

JavaScript

Ana G. Caesar
778.6008.8769

COD.: TE 1545/2_WEB



JavaScript

Ana G. Caetano
778.6008.8779



IMPACTA
EDITORA

Créditos

Copyright © TechnoEdition Editora Ltda.

Todos os direitos autorais reservados. Este manual não pode ser copiado, fotocopiado, reproduzido, traduzido ou convertido em qualquer forma eletrônica, ou legível por qualquer meio, em parte ou no todo, sem a aprovação prévia, por escrito, da TechnoEdition Editora Ltda., estando o contrafator sujeito a responder por crime de Violação de Direito Autoral, conforme o art.184 do Código Penal Brasileiro, além de responder por Perdas e Danos. Todos os logotipos e marcas utilizados neste material pertencem às suas respectivas empresas.

"As marcas registradas e os nomes comerciais citados nesta obra, mesmo que não sejam assim identificados, pertencem aos seus respectivos proprietários nos termos das leis, convenções e diretrizes nacionais e internacionais."

JavaScript

Coordenação Geral

Marcia M. Rosa

Coordenação Editorial

Henrique Thomaz Bruscagin

Supervisão Editorial

Simone Rocha Araújo Pereira

Redação

André Ricardo Stern

Cristiana Hoffmann Pavan

Maria Rosa Carnicelli Kushnir

Rafael Farinaccio

Elaboração de Exemplos,

Exercícios e Laboratórios

João Henrique Cunha Rangel

Revisão Ortográfica e

Gramatical de Exemplos,

Exercícios e Laboratórios

Rafael Farinaccio

Diagramação

Shelida Letícia Lopes

Equipe de Apoio

Rodrigo Henrique Sobrinho

Edição nº 1 | Cód.: 1545/2_WEB

Novembro/2012



IMPACTA
EDITOR A

Sumário

Informações sobre o treinamento 13

Capítulo 1 - Introdução 15

1.1.	A linguagem JavaScript	16
1.1.1.	JavaScript e HTML	17
1.2.	Um pouco de história	17
1.3.	As versões da JavaScript	18
1.4.	Camadas de desenvolvimento	20
1.5.	Plugins e Frameworks.....	20
1.6.	Consoles para depuração na JavaScript.....	21
1.7.	Criando scripts.....	22
1.7.1.	Definindo a linguagem usada no código.....	24
1.8.	Primeiros códigos em JavaScript.....	24
1.8.1.	Código de ocultamento	24
1.8.2.	Códigos de comentários.....	26
1.9.	Inserindo um arquivo com extensão .js em HTML.....	27
1.10.	Orientação a objetos	28
	Pontos principais	29

Teste seus conhecimentos 31

Capítulo 2 - Variáveis 35

2.1.	O que é uma variável?	36
2.2.	Criando variáveis	36
2.2.1.	Declarando variáveis	37
2.2.2.	Variáveis locais	38
2.2.3.	Variáveis globais	38
2.2.4.	Aritmética com a JavaScript	38
2.3.	Palavras reservadas	39
2.4.	Tipos de variáveis	41
2.4.1.	Convertendo strings em números.....	46
2.5.	Caracteres especiais	49
2.6.	Concatenação.....	51
2.7.	Constantes	52
2.8.	Objetos globais	52
2.8.1.	Variáveis e propriedades dos objetos	53
2.8.2.	null	53
2.8.3.	undefined	54
2.8.4.	NaN.....	54

HTML5

2.8.5. Infinity	54
Pontos principais	55
Teste seus conhecimentos.....	57
Mãos à obra	61
Capítulo 3 - Operadores	63
3.1. Utilizando operadores em JavaScript.....	64
3.2. Expressões.....	64
3.3. Tipos de operadores	65
3.4. Operadores de atribuição	65
3.5. Operadores de comparação.....	67
3.6. Operadores aritméticos	69
3.7. Operadores bitwise	70
3.7.1. Operadores bitwise lógicos	70
3.7.2. Operadores bitwise de deslocamento	72
3.8. Operadores lógicos	73
3.8.1. Avaliação de curto-círcuito	75
3.9. Operadores de string	76
3.10. Operadores especiais	76
3.10.1. Operador condicional.....	77
3.10.2. Operador separador	78
3.10.3. delete.....	78
3.10.4. in	80
3.10.5. instanceof	80
3.10.6. new	81
3.10.7. this	82
3.10.8. typeof	82
3.10.9. void	83
3.11. Precedência dos operadores	84
Pontos principais	86
Teste seus conhecimentos.....	87
Mãos à obra	91
Capítulo 4 - Declarações	95
4.1. Introdução	96
4.1.1. Declarações em JavaScript	96
4.2. Estruturas condicionais	97
4.2.1. Declaração if/else	98

Sumário

4.2.2.	Declaração switch/case	102
4.3.	Estruturas para loops	106
4.3.1.	Declaração while	107
4.3.2.	Declaração do/while	108
4.3.3.	Declaração for	109
4.3.4.	Declaração for/in	115
4.3.5.	Declaração break	116
4.3.6.	Declaração continue	117
	Pontos principais	119
	Teste seus conhecimentos	121
	Mãos à obra	125

	Capítulo 5 - Funções	129
5.1.	O que é uma função?	130
5.2.	Definindo uma função	130
5.2.1.	Inserindo funções	131
5.3.	Chamando funções	133
5.4.	Escopo de uma função	134
5.5.	Closures	136
5.6.	Inserindo variáveis nos parâmetros	137
5.6.1.	Um parâmetro	138
5.6.2.	Múltiplos parâmetros	139
5.6.3.	Parâmetros passados por valores	140
5.7.	Retornando valores	140
5.8.	Funções predefinidas	141
5.9.	Propriedades das funções	146
5.9.1.	arguments.length	146
5.9.2.	arguments.callee	146
5.9.3.	length	147
5.9.4.	constructor	148
5.9.5.	prototype	149
5.10.	Métodos para funções	150
	Pontos principais	153
	Teste seus conhecimentos	155
	Mãos à obra	159

HTML5

Capítulo 6 - Eventos	161
6.1. Introdução	162
6.2. Definindo um evento.....	162
6.3. Manipuladores de eventos.....	163
6.3.1. Eventos do mouse	164
6.3.2. Eventos do teclado	170
6.3.3. Eventos de frame/objeto	172
6.3.4. Eventos de formulário	178
6.4. Objeto Event	183
6.4.1. Objeto Event no DOM	184
6.4.1.1. Propriedades do objeto Event no DOM	185
6.4.1.2. Métodos do objeto Event.....	192
6.4.2. Objeto Event no Internet Explorer.....	194
6.4.2.1. Propriedades do objeto Event no Internet Explorer	195
Pontos principais	199
Teste seus conhecimentos.....	201
Mãos à obra	205

Capítulo 7 - Objetos	209
7.1. Introdução	210
7.1.1. Categorias de objeto	210
7.2. Conhecendo a Programação Orientada a Objetos (POO).....	211
7.3. Criando objetos.....	216
7.4. As propriedades	216
7.5. Os métodos	217
7.6. Objetos nativos	218
7.6.1. Objeto Array	219
7.6.1.1. Criando objetos Array	219
7.6.1.2. Propriedades do objeto Array	221
7.6.1.3. Métodos do objeto Array	224
7.6.2. Objeto Boolean.....	231
7.6.3. Objeto Date.....	232
7.6.3.1. Propriedades do objeto Date	238
7.6.3.2. Métodos do objeto Date	239
7.6.4. Objeto Math	242
7.6.4.1. Propriedades do objeto Math.....	243
7.6.4.2. Métodos do objeto Math	244

Sumário

7.6.5.	Objeto Number	246
7.6.5.1.	Propriedades do objeto Number	247
7.6.5.2.	Métodos do objeto Number	248
7.6.6.	Objeto String.....	249
7.6.6.1.	Propriedades do objeto String	249
7.6.6.2.	Métodos do objeto String	252
7.6.7.	Objeto RegExp	253
7.6.7.1.	Modificadores RegExp	254
7.6.7.2.	Agrupadores	254
7.6.7.3.	Metacaracteres	256
7.6.7.4.	Quantificadores.....	257
7.6.7.5.	Propriedades do objeto RegExp.....	259
7.6.7.6.	Métodos do objeto RegExp.....	262
7.6.7.7.	O objeto RegExp na prática	264
7.7.	BOM (Browser Object Model)	269
7.7.1.	Objeto Window.....	269
7.7.1.1.	Propriedades do objeto Window	269
7.7.1.2.	Métodos do objeto Window	272
7.7.2.	Objeto Navigator	275
7.7.2.1.	Propriedades do objeto Navigator	275
7.7.2.2.	Métodos do objeto Navigator	276
7.7.3.	Objeto Screen.....	277
7.7.3.1.	Propriedades do objeto Screen	277
7.7.4.	Objeto History.....	279
7.7.4.1.	Propriedade do objeto History	280
7.7.4.2.	Métodos do objeto History	280
7.7.5.	Objeto Location.....	281
7.7.5.1.	Propriedades do objeto Location	282
7.7.5.2.	Métodos do objeto Location	283
7.8.	JSON	284
7.8.1.	Tipos básicos de JSON	285
7.8.2.	Função JSON.parse	286
7.8.3.	Função JSON.stringify	288
7.8.3.1.	Ordem de execução	292
	Pontos principais	293
	Teste seus conhecimentos.....	295
	Mãos à obra	299

Capítulo 8 - DOM	303
8.1. Introdução	304
8.2. DOM HTML.....	304
8.2.1. Objeto Document	306
8.2.1.1. Propriedades do objeto Document	307
8.2.1.2. Métodos do objeto Document	308
8.2.2. Objeto Event	309
8.2.3. Objeto HTMLElement.....	309
8.2.3.1. Propriedades de HTMLElement	309
8.2.3.2. Método de HTMLElement.....	312
8.2.4. Objeto Anchor.....	312
8.2.4.1. Propriedades do objeto Anchor	313
8.2.5. Objeto Area.....	314
8.2.5.1. Propriedades do objeto Area	315
8.2.6. Objeto Base.....	316
8.2.6.1. Propriedades do objeto Base	316
8.2.7. Objeto Body	317
8.2.7.1. Propriedades do objeto Body	317
8.2.7.2. Eventos do objeto Body	318
8.2.8. Objeto Button.....	319
8.2.8.1. Propriedades do objeto Button	320
8.2.9. Objeto Form.....	320
8.2.9.1. Coleção do objeto Form	321
8.2.9.2. Propriedades do objeto Form	322
8.2.9.3. Métodos do objeto Form	322
8.2.9.4. Eventos do objeto Form.....	323
8.2.10. Objetos Frame e IFrame	324
8.2.10.1. Propriedades dos objetos Frame e IFrame	324
8.2.10.2. Evento dos objetos Frame e IFrame	327
8.2.11. Objetos Input	327
8.2.11.1. Objeto Button.....	327
8.2.11.2. Objeto Checkbox	328
8.2.11.3. Objeto FileUpload.....	330
8.2.11.4. Objeto Hidden.....	332
8.2.11.5. Objeto Password	333
8.2.11.6. Objeto Radio	335
8.2.11.7. Objeto Reset	337
8.2.11.8. Objeto Submit	338

Sumário

8.2.11.9. Objeto Text	339
8.2.12. Objeto Link	341
8.2.12.1. Propriedades do objeto Link	342
8.2.13. Objeto Meta	342
8.2.13.1. Propriedades do objeto Meta	343
8.2.14. Objeto Object	344
8.2.14.1. Propriedades do objeto Object	344
8.2.15. Objeto Option	345
8.2.15.1. Propriedades do objeto Option	345
8.2.16. Objeto Select	346
8.2.16.1. Coleção do objeto Select	346
8.2.16.2. Propriedades do objeto Select	347
8.2.16.3. Métodos do objeto Select	347
8.2.17. Objeto Style	349
8.2.18. Objeto Table	350
8.2.18.1. Coleções do objeto Table	350
8.2.18.2. Propriedades do objeto Table	350
8.2.18.3. Métodos do objeto Table	351
8.2.19. Objeto Textarea	352
8.2.19.1. Propriedades do objeto Textarea	352
8.2.19.2. Métodos do objeto Textarea	353
8.3. DOM Core	353
8.3.1. Objeto Node	355
8.3.1.1. Propriedades do objeto Node	356
8.3.1.2. Métodos do objeto Node	358
8.3.2. Objeto NodeList	359
8.3.2.1. Propriedade do objeto NodeList	360
8.3.2.2. Método do objeto NodeList	360
8.3.3. Objeto NamedNodeMap	360
8.3.3.1. Propriedades do objeto NamedNodeMap	361
8.3.3.2. Métodos do objeto NamedNodeMap	361
8.3.4. Objeto Document	362
8.3.4.1. Propriedades do objeto Document	362
8.3.4.2. Métodos do objeto Document	363
8.3.5. Objeto Element	364
8.3.5.1. Propriedades do objeto Element	365
8.3.5.2. Métodos do objeto Element	365
8.3.6. Objeto Attr	366

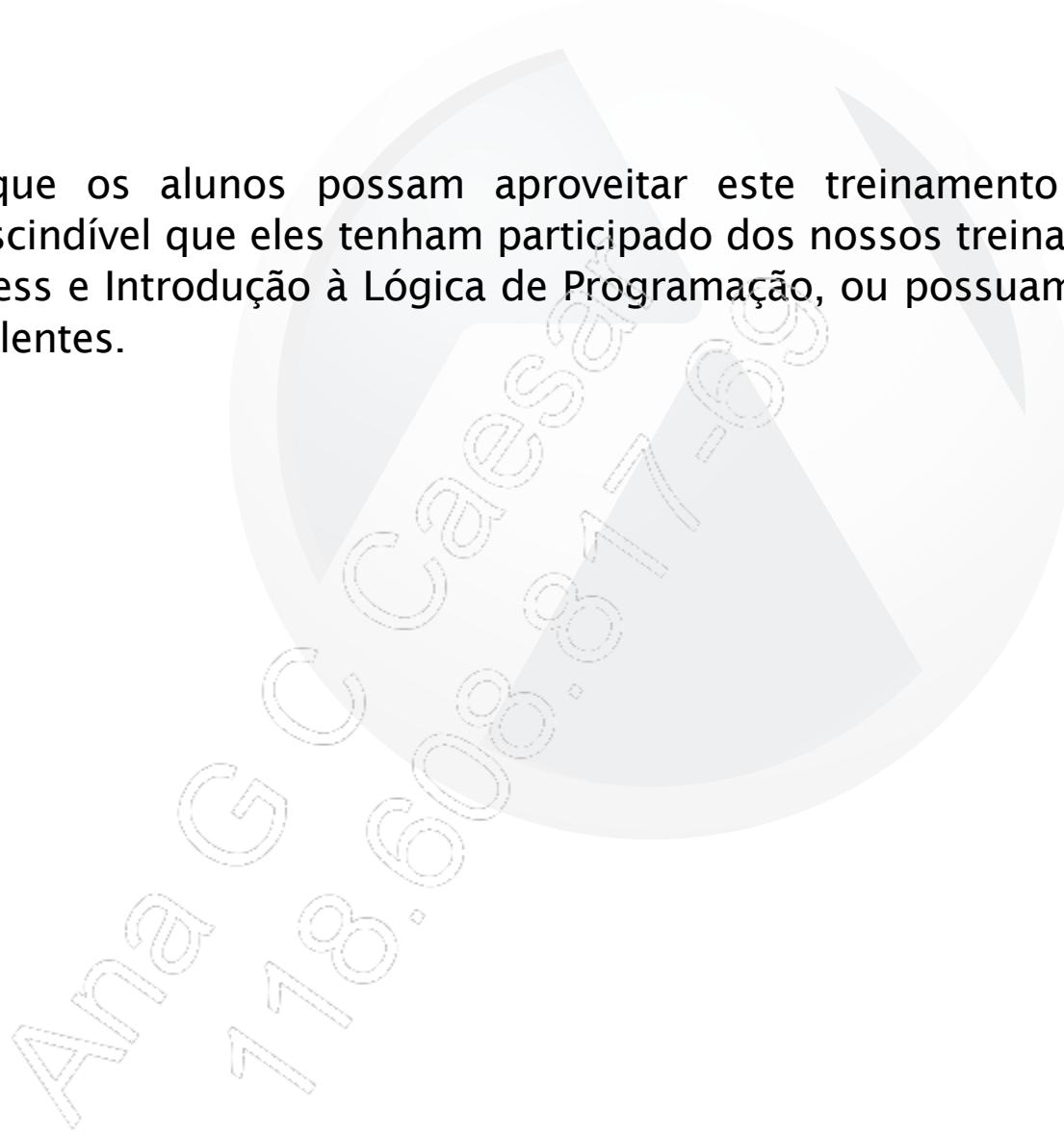
HTML5

8.3.6.1.	Propriedades do objeto Attr	367
8.4.	DOM Storage.....	367
8.4.1.	Local Storage	368
8.4.2.	A interface Storage.....	368
8.4.3.	Eventos DOM Storage	370
	Pontos principais	371
	Teste seus conhecimentos.....	373
	Mãos à obra	377

	Capítulo 9 - AJAX.....	381
9.1.	Introdução	382
9.2.	O objeto XMLHttpRequest	383
9.2.1.	Criando o objeto	384
9.2.2.	Enviando uma requisição para o servidor.....	384
9.2.2.1.	Método open.....	385
9.2.2.2.	Método send	386
9.2.2.3.	Método setRequestHeader.....	388
9.2.2.4.	Ação disparadora de evento onreadystatechange	390
9.2.3.	Respostas do servidor	392
9.2.3.1.	Propriedade responseText.....	392
9.2.3.2.	Propriedade responseXML.....	393
9.3.	Exemplos de uso	394
9.3.1.	Solicitando HTML	395
9.3.2.	Solicitando JSON.....	396
9.3.3.	Solicitando JavaScript	397
	Pontos principais	399
	Teste seus conhecimentos.....	401
	Mãos à obra	405

Informações sobre o treinamento

Para que os alunos possam aproveitar este treinamento ao máximo, é imprescindível que eles tenham participado dos nossos treinamentos XHTML - Tableless e Introdução à Lógica de Programação, ou possuam conhecimentos equivalentes.



Introdução

1

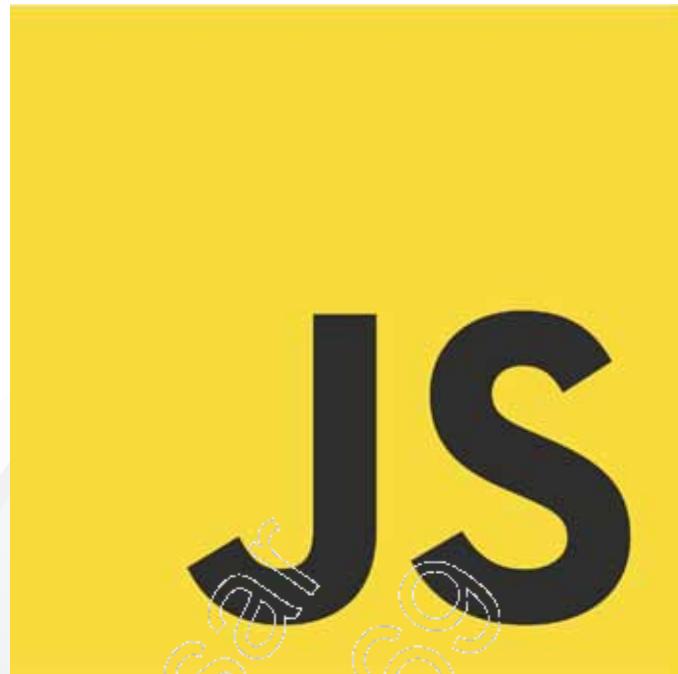
- ✓ A linguagem JavaScript;
- ✓ Versões da JavaScript;
- ✓ Camadas de desenvolvimento;
- ✓ Plugins e Frameworks;
- ✓ Consoles para depuração na JavaScript;
- ✓ Criação de scripts;
- ✓ Código de ocultamento e de comentário;
- ✓ Inserção de arquivos com extensão .js em HTML;
- ✓ Orientação a objetos.



IMPACTA
EDITORA

1.1. A linguagem JavaScript

A JavaScript é uma linguagem de script orientada a objeto que pode ser executada em várias plataformas.



1.1. Logotipo da plataforma JavaScript (a partir de 2011)

Essa linguagem é leve, concisa e de fácil integração com outras aplicações, mas não é uma boa opção utilizá-la como linguagem independente. É possível também conectar a linguagem JavaScript a outros objetos existentes dentro de um ambiente para poder controlá-los de forma programática.

Você encontra um conjunto básico tanto de objetos quanto de elementos de linguagem na JavaScript, mas também pode adicionar objetos. No lado do cliente, é possível estender a linguagem básica ao disponibilizar objetos para controlar um navegador e seu DOM. Assim, elementos podem ser inseridos para executar determinadas ações, por exemplo. Já do lado do servidor, a extensão da linguagem se dá por meio de objetos relevantes para a execução de JavaScript no servidor. Essa extensão pode, por exemplo, viabilizar a comunicação entre uma aplicação e um banco de dados.

É importante saber que Java e JavaScript não são a mesma coisa! Cada linguagem tem um conceito e um design diferente. A linguagem Java é bem mais complexa e está no mesmo patamar que outras linguagens de programação como C. Além disso, o Java foi desenvolvido pela Sun Microsystems, enquanto o JavaScript foi desenvolvido por Brendan Eich da Netscape.

Luma curiosidade: O nome original do JavaScript era Mocha. Depois a linguagem foi renomeada e passou a se chamar LiveScript. Por fim, acabou mudando para JavaScript devido à grande influência da linguagem Java sobre ela. Além da mudança do nome, a Netscape passou a oferecer suporte ao Java em seu navegador, o que acabou causando uma confusão, pois parecia que a linguagem JavaScript era apenas um spin-off do Java.

1.1.1. JavaScript e HTML

A HTML é uma linguagem de marcação, o que quer dizer que ela tem a função de estruturar as páginas web. Os elementos que compõem a página não podem ser atribuídos a partir de uma linguagem de marcação. Mesmo a versão atual da HTML (atual HTML 4.01) não oferece meios para implementar algum tipo de interatividade mais avançada. Para tarefas como essa, é necessário utilizar uma linguagem de programação, como JavaScript.

1.2. Um pouco de história

Quando foi lançada nas versões beta do Netscape Navigator 2.0, a linguagem JavaScript se chamava LiveScript. Isso foi em setembro de 1995. Logo em dezembro do mesmo ano, seu nome foi alterado para JavaScript em um pronunciamento conjunto com a Sun Microsystems. Nessa ocasião, a linguagem passou a ser distribuída na versão 2.0B3 do Netscape.

JavaScript

Como a JavaScript rapidamente se tornou popular entre os usuários, a Microsoft teve que nomear sua implementação JScript, para evitar problemas relacionados aos direitos sobre a marca. Além disso, foram adicionados à JScript novos métodos de data para resolver os problemas com os métodos Y2K existentes em JavaScript. A JScript foi incluída no Internet Explorer 3.0 (lançado em meados de 1996).

A Netscape submeteu a linguagem JavaScript à ECMA International em novembro de 1996, para que fosse definida como padrão na indústria. Isso acabou resultando na padronização da versão chamada ECMAScript.

Hoje em dia, a JavaScript é uma das linguagens de script na Internet mais conhecidas. Você pode trabalhar com JavaScript nos navegadores mais populares (Chrome, Firefox, Internet Explorer, Opera e Safari). No entanto, essa linguagem foi muito criticada no início por programadores profissionais, principalmente por também poder ser utilizada por amadores. O que fez com que ela conquistasse os programadores profissionais foi o advento do AJAX (Asynchronous Javascript and XML). A partir daí, as práticas de programação via JavaScript foram otimizadas e houve um considerável crescimento de bibliotecas e Frameworks, bem como do uso de JavaScript fora de navegadores web, já que as plataformas JavaScript do lado do servidor também apresentaram aumento.

Por fim, foi fundado, em janeiro de 2009, um projeto chamado CommonJS, com o intuito de definir uma biblioteca padrão voltada para o desenvolvimento de JavaScript fora de navegadores.

1.3. As versões da JavaScript

O nome oficial da JavaScript é ECMAScript e a linguagem está na versão ECMA-262 v5, mas, desde a primeira versão, chamada JavaScript 1.0, até hoje, existiram várias versões da linguagem. Dê uma olhada na tabela a seguir:

Versão	Implementação	Data
JavaScript 1.0	Netscape 2.0	03/1996
JavaScript 1.1	Netscape 3.0	08/1996
JavaScript 1.2	Netscape 4.0 e 4.05	06/1997
JavaScript 1.3	Netscape 4.06 e 4.07x	10/1998

Versão	Implementação	Data
JavaScript 1.4	Servidores Netscape	
JavaScript 1.5	Netscape 6.0 – Firefox 1.0 – Opera 6.0 a 9.0	11/2000
JavaScript 1.6	Firefox 1.5 – Safari 3.0 e 3.1	11/2005
JavaScript 1.7	Firefox 2.0 – Safari 3.2 e 4.0 – Chrome 1.0	10/2006
JavaScript 1.8	Firefox 3.0	06/2008
JavaScript 1.8.1	Firefox 3.5	2008
JavaScript 1.9	Firefox 4.0	2009
JavaScript 1.0	Internet Explorer 3	08/1996
JavaScript 2.0	Internet Explorer 3 – Windows IIS 3	01/1997
JavaScript 3.0	Internet Explorer 4	10/1997
JavaScript 4.0	Visual Studio 6.0	
JavaScript 5.0	Internet Explorer 5	03/1999
JavaScript 5.1	Internet Explorer 5.01	
JavaScript 5.5	Internet Explorer 5.5	07/2000
JavaScript 5.6	Internet Explorer 6	10/2001
JavaScript 5.7	Internet Explorer 7	11/2006
JavaScript 5.8	Internet Explorer 8	03/2009
ECMA-262 v1	Navegadores versão 4	1998
ECMA-262 v2	Versão de testes	1998
ECMA-262 v3	Navegadores versão 6	1999
ECMA-262 v4	Navegadores versão 6+	2002
ECMA-262 v5	Versão Atual	2009

1.4. Camadas de desenvolvimento

As aplicações de Web hoje são geralmente construídas respeitando o conceito de desenvolvimento em camadas, que nada mais é do que a separação dos códigos de desenvolvimento em três camadas. Basicamente, se quiser seguir o conceito do desenvolvimento em camadas, você vai escrever os códigos de cada camada em arquivos separados e fazer a conexão deles através de links. Com isso, fica mais fácil reaproveitar pedaços de códigos em projetos novos, você vai ter menos retrabalho e os códigos ficarão mais fáceis de entender, corrigir e administrar. As três camadas de desenvolvimento são:

- Camada de estruturação de conteúdos, que consiste na marcação HTML;
- Camada de apresentação, representada pelas folhas de estilos;
- Camada de comportamento, onde entram os scripts que definem o comportamento das aplicações. É aqui que entra o JavaScript e a interatividade com o usuário é inserida.

1.5. Plugins e Frameworks

Frameworks são coleções de bibliotecas de software, que incluem programas de suporte, compiladores, bibliotecas de códigos e todo um conjunto de ferramentas que formam uma interface de programação de aplicativos (API). Com os Frameworks, você consegue desenvolver aplicações, produtos e soluções.

Para ter uma ideia melhor dos tipos de Frameworks e plugins que você pode utilizar com JavaScript, veja, a seguir, uma lista com alguns deles:

- **JQuery**: Bem documentado e simples, é um dos Frameworks mais populares, que facilita a criação de códigos limpos e organizados. Seu site oficial é <http://jquery.com>;
- **Prototype**: Opção de muitas empresas internacionalmente reconhecidas, esse Framework é uma alternativa um pouco mais profissional, porém muita gente reclama do seu peso e da lentidão dos sites que usam o Prototype. Seu endereço oficial é <http://www.prototypejs.org/>;

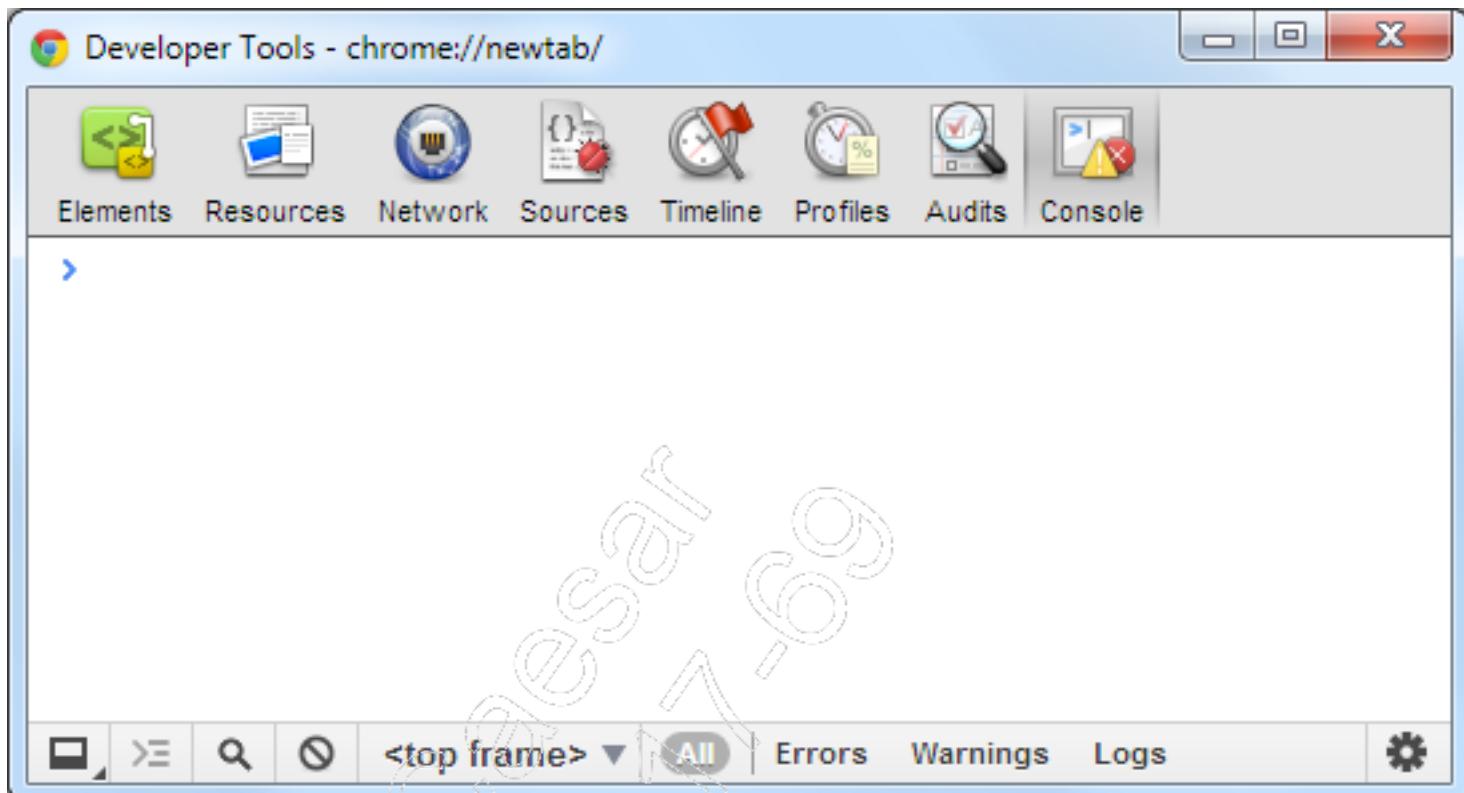
- **Mootools:** Um Framework rápido e simples, que permite escolher as partes do Framework que serão inclusas, para que os scripts sejam carregados com mais agilidade. O site é <http://mootools.net/>;
- **YUI:** Esse Framework é usado pelos desenvolvedores do Yahoo! E disponibilizado para uso. Bastante eficiente e com muitos controles e componentes, mas um pouco pesado. O site é <http://developer.yahoo.com/yui/>;
- **Backbone:** Ao separar as características comuns entre as aplicações Web em JavaScript, o Framework Backbone busca resolver a falta de estrutura das aplicações. Seu site é <http://backbonejs.org/>.

1.6. Consoles para depuração na JavaScript

Como você já deve saber, a depuração é o processo através do qual você busca e conserta erros num software ou hardware eletrônico, antes de eles acontecerem, para gerar um comportamento tão próximo do esperado quanto possível na hora do uso.

Diversos consoles de depuração podem ser utilizados para encontrar problemas em trechos de código programado em JavaScript. Todos os navegadores mais populares, como Internet Explorer, Firefox, Safari, Chrome e Opera, possuem ferramentas para depuração de códigos JavaScript. Além deles, existem outros consoles muito utilizados, como o Firebug, a Web Developer's Toolbar, etc.

O navegador Google Chrome disponibiliza um bom console para depuração de códigos JavaScript. O Developer Tools, mostrado na próxima imagem, é baseado no Webkit Web Inspector, que faz parte do projeto open source WebKit, e permite que você acesse níveis profundos do navegador e das aplicações, através de uma caixa de busca e de uma barra de ferramentas com ícones e painéis voltados para tipos específicos de informações de aplicações e páginas.



1.2. Console para depuração do navegador Google Chrome.

1.7. Criando scripts

Podemos definir um script como uma linguagem que será interpretada pelo navegador. Para criar um script, basta um editor de texto puro e um navegador. Um exemplo de editor de texto é o bloco de notas, disponível em qualquer sistema operacional. Com relação ao navegador, podemos usar qualquer navegador disponível no mercado, já que todos suportam a linguagem JavaScript atualmente.

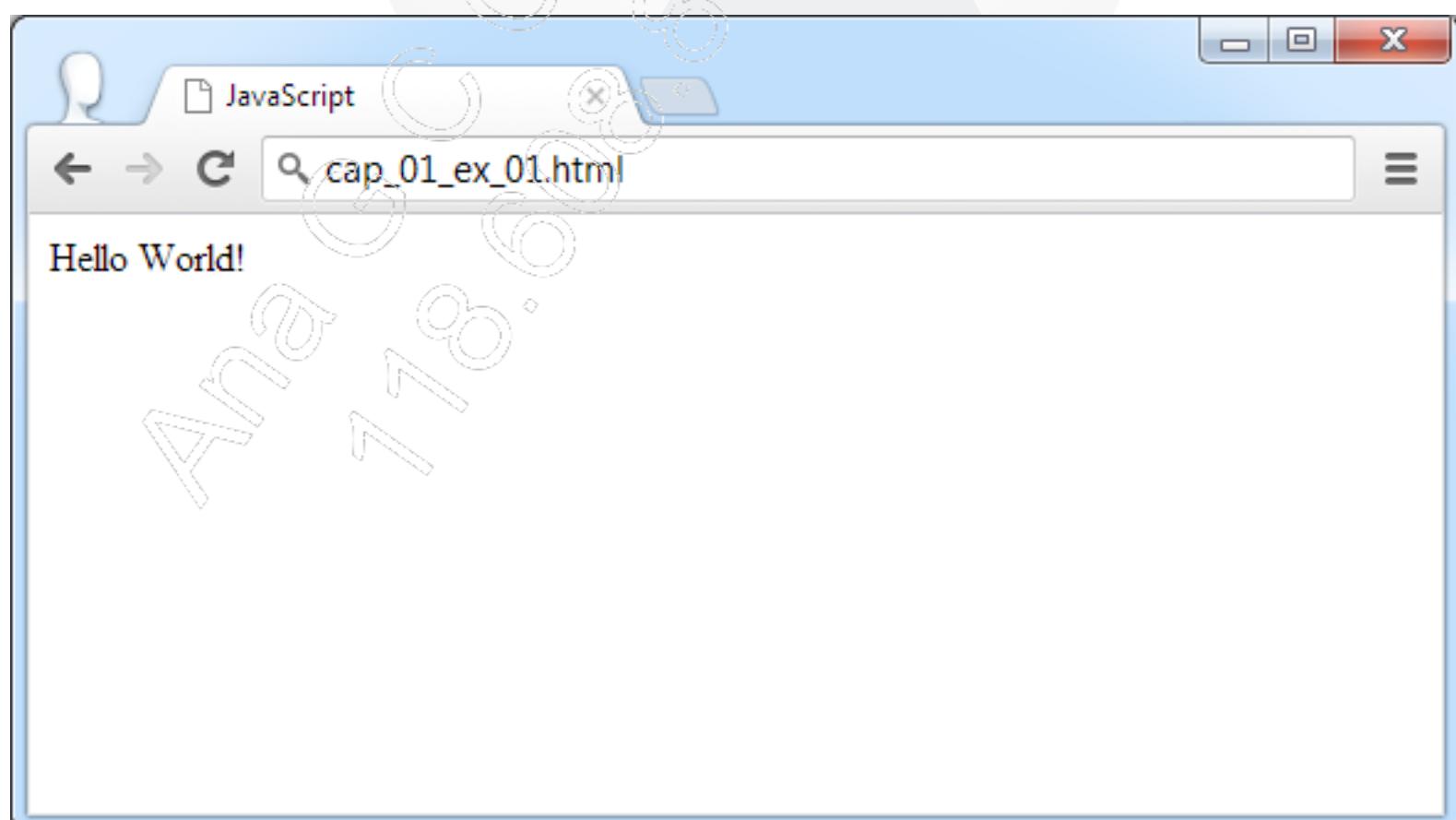
Os códigos JavaScript, em documentos HTML, funcionam sob a forma de funções/métodos, que podem estar presentes em qualquer trecho do código e são chamados em resposta a determinados eventos ou em situações específicas. Essas funções são iniciadas com a declaração `<script>` e finalizadas com `</script>`.

Para que o JavaScript seja carregado antes do **<body>**, ou seja, antes de o usuário interagir com a página Web, as funções são normalmente colocadas entre as tags **<head>** e **</ head>**, no início do documento.

No próximo exemplo, você vê o uso das tags **<script>** e **</ script>**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 document.write("Hello World!");
8 </script>
9 </head>
10
11 <body>
12
13 </body>
14 </html>
```

O código anterior, que deve ser salvo como **.html** ou **.htm**, produzirá o seguinte resultado:



1.7.1. Definindo a linguagem usada no código

Para seguir o padrão definido pela W3C (World Wide Web Consortium), é fundamental apontar a linguagem utilizada - neste caso, JavaScript - por meio do parâmetro **script type**, como apresentado adiante:

```
<script>  
...  
</script>
```

1.8. Primeiros códigos em JavaScript

Nos subtópicos adiante, você vai ver dois tipos de código muito utilizados em JavaScript: o código de ocultamento e os códigos de comentários.

1.8.1. Código de ocultamento

Esse tipo de código deve ser empregado só quando você estiver usando navegadores antigos que não suportam JavaScript, ou seja, que não reconhecem e ignoram as TAGs `<script>` e `</script>`, inclusive os códigos que se encontram limitados por elas.

Para evitar esse transtorno, que acaba fazendo com que a tela seja exibida como se fosse um texto HTML, você pode usar os códigos de ocultamento `<!--`, no início, e `-->`, no final, para ocultar os códigos JavaScript dos navegadores que não reconhecem tal linguagem.

No exemplo a seguir, veja a ocultação da linha de código em JavaScript:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 <!--
8 document.write("Hello World!");
9 //-->
10 </script>
11 </head>
12
13 <body>
14
15 </body>
16 </html>
```

Lembre que o procedimento de ocultamento não esconde o código JavaScript por completo, só garante a exibição dele por parte dos navegadores que não suportam a linguagem. Mas, por meio da opção **Exibir Código Fonte** (CTRL + U), você pode acessar o código-fonte da página em HTML ou JavaScript, como apresentado a seguir:



1.8.2. Códigos de comentários

Os códigos de comentários são linhas que não são lidas pelo navegador, usadas quando você quiser inserir observações em trechos de código, para facilitar o reconhecimento e a identificação de tarefas por parte do programador. São basicamente uma ou mais linhas, dentro de um código, que servem apenas como comentários para esclarecimentos gerais.

Em um código, você pode definir comentários de uma linha ou de múltiplas linhas. No primeiro caso, o comentário é representado pelos caracteres `//`. Já para múltiplas linhas, o comentário é iniciado com `/*` e finalizado com `*/`.

Por não exigir um limite para a quantidade de linhas, nos comentários de múltiplas linhas, você pode usar o comentário para outras finalidades, como para isolar trechos de códigos que não serão lidos, ajudar na identificação de erros em códigos amplos, entre outros fins.

Veja um exemplo de código de comentário de uma linha:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 // Este é um comentário de uma linha
8 document.write("Hello World!");
9 </script>
10 </head>
11 <body>
12 </body>
13 </body>
14 </html>
```

E agora um exemplo de código de comentário de múltiplas linhas:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 /*
8     Este é um comentário de
9     múltiplas linhas
10 */
11 document.write("Hello World!");
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

1.9. Inserindo um arquivo com extensão .js em HTML

A partir do bloco de notas, é possível escrever os códigos JavaScript de forma separada, em um arquivo de extensão .js, e depois incluir esse arquivo no documento HTML. Mas, nesse caso, as tags `<script>` e `</script>` não devem ser incluídas.

O seguinte código deve ser definido para inserir um arquivo .js em um documento HTML:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script src="cap_01_ex_05.js"></script>
7 </head>
8
9 <body>
10
11 </body>
12 </html>
```

O nome do arquivo separado que contém o script em JavaScript é especificado em “**cap_01_ex_05.js**”.

1.10. Orientação a objetos

A JavaScript possui suporte a vários aspectos da programação orientada a objetos, como você verá ao longo deste treinamento. Por enquanto, basta saber que os objetos podem ser internos da linguagem, do ambiente de hospedagem ou personalizados, ou seja, criados por quem desenvolve a aplicação, e que eles possuem propriedades e métodos, que você pode acessar com estas sintaxes:

```
propriedade = Objeto.propriedade;
```

```
Objeto.método();
```

Uma dica que você pode querer seguir é escrever os nomes dos objetos sempre com a inicial maiúscula, para facilitar a busca no código.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- A JavaScript é uma linguagem de script orientada a objeto que pode ser executada em várias plataformas. Essa linguagem é leve, concisa e de fácil integração com outras aplicações, mas não é uma boa opção utilizá-la como linguagem independente;
- As aplicações de Web hoje são geralmente construídas respeitando o conceito de desenvolvimento em camadas, que nada mais é do que a separação dos códigos de desenvolvimento em três camadas. Basicamente, se quiser seguir o conceito do desenvolvimento em camadas, você vai escrever os códigos de cada camada em arquivos separados e fazer a conexão deles através de links. Com isso, fica mais fácil reaproveitar pedaços de códigos em projetos novos, você vai ter menos retrabalho e os códigos ficarão mais fáceis de entender, corrigir e administrar;
- Frameworks são coleções de bibliotecas de software, que incluem programas de suporte, compiladores, bibliotecas de códigos e todo um conjunto de ferramentas que formam uma interface de programação de aplicativos (API). Com os Frameworks, você consegue desenvolver aplicações, produtos e soluções;
- Depuração é o processo através do qual você busca e conserta erros num software ou hardware eletrônico, antes de eles acontecerem, para gerar um comportamento tão próximo do esperado quanto possível na hora do uso. Diversos consoles de depuração podem ser utilizados para encontrar problemas em trechos de código programado em JavaScript;
- Script é uma linguagem que será interpretada pelo navegador. Para criar um script, basta um editor de texto puro e um navegador;

JavaScript

- A partir do bloco de notas, é possível escrever os códigos JavaScript de forma separada, em um arquivo de extensão .js, e depois incluir esse arquivo no documento HTML. Mas, nesse caso, as tags `<script>` e `</script>` não devem ser incluídas;
- A JavaScript possui suporte a vários aspectos da programação orientada a objetos. Os objetos podem ser internos da linguagem, do ambiente de hospedagem ou personalizados, ou seja, criados por quem desenvolve a aplicação, e possuem propriedades e métodos.



1

Introdução

Teste seus conhecimentos

Ana G. C.
778.6008.



IMPACTA
EDITORA

1. O projeto CommonJS foi fundado com qual intuito?

- a) Definir padrões de desenvolvimento em JavaScript orientado a objetos.
- b) Definir uma biblioteca padrão voltada para desenvolvimento de JavaScript fora de navegadores.
- c) Definir regras de utilização do JavaScript em Servidores.
- d) Definir regras de utilização da linguagem JavaScript nos navegadores do mercado.
- e) Nenhuma das alternativas anteriores está correta.

2. Qual a versão atual do JavaScript?

- a) JavaScript 1.7
- b) ECMA-262 v5
- c) JavaScript 1.8
- d) ECMA-262 v6
- e) JavaScript 5.2

3. Escolha abaixo o Framework que não é JavaScript:

- a) JQuery
- b) Backbone
- c) .NET Framework 3.0
- d) YUI
- e) Nenhuma das alternativas anteriores está correta.

4. Qual a tag utilizada para iniciar uma instrução em JavaScript?

- a) <script>
- b) <jscript>
- c) <javascript>
- d) <js>
- e) Nenhuma das alternativas anteriores está correta.

5. Qual a instrução utilizada para inserir um arquivo com extensão JS em um HTML?

- a) <script src="nome_do_arquivo.js"></script>
- b) <jssrc="nome_do_arquivo.js"></js>
- c) <script source="nome_do_arquivo.js"></script>
- d) <jscriptsrc="nome_do_arquivo.js"></jscript>
- e) Nenhuma das alternativas anteriores está correta.



Variáveis

2

- ✓ Definição de variável;
- ✓ Criação de variáveis;
- ✓ Palavras reservadas;
- ✓ Tipos de variáveis;
- ✓ Caracteres especiais;
- ✓ Concatenação;
- ✓ Constantes;
- ✓ Objetos globais.



IMPACTA
EDITORA

2.1. O que é uma variável?

Uma variável é um espaço na memória onde um dado é armazenado. Você pode salvar, em uma variável, qualquer tipo de informação necessária para que os programas possam realizar suas tarefas. Os nomes de variáveis podem ser tão curtos quanto um só caractere ou descriptivos como **nomecompleto**.

2.2. Criando variáveis

As variáveis na JavaScript podem começar com uma letra, um sublinhado (_) ou um cífrão (\$) e, a partir do segundo caractere, podem apresentar números também. A partir da versão 1.5, você pode usar caracteres ISO 8859-1 ou Unicode, como ü e å. As sequências de escape Unicode \uXXXX também podem ser usadas.

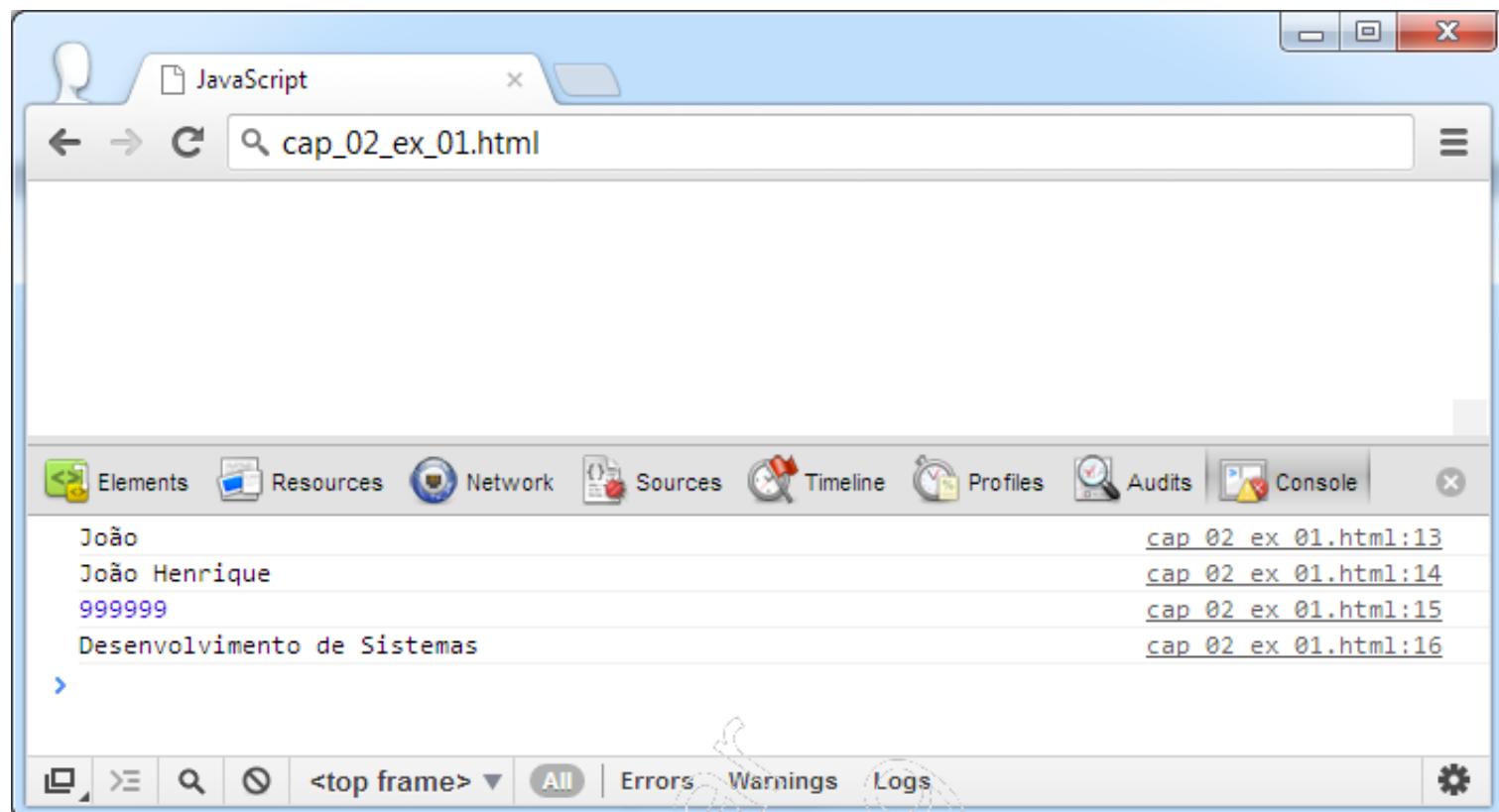
Lembre que a JavaScript diferencia letras maiúsculas de minúsculas, ou seja, **variável** não é a mesma coisa que **Variável**.

