

JavaScript

Guia do Programador

Maurício Samy Silva

Copyright © Novatec Editora Ltda. 2010.

Todos os direitos reservados e protegidos pela Lei 9610 de 19/02/1998.

É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito, do autor e da Editora.

Editor: Rubens Prates

Revisão gramatical: Patrizia Zagni

Editoração eletrônica: Camila Kuwabata e Carolina Kuwabata

Capa: Victor Bittow

ISBN: 978-85-7522-248-5

Histórico de impressões:

Setembro/2010 Primeira edição

Novatec Editora Ltda.

Rua Luís Antônio dos Santos 110

02460-000 – São Paulo, SP – Brasil

Tel.: +55 11 2959-6529

Fax: +55 11 2950-8869

E-mail: novatec@novatec.com.br

Site: www.novatec.com.br

Twitter: twitter.com/novateceditora

Facebook: facebook.com/novatec

LinkedIn: linkedin.com/in/novatec

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Silva, Maurício Samy
JavaScript : guia do programador / Maurício
Samy Silva. -- São Paulo : Novatec Editora, 2010.

Bibliografia.
ISBN 978-85-7522-248-5

1. JavaScript (Linguagem de programação)
I. Título.

10-10573

CDD-005.133

Índices para catálogo sistemático:

1. JavaScript : Linguagem de programação :
Computadores : Processamentos de dados
005.133
CRM20100922



CAPÍTULO 1

Introdução à JavaScript

Neste capítulo, será apresentada a linguagem JavaScript, relatando-se suas origens, finalidades e destinação. Será feito um breve relato histórico de sua evolução, examinando as diferentes versões por que passou e desmistificando alguns conceitos malformados que acabaram por estigmatizar a linguagem como uma técnica inferior e em desacordo com as boas práticas de programação para web.

Com a finalidade de criar uma base de estudos para os tópicos e capítulos que se seguem, serão apresentados os **métodos para criar caixas de alerta**, escrever e atrelar um evento na marcação HTML.

Estudaremos, ainda, os conceitos básicos de objetos da linguagem JavaScript e aprofundaremos o estudo de variáveis mostrando como, onde e quando as declarar.

Ao final do estudo deste capítulo, o leitor terá uma sólida visão dos princípios e da filosofia de emprego de programação JavaScript em conformidade com os Padrões Web, bem como adquirido os conhecimentos necessários para **criar caixas de alerta, escrever HTML com o uso de JavaScript** e atrelar, ainda que em caráter precário, um evento a um botão HTML. Esses conhecimentos auxiliarão o leitor a criar visualização para os resultados de seus primeiros scripts.

1.1 Visão geral

A linguagem de marcação **HTML** destina-se a **estruturar uma página web**, não se devendo empregá-la para adicionar estilos ou apresentação visual aos elementos que constituem a página, sendo tais tarefas função das folhas de estilo em cascata. A HTML, em sua versão atual _ HTML 4.01 _, também não possui funcionalidades que permitam adicionar **interatividade avançada à página**, sendo tal tarefa função das linguagens de programação.

JavaScript foi criada pela Netscape em parceria com a Sun Microsystems, com a finalidade de fornecer um meio de adicionar **interatividade a uma página web**. A primeira versão, denominada JavaScript 1.0, foi lançada em 1995 e implementada em **março de 1996** no navegador Netscape Navigator 2.0 quando o mercado era dominado pela Netscape.

Logo a seguir, veio a época da chamada guerra dos browsers, cujos efeitos nocivos se fazem sentir até os dias atuais. Para não fugir à regra, a Microsoft, em resposta à Netscape, criou a linguagem **JScript** baseada em **Visual Basic** cuja primeira versão denominada JScript 1.0 foi lançada com o navegador Internet Explorer 3.0.

Não há como fazer funcionar um formulário HTML com o uso de elementos HTML. A HTML limita-se a criar os rótulos e campos de um formulário para serem preenchidos pelo usuário e nada mais. **Com HTML**, não conseguimos processar os dados nem mesmo enviá-los ao servidor ou a outra máquina qualquer. Para cumprir essas tarefas, é necessário utilizar um programa que consiga **manipular e processar os dados**. Entre as várias linguagens de programação destinadas a adicionar e processar dados em páginas web, destacam-se **PHP, ASP, Java, Ruby, Python e ColdFusion**, entre outras.

As linguagens de programação como as citadas anteriormente foram desenvolvidas para rodar no **lado do servidor**, isto é, dependem de uma máquina remota onde estão hospedadas as funcionalidades capazes de interpretar e fazer funcionar os programas.

JavaScript é uma linguagem desenvolvida para **rodar no lado do cliente**, isto é, a interpretação e o funcionamento da linguagem dependem de funcionalidades hospedadas no navegador do usuário. Isso é possível porque existe um interpretador JavaScript hospedado no navegador.

Tanto a Netscape como a Microsoft desenvolveram interpretadores JavaScript para serem hospedados no servidor, tornando possível rodar JavaScript no lado do servidor. Esses interpretadores foram disponibilizados para uso público e podem ser usados pelos desenvolvedores para serem embutidos em aplicações gerais. A organização Mozilla lançou, sob a bandeira de código aberto e livre, duas versões do interpretador: **Spider Monkey escrita em C e Rhino escrita em Java**.



Neste livro, trataremos exclusivamente de JavaScript para rodar no lado do cliente. Assim sendo, fica subentendido que daqui para frente a palavra JavaScript se refere à JavaScript a ser interpretada pelo navegador.

Em tese, precisamos apenas de um navegador para fazer funcionar scripts desenvolvidos com a linguagem JavaScript. Ao contrário, programas escritos em PHP, por exemplo, precisam ser hospedados em um servidor remoto configurado para rodar PHP ou visualizados localmente em uma máquina na qual tenha sido instalado um servidor local com suporte para PHP.

1.2 Funcionalidades gerais da JavaScript

1.2.1 Manipular conteúdo e apresentação

Com JavaScript, podemos escrever marcação HTML e inseri-la na marcação de um documento existente. Por exemplo: inserção de data/hora no documento, inserção de uma mensagem de boas-vindas ou, ainda, inserção de conteúdos diferenciados e escolhidos de acordo com o navegador do usuário. Podemos, até mesmo, gerar o HTML completo de uma página web.

Na verdade, esse poder da **JavaScript em gerar marcação HTML** foi um dos fatores responsáveis pela má fama da linguagem. Desenvolvedores, maravilhados com as possibilidades dessa linguagem, começaram a gerar, indiscriminadamente, HTML, tornando os conteúdos completamente inacessíveis. Antigamente, tal prática era comum, aceitável e amplamente empregada, porém com a chegada dos Padrões Web, **não é admissível gerar indiscriminadamente HTML com JavaScript**. É preciso uma análise criteriosa para determinar que HTML pode ou não ser gerado com JavaScript.



Usar JavaScript em conformidade com os Padrões Web implica, entre outras práticas, que, na ausência de um interpretador JavaScript, não se comprometa o acesso ao conteúdo.

Como regra geral, evite **gerar marcação HTML com JavaScript**. Ao longo deste livro, sempre que for o caso, ressaltaremos as exceções a essa recomendação.

JavaScript é capaz de definir, alterar e controlar de forma dinâmica a apresentação de um documento HTML, como os aspectos relacionados à cor de fundo, de textos e de links, ou mesmo interferir no posicionamento dos elementos HTML de um documento. É possível manipular a folha de estilos associada ao documento criando novas regras CSS ou anulando regras existentes.

1.2.2 Manipular o navegador

Com JavaScript, podemos controlar o comportamento do navegador em diversos aspectos, como criar janelas pop-up, apresentar mensagens ao usuário, alterar as dimensões do navegador, interferir na barra de status, retirar menus, fechar e abrir janelas.

É muito provável que você já tenha passado pela desagradável experiência de entrar em um site e, surpreendentemente, constatar que toda a parte superior do navegador simplesmente desapareceu, impedindo-o de fechar a janela. Ou, ainda,

a cada clique em um link, ver abrir uma janela pop-up com propaganda de um produto que em nada se relaciona a seus desejos consumistas, ou, ao sair de um site ou fechar uma janela, descobrir uma outra janela não solicitada.

Esses comportamentos, inesperados e não solicitados pelo usuário, inseridos por JavaScript também contribuíram para a má fama da linguagem. Portanto, não perpetue essa prática nociva e ultrapassada, desenvolvendo seus scripts em conformidade com os Padrões Web e voltando-se exclusivamente para enriquecer a experiência do usuário sem criar barreiras para a acessibilidade e a usabilidade.

Adote como regra geral não manipular a janela do navegador do usuário, pois é ele quem decide se quer abrir, fechar, redimensionar, voltar, sair e fazer o que bem entender com o navegador.

1.2.3 Interagir com formulários

JavaScript é capaz de acessar os campos e valores digitados em um formulário HTML e proceder à validação dos dados, realizar cálculos e fornecer dicas de preenchimento dos campos.

1.2.4 Interagir com outras linguagens dinâmicas

JavaScript pode ser usada em conjunto com outras linguagens para cumprir tarefas complementares relacionadas ao fluxo da programação.

1.3 JavaScript em conformidade com os Padrões Web

O conceito de desenvolvimento em conformidade com os Padrões Web trouxe como consequência imediata para a linguagem JavaScript a necessidade de se rever os seus critérios de uso. Daí surgiram dois princípios básicos que norteiam seu uso: o princípio de Javascript não obstrutivo e o da melhoria progressiva.

Não há uma clara divisão entre eles, na verdade se completam e são interdependentes.

Escrever JavaScript não obstrutivo implica:

- que o conteúdo da página deve estar presente e funcional, ainda que se perca em usabilidade, caso o usuário esteja visualizando o documento em um dispositivo (por exemplo, navegador) sem suporte para JavaScript;

- usar a linguagem com vistas a unicamente incrementar a usabilidade da página;
- escrever scripts em arquivos externos para serem linkados ao documento e não inserir script na marcação HTML.

O primeiro conceito traz a grande novidade e separa definitivamente os princípios de uso da JavaScript à maneira ultrapassada e à maneira moderna, em conformidade com os Padrões Web.

