40.         <center>
41.             <input type="button" value="0 segundo(s)" name="tempo"
42.                 id="tempo" disabled class="botao">
43.             <br><br>
44.             <input type="button" value="Parar tempo" name="parar"
45.                 class="botao" onClick="clearTimeout( st );">
46.         </center>
47.     </body>
48. </html>
```

Quando o botão "Parar tempo" é clicado, o método ***clearTimeout()*** finaliza o identificador ***st***, que estava referenciando a rotina ***obtemSegundos()***.

A seguir, trataremos de mais um objeto Javascript: o objeto History.

8.4 Objeto History

O objeto History é responsável por guardar e referenciar as páginas que o usuário navegou em uma determinada sessão de uso.

Seus principais métodos e propriedades são:

- ***length*** – retorna o número de entradas do histórico.

- ***go()*** – carrega a página especificada no parâmetro. Deve ser informado um valor inteiro, que corresponderá à posição desta página no array de histórico.
- ***back()*** – carrega a página visitada anteriormente, tal como um botão "Voltar" do navegador.
- ***forward()*** – simula um botão "Avançar", como o do navegador.

O exemplo abaixo, irá montar uma seqüência de acessos a algumas páginas que criaremos. Para navegar e obter um resultado satisfatório da utlização dos métodos e propriedades do objeto history, procure acessar as páginas na ordem proposta.

```
01. <!−− principal.htm −−>
02. <html>
03.     <head>
04.         <title>Objeto History</title>
05.         <script language="Javascript">
06.             document.write( 'Número de entradas no histórico: ' +
07.                 history.length, '<hr>' );
08.         </script>
09.     </head>
10.     <body>
11.         <a href="pagina1.htm">Página 1</a>
12.         <br>
13.         <a href="pagina2.htm">Página 2</a>
14.         <br>
15.         <a href="Javascript: history.forward();">Avançar</a>
16.    </body>
17. </html>
```

Utilizar a sintaxe "Javascript:" dentro da propriedade "href" da tag "" é permitido e muito útil. Isso se deve ao fato de haver implicitamente nos links, um evento onClick.

```
01. <!−− pagina1.htm −−>
02. <html>
03.     <head>
04.         <title>Objeto History</title>
05.     </head>
06.     <body>
07.         <a href="Javascript: history.back();">Voltar</a>
08.     </body>
09. </html>
```

O método ***back()*** do objeto history, carregará a página visitada anteriormente.

```
01. <!−pagina2.htm −−>
02. <html>
03.     <head>
04.         <title>Objeto History</title>
05.     </head>
06.     <body>
07.         <a href="Javascript: history.go( −1 );">Voltar</a>
08.     </body>
09.  </html>
```

O método ***go(−1)***, voltará uma página na lista de histórico.

Informe qualquer número inteiro(positivo ou negativo) para navegar entre os registros do histórico. Se for negativo, as páginas carregadas serão as últimas da lista. Se for positivo, as próximas.

Vimos neste capítulo quatro objetos muito importantes da linguagem Javascript: ***location***, ***navigator***, ***window*** e ***history***. O primeiro é responsável por manter propriedades e métodos a respeito da url, servidores e protocolos do documento. O objeto ***navigator*** guarda valores sobre o ambiente de interpretação da linguagem, ou seja, os navegadores. ***Window*** é o principal objeto do Javascript. Todos os outros objetos derivam dele. E finalmente, ***history*** armazena as entradas de páginas visitadas e fornece métodos para manipulação desses valores.

8.5 Resumo do capítulo

- O objeto ***location*** armazena informações sobre a URL, servidor da página corrente, sub−diretórios, protocolos e porta de comunicação que está sendo utilizada.
- Uma URL pode ser composta de protocolos, servidores, arquivos de scripts e parâmetros de valores.
- A propriedade ***href*** do objeto ***location*** armazena toda a URL. Ela pode ser muito útil quando houver a necessidade de se criar uma determinada URL ou fazer um redirecionamento de páginas.
- O objeto ***navigator*** guarda informações sobre os navegadores, suas características e funcionalidades. É importante porque através dessas informações, podemos produzir páginas e códigos específicos para satisfazer a necessidade de cada interpretador.
- Identificação de versões, aceitação de cookies no cliente e execução de aplicações Java são fornecidas pelo objeto ***navigator***.
- O objeto ***window*** é o principal objeto da linguagem ***Javascript***. Dele derivam todos os outros.
- A possibilidade de se criar novas janelas, manipular suas características estéticas, interagir com o usuário através de caixas de diálogo e várias outras funcionalidades pertencem a ***window***.
- Utilize os métodos ***open()*** e ***close()*** para gerar e fechar novas janelas.
- Os métodos ***confirm()***, ***alert()*** e ***prompt()*** permitem interagir com o usuário.
- Métodos como ***setTimeout()*** e ***clearTimeout()*** podem executar rotinas e finaliza−las em intervalos de tempo pré−definidos.
- O objeto ***history*** armazena todas as páginas visitadas pelo computador cliente.
- Ações que simulam a navegação, tais como, avançar para outras páginas, voltar no histórico e atualizações são fornecidas pelo objeto ***history***.
- Os métodos ***back()***, ***forward()*** e ***go()*** podem simular os botões ***Voltar***, ***Avançar*** e ***Atualizar*** de um navegador.

8.6 Exercícios de fixação

1. Crie uma página que contenha um objeto ***select*** com várias ***options*** contendo URLs válidas. Utilize eventos e o objeto ***location*** para redirecionar a página de acordo com a escolha do usuário.
2. Escreva uma rotina que identifique qual o navegador utilizado pelo cliente e carregue páginas diferentes para cada resposta que obtiver. Será necessário a utilização em conjunto dos objetos ***navigator*** e ***location***.
3. Crie uma página que irá carregar uma janela popup. Esta janela popup deverá conter uma caixa de texto onde o usuário deverá digitar seu nome, e um botão para fechar a página. Este botão deve conter um código para transportar este nome para um outro campo texto da página que originou a janela popup.
4. Acrescente ao exercício 2, uma rotina para descobrir qual a resolução utilizada na máquina cliente.
5. Crie uma rotina para exibir um relógio no formato "hh:mm:ss" que esteja em constante atualização.

9 Objetos IV − Criando Objetos

Até o momento, temos trabalhado com objetos oriundos de tags HTML e da própria linguagem Javascript. Aprendemos que estes objetos possuem suas características e ações que podem executar. São as propriedades e métodos.

Neste capítulo aprenderemos a criar nossos próprios objetos. Veremos como atribuir propriedades e métodos a esses objetos.

A princípio, utilizaremos uma sintaxe mais simples, onde o objeto é criado com propriedades e valores já definidos. Logo em seguida, utilizaremos um construtor para criar vários objetos semelhantes utilizando apenas uma estrutura.

Criaremos e observaremos nossa própria hierarquia de objetos. E finalmente, vamos escrever métodos que serão atribuídos aos nossos objetos.

A sintaxe básica para a criação de um objeto é muito simples:

objeto = { prop1: valor, prop2: valor, ..., propN: valor }

Vejam o exemplo abaixo:

```
01. <html>
02.     <head>
03.         <title>Criando Objetos</title>
04.         <script>
05.             /*
06.                 A sintaxe para se criar um objeto é semelhante a
07.                 criação de uma classe CSS, exceto que as propriedades
08.                 são separadas por "vírgula".
09.             */
10.             pessoa = { nome: 'Marcello', sexo: 'M', idade: '27' };
11.
12.             document.write( pessoa.nome, '<br>', pessoa.sexo, '<br>',
13.                             pessoa.idade );
14.         </script>
15.     </head>
16.     <body>
17.     </body>
18. </html>
```

No exemplo acima, criamos um objeto ***pessoa***. Este objeto possui três propriedades: nome, sexo e idade. A forma para se acessar cada uma dessas propriedades é idêntica aos objetos que já conhecemos: ***objeto.propriedade***.

Mas e se tivéssemos um outro objeto pessoa? Da forma acima, precisaríamos reescrever tudo novamente. Isto não seria muito agradável, a partir do momento em que as "pessoas" fossem numerosas.

Para resolver este problema, surgiu o conceito de ***construtores***. Em Javascript, um construtor nada mais é, do que uma função. Esta função tem a função de ser padrão, para vários objetos do mesmo tipo e mesmas características. Assim, basta que a invoquemos, passando como parâmetros os valores das propriedades já pré−definidas.

Observem o primeiro exemplo, utilizando construtores:

```
01. <html>
02.     <head>
03.         <title>Criando Objetos com construtores</title>
04.         <script>
05.             function pessoa( nome, sexo, idade ) {
06.                 this.nome  = nome;
07.                 this.sexo  = sexo;
08.                 this.idade = idade;
09.             }
10.
11.             // Criando vários objetos "pessoa"
12.             funcionario = new pessoa( 'Marcello', 'M', '27' );
13.             secretaria  = new pessoa( 'Maria', 'F', '22' );
14.             instrutor   = new pessoa( 'João', 'M', '24' );
15.
16.             // Imprimindo alguns valores
17.             document.write( funcionario.nome, '<br>' );
18.             document.write( secretaria.sexo, '<br>' );
19.             document.write( instrutor.idade, '<br>' );
20.         </script>
21.     </head>
22.     <body>
23.     </body>
24. </html>
```

Criamos um construtor chamado ***pessoa***. Este construtor receberá como parâmetros três valores: nome, sexo e idade. Em seguida, esses valores são atribuídos às propriedades de mesmo nome.

⚠ O objeto ***this*** é necessário em um construtor para fazer valer a independência do objeto. Suponhamos que, ao invés de "this" tivéssemos o nome do primeiro objeto: "funcionario". Teríamos:

```
.
.
05.     function pessoa( nome, sexo, idade ) {
        06.         funcionario.nome = 'Marcello';
        07.         funcionario.sexo = 'M';
        08.         funcionario.idade = 27;
        09.     }
.
.
```

No caso acima, não poderíamos criar outros objetos. Não teríamos como escrever por exemplo o nome do objeto "secretaria" ou a idade do objeto "instrutor". Todos os valores iriam para o objeto "funcionario". Por esse motivo, utiliza−se "this", para que todos os valores sejam destinados a seus respectivos objetos.

Outro detalhe importante é o uso do operador "new". Este operador cria uma referência do objeto pessoa e a atribui à variável que o está chamando. Dessa forma, "funcionario" aponta para um objeto em memória, "secretaria" para outro objeto e assim por diante.

Vamos aproveitar o último exemplo para acrescentar um novo objeto, chamado ***habilidades***. Este novo objeto será um sub−objeto do objeto ***pessoa***. Dessa forma, uma hierarquia será criada, assim como temos utilizado nos objetos da linguagem Javascript.

Vejamos:

```
01. <html>
02.     <head>
03.         <title>Criando Objetos com construtores</title>
04.         <script>
05.             function pessoa( nome, sexo, idade, habilidades ) {
```

```
06.                 this.nome        = nome;
07.                 this.sexo        = sexo;
08.                 this.idade       = idade;
09.                 this.habilidades = habilidades;
10.             }
11.
12.
13.             function habilidades( habil1, habil2 ) {
14.                 this.habil1 = habil1;
15.                 this.habil2 = habil2;
16.             }
17.
18.             // Criando objetos de habilidades
19.             habilidades1 = new habilidades( ‘comunicativo’, ‘dedicado’ );
20.             habilidades2 = new habilidades( ‘prestatividade’, ‘pontualidade’ );
21.
22.             // Criando objetos “pessoa”
23.             funcionario = new pessoa( ‘Marcello’, ‘M’, ‘27’, habilidades1 );
24.             secretaria  = new pessoa( ‘Maria’, ‘F’, ‘22’, habilidades2 );
25.
26.             // Imprimindo alguns valores
27.             document.write( funcionario.nome, ‘<br>’ );
28.             document.write( funcionario.habilidades.habil1, ‘<br>’ );
29.             document.write( secretaria.sexo, ‘<br>’ );
30.             document.write( secretaria.habilidades.habil2, ‘<br>’ );
31.         </script>
32.     </head>
33.     <body>
34.     </body>
35. </html>
```

O exemplo acima possui dois construtores de objetos: ***pessoa()*** e ***habilidades()***. Observem que objetos do tipo ***pessoa()*** tem uma 4ª propriedade: ***habilidades***. Essa propriedade, na verdade é um outro objeto.

O objeto ***habilidades*** possui duas propriedades. Instanciamos dois objetos ***habilidades*** com valores diferentes. Em seguida, ao instanciarmos os objetos ***funcionário*** e ***secretaria***, dizemos ao construtor que a 4ª propriedade é um objeto do tipo ***habilidades***.

Daí por diante é só observar a hieraquia. Se quisermos obter os valores das habilidades de um funcionario ou secretaria, basta que os acessemos da seguinte forma:

funcionario.habilidades.habil1 ou secretaria.habilidades.habil2

Nosso próximo passo é incluir métodos aos nossos objetos.

Observem:

```
<html>
<head>
<title>Criando Objetos</title>
<script language="Javascript">
// construtor para objetos do tipo animal
function animal( especie, alimento, mensagem ) {
        this.especie  = especie;
        this.alimento = alimento;
        this.mensagem = mensagem;
        this.dizer = dizer;
}

// método para objetos do tipo animal
function dizer() {
```

```
        document.write( this.mensagem, '<br>' );
}

// instanciando os objetos
rex   = new animal( "Cachorro", "Ossos", "Au Au" );
felix = new animal( "Gato", "Sardinhas", "Miau Miau" );

// imprimindo algumas propriedades dos objetos
document.write( rex.especie, '<br>' );
document.write( felix.especie, '<br>' );

