

André Milani
Softblue



JavaScript

e-book



Sumário

Apresentação.....	4
Disclosure	5
1. Introdução	6
2. O que é JavaScript	7
2.1. Versões e compatibilidade com navegadores	7
2.2. Como programar em JavaScript.....	8
3. Variáveis	10
3.1. Tipos de dados.....	12
3.2. Propriedades de strings.....	14
4. Operadores	19
4.1. Operadores aritméticos	19
4.2. Operadores lógicos.....	21
5. Comandos de tomada de decisão	25
5.1. Comando if.....	25
5.2. Comando switch.....	29
6. Comandos de repetição	32
6.1. Comando for	32
6.2. Comando for in.....	34
6.3. Comando while	35
6.4. Comando do while.....	36
7. Funções	39
8. Eventos.....	43
9. Caixas de diálogo	47
9.1. Caixa de diálogo alert	47
9.2. Caixa de diálogo confirm	47
9.3. Caixa de diálogo prompt.....	49
10. Abrindo uma URL.....	51
11. Agendando execução de códigos.....	53
12. Alterando uma imagem via JavaScript	55
13. Objeto form	57
13.1. Montando um formulário HTML	57
13.2. Capturando dados de campos texto	60

13.3. Capturando dados de campos de seleção (combo box).....	60
13.4. Capturando dados em campos de marcação (checkbox)	62
13.5. Capturando dados de campos de rádio (radio button)	63
13.6. Validando dados de um formulário e enviando-o ao servidor	64
14. Próximos passos	68
Lista de Figuras.....	69
Lista de Tabelas	70
Lista de Códigos-Fonte	71
Índice Remissivo.....	72

Apresentação

A Softblue é uma empresa de cursos online na área de programação. Fundada em 2003 na cidade de Curitiba-PR, a empresa conta atualmente com um grande portfólio de cursos e dezenas de milhares de alunos espalhados em dezenas de países pelo mundo.

Este e-book foi criado por André Milani, sócio fundador da Softblue. André Milani é formado em Ciência da Computação pela PUC-PR, pós-graduado em Business Intelligence pela mesma instituição e possui diversas certificações na área de TI. É também autor de vários livros na área de informática, entre eles o *Programando para iPhone e iPad*, *MySQL - Guia do Programador* e *Construindo Aplicações Web com PHP & MySQL*, todos pela editora Novatec. Atua desde 2003 com desenvolvimento web e treinamentos de profissionais. Também é desenvolvedor de aplicativos para o ambiente iOS da Apple, possuindo aplicações que juntas somam mais de 50.000 downloads na AppStore.

Disclosure

Este e-book foi elaborado pela Softblue e é de uso exclusivo de seu destinatário. Seu conteúdo não pode ser reproduzido ou distribuído, no todo ou em parte, a qualquer terceiro sem autorização expressa.

A reprodução indevida, não autorizada, deste e-book ou de qualquer parte dele sujeitará o infrator à multa de até 3 (três) mil vezes o valor do e-book, à apreensão das cópias ilegais, à responsabilidade reparatória civil e persecução criminal, nos termos dos artigos 102 e seguintes da Lei 9.610/98.

1. Introdução

Ter conhecimentos em JavaScript é outra característica que deve ser possuída pelos desenvolvedores que pretendem atuar no segmento web de alguma forma. Portanto, se você não conhece ou não tem praticado sua programação em JavaScript recentemente, aproveite este livro para conhecer ou até mesmo relembrar os principais fundamentos desta tecnologia.

Este livro apresenta a linguagem JavaScript de uma forma simples e direta, abordando inicialmente seus principais conceitos e fundamentos e, em seguida, contempla o assunto com os recursos necessários para se programar nesta linguagem.

Para um melhor aproveitamento do conteúdo abordado, é recomendado que você tenha bons conhecimentos de lógica de programação, lembrando que se não for este o seu caso, a Softblue oferece um curso completo e gratuito sobre o tema. Para maiores informações, basta acessar o site da Softblue (www.softblue.com.br) e procurar pelo curso de *Lógica de Programação*.

2. O que é JavaScript

O JavaScript é uma linguagem de programação interpretada por navegadores web, tais como o Google Chrome, Firefox, Opera e outros. Ela não é uma linguagem compilada como as linguagens C ou Java, ou seja, o seu código-fonte não gera um código de máquina para ser executado. O seu código é enviado para os navegadores, e estes, por sua vez, interpretam-no e executam suas ações.

Apesar da similaridade dos nomes JavaScript e Java, essas tecnologias são totalmente diferentes. Entre as diferenças, a linguagem Java utiliza compilação para gerar seus programas (códigos de máquina) e também é utilizada para gerar aplicações clientes (instaláveis), que executam nos computadores sem a necessidade específica de um navegador web. Ela também é amplamente utilizada no lado servidor de diversas aplicações web.

Já o JavaScript foi criado pela Netscape em 1995 com o intuito de transformar as páginas web, até então estáticas no lado cliente, em páginas um pouco mais dinâmicas. Considere que lado servidor é o termo utilizado para representar as operações que são executadas no servidor que hospeda determinado site ou aplicação e que lado cliente é o termo que define as operações que são realizadas no computador do usuário da aplicação, do visitante do site, mais especificamente no seu navegador web.

Por ser uma linguagem interpretada no lado cliente, nos navegadores web dos visitantes de sites e portais na internet, sua utilização permite programar ações para serem realizadas diretamente neste lado da operação, como validação de campos em formulários, interações com o usuário e diversas outras funcionalidades.

A evolução da linguagem continua até hoje. Por ser uma linguagem interpretada pelos navegadores web, é necessário manter o seu navegador sempre atualizado para a versão mais recente, pois somente assim a linguagem poderá ser interpretada de acordo com as novidades mais atuais da tecnologia.

2.1. Versões e compatibilidade com navegadores

Cada navegador web possui suas próprias implementações para atender as novas versões do JavaScript, o que implica em versões diferentes de navegadores poderem apresentar diferentes versões da linguagem. Podem ocorrer também pequenas diferenças na interpretação dos códigos JavaScript em dois navegadores distintos, mesmo que atuais. Isto ocorre pelo fato de alguns dos comandos mais recentes não terem sido implementados ainda por um ou outro navegador, ou terem sido

implementados de maneiras diferentes. Desta forma, recomenda-se sempre testar seus códigos JavaScript nos mais variados navegadores, para garantir o correto funcionamento deles independente do navegador utilizado pelos visitantes das páginas dos seus sistemas web.

Para organizar de uma melhor forma a implementação da linguagem JavaScript pelos navegadores web, foi criada uma norma internacional para padronizá-lo, conhecida como ECMA-262. Ela determina o padrão da linguagem, conhecida como ECMAScript. É com base nesta normalização que as empresas criadoras de navegadores web implementam a linguagem em seus produtos.

Atualmente, o JavaScript encontra-se na versão 1.8.5, atualizada pela última vez em 2010, e já foi implementado pelos principais navegadores web disponíveis no mercado.

2.2. Como programar em JavaScript

Para programar em JavaScript dentro de páginas web, dentro de arquivos HTML, é necessário abrir e fechar a marcação `<script>`, como é mostrado na sintaxe a seguir:

```
<script>
/*
  Bloco de código JavaScript
*/
</script>
```

Antigamente era necessário complementar a marcação `<script>` com o atributo `language="JavaScript"`, mas essa informação não é mais necessária, pois a marcação `<script>` a partir da versão 5 do HTML assume que, se a linguagem não for informada, trata-se da linguagem JavaScript.

Observe no código-fonte apresentado que dentro da marcação `<script>` foi inserida a frase `Bloco de código JavaScript` iniciando com o operador `/*` uma linha antes e finalizando uma linha depois com o operador `*/`. Estes operadores são utilizados para representar o início de um bloco de comentário e o seu término. Portanto, todo o conteúdo existente entre estes dois operadores será ignorado pelo JavaScript, servindo apenas para que o autor do código possa adicionar comentários sobre as funcionalidades que estiver implementado.

Comentários ainda podem ser realizados por meio do operador `//`, que representa o comentário da linha em questão como comentário, conforme mostra o exemplo a seguir:

```
<script>
// Bloco de código JavaScript
</script>
```


É possível ainda criar um arquivo exclusivo de códigos JavaScript, cuja extensão é `.js`, e vinculá-lo às suas páginas HTML da mesma forma que estilos de formatação podem ser organizados em um arquivo a parte com a extensão `.css` e utilizados pelas páginas web. Esta é uma prática altamente recomendada, tendo em vista que diversas páginas poderão fazer uso deste arquivo JavaScript, o qual provavelmente estará codificado com diversas funções. Desta forma fica fácil realizar a manutenção dos códigos JavaScript, pois estarão centralizados em arquivos próprios. Porém, vale a pena reforçar que pequenos trechos de códigos JavaScript, específicos de uma determinada página, podem ser codificados diretamente nas páginas em questão por meio da marcação `<script>`.

Para programar em JavaScript utilizando arquivos próprios da linguagem, crie um arquivo `.js` e, dentro dele, codifique suas funções diretamente, sem a necessidade de abrir a marcação `<script>`. Depois, no arquivo HTML, insira a marcação `<script>` vinculando-a com o arquivo criado, da seguinte forma:

```
<script src="nome_do_seu_arquivo.js"></script>
```

A instrução utilizada no código-fonte apresentado relaciona a página HTML com o arquivo JavaScript, permitindo assim que a sua página faça o uso dos recursos codificados no arquivo `.js`. Outro detalhe importante é que dentro de arquivos do tipo `.js` não há a necessidade de utilizar nenhuma marcação HTML como a `<script>`, ou seja, no arquivo pode-se programar diretamente em JavaScript já a partir da primeira linha. Fique tranquilo, todos esses conceitos serão explorados no decorrer deste e-book.

3. Variáveis

Agora que você já sabe um pouco mais sobre como relacionar suas páginas com os códigos JavaScript, vamos começar a aprender a programar nessa linguagem. O primeiro conceito que será abordado é a criação e uso de variáveis. Declarar e utilizar uma variável nesta linguagem é algo muito simples de ser feito. Basta utilizar a palavra reservada `var` e indicar o nome da variável, como mostra o exemplo a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       // Declarando uma variável chamada var1
11       var var1;
12
13       // Declarando uma variável chamada larguraTabela
14       var larguraTabela;
15
16       // Declarando uma variável chamada alfa, e outra chamada beta, na mesma linha
17       var alfa, beta;
18
19     </script>
20   </body>
21 </html>
```

Código-Fonte 3-1: criandoVariaveis.htm

Observe o Código-Fonte 3-1. Inicialmente, ele define a estrutura básica de uma página HTML por meio de diversas marcações, tais como `<!DOCTYPE html>`, `<html>`, `<head>`, `<title>` e `<body>`. O JavaScript entra em ação neste exemplo a partir da linha 8, com a abertura da marcação `<script>`. Neste código há a criação da variável `var1` na linha 11, da variável `larguraTabela` na linha 14 e das variáveis `alfa` e `beta` na linha 17, seguido do fechamento da marcação `<script>` na linha 19, encerrando o bloco JavaScript. Alguns comentários foram apresentados nas linhas 10, 13 e 16 para demonstrar utilizações deste recurso.

Cada instrução da linguagem JavaScript deve ser finalizada por meio do operador ponto e vírgula (`;`), como é apresentado no final de cada linha de declaração de variável, por exemplo. A falta ou esquecimento deste operador poderá gerar erros na execução do código.

Após declarar variáveis, é possível atribuir valores para elas. Para atrelar valores a uma variável utilize o operador de atribuição `=` (sinal de igual) seguido do valor que

deseja atribuir. Caso o valor em questão seja numérico, o mesmo pode ser informado diretamente, utilizando o operador ponto (.) para separar as casas decimais, caso existam. Para atribuir textos, use aspas no início e fim dos mesmos. Observe no exemplo a seguir o uso do operador de atribuição de valores para definir algumas variáveis:

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       var alfa;
11       alfa = 3;
12
13       var beta = 'E-book de JavaScript';
14
15       document.write('O valor de alfa é: ' + alfa);
16       document.write('<br>');
17       document.write('O valor de beta é: ' + beta);
18
19     </script>
20   </body>
21 </html>

```

Código-Fonte 3-2: atribuindoValores.htm

O resultado da interpretação do Código-Fonte 3-2 pode ser visualizado a seguir:

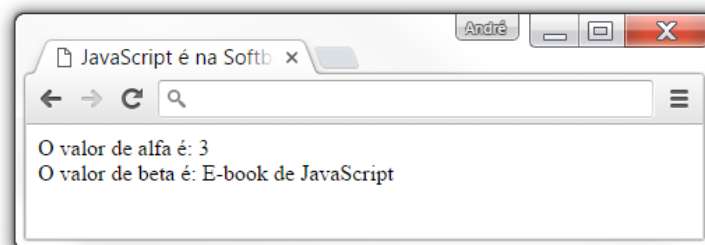


Figura 3-1: Atribuindo e exibindo valores em variáveis

No Código-Fonte 3-2 foi criada a variável `alfa` na linha 10, e o valor `3` numérico foi atribuído para ela na linha 11. Já no caso da variável `beta` apresentada na linha 12, a atribuição de valor foi feita na mesma linha da sua criação, atrelando neste caso o texto `E-book de JavaScript` a ela.

As linhas 15, 16 e 17 do Código-Fonte 3-2 apresentam a instrução `document.write()` do JavaScript, utilizada para imprimir textos no HTML. A palavra reservada `document` do JavaScript representa a instância do documento HTML em si, representada por um objeto do tipo *DOM Document* do JavaScript. Este objeto, por sua vez, possui uma funcionalidade chamada `write`, para imprimir valores no documento HTML. Nos

exemplos apresentados foram impressas no documento algumas frases apresentando os valores das variáveis. Observe ainda nestas operações o uso do operador `+` (sinal de mais), que quando utilizado entre textos e variáveis, concatena (une) informações. Por este motivo todo o conteúdo unido dentro dos parênteses da instrução `document.write()` é impresso na tela, tendo as variáveis JavaScript seus valores interpretados para apresentação.

Quanto ao uso do operador `+` (sinal de mais) em operações matemáticas, fique tranquilo, que já estamos chegando neste ponto, logo nos próximos tópicos.

3.1. Tipos de dados

Como foi possível constatar nos exemplos apresentados nos tópicos anteriores, na declaração de variáveis em JavaScript não é necessário informar o tipo de dado que a mesma irá gerenciar (se é texto, se é número, etc.). Isto ocorre porque o JavaScript não é uma linguagem tipada, ou seja, o próprio JavaScript define o tipo dinamicamente, analisando o valor que está sendo atribuído para cada variável. Por este motivo, uma variável que em um momento armazena um valor numérico pode vir a armazenar, em outro momento, um valor do tipo texto ou qualquer outro disponível pela linguagem.

Basicamente existem quatro tipos de dados no JavaScript. São eles:

- **String:** representa textos e caracteres. Deve sempre ser informado entre aspas de mesmo tipo, podendo ser utilizadas aspas simples ou aspas duplas. Se o texto informado utilizou aspas duplas em sua abertura, deve utilizar aspas duplas também para delimitar o seu fechamento, valendo a mesma regra para o uso de aspas simples.
- **Número:** representa números inteiros e decimais. Deve ser informado sem o uso de aspas, caso contrário será considerado como texto pelo JavaScript e poderá não funcionar em operações matemáticas adequadamente. Para separar a parte inteira da parte decimal de números que contenham essa característica, deve ser utilizado o operador ponto (`.`).
- **Booleano:** representa os valores booleanos `true` e `false`, informados desta forma, sem aspas, utilizados em expressões condicionais e comandos de tomada de decisão, que serão vistos mais adiante.
- **Objeto:** o JavaScript disponibiliza uma série de classes e objetos para serem utilizados, tais como a referência para o documento que está sendo manipulado, janela do navegador, array, entre outros. Variáveis podem armazenar referências para esses objetos, que também serão vistos neste livro.