Até o advento dos Padrões Web, a premissa em vigor era a de que JavaScript e acessibilidade eram incompatíveis. Não havia como se pensar em desenvolvimento JavaScript contemplando a acessibilidade. Se desabilitarmos JavaScript em nosso navegador e passarmos algum tempo navegando por sites na Web, certamente teremos uma boa ideia da incompatibilidade de JavaScript mal desenvolvido com acessibilidade. Assim, não é exagero afirmar que o primeiro conceito da listagem anterior revolucionou a forma de escrever JavaScript.

O segundo conceito agrega um valor eminentemente prático à linguagem, acabando definitivamente com a prática de criar scripts voltados unicamente a acrescentar efeitos espetaculares na página, mas sem qualquer utilidade, como figuras acompanhando o cursor, objetos voando pela página, luzes piscando e uma parafernália de efeitos tão próprios do desenvolvimento em anos passados.

O terceiro conceito alinha-se com o moderno conceito de separação de camadas de desenvolvimento. JavaScript deve ser mantido na camada de comportamento, não invadindo a camada de estruturação do conteúdo (marcação HTML) nem a camada de apresentação (CSS).

1.4 Camadas de desenvolvimento

Com a chegada dos Padrões Web, o conceito de desenvolvimento em camadas tornou-se um importante ponto a ser considerado na construção de aplicações Web.

Tal conceito preconiza a separação dos códigos de desenvolvimento em três camadas separadas: a camada de estruturação de conteúdos constituída pela marcação HTML, a camada de apresentação constituída pelas folhas de estilos e a camada de comportamento constituída pelos scripts que determinam comportamentos como scripts desenvolvidos com JavaScript.

Desenvolver segundo o princípio da separação das camadas de desenvolvimento implica escrever os códigos específicos de cada camada em arquivos separados e

estabelecer a conexão entre eles com o uso de links. As principais vantagens de adotar a prática de separação das camadas são:

- elimina a necessidade de repetição de códigos em diferentes páginas;
- facilita o reaproveitamento de trechos de códigos em outros projetos;
- facilita a busca e correção de eventuais bugs nos códigos;
- facilita a manutenção e o entendimento dos códigos.

Outro conceito intimamente relacionado ao princípio de separação das camadas de desenvolvimento é o moderno conceito, introduzido com a chegada dos Padrões Web, conhecido como “melhoria progressiva” (progressive enhancement). Segundo esse princípio, o desenvolvimento de uma página Web deve ser feito em três etapas:

- Na primeira etapa, **estruturar os conteúdos usando a linguagem HTML**. Ao final dessa etapa, todos os conteúdos devem estar disponíveis para qualquer visitante que esteja utilizando qualquer dispositivo de usuário.
- Na segunda etapa, **incrementar a apresentação da página com o uso das CSS**. Essa etapa visa a melhorar a experiência dos usuários aptos a visualizar folhas de estilos.
- Finalmente, na terceira etapa, introduzir **JavaScript com a finalidade de acrescentar interatividade à página**, melhorando ainda mais a experiência do usuário.

1.5 Introdução à linguagem

A linguagem JavaScript foi inventada por Brendan Eich, da Netscape, e a primeira versão da linguagem denominada JavaScript 1.0 foi introduzida no navegador Netscape 2.0 em 1996. Nesse mesmo ano, a Microsoft lançou sua versão com o nome JScrip 1.0 e introduziu-a no então Internet Explorer 3.0. Àquela época, em plena guerra dos navegadores, as diferentes implementações das funcionalidades da linguagem nos dois navegadores não seguiam um padrão unificado, causando um verdadeiro martírio para o desenvolvedor implantar scripts para servir ambos os navegadores. Não iremos entrar em detalhes históricos com relação às implementações proprietárias, mas convém ressaltar que até os dias atuais ainda sentimos os efeitos nefastos daquela era, conforme veremos no decorrer dos capítulos deste livro.

Atualmente, o nome oficial da JavaScript é ECMAScript e a versão da linguagem é a ECMA-262 v5. Na tabela 1.1, há um resumo da evolução da JavaScript em suas diferentes versões e implementações.

Tabela 1.1 – Versões da JavaScript

Versão	Implementação	Mês/ano
JavaScript 1.0	Netscape 2.0	Março 1996
JavaScript 1.1	Netscape 3.0	Agosto 1996
JavaScript 1.2	Netscape 4.0 e 4.05	Junho 1997
JavaScript 1.3	Netscape 4.06 e 4.07x	Outubro 1998
JavaScript 1.4	Servidores Netscape	-
JavaScript 1.5	Netscape 6.0 – Firefox 1.0 – Opera 6.0 a 9.0	Novembro 2000
JavaScript 1.6	Firefox 1.5 – Safari 3.0 e 3.1	Novembro 2005
JavaScript 1.7	Firefox 2.0 – Safari 3.2 e 4.0 – Chrome 1.0	Outubro 2006
JavaScript 1.8	Firefox 3.0	Junho 2008
JavaScript 1.8.1	Firefox 3.5	2008
JavaScript 1.9	Firefox 4.0	2009
JavaScript 1.0	Internet Explorer 3	Agosto 1996
JavaScript 2.0	Internet Explorer 3 – Windows IIS 3	Janeiro 1997
JavaScript 3.0	Internet Explorer 4	Outubro 1997
JavaScript 4.0	Visual Studio 6.0	-
JavaScript 5.0	Internet Explorer 5	Março 1999
JavaScript 5.1	Internet Explorer 5.01	-
JavaScript 5.5	Internet Explorer 5.5	Julho 2000
JavaScript 5.6	Internet Explorer 6	Outubro 2001
JavaScript 5.7	Internet Explorer 7	Novembro 2006
JavaScript 5.8	Internet Explorer 8	Março 2009
ECMA-262 v1	Navegadores versão 4	1998
ECMA-262 v2	Versão de testes	1998
ECMA-262 v3	Navegadores versão 6	1999
ECMA-262 v4	Navegadores versão 6+	2002
ECMA-262 v5	Versão atual	2009

1.5.1 Definições

ECMA

Abreviatura para European Computer Manufacturers Association, trata-se de uma associação fundada em 1961 dedicada à padronização de sistemas de informação. Desde 1994, passou a se denominar ECMA International para refletir suas atividades internacionais. A associação é aberta a companhias que produzem, comercializam ou desenvolvem sistemas de computação ou de comunicação na Europa. Em 1996, a

ECMA Internacional começou a desenvolver a ECMA-262 que vem a ser a norma para JavaScript. Em 1997, foi lançada a primeira edição da norma e, em 1998, a norma foi reconhecida oficialmente pela International Organization for Standardization (ISO).

ECMAScript

É uma linguagem de programação orientada a objetos que se destina a realizar cálculos e manipular objetos computacionais de um ambiente de hospedagem. ECMAScript não se destina a ser computacionalmente autossuficiente, visto que não existem disposições nas especificações que normatizem entrada de dados externos ou saída de resultados calculados. Em vez disso, espera-se que o ambiente computacional de um programa ECMAScript proporcione não apenas os objetos e outras funcionalidades descritas nas especificações, mas também **objetos específicos a um ambiente de hospedagem**, cuja descrição e cujo comportamento estão além do escopo da especificação.

Linguagem de script

É uma linguagem de programação usada para **manipular, personalizar e automatizar** as **funcionalidades de um sistema** existente. Em tais sistemas, as funcionalidades já se encontram disponíveis por meio de uma interface de usuário e a linguagem de script provê um mecanismo para acessá-las. Dessa forma, o sistema existente oferece um ambiente de hospedagem para **objetos e funcionalidades** que complementa os recursos da linguagem de script. A linguagem de script destina-se a programadores profissionais e não profissionais.

Ambiente de hospedagem

Um navegador web é um exemplo de ambiente de hospedagem para ECMAScript que funciona no lado do cliente. Ele hospeda objetos que representam janelas, menus pop-ups, caixas de diálogo, áreas de texto, âncoras, quadros, histórico e cookies. Além disso, o ambiente fornece funcionalidades para que o script manipule eventos como mudança de foco, carregamento e fechamento de documentos, erros, seleção, apresentação de formulários e variadas ações do mouse.

Um servidor web é outro exemplo de ambiente de hospedagem para ECMAScript que funciona no lado do servidor. Hospeda objetos que representam as requisições ao servidor, clientes, arquivos e mecanismos de bloqueio e compartilhamento de dados. Ao desenvolver scripts, em conjunto, no lado do cliente e no lado do servidor, é possível estabelecer interação entre o cliente e o servidor, proporcionando uma interface de usuário personalizada para um aplicativo baseado na web.

Cada navegador ou servidor que suporta ECMAScript fornece seu ambiente próprio de hospedagem que completa o ambiente de execução ECMAScript.

1.6 Criando uma base para estudos

Antes de prosseguirmos com a apresentação dos fundamentos da linguagem, vamos apresentar alguns conceitos e funcionalidades da JavaScript que servirão de base para desenvolver nossos estudos e exemplos práticos disponíveis no site do livro.

O que iremos estudar a seguir são temas básicos da JavaScript e se o leitor já tem alguma intimidade com essa linguagem, sinta-se à vontade para ler o capítulo seguinte. Contudo, convido-o a ler o que se segue com o propósito de fazer uma revisão de conceitos.

1.6.1 Orientação a objetos

A linguagem JavaScript suporta programação orientada a objetos – Object-Oriented Programming (OOP). É mais apropriado dizer que JavaScript é uma linguagem capaz de simular muitos dos fundamentos de OOP, embora não plenamente alinhada com todos os conceitos de orientação a objetos. Veremos a seguir os fundamentos da OOP para JavaScript, sem, contudo, entrar em considerações ou diferenciações com OOP para outras linguagens de programação, pois este não é o escopo deste livro.

OOP é um paradigma (estilo) de programação que usa analogia com objetos do mundo real e suas características para definir estruturas de dados agrupados em campos ou métodos, bem como a interação entre esses dados, com a finalidade de desenvolver aplicações e escrever programas de computador.

A melhor maneira de entender o paradigma da programação OOP é mostrar a analogia com objetos do mundo real. Para nosso exemplo, vamos considerar o objeto: pessoa.

Objetos possuem propriedades e métodos.