Veja alguns exemplos de nomes válidos:

```
<script>
var
    nome = 'João',
    Nome = 'João Henrique',
    $salario = 99999,
    departamento = 'Desenvolvimento de Sistemas';

    console.log(nome);
    console.log(Nome);
    console.log($salario);
    console.log(_departamento);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



2.2.1. Declarando variáveis

O processo de criar variáveis, na JavaScript, é chamado de declaração. Para declarar uma variável sem valores, você usa a palavra-chave **var**, como mostrado a seguir:

```
var nome_da_variavel;
```

Mas também dá para declarar uma variável já com valores atribuídos. Dê uma olhada:

```
<script>
  var treinamento = 'JavaScript';
</script>
```

Para atribuir um valor textual, não se esqueça de usar aspas antes e depois do valor. Outra coisa que você precisa lembrar é que uma variável não perde seu valor se for redeclarada.

2.2.2. Variáveis locais

Chamamos de variáveis locais aquelas declaradas dentro de uma função da JavaScript e só reconhecidas dentro dessa função. Quando a função é completada, as variáveis locais são imediatamente excluídas.

Como só são reconhecidas dentro da função onde são declaradas, você pode usar o mesmo nome para variáveis locais que estiverem dentro de funções diferentes sem problema algum.

2.2.3. Variáveis globais

Ao contrário das variáveis locais, as globais são aquelas declaradas fora de uma função, e tem escopo sobre toda a página web. São excluídas quando você fecha a página.

Outra forma de criar uma variável global é atribuindo valores a variáveis que ainda não existem, ou seja, que ainda não tenham sido declaradas, como você vê no exemplo a seguir:

```
<script>
global = 'Impacta';
</script>
```

2.2.4. Aritmética com a JavaScript

É possível realizar operações aritméticas com as variáveis da JavaScript. Dê uma olhada:

```
<script>
var salario = 10000;
var aumento = 1.1;
var salario_com_aumento = salario * aumento;

console.log(salario_com_aumento);
</script>
```

2.3. Palavras reservadas

As palavras reservadas são palavras da JavaScript que o interpretador (browser) utiliza internamente. Na criação de variáveis, funções, métodos ou identificadores de objetos, você não deve usar palavras reservadas para a nomeação, já que variáveis ou funções com nomes de palavras reservadas podem gerar erros de código. São elas:

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

JavaScript

Tem ainda algumas palavras que não possuem funcionalidade atualmente, mas que já foram reservadas pelo ECMAScript, caso venham a ter alguma utilidade, e não podem ser usadas como identificadores, tanto no modo estrito quanto não-estrito. São elas:

- class
- enum
- export
- extends
- import
- super

Para quem usa o Mozilla, as palavras export e import já tiveram sua utilidade prática no passado, mas atualmente estão apenas reservadas.

Por fim, estas palavras estão reservadas apenas para o modo estrito:

- implements
- interface
- let
- package
- private
- protected
- public
- static
- yield

No Mozilla, as palavras let e yield possuem funcionalidades próprias na versão JavaScript 1.7 ou superior.

As palavras **null**, **true** e **false** são reservadas na ECMAScript para seus usos normais. Já a **const**, que também é reservada, tem uma função atribuída a ela em uma extensão não padrão no Mozilla e na maioria dos navegadores. Essa extensão pode virar padrão no futuro.

2.4. Tipos de variáveis

Uma variável pode armazenar diversas classes de informação, como textos, números inteiros, números reais, entre outros. Essas classes são conhecidas como tipos de variáveis, cada uma com características e usos próprios.

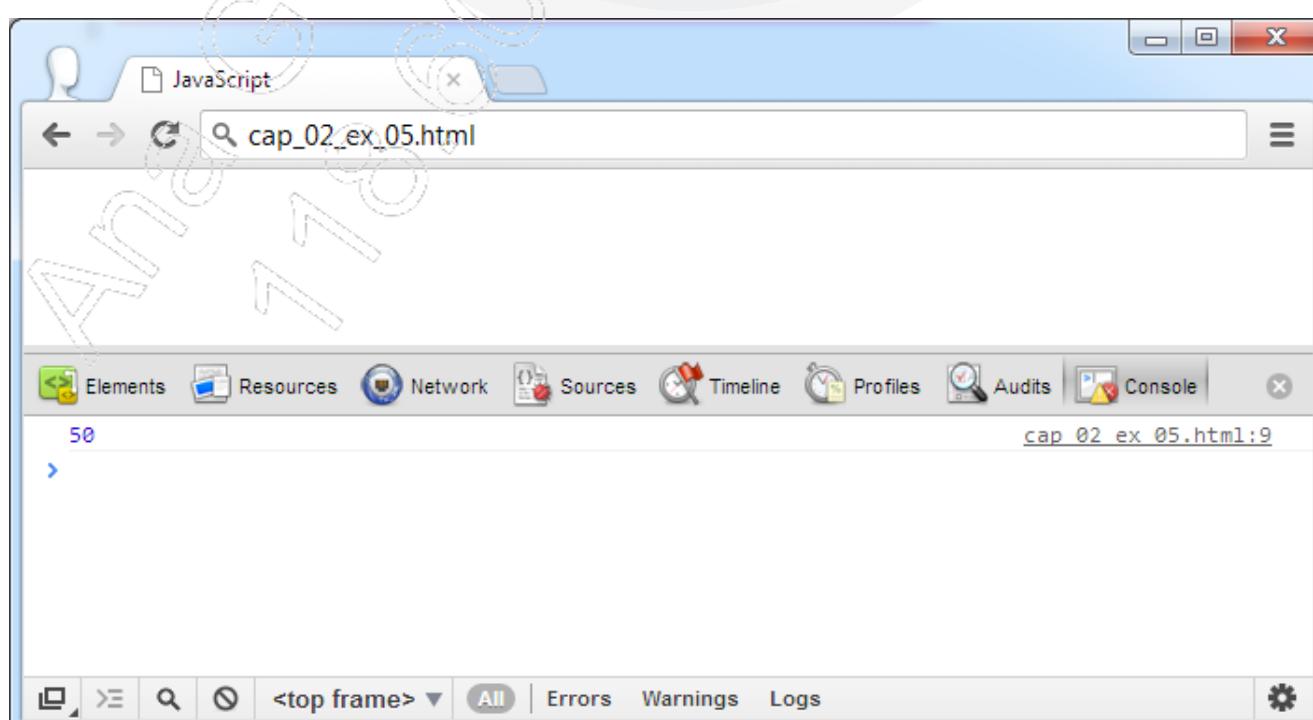
A seguir, você vai ver cada um dos tipos de variáveis da JavaScript:

- **Numéricas inteiras**

Essas variáveis armazenam números inteiros, como mostrado a seguir:

```
<script>
var inteiro = 50;
console.log(inteiro);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



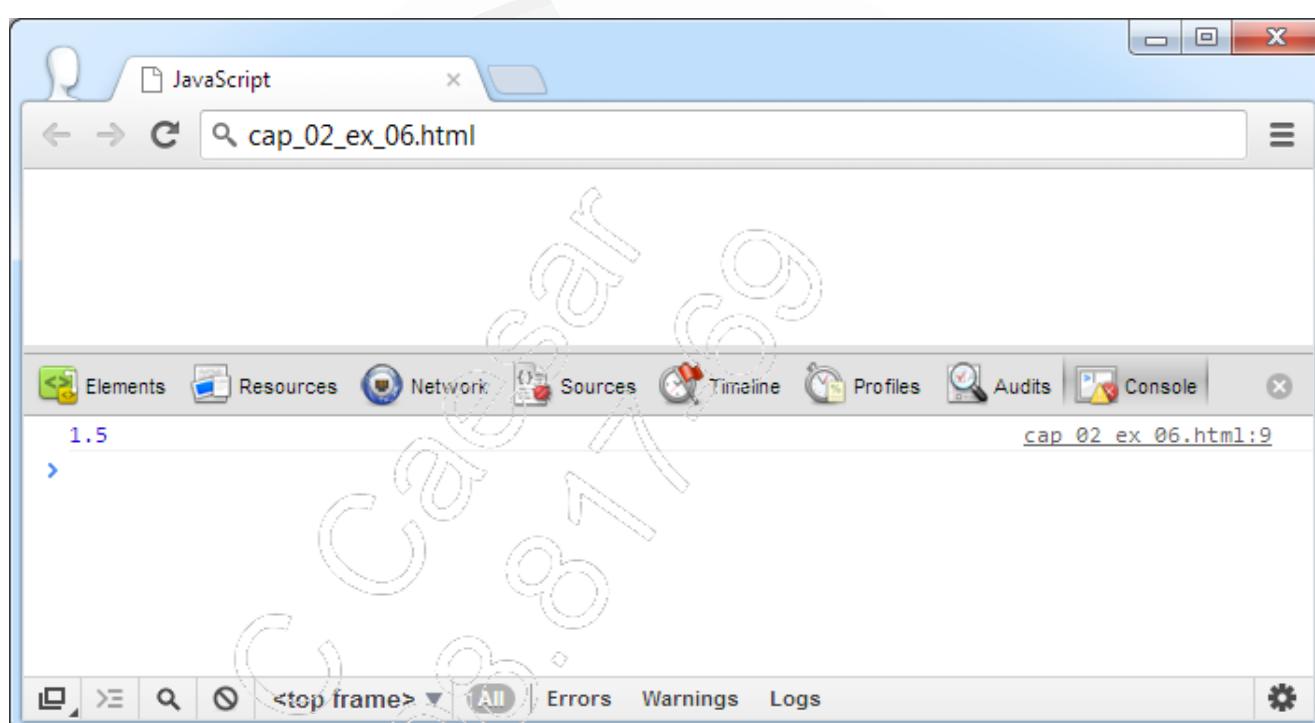
- **Numéricas Fracionárias (Ponto Flutuante)**

As variáveis numéricas fracionárias armazenam números fracionários. A casa decimal é marcada pelo ponto (.), como mostrado a seguir:

```
<script>
var flutuante = 1.5;

console.log(flutuante);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



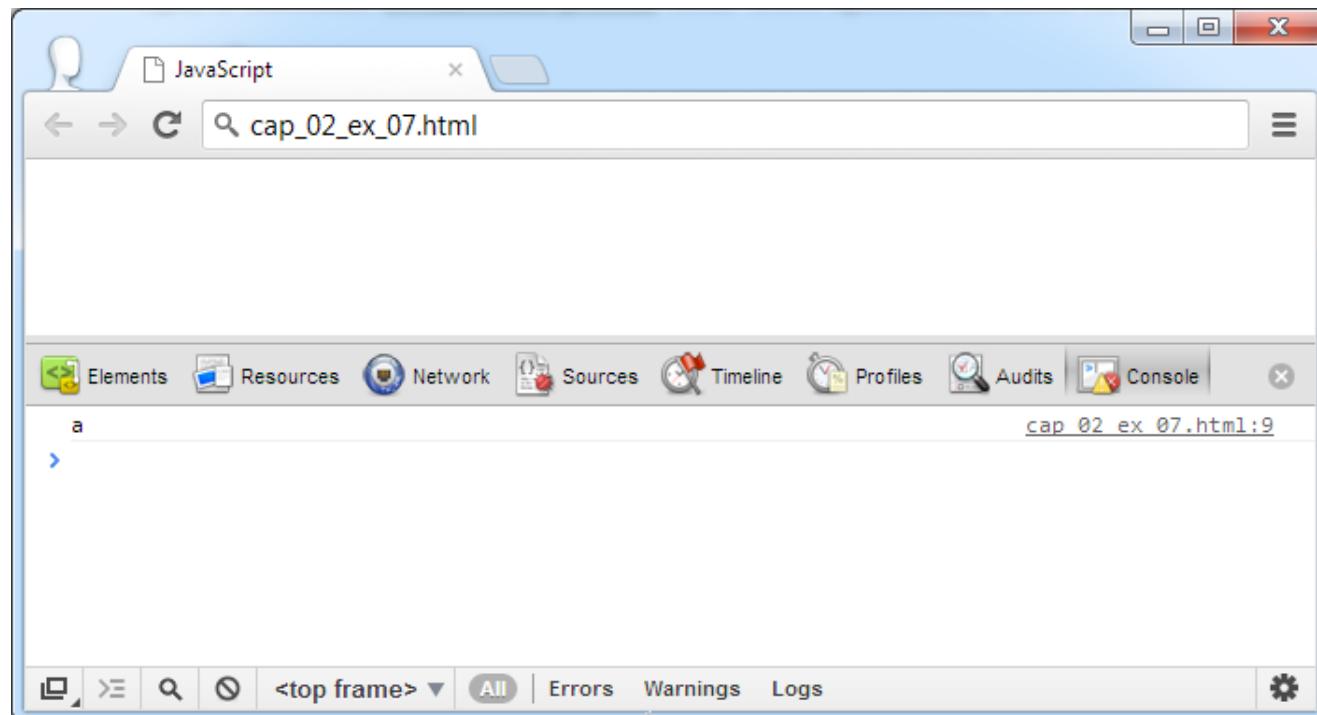
- **Caractere**

Essas variáveis armazenam apenas um caractere. Ao definir uma variável desse tipo, o conteúdo armazenado precisa estar entre apóstrofos, como no próximo exemplo:

```
<script>
var caractere = 'a';

console.log(caractere);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:

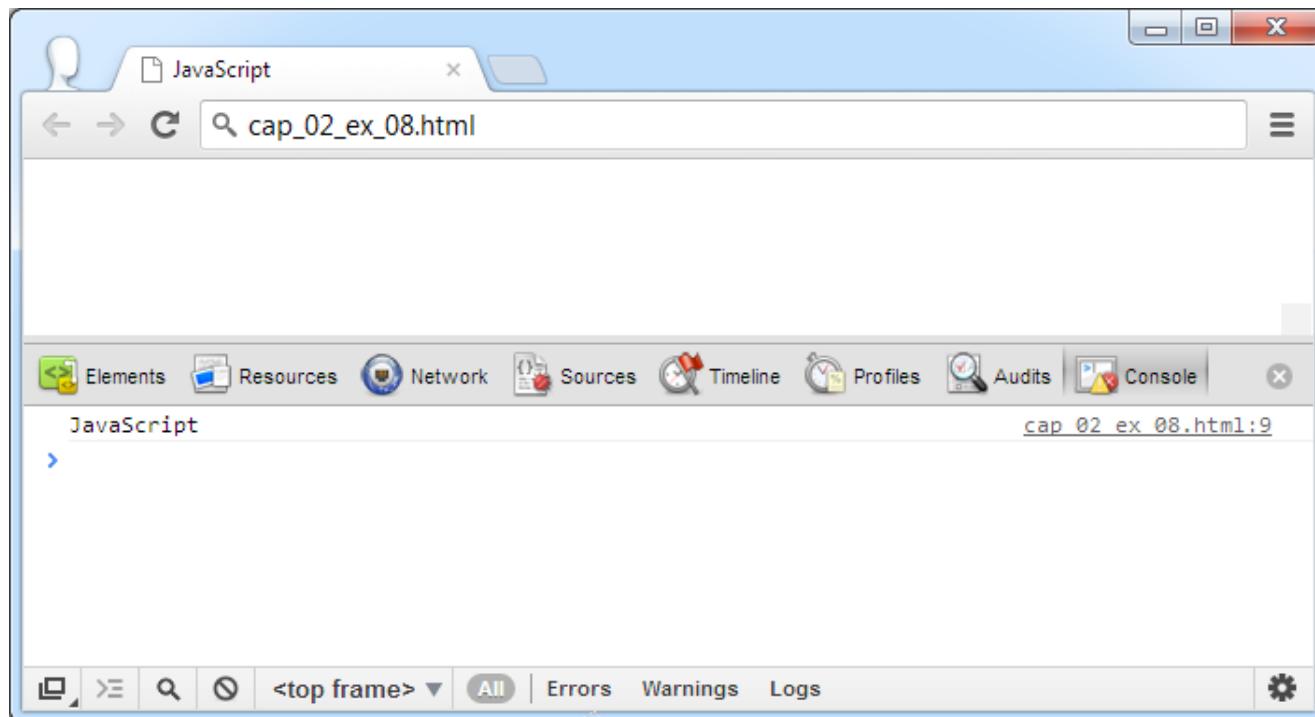


- **String**

Variáveis desse tipo armazenam uma sequência de caracteres. Na definição de strings, o conteúdo precisa estar entre aspas, como no código seguinte:

```
<script>
var texto = "JavaScript";
console.log(texto);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



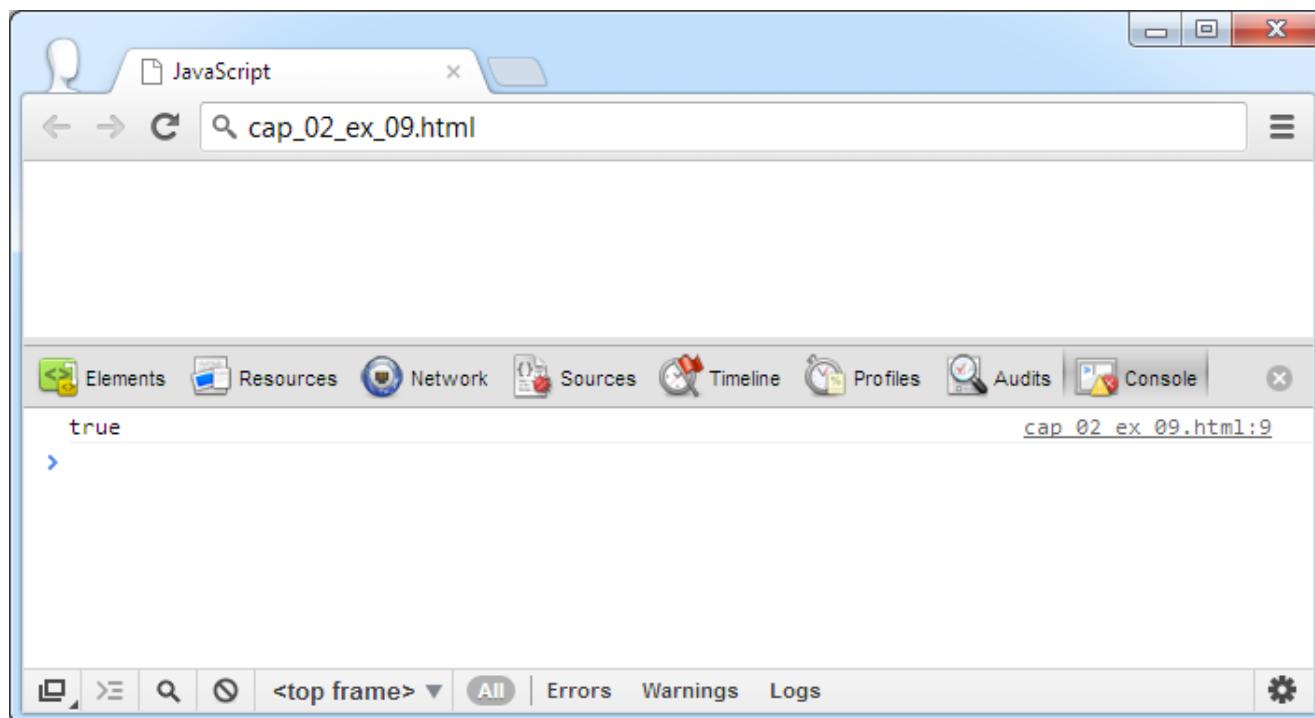
- **Booleanos**

Variáveis desse tipo armazenam estados, isto é, **true** (verdadeiro) ou **false** (falso), como demonstra o próximo código:

```
<script>
var booleano = true;

console.log(booleano);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



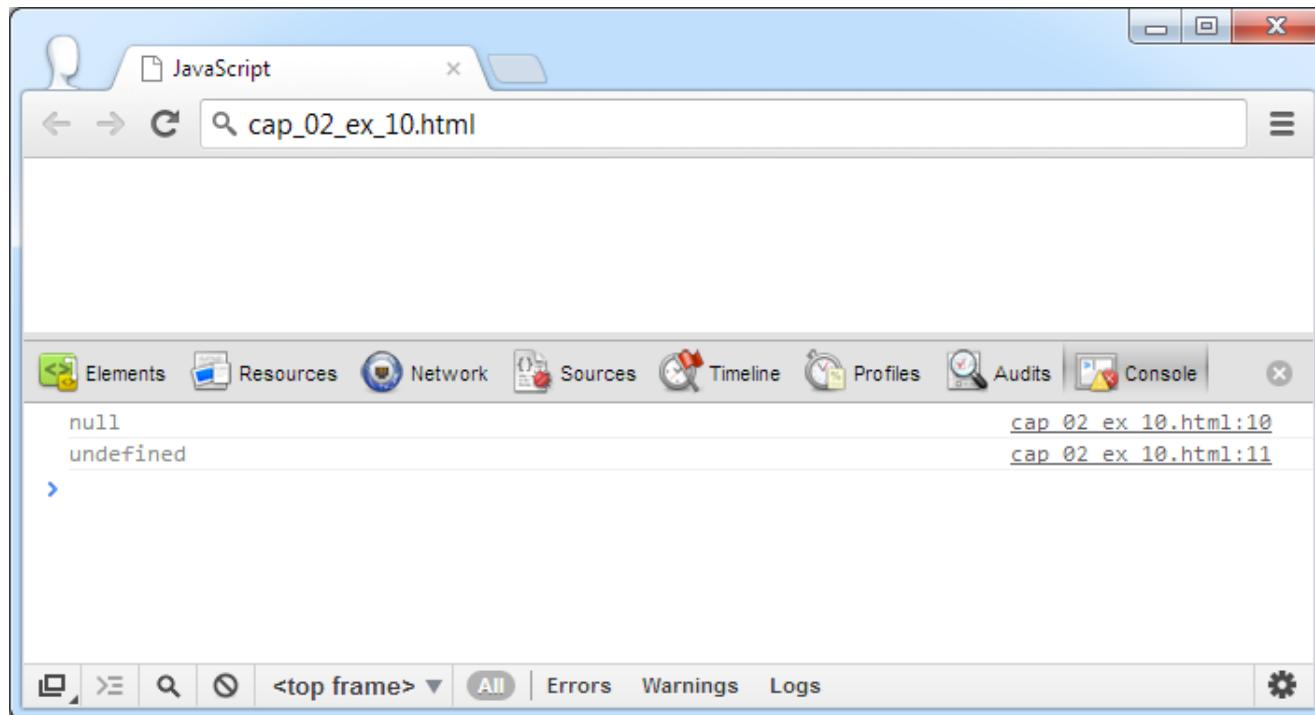
- **null**

Os dados nulos são representados pela palavra-chave **null**. Uma variável receberá o valor **undefined** caso não tenha valor atribuído.

```
<script>
var nulo = null;
var indefinido;

console.log(nulo);
console.log(indefinido);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



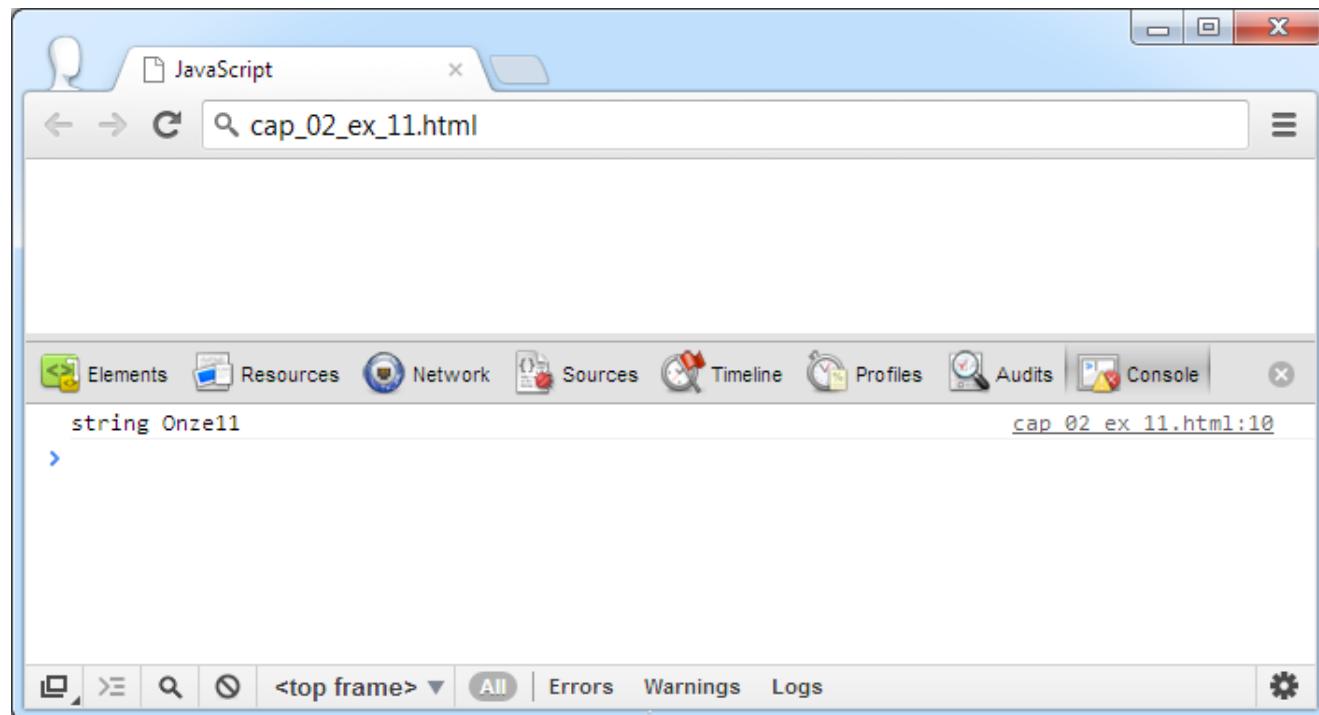
2.4.1. Convertendo strings em números

A mistura de tipos de dados diferentes pode acontecer quando as variáveis estão sendo manipuladas. Podemos ter strings junto de números. Nesse caso, é uma tendência do JavaScript interpretar esses dados como strings.

```
<script>
var numero = 11;
var texto = "Onze" + numero;

console.log(typeof texto, texto);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



Na utilização de strings com números, o resultado obtido nem sempre é o esperado. Para evitar erros decorrentes da mistura de dados, existem funções de conversão. As duas principais delas são descritas a seguir:

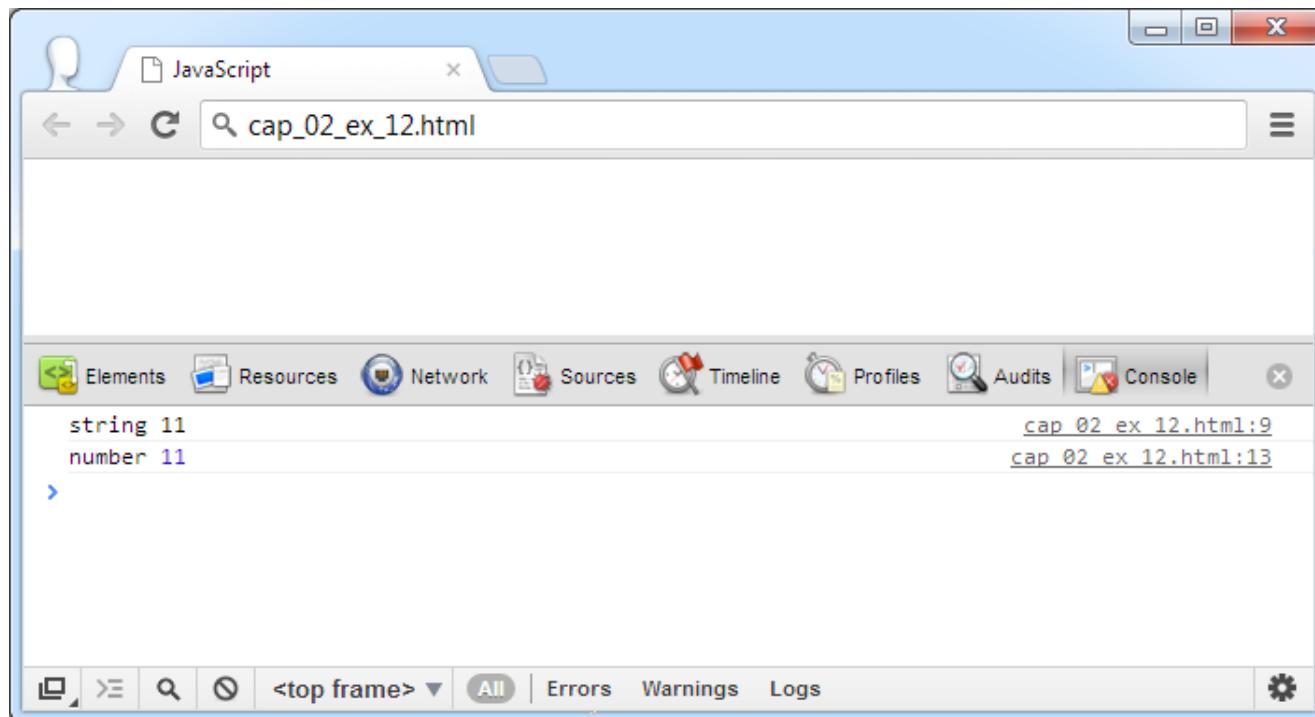
- **parseInt()**

Para converter uma string em número inteiro, você pode usar essa função. No código, a variável ou string que você queira converter em número deve ser colocada entre parênteses, como mostra o exemplo seguinte:

```
<script>
var numero = "11";
console.log(typeof numero, numero);
numero = parseInt(numero);
console.log(typeof numero, numero);
</script>
```

JavaScript

Após salvar e executar o código anterior, a seguinte página será exibida:



- **parseFloat()**

Essa função converte uma string em um número de ponto flutuante. A seguir, você vê um exemplo da utilização de **parseFloat()**:

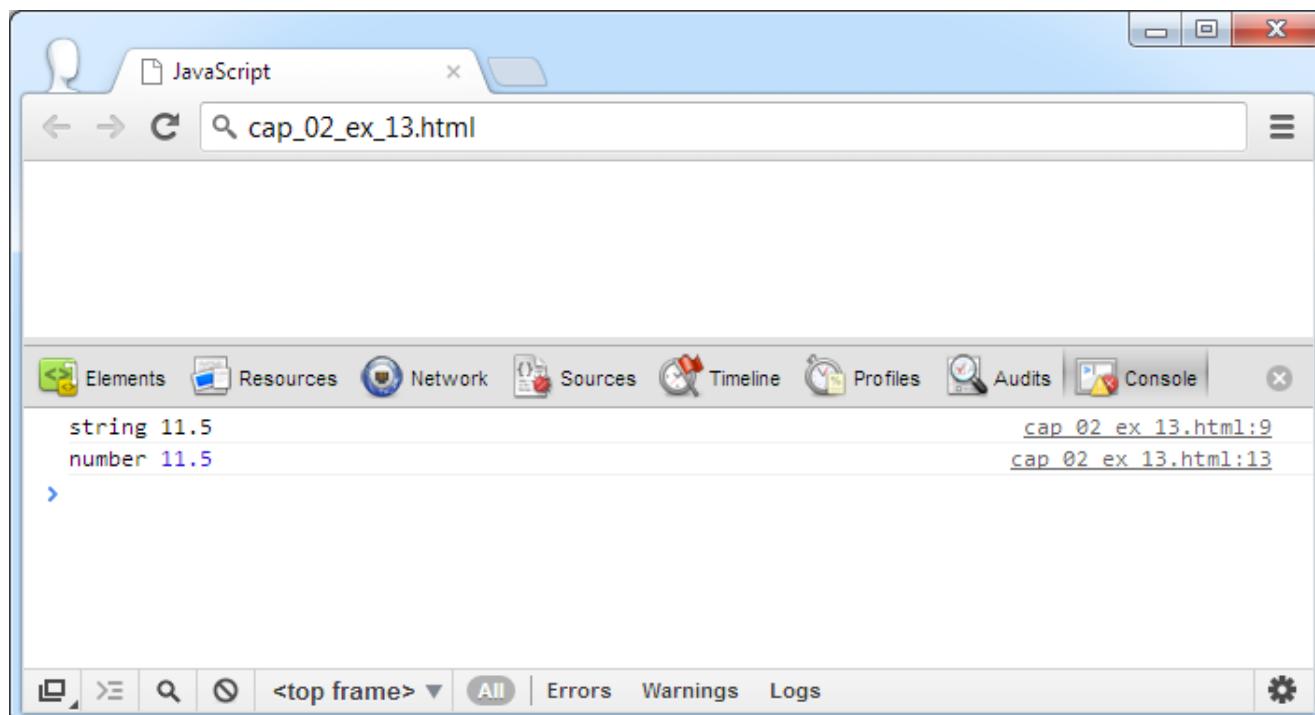
```
<script>
var numero = "11.5";

console.log(typeof numero, numero);

numero = parseFloat(numero);

console.log(typeof numero, numero);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



2.5. Caracteres especiais

A JavaScript faz uso de caracteres especiais em uma string. Um deles é o sinal de aspas, um caractere reservado dentro da JavaScript. Como sabemos, uma string deve ser declarada sempre entre aspas. Mas o programador pode querer inserir, dentro de uma string, uma palavra acompanhada do sinal de aspas. Para isso, deve usar a barra invertida (\) junto das aspas, e isso serve para os demais caracteres especiais.

A tabela a seguir descreve os tipos de caracteres especiais e os respectivos códigos utilizados no script:

Tipo	Código
Aspas ("")	\"
Apóstrofo ('')	\'
Barra invertida (\)	\\\
Alimentação do formulário	\f
Retrocesso	\
Nova linha	\n

JavaScript

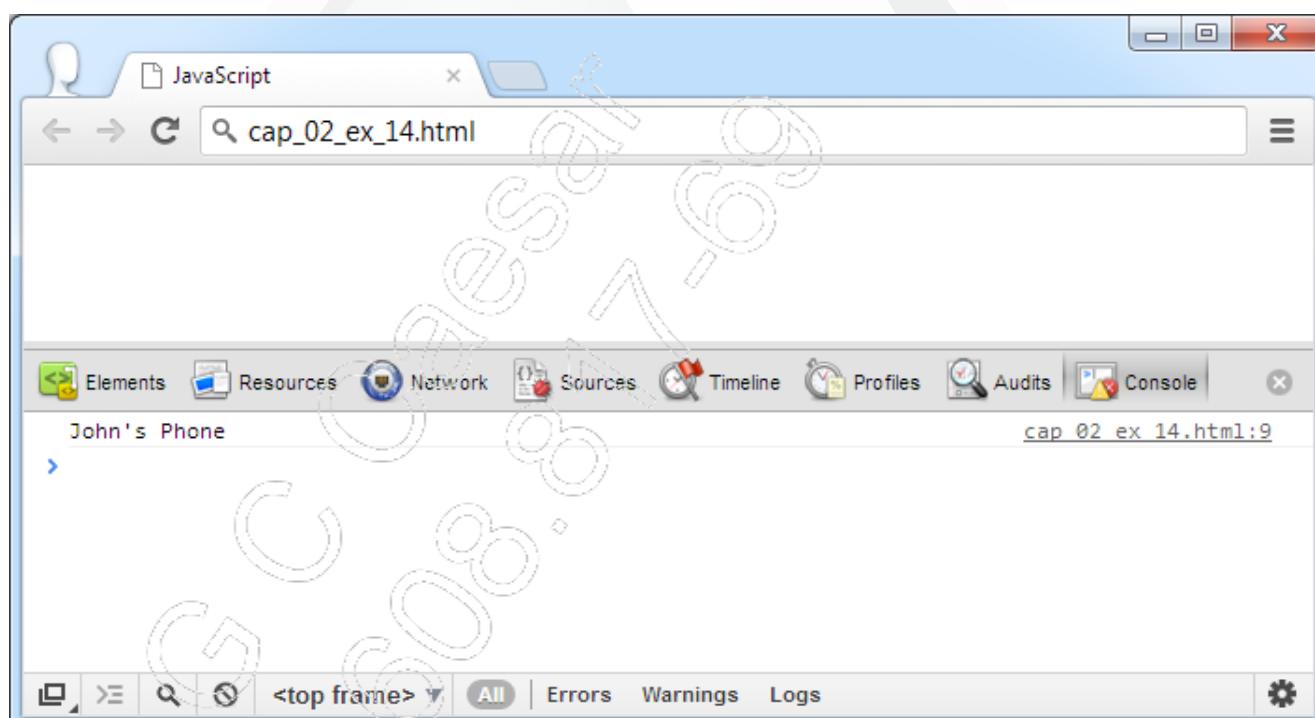
Tipo	Código
Tabulação	\t
Retorno de carro	\r

Veja um exemplo de uso de caracteres especiais:

```
<script>
var texto = 'John\'s Phone';

console.log(texto);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



Alguns caracteres especiais, como o de nova linha, só funcionam dentro de caixas de alerta. O navegador reconhece tags HTML dentro do texto, quando você usa o comando `document.write` (faz a escrita diretamente na página HTML). Em caixas de alerta, as tags HTML não são reconhecidas.

2.6. Concatenação

Na JavaScript, você pode somar strings e variáveis de todos os tipos. Isso é chamado de concatenação. A JavaScript faz a conversão automática na concatenação de uma string com qualquer outro tipo de variável. Comandos do HTML também podem ser concatenados. Para isso, eles devem ser inseridos entre aspas, como mostrado adiante:

```
<script>
var texto = 'JavaScript';

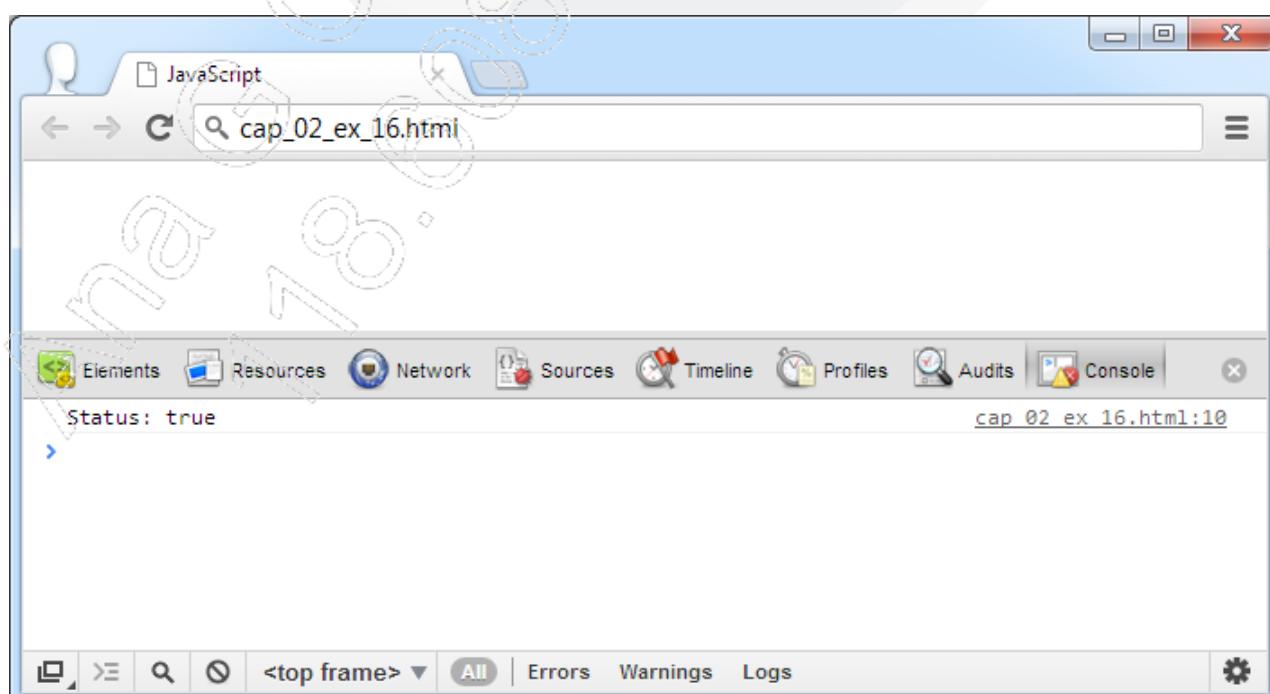
document.write("<h1>" + texto + "</h1>");
</script>
```

O exemplo a seguir demonstra a declaração e a concatenação de dados:

```
<script>
var status = true;
var texto = 'Status:';

console.log(texto+status);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



2.7. Constantes

As constantes seguem as mesmas regras das variáveis para declaração e nomeação, com a diferença de usarem a palavra-chave **const**, sempre necessária, mesmo para constantes globais. Se você não incluir a palavra-chave **const**, a JavaScript interpretará que aquilo é uma variável.

O nome escolhido para a constante não pode ser o mesmo de nenhuma variável ou função do mesmo escopo. Além disso, o valor de uma constante não pode ser alterado ou redeclarado enquanto o script estiver sendo executado.

```
<script>
const navegador = 'Chrome';
</script>
```

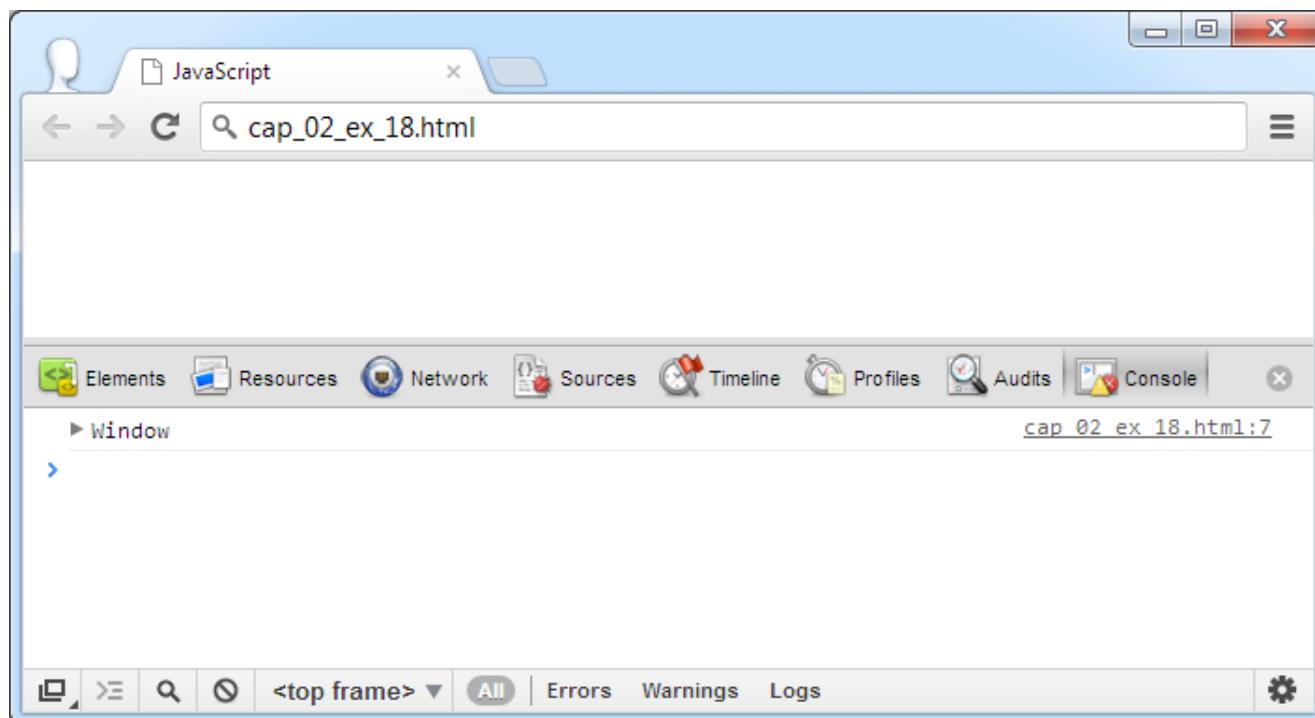
2.8. Objetos globais

Objeto global é onde ficam todas as propriedades e métodos da linguagem e do navegador que você está usando. Um objeto global é automaticamente criado sempre que você iniciar um ambiente de hospedagem da JavaScript. É no objeto global que ficam declaradas as variáveis globais, que você já estudou.

Para fazer referência ao objeto global, você pode usar a palavra-chave **this** (ou **window**), mas preste atenção para que ela esteja fora do corpo de uma função, caso contrário, a referência será feita a outro objeto da própria função, que chamamos genericamente de call object.

```
<script>
console.log(this);
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



2.8.1. Variáveis e propriedades dos objetos

As propriedades de um objeto são os valores de suas variáveis, ou seja, propriedades e variáveis de objetos são exatamente a mesma coisa.

2.8.2. null

A palavra-chave **null**, quando usada, mostra que não existe nenhum valor associado a uma variável, seja ele do tipo que for.

```
<script>
var variavel = null;

console.log(variavel);
</script>
```

2.8.3. undefined

Diferente da **null**, a palavra-chave **undefined**, que é uma propriedade do objeto global, define um valor indefinido a uma variável. Isso acontece quando ela é declarada e não iniciada.

```
<script>
var variavel;

console.log(variavel);
</script>
```

2.8.4. NaN

Outra propriedade do objeto global que você precisa conhecer é a **NaN**, que serve para representar um valor que não seja um número.

```
<script>
var variavel = "X";
console.log(parseInt(variavel));
</script>
```

2.8.5. Infinity

A última propriedade do objeto global que você verá neste capítulo é a **infinity**, que representa um valor infinito positivo. O interpretador JavaScript consegue manipular os números encontrados entre $-1.7976931348623157 \times 10^{308}$ e $1.7976931348623157 \times 10^{308}$.

```
<script>
var variavel = 1.7976931348623157*Math.pow(10, 308);

console.log(variavel);
</script>
```

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Uma variável é um espaço na memória onde um dado é armazenado. Você pode salvar, em uma variável, qualquer tipo de informação necessária para que os programas possam realizar suas tarefas. Os nomes de variáveis podem ser tão curtos quanto um só caractere ou descritivos como **nomecompleto**;
- O processo de criar variáveis, na JavaScript, é chamado de declaração. Para declarar uma variável sem valores, você usa a palavra-chave **var**. As variáveis na JavaScript podem começar com uma letra, um sublinhado (_) ou um cífrão (\$) e, a partir do segundo caractere, podem apresentar números também;
- As palavras reservadas são palavras da JavaScript que o interpretador (browser) utiliza internamente. Na criação de variáveis, funções, métodos ou identificadores de objetos, você não deve usar palavras reservadas para a nomeação, já que variáveis ou funções com nomes de palavras reservadas podem gerar erros de código;
- Uma variável pode armazenar diversas classes de informação, como textos, números inteiros, números reais, entre outros. Essas classes são conhecidas como tipos de variáveis, cada uma com características e usos próprios. As variáveis podem ser numéricas inteiras, numéricas fracionárias, caracteres, strings, booleanos ou null;
- A JavaScript faz uso de caracteres especiais em uma string. Um deles é o sinal de aspas, um caractere reservado dentro da JavaScript. Como sabemos, uma string deve ser declarada sempre entre aspas. Mas o programador pode querer inserir, dentro de uma string, uma palavra acompanhada do sinal de aspas. Para isso, deve usar a barra invertida (\) junto das aspas, e isso serve para os demais caracteres especiais;

- Na JavaScript, você pode somar strings e variáveis de todos os tipos. Isso é chamado de concatenação. A JavaScript faz a conversão automática na concatenação de uma string com qualquer outro tipo de variável. Comandos do HTML também podem ser concatenados. Para isso, eles devem ser inseridos entre aspas;
- As constantes seguem as mesmas regras das variáveis para declaração e nomeação, com a diferença de usarem a palavra-chave **const**, sempre necessária, mesmo para constantes globais. Se você não incluir a palavra-chave **const**, a JavaScript interpretará que aquilo é uma variável;
- Objeto global é onde ficam todas as propriedades e métodos da linguagem e do navegador que você está usando. Um objeto global é automaticamente criado sempre que você iniciar um ambiente de hospedagem da JavaScript. É no objeto global que ficam declaradas as variáveis globais.