Acompanhe no exemplo apresentado a seguir algumas utilizações dos tipos de dados apresentados anteriormente:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
```

```

07 <body>
08   <script>
09
10     var alfa = "André ";
11     var beta = 'Milani';
12     var gama = alfa + beta;
13     document.write(gama + "<br>");
14
15     var x = 10;
16     var y = 15.5;
17     var z = x + y;
18     document.write(z + "<br>");
19
20     var resultado = true;
21     document.write(resultado + "<br>");
22
23     var resultado = false;
24     document.write(resultado + "<br>");
25
26     var lista = new Array("Maçã", "Laranja", "Abacaxi");
27     document.write(lista[0] + "<br>");
28     document.write(lista[1] + "<br>");
29     document.write(lista[2] + "<br>");
30
31   </script>
32 </body>
33 </html>

```

Código-Fonte 3-3: tiposDeDados.htm

O resultado da interpretação do Código-Fonte 3-3 pode ser visualizado a seguir:

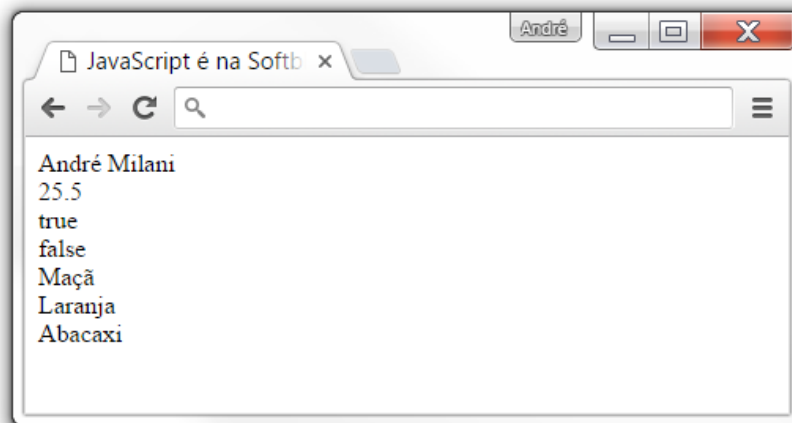


Figura 3-2: Diferentes tipos de dados impressos no HTML

No Código-Fonte 3-3 foram construídos exemplos para vários tipos de dados, iniciando pelas linhas 10 e 11 com a criação de duas variáveis do tipo texto. Observe que na linha 10 foram utilizadas aspas duplas para delimitar o texto, enquanto que na linha 11 foram utilizadas aspas simples. Ambas podem ser utilizadas para definição de textos, sendo igualmente válidas, desde que sejam iguais na abertura e fechamento do texto em questão.

Uma operação de concatenação foi realizada na linha 13, unindo os textos contidos em `alfa` e `beta` e atribuindo este resultado à variável `gama`, impressa na linha 14, concatenando na impressão também a marcação `
`. Neste momento, a variável `gama` contém o texto `André Milani`, que é impresso na tela junto com uma quebra de linha (`
`).

Um pouco mais adiante no Código-Fonte 3-3 temos a criação da variável `x` definida com o valor inteiro `10` na linha 15, e na linha 16 a criação da variável `y` definida com o valor decimal `15.5`. Observe na linha 17 que o operador `+` é utilizado para realizar uma operação entre as variáveis `x` e `y`, atribuindo o resultado para a variável `z`. Neste caso, os valores das variáveis são somados, resultando em `25.5`, valor que é atribuído para a variável `z` e impresso na linha 18.

Isso ocorre porque o operador `+` apresenta diferentes comportamentos dependendo dos tipos de dados envolvidos na operação. Se um dos operandos da expressão for do tipo texto, o operador `+` concatena os valores. Já se todos os operandos forem do tipo numérico, a operação matemática de soma é realizada.

As linhas 20 e 23 apresentam exemplos de variáveis armazenando valores booleanos `true` e `false`, respectivamente. Estes valores ficarão mais interessantes mais adiante, quando operações condicionais forem abordadas.

Por último, a linha 26 apresenta uma variável armazenando um objeto do tipo `Array` do JavaScript. Arrays são conjuntos de elementos ou listas, se você preferir. Neste caso, para acessar cada uma das posições, é necessário acrescentar colchetes depois do nome da variável que faz referência ao array e, dentro deles, colocar a posição desejada, como mostram as linhas 27, 28 e 29, imprimindo as posições 0, 1 e 2 do array em questão. A numeração dos índices dos elementos contidos em arrays sempre inicia em 0.

3.2. Propriedades de strings

Entre os diferentes tipos de dados disponibilizados pelo JavaScript vimos o tipo string, que representa cadeias de caracteres (textos). Este tipo de dado apresenta algumas propriedades extras para sua manipulação, entre elas o seu tamanho (número de caracteres), métodos de busca por trechos de textos e operações para extração de partes de textos. Para conhecer um pouco mais sobre estas funcionalidades, considere o código-fonte apresentado a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
```

```

06 </head>
07 <body>
08   <script>
09
10     var texto = "JavaScript é na Softblue!";
11     document.write("O texto tem " + texto.length + " posições.<br>");
12
13     var posicao = texto.indexOf("Softblue");
14     document.write("Resultado da busca 'Softblue': " + posicao + "<br>");
15
16     posicao = texto.indexOf("Servidor");
17     document.write("Resultado da busca 'Servidor': " + posicao + "<br>");
18
19     var trecho = texto.substring(4, 10);
20     document.write("Trecho entre posições 4 e 10: " + trecho);
21
22   </script>
23 </body>
24 </html>

```

Código-Fonte 3-4: strings.htm

O resultado da interpretação do Código-Fonte 3-4 pode ser visualizado a seguir:

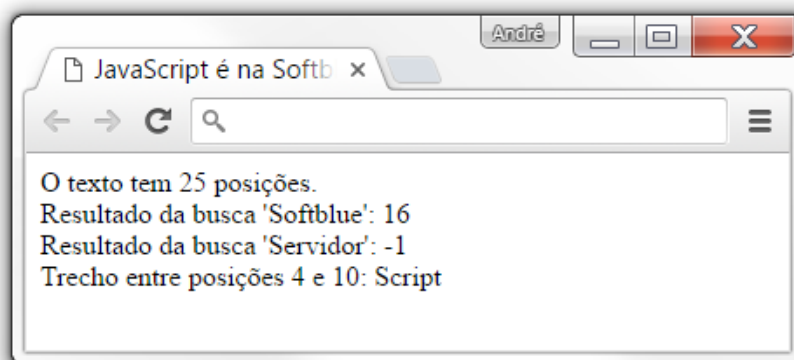


Figura 3-3: Propriedades de strings

No Código-Fonte 3-4 foi declarada inicialmente a variável `texto` na linha 10 com o conteúdo `JavaScript é na Softblue!`. Os textos em JavaScript são organizados, internamente, vinculando um índice numérico para cada posição de caractere, iniciando em 0. Isso quer dizer que a letra `J` que inicia o texto está na posição 0, a primeira letra `a` está na posição 1 e assim sucessivamente, até o ponto de exclamação, que está na posição 24, como mostra o exemplo a seguir com as posições numeradas:

```

JavaScript é na Softblue!
000000000011111111122222
0123456789012345678901234

```

Como é possível observar, até mesmo os espaços em branco contabilizam posições. E a pergunta mais interessante é: quantas posições esse texto possui? Se você respondeu 24, você errou, pois não contabilizou a posição 0 inicial. Este texto possui 25 posições, como é possível contar no exemplo a seguir:

```
JavaScript é na Softblue!
000000001111111111222222
1234567890123456789012345
```

Não confunda índice de posição com a contagem de posições. Os índices de posição de caracteres iniciam sua contagem em `0`, enquanto que a contabilização do número de caracteres existentes inicia a contagem em `1`. Não se preocupe, a diferença entre essas duas informações ficará mais clara no decorrer deste tópico.

Primeiro vamos conhecer a propriedade que informa o tamanho do texto, que é a propriedade `length`. Para acessá-la, utilize o nome da variável que contém a string, seguido de `.length`, como foi realizado na linha 11 do Código-Fonte 3-4. Observe que o valor impresso na tela por esta operação é `25`. A string em questão tem 25 caracteres, organizados numericamente entre os índices `0` e `24`.

O uso dos índices numéricos fica ainda mais evidente ao utilizar o método `substring`, conforme apresentado na linha 19 do Código-Fonte 3-4. Este método captura uma cópia do conteúdo existente entre o índice numérico informado no primeiro parâmetro (no caso, `4`) e o índice informado no segundo parâmetro (no caso, `10`), apresentando a palavra `Script` como retorno da operação.

Observe que este método inclui o caractere existente na posição `4` (letra `S`), mas não inclui o caractere da posição `10` (espaço em branco). Para que a letra `t` da palavra `Script` aparecesse no resultado, mesmo ela estando posicionada na posição `9` da string, seria necessário informar um caractere a mais para o método `substring`.

Para complementar um pouco mais seus conhecimentos sobre strings, observe os comandos apresentados entre as linhas 13 e 17 do Código-Fonte 3-4. Por meio do uso do método `indexOf` a partir de uma variável que contém uma string (`texto.indexOf`, por exemplo) é possível realizar uma busca no texto pelo parâmetro informado. A linha 13 ilustra a busca pela palavra `Softblue` no texto contido na variável `texto`, a qual existe e, por este motivo, retorna o valor da posição inicial deste texto no texto principal, que é a posição `16`.

No caso da busca por um trecho de texto não encontrar resultados, o valor `-1` é retornado, como pode ser observado nas linhas 16 e 17 ao fazer a busca pelo termo `Servidor`, que não existe no texto contido pela variável `texto`.

Voltando na busca pela palavra `Softblue`. E se houvesse mais de uma ocorrência dessa palavra no texto? Neste caso o JavaScript retornaria apenas a posição da primeira ocorrência, sendo necessário utilizar outras buscas para acessar as demais ocorrências. Mas não basta apenas repetir a busca como ela foi feita inicialmente, pois desta forma é sempre a primeira ocorrência que será retornada.

O método `indexOf` aceita um segundo parâmetro, opcional, que indica a posição a partir de onde a busca deve iniciar. Conhecendo a posição da primeira ocorrência fica fácil solicitar outras buscas avançando essa posição, como mostra o código-fonte apresentado a seguir:

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       var texto = "JavaScript é na Softblue!";
11       var posicao = texto.indexOf("a");
12
13       while(posicao != -1)
14       {
15         document.write("Posição: " + posicao + "<br>");
16         posicao = texto.indexOf("a", posicao+1);
17       }
18
19     </script>
20   </body>
21 </html>

```

Código-Fonte 3-5: buscaRecorrente.htm

O resultado da interpretação do Código-Fonte 3-5 pode ser visualizado a seguir:

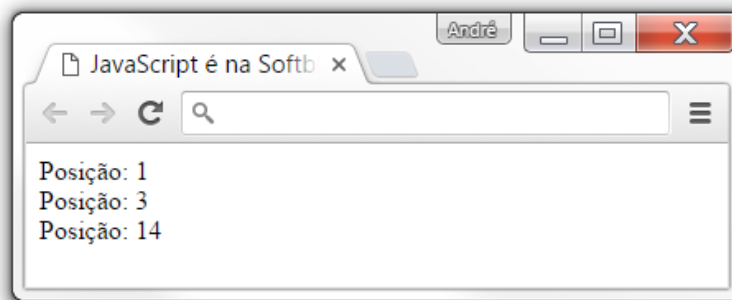


Figura 3-4: Todos os resultados para a busca do caractere 'a'

No Código-Fonte 3-5 foi realizada uma primeira busca pelo caractere `a`, na linha 12, armazenando o índice na variável `posicao`. Se nenhuma ocorrência for encontrada, a condição apresentada na linha 14 para execução do laço de repetição `while` não é atendida, pois `-1` é retornado, não imprimindo o valor da variável `posicao` e não realizando uma nova busca pela próxima ocorrência.

Já se a busca retornar um valor diferente de `-1`, significa que uma ocorrência foi encontrada. Neste caso, o laço de repetição `while` imprime na tela a posição em questão por meio da linha 16 do Código-Fonte 3-5 e, logo em seguida na linha 17, realiza a

busca pela próxima ocorrência. Observe que esta operação de nova busca faz o uso do método `indexOf` com dois parâmetros, informando no segundo parâmetro a posição de onde a nova busca deve iniciar, que no caso é a posição atual da ocorrência encontrada (valor contido na variável `posicao`) acrescida de uma posição. Ou seja, a busca é iniciada a partir do próximo caractere.

Desta forma temos as posições 1, 3 e 14 impressas na tela, representando os índices numéricos das três ocorrências do caractere `a` na string `JavaScript é na Softblue!`.

4. Operadores

Este tópico apresenta os principais operadores disponíveis no JavaScript, que tornam possíveis a criação e avaliação de expressões de diversos tipos. Basicamente, existem dois tipos: operadores aritméticos e operadores lógicos. O primeiro tipo está relacionado a operações matemáticas entre números geralmente apresentados em constantes ou variáveis declaradas. Já o segundo tipo de operadores está relacionado aos comandos lógicos e expressões condicionais, utilizados em boa parte dos casos juntos com comandos de tomada de decisão.

Caso você já tenha conhecimento em alguma outra linguagem de programação, é muito provável que você esteja familiarizado com os operadores que serão apresentados. Mas se o JavaScript é a primeira linguagem de programação que você está aprendendo, convido você a conhecer pela primeira vez os operadores matemáticos e lógicos do mundo da programação!

4.1. Operadores aritméticos

Os operadores aritméticos são responsáveis por realizar operações matemáticas entre dois ou mais operandos declarados no JavaScript. As principais operações aritméticas são:

Sinal	Exemplo	Descrição
+	<code>a + b</code>	Soma o valor de <code>a</code> com o valor de <code>b</code> .
-	<code>a - b</code>	Subtrai do valor de <code>a</code> o valor de <code>b</code> .
*	<code>a * b</code>	Multiplica o valor de <code>a</code> com o valor de <code>b</code> .
%	<code>a % b</code>	Retorna o resto da divisão do valor de <code>a</code> pelo valor de <code>b</code> .
/	<code>a / b</code>	Divide o valor de <code>a</code> com o valor de <code>b</code> .
++	<code>a++</code>	Soma 1 ao valor de <code>a</code> .
--	<code>a--</code>	Subtrai 1 do valor de <code>a</code> .

Tabela 4-1: Operadores aritméticos

A seguir, alguns exemplos de utilização dos operadores aritméticos:

```
01 <!DOCTYPE html>
```

```

02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       var a = 1;
11       var resultado = 0;
12       resultado = a + a + a;
13       document.write(resultado + "<br>"); // Exibe o valor 3
14
15       var b = 5, c = 8;
16       resultado = a + 1 - b * (c % b);
17       document.write(resultado + "<br>"); // Exibe o valor -13
18
19     </script>
20   </body>
21 </html>

```

Código-Fonte 4-1: operacoesAritmeticas.htm

O resultado da interpretação do Código-Fonte 4-1 pode ser visualizado a seguir:

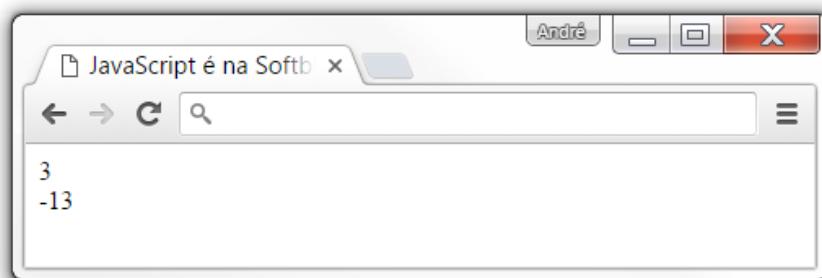


Figura 4-1: Resultado das operações aritméticas

No Código-Fonte 4-1 foram inicialmente declaradas e inicializadas as variáveis `a` e `resultado` nas linhas 10 e 11. Uma operação matemática é realizada na linha 12 atribuindo para `resultado` a soma do operando `a` com ele mesmo mais duas vezes. Neste caso, assumindo que o valor do operando `a` é 1, a expressão matemática fica `1 + 1 + 1`, que resulta em 3. Por este motivo, o valor 3 armazenado em `resultado` na linha 12 e é impresso na linha 13.

Já na linha 15 são declaradas mais duas variáveis: `b` com o valor 5 e `c` com o valor 8. Dessa forma, uma nova expressão matemática é construída na linha 16 da seguinte forma: `a + 1 - b * (c % b)`. Substituindo as variáveis por seus respectivos valores numéricos, a expressão pode ser interpretada desse modo: `1 + 1 - 5 * (8 % 5)`. Neste caso, a ordem em que as operações são realizadas obedecem as regras da matemática. Inicialmente temos a resolução do que está entre parênteses: `(8 % 5)`. Esta operação calcula o resto da divisão de 8 por 5, que é 3, avançando nossa expressão para `1 + 1 - 5 * 3`. Em seguida, a operação de multiplicação tem preferência, multiplicando 5 por 3,

que resulta em 15, avançando a expressão para $1 + 1 - 15$. Na sequência, temos a soma de $1 + 1$ subtraído 15, resultando em -13, valor que é impresso na linha 17.

4.2. Operadores lógicos

Os operadores lógicos permitem a avaliação de expressões condicionais e, consequentemente, a tomada de decisão dentro das linguagens de programação. A partir deles é possível verificar uma condição e, a partir disso, decidir se uma determinada ação deve ou não ser executada.

Neste tópico serão visto os operadores lógicos e no tópico seguinte os comandos de decisão que fazem uso destes operadores. É a partir destes tópicos que as linguagens de programação começam a ficar mais interessantes!

Expressões condicionais e operadores lógicos estão relacionados aos valores booleanos `true` e `false`. Isso quer dizer que somente esses valores poderão ser resultados dessas expressões. Apesar de isso parecer pouco produtivo, você vai compreender no decorrer deste tópico o quão valiosos são estes recursos.

Os principais operadores lógicos estão listados na tabela a seguir:

Sinal	Exemplo	Descrição
!	!a	Retorna a negação do valor de <code>a</code> , ou seja, o valor contrário. Se <code>a</code> contiver <code>true</code> , retorna <code>false</code> . Se contiver <code>false</code> , retorna <code>true</code> .
&&	a && b	Retorna <code>true</code> se ambos os valores de <code>a</code> e de <code>b</code> forem <code>true</code> , deixando de avaliar o lado direito da expressão caso o lado esquerdo já apresente o valor <code>false</code> , pois <code>false</code> e qualquer outro valor sempre resultará em <code>false</code> .
&	a & b	Retorna <code>true</code> se ambos os valores de <code>a</code> e <code>b</code> forem <code>true</code> , avaliando sempre ambos os lados da expressão.
	a b	Retorna <code>true</code> se pelo menos um dos valores de <code>a</code> ou de <code>b</code> for <code>true</code> , deixando de avaliar o lado direito da expressão caso o lado esquerdo já apresente o valor <code>true</code> , pois <code>true</code> ou qualquer outro valor sempre resultará em <code>true</code> .
	a b	Retorna <code>true</code> se pelo menos um dos valores de <code>a</code> ou <code>b</code> for <code>true</code> , avaliando sempre ambos os lados da expressão.
==	a == b	Retorna <code>true</code> se o valor de <code>a</code> e <code>b</code> forem iguais.
!=	a != b	Retorna <code>true</code> se o valor de <code>a</code> e <code>b</code> forem diferentes.
===	a === b	Retorna <code>true</code> se o valor de <code>a</code> e <code>b</code> forem iguais e do mesmo tipo de dados.

<code>!==</code>	<code>a !== b</code>	Retorna <code>true</code> se o valor de <code>a</code> e <code>b</code> forem diferentes ou caso sejam de tipos de dados diferentes.
<code><</code>	<code>a < b</code>	Retorna <code>true</code> se o valor de <code>a</code> for menor que o valor de <code>b</code> .
<code><=</code>	<code>a <= b</code>	Retorna <code>true</code> se o valor de <code>a</code> for menor ou igual ao valor de <code>b</code> .
<code>></code>	<code>a > b</code>	Retorna <code>true</code> se o valor de <code>a</code> for maior que o valor de <code>b</code> .
<code>>=</code>	<code>a >= b</code>	Retorna <code>true</code> se o valor de <code>a</code> for maior ou igual ao valor de <code>b</code> .

Tabela 4-2: Operadores lógicos

Para ficar mais claro o uso dos operadores lógicos, serão apresentados a seguir alguns exemplos:

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       var a;
11       var resultado;
12
13       a = true;
14       resultado = !a;
15       document.write(resultado + "<br>"); // Imprime false
16
17       resultado = true && true;
18       document.write(resultado + "<br>"); // Imprime true
19
20       resultado = true && false;
21       document.write(resultado + "<br>"); // Imprime false
22
23       resultado = true || false;
24       document.write(resultado + "<br>"); // Imprime true
25
26       resultado = 5 == 5;
27       document.write(resultado + "<br>"); // Imprime true
28
29       resultado = 5 == 6;
30       document.write(resultado + "<br>"); // Imprime false
31
32       resultado = 5 < 6;
33       document.write(resultado + "<br>"); // Imprime true
34
35       resultado = 3 == "3";
36       document.write(resultado + "<br>"); // Imprime true
37
38       resultado = 3 == 3;
39       document.write(resultado + "<br>"); // Imprime true
40
41       resultado = 3 === "3";

```

```

42     document.write(resultado + "<br>"); // Imprime false
43
44     resultado = 3 === 3;
45     document.write(resultado + "<br>"); // Imprime true
46
47     </script>
48 </body>
49 </html>

```

Código-Fonte 4-2: operacoesLogicas.htm

O resultado da interpretação do Código-Fonte 4-2 pode ser visualizado a seguir:



Figura 4-2: Resultado das operações lógicas

No Código-Fonte 4-2 temos a primeira operação lógica na linha 14, negando o valor contido na variável `a`, que foi definida na linha anterior com o valor `true`. Nesta operação, seu valor negado (`false`) foi atribuído para a variável `resultado`, que é impressa na tela por meio da instrução existente na linha 15.

A próxima operação lógica existente é a da linha 17, que faz o uso do operador `&&`. Este operador só resulta em `true` caso ambos os valores sejam verdadeiros, que é o que ocorre nesta operação, impresso pela linha 18.

O mesmo operador `&&` é utilizado na expressão da linha 20, mas desta vez com um dos operandos sendo `false`, apenas para ilustrar que neste caso a operação resulta em `false`, valor impresso pela linha 21.

A linha 23 ilustra um exemplo de uso do operador `||`, que retorna `true` caso um dos operandos seja `true`. Neste caso, `true` é impresso na tela por meio do comando existente na linha 24.

Em seguida temos o operador `==` avaliando se `5` é igual a `5` na linha 26, e se `5` é igual a `6` na linha 29. Ambos os resultados podem ser conferidos nas linhas 27 e 30,

respectivamente. Observe nestes exemplos a diferença entre `=` e `==`. O operador `=` é o de atribuição, vinculando o valor resultante da expressão lógica à variável `resultado`, enquanto que o operador `==` é de comparação e verifica se os operandos são iguais ou não. O resultado é um valor lógico `true` ou `false`, que indica o resultado desta comparação.

Não querendo dificultar ainda mais o processo, mas fica mais interessante ainda conhecer a diferença de `=` e `==` para o operador `===`. O operador `===` compara se os operandos possuem o mesmo valor e se são do mesmo tipo de dado.

Por exemplo, na linha 35 temos a comparação `3 == "3"`, que resulta em `true`, pois o operador `==` compara se ambos os valores são iguais, sem levar em consideração seus tipos de dados. Observe neste exemplo que o valor `3` é do tipo numérico, enquanto `"3"`, por ser apresentado entre aspas, é do tipo texto. Essa mesma comparação feita com o operador `===` na linha 41 retorna `false`, pois apesar dos valores serem os mesmos, não são do mesmo tipo de dado. Portanto muita atenção com estes pequenos detalhes das linguagens de programação, pois eles tiram o sono de muitos programadores!

5. Comandos de tomada de decisão

Um dos principais recursos responsáveis por dar vida aos sistemas no mundo da programação são os comandos de tomada de decisão, por permitirem mudar os rumos da execução de um código baseado na análise de expressões condicionais.

Por exemplo, ao avaliar se a idade de um usuário é menor que 18 anos, encaminhar a execução do programa para indicar uma mensagem que ele ainda não pode se matricular na autoescola; ou, se for maior de 18 anos, realizar outra ação, como exibir o formulário de inscrição de uma autoescola.

Veremos neste tópico os principais comandos de tomada de decisão da linguagem JavaScript, começando pelo mais clássico e tradicional de todos: o comando `if`.

5.1. Comando `if`

O comando `if` analisa uma sentença condicional e, caso a mesma resulte em `true`, seu bloco de código é executado. Caso a sentença avaliada retorne `false`, o bloco de código associado ao comando `if` é ignorado, não sendo executado.

A sintaxe mais simples do comando `if` está descrita a seguir:

```
if(condição)
{
    /* Bloco de código caso a sentença retorne true */
}
```

A seguir, alguns exemplos de utilização do comando `if`:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       var alfa = 4, beta = 5;
11
12       if(alfa > beta)
13       {
14         document.write('alfa é maior do que beta');
15       }
16
17       if(alfa == 4)
18       {
19         document.write('alfa é quatro');
20       }
21
```

```

22     </script>
23 </body>
24 </html>

```

Código-Fonte 5-1: comandoIf.htm

O resultado da interpretação do Código-Fonte 5-1 pode ser visualizado a seguir:

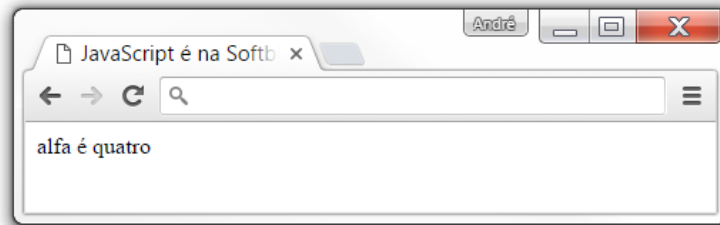


Figura 5-1: Exemplo de uso do comando if

No exemplo apresentado no Código-Fonte 5-1 temos um primeiro comando `if` entre as linhas 12 e 15, que não apresenta nenhum resultado na tela. Isso ocorre pelo fato de que o comando `if` apresentado na linha 12 avalia se o valor contido na variável `alfa` é maior do que o valor contido na variável `beta`. Como `alfa` contém 4 e `beta` contém 5, é avaliado se 4 é maior que 5, o que resulta em `false` e o que faz com que o comando `if` ignore o bloco de código definido entre as linhas 13 e 15.

Sinta-se a vontade para mudar o valor de `alfa` para 6 ou qualquer outro valor maior do que o definido para a variável `beta`, salvar o arquivo e executá-lo novamente, para ver desta vez a sentença `alfa é maior do que beta` ser impressa na tela.

Já o comando `if` apresentado entre as linhas 17 e 20 imprime na tela a sentença `alfa é quatro`, pois ele avalia se o valor contido em `alfa` é igual a 4. Lembre-se que comparação de valores deve ser feita com o operador `==` (dois iguais).

O comando `if` pode ser ainda complementado por um bloco `else`, que por sua vez será executado somente se a condição do bloco `if` não for avaliada como verdadeira. Por exemplo, caso você esteja avaliando a idade de um usuário para permitir ou não a sua inscrição na autoescola, você pode avaliar na condição do comando `if` a sentença `idade >= 18` para que o bloco de código seja executado caso a idade seja igual ou maior que 18, e pode opcionalmente criar o bloco `else` indicando alguma ação para ser realizada caso a condição avaliada no comando `if` não seja verdadeira. A seguir, um exemplo de utilização do comando `else`:

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>

```

```

08  <script>
09
10      var alfa = 4, beta = 5;
11
12      if(alfa > beta)
13      {
14          document.write('alfa é maior do que beta');
15      }
16      else
17      {
18          document.write('alfa não é maior do que beta');
19      }
20
21  </script>
22  </body>
23  </html>

```

Código-Fonte 5-2: comandIfElse.htm

O resultado da interpretação do Código-Fonte 5-2 pode ser visualizado a seguir:

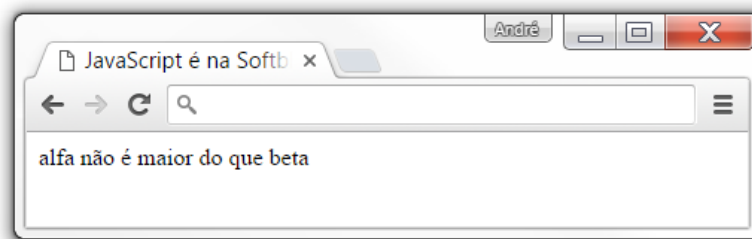


Figura 5-2: Exemplo de uso do comando if/else

No exemplo apresentado no Código-Fonte 5-2, o comando `if` avalia se `alfa` é maior do que `beta` na linha 12. Como a expressão resulta em `false`, pois `alfa` contém 4 e `beta` contém 5, o bloco de código `if` existente entre as linhas 12 e 15 é ignorado, e por esse motivo o bloco `else` existente entre as linhas 16 e 19 é executado, imprimindo `alfa não é maior do que beta` na tela.

Cada comando `if` pode ter no máximo um bloco `else`. Contudo, existe ainda o bloco `else if`, que pode ser utilizado mais de uma vez. Blocos do tipo `else if` realizam uma nova avaliação condicional caso a avaliação apresentada no comando anterior tenha resultado em `false`.

Por exemplo, suponha que você esteja programando uma funcionalidade que deva dizer `Bom dia` se o horário em que a página for executada for entre 0 e 12 horas, `Boa tarde` se for entre 13 e 18 horas ou `Boa noite` se for entre 19 e 23 horas. Neste caso, o seguinte comando `if` poderia ser montado:

```

01  <!DOCTYPE html>
02  <html lang="pt-br">
03  <head>
04      <meta charset="utf-8">
05      <title>JavaScript é na Softblue!</title>
06  </head>

```

```

07 <body>
08   <script>
09
10     var horario = 12;
11
12     if(0 <= horario && horario <= 12)
13     {
14       document.write('Bom dia');
15     }
16     else if(13 <= horario && horario <= 18)
17     {
18       document.write('Boa tarde');
19     }
20     else if(19 <= horario && horario <= 23)
21     {
22       document.write('Boa noite');
23     }
24     else
25     {
26       document.write('Horário inválido');
27     }
28
29   </script>
30 </body>
31 </html>

```

Código-Fonte 5-3. comandIfElseIfElse.htm

O resultado da interpretação do Código-Fonte 5-3 pode ser visualizado a seguir:

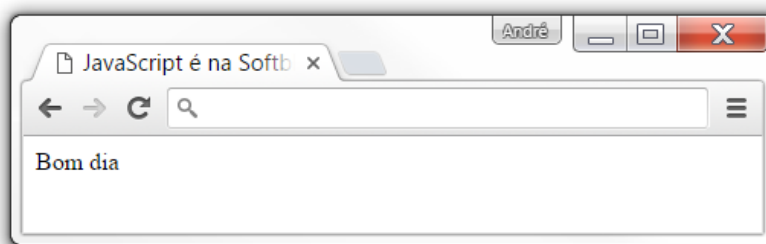


Figura 5-3: Exemplo de uso do comando if/else if/else

No exemplo apresentado no Código-Fonte 5-3 temos um exemplo de comando `if` completo. O exemplo começa criando a variável `horario` na linha 10 e atribuindo o valor `12` para ela. Em seguida, o comando `if` apresentado na linha 12 verifica se o valor contido em `horario` é maior ou igual a `0` e também menor ou igual a `12`, imprimindo `Bom dia` na tela caso essas condições sejam verdadeiras. Caso não sejam, o bloco `else if` apresentado na linha 16 será avaliado, verificando se o valor contido em `horario` é maior ou igual a `13` e menor ou igual que `18`, imprimindo `Boa tarde` se essas condições forem verdadeiras.