Propriedades são valores associados ao objeto. Algumas propriedades do nosso objeto pessoa são altura, peso, sexo, cor da pele, naturalidade, profissão etc.

Métodos (ou funções) são ações que podem ser executadas pelo objeto. Alguns métodos do objeto pessoa são andar(), dormir(), comer(), trabalhar(), divertir() etc.

Para acessar as propriedades de um objeto, usamos a sintaxe mostrada a seguir:

```
altura = Pessoa.altura;  
sexo = Pessoa.sexo;  
profissao = Pessoa.profissao;  
...
```

Para acessar os métodos de um objeto, a sintaxe é conforme mostrada a seguir:

```
Pessoa.andar();  
Pessoa.dormir();  
Pessoa.trabalhar();  
...
```

Os objetos da linguagem JavaScript podem ser agrupados em três categorias distintas:

- objetos internos da linguagem, como, strings, arrays e datas;
- objetos do ambiente de hospedagem (por exemplo, o navegador), como, window e document;
- objetos personalizados criados pelo desenvolvedor.

Programação orientada a objetos e consequente criação de objetos não são o foco deste livro, contudo abordaremos esse assunto com maiores detalhes nos capítulos subsequentes, quando for necessário.

No momento, é suficiente que se entenda que objetos possuem propriedades e métodos acessados com o uso de sintaxe apropriada e saber também que a linguagem JavaScript possui objetos nativos com suas propriedades e métodos.



É de boa prática escrever o nome de objetos começando com letra maiúscula para diferenciá-los de outros tipos de dados da linguagem, como das variáveis. Isso permite localizar, com maior facilidade, objetos no script, facilitando a manutenção do código.

1.6.2 Caixas de diálogo

Caixa de diálogo é uma janela do tipo pop-up que se destina a fornecer informações ou coletar dados do usuário.

A linguagem JavaScript possui três métodos (ou funções), para o objeto Window, destinadas a criar três tipos de caixa de diálogo.

Vamos apresentar esses métodos, examinando como criar cada uma dessas caixas de diálogo que servirão de apoio para demonstrar o funcionamento dos exemplos que iremos desenvolver nos primeiros capítulos do livro.

Caixa de alerta

Alert Box, ou caixa de alerta, destina-se a colocar na interface do usuário uma caixa de diálogo contendo uma mensagem a ele. Conforme vimos anteriormente, a sintaxe para se chamar um método de um objeto JavaScript consiste em escrever o objeto e o método separados por um ponto. Quando se trata de uma propriedade ou método do objeto Window, podemos abreviar a sintaxe omitindo o objeto. Isso porque Window é o objeto JavaScript global ou raiz que contém todos os demais objetos do navegador. A seguir, veja a sintaxe para criar uma caixa de alerta:

```
window.alert("arg");
```

ou simplesmente:

```
alert("arg");
```

Esse método ou função admite um argumento — *arg* — que é uma string contendo a mensagem a ser apresentada ao usuário.

Observe o código mostrado a seguir:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>
<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <script type="text/javascript">
    alert("Alô Mundo!");
  </script>
</head>
<body>
</body>
</html>
```



[c1-alertbox.html]

Na figura 1.1, observe a caixa de alerta renderizada em diferentes navegadores.

As caixas de diálogo criadas com JavaScript têm sua apresentação determinada pelo fabricante do navegador, conforme podemos observar na figura 1.1.

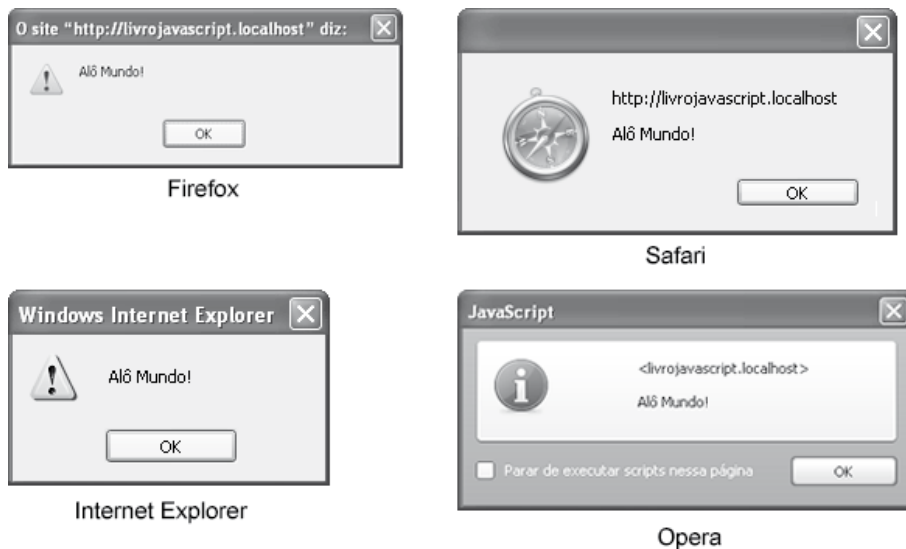


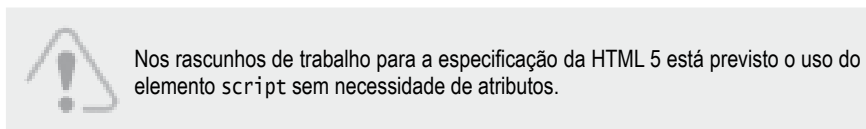
Figura 1.1 – Alert Box em diferentes navegadores.

Note no script anterior que para inserir JavaScript diretamente em uma página HTML usamos o elemento `script` e seu atributo `type`. A sintaxe para o elemento `script` prevê uma tag de abertura `<script>` e uma tag de fechamento `</script>` dentro das quais deverá ser inserido o código JavaScript.

Encontramos muitos códigos fazendo uso do atributo `language` para o elemento `script` como mostrado a seguir:

```
<script language="javascript1.2">
```

Esse atributo visa a identificar a linguagem em que foi escrito o script. Contudo, por não haver uma normatização dos identificadores de linguagens, o atributo está em desuso segundo as recomendações do W3C e não deverá ser empregado. Use o atributo `type` como mostrado anteriormente.



O script mostrado anteriormente consta de uma linha contendo um método (ou função) nativo da linguagem JavaScript e pertencente ao objeto `Window`, denominado `alert()`, que admite uma string como parâmetro. A string é o texto a ser apresentado na caixa de alerta e deverá ser escrita entre aspas duplas (" ") ou aspas simples (' '). A linha de declaração da função termina com ponto-e-vírgula.

Abra o arquivo [c1-alertbox.html], substitua a string por outra qualquer e veja o efeito no navegador.

Note que inserimos nosso script para a caixa de alerta na seção `head` do documento. Se retirarmos o script dessa seção e o inserirmos na seção `body`, irá funcionar normalmente. Experimente fazer isso alterando o arquivo e visualizando no navegador.

Mas, de acordo com o que vimos anteriormente, misturar scripts com a marcação HTML contraria os Padrões Web, ferindo os conceitos de JavaScript não obstrutivo e de separação das camadas de desenvolvimento. Assim, o script inserido na seção `body` não é apropriado e deve ser evitado.

Para a visualização imediata do funcionamento dos nossos scripts, iremos adotar inicialmente a prática de inseri-los diretamente na marcação HTML. Tão logo tenhamos passado uma base sólida que permita entender os princípios de funcionamento da linguagem, mostraremos a maneira correta de separar nossos scripts em um arquivo independente da marcação. Mas, desde já, esteja consciente de que script misturado com HTML não é uma prática recomendável.

Suponha que pretendemos apresentar ao usuário uma caixa de alerta com uma mensagem em duas linhas, como mostra a figura 1.2.

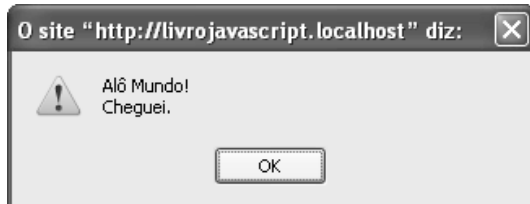


Figura 1.2 – Alert Box com mensagem em duas linhas.

Talvez você seja tentado a escrever o seguinte script:

```
<head>
...
<script type="text/javascript">
    alert("Alô Mundo!<br />Cheguei.");
</script>
</head>
...
```

Isso não vai funcionar, pois `
` será interpretado como uma string e não como um elemento HTML. Experimente fazer isso e visualize no navegador.

Felizmente, para resolver esse e outros problemas de formatação de strings, a linguagem JavaScript, tal como outras linguagens de programação, aceita os chamados caracteres de controle ASCII. Um deles é `\n`, que causa uma quebra de linha na string.

Assim, para pular uma linha na mensagem ao usuário passada com o uso de uma caixa de alerta, o código utilizado é o seguinte:

```
<head>
...
<script type="text/javascript">
    alert("Alô Mundo!\nCheguei.");
</script>
</head>
...
```

No item [1.8.5.] apresentamos uma tabela com os caracteres de controle ASCII para JavaScript.

Caixa de diálogo de confirmação

O método `confirm()` do objeto `Window` destina-se a colocar na interface do usuário uma caixa de diálogo contendo dois botões, normalmente denominados **OK** e **Cancelar**, e, ao clicar um deles, ele confirma ou cancela uma determinada ação.

Observe o código mostrado a seguir:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>
<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <script type="text/javascript">
        confirm("Você tem certeza que quer apagar o arquivo?\n
        Esta operação não poderá ser desfeita.");
    </script>
</head>
<body>
</body>
</html>
```

 [c1-confirm.html]

Na figura 1.3, observe a renderização da caixa de diálogo do tipo `confirm()` criada com o uso do script mostrado.

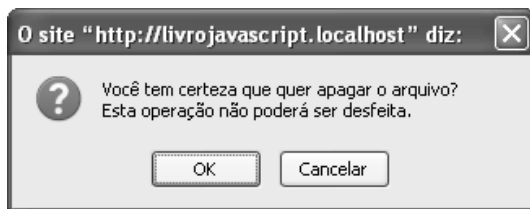


Figura 1.3 – Caixa de diálogo confirm.