2

Variáveis Teste seus conhecimentos

Ana Gomes
778.6008-0000



IMPACTA
EDITORA

1. Qual das palavras não é reservada no JavaScript?

- a) Break
- b) Finally
- c) Var
- d) Void
- e) Foreach

2. Qual o comando utilizando para você fazer referência a um objeto global?

- a) this
- b) var
- c) const
- d) super
- e) new

3. Podemos definir variáveis locais como:

- a) Aquelas variáveis declaradas dentro de uma função e só reconhecidas dentro dessa função.
- b) Aquelas variáveis declaradas fora de uma função e só reconhecidas dentro da função.
- c) Aquelas variáveis declaradas dentro de uma função e também pode ser reconhecida fora dessa função.
- d) Aquelas variáveis declaradas fora de uma função e podem ser acessadas por qualquer outra função.
- e) Nenhuma das alternativas anteriores está correta.

4. Qual dos caracteres corresponde ao caractere de Tabulação?

- a) \t
- b) \'
- c) \f
- d) \
- e) \\

5. A propriedade do objeto global infinity representa um valor:

- a) Infinito negativo
- b) Inteiro
- c) Infinito positivo
- d) Booleano
- e) String infinita



2

Variáveis Mãos à obra!

Ana Góes
778.6000



IMPACTA
EDITORA

Laboratório 1

A - Criando variáveis, atribuindo valores, concatenando e mostrando resultado na tela

1. Crie um novo arquivo de texto e renomeie com a extensão **.html**;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD**, **BODY**;
4. Inclua dentro da tag **HEAD** a tag **<script></script>**;
5. Declare dentro da tag **<script>** duas variáveis:
 - A primeira deve-se chamar **habitante**. O valor dela é: **200000**;
 - A segunda deve-se chamar **renda**. O valor dela é: **1000**.
6. Crie uma variável chamada **resultado** e atribua a ela o resultado da divisão da variável **habitante** pela variável **renda**;
7. Declare uma variável chamada **descricao**:
 - O valor desta variável deve ser: “**Renda por habitante é:**”
8. Concatene e mostre na tela o valor da variável **descricao** com o resultado do cálculo da renda por habitante.

Operadores

3

- ✓ Utilização de operadores em JavaScript;
- ✓ Expressões;
- ✓ Tipos de operadores;
- ✓ Operadores de atribuição;
- ✓ Operadores de comparação;
- ✓ Operadores aritméticos;
- ✓ Operadores bitwise;
- ✓ Operadores lógicos;
- ✓ Operadores de string;
- ✓ Operadores especiais;
- ✓ Precedência dos operadores.

3.1. Utilizando operadores em JavaScript

Os operadores são utilizados em praticamente todas as linguagens de programação existentes no mercado. Como você pode ver pelo próprio nome, estes operadores permitem realizar operações e cálculos, assim como você aprendeu em matemática, e são empregados especialmente durante o desenvolvimento dos programas.

O resultado dessas operações permite ao programa variar o seu comportamento conforme as informações obtidas.

Existem diferentes tipos de operadores, e estes podem ser destinados a operações simples ou complexas que envolvem operandos de tipos de dados distintos, por exemplo, textos ou números. A partir de agora, você vai ver mais detalhes sobre os tipos de operadores disponíveis na linguagem JavaScript.

3.2. Expressões

Quando tratamos de JavaScript (e também outras linguagens de programação), chamamos de expressão qualquer unidade de código que é válida e que resulta em um valor.

Sabendo disso, considere que existam dois tipos de expressões. Na primeira, o valor resultante é atribuído a uma variável. Na segunda, você tem apenas o valor que resulta dela.

Quando usamos operadores de atribuição, temos, por exemplo, o seguinte caso:

```
x=3
```

Isso define que a variável `x` possui o valor 3.

Já no caso de uma expressão como `8 - 5`, cujo resultado é 3, você tem apenas o valor definido, pois ele não é atribuído a nenhuma variável. Nestas expressões são usados operadores simples, que você pode chamar simplesmente de operadores.

As expressões podem ser classificadas pelo tipo de valor que ela resulta. Veja a seguir:

Tipo de expressão	Resultado
Aritmética	Valor numérico, por exemplo, 1,61803399. Geralmente usa operadores aritméticos.
String	String de caracteres, por exemplo, "Impacta". Geralmente usa operadores de string.
Lógica	Valores verdadeiro (true) ou falso (false). Geralmente usa operadores lógicos.
Objeto	Um objeto. Geralmente usa operadores especiais.

3.3. Tipos de operadores

Os tipos de operadores disponíveis para a linguagem de programação JavaScript são os seguintes:

- Operadores de atribuição;
- Operadores de comparação;
- Operadores aritméticos;
- Operadores bitwise;
- Operadores de string;
- Operadores especiais.

3.4. Operadores de atribuição

Com os operadores de atribuição, você pode armazenar informações nas variáveis de memória do sistema.

JavaScript

Veja os diferentes tipos de operadores de atribuição na tabela adiante:

Operadores de atribuição	Descrição
Sinal de igualdade (=)	Este operador, que indica apenas uma atribuição, faz com que a parte direita do sinal de igual seja atribuída à parte da esquerda. Ao passo que a parte da esquerda costuma receber uma variável onde se deseja salvar o dado, a parte da direita geralmente recebe os valores finais.
Sinais de adição e igualdade (+=)	Este operador, que indica uma atribuição com soma, faz com que a parte da direita seja somada com a parte da esquerda. O resultado da operação, por sua vez, é salvo na parte da esquerda.
Sinais de subtração e igualdade (-=)	Este operador é responsável por indicar uma atribuição com subtração.
Sinais de multiplicação e igualdade (*=)	Este operador tem a finalidade de indicar uma atribuição da multiplicação.
Sinais de barra e igualdade (/=)	Este operador é responsável por indicar uma atribuição da divisão.
Sinais de porcentagem e igualdade (%=)	Este operador permite obter o resto e, então, atribuí-lo.

Veja agora um exemplo que demonstra a utilização de operadores de atribuição:

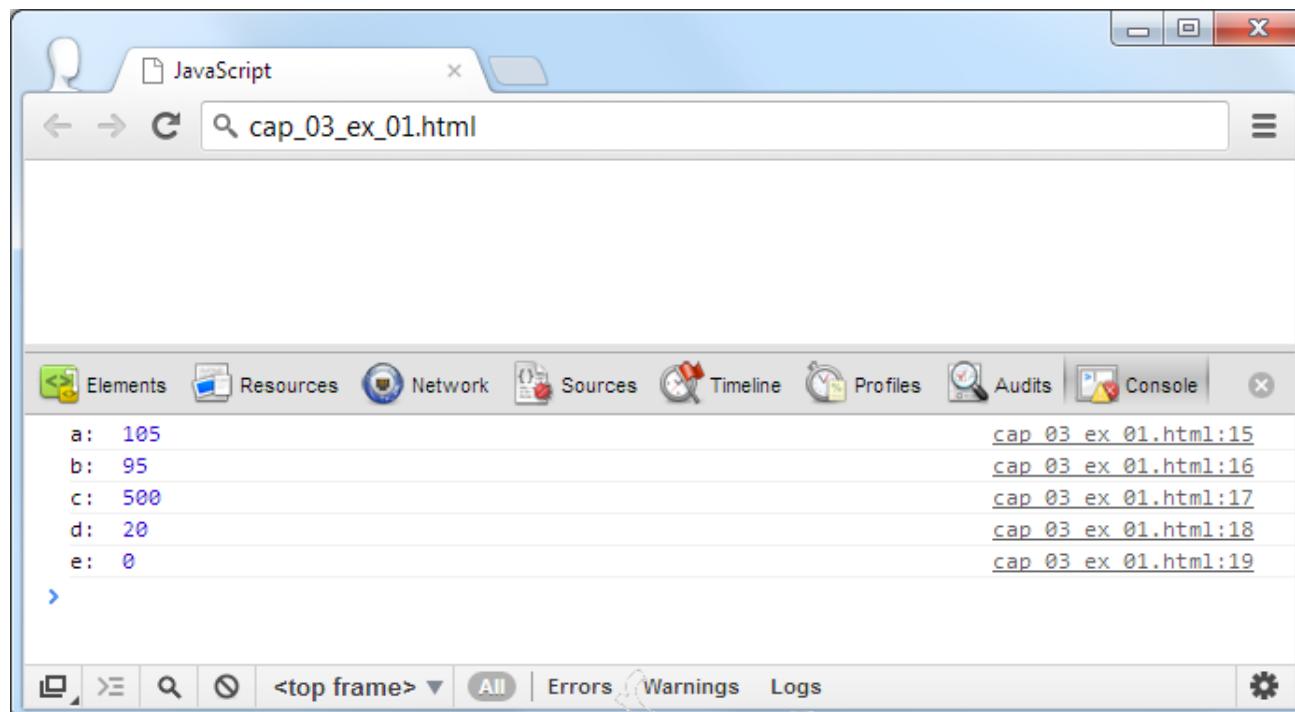
```
<script>
var a = b = c = d = e = 100;

a += 5;
b -= 5;
c *= 5;
d /= 5;
e %= 5;

console.log('a: ', a);
console.log('b: ', b);
console.log('c: ', c);
console.log('d: ', d);
console.log('e: ', e);

</script>
```

Como resultado, a seguinte página será exibida depois que salvar o código anterior e executá-lo:



3.5. Operadores de comparação

Utilizando os operadores de comparação, você pode realizar a comparação entre conteúdos de variáveis de memória.

Veja os diferentes tipos de operadores condicionais na tabela adiante:

Operadores condicionais	Descrição
Dois sinais de igualdade (<code>= =</code>)	Este operador permite verificar se dois números são idênticos.
Sinais de exclamação e igualdade (<code>!=</code>)	Por meio deste operador, é possível comprovar se dois números são diferentes.
Sinal de maior (<code>></code>)	Este operador, que indica “maior que”, é responsável por retornar o valor true se o elemento da esquerda for maior que o da direita.
Sinais de maior e igualdade (<code>>=</code>)	Este operador, que indica “maior igual”, retorna o valor true se o primeiro elemento for maior ou igual ao segundo.

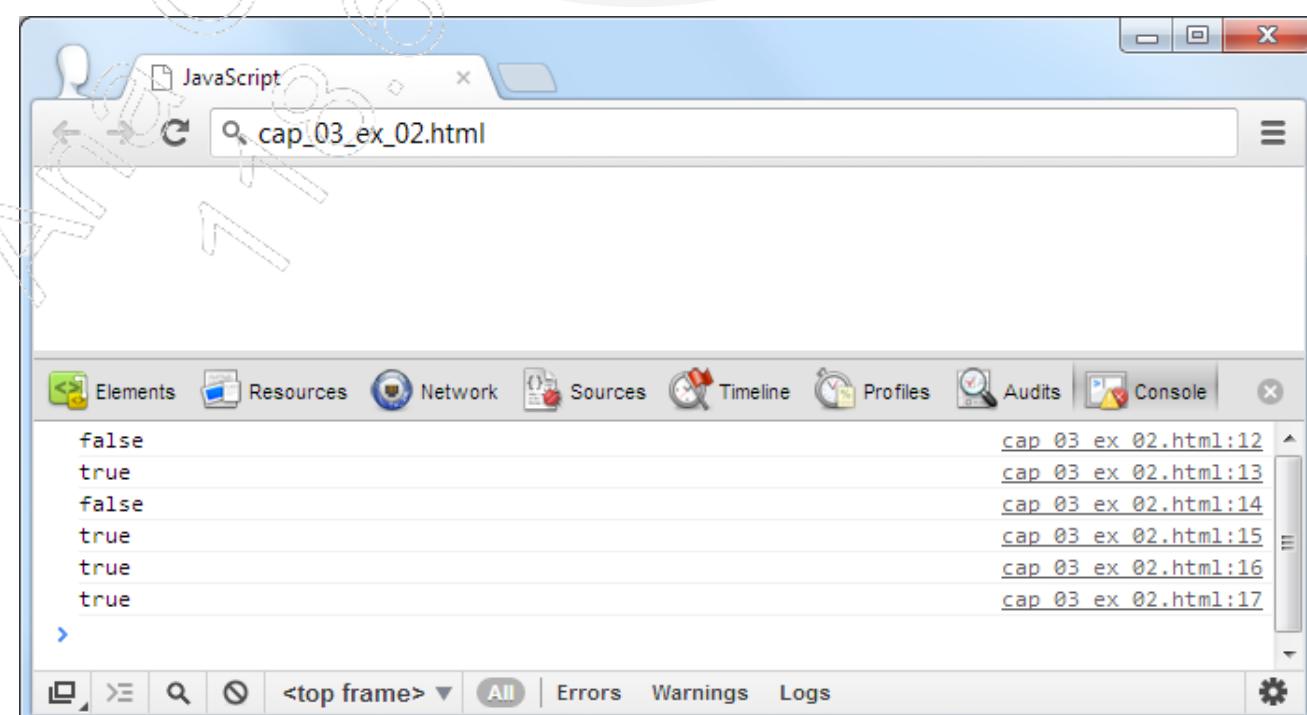
Operadores condicionais	Descrição
Sinal de menor (<)	Este operador, que indica “menor que”, é responsável por retornar o valor true quando o elemento da esquerda for menor que o da direita
Sinais de menor e igualdade (<=)	Este operador, que indica “menor igual”, retorna o valor true se o primeiro elemento for menor ou igual ao segundo.

Veja agora um exemplo que demonstra a utilização de operadores condicionais:

```
<script>
var a = 10;
var b = 20;
var c = 30;
var d = 40;

console.log(a == b);
console.log(b != c);
console.log(c > d);
console.log(d >= a);
console.log(a < b);
console.log(c <= d);
</script>
```

Como resultado, a seguinte página será exibida depois que salvar o código anterior e executá-lo:



3.6. Operadores aritméticos

Os diferentes tipos de operadores aritméticos disponíveis em JavaScript – os quais são empregados em referências indexadoras, cálculos e manuseio de strings – são descritos na tabela a seguir:

Operadores aritméticos	Descrição
Sinal de adição (+)	Este operador permite obter a soma de dois valores.
Sinal de subtração (-)	Este operador permite obter a diferença de dois valores.
Sinal de multiplicação (*)	Este operador é responsável por multiplicar dois valores.
Sinal de barra (/)	Este operador tem a finalidade de dividir dois valores.
Sinal de porcentagem (%)	Este operador permite obter o resto da divisão de dois valores.
Dois sinais de adição (++)	Este operador é utilizado com um único operando e permite obter o incremento em uma unidade.
Sinal de decremento (--)	Este operador é utilizado com um único operando e permite obter o decremento em uma unidade.

Veja um exemplo que demonstra a utilização dos operadores aritméticos:

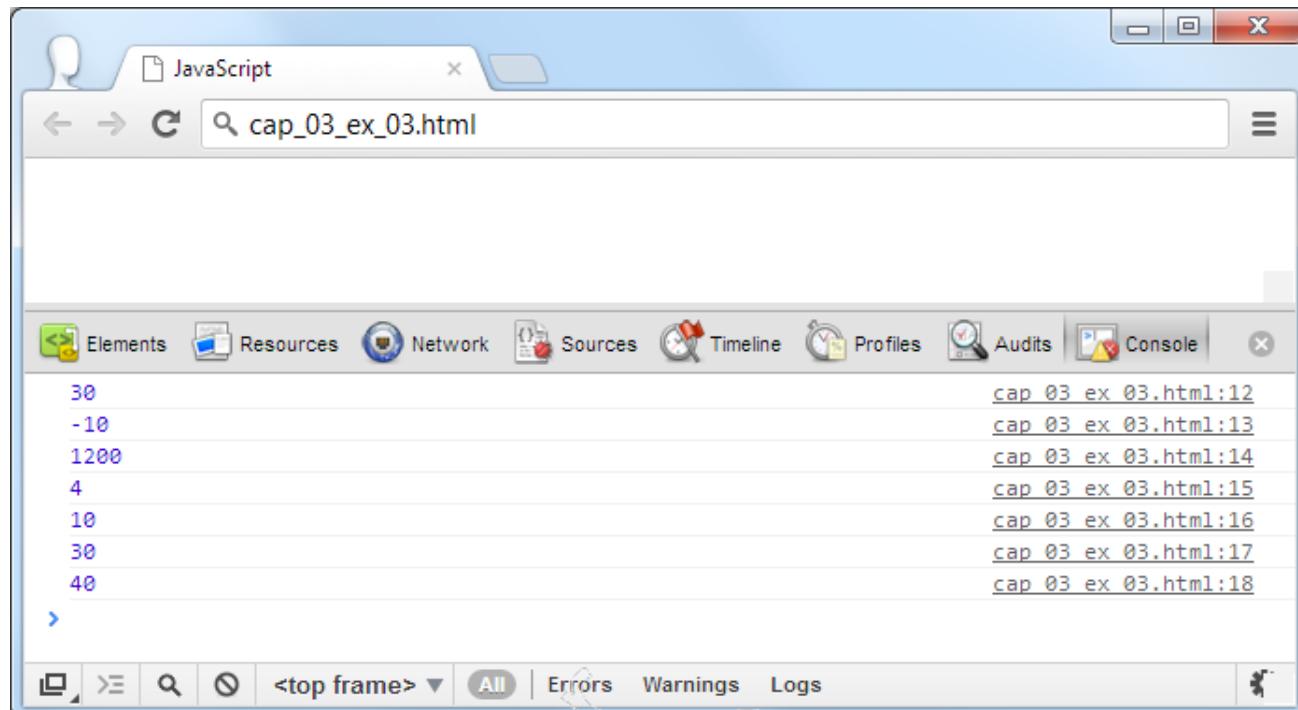
```

<script>
var a = 10;
var b = 20;
var c = 30;
var d = 40;

console.log(a + b);
console.log(b - c);
console.log(c * d);
console.log(d / a);
console.log(a % b);
console.log(c++);
console.log(d--);
</script>

```

Como resultado, a seguinte página será exibida depois que salvar o código anterior e o executá-lo:



3.7. Operadores bitwise

Os operadores bitwise trabalham com valores binários, ou seja, seus operandos são formados por conjuntos de 32 bits, ou dígitos, e cada um deles tem o valor de 1 ou 0. Uma operação bitwise vai sempre fornecer uma resposta em valor numérico padrão, mesmo tendo as operações realizadas em valores binários.

3.7.1. Operadores bitwise lógicos

Os operadores bitwise lógicos trabalham com pares destes operandos binários de 32 dígitos, onde, em um operando, cada dígito corresponde a outro dígito do outro operando, em ordem respectiva.

Os operadores bitwise lógicos são:

Operador	Sintaxe	Descrição
AND Bitwise	a & b	Define o valor 1 em cada posição onde ambos os operandos possuem o valor 1.

Operador	Sintaxe	Descrição
OR Bitwise	$a b$	Define o valor 1 em cada posição onde é encontrado o valor 1 , seja em um operando ou no outro, ou em ambos.
XOR Bitwise	$a ^ b$	Define o valor 1 para cada posição onde, correspondentemente, existe o valor 1 em apenas um dos operandos.
NOT Bitwise	$\sim a$	Define o valor final como o inverso do valor inicial, ou seja, valores 1 tornam-se 0 e vice-versa.

Considere como exemplo os valores binários de 13 e 8, que são, respectivamente, 1101 e 1000. Veja como os operadores bitwise lógicos funcionam:

Expressão	Resultado	Descrição binária
13 & 8	8	$1101 \& 1000 = 1000$
13 8	13	$1101 1000 = 1101$
13 ^ 8	5	$1101 ^ 1000 = 0101$
~ 13	-14	$\sim 1101 = 0010$
~ 8	-9	$\sim 1000 = 0111$

Veja um exemplo que demonstra a utilização dos operadores bitwise lógicos:

```

<script>
var a = 13;
var b = 8;

console.log(a & b);
console.log(a | b);
console.log(a ^ b);
console.log(~a);
console.log(~b);

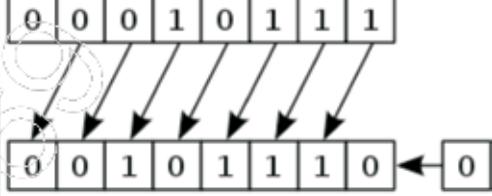
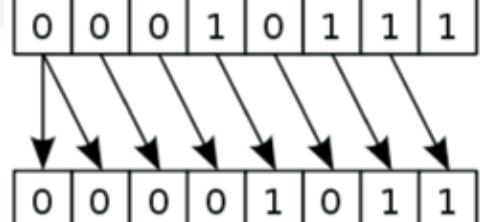
</script>

```

3.7.2. Operadores bitwise de deslocamento

Você vai encontrar três tipos de operadores bitwise de deslocamento, e seu funcionamento é muito simples.

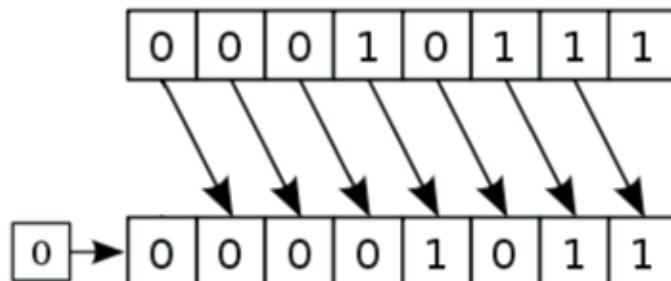
Nos três tipos de operadores, destacados logo adiante, você deve trabalhar com dois tipos de valores: o valor a ser deslocado, e a quantidade de dígitos que este valor sofre de deslocamento. É aí que entram os operadores, que definem para que lado este deslocamento ocorrerá (direito ou esquerdo) e o que ocorre com os dígitos afetados. Veja na tabela como isso funciona:

Deslocamento para a esquerda	$a << b$	Desloca o valor a , em representação binária, b dígitos para a esquerda, preenchendo os dígitos à direita com 0. 
Deslocamento para a direita	$a >> b$	Desloca o valor a , em representação binária, b dígitos para a direita, descartando os valores dos dígitos deslocados e copiando o valor do último dígito da esquerda na esquerda. 

Deslocamento para a direita preenchendo com zeros.

a >>> b

Desloca o valor **a**, em representação binária, **b** dígitos para a direita, descartando os valores dos dígitos deslocados e inserindo **0** à esquerda.



Veja um exemplo que demonstra a utilização dos operadores bitwise de deslocamento:

```
<script>
var a = 10;
var b = 20;

console.log(a << b);
console.log(a >> b);
console.log(a >>> b);

</script>
```

3.8. Operadores lógicos

Os operadores lógicos permitem que você realize operações lógicas, que nada mais são do que operações que geram como resultado um valor booleano, ou seja, verdadeiro (true) ou falso (false). Este operador é muito útil quando é necessário tomar decisões nos scripts.

É importante ressaltar que os operandos booleanos são relacionados pelos operadores lógicos no intuito de gerar como resultado outro operando booleano.

JavaScript

Veja adiante na tabela os operadores lógicos e suas respectivas descrições:

Operadores lógicos	Descrição	Exemplo $x==50$ e $y==5$
$&&$ (e)	Quando este operador é utilizado em um teste lógico, é necessário que todas as expressões lógicas sejam avaliadas como verdadeiras para que o resultado seja verdadeiro (true).	$(x==50)&&(y<50)$ é verdadeiro .
$ $ (ou)	Quando este operador é utilizado em um teste lógico, é necessário que apenas uma expressão lógica seja avaliada como verdadeira para que o resultado seja verdadeiro (true).	$(x==50) (y==50)$ é verdadeiro .
$!$ (não)	Com este operador, a lógica da expressão é invertida, ocasionando uma negação.	$x!=y$ é verdadeiro.

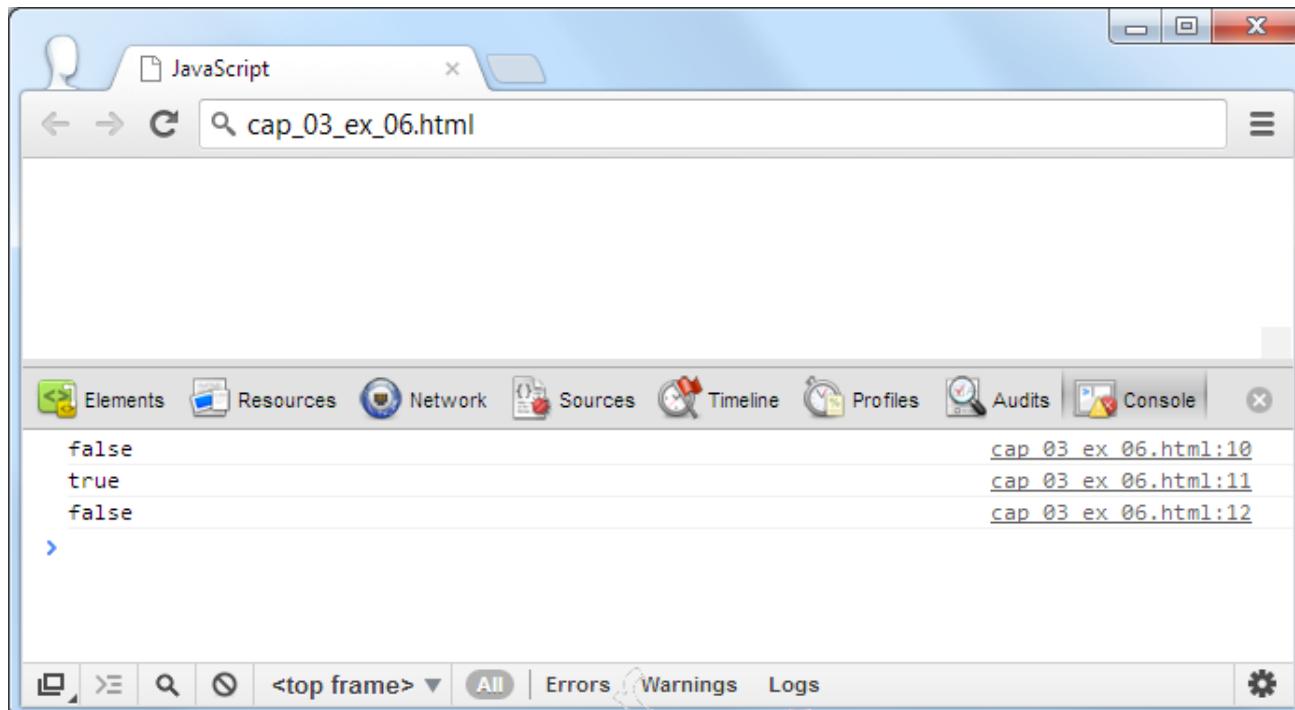
Veja um exemplo que demonstra a utilização dos operadores lógicos:

```
<script>
var a = 10;
var b = 20;

console.log(a > 15 && b > 15);
console.log(a > 15 || b > 15);
console.log(!a == 20);

</script>
```

Como resultado, a seguinte página será exibida depois que salvar o código anterior e o executá-lo:



3.8.1. Avaliação de curto-circuito

Existem algumas regras que tornam possível que você avalie uma expressão lógica para ver se ela sofrerá o que chamamos de “curto-circuito”. Isso ocorre pois as expressões são processadas da esquerda para a direita, e alguns resultados não precisam passar por certos processos, visto que sempre terão o mesmo resultado segundo a lógica. Essas regras são:

- **falso (false) && qualquer valor:** resulta sempre em **falso (false)**;
- **verdadeiro (true) || qualquer valor:** é sempre **verdadeiro (true)**.

Sendo assim, nestes dois casos específicos, qualquer valor que vier depois dos operadores **&&** ou **||** não precisam ser processados pois a resposta será sempre a mesma.

3.9. Operadores de string

Para lidar com expressões de string, você pode utilizar os operadores de comparação, já vistos neste mesmo capítulo, porém, com valores de string. Além destes, um operador que é muito utilizado com strings é o concatenador (sinal de +). Com ele, você pode unir diferentes valores de string e ter como resultado uma string que é a junção destes.

Por exemplo: usando o concatenador nas strings “Impacta” + “Tecnologia” você obtém como resultado a string “Impacta Tecnologia”.

Outra forma de concatenar strings em JavaScript é utilizando o operador +=. Neste caso, ele pega uma string armazenada em uma variável, soma com outra string e associa o valor final à mesma variável da primeira string. Veja o exemplo:

Considere uma variável **novestring** que possui o valor “Java”. Usando a variável na expressão **novestring** += “Script”, você obtém como resultado a string “JavaScript”, que fica armazenada na mesma variável **novestring**.

3.10. Operadores especiais

Além dos operadores que você já viu anteriormente neste capítulo, a linguagem JavaScript apresenta alguns operadores chamados especiais. São eles:

- Operador condicional;
- Operador separador;
- **delete**;
- **in**;
- **instanceof**;
- **new**;
- **this**;

- `typeof`;
- `void`.

3.10.1. Operador condicional

Com o operador condicional, você pode definir uma condição e dois valores diferentes, que serão utilizados com base nessa mesma condição. É o único operador da linguagem JavaScript que utiliza três operandos.

Veja a sintaxe:

```
condição ? val1 : val2
```

Onde:

- **condição**: é uma condição que será avaliada pela expressão;
- **val1**: valor que o operador assume caso a condição definida seja verdadeira;
- **val2**: valor que o operador assume caso a condição não seja verdadeira.

O operador condicional pode ser usado em qualquer lugar onde possa ser usado um operador padrão.

Veja um exemplo que demonstra a utilização do operador condicional:

```
<script>

var resultado = (new Date().getDay() > 3) ? 'Maior que três' : 'Menor que três';

console.log(resultado);

</script>
```

3.10.2. Operador separador

Este operador é muito comum quando utilizado dentro de um loop **for**. Ele retorna sempre o valor do seu segundo operando, depois de calcular ambos, funcionando assim como um loop. Dentro do loop **for**, ele permite a atualização de múltiplas variáveis.

Veja um exemplo que demonstra a utilização do operador separador:

```
<script>

var a = [1,2,3,4];
var b;
for ( i = 0; i < a.length, b = a[i]; i++ ) {
    console.log(a[i], b);
}

</script>
```

3.10.3. delete

Quando você desejar apagar um objeto, a propriedade de um objeto ou um elemento em um índice específico, utilize o operador **delete**. Ele vai retornar o valor true (verdadeiro) caso seja realmente possível realizar a ação, ou false (falso) caso não seja permitido.

Ele possui as seguintes sintaxes:

```
delete nomeObjeto;

delete nomeObjeto.propriedade;

delete nomeObjeto[índice];

delete propriedade;
```

Em que:

- **nomeObjeto**: o nome do objeto que você deseja inserir na expressão;

- **propriedade**: qualquer propriedade existente;
- **índice**: um número que representa a localização de um elemento em um array.

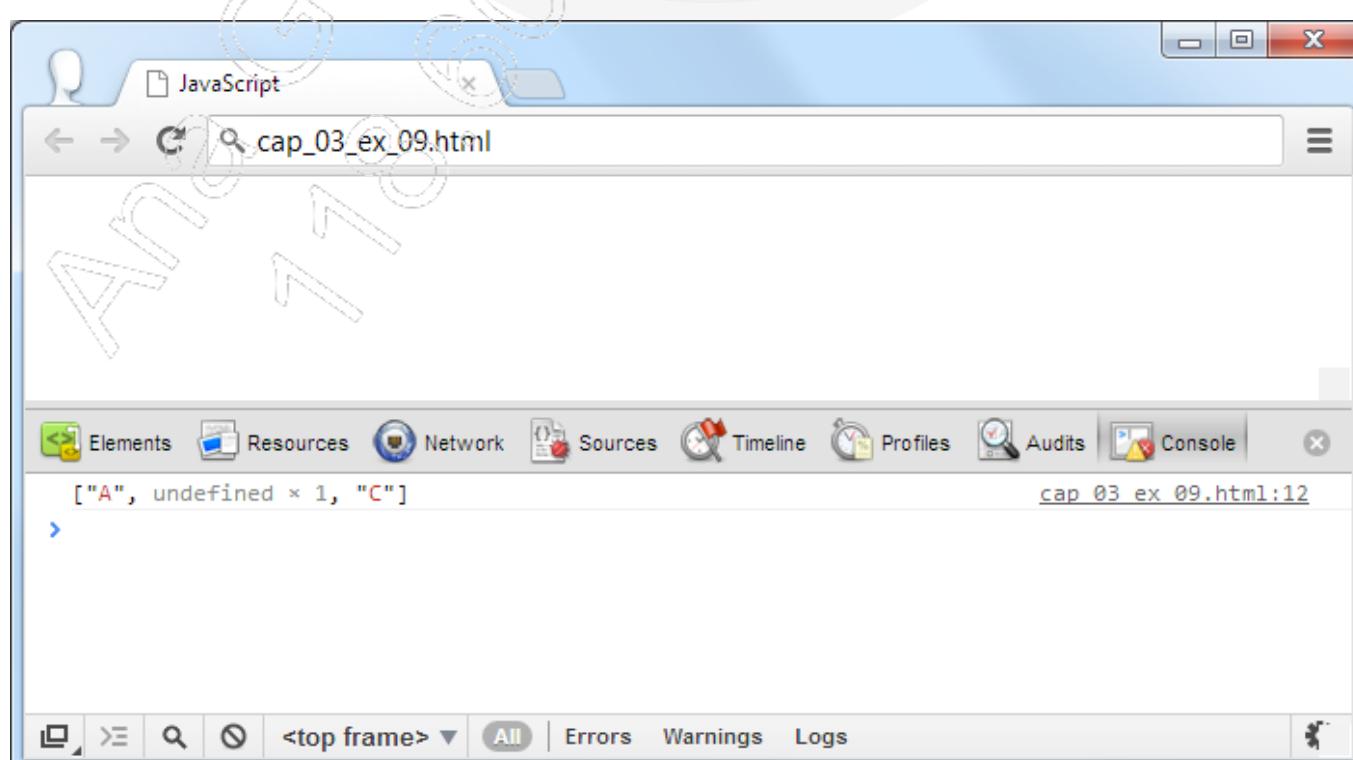
ATENÇÃO! A última sintaxe pode ser utilizada apenas para deletar uma propriedade de um objeto dentro de uma declaração **with**.

Variáveis declaradas com a declaração **var** não podem ser apagadas com o operador **delete**, apenas as declaradas implicitamente.

Veja um exemplo que demonstra a utilização do operador **delete**:

```
<script>  
  
var items = ['A', 'B', 'C'];  
  
delete items[1];  
  
console.log(items);  
  
</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



3.10.4. in

Este operador indica se certa propriedade encontra-se em um objeto definido na expressão. Veja como o operador **in** funciona:

```
propNome in nomeObjeto
```

Em que:

- **propNome**: pode ser uma string que representa um nome de propriedade ou mesmo um índice de array;
- **nomeObjeto**: é o nome do objeto a ser utilizado.

Veja um exemplo que demonstra a utilização do operador **in**:

```
<script>

var Pessoa = {
    nome:'João',
    idade:23
};

for(propriedade in Pessoa){
    console.log(propriedade);
}

</script>
```

3.10.5. instanceof

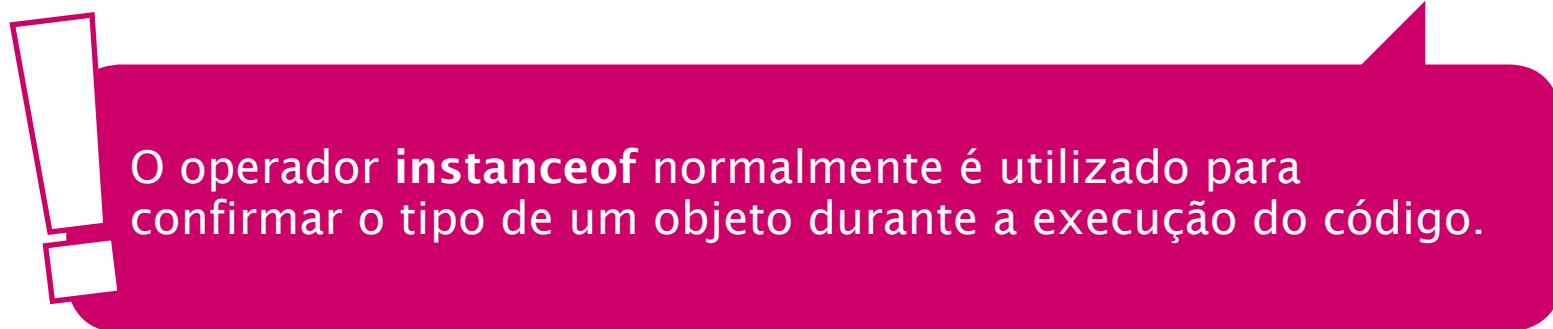
Para verificar se certo objeto faz parte de um tipo definido de objeto, você deve utilizar o operador **instanceof**. Veja sua sintaxe:

```
nomeObjeto instanceof tipoObjeto
```

Em que:

- **nomeObjeto**: nome do objeto a ser analisado;

- **tipoObjeto**: tipo do objeto (Array, Date, etc.) com o qual nomeObjeto será comparado.



Veja um exemplo que demonstra a utilização do operador **instanceof**:

```
<script>

var frutas = ['Banana', 'Maça', 'Abacaxi'];

console.log(frutas instanceof Array);

</script>
```

3.10.6. new

O operador **new** tem a função de criar uma ocorrência seja de um dos tipos predefinidos de objetos, seja de tipos criados pelo usuário. Também é possível usar o operador **new** no servidor, com DbPool, Lock, File e SendMail. Veja a sintaxe:

```
var nomeObjeto = new tipoObjeto([parametro1, parametro2, ..., parametroN]);
```

Veja um exemplo que demonstra a utilização do operador **new**:

```
<script>

var navegadores = new Array('Chrome', 'Firefox', 'Safari');

</script>
```

3.10.7. this

Para acessar o objeto atual, utilize o operador **this**, que normalmente se refere ao objeto de chamada em um método. Veja a sintaxe:

```
this["nomePropriedade"]
```

```
this.nomePropriedade
```

Veja um exemplo que demonstra a utilização do operador **this**:

```
<script>

var Pessoa = function(nomeDaPessoa) {

    this.nome = nomeDaPessoa;

}

var a = new Pessoa('Massa');
var b = new Pessoa('Renan');

console.log(a, b);
</script>
```

3.10.8. typeof

O operador **typeof** fornece em forma de string o tipo do operando inserido, que pode ser uma string, variável, palavra-chave ou objeto.

Existem duas sintaxes possíveis para o operador **typeof**:

```
typeof operando
```

```
typeof (operando)
```

Veja um exemplo que demonstra a utilização do operador **typeof**:

```
<script>

var a = 1;
var b = "HTML5";
var c = true;
var d = {};

console.log(typeof a);
console.log(typeof b);
console.log(typeof c);
console.log(typeof d);

</script>
```

3.10.9. void

O operador **void** analisa uma expressão sem retornar um valor. O operador **void** é utilizado especialmente para definir uma expressão como um link de hipertexto onde a expressão seja avaliada mas não seja carregada no lugar do documento atual.

Veja a sintaxe:

```
void (expressão)
```

```
void expressão
```

Recomenda-se a utilização dos parênteses na sintaxe para facilitar a organização do código.

Veja um exemplo que demonstra a utilização do operador **void**:

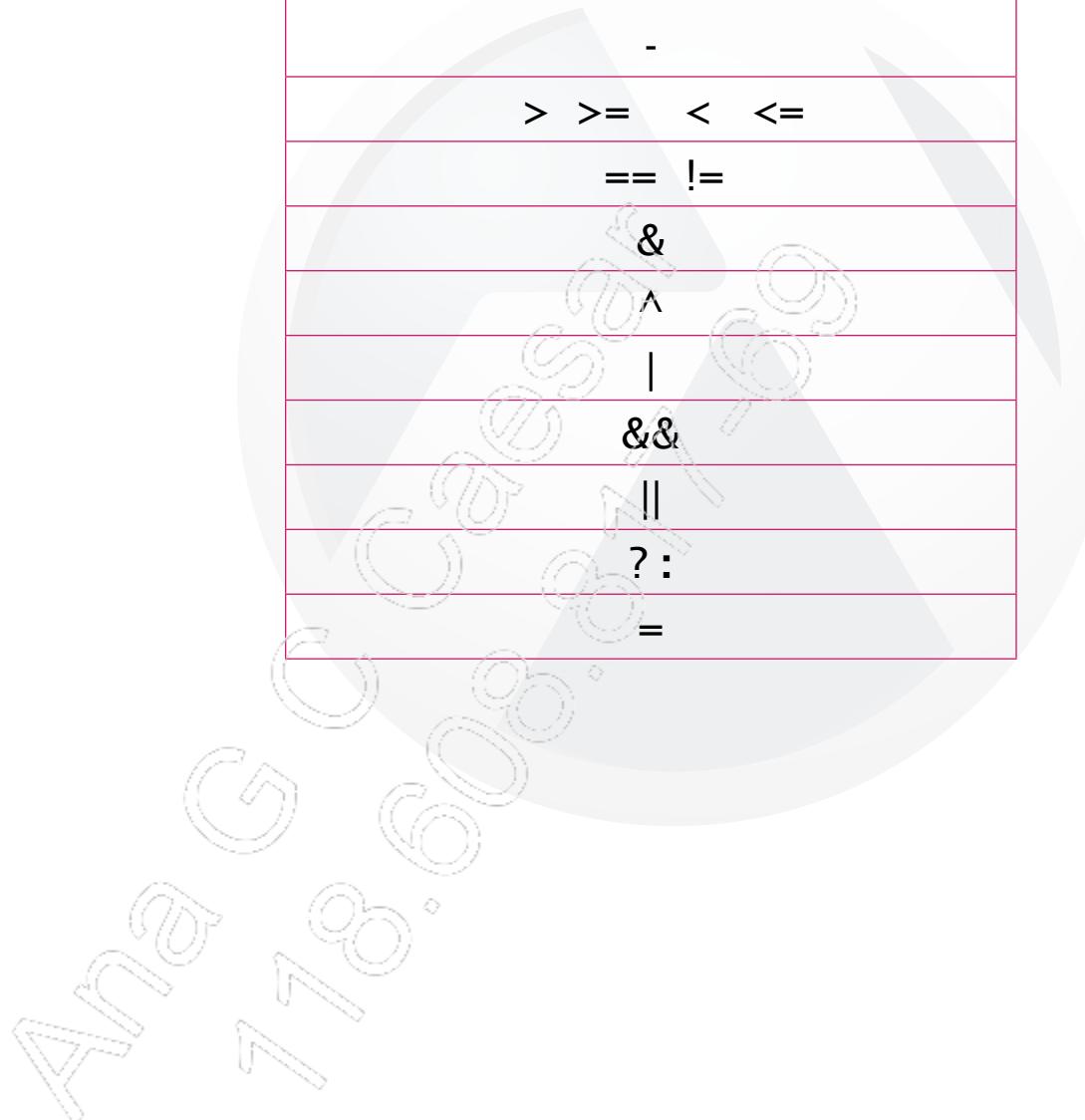
```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7
8  <body>
9
10 <p><a href="javascript:void(0);">Não faz nada</a></p>
11
12 <p><a href="javascript:void(document.body.style.backgroundColor='black');">
13     Mudar a cor do fundo da página para preto</a></p>
14
15 <p><a href="javascript:void(document.body.style.backgroundColor='white');">
16     Mudar a cor do fundo da página para branco</a></p>
17
18 </body>
19 </html>
```

3.11. Precedência dos operadores

A ordem de resolução para uma expressão que possui vários operadores e operandos é definida com base na precedência dos operadores. Portanto, deve ser efetuada a operação que apresenta maior precedência em primeiro lugar, e assim por diante.

A tabela adiante mostra a ordem de prioridade dos operadores:

Ordem de prioridade dos operadores
() [].
++ -- ~
!
* /
%
+
-
> >= < <=
== !=
&
^
&&
? :
=



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Operadores são utilizados em praticamente todas as linguagens de programação existentes no mercado. Os operadores permitem realizar operações e cálculos e são empregados especialmente durante o desenvolvimento dos programas;
- Qualquer unidade de código que é válida e que resulta em um valor é chamada de expressão. Considere que existam dois tipos de expressões: na primeira, o valor resultante é atribuído a uma variável; na segunda, você tem apenas o valor que resulta dela;
- Os tipos de operadores disponíveis para a linguagem de programação JavaScript são os seguintes: operadores de atribuição, operadores de comparação, operadores aritméticos, operadores bit a bit, operadores de string, operadores especiais;
- A linguagem JavaScript apresenta alguns operadores chamados especiais. São eles: operador condicional, operador separador, delete, in, instanceof, new, this, typeof, void.

3

Operadores

Teste seus conhecimentos

Ana G. C.
778.6008.



IMPACTA
EDITORA

1. Assinale o tipo de expressão incorreto.

- a) Aritmética
- b) String
- c) Condicional
- d) Lógica
- e) Objeto

2. Qual dos operadores a seguir não é caracterizado como um operador especial?

- a) Delete
- b) In
- c) Operador aritmético
- d) New
- e) Typeof

3. O operador instanceof normalmente é utilizado quando:

- a) Deseja confirmar o tipo de um objeto durante a execução do código.
- b) Deseja confirmar o nome do objeto durante a execução do código.
- c) Deseja confirmar o tipo e nome do objeto durante a execução do código.
- d) Deseja confirmar a existência de um objeto durante a execução do código.
- e) Nenhuma das alternativas anteriores está correta.

4. O operador void analisa uma expressão e retorna:

- a) Não retorna valor.
- b) Um inteiro.
- c) Um booleano.
- d) Uma função.
- e) Um objeto.

5. Qual dos operadores a seguir não é caracterizado como um operador de atribuição?

- a) =
- b) ==
- c) -=
- d) +=
- e) %=



3

Operadores Mãos à obra!

Ana G. Cesar
778.6008.



IMPACTA
EDITORA

Laboratório 1

A - Realizando opções aritméticas

1. Crie um novo arquivo de texto e renomeie com a extensão **.html**;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD**, **BODY**;
4. Inclua dentro da tag **HEAD** a tag **<script></script>**;
5. Dentro da tag **<script>**, desenvolva o seguinte sistema:
 - Uma concessionária em 2011 possuía 3 produtos:
 - Carros que custam a partir de R\$ 20.000,00;
 - Motos que custam a partir de R\$ 3.000,00;
 - Caminhões que custam a partir de R\$ 80.000,00.
 - O lucro na venda de:
 - Carros é 20%;
 - Motos é 30%;
 - Caminhões é 40%.
 - A concessionária vendeu em 2011:
 - Carros: 100;
 - Motos: 80;
 - Caminhões: 60.

6. Calcule o lucro da concessionária com a venda dos três produtos e imprima na tela:

- O valor deve ser arredado.

Laboratório 2

A – Incrementando o código

Utilizando o laboratório anterior, faça a seguinte incrementação:

1. Verifique se o lucro da venda com carros foi maior que R\$ **600.000,00** no ano:

- Imprimir na tela se sim ou não.

2. Verifique se o lucro da venda de motos somado com caminhões foi maior que R\$ **1.000.000,00**:

- Imprimir na tela se sim ou não.

3. Verifique se o lucro da venda de motos foi maior que R\$ **70.000,00** e menor que R\$ **71.000,00**:

- Imprimir na tela se sim ou não.

Declaracões

4

- ✓ Estruturas condicionais;
- ✓ Estruturas para loops.

Ana Góes
Cesar
Góes
778.6000



IMPACTA
EDITORA

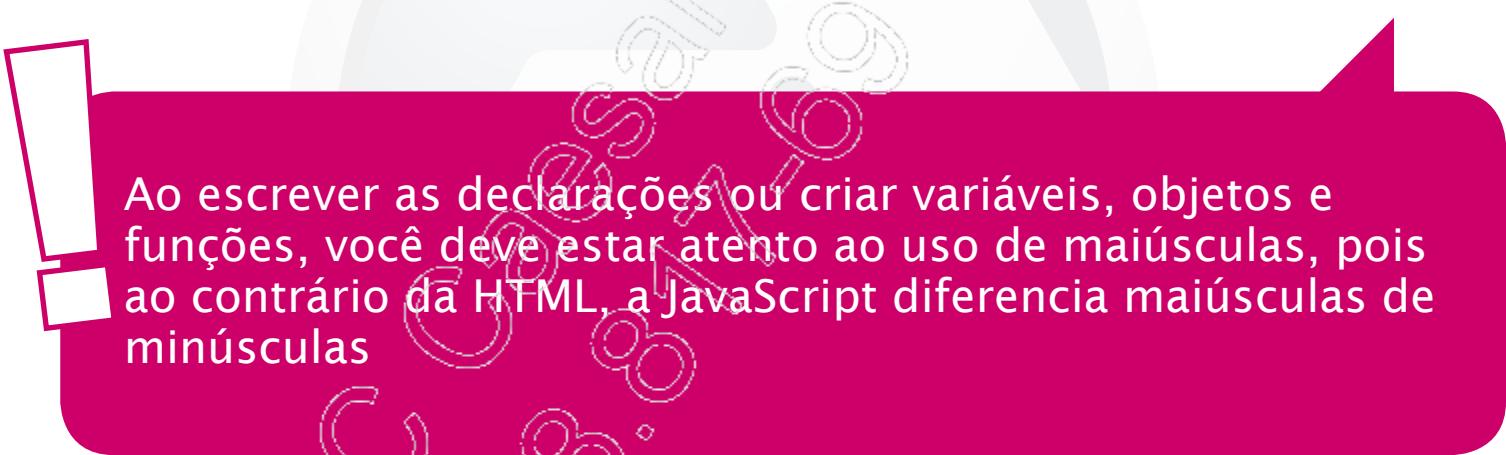
4.1. Introdução

A JavaScript é responsável por grande parte dos recursos interativos nas páginas HTML. Para incorporação desses recursos, faz uso de um grupo de declarações com características e funcionalidades específicas que serão apresentadas no decorrer deste capítulo.

O controle do fluxo de um programa é feito por bloco de declarações, ou seja, uma sequência de declarações entre chaves ({}).

Uma das características dessas declarações em bloco é o aninhamento. Por meio do aninhamento é possível agrupar blocos de declarações em outro bloco.

As estruturas condicionais e as estruturas para loop constituem blocos de declarações, que serão apresentados a seguir.



Ao escrever as declarações ou criar variáveis, objetos e funções, você deve estar atento ao uso de maiúsculas, pois ao contrário da HTML, a JavaScript diferencia maiúsculas de minúsculas

4.1.1. Declarações em JavaScript

Na tabela a seguir apresentamos as declarações para estruturas JavaScript e suas respectivas finalidades:

Declaração	Finalidade
var	Define uma variável.
function	Define uma função.
return	Retorna um valor.
if/else	Cria uma estrutura condicional.
switch	Cria uma estrutura condicional.

Declaração	Finalidade
case	Usada na estrutura condicional switch.
break	Usada na estrutura condicional switch.
default	Usada na estrutura condicional switch.
for	Cria um loop.
continue	Reinicia um loop.
while	Cria uma estrutura de repetição.
do/while	Cria uma estrutura de repetição.
for/in	Cria um loop em objeto.
throw	Sinaliza erros.
try/catch/finally	Trata erros.
with	Altera escopo.
;	É uma declaração vazia.

4.2. Estruturas condicionais

Para que você decida qual ação será realizada em uma parte específica de um programa, é necessário solicitar que o programa avalie uma determinada expressão. Se esta expressão for avaliada como verdadeira, uma sequência de declarações será executada.

Então, as estruturas de condição a serem consideradas são:

- **if/else;**
- **switch/case.**

4.2.1. Declaração if/else

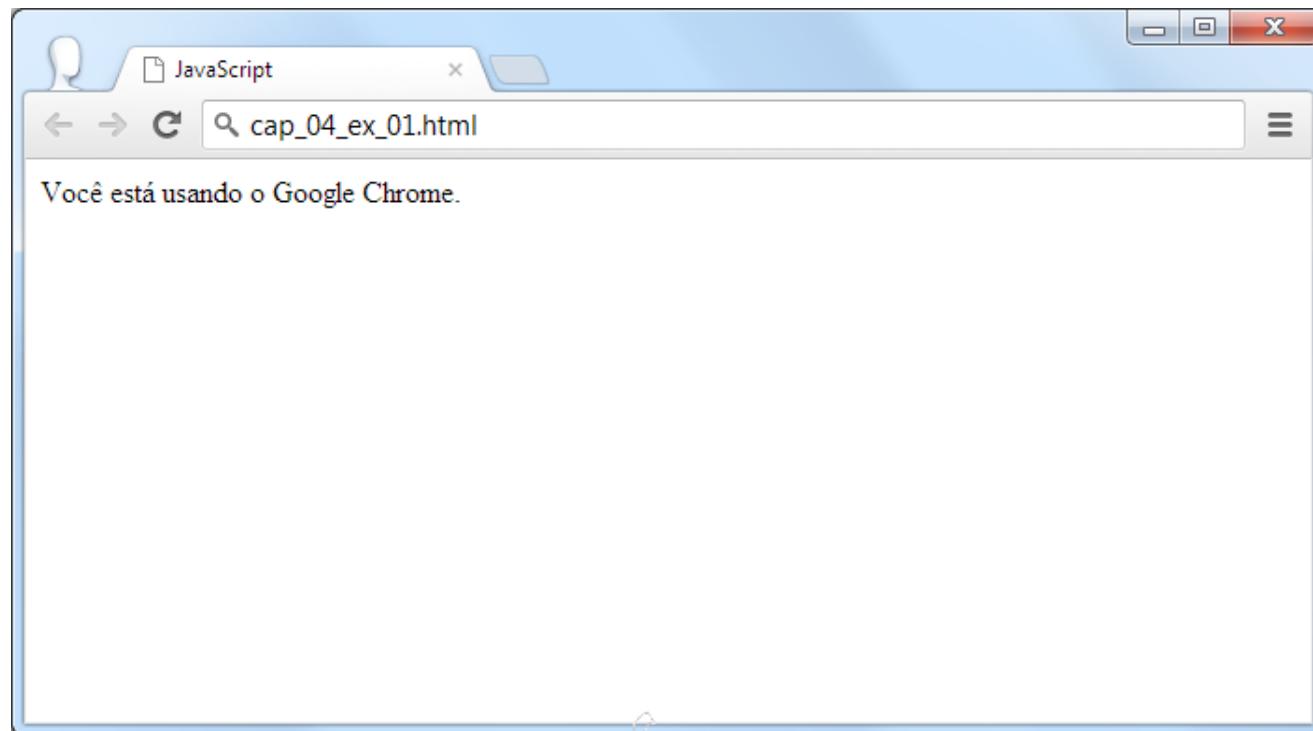
A instrução **if** avalia somente expressões com resultados booleanos. Caso a avaliação tenha um resultado verdadeiro, será executado um bloco de declarações localizado entre chaves. Caso contrário, ou seja, se o resultado for falso, o bloco de declarações não será executado ou, ainda, pode ser executado outro bloco de declarações. Veja a sintaxe a seguir:

```
if (Teste Condicional) {  
    comando  
}
```

Agora veja um exemplo da utilização de **if**:

```
<script>  
  
    if(navigator.userAgent.indexOf('Chrome') > -1){  
        document.write("Você está usando o Google Chrome.");  
    }  
  
</script>
```

Ao salvar e executar o código anterior, você visualizará a seguinte página:



! Ao utilizar `if`, o emprego de chaves não é obrigatório se o bloco de declarações possuir uma única instrução. Mesmo assim, é melhor que você use as chaves, pois proporcionam uma legibilidade melhor. Por esse mesmo motivo, recomenda-se que as instruções `if/else` sejam moderadamente aninhadas.

A cláusula `else` deve ser utilizada ao executar declarações, cuja condição é falsa. Para isso:

1. Digite `else` após as instruções da condição verdadeira;

JavaScript

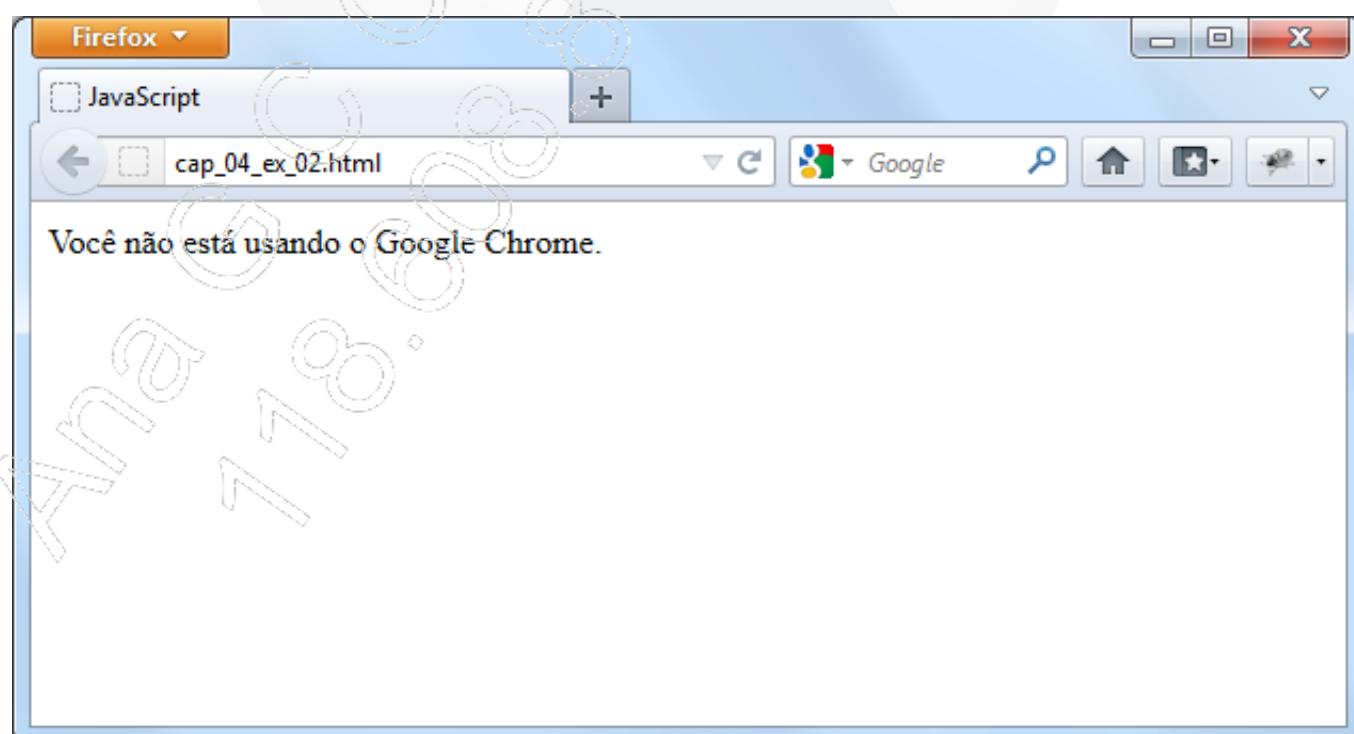
2. A seguir, na linha abaixo, insira o bloco de instruções a ser executado. Veja a sintaxe:

```
if (Teste Condicional) {  
    comando 1  
}  
else{  
    comando 2  
}
```

Veja um exemplo da utilização das instruções **if/else**:

```
<script>  
  
    if(navigator.userAgent.indexOf('Chrome') > -1){  
        document.write("Você está usando o Google Chrome.");  
    }else{  
        document.write("Você não está usando o Google Chrome.");  
    }  
  
</script>
```

Ao salvar e executar o código acima, você visualizará a seguinte página:



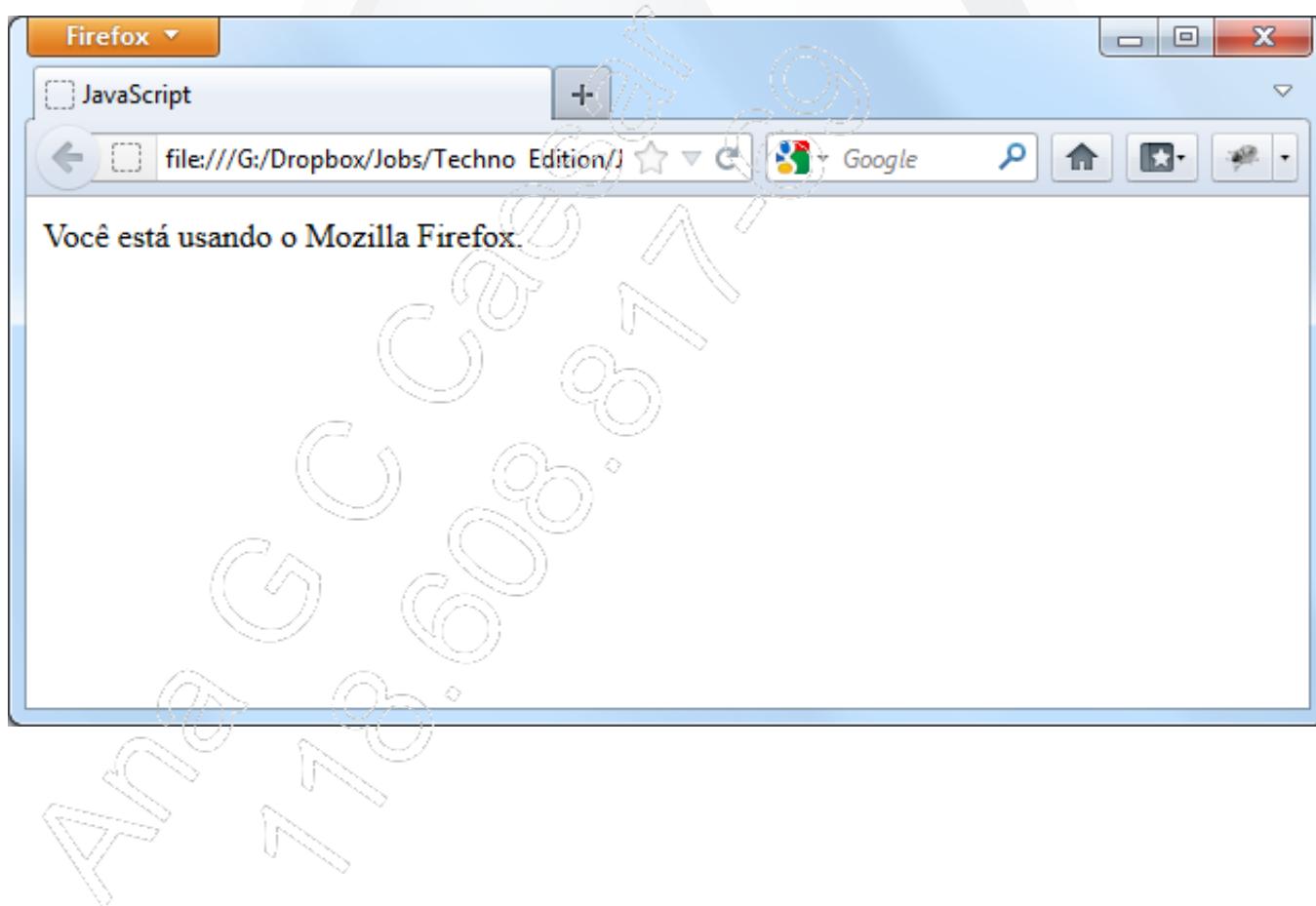
Se você quiser que múltiplas condições sejam testadas em sequência, você deve compor as declarações usando **else/if**. Veja:

```
<script>

if(navigator.userAgent.indexOf('Chrome') > -1){
    document.write("Você está usando o Google Chrome.");
} else if(navigator.userAgent.indexOf('Firefox') > -1){
    document.write("Você está usando o Mozilla Firefox.");
} else{
    document.write("Você não está usando o Google Chrome.");
}

</script>
```

Você visualizará a seguinte página ao usar o código anterior:



4.2.2. Declaração switch/case

Você também pode adotar a instrução **switch** para simular o uso de diversas instruções **if**. Veja sintaxe de **switch/case**:

```
switch (expressão) {  
    case valor1:  
        instruções;  
        break;  
    case valor2:  
        instruções;  
        break;  
}
```

Switch verifica uma relação de igualdade. Deste modo, ao executá-lo, o programa pode encontrar uma declaração **break**, a qual indica que a instrução seguinte ao bloco **switch** deve ser executada. Portanto, a instrução **case** será executada enquanto **switch** não for finalizado ou até encontrar uma declaração **break**, caso o código-fonte não a possua.

Assim, enquanto é executado, é possível que o programa passe de uma instrução **case** para outra. Este procedimento é conhecido como passagem completa, na qual todas as instruções **case** são executadas até o final de **switch**.

Veja um exemplo de **switch/case**:

```
<script>

    var semana = 3, resultado;

    switch(semana) {
        case 1:
            resultado = 'Domingo';
            break;
        case 2:
            resultado = 'Segunda';
            break;
        case 3:
            resultado = 'Terça';
            break;
        case 4:
            resultado = 'Quarta';
            break;
        case 5:
            resultado = 'Quinta';
            break;
        case 6:
            resultado = 'Sexta';
            break;
        case 7:
            resultado = 'Sábado';
            break;
    }

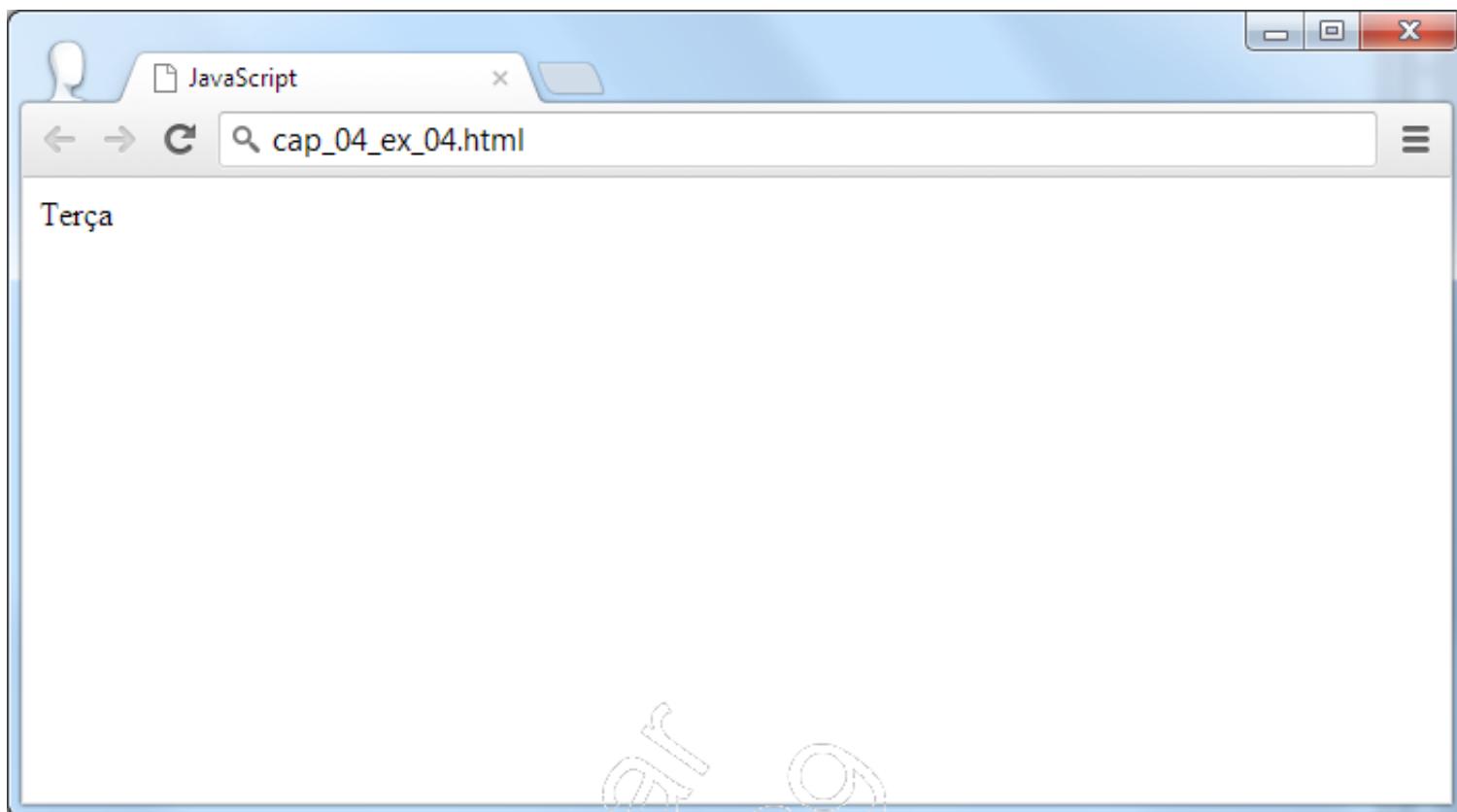
    document.write(resultado);

</script>
```

Ana G.
778.

JavaScript

Você visualizará a seguinte página ao usar o código anterior:



No exemplo, a declaração **break** é usada para separação do que for restritamente executado em cada **case**, bem como para finalizar o **case** anterior.

Você ainda pode usar a declaração **default** para concluir a utilização de **switch/case**. Esta declaração será executada sempre que um valor **case** correspondente não seja identificado pelo valor assumido pela variável não.

```
<script>

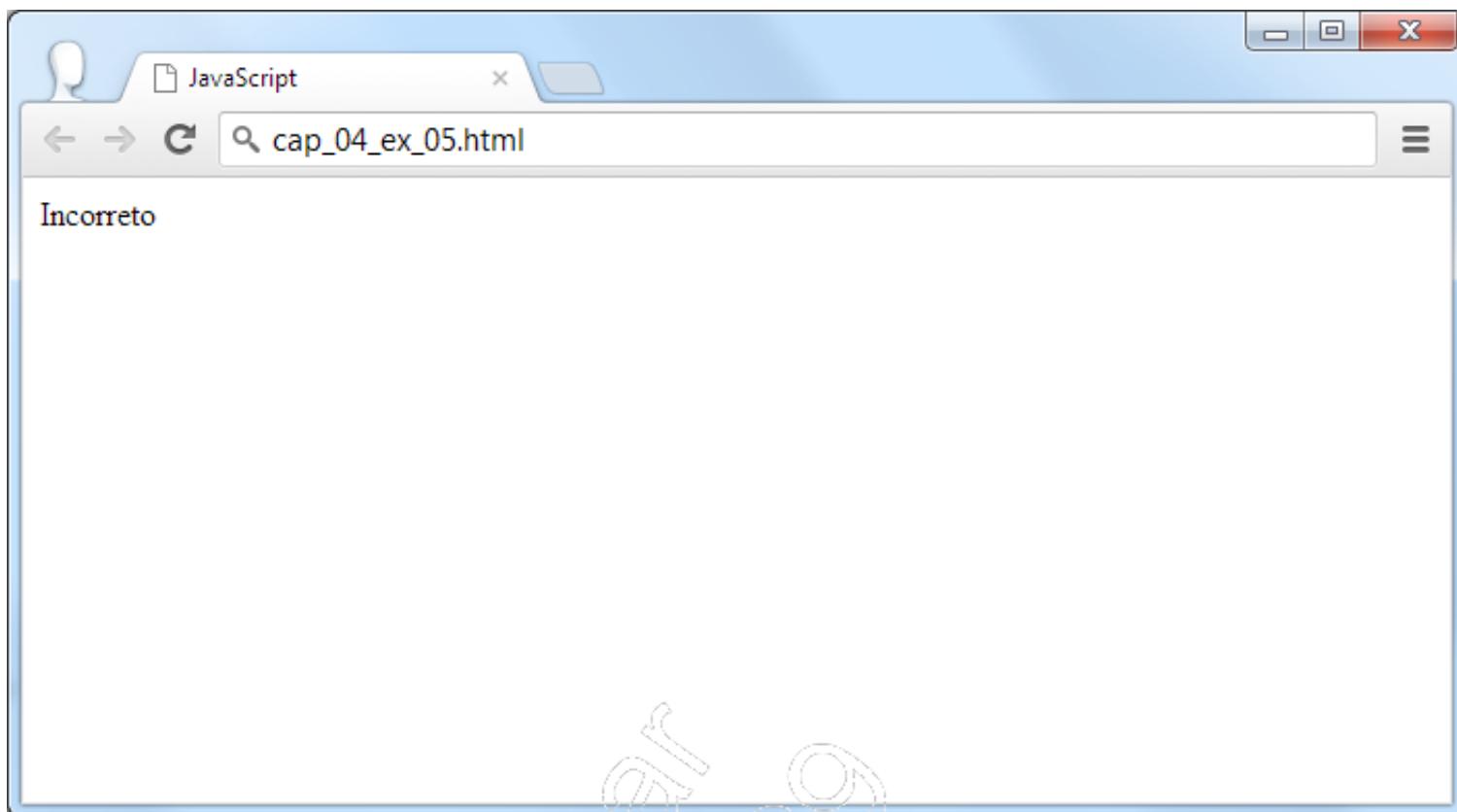
    var semana = 8, resultado;

    switch(semana) {
        case 1:
            resultado = 'Domingo';
            break;
        case 2:
            resultado = 'Segunda';
            break;
        case 3:
            resultado = 'Terça';
            break;
        case 4:
            resultado = 'Quarta';
            break;
        case 5:
            resultado = 'Quinta';
            break;
        case 6:
            resultado = 'Sexta';
            break;
        case 7:
            resultado = 'Sábado';
            break;
        default:
            resultado = 'Incorreto';
            break;
    }

    document.write(resultado);

</script>
```

Você visualizará a seguinte página ao usar o código anterior:



4.3. Estruturas para loops

As estruturas para loops são declarações usadas para executar um bloco de códigos repetidas vezes, até se atingir certo número de execuções ou até que uma determinada condição seja verdadeira.

Em JavaScript, as estruturas para loops são:

- **while;**
- **do/while;**
- **for;**
- **for/in.**

Também é possível usar **break** e **continue** nas estruturas para loop.

4.3.1. Declaração while

Quando você não souber a quantidade de vezes que um bloco de instruções deve ser repetido, use a declaração para repetição **while**.

Com a declaração **while**, uma instrução é continuamente executada até que uma condição seja verdadeira. A condição analisada retorna um valor booleano.

A declaração **while** não será executada se uma condição for falsa na primeira vez em que é verificada. Neste caso, será executada a instrução seguinte ao loop.

Observe a sintaxe:

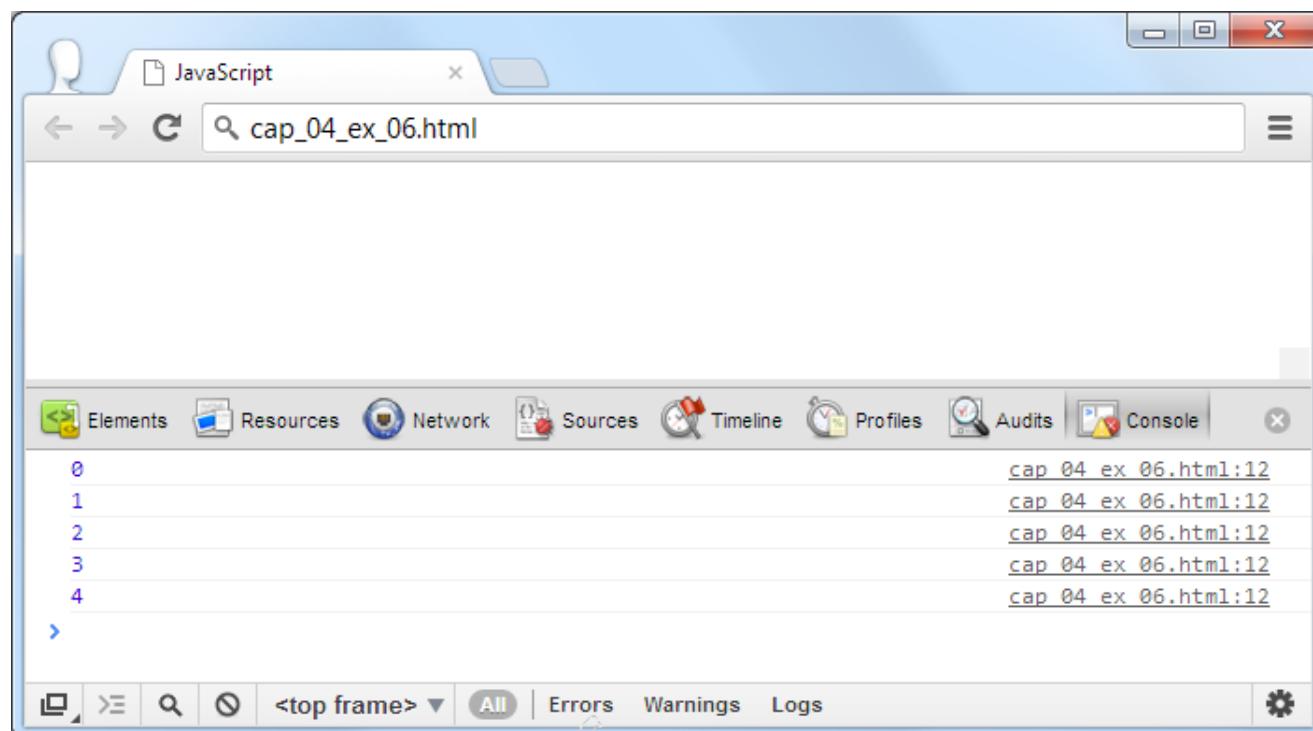
```
while (teste condicional) {  
    declarações // será executado enquanto teste condicional  
    = true  
}
```

Pela sintaxe, você deve ter notado que somente se a condição for verdadeira o corpo da estrutura será executado.

Veja o exemplo a seguir:

```
<script>  
var x = 0;  
while(x < 5){  
    console.log(x);  
    x++;  
}  
</script>
```

Você visualizará a seguinte página ao usar o código anterior:



4.3.2 Declaração do/while

Apesar de ter fundamentalmente o mesmo funcionamento de **while**, com **do/while**, independentemente da condição ser verdadeira ou não, as declarações são executadas pelo menos uma vez. Observe a sintaxe:

```
do {
    declaração
}
while (condição)
```

Agora, veja o exemplo:

```
<script>

    var x = 0;

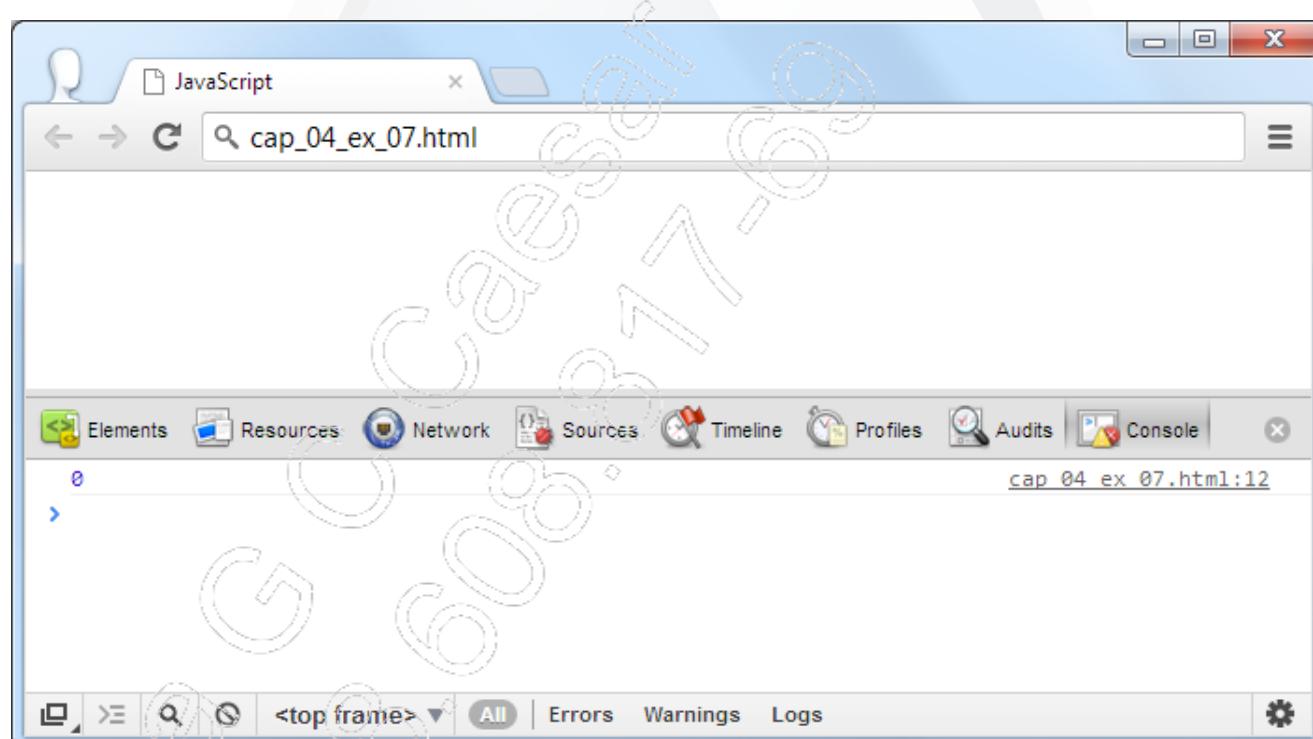
    do {

        console.log(x);
        x++;

    } while(x < 0);

</script>
```

Você visualizará a seguinte página ao usar o código anterior:



4.3.3. Declaração for

Se você souber exatamente a quantidade de vezes que deseja repetir um bloco de instruções, use a declaração **for**.

A declaração **for** é formada pelo corpo do loop, bem como pelas seguintes partes:

- **Declaração e inicialização de variáveis**

Se, na primeira parte de **for**, houver uma ou mais variáveis, estas poderão ser declaradas ou inicializadas. Para isso, você deve inseri-las entre parênteses, após a palavra-chave **for**. Além disso, use vírgulas para separar variáveis do mesmo tipo. Veja:



A declaração e inicialização de variáveis em **for** sempre ocorrem antes das outras declarações. Além disso, ocorrem uma única vez no loop, diferentemente da expressão de iteração e do teste booleano, que são executados a cada loop.

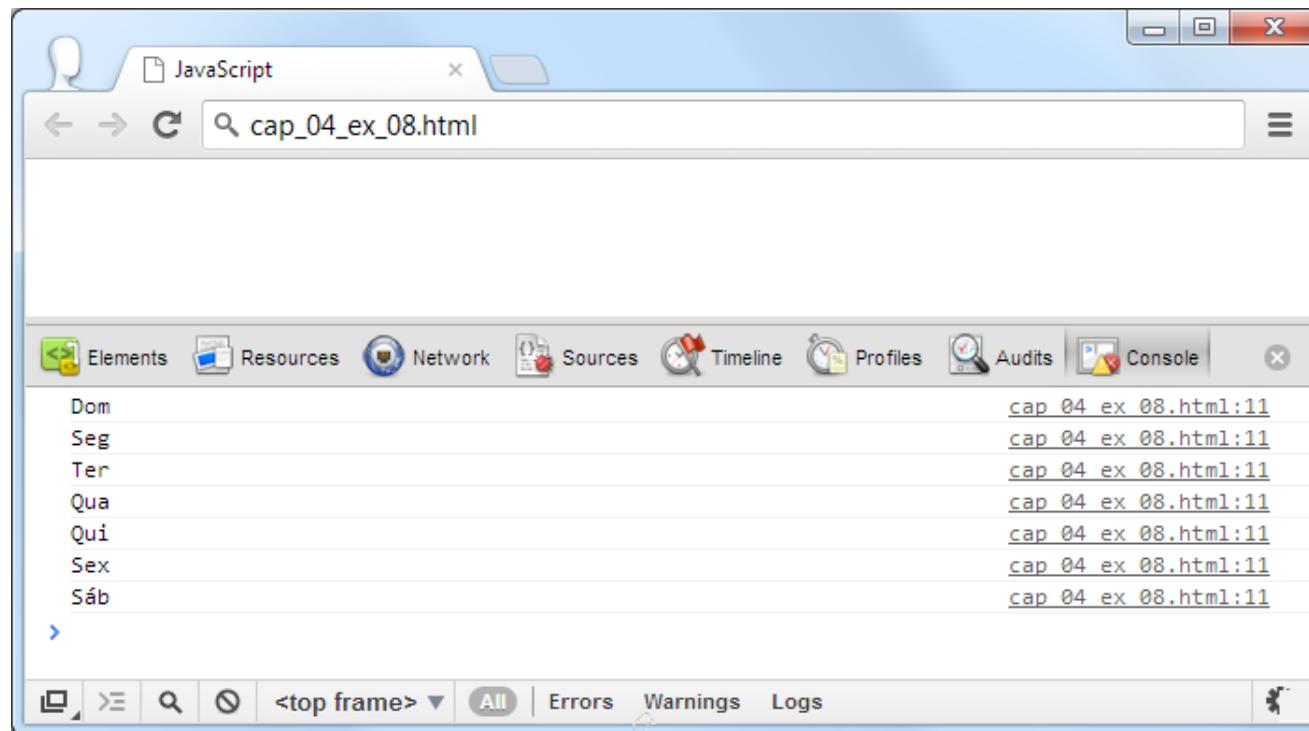
- **Expressão condicional**

A expressão condicional se localiza na segunda parte da instrução do loop. Ela é um teste e, quando executada, retorna um valor booleano. Por isso você deve especificar uma expressão lógica como expressão condicional. Porém seja cauteloso com códigos que usam várias expressões lógicas (a complexidade pode ser uma característica de uma expressão lógica).

Veja o exemplo de código seguinte, cuja expressão condicional é válida:

```
for(var i = 0, diaAtual; i < dias.length, diaAtual = dias[i]; i++) {
    console.log(diaAtual);
}
```

Você visualizará a seguinte página ao usar o código anterior:



Segundo a sintaxe da declaração **for**, podemos ter apenas uma expressão de teste. Assim, se você utilizar dois testes booleanos separados por vírgulas, a instrução não será executada e um erro ocorrerá, ou seja, a expressão condicional não será válida. Veja:

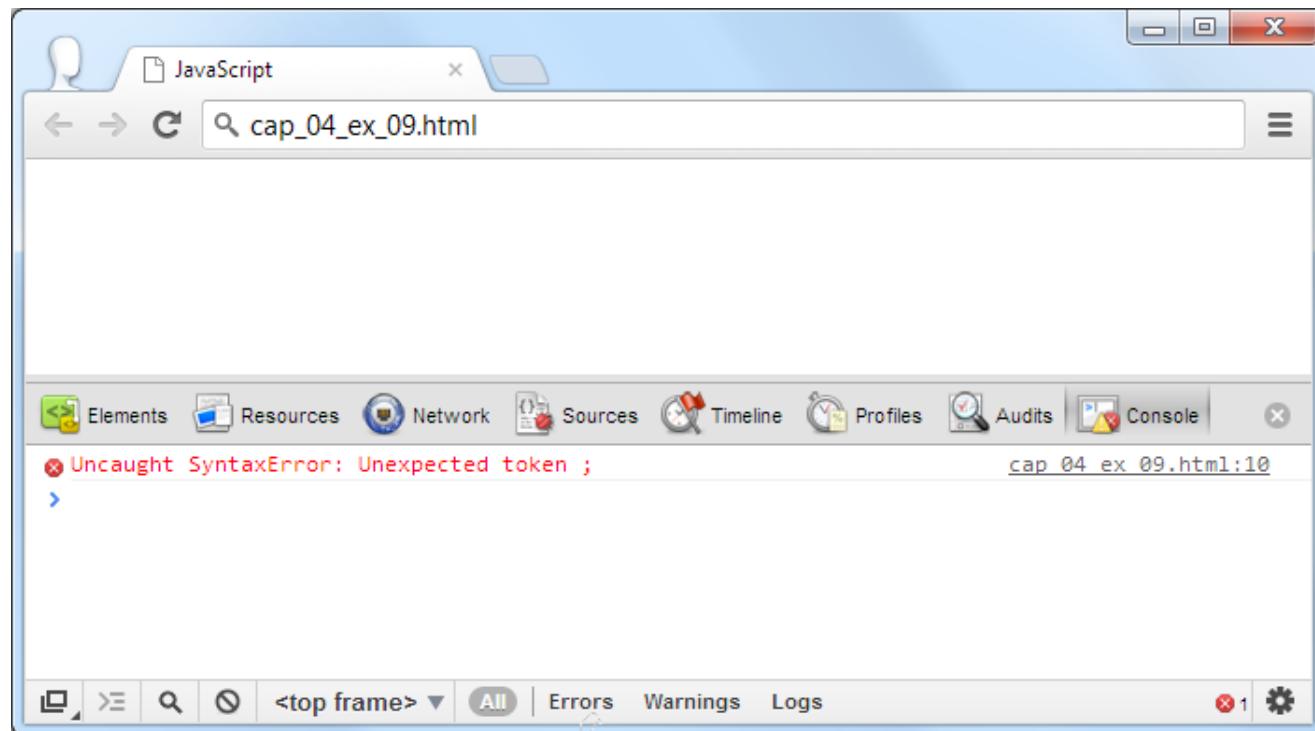
```
<script>

var dias = ['Dom', 'Seg', 'Ter', 'Qua', 'Qui', 'Sex', 'Sáb'];

for(var i = 0, diaAtual; i < dias.length, diaAtual = dias[i]; diaAtual != 'Dom'; i++){
    console.log(diaAtual);
}

</script>
```

Após salvar e executar o código anterior, a seguinte página será exibida:



- **Expressão de iteração**

Embora seja a última a ser executada na instrução do loop **for**, esta expressão é sempre processada após a execução do corpo do loop. É nela que você especificará o que deve ocorrer assim que o corpo do loop de repetição **for** executado.

Observe a sintaxe completa da instrução **for**:

```
for (declaração e inicialização de variável; condição; iteração) {  
    instrução do corpo do loop for;  
}
```

Agora veja o exemplo:

```
<script>  
  
var dias = ['Dom', 'Seg', 'Ter', 'Qua', 'Qui', 'Sex', 'Sáb'];  
  
for(var i = 0, diaAtual; i < dias.length, diaAtual = dias[i]; i++){  
    console.log(diaAtual);  
}  
  
</script>
```

Se você optar por não declarar uma dessas três partes de **for**, o que não é indicado, pode ter como consequências:

- Loop infinito;
- Loop de repetição **for** atuando como um loop de repetição **while** caso não apresente seções de declaração e inicialização.

Veja o próximo exemplo:

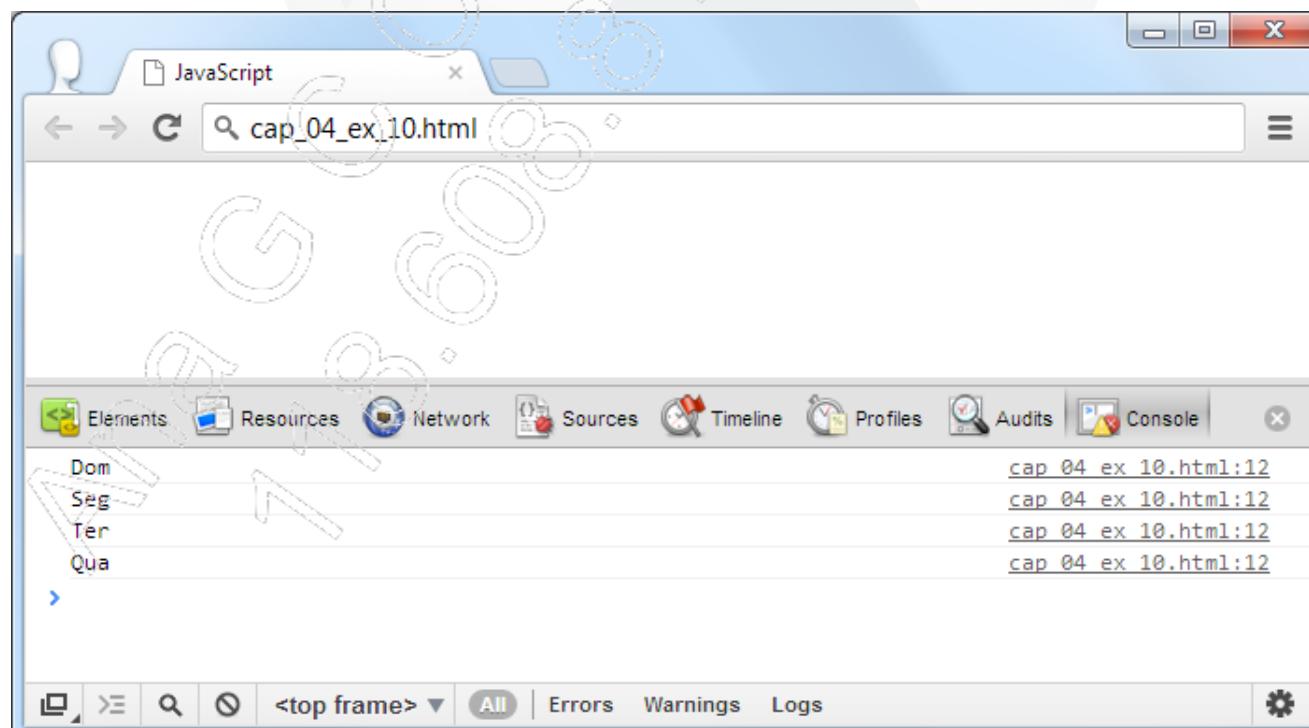
```
<script>

var dias = ['Dom', 'Seg', 'Ter', 'Qua', 'Qui', 'Sex', 'Sáb'];

for(var i = 0, diaAtual; i < dias.length, diaAtual = dias[i]; i++){
    if(diaAtual == 'Qui') break;
    console.log(diaAtual);
}

</script>
```

Você visualizará a seguinte página ao usar o código anterior:



Repare também que as seções da instrução **for** são independentes e por isso não precisam operar sobre as mesmas variáveis.

JavaScript

Uma expressão de iteração não tem necessariamente como função a configuração ou incrementação de algo. Observe o exemplo a seguir:

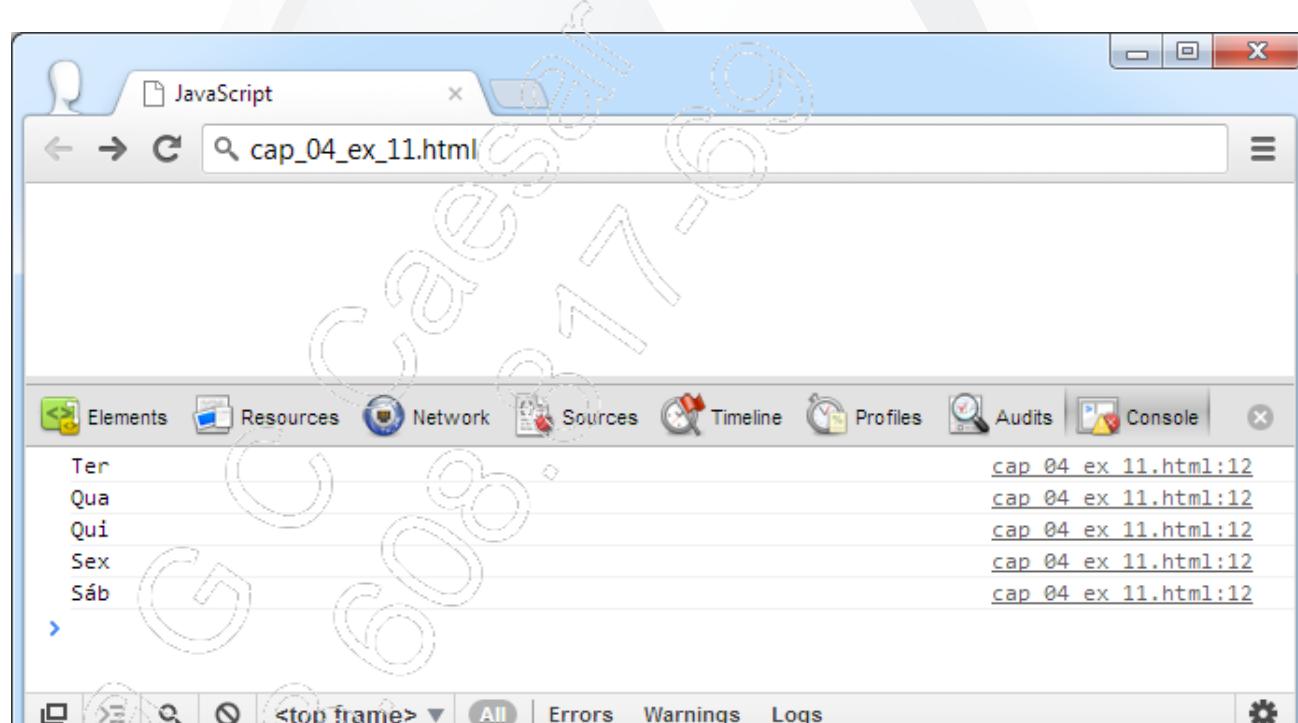
```
<script>

var dias = ['Dom', 'Seg', 'Ter', 'Qua', 'Qui', 'Sex', 'Sáb'];

var i=2,len=dias.length;
for (; i<len; ){
    console.log(dias[i]);
    i++;
}

</script>
```

Você visualizará a seguinte página ao usar o código anterior:



4.3.4. Declaração for/in

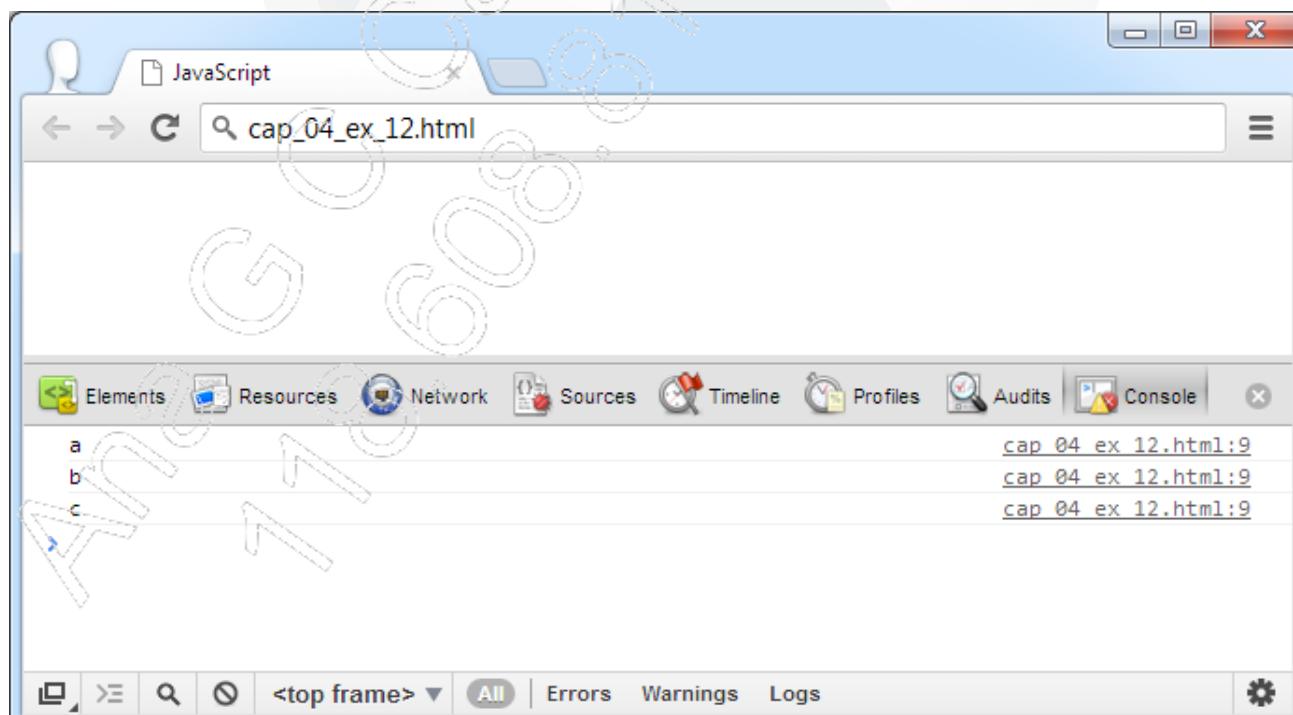
A declaração **for/in** repete uma variável especificada em todas as propriedades de um objeto. Para cada uma dessas propriedades, a JavaScript executa as instruções especificadas, ou seja, um código interno ao loop. Veja:

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

Agora veja um exemplo da utilização dessa declaração:

```
<script>  
  
    for(var valor in {a:1, b:2, c:3}){  
        console.log(valor);  
    }  
  
</script>
```

Você visualizará a seguinte página ao usar o código anterior:



Se você fizer modificações no objeto **array**, adicionar propriedades ou métodos personalizados, a declaração **for/in** retorna o nome de suas propriedades definidas pelo usuário, além de índices numéricos; uma vez que esta declaração também itera sobre propriedades definidas pelo usuário. Por isso, sempre que iterar sobre **array**, recomenda-se usar um loop **for** com um índice numérico.

4.3.5. Declaração **break**

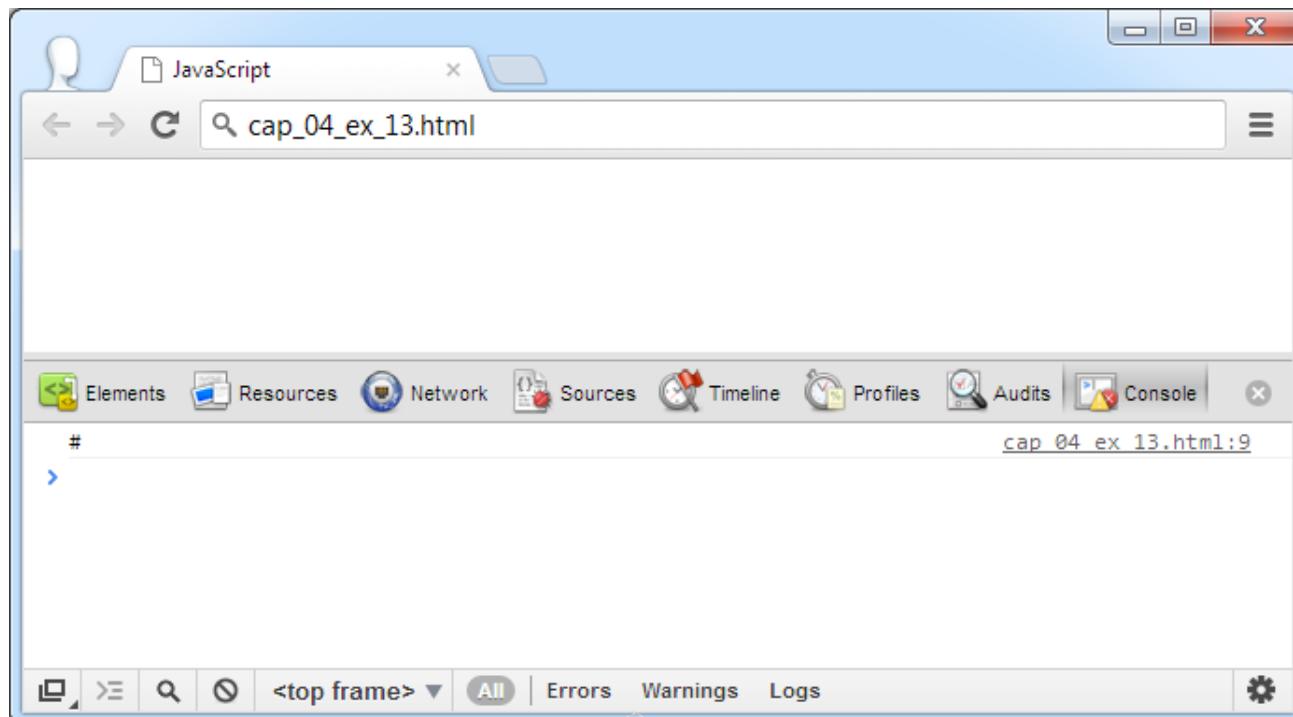
Nos loops de repetição, a declaração **break** interrompe o loop imediatamente, além de manter a execução do programa na linha seguinte ao loop, se a condição imposta for atendida.

Como você pôde ver anteriormente, **break** também pode ser usada com **switch/case**.

Veja um exemplo de utilização de **break**:

```
<script>
  while(new Date().getDay() < 7) {
    console.log('#');
    break;
  }
</script>
```

Você visualizará a seguinte página ao usar o código anterior:



4.3.6. Declaração continue

Esta declaração geralmente é usada em um teste **if** e faz com que o loop retorne imediatamente para o teste de condição do loop de repetição.

Pelo exemplo, você deve ter percebido que o código verifica a existência de erros. Assim, se a condição for verdadeira, a leitura do próximo campo será feita até que se atinja o final do arquivo.

JavaScript

Veja o exemplo a seguir, em que a declaração **continue** é usada com o loop de repetição **while**:

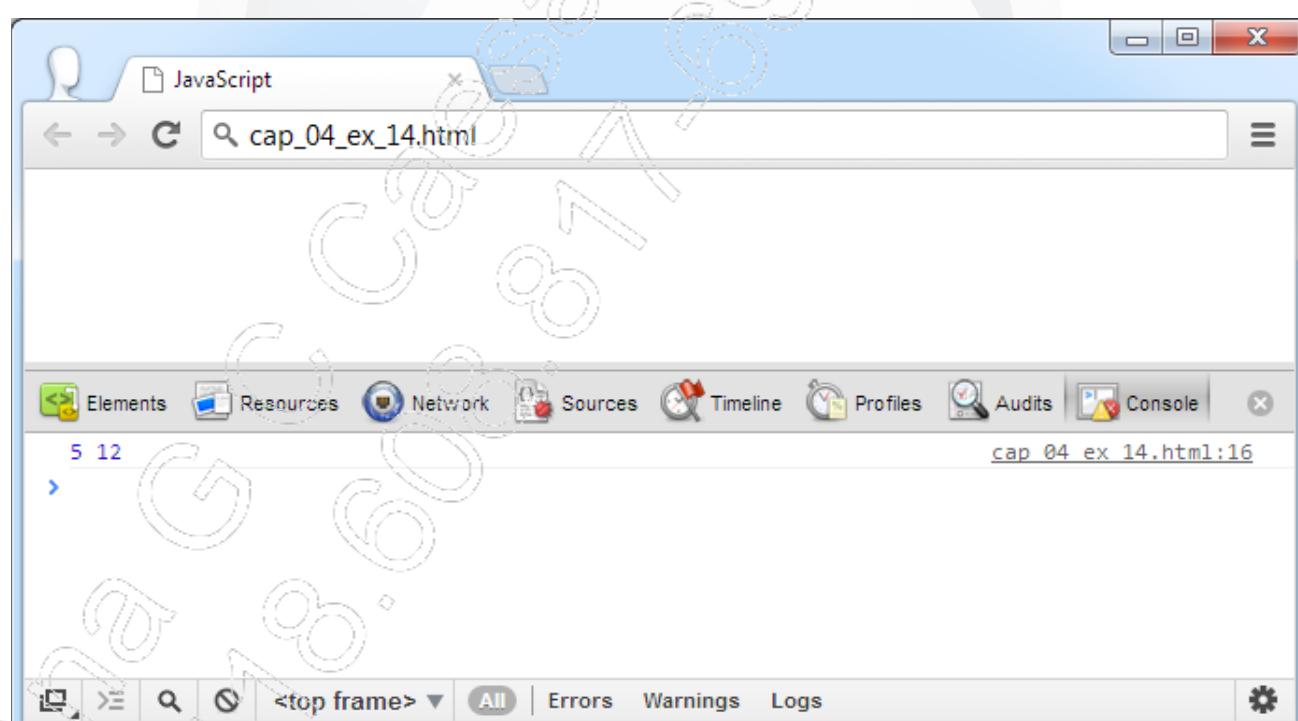
```
<script>

    a = 0;
    b = 0;
    while (a < 5) {
        a++;
        if (a == 3) continue;
        b += a;
    }

    console.log(a, b);

</script>
```

Você visualizará a seguinte página ao usar o código anterior:

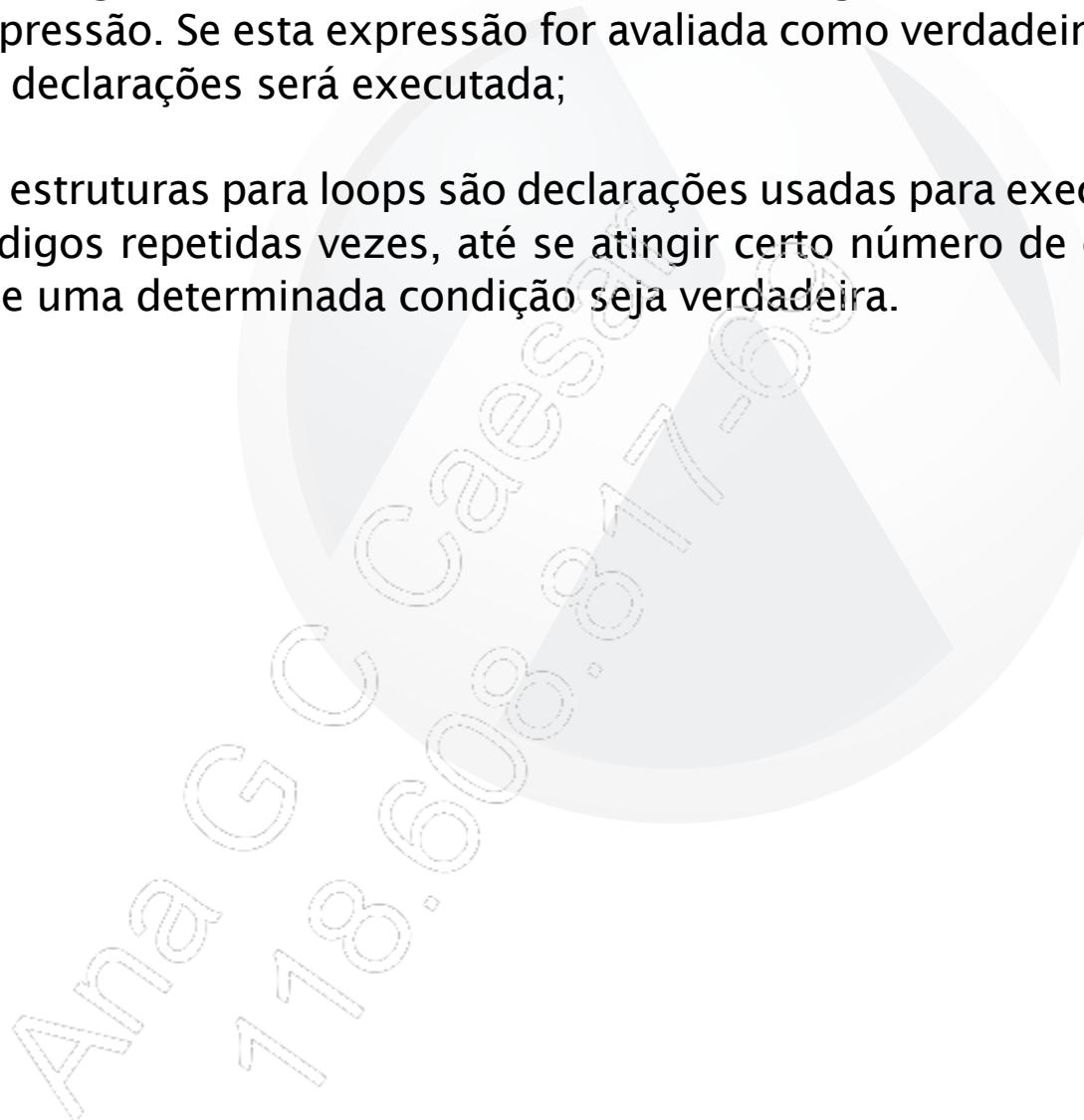


A declaração **continue** é usada somente em estruturas para loops.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- A JavaScript é responsável por grande parte dos recursos interativos nas páginas HTML. Para incorporação desses recursos, faz uso de um grupo de declarações com características e funcionalidades específicas;
- Para que você decida qual ação será realizada em uma parte específica de um programa, é necessário solicitar que o programa avalie uma determinada expressão. Se esta expressão for avaliada como verdadeira, uma sequência de declarações será executada;
- As estruturas para loops são declarações usadas para executar um bloco de códigos repetidas vezes, até se atingir certo número de execuções ou até que uma determinada condição seja verdadeira.



4

Declarações

Teste seus conhecimentos

Ana G. C.
778.6008.



IMPACTA
EDITORA

1. Qual a finalidade do comando throw?

- a) Sinalizar erros.
- b) Tratar erros.
- c) Declarar erros.
- d) Remover erros.
- e) Nenhuma das alternativas anteriores está correta.

2. Qual das estruturas abaixo não é estrutura de loop:

- a) While
- b) Do/while
- c) For
- d) If/else
- e) For/in

3. Qual o comando responsável por interromper imediatamente um loop for?

- a) Break
- b) Continue
- c) Pause
- d) Clear
- e) Nenhuma das anteriores

4. Qual é a sintaxe correta na utilização de um loop de repetição for:

- a) for (declaração e inicialização de variável; condição; iteração){}
- b) for (declaração e inicialização de variável; iteração; condição){}
- c) for (declaração e inicialização de variável; condição){}
- d) for (condição; iteração; declaração e inicialização de variável){}
- e) Nenhuma das alternativas anteriores está correta.

5. Qual a finalidade do comando `with`?

- a) Reinicia loop.
- b) Sinaliza erros.
- c) Altera escopo.
- d) Definir um valor em um método.
- e) Definir uma propriedade de uma função.



4

Declarações Mãos à obra!

Ana Gómez
778.6000



IMPACTA
EDITORA

Laboratório 1

A - Criando um código gerador de tabuadas

1. Crie um novo arquivo de texto e renomeie com a extensão.html;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD**, **BODY**;
4. Inclua dentro da tag **HEAD** a tag **<script></script>**;
5. Dentro da tag **<script>**, desenvolva o seguinte sistema:
 - Utilizando a instrução **for**, desenvolva um código que mostre na tela a tabuada de 1 a 10.

Laboratório 2

A - Alternando cores

1. Utilizando o laboratório anterior, faça a seguinte incrementação:
 - As linhas devem ser alternadas com cores diferentes.

Laboratório 3

A - Calculando valores de comissão

1. Crie um novo arquivo de texto e renomear com a extensão.html;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD**, **BODY**;

4. Inclua dentro da tag **HEAD** a tag **<script></script>**;

5. Dentro da tag **<script>**, desenvolva o seguinte sistema:

- A empresa BBB LTDA. precisa criar um sistema que imprima para os seus três vendedores (Bruno, João e Fernando) a comissão que cada um irá ganhar na ultima semana de trabalho:
 - A empresa funcionou de segunda-feira a sexta-feira;
 - Na segunda-feira a empresa vendeu R\$100.000,00. Bruno vendeu R\$20.000,00, João vendeu R\$30.000,00, Fernando vendeu R\$50.000,00;
 - Na terça-feira a empresa vendeu R\$120.000,00. Bruno vendeu R\$60.000,00, João vendeu R\$40.000,00, Fernando vendeu R\$20.000,00;
 - Na quarta-feira a empresa vendeu R\$80.000,00. Bruno vendeu R\$10.000,00, João vendeu R\$60.000,00, Fernando vendeu R\$10.000,00;
 - Na quinta-feira a empresa vendeu R\$150.000,00. Bruno vendeu R\$80.000,00, João vendeu R\$10.000,00, Fernando vendeu R\$60.000,00;
 - Na sexta-feira a empresa vendeu R\$250.000,00. Bruno vendeu R\$120.000,00, João vendeu R\$70.000,00, Fernando vendeu R\$60.000,00.
- O valor de comissão é 1% do valor vendido e somente de sexta-feira o valor é 3%.

6. Imprima na tela o valor de comissão de cada vendedor.

Funções

5

- ✓ O que é uma função?;
- ✓ Definindo uma função;
- ✓ Chamando funções;
- ✓ Escopo de uma função;
- ✓ Closures;
- ✓ Inserindo variáveis nos parâmetros;
- ✓ Retornando valores;
- ✓ Funções predefinidas;
- ✓ Propriedades das funções;
- ✓ Métodos para funções.

5.1. O que é uma função?

Durante a execução de um programa é comum ser necessário realizar alguns processos repetidas vezes. Para evitar justamente a repetição de códigos nos scripts, podemos agrupá-los em funções. Ao agrupar os processos em uma função, basta apenas chamá-la e ela ficará encarregada de realizar as tarefas necessárias.

Você pode, então, entender uma função como um conjunto de instruções em um mesmo processo, o qual pode ser executado ao ser chamado. Considere, por exemplo, a criação de uma função para mudar a cor de fundo de uma página Web. Neste caso, se você quisesse mudar a cor de fundo em outra parte da página, não seria necessário escrever todo o código de novo, bastaria apenas chamar a mesma função, já definida.

É importante levar em conta que o uso das funções não se restringe apenas aos desenvolvedores. Além das funções que eles escrevem, também há aquelas já definidas no sistema, que são empregadas em processos rotineiros, como na exibição de mensagens na tela, obtenção de hora, entre outros.

Neste capítulo, você aprenderá a criar e chamar uma função, sua utilização e manipulação por meio de propriedades e métodos.

5.2. Definindo uma função

Para criar uma função, devemos utilizar a seguinte sintaxe:

```
function nome_funcao() {  
    instruções da função  
}
```

Em que:

- **function**: Indica a criação de uma função;
- **nome da função**: Este parâmetro nomeia a função a ser criada. Pode conter números, letras e algum caractere adicional.

Depois de especificado o nome para a variável, basta inserir as instruções da função entre as chaves de abertura e de fechamento.

A utilização das chaves de abertura e de fechamento não é opcional para funções. Elas contribuem, inclusive, para uma visualização mais fácil da estrutura que é formada pelas instruções da função.

5.2.1. Inserindo funções

As funções podem ser definidas em qualquer parte de uma página, desde que entre as tags `<script>` e `</script>`. É indiferente, também, se a chamada está antes ou depois da função, desde que esteja no mesmo bloco `<script>`. Em outros termos, a função deve ser definida no mesmo bloco de sua chamada.

Mas, para evitar enganos, o mais comum é inserir a função antes de qualquer chamada, conforme o exemplo a seguir:

```
<script>
function saudacoes(){
    document.write("Olá!");
}
saudacoes();
</script>
```

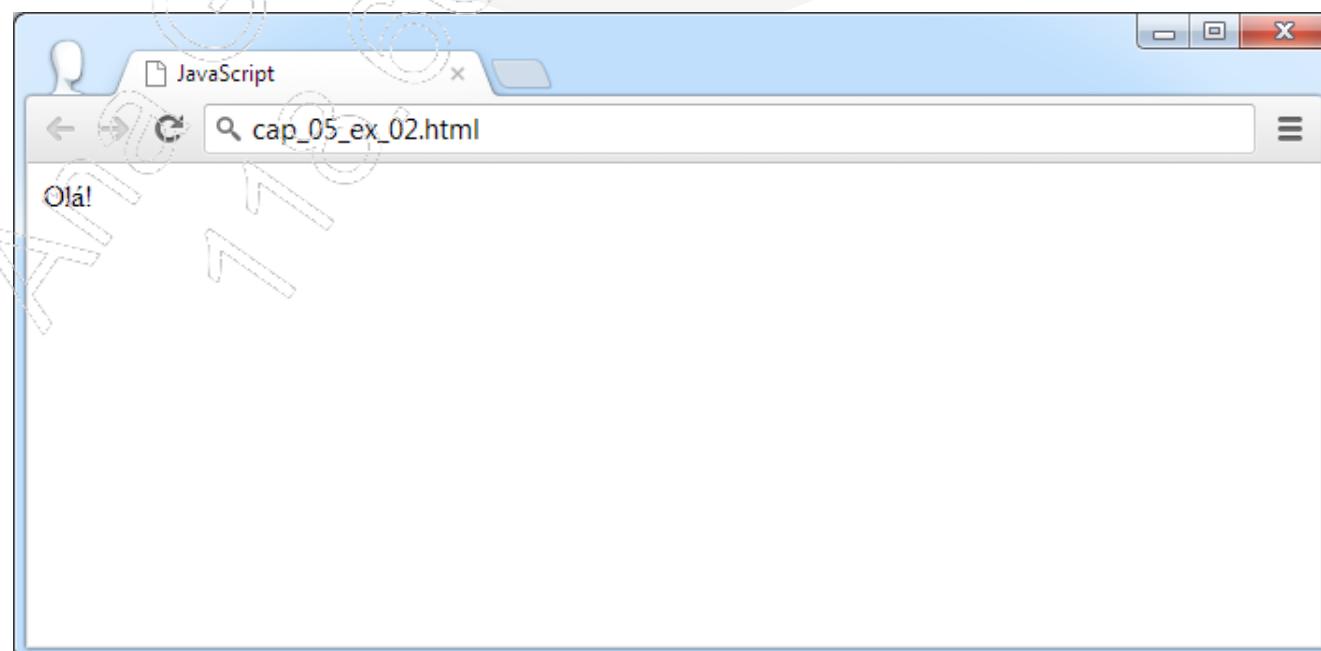
O funcionamento é garantido pelo fato de a função estar declarada no mesmo bloco `<script>` de sua chamada.

JavaScript

A função também pode ser inserida em um bloco `<script>` que esteja antes do bloco da chamada. Veja o exemplo adiante:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function saudacoes(){
9     document.write("Olá!");
10 }
11
12 </script>
13 </head>
14
15 <body>
16
17 <script>
18 saudacoes();
19 </script>
20 </body>
21 </html>
```

A seguinte página será exibida depois que você salva e executa o código anterior:



5.3. Chamando funções

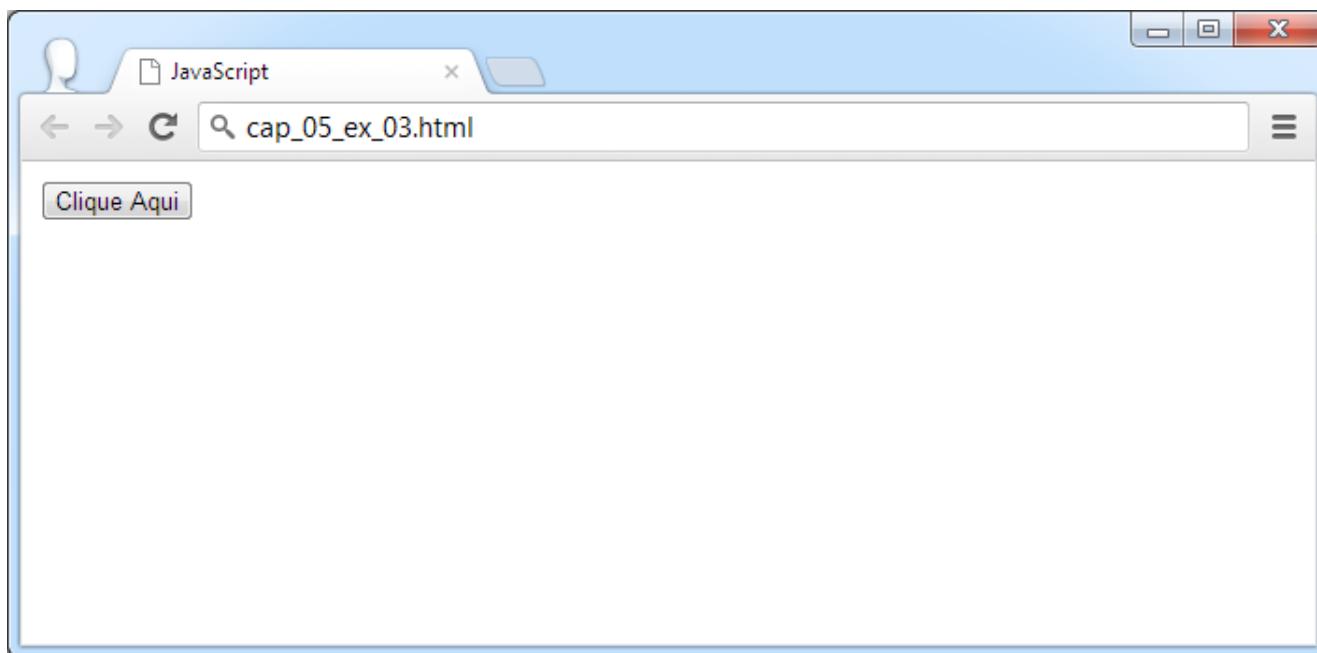
Para chamar funções, utilize o formato nome da função seguido por parênteses: **nome_função()**. É possível chamá-las a partir de qualquer parte da página. Dessa maneira, o conjunto de instruções que possui a função entre as duas chaves será executado de forma correta.

Considere o seguinte exemplo, no qual a função é chamada a partir do momento em que um botão é pressionado:

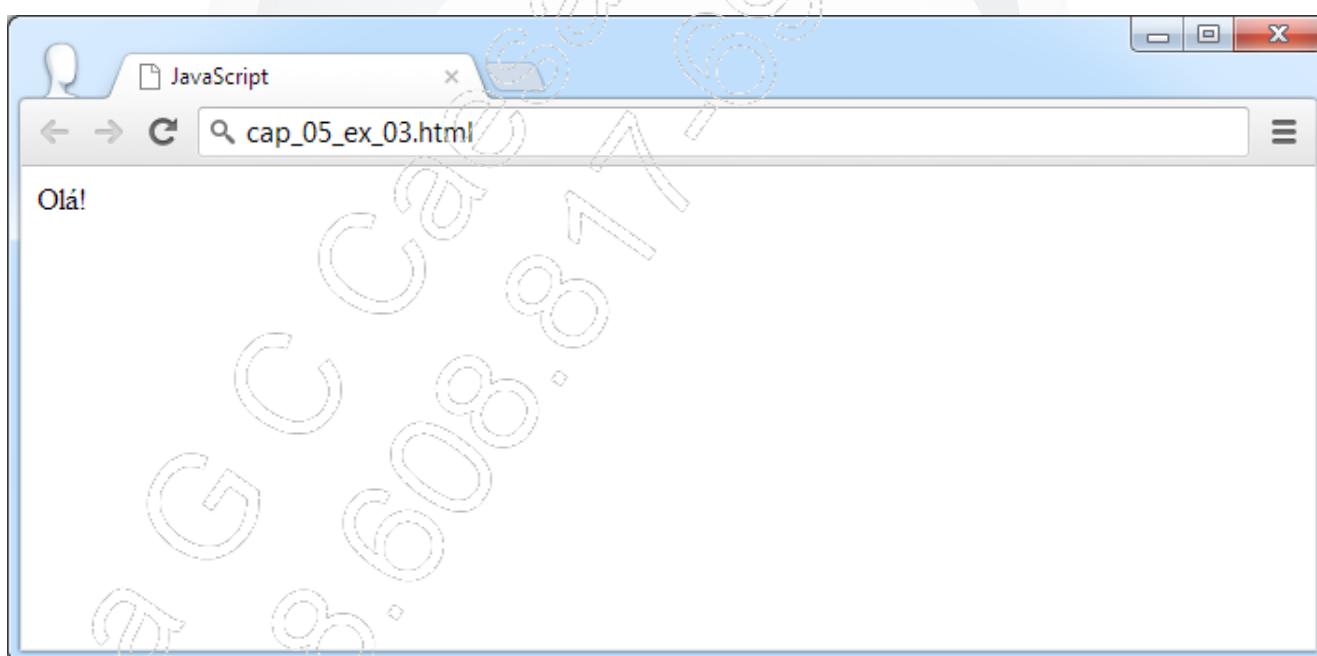
```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function saudacoes(){
9     document.write("Olá!");
10 }
11
12 </script>
13 </head>
14
15 <body>
16
17 <button onclick="saudacoes()">Clique Aqui</button>
18
19 </body>
20 </html>
```

JavaScript

A seguinte página será exibida depois de salvar e executar o código anterior:



Após clicar no botão:



5.4. Escopo de uma função

Como as variáveis são definidas no escopo da função, elas não podem ser acessadas a partir de qualquer lugar fora da função. Basicamente, uma função pode acessar todas as variáveis e demais funções do mesmo escopo em que foi definida.

Uma função que foi definida no escopo global, por exemplo, poderá acessar todas as variáveis definidas nesse mesmo escopo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var texto = 'Mundo!';
9
10 function saudacoes(){
11     document.write("Olá " + texto);
12 }
13
14 </script>
15 </head>
16
17 <body onLoad="saudacoes()">
18
19 </body>
20 </html>
```

Quando uma função é definida dentro de outra função, elas são consideradas como função-pai e função-filha. A função-filha poderá acessar as variáveis da função-pai, assim como as variáveis que esta tiver acessado:

```
<script>
var texto = 'Mundo!';

function saudacoes(){
    function funcao_filha(){
        return '<i>' + texto + '</i>';
    }

    document.write("Olá " + funcao_filha());
}

saudacoes();

</script>
```

5.5. Closures

Um dos recursos mais interessantes da JavaScript são os **closures**. Eles representam o aninhamento de funções, isto é, uma função é definida dentro de outra função. Um closure é criado quando a função interna passa a ser disponibilizada fora do escopo da função externa.

Com isso, a função interna possui acesso total às variáveis e funções dentro da função em que foi definida, além do acesso às variáveis e funções que esta tiver acessado.

Considere o seguinte exemplo:

```
<script>

function multiplicarPor(n) {
    return function(x) { return x * n; };
}

var por10 = multiplicarPor(10);

console.log(por10(2)); //20
console.log(por10(3)); //30

</script>
```

As variáveis e funções nela incluídas podem ter a sua vida útil estendida em relação à própria função externa, uma vez que as funções internas podem acessar o seu escopo e são destinadas a uma vida útil maior.

Já a função externa, em contrapartida, não tem acesso às variáveis e funções que estão dentro da função interna, garantindo segurança a elas:

```
<script>

function multiplicarPor(n) {

    console.log('1. x existe? ', (typeof x));

    return function(x) { return x * n;};

}

var por10 = multiplicarPor(10);

console.log(por10(4)); //40
console.log(por10(5)); //50

console.log('2. x existe? ', (typeof x));

</script>
```

5.6. Inserindo variáveis nos parâmetros

Para receber e devolver valores, as funções apresentam uma entrada e uma saída. Por meio dos parâmetros, é possível definir valores de entrada para a função, os quais serão empregados por ela para realizar as ações.

Para uma função que deve realizar a soma de dois números, por exemplo, os parâmetros serão esses dois números, ou seja, ambos serão a entrada. O resultado, por sua vez, será a saída. Veja o seguinte exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function somarCom(n) {
9
10     return function(x) { return x + n;};
11 }
12
13 var com2 = somarCom(2);
14
15 for(var i = 0; i <= 10; i++) {
16     console.log(i+' + 2 = '+com2(i));
17 }
18
19 </script>
20 </head>
21
22 <body>
23
24 </body>
25 </html>
```

A utilização dos parâmetros será explicada nos próximos subtópicos.

5.6.1. Um parâmetro

Para especificar um parâmetro na função, é necessário inserir o nome da variável que contém o dado a ser passado para a função quando a chamarmos.

O tempo de vida útil da variável dura até o término da execução da função. Assim que a função é executada, a variável deixa de existir.

Quando você quer chamar a função, o valor do parâmetro deve ser colocado entre parênteses. Veja o exemplo adiante, em que chamamos a função do exemplo anterior:

```
14 var com2 = somarCom(2);
```

Os parâmetros aceitam quaisquer tipos de dados, sejam eles textuais, numéricos ou mesmo booleanos. Quando não definimos o tipo de parâmetro, é necessária atenção especial ao especificar as ações que serão efetuadas dentro da função.

Também é importante ter cuidado ao passar valores para a função, para que os valores passados não entrem em conflito com os tipos já definidos para as variáveis ou parâmetros.

5.6.2. Múltiplos parâmetros

Podemos inserir a quantidade desejada de parâmetros dentro de uma função. Para defini-los, basta separá-los por vírgulas dentro dos parênteses, conforme o exemplo adiante:

```
<script>  
  
function calcularRegra(n1, n2) {  
  
    return function(x) { return (x * n1) + n2; };  
}  
  
var calc = calcularRegra(10, 5);  
  
console.log(calc(3));  
  
</script>
```

5.6.3. Parâmetros passados por valores

Os parâmetros passados por valores fazem com que a variável original não tenha seu valor modificado mesmo quando você altera um parâmetro em uma função.

Para compreender melhor a utilização desses parâmetros, veja o exemplo adiante:

```
<script>

function calcularRegra(n1) {

    var n2 = 0;

    return function(x) { n2++; return (x * n1) + n2;};

}

var calc = calcularRegra(10);

console.log(calc(10));
console.log(calc(10));
console.log(calc(10));

</script>
```

5.7. Retornando valores

Na linguagem JavaScript, há funções que têm como objetivo retornar valores, ou seja, atribuí-los às variáveis. Isto amplia as possibilidades de utilização dos aplicativos.

Podemos retornar valores por meio da palavra-chave **return**, conforme utilizado no exemplo anterior:

```
return function(x) { n2++; return (x * n1) + n2;};
```

5.8. Funções predefinidas

Existem algumas funções predefinidas disponíveis para utilização na linguagem JavaScript, são elas: **eval**, **isFinite**, **isNaN**, **parseInt**, **parseFloat**, **number**, **string**, **encodeURI**, **decodeURI**, **encodeURIComponent** e **decodeURIComponent**.

A seguir, estas funções predefinidas serão descritas:

- **eval**

Com a utilização desta função, é possível retornar um número a partir de uma operação que contém uma string.

Veja o seguinte exemplo:

```
<script>  
  
eval("var resultado = 10 + 3;");  
  
console.log(resultado);  
  
</script>
```

- **isFinite**

Com a utilização desta função, disponível na linguagem JavaScript a partir da versão 1.3, é possível verificar se um número é finito ou não. Caso o número seja finito, o valor **true** será retornado; caso contrário, o valor **false** será retornado.

Veja o exemplo adiante:

```
<script>  
  
var resultado = 10 * Math.pow(10, 309);  
  
console.log(isFinite(resultado));  
  
</script>
```

- **isNaN**

Com a utilização desta função, é possível verificar se um argumento é ou não um número. O argumento pode ser tanto um objeto quanto uma variável.

O retorno da função será um valor booleano: caso o valor não seja um número, ela retorna **true**, se for um número, ela retorna **false**.

Veja o seguinte exemplo:

```
<script>

    var numero = parseInt('A1');

    console.log(isNaN(numero));

</script>
```

- **parseInt**

Por meio desta função, é possível transformar uma string em um número inteiro. O número inteiro é retornado mesmo quando a string possui um valor com casa decimal.

Observe o exemplo adiante:

```
<script>

    var texto = '1A';

    console.log(texto);

    texto = parseInt(texto);

    console.log(texto);

</script>
```

- **parseFloat**

Com a utilização desta função, você consegue transformar uma string em um ponto flutuante.

Veja um exemplo demonstrando a utilização da função **parseFloat**:

```
<script>

var texto = '1.2A';

console.log(texto);

texto = parseFloat(texto);

console.log(texto);

</script>
```

- **Number**

Por meio desta função, é possível converter um objeto em um número.

Observe o exemplo adiante:

```
<script>

var data = new Date();

var numero = Number(data);

console.log(numero);

</script>
```

- **String**

Por meio desta função, é possível converter um objeto em uma string.

Observe o exemplo adiante:

```
<script>

    var data = new Date();

    var texto = String(data);

    console.log(texto);

</script>
```

- **encodeURI**

Com a utilização desta função, você consegue codificar um determinado texto. Com isso, os caracteres são substituídos por caracteres Unicode 8 bits.

Veja o exemplo demonstrando a utilização da função **encodeURI**:

```
<script>
    var texto = 'João Henrique';
    console.log(encodeURI(texto));
</script>
```

- **decodeURI**

Com a utilização desta função, que atua de forma oposta à função **encodeURI**, é possível decodificar um determinado texto que possua caracteres UTF-8.

Veja um exemplo demonstrando a utilização da função **decodeURI**:

```
<script>

var texto = 'Jo%C3%A3o%20Henrique';

console.log(decodeURI(texto));

</script>
```

- **encodeURIComponent**

Por meio desta função, é possível codificar para caracteres UTF-8 os caracteres especiais de um componente URL.

Observe o seguinte exemplo:

```
<script>

console.log(encodeURIComponent(window.location));

</script>
```

- **decodeURIComponent**

Por meio desta função, oposta à função **encodeURIComponent**, é possível decodificar um texto que possua caracteres UTF-8.

Observe o seguinte exemplo:

```
<script>

console.log(decodeURIComponent(encodeURIComponent(window.location)));

</script>
```

5.9. Propriedades das funções

As funções possuem algumas propriedades. São elas: **arguments.length**, **arguments.callee**, **length**, **constructor** e **prototype**.

Os próximos subtópicos descrevem as propriedades das funções.

5.9.1. arguments.length

Trata-se da propriedade **length** do objeto **arguments**. Por meio dela, você consegue saber quantos argumentos a função possui, isto é, o retorno é o número de argumentos da função.

Veja um exemplo de utilização desta propriedade:

```
<script>
  function calcula(a, b, c) {
    console.log(arguments.length);
  }
  calcula();
  calcula(1, 2, 3);
</script>
```

5.9.2. arguments.callee

Trata-se da propriedade **callee** do objeto **arguments**. Por meio desta propriedade, é possível fazer referência a uma função anônima, sem que seja necessário criar um nome para ela. Com isso, você consegue chamar a função anônima dentro de si mesma.

Para compreender melhor o funcionamento desta propriedade, veja o exemplo a seguir:

```
<script>

function factorial() {
    return function(n) {
        if (n <= 1)
            return 1;
        return n * arguments.callee(n - 1);
    }
}

var resultado = factorial()(5);

console.log(resultado);

</script>
```

A chamada da função dentro de si mesma é denominada mecanismo recursivo, o que qualifica a função como recursiva.

5.9.3. length

Trata-se da propriedade **length** do objeto **função**. Por meio dela, é possível verificar quantos argumentos a função espera receber, isto é, o retorno é o número de argumentos que podem ser passados à função. É diferente da propriedade **arguments.length**.

A tabela a seguir ajuda a identificar a diferença entre essas propriedades:

TABELA COMPARATIVA	
Propriedade lenght	Propriedade arguments.length
Retorna o número de argumentos que podem ser passados para a função.	Retorna o número de argumentos passados para a função.
Pode ser acessada a partir de fora da função.	Só existe dentro do corpo da função.

Veja o exemplo de utilização da propriedade **length**:

```
<script>

    function factorial() {
        return function(n) {
            if (n <= 1)
                return 1;
            return n * arguments.callee(n - 1);
        };
    }

    console.log(factorial.length);
    console.log(factorial().length);

</script>
```

5.9.4. constructor

Esta propriedade é uma referência a uma função responsável por criar um objeto.

Para compreender melhor a propriedade **constructor**, veja o exemplo adiante:

```
<script>

var a = Object;

console.log(a.constructor);

a.constructor = function(){ alert('Objeto'); }

console.log(a.constructor);

</script>
```

5.9.5. prototype

Por meio desta propriedade, você consegue adicionar tanto propriedades quanto métodos a um objeto preexistente.

Veja um exemplo de utilização de **prototype**:

```
<script>

Date.prototype.diasDaSemana = [
  'Domingo',
  'Segunda',
  'Terça',
  'Quarta',
  'Quinta',
  'Sexta',
  'Sábado'
];

Date.prototype.qualDia = function(){

  return this.diasDaSemana[this.getDay()];
}

var data = new Date();

console.log('Hoje é ' + data.qualDia());

</script>
```

5.10. Métodos para funções

Além das propriedades, você pode utilizar os seguintes métodos para as funções: **call**, **apply** e **toString**. Estes métodos são descritos adiante:

- **call(this, argumentos)**

Este método permite inserir dois ou mais métodos em um só objeto e mesclá-los. O resultado é um objeto com vários métodos. Para isso, basta chamar um método dentro de outro método e passar o objeto como parâmetro.

Obrigatoriamente, os seguintes parâmetros devem ser utilizados:

- Palavra-chave **this**;
- Argumentos do método que será mesclado.

Para compreender melhor a utilização do método **call**, veja o seguinte exemplo:

```
<script>
  function calcula(a, b, c) {
    console.log(a, b, c);
  }
  calcula.call(this, 1, 2, 3);
</script>
```

- **apply(this, array)**

Este método é semelhante ao **call**, diferindo apenas em como os parâmetros são passados. Em vez de listar entre vírgulas os valores dos argumentos como é feito no método **call**, você utilizará um array para passar os parâmetros.

Para criar um array de parâmetros, podemos utilizar:

- **Sintaxe literal**

```
<script>

    function calcula(a, b, c) {

        console.log(a, b, c);

    }

    calcula.apply(this, [1, 2, 3]);

</script>
```

- **Operador new**

```
<script>

    function calcula(a, b, c) {

        console.log(a, b, c);

    }

    calcula.apply(this, new Array(1, 2, 3));

</script>
```

- **Propriedade arguments**

Esta propriedade é um pseudo-array dos argumentos de uma função. Sua utilização é a seguinte:

```
<script>

    function calcula(a, b, c) {

        console.log(arguments);

    }

    calcula.apply(this, new Array(1, 2, 3));

</script>
```

- **toString()**

Com a utilização do método **toString()**, o retorno será uma string contendo o código da função.

Veja o exemplo adiante:

```
<script>

    function calcula(a, b, c) {

        console.log(arguments);

    }

    console.log(calcula.toString());

</script>
```



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Funções podem ser consideradas um conjunto de instruções em um mesmo processo, que pode ser executado ao ser chamado. Com isso, não precisamos executar tais processos repetidas vezes, basta chamar a mesma função;
- É necessário prestar atenção à sintaxe e os formatos utilizados para a criação de uma função. Por exemplo, ela pode ser definida em qualquer parte de uma página, desde que entre as tags `<script>` e `</ script>`;
- Para chamar funções, utilize o formato nome da função seguido por parênteses: **nome_função ()**;
- Com relação ao escopo de uma função, ela poderá acessar todas as variáveis e demais funções do mesmo escopo em que foi definida;
- Os closures, isto é, quando uma função é definida dentro de outra função, permitem que a função interna acesse as variáveis e funções da função externa, além do acesso às variáveis e funções que esta tiver acessado. Já a função externa, em contrapartida, não tem acesso às variáveis e funções que estão dentro da função interna, garantindo segurança a elas;
- Para especificar um parâmetro na função, é necessário inserir o nome da variável que contém o dado a ser passado para a função quando ela for chamada. Quando você quer chamar a função, o valor do parâmetro deve ser colocado entre parênteses e a variável deixará de existir;
- Para retornar valores, ou seja, atribuí-los às variáveis, utilize a palavra-chave **return**;

- As seguintes funções são predefinidas na linguagem JavaScript: **eval**, **isFinite**, **isNaN**, **parseInt**, **parseFloat**, **number**, **string**, **encodeURI**, **decodeURI**, **encodeURIComponent** e **decodeURIComponent**;
- As funções possuem as seguintes propriedades: **arguments_length**, **arguments_callee**, **length**, **constructor** e **prototype**;
- Você pode utilizar os seguintes métodos para as funções: **call**, **apply** e **toString**.



5

Funções

Teste seus conhecimentos

Ana Gomes
778.6008-0609



IMPACTA
EDITORA

1. Na JavaScript, quando um Closure é criado?

- a) Quando a função interna passa a ser disponibilizada fora do escopo da função externa.
- b) Quando a função externa passa a ser disponibilizada fora do escopo da função interna.
- c) Quando a função interna passa a ser disponibilizada e declarada no escopo da função interna.
- d) Quando é a função interna é excluída do escopo da função externa.
- e) Nenhuma das alternativas anteriores está correta.

2. Qual o delimitador necessário para separar parâmetros de uma função?

- a) ;
- b) And
- c) ,
- d) .
- e) &

3. Qual o objetivo da função number?

- a) Converter um objeto em um número.
- b) Converter um número em string.
- c) Converter um número em decimal.
- d) Converter uma string em número.
- e) Nenhuma das alternativas anteriores está correta.

4. Qual são os métodos das funções que podem ser utilizados na JavaScript?

- a) Call, apply, new
- b) Apply, new, toString
- c) Call, apply, toString.
- d) Number, toString, new
- e) Nenhuma das alternativas anteriores está correta.

5. Qual dos tipos abaixo não é considerado uma propriedade das funções na JavaScript?

- a) Arguments_length
- b) Length
- c) arguments_callee
- d) toString
- e) prototype



5

Funções Mãos à obra!

Ana Góesar
778.6008.



IMPACTA
EDITORA

Laboratório 1

A - Criando função para calcular regra de três

1. Crie um novo arquivo de texto e renomeie com a extensão **.html**;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD**, **BODY**;
4. Inclua dentro da tag **HEAD** a tag **<script></script>**;
5. Crie uma função, defina o nome da função como **fnRegradeTres** e atribua três parâmetros a ela;
6. No retorno da função, realize o cálculo do valor;
7. Utilize a função para calcular os seguintes problemas:
 - Bruno trabalhou 30 dias e recebeu R\$ 1.000,00 reais. Quantos dias terá que trabalhar para receber R\$ 10.000,00 reais?
 - João trabalhou 40 dias e recebeu R\$ 1.500,00 reais. Quantos dias terá que trabalhar para receber R\$ 10.000,00 reais?

Laboratório 2

A - Otimizando código

1. Baseado no Laboratório 3 do capítulo anterior, crie uma função para realizar o cálculo de comissão dos funcionários:
 - Caso a comissão seja maior igual a R\$ 5.000,00, acrescentar um bônus de R\$ 1.000,00;
 - Caso a comissão seja maior que R\$ 500,00, acrescentar R\$ 100,00 de bônus.

Eventos

6

- ✓ Definindo um evento;
- ✓ Manipuladores de eventos;
- ✓ Objeto Event.

Ana Caesar
778.6657-69



IMPACTA
EDITORA

6.1. Introdução

Na linguagem JavaScript, é possível controlar as ações dos visitantes de uma página e definir um comportamento para ela quando as ações são criadas. Para isso servem os eventos, que também são aplicáveis a botões e outros elementos que compõem uma página.

Os eventos acontecem quando um usuário acessa uma página da Web e passa a interagir com ela. Assim, você pode utilizar JavaScript para determinar a ação que deverá ocorrer na criação dos eventos. Clicar sobre um botão da página ou pressionar uma tecla são procedimentos considerados como eventos.

Por meio dos eventos, você pode responder às ações dos usuários, o que faz da criação dos eventos um recurso muito importante para desenvolver páginas interativas.

6.2. Definindo um evento

São os manipuladores de evento que determinam as ações a serem realizadas na produção de um evento. Para cada tipo de ação do usuário existe um manipulador de evento específico. Você deve colocar o manipulador de eventos na tag referente ao elemento da página que responderá às ações do usuário.

Para descrever ações executadas assim que um clique é dado sobre um botão, utiliza-se o manipulador de eventos **onclick**, que deve ser escrito na tag **<input type=button>**. Veja a sintaxe a seguir:

```
<input type=button value="Clique aqui!"  
      onclick="Instruções_JavaScript">
```

Normalmente utiliza-se somente uma instrução dentro dos manipuladores de eventos, mas você pode inserir quantas instruções quiser. Neste caso, preste atenção aos seguintes aspectos:

- As instruções devem ser separadas por ponto-e-vírgula;
- É uma prática comum criar uma função com todas as instruções e, no manipulador de eventos, inserir uma única instrução, que, por sua vez, chamará à função.

