

Javascript



Javascript

Autor:
Roberson Luiz

1 INTRODUÇÃO	3
2 JAVASCRIPT.....	3
2.1 O QUE É JAVASCRIPT	3
2.1.1 CARACTERÍSTICAS BÁSICAS.....	3
2.1.2 O QUE JAVASCRIPT NÃO É.....	4
2.1.3 PARTICULARIDADES E LIMITAÇÕES.....	4
2.2 ELEMENTO SCRIPT	5
2.3 EVENTOS EM JAVASCRIPT	8
2.4 MANIPULADORES DE EVENTOS.....	8
2.4.1 ONLOAD	9
2.4.2 ONUNLOAD.....	9
2.4.3 ONCLICK.....	10
2.4.4 ONFOCUS	10
2.4.5 ONBLUR.....	11
2.4.6 ONCHANGE.....	11
2.4.7 ONSELECT.....	12
2.4.8 ONSUBMIT	14
2.4.9 ONMOUSEOVER	16
3 CONSTRUÇÕES DE JAVASCRIPT.....	17
3.1 CONCEITOS BÁSICOS DE PROGRAMAÇÃO	17
3.1.1 CONSTRUÇÃO DE NOMES	17
3.1.2 DECLARAÇÃO DE VARIÁVEIS	17
3.1.3 TIPOS DE VALORES	18
3.1.4 CARACTERES ESPECIAIS.....	19
3.1.5 EXPRESSÕES	19
3.1.6 OPERADORES	19
3.1.7 DECLARAÇÕES	24
3.1.8 FUNÇÕES.....	29
3.2 OBJETOS	30
3.2.1 HIERARQUIA	30
3.2.2 A NATUREZA ORIENTADA A OBJETOS DE HTML.....	31
3.2.3 OBJETO NAVIGATOR	32
3.2.4 OBJETO LOCATION	32
3.2.5 OBJETO CHECKBOX.....	33
3.2.6 OBJETO RADIO	34
3.2.7 OBJETO HIDDEN	34
3.2.8 OBJETO TEXT	34
3.2.9 OBJETO RESET.....	34
3.2.10 OBJETO SUBMIT	35
3.2.11 OBJETO BUTTON.....	35
3.2.12 OBJETO TEXTAREA.....	35
3.2.13 OBJETO SELECT.....	35
3.3 OBJETOS DO CORE JAVASCRIPT.....	36

Javascript

3.3.1 OBJETO ARRAY.....	36
3.3.2 OBJETO DATE.....	36
3.3.3 OBJETO STRING.....	37
3.3.4 OBJETO WINDOW.....	38

1 Introdução

O objetivo deste material é apresentar a linguagem Javascript, muito utilizada atualmente, em conjunto com HTML, para construir páginas dinâmicas e interfaces de aplicações no ambiente *Web*.

A partir deste estudo, espera-se posicionar a tecnologia Javascript no contexto do desenvolvimento *Web*, a fim de que desenvolvedores conheçam o potencial das linguagens interpretadas pelos navegadores.

2 Javascript

2.1 O que é Javascript

Javascript é uma linguagem de *script* orientada a objetos, utilizada para desenvolver aplicações cliente para *Internet/Intranet*. Ela foi criada pela *Netscape* a fim de expandir a funcionalidade de seu popular *browser*: o *Navigator*.

2.1.1 CARACTERÍSTICAS BÁSICAS

Javascript é uma linguagem de *script* (*scripts* são “miniprogramas” interpretados e voltados para execução de tarefas específicas) com uma sintaxe bastante similar a C, C++, Pascal e Delphi.

Os comandos e funções de Javascript são inseridos dentro de um documento da *Web*, junto com *tags* HTML e texto. Quando o navegador de um usuário acessa este documento, ele formata a página, executando o programa nela inserido. Para acessar uma página que possui *scripts*, o navegador deve ser capaz de interpretar a linguagem.

Javascript é uma linguagem baseada em objetos. Uma linguagem baseada em objetos é uma linguagem orientada a objetos com um conjunto de objetos já embutidos.

Sempre que algo acontece em uma página *Web*, ocorre um evento. Eventos podem ser qualquer coisa – um botão recebe um clique, o *mouse* é arrastado, uma página é carregada, um formulário é enviado, e assim por diante. Javascript é uma linguagem dirigida por eventos, no sentido de que é projetada para reagir quando um evento ocorre.

A linguagem Javascript foi projetada para manipular e apresentar informação através de um navegador. Ela não é capaz de recuperar informações de outro arquivo ou salvar dados em um servidor da *Web*, ou no computador do usuário. Isto significa que não é possível escrever um programa Javascript que, por exemplo, varra os diretórios de um computador, lendo ou apagando arquivos do usuário.

JavaScript

JavaScript é uma linguagem independente de plataforma, ou seja, o código escrito nesta linguagem não depende de uma plataforma específica (*Windows*, *Macintosh*, *UNIX*, etc), depende apenas do navegador que a interpreta. Dessa forma, quer o usuário tenha um navegador para *Windows*, *Macintosh* ou *UNIX*, o código JavaScript será executado sem que nenhuma adaptação seja necessária.

2.1.2 O QUE JAVASCRIPT NÃO É

Ainda é muito comum alguém confundir a linguagem JavaScript com a linguagem Java, mas, atenção, **JavaScript não é Java**. Java (desenvolvida pela *Sun Microsystems*) é uma linguagem de programação orientada a objetos completa, que pode ser usada para projetar aplicações isoladas (que não exigem um *browser* para rodar) ou mini-aplicações (*applets*).

Principais diferenças entre JavaScript e Java:

- JavaScript é baseada em objetos - tem seus próprios objetos embutidos. Java é orientada a objetos - os objetos são construídos a partir de classes;
- código JavaScript é embutido dentro de um documento HTML como texto simples. *Applets* Java são referenciados a partir de um documento, mas o código é mantido em um arquivo separado (em um formato binário);
- JavaScript é identificada em um documento HTML através da *tag* <SCRIPT>. *Applets* Java, através da *tag* <APPLET>;
- JavaScript é passada ao cliente (*browser*) como texto e é interpretada. Java é compilada em um tipo especial de código (*bytecodes*), que são passados ao cliente para serem executados;
- JavaScript usa tipagem fraca - as variáveis não precisam ser declaradas, e uma variável ora pode guardar *strings*, ora números. Java usa tipagem forte - as variáveis precisam ser declaradas e usadas para um tipo de dados específico;
- JavaScript usa ligação dinâmica - referências a objetos são verificadas e resolvidas em tempo de execução. Java usa ligação estática - referências a objetos devem ser resolvidas quando o programa é compilado.

2.1.3 PARTICULARIDADES E LIMITAÇÕES

JavaScript é importante para desenvolvedores de páginas *Web* que desejam estender a capacidade de seus documentos HTML, tornando-os dinâmicos. No entanto, esta linguagem apresenta algumas limitações, que se encaixam em três categorias:

Javascript

- Limitações de *Browsers*

Como o código Javascript é executado no cliente (*browser*), seu interpretador deve ser implementado dentro do *browser*. A maioria dos *browsers* disponíveis no mercado atualmente (versões recentes do *Internet Explorer* e do *Netscape*) dão suporte à Javascript.

- Limitações de Plataformas

Embora Javascript rode em todas as plataformas para as quais *browsers* compatíveis existem, nem todas as funções de Javascript rodarão do mesmo modo em plataformas distintas.

- Limitações de Segurança

A linguagem Javascript foi projetada para ser segura com respeito à *Web*. Para isso, ela foi concebida com algumas restrições:

- Ela não pode abrir, ler, gravar ou salvar arquivos no computador do usuário. A única informação que ela pode acessar é a que está na página *Web* onde reside (ou em outras páginas carregadas ao mesmo tempo, como ocorre quando se usam *frames*);
- Ela não pode abrir, ler, gravar ou salvar arquivos no servidor *Web*;
- Ela não pode ser usada para criar vírus que danifique o computador do usuário.

Resumindo, Javascript é uma linguagem segura e não consegue interagir diretamente com nada no computador do usuário fora da página *Web* que está sendo exibida no navegador.

2.2 Elemento SCRIPT

container tag: <SCRIPT>...</SCRIPT>

atributos: LANGUAGE, SRC

Dentro de um documento HTML, a linguagem Javascript é delimitada pelo par de *tags* <SCRIPT> e </SCRIPT>. Você pode posicionar a *tag* <SCRIPT> dentro dos elementos <HEAD> e </HEAD>, <BODY> e </BODY>, ou ambos – é possível incorporar múltiplos elementos <SCRIPT> dentro de um documento.

Scripts dentro da *tag* <HEAD> são carregados antes que o resto da página seja carregado, tornando-se um excelente lugar para colocar suas funções Javascript (para garantir que elas estejam disponíveis para outras partes do

Javascript

documento). Colocar *scripts* dentro da *tag* <BODY> faz com que seja possível criar, dinamicamente, partes de seu documento (exibindo, por exemplo, a hora).

A sintaxe do elemento <SCRIPT> é:

```
<SCRIPT LANGUAGE="javascript">
código-fonte javascript
</SCRIPT>
```

A *tag* <SCRIPT> informa o interpretador Javascript do navegador que um *script* encontra-se embutido. O atributo **LANGUAGE** identifica a linguagem que deve ser utilizada para interpretar o código do *script*. Isto é necessário porque existem outras linguagens para construção de *scripts*, como, por exemplo, VBScript (Microsoft).

O atributo **SRC** permite chamar um arquivo de *scripts* externo (arquivos com extensão ".js"). A fim de facilitar a manutenção, recomenda-se armazenar as funções Javascript de uma página em um arquivo separado, que pode ser incluído no documento HTML com a construção abaixo.

```
<SCRIPT LANGUAGE="javascript" SRC="funcoes.js"></SCRIPT>
```

- Criando o primeiro *script*

O exemplo abaixo imprime a frase "Bom dia!" na página. Observe que, apesar de não existirem elementos entre as marcações <BODY> e </BODY>, esta frase é exibida. Isto ocorre porque a função `document.write()` instrui o navegador a colocar o que estiver entre apóstrofos na página.

```
<html>
<head>
<script language="javascript">
document.write("Bom dia!<BR>");
</script>
</head>
<body>
</body>
</html>
```

- Criando o segundo *script*

Uma das razões de utilizar Javascript é a possibilidade de montar um texto para ser exibido, incluindo valores de variáveis que podem ser inseridos pelo usuário.

```
<html>
<head>
<script language="javascript">
```

Javascript

```
var nome
nome = window.prompt("Digite o seu nome:");
document.write("Bom dia, " + nome + "!<BR>");
</script>
</head>
<body>
</body>
</html>
```

Neste ponto, é importante lembrar que nem todos os navegadores interpretam Javascript. Para evitar resultados indesejados (por exemplo, listagem do código da função Javascript que deveria ser interpretada), usam-se as marcações de comentários “<!--” e “-->”, conforme exemplo a seguir.

```
<html>
<head>
<script language="javascript">
<!--
var nome
nome = window.prompt("Digite o seu nome:");
document.write("Bom dia, " + nome + "!<BR>");
//-->
</script>
</head>
<body>
</body>
</html>
```

2.3 Eventos em Javascript

JavaScript é uma linguagem dirigida por eventos. Eventos (tais como, clicar no *mouse*, ou pressionar um botão) são utilizados para controlar a interação do usuário com o aplicativo.

Programas convencionais funcionam de maneira diferente. Um programa convencional executa seu código sequencialmente.

```
function exhibe() {  
    if ( !(confirm("Quer encerrar?")) ) {  
        documento.write ("Oi, pessoal da Internet!");  
    }  
}
```

Um programa que queira que o usuário confirme a exibição de uma frase poderia usar a função acima para obter a entrada do usuário. Entretanto, este programa ficaria preso na função `exibe()`, esperando por uma resposta. Enquanto isso, não é possível ter outra operação sendo executada. Quaisquer outras entradas e operações são suspensas até que o usuário responda à pergunta.

Uma abordagem melhor seria usar um dos manipuladores de eventos de JavaScript para ativar a função abaixo.

```
function exhibe() {  
    documento.write ("Oi, pessoal da Internet!");  
}
```

Manipuladores JavaScript são representados como atributos especiais que modificam o comportamento de uma *tag* HTML à qual são anexados. Atributos de manipulação de eventos começam todos com "On" e identificam os diversos eventos que podem ocorrer. O valor associado ao manipulador pode ser uma sequência de declarações JavaScript, ou uma chamada de função JavaScript.

2.4 Manipuladores de eventos

Manipuladores de eventos JavaScript servem para interfacear um *script* com atividades do sistema ou ações do usuário. Eles são divididos em 2 categorias: eventos de sistema e eventos de *mouse*.

Os eventos de sistema disponíveis são: *OnLoad* e *OnUnload*. Eles não exigem a interação do usuário para serem ativados.

Javascript

Os eventos de *mouse* disponíveis são: *OnClick*, *OnFocus*, *OnBlur*, *OnChange*, *OnSelect*, *OnSubmit* e *OnMouseOver*. Eles exigem a interação do usuário (através do *mouse* ou não) para serem ativados.

2.4.1 ONLOAD

Este evento é ativado após a página HTML ser completamente carregada. Ele pode ser associado às *tags* <BODY> ou <FRAMESET>.

Exemplo:

```
<html>
<head>
<script language="Javascript">
function chegada() {
    window.alert("Seja bem-vindo ao nosso site!");
}
</script>
</head>
<body OnLoad="chegada()">
Veja que interessante utilização do evento <I>OnLoad</I>.
</body>
</html>
```

2.4.2 ONUNLOAD

Este evento é ativado após a página HTML ser abandonada (seja através do clique sobre algum *link*, ou sobre os botões de avanço/retrocesso do *browser*). Ele pode ser associado às *tags* <BODY> ou <FRAMESET>.

Exemplo:

```
<html>
<head>
<script language="Javascript">
function saida() {
    window.alert("Volte sempre!");
}
</script>
</head>
<body OnUnload="saida()">
Veja que interessante utilização do evento <I>OnUnLoad</I>.
</body>
</html>
```

2.4.3 ONCLICK

Javascript

O evento mais básico de *mouse* é tratado pelo manipulador *OnClick*. Este evento é ativado sempre que se dá um clique sobre um objeto que aceita evento de clique de *mouse*. Objetos que aceitam um evento *OnClick* são *links*, caixas de verificação e botões.

Exemplo:

```
<html>
<head>
<script language="Javascript">
function mensagem() {
    window.alert("Você clicou neste campo");
}
</script>
</head>
<body>
<a href="exemplo3.html" OnClick="mensagem()">
<i>Link</i> comum</a><br>
<form>
<input type="radio" OnClick="mensagem()"><i>Radio</i>
<br>
<input type="checkbox" OnClick="mensagem()"><i>Checkbox</i>
<br>
<input type="reset" OnClick="mensagem()">
<br>
<input type="submit" OnClick="mensagem()">
<br>
</form>
</body>
</html>
```

2.4.4 ONFOCUS

O foco ocorre quando um objeto torna-se o item em foco. Isto acontece quando o usuário clicar ou alternar para um objeto específico na página. Este evento pode ser associado aos objetos *text*, *password*, *textarea* e *select* (definidos pelas *tags* <INPUT>, <TEXTAREA> e <SELECT>).

Exemplo:

```
<html>
<head>
<script language="Javascript">
function foco() {
    window.alert("O campo EMAIL está em foco");
}
</script>
</head>
```

Javascript

```
<body><form>
Nome: <input name="nome" type="text"><br>
Email: <input name="email" type="text" OnFocus="foco()"><br>
Telefone: <input name="telefone" type="text">
</form></body>
</html>
```

2.4.5 ONBLUR

Este evento é ativado quando um objeto torna-se fora de foco - quando se muda para outra janela, ou aplicativo, ou quando se passa para outro objeto utilizando cliques do *mouse*, ou a tecla TAB. Ele é associado aos objetos *text*, *password*, *textarea* e *select* (definidos pelas *tags* <INPUT>, <TEXTAREA> e <SELECT>).

Exemplo:

```
<html><head><script language="Javascript">
function semfoco() {
    window.alert("O campo EMAIL perdeu o foco");
}
</script></head>
<body><form>
Nome: <input name="nome" type="text">
<br>
Email:<input name="email" type="text" OnBlur="semfoco()">
<br>
Telefone: <input name="telefone" type="text">
</form></body></html>
```

2.4.6 ONCHANGE

Este evento é ativado sempre que um objeto perde o foco e o seu valor é alterado. Ele é associado aos objetos *text*, *password*, *textarea* e *select* (definidos pelas *tags* <INPUT>, <TEXTAREA> e <SELECT>).

Exemplo:

```
<html>
<head>
<script language="Javascript">

function mudoul() {
document.form1.completo.value=document.form1.nome.value;
}

```

Javascript

```
function mudou2() {
document.form1.completo.value=document.form1.completo.value
    + " " + document.form1.sobrenome.value;
}

</script>
</head>
<body bgcolor=white link=blue vlink=blue alink=blue>
<form name=form1>
<pre>
Nome:
<input name="nome" type="text" OnChange="mudou1()">
Sobrenome:
<input name="sobrenome" type="text" OnChange="mudou2()">
Nome completo:
<input name="completo" type="text">
</pre>
</form>
</body>
</html>
```

2.4.7 ONSELECT

Este evento é ativado quando o usuário seleciona (deixa em destaque) parte do texto em um dos objetos aos quais está associado. São eles: *text*, *password* e *textarea* (definidos pelas *tags* <INPUT> e <TEXTAREA>).

Javascript

Exemplo:

```
<html>
<head>
<script language="Javascript">
function selecao() {
    window.alert("Campo selecionado");
}
</script>
</head>
<body bgcolor=white link=blue vlink=blue alink=blue>
<form>
<pre>
Campo input texto:
<input type="text" OnSelect="selecao()">
Campo input senha:
<input type="password" OnSelect="selecao()">
Campo textarea:
<textarea OnSelect="selecao()"></textarea>
</pre>
</form>
</body>
</html>
```

2.4.8 ONSUBMIT

Este evento é ativado no momento de enviar os dados do formulário. Ele é associado ao objeto *form* (definido pela tag <FORM>).

Exemplo:

```
<html>
<head>
<script language="Javascript">
function submete() {
    window.alert("Evento OnSubmit ativado!");
}
</script>
</head>
<body bgcolor=white link=blue vlink=blue alink=blue>
<form name=form1 OnSubmit="submete()">
Campo 1: <input type="text" size=10 name=campo1><br>
Campo 2: <input type="text" size=10 name=campo2><p>
<input type=submit>
</form>
</body>
</html>
```

Javascript

Muitas vezes, os dados que são inseridos em um formulário precisam ser separados, analisados, manipulados ou verificados quanto a erros antes de serem transmitidos para o servidor. O evento *OnSubmit* permite a captura e, se necessário, a interrupção do envio dos dados de um formulário. Isto é realizado chamando-se a função a partir do manipulador *OnSubmit*, fazendo com que ela retorne verdadeiro ou falso. Se a função associada ao manipulador retornar falso, os dados do formulário não serão enviados. Esta funcionalidade pode ser verificada a partir do código a seguir.

Exemplo:

```
<html>
<head>
<script language="Javascript">
function submete() {
if (document.form1.campo1.value == "" ||
    document.form1.campo2.value == "") {
    return false;
}
else {
    return true;
}
}
</script>
</head>
<body bgcolor=white link=blue vlink=blue alink=blue>
<form name=form1 action="exemplo8b.html"
    OnSubmit="return submete()">
Campo 1: <input type="text" size=10 name=campo1>
<br>
Campo 2: <input type="text" size=10 name=campo2>
<p>
<input type=submit>
</form>
</body>
</html>
```

Javascript

2.4.9 ONMOUSEOVER

Este evento é ativado quando o ponteiro do *mouse* passa sobre um objeto do tipo *links* ou botões.

Exemplo:

```
<html>
<head>
<script language="Javascript">
function ativa() {
    window.alert("Evento OnMouseOver ativado!");
}
</script>
</head>
<body bgcolor=white link=blue vlink=blue alink=blue>
<a href="exemplo9.html" OnMouseOver="ativa()">
Passe o <i>mouse</i> sobre este <i>link</i></a>
<form>
<input type="reset" value="Botão Reset"
OnMouseOver="ativa()"><P>
<input type="submit" value="Botão Submit"
OnMouseOver="ativa()">
</form>
</body>
</html>
```

3 Construções de Javascript

3.1 Conceitos básicos de programação

3.1.1 CONSTRUÇÃO DE NOMES

JavaScript apresenta algumas restrições quanto ao nome de variáveis/funções:

- não é permitido colocar espaço em branco em um nome;
- não é permitido incluir um hífen ("-") em um nome;
- não é permitido colocar os seguintes caracteres em um nome: . , ; " ' ?
- embora seja possível usar dígitos em um nome, ele precisa começar com uma letra;
- não é permitido utilizar, como nome de uma nova variável/função, alguma das palavras reservadas de JavaScript.

Segue a relação de palavras que não podem ser usadas como nomes de variáveis ou funções em JavaScript:

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

3.1.2 DECLARAÇÃO DE VARIÁVEIS

Cada variável tem que ser declarada como global ou local. A única diferença entre estes dois tipos em JavaScript é onde elas estão localizadas dentro do código. É possível definir variáveis antes de atribuir um valor a elas, ou no momento em que fizer a atribuição.

- Variáveis Locais

São definidas dentro do corpo de uma função. Elas são válidas apenas dentro do corpo da função onde foram definidas (escopo limitado).

Exemplo:

JavaScript

```
function adiciona(valor){
var a
a = valor + 10;
...
}

function subtrai(valor){
var b
b = valor - 10;
...
}
```

Neste exemplo, a variável `a` é vista apenas pela função `adiciona()`, enquanto que a variável `b` é vista apenas por `subtrai()`.

- Variáveis Globais

São definidas fora de todos os corpos de funções de um arquivo Javascript. Elas são válidas dentro de qualquer função do arquivo.

Exemplo:

```
var total = 0;

function adiciona(valor){
var a = valor + 10;
total = total + a;
}

function subtrai(valor){
var b = valor - 10;
total = total - b;
}
```

Neste exemplo, a variável `total` é definida tanto fora da função `adiciona()`, quanto fora da função `subtrai()`, de forma que está acessível a ambas.

3.1.3 TIPOS DE VALORES

Existem 4 (quatro) tipos de variáveis reconhecidos por Javascript: números, valores lógicos (booleanos), *strings* e nulos. O tipo específico de uma variável depende dos dados atribuídos a ela.

- Número: qualquer número positivo ou negativo. Este número pode ser inteiro no formato decimal, hexadecimal ou octal. Pode também ser número de ponto flutuante com/sem exponencial;

JavaScript

- Booleano: *true* ou *false* (sem aspas);
- String: conjunto de caracteres limitados por aspas/apóstrofos. Um *string* exige um par de aspas (ou apóstrofos) para identificá-lo como *string*;
- Nulo: *null* (palavra chave que denota o valor nulo).

3.1.4 CARACTERES ESPECIAIS

Alguns caracteres especiais que são permitidos em valores do tipo *string*:

- `\b`: retrocesso (*backspace*);
- `\f`: nova página (*form feed*);
- `\n`: nova linha (*line feed*);
- `\r`: retorno de carro (*carriage return*);
- `\t`: caracter de tabulação (*tab*).

3.1.5 EXPRESSÕES

Expressão é um conjunto de literais (constantes), variáveis e operadores que, avaliados, resultam em um único valor (número, *string* ou booleano).

Existem 3 (três) tipos de expressões em Javascript:

- expressões aritméticas: resultam em um número;
- expressões de *string*: resultam em uma sequência de caracteres (*string*);
- expressões lógicas: resultam em verdadeiro ou falso, representados, respectivamente, pelas palavras reservadas *true* e *false*.

3.1.6 OPERADORES

Operadores são símbolos especiais que controlam como uma expressão deve ser avaliada. Um operador pode ser binário - exige dois operandos; ou unário - exige apenas um operando (antes ou depois do operador).

Os operadores podem, ainda, ser classificados de acordo com o tipo dos operandos que manipulam: operadores aritméticos, operadores de comparação, operadores de *string*, operadores lógicos, operadores *bit a bit* e operadores de atribuição.

• OPERADORES ARITMÉTICOS

Operadores aritméticos constroem expressões aritméticas. Eles recebem e retornam números.

JavaScript

A tabela 1 mostra os operadores aritméticos básicos de Javascript.

Operador	Função
+	Soma
-	Subtração
*	Multiplicação
/	Divisão

Tabela 1 - Operadores aritméticos básicos

Existem outros operadores aritméticos em Javascript. São eles:

- Módulo (%)
Retorna o resto da operação de divisão inteira entre os operandos.
Exemplo: $22 \% 5 \Rightarrow 2$
- Incremento (++)
É uma forma abreviada de adicionar 1 a um operando (o operando deve ser uma variável).
Exemplo: `operando++ => operando = operando + 1`

O operador de incremento pode ser escrito como `operando++` ou `++operando`. O lado em que o operador está controla quando 1 é adicionado ao operando (com respeito ao resto da expressão). Se ele é usado depois do operando, retorna o valor do operando antes de incrementá-lo. Por exemplo, se $x=3$, a declaração `y=x++` coloca primeiro o valor 3 em `y` e, depois, incrementa `x` para 4. Se, por outro lado, o operador de incremento for colocado antes do operando, ele retorna o valor do operando depois de o operando ser incrementado. O código `y=++x` primeiro incrementa o `x` para 4 e, então, coloca o valor 4 em `y`.

- Decremento (--)
É uma forma abreviada de subtrair 1 de um operando (o operando deve ser uma variável).
Exemplo: `operando-- => operando = operando - 1`

De forma semelhante ao operador de incremento, o operador de decremento pode ser escrito como `operando--` ou `--operando`.

- Negação Unária (-)
Este operador nega o seu operando.
Exemplo: `x = 4; y = -x => y = -4`

- **OPERADORES DE COMPARAÇÃO**

JavaScript

Um operador de comparação compara seus operandos e retorna um valor booleano. Estes operandos podem ser números ou *strings*.

A tabela 2 mostra os operadores de comparação de JavaScript.

Operador	Função
==	Igual a
!=	Diferente de
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a

Tabela 2 – Operadores de comparação

- **OPERADORES DE STRING**

O operador de *string*, "+", serve para concatenar *strings*. Ele recebe e retorna *strings*.

Exemplo: "string1" + "string2" => "string1string2"

- **OPERADORES LÓGICOS**

Os operadores lógicos retornam valores booleanos.

A tabela 3 mostra os operadores lógicos de JavaScript.

Operador	Função
&&	E, AND
	OU, OR
!	NÃO, NOT

Tabela 3 – Operadores lógicos

- **OPERADORES BIT A BIT**

Os operadores *bit a bit* servem para manipular um número a nível de *bit*.

A tabela 4 mostra os operadores lógicos de JavaScript.

JavaScript

Operador	Função
&	E, AND
	OU, OR
^	OU Exclusivo, XOR
~	NÃO, NOT
<<	Deslocamento à esquerda
>>	Deslocamento à direita
>>>	Deslocamento à direita com preenchimento de zeros

Tabela 4 - Operadores bit a bit

- **OPERADORES DE ATRIBUIÇÃO**

JavaScript dá suporte a um método abreviado de escrever operações aritméticas padrão e *bit a bit*. São os operadores de atribuição, apresentados na tabela 5.

Javascript

Operador	Função
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x \% = y$	$x = x \% y$
$x \ll = y$	$x = x \ll y$
$x \gg = y$	$x = x \gg y$
$x \gg\gg = y$	$x = x \gg\gg y$
$x \& = y$	$x = x \& y$
$x \wedge = y$	$x = x \wedge y$
$x = y$	$x = x y$

Tabela 5 - Operadores de atribuição

JavaScript

• Precedência de Operadores

Em Javascript, os operadores têm certa ordem de precedência, ou que operadores são avaliados em primeiro lugar e em que ordem. A ordem de precedência, da mais alta (primeira) à mais baixa (última), é a seguinte:

- Chamadas e membros (() e []);
- Negação, incremento e decremento (!, ~, -, ++ e --);
- Multiplicação e divisão (*, / e %);
- Adição e subtração (+ e -);
- Deslocamentos (<<, >> e >>>);
- Relacionais (<, <=, > e >=);
- Igualdade (== e !=);
- E *bit a bit* (&);
- OU Exclusivo *bit e bit* (^);
- OU *bit a bit* (|);
- E Lógico (&&);
- OU Lógico (||);
- Atribuição.

3.1.7 DECLARAÇÕES

Declaração é uma sequência de palavras-chave, operadores, operandos e/ou expressões terminados por um ponto-e-vírgula.

Todas as declarações em Javascript podem ser agrupadas em uma das seguintes categorias:

- Comandos;
- Declarações Condicionais;
- Declarações de Laço;
- Declarações de Manipulação de Objetos.

• COMANDOS

- Declaração de variáveis

A declaração `var` indica que a palavra imediatamente subsequente é uma variável, à qual é possível atribuir qualquer valor.

Exemplo: `var i;`

`var i=10;`

`var x, y, z=1;`

- Comentário em uma única linha

JavaScript

Para acrescentar comentários que ocupam apenas uma linha, deve-se colocar a sequência “//” antes do texto de comentário.

Exemplo: `total = total + 1; //acumula total`

– Comentário em mais de uma linha

Para acrescentar comentários que ocupam mais de uma linha, deve-se colocar a sequência “/*” na primeira linha e “*/” na última linha.

Exemplo: `/* aqui começa o comentário
aqui termina o comentário */`

• DECLARAÇÕES CONDICIONAIS

Permitem controlar o conjunto de declarações que serão executadas, com base em uma condição (uma expressão ser verdadeira ou falsa). A declaração condicional de Javascript usa as palavras-chave *if* e *else*, e *switch*.

Sintaxe: declaração *if* e *else*

```
if (condição) {  
  // faz algo se a condição for verdadeira  
}  
else {  
  // faz algo se a condição for falsa  
}
```

Sintaxe: declaração *switch*

```
switch (fruta) {  
  case "Laranja":  
    XYZ = 1  
    break  
  case "Banana":  
    XYZ = 2  
    break  
  default:  
    XYZ = 3  
}
```

As chaves (“{ }”) definem um bloco de declarações que são tratadas e executadas como uma unidade.

JavaScript

• DECLARAÇÕES DE LAÇO

Possibilitam repetir a execução de um conjunto de declarações mais de uma vez.

As declarações de laço de Javascript usam as palavras-chave *for* e *while*.

Sintaxe: declaração *for*

```
for (inicialização; condição; incremento) {  
  declaração 1;  
  ...  
  declaração n;  
}
```

De acordo com a sintaxe exposta anteriormente:

- *inicialização* é uma expressão usada para inicializar uma variável contador (a variável utilizada para controlar o número de vezes que o laço é executado);
- *condição* é uma expressão booleana avaliada a cada repetição do laço, antes do corpo do laço ser executado. Enquanto a expressão *for* verdadeira, o conteúdo do laço é executado;
- *incremento* é uma expressão usada para atualizar (através de incremento ou decremento) o contador.

Exemplo:

```
for (i=1; i <=5; i++) {  
  total = total + 1;  
}
```

Na declaração *for*, pode-se omitir as partes de inicialização, condição e incremento, contanto que as vírgulas permaneçam.

Sintaxe: declaração *while*

```
while (condição) {  
  declaração 1;  
  ...  
  declaração n;  
}
```

Dentro do corpo de um laço *for* e *while*, é possível controlar até quando o laço deve ser executado. Isto é obtido através das declarações *break* e *continue*:

JavaScript

- *break*: cancela a execução dos laços *for* e *while*, passando o controle para a próxima instrução fora do laço;
- *continue*: em um laço *for*, ela cancela a execução, passando o controle para a próxima iteração; em um laço *while*, ela cancela a execução, voltando à condição.

• DECLARAÇÕES DE MANIPULAÇÃO DE OBJETOS

JavaScript é uma linguagem baseada em objetos. Um objeto é uma entidade que contém componentes associados que podem armazenar diversos valores.

Algumas vezes pode ser preciso efetuar operações em todas as propriedades de um objeto, ou, até mesmo, listar as propriedades de um objeto. Estas tarefas são facilmente executadas através das declarações *for...in* e *with*. Existe também o operador especial para objetos *new* e uma palavra-chave especial *this*.

- **Operador *new***
Cria uma nova instância de um objeto.

Sintaxe:

novaVar = new *tipoObjeto* (*parâmetros*);

onde, *novaVar* é a variável de objeto criada, *tipoObjeto* é o tipo do objeto que está sendo criado (um dos tipos embutidos de JavaScript), e *parâmetros* são quaisquer parâmetros de que o objeto precisa para ser criado corretamente.

- **Palavra-chave *this***
Objetos são constituídos de propriedades e métodos. Muitos dos objetos embutidos de JavaScript contêm, por sua vez, outros objetos, que podem conter outros objetos, e assim por diante. Às vezes, torna-se difícil saber em que ponto se está na árvore de objetos. JavaScript oferece um “atalho” para referenciar o objeto corrente – a palavra-chave *this*.

Sintaxe:

this.propriedade

- **Declaração *for... in***

JavaScript

Repete uma variável sobre todas as propriedades de um objeto. Este é um tipo especial de declaração de laço que varre todas as propriedades de um objeto.

Sintaxe:

```
for (propriedade in objeto) {  
  declarações  
}
```

onde, *objeto* é o objeto em que se está trabalhando, e *propriedade* é o nome da propriedade do objeto. A cada iteração do laço, *propriedade* recebe uma propriedade diferente do objeto. O laço se encerra quando todas as propriedades de um objeto já foram visitadas.

Exemplo:

```
<html>  
<head><title>JavaScript</title></head>  
<body>  
<h3>Propriedades do Documento</h3>  
<hr>  
<script language="javascript">  
for (propriedade in document)  
document.write(propriedade + "<br>");  
</script>  
</body>  
</html>
```

– Declaração *with*

Indica que, no corpo da declaração, todas as referências a variáveis são propriedades do objeto em questão.

Sintaxe:

```
with (objeto) {  
  declarações  
}
```

Exemplo:

```
with (document) {  
  fgColor = "#000000";  
  bgColor = "#FFFFFF";  
}
```

O código acima pode ser usado no lugar de:

JavaScript

```
document.fgColor = "#000000";  
document.bgColor = "#FFFFFF";
```

3.1.8 FUNÇÕES

Uma função é um conjunto de declarações que executam uma tarefa específica.

Existem dois tipos de funções: funções desenvolvidas pelo usuário e funções pré-definidas da linguagem.

Em Javascript, uma função é identificada pela palavra-chave *function*, uma palavra chamada *nomeFunção*, e um par de parênteses ("()"), que delimitam zero ou mais parâmetros. Todas as declarações dentro de uma função estão dentro de um par de chaves ("{ }").

Sintaxe:

```
function nomeFunção () {  
  declaração 1;  
  ...  
  declaração n;  
}
```

A declaração *return* é utilizada, dentro do corpo de uma função, para retornar um valor, ou para cancelar imediatamente a execução da função.

Exemplo:

```
function aoCubo(valor) {  
  return valor * valor * valor;  
}
```

Algumas funções pré-definidas de Javascript são:

- *eval (string)*, onde *string* representa uma expressão a ser resolvida, retornando um valor numérico.

Exemplo:

```
x = 4;  
eval ("3 + x + 8");
```

Resultado = 15

- *parseInt(string, base)*, onde o primeiro parâmetro representa uma *string* a ser convertida para a base especificada no segundo parâmetro.

Javascript

Exemplo:

```
parseInt("45", 10);
```

Resultado = número 45 na base decimal

- `isNaN` recebe um argumento e determina se ele é ou não um número. Caso ele seja um número, retorna o valor *false*. Se não for, retorna *true*.

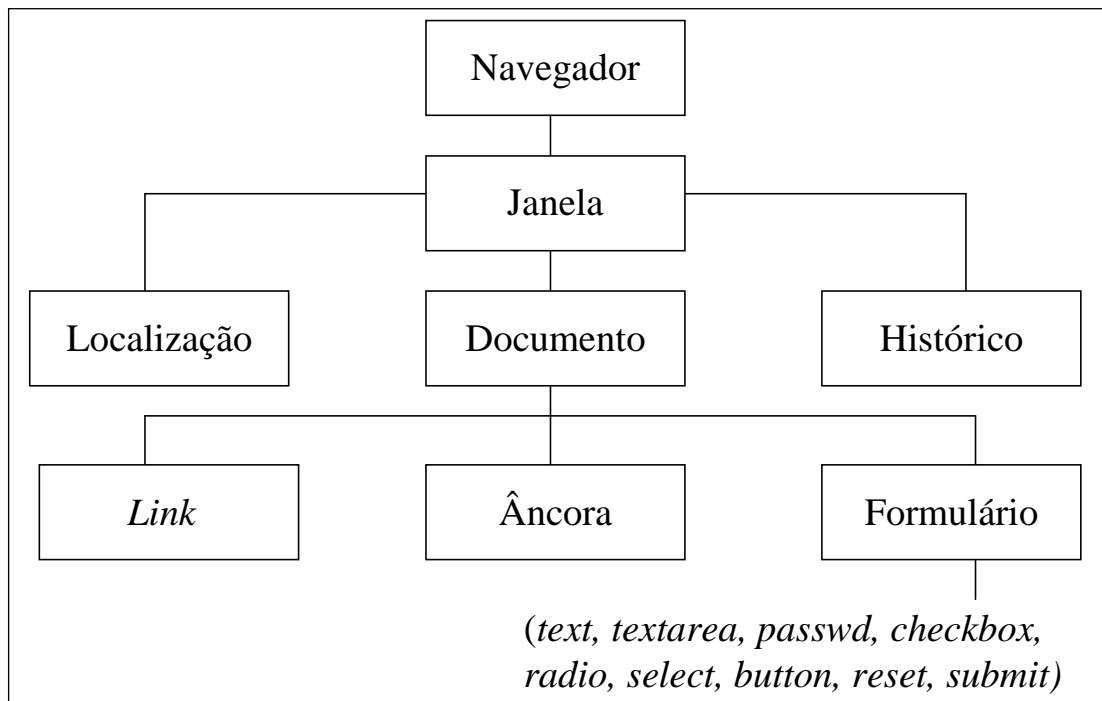
3.2 Objetos

3.2.1 HIERARQUIA

O nível mais alto de objetos em Javascript consiste naqueles objetos que pertencem a *navigator* (navegador). Diretamente abaixo deste nível, estão os objetos *window* (janela). Cada janela tem uma árvore de níveis que se ramifica a partir dela. Estas árvores consistem em *location* (localização), *history* (histórico) e *document* (documento). A cada nível há outros objetos e abaixo da árvore de documentos há outra ramificação. Neste nível, há três objetos array - *forms* (formulários), *anchors* (âncoras) e *links*. A figura 1 mostra a hierarquia de objetos de Javascript.

No *browser*, os objetos seguem a mesma estrutura hierárquica da página HTML: de acordo com essa hierarquia, os descendentes dos objetos são propriedades desses objetos.

Quando uma página é carregada no *browser*, ele cria um número de objetos de acordo com o conteúdo da página. Os seguintes objetos são sempre criados, independentemente do conteúdo da página: *window*, *location*, *history* e *document*.



JavaScript

Figura 1 – Hierarquia de objetos

3.2.2 A NATUREZA ORIENTADA A OBJETOS DE HTML

JavaScript considera HTML uma linguagem orientada a objetos, na qual os diversos *tags* HTML correspondem a diferentes tipos de objetos JavaScript.

Exemplo:

```
<html>
<head><title>Minha pagina</title></head>
<body>
<form name="formulario1">
<input type="button" name="botaol">
</form>
</body>
</html>
```

A partir do código acima, obtém-se os seguintes objetos JavaScript:

- `document.title` : título da página
- `document.formulario1` : formulário da página
- `document.formulario1.botaol` : botão do formulário

Este é apenas um exemplo dos objetos que JavaScript cria automaticamente de HTML. Observe que quase todo elemento HTML pode ser usado como um objeto. Além disso, é possível ler e atribuir valores a estes objetos dinamicamente.

Tenha em mente que à medida em que se estrutura uma página, também está se definindo objetos e seus valores para JavaScript. As próximas seções apresentam detalhadamente o conjunto de objetos da linguagem JavaScript.

3.2.3 OBJETO NAVIGATOR

Este objeto dá informações sobre o navegador. Ele apresenta as seguintes propriedades: *appName*, *appVersion*, *appCodeName*, *userAgent*.

- Propriedade *appName*: retorna o nome do *browser* do usuário.

Javascript

Exemplo: `navigator.appName = Netscape`

- Propriedade *appVersion*: retorna a versão do *browser* e a versão do sistema operacional em que ele está rodando. [formato: número da versão (plataforma; país)]

Exemplo: `navigator.appVersion = 2.0 (Win95; I)`

- Propriedade *appCodeName*: retorna o nome do código de desenvolvimento interno do desenvolvedor de um *browser* específico.

Exemplo: `navigator.appCodeName = Mozilla`

- Propriedade *userAgent*: usada em cabeçalhos HTTP para fins de identificação, é a combinação das propriedades *appCodeName* e *appVersion*. Servidores *Web* usam esta informação para identificar os recursos que o navegador dispõe.

Exemplo: `navigator.userAgent=Mozilla/2.0 (Win95; I)`

3.2.4 OBJETO LOCATION

Este objeto é utilizado para identificar o documento corrente. Ele consiste em uma URL completa no formato *protocolo//servidor:porta/caminho* seguidos de *?search* ou *#hash*.

Suas propriedades são: *protocol*, *hostname*, *port*, *pathname*, *search*, *hash*. Cada uma de suas propriedades representa uma parte da URL total.

- Propriedade *protocol*: retorna o protocolo de transporte do documento.

Exemplo: `location.protocol = http:`

- Propriedade *hostname*: identifica o nome do computador hospedeiro.
- Propriedade *port*: especifica a porta para o endereço. Esta informação é utilizada apenas se uma porta não-padrão estiver sendo usada.
- Propriedade *pathname*: define o caminho e o nome do arquivo.
- Propriedade *search*: retorna quaisquer comandos de consulta que possam estar embutidos na URL corrente. Valores de *search* são separados do resto da URL por um sinal de interrogação (“?”).

Exemplo: `location.search = nome=Joao`

JavaScript

- Propriedade *hash*: retorna quaisquer âncoras que possam ter sido passadas na URL. Valores de *hash* são separados do resto da URL por um sinal de cerquilha (“#”).

Exemplo: `location.hash = capitulo1`

3.2.5 OBJETO CHECKBOX

Utilizado na construção de caixas de verificação. Suas propriedades são: *name*, *value*, *checked*, *defaultChecked*.

- Propriedade *name*: especifica o nome da caixa.
- Propriedade *value*: especifica o valor da caixa.

Exemplo:

```
nomeForm.nomeCheckbox.value = "1"
```

- Propriedade *checked*: valor booleano que especifica o estado de seleção da caixa (selecionada ou não-selecionada).

Exemplo:

```
if ( nomeForm.nomeCheckbox.checked == true ) {  
    funcao();  
}
```

- Propriedade *defaultChecked*: valor booleano que especifica o estado *default* de seleção da caixa.

3.2.6 OBJETO RADIO

Corresponde a um *array* de botões, onde todos os botões compartilham a mesma propriedade *name*. Suas propriedades são: *name*, *checked*, *defaultChecked*, *length*.

- Propriedade *name*: especifica o nome do objeto.
- Propriedade *checked* e *defaultChecked*: funcionamento idêntico ao definido em *checkbox*.
- Propriedade *length*: especifica o comprimento do *array*.

Javascript

3.2.7 OBJETO *HIDDEN*

Utilizado para enviar informações quando o formulário é submetido (este objeto não aparece no formulário). Suas propriedades são: *name*, *value*.

- Propriedade *name*: especifica o nome do objeto.
- Propriedade *value*: especifica a informação que está sendo passada.

3.2.8 OBJETO *TEXT*

Utilizado para entrada/saída de dados. Suas propriedades são: *name*, *value*, *defaultValue*.

- Propriedade *name*: especifica o nome do objeto.
- Propriedade *value*: especifica o valor do objeto.
- Propriedade *defaultValue*: especifica o valor *default* do objeto.

3.2.9 OBJETO *RESET*

Utilizado para limpar dados de um formulário. Suas propriedades são: *name*, *value*.

- Propriedade *name*: especifica o nome do botão.
- Propriedade *value*: especifica o título colocado na face do botão.

Exemplo: `document.form.botao.value="novo titulo"`

3.2.10 OBJETO *SUBMIT*

Utilizado para interfacear Javascript e outros *scripts*/programas. Suas propriedades são: *name*, *value*.

- Propriedade *name*: especifica o nome do botão.
- Propriedade *value*: especifica o título colocado na face do botão.

JavaScript

3.2.11 OBJETO *BUTTON*

Utilizado na construção de *interfaces*. Suas propriedades são: *name*, *value*.

- Propriedade *name*: especifica o nome do botão.
- Propriedade *value*: especifica o título colocado na face do botão.

3.2.12 OBJETO *TEXTAREA*

Utilizado para entrada/saída de dados. Suas propriedades são: *name*, *value*, *defaultValue*.

- Propriedade *name*: especifica o nome do objeto.
- Propriedade *value*: especifica o valor do objeto.
- Propriedade *defaultValue*: especifica o valor *default* do objeto.

3.2.13 OBJETO *SELECT*

Utilizado para construir caixas de seleção. Suas propriedades são: *name*, *options*, *length*.

- Propriedade *name*: especifica o nome do objeto.
- Propriedade *options*: *array* que contém uma entrada para cada opção de uma caixa de seleção.
- Propriedade *length*: especifica o comprimento do *array* de opções.

O exemplo a seguir identifica que opções foram selecionadas na caixa de seleção.

Exemplo:

```
function listSelected(obj) {
  for (i=0; i < obj.length; i++) {
    document.write(" " + obj.options[i].name + " ");
    if (!obj.options[i].selected) {
      document.write("não está selecionada<BR>");
    }
    else {
```

Javascript

```
        document.write("está selecionada<BR>");  
    }
```

3.3 Objetos do CORE Javascript

Os objetos do CORE Javascript são os objetos instrínsecos da linguagem, isto é existem tanto no cliente (navegador), quanto no servidor *Web*.

Os objetos do *Client-Side* Javascript e do *Server-Side* Javascript somente funcionam, respectivamente, em programas escritos para o navegador e para o servidor *Web*.

3.3.1 OBJETO ARRAY

É possível criar um vetor através do objeto Array, pré-definido no Javascript.

```
nome_do_array = new Array(10)
```

Para inserir e consultar elementos de um array, deve-se utilizar a seguinte sintaxe: `nome_do_array[x]`

A variável *x* representa o índice de um elemento. O primeiro elemento é o de número 0.

3.3.2 OBJETO DATE

O objeto Date lhe ajuda a manipular datas. Para criar um objeto do tipo Date, deve-se utilizar a seguinte sintaxe:

```
nome_do_objeto = new Date(parâmetros)
```

Se não for indicado nenhum parâmetro, será criado um objeto com a data e a hora atual do sistema. É possível passar uma *string* representando uma data e hora como parâmetro:

```
x = new Date("October 01, 2001 08:00:00")
```

Caso sejam omitidos hora, minuto e segundo, o valor padrão será 0. Outra forma de indicar data e hora é através de uma série de parâmetros numéricos, representando o ano, mês, dia, hora, minutos e segundos.

A função *Date()* retorna a data atual.

JavaScript

Propriedades mais utilizadas:

- `defaultStatus`: a mensagem que será exibida quando não tiver nenhuma outra na *status bar* do navegador. Cuidado para não confundir com a propriedade *status*, que reflete umas mensagens transitórias, adequadas para um certo momento ou ação do usuário;
- `Height`: esta propriedade contém a altura, em pixels, da janela do navegador;
- `Width`: semelhante à propriedade anterior, porém trabalha com a largura;
- `name`: representa o nome da janela;
- `status`: especifica a mensagem a ser exibida na *status bar* do navegador. É muito útil para comunicar ao usuário pequenas mensagens.

Javascript

Métodos mais utilizados:

- `alert()`: exibe uma mensagem para o usuário. A *string* com a mensagem deve ser passada para o método como parâmetro;
- `back()`: é equivalente a apertar o botão *back* do navegador. Ou seja, volta atrás na última navegação feita pelo usuário;
- `forward()`: tem o mesmo efeito do botão *forward* do navegador. Ou seja, se o usuário tiver apertado o botão *back* para voltar à última página visitada, o *forward* avança novamente para a página mais recente;
- `open()`: abre uma nova janela. O método recebe como parâmetros uma URL (o endereço da página que vai ficar na nova janela), o nome da janela e uma *string* com suas características;
- `close()`: fecha a janela especificada. O Javascript somente pode fechar automaticamente janelas abertas por ele. Caso contrário, aparece uma caixa de confirmação para o usuário;
- `confirm()`: exibe uma caixa de mensagem para o usuário com duas opções: OK e Cancel. Caso o usuário pressione OK, o método retorna `true`. Caso contrário, `false`. Ele recebe como parâmetro uma *string* com a mensagem a ser exibida para o usuário;
- `prompt()`: exibe uma caixa de mensagem e campo para o usuário entrar com uma *string*. O método retorna a *string* digitada pelo usuário. São aceitos dois parâmetros. O primeiro é uma *string* com a mensagem a ser exibida e o segundo é o valor padrão da *string* a ser digitada pelo usuário.

Exemplo:

```
<HTML>
<HEAD>
<TITLE>Exemplo</TITLE>
</HEAD>
<BODY>
<SCRIPT>
x = prompt("Qual a sua idade?", "25");
janela = window.open('', '',
'innerHeight=100,innerWidth=150,resizable=yes');
janela.document.write("<BR> <B> <CENTER>");
janela.document.write("Você tem " + x + " anos!");
janela.document.write("</CENTER> </B>");</SCRIPT>
</BODY>
</HTML>
```