Em uma situação real, o script mostrado deveria ser complementado, ficando em condições de receber a resposta do usuário e tomar uma decisão baseado na resposta. Apenas quando um dos botões é clicado, a linguagem fornece mecanismos internos capazes de identificar o botão clicado e manipular a resposta.

Caixa de diálogo para entrada de string

O método `prompt()` do objeto `Window` destinada-se a colocar na interface do usuário uma caixa de diálogo contendo um campo para que ele digite uma string. Essa função admite dois argumentos conforme mostrado a seguir:

```
prompt ("arg1", ["arg2"]);
```

Os argumentos devem estar entre aspas duplas ou simples e separados por vírgula, sendo o segundo argumento facultativo. O primeiro argumento, `arg1`, é uma instrução ao usuário para o que se espera que ele digite no campo de texto e o segundo argumento, `arg2`, um valor default, facultativo, inserido no campo de texto, normalmente para fornecer uma dica da forma de preenchimento do campo.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>
<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <script type="text/javascript">
    prompt("Olá visitante\nInforme a data do seu nascimento.", "dd/mm/aaaa");
  </script>
</head>
<body>
</body>
</html>
```



[c1-prompt.html]

Na figura 1.4, observe a renderização da caixa de diálogo do tipo `prompt()` criada com o uso do script mostrado.

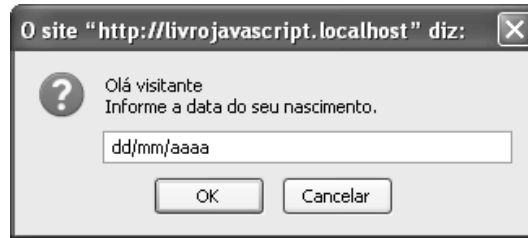


Figura 14 – Caixa de diálogo prompt.

Em uma situação real, o script mostrado deveria ser complementado, ficando em condições de receber a resposta do usuário e manipulá-la. Apenas quando um dos botões é clicado, a linguagem fornece mecanismos internos capazes de identificar a string entrada pelo usuário.

1.6.3 Escrever HTML com JavaScript

Vimos os três métodos do objeto Window destinados a criar caixas de diálogo. Vamos examinar a seguir o método `write()` do objeto Document cuja finalidade é escrever marcação HTML em um documento. Uma vez que Document é um objeto de Window, a sintaxe para esse método é mostrada a seguir:

```
window.document.write("arg1");
```

ou simplesmente:

```
document.write("arg1");
```

Esse método admite um argumento que é a marcação HTML a ser inserida no documento. Ao contrário do método `alert()`, aceita como argumento marcação HTML e não somente strings. Assim, veja o script a seguir:

```
document.write("<p>Aô mundo<br />Cheguei.</p>");
```

Causa a renderização de um parágrafo com uma quebra de linha. Esse método escreve diretamente na página, tal como se tivéssemos inserido a marcação no arquivo HTML. Cria conteúdo completamente inacessível para dispositivos não visuais, como um leitor de tela. Por essa razão, de acordo com os Padrões Web, usar esse método em scripts requer uma análise acurada e deve se restringir a casos muito específicos. Como regra geral, evite utilizá-lo.

Veja uma demonstração prática da inacessibilidade desse método. Considere o script a seguir:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```

<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>
<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
...
<script type="text/javascript">
    document.write("<h2>0 método <code>write()</code></h2>");
    document.write("<p>0 método <code>write()</code> do objeto <code>document</code>
        destina-se a escrever marcação HTML com uso da JavaScript</p>");
</script>
...
</body>
</html>

```

 [c1-write.html]

Na figura 1.5, veja a renderização da marcação HTML criada com o script e mostrada em arquivo disponível no site do livro.

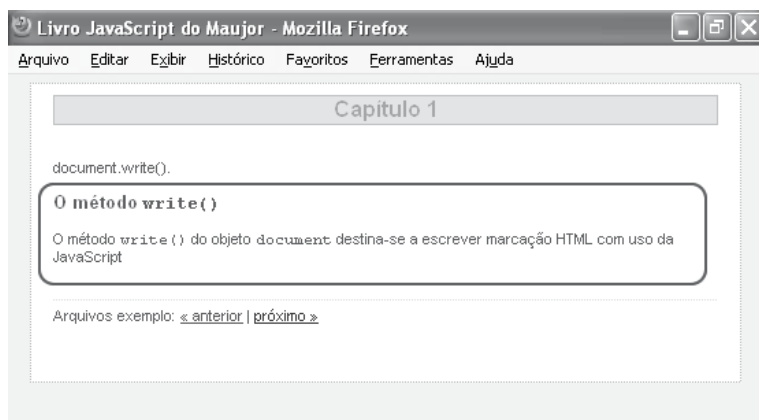



Figura 1.5 – Método write().

 O método write() só funciona como mostrado no exemplo anterior, quando inserido no fluxo do documento que está sendo processado. Se chamarmos esse método com o uso de link na seção head ou de um evento como veremos a seguir, em um documento já carregado, ele irá sobrescrever todo o HTML carregado.

O método writeIn() é semelhante ao método write(), com a diferença que acrescenta um caractere de nova linha ao final do conteúdo inserido. Em marcação HTML, pular uma linha não causa renderização em nova linha (sabemos que para pular uma linha, devemos usar o elemento br). Assim, o efeito prático de writeIn() é o mesmo de write().

1.6.4 Atrelar um evento com JavaScript

Outra funcionalidade da linguagem que usaremos para servir de apoio aos exemplos diz respeito a eventos. Apresentaremos sumariamente o objeto evento mostrando os princípios básicos do seu uso e, em capítulos posteriores, iremos aprofundar os conceitos.

Genericamente falando, podemos dizer que eventos são ações capazes de disparar uma reação. Veja este exemplo: clicar um link ou colocar o ponteiro do mouse sobre um elemento são eventos. Ao clicar um link, desencadeamos uma reação que poderá ser a de abrir uma nova página. Ao colocar o ponteiro do mouse sobre um elemento, podemos, por exemplo, alterar a opacidade dele.

Eventos são muito usados em JavaScript e viabilizam a interatividade em uma página Web. Na verdade, eventos viabilizam a própria existência da JavaScript. Sem eles, não teríamos como fazer funcionar os scripts.

Neste capítulo, não iremos aprofundar o estudo de eventos. Mostraremos apenas os eventos denominados `onclick` e `onload` que ocorrem respectivamente quando o usuário clica um elemento qualquer da página (não necessariamente um link) e quando um documento é carregado. Eventos podem ser disparados pelo usuário, como é o caso de `onclick`, ou ocorrer independentemente da interferência do usuário, como é o caso de `onload`.

Observe o script a seguir e, antes de visualizar o seu funcionamento no site do livro, tente entendê-lo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>
<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<button type="button" onclick="alert('Caixa de alerta');">Alert</button>
<button type="button" onclick="confirm('Caixa de confirmação');">Confirm</button>
<button type="button" onclick="prompt('Caixa de dados');">Prompt</button>
</body>
</html>
```



[c1-onclick.html]

Observe que atrelamos o evento `onclick` a três botões que, ao serem clicados, criarão, cada um deles, uma caixa de diálogo. Note que os eventos previstos na linguagem JavaScript são atributos da HTML cujo valor determina a ação a ser disparada. Ao usarmos eventos como atributos de um elemento HTML, estamos misturando as camadas de comportamento com a camada de estruturação e esta não é uma prática em conformidade com os Padrões Web. Em capítulos posteriores, iremos mostrar como fazer a separação.

No exemplo a seguir, vamos demonstrar o uso do método `write()` sendo chamado a partir de um evento, após a página ter sido carregada, para ilustrar o que dissemos no final do item [1.6.3]:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>
<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div id="tudo">
<h1>Capítulo 1</h1>
<p class="xpto" id="teste">document.write() - Após carregamento da página.</p>
<p class="nav-arq" id="referencia">Arquivos exemplo: <a href="c1-onclick.html">
&laquo; anterior</a> | <a href="c1-onclick.html">próximo &raquo;</a></p>
<button type="button"
  onclick="document.write('Conteúdo inserido após carregamento da página')">
  Inserir conteúdo</button>
</div>
</body>
</html>
```



[c1-write-apos-carregamento.html]

Note que ao clicar o botão para executar o método `write()` todo o conteúdo da página é substituído pelo conteúdo inserido.

1.7 Inserir JavaScript na HTML

Para escrever código JavaScript, não há necessidade de instalar um software especial. Escrevemos JavaScript com um editor de textos simples e visualizamos o resultado em um navegador. Arquivos escritos com o uso dessa linguagem devem ser gravados com a extensão `.js` e são criados para serem executados dentro de um arquivo HTML.

Então, scripts devem ser atrelados a documentos HTML. Existem três maneiras de se inserir JavaScript em um documento HTML:

- **Inline:** inserir o script diretamente na seção `body` do documento. Trata-se de prática não recomendada de acordo com o princípio da separação das camadas de desenvolvimento.
- **Incorporado:** inserir o script na seção `head` do documento.
- **Externo:** escrever o script em um arquivo externo e inserir com um link na seção `head` do documento.

Inserir script inline é uma prática que pertence ao passado e deve ser evitada pelo desenvolvedor comprometido com os Padrões Web. Ao escrevermos scripts dentro da marcação HTML, estaremos misturando as camadas de marcação e comportamento, dificultando a manutenção e o entendimento dos códigos.



Todos os scripts que mostramos anteriormente foram inseridos na marcação HTML. Fizemos assim para facilitar a demonstração das funcionalidades básicas da linguagem e o entendimento dos exemplos ilustrativos. Com a sequência de estudos, iremos mostrar como fazer a separação.

Exemplo de script inline:

```
<button type="button" onclick="alert('Olá visitante!')">
```

Incorporar script ao documento pode ser uma prática tolerável em situações muito particulares, como quando se trata de um só documento utilizando um script exclusivo.

Exemplo de script incorporado:

```
...  
<head>  
...  
  <script type="text/javascript">  
    // script vai aqui  
  </script>  
</head>  
...
```

As modernas práticas de desenvolvimento preconizam o uso de scripts externos a serem linkados ao documento. Cria-se um ou mais arquivos contendo os scripts e gravados com a extensão `.js` e usa-se o elemento `script` na seção `head` do documento para linká-lo à página. Essa técnica permite linkar o script a várias páginas do site, facilitando a manutenção.