Um novo bloco `else if` é apresentado na linha 20 e somente será avaliado se até o momento nenhuma das condições anteriores tiver sido satisfeita, ou seja, se nenhum dos blocos de código tiver sido executado ainda. Neste caso, o comando `else if` da linha 20 vai entrar em ação verificando se `horario` é maior ou igual a `19` e menor ou

igual a 23, imprimindo nesta situação `Boa noite`. Se novamente a condição não for aceita (suponha que a variável `horario` contenha o número 50, por exemplo), o bloco `else` final apresentado na linha 24 será executado, imprimindo `Horário inválido` na tela.

5.2. Comando switch

Assim como o comando `if`, existe o comando `switch`, que também é um comando de tomada de decisão. A diferença é que o comando `if` apenas avalia se a sua condição é verdadeira ou falsa, enquanto o comando `switch` avalia geralmente uma variável, podendo executar especificamente um bloco de código para cada valor que essa variável possa conter.

O uso do comando `switch` está mais relacionado com a análise de expressões numéricas e de textos, e não apenas de valores ou expressões condicionais como no caso do comando `if`, apesar de atender a esta demanda também.

A sintaxe do comando `switch` está descrita a seguir:

```
switch(variável)
{
    case valor1:
        /* Bloco de código caso variável == valor1 */
        break;

    case valor2:
        /* Bloco de código caso variável == valor2 */
        break;

    default:
        /* Bloco de código se nenhuma condição for atendida */
        break;
}
```

Para cada valor que a variável avaliada pelo comando `switch` possa apresentar, um bloco de código `case` relacionado deve ser construído. Caso nenhum resultado seja encontrado nos blocos `case` informados, o bloco `default` é executado (a construção do bloco `default` é opcional).

A seguir, um exemplo de utilização do comando `switch`:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10     var alfa = 4;
11
12     switch(alfa)
```

```

13  {
14      case 2:
15          document.write('alfa é igual a 2');
16          break;
17
18      case 3:
19          document.write('alfa é igual a 3');
20          break;
21
22      case 4:
23          document.write('alfa é igual a 4');
24          break;
25
26      default:
27          document.write('alfa não é 2, nem 3 e nem 4');
28          break;
29  }
30
31  </script>
32  </body>
33  </html>

```

Código-Fonte 5-4: comandoSwitch.htm

O resultado da interpretação do Código-Fonte 5-4 pode ser visualizado a seguir:

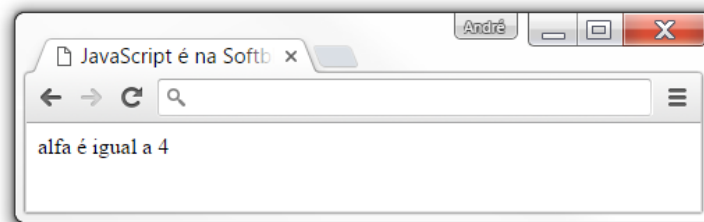


Figura 5-4: Exemplo de uso do comando switch

No exemplo apresentado no Código-Fonte 5-4, um comando `switch` começa a ser construído na linha 12, avaliando o valor contido na variável `alfa`. Dentro do bloco de código do comando `switch`, vários blocos `case` foram construídos. O primeiro deles entre as linhas 13 e 16, que será executado somente se o valor contido em `alfa` for 2, imprimindo `alfa é igual a 2` na tela.

Cada bloco `case` deve ser encerrado por meio da instrução `break`, como ocorre na linha 16. Caso contrário, a execução do programa continuará interpretando todas as linhas de código apresentadas até o final do comando `switch` ou até encontrar uma instrução `break`.

Em seguida, no comando `switch` há um segundo bloco `case`, entre as linhas 18 e 20, que será executado somente se o valor contido na variável `alfa` for 3. Há um terceiro bloco `case` entre as linhas 22 e 24, que somente será executado caso o valor de `alfa` seja 4 e, por último, há um bloco `default`, que é opcional, e é executado somente se o valor de

`alfa` não for encontrado em nenhum dos blocos `case` apresentados anteriormente no comando.

No exemplo apresentado no Código-Fonte 5-4, o bloco `case 4` será executado, pois a variável `alfa` foi definida com o valor `4` na linha 10. Altere os valores e as condições dos blocos `case` para criar outros exemplos e comparar os resultados.

6. Comandos de repetição

Agora que você conheceu um pouco mais sobre os comandos de tomada de decisão do JavaScript, chegou a hora de um outro conjunto de comandos bastante úteis serem apresentados: os comandos de repetição, que são utilizados para realizar ações uma ou mais vezes ou enquanto uma condição for verdadeira.

A seguir estão apresentados os comandos de repetição existentes no JavaScript, começando pelo mais popular de todos: o `for`.

6.1. Comando `for`

O comando `for` executa `n` vezes seu bloco de código, baseado em um valor de variável informada inicialmente, uma condição de até quando o mesmo deve ser executado e valor de incremento da variável para ser atualizada a cada execução do comando. Vamos conhecer a sintaxe do comando `for`:

```
for(inicialização; condição; incremento)
{
    /* Bloco de código */
}
```

No comando `for` deve ser informado como primeiro parâmetro (*inicialização*) a variável de controle de repetição. Essa instrução será executada antes das repetições do comando `for` iniciarem.

As repetições do comando `for` irão ocorrer enquanto a condição informada no segundo parâmetro do comando (*condição*) resultar em `true`, sendo que no final de cada execução do bloco de código a instrução definida no terceiro parâmetro do comando (*incremento*) é executada, atualizando eventuais variáveis de controle de repetição. Para que esses conceitos fiquem mais claros, vamos visualizar um exemplo do uso deste comando no código-fonte apresentado a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       for(a=1; a<=10; a++)
11       {
12         document.write('Valor de a: ' + a + '<br>');
13       }
14
```



```
15     </script>
16   </body>
17 </html>
```

Código-Fonte 6-1: comandoFor.htm

O resultado da interpretação do Código-Fonte 6-1 pode ser visualizado a seguir:

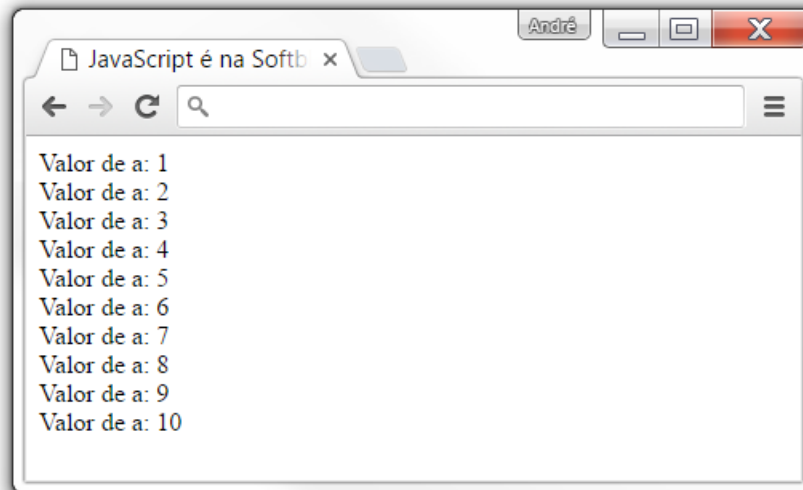


Figura 6-1: Exemplo de uso do comando for

No exemplo apresentado no Código-Fonte 6-1 um comando `for` é construído entre as linhas 10 e 13. Em seu primeiro parâmetro é definido que uma variável de controle chamada `a` será utilizada, iniciando com o valor `1`. Em seguida, no segundo parâmetro, é informado que o comando deve repetir enquanto o valor de `a` for menor ou igual a `10`. No terceiro parâmetro é informado que o valor de `a` deve ser atualizado somando 1 (operação `++`) sempre que uma repetição do comando ocorrer. Neste caso o comando começa a executar o seu bloco de código com `a` valendo `1` inicialmente, imprimindo `Valor de a: 1` na tela.

Ao encerrar a execução do bloco de código, a variável `a` é incrementada, passando a valer `2` porque a operação `a++` definida no terceiro parâmetro do comando `for` é executada. Neste ponto, o comando `for` volta a avaliar a condição informada no segundo parâmetro, para saber se `a` continua sendo menor ou igual a `10`. Como o resultado desta análise é verdadeiro, o bloco de código é executado novamente, dessa vez com `a` valendo `2`, imprimindo `Valor de a: 2` na tela, e assim sucessivamente, até que `a` seja atualizado para `11`, fazendo com que a condição `a <= 10` retorne `false`, encerrando a execução do comando `for`.

6.2. Comando for in

O comando `for in` executa `n` vezes seu bloco de código, baseado em um array informado e seu número de posições, executando uma vez para cada elemento presente no array em questão, da seguinte forma:

```
for(elemento in array)
{
    /* Bloco de código */
}
```

Para cada elemento existente no array informado, o bloco será executado, disponibilizando o elemento em questão na variável informada à esquerda do operador `in`. Por exemplo, considere o código-fonte apresentado a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10     var minhaLista = new Array(2, 4, 5, 7);
11
12     for(item in minhaLista)
13     {
14       document.write('Processando ' + item + '<br>');
15     }
16
17   </script>
18 </body>
19 </html>
```

Código-Fonte 6-2: comandoForIn.htm

O resultado da interpretação do Código-Fonte 6-2 pode ser visualizado a seguir:

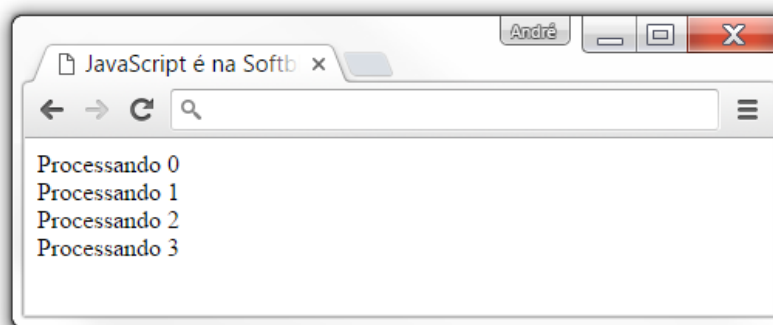


Figura 6-2: Exemplo de uso do comando for in

No exemplo apresentado no Código-Fonte 6-2, é possível observar a criação de um array na linha 10 chamado `minhaLista` com alguns elementos numéricos. O comando `for` apresentado na linha 12 define por meio do parâmetro `item in minhaLista` que cada

elemento do array contido em `minhaLista` será alocado na variável `item` e processado pelo bloco de código disponibilizado entre as linhas 13 e 15. Dessa forma, o bloco de códigos executará 4 vezes, sendo uma para cada elemento presente no array `minhaLista`.

6.3. Comando `while`

O comando `while` é um comando de repetição similar ao comando `for`. No entanto, ele não define valores iniciais para suas variáveis de controle e nem mesmo as alterações que as mesmas deverão sofrer no final de cada execução de seu bloco de código. Por este motivo, o comando `while` deve ser utilizado com mais atenção, pois, por deixar seus parâmetros de controle mais livres, pode facilmente entrar em loop infinito, travando a execução do código.

Quando ativado, o comando `while` executa constantemente seu bloco de código enquanto sua condição for satisfeita. Geralmente suas variáveis de controle são tratadas dentro de seu próprio bloco, devendo ser gerenciadas pelo programador isoladamente. A sintaxe do comando `while` é a seguinte:

```
while(condição)
{
    /* Bloco de código */
}
```

Para ilustrar a utilização do comando `while`, considere o exemplo a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10     var a = 1;
11
12     while(a <= 10)
13     {
14       document.write('Valor de a: ' + a + '<br>');
15       a++;
16     }
17
18   </script>
19 </body>
20 </html>
```

Código-Fonte 6-3: comandoWhile.htm

O resultado da interpretação do Código-Fonte 6-3 pode ser visualizado a seguir:

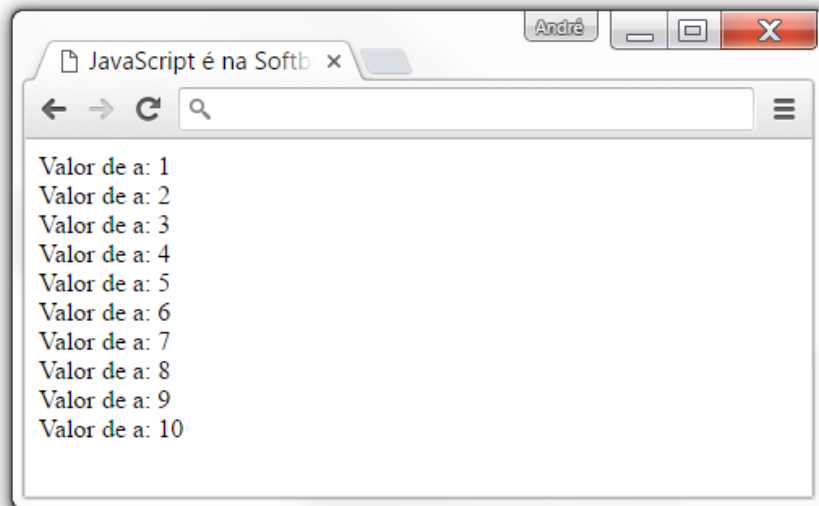


Figura 6-3: Exemplo de uso do comando while

No exemplo apresentado no Código-Fonte 6-3 temos um comando `while` construído entre as linhas 12 e 16, que repete sua execução enquanto o valor contido na variável `a` for menor ou igual a `10`. Observe neste caso que foi necessário criar e inicializar com algum valor a variável `a` na linha 10, antes do comando `while` iniciar, pois diferentemente do comando `for`, o `while` não inicializa as variáveis de controle de repetição.

Ainda, para que o valor de `a` sofra alterações, tornando possível sair do bloco de código do comando `while` em algum momento, foi necessário incrementar o seu valor na linha 15 de alguma forma. O comando `while` deste exemplo foi construído para realizar a mesma contagem de 1 até 10 já realizada na apresentação do comando `for`.

6.4. Comando do while

Encerrando o grupo de comandos de repetição do JavaScript, será apresentado o `do while`, uma variação do comando `while` tradicional. Basicamente o comando é o mesmo. Contudo, a validação da condição de repetição do comando é realizada no final do bloco de código e não no começo, garantindo que o comando execute pelo menos uma vez antes da primeira condição ser avaliada, como ilustra a sintaxe apresentada a seguir:

```
do
{
    /* Bloco de código */
} while(condição);
```

Para ilustrar o uso do comando `do while`, considere o código-fonte apresentado a seguir:

```
01 <!DOCTYPE html>
```

```

02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10     var a = 1;
11
12     do
13     {
14       document.write('Valor de a: ' + a + '<br>');
15       a++;
16     } while(a <= 10)
17
18     </script>
19   </body>
20 </html>

```

Código-Fonte 6-4: comandoDoWhile.htm

O resultado da interpretação do Código-Fonte 6-4 pode ser visualizado a seguir:

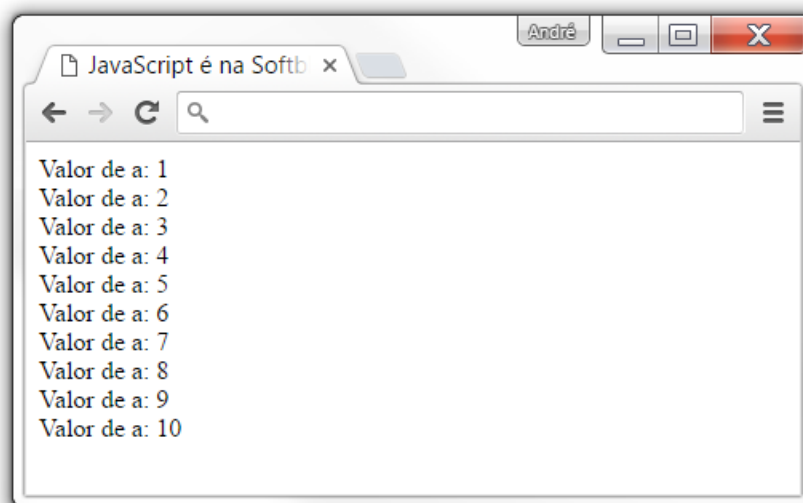


Figura 6-4: Exemplo de uso do comando do while

No exemplo apresentado no Código-Fonte 6-4, observe que o resultado apresentado é o mesmo que nos exemplos do comando `for` e do comando `while` tradicional. A diferença neste caso é que a condição fica no final do bloco de código.

Se você alterar o Código-Fonte 6-3 e o Código-Fonte 6-4 para que em ambos a variável `a` seja iniciada com o valor `15`, por exemplo, a diferença poderá ser notada. No caso do Código-Fonte 6-3 nenhuma impressão será realizada na tela, pois o valor de `a` sendo `15` fará com que a condição inicial do comando `while` resulte em `false`, não executando o bloco de código nenhuma vez. Já no Código-Fonte 6-4, apresentado neste tópico, o comando `while` executará uma única vez, mesmo com `a` contendo `15`, pois

somente no final da execução é que a condição de controle é avaliada para definir se outra repetição será executada ou não.

A escolha entre a utilização de `for`, `while` ou `do while` ocorrerá naturalmente com sua prática de uso. Algumas situações são melhores atendidas com o comando `for`, outras com o `while`, `do while` e assim por diante. Fique tranquilo, pois com o passar do tempo, e praticando, você terá cada vez mais experiência no assunto.

7. Funções

Funções são blocos de códigos customizados pelos programadores, nomeados com um título específico, com a intenção de realizar alguma tarefa ou ação que ocorrerá mais de uma vez em seu código. Entre os objetivos das funções está a diminuição da repetição de código, bem como a melhor manutenção dos projetos, pois são blocos que podem ser invocados a qualquer momento da execução do código-fonte.

Considere a seguinte situação. Em um projeto fictício há a necessidade de converter o valor de produtos de real (R\$) para dólares americanos (US\$). É possível fazer essa conversão em todo o seu código, repetindo a fórmula matemática que realiza essa operação toda vez que essa conta se fizer necessária.

O problema é que se esta fórmula mudar (se for necessário acrescentar algum tributo, por exemplo) será necessário mudar isso no código em todos os locais em que essa fórmula tenha sido utilizada. Neste caso, é possível criar um bloco de código customizado (função) que centralize em um único lugar a expressão matemática em questão, que recebe chamadas sempre que for necessário realizar o cálculo. Assim, se for preciso alterar a expressão matemática desta operação, este procedimento será feito somente em um lugar.

Funções geralmente são blocos de códigos de tratamento específico, tais como validação de campos numéricos, de texto, ou tratamento específico de algo que ocorre em várias páginas do site, fazendo com que o código seja escrito uma única vez, e não repetido a todo momento que precisa ser executado.

Para escrever uma função, utilize a sintaxe apresentada a seguir:

```
function nomeDaFunção(parâmetros)
{
    /* Bloco de código */

    // Retorno opcional
    return valor;
}
```

É possível criar funções que recebam nenhum, um ou mais argumentos de entrada, também conhecidos como *parâmetros*. Estes valores são passados na chamada da função, entre parênteses, no código-fonte. Ainda, a função pode retornar algum valor para quem a chamou, por meio da palavra reservada `return`, seguido do valor a ser retornado. Esse e os demais conceitos relacionados às funções ficarão mais claros por meio de um exemplo prático, que será exemplificado a seguir:

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10     function soma(a, b)
11     {
12       var resultado;
13       resultado = a + b;
14
15       return resultado;
16     }
17
18     var x=1, y=2, z=0;
19     z = soma(x, y);
20     document.write(z);
21
22   </script>
23 </body>
24 </html>

```

Código-Fonte 7-1: comandoFunction.htm

O resultado da interpretação do Código-Fonte 7-1 pode ser visualizado a seguir:

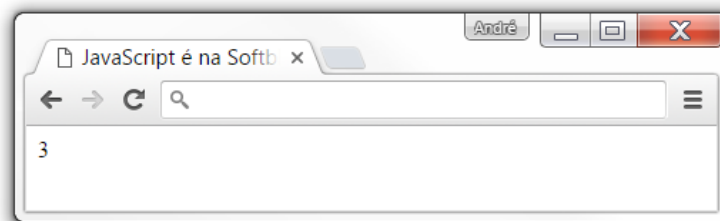


Figura 7-1: Exemplo de função customizada

No exemplo apresentado no Código-Fonte 7-1, observe que uma função é criada entre as linhas 10 e 16. A primeira etapa de criação da função é na linha 10, onde a instrução `function` deve ser informada, seguida do nome da função. Este nome é você, programador, que deve criar, não devendo utilizar nomes de funções já existentes do JavaScript. O nome também não deve ter espaços em branco, caracteres especiais ou começar com números. Essas são algumas regras de nomenclaturas de variáveis e funções do JavaScript e de boa parte das linguagens de programação.

Após o nome, você deve abrir e fechar parênteses e, caso você queira que essa função receba algum valor como parâmetro para ser executada, é necessário preencher entre os parênteses os nomes dos parâmetros que serão recebidos pela função. Esses parâmetros poderão ser acessados de dentro do bloco de código da função. No caso da função do Código-Fonte 7-1 o objetivo é criar um código que some dois valores, retornando o resultado final para o código que invocar essa função. Neste caso, o nome

dado para a função foi `soma` e, entre parênteses, foram definidos dois parâmetros para representar os operandos que serão somados. Esses parâmetros poderão ser acessados pelos seus respectivos nomes de parâmetros apresentados ainda na linha 10: `a` e `b`.

O corpo da função `soma` propriamente dita começa na linha 11. Sua primeira instrução é a criação de uma variável chamada `resultado` na linha 12, que receberá a soma de `a` e `b` conforme expressão apresentada na linha 13. Para retornar este valor ao código que fará uso dessa função, a instrução `return` foi utilizada na linha 15. No lado direito da instrução `return`, um único valor deve ser apresentado: é este o valor que será retornado ao código que utilizar esta função. No caso da função `soma`, é o valor contido na variável `resultado` que será retornado.

O uso da função `soma` pode ser observado no final do Código-Fonte 7-1. Observe que na linha 18 são criadas três variáveis: `x`, inicializada com o valor `1`; `y`, inicializada com o valor `2`; e `z`, inicializada com o valor `0`. Em seguida, na linha 19, finalmente a função `soma` é utilizada, passando para os seus parâmetros os valores contidos nas variáveis `x` e `y`, e armazenando o resultado retornado pela função na variável `z`. Neste caso a função `soma` irá realizar a operação de adição entre `a` e `b`, que contém os valores `1` e `2`, respectivamente, e retornará o valor `3` para ser armazenado em `z`, que tem o seu valor impresso na tela por meio do comando apresentado na linha 20.

É possível ainda construir funções que não retornem nenhum valor. Neste caso, considere o exemplo apresentado a seguir.

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10     function cumprimentar(horario, nome)
11     {
12       var frase;
13
14       if(horario < 12)
15       {
16         frase = "Bom dia ";
17       }
18       else if(horario < 18)
19       {
20         frase = "Boa tarde ";
21       }
22       else
23       {
24         frase = "Boa noite ";
25       }
26
27       frase = frase + nome;

```

```

28     document.write(frase);
29 }
30
31     cumprimentar(10, "André");
32
33     </script>
34 </body>
35 </html>

```

Código-Fonte 7-2: comandoFunctionSemRetorno.htm

O resultado da interpretação do Código-Fonte 7-2 pode ser visualizado a seguir:

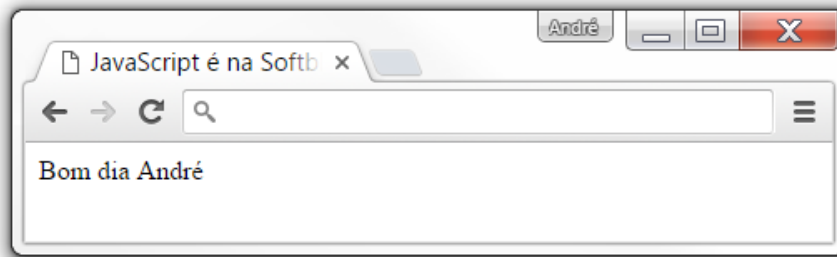


Figura 7-2: Exemplo de função customizada sem retorno

No exemplo apresentado no Código-Fonte 7-2, observe a criação da função `cumprimentar` entre as linhas 10 e 29. Essa função recebe dois parâmetros: `horario` e `nome`. O objetivo desse código é dar bom dia, boa tarde ou boa noite, citando o nome do usuário, dependendo do horário informado no parâmetro `horario`.

Neste caso, alguns comandos de decisão foram utilizados entre as linhas 14 e 25 para definir qual cumprimento será realizado e, na linha 27, o cumprimento em questão armazenado na variável `frase` será concatenado com o nome do usuário, que é também um parâmetro recebido pela função. Neste caso, ao fazer o uso da função na linha 31 com os parâmetros `10` e `André`, teremos a sentença `Bom dia André` formada pela função `cumprimentar`, e impressa na tela por meio da linha 28, que faz parte da função. Dessa forma temos um exemplo de uso de função que não precisa retornar valores.

É possível ainda criar funções que não recebam nenhum parâmetro, que apenas escrevam `Bom dia`, por exemplo. Para isso, basta ocultar os parâmetros informados na linha de criação da função, como por exemplo, `function cumprimentar()`.

8. Eventos

O JavaScript permite iniciar a execução de comandos ou blocos de código a partir de eventos pré-definidos que podem ocorrer em objetos HTML de uma página ou um site. Em outras palavras, é possível executar comandos JavaScript a partir do clique do mouse em alguma parte da página, ao preencher determinado campo ou ainda ao apertar algum botão, entre outros.

Existem vários eventos tratando diversas situações que podem ocorrer na interface ou comportamento de uma página web. Os principais estão listados a seguir:

Evento	Descrição
onBlur	Executa o código quando o usuário retira a ação do campo em questão. Utilizado em campos de formulários.
onChange	Executa o código quando o usuário altera o valor do campo em questão. Utilizado em campos de formulários.
onClick	Executa o código quando o usuário clica sobre o objeto da marcação em questão.
onFocus	Executa o código quando o usuário ativa o campo em questão. Utilizado em campos de formulários.
onLoad	Executa o código quando a página termina de ser carregada. Geralmente utilizada na marcação <code><body></code> .
onUnload	Executa o código quando o usuário sai da página, seja ao clicar em um link ou fechar o navegador. Geralmente utilizada na marcação <code><body></code> .
onMouseOver	Executa o código quando o usuário passa com o mouse sobre o objeto da marcação em questão.
onMouseOut	Executa o código quando o usuário retira o mouse de cima do objeto da marcação em questão.
onMouseDown	Executa o código quando o usuário clica sobre o objeto da marcação em questão, mais precisamente ao apertar o botão do mouse.
onMouseUp	Executa o código quando o usuário clica sobre o objeto da marcação em questão, mais precisamente ao soltar o botão do mouse.
onKeyPress	Executa o código quando o usuário digita (pressiona e solta) uma tecla com o objeto da marcação em questão ativado.

onKeyDown	Executa o código quando o usuário pressiona uma tecla com o objeto da marcação em questão ativado.
onKeyUp	Executa o código quando o usuário solta uma tecla com o objeto da marcação em questão ativado.
onSubmit	Executa o código quando o usuário pressiona o botão de <i>submit</i> de um formulário. Geralmente utilizado com a marcação <code><FORM></code> .

Tabela 8-1: Principais eventos JavaScript

A utilização destes recursos deve ser feita da seguinte forma: incluir o nome do evento que deseja tratar como um atributo de uma marcação HTML de abertura, atribuindo a este o(s) comando(s) que deseja executar. Antes de visualizarmos alguns exemplos de utilização destes recursos, convido você a conhecer um comando JavaScript que apresenta uma pequena janela na tela com uma mensagem customizada para o usuário. Esse comando será útil para a apresentação dos eventos JavaScript e se chama `alert`. Considere inicialmente o código apresentado a seguir:

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       alert("Seja bem-vindo(a)!");
11
12     </script>
13   </body>
14 </html>

```

Código-Fonte 8-1: comandoAlert.htm

O resultado da interpretação do Código-Fonte 8-1 pode ser visualizado a seguir:

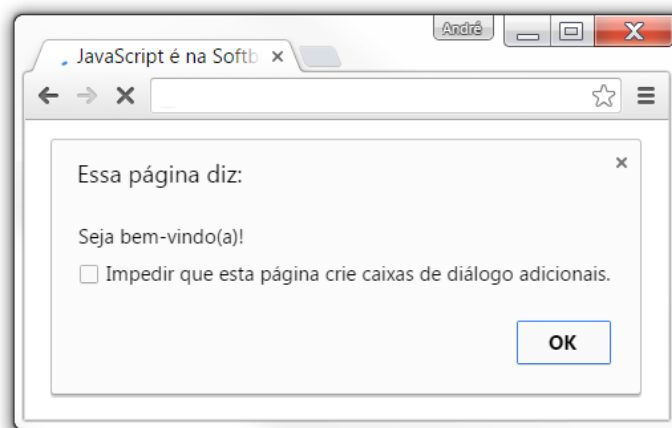


Figura 8-1: Caixa de diálogo de exibição de mensagens

No exemplo apresentado no Código-Fonte 8-1, temos a utilização do comando `alert` na linha 10, que recebe como parâmetro uma sentença de texto e a apresenta na tela para o usuário, com um botão com o texto `OK` para o fechamento da janela. Este comando é bastante simples e ao mesmo tempo bastante útil.

Agora que você conhece o comando `alert`, vamos dar uma olhada em um exemplo com vários casos de uso de eventos JavaScript em uma página web. Para isso, considere o código apresentado a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body onLoad="alert('Página carregada!');">
08
09     <a href="#" onMouseOver="alert('Passou o mouse');">Passe o mouse aqui</A>
10     <a href="#" onClick="alert('Realizou um clique');">Clique aqui</A>
11
12     <form>
13       <input onFocus="alert('Ativou o campo');" value='Clique neste campo'>
14       <input onKeyPress="alert('Digitou');" value='Digite neste campo'>
15     </form>
16
17   </body>
18 </html>
```

Código-Fonte 8-2: exemplosEventos.htm

No exemplo apresentado no Código-Fonte 8-2, várias coisas diferentes podem acontecer dependendo de como você interagir com a página. Inicialmente a página será carregada e imediatamente será apresentada na tela a mensagem `Página carregada!`, por meio do uso de um comando `alert`. Isso irá ocorrer porque foi programado na linha 7 o evento `onLoad`, dentro da marcação `<body>`. Observe que o atributo `onLoad` deve apresentar entre aspas as instruções JavaScript que deverão ser realizadas quando a página for carregada. Neste caso foram utilizadas aspas duplas para delimitar o começo e o fim do código que será executado.

Preste bastante atenção no comando `alert` existente na linha 7, entre as aspas do atributo `onLoad`. O comando `alert` poderá causar conflito de aspas caso a sentença que ele imprimirá também seja delimitada por aspas duplas, já utilizadas para delimitar o código que será executado. Por este motivo o uso de outro tipo de aspas foi sugerido, no caso, aspas simples. Esta é uma forma de resolver este problema.

Depois de fechar a mensagem que aparece quando a página é carregada, experimente passar o mouse sobre o link do texto `Passe o mouse aqui` para ver o que acontece. Como foi programado na linha 9, a marcação `<a>` define que o evento

`onMouseOver` executará algum código JavaScript. Por este motivo, ao passar o mouse neste texto, o comando `alert` com a frase `Passou o mouse` será executado.

Já na linha 10 do Código-Fonte 8-2, outro link de texto é apresentado, com o texto `Clique aqui`, mas neste caso somente disparará o comando `alert` quando o usuário clicar sobre o link, uma vez que foi utilizado o evento `onClick`.

Alguns eventos específicos de campos de formulários também foram utilizados no exemplo do Código-Fonte 8-2. O primeiro deles está apresentado na linha 13, disparando um comando `alert` com o texto `Ativou o campo` quando o usuário ativar o campo em questão para digitar algum valor. E o segundo na linha 14, que somente será disparado quando o usuário apertar alguma tecla com o campo ativo, exibindo na tela a sentença `Digitou`.

No exemplo apresentado neste tópico, somente comandos `alert` foram executados a partir dos eventos JavaScript, mas vale a pena reforçar que esses eventos podem disparar qualquer tipo de código JavaScript. Sinta-se livre para brincar com os demais eventos existentes, testando e conhecendo ainda mais suas capacidades.

Neste tópico vimos o comando `alert`, utilizado para apresentar mensagens para o usuário. O JavaScript apresenta outros tipos de caixas de diálogos bem interessantes, não apenas para apresentação de mensagens, mas também para tomada de decisão e até mesmo preenchimento de informações. Esses recursos serão abordados a partir do próximo tópico deste e-book.

9. Caixas de diálogo

O JavaScript disponibiliza alguns métodos de interação com o usuário por meio de caixas de diálogo pré-existentes, que podem ser customizadas pelo programador. Por meio destes recursos é possível exibir alertas na tela, bem como solicitar informações ao usuário.

A seguir estão descritas as três formas mais comuns de exibir caixas de diálogos utilizando JavaScript.

9.1. Caixa de diálogo alert

O comando `alert`, abordado anteriormente neste livro, permite exibir uma mensagem para o usuário. Este método exibe uma janela que se sobrepõe ao conteúdo apresentado pelo navegador, dando prioridade para que sua mensagem seja lida pelo usuário.

A sintaxe de execução deste comando é a seguinte:

```
alert(mensagem);
```

Um exemplo de uso deste comando já foi abordado no Código-Fonte 8-2, podendo o resultado ser observado na Figura 8-1.

9.2. Caixa de diálogo confirm

O comando `confirm` é uma extensão do comando `alert`. Assim como o primeiro, este também exibe uma mensagem para o usuário na tela, por meio de uma caixa de diálogo. A diferença é que o método `confirm` apresenta também para o usuário dois botões: um botão `OK` e um botão `Cancel`. Ambos os botões encerram a apresentação da caixa de diálogo. No entanto, é possível detectar no código JavaScript se o usuário utilizou a opção `OK` ou `Cancel` e, com base nisso, executar um determinado trecho de código JavaScript ou não.

Como já vimos no decorrer deste e-book, quando o tema de funções foi tratado, funções podem retornar valores. Por meio do retorno do método `confirm` é possível saber qual botão o usuário pressionou, como será possível observar no exemplo apresentado logo mais, ainda neste tópico. Antes, vamos conhecer a sintaxe do comando `confirm`:

```
confirm(conteúdo);
```

O código-fonte apresentado na sequência aborda um exemplo de utilização do `confirm`:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       resposta = confirm('Deseja realizar a operação?');
11
12       if(resposta == true)
13       {
14         alert('Pressionou Ok');
15       }
16       else
17       {
18         alert('Pressionou Cancel');
19       }
20
21     </script>
22   </body>
23 </html>
```

Código-Fonte 9-1: comandoConfirm.htm

O resultado da interpretação do Código-Fonte 9-1 pode ser visualizado a seguir:

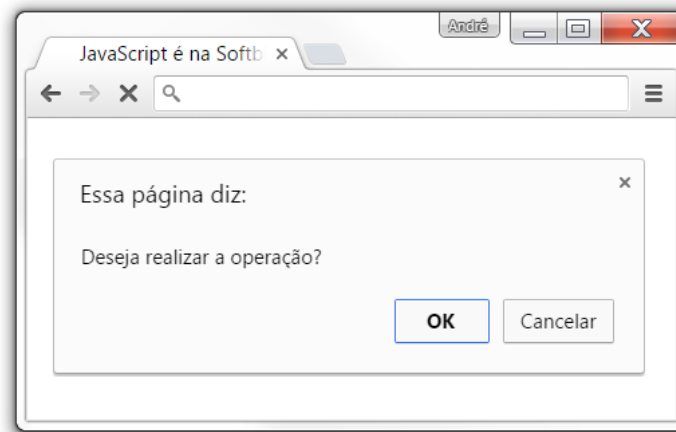


Figura 9-1: Caixa de diálogo de tomada de decisão

No exemplo apresentado no Código-Fonte 9-1, uma caixa de diálogo de tomada de decisão é exibida para o usuário, junto com uma mensagem. Caso o usuário clique sobre a opção `OK`, o valor booleano `true` será retornado pela função `confirm`. Já se o usuário clicar em `Cancel`, o valor booleano `false` será retornado. Observe na linha 10 que o retorno de `confirm` é atribuído à variável `resposta`, que é avaliada pelo comando `if` apresentado na linha 12. Dependendo da opção escolhida pelo usuário, um ou outro bloco de código JavaScript será executado.

Caixas de diálogo do tipo `confirm` são bastante úteis quando a sua página precisa confirmar a realização de uma determinada ação do usuário.

9.3. Caixa de diálogo `prompt`

Outra forma de interação com o usuário é por meio das caixas de diálogo do tipo `prompt`. Esta caixa exibe uma mensagem para o usuário, assim como as demais, porém disponibiliza um campo de texto para que o usuário preencha alguma informação. Esse valor, preenchido pelo usuário, é retornado pela função `prompt` para o seu código JavaScript fazer uso dele.

É possível ainda fazer com que o campo texto que será apresentado para o usuário já apareça com algum texto preenchido. Para fazer isso, o segundo parâmetro da função `prompt`, que é opcional, deve ser informado. A sintaxe de execução deste comando é a seguinte:

```
prompt(conteúdo [, texto_sugerido]);
```

Para ilustrar o uso do comando `prompt`, considere o seguinte código-fonte:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       resposta = prompt('Qual o seu nome?', 'Digite aqui o seu nome');
11
12       alert('Bem vindo(a) ao site ' + resposta);
13
14     </script>
15   </body>
16 </html>
```

Código-Fonte 9-2: comandoPrompt.htm

O resultado da interpretação do Código-Fonte 9-2 pode ser visualizado a seguir:

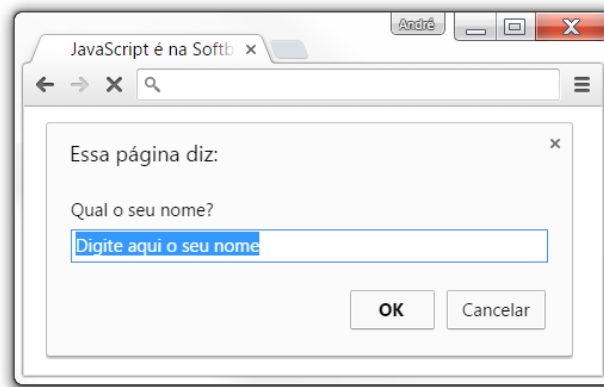


Figura 9-2: Caixa de diálogo de solicitação de informação

No exemplo apresentado no Código-Fonte 9-2, o comando `prompt` é apresentado na linha 10, exibindo a sentença `Qual o seu nome?` para o usuário, e apresenta o campo para preenchimento com o texto `Digite aqui o seu nome`, ajudando a orientar o usuário.

Neste caso, assim que o usuário termina de preencher o seu nome e fecha a caixa de diálogo, o JavaScript continua sua execução, atribuindo a informação preenchida à variável `resposta` e, em seguida, dá boas-vindas para a pessoa em questão por meio de um comando `alert`.

10. Abrindo uma URL

Entre os diversos objetos que o JavaScript disponibiliza para você utilizar, está o que representa a barra de endereços do navegador: o objeto `location`. Por meio deste objeto é possível acessar a URL da página aberta no navegador e também definir um novo endereço, redirecionando o usuário para outra página dinamicamente via JavaScript.

Para compreender o uso do objeto `location` e de sua propriedade `href`, responsável pela informação de endereço de URL do navegador, considere o código-fonte apresentado a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08
09     <input type=BUTTON onClick="acessarUrl();" value="Acessar URL">
10
11     <script>
12
13       function acessarUrl()
14       {
15         location.href = "http://www.softblue.com.br";
16       }
17
18     </script>
19   </body>
20 </html>
```

Código-Fonte 10-1: propriedadeLocationHref.htm

No Código-Fonte 10-1 é criado um botão HTML por meio da marcação `<input>` apresentada na linha 9. Este botão, por sua vez, dispara um código JavaScript quando clicado, conforme o evento `onClick` foi programado na mesma linha 9. O código JavaScript em questão invoca a função `acessarUrl`, cujo corpo de código foi definido entre as linhas 13 e 16. Assim que o usuário clica sobre o botão, a função JavaScript é executada, definindo um novo endereço para ser acessado pelo navegador, por meio da instrução apresentada na linha 15. Neste caso, o usuário é redirecionado para o site da Softblue.

Como é possível observar executando o exemplo apresentado no Código-Fonte 10-1, a URL informada é carregada na mesma guia do navegador web do visitante da página. Caso a sua intenção seja a de abrir uma URL em uma nova guia, mantendo a

guia atual no endereço corrente da página, você precisará utilizar outra instrução JavaScript, que é o comando `open`, disponibilizado pelo objeto `window`.

A instrução `open` recebe como parâmetro o endereço da URL que deve ser aberta em uma nova guia do navegador. Observe a seguir uma adaptação do código-fonte anteriormente apresentado neste tópico, desta vez fazendo o uso do comando `open` do JavaScript:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08
09     <input type=BUTTON onClick="acessarUrlNovaJanela();" value="Acessar em nova janela">
10
11     <script>
12
13       function acessarUrlNovaJanela()
14       {
15         open("http://www.youtube.com/softbluecursos");
16       }
17
18     </script>
19   </body>
20 </html>
```

Código-Fonte 10-2: comandoOpen.htm

O Código-Fonte 10-2 apresenta novamente um botão HTML criado por meio da marcação `<input>` que invoca uma função JavaScript a partir do evento `onClick`. Neste caso a função `acessarUrlNovaJanela`, apresentada entre as linhas 13 e 16, faz uso do comando `open`, abrindo em uma nova guia do navegador no endereço informado.

11. Agendando execução de códigos

Outra funcionalidade muito interessante do JavaScript é o agendamento de execução de códigos. Suponha que após alguns segundos de visita de um visitante na página, você deseje apresentar uma mensagem a ele. Essa e outras necessidades podem ser atendidas por meio do uso do comando `setTimeout`.

O comando `setTimeout` basicamente opera com dois parâmetros: o primeiro deles é o nome da função JavaScript que você pretende invocar. Por este motivo, é recomendado que o código em questão seja criado dentro de uma função customizada. O segundo parâmetro dessa instrução é o tempo, em milissegundos, que deve haver de intervalo entre a execução do comando `setTimeout` e a execução do código desejado.

Basicamente, este comando informa via JavaScript a função que deve ser executada e daqui há quanto tempo ela deve ser executada.

Um exemplo de utilização da função `setTimeout` pode ser observado no código-fonte apresentado a seguir:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       function minhaTarefa()
11       {
12         alert("Tarefa executada!");
13       }
14
15       setTimeout(minhaTarefa, 5000);
16
17     </script>
18   </body>
19 </html>
```

Código-Fonte 11-1: comandoSetTimeout.htm

No exemplo apresentado no Código-Fonte 11-1, há inicialmente a construção de uma função customizada chamada `minhaTarefa` entre as linhas 10 e 13. O código apresentado no corpo dessa função será executado de forma agendada neste exemplo.

O agendamento propriamente dito é construído na linha 15, por meio da função `setTimeout`. O primeiro parâmetro deste comando é o nome da função que deverá ser disparada e o segundo parâmetro é o tempo em milissegundos de intervalo entre a

execução da linha 15 e a chamada da função em questão. Neste caso foi definido que a função `minhaTarefa` seja invocada depois de 5000 milissegundos. Como cada segundo é composto de 1000 milissegundos, a função será invocada depois do intervalo de 5 segundos. Neste caso, o usuário irá abrir a página e, 5 segundos depois, a mensagem `Tarefa executada!` será exibida na tela.

12. Alterando uma imagem via JavaScript

Com o JavaScript, é possível acessar boa parte dos recursos das páginas web, se devidamente programados. Este acesso é realizado capturando uma referência para o elemento em questão por meio da instrução `getElementById`, disponibilizada por meio do objeto `document` do JavaScript.

Mas para que isso funcione, o elemento deve apresentar o atributo `id` em sua respectiva marcação HTML, atrelando o elemento em questão a um nome único, o que permite o seu acesso por meio deste nome, via JavaScript.

Para ilustrar um exemplo de manipulação de elemento de página via JavaScript, considere o código-fonte apresentado a seguir, que construirá um botão HTML responsável por trocar uma imagem da página:

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08
09     
10     <input type="BUTTON" onClick="trocarImagem();" value="Trocar imagem">
11
12     <script>
13
14       function trocarImagem()
15       {
16         var imagem = document.getElementById("corImagem");
17         imagem.src = "images/red.png";
18       }
19
20     </script>
21   </body>
22 </html>
```

Código-Fonte 12-1: propriedadeImgSrc.htm

O resultado da interpretação do Código-Fonte 12-1 pode ser visualizado a seguir:

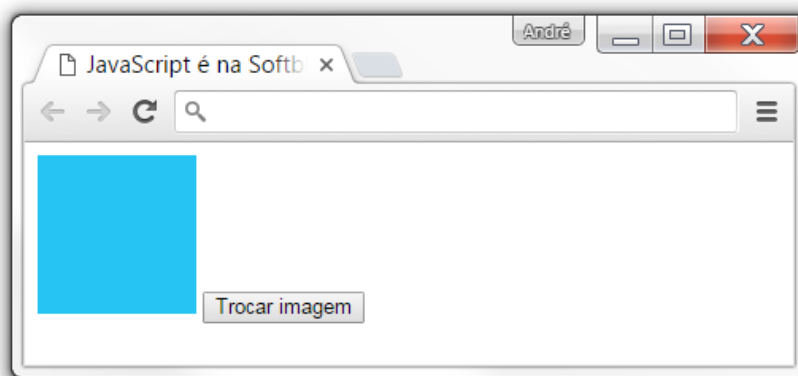


Figura 12-1: Alterando imagem via JavaScript

No exemplo apresentado no Código-Fonte 12-1, inicialmente é apresentada uma imagem na tela por meio da marcação `` construída na linha 9. Observe que esta marcação aponta para uma imagem chamada `blue.png` e que foi atrelada a esta marcação o atributo `id` com o valor `corImagem`. É por meio deste nome que será possível acessar este elemento via JavaScript.

Um botão HTML foi construído na linha 10, responsável por disparar a função `trocarImagem`, apresentada entre as linhas 14 e 18. O disparo desta função é atrelado a este botão por meio do evento `onClick`, programado para a marcação `<input>` na mesma linha 10.

A engrenagem principal do exemplo apresentado neste tópico está nas linhas 16 e 17. Na linha 16 uma referência do elemento de imagem é capturada e armazenada na variável `imagem`, para que possa ser manipulada no JavaScript. Observe que a captura desta referência é realizada por meio da instrução `document.getElementById`, passando como parâmetro o nome dado ao elemento que deve ser capturado, por meio do atributo `id`. Neste caso, o `id` é `corImagem`, definido para o elemento `` na linha 9 do arquivo.

Com a referência para o elemento de imagem da página em mãos e disponibilizado pela variável `imagem`, basta trocar a propriedade `src` desta marcação dinamicamente, para apontar para outra imagem. É exatamente isso que a linha 17 faz, ao acessar o elemento contido na variável `imagem`, acessando sua propriedade `src`, que representa o caminho do arquivo, e definindo o caminho de outra imagem para ser apresentada na tela.

No exemplo apresentado no Código-Fonte 12-1, a página será carregada com uma imagem de cor azul e será trocada por uma imagem de cor vermelha quando o botão HTML for clicado pelo usuário. As imagens apresentadas neste exemplo podem ser obtidas junto com os códigos-fonte deste e-book no site da Softblue, na área de aluno.

13. Objeto form

Um dos objetos mais importantes do JavaScript é o que permite a manipulação de formulários HTML, muito utilizado para validar informações e realizar outros processamentos. O objeto `form` representa um formulário contido em uma página HTML e, por meio dele, é possível acessar todos os seus campos e valores e manipulá-los de acordo com a necessidade.

Os principais recursos do `form` são:

Comando	Descrição
<code>form.action</code>	Representa o atributo <code>action</code> informado na marcação <code><form></code> .
<code>form.name</code>	Representa o nome do formulário.
<code>form.method</code>	Representa o atributo <code>method</code> informado na marcação <code><form></code> .
<code>form.submit()</code>	Envia o formulário e suas informações para o destino informado no atributo <code>action</code> .
<code>form.nomecampo</code>	Permite acessar as informações do campo cujo nome tenha sido informado no lugar da propriedade <code>nomecampo</code> .

Tabela 13-1: Principais recursos do objeto form

Com os recursos apresentados no decorrer deste livro, e também com as informações que serão abordadas ainda neste tópico, várias validações interessantes poderão ser realizadas no seu formulário HTML, a fim de verificar se o usuário preencheu determinado campo, selecionou determinado valor, entre outras possibilidades. Para isso, vamos ver na prática a validação de um formulário utilizando JavaScript, começando com a montagem do código HTML inicial dessa página de exemplo.

13.1. Montando um formulário HTML

O formulário HTML que será utilizado nos próximos tópicos será um formulário simples de cadastro, com algumas características. Ele terá os atributos `method` e `action` da marcação `<form>`, terá o seu botão de enviar definido como um botão comum (`type=BUTTON`) e receberá o nome `meu_form` por meio do atributo `name` utilizado na marcação `<form>`. Desta forma o botão de envio do formulário, ao invés de enviar os

dados para o servidor, chamará uma função JavaScript para validar o formulário, e, se tudo estiver correto, encaminhará o mesmo para o servidor processar.

O processamento por parte do servidor não será abordado neste e-book, pois ele depende de tecnologias que executem do lado servidor, como PHP ou outras linguagens de programação voltadas para web.

Utilize o código a seguir como base para os exemplos que serão discutidos nos próximos tópicos:

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09       function validarFormulario()
10       {
11         /* Código de validação */
12       }
13     </script>
14
15     <form name="meu_form" method="get" action="?">
16
17       Nome completo:
18       <input name="campo_nome" type="text">
19       <br>
20
21       Sexo:
22       <input name="campo_sexo" type="radio" value="M">Masculino
23       <input name="campo_sexo" type="radio" value="F">Feminino
24       <br>
25
26       <input name="campo_maior" type="checkbox" value="maior">
27       Declaro ser maior de 18 anos
28       <br>
29
30       Como conheceu o site:
31       <select name="campo_como">
32         <option value="Nenhum" default>Selecione...</option>
33         <option value="Ind">Indicação de amigo</option>
34         <option value="Out">Outro site</option>
35         <option value="Goo">Google</option>
36       </select>
37       <br>
38
39       <input type="reset" value="Limpar dados">
40       <input type="button" value="Enviar" onClick="validarFormulario();">
41
42     </form>
43   </body>
44 </html>

```

Código-Fonte 13-1: validandoFormularioBaseHTML.htm

O resultado da interpretação do Código-Fonte 13-1 pode ser visualizado a seguir:

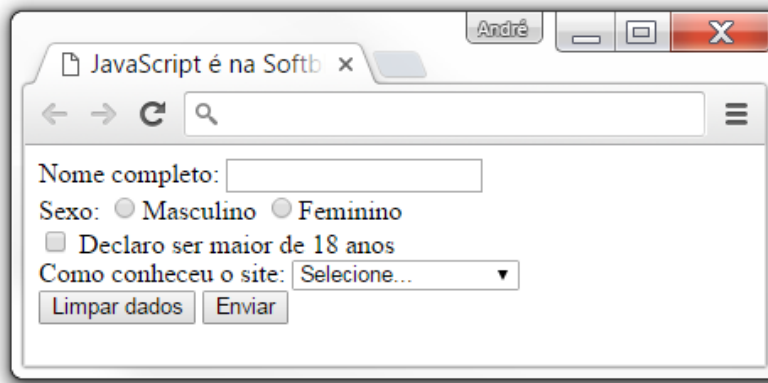
A screenshot of a web browser window. The title bar shows a single tab titled 'JavaScript é na Softb' and a window name 'André'. The browser's address bar is empty. The main content area displays an HTML form. The form starts with a text input field labeled 'Nome completo:'. Below this is a 'Sexo:' label followed by two radio buttons, 'Masculino' and 'Feminino'. Then there is a checkbox labeled 'Declaro ser maior de 18 anos'. Below the checkbox is a dropdown menu labeled 'Como conheceu o site:' with the text 'Selecione...'. At the bottom of the form are two buttons: 'Limpar dados' and 'Enviar'.

Figura 13-1: Formulário HTML

No exemplo apresentado no Código-Fonte 13-1, temos a construção de uma página web com um formulário fictício de cadastro de usuário com campos dos mais variados tipos disponibilizados pelo HTML. Até este ponto não há grandes segredos, nem mesmo novidades.

Como já foi visto no e-book de HTML da Softblue, é recomendado o uso do atributo `name` para os campos do formulário, pois é por meio deste atributo que os campos poderão ser processados posteriormente por uma linguagem de programação no lado servidor da aplicação. É por meio deste mesmo atributo que acessaremos os campos via JavaScript na validação que será abordada neste tópico. Por este motivo, todos os campos foram nomeados em suas respectivas marcações de criação nas linhas 18, 22, 23, 26 e 31.

Observe também que a própria marcação `<form>` recebeu um nome específico por meio da instrução `name="meu_form"` na linha 15, para que o formulário em si possa ser manipulado via JavaScript. A intenção deste exemplo é submeter o formulário para o servidor via programação JavaScript se a validação ocorrer conforme o esperado. Portanto será necessário acessar o formulário para realizar este envio.

Ainda no exemplo apresentado no Código-Fonte 13-1, observe que o botão `Enviar` do formulário invoca uma função JavaScript chamada `validarFormulario()` na linha 40. Isso se deve ao fato de que não queremos que o formulário seja submetido automaticamente para o servidor, como ocorreria se a marcação `<input>` da linha 40 fosse do tipo `submit`. O que queremos neste exemplo é disparar uma função JavaScript que avalie o preenchimento do formulário e, dependendo do resultado desta avaliação, que o formulário seja enviado para o servidor ou alguma mensagem de erro seja apresentada para o usuário na tela.

O botão `Enviar` dispara a execução da função `validarFormulario()` apresentada entre as linhas 9 e 12, que neste momento ainda está vazia. Não se preocupe, a ideia é construir o corpo desta função no decorrer dos próximos tópicos.

13.2. Capturando dados de campos texto

Os campos de textos são os mais simples e intuitivos de serem manipulados por JavaScript, por permitirem que seus valores sejam acessados diretamente pela propriedade `value`. Mas antes, para que possamos acessar as propriedades de um campo de formulário, é necessário primeiro acessar o formulário em si via JavaScript.

Os formulários HTML são disponibilizados no JavaScript por meio do objeto `document`, que representa os elementos da página web, mais precisamente por um array de formulários contido na propriedade `forms` deste objeto. Por este motivo, é necessário acessar o elemento `document`, sua propriedade `forms`, seguido do nome do formulário entre colchetes e aspas, indicando seu índice de acesso, construído na marcação `<form>` na linha 15. Como o nome do formulário é `meu_form`, é este o valor que deve ser informado entre colchetes. O código-fonte a seguir demonstra como fazer este acesso completo para capturar o valor de um campo texto, no caso o campo `nome`, e exibi-lo na tela:

```
function validarFormulario()
{
    conteudo_nome = document.forms['meu_form
'].campo_nome.value;
    alert('O nome informado no formulário é: ' + conteudo_nome);
}
```

Observe a instrução `document.forms['meu_form'].campo_nome.value`. Ela inicia acessando a página web por meio de `document`. Em seguida o formulário por meio de `forms['meu_form']`, seguido do nome do campo `campo_nome` e, por último, da propriedade que acessa o valor contido em um campo de texto, que é a propriedade `value`. Este valor é atribuído para a variável `conteudo_nome`, sendo impresso na tela pelo comando `alert` apresentado na linha seguinte, quando o botão `Enviar` for clicado.

Antes de validarmos esse campo, vamos dar uma olhada em como acessar os valores preenchidos e/ou selecionados nos outros tipos de campos, via JavaScript.

13.3. Capturando dados de campos de seleção (combo box)

Com a mesma facilidade de acesso aos conteúdos dos campos de texto, é possível acessar o valor selecionado dentro de um campo de seleção (marcação `<select>`), por meio da mesma propriedade `value`.

Existe ainda uma propriedade chamada `selectedIndex` para os campos deste tipo, que permite saber o número da opção que foi escolhida, seguindo a ordem de apresentação no campo HTML. Esta alternativa pode ser útil em casos como este, para saber se o usuário selecionou alguma opção ou se manteve o campo sem nenhuma seleção, pois geralmente a primeira opção dos campos de seleção é justamente o texto `Selecione...`, que ocupa o índice `0` dessa listagem.

O código-fonte a seguir exhibe as duas propriedades comentadas neste tópico: o valor selecionado pelo usuário e o índice numérico que este valor possui na listagem de opções do campo:

```
function validarFormulario()
{
    conteudo_como = document.forms['meu_form'].campo_como.value;
    numero_como   = document.forms['meu_form'].campo_como.selectedIndex;
    alert('A opção escolhida é: ' + numero_como + ': ' + conteudo_como);
}
```

Neste exemplo, o comando `alert` irá informar na tela a opção selecionada pelo campo `campo_como` e também o número da opção escolhida. Lembre-se que a numeração de ordenamento das opções começa sempre em zero.

Vale a pena citar que quando se fala em informar o valor da opção selecionada, fala-se em exibir o valor preenchido no atributo `value` da opção em questão. Por exemplo, ao selecionar a opção `Google` no `campo_como`, este exemplo exibirá na tela o texto `3: Goo`, pois é o valor `Goo` que foi atribuído para esta opção na marcação `<option>` existente na linha 35 do Código-Fonte 13-1.

Se for necessário acessar o valor estendido da opção, aquele definido entre a abertura e o fechamento de cada marcação `<option>`, utilize o seguinte código:

```
function validarFormulario()
{
    numero_como   = document.forms['meu_form'].campo_como.selectedIndex;
    descricao_como = document.forms['meu_form'].campo_como.options[numero_como].text;
    alert('A descrição da opção escolhida é: ' + descricao_como);
}
```

O primeiro passo para exibir a descrição de uma opção em um campo de seleção é capturar a posição que foi selecionada. Neste caso, a informação pode ser acessada por meio da propriedade `selectedIndex` vista anteriormente, a qual foi armazenada na variável `numero_como` neste exemplo.

Em seguida, sabendo a posição numérica da opção selecionada, deve ser realizado um acesso à propriedade `options` do campo `campo_como`, que apresenta um array com a lista de opções escritas por extenso, e, nessa lista, a posição numérica da opção selecionada deve ser informada entre colchetes. Como a posição numérica está contida

na variável `numero_como`, basta informar essa variável entre colchetes. Por último, é necessário acessar a propriedade `text` da opção em questão.

Como você pôde perceber, são vários acessos, o que pode parecer complicado neste primeiro momento, mas fica bastante simples quando se analisa cada acesso isoladamente. Para deixar ainda mais claro, observe a ordem em que os acessos ocorrem:

```
document
```

Acessa a página web que contém o formulário.

```
document.forms['meu_form']
```

Acessa o formulário `meu_form`, que contém o campo.

```
document.forms['meu_form'].campo_como
```

Acessa o campo `campo_como`, que contém as opções de seleção.

```
document.forms['meu_form'].campo_como.options[numero_como]
```

Acessa o elemento que representa a opção de seleção na posição `numero_como`.

```
document.forms['meu_form'].campo_como.options[numero_como].text
```

Acessa a descrição da opção de seleção em questão.

13.4. Capturando dados em campos de marcação (checkbox)

Os campos de marcação também possuem características específicas para terem suas informações acessadas. Se o código JavaScript tentar acessar o seu valor utilizando a propriedade `value`, a mesma irá trazer o valor definido na caixa de marcação por meio do atributo `value`, independente se a mesma tiver sido marcada ou não pelo usuário.

Para saber se uma caixa de marcação foi assinalada, é necessário acessar a propriedade `checked` do campo em questão. Observe no código-fonte a seguir as duas situações comentadas neste tópico: o acesso ao valor definido pelo atributo `value` de um campo de marcação e, em seguida, o acesso à propriedade que indica se o mesmo está marcado ou não:

```
function validarFormulario()
{
    conteudo_maior = document.forms['meu_form'].campo_maior.value;
    alert('O valor da caixa de marcação é: ' + conteudo_maior);

    marcacao_maior = document.forms['meu_form'].campo_maior.checked;
    alert('A caixa de marcação foi marcada? ' + marcacao_maior);
}
```

Neste exemplo, note que o primeiro comando `alert` sempre apresentará o valor `maior` que foi definido no formulário HTML na linha 26, independente se a caixa de marcação `Declaro ser maior de 18 anos` estiver marcada ou não. Para saber se está marcada, o segundo comando `alert` captura a propriedade `checked`, que retorna `true` caso a mesma esteja marcada, ou `false` caso contrário.

13.5. Capturando dados de campos de rádio (radio button)

De todos os tipos de campos existentes no HTML, o tipo rádio é o mais complexo de ser manipulado via JavaScript, pois as diferentes opções rádio para um mesmo campo devem ser definidas com o mesmo nome de campo.

Parece confuso, não é mesmo? Mas observe o exemplo apresentado no Código-Fonte 13-1, onde o campo de nome `campo_sexo` deve apresentar as opções `Masculino` e `Feminino` utilizando botões do tipo rádio, nas linhas 22 e 23. Notou que ambas apresentam o atributo `name="campo_sexo"`? Isso deve ser feito assim para que o navegador web possa desmarcar uma opção quando outra for selecionada. Experimente marcar este campo clicando na bolinha que representa a opção `Feminino` e depois na bolinha que representa a opção `Masculino`. Você vai perceber que, automaticamente, a opção `Feminino` será desmarcada quando você clicar em `Masculino`.

A ideia envolvida na busca pela opção de rádio selecionada para um determinado campo inclui uma varredura em todas as opções de rádio existentes no formulário, referente ao campo em questão, verificando cada uma até encontrar qual foi selecionada. O exemplo a seguir demonstra esta funcionalidade, com alguns comentários na sequência:

```
function validarFormulario()
{
    total_opcoes = document.forms['meu_form'].campo_sexo.length;
    opcao_escolhida = null;

    for(i=0; i<total_opcoes; i++)
    {
        if(document.forms['meu_form'].campo_sexo[i].checked == true)
        {
            opcao_escolhida = document.forms['meu_form'].campo_sexo[i].value;
        }
    }
    alert('O sexo marcado no formulário é: ' + opcao_escolhida);
}
```

O primeiro passo do exemplo apresentado foi acessar a propriedade `length` do campo `campo_sexo`, que retorna quantas opções de rádio existem no formulário com este nome de campo. Com esta informação em mãos e armazenada na variável `total_opcoes`, é realizado um laço de repetição `for` que irá começar definindo a variável `i` com o valor

0 e irá repetir até que `i` atinja o valor `total_opcoes`, ou seja, estamos preparando um laço de repetição que irá navegar por todos os campos rádio de nome `campo_sexo`.

Dentro do laço de repetição `for` é acessada a propriedade `checked` do campo rádio em questão. Observe que `document.forms['meu_form'].campo_sexo` representa um array com as opções de rádio apresentadas para este campo. Por este motivo, ao acessar `campo_sexo[i]` você estará acessando cada uma dessas opções, verificando se a mesma foi selecionada, por meio da propriedade `checked`. O elemento da lista que retornar `true` para este acesso é justamente o elemento selecionado. A partir daí fica fácil: basta acessar a propriedade `value` do elemento `campo_sexo[i]` para saber o valor do campo selecionado.

13.6. Validando dados de um formulário e enviando-o ao servidor

Agora que você viu uma coleção de propriedades de acesso aos campos de formulários HTML em JavaScript, fica mais fácil pensar em um código que valide os mesmos, como faz o código-fonte apresentado a seguir:

```

01 <!DOCTYPE html>
02 <html lang="pt-br">
03   <head>
04     <meta charset="utf-8">
05     <title>JavaScript é na Softblue!</title>
06   </head>
07   <body>
08     <script>
09
10       function validarFormulario()
11       {
12         // Captura o campo de texto que representa o nome do usuário
13         conteudo_nome = document.forms["meu_form"].campo_nome.value;
14
15         // Validação se o nome foi preenchido
16         if(conteudo_nome == "")
17         {
18           alert("É necessário preencher o campo nome corretamente.");
19           return;
20         }
21
22         // Captura o valor do campo que representa o sexo do usuário
23         total_opcoes = document.forms["meu_form"].campo_sexo.length;
24
25         // Validação se o sexo foi selecionado
26         opcao_escolhida = null;
27
28         for(i=0; i<total_opcoes; i++)
29         {
30           if(document.forms["meu_form"].campo_sexo[i].checked == true)
31           {
32             opcao_escolhida = document.forms["meu_form"].campo_sexo[i].value;
33           }
34         }
35
36         if(opcao_escolhida == null)

```



```

37     {
38         alert("É necessário selecionar o campo sexo corretamente.");
39         return;
40     }
41
42     // Captura se foi marcado a caixa de maior que 18 anos
43     // Mas neste exemplo nenhuma validação se faz necessária.
44     marcacao_maior = document.forms["meu_form"].campo_maior.checked;
45
46     // Captura o número e o valor da opção selecionada em COMO CONHECEU O SITE
47     conteudo_como = document.forms["meu_form"].campo_como.value;
48     numero_como    = document.forms["meu_form"].campo_como.selectedIndex;
49
50     // Validação se o campo COMO CONHECEU O SITE foi selecionado
51     // diferente da opção "Selecione..."
52     if(numero_como == 0)
53     {
54         alert("É necessário selecionar como conheceu o site.");
55         return;
56     }
57
58     // Se não tiver realizado um return até este momento
59     // envia os dados para o servidor
60     document.forms["meu_form"].submit();
61 }
62 </script>
63
64 <form name="meu_form" method="get" action="?">
65
66     Nome completo:
67     <input name="campo_nome" type="text">
68     <br>
69
70     Sexo:
71     <input name="campo_sexo" type="radio" value="M">Masculino
72     <input name="campo_sexo" type="radio" value="F">Feminino
73     <br>
74
75     <input name="campo_maior" type="checkbox" value="maior">
76     Declaro ser maior de 18 anos
77     <br>
78
79     Como conheceu o site:
80     <select name="campo_como">
81         <option value="Nenhum" default>Selecione...</option>
82         <option value="Ind">Indicação de amigo</option>
83         <option value="Out">Outro site</option>
84         <option value="Goo">Google</option>
85     </select>
86     <br>
87
88     <input type="reset" value="Limpar dados">
89     <input type="button" value="Enviar" onClick="validarFormulario();">
90
91 </form>
92 </body>
93 </html>

```

A validação dos campos do formulário começa basicamente na linha 13, capturando o valor preenchido no campo que representa o nome do usuário e verificando na linha 16, por meio de um comando `if`, se o valor contido no campo é uma string vazia. Se

for, uma caixa de diálogo `alert` exibe uma mensagem para o usuário, informando que ele precisa preencher o campo corretamente e uma instrução `return` existente na linha 19 interrompe a execução do restante da função `validarFormulario`. Seguindo estes mesmos princípios, os demais campos do formulário também são validados.

Supondo que o campo que representa o nome do usuário esteja com um texto preenchido, a instrução `return` da linha 19 não será executada, permitindo a sequência da função de validação. A função continua na linha 23, validando o campo que representa o sexo do usuário.

Uma variável auxiliar chamada `opcao_escolhida` é criada e inicializada com o valor `null`. Se, após o laço de repetição que procura pela opção marcada, a variável `opcao_escolhida` continuar com o valor `null`, isso indicará que nenhuma opção de sexo foi marcada pelo usuário. Neste caso, o comando `if` apresentado na linha 36 irá executar seu bloco de código, apresentando uma mensagem para o usuário selecionar o campo sexo corretamente.

Para o campo de checagem `campo_maior` nenhuma validação foi realizada, pois neste exemplo fictício é permitido que o usuário envie o formulário sendo menor de 18 anos (sem marcar o campo) ou sendo maior de 18 anos (marcando o campo). Se fosse necessário validar este campo, como campos de checagem que verificam se o usuário leu e está de acordo com os termos de uso do site, por exemplo, o seguinte comando `if` poderia ser utilizado (assumindo que o campo de checagem se chamasse `campo_de_checagem`):

```
if(document.forms["meu_form"].campo_de_checagem.checked != true)
{
    alert("É necessário concordar com os termos para prosseguir.");
    return;
}
```

Por último, na linha 52 é validado se o usuário selecionou alguma opção para o campo que representa como o usuário conheceu o site. Se nenhuma opção tiver sido selecionada, a variável `numero_como` conterà o valor `0`, capturado anteriormente na linha 47.

Se os campos tiverem sido preenchidos de acordo com as validações abordadas neste tópico, nenhuma instrução `return` interromperá a execução da função `validarFormulario`, fazendo com que o código chegue e execute o envio do formulário, conforme instrução existente na linha 60.

Claro que neste exemplo, ao chegar nesta linha e enviar o formulário para o servidor, faltam programar ainda as instruções do lado servidor para poder gravar o usuário em um banco de dados, por exemplo. Esta tarefa não será abordada neste e-

book, pois as linguagens de programação do lado servidor são outras tecnologias, assunto para outros e-books ou cursos da Softblue. Mas do ponto de vista de lado cliente, você certamente já está com bons conhecimentos em JavaScript para utilizá-lo em seus projetos!

14. Próximos passos

`Alert("Ufa!")`! Chegamos ao final de mais uma jornada! Conforme proposto por este e-book, você já está capacitado para trabalhar com JavaScript em seus projetos. Logicamente, seus estudos não acabam por aqui. Faz parte pesquisar e continuar estudando o tema de acordo com as necessidades dos seus projetos.

Novamente deixo aqui um grande abraço e meus mais sinceros parabéns por você ter concluído mais esta etapa! Encontro você em breve nos eventos ou cursos da Softblue! Até mais!

Lista de Figuras

Figura 3-1: Atribuindo e exibindo valores em variáveis	11
Figura 3-2: Diferentes tipos de dados impressos no HTML	13
Figura 3-3: Propriedades de strings	15
Figura 3-4: Todos os resultados para a busca do caractere 'a'	17
Figura 4-1: Resultado das operações aritméticas	20
Figura 4-2: Resultado das operações lógicas.....	23
Figura 5-1: Exemplo de uso do comando if	26
Figura 5-2: Exemplo de uso do comando if/else.....	27
Figura 5-3: Exemplo de uso do comando if/else if/else	28
Figura 5-4: Exemplo de uso do comando switch	30
Figura 6-1: Exemplo de uso do comando for	33
Figura 6-2: Exemplo de uso do comando for in	34
Figura 6-3: Exemplo de uso do comando while	36
Figura 6-4: Exemplo de uso do comando do while.....	37
Figura 7-1: Exemplo de função customizada.....	40
Figura 7-2: Exemplo de função customizada sem retorno.....	42
Figura 8-1: Caixa de diálogo de exibição de mensagens	44
Figura 9-1: Caixa de diálogo de tomada de decisão	48
Figura 9-2: Caixa de diálogo de solicitação de informação.....	50
Figura 12-1: Alterando imagem via JavaScript.....	56
Figura 13-1: Formulário HTML	59

Lista de Tabelas

Tabela 4-1: Operadores aritméticos.....	19
Tabela 4-2: Operadores lógicos.....	22
Tabela 8-1: Principais eventos JavaScript.....	44
Tabela 13-1: Principais recursos do objeto form.....	57

Lista de Códigos-Fonte

Código-Fonte 3-1: criandoVariaveis.htm	10
Código-Fonte 3-2: atribuindoValores.htm	11
Código-Fonte 3-3: tiposDeDados.htm	13
Código-Fonte 3-4: strings.htm	15
Código-Fonte 3-5: buscaRecorrente.htm	17
Código-Fonte 4-1: operacoesAritmeticas.htm	20
Código-Fonte 4-2: operacoesLogicas.htm	23
Código-Fonte 5-1: comandoIf.htm	26
Código-Fonte 5-2: comandoIfElse.htm	27
Código-Fonte 5-3: comandoIfElseIfElse.htm	28
Código-Fonte 5-4: comandoSwitch.htm	30
Código-Fonte 6-1: comandoFor.htm	33
Código-Fonte 6-2: comandoForIn.htm	34
Código-Fonte 6-3: comandoWhile.htm	35
Código-Fonte 6-4: comandoDoWhile.htm	37
Código-Fonte 7-1: comandoFunction.htm	40
Código-Fonte 7-2: comandoFunctionSemRetorno.htm	42
Código-Fonte 8-1: comandoAlert.htm	44
Código-Fonte 8-2: exemplosEventos.htm	45
Código-Fonte 9-1: comandoConfirm.htm	48
Código-Fonte 9-2: comandoPrompt.htm	49
Código-Fonte 10-1: propriedadeLocationHref.htm	51
Código-Fonte 10-2: comandoOpen.htm	52
Código-Fonte 11-1: comandoSetTimeout.htm	53
Código-Fonte 12-1: propriedadeImgSrc.htm	55
Código-Fonte 13-1: validandoFormularioBaseHTML.htm	58

Índice Remissivo

- .js, 9
- /*, 8, 25, 29, 32, 34, 35, 36, 39, 58
- //, 8, 10, 20, 22, 23, 39, 64, 65
- <body>, 10, 11, 13, 15, 17, 20, 22, 25, 26, 28, 29, 32, 34, 35, 37, 40, 41, 43, 44, 45, 48, 49, 51, 52, 53, 55, 58, 64
-
, 11, 13, 14, 15, 17, 20, 22, 23, 32, 34, 35, 37, 58, 65
- <script>, 8, 9, 10, 11, 13, 15, 17, 20, 22, 25, 27, 28, 29, 32, 34, 35, 37, 40, 41, 44, 48, 49, 51, 52, 53, 55, 58, 64
- action, 57
- alert, 44, 45, 46, 47, 48, 49, 50, 53, 60, 61, 62, 63, 64, 65, 66
- argumentos, 39
- array, 12, 14, 34, 60, 61, 64
- booleano, 48
- break, 29, 30
- case, 29, 30, 31
- checked, 62, 63, 64, 65, 66
- confirm, 47, 48, 49
- default, 29, 30, 58, 65
- do while, 36, 38
- document, 11, 13, 15, 17, 20, 22, 23, 25, 27, 28, 30, 32, 34, 35, 37, 40, 42, 55, 56, 60, 61, 62, 63, 64, 65, 66
- DOM Document, 11
- else, 26, 27, 28, 41, 48
- eventos, 43, 44, 45, 46, 68
- false, 12, 13, 14, 21, 22, 23, 24, 25, 26, 27, 33, 37, 48, 63
- for, 29, 32, 33, 34, 35, 36, 37, 38, 63, 64
- for in, 34
- form, 57, 58, 59, 60, 65
- forms, 60, 61, 62, 63, 64, 65, 66
- function, 39, 40, 41, 42, 51, 52, 53, 55, 58, 60, 61, 62, 63, 64
- getElementById, 55, 56
- href, 45, 51
- if, 25, 26, 27, 28, 29, 41, 48, 63, 64, 65, 66
- indexOf, 15, 16, 17, 18
- length, 15, 16, 63, 64
- location, 51
- lógica de programação, 6
- method, 57, 58, 65
- name, 57, 58, 59, 63, 65
- null, 63, 64, 66
- numérico, 11, 12, 14, 15, 16, 24, 61
- objeto, 11, 14, 43, 44, 51, 52, 55, 57, 60
- onBlur, 43
- onChange, 43
- onClick, 43, 45, 46, 51, 52, 55, 56, 58, 65
- onFocus, 43, 45
- onKeyDown, 44
- onKeyPress, 43, 45
- onKeyUp, 44
- onLoad, 43, 45
- onMouseDown, 43
- onMouseOut, 43
- onMouseOver, 43, 45, 46
- onMouseUp, 43
- onSubmit, 44
- onUnload, 43
- open, 52
- operadores aritméticos, 19
- operadores lógicos, 19, 21, 22
- options, 61, 62
- parâmetros, 18, 35, 39, 40, 41, 42, 53
- prompt, 49, 50
- return, 39, 40, 41, 64, 65, 66
- selectedIndex, 61, 65
- setTimeout, 53
- submit, 44, 57, 59, 65
- substring, 15, 16
- switch, 29, 30
- text, 58, 61, 62, 65
- true, 12, 13, 14, 21, 22, 23, 24, 25, 32, 48, 63, 64, 66
- value, 45, 51, 52, 55, 58, 60, 61, 62, 63, 64, 65
- var, 10, 11, 13, 15, 17, 20, 22, 25, 27, 28, 29, 34, 35, 37, 40, 41, 55
- variáveis, 10, 11, 12, 13, 14, 19, 20, 32, 35, 36, 40, 41

while, 17, 35, 36, 37, 38

window, 52