6.3. Manipuladores de eventos

Como já foi mencionado, JavaScript disponibiliza uma série de manipuladores de eventos. Veja como funciona cada um deles a seguir.



6.3.1. Eventos do mouse

São eventos relacionados a ações envolvendo o mouse, tanto em relação a cliques quanto ao posicionamento do cursor.

- **onclick**

Este manipulador de eventos faz com que um código seja executado assim que certos elementos forem clicados.

Considere o seguinte exemplo:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <style>
7  #A {
8      margin:10%;
9      width:100px;
10     height:100px;
11     background-color:rgb(0,0,0);
12 }
13 </style>
14 <script>
15
16 function evento(){
17
18     alert('onclick');
19
20 }
21
22 </script>
23 </head>
24
25 <body>
26
27     <div id="A" onclick="evento()></div>
28
29 </body>
30 </html>
```

- **ondblclick**

Este manipulador de eventos faz com que um código seja executado assim que certos elementos receberem um clique duplo.

O evento **ondblclick** identifica com exatidão a ação do usuário sobre os botões, assim como nos eventos **onclick**, **onmouseup** e **onmousedown**. Por meio da propriedade **button** do objeto **event**, você pode saber qual botão foi pressionado. Caso o botão pressionado pelo usuário tenha sido o esquerdo, **button** terá 1 como valor. Se for o direito, 2 será o valor de **button**. O clique com o botão direito em muitos sites atualmente possibilita a cópia de seu código-fonte.

Considere o seguinte exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <style>
7 #A {
8   margin:10%;
9   width:100px;
10  height:100px;
11  background-color:rgb(0,0,0);
12 }
13 </style>
14 <script>
15
16 function evento() {
17
18   alert('ondblclick');
19
20 }
21
22 </script>
23 </head>
24
25 <body>
26
27   <div id="A" ondblclick="evento()"></div>
28
29 </body>
30 </html>
```

- **onmousedown**

Este manipulador de eventos executa uma ação assim que o botão esquerdo ou direito do mouse é pressionado sobre um parágrafo.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function evento(){
9
10     alert('onmousedown');
11
12 }
13
14 </script>
15 </head>
16
17 <body>
18
19     <p onmousedown="evento()">Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
 reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
 Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.</p>
20
21 </body>
22 </html>
```

- **onmousemove**

Este manipulador de eventos é utilizado para executar uma ação sempre que o ponteiro do mouse é movimentado.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function evento() {
9
10     console.log('onmousemove');
11 }
12
13 </script>
14 </head>
15
16 <body>
17
18     <p onmousemove="evento()">Lorem ipsum dolor sit amet, consectetur
19 adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
20 aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
21 nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
22 reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Exepteur sint occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.</p>
23
24 </body>
25 </html>
```

- **onmouseover**

Este manipulador de eventos é utilizado para executar uma ação assim que posicionarmos o ponteiro do mouse sobre um link.

onmouseover é muito utilizado para a exibição de camadas ocultas, inserção de mensagens na barra de status e troca de imagens para a construção de um menu dinâmico, o que proporciona à página alguns efeitos visuais interessantes.

Considere o seguinte exemplo:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function evento(a){
9
10     a.style.color = 'red';
11
12 }
13
14 </script>
15 </head>
16
17 <body>
18
19 <a onmouseover="evento(this)">Link</a>
20
21 </body>
22 </html>
```

- **onmouseout**

Este manipulador de eventos tem a função parecida com a de **onmouseover**, com a diferença de que a ação é executada quando o ponteiro do mouse é retirado de cima do link.

Considere o seguinte exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function over(a) {
9
10     a.style.color = 'red';
11 }
12
13 function out(a) {
14
15     a.style.color = 'black';
16 }
17
18 </script>
19 </head>
20
21 <body>
22
23     <a onmouseover="over(this)" onmouseout="out(this)">Link</a>
24
25 </body>
26
27 </html>
```

- **onmouseup**

Este manipulador de eventos, executa uma ação no momento em que qualquer um dos botões do mouse que já esteja pressionado é solto (incluindo o botão do meio).

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function evento() {
9
10    console.log('Soltou');
11
12 }
13 </script>
14 </head>
15 <body>
16
17 <a onmouseup="evento()">Link</a>
18
19 </body>
20 </html>
```

6.3.2. Eventos do teclado

São eventos relacionados ao uso do teclado, como o pressionamento de certas teclas.

- **onkeydown**

Este manipulador de evento é utilizado para executar uma ação assim que uma tecla é pressionada.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function evento(input){
9
10     console.log('onkeydown', 'Valor digitado no momento do evento: '+input.value);
11 }
12
13 </script>
14 </head>
15
16 <body>
17
18     <input type="text" onkeydown="evento(this)">
19
20 </body>
21 </html>
```

- **onkeyup**

Este manipulador de evento é utilizado para executar uma ação assim que uma tecla já pressionada é solta.

Por meio dos eventos **onkeydown** e **onkeyup**, é possível saber se uma tecla foi pressionada ou solta, respectivamente.

- **onkeypress**

O **onkeypress** realiza uma ação assim que o usuário pressiona uma determinada tecla.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function up(input){
9
10     console.log('onkeyup', 'Valor digitado no momento do evento: '+input.value);
11
12 }
13
14 function press(input){
15
16     console.log('onkeypress', 'Valor digitado no momento do evento: '+input.value);
17
18 }
19
20 </script>
21 </head>
22
23 <body>
24
25     <input type="text" onkeyup="up(this)" onkeypress="press(this)">
26
27 </body>
28 </html>
```

6.3.3. Eventos de frame/objeto

São eventos relacionados ao carregamento, fechamento, redimensionamento e movimentação de elementos como janelas, imagens, documentos e objetos.

- **onabort**

Este manipulador de eventos executa uma instrução caso o usuário interrompa o carregamento de uma imagem pressionando o botão **Stop (Parar)** do browser ou carregando outra página.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function abort(){
9
10    console.log('Abortou!');
11
12 }
13
14 </script>
15 </head>
16
17 <body>
18
19     
20
21 </body>
22 </html>
```

- **onerror**

Este manipulador de eventos executa um alerta quando uma imagem não puder ser carregada.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function erro(){
9
10    console.error('Não foi possível carregar a imagem');
11
12 }
13
14 </script>
15 </head>
16
17 <body>
18
19     
20
21 </body>
22 </html>
```

- **onload**

O manipulador **onload** faz com que o navegador execute uma ação no carregamento da página ou de uma imagem.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var dt1 = new Date();
9
10 function carregou(){
11
12     var dt2 = new Date();
13
14     console.info('A imagem foi carregada em: ' + (dt2.getTime() - dt1.getTime()) + ' milisegundos.');
15 }
16
17 </script>
18 </head>
19
20 <body>
21
22     
23
24 </body>
25 </html>
```

- **onresize**

Este manipulador de evento é usado para criar uma ação quando a janela do navegador, ou um frame, caso exista, é redimensionado.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <style>
7 * {
8     font-family:Arial, Helvetica, sans-serif;
9 }
10 </style>
11 <script>
12
13 window.onresize = function(){
14
15     document.getElementById('tamanho').innerHTML = (window.innerWidth + 'x' + window.innerHeight);
16
17 }
18
19 </script>
20 </head>
21
22 <body>
23
24     <center>
25         <p>Tamanho da Janela</p>
26         <h1 id="tamanho">Altere o tamanho da janela!</h1>
27     </center>
28
29 </body>
30 </html>
```

- **onscroll**

O manipulador **onscroll** cria uma ação quando o usuário usa a barra de rolagem.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <style>
7 * {
8     font-family:Arial, Helvetica, sans-serif;
9 }
10 body {
11     height:3000px;
12 }
13 h1 {
14     position:absolute;
15     top:0;
16 }
17 </style>
18 <script>
19
20 window.onscroll = function() {
21
22     var h1 = document.getElementById('scroll');
23
24     h1.innerHTML = window.scrollY+'px';
25     h1.style.top = window.scrollY+'px';
26
27 }
28
29 </script>
30 </head>
31
32 <body>
33
34     <h1 id="scroll">0px</h1>
35
36 </body>
37 </html>
```

- **onunload**

Este manipulador de eventos é executado quando a janela do navegador é fechada ou clicando em um link direcionado a outra página, abandonando a página anterior.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  window.onunload = function () {
9
10     return confirm("Ja vai embora mesmo?");
11
12 }
13
14 </script>
15 </head>
16
17 <body>
18
19 <h1>Feche o navegador!</h1>
20
21 </body>
22 </html>
```

6.3.4. Eventos de formulário

São eventos relacionados ao preenchimento de formulários e a outros elementos relativos.

- **onblur**

Este manipulador é utilizado para a execução de um código quando o foco é movido para fora do formulário. É um recurso bastante útil para confirmar se um formulário foi preenchido com informações corretas.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function evento(input){
9
10    console.log('onblur');
11
12  }
13
14 </script>
15 </head>
16
17 <body>
18
19 <input type="text" onblur="evento(this)">
20
21 </body>
22 </html>
```

- **onchange**

Este manipulador é parecido com **onblur**, mas é executado no momento em que o usuário inserir ou alterar algo nos elementos de texto do formulário ou quando o usuário selecionar algo diferente em um menu de seleção.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function evento(input){
9
10     console.log('onchange', 'Valor atual: ' + input.value);
11
12 }
13
14 </script>
15 </head>
16
17 <body>
18
19     <input type="text" onchange="evento(this)">
20
21 </body>
22 </html>
```

- **onfocus**

Este manipulador de eventos executa uma ação quando o foco está se movendo para o elemento do formulário, e não para fora dele, como acontece com **onblur**.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function evento(input) {
9
10    console.log('onfocus');
11
12 }
13
14 </script>
15 </head>
16
17 <body>
18
19 <input type="text" onfocus="evento(this)">
20
21 </body>
22 </html>
```

- **onreset**

O evento **onreset** ocorre assim que o usuário clica no botão de reset de um formulário.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function evento(){
9
10     console.log('onreset');
11 }
12 </script>
13 </head>
14
15 <body>
16
17 <form onreset="evento()">
18
19     <input type="reset">
20
21 </form>
22
23 </body>
24
25 </html>
```

- **onselect**

Quando utilizado, este manipulador de eventos executa um código no momento em que o usuário utiliza o ponteiro do mouse para selecionar um texto existente em um campo de texto de um formulário. Este campo pode ser textfield ou textarea.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8     function evento() {
9
10         console.log('onselect');
11
12     }
13
14 </script>
15 </head>
16
17 <body>
18
19     <textarea style="width:100%; height:200px;" onselect="evento(this)">Valor Padrão</textarea>
20
21 </body>
22 </html>
```

- **onsubmit**

O evento **onsubmit** é executado no momento em que o usuário pressiona o botão **Enviar (Submit)** e normalmente é utilizado para a validação de formulários.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function evento(){
9
10     alert('onsubmit');
11 }
12 </script>
13 </head>
14 <body>
15
16     <form onsubmit="evento()">
17         <input type="submit">
18     </form>
19
20
21
22
23
24
25 </body>
26 </html>
```

6.4. Objeto Event

O objeto **Event** nada mais é do que um objeto criado quando um evento ocorre. Ele armazena dados diversos sobre o evento, como, por exemplo, o posicionamento exato do ponteiro do mouse na hora de um clique, ou o botão que foi utilizado para o clique, no caso de um evento onde o usuário deve clicar em um elemento do DOM.

O suporte deste objeto é encontrado de modos diferentes nos navegadores mais populares do mercado.

O DOM será tratado com muito mais detalhes em um capítulo posterior. Porém, para uma melhor compreensão do objeto **Event**, você deve saber que o DOM é uma interface de programação padronizada, desenvolvida para utilização em qualquer linguagem de programação. Você pode aplicar DOM em qualquer tipo de ambiente e aplicação. Sua versatilidade faz com que seja muito utilizada na linguagem JavaScript.

A especificação mais suportada de DOM é o Nível 2, apesar de a especificação atual ser Nível 3. Para facilitar sua utilização, vamos tomar como padrão a referência DOM - Nível 2. Caso a referência deva ser feita para o Nível 3, será especificado no texto.

Normalmente, os eventos são combinados com funções, e apenas quando o evento ocorrer é que essas funções serão executadas.

6.4.1. Objeto Event no DOM

O objeto **Event** no DOM trabalha com propriedades e métodos específicos, que são compatíveis com os principais navegadores. A exceção é o Internet Explorer, que trabalha com propriedade e métodos diferentes, e que você verá mais adiante neste capítulo.

6.4.1.1. Propriedades do objeto Event no DOM

Veja agora as principais propriedades do objeto Event no DOM:

- **bubbles**: Informa valores booleanos (true ou false) para indicar se um evento é um evento de propagação de efeito bolha.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  document.addEventListener('click', function(event){
9
10     console.log(event.bubbles);
11
12 }, false);
13
14 </script>
15 </head>
16
17 <body>
18
19     <input type="button" id="botao" value="Clique Aqui">
20
21 </body>
22 </html>
```

- **cancelable**: Informa valores booleanos (true ou false) para indicar se é ou não possível o cancelamento do comportamento padrão do navegador.

Veja o exemplo a seguir:

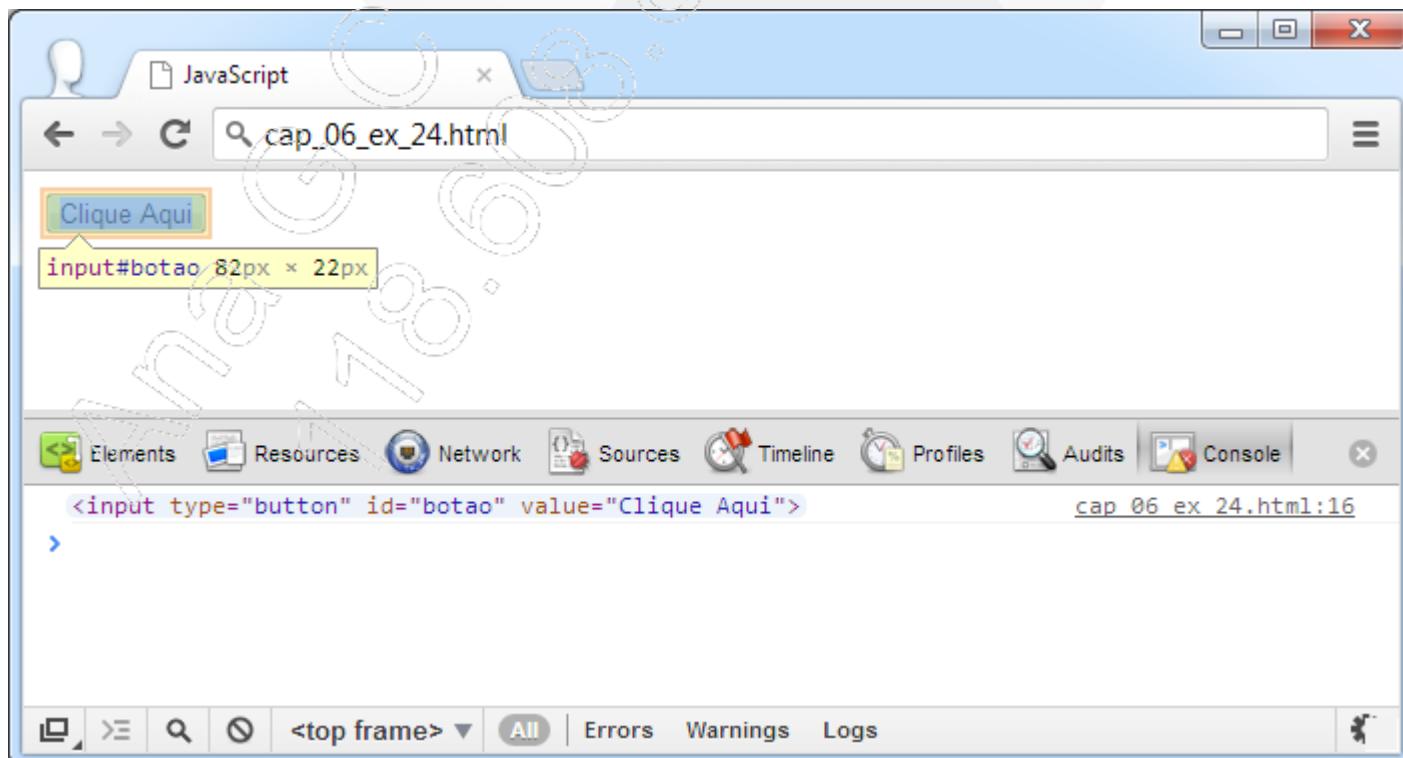
```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  document.addEventListener('click', function(event) {
9
10     console.log(event.cancelable);
11
12 }, false);
13
14 </script>
15 </head>
16
17 <body>
18
19     <input type="button" id="botao" value="Clique Aqui">
20
21 </body>
22 </html>
```

- **currentTarget**: Indica qual é o objeto ao qual se liga o evento naquele momento. É uma propriedade somente leitura.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7
8 <body>
9
10 <input type="button" id="botao" value="Clique Aqui">
11
12 <script>
13
14 document.getElementById('botao').addEventListener('click', function(event) {
15
16     console.log(event.currentTarget);
17
18 }, false);
19
20 </script>
21 </body>
22 </html>
```

Ao salvar e executar o código acima, você visualizará a seguinte página:



- **eventPhase**: Essa propriedade fornece dados sobre a propagação do evento, listando três tipos de respostas, e cada uma delas retorna um número: 1 para evento que ocorre na fase da captura, 2 para evento que ocorre no alvo e 3 para evento que ocorre na fase bolha. A propriedade **eventPhase** é somente leitura.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7
8 <body>
9
10    <input type="button" id="botao" value="Clique Aqui">
11
12 <script>
13
14 document.getElementById('botao').addEventListener('click', function(event) {
15
16     console.log(event.eventPhase);
17
18 }, false);
19
20 </script>
21 </body>
22 </html>
```

- **target**: O elemento que disparou o evento é informado quando essa propriedade, também do tipo somente leitura, é usada.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7
8  <body>
9
10     <input type="button" id="botao" value="Clique Aqui">
11
12 <script>
13
14 document.getElementById('botao').addEventListener('click', function(event) {
15
16     console.log(event.target);
17
18 }, false);
19
20 </script>
21 </body>
22 </html>
```



- **timeStamp**: Fornece informação sobre tempo em que o evento foi criado, em milissegundos.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7
8 <body>
9
10    <input type="button" id="botao" value="Clique Aqui">
11
12 <script>
13
14 document.getElementById('botao').addEventListener('click', function(event) {
15
16     var data = new Date(event.timeStamp);
17
18     console.log(event.timeStamp, data);
19
20 }, false);
21
22 </script>
23 </body>
24 </html>
```

- **type:** Essa propriedade indica o tipo de evento. É muito eficiente quando usado com um manipulador de evento para definir qual função deve ser acionada de acordo com o tipo de evento que ocorrer.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7
8  <body>
9
10     <input type="button" id="botao" value="Clique Aqui">
11
12 <script>
13
14 document.getElementById('botao').addEventListener('click', function(event) {
15
16     console.log(event.type);
17
18 }, false);
19
20 </script>
21 </body>
22 </html>
```

6.4.1.2. Métodos do objeto Event

Veja agora os principais métodos do objeto Event:

- **initEvent**: Com o método **initEvent**, o tipo de evento é especificado, a possibilidade de o evento ser propagado em efeito bolha também é verificada, e especifica se é possível ou não impedir a ação padrão do evento.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7
8 <body>
9
10    <input type="button" id="botao" value="Clique Aqui">
11
12 <script>
13
14 document.getElementById('botao').addEventListener('click', function(event){
15
16     console.log(event.initEvent('click', true, false));
17
18 }, false);
19
20 </script>
21 </body>
22 </html>
```

- **preventDefault:** Para cancelar o evento se ele for cancelável, significando que qualquer ação padrão tomada normalmente pela implementação como resultado do evento não ocorrerá.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7
8  <body>
9
10     <p>Clique com o botão direito do mouse.</p>
11
12 <script>
13
14 document.addEventListener('contextmenu', function(event) {
15
16     event.preventDefault();
17
18 }, false);
19
20 </script>
21 </body>
22 </html>
```



- **stopPropagation:** Este método previne que um evento continue sendo propagado durante o fluxo do evento.

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7
8 <body>
9
10    <button id="botao">Clique</button>
11
12 <script>
13
14 document.getElementById("botao").addEventListener('click', function(event) {
15
16     event.stopPropagation();
17
18 }, false);
19 </script>
20 </body>
21 </html>
```

6.4.2. Objeto Event no Internet Explorer

O objeto **Event** funciona no Internet Explorer de jeitos diferentes, baseado no método que você escolher para manipular o evento. Entenda melhor observando a tabela:

Método	Funcionamento
Método DOM 0	O objeto Event é uma propriedade do objeto window .
Método proprietário attachEvent	Como objeto Event semelhante ao DOM, ou;
	Como propriedade do objeto window .
Método HTML	O objeto Event é uma variável denominada event .

Veja o exemplo a seguir para entender melhor o uso do objeto **Event** no Internet Explorer:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7
8 <body>
9
10 <p>Clique aqui...</p>
11
12 <script>
13
14 document.attachEvent('onclick', function(event) {
15
16     console.log(event);
17 }, false);
18
19 </script>
20 </body>
21 </html>
```

6.4.2.1. Propriedades do objeto Event no Internet Explorer

Veja agora as principais propriedades do objeto **Event** no Internet Explorer:

- **returnValue**

A propriedade **returnValue** apresenta valores booleanos, ou seja, **true** ou **false**, e cancela o comportamento padrão que um navegador possui para um evento quando apresentar o valor **false**. É a propriedade equivalente a **preventDefault** em navegadores que usam DOM.

JavaScript

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7
8  <body>
9
10 <p>Clique aqui...</p>
11
12 <script>
13
14 document.attachEvent('onclick', function(event) {
15
16     console.log(event.returnValue);
17
18 }, false);
19
20 </script>
21 </body>
22 </html>
```

- **cancelBubble**

Essa propriedade também trabalha com valores booleanos. Como seu valor padrão é **false**, deve ser definida como **true**, para que a propagação do evento seja cancelada. É a propriedade equivalente a **stopPropagation** em navegadores que usam DOM.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7
8  <body>
9
10 <p>Clique aqui...</p>
11 <script>
12
13 document.attachEvent('onclick', function(event) {
14
15     console.log(event.cancelBubble);
16
17 }, false);
18
19 </script>
20 </body>
21 </html>
```

- **srcElement**

É a propriedade equivalente a **target** em navegadores que usam DOM, portanto, ela vai retornar o objeto alvo do evento. É uma propriedade somente leitura.

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7
8  <body>
9
10     <p>Clique aqui...</p>
11
12 <script>
13
14 document.attachEvent('onclick', function(event) {
15
16     console.log(event.srcElement);
17
18 }, false);
19
20 </script>
21 </body>
22 </html>
```

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Os eventos tem a função de controlar as ações dos visitantes de uma página e definir um comportamento para ela quando as ações são criadas. Eles podem ser aplicados a botões e outros elementos que compõem uma página;
- As ações a serem realizadas na produção de um evento são determinadas pelos manipuladores de evento. Para cada tipo de ação do usuário existe um manipulador de evento específico. Eles se dividem em: manipuladores do mouse, eventos do teclado, eventos de frame/objeto e eventos de formulário;
- O objeto **Event** é um objeto criado quando um evento ocorre, armazenando diversas informações sobre um evento. Normalmente, os eventos são combinados com funções, e apenas quando o evento ocorrer é que essas funções serão executadas;
- No Internet Explorer, o objeto **Event** funciona de modos diferentes, de acordo com o método que você escolher para manipular o evento.

6

Eventos

Teste seus conhecimentos

Ana Gesar
778.6008.
Ano 2009.



IMPACTA
EDITORA

1. O evento onmouseup é utilizado quando?

- a) No momento em que qualquer um dos botões do mouse que já esteja pressionado é solto (incluindo o botão do meio).
- b) O ponteiro do mouse for retirado de cima do link.
- c) Posicionamos o ponteiro do mouse sobre um link.
- d) Certos elementos forem clicados.
- e) Nenhuma das alternativas anteriores está correta.

2. O evento onkeydown é utilizado quando?

- a) Este manipulador de evento é utilizado para executar uma ação assim que uma tecla é pressionada.
- b) Este manipulador de evento é utilizado para executar uma ação assim que uma tecla já pressionada é solta.
- c) Este manipulador de evento é utilizado para executar uma ação assim que uma tecla é pressionada e depois enviar um aviso na tela.
- d) Este manipulador de evento é utilizado para executar uma ação assim que uma tecla é pressionada é solta e depois enviar um aviso na tela.
- e) Nenhuma das alternativas anteriores está correta.

3. Qual o navegador que o objeto Event trabalha de maneira diferente?

- a) Google Chrome
- b) Safari
- c) Opera
- d) Internet Explorer
- e) Mozilla Firefox.

4. Qual propriedade não faz parte das principais do objeto Event?

- a) Bubbles
- b) Cancelable
- c) Type
- d) Number
- e) Timestamp

5. Quais são os principais métodos do objeto Event?

- a) initEvent, preventEvent, stopPropagation
- b) initEvent, defaultEvent, preventEvent
- c) initEvent, preventEvent
- d) defaultEvent, resumeEvent, preventEvent
- e) Nenhuma das alternativas anteriores está correta.



6

Eventos Mãos à obra!

Ana G. Cesar
778.6000.

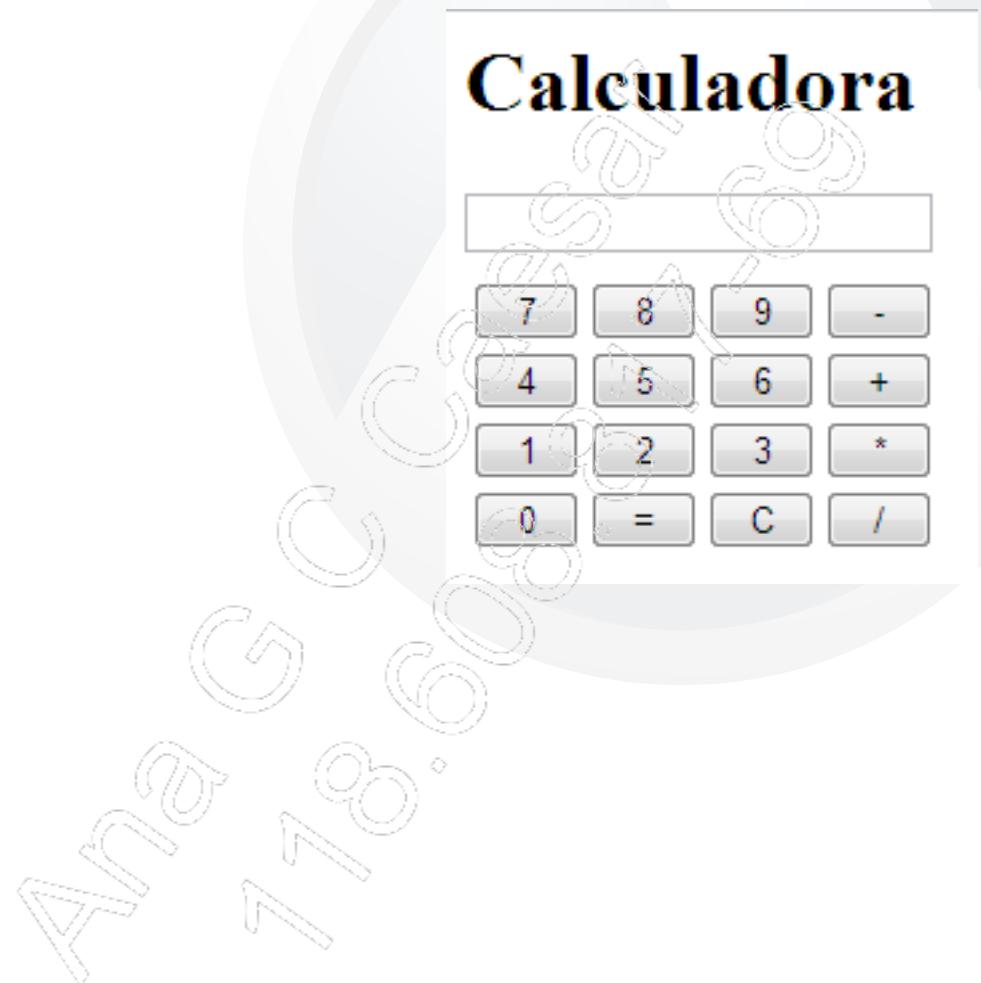


IMPACTA
EDITORA

Laboratório 1

A – Manipulando eventos: Criando uma calculadora com as quatro operações básicas

1. Crie um novo arquivo de texto e renomeie com a extensão **.html**;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD**, **BODY**;
4. Crie o HTML dentro da tag **BODY** para gerar o seguinte modelo de calculadora:



```
<html>
<head>
    <style>
        input[type=button]{ width:40px; margin:0px; }
        input[type=text]{ width:175px; }
    </style>
<body>
    <h1>Calculadora</h1>
    <form name="calc">
        <div>
            <input type="text" name="visor" />
        </div>
        <table>
            <tbody>
                <tr>
                    <td><input type="button" value="7"/></td>
                    <td><input type="button" value="8"/></td>
                    <td><input type="button" value="9"/></td>
                    <td><input type="button" value="-"/></td>
                </tr>
                <tr>
                    <td><input type="button" value="4"/></td>
                    <td><input type="button" value="5"/></td>
                    <td><input type="button" value="6"/></td>
                    <td><input type="button" value="+"/></td>
                </tr>
                <tr>
                    <td><input type="button" value="1"/></td>
                    <td><input type="button" value="2"/></td>
                    <td><input type="button" value="3"/></td>
                    <td><input type="button" value="*"/></td>
                </tr>
                <tr>
                    <td><input type="button" value="0"/></td>
                    <td><input type="button" value="="/></td>
                    <td><input type="button" value="C"/></td>
                    <td><input type="button" value="/"></td>
                </tr>
            </tbody>
        </table>
    </form>
</body>
</head>
</html>
```

5. Inclua dentro da tag **HEAD** a tag **<script></script>**;

- Crie uma função que pode ser chamada **fnValor** que recebe um parâmetro;
- Atribua esta função ao evento **click** de cada botão do formulário HTML, com exceção do botão **igual(=)** e **C (limpar)**. Esta função irá receber os valores selecionados para operação matemática e mostrar no visor.

- Crie uma função que pode ser chamada **fnLimparVisor**:
 - Atribua esta função ao evento **click** do botão **C (limpar)**. Esta função irá limpar o visor.
 - Crie uma função que pode ser chamada **fnCalcular**:
 - Atribua esta função ao evento **click** do botão **igual (=)**. Esta função irá realizar o cálculo. Para realizar o cálculo, recupere o valor que está preenchido no visor e utilize a função **eval()**.
6. Mostre o resultado no campo **visor** a cada operação realizada.



Objetos

7

- ✓ Conhecendo a Programação Orientada a Objetos (POO);
- ✓ Criando objetos;
- ✓ Propriedades;
- ✓ Métodos;
- ✓ Objetos nativos;
- ✓ BOM (Browser Object Model);
- ✓ JSON.



IMPACTA
EDITORA

7.1. Introdução

Este capítulo aborda um tema bastante importante para a linguagem JavaScript: os objetos. Especialmente, se considerarmos que quase todos os elementos desta linguagem são objetos e que a manipulação deles está presente em todos os códigos, dos mais simples aos mais complexos.

Por exemplo, com exceção de **null** e de **undefined**, todos os tipos primitivos são tratados como objetos. Além de possuírem as características comuns aos objetos, podemos atribuir propriedades a eles.

Apesar da possibilidade de criação e utilização de objetos, a maneira que o desenvolvedor vai programar não muda muito, podendo seguir a mesma lógica para utilizar as funções, as sintaxes e os outros recursos já apresentados.

7.1.1. Categorias de objeto

Os objetos da JavaScript podem ser classificados em objetos customizados, nativos ou do ambiente de hospedagem.

A tabela a seguir descreve as diferentes categorias de objetos:

Categoria	Descrição
Objetos customizados	São criados pelo desenvolvedor.
Objetos nativos	São predefinidos e próprios da linguagem JavaScript.
Objetos do ambiente de hospedagem	São próprios do dispositivo utilizado para interpretar a linguagem, como um navegador gráfico.

7.2. Conhecendo a Programação Orientada a Objetos (POO)

A partir do início da década de 90, o conceito de orientação a objetos tornou-se bastante importante. Isso porque os sistemas orientados a objetos oferecem uma visão mais natural, uma forte arquitetura e um alto grau de reusabilidade do código.

Basicamente, a Programação Orientada a Objetos (POO) apresenta uma maneira especial de programar, que se aproxima mais da realidade em comparação aos outros tipos de programação.

Para compreender melhor sua estrutura, é necessário conhecer alguns conceitos básicos importantes:

- **Objetos simples**

O objeto mais simples da linguagem Java Script é o tipo de dado **Object**. Ele é implementado como uma coleção de propriedades nomeadas, que podem ser criadas e adicionadas a qualquer tempo, não necessitando que sejam predefinidas em um construtor ou em uma declaração de objeto.

- **Definindo classes**

Definimos classe como sendo um conjunto de objetos agrupados por possuírem comportamento e características semelhantes. Sendo assim, as propriedades ou os atributos de um objeto, bem como seu comportamento, são descritos a partir da definição de uma classe.

O comportamento de um objeto é a funcionalidade que pode ser aplicada a ele e é descrito por um método. Embora os métodos sejam correspondentes a procedimentos e funções, eles são capazes de manipular somente os atributos definidos para um objeto e suas variáveis locais.

- **Herança e conceito de Prototype**

Protótipo (Prototype) é a denominação do objeto a partir do qual outro objeto herda propriedades. Esse mecanismo de herança é a maior diferença entre a programação orientada a objetos e os outros tipos de programação.

Quando a hierarquia de classes é planejada de forma adequada, temos a base para que um código possa ser reutilizado, permitindo poupar esforço e tempo no desenvolvimento.

Antes que uma expressão retorne uma propriedade, a linguagem JavaScript avalia, primeiramente, se a propriedade foi definida diretamente no objeto, caso contrário, verifica o protótipo do objeto e, caso haja uma cadeia de protótipos, continua a verificação até localizar o protótipo raiz.



- **Métodos**

Definimos métodos como sendo ações executadas pelo objeto ou ações executadas sobre ele. Por exemplo, quando você abre a porta de sua casa, executa a ação de abrir sobre essa porta.

- **Polimorfismo**

O polimorfismo é um mecanismo por meio do qual você pode selecionar as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução. Os códigos que o polimorfismo permite criar apresentam diversas características, dentre as quais destacamos a lógica, a robustez, a legibilidade e a limpeza.

- **Subclasse**

A cadeia de protótipos é utilizada, por padrão, para implementar a herança dos métodos a partir de uma superclasse. Com isso, os métodos que forem definidos em uma subclasse irão substituir aqueles definidos na superclasse.

Quando a subclasse é definida, há a necessidade de chamar o construtor da superclasse a fim de inseri-la na cadeia de protótipos. Em contrapartida, para que o construtor da superclasse possa ser chamado para cada instância de subclasse, é necessário que o código seja adicionado explicitamente ao construtor da subclasse.

- **Encapsulamento**

Quando você cria algo dentro de um construtor, será acessível tanto a partir dele quanto do restante do script, isto é, o escopo será público. Para que possa ser acessado somente a partir do próprio construtor e de seus métodos, a alternativa é simular variáveis privadas por meio do escopo local. Essa é a forma que temos, também, de utilizar o recurso de encapsulamento disfarçado, já que ele ainda não existe na linguagem JavaScript.

JavaScript

Veja a seguir um exemplo da utilização de encapsulamento:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  function Carro(hp) {
9
10     var hpMax = 500;//Encapsulamento
11     var hpPadrao = 100;//Encapsulamento
12
13     this.hp = (hp > 0 && hp <= hpMax) ? hp : hpPadrao;
14
15     this.ligado = false;
16
17     this.ligar = function() {
18
19         this.logado = true;
20
21     }
22
23 }
24
25 var Gol = new Carro(94);
26
27 console.log(Gol); //Objeto
28 console.log(Gol.hp); //94
29 console.log(Gol.hpMax); //undefined
30
31
32 </script>
33 </head>
34
35 <body>
36
37 </body>
38 </html>
```

O único problema da utilização dos métodos privados é que não permitem o acesso às propriedades e aos métodos públicos diretamente, a não ser que o escopo da função seja alterado. Para alterar o escopo em que a função é executada, a função **Function.bind** deve ser utilizada:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  Function.prototype.bind = function(scope) {
9    var _function = this;
10
11   return function() {
12     return _function.apply(scope, arguments);
13   }
14 }
15
16 Carro = {
17   name: "carro",
18
19   velocidade: 0,
20
21   acoes: function() {
22
23     acelerar = function(velocidade) {
24       this.velocidade += velocidade;
25     }.bind(this);
26
27     acelerar(20);
28   }
29 }
30
31
32 Carro.acoes();
33
34 console.log(Carro);
35
36
37 </script>
38 </head>
39
40 <body>
41
42 </body>
43 </html>
```

7.3. Criando objetos

A instanciação (criação) de um objeto é um processo por meio do qual é criado o exemplar de uma classe para podermos utilizá-la. É um processo obrigatório, isto é, uma classe deve ser instanciada para que possamos trabalhar com ela. Sendo assim, devemos criar sua instância, que nada mais é que um objeto com o tipo de dado que foi definido pela classe.

Para a criação de um objeto a partir de uma classe, utilize a instrução `new`, como no exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var objeto = new Object();
9
10 console.log(objeto);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

7.4. As propriedades

O operador ponto (.) é utilizado para acessar as propriedades e os métodos de objetos. Este procedimento é semelhante ao realizado em outras linguagens de programação.

Para isso, é necessário especificar o nome do objeto, com o seguinte formato: deve conter um operador ponto à sua frente e, então, o nome da propriedade a ser acessada. Observe o exemplo adiante:

```
meuObjeto.minhaPropriedade
```

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var objeto = new Object();
9
10 objeto.nome = 'Objeto da Silva';
11
12 console.log(objeto);
13
14 </script>
15 </head>
16
17 <body>
18
19 </body>
20 </html>
```

7.5. Os métodos

Os métodos podem ser definidos como sendo ações executadas pelos objetos, ou seja, são funções que pertencem aos objetos.

A maneira de chamar métodos é parecida com a empregada para chamar as propriedades. No entanto, ao final da sintaxe, é necessário colocar entre parênteses os parâmetros passados aos métodos, conforme o exemplo adiante:

```
meuObjeto.meuMetodo(parâmetro1, parâmetro2)
```

Quando o método não recebe parâmetros, os parênteses devem ser especificados vazios, conforme o exemplo a seguir:

```
meuObjeto.meuMetodo()
```

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var objeto = new Object();
9
10 objeto.mudaNome = function(novoNome) {
11
12     this.nome = novoNome;
13
14 }
15
16 objeto.mudaNome('Objeto J. S.');
17
18 console.log(objeto);
19
20 </script>
21 </head>
22
23 <body>
24
25 </body>
26 </html>
```

7.6. Objetos nativos

Objetos nativos são aqueles que já estão embutidos na linguagem JavaScript. Veja a seguir como funcionam os principais objetos nativos.

7.6.1. Objeto Array

A linguagem Java Script não possui um tipo de dado array. Em vez disso, para trabalhar com arrays, você fará uso do objeto **Array** e de seus métodos. Além deles, que permitirão que você manipule os arrays normalmente, existem propriedades para utilização com as expressões regulares e uma que permite definir o comprimento do array.

O que é array?

Um array é um conjunto de valores ordenados, ao qual nos referimos por meio de um nome e num índice.

Considere, por exemplo, o array **func**, contendo os nomes dos funcionários de uma empresa, ordenados por um número de registro. Nesse caso, **func[1]** representará o primeiro funcionário e assim por diante.

7.6.1.1. Criando objetos Array

Para a criação de um objeto **Array**, você pode utilizar uma das seguintes sintaxes:

```
var arr = new Array(elemento0, elemento1, ..., elementoN);
```

```
var arr = Array(elemento0, elemento1, ..., elementoN);
```

```
var arr = [elemento0, elemento1, ..., elementoN];
```

Em que **elemento0**, **elemento1**, ..., **elementoN** é uma lista de valores, que constitui os elementos do array. Quando estes elementos são especificados na sintaxe, o array é criado com os valores definidos e a propriedade **length** exibe o número de argumentos.

A última sintaxe é denominada array literal ou array initializer e é a utilizada com mais frequência.

JavaScript

Veja o exemplo a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var meses = [
9      'Jan', 'Fev', 'Mar',
10     'Abr', 'Mai', 'Jun',
11     'Jul', 'Ago', 'Set',
12     'Out', 'Nov', 'Dez'
13 ];
14
15 console.log(meses);
16
17 </script>
18 </head>
19
20 <body>
21
22 </body>
23 </html>
```

7.6.1.2. Propriedades do objeto Array

As propriedades do objeto **Array** são as seguintes:

- **constructor**: O retorno é a função que criou o protótipo do objeto **Array**. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var meses = [
9   'Jan', 'Fev', 'Mar',
10  'Abr', 'Mai', 'Jun',
11  'Jul', 'Ago', 'Set',
12  'Out', 'Nov', 'Dez'
13];
14
15 console.log(meses.constructor);
16
17</script>
18</head>
19
20<body>
21
22</body>
23</html>
```

- **length**: Permite configurar o número de elementos no array ou retorna esse número. Veja o exemplo:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var meses = [
9      'Jan', 'Fev', 'Mar',
10     'Abr', 'Mai', 'Jun',
11     'Jul', 'Ago', 'Set',
12     'Out', 'Nov', 'Dez'
13 ];
14
15 console.log('Total de '+meses.length+' meses');
16
17 </script>
18 </head>
19
20 <body>
21 </body>
22 </html>
```

- **prototype:** Permite adicionar propriedades e métodos a um objeto Array.
Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 Array.prototype.procurar = function(texto){
9
10    for (var i = 0; i < this.length, atual = this[i]; i++) {
11
12        if(atual == texto) return "'" + texto + "' foi encontrado no index ' + i;
13
14    }
15
16    return 'Não foi encontrado!';
17
18 }
19
20 var meses = [
21     'Jan', 'Fev', 'Mar',
22     'Abr', 'Mai', 'Jun',
23     'Jul', 'Ago', 'Set',
24     'Out', 'Nov', 'Dez'
25 ];
26
27 console.log(meses.procurar('Ago'));
28
29 </script>
30 </head>
31
32 <body>
33
34 </body>
35 </html>
```

7.6.1.3. Métodos do objeto Array

Os métodos do objeto **Array** são os seguintes:

- **concat()**: Une dois ou mais arrays e retorna uma cópia dos arrays que foram unidos. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americaDoSul = ['Brasil', 'Argentina', 'Venezuela'];
9 var americaDoNorte = ['Estados Unidos', 'Canadá', 'México'];
10 var americas = [].concat(americaDoSul, americaDoNorte);
11
12 console.log(americas);
13
14 </script>
15 </head>
16
17 <body>
18
19 </body>
20 </html>
```

- **indexOf()**: Procura por um elemento no array, retornando a posição dele. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.indexOf('Japão'));
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

- **join()**: Junta todos os elementos de um array dentro de uma string. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.join(' / '));
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

- **lastIndexOf()**: Semelhante ao método **indexOf()**, procura um elemento em um array e retorna a sua posição. A diferença é que este método procura o elemento a partir do final do array. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.lastIndexOf('Canadá'));
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

- **pop()**: Exclui o último elemento de um array e retorna o elemento excluído. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.pop(), americas);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

- **push()**: Adiciona elementos ao final do array e retorna o comprimento do array após a adição. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 americas.push('Peru');
11
12 console.log(americas);
13
14 </script>
15 </head>
16
17 <body>
18
19 </body>
20 </html>
```

- **reverse()**: Inverte a ordem dos elementos dentro do array. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 americas = americas.reverse();
11
12 console.log(americas);
13
14 </script>
15 </head>
16
17 <body>
18
19 </body>
20 </html>
```

JavaScript

- **shift()**: Exclui o primeiro elemento do array e retorna o elemento excluído. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.shift(), americas);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

- **slice()**: Seleciona uma parte do array e retorna esse novo array. Veja o exemplo. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.slice(1, 4));
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

- **sort()**: Ordena os elementos do array. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.sort());
11
12 </script>
13 </head>
14
15 <body>
16 </body>
17 </html>
```

- **splice()**: Permite adicionar ou excluir elementos do array. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 americas.splice(2,0, "Peru", "Chile");
11
12 console.log(americas);
13
14 </script>
15 </head>
16
17 <body>
18 </body>
19 </html>
```

- **toString()**: Faz a conversão de um array em uma string, retornando o resultado. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.toString());
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

- **unshift()**: Adiciona elementos ao começo da array e retorna o comprimento após a adição. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 americas.unshift("Colômbia", "Uruguai");
11
12 console.log(americas);
13
14 </script>
15 </head>
16
17 <body>
18
19 </body>
20 </html>
```

- **valueOf()**: Retorna o valor primitivo de um array. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var americas = ["Brasil", "Argentina", "Venezuela", "Estados Unidos", "Canadá", "México"];
9
10 console.log(americas.valueOf());
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

7.6.2. Objeto Boolean

Este objeto, cuja finalidade é converter um valor que não seja booleano em um valor booleano, pode ser considerado uma embalagem para o tipo de dado booleano primitivo.

Para criar um objeto **Boolean**, use a seguinte sintaxe:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var js = new Boolean('JavaScript');
9
10 </script>
11 </head>
12
13 <body>
14
15 </body>
16 </html>
```

É importante não confundir os valores booleanos primitivos **true** e **false** com os valores **true** e **false** do objeto booleano. Quando passado a uma instrução condicional, qualquer valor diferente de **undefined**, **null**, **0**, **NaN**, uma string vazia ou mesmo do valor **false** do objeto booleano, é alterado para **true**.

7.6.3. Objeto Date

Este objeto é um construtor de datas e horários, a partir da extração desses dados diretamente do sistema operacional utilizado pelo usuário. Para isso, a linguagem JavaScript é executada dentro do calendário interno do sistema operacional, onde tem acesso ao grupo data-hora atual. Quando você altera o relógio do computador, por exemplo, o objeto **Date** utilizará os novos dados.

Para criar um objeto **Date**, podemos utilizar as seguintes sintaxes:

```
var d = new Date();  
  
var d = new Date(argumento_em_milissegundos);  
  
var d = new Date(argumento_em_string);  
  
var d = new Date(ano, mês, dia, hora, minuto, segundo, milissegundo)
```

A data-base utilizada como referência inicial para a contagem de tempo é 01 de janeiro de 1970 às 00:00:00. Esta referência inicial é denominada Unix Time ou POSIX Time. Qualquer momento anterior ou posterior à data-base é calculado em milissegundos transcorridos em relação a ela. Assim, o objeto **Date** é expresso em uma faixa que vai de -100,000,000 a 100,000,000 dias em relação a 01 de Janeiro de 1970 UTC.



- **new Date()**

Quando o construtor **Date** é utilizado sem argumentos, o objeto criado conterá a data-hora atual. Veja um exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var data = new Date();
9
10 console.log(data);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

Lembre-se sempre que:

- Independentemente do sistema operacional utilizado e de seu idioma, os dias da semana e os meses são expressos em inglês.
- Cada navegador tem seu próprio padrão para a string de saída de data-hora criada pelo objeto Date. Não há um padrão.

- **new Date(argumento_em_milissegundos)**

Quando o construtor **Date** é utilizado com o argumento em milissegundos, o objeto criado conterá a data-hora correspondente ao número de milissegundos especificado.

Considerando os milissegundos transcorridos a partir da data-base, quando o argumento é positivo, a data é posterior a ela e quando o argumento é negativo, a data é anterior. Observe os exemplos a seguir:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var data = new Date(606193200000);
9
10 console.log(data);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```



- **new Date(argumento_em_string)**

Quando o conjunto data-hora é passado como argumento para o construtor, é possível criar um grupo data-hora predefinido. Diversas sintaxes podem ser utilizadas nessa opção:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var data = new Date("December 21, 2012 21:12");
9
10 console.log(data);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

 Os meses devem ser escritos sempre em inglês, e suas respectivas abreviaturas são formadas pelas três primeiras letras de cada mês, também em inglês.

 Nem todas as sintaxes possíveis são renderizadas corretamente por todos os navegadores.

- **new Date(ano, mês, dia, hora, minuto, segundo, milissegundo)**

Quando o construtor Date é utilizado com os sete argumentos do grupo data-hora, o objeto criado conterá a data-hora especificada. Com exceção dos argumentos ano e mês, que são obrigatórios, os demais assumirão o valor 0 (zero) quando forem omitidos.

! Para especificar a numeração adequada para os meses você deve seguir a contagem específica da linguagem JavaScript, em que Janeiro, o primeiro mês, é representado pelo número 0.

Veja um exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var data = new Date(2007, 11, 2);
9
10 console.log(data);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

7.6.3.1. Propriedades do objeto Date

As propriedades do objeto **Date** são as seguintes:

- **constructor**: Retorna a função que criou o protótipo do objeto **Date**;

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var data = new Date();
9
10 console.log(data.constructor);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

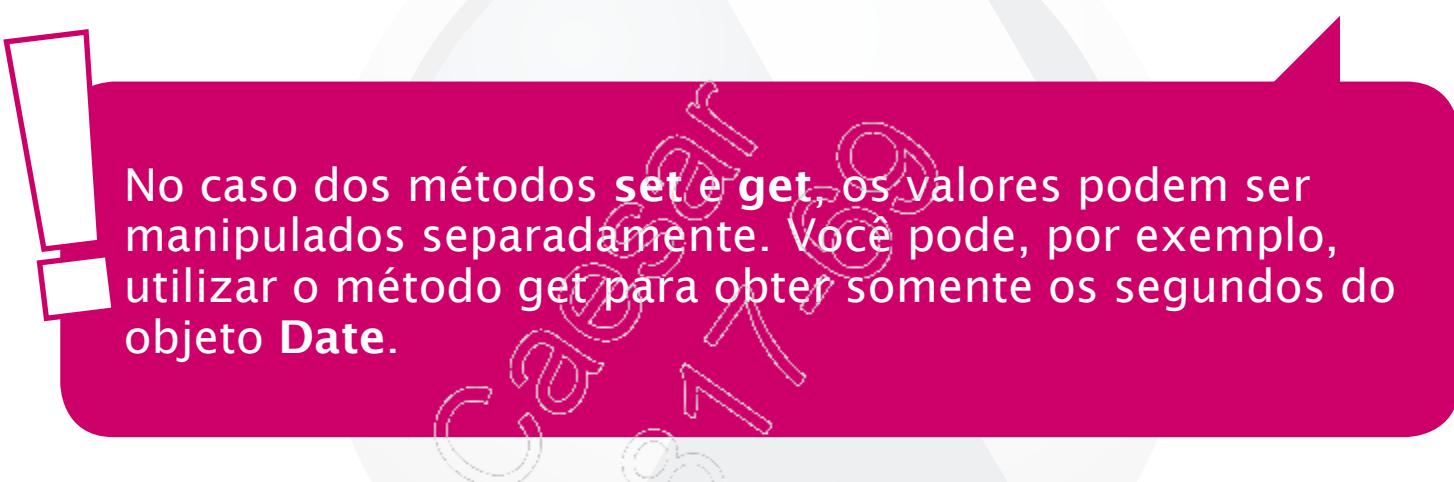
- **prototype**: Permite adicionar propriedades e métodos ao objeto **Date**.

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  Date.prototype.DIASDASEMANA = ['D', 'S', 'T', 'Q', 'Q', 'S', 'S'];
9  Date.prototype.queDiaDaSemana = function(){
10     return this.DIASDASEMANA[this.getDay()];
11 }
12 var data = new Date();
13
14 console.log(data.queDiaDaSemana());
15
16 </script>
17 </head>
18
19 <body>
20
21 </body>
22 </html>
```

7.6.3.2. Métodos do objeto Date

Os métodos do objeto **Date** podem ser classificados da seguinte forma:

- Métodos **set**: Permitem definir os valores para a data e o tempo no objeto **Date**;
- Métodos **get**: Permitem extrair os valores da data e de tempo do objeto **Date**;
- Métodos **to**: Permitem retornar os valores da string do objeto **Date**;
- Métodos **parse** e **UTC**: Permitem interpretar as strings do objeto **Date**.



No caso dos métodos **set** e **get**, os valores podem ser manipulados separadamente. Você pode, por exemplo, utilizar o método **get** para obter somente os segundos do objeto **Date**.

Os métodos do objeto **Date** são descritos na tabela a seguir:

Método	Descrição
getDate()	O retorno é o dia do mês (1 a 31).
getDay()	O retorno é o dia da semana (0 a 6).
getFullYear()	O retorno é o ano, expresso com quatro dígitos.
getHours()	O retorno a hora (0 a 23).
getMilliseconds()	O retorno são os milissegundos (0 a 999).
getMinutes()	O retorno são os minutos (0 a 59).
getMonth()	O retorno é o mês (0 a 11). Inicia em 0, que representa Janeiro.
getSeconds()	O retorno são os segundos (0 a 59).
getTime()	O retorno é o número de milissegundos transcorridos a partir da meia noite do dia 01 de Janeiro de 1970.

Método	Descrição
getTimezoneOffset()	O retorno é a diferença, em minutos, entre o tempo UTC e o tempo local.
getUTCDate()	O retorno o dia do mês, conforme o tempo universal (1 a 31).
getUTCDay()	O retorno é o dia da semana, conforme o tempo universal (0 a 6).
getUTCFullYear()	O retorno é o ano, conforme o tempo universal, expresso com quatro dígitos.
getUTCHours()	O retorno é a hora, conforme o tempo universal (0 a 23).
getUTCMilliseconds()	O retorno são os milissegundos, conforme o tempo universal (0 a 999).
getUTCMinutes()	O retorno são os minutos, conforme o tempo universal (0 a 59).
getUTCMonth()	O retorno é o mês, conforme o tempo universal (0 a 11).
getUTCSeconds()	O retorno são os segundos, conforme o tempo universal (0 a 59).
parse()	Interpreta uma string de data e retorna o número de milissegundos transcorridos a partir de 01 de Janeiro de 1970.
 setDate()	Permite configurar o dia do mês de um objeto Date .
 setFullYear()	Permite configurar o ano, expresso com quatro dígitos, de um objeto Date .
 setHours()	Permite configurar a hora do objeto Date .
 setMilliseconds()	Permite configurar os milissegundos de um objeto Date .
 setMinutes()	Permite configurar os minutos de um objeto Date .
 setMonth()	Permite configurar o mês de um objeto Date .
 setSeconds()	Permite configurar os segundos de um objeto Date .
 setTime()	Permite definir uma data e um horário por meio da adição ou subtração de milissegundos da data-base.
 setUTCDate()	Definir o dia do mês de um objeto Date , conforme o tempo universal.
 setUTCFullYear()	Define o ano de um objeto Date , com quatro dígitos, conforme o tempo universal.

Método	Descrição
setUTCHours()	Define a hora de um objeto Date , conforme o tempo universal.
setUTCMilliseconds()	Define os milissegundos de um objeto Date , conforme o tempo universal.
setUTCMinutes()	Define os minutos de um objeto Date , conforme o tempo universal.
setUTCMonth()	Define o mês de um objeto Date , conforme o tempo universal.
setUTCSeconds()	Define os segundos de um objeto Date , conforme o tempo universal.
toDateString()	Responsável pela conversão de parte da data do objeto Date em uma string legível.
toISOString()	O retorno é a data como uma string, de acordo com o padrão ISO.
toJSON()	O retorno é a data como uma string, de acordo com o formato de data JSON.
toLocaleDateString()	Retorna parte da data do objeto Date como uma string, conforme convenções locais.
toLocaleTimeString()	Retorna parte do tempo do objeto Date como uma string, conforme convenções locais.
toLocaleString()	Responsável pela conversão do objeto Date em uma string, conforme convenções locais.
toString()	Responsável pela conversão do objeto Date em uma string.
toTimeString()	Responsável pela conversão de parte do tempo do objeto Date em uma string.
toUTCString()	Responsável pela conversão do objeto Date em uma string, conforme o tempo universal.
UTC()	O retorno é o número de milissegundos em uma string de data, transcorrido a partir da data-base e conforme o tempo universal.
valueOf()	O retorno é o valor primitivo do objeto Date .

Veja um exemplo da utilização de alguns métodos do objeto **Date**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var Data = new Date();
9
10 Data.setTime(0, 0, 0);
11
12 var dia = Data.getDate();
13 var mes = Data.getMonth() + 1;
14 var ano = Data.getFullYear();
15 var hora = Data.getHours();
16 var minuto = Data.getMinutes();
17 var segundo = Data.getSeconds();
18
19 console.log(dia+'/'+mes+'/'+ano+' '+hora+':'+minuto+':'+segundo);
20
21 </script>
22 </head>
23
24 <body>
25
26 </body>
27 </html>
```

7.6.4. Objeto Math

Diferentemente do objeto **Date**, o objeto **Math** não é um construtor. Ele oferece propriedades e métodos para as constantes e funções matemáticas, que podem ser chamados utilizando **Math** como objeto, mesmo sem criá-lo.

! Não é possível criar um objeto **Math, você sempre fará uso do objeto predefinido.**

7.6.4.1. Propriedades do objeto Math

Estas propriedades armazenam constante matemáticas notáveis, como o número de Euller (E) e o valor de π (pi).

As propriedades do objeto **Math** são descritas na tabela a seguir:

Propriedade	Descrição
Math.E	O retorno é o número de Euller, isto é, aproximadamente 2.718.
Math.LN2	O retorno é o logaritmo natural de 2, ou seja, aproximadamente 0.693.
Math.LN10	O retorno é o logaritmo natural de 10, ou seja, aproximadamente 2.302.
Math.LOG2E	O retorno é o logaritmo de base 2 de E(número de Euller), isto é, aproximadamente 1.442.
Math.LOG10E	O retorno é o logaritmo de base 10 de E, ou seja, aproximadamente 0.434.
Math.PI	O retorno é π , aproximadamente 3.14.
Math.SQRT1_2	O retorno é a raiz quadrada de $\frac{1}{2}$, ou seja, aproximadamente 0.707.
Math.SQRT2	O retorno é a raiz quadrada de 2, isto é, aproximadamente 1.414.



Veja um exemplo da utilização de algumas propriedades do objeto **Math**:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  console.log('PI', Math.PI);
9  console.log('E', Math.E);
10 console.log('LN2', Math.LN2);
11 console.log('LN10', Math.LN10);
12 console.log('LOG2E', Math.LOG2E);
13 console.log('LOG10E', Math.LOG10E);
14 console.log('SQRT1_2', Math.SQRT1_2);
15 console.log('SQRT2', Math.SQRT2);
16
17 </script>
18 </head>
19
20 <body>
21
22 </body>
23 </html>
```

7.6.4.2. Métodos do objeto Math

Os métodos do objeto **Math** são funções matemáticas padrão, como as trigonométricas, logarítmicas ou exponenciais.

Os métodos do objeto **Math** são descritos na tabela a seguir:

Método	Descrição
abs(x)	O retorno é o valor absoluto de x.
acos(x)	O retorno é o arco cosseno, em radianos, de x.
asin(x)	O retorno é o arco seno, em radianos, de x.
atan(x)	O retorno é o arco tangente de x, sendo um valor numérico entre $-\pi/2$ e $\pi/2$ radianos.

Método	Descrição
atan2(y,x)	O retorno é o arco tangente do quociente de seus argumentos.
ceil(x)	O retorno é x, arredondado para o próximo valor inteiro maior.
cos(x)	O retorno é o cosseno de x. O valor de x é expresso em radianos.
exp(x)	O retorno é o valor de E^x .
floor(x)	O retorno é x, arredondado para o próximo valor menor.
log(x)	O retorno é o logaritmo natural, de base E, de x.
max(x,y,z,...,n)	O retorno é o número com valor mais alto.
min(x,y,z,...,n)	O retorno é o número com valor mais baixo.
pow(x,y)	O retorno é o valor de x sobre y.
random()	O retorno é um número aleatório entre 0 e 1.
round(x)	Arredonda o valor de x para o próximo valor inteiro.
sin(x)	O retorno é o seno de x. O valor de x é expresso em radianos.
sqrt(x)	O retorno é a raiz quadrada de x.
tan(x)	O retorno é a tangente de um ângulo.

Todos os métodos trigonométricos de **Math** utilizam os argumentos em radianos.

Veja um exemplo da utilização de alguns métodos do objeto **Math**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8     function gerarNumeroAte(n) {
9         return parseInt(Math.random() *n);
10    }
11
12    function areaDoCirculo(r) {
13        return Math.PI * Math.pow(r, 2);
14    }
15
16    function hipotenusa(a, b) {
17
18        return Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));
19    }
20
21
22    console.log('gerarNumeroAte: '+gerarNumeroAte(100));
23    console.log('areaDoCirculo: '+areaDoCirculo(10));
24    console.log('hipotenusa: '+hipotenusa(4, 3));
25
26 </script>
27 </head>
28
29 <body>
30
31 </body>
32 </html>
```

7.6.5. Objeto Number

O objeto **Number** é como uma embalagem para os valores numéricos primitivos. Este tipo de objeto possui propriedades para as constantes numéricas, como valores mínimo e máximo. Tais valores não podem ser alterados.

A sintaxe para criar um objeto **Number** é a seguinte:

```
var num = new Number(valor);
```

 Quando o valor não puder ser convertido em um número, o retorno será **NaN**, isto é, **Not-a-Number** (não é um número).

7.6.5.1. Propriedades do objeto Number

As propriedades do objeto **Number** são descritas na tabela a seguir:

Propriedade	Descrição
constructor	O retorno é a função que criou o protótipo do objeto Number .
MAX_VALUE	O retorno é o maior número possível na linguagem JavaScript.
MIN_VALUE	O retorno é o menor número possível na linguagem JavaScript.
NEGATIVE_INFINITY	Representação da infinidade negativa.
POSITIVE_INFINITY	Representação da infinidade positiva.
prototype	Permite adicionar propriedades e métodos ao objeto Number .

Veja um exemplo da utilização de algumas propriedades do objeto **Number**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log('MAX_VALUE', Number.MAX_VALUE);
9 console.log('MIN_VALUE', Number.MIN_VALUE);
10 console.log('NEGATIVE_INFINITY', Number.NEGATIVE_INFINITY);
11 console.log('POSITIVE_INFINITY', Number.POSITIVE_INFINITY);
12
13 </script>
14 </head>
15
16 <body>
17
18 </body>
19 </html>
```

7.6.5.2. Métodos do objeto Number

Os métodos do objeto **Number** são descritos na tabela a seguir:

Método	Descrição
toExponential(x)	Responsável pela conversão de um número em uma notação exponencial.
toFixed(x)	Permite formatar o número com a quantidade de dígitos especificada (x) após o ponto decimal.
toPrecision(x)	Permite formatar o número com o comprimento especificado para x.
toString()	Responsável pela conversão do objeto Number em uma string.
valueOf()	O retorno é o valor primitivo do objeto Number .

Veja um exemplo da utilização de alguns métodos do objeto **Number**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function dollar(n) {
9     var number = new Number(n);
10    return '$' + number.toFixed(2);
11 }
12
13 console.log(dollar(2.568));
14
15 </script>
16 </head>
17
18 <body>
19
20 </body>
21 </html>
```

7.6.6. Objeto String

Uma string, que é um objeto global, é utilizada para criar instâncias de strings, as quais têm por finalidade o armazenamento de textos. Por meio das propriedades e métodos você consegue manipular os textos armazenados no objeto **String**.

Para criar um objeto **String**, uma das seguintes sintaxes deve ser utilizada:

```
var txt = new String("string");
```

ou

```
var txt = "string";
```

7.6.6.1. Propriedades do objeto String

As propriedades do objeto **String** são as seguintes:

- **constructor**: O retorno é a função que criou o protótipo do objeto **String**. Veja o exemplo:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var texto = new String('JavaScript');
9
10 console.log(texto.constructor);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```

JavaScript

- **length:** O retorno é o comprimento da string. Veja o exemplo:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var texto = new String('JavaScript');
9
10 console.log(texto.length);
11
12 </script>
13 </head>
14
15 <body>
16
17 </body>
18 </html>
```



- **prototype:** Permite adicionar propriedades e métodos ao objeto **String**.
Veja o exemplo:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8      String.prototype.converteParaUnicode = function(){
9
10         var resultado = [];
11         for(var i = 0; i < this.length; i++){
12
13             resultado.push(this[i].charCodeAt());
14
15         }
16
17         return resultado;
18     }
19
20
21     var texto = new String('JavaScript');
22
23     console.log(texto.converteParaUnicode());
24
25 </script>
26 </head>
27
28 <body>
29
30 </body>
31 </html>
```

7.6.6.2. Métodos do objeto String

Os métodos do objeto **String** são descritos na tabela a seguir:

Método	Descrição
charAt()	O retorno é o caractere do índice especificado.
charCodeAt()	O retorno é o valor Unicode do caractere no índice especificado.
concat()	Permite unir duas ou mais strings e retorna uma cópia das strings após a junção.
fromCharCode()	Responsável pela conversão dos valores Unicode em caracteres.
indexOf()	O retorno é a posição da primeira ocorrência localizada de um valor especificado em uma string.
lastIndexOf()	O retorno é a posição da última ocorrência localizada de um valor especificado em uma string.
match()	Permite procurar por uma correspondência entre uma expressão regular e uma string, retornando a correspondência localizada.
replace()	Permite procurar por uma correspondência entre uma substring ou expressão regular e uma string, substituindo a substring correspondente por uma nova.
search()	Permite procurar por uma correspondência entre uma expressão regular e uma string, retornando a posição da correspondência localizada.
slice()	Permite extrair uma parte da string, retornando a nova string extraída.
split()	Permite dividir uma string em um array de substrings.
substr()	Permite extrair os caracteres de uma string, a partir de uma posição inicial especificada, até o número especificado de caracteres.
substring()	Permite extrair os caracteres de uma string que estiverem entre dois índices.
toLowerCase()	Responsável por converter uma string em letras minúsculas.

Método	Descrição
toUpperCase()	Responsável por converter uma string em letras maiúsculas.
valueOf()	O retorno é o valor primitivo de um objeto String .

Veja um exemplo da utilização de alguns métodos do objeto **String**:

```

1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log("JavaScript".toUpperCase());
9 console.log("JavaScript".toLowerCase());
10 console.log("JavaScript".replace(/Java/, ''));
11 console.log("JavaScript".split(''));
12
13 </script>
14 </head>
15
16 <body>
17
18 </body>
19 </html>
```

7.6.7. Objeto RegExp

O objeto **RegExp** (Regular Expression) representa uma expressão regular, isto é, a criação de um texto por meio do uso de um caractere curinga. Com base no caractere curinga, é possível buscar e trocar conteúdos.

Existem duas sintaxes possíveis para o objeto **RegExp**:

```
var patt=new RegExp(pattern,modifiers);
```

ou

```
var patt=/pattern/modifiers;
```

Em que:

- **pattern**: Define o padrão para uma expressão;

- **modifiers:** Define o padrão para a busca, ou seja, se ela será global, case-sensitive ou realizada em múltiplas linhas.

Para compreender a sintaxe de construção de uma expressão regular, é necessário conhecer os modificadores, os agrupadores, os metacaracteres, os quantificadores, as propriedades e os métodos do objeto **RegExp**, os quais serão descritos adiante.

7.6.7.1.Modificadores RegExp

Os modificadores estabelecem o tipo de busca a ser realizada. Por meio deles, é possível definir se a busca será case-sensitive ou case-insensitive, global e se será realizada em múltiplas linhas.

Os modificadores do objeto **RegExp** são os seguintes:

- **i:** Define que a busca será case-insensitive, isto é, a diferença entre letras maiúsculas e minúsculas não será considerada;
- **g:** Define que a busca será global, ou seja, em vez de parar após encontrar o primeiro resultado, a busca retornará todos os resultados possíveis;
- **m:** Define que a busca será realizada em múltiplas linhas.

7.6.7.2.Agrupadores

Os agrupadores, representados pelo par de colchetes, são utilizados pra buscar por um intervalo de caracteres. A tabela adiante descreve a utilização dos agrupadores com determinadas expressões:

Expressão	Descrição
[abc]	Busca os caracteres especificados entre os agrupadores.
[^abc]	Busca quaisquer caracteres, exceto aqueles especificados entre os agrupadores.
[0-9]	Busca todos os dígitos que estejam no intervalo entre 0 e 9.
[A-Z]	Busca quaisquer caracteres que estejam no intervalo entre as letras maiúsculas A e Z.

Expressão	Descrição
[a-z]	Busca quaisquer caracteres que estejam no intervalo entre as letras minúsculas a e z.
[A-z]	Busca quaisquer caracteres que estejam no intervalo entre a letra maiúscula A e a letra minúscula z.
[adgk]	Busca por qualquer um dos caracteres definidos dentro dos agrupadores.
[^adgk]	Busca por qualquer caractere, exceto aqueles especificados entre os agrupadores.
(red blue green)	Busca por qualquer uma das alternativas especificadas dentro dos agrupadores.

Veja um exemplo da utilização de alguns agrupadores:

```

1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7
8  var texto = "impacta treinamentos e certificações";
9
10 texto = texto.replace(/\^([a-z])|\s+([a-z])/g, function (letra) {
11     return letra.toUpperCase();
12 });
13
14 console.log(texto);
15
16 </script>
17 </head>
18
19 <body>
20
21 </body>
22 </html>
```

7.6.7.3. Metacaracteres

Metacaracteres são caracteres especiais, com funções específicas na realização de buscas. Cada um deles possui um significado próprio, alterando o padrão da busca a ser realizada.

A tabela adiante descreve a utilização dos metacaracteres:

Metacaractere	Descrição
.	Busca por qualquer caractere, excluindo apenas os caracteres de nova linha e terminador de linha.
\w	Busca por um caractere que seja uma letra.
\W	Busca por um caractere que não seja uma letra.
\d	Busca por um caractere que seja um dígito.
\D	Busca por um caractere que não seja um dígito.
\s	Busca por um caractere que seja um espaço em branco.
\S	Busca por um caractere que não seja um espaço em branco.
\b	Busca por uma correspondência com o começo ou o final de uma palavra.
\B	Busca por uma correspondência que não esteja no começo ou no final de uma palavra.
\0	Busca por um caractere NUL.
\n	Busca por um caractere de nova linha.
\f	Busca por um caractere de preenchimento de formulário (form feed).
\r	Busca por um caractere de quebra de linha.
\t	Busca por um caractere de tabulação.
\v	Busca por um caractere de tabulação vertical.
\xxx	Busca por um caractere definido por um número octal xxx (de base 8).
\xdd	Busca por um caractere definido por um número hexadecimal dd .
\xxxxx	Busca por um caractere Unicode definido por um número hexadecimal xxxx .

7.6.7.4. Quantificadores

Assim como os metacaracteres, os quantificadores são caracteres especiais que exercem uma função própria, mas inserem uma quantificação ao padrão de busca.

Os quantificadores são descritos na tabela a seguir:

Quantificador	Descrição	Exemplo de sintaxe
n+	Busca por qualquer string que contenha pelo menos um n.	/sit+/g
n*	Busca por qualquer string que contenha nenhuma ou mais ocorrências de n.	/sit*/g
n?	Busca por qualquer string que contenha nenhuma ou uma só ocorrência de n.	/sit?/g
n{X}	Busca por qualquer string que contenha uma sequência da quantidade X de n.	/in{1}/g
n{X,Y}	Busca por qualquer string que contenha uma sequência da quantidade X a Y do caractere n.	/in{0,4}/g
n{X,}	Busca por qualquer string que contenha uma sequência com, no mínimo, a quantidade X de n.	/a{1,}/g
n\$	Busca por qualquer string que contenha n em seu final.	/.\$/g
^n	Busca por qualquer string que contenha n em seu início.	/^L/g
?=n	Busca por qualquer string que seja seguida rpor uma string n específica.	/r(?= a)/g
?!n	Busca por qualquer string que não seja seguida por uma string n específica.	/r(?! a)/ig

JavaScript

O exemplo adiante é feito com base no texto:

“Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magnr aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ur aliquip ex ea commodr consequat. Duis aute irure dolor in reprehenderit inin voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt inin culpa qui officia deserunt mollit anim id est laborum.”

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magnr aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ur aliquip ex ea commodr consequat. Duis aute irure dolor in reprehenderit inin voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt inin culpa qui officia deserunt mollit anim id est laborum.";
8
9 var resultado_1 = texto.replace(/sit+/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
10 document.write(resultado_1 + '<hr>');
11
12 var resultado_2 = texto.replace(/sit*/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
13 document.write(resultado_2 + '<hr>');
14
15 var resultado_3 = texto.replace(/sit?/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
16 document.write(resultado_3 + '<hr>');
17
18 var resultado_4 = texto.replace(/in{1}/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
19 document.write(resultado_4 + '<hr>');
20
21 var resultado_5 = texto.replace(/in{0,4}/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
22 document.write(resultado_5 + '<hr>');
23
24 var resultado_6 = texto.replace(/a{1,}/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
25 document.write(resultado_6 + '<hr>');
26
27 var resultado_7 = texto.replace(/./g, function($){ return '<span style="color:red">' +$+ '</span>'; });
28 document.write(resultado_7 + '<hr>');
29
30 var resultado_8 = texto.replace(/^L/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
31 document.write(resultado_8 + '<hr>');
32
33 var resultado_9 = texto.replace(/r(?! a)/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
34 document.write(resultado_9 + '<hr>');
35
36 var resultado_10 = texto.replace(/r(?= a)/g, function($){ return '<span style="color:red">' +$+ '</span>'; });
37 document.write(resultado_10 + '<hr>');
38 </script>
39 </head>
40 <body>
41 </body>
42 </html>
```

7.6.7.5. Propriedades do objeto RegExp

As propriedades do objeto **RegExp** são as seguintes:

- **global**: Especifica o modo de busca global, por meio do modificador **g**. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "Lorem ipsum dolor sit amet, consectetur adipisicing elit,
     sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
     enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
     ut aliquip ex ea commodo consequat. Duis aute irure dolor in
     reprehenderit inin voluptate velit esse cillum dolore eu fugiat nulla
     pariatur. Excepteur sint occaecat cupidatat non proident, sunt inin
     culpa qui officia deserunt mollit anim id est laborum.";
8
9 var resultado = texto.replace(/a/g, function($){ return '<span
style="color:red; font-size:200%">'+$+'</span>'; });
10 document.write(resultado);
11
12 </script>
13 </head>
14 <body>
15 </body>
16 </html>
```

Ana
778.600

JavaScript

- **ignoreCase**: Especifica o modo de busca case-sensitive, por meio do modificador **i**. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "Lorem ipsum dolor sit amet, consectetur adipisicing elit,
     sed do eiusmod tempor incididunt ut labore et dolore magnr aliqua. Ut
     enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
     ur aliquip ex ea commodr consequat. Duis aute irure dolor in
     reprehenderit inin voluptate velit esse cillum dolore eu fugiat nulla
     pariatur. Excepteur sint occaecat cupidatat non proident, sunt inin
     culpa qui officia deserunt mollit anim id est laborum.";
8
9 var resultado = texto.replace(/d/ig, function($){ return '<span'
style="color:red; font-size:200%">'+'$+'</span>'; });
10 document.write(resultado);
11
12 </script>
13 </head>
14 <body>
15 </body>
16 </html>
```

- **lastIndex**: Define o índice em qual a próxima busca deve ser iniciada. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "Lorem ipsum dolor sit amet, consectetur adipisicing elit,
     sed do eiusmod tempor incididunt ut labore et dolore magnr aliqua. Ut
     enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
     ur aliquip ex ea commodr consequat. Duis aute irure dolor in
     reprehenderit inin voluptate velit esse cillum dolore eu fugiat nulla
     pariatur. Excepteur sint occaecat cupidatat non proident, sunt inin
     culpa qui officia deserunt mollit anim id est laborum.";
8
9 var patt = /m/g;
10
11 while (patt.test(texto)==true){
12     console.log("'m' encontrado. O index está em: "+patt.lastIndex);
13 }
14
15 </script>
16 </head>
17 <body>
18 </body>
19 </html>
```

- **multiline**: Especifica o modo de busca em múltiplas linhas, por meio do modificador **m**. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "Lorem ipsum dolor sit amet, consectetur adipisicing elit,
8 sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
9 enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
10 ur aliquip ex ea commodo consequat. Duis aute irure dolor in
11 reprehenderit inin voluptate velit esse cillum dolore eu fugiat nulla
12 pariatur. Excepteur sint occaecat cupidatat non proident, sunt inin
13 culpa qui officia deserunt mollit anim id est laborum.";
14
15 var patt = /o/gi;
16
17 if(patt.multiline){
18     console.log('modificador m está definido!');
19 }else{
20     console.log('modificador m não está definido!');
21 }
```

- **source**: Define o texto do padrão do objeto **RegExp**. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "Lorem ipsum dolor sit amet, consectetur adipisicing elit,
8 sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
9 enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
10 ur aliquip ex ea commodo consequat. Duis aute irure dolor in
11 reprehenderit inin voluptate velit esse cillum dolore eu fugiat nulla
12 pariatur. Excepteur sint occaecat cupidatat non proident, sunt inin
13 culpa qui officia deserunt mollit anim id est laborum.";
14
15 var patt = /^o/gi;
16
17 console.log('A expressão regular usada é: "'+patt.source+'"');
```

7.6.7.6. Métodos do objeto RegExp

Os métodos do objeto **RegExp** são os seguintes:

- **compile()**: Compila uma expressão regular. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "JavaScript foi originalmente desenvolvido por Brendan
Eich da Netscape sob o nome de Mocha, posteriormente teve seu nome
mudado para LiveScript e por fim JavaScript.";
8
9 var patt=/JavaScript/g;
10 var texto2 = texto.replace(patt,"<strong>JavaScript</strong>");
11 document.write(texto2+"<br />");
12 patt = /LiveScript/g;
13 patt.compile(patt);
14 texto2 = texto.replace(patt,'<span style="color:blue">LiveScript
</span>');
15 document.write(texto2);
16
17 </script>
18 </head>
19 <body>
20 </body>
21 </html>
```

- **exec()**: Executa um teste, procurando por uma correspondência em uma string e retorna o primeiro resultado. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "JavaScript foi originalmente desenvolvido por Brendan
Eich da Netscape sob o nome de Mocha, posteriormente teve seu nome
mudado para LiveScript e por fim JavaScript.";
8
9 var patt = /JavaScript/g;
10
11 console.log(patt.exec(texto));
12
13 </script>
14 </head>
15 <body>
16 </body>
17 </html>
```

- **test()**: Executa um teste, procurando por uma correspondência em uma string e retorna **true** (verdadeiro) ou **false** (falso). Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var texto = "JavaScript foi originalmente desenvolvido por Brendan
Eich da Netscape sob o nome de Mocha, posteriormente teve seu nome
mudado para LiveScript e por fim JavaScript.";
8
9 var patt = /JavaScript/g;
10
11 console.log(patt.test(texto));
12
13 </script>
14 </head>
15 <body>
16 </body>
17 </html>
```

Os métodos **replace()**, **search()** e **match()** do objeto **String** podem ser aplicados na manipulação de objetos **RegExp**. Eles são chamados de métodos auxiliares a expressões regulares.

7.6.7.7. O objeto RegExp na prática

Para preenchimento de campos de formulários, o mais comum é que seja criada uma máscara de campo. Para exemplificar o uso das expressões regulares, vamos criar algumas com a finalidade de validar as seguintes entradas de dados de um usuário em campos de um formulário:

- **Validação de CPF**

A validação de um campo de formulário para inserção do número de CPF requer as seguintes etapas:

- Primeiro, verifique se a forma do número está correta;
- Depois, verifique se a forma do número está de acordo com o algoritmo gerador do número de CPF.

 Para informações sobre o algoritmo gerador do número de CPF, consulte http://www.geradorcpf.com/algoritmo_do_cpf.htm.

O número de CPF é constituído por 11 algoritmos, no seguinte formato: **xxx.xxx.xxx-xx**. Observe que os nove primeiros algoritmos são divididos em grupos de três algoritmos, separados por pontos e, em seguida, os dois algoritmos restantes são separados por um traço. Essa é a parte fixa da ER.

A máscara de campo para esse número tem a finalidade de preencher, automaticamente, os pontos e o traço que separam os algoritmos. Com isso, o usuário digita o número ininterruptamente e os pontos e o traço são inseridos automaticamente, durante a digitação.

Para exemplificar a utilização das expressões regulares, vamos validar a entrada dos 11 algoritmos, mas não a do algoritmo gerador do número, criando uma validação simplificada:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7 <body>
8
9 <input type="text" placeholder="CPF" id="inputCPF">
10 <button id="validar">Validar</button>
11
12 <script>
13
14 document.getElementById("validar").addEventListener('click', function() {
15
16     var regex = /^{\d{3}}\.{\d{3}}\.{\d{3}}\-\d{2}$/;
17
18     if(regex.test(document.getElementById("inputCPF").value)){
19         alert('É um formato de CPF!');
20     }else{
21         alert('Não é um formato de CPF!');
22     }
23
24 }, false);
25
26 </script>
27 </body>
28 </html>
```

- **Validação de CNPJ**

A validação do CNPJ é semelhante à validação do CPF, com exceção do número. O registro do CNPJ é constituído por 14 algoritmos, no seguinte formato: **xx.xxx.xxx/xxxx-xx**. Observe que há dois pontos finais, uma barra e um traço separando os algoritmos, de forma a criar sua estrutura.

A máscara de campo para esse número tem a finalidade de preencher, automaticamente, os pontos, a barra e o traço que separamos os algoritmos. Essa é a parte fixa da ER. Com isso, o usuário digita o número ininterruptamente e eles são inseridos automaticamente, durante a digitação.

Para exemplificar a utilização das expressões regulares, vamos validar a entrada dos 14 algoritmos, criando uma validação simplificada:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7 <body>
8
9 <input type="text" placeholder="CNPJ" id="inputCNPJ">
10 <button id="validar">Validar</button>
11
12 <script>
13
14 document.getElementById("validar").addEventListener('click', function(){
15
16     var regex = /^\d{2}.\d{3}.\d{3}\/\d{4}-\d{2}$/;
17
18     if(regex.test(document.getElementById("inputCNPJ").value)){
19         alert('É um formato de CNPJ!');
20     }else{
21         alert('Não é um formato de CNPJ!');
22     }
23 }, false);
24
25 </script>
26 </body>
27 </html>
```

- **Validação de e-mail**

Considere o endereço de e-mail **usuario@impacta.com**. A parte fixa da ER são os caracteres arroba (@) e ponto (.), presentes em todos os endereços de e-mail.

Lembre-se de considerar as condições para a formação da string de um endereço de e-mail, definidas pela IETF (Internet Engineering Task Force), como a que impede sua criação se este for iniciado pelo caractere arroba (@).

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7 <body>
8
9 <input type="text" placeholder="E-mail" id="inputEmail">
10 <button id="validar">Validar</button>
11
12 <script>
13
14 document.getElementById("validar").addEventListener('click', function(){
15
16     var regex = /^[a-zA-Z0-9_.-]+@[a-zA-Z0-9_.-]+\.( [a-zA-Z])+([a-zA-Z])+/;
17
18     if(regex.test(document.getElementById("inputEmail").value)){
19         alert('É um formato de E-MAIL!');
20     }else{
21         alert('Não é um formato de E-MAIL!');
22     }
23
24 }, false);
25
26 </script>
27 </body>
28 </html>
```

- **Validação de URL**

Para a validação de URL, utiliza-se o mesmo raciocínio das outras validações. Considere as partes que devem ser fixas em um endereço URL (o prefixo **http**, **ftp** ou **https**), a colocação dos pontos após o www, e no final, antes de definir o último fragmento (.com, .net, .gov, etc.). Um código padrão para definir a validação de URL é o seguinte:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7 <body>
8
9 <input type="text" placeholder="URL" id="inputURL">
10 <button id="validar">Validar</button>
11
12 <script>
13
14 document.getElementById("validar").addEventListener('click', function() {
15
16     var regex =
17         /(^http|https) :\/\/[\w-]+([\w-]+)+([\w-]+[\w@?#=;&:&/~+#!]*[\w@?#=;&:&/~+#!])?/;
18
19     if(regex.test(document.getElementById("inputURL").value)){
20         alert('É um formato de URL!');
21     }else{
22         alert('Não é um formato de URL!');
23     }
24 }, false);
25
26 </script>
27 </body>
28 </html>
```

 Quando você aceita uma expressão regular como string, é necessário utilizar o construtor RegExp e torná-lo mais legível, como no exemplo anterior.

7.7. BOM (Browser Object Model)

A função do **Browser Object Model** (BOM) na linguagem JavaScript é descrever os objetos que integram o browser ou navegador, bem como seus métodos e propriedades. Ele é implementado normalmente pela maioria dos navegadores e muitas de suas funcionalidades são equivalentes a funcionalidades do DOM.

Mas como não está previsto nas ECMAScript ou em nenhuma outra especificação, é recomendável, sempre que possível, usar o DOM em vez do BOM.

7.7.1. Objeto Window

Este objeto é uma representação de uma janela que esteja aberta em um navegador. Sempre que um documento possui frames, representados pelas tags `<frame>` e `<iframe>`, o navegador cria um objeto **Window** para o documento HTML e um objeto **Window** adicional para cada um dos frames.

7.7.1.1. Propriedades do objeto Window

As propriedades do objeto **Window** são descritas na tabela adiante:

Propriedade	Descrição
closed	O retorno é um valor booleano, que indica se a janela foi fechada ou se continua aberta.
defaultStatus	Permite configurar ou retorna o texto padrão da barra de status de uma janela.
document	O retorno é o objeto Document da janela.
frames	O retorno é um array contendo todos os frames, inclusive iframes, da janela.
history	O retorno é o objeto History da janela.
innerHeight	Permite configurar ou retorna a altura interna da área de conteúdo da janela.
innerWidth	Permite configurar ou retorna a largura interna da área de conteúdo da janela.
length	O retorno é o número de frames, inclusive iframes, da janela.

Propriedade	Descrição
location	O retorno é o objeto Location da janela.
name	Permite configurar ou retorna o nome da janela.
navigator	O retorno é o objeto Navigator da janela.
opener	O retorno é uma referência à janela que criou a janela atual.
outerHeight	Permite configurar ou retorna a altura externa da janela, contando com as barras de ferramenta e scrolls.
outerWidth	Permite configurar e retorna a largura externa da janela, contando com as barras de ferramenta e scrolls.
pageXOffset	O retorno é o número de pixels movidos horizontalmente por meio do scroll, a partir do canto esquerdo superior da janela.
pageYOffset	O retorno é o número de pixels movidos verticalmente por meio do scroll, a partir do canto esquerdo superior da janela.
parent	O retorno é a janela pai da janela atual.
screen	O retorno é o objeto Screen da janela.
screenLeft	O retorno é a coordenada x da janela em relação à tela.
screenTop	O retorno é a coordenada y da janela em relação à tela.
screenX	O retorno é a coordenada x da janela em relação à tela.
screenY	O retorno é a coordenada y da janela em relação à tela.
self	O retorno é a janela atual.
status	Permite configurar o texto da barra de status da janela.
top	O retorno é a janela superior do navegador.

Veja um exemplo da utilização de algumas propriedades do objeto **Window**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log("closed", window.closed);
9 console.log("defaultStatus", window.defaultStatus);
10 console.log("document", window.document);
11 console.log("frames", window.frames);
12 console.log("history", window.history);
13 console.log("innerHeight", window.innerHeight);
14 console.log("innerWidth", window.innerWidth);
15 console.log("length", window.length);
16 console.log("location", window.location);
17 console.log("name", window.name);
18 console.log("navigator", window.navigator);
19 console.log("opener", window.opener);
20 console.log("outerHeight", window.outerHeight);
21 console.log("outerWidth", window.outerWidth);
22 console.log("pageXOffset", window.pageXOffset);
23 console.log("pageYOffset", window.pageYOffset);
24 console.log("parent", window.parent);
25 console.log("screen", window.screen);
26 console.log("screenLeft", window.screenLeft);
27 console.log("screenTop", window.screenTop);
28 console.log("screenX", window.screenX);
29 console.log("screenY", window.screenY);
30 console.log("self", window.self);
31 console.log("status", window.status);
32 console.log("top", window.top);
33
34 </script>
35 </head>
36 <body>
37
38 </body>
39 </html>
```

7.7.1.2. Métodos do objeto Window

Os métodos do objeto **Window** são descritos na tabela a seguir:

Método	Descrição
alert()	Exibe uma caixa de aviso contendo uma mensagem em um botão OK .
blur()	Retira o foco da janela atual.
clearInterval()	Exclui o horário que foi configurado por meio do método setInterval() .
clearTimeout()	Exclui o horário que foi configurado por meio do método setTimeout() .
close()	Fechá a janela.
confirm()	Exibe uma caixa de diálogo contendo uma mensagem e os botões OK e Cancel .
createPopup()	Permite criar uma janela pop-up.
focus()	Coloca o foco na janela atual.
moveBy()	Movimenta a janela em relação à posição atual.
moveTo()	Movimenta a janela para a posição especificada.
open()	Abre uma nova janela do navegador.
print()	Permite imprimir o conteúdo da janela atua.
prompt()	Exibe uma caixa de diálogo que solicita uma entrada ao visitante.
resizeBy()	Redimensiona a janela, conforme o número de pixels definido.
resizeTo()	Redimensiona a janela, conforme altura e largura especificadas.
scrollBy()	Movimenta o conteúdo, conforme o número de pixels especificado.
scrollTo()	Movimenta o conteúdo, conforme as coordenadas especificadas.
setInterval()	Permite chamar a função ou avaliar uma expressão em intervalos especificados em milissegundos.
setTimeout()	Permite chamar uma função ou avaliar uma expressão após um número de milissegundos especificado.

Veja um exemplo da utilização de alguns métodos do objeto **Window**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8     function fechar() {
9         window.close();
10    }
11
12    function alertaEm(tempo) {
13        window.setTimeout(function() {
14            alert("Alerta!");
15        }, tempo * 1000);
16    }
17
18    function imprimir() {
19        window.print();
20    }
21
22 </script>
23 </head>
24 <body>
25
26 <button onClick="fechar()">Fechar</button>
27 <button onClick="alertaEm(2)">Alerta em 2 segundos...</button>
28 <button onClick="alertaEm(5)">Alerta em 5 segundos...</button>
29 <button onClick="imprimir()">Imprimir</button>
30
31 </body>
32 </html>
```

Veja em destaque os dois métodos mais importantes do objeto **Window**:

- **setInterval()**

Este método permite chamar uma função ou avaliar uma expressão em intervalos especificados. O retorno é um valor de ID.

A função continuará a ser chamada até que a janela que está sendo utilizada seja fechada, ou até que o método **clearInterval()** seja chamado. Este método fará uso do retorno do método **setInterval()** como parâmetro, isto é, do valor de ID retornado.

JavaScript

Para utilizar o método **setInterval()**, a sintaxe é a seguinte:

```
setInterval(code,millisec,lang)
```

Em que:

- **code**: É uma referência à função ou um código a ser executado;
 - **millisec**: Define, em milissegundos, os intervalos de frequência com que o código deve ser executado;
 - **lang**: Define a linguagem, se JScript, VBScript ou JavaScript. Este parâmetro é opcional.
- **setTimeout()**

Este método permite chamar uma função ou avaliar uma expressão após o tempo especificado. Para utilizá-lo, a sintaxe é a seguinte:

```
setTimeout(code,millisec,lang)
```

Em que:

- **code**: É uma referência à função ou ao código a ser executado;
- **millisec**: Define, em milissegundos, o tempo antes de o código deve ser executado;
- **lang**: Define a linguagem, se JScript, VBScript ou JavaScript. Este parâmetro é opcional.

7.7.2. Objeto Navigator

Este objeto contém as informações relacionadas ao navegador em uso.



A maioria dos navegadores oferece suporte para o objeto **Navigator**, embora não haja padrões públicos aplicáveis a ele.

7.7.2.1. Propriedades do objeto Navigator

As propriedades do objeto **Navigator** são descritas na tabela a seguir:

Propriedade	Descrição
appCodeName	O retorno é o codinome do navegador.
appName	O retorno é o nome do navegador.
appVersion	O retorno é a informação sobre a versão do navegador.
cookieEnabled	Define se os cookies serão habilitados no navegador.
platform	O retorno é a plataforma em que o navegador foi compilado.
userAgent	O retorno é o cabeçalho user-agent enviado do navegador ao servidor.

Veja um exemplo da utilização de algumas propriedades do objeto **Navigator**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log('appCodeName', navigator.appCodeName);
9 console.log('appName', navigator.appName);
10 console.log('appVersion', navigator.appVersion);
11 console.log('cookieEnabled', navigator.cookieEnabled);
12 console.log('platform', navigator.platform);
13 console.log('userAgent', navigator.userAgent);
14
15 </script>
16 </head>
17 <body>
18
19 </body>
20 </html>
```

7.7.2.2. Métodos do objeto Navigator

Os métodos do objeto **Navigator** são os seguintes:

- **javaEnabled()**: Define se o navegador terá Java habilitado ou não. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log('Suporta JAVA? ', navigator.javaEnabled());
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **taintEnabled()**: Define se o navegador terá marcação de dados (data tainting) habilitada ou não. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log('Marcação de Dados Tainting', navigator.taintEnabled());
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

7.7.3. Objeto Screen

Este objeto contém as informações relacionadas à tela do usuário.

7.7.3.1. Propriedades do objeto Screen

As propriedades do objeto **Screen** são as seguintes:

- **availHeight**: O retorno é a altura da tela, com exceção da barra de tarefas do Windows. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(screen.availHeight);
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **availWidth:** O retorno é a largura da tela, com exceção da barra de tarefas do Windows. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(screen.availWidth);
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **colorDepth:** O retorno é a profundidade de cor da paleta de cores para a exibição de imagens. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(screen.colorDepth);
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **height:** O retorno é a altura total da tela. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(screen.height);
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **pixelDepth**: O retorno é a resolução de cor da tela, em bits por pixel (bpp). Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(screen.pixelDepth);
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **width**: O retorno é a largura total da tela. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(screen.width);
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

7.7.4. Objeto History

Este objeto faz parte do objeto **Window** e pode ser acessado por meio de sua propriedade **history**. O objeto **History** contém o histórico das URLs que o usuário visitou, dentro de uma janela do navegador.

7.7.4.1. Propriedade do objeto History

O objeto **History** possui somente uma propriedade: **length**. O retorno dessa propriedade é o número de URLs na lista do histórico, isto é, os URLs acessados pelo usuário dentro da janela do navegador. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(history.length);
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

7.7.4.2. Métodos do objeto History

Os métodos do objeto **History** são os seguintes:

- **back()**: Carrega o URL anterior na lista do histórico. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(history.back());
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **forward()**: Carrega o próximo URL da lista do histórico. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(history.forward());
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **go()**: Carrega um URL específico na lista do histórico. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(history.go(-1));
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

7.7.5. Objeto Location

Este objeto também é parte do objeto **Window** e pode ser acessado a partir de sua propriedade **location**. O objeto **Location** contém as informações relacionadas com o URL acessado atualmente pelo usuário.

7.7.5.1. Propriedades do objeto Location

As propriedades do objeto **Location** são descritas na tabela a seguir:

Propriedade	Descrição
hash	O retorno é a parte do URL que explicita o tipo de arquivo ou página para a qual o usuário será direcionado (âncora).
host	O retorno é o hostname e a porta do URL.
hostname	O retorno é o hostname do URL.
href	O retorno é o URL completo.
pathname	O retorno é o nome do caminho de um URL.
port	O retorno é o número da porta utilizada pelo servidor para o URL.
protocol	O retorno é o protocolo de um URL.
search	O retorno é a parte de query do URL.

Veja um exemplo da utilização de algumas propriedades do objeto **Location**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log('hash', location.hash);
9 console.log('host', location.host);
10 console.log('hostname', location.hostname);
11 console.log('href', location.href);
12 console.log('pathname', location.pathname);
13 console.log('port', location.port);
14 console.log('protocol', location.protocol);
15 console.log('search', location.search);
16
17 </script>
18 </head>
19 <body>
20
21 </body>
22 </html>
```

7.7.5.2. Métodos do objeto Location

Os métodos do objeto **Location** são os seguintes:

- **assign()**: Carrega um novo documento. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 window.location.assign("http://www.impacta.com.br");
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **reload()**: Recarrega o documento atualmente aberto. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 function atualizar(){
9
10    window.location.reload();
11 }
12
13 </script>
14 </head>
15 <body>
16
17 <button onClick="atualizar()">Atualizar (F5)</button>
18
19
20 </body>
21 </html>
```

- **replace()**: Substitui o documento atualmente aberto por um novo documento. Veja o exemplo:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 window.location.replace("http://www.impacta.com.br");
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

7.8. JSON

O formato JSON (JavaScript Object Notation), descrito pelo RFC 4627, foi originalmente especificado por Douglas Crockford. É um subconjunto da notação de objeto JavaScript, de formato leve, para intercâmbio de dados computacionais.

JSON tem como tipo de mídia padrão `application/json` e sua extensão é `.json`.

Ele é utilizado, principalmente, como uma alternativa ao XML em AJAX, porque é muito mais simples escrever um analisador JSON. Para utilizá-lo, não é necessário adotar a linguagem JavaScript, é possível utilizar outras.

Para analisar JSON utilizando JavaScript, a forma mais simples é utilizando a função `eval()`. Como há esse recurso em todos os navegadores modernos, a comunidade AJAX aceitou bem a utilização do formato JSON.

Veja a seguir um exemplo da utilização do JSON:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var jsonString = '{"trenamento":"JavaScript"}';
9
10 eval("var json = "+jsonString);
11
12 console.log(json);
13
14 </script>
15 </head>
16 <body>
17
18 </body>
19 </html>
```

7.8.1. Tipos básicos de JSON

Os tipos de dados básicos de JSON são os seguintes:

- **Number**

Este tipo de JSON é semelhante ao tipo **number** na linguagem C ou Java. A diferença é que não utiliza os formatos octal e hexadecimal.

 O formato de ponto flutuante com precisão dupla na linguagem JavaScript requer implementação na maioria das vezes.

- **String**

Semelhantemente ao tipo **String** das linguagens C e Java, este tipo é uma sequência de caracteres Unicode, que deve ser delimitada por aspas e seguida por uma barra invertida. Quando há um caractere, ele é representado por uma string com um caractere somente.

- **Boolean**

Este tipo representa os dois valores booleanos: **true** ou **false**.

- **Array**

Este tipo é uma sequência de valores ordenados, que não precisam ser do mesmo tipo. A sequência deve vir delimitada por colchetes e seus elementos separados por vírgulas.

- **Object**

Este tipo é uma coleção de chaves, em que não há ordem, desde que os pares nome/valor sejam separados por vírgulas (,), delimitados por chaves ({}) e possuam dois pontos (:) separando as chaves dos valores. Além disso, as chaves devem ser strings diferentes entre si.

- **Value**

Um número, os valores **true**, **false** e **null**, uma string entre aspas, um objeto ou mesmo um array são diferentes tipos de valores. Quando você utiliza estes valores, é importante lembrar que eles podem ser aninhados.

- **null (empty)**

Você pode ficar à vontade para utilizar espaços em branco (**null**) antes ou depois dos caracteres estruturais, já que eles não afetarão o código.

7.8.2. Função **JSON.parse**

Esta função tem a finalidade de realizar a conversão de uma string JSON em um objeto. Para isso, utilize o seguinte código:

```
JSON.parse(text [, reviver])
```

Em que:

- **text**: Especifica uma string JSON válida;

- **reviver:** Esta função, que transforma o resultado, pode ser chamada para cada membro do objeto. Quando um membro possui objetos aninhados, a função é aplicada primeiro a eles. Para cada membro de um objeto, pode acontecer o seguinte:
 - O retorno da função é um valor válido. Com esse resultado, o valor transformado substituirá o valor do membro do objeto;
 - O retorno da função é o mesmo valor recebido. Com isso, o membro do objeto não será alterado;
 - O retorno da função é **null** ou **undefined**. Com isso, o membro do objeto será excluído.

A aplicação da função **reviver** é opcional.

Com a utilização da função **JSON.parse** pode acontecer um erro do analisador, caso em que o texto de entrada não está em conformidade com a sintaxe JSON. Para resolver esse problema, temos duas alternativas: alterar o argumento do texto, a fim de que fique de acordo com a sintaxe JSON ou garantir que ele seja serializado por uma implementação que o torne compatível, como a função **JSON.stringify**. Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 var jsonString = '{"trenamento":"JavaScript"}';
9
10 console.log(JSON.parse(jsonString));
11
12 </script>
13 </head>
14 <body>
15
16 </body>
17 </html>
```

7.8.3. Função JSON.stringify

Esta função tem a finalidade de realizar a conversão de um valor JavaScript em uma string JSON. Para isso, utilize o seguinte código:

```
JSON.stringify(value [, replacer] [, space])
```

Em que:

- **value**: Especifica o valor JavaScript a ser convertido. Geralmente, este valor é um objeto ou um array;
- **replacer**: Pode ser tanto uma função quanto um array, que transformará o resultado. Funciona da seguinte forma:
 - Quando **replacer** é uma função, ela é chamada por **JSON.stringify**, passando a chave e o valor para cada membro. O valor que é retornado substituirá o valor original. Somente quando o valor retornado for **undefined**, o membro será excluído;

 A chave para o objeto raiz é uma string vazia ("").

- Quando **replacer** é um array, somente os membros com valores chave no array serão convertidos. A conversão é feita na mesma ordem em que as chaves estão dispostas no array. Mas se o argumento **value** for também um array, a aplicação de **replacer** será ignorada.

 A aplicação da função ou do array **replacer** é opcional.

- **space**: Responsável por adicionar caracteres de endentação, espaços em branco ou quebras de linha ao texto JSON com o valor retornado, a fim de que ele se torne mais legível. Sua aplicação é opcional e quando não é utilizado, o texto não conterá espaços em branco. As hipóteses são as seguintes:
 - O argumento **space** é um número. Com isso, o texto será endentado com o número definido de espaços em branco para cada nível. Quando o valor é maior que 10, o texto será endentado em 10 espaços;
 - O argumento **space** é uma string que não está vazia. O texto, então, será endentado com a quantidade de caracteres da string em cada nível. Quando a string possui mais que 10 caracteres, somente os 10 primeiros serão utilizados.



A aplicação de **space** é opcional.

O argumento **value** pode conter um método **toJSON**, caso em que a função vai utilizar o valor retornado deste método. Somente se o valor retornado for **undefined** o membro não será convertido. Com isso, o objeto consegue determinar sua própria representação JSON.

JavaScript

Os valores que não possuírem representação JSON não serão convertidos e, se estiverem em objetos serão pulados ou em arrays serão substituídos por **null**.

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7
8 console.log(JSON.stringify(screen));
9
10 </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```



É importante lembrar que:

- Os valores em uma string devem vir delimitados por aspas;
- Todos os caracteres Unicode devem ser delimitados por aspas, com exceção dos caracteres que devem ser antecedidos por uma barra invertida;
- Os seguintes caracteres devem ser antecedidos por uma barra invertida:
 - Aspas duplas (“);
 - Barra invertida (\);
 - Backspace (b);
 - Formfeed (f);
 - Nova linha (n);
 - Quebra de linha (r);
 - Tabulação horizontal (t);
 - Quatro dígitos hexadecimais (uhhh).

7.8.3.1. Ordem de execução

A ordem de execução depende se há ou não um método `toJSON` para o argumento `value`. Quando o argumento possui o método, a função `JSON.stringify` chama primeiro o método, já que ele transformará os valores. Na outra hipótese, isto é, quando o argumento não possui o método, os valores originais são utilizados.

O próximo passo tem a ver com o argumento `replacer`. Quando ele foi fornecido na sintaxe, o valor original ou o valor de retorno do método JSON é substituído pelo valor do argumento `replacer`.

Por último, se o argumento `space` foi utilizado, os espaços em branco serão adicionados em conformidade com a sua definição. Com isso, o texto JSON final é gerado.



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Quase todos os elementos da linguagem JavaScript são objetos e a manipulação deles está presente em todos os códigos, dos mais simples aos mais complexos. Eles podem ser classificados em objetos customizados, nativos ou do ambiente de hospedagem;
- O objeto mais simples da linguagem Java Script é o tipo de dado **Object**. Trata-se de uma coleção de propriedades nomeadas, as quais podem ser criadas e adicionadas a qualquer tempo. Ou seja, não é necessário que elas sejam predefinidas em um construtor ou em uma declaração de objeto;
- A criação de um objeto é, na verdade, chamar o construtor de uma classe para a criação de uma instância, que nada mais é que um objeto com o tipo de dado que foi definido pela classe. Para a criação de um objeto a partir de uma classe, utilize a instrução **new**;
- Para acessar tanto as propriedades quanto os métodos de objetos, utilizamos o operador ponto (.). Este procedimento é semelhante ao realizado em outras linguagens de programação;
- Objetos nativos são aqueles que já estão embutidos na linguagem JavaScript. São eles: **array, boolean, date, math, number, string, RegExp**;
- A função do **Browser Object Model** (BOM) na linguagem JavaScript é descrever os objetos que integram o browser ou navegador, bem como seus métodos e propriedades. Para isso, contamos com os objetos **Window, Navigator, Screen, History e Location**;
- O formato JSON visa o intercâmbio de dados computacionais. O tipo de mídia padrão é **application/json** e sua extensão é **.json**. Para analisar JSON utilizando JavaScript, a forma mais simples é utilizando a função **eval()**;
- Utilizamos a função **JSON.parse** para converter uma **string JSON** em um objeto e a função **JSON.stringify** para converter um valor JavaScript em uma **string JSON**.

7

Objetos

Teste seus conhecimentos

Ana Gomes
778.6008-0000
cesar
778.6008-0000



IMPACTA
EDITORA

1. Qual dos conceitos básicos não faz parte do conceito da estrutura de programação orientada a objetos?

- a) Objeto Simples
- b) Classes
- c) Métodos
- d) Polimorfismo
- e) Variáveis

2. Qual operador é utilizado para acessar as propriedades e os métodos de um objeto na JavaScript?

- a) , (virgula)
- b) . (ponto)
- c) &
- d) And
- e) Nenhuma das alternativas anteriores está correta.

3. Qual das sintaxes esta correta na declaração de um array?

- a) var arr = new Array(element0, element1, ..., elementN);
- b) var arr = new array(element0, element1, ..., elementN);
- c) var arr = array(element0, element1, ..., elementN);
- d) var arr = Array(element0, element1, ..., elementN);
- e) Nenhuma das alternativas anteriores está correta.

4. O método indexOf do objeto Array é utilizado para:

- a) Encontrar um elemento no array, retornando a posição dele.
- b) Encontrar um elemento no array, retornando a posição e o tipo do array.
- c) Encontrar um elemento no array, retornando se existe.
- d) Encontrar um elemento no array, retornando a posição e invertendo a ordem dos elementos dentro do array.
- e) Nenhuma das alternativas anteriores está correta.

5. O método `getDay()` do objeto `Date` retorna qual valor?

- a) Dia do mês.
- b) Dia da semana.
- c) Dia do ano.
- d) Dia e hora do mês.
- e) Dia, mês e ano.



7

Objetos Mãos à obra!

Ana Góes
778.6000



IMPACTA
EDITORA

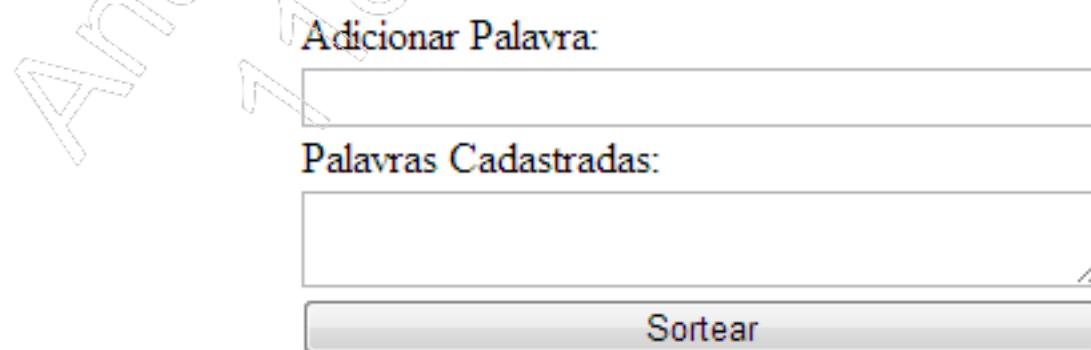
Laboratório 1

A - Criando uma lista de palavras com array e utilizando esta lista para gerar um sorteio

1. Crie um novo arquivo de texto e renomeie com a extensão .html;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD** e **BODY**;
4. Inclua dentro da tag **BODY** o seguinte código HTML:

```
1 <html>
2 <head>
3   <style>
4     input, textarea{width:300px;}
5   </style>
6 </head>
7 <body>
8   <label>Adicionar Palavra:</label>
9   <div><input type="text" id="palavras"></div>
10  <label>Palavras Cadastradas:</label>
11  <div><textarea id="listapalavras"></textarea></div>
12  <div><input type="button" value="Sortear" /></div>
13  <div id="sorteado"></div>
14 </body>
15 </html>
```

Isso vai resultar no seguinte:



5. Inclua dentro da tag **HEAD** a tag **<script></script>**;

6. Crie um array de palavras;
7. Crie uma função para capturar o valor que o usuário preencheu no campo **Palavras Cadastradas** e adicione no array de palavras;
 - Deve-se utilizar o evento **onKeyPress**.
8. Crie uma função no evento **click** do botão **Sortear** para realizar o sorteio;
 - O sorteio deve ser aleatório;
 - Deve-se levar em consideração o número de palavras adicionadas no array para realizar o sorteio.
9. O resultado final vai mostrar a cada clique no botão **Sortear** o valor sorteado de forma aleatória.

Laboratório 2

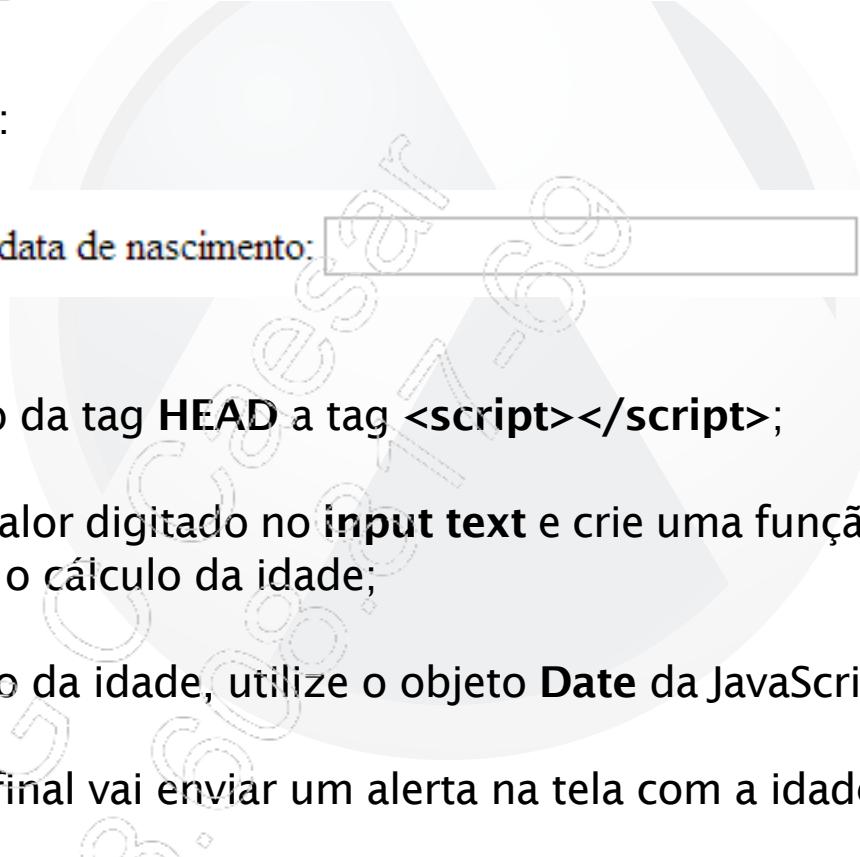
A - Calculando idade do usuário após ser inserido a data de nascimento

1. Crie um novo arquivo de texto e renomeie com a extensão **.html**;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD**, **BODY**;

4. Inclua dentro da tag **BODY** o seguinte código HTML:

```
1 <html>
2 <head>
3   <style>
4     input {width:200px;}
5   </style>
6 </head>
7 <body>
8   <label>Insira a data de nascimento:</label>
9   <input type="text" id="valor">
10  <button>Calcular</button>
11  <div id="idade"></div>
12 </body>
13 </html>
```

Que resulta em:



Insira a data de nascimento: Calcular

5. Inclua dentro da tag **HEAD** a tag **<script></script>**;

6. Recupere o valor digitado no **input text** e crie uma função no botão **Calcular** que irá realizar o cálculo da idade;

7. Para o cálculo da idade, utilize o objeto **Date** da JavaScript;

8. O resultado final vai enviar um alerta na tela com a idade calculada.

DOM

8

- ✓ DOM HTML;
- ✓ DOM Core;
- ✓ DOM Storage.

Ana Gómez
778.6008.7769



IMPACTA
EDITORA

8.1. Introdução

A maneira padrão de se acessar e manipular documentos HTML é definida pelo Document Object Model (Modelo de Objetos de Documento), também conhecido como DOM, que foi lançado em 1998 pela W3C (World Wide Web Consortium) com a especificação Nível 1, permitindo o acesso e a manipulação de cada elemento em uma página HTML.

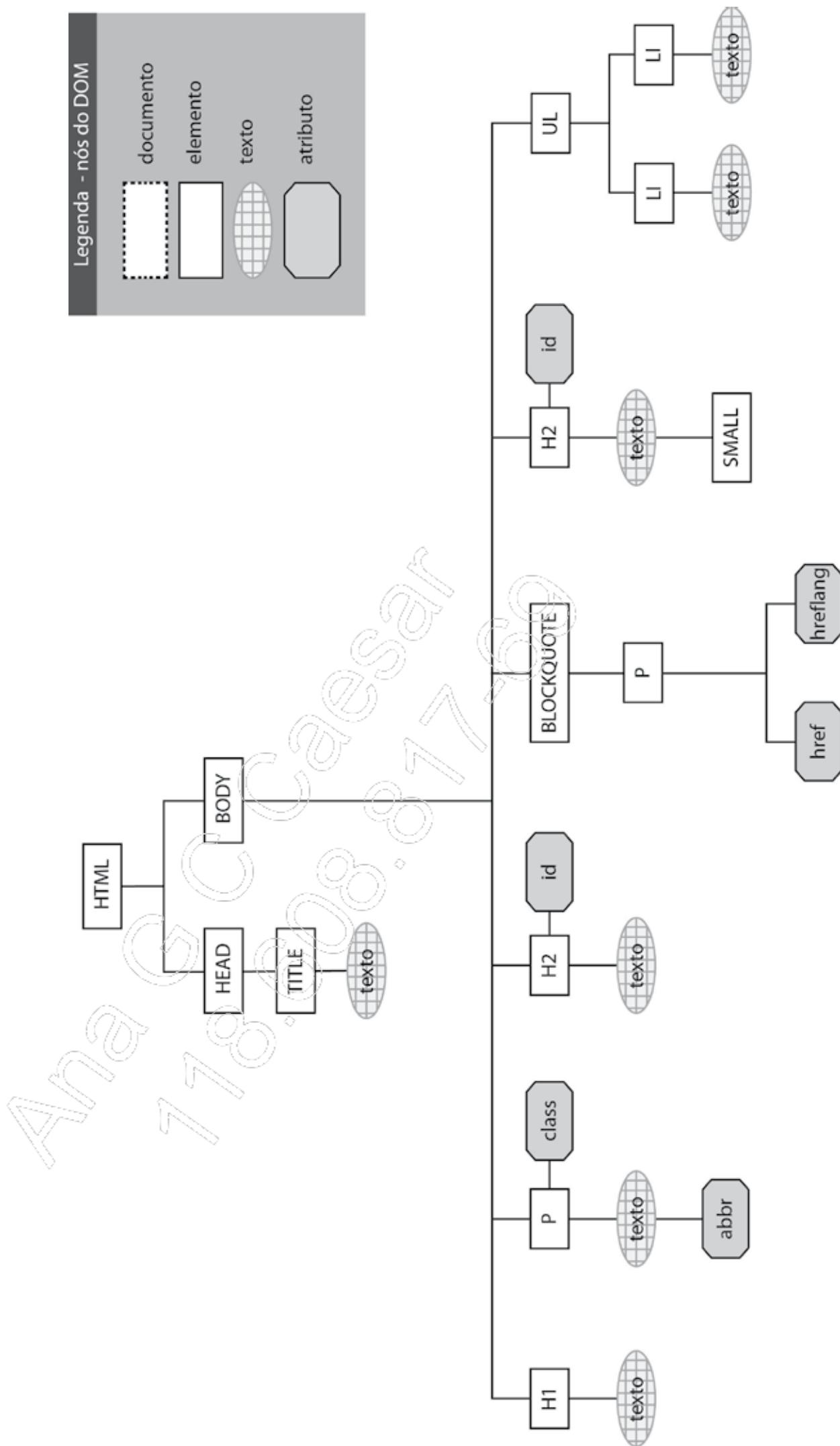
Antes de tudo, é bom lembrar que, em virtude da sua implementação por parte dos navegadores disponíveis no mercado, praticamente não ocorrem problemas de incompatibilidade com DOM atualmente.

O objeto DOM permite que a linguagem JavaScript acesse todos os elementos de um documento HTML, o que é fundamental para realizar qualquer alteração em páginas da Web. Estas alterações podem ser realizadas com a utilização dos métodos e das propriedades que permitem adicionar, mover, alterar ou remover elementos HTML da página.

JavaScript também utiliza os recursos do objeto DOM para realizar tanto a leitura quanto a alteração em documentos HTML, XHTML e XML.

8.2. DOM HTML

A estrutura de um documento HTML pode ser representada pelo DOM HTML. É possível retratar o DOM HTML por meio de um diagrama em formato de árvore, do mesmo modo que uma árvore genealógica de uma família. Nesse diagrama é possível compreender os graus de “parentesco” entre elementos, além das ascendências e descendências. O DOM é designado pelo termo Modelo de Objetos, como já foi mencionado, de acordo com a definição comum para esse termo vista quando você estuda Programação Orientada a Objetos.



Veja a seguir os principais objetos do DOM HTML.



Todos os objetos DOM HTML apresentam suporte para todas as propriedades e eventos padrão, além de suas próprias propriedades, métodos e eventos.

8.2.1. Objeto Document

O objeto **Document** dá acesso a todos os elementos HTML de uma página por meio de scripts. Sendo assim, todo documento HTML que é aberto em um navegador torna-se um objeto **Document**, e por meio de suas propriedades é possível visualizar as informações sobre um documento HTML e seus elementos.



Por meio da propriedade **window.document** é possível acessar o objeto **Document**, pois ele é parte do objeto **Window**.

As propriedades e métodos do objeto **Node** também podem ser usados pelo objeto **Document**.

Veja a seguir um exemplo da utilização do objeto **Document**:

```
<script>  
  console.log(window.document);  
  
  console.log(document);  
  
</script>
```

8.2.1.1. Propriedades do objeto Document

A tabela a seguir mostra as principais propriedades do objeto **Document**:

Propriedade	Descrição
anchors	Mostra uma relação com todas as âncoras do documento.
applets	Mostra uma relação com todos os applets do documento.
body	Mostra o elemento body do documento.
cookie	Todos os pares nome/valor de cookies encontrados no documento são mostrados.
documentMode	Mostra qual modo o navegador utiliza na renderização do documento.
domain	O servidor que carregou o documento tem seu nome de domínio mostrado.
forms	Mostra uma relação com todos os formulários encontrados no documento.
images	Mostra uma relação com todas as imagens encontradas no documento.
lastModified	A hora e a data da última modificação feita no documento são mostradas.
links	Mostra uma relação com todos os links encontrados no documento.
readyState	O status de carregamento do documento é mostrado.
referrer	O URL do documento que carregou o documento corrente é mostrado.
title	Mostra o título do documento ou permite que o mesmo seja definido.
URL	O URL completo do documento é mostrado.

Veja a seguir um exemplo da utilização de algumas propriedades do objeto **Document**:

```
<script>

    console.log("anchors", document.anchors);
    console.log("applets", document.applets);
    console.log("body", document.body);
    console.log("cookie", document.cookie);
    console.log("documentMode", document.documentElement);
    console.log("forms", document.forms);
    console.log("images", document.images);
    console.log("lastModified", document.lastModified);
    console.log("links", document.links);
    console.log("readyState", document.readyState);
    console.log("referrer", document.referrer);
    console.log("title", document.title);
    console.log("URL", document.URL);

</script>
```

8.2.1.2. Métodos do objeto Document

A tabela a seguir mostra os principais métodos do objeto **Document**:

Método	Descrição
close()	O fluxo de saída que foi aberto com open() é fechado com este método.
getElementsByName()	Os elementos com um nome específico podem ser acessados por este método.
open()	Com este método, um fluxo de saída é aberto para coletar a saída de write() ou writeln() .
write()	O método write() escreve um código JavaScript, ou mesmo uma expressão HTML, em um documento.
writeln()	Este método tem a mesma função que write() , porém, ele adiciona um caractere em uma nova linha após cada declaração.

8.2.2. Objeto Event

O objeto **Event** é tratado nesta apostila no capítulo referente a Eventos. Lá estão listados todos os métodos e propriedades relativos a este objeto, entre outras informações.

8.2.3. Objeto HTMLElement

Veja a seguir as propriedades e métodos que você pode utilizar em todos os elementos HTML.



As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados por todos os elementos.

8.2.3.1. Propriedades de HTMLElement

A tabela a seguir mostra as principais propriedades do objeto **HTMLElement**:

Propriedade	Descrição
accessKey	Essa propriedade informa a chave de acesso para um elemento ou permite que você a defina.
className	Essa propriedade informa o atributo de classe de um elemento ou permite que você o defina.
clientHeight	A altura visível do conteúdo de uma página é mostrada, porém, essa informação não inclui bordas, margens ou barras de rolagem.
clientWidth	A largura visível do conteúdo de uma página é mostrada, porém, essa informação não inclui bordas, margens ou barras de rolagem.
dir	Essa propriedade informa a direção do texto em um elemento ou permite que você a defina.

Propriedade	Descrição
id	A ID de um elemento é mostrada ou definida usando essa propriedade.
innerHTML	Essa propriedade retorna os conteúdos HTML de um elemento, incluindo o texto, ou permite que você os defina.
lang	O código de idioma de um elemento é mostrado, e também pode ser definido, por meio dessa propriedade!
offsetHeight	Essa propriedade mostra a altura de um elemento sem incluir as margens, mas incluindo bordas e padding, caso existam.
offsetLeft	A posição de offset horizontal do elemento corrente é mostrada, sempre de modo relativo a seu contêiner de offset.
offsetParent	Essa propriedade mostra o contêiner de offset de um elemento.
offsetTop	A posição de offset vertical do elemento corrente é mostrada, sempre de modo relativo a seu contêiner de offset.
offsetWidth	Essa propriedade mostra a largura de um elemento sem incluir as margens, mas incluindo bordas e padding, caso existam.
scrollHeight	Essa propriedade mostra a altura total de um elemento, inclusive considerando áreas que as barras de rolagem possam ocultar.
scrollLeft	A distância entre o atual limite visível do lado esquerdo e o verdadeiro limite do lado esquerdo de um elemento é mostrada usando essa propriedade.
scrollTop	A distância entre o atual limite superior visível e o verdadeiro limite superior de um elemento é mostrada usando essa propriedade.
scrollWidth	Essa propriedade mostra a largura total de um elemento, inclusive considerando áreas que as barras de rolagem possam ocultar.
style	Esta propriedade mostra o atributo de estilo (style) de um elemento ou permite que você o defina.

Propriedade	Descrição
tabIndex	Esta propriedade mostra a ordem de guias em um elemento ou permite que você a defina.
title	Esta propriedade mostra o atributo de título (title) em um elemento ou permite que você a defina.

Veja a seguir um exemplo da utilização de algumas propriedades do objeto **HTMLElement**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7 <body>
8
9     <div name="elemento"></div>
10
11 <script>
12
13 var div = document.getElementsByName("elemento")[0];
14
15 div.style.width = '400px';
16 div.style.height = '200px';
17 div.style.backgroundColor = 'lightblue';
18 div.style.color = 'white';
19
20 div.innerHTML = "DIV";
21
22 </script>
23 </body>
24 </html>
```

8.2.3.2. Método de HTMLElement

O único método de HTMLElement, **toString()**, tem a função de converter um elemento em uma string.

Veja a seguir um exemplo da utilização do método do objeto **HTMLElement**:

```
<script>

var div = document.getElementsByName("elemento")[0];

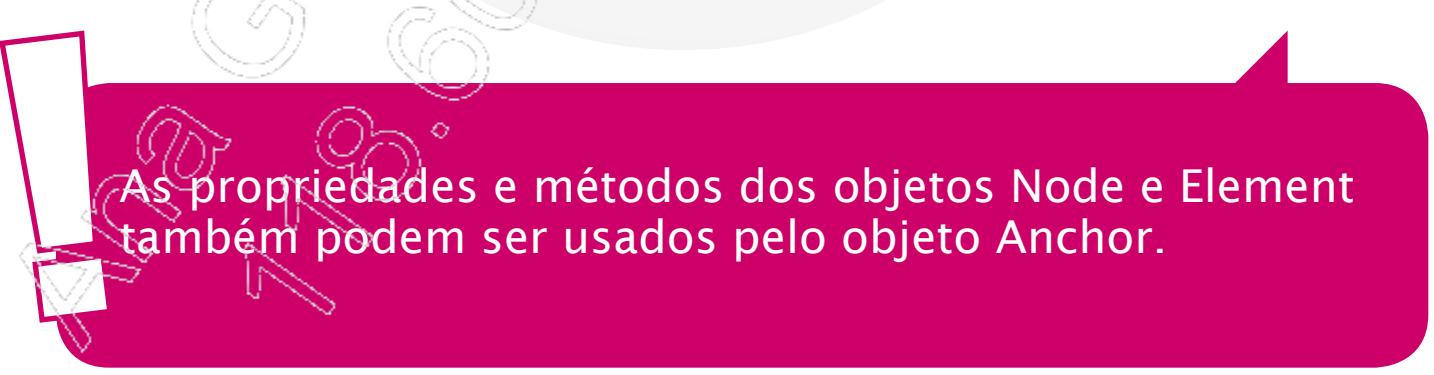
console.log(div.toString());

</script>
```

8.2.4. Objeto Anchor

Com a função de representar um hiperlink HTML, o objeto **Anchor** é criado sempre que a tag `<a>` for adicionada a um documento HTML. Com este objeto você pode criar um link para um documento diferente, ou até mesmo para um diferente ponto do mesmo documento. Para isso, você usa os atributos **href** e **name**, respectivamente. Veremos mais detalhes sobre esses atributos adiante.

Você pode encontrar uma âncora específica utilizando a propriedade **anchors** do objeto **Document**, ou acessá-la por meio de **getElementById()**.



As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Anchor**.

Veja a seguir um exemplo da utilização do objeto **Anchor**:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7  <body>
8
9      <a href="#1" name="link_1">Link 1</a>
10     <a href="#2" name="link_2">Link 2</a>
11     <a href="#3">Link 3</a>
12
13 <script>
14
15 var anchors = document.anchors;
16
17 console.log(anchors); //Apenas os elementos "a" com "name" são exibidos
18
19 </script>
20 </body>
21 </html>
```

8.2.4.1. Propriedades do objeto Anchor

A tabela a seguir mostra as principais propriedades do objeto **Anchor**. Em todos os casos, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Descrição
charset	Este atributo tem a função de especificar qual o conjunto de caracteres é usado no documento definido pelo atributo href .
href	href é o atributo que define o destino de um link.
hreflang	Este atributo tem a função de especificar o idioma usado no documento definido pelo atributo href .
name	O atributo name cria um marcador de página dentro de um documento, especificando o nome de uma âncora, sendo possível criar um link para pontos específicos de uma mesma página.
rel	Este atributo mostra a relação entre o documento atual e o documento referido no link.

Propriedade	Descrição
rev	Este atributo mostra a relação entre o documento referido no link e o documento atual.
target	O atributo target indica onde será aberto o documento referido no link.
type	O atributo type indica qual o tipo MIME do documento referido no link.

Veja a seguir um exemplo da utilização de algumas propriedades do objeto **Anchor**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7 <body>
8
9   <a href="#1" name="link_1">Link 1</a>
10  <a href="#2" name="link_2">Link 2</a>
11    <a href="#3">Link 3</a>
12
13 <script>
14
15 var anchors = document.anchors;
16
17 var anchor_1 = anchors[0];
18
19 console.dir(anchor_1); //Utilize o método "dir" do console para visualizar as propriedades e métodos de um HTMLElement
20
21 </script>
22 </body>
23 </html>
```

8.2.5. Objeto Area

Este objeto tem a função de representar uma área dentro uma imagem com regiões que podem ser clicadas, que chamamos de imagem mapeada HTML. Essas imagens mapeadas são muito utilizadas para criar menus interativos sobre imagens, como, por exemplo, um mapa nacional que oferece diferentes links quando você clica em cada estado. Um objeto **Area** é criado sempre que for encontrada uma tag **<area>** em um documento HTML.

As propriedades e métodos dos objetos **Node e **Element** também podem ser usados pelo objeto **Area**.**

8.2.5.1. Propriedades do objeto Area

A tabela a seguir mostra as principais propriedades do objeto **Area**. Em todos os casos, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Descrição
alt	O atributo alt consiste em um texto que é exibido no lugar de uma imagem que não pode ser carregada.
coords	O atributo coords representa as coordenadas x e y de uma região.
hash	O destino de um link é definido no atributo hash .
host	Este atributo refere-se à parte hostname:port do valor do atributo href .
hostname	Este atributo refere-se à parte hostname do valor do atributo href .
href	Refere-se ao valor do atributo href de uma área específica.
noHref	Este atributo determina que certa área não possui um link definido.
pathname	É a parte do nome de caminho encontrada no valor do atributo href .
port	Este atributo refere-se à parte port do valor do atributo href .
protocol	Este atributo refere-se à parte do protocolo do valor do atributo href .
search	Este atributo refere-se à parte de querystring do valor do atributo href .
shape	O atributo shape refere-se à forma de uma área.
target	Este atributo define o local onde será aberto o documento ao qual o link se refere.

8.2.6. Objeto Base

Para definir um endereço ou alvo padrão para todos os links em uma página você pode usar um objeto **Base**, que representa um elemento base HTML.

Você pode criar um objeto **Base** inserindo a tag `<base>` em um documento HTML.

As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Base**.

8.2.6.1. Propriedades do objeto Base

A tabela a seguir mostra as principais propriedades do objeto **Base**, que têm a função de visualizar um valor de atributo ou defini-lo.

Propriedade	Descrição
<code>href</code>	Esse atributo define um endereço de URL comum para todos os URLs relativos de uma mesma página.
<code>target</code>	Este atributo define o local onde serão abertos todos os links de uma página.

8.2.7. Objeto Body

O elemento de corpo do HTML, que pode ser chamada também de body, é representado pelo objeto **Body**, e é onde você encontra todo o conteúdo de um documento. Através desse objeto, podemos manipular elementos como tabelas, texto, links, listas, etc.

As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Body**.

8.2.7.1. Propriedades do objeto Body

A tabela a seguir mostra as principais propriedades do objeto **Body**. Em todos os casos, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Descrição
aLink	Este atributo define a cor de um link quando é clicado, ou seja, quando se torna ativo.
background	Este atributo define a imagem de fundo que será apresentada numa página.
bgColor	A cor de fundo de um documento é definida por meio desse atributo.
link	Links que ainda não foram visitados têm sua cor definida por esse atributo.
text	O texto presente em um documento pode ter sua cor especificada nesse atributo.
vLink	Este atributo define a cor de links já visitados por um usuário.

Veja a seguir um exemplo da utilização de algumas propriedades do objeto **Body**:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7  <body>
8
9      <input type="color" id="inputCor" value="#FFFFFF">
10
11 <script>
12
13 var input = document.getElementById("inputCor");
14
15 input.addEventListener("change", function(event) {
16
17     document.body.bgColor = this.value;
18
19 }, false);
20
21 </script>
22 </body>
23 </html>
```

8.2.7.2. Eventos do objeto Body

Você pode definir o que acontece assim que uma página é carregada utilizando o evento do objeto **Body** que se chama **onload**.

Sua sintaxe é:

```
onload="Codigo_JavaScript"
```

Em que:

- **Codigo_JavaScript**: é um script que será executado quando o evento ocorre, ou seja, quando uma página é carregada.

Veja a seguir um exemplo da utilização do evento **onload**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 var input_1 = document.getElementById("inputCor");
8
9 console.log("1. Neste momento o input existe?", (input_1));
10
11 function bodyPronto() {
12
13     var input_2 = document.getElementById("inputCor");
14
15     console.log("2. Neste momento o input existe?", (input_2));
16
17 }
18 </script>
19 </head>
20 <body onLoad="bodyPronto()">
21
22     <input type="color" id="inputCor" value="#FFFFFF">
23
24 </body>
25 </html>
```

8.2.8. Objeto Button

Os botões que você encontra em um documento HTML são representados pelo objeto **Button**. Sempre que a tag **<button>** é inserida em um documento, um botão é criado, e seu objeto **Button** permite que você defina seu conteúdo, como texto, imagens e suas funções, o que difere do processo de criação de botões por meio do elemento **input**.

As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Button**.

8.2.8.1. Propriedades do objeto Button

A tabela a seguir mostra as principais propriedades do objeto **Button**. Com exceção do atributo **form**, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Descrição
disabled	Este atributo habilita ou desabilita um botão.
form	Indica o formulário que possui o botão.
name	Este atributo define o nome do botão.
type	Este atributo refere-se ao tipo de um botão, que pode ser: <ul style="list-style-type: none">• submit: botão de envio do formulário;• button: qualquer botão clicável;• reset: limpa os dados inseridos em um formulário. Lembre-se sempre que o valor padrão para botões no Internet Explorer é o tipo button . Já nos outros navegadores, o tipo padrão é submit .
value	O valor fundamental associado a um botão é definido por meio desse atributo.

8.2.9. Objeto Form

Um meio possível para enviar dados para um servidor é o formulário. Sempre que você criar uma tag **<form>** em um documento HTML, você irá gerar um formulário, e por consequência, um objeto **Form**. Um formulário fornece diversos elementos de entrada de dados, como campos de texto, botões de verificação, botões de envio, entre diversas outras opções que permitam a entrada de dados por parte de um usuário.

As propriedades e métodos dos objetos Node e Element também podem ser usados pelo objeto Form.

8.2.9.1.Coleção do objeto Form

Todos os elementos de um formulário podem ser mostrados com a utilização da coleção **elements[]**. Essa coleção mostra esses dados na forma de um array.

Sua sintaxe é:

```
formObject.elements[] .propriedade
```

Veja a seguir um exemplo da utilização da coleção do objeto **Form**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 function bodyPronto(){
8
9     console.log(document.formulario); //Obter formulário pelo "name"
10    console.log(document.forms[0]); //Obter formulário pelo array "forms"
11
12    console.log(document.formulario.nome)
13    console.log(document.forms[0].nome);
14
15 }
16 </script>
17 </head>
18 <body onLoad="bodyPronto()">
19
20     <form name="formulario">
21
22         <input type="text" name="nome" placeholder="Digite o nome...">
23
24         <button type="submit">Enviar</button>
25
26     </form>
27
28 </body>
29 </html>
```

8.2.9.2. Propriedades do objeto Form

A tabela a seguir mostra as principais propriedades do objeto **Form**. Com exceção do atributo **length**, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Descrição
acceptCharset	Este atributo representa quais conjuntos de caracteres são aceitos pelo servidor que receberá os dados do formulário.
action	Quando um formulário é enviado, este atributo mostra para onde enviar seus dados.
enctype	Este atributo mostra como os dados enviados pelo formulário são codificados.
length	Este atributo mostra o número de elementos contidos em um formulário.
method	O método pelo qual os dados são enviados para o servidor é definido nesse atributo.
name	Este atributo define o nome do formulário.
target	Este atributo especifica onde abrir o URL da ação de resposta do servidor.

8.2.9.3. Métodos do objeto Form

A tabela a seguir mostra os principais métodos do objeto **Form**:

Método	Descrição
reset()	Este método apaga os dados de um formulário por completo.
submit()	Este método envia os dados de um formulário.

Veja a seguir um exemplo da utilização de alguns métodos do objeto **Form**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 function enviar() {
8
9     if(confirm("Deseja buscar realmente?")){
10         document.forms[0].submit();
11     }else if(confirm("Deseja limpar o formulário?")){
12         document.forms[0].reset();
13     }
14 }
15 </script>
16 </head>
17 <body>
18
19     <form action="https://www.google.com" target="_blank">
20
21         <input type="text" name="q" placeholder="Buscar por...">
22
23         <button type="button" onClick="enviar()">Buscar</button>
24
25     </form>
26
27 </body>
28 </html>
```

8.2.9.4. Eventos do objeto Form

A tabela a seguir mostra os principais eventos do objeto **Form**:

Evento	Ocorrência do evento
onreset	Ocorre quando você clica o botão Reset.
onsubmit	Ocorre quando você clica o botão Enviar (submit).

8.2.10. Objetos Frame e IFrame

O objeto **Frame** representa um frame HTML. A tag `<frame>` gera uma janela específica (frame) dentro de um conjunto frames (frameset). Já o objeto **IFrame** constitui um frame inline HTML, ou seja, um frame interno que pode carregar páginas individualmente. Sempre que suas tags são inseridas, `<frame>` ou `<iframe>`, cria-se o objeto **Frame** ou **IFrame**, respectivamente.



As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelos objetos **Frame** e **IFrame**.

8.2.10.1. Propriedades dos objetos Frame e IFrame

A tabela a seguir mostra as principais propriedades dos objetos **Frame** e **IFrame**. Com exceção dos atributos **contentDocument** e **contentWindow**, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Descrição
align	Este atributo especifica, de acordo com o elemento ao redor, o alinhamento horizontal e vertical de um iframe.
contentDocument	Retorna o objeto Document criado por um frame/iframe.
contentWindow	Retorna o objeto Window criado por um frame/iframe.
frameBorder	Neste atributo você define se há ou não a visualização de uma borda em torno de um frame.
height	A altura de um iframe é definida nesse atributo.

Propriedade	Descrição
longDesc	Este atributo define um URL para uma página que tenha uma descrição muito longa do conteúdo de um frame/iframe.
marginHeight	As margens superior e inferior de um frame/iframe são definidas nesse atributo. As medidas são sempre em pixels.
marginWidth	As margens esquerda e direita de um frame/iframe são definidas nesse atributo. As medidas são sempre em pixels.
name	Este atributo representa o nome de um frame/iframe.
noResize	Para que um frame não possa ser redimensionado, utilize esta propriedade.
scrolling	Este atributo permite que você escolha entre mostrar barras de rolagem ou não em um frame/iframe.
src	Este atributo define o URL do documento que é mostrado em um frame/iframe.
width	A largura de um iframe é definida nesse atributo.

JavaScript

Veja a seguir um exemplo da utilização de algumas propriedades dos objetos **Frame** e **IFrame**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 function bodyPronto(){
8
9     var iframe = document.getElementsByName("frame").item(0);
10
11    iframe.height = window.innerHeight - 100;
12
13    var input = document.forms[0].endereco;
14    var button = document.getElementsByTagName("button").item(0);
15
16    button.addEventListener("click", function(event){
17
18        iframe.src = input.value;
19
20    }, false);
21
22 }
23 </script>
24 </head>
25 <body onLoad="bodyPronto()">
26     <form action="#">
27         <input
28             type="url"
29             name="endereco"
30             placeholder="Digite o endereço..."
31             value="http://www.jotaquery.com.br"
32             style="width:90%">
33         <button type="button">Ir</button>
34     </form>
35
36     <iframe width="100%" border="0" name="frame"></iframe>
37
38 </body>
39 </html>
```

8.2.10.2. Evento dos objetos Frame e IFrame

Os objetos Frame e Iframe possuem um evento chamado onload, que define o que ocorre assim que um frame ou iframe é carregado.

Sua sintaxe é:

```
onload="Codigo_JavaScript"
```

Em que:

- **Codigo_Javascript**: é um script que será executado quando o evento ocorre.

8.2.11. Objetos Input

Existem diversos tipos de objeto **Input**, conforme você verá a seguir, cada um deles com uma função específica.

8.2.11.1. Objeto Button

Um botão clicável em um formulário pode ser representado pelo objeto **Button**, criado em um formulário HTML por meio da tag `<input type="button">`. Esse botão vai ter a função de ativar um script assim que for clicado. Também é possível criar um botão usando a tag `<button>`, conforme já foi mostrado nesse mesmo capítulo.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar o objeto **Button** que deseja.

 As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelos objeto **Button**.

- **Propriedades do objeto Button**

A tabela a seguir mostra as principais propriedades do objeto **Button**:

Propriedade	Descrição
disabled	Por meio dessa propriedade você pode definir se um botão está desabilitado ou não, ou apenas visualizar o valor do atributo.
form	Indica o formulário que possui o botão de entrada (input).
name	Por meio dessa propriedade você pode definir o nome de um botão de entrada, ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
type	Indica qual o tipo de elemento de formulário do qual o botão faz parte.
value	O texto que é mostrado no botão é definido pelo valor deste atributo.

8.2.11.2. Objeto Checkbox

Uma caixa de verificação é representada pelo objeto **Checkbox**, criado em um formulário HTML por meio da tag `<input type="checkbox">`. Com elas, você pode escolher entre um número determinado de opções, e sua resposta pode ser uma ou mais de uma opção.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar o objeto **Checkbox** que deseja.

! As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelos objeto **Checkbox**.

- **Propriedades do objeto Checkbox**

A tabela a seguir mostra as principais propriedades do objeto **Checkbox**:

Propriedade	Descrição
disabled	Por meio dessa propriedade você pode definir se uma caixa de verificação está desabilitada ou não, ou apenas visualizar o valor do atributo.
checked	Este atributo mostra se a caixa de verificação está ou não marcada.
defaultChecked	Este atributo define qual será o valor padrão da caixa de verificação: marcado ou não marcado.
form	Indica o formulário que possui a caixa de verificação.
name	Por meio dessa propriedade você pode definir o nome de uma caixa de verificação, ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
type	Indica qual o tipo de elemento de formulário do qual a caixa de verificação faz parte.
value	O conteúdo dessa propriedade não é mostrada na interface do usuário. Seu valor apenas é enviado para o servidor quando o formulário é enviado, e consiste no nome da caixa de verificação quando esta está marcada. Quando não existe marcação numa caixa de verificação, seu valor simplesmente não é informado ao servidor.

Veja a seguir um exemplo da utilização de algumas propriedades do objeto **Checkbox**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <style>
7 .ok { text-decoration:line-through; color:rgba(0,0,0,.5); }
8 </style>
9 <script>
10 function bodyPronto() {
11
12     var checkboxes = document.querySelectorAll('[type="checkbox"]');
13
14     for(var i = 0; i < checkboxes.length, checkbox = checkboxes[i]; i++) {
15
16         checkbox.addEventListener("change", function(){
17
18             this.parentNode.className = (this.checked)?'ok':'';
19
20         }, false);
21
22     }
23
24 }
25 </script>
26 </head>
27 <body onLoad="bodyPronto()">
28     <form action="#">
29
30         <label><input type="checkbox" name="opcao_1" value="Google Chrome"> Google Chrome</label><br/>
31         <label><input type="checkbox" name="opcao_2" value="Mozilla Firefox"> Mozilla Firefox</label><br/>
32         <label><input type="checkbox" name="opcao_3" value="Opera"> Opera</label><br/>
33         <label><input type="checkbox" name="opcao_4" value="Safari"> Safari</label><br/>
34         <label><input type="checkbox" name="opcao_5" value="Internet Explorer"> Internet Explorer</label><br/>
35
36     </form>
37 </body>
38 </html>
```

8.2.11.3. Objeto FileUpload

Quando é necessário o envio de arquivos em um formulário, você pode usar o objeto **FileUpload**, usando a tag **<input type="file">**. Com isso, você insere no formulário um controle de entrada de texto juntamente com um botão de navegação. Clicando nesse botão, o usuário acessa uma caixa de diálogo que possibilita a escolha do arquivo que será enviado.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar o objeto **FileUpload** que deseja.

As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelos objeto **FileUpload**.

- **Propriedades do objeto FileUpload**

A tabela a seguir mostra as principais propriedades do objeto **FileUpload**:

Propriedade	Descrição
disabled	Por meio dessa propriedade você pode definir se os comandos de FileUpload estão desabilitados ou não, ou apenas visualizar o valor do atributo.
accept	Este atributo define qual o tipo de conteúdo é aceito para upload em uma lista separada por vírgulas, ou apenas visualiza o valor do atributo.
form	Indica o formulário que possui o objeto FileUpload .
name	Por meio dessa propriedade você pode definir o nome de um objeto FileUpload , ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
type	Indica qual o tipo de elemento de formulário do qual o objeto FileUpload faz parte.
value	Para visualizar o caminho, ou nome, do arquivo escolhido, utilize esta propriedade.

Veja a seguir um exemplo da utilização de algumas propriedades do objeto **FileUpload**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 function bodyPronto() {
8
9     document.getElementsByTagName("button").item(0).addEventListener("click", function() {
10
11         var fileUpload = document.createElement("input");
12
13         fileUpload.setAttribute("type", "file");
14         fileUpload.setAttribute("name", "arquivo[]");
15         fileUpload.setAttribute("accept", "image/*");
16
17         document.getElementsByTagName("fieldset").item(0).appendChild(fileUpload);
18
19     }, false);
20 }
21 </script>
22 </head>
23 <body onLoad="bodyPronto()">
24     <form action="#">
25
26         <button type="button">Adicionar Arquivo</button>
27
28         <fieldset></fieldset>
29
30     </form>
31
32 </body>
33 </html>
```

8.2.11.4. Objeto Hidden

Você pode criar, em um formulário HTML, campos de entrada ocultos gerando o objeto **Hidden** por meio da tag `<input type="hidden">`. Com esse recurso, você cria um campo de entrada que é invisível para o usuário, mas que permite a você enviar para o servidor dados desses campos ocultos.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar o campo de entrada oculto que deseja.

As propriedades e métodos dos objetos **Node e **Element** também podem ser usados pelo objeto **Hidden**.**

- **Propriedades do objeto Hidden**

A tabela a seguir mostra as principais propriedades do objeto **Hