Tutorial JavaScript

Por: Izaias Lisboa
ultima revisão: 13/02/2001
http://www.codefactory.com.br

Índice:

O que é JavaScript?

O JavaScript é uma linguagem de programação baseada na linguagem JAVA. É destinada para o uso em páginas Web(client-side) ou em servidores web (server-side). Neste tutorial irei falar somente sobre o JavaScript em client-side, ou seja, irei falar sobre os códigos que são inseridos nas páginas HTML sendo executadas pelo browser do cliente.

Esta linguagem permite ao programador ter acesso à elementos de uma página web, como imagens, elementos de um formulário links etc. Este objetos podem ser manipulados ou mudados via programação, pois o JavaScript permite também ao programador capturar eventos, como um click do mouse ou uma tecla pressionada de seu teclado. Isto lhe dá a capacidade de poder criar ações baseadas nas ações do usuário.

JavaScript é em si, uma grande linguagem de programação, que oferece ao programador web muitas recursos, os quais podem ser executados facilmente.

Que browsers suportam JavaScript?

A Netscape foi a primeira a introduzir o JavaScript em seu browser na versão 2.0. Já a Microsoft passou a incorporar algumas recursos apenas no Internet Explorer 3.0. Ao decorrer dos tempos outros browsers também incorporaram o JavaScript, e esta linguagem foi crescendo cada vez mais e estendendo seu suporte. logo os fabricantes de browsers também deveriam ir se atualizando, mas a Microsoft para não ficar para trás inventou um tal de JScript (http://msdn.microsoft.com/scripting/default.htm), que na verdade é a mesma linguagem que o JavaScript, porém com alguns recursos adicionais, o que gerou incompatibilidade da linguagem entre os browsers, como por exemplo a propriedade document.all, que traz grandes problemas para programadores em JavaScript, porque ela não existe no Netscape, somente no Internet Explorer...coisa do JScript e da Microsoft.

Iniciando...

Neste passo vamos inserir os códigos JavaScript em um documento HTML. Antes de você começar a mexer nos códigos do JavaScript, você precisa saber algumas regras e sintaxes básicas sobre a linguagem. Para usar JavaScript em suas páginas, você precisa saber primeiro onde coloca-los.
Para inserir seus scripts em um documento HTML você precisará coloca-los entre as tags <Script> e

</Script>, ficando mais ou menos assim:

```
<Script>
    comandos...
</Script>
```

Mas você pode também usar seus códigos em arquivos externos e usar o atributo SRC=, mas lembre-se que sempre que você colocar seus códigos em arquivos externos você precisará nomea-los com a extensão .JS exemplo :

<SCRIPT SRC=”meucodigo.js”></SCRIPT>

E onde colocar estas tags <Script> </Script> ? Bem, estas tags podem ser colocadas em qualquer lugar de seu documento HTML, ou, de acordo com sua necessidade. Se você necessita que seu código JavaScript seja interpretado antes que seu código HTML, você deve colocar as tags <Script></Script> dentro do cabeçalho das páginas HTML que são as tags <HEAD> e </HEAD>. Mas em apenas alguns casos isso é necessário, no começo de seu aprendizado, com certeza não será necessário colocar as tags dentro dos cabeçalhos HTML, mas é bom saber desde o início, que quando seu código JavaScript ali está(<head></head>), será lido “antes que tudo”.

A tag “SCRIPT”

A tag <Script> pode também especificar qual é a versão do JavaScript a ser usada, através do atributo “LANGUAGE=”. O que vai neste atributo pode ser apenas o comum “JavaScript”, “JavaScript1.1”, “JavaScript1.2”. A maioria dos browsers irão ignorar qualquer Script que ele não reconheça ou não suporte. Isto permite à você separar os scripts para diferentes versões de browsers. esta lista abaixo irá mostrar as versões do JavaScript suportadas por diferentes versões do Netscape.

- JavaScript - Netscape 2.0
- JavaScript1.1 - Netscape 3.0
- JavaScript1.2 - Netscape 4.0 -4.05
- JavaScript1.3 - Netscape 4.06 -4.07x
- JavaScript 1.4 - A Netscape não referencia nenhum navegador para esta versão.
- JavaScript 1.5 - Netscape 6.0 - Mozilla (open source browser)

Então quando um browser do Netscape em sua versão 2.0 ler o código

```
<SCRIPT LANGUAGE=”JavaScript1.2”>
  comandos...
</SCRIPT>
```

ele irá ignorara tudo, pois não entenderá nem suportará aquela versão do JavaScript, portanto o Netscape 4.0 ou superior irá executar o código normalmente. Para ver mais sobre este detalhe, use o ’Netscapes JavaScript Reference <http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm> (documentação em idioma inglês)..

A sintaxe do JavaScript, é similar a sintaxe do C/C++ e do JAVA, para uma referência maior sobre as sintaxes visite ’Netscapes JavaScript Reference <http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm> ou Microsoft´s JScript Reference <http://www.microsoft.com/JScript/us/Jslang/Jstoc.htm>. Lá você poderá encontrar um artigo muito extenso e rico para leitura, cheio de coisas que você poderá utilizar muito, e você pode também fazer o download desses artigos no formato .exe, que eu disponibilizo também em <http://www.codefactory.com.br/>.

Introdução ao ECMAScript

ECMAScript provêm de: European Computer Manufacturers Association.
Em 1996 os desenvolvedores web começaram a reclamar que a Netscape estava indo em uma direção com o JavaScript e a Microsoft se encaminhava em uma direção de certa forma compatível, mas diferente com o JScript. Ninguém gosta de ter que codificar páginas que manipulem dialetos diferentes do JavaScript, ou que

tenham seu código funcionando em um navegador, mas não em outro.
Em resumo, os desenvolvedores queriam um padrão. Assim a Netscape foi até um corpo internacional de padrões chamado ECMA e submeteu a eles a especificação da linguagem JavaScript, enquanto a Microsoft apresentava seus próprios comentários e sugestões. A ECMA fez o que costumam fazer os corpos de padrões e, em junho de 1997, produziu um padrão chamado ECMA-262 (também conhecido como ECMAScript).
Veja abaixo uma pequena tabela com versões e suas compatibilidades no JavaScript.

Versão do JavaScript	Relacionamento com a versão do ECMA
JavaScript 1.1	ECMA-262, Edição 1 é baseada no JavaScript 1.1.
JavaScript 1.2	O JavaScript 1.2 não é completamente compatível com o ECMA-262, Edição 1,e uma das razões disso, se dá ao fato de que a Netscape desenvolveu recursos adicionais no JavaScript 1.2 que não foram considerados pelo ECMA-262.
JavaScript 1.3	JavaScript 1.3 é completamente compatível com o ECMA-262, Edição 1.
JavaScript 1.4	JavaScript 1. 4 é completamente compatível com o ECMA-262, Edição 1.
JavaScript 1.5	JavaScript 1.5 é completamente compatível com o ECMA-262, Edição 3.

Se estiver interessado em ler a especificação completa do ECMAScript, poderá fazer o download no formato PDF, usando este link <http://developer.netscape.com/docs/javascript/e262-pdf.pdf>.
(http://developer.netscape.com/docs/javascript/e262-pdf.pdf)
O fato é que, desde que você escreva seu código compatível como ECMAScript, ele deverá funcionar muito bem no Internet Explorer, e provavelmente no Netscape Navigator versões 4.0 e superiores, porém é recomendável testar o código em diferentes navegadores e plataformas.

Escrevendo no documento - Seu primeiro script (document.write)

A função Document.Write do Java Script, é muito importante para nossa programação, sendo muito simples de usar. A sintaxe básica é:

```
<Script>
document.write (“Hello world”);
</Script>
```

O ideal, durante este tutorial, é estar testando todos os scripts mostrados, para que assim, você entenda o que irá acontecer quando usarmos cada função. Apesar da simples facilidade de escrever no documento, há alguns pequenos enganos que não devemos cometer para que a programação funcione perfeitamente. Para detalhar este comentário vamos fazer o seguinte exemplo :

Teste o código abaixo em seu browser :

```
<!-- ** código 1 ** - - >

<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0 Transitional//EN”>
<html>
<head>
<title>Untitled</title>
</head>
<body>
<Script Language=JavaScript>
var browserName
browserName = navigator.appName;
document.write “seu browser é “ + browserName;
</Script>
</body>
</html>
```

Não funcionou nada não é mesmo ? tudo bem, agora teste este outro código abaixo :

```
<!-- ** código 2 ** - - >
```

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0 Transitional//EN”>
<html>
<head>
<title>Untitled</title>
</head>
<body>
<Script Language=JavaScript>
var browserName
browserName = navigator.appName;
document.write (“seu browser é “ + browserName);
</Script>
</body>
</html>
```

Bom, como podemos observar no código 1 nossa string (texto) e nossa variável, estão fora de um parênteses, ao contrário do código 2, uma diferença de código tão simples que faz com que um Script funcione e o outro não, então é sempre bom lembrar desta pequena dica, use sempre parênteses em seu document.write do Java Script, mesmo se o conteúdo à ser exibido for uma string ou uma variável, sempre use os parênteses !

Outra coisa que é bom ser lembrada, é que o ponto e vírgula deve ser fora de seu parênteses, ou seja, indicando que é um final de uma instrução. Para Ter melhores noções, e uma breve dica sobre o ponto e vírgula no JavaScript clique aqui.

Operadores

Operadores lógicos

São operadores que normalmente são utilizados em comandos condicionais, assim como: IF , FOR e WHILE. Estes comandos condicionais serão vistos mais adiante em nosso tutorial.

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

Operadores Matemáticos

- **+**
  este operador serve para adição de valores e ao mesmo tempo, no JavaScript, este operador pode ser usado na concatenação de strings. (concatenação = junção ou união) exemplo:
  “Izaias” + “Lisboa” //retornaria Izaias Lisboa

- **-**
  utilizado na subtração de valores

- ***
  utilizado em multiplicação de valores

- **/**
  utilizado para divisão de valores

- **%**
  obtém o resto de uma divisão:
  Exemplo: 150 % 13 retornará 7
  7 % 3 retornará 1

Expressões Simples com operadores

+=
concatena /adiciona ao string/valor já existente. Ou seja:
x += y é o mesmo que x = x + y, da mesma forma podem ser utilizados: -= , *= , /= ou %=

A maioria das sintaxes e expressões usadas em C/C++ e JAVA, são suportadas pelo JavaScript para exemplificar vou descrever expressões válidas:

total += 4;
i++;
Na primeira linha do exemplo acima, significa que quando temos numa variável já declarada com um **var** um valor, podemos usar += para adicionar um valor à esta variável, ou seja, quando dizemos:
total +=4;
Estaríamos dizendo para o compilador:
Tome o valor de total e acrescente mais quatro à este valor.
Logo se total for igual à 3, então total passará a valer 7, pois três acrescido de 4 resultam em 7.
O mesmo é válido para outras expressões assim como:

```
<Script>
subtotal = 10;
total += subtotal;  // tome o valor de total a acrescente á ele subtotal
</Script>
```

Na segunda linha do exemplo acima usamos i++; o que significa que estamos pedindo para que tome o valor de i e acrescente um (1) à ele. Se usarmos sempre esta "dica" de dizer "tome o valor de X e acrescente Y à ele", iremos intender qualquer código perfeitamente. È realmente uma dica muito útil, sempre que você ler estas expressões, leia-as desta maneira, e compreenderás muito facilmente.

Variáveis & Constantes

Esta é uma importante parte para se aprender. No JavaScript use a palavra var para declarar suas variáveis, podendo ela ser de qualquer tamanho, conter números, dígitos e underscores/underlines ('_').

Uma variável pode ser chamada de um identificador. Um identificador JavaScript deve iniciar com uma letra, underscore (_), ou sinal de dólar ($); caracteres subseqüentes podem ser também dígitos de zero à nove (0-9). Por causa do JavaScript ser case sensitive, letras incluindo os caracteres "A" até "Z" (maiúsculo) e caracteres "a" até "z" (minúsculo), também pode ser um identificador.

Inciando no JavaScript com JavaScript 1.5, você pode usar letras ISO 8859-1 ou letras Unicode assim como å e ü, para serem um identificador. Você pode também usar as \uXXXX seqüências de Unicode escape listadas na página 34 de caracteres e identificadores do documento oficial do JavaScript versão 1.5

Exemplo de declaração de um identificador:

var calculo_diferencial;

Note que o JavaScript é case-sensitive nomes como conta_total e Conta_total referem-se à diferentes variáveis. Variáveis declaradas fora de uma função, ficam sendo como globais, podendo estas serem acessadas por qualquer lugar do Script ou por qualquer função. Observe que a palavra var é opcional fora das funções, mas é requerida se você quer atribuir um valor ao uma variável que não está definida, portanto, é recomendável, para que não se obtenha erros estranhos, declarar-se sempre todas as variáveis. Por exemplo:

- Se a variável que não teve um valor atribuído á ela foi declarada sem o ‘var’, a avaliação dela retorna em um runtime error (erro de execução).
- Se a variável não atribuída, foi declarada com um var, a avaliação dela resulta em um valor, ou NaN em contexto numérico, portanto não mostra nenhuma mensagem de erro.

O seguinte código mostra um exemplo de variáveis que não tiveram nenhum valor atribuído à elas e foram ou não declaradas com um var.

```
function f1() {
  return y - 2;
}
f1() //Causa um  runtime error
function f2() {
  return var y - 2;
}
f2() //retorna NaN, e não cause um runtime error
```

As **constantes**, têm a mesma função das variáveis, porém um valore de uma constante não pode ser alterado, apenas isso que diferencia-a de uma variável. Para se declarar constantes usa-se :

```
const maximo=10;
```

Nota: Você não pode declarar uma constante dentro do escopo de uma função, e também não pode declarar uma constante com o nome de uma variável. Tendo isso em mente veja os códigos abaixo que resultarão em erro:

```
//ISTO IRÁ CAUSAR UM ERRO!
function f{};
const f = 5;


//ISTO IRÁ CAUSAR UM ERRO TAMBÉM !
function f{
const g=5;
var g;
}
```

ATENÇÂO: você só poderá usar constantes no JavaScript 1.5, pois isso é uma inovação desta versão do JavaScript. Caso tente usar o comando **const** para declarar uma constante usando JavaScript 1.4 e anteriores, ocorrerá uma mensagem de erro de sintaxe.

Usando Constantes de Strings:

As constantes de strings podem ser usadas tanto com o nosso genérico aspas(“) quanto com o apóstrofo(‘). Você também pode usar as barras invertidas para separar os aspas para que o JavaScript não entenda bobagens...como por exemplo :

```
<Script>
var Izaias = “ \”texto\” “
document.write(Izaias); // o resultado seria “texto”
</Script>
```

Este Script acima nos dá um exemplo de como atribuir um valor para uma variável, sendo que esse valor contém caracteres que são reservados, assim como o aspas no exemplo acima, ou mesmo com um apóstrofo no exemplo abaixo:

```
<Script>
var Izaias = ‘ \’texto\’ ‘
document.write(Izaias); // o resultado seria ‘texto’
</Script>
```

Esta sintaxe, como disse anteriormente é semelhante à sintaxe do Java, C/C++, programadores que migram desta linguagem, não encontrarão maiores dificuldades.

Conversões de tipos de dados

JavaScript é uma linguagem de tipos dinâmicos. Isso quer dizer que você não têm que especificar o tipo de dado de uma variável quando você a declara, além disso tipos de dados podem ser convertidos automaticamente como necessário durante a axecução do script. Assim, por exemplo, você pode definir variáveis como segue:

var answer = 42

E mais tarde, você pode atribuir à mesma variável um valor de uma string, por exemplo:

answer = "Thanks for all the fish..."

Pelo motivo do JavaScript ser uma linguagem de tipos dinâmico, esta atribuição não causa uma mensagem de erro.

Em expressões envolvendo valores numéricos e strings com o operador +, o JavaScript converte valores numéricos para strings. Por exemplo, considere as seguintes expressões.

x = "The answer is " + 42 // retona "The answer is 42"

y = 42 + " is the answer" // retorna "42 is the answer"

Em expressões envolvendo outros operadores, o JavaScript não converte valores numéricospara strings. Por exemplo:

"37" - 7 // retorna 30

"37" + 7 // retorna 377

parseInt e parseFloat

Ambos Servem para converter strings, o parseInt para converter strings para Inteira e o Float como o nome já diz, serve para converter para números de Ponto Flutuante.

```
<Script Language=“JavaScript”>
var isNav, isIE
if (parseInt(navigator.appVersion) >= 4){
        if (navigator.appName == “Netscape”) {
                isNav = true
        } else {
                isIE = true
        }
} else {
        document.write (‘Desculpe, mas seu browser é muito antigo!‘,’)
```

```
        + (‘faça download de uma versão atualizada’);
}
if (isNav) document.write(‘netscape’);
if (isIE) document.write(‘internet explorer’);
</Script>
```

No exemplo acima convertemos a String da versão do browser para um número inteiro comparando se a versão é maior ou igual a 4. Caso você queira testar o código sem usar a função parseInt o código não funcionará.

Comandos estruturados

Comando IF

O comando IF todo mundo deve saber, mas vou falar para os leigos que ele é usado simplesmente para testar uma condição. Exemplo :

```
<Script>
var i=10
if (i==10) {
Document.Write(“é dez”);
}else{
Document.Write(“não é dez”);
}
</Script>
```

Acima o comando IF avaliou a condição, se a variável i fosse igual a dez ele escreveria no documento que é dez, senão ele escreveria que não é dez. O mais importante neste exemplo, é observar como é feita a regra das chaves (em azul) que indicam o inicio e o término do comando estruturado. È importante dizer que no comando IF (no JavaScript) a condição deve sempre estar entre parênteses seguindo a sintaxe abaixo :

```
if ( condição_vai_aqui )
```

As chaves indicam o início do comando estruturado, assim como nos comandos for e while. Elas são necessárias apenas quando temos mais de um comando dentro de uma condição, exemplo:

```
If(a==b)document.write “A é igual à B”;
```

Este exemplo acima é válido pois só têm um comando, ou só uma ação, que é o document.write, agora observe o código abaixo:

```
If(a==b)document.write “A é igual a B”; document.write “você acertou !“;
```

Este exemplo está incorreto e não deve funcionar como desejado, pois têm dois comandos dentro de um if que não têm chaves. Se quiséssemos fazer esta verificação acima com as mesmas ações deveríamos usar as chaves como no exemplo abaixo:

```
If(a==b){
    document.write “A é igual a B”;
    document.write “ você acertou !“;
}
```

E assim funcionará perfeitamente como desejado.

Também podemos usar outro modo que funciona semelhantemente ao if. É um pouco mais complexo para se entender, e ao início de aprendizado é recomendável usar somente o if sózinho, mas para que voc~e conheça essa sintaxe vamos vê-la agora:

var teste = (a==b)**?** “A é igual à B “ **:** “A não é igual à B “;

A variável teste retornará uma string “A é igual à B “ se for verdadeira a condição entre parênteses, e retornará uma string “A não é igual à B “ se a condição entre parênteses for falsa.

Comando FOR

O comando para no JavaScript é, como em todas as outras linguagens indispensável, pois ele é quem faz um trabalho simples que se fosse feito manualmente daria um grande trabalho. Para exemplificar vamos supor que você tenha tenha que escrever em seu código HTML os números de 1 até 20. Para que você vai escreve-los um a um se você pode usar um simples comando em JavaScript para fazer isto ?

A sintaxe para isso é a seguinte :

for (sua_variavel = inicio_do_laço ; sua_variavel <= fim_do_laço ; sua_variavel++)

Sintaxe em uso :

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0 Transitional//EN”>
<html>
<head>
<title>Titulo</title>
</head>
<body>
<Script Language=”JavaScript”>
for (i=1;i<=20;i++)
document.write (i + “<BR>”);
</script>
</body>
</html>
```

Explicação passo-à-passo do código acima:

for
Este é o comando formalmente nomeado de “Para”

i=1;
quando fazemos esta atribuição, estamos dizendo que nossa contagem deve começar do 1(um).

i<=20;
Isto é ATÉ onde vai sua contagem, ou seja, o fim da contagem, o número de voltas que o laço vai dar.

i++
Isto significa que ao final do laço (volta) será incrementado 1 (um) para “ i “ no final de cada volta.

Lendo este código pela lógica de programação, ficaria assim :

```
Para i começando por 1 até 20 incrementando 1
    Escreva i + “<BR>”
Fim Para
```

Sendo que, tudo isso vai entre parênteses, e não devemos jamais esquecer que no Java Script é tudo CaSe

SeNSeTiVe, ou seja, tudo que é minúsculo é minúsculo e não há exceções , portanto se você quiser escrever FOR (em maiúsculo) em vez de for (em minúsculo, o correto), é 100 % de chances de um código não funcional.

Comando While

O comando while é semelhante ao comando for, porém na maioria das vezes o while é aplicado quando não podemos determinar a quantidade de voltas que nosso laço vai ter, um bom exemplo que desenvolvi migrando do PHP está aí em baixo:

```
<Script Language=”JavaScript”>
var i=1;
while(i <= 10){
	document.write(i++ +’<br>’); //escrevendo e ao mesmo tempo adicionando um para a variável i
}
</Script>
```

O comando switch

O comando switch do JavaScript1.2 nada mais é do que um case, assim como no C ou no PHP, ou até podendo se comparar com um simples case do VB. Caso você trabalhe com C, PHP ou outras linguagens, você provavelmente já sabe o que o comando faz. Mas lembre-se que este comando no JavaScript serve apenas para o JavaScript1.2, ou seja, serve apenas para Netscape e Internet Explorer nas suas versões 4.X.

Sintaxe:

```
switch (variavel_de_controle) {

	opção1 : comandos ;

	break;

	opção2 : comandos ;

	break;

	default : comandos;

}
```

vamos para um exemplo prático mas primeiramente veja o código abaixo:

```
	<Script Language=JavaScript1.2>
	var teste = prompt(“digite um valor de 1 até 5”);
	switch(teste){
		case ‘1’: alert(‘seu valor foi 1’);break;
		case ‘2’: alert(‘seu valor foi 2’); break;
		case ‘3’: alert(‘seu valor foi 3’); break;
		case ‘4’: alert(‘seu valor foi 4’); break;
		case ‘5’: alert(‘seu valor foi 5’); break;
		default: alert(‘seu valor nao foi nenhum valor entre 1 e 5’);
	}
</Script>
```

Objetos

Primeiro, vamos pensar nos objetos. Um objeto é alguma coisa. Um gato um carro, uma bicicleta são todos objetos no mundo físico. Para o JavaScript, existem objetos com que ele lida em navegadores da Web, como janelas, formulários e os elementos do formulário, como botões e caixas de seleção.

Como é possível haver mais de um gato, ou mais de uma janela, faz sentido dar nomes aos objetos.

Embora você pudesse se referir a seus bichinhos como Gato1 e Gato2. No JavaScript você também pode fazer isso !. Como por exemplo window[0], forms[1]. Uma pequena dica que gostaria de mencionar é que você sempre utilize o nome real dos objetos em vez de utilizar o exemplo acima de window[0], forms[1] ...Gato1, Gato2 (no mundo real), pois assim facilitará mais sua vida como um programador em JavaScript. Para não complicar o que eu estou dizendo aqui vou fazer dois exemplos. O primeiro utilizando seus "apelidos" e o segundo utilizando os nomes reais dos objetos.

```
<form name="preenchimento" method="post">
<input type="text" name="nome">
<input type="button" name="botaoenvia">
</form>
<Script Language="JavaScript">
minhavar = document.forms[0].nome.value ;
</Script>
```

Neste exemplo utilizamos o objeto forms em vez do nome do objeto que criamos agora veremos o exemplo abaixo, desta vez utilizando o nome real dos objetos

```
<form name="preenchimento" method="post">
<input type="text" name="nome">
<input type="button" name="botaoenvia">
</form>
<Script Language="JavaScript">
minhavar = document.preenchimento.nome.value;
</Script>
```

Agora sabemos que há duas formas de se tratar objetos. Uma delas é chamando por um nome ordenado de uma propriedade do documento e a outra é, chamando pelo próprio nome como no segundo exemplo, o que é mais fácil e recomendado, não somente por mim, mas por livros e outros programadores.

Propriedades

Os objetos têm propriedades. Um gato tem pele, o carro têm uma porta, e a bicicleta têm rodas. No mundo do JavaScript, uma janela tem título, e um formulário têm uma caixa de seleção. As propriedades podem modificar os objetos, e a mesma propriedade pode se aplicar a objetos completamente diferentes. Digamos que você tem uma propriedade chamada vazia. É certo usar vazia onde ela for aplicável; assim, você poderia dizer que a barriga do gato está vazia. Observe que a roda da bicicleta e a porta do carro são apenas propriedades; elas mesmas também são objetos, os quais podem ter suas próprias propriedades.

Portanto objetos podem ser subObjetos.

Métodos

As ações que os objetos podem realizar são chamadas de métodos. Gatos miam, carros se correm e bicicletas andam...Os Objetos do JavaScript também tem métodos:

botões: click(), janelas open(), e texto pode ser selected(),. Os parênteses significam que estamos fazendo referência à um método, e não a uma propriedade.

Dica:
talvez ajuda a imaginar os objetos e as propriedades como nomes (substantivos) e seus métodos como verbos. os primeiros são coisas, enquanto os métodos são ações que essas coisas podem realizar, ou então que podem ser realizadas sobre eles.

Juntando as peças

É possível reunir objetos, propriedades e métodos para conseguir uma descrição melhor de um objeto, ou ainda para descrever um processo. Em JavaScript estes itens são separados por pontos. Isso denomina-se sintaxe de ponto. Aqui estão alguns exemplos de objetos e suas propriedades, representados dessa maneira:

bicicleta.rodas
gato.pata.dianteira.esquerda
computador.disco.flexivel
documento.imagens.nome
janela.status

E aqui estão alguns exemplos de objetos e métodos escritos em sintaxe de ponto:

gato.miar()
documento.escrever()
formulários.elementos.rádio.clique()

Abaixo, segue uma tabela que demonstra os principais eventos

Eventos

O browser interpreta o JavaScript esperando que eventos aconteçam, eles podem vir quando a página termina de ser lida ou quando o usuário move o mouse sobre um link ou clica em um botão, e são inseridos dentro de nossas tags HTML. A maioria dos eventos é precedido da palavra ON, como por exemplo onMouseOver.

Mas os eventos do JavaScript nem sempre permitem que você possa trabalhar com qualquer tag do HTML, como por exemplo você não pode colocar um evento OnClick em uma tag <IMG do HTML...isto nem sempre é observado pelo programadores JavaScript, bom para ficar claro vou colocar abaixo alguns dos eventos mais usados no JavaScript e citar o que eles fazem e também em que tag HTML que podemos usa-lo. Para ter uma referência ainda melhor use o site da Netscape

A Tag	***Possui o evento***	***Que é chamado quando...***
<A>	OnClick	O usuário clica no link.
	OnMouseOver	O ponteiro do mouse é colocado em cima do link.
	onMouseOut	Quando o mouse sai fora do link.
<AREA>	onMouseOver	O ponteiro do mouse é colocado em cima da área de mapeamento da imagem.
	onMouseOut	O ponteiro do mouse sai fora da área de mapeamento da imagem.
<BODY>	onBlur	A janela ou frame contendo esta página perde o fócus.
	onFocus	Ao contrário do onBlur a página recebe o fócus.
	onLoad	Quando a página foi terminada de ler...
	onUnload	Quando o usuário sai da página.
<FORM>	onReset	Quando o botão RESET é pressionado.

	onSubmit	Quando o Botão submit é pressionado.
<IMG>	onAbort	A leitura da imagem tem sido parada por uma ação do usuário.
	onLoad	Quando a leitura da imagem é terminada.
	onError	Quando um erro ocorreu enquanto a imagem foi lida.
<INPUT> com TYPE="BUTTON" ou "CHECKBOX" ou "RADIO" ou "RESET"	onClick	Quando é dado um clique de mouse em um elemento do formulário.
<INPUT> com TYPE="TEXT" ou <TEXTAREA>	onBlur	Quando um elemento do formulário perde o fócus.
	onChange	Quando um texto de um elemento do formulário é atualizado.
	onFocus	Quando um elemento do formulário recebe o fócus.
	onSelect	Quando o usuário seleciona tudo ou apenas uma parte de um texto de um elemento do formulário.
<INPUT> com TYPE="SELECT"	onBlur	Quando o elemento do formulário perde o fócus
	onChange	Quando o usuário muda o item selecionado.
	onClick	Quando um usuário clica em um item
	onFocus	Quando um elemento do formulário recebe o fócus

Manipulando strings

substring

Esta é uma função de manipulação de strings, e serve para que você possa obter um trecho ou uma pequena parte de uma grande string, isto sendo informado por intervalos de números, os quais você pode contar os caracteres. veja meu exemplo :

```
<Script>
var frase = “Alface é bom para saúde”;
var parte = frase.substring(0,2);
document.write(parte);
</Script>
```

Como podemos ver a função substring arrancou uma parte da minha frase, esse é o objetivo desta função, como vimos onde eu tinha “Alface é bom para saúde” eu fiquei apenas com a palavra Al. Isso aconteceu porque eu mandei que o Script começasse a contar à partir do zero, ou seja, contar à partir da primeira string encontrada (da esquerda para a direita), e terminar quando chegar o total da conta em 2 caracteres. Não entendeu ? ...vamos para mais uma explicação como outro exemplo.

```
<Script>
var frase = “Alface é bom para saúde”;
var parte = frase.substring(1,2);
document.write(parte); //retornará L
</Script>
```

Agora, depois de testarmos este Script, vimos que foi retornado apenas o caracter “l”(L) , pois agora mandamos o Script começar a contar depois que ele visse o primeiro caracter (da esquerda para a direita), e terminar sua contagem, assim que visse o segundo caracter. Acho que agora deu para entender e ter uma noção básica desta função.

substr

Idêntico à função substring.
Função: Retorna o conteúdo de uma string dentro de um intervalo dado.
Sintaxe: string.substr(inicio_contagem, fim_contagem);
Exemplo:

```
<script Language=”JavaScript” Type=”text/javaScript”>
        aString = “Izaias”;
        document.write (  aString.substr(0,1)  ); //retornará ‘I’
</script>
```

length

Função: Retorna o número de caracteres que um string têm.
Sintaxe: string.length;
Exemplo:

```
<script Language=”JavaScript” Type=”text/javaScript”>
        aString = “Izaias”;
        document.write (  aString.length ); //retornará 6, pois o nome Izaias contém 6 caracteres
</script>
```

charAt

Função: Retorna o caracter da posição especificada.
Sintaxe: string.charAt(numero_da_posicao);
Exemplo:

```
<script Language="JavaScript" Type="text/javaScript">
        aString = "Lisboa";
        document.write (  aString.charAt(4) ); //retornará 'o'
</script>
```

indexOf

Função: Retorna a posição do primeiro caracter encontrado em uma string.
Sintaxe: string.indexOf(“parte_da_string”);
Exemplo:

```
<script Language=”JavaScript” Type=”text/javaScript”>
        aString = “Izaias”;
        document.write (aString.indexOf(“z”)); //retornará 1, pois I = 0 e z = 1, ou seja, começa do zero
</script>
```

Exemplo 2:

```
<script Language=”JavaScript” Type=”text/JavaScript”>
        aString = “Izaias”;
        document.write (aString.indexOf(“x”)); //retornará -1 pois x não existe no nome Izaias
</script>
```

Exemplo 3:

```
<script Language=”JavaScript” Type=”text/JavaScript”>
    aString = “Izaias”;
    document.write (aString.indexOf(“a”));      //retornará 2, pois I=0, z=1,  e a=2,
                                                //e o primeiro ‘a’ encontrado será o 2º caracter
                                                //encontrado iniciando pelo  zero...
    </script>
```

lastIndexOf

Função: Quase igual ao indexOf, porém com o objetivo de trazer a posição do último caracter encontrado, e não do primeiro assim como no indexOf.
Sintaxe: string.lastIndexOf(“parte_da_string”);
Exemplo:

```
<script Language=”JavaScript” Type=”text/JavaScript”>
    aString = “Izaias”;
    document.write (aString. lastIndexOf (“a”));    //retornará 4, pois estará retornando a posição do
                                                    //do último caracter encontrado.
</script>
```

toUpperCase

Função: Muda a string para maiúsculo
Sintaxe: string.toUpperCase();
Exemplo:

```
    <script language=”JavaScript” Type=”Text/JavaScript”>
    minhaString= “Izaias”;
    document.write ( minhaString.toUpperCase());
    </script>
```

toLowerCase

Função: Muda a string para minúsculo.
Sintaxe: string.toLoewrCase();
Exemplo:

```
    <script language=”JavaScript” Type=”Text/JavaScript”>
    minhaString= “Izaias”;
    document.write ( minhaString.toLowerCase());
    </script>
```

escape

Função: Retorna o valor ASCII da string.
Pode ser usado como uma função de codificação de URL (URLEncode).
Sintaxe: escape(“string”);
Exemplo:

```
    <script language=”JavaScript” Type=”Text/JavaScript”>
    document.write ( escape("Iza i & as"));   //Retornará Iza%20i%20%26%20as
    </script>
```

unescape

Função: Retorna um caracter á partir de seu código ASCII.
Ao contrário do escape
Sintaxe: unescape(“string”);
Exemplo:

```
    <script language=”JavaScript” Type=”Text/JavaScript”>
```

```
document.write ( unescape("Iza%20i%20%26%20as ")); //Retornará Iza i & as
</script>
```

Janelas

O JavaScript oferece muitas rotinas para se manipular janelas. Vamos ver algumas abaixo:

Abrindo uma nova Janela

Abrir uma nova janela no JavaScript é extremamente fácil. Veja o exemplo abaixo:

```
<Script Language=”JavaScript”>
    window.open(“Janela_nome”,”target”,”opções”);
</Script>
```

Onde:
janela_nome é o nome da página que você deseja abrir em uma nova janela.
target é o alvo desta janela, ou seu nome, caso use o target poderá ser:
_blank
_parent
_self
_top

Um detalhe interessante sobre janelas no JavaScript, e ainda pouco divulgado é o uso do “_self”, pois ele abrirá um pop-up na mesma janela, ou seja, ele transformará sua janela comum em um pop-up. Por exemplo, se quiséssemos abrir a página inicial de nosso site já em um pop-up, e não em uma janela padrão, poderíamos usar esta opção de target, porém é importante informar que isso só funciona no JavaScript, ou seja, só funciona no Netscape até as versões 4.7.6. Se usarmos esse método de target no IE, a página até se abrirá na mesma janela, porém você verá que as opções de toolbar, location etc. não farão efeito, ou seja, a página se abrirá na mesma janela porém não ficará em formato físico de um pop-up. Já os outros métodos são suportados normalmente pelo IE, assim como: o _top (para se abrir a janela no topo do documento), o _blank (para se abrir numa outra página) , e assim os restantes.
Continuando...

opções são os atributos da janela, alguns deles podem ser:
toolbar - Quando setado para “yes” ativa a barra de ferramentas que contém os botos Back, Stop, entre outros
location - Quando setado para “yes” ativa Abre a barra de location do browser
directories - Quando setado para “yes” ativa os botões What's New, Handbook, etc.
status - Quando setado para “yes” ativa status do browser no rodapé da janela
scrollbars - Quando setado para “yes” ativa o scroll do browser, na vertical e horizontal
menubar - Quando setado para “yes” ativa os menus de aplicativo, por exemplo: File, Edit, Help, etc.
resizable - Quando setado para “yes” permite o usuário alterar o tamanho da janela.
fullscreen - Quando setado para “yes” abre a tela cheia (somente IE).
width - Atribui a largura da janela que será aberta, usando coordenadas em pixels
height - Atribui a altura da janela que será aberta, usando coordenadas em pixels
top - Atribui a distância da nova janela em relação ao topo da tela usando pixels como coordenadas.
left - Atribui a distância da nova janela em relação ao lado esquerdo da tela em pixels.

Usando a função moveTo para mover janelas

Quando abrimos uma janela, nas opções de abertura, podemos determinar a distância que nossa nova janela terá em relação ao topo da tela e a distância que a nova janela terá em relação ao lado esquerdo da

tela. Contudo ainda temos mais uma opção, que é usar a função moveTo() para mover esta janela para um lugar específico da tela, usando as coordenadas de pixels.

Sintaxe: objeto.moveTo (distancia_do_canto_esquerdo-da_tela , distancia_do_canto_direito-da_tela,);

Exemplo:

```
<script>
cyberpunk = window.open(“minhaPagina.htm”, “_blank”, “width=290,height=250” );
cyberpunk.moveTo(0,0);
</script>
```

A propriedade opener

A propriedade opener pode ser usada quando desejarmos referenciar a janela pai, depois que o pop-up (ou janela) foi aberto.
Sintaxe: opener.comandos...
Exemplo:
Este exemplo será demonstrado usando-se 2 páginas.
Código da primeira página: 1.htm

```
<html>
<head>
	<title>Untitled</title>
</head>
<body>
<script>
   remote =
window.open('2.htm','promocao','toolbar=no,location=no,directories=no,status=no,menubar=no,scrollbars
=yes,resizable=no,menubar=no,width=290,height=250')
</script>
</body>
</html>
```

Código da segunda página: 2.htm

```
<html>
<head>
	<title>Untitled</title>
</head>
<body>
<script>
function linkar(){
	opener.document.location.href="http://www.encode.com.br";
}
</script>
<a href="" onclick="linkar(); return false;">Link Teste</a>
</body>
</html>
```

Como percebemos olhando os códigos acima, a página 1.htm através de seu script, abre uma outra janela, chamada 2.htm. A página 2.htm contém um link que chamamos de link teste, que chama a função linkar, e lá vemos finalmente o uso da propriedae opener, que indica a a bertura de uma outra url dentro da primeira janela (página 1.htm). Isso demonstra como usarmos comandos de uma janela para outra, assim desta maneira, poderíamos manipular muitos outros dados assim como formulários e diversos outros dados que estão dentro da página pai - que no nosso exemplo foi a página 1.htm - através da propriedade opener.

Fechando uma janela

Para fechar uma janela pode-se usar o método close(), atribuindo-se à propriedade window.

Exemplo:

```
<script>
        window.close();
</script>
```

Se a janela que possuir esse comando, for um “pop-up”, ou seja, se for uma janela “filha” de outra, então quando chamar-mos a função de algum modo, a janela se fechará automaticamente. Caso a janela que conter este comando for uma janela comum, não aberta através de outra, por motivos de segurança, aparecerá uma alerta interna do browser perguntando
A novidade neste método é que nos navegadores mais recentes (IE 5.5 e Netscape 6), pode-se fechar uma janela “pai“ automaticamente, sem que seja exibida uma mensagem de alerta interna do navegador.
Há ainda algumas propriedades nestes navegadores mais recentes que permitem manipular muitas propriedades de uma janela, como cores do título da janela, cores do sistema da janela, mudar botões de maximizar e minimizar de uma janela e enfim fazer muita coisa diferente, mas isso é possível somente aos navegadores com suporte a tecnologia DOM. Se quiser obter mais informações sobre o DOM use este link <http://www.codefactory.com.br/>

Arrays

Array, são também chamadas de vetores. Um vetor é usado em situações quando precisamos de que uma única variável possua vários valores diferentes e que estes valores estejam preservados, guardados dentro da variável, e quando precisamos de usar isto pegamos desta variável, de uma forma simples.

Vou citar um exemplo:

```
<Script Type=”text/javascript” Language=“JavaScript”>
var nome = new Array() //declarando as arrays
var sobrenome = new Array()
var link = new Array()
var descricao = new Array()

/* Cuidado ! a palavra "Array" é mais uma das Case sensitive do JavaScript */

/* Atribuindo os valores */
nome[0] = "Izaias"
nome[1] = "Linus"
nome[2] = "Patrick"

sobrenome[0] = "Lisboa"
sobrenome[1] = "Tourvals"
sobrenome[2] = "Volkerding"

link[0] = "http://izaiaslisboa.cjb.net"
link[1] = "http://linux.org"
link[2] = "http://www.slackware.org"

descricao[0] = "Site para programadores Web de Izaias Lisboa"
descricao[1] = "Site oficial do sistema de Linus Tourvals 'o inventor do linux' "
descricao[2] = "Site oficial da distribuição mais usada por 'NERDS Linuxers' "

/* pronto agora as variáveis nome, sobrenome, link, descrição contem todos os valores */

/* Escrevendo todos os links e descrições, através o comando PARA */

for(i=0;i<=descricao.length -1;i++){
document.write ('Nome do Autor :' + ' ' + nome[i] + ' ' + sobrenome[i] + '<br><a href=' +
```

```
link[i] + '>' + link[i] + '</a><br>' + descricao[i] + '<br><br><br>');
}
</Script>
```

Testando-se este exemplo teríamos nome do autor, url e descrição escritas no documento, de uma forma dinâmica. Ou seja, primeiramente atribuímos valores às arrays depois usamos um comando for para lê-las uma por vez, e em seguida escrevê-las no documento.

Como vimos, para se atribuir um valor a uma variável podemos usar:

```
minhavariavel[0]=1;
minhavariavel[1]=5;
```

e assim sucessivamente. Mas também podemos atribuir os valores às arrays de diferentes modos, veja abaixo:

```
var minhaarray= new Array(“ae”,”io”, “uao”);
```

e assim seria o mesmo que se tivéssemos atribuído da seguinte forma:

```
minhaarray[0]= “ae”;
minhaarray[1]= “io”;
minhaarray[2]= “uao”;
```

E também poderíamos usar o seguinte:

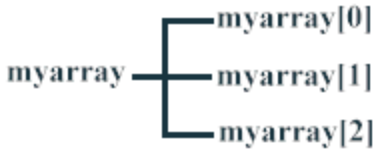
```
fish = ["Lion", , "Angel"];
alert (fish[1]); //o resultado será uma string vazia, ou seja será alertado um espaço em branco
```

No exemplo acima atribuímos no primeiro elemento(primeiro pois as arrays começam com zero), um valor nulo.

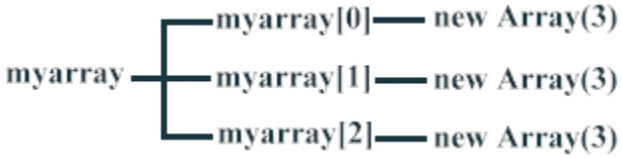
Arrays de 2 dimensões

Criar arrays de 2 dimensões não é uma tarefa tão difícil. Vamos primeiro relembrar como criamos um Array simples.

```
var myarray = new Array(3);
```



Este código acima cria uma Array com três compartimentos. Felizmente isto é um artigo velho para você que está na parte avançada. Graficamente isto é representado da seguinte maneira:



Agora que você tem representado graficamente o que é um Array simples, veremos agora o que é um Array de duas dimensões:

Como você pode perceber, a diferença entre a Array comum e a de duas dimensões é que na segunda, temos que criar um novo Array, no topo de cada Array "comum". A tradução desta idéia esta sendo exemplificada no código abaixo:

```
<Script Language="JavaScript">
var myarray = new Array(3)
for (i=0; i<3; i++)
myarray[i]=new Array(2);


myarray[0][0]="teste esta é 0-0";
myarray[0][1]="teste esta é 0-1";
myarray[0][2]="teste esta é 0-2";
myarray[0][3]="teste esta é 0-3";
myarray[1][0]="teste esta é 1-0";
myarray[1][1]="teste esta é 1-1";
myarray[1][2]="teste esta é 1-2";
myarray[1][3]="teste esta é 1-3";

      for(i=0;i<myarray.length-1;i++){
          for(j=0;j<=myarray.length;j++){
                    document.write(myarray[i][j]+"<br>");
          }
      }
</Script>
```

Sort

Supondo que queremos escrever alguns nomes de mulheres, sendo que estes nomes estão dentro de uma Array. Faríamos um Array com cada nome e depois usaríamos um for para escrever todas. Mas e se fosse necessário ordenar os nomes em ordem alfabética? Para fazer isto de uma forma dinâmica, usamos o comando sort(), veja os seguintes códigos:

```
<Script>
var minhArray =new Array("Patricia","Beatriz","Amanda","Solange");

minhArray.sort(); //agora minhArray[0]="Amanda"...
for(i=0;i<=minhArray.length-1;i++) document.write(minhArray[i]+'<br>');
</Script>
```

Ou podíamos fazer a mesma coisa da seguinte forma:

```
<Script>
var minhArray =new Array();
 minhArray[0]="Patricia";
 minhArray[1]="Beatriz";
 minhArray[2]"Amanda";
 minhArray[3]="Solange";
 minhArray.sort(); //agora minhArray[0]="amanda"...
 for(i=0;i<=minhArray.length-1;i++) document.write(minhArray[i]+'<br>');
</script>
```

Ambos códigos resultariam nisso <http://www.codefactory.com.br/>

Bem, esta função apenas ordena, tanto os textos, quanto números, colocando-os em ordem alfabética ou

ordem numérica.

Opção de uso de expressão condicional

Além do comando if para avaliar uma expressõ, no JavaScript, assim como no Java e no C/C++ temos uma expressão que também checa condições.
Sintaxe:
variável = (condição) ? x : y ;
a expressão acima retorna x se a condição entre parênteses for verdadeira e retorna y se a condição entre parênteses for falsa.
Exemplos:

```
<script>
var navegador = ( navigator.appName.indexOf(“Netscape”)>=0  ) ?  “Netscape” : “Não Netscape”;
document.write (navegador);
</script>
```

A variável navegador terá uma condição avaliada, se o browser do cliente for Netscape então a variável navegador recebe “Netscape “ caso contrário recebe “Não Netscape”.

Comando FOR(avançado)

É claro que você que programa em JavaScript não prescisa saber este comando for avançado, pois da primeira forma que escrevi em meu tutorial, também funciona do mesmo jeito, mas, para os viciados em JavaScript...isso pode ajudar a entender o funcionamento do comando break. Bom então vamos lá escrever o comando for de umas 3 maneiras diferentes.

1 - Em apenas uma linha...

```
<script> for(i=0;i<10;document.write(i+'<br>'),i++);</script>
```

2 - Usando o comando break como auxílio (repare nos pontos e vírgulas dentro do for)

```
<Script>

i = 1;
for (;;) {
    if (i > 10) {
      break;

    }
    document.write(i);
    i++;
}                       ;
</Script>
```

3 - Usando o comando break como auxílio (repare dentro do for, como a expressão mudou)

```
<Script>

for (i = 1;;i++) {
    if (i > 10) {
    break;

    }
    document.write(i);

}
</Script>
```

Este tutorial foi totalmente escrito por *Izaias Lisboa* autor da CodeFactory - http://www.codefactory.com.br, para ver mais artigos sobre JavaScript e sobre DHTML em geral, por favor visite este web site.