// executando os métodos associados
rex.dizer();
felix.dizer();
</script>
</head>
<body>
</body>
</html>
```

Note a criação de um construtor para objetos do tipo ***animal***, note a instanciação de dois objetos utilizando esse construtor: ***rex*** e ***Felix***.

Note também a criação do método dizer e a sua associção como método do objeto animal, através da linha 'this.dizer = dizer'

Pronto. O método já pertence a nossos objetos ***rex*** e ***felix***. Basta que os chamemos com seus respectivos valores.

Aprendemos neste capítulo a criar objetos próprios, com as caracteríticas e ações do nosso interesse. Os objetos são a base do Javascript. Toda a linguagem gira em torno deles.

Por isso, compreendê−los torna−se essencial para o sucesso do aprendizado.

A seguir, um capítulo onde há a possibilidade de se utilizar a linguagem Javascript para manipular objetos de documentos diferentes dentro um mesmo objeto ***window***. Estamos falandos dos frames.

9.1 Revisão do Capítulo

- O Javascript permite a criação de objetos com propriedades e métodos, além dos que ele já fornece.
- A sintaxe de criação de um objeto é semelhante à sintaxe de uma classe CSS.
- Propriedades de objetos são acessadas da seguinte forma: ***objeto.propriedade***.
- Métodos de objetos são acessados da seguinte forma: ***objeto.metodo()***.
- Construtores permitem oficializar um molde para diversos objetos de um mesmo tipo.
- O operador ***new*** permite instanciar uma cópia de um objeto na memória.
- Métodos podem ser associados a objetos utilizando−se a propriedade ***prototype*** que todos os objetos possuem.

9.2 Exercícios de Revisão

1. Escreva uma aplicação onde um objeto do tipo ***POLITICO***. Faça−o ter propriedades como ***partido*** e ***cargo disputado***. Instancie dois ou três objetos deste tipo. Imprima os valores das propriedades desse objetos.
2. Acrescente ao exercício anterior, um outro objeto chamado ***CARATER***, com propriedades ***carater1*** e ***carater2***. Altere o objeto ***POLITICO*** para conter mais estas propriedades de ***CARATER***. Instancie e

imprima os valores obtidos.

3. Crie um método ***DISCURSAR***. Associe ao objeto ***POLITICO*** e utilize este método para imprimir algum texto.

10 Janelas e Frames

Este capítulo irá abordar as diferentes nuances de se trabalhar com frames. Um ***frameset*** é uma coleção de documentos, todos exibidos dentro de uma mesma janela. O importante é podermos conhecer a hierarquia e saber aplicar tudo o que aprendemos até agora, nas diferentes páginas a que teremos acesso.

Como vários ***documents*** estão dentro de uma mesma janela, precisamos referenciar esta janela através de um objeto. Este objeto é chamado de ***top*** ou ***parent***. As páginas ou documentos que ele contiver serão acessadas através da sua ordem de declaração, como em um array.

Observem:

A imagem acima mostra três documentos dentro de uma mesma janela. São os frames. Como todos os objetos do mesmo tipo que estivemos conhecendo durante os capítulos anteriores, os frames também podem ser acessados através de índices de array. Sabemos que o objeto que os comporta é o ***top***, podemos então chamá−los assim:

top.frames[0]
top.frames[1]
top.frames[2]

O exemplo abaixo representa o código HTML para a tela acima:

```
01. <html>
02.     <head>
03.         <title>Trabalhando com Frames</title>
04.     </head>
05.     <frameset cols="20%, *" frameborder="yes">
06.         <frame src="frame1.htm" name="esquerda">
07.         <frameset rows="20%, *" frameborder="yes">
08.             <frame src="frame2.htm" name="topo">
09.             <frame src="frame3.htm" name="principal">
10.         </frameset>
11.     </frameset>
12. </html>
```

Um arquivo HTML que contém frames não possui tag ***body***. Na verdade, cada ***frame*** terá o seu próprio ***body***, mas nunca o frameset.
Cada ***document*** é representado pela tag ***frame***. Sendo assim, no exemplo acima temos três documentos.
A ordem de acesso dos mesmos, é a ordem de declaração.

Vamos então praticar um pouco. A princípio criaremos um frameset que possuirá quatro documentos. Vamos explorar algumas propriedades e métodos de vários objetos neste frameset.

O código abaixo é o nosso frameset:

```
01. <!−− frameset.htm −−>
02. <html>
03.     <head>
04.         <title>Trabalhando com Frames</title>
05.     </head>
06.     <frameset cols="50%, *" rows="50%, *" frameborder="yes">
07.         <frame src="frame1.htm" name="f1">
08.         <frame src="frame2.htm" name="f2">
09.         <frame src="frame3.htm" name="f3">
```

```
10.         <frame src="frame4.htm" name="f4">
11.     </frameset>
12. </html>
```

A seguir temos o código para a página que ocupará o primeiro frame, a qual denominamos ***frame1.htm***. Iremos adicionar vários elementos nesta página aos poucos. Enquanto isso, salve as páginas ***frame2.htm***, ***frame3.htm*** e ***frame4.htm*** todas em branco, sem nenhum conteúdo.

```
01. <!-- frame1.htm -->
02. <html>
03.     <head>
04.         <title>Trabalhando com Frames</title>
05.     </head>
06.     <body>
07.         <!-- Este botão muda cor de fundo do 2º frame -->
08.         <input type="button" name="cor" value="Mudar a cor do 2º frame" onClick="top.f
09.     </body>
10. </html>
```

Lembrem-se: a ordem de acesso aos frames é a mesma ordem de declaração. Sendo assim, ***top.frames[1]*** é o 2º frame declarado.
Frames é um array de frames. Os índices de um array comecam por 0.

Vamos inserir um outro botão que carregará uma determinada página no quarto frame declarado:

```
01. <!-- frame1.htm -->
02. <html>
03.     <head>
04.         <title>Trabalhando com Frames</title>
05.     </head>
06.     <body>
07.         <!-- Este botão muda cor de fundo do 2º frame -->
08.         <input type="button" name="cor" value="Mudar a cor do 2º frame"
09.             onClick="top.frames[1].document.bgColor = '#FF0000';">
10.         <br>
11.         <!-- Este botão carrega uma página no 4º frame -->
12.         <input type="button" name="pag" value="Carrega página no 4º frame"
13.             onClick="top.frames[3].document.location.href = 'http://www.sistemasaberto
14.     </body>
15. </html>
```

Para finalizar este primeiro exemplo, acrescentaremos mais um exemplo, onde o método ***write()*** do 3º frame declarado para imprimir uma mensagem.

```
01. <!-- frame1.htm -->
02. <html>
03.     <head>
04.         <title>Trabalhando com Frames</title>
05.     </head>
06.     <body>
07.         <!-- Este botão muda cor de fundo do 2º frame -->
08.         <input type="button" name="cor" value="Mudar a cor do 2º frame"
09.             onClick="top.frames[1].document.bgColor = '#FF0000';">
10.         <br><br>
11.         <!-- Este botão carrega uma página no 4º frame -->
12.         <input type="button" name="pag" value="Carrega página no 4º frame"
```

```
13.             onClick="top.frames[3].document.location.href = 'http://www.sistemasaberto
14.         <br><br>
15.         <!-- Este botão imprime uma mensagem no 3º frame -->
16.         <input type="button" name="msg" value="Imprime mensagem no 3º frame"
17.             onClick="top.frames[2].document.write( 'Trabalhando com frames.' );">
18.     </body>
19. </html>
```

Nosso próximo exemplo, mostrará como manipular outros objetos em frames diferentes. Utilizaremos a mesma estrutura do frameset do exemplo anterior. Observem:

```
01. <!-- frameset.htm -->
02. <html>
03.     <head>
04.         <title>Trabalhando com Frames</title>
05.     </head>
06.     <frameset cols="50%, *" rows="50%, *" frameborder="yes">
07.         <frame src="frame1.htm" name="f1">
08.         <frame src="frame2.htm" name="f2">
09.         <frame src="frame3.htm" name="f3">
10.         <frame src="frame4.htm" name="f4">
11.     </frameset>
12. </html>
```

O código acima reproduz o frameset. O próximo código representará um formulário do 2º frame. Será um formulário com um campo texto, que receberá um valor demandado pela página de controle, que será construída posteriormente.

```
01. <!-- frameset.htm -->
02. <html>
03.     <head>
04.         <title>Frame 2</title>
05.     </head>
06.     <body>
07.         <center>
08.             <form name="f1">
09.                 <input type="text" name="msg" size="50">
10.             </form>
11.         </center>
12.     </body>
13. </html>
```

O 3º frame terá alguns botões de radio para que sejam selecionados de acordo com a escolha do usuário. Porém isso será feito através da página de controle. No momento, codifiquemos apenas o formulário desejado.

```
01. <!-- frameset.htm -->
02. <html>
03.     <head>
04.         <title>Trabalhando com Frames</title>
05.     </head>
06.     <body>
07.         <form name="f1">
08.             <input type="radio" name="opcao" value="1"> Opção 1
09.             <br>
10.             <input type="radio" name="opcao" value="2"> Opção 2
11.             <br>
12.             <input type="radio" name="opcao" value="3"> Opção 3
```

```
13.         </form>
14.     </body>
15. </html>
```

A seguir o 4º frame. Este frame será um novo formulário, agora com um único elemento checkbox, que também será selecionado pela página de controle.

```
01. <!-- frameset.htm -->
02. <html>
03.     <head>
04.         <title>Trabalhando com Frames</title>
05.     </head>
06.     <body>
07.         <form name="f1">
08.             <center>
09.                 <input type="checkbox" name="opcao" value="1"> Opção 1
10.             </center>
11.         </form>
12.     </body>
13. </html>
```

Finalmente, o 1º frame. O 1º frame será nossa página controladora. Através dela, as ações serão disparadas. Porém os efeitos serão sofridos pelos outros frames. Observem:

```
01. <!-- frameset.htm -->
02. <html>
03.     <head>
04.         <title>Trabalhando com Frames</title>
05.     </head>
06.     <body>
07.         <!-- Este botão exibirá uma mensagem na caixa de texto do 2º frame -->
08.         <input type="button" name="msg" value="Criar mensagem!"
09.             onClick="top.frames[1].document.f1.msg.value = 'Curso de Javascript';">
10.
11.         <hr>
12.         <!-- Este botões selecionam as opções existentes no 3º frame -->
13.         <input type="button" name="opcoes" value="Seleciona 1ª"
14.             onClick="top.frames[2].document.f1.opcoes[0].checked = true;">
15.         <br>
16.         <input type="button" name="opcoes" value="Seleciona 2ª"
17.             onClick="top.frames[2].document.f1.opcoes[1].checked = true;">
18.         <br>
19.         <input type="button" name="opcoes" value="Seleciona 3ª"
20.             onClick="top.frames[2].document.f1.opcoes[2].checked = true;">
21.
22.         <hr>
23.         <!-- Estes botões marcam e desmarcam a checkbox do 4º frame -->
24.         <input type="button" name="marcar" value="Marcar"
25.             onClick="top.frames[3].document.f1.opcao.checked = true;">
26.         <br>
27.         <input type="button" name="desmarcar" value="Desmarcar"
28.             onClick="top.frames[3].document.f1.opcao.checked = false;">
29.     </body>
30. </html>
```

Como vimos, o segredo para se trabalhar com frames está simplesmente na sua hierarquia vetorial. Compreendendo que um frame nada mais é do que um objeto ***document***, todos as propriedades e metódos

deste também estarão disponíveis.

10.1 Resumo do capítulo

- Um ***Frameset*** é uma coleção de objetos ***document***, todos contidos dentro de uma mesma janela.
- O array ***frames*** é o objeto Javascript que representa os frames de uma página. O objeto ***frames*** por sua vez pertence ao objeto ***top*** ou ***parent***, aqui representando o objeto ***window***.
- Os índices do array ***frames*** começam do 0(zero) como todo array Javascript e a ordem de acesso de cada frame, é a mesma ordem de declaração no código.
- Um documento do tipo ***Frameset*** não possui a tag ***body***, uma vez que cada elemento seu(frame) terá uma.
- Propriedades e métodos de um objeto ***document*** estão disponíveis para cada elemento de um array de frames, uma vez que esses elementos também são objetos ***document***.

10.2 Exercícios de fixação

1. O que o código abaixo faz:

```
top.frames[1].sexo[1].checked = false;//considere sexo um objeto do tipo radio button
```

- ♦ (a) Marca a primeira opção do botão chamado ***sexo*** do primeiro frame.
- ♦ (b) Desmarca a primeira opção do botão chamado ***sexo*** do primeiro frame.
- ♦ (c) Desmarca a segunda opção do botão chamado ***sexo*** do primeiro frame.
- ♦ (d) Desmarca a segunda opção do botão chamado ***sexo*** do segundo frame.
- ♦ (e) Marca a segunda opção do botão chamado ***sexo*** do segundo frame.

1. Monte um frameset dividido em duas partes. Inclua na primeira parte um objeto do tipo ***select*** com várias opções de sites que você goste. Faça um script para trocar a página do segundo frame, obedecendo a escolha da caixa de seleção criada no primeiro frame.

11 Arrays e Matrizes

Neste capítulo faremos uma revisão do objeto ***Array***, uma vez que já o estamos utilizando desde o início do livro. Percebemos que todos os objetos de um mesmo tipo, ao se unirem dentro de um mesmo documento, podem ser agrupados dentre de um array.

Matrizes serão o outro tópico de estudo dessa fase. Veremos que seu uso é bastante e simples e também aprenderemos que elas são tão somente arrays de outros arrays. Então, mãos a obra.

11.1 Objeto Array

O objeto ***Array*** permite que representemos uma coleção de objetos do mesmo tipo. Por exemplo, todas as imagens de uma página podem ser representadas pelo array ***images***. Existem duas formas simples de se criar um array. A primeira é informando diretamente ao construtor, os elementos desejados. Observem:

```
...      carros = new Array( "GOL", "CORSA", "PALIO" );
```

- Ao se instanciar um array lembrem−se de que a primeira letra da palavra array deve estar maiúscula.
- Arrays são criados através do comando ***new***.

A outra forma de se criar o array é a seguinte:

```
...      carros = new Array( 3 );//indica que o array terá três elementos
...      //Neste caso os elementos são inseridos através de seus índices
...      carros[0] = "GOL";
...      carros[1] = "CORSA";
...      carros[2] = "PALIO";
```

Uma vez criados, a forma mais fácil de se obter os valores novamente, é através de um laço de repetição. Vamos a um exemplo prático:

```
01. <html>
02.     <head>
03.         <title>Trabalhando com arrays</title>
04.         <script language="Javascript">
05.             notas = new Array( 3 );
06.
07.             //Este laço obtém do usuário valores que serão atribuídos ao array de notas
08.             for( i = 0; i < 3; i++ ) {
09.                 notas[i] = prompt( 'Informe a nota do aluno ' + ( i + 1 ) , ''
10.             }
11.
12.             //Uma vez o array criado, iremos imprimir os valores na tela
13.             for( i = 0; i < 3; i++ ) {
14.                 document.write( 'Aluno ', i + 1, ': ', notas[i], '<br>' );
15.             }
16.         </script>
17.     </head>
18.     <body>
19.     </body>
20. </html>
```

O acesso a cada elemento é feito pela sua posição dentro do array, como mostrado no exemplo. O laço de repetição é muito útil, porque sua estrutura permite o acesso de todos os elementos de array em uma só rotina de programação. Imaginem uma impressão de relatório de notas de 1000 alunos feitas sem o recurso do laço?

11.2 Matrizes

Matrizes em Javascript, são na verdade, uma forma de se utilizar múltiplos arrays, cada um se originando do outro. Torna−se meramente uma questão algorítmica e mecânica o seu uso.

Por exemplo, se quisermos utilizar uma matriz 3x3, teremos na verdade 4 arrays. Podemos dizer que, cada elemento do primeiro array de 3 elementos, será um novo array de 3 elementos.

Observem o exemplo abaixo. Montaremos uma tabela de alunos e notas. O script irá solicitar do usuário o nome e a notas de cada aluno e depois imprimirá os dados na tela.

```
01. <html>
02.     <head>
03.         <title>Matrizes</title>
04.         <script language="Javascript">
05.             //criando o array de alunos
06.             alunos = new Array( 3 );
07.
08.             //contador de alunos − apenas para informar ao usuário com qual aluno está
09.             numAluno = 1;
10.             //solicitando os nomes dos alunos
11.             for( i = 0; i < 3; i++ ) {
        12.                 alunos[i] = prompt( 'Informe o nome do aluno ' + numAluno + ':
        13.
        14.                 //incrementamos numAluno para informar ao usuário qual o próxi
        15.                 numAluno++;
        16.             }
17.
18.             //contador de notas com mesma funcionalidade informativa do contador de al
19.             numNotas = 1;
20.             //voltando contador de alunos para posição inicial
21.             numAluno = 1;
22.
23.             //criando a matriz e solicitando os valores das notas ao usuário
24.             notas = new Array( 3 );
25.             for( i = 0; i < notas.length; i++ ) {
        26.                 //observem que cada posição de notas será um novo array
        27.                 notas[i] = new Array( 3 );
        28.
        29.                 for( j = 0; j < notas[i].length; j++ ) {
                30.                     //agora existem índices para cada dimensão do arra
                31.                     notas[i][j] = prompt( 'Informe a ' + numNotas + 'ª
                32.                     numNotas++;
                33.                 }
        34.                 numAluno++;
        35.
        36.                 //voltando o contador de notas para a posição inicial
        37.                 numNotas = 1;
        38.             }
39.
40.             //imprimindo os valores obtidos
41.             for( i = 0; i < alunos.length; i++ ) {
        42.                 document.write( 'Aluno ', alunos[i], '<br>' );
        43.                 for( j = 0; j < notas[i].length; j++ ) {
                44.                     document.write( ' >>> ', notas[i][j], '<br>' );
                45.                 }
        46.                 document.write( '<br>' );
        47.             }
48.         </script>
49.     </head>
50. </html>
```

Como vimos, a utilização de arrays é bastante simples e passa a ser mecânica a partir do momento que entendemos que cada posição do array superior, é um novo array, seja ele de qualquer dimensão.

Matematicamente, sabemos que dimensões além da 3ª podem ser representadas. O mesmo acontece aqui. Quanto mais dimensões, mais índices você terá que controlar.

⚠ Quanto mais dimensões utilizar, mais código terá que escrever.

- ***Algumas propriedades e métodos de arrays***
  - ♦ length − retorna a quantidade de elementos de um array ou matriz
  - ♦ join() − une os elementos de um array através do separador informado como parâmetro:
    - ◊ ex. array.join(' − ')
    - ◊ resultado: elemento 1 − elemento 2 − ... − elemento n
  - ♦ reverse() − inverte a ordem dos elementos
    - ◊ ex. array.reverse()
  - ♦ sort() − ordena os elementos do array
    - ◊ ex. array.sort()

⚠ Outros métodos como pop(), shift(), unshift() entre outros, devem ser utilizados com cuidado, pois nem todos os navegadores os interpretarão.

11.3 Resumo do capítulo

- Em Javascript, todos os objetos de um mesmo tipo podem ser representados em um array. Ex. images[], forms[], links[] etc.
- Os arrays tem índice inicial igual a 0(zero).
- Ao declarar um array, lembre−se que a primeira letra deve ser maiúscula. Ex. carros = new Array(4).
- Os arrays podem ser declarados informando diretamente os seus elementos, ou informando o seu tamanho.
- A forma mais simples de se percorrer um array é através do uso de laços de repetição.
- Matrizes são arrays que têm como elementos, outros arrays.

11.4 Exercícios de fixação

1. O array frutas("mamão", "uva", "melancia", "banana") deve ser impresso em ordem decrescente. Utilize os métodos de arrays e os imprima na tela.
2. Observem o seguinte array: estados("GO", "MG", "SP", "RJ", "AM", "PA"). Faça um laço de repetição inserindo todos esses elementos dentro de um objeto "select", onde o usuário poderá escolher o valor que lhe convier.
3. Crie uma matriz chamada ***agenda*** de dimensão 5x2. Peça ao usuário para informar os dias úteis da semana e um compromisso para cada dia. Imprima a agenda.

12 Referência de Métodos

- abs()
  - ♦ objeto: Math
  - ♦ ação: retorna o valor absoluto de um número
  - ♦ parâmetros: valor numérico
  - ♦ ex. Math.abs();
- alert()
  - ♦ objeto: window
  - ♦ ação: exibir caixa de diálogo(janela de alerta)
  - ♦ parâmetros: strings e propriedades de objetos
  - ♦ ex. alert('*** ERRO!');
- anchor()
  - ♦ objeto: String
  - ♦ ação: cria uma âncora para um link
  - ♦ parâmetros: nome da âncora a ser chamada
  - ♦ ex. string.anchor('nome da âncora');
- back()
  - ♦ objeto: history
  - ♦ ação: carrega a página visitada anteriormente, tal como um botão "Voltar" do navegador
  - ♦ parâmetros: não
  - ♦ ex. history.back();
- big()
  - ♦ objeto: String
  - ♦ ação: retorna uma cópia da string formatada com a tag big
  - ♦ parâmetros: não
  - ♦ ex. string.big();
- blink()
  - ♦ objeto: String
  - ♦ ação: retorna uma string com a formatação da tag blink
  - ♦ parâmetros: não
  - ♦ ex. string.blink()
- bold()
  - ♦ objeto: String
  - ♦ ação: retorna uma string envolta pela tag b
  - ♦ parâmetros: não
  - ♦ ex. string.bold();
- ceil()
  - ♦ objeto: Math
  - ♦ ação: arredonda para cima
  - ♦ parâmetros: número a ser arredondado
  - ♦ ex. Math.ceil(2.1);
- charAt()
  - ♦ objeto: String
  - ♦ ação: retorna o caracter na posição informada
  - ♦ parâmetros: posição desejada
  - ♦ ex. string.charAt(5);
- charCodeAt()
  - ♦ objeto: String
  - ♦ ação: retorna o código ASCII do caracter na posição informada
  - ♦ parâmetros: posição desejada
  - ♦ ex. string.charCodeAt(5);
- clearTimeout()
  - ♦ objeto: window

- ♦ ação: tem a função de cancelar a execução de um procedimento iniciado pelo setTimeout()
- ♦ parâmetros: identificador do procedimento a ser cancelado
- ♦ ex. clearTimeout(identificador)

- close()
  - ♦ objeto: window
  - ♦ ação: fechar uma janela
  - ♦ parâmetros: não
  - ♦ ex. close();
- concat()
  - ♦ objeto: String
  - ♦ ação: concatena duas ou mais strings
  - ♦ parâmetros: strings a serem concatenadas
  - ♦ ex. string1.concat(string2, string3, ..., stringn);
- confirm()
  - ♦ objeto: window
  - ♦ ação: exibir caixa de diálogo de confirmação para o usuário, retorna true ou false
  - ♦ parâmetros: mensagem de interação com o usuário
  - ♦ ex. confirm('Deseja sair desta página?');
- eval()
  - ♦ objeto: window
  - ♦ ação: avalia e executa uma expressão matemática
  - ♦ parâmetros: requer uma expressão matemática válida
  - ♦ ex. eval(document.f1.numero1.value + document.f1.numero2.value);
- fixed()
  - ♦ objeto: String
  - ♦ ação: utiliza a tag tt para formatar a string
  - ♦ parâmetros: não
  - ♦ ex. string.fixed();
- floor()
  - ♦ objeto: Math
  - ♦ ação: arredonda para baixo
  - ♦ parâmetros: número a ser arredondado
  - ♦ ex. Math.floor(3.8);
- focus()
  - ♦ objeto: vários
  - ♦ ação: muda o foco da aplicação para o objeto que chamar este método
  - ♦ parâmetros: não
  - ♦ ex. document.f1.caixadetexto.focus();
- fontColor()
  - ♦ objeto: String
  - ♦ ação: utiliza o atributo color da tag font
  - ♦ parâmetros: cor em hexadecimal ou correspondente em inglês
  - ♦ ex. string.fontColor('#FF0000');
- fontSize()
  - ♦ objeto: String
  - ♦ ação: utiliza o atributo size da tag font
  - ♦ parâmetros: tamanho da fonte
  - ♦ ex. string.fontSize(16);
- forward()
  - ♦ objeto: history
  - ♦ ação: simula um botão "Avançar", como o do navegador
  - ♦ parâmetros: não
  - ♦ ex. history.forward()
- fromCharCode()

- objeto: String
- ação: constrói uma string a partir do código ASCII informado
- parâmetros: código ASCII dos caracteres desejados
- ex. String.fromCharCode(74, 97, 118, 97, 115, 99, 114, 105, 112, 116);

- getDate()
  - objeto: Date
  - ação: retorna o dia de uma data específica
  - parâmetros: não
  - ex. data.getDate();
- getElementById()
  - objeto: document
  - ação: retorna o objeto solicitado pelo seu id
  - parâmetros: id do objeto desejado
  - ex. document.getElementById(1), onde 1 é o id do objeto
- getFullYear()
  - objeto: Date
  - ação: retorna o ano com 4 dígitos de uma data específica
  - parâmetros: não
  - ex. data.getFullYear();
- getMonth()
  - objeto: Date
  - ação: retorna o mês de uma data específica
  - parâmetros: não
  - ex. data.getMonth()
- go()
  - objeto: history
  - carrega a página especificada no parâmetro. Deve ser informado um valor inteiro, que corresponderá à posição desta página no array de histórico
  - parâmetros: posição do array de histórico que corresponde a página desejada
  - ex. history.go(2);
- indexOf()
  - objeto: String
  - ação: procura pela primeira ocorrência do caracter ou string informada
  - parâmetros: parte ou todo de uma string
  - ex. string.indexOf('a');
- italics()
  - objeto: String
  - ação: transforma a string em itálico, equivalente à tag i
  - parâmetros: não
  - ex. string.italics();
- javaEnabled()
  - objeto: navigator
  - ação: verifica se o navegador suporta ou não Java. Retorna true ou false
  - parâmetros: não
  - ex. navigator.javaEnabled();
- join()
  - objeto: Array
  - ação: une os elementos de um array através do separador informado como parâmetro
  - parâmetros: separador desejado
  - ex. array.join(':');
- lastIndexOf()
  - objeto: String
  - ação: Procura pela última ocorrência do caracter ou string informada
  - parâmetros: parte ou todo de uma string

    - ♦ ex. string.lastIndexOf('a');
- link()
    - ♦ objeto: String
    - ♦ ação: cria um link
    - ♦ parâmetros: página a ser chamada
    - ♦ ex. string.link('http://www.sistemasabertos.com.br');
- max()
    - ♦ objeto: Math
    - ♦ ação: retorna o maior dos parâmetros informados
    - ♦ parâmetros: números a serem comparados
    - ♦ ex. Math.max(5, 2, 7, 3);
- min()
    - ♦ objeto: Math
    - ♦ ação: retorna o menor dos parâmetros informados
    - ♦ parâmetros: números a serem comparados
    - ♦ ex. Math.min(5, 2, 7, 3);
- open()
    - ♦ objeto: window
    - ♦ ação: abre uma nova janela
    - ♦ parâmetros: arquivo a ser aberto, nome do objeto, características da janela respectivamente
    - ♦ ex. open('popup.htm', '', 'status=yes, menubar=no, resizable=no, width=400, height=400');
- pow()
    - ♦ objeto: Math
    - ♦ ação: eleva o primeiro valor informado, à potência representada pelo segundo valor
    - ♦ parâmetros: base e potência, respectivamente
    - ♦ ex. Math.pow(2, 3);
- prompt()
    - ♦ objeto: window
    - ♦ ação: solicita uma entrada de dados ao usuário
    - ♦ parâmetros: o primeiro parâmetro é a mensagem de interação com o usuário. O segundo é opcional e informa qual texto será default na caixa de diálogo
    - ♦ ex. prompt('Informe seu nome:', '');
- reverse()
    - ♦ objeto: Array
    - ♦ ação: inverte a ordem dos elementos
    - ♦ parâmetros: não
    - ♦ ex. array.reverse()
- round()
    - ♦ objeto: Math
    - ♦ ação: arredonda para cima se o valor passado é x.5 ou maior, caso contrário, arredonda para baixo
    - ♦ parâmetros: valor a ser arredondado
    - ♦ ex. Math.round(5.5);
- select()
    - ♦ objeto: caixas de texto e textarea
    - ♦ ação: seleciona os valores(texto) dos objetos que invocam este método
    - ♦ parâmetros: não
    - ♦ ex. document.f1.caixadetexto.select();
- setFullYear()
    - ♦ objeto: Date
    - ♦ ação: altera o ano de uma data para o ano informado como parâmetro
    - ♦ parâmetros: ano válido
    - ♦ ex. data.setFullYear(ano);
- setTimeout()

- objeto: window
- ação: utilizado para determinar um intervalo de tempo, onde um procedimento é executado
- parâmetros: expressão a ser executada e intervalo de tempo em millisegundos, respectivamente
- ex. setTimeout(‘document.write(‘Esta frase demorou 5 segundos para aparecer’, 5000);

• slice()
- objeto: String
- ação: retorna uma substring delimitada pelas posições de início e fim informadas
- parâmetros: posição inicial e posição final da string desejada
- ex. string.slice(0, 5);

• small()
- objeto: String
- ação: retorna uma string formatada com a tag small
- parâmetros: não
- ex. string.small();

• sort()
- objeto: Array
- ação: ordena os elementos de um array
- parâmetros: não
- ex. array.sort();

• split()
- objeto: String
- ação: divide a string onde houver ocorrência do parâmetro informado e atribui os pedaços a um array
- parâmetros: separador
- ex. data.split("/");

• sqrt()
- objeto: Math
- ação: retorna a raiz quadrada
- parâmetro: valor numérico
- ex. Math.sqrt(9);

• sub()
- objeto: String
- ação: formata a string utilizando a tag sub
- parâmetros: não
- ex. string.sub();

• substr()
- objeto: String
- ação: retorna uma substring delimitada pela posição inicial constante no 1º parâmetro e pelo comprimento, informado no segundo parâmetro
- parâmetros: posição inicial e tamanho desejado da string
- ex. string.substr(5, 10);

• substring()
- objeto: String
- ação: idêntico ao método slice()
- parâmetros: posição inicial e posição final da string desejada
- ex. string.substring(0, 5);

• sup()
- objeto: String
- ação: formata a string utilizando a tag sup
- parâmetros: não
- ex. string.sup();

• toLowerCase()
- objeto: String

- ♦ ação: converte a string para minúsculas
- ♦ parâmetros: não
- ♦ ex. string.toLowerCase()

- toUpperCase()
  - ♦ objeto: String
  - ♦ ação: converte a string para maiúsculas
  - ♦ parâmetros: não
  - ♦ ex. string.toUpperCase()
- write()
  - ♦ objeto: document
  - ♦ ação: utilizado para exibir saída de dados na tela
  - ♦ parâmetros: strings e propriedades de objetos
  - ♦ ex. document.write('Alô Mundo');

13 Repostas dos Exercícios de Fixação

13.1 Capítulo II

13.1.1 Exercício 1

```
var nome  = "Marcello";
var idade = 27;
var peso  = 69.5;
```

13.1.2 Exercício 2

```
01. <script language="Javascript">
02.     function parImpar( num ) {
        03.         if( num%2 == 0 ) //se resto da divisão por 2 for zero, o número é par
        04.             document.write( 'O número ', num, ' é par' );
        05.         else
        06.             document.write( 'O número ', num, ' é ímpar' );
        07.     }
08.
09.     parImpar( 5 );
10. </script>
```

13.1.3 Exercício 3

```
01. <script language="Javascript">
02.     for( i = 2004; i >= 1977; i-- )
03.         document.write( i, '<br>' );
04. </script>
```

13.2 Capítulo III

13.2.1 Exercício 1

```
01. <script language="Javascript">
02.     var nome      = "José";
03.     var sobrenome = "Silva";
04.
05.     document.write( '<font color="#FF0000">', nome, ' ', sobrenome, '</font>' );
06. </script>
```

13.2.2 Exercício 2

- arquivoexterno.js

```
01.     var nome      = "José";
02.     var sobrenome = "Silva";
03.
04.     document.write( '<font color="#FF0000">', nome, ' ', sobrenome, '</font>' );
```

- pagina.htm

```
01. <script language="Javascript" src="arquivoexterno.js"></script>
```

13.2.3 Exercício 3

O script não é executado corretamente quando a letra w do método write estiver maiúscula. A mensagem "O objeto não dá suporte para a propriedade ou método" indica que o Javascript não reconheceu o comando.

13.2.4 Exercício 4

Letra d.

13.3 Capítulo IV

13.3.1 Exercício 1

Letra d ou 4ª opção.

13.3.2 Exercício 2

```
01. <script language="Javascript">
02.     if( confirm( 'Deseja trocar a cor de fundo da página? ' ) ) {
        03.         cor = prompt( 'Escolha uma cor dentre essas: BLUE, RED ou GREEN', '' );
        04.         document.bgColor = cor;
        05.     }
06.     else
07.         alert( 'Você escolheu não trocar a cor de fundo da página.' );
08. </script>
```

13.3.3 Exercício 3

```
01. <script language="Javascript">
02.     nome = prompt( 'Informe o seu nome:', '' );
03.     alert( 'Bem Vindo ' + nome + '!' );
04. </script>
```

13.3.4 Exercício 4

```
01. <script language="Javascript">
02.     var num1, num2, num3;
03.     num1 = eval( prompt( 'Informe o 1º número:', '' ) );
04.     num2 = eval( prompt( 'Informe o 2º número:', '' ) );
05.     num3 = eval( prompt( 'Informe o 3º número:', '' ) );
06.
07.     function media( n1, n2, n3 ) {
        08.         return ( n1 + n2 + n3 ) / 3;
        09.     }
10.
11.     document.write( media( num1, num2, num3 ) );
12. </script>
```

13.3.5 Exercício 5

Letra d ou 4ª opção

13.4 Capítulo V

13.4.1 Exercícios 1 e 2

```
01. <html>
02.     <head>
03.         <title>Formulários</title>
04.     </head>
05.     <body>
06.         <form name="f1" method="get">
07.             <h3>Formulário 1 − Login</h3><hr>
08.             Usuário:<br>
09.             <input type="text" name="user"><br>
10.             Senha:<br>
11.             <input type="password" name="senha" maxlength="10"><br>
12.             <input type="submit" name="enviar" value="Login">
13.         </form>
14.         <br>
15.         <form name="f2" method="get">
16.             <h3>Formulário 2 − Esqueceu sua senha?</h3><hr>
17.             Preencha o campo abaixo e a senha será enviada para você:<br>
18.             <input type="text" name="email" value="Digite aqui seu email"><br>
19.             <input type="submit" name="enviar" value="Enviar">
20.         </form>
21.         <!−−
22.             Neste ponto, vamos adicionar o código Javascript. Faremos isso aqui, porque,
23.             na tag <head> ainda não teríamos o código dos formulários executado. Assim,
24.             o Javascript não conseguiria devolver as informações solicitadas.
25.         −−>
26.         <script language="Javascript">
27.             document.write( "Número de formulários: ", document.forms.length, "<br>" );
28.             document.write( "Nome do 1º formulário: ", document.forms[0].name, "<br>" );
29.             document.write( "Quantidade de elementos do 1º formulário :", document.forms[0].e
30.             document.write( "Nome do 1º campo do 1º formulário: ", document.forms[0].elements
31.             document.write( "Nome do 2º campo do 1º formulário: ", document.forms[0].elements
32.             document.write( "Nome do 3º campo do 1º formulário: ", document.forms[0].elements
33.             document.write( "Valor do 3º campo do 1º formulário: ", document.forms[0].element
34.             document.write( "Nome do 2º formulário: ", document.forms[1].name, "<br>" );
35.             document.write( "Quantidade de elementos do 2º formulário :", document.forms[1].e
36.             document.write( "Nome do 1º campo do 2º formulário: ", document.forms[1].elements
37.             document.write( "Valor do 1º campo do 2º formulário: ", document.forms[1].element
38.             document.write( "Nome do 2º campo do 2º formulário: ", document.forms[1].elements
39.             document.write( "Valor do 2º campo do 2º formulário: ", document.forms[1].element
40.
41.             //resposta do exercício 2
42.             document.f1.user.value = "João";
43.         </script>
44.     </body>
45. </html>
```

13.4.2 Exercício 3

Letra d.

13.5 Capítulo VI

13.5.1 Exercício 1

Letra d.

13.5.2 Exercício 2

```
01. <html>
02.     <head>
03.         <title>Formulário</title>
04.         <script language="Javascript">
05.             function numerico() {
        06.                 if( !(event.keyCode >= 48 && event.keyCode <= 57 ) )
        07.                     return false;
        08.             }
09.
10.             function formataData() {
        11.                 //Criaremos esta variável para evitar a digitação de nomes longos
        12.                 var data = document.getElementById('data');
        13.
        14.                 if( data.value.length == 2 || data.value.length == 5 ) {
                15.                     /* Aqui vemos uma outra forma de se fazer acesso ao objet
                16.                        Procurem pela propriedade id na declaração da tag, e a
                17.                        Traduzindo: o valor do objeto, cujo id é igual a 'data
                18.                        ele mesmo +(concatenando) a barra de separação '/'
                19.                     */
                20.                     document.getElementById('datanascimento').value += "-";
                21.                 }
        22.             }
23.
24.             function checaCampos() {
        25.                 f = document.f1;
        26.
        27.                 if( f.nome.value == '' || f.cpf.value == '' || f.datanascimento.value
                28.                     alert( '***ERRO: Todos os campos devem ser preenchidos' )
                29.                     return false;
                30.                 }
        31.             }
32.         </script>
33.     </head>
34.     <body>
35.         <form name="f1" onSubmit="return checaCampos();">
36.             <h3>Formulário de Inscrição:</h3><hr>
37.             Nome:<br>
38.             <input type="text" name="nome"><br>
39.             CPF:<br>
40.             <input type="text" name="cpf" onKeyPress="return numerico();">(somente números)<b
41.             Data de Nascimento:<br>
42.             <input type="text" name="datanascimento" onKeyUp="formataData();" onKeyPress="ret
43.             (dd-mm/yyyy)<br><br>
44.             <input type="submit" name="enviar" value="Enviar">
45.         </form>
46.     </body>
47. </html>
```

13.5.3 Exercício 3

Letra d.

13.5.4 Exercício 4

```
01. <html>
02.     <head>
03.         <title>Formulário</title>
04.         <script language="Javascript">
05.             function numerico() {
        06.                 if( !(event.keyCode >= 48 && event.keyCode <= 57 ) )
        07.                     return false;
        08.             }
```

```
09.
10.             function formataCPF() {
        11.                 //Criaremos esta variável para evitar a digitação de nomes longos
        12.                 var auxCPF = document.getElementById('cpf');
        13.
        14.                 if( auxCPF.value.length == 3 || auxCPF.value.length == 7 )
        15.                     document.getElementById('cpf').value += ".";
        16.
        17.                 if( auxCPF.value.length == 11 )
        18.                     document.getElementById('cpf').value += "-";
        19.             }
20.         </script>
21.     </head>
22.     <body>
23.         <form name="f1">
24.             CPF:<input type="text" id="cpf" name="cpf" onKeyUp="formataCPF();" onKeyPress="re
25.                     maxlength="14">
26.             000.000.000-00
27.         </form>
28.     </body>
29. </html>
```

13.5.5 Exercício 5

Letra c.

13.6 Capítulo VII

13.6.1 Exercício 1

```
01. <script language="Javascript">
02.     var str = "Cachorro, gato, rato, leão, girafa, veado";
03.
04.     str = str.split( ", " );
05.
06.     for( i = 0; i < str.length; i++ )
07.         document.write( str[i], "<br>" );
08. </script>
```

13.6.2 Exercício 2

```
01. <script language="Javascript">
02.     var str1 = "CEF abre concurso para Advogados.";
03.     var str2 = "Abel pede demissão do Flamengo.";
04.     var str3 = "Bancos utilizam Java em operações financeiras online.";
05.     var str4 = "Este script deve ser concluído em 5 minutos!!!";
06.
07.     str1 = str1.slice( 12, 18 );
08.     str2 = str2.substr( 7, 2 );
09.     str3 = str3.substring( 16, 20 );
10.     str4 = str4.slice( 5, 11 );
11.
12.     str  = str1 + " " + str2 + " " + str3 + str4;
13.
14.     document.write( str.toUpperCase() );
15. </script>
```

13.6.3 Exercício 3

Letra c. Um erro será disparado porque "math.PI" está com a letra "m" minúscula.