Exemplo de script externo:

```
...  
<head>  
...  
  <script type="text/javascript" src="scripts/meu_script.js"></script>  
</head>  
...
```

Note o uso do atributo `src` para indicar o caminho para o arquivo que contém o script.

1.8 Sistema léxico da JavaScript

Entende-se por sistema léxico de uma linguagem o conjunto de palavras que compõem a linguagem. Neste item, iremos estudar a estrutura léxica da JavaScript, bem como as regras gramaticais e sintáticas para escrever scripts.

1.8.1 Tamanho de caixa

A linguagem JavaScript é sensível ao tamanho de caixa (case sensitive). Isso significa que nomes de variáveis, funções e demais identificadores são diferenciados quando escritos com letras maiúsculas ou minúsculas. Por exemplo: as variáveis `total`, `Total`, `toTa1` e `TOTAL` são diferentes. O método `write()` deve ser escrito em minúscula, pois escrever `Write()` ou `WRITE()` causará um erro no script.

Observe a marcação HTML a seguir:

```
<button type="button" onClick="alert('Olá visitante!')">
```

Estamos atrelando o evento `onClick`, que, na sintaxe JavaScript, deve ser grafado em minúsculas, a um botão HTML. Mas, na marcação, a grafia é `onClick` e não causará erro no script.

Na marcação, estamos escrevendo HTML (`onClick` é um atributo HTML) e sabemos que HTML não é sensível ao tamanho da caixa, portanto é válida a grafia apresentada. Contudo, no script, deveremos usar `onclick`, caso contrário causaremos um erro. Isso pode parecer um pouco confuso, mas estamos escrevendo em linguagens diferentes, uma case insensitive – HTML – e outra case sensitive – JavaScript.

Convém notar que se a marcação for XHTML e não HTML, então o atributo `onclick` terá que ser escrito também em minúscula, pois a linguagem XHTML, por ser compatível com XML, adota a sintaxe desta, na qual a marcação deve ser escrita em minúsculas.

1.8.2 Comentários

Comentários são pequenos textos que o desenvolvedor insere ao longo do script com a finalidade de facilitar o entendimento e a manutenção do código. A linguagem JavaScript admite três tipos de marcadores para comentários:

- Comentário em linha única (variante 1):

```
<script type="text/javascript">  
    // caixas de diálogo da JavaScript  
    alert("Alô Mundo!");           // caixa de alerta  
    confirm("Você tem certeza?");  // caixa de confirmação  
    prompt("Entre seu CPF");       // caixa de dados  
</script>
```

- Comentário em linha única (variante 2):

```
<script type="text/javascript">  
    <!-- caixas de diálogo da JavaScript  
    alert("Alô Mundo!");           <!-- caixa de alerta  
    confirm("Você tem certeza?");  <!-- caixa de confirmação  
    prompt("Entre seu CPF");       <!-- caixa de dados  
</script>
```

- Comentário em múltiplas linhas:

```
<script type="text/javascript">  
    /* Caixas de diálogo da JavaScript  
    - caixa de alerta  
    - caixa de confirmação  
    - caixa de dados */  
    alert("Alô Mundo!");  
    confirm("Você tem certeza?");  
    prompt("Entre seu CPF");  
</script>
```

Use duas barras // para iniciar um comentário em linha única ou coloque as múltiplas linhas de comentário entre os sinais /* e */.

Note que a variante 2, para marcar comentários em uma linha, usa uma sintaxe semelhante à adotada para comentários na marcação HTML, com a diferença de que não é necessário o sinal de fechamento como mostrado a seguir:

```
<!-- Aqui comentário para a marcação HTML -->  
<!-- Aqui comentário para JavaScript
```

Essa variante normalmente não é usada e consta aqui apenas como informação. Use a variante 1 para comentários em uma linha.

1.8.3 Declarações

Um script consiste em uma série de instruções escritas segundo uma sintaxe própria e rígida. As instruções, escritas em uma sequência lógica, determinam a realização de tarefas com a finalidade de obter um resultado final.

Cada uma das instruções de um script constitui uma declaração independente e existem duas sintaxes para separar uma declaração da outra. Separe-as com ponto-e-vírgula ou coloque cada declaração em uma linha separada. Observe a seguir:

```
// declarações na mesma linha
a = 5; b = 8; alert("Alô Mundo!");

// declarações em linhas separadas
a = 5
b = 8
alert("Alô Mundo!")
```

A sintaxe determina que o ponto-e-vírgula seja obrigatório para separar declarações em uma mesma linha e facultativo para separar declarações em linhas diferentes, contudo é uma boa prática usar ponto-e-vírgula para separar declarações em linhas diferentes.

```
// declarações em linhas separadas - recomendado usar ponto-e-vírgula para separar
a = 5;
b = 8;
alert("Alô Mundo!");
```

1.8.4 Espaços em branco e quebras de linha

Quebras de linhas e espaços em branco, quando inseridos entre nomes de variáveis, nomes de funções, números e entidades similares da linguagem, são ignorados na sintaxe JavaScript. Contudo, para strings e expressões regulares, tais espaçamentos são considerados. Vejamos alguns exemplos que esclarecem o uso desses espaços:

- As duas sintaxes a seguir são válidas:

```
a=27; e a = 27;
alert("Olá") e alert ( "Olá" )
function(){...} e function () {...}
```

- As sintaxes a seguir causam um erro no script:

```
a = 2 7;          // espaço entre os algarismos de um número não é permitido
document.write("<p> Eu sou
uma string</p>")   // quebra de linha em uma string
```

- A sintaxe a seguir é válida:

```
document.write("<p> Eu sou \\  
uma string</p>")    // uso de \ para quebrar linha em string
```

- A sintaxe a seguir causa um erro no script:

```
document.write \  
("<p> Eu sou uma string</p>")    // uso de \ para quebrar linha fora de string
```

- A sintaxe a seguir é válida:

```
document.write  
("<p> Eu sou uma string</p>")    // quebra de linha entre nome de funções
```

1.8.5 Literais

Na terminologia JavaScript, a palavra literal designa qualquer dado – valor fixo (não variável) – que se insere no script. Nos exemplos a seguir, os valores 45 e Alô Mundo! são literais:

```
a = 45;  
mensagem = "Alô Mundo!";
```

Existem seis tipos de dados literais conforme descritos a seguir:

- inteiros;
- decimais;
- booleanos;
- strings;
- arrays;
- objetos.

Inteiros

Os literais inteiros, na sintaxe JavaScript, são os números inteiros escritos em base decimal (base 10), hexadecimal (base 16) ou octal (base 8). A base octal está em desuso segundo a especificação ECMA-262, embora seja válida na versão JavaScript 1.5.

A base 10 é o nosso conhecido sistema de numeração e os números inteiros nela são aqueles do conjunto \mathbb{Z} da matemática que aprendemos no Ensino Médio: $\mathbb{Z} = \{\dots -2, -1, 0, 1, 2, \dots\}$ ou conjunto dos números inteiros positivos e negativos mais o número zero.

A base hexadecimal utiliza 16 símbolos (base 16) para representar os números. Nela, os símbolos são os dez números de 0 a 9 mais as seis primeiras letras do alfabeto A a F. Aqueles familiarizados com CSS certamente conhecem a declaração de cores com o uso do sistema hexadecimal cuja sintaxe é mostrada a seguir:

```
div { background: #FFD38A; }
```

Outros exemplos de número hexadecimal: F2, b43, EC65D1. Na sintaxe CSS, números hexadecimais deverão ser precedidos do sinal # (tralha). Na sintaxe JavaScript, números hexadecimais deverão ser precedidos de 0x (zero xis) e os exemplos anteriores, em sintaxe JavaScript, serão escritos: 0xF2, 0xb43, 0xEC65D1.

Exemplo de declarações usando o literal inteiro:

```
c = 32;           // base 10
d = -119;         // base 10
e = 0x110B6;      // base hexadecimal
f = 0xf56a2;      // base hexadecimal
```

Note que quando se trata de sintaxe na base hexadecimal, as letras A a F podem ser escritas tanto em maiúsculas quanto em minúsculas.

Decimais

Os literais decimais, na sintaxe JavaScript, são os números constituídos por um número inteiro e um número fracionário. As casas decimais são separadas por um ponto (.), como mostrado nos exemplos a seguir:

```
a = 3.1416;
b = -76.89;
c = .33333;
```

A sintaxe JavaScript admite a notação científica (ou notação exponencial) para escrever tanto literais inteiros como literais fracionários. Notação científica é uma maneira alternativa de representar números compostos por muitos algarismos e consiste em acrescentar a letra E (maiúscula) ou e (minúscula) a um número para indicar expoentes de 10. Observe os exemplos a seguir;

```
a = 3E5;           // equivale a 3x10**5 = 300000
b = -47.78965432E10; // equivale a -47.78965432x10**10 = 477896543200
```

Booleanos

Os literais booleanos, na sintaxe JavaScript, são as palavras-chave `true` e `false` (minúsculas) destinadas a definir as condições de verdadeiro e falso, respectivamente, para uma variável ou equivalente, como mostrado a seguir:

```
a = true;
b = false;
```

Booleanos são amplamente usados em estruturas de controle com a finalidade de testar a veracidade (ou validade) de uma determinada condição, permitindo ao script tomar uma decisão baseado na condição de falso ou verdadeiro da condição. O exemplo a seguir esclarece em linguagem corrente o uso mais comum dos booleanos em programação:

```
a = 10;
se a < 10 faça isto se não faça aquilo;
```

Quando transformamos a lógica mostrada em sintaxe JavaScript, a condição de teste da variável `a` retorna `true` sempre que `a` for menor que 10 e `false`, caso contrário. Em capítulos posteriores, teremos oportunidade de usar com frequência os booleanos e eventuais dúvidas sobre eles serão esclarecidas.

Strings

Os literais strings, na sintaxe JavaScript, são os conjuntos de zero ou mais caracteres envolvidos por aspas duplas (`" "`) ou aspas simples (`' '`). Strings são sequências de caracteres. Observe os exemplos a seguir:

```
nome = "Maurício Samy Silva";           // string com aspas duplas
outroNome = 'Pedro Nascimento Júnior';  // string com aspas simples
```

Podemos obter alguma formatação básica ao escrevermos strings com o uso de caracteres especiais para a linguagem JavaScript. No exemplo a seguir:

```
mensagem = "Obrigado pela visita.\nVolte em breve.";
```

Inserimos o caractere `\n` na string. Ele faz com que haja o pulo de uma linha quando renderizada a string no navegador, resultando em algo como:

```
Obrigado pela visita.
Volte em breve.
```

Na tabela 1.2, observe alguns dos caracteres especiais da linguagem e a sintaxe geral para caracteres Latin-1 e Unicode.

Tabela 1.2 – Tabela de caracteres especiais da JavaScript

Caractere	Descrição – Caractere Unicode hexadecimal
\b	Backspace – \u0008
\f	Form feed – \u000C
\n	Nova linha – \u000A
\r	Retorno do carro – \u000D
\v	Tabulação vertical – \u000B
\t	Tabulação horizontal – \u0009
\'	Apóstrofo ou aspas simples – \u0027
\"	Aspas duplas – \u0022
\\	Barra invertida – \u005C
\XXX	Caractere Latin-1 expresso por dígitos octais de 1 a 377
\xXX	Caractere Latin-1 expresso por dois dígitos hexadecimais de 00 a FF
\uXXXX	Caractere Unicode expresso por quatro dígitos hexadecimais

Nas três últimas linhas da tabela 1.2, consta a sintaxe geral para a representação de caracteres em strings nos sistemas octal e hexadecimal. O sistema octal está em desuso e, assim, deve ser evitado.

O sistema hexadecimal admite uma sintaxe para Latin-1 e uma sintaxe para Unicode. Para consultar as tabelas de caracteres, visite o site <http://www.unicode.org/>. Lá, você encontrará toda a codificação Unicode nos seus variados formatos. Para demonstrar o funcionamento da codificação, mostramos a seguir três caracteres e seus respectivos códigos e exemplos de uso na sintaxe JavaScript (Tabela 1.3).

Tabela 1.3 – Caracteres Unicode na sintaxe JavaScript

Caractere	Octal	Latin-1 hexadecimal	Unicode hexadecimal
© (Copyright)	\251	\xA9	\u00A9
® (Marca registrada)	\256	\xAE	\u00AE
¶ (Parágrafo)	\247	\xA7	\u00A7

Observe no script a seguir que utilizamos as três sintaxes mostradas na tabela 1.3 para obter a renderização dos caracteres em uma caixa de alerta:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>
<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<link href="../../main.css" rel="stylesheet" type="text/css" />
```

```

<script type="text/javascript">
  var executar = function() {
    var caractere = "\
Octal:.....\251, \256, \266 \n\
Latin 1 Hexadecimal:....\xA9, \xAE, \xB6 \n\
Latin 1 Unicode:.....\u00A9, \u00AE, \u00B6"
    alert(caractere);
  }
</script>
</head>
<body>
<div id="tudo">
<h1>Capítulo 1</h1>
  <p class="xpto" id="teste">Caracteres especiais.</p>
  <p class="nav-arq" id="referencia">Arquivos exemplo:
    <a href="c1-write-apos-carregamento.html">&laquo; anterior</a> |
    <a href="c1-#.html">próximo &raquo;</a></p>
    <button type="button" onclick="executar()">Mostrar caracteres</button>
  </div>
</body>
</html>

```



[c1-caracteres.html]

No arquivo [c1-caracteres.html], no site do livro, ao clicar o botão para mostrar os caracteres, será apresentada uma caixa de alerta conforme mostra a figura 1.6.

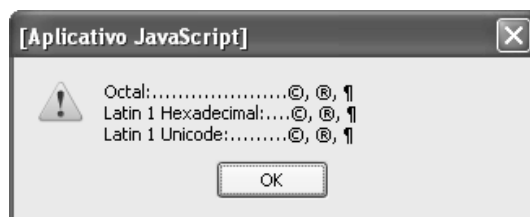


Figura 1.6 – Renderização de caracteres.

Na tabela 1.2, note que os caracteres ali mostrados têm seus equivalentes Unicode hexadecimal. Por exemplo, \u000A equivale a \n.

Caractere de escape

Na sintaxe para a codificação de caracteres que acabamos de estudar, podemos notar a presença de uma barra invertida “\” no início de cada caractere, tal como \n, \t, \251, \xAE, \u00AE etc.

A barra invertida tem um emprego específico na escrita de strings da JavaScript. É conhecida como caractere de escape e usada para representar caracteres especiais como mostrado e também para escapar caracteres normalmente não permitidos dentro de uma string. O caractere de escape, barra invertida, é ignorado dentro da string.

Se escrevermos uma string contendo um caminho no diretório C do HD como mostrado a seguir:

```
alert("C:\\localhost\\index.php");
```

Iremos obter um resultado conforme o mostrado na figura 1.7, o que obviamente não é o que estamos esperando.

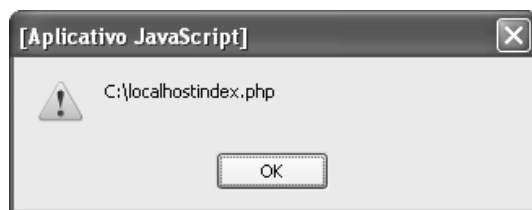


Figura 1.7 – Caractere de escape.

Isso porque o primeiro caractere de escape foi ignorado e escapou, o segundo que aparece depois de C: e o terceiro caractere de escape entre localhost e index.php também foram ignorados. Para resolver isso, devemos escapar todas as barras invertidas que aparecem na string como mostrado a seguir:

```
alert("C:\\\\localhost\\\\index.php");
```

Sabemos que strings devem ser marcadas com aspas duplas (") ou aspas simples ('). Mas o que ocorre quando uma string contém um texto entre aspas ou apóstrofo? Considere as strings a seguir:

- Os livros da editora O'Reilly são muito bons.
- Mac'Neil disse: "Vim para ficar".

As seguintes sintaxes são válidas para escrever as strings mostradas:

```
"Os livros da editora O'Reilly são muito bons." // aspas duplas fora e simples dentro  
'Os livros da editora O'Reilly são muito bons.' // aspas simples fora e simples escapadas dentro  
'Mac\\Neil disse: "Vim para ficar"' // aspas simples fora, duplas e simples escapadas dentro  
"Mac'Neil disse: \\\"Vim para ficar\\\"." // aspas duplas fora, duplas escapadas e simples dentro
```

Como regra geral, aspas que envolvem a string e estão contidas na string devem ser de natureza (simples e dupla) oposta. Se forem de mesma natureza, deveremos usar o caractere de escape nas internas.

Arrays

Os literais arrays, na sintaxe JavaScript, são os conjuntos de zero ou mais valores, separados por vírgula e envolvidos por colchetes ([]). Os valores contidos em um array recebem um índice sequencial começando com zero. A sintaxe para criar um array literal é mostrada a seguir:

```
frutas = ["laranja", "pera", "goiaba", "morango"];
```

Para recuperarmos os valores de um array, usamos a sintaxe composta pelo nome do array seguida de um índice, entre colchetes, como mostra o exemplo a seguir:

frutas[0] contém o valor laranja.

frutas[3] contém o valor morango.

Os valores do array que mostramos são strings e devem ser escritos entre aspas como estudamos anteriormente. Um array pode conter qualquer tipo de dado da JavaScript, incluindo expressões, objetos e outras arrays. Por exemplo:

```
ArrayMisto = ["laranja", 34, "casa", true, [1,3,5], a+b];
```

Veja no site do livro o exemplo a seguir, que mostra a extração de dados de um array:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>
<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<link href="../main.css" rel="stylesheet" type="text/css" />
<script type="text/javascript">
  a = 4; b = 12;
  ArrayMisto = [ "laranja", 34, "casa", true, [1,3,5], a+b ];
  var executar = function() {
    alert(ArrayMisto[0]);
    alert(ArrayMisto[1]);
    alert(ArrayMisto[3]);
    alert(ArrayMisto[4][1]);
    alert(ArrayMisto[5]);
  }
</script>
</head>
<body>
<div id="tudo">
```



```

<h1>Capítulo 1</h1>
<p class="xpto" id="teste">Literais arrays.</p>
<p class="nav-arq" id="referencia">Arquivos exemplo: <a href="c1-caracteres.html">&laquo;
anterior</a> | <a href="c1-objetos.html">próximo &raquo;</a></p>
<button type="button" onclick="executar()">Mostrar array</button>
</div>
</body>
</html>

```



[c1-arrays.html]

Objetos

Os literais objetos, na sintaxe JavaScript, são os conjuntos de pares nome/valor separados por vírgula e envolvidos por chaves ({ }). O par nome/valor pode ser uma propriedade do objeto e seu respectivo valor, um método do objeto e seu valor (normalmente uma função) ou mesmo outro objeto. A sintaxe para criar um objeto literal é mostrada a seguir:

```

Carro =
{
  marca: "Renault",      // propriedade/valor
  modelo: "Logan",       // propriedade/valor
  ipva: "valor('rb15')", // propriedade/método
  dimensoes:             // propriedade/objeto
  {
    c: "4.250mm",
    l: "1.735mm",
    h: "1.525mm"
  }
};

```

O objeto Carro mostrado possui as propriedades *marca*, *modelo* e *dimensoes*. A propriedade *dimensoes* é um objeto com as propriedades *c*, *l* e *h* que expressam o comprimento, a largura e a altura do carro. O objeto possui ainda um método de nome *ipva*, que é uma função denominada *valor()*, que admite um argumento de cálculo *rb15* destinado a calcular o valor do IPVA.

Para extrair os dados de um objeto, usa-se a sintaxe *propriedade.valor*. Observe o exemplo a seguir disponível para consulta no site do livro:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
<head>

```

```

<title>Livro JavaScript do Maujor</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<link href="../main.css" rel="stylesheet" type="text/css" />
<script type="text/javascript">
    Carro =
    {
        marca: "Renault",
        modelo: "Logan",
        ipva: "valor('rb15')",
        dimensoes:
        {
            c: "4.250mm",
            l: "1.735mm",
            h: "1.525mm"
        }
    }
    marca = Carro.marca;
    modelo = Carro.modelo;
    comprimento = Carro.dimensoes.c;

    executar = function() {
        alert(marca);
        alert(modelo);
        alert(comprimento);
    }
</script>
</head>
<body>
<div id="tudo">
<h1>Capítulo 1</h1>
<p class="xpto" id="teste">Literais objetos.</p>
<p class="nav-arq" id="referencia">Arquivos exemplo: <a href="c1-caracteres.html">&laquo;
anterior</a> | <a href="c1-#.html">próximo &raquo;</a></p>
<button type="button" onclick="executar()">Mostrar propriedades</button>
</div>
</body>
</html>

```



[c1-objetos.html]

1.8.6 Seção CDATA

Seções CDATA se destinam a esconder do parseador XML blocos de texto que contenham caracteres de dados ou marcação.



Parseador XML é um processador nativo do navegador (Internet Explorer não tem) cuja finalidade é transformar uma marcação XML em um objeto DOM.

Na sintaxe XML caracteres tais como &, " e ' devem ser codificados para & " e ' respectivamente. O mesmo ocorre com o caractere < quando usado de outra maneira que não para indicar abertura de tag, caso em que deverá ser codificado para <.

Considere que a marcação a seguir está inserida em um documento XML ou XHTML.

```
<p>0 sinal matemático < indica menor que</p>
```

Essa marcação é ilegal segundo a sintaxe XML, pois o parseador ao encontrar o caractere < inserido no meio do parágrafo o interpretará como uma abertura de tag e não encontrando o respectivo caractere de fechamento da tag gerará um erro.

A solução é usar codificação como mostrado a seguir.

```
<p>0 sinal matemático &lt; indica menor que</p>
```

Scripts JavaScript podem conter, e normalmente contém, caracteres ilegais segundo a sintaxe XML. Se tais scripts forem inseridos diretamente no documento (não linkados), serão parseados e gerarão erro XML. Observe a seguir.

```
<script type="text/javascript">
  if (a<8) {
    alert("a é menor do que 8");
  }
</script>
```

A presença do sinal <, menor que, no script causa um erro XML. Se o documento for servido com DOCTYPE XML simplesmente ele não funcionará. Se for servido com DOCTYPE XHTML ele não será validado.

O uso de seções CDATA em JavaScript resolvem o problema tornando o código invisível para o parser XML e portanto válido. Observe a seguir um exemplo demonstrando a sintaxe e uso de seções CDATA em JavaScript.

```
<script type="text/javascript">
  // <![CDATA[
  if (a < 8) {
    alert("a é menor do que 8");
  }
  // ]]>
</script>
```

Se o documento usar um DOCTYPE para HTML não é necessário o uso de seções CDATA pelo fato de que HTML não segue a sintaxe XML.

Por mais estranho que possa parecer a grafia para a seção CDATA deve ser em maiúscula embora a sintaxe XML imponha minúsculas.

1.9 Variáveis

Variável é um nome qualquer ao qual se atribui um valor ou dado. Uma variável pode conter uma string, um número inteiro, um número fracionário, um array, um booleano, uma função, um objeto etc. Variáveis são também chamadas de identificadores.

O nome da variável é de livre escolha do programador, ressalvadas as seguintes restrições sintáticas: o nome da variável deve começar com um dos três seguintes caracteres: uma letra, um caractere underscore (_) ou um caractere cifrão (\$). Os caracteres que se seguem podem ser letras maiúsculas ou minúsculas (A-Z ou a-z), números (0-9), underscores (_) ou caractere cifrão (\$).

É interessante não utilizar o caractere cifrão (\$) para não haver confusão com códigos específicos de bibliotecas JavaScript, pois é amplamente empregado na sintaxes desses frameworks.

A partir da versão JavaScript 1.5, permite-se o uso de letras acentuadas, dígitos, caracteres escapados e demais caracteres Unicode em nome de variáveis.

Exemplos de nomes de variáveis:

```
a
_xpto
minha_variavel
minhaVariavel
minha-variavel
cd456
```

Embora a sintaxe para escrever variáveis permita uma infinidade de combinações, e o programador é livre para escolher aquela que bem entender, algumas regras práticas consagradas pelo uso devem ser seguidas não em caráter obrigatório, mas como uma técnica de boas práticas de programação. Procure adotar as seguintes regras ao escolher nomes para variáveis:

- Escolha nomes que transmitam uma dica do conteúdo da variável. Por exemplo: `qdeLivros` para uma variável destinada a armazenar uma quantidade de livros.

- Evite nomes de variáveis como `a`, `f34`, `variavelParaArmazenarUmObjeto`, pois são muito sucintas ou verbosas. Use as letras `i`, `j`, `k`,... para nomear variáveis contadoras.
- Variáveis nomeadas com um nome composto podem ser escritas de várias formas. Por exemplo: `nomevendedor`, `nome-vendedor`, `nome_vendedor` e `nomeVendedor` são todos nomes válidos e podemos escolher qualquer um deles. As formas mais usadas são `nome_vendedor` e `nomeVendedor`, e essa última usa a grafia conhecida como CamelCase. O importante é que o desenvolvedor escolha uma das formas mostradas ou mesmo outra forma válida por ele inventada e adote-a para desenvolver todos os seus projetos, mantendo a consistência de escrita.
- Há, ainda, restrição final para escolher nomes de variáveis. Algumas palavras fazem parte da sintaxe da linguagem JavaScript e não podem ser usadas para nomear variáveis. Tais palavras estão agrupadas em três categorias distintas, a saber: palavras próprias e internas da JavaScript (Tabela 1.4), palavras reservadas pelas especificações ECMA-262 para uso em futuras versões da JavaScript (Tabela 1.5) e palavras que fazem parte da implementação JavaScript nos dispositivos que hospedam a linguagem, como um navegador, por exemplo (Tabela 1.6).

Tabela 1.4 – Palavras-chave JavaScript

<code>break</code>	<code>else</code>	<code>new</code>	<code>var</code>
<code>case</code>	<code>finally</code>	<code>return</code>	<code>void</code>
<code>catch</code>	<code>for</code>	<code>switch</code>	<code>while</code>
<code>continue</code>	<code>function</code>	<code>this</code>	<code>with</code>
<code>default</code>	<code>if</code>	<code>throw</code>	
<code>delete</code>	<code>in</code>	<code>try</code>	
<code>do</code>	<code>instanceof</code>	<code>typeof</code>	

Tabela 1.5 – Palavras reservadas pela especificação ECMA-262

<code>abstract</code>	<code>enum</code>	<code>int</code>	<code>short</code>
<code>boolean</code>	<code>export</code>	<code>interface</code>	<code>static</code>
<code>byte</code>	<code>extends</code>	<code>long</code>	<code>super</code>
<code>char</code>	<code>final</code>	<code>native</code>	<code>synchronized</code>
<code>class</code>	<code>float</code>	<code>package</code>	<code>throws</code>
<code>const</code>	<code>goto</code>	<code>private</code>	<code>transient</code>
<code>debugger</code>	<code>implements</code>	<code>protected</code>	<code>volatile</code>
<code>double</code>	<code>import</code>	<code>public</code>	<code>public</code>

Tabela 1.6 – Palavras reservadas dos dispositivos de hospedagem

alert	eval	location	open
array	focus	math	outerHeight
blur	function	name	parent
boolean	history	navigator	parseFloat
date	image	number	RegExp
document	isNaN	object	status
escape	length	onLoad	string



Lembre-se de que JavaScript é sensível ao tamanho da caixa (case sensitive). Assim, as variáveis nomevendedor, nome-vendedor, nome_vendedor e nomeVendedor são diferentes.

1.9.1 Declarar variáveis

JavaScript é uma linguagem de programação não tipificada, pois, ao contrário da maioria das linguagens de programação, as variáveis da JavaScript podem conter qualquer tipo de dado. Observe o exemplo a seguir:

```
...  
minhaVariavel = "Alô Mundo";  
// aqui script  
minhaVariavel = 235;  
// aqui mais script  
...
```

Note que a variável `minhaVariavel` começa com um valor do tipo `string` e, em um ponto posterior no script, a mesma variável assume um valor tipo numérico. Tal fato é legal em JavaScript, pois a linguagem reconhece o tipo de dado quando a variável é declarada. Na maioria das linguagens de programação, isso é ilegal, pois uma determinada variável deverá ser declarada para um tipo de dado e conter sempre esse tipo.

Um conceito importante ao declararmos uma variável é o chamado escopo da variável, que é o ambiente ou, mais precisamente, o trecho ou região do script no qual a variável assume o valor que foi para ela declarado.

Uma variável pertence ao escopo global e denomina-se variável global quando seu valor é reconhecido em qualquer trecho do script. Por outro lado, diz-se que uma variável é local e pertence ao escopo local quando seu valor é reconhecido somente no trecho do script no qual ela foi declarada.

Para declarar uma variável pertencente ao escopo local, usa-se a palavra-chave `var`, que é própria da sintaxe JavaScript. Declarar uma variável sem uso da palavra-chave `var` implica que a variável pertença ao escopo global. Observe a seguir:

```
var a = 30;    // a variável a pertence ao escopo local
b = 25;        // a variável b pertence ao escopo global
```

Vamos fixar a noção de escopo de uma variável com o uso do script mostrado a seguir. Caso você não esteja familiarizado com o funcionamento de funções na JavaScript, não se preocupe em entendê-lo agora. No capítulo 6, trataremos de funções com mais detalhes.

```
<script type="text/javascript">
  a = 10;
  funcaoUm = function() {
    var a = 20;
    alert(a);    // alerta o valor 20
  };
  funcaoUm();    // executa a funcaoUm

  funcaoDois = function() {
    alert(a);    // alerta o valor 10
  };
  funcaoDois();  // executa a funcaoDois
  // mais script ...
  alert(a);      // alerta o valor 10
</script>
```

Podemos identificar três regiões distintas no script mostrado. A região da `funcaoUm`, a região da `funcaoDois` e a região compreendendo todo o script (dentro e fora das duas funções).

Para a `funcaoUm`, o valor da variável `a` é 20, pois a declaramos no corpo da função com o uso da palavra-chave `var`. Trata-se de uma variável local cujo valor vale somente para a função.

Para a `funcaoDois`, não foi declarado um valor local para a variável `a`, portanto a função procura o valor da variável no escopo global e encontra o valor 10. Se não existisse uma variável `a = 10` no escopo global, o valor retornado para ela seria considerado indefinido (`undefined`), causando um erro no script.

Além das duas funções, o último alerta usa a variável do escopo global com valor 10.

Vamos introduzir uma modificação no exemplo anterior com a finalidade de esclarecer melhor o escopo de uma variável. A modificação consiste em declarar um novo valor para a variável `a` dentro da `funcaoDois`, mas sem uso da palavra-chave `var`, ou seja, trata-se de uma variável do escopo global definida no corpo de uma função.

```
<script type="text/javascript">
  a = 10;
  funcaoUm = function() {
    var a = 20;
    alert(a);    // alerta o valor 20
  };
  funcaoUm();    // executa a funcaoUm

  funcaoDois = function() {
    a = 40;
    alert(a);    // alerta o valor 40
  };
  funcaoDois();  // executa a funcaoDois
  // mais script ...
  alert(a);      // alerta o valor 40
</script>
```

A declaração do valor global – 40 – para a variável `a` dentro da `funcaoDois` sobrepõe o valor global – 10 – declarado anteriormente e, em consequência, neste caso, o alerta seguinte e o último alerta retornarão 40 e não mais 10.

A seguir, mais alguns exemplos esclarecendo o escopo de variáveis. Estude-os com atenção para entendê-los, pois os conceitos são os mesmos mostrados nos exemplos anteriores.

■ Exemplo 1:

```
<script type="text/javascript">
  funcaoUm = function() {
    a = 20;
    alert(a);    // alerta o valor 20
  };
  funcaoUm();    // executa a funcaoUm
  alert(a);      // alerta o valor 20
</script>
```

■ Exemplo 2:

```
<script type="text/javascript">
  funcaoUm = function() {
    var a = 20;
    alert(a);    // alerta o valor 20
  };
  funcaoUm();    // executa a funcaoUm
  alert(a);      // alerta o valor 10
</script>
```



```
};
funcaoUm();      // executa a funcaoUm
alert(a);        // retorna undefined
</script>
```

■ Exemplo 3:

```
<script type="text/javascript">
  funcaoUm = function() {
    var a = 20;
    alert(a);      // alerta o valor 20
    function funcaoAninhada() {
      var a = 40;  // alerta o valor 40
      alert(a);
    };
    funcaoAninhada(); // executa a funcaoAninhada
  };
  funcaoUm();      // executa a funcaoUm
</script>
```

■ Exemplo 4:

```
<script type="text/javascript">
  a = 40;
  funcaoUm = function() {
    alert(a);      // alerta undefined
    var a = 10;
    alert(a);      // alerta o valor 10
  };
  funcaoUm();      // executa a funcaoUm
</script>
```

O exemplo 4 expressa um conceito de escopo de variáveis que diz o seguinte: uma variável local cria, na região onde for declarada, um contexto para sua validade, independentemente de onde estiver posicionada na região. No exemplo 4, era de se esperar que o primeiro alerta resultasse no valor 40, pois existe uma variável `a` do escopo global definida anteriormente com esse valor. Contudo, como existe uma variável `a` local de valor 10, definida dentro da função, ela é válida dentro de toda a função e o primeiro alerta por estar antes da declaração da variável local “sabe” que foi criado um contexto de validade para `a`, mas não conhece seu valor, retornando indefinido.

Para evitar resultados inesperados como o mostrado no exemplo 4, é importante declarar as variáveis logo no início da função.

Em um script muito extenso desenvolvido individualmente ou em equipe, é muito difícil manter o controle sobre o nome das variáveis dele e a possibilidade de se escolher um mesmo nome para uma variável em diferentes locais dele deve ser considerada. Isso poderá ocasionar sobrescrição de variáveis anteriormente declaradas, causando resultados inesperados e, dependendo do contexto, difíceis de serem detectados pelo desenvolvedor. É interessante usar sempre a palavra-chave `var` ao se declarar variáveis, tornando-as todas variáveis locais.

Tal prática não é mandatória, mas sim altamente recomendada, ressalvados os casos em que declarar variáveis globais tenha sido uma decisão fundamentada em propósitos consistentes para o script e que garanta o pleno controle de tais variáveis ao longo de todo o processo de desenvolvimento.

Sintaxe para declarar variáveis

É importante declarar as variáveis na primeira linha de código da região para a qual ela será válida, mas não há uma sintaxe rígida para a forma de declará-las. Observe nos exemplos a seguir três formas diferentes de se declarar variáveis.

```
...  
var a = 40; var b = 60; var c = "A!ô Mundo!";  
  
var a = 40;  
var b = 60;  
var c = "A!ô Mundo!";  
  
var a = 40, b = 60, c = "A!ô Mundo!";  
...
```

A terceira forma é a mais simplificada e tem sido preferida pela maioria dos desenvolvedores, contudo você é livre para escolher a que lhe pareça mais apropriada e seguir com ela por todos os scripts que venha a desenvolver. Exclua-se os casos específicos nos quais talvez fosse melhor escolher outra forma com a finalidade, por exemplo, de tornar o código mais claro ou legível.

Note que na forma mais simplificada a palavra-chave `var` aparece somente na declaração da primeira variável e as declarações seguintes são separadas por vírgula. Nessa sintaxe, a palavra-chave `var` é válida para todas as variáveis da lista e, em consequência, todas elas são variáveis locais.

null, undefined, NaN, Infinity

Faremos uma introdução sumária ao significado de um valor e três propriedades da linguagem JavaScript com a finalidade de fornecer uma base mínima para compreender alguns exemplos que desenvolveremos na sequência deste capítulo e no capítulo 2. Posteriormente, detalharemos com mais precisão os conceitos estudados a seguir.

Na sintaxe JavaScript, `null` é uma palavra-chave que indica ausência de valor, ou um objeto sem um valor que o represente. Se uma variável é declarada `null`, não existe um objeto, um array, uma string, um literal ou qualquer outro valor a ela associado. No exemplo a seguir, a variável `a` foi declarada `null`:

```
var a = null;
```

Para representar um valor indefinido, a sintaxe JavaScript usa uma propriedade do objeto global denominada `undefined` (indefinido). Diz-se que uma variável assume o valor `undefined` quando é declarada e não inicializada, ou seja, não se atribui um valor a ela. No exemplo a seguir, a variável `a` é indefinida:

```
var a;
```

Para representar um valor que não seja um número, a sintaxe JavaScript usa uma propriedade do objeto global denominada `NaN`. No exemplo a seguir, a variável `a` é representada por `NaN`:

```
var a = 3 * "01á";
```

Para representar um valor infinito positivo, a sintaxe JavaScript usa uma propriedade do objeto global denominada `Infinity`. No exemplo a seguir, a variável `a` tem valor infinito. A faixa de números manipuláveis pelo interpretador JavaScript está compreendida entre:

```
-1.7976931348623157e+308 e 1.7976931348623157e+308
```

Os números mostrados estão escritos em notação científica. A notação científica é da forma $ae+b$ e $ae-b$ e é usada para representar números muito grandes ou muito pequenos. Observe um exemplo mostrando a equivalência de números na notação decimal e científica:

```
3.2e+4 = 3.2x104 = 32000
```

```
347e-5 = 347x10-5 = 0,00345
```

Valores abaixo desses números são considerados `-Infinity` e acima, `+Infinity`. Estudaremos com mais detalhes essas propriedades do objeto `Number` no capítulo 9.

```
var a = 12 + Infinity;
```

1.10 Variáveis e propriedades dos objetos

Fizemos uma introdução à sintaxe JavaScript para objetos no item [1.8.5]. Vimos que propriedades de um objeto nada mais são do que valores de uma variável do objeto. Nesse sentido, não há diferenças entre variáveis e propriedades dos objetos. Ambas são conceitualmente idênticas. O item a seguir esclarece melhor esse conceito.

1.11 Objeto global

Um ambiente de hospedagem da JavaScript possui, nativamente instalado, um interpretador da linguagem. Toda vez que se inicia um desses ambientes, ele cria um objeto global contendo todas as propriedades e métodos da linguagem e do ambiente de hospedagem.

Para esclarecer e identificar o objeto global, tomemos como exemplo um navegador web que é um ambiente de hospedagem para JavaScript.

Quando abrimos o navegador, cria-se um objeto global que, na sintaxe JavaScript, se denomina `window`. Esse objeto contém todas as propriedades e métodos da linguagem e do respectivo navegador. Os métodos para criar caixas de diálogo que estudamos anteriormente são do objeto `Window` e, assim sendo, as sintaxes a seguir são perfeitamente válidas:

```
window.alert("Olá Mundo");  
window.confirm("Você tem certeza?");  
window.prompt("Informe seu nome", " ");
```

Pode-se fazer referência ao objeto global `window` com o uso de uma palavra-chave própria da linguagem JavaScript denominada `this`. Assim, as sintaxes a seguir são equivalentes às mostradas anteriormente e perfeitamente válidas:

```
this.alert("Olá Mundo");  
this.confirm("Você tem certeza?");  
this.prompt("Informe seu nome", " ");
```

A palavra-chave `this` refere-se ao objeto `Window` quando usada fora do corpo de uma função. Quando utilizada no corpo de uma função, refere-se a um objeto que não `window`. É um objeto próprio da função denominado genericamente de `call object`. Trataremos desse objeto em capítulos posteriores deste livro.

Quando declaramos, em nossos scripts, variáveis globais, estas são automaticamente declaradas como propriedades do objeto global `window`. Assim, de acordo com os conceitos já estudados, as seguintes sintaxes são válidas:

```
a = 50;
alert(window.a); // alerta 50
...
funcaoUm = function() {
    var b = 30;
};
funcaoUm();
alert(window.b); // alerta undefined, pois b é local e não global
```

Os exemplos anteriores podem ser escritos com a palavra-chave `this` como mostrado a seguir:

```
a = 50;
alert(this.a); // alerta 50
...
funcaoUm = function() {
    var b = 30;
};
funcaoUm();
alert(this.b); // alerta undefined, pois b é local e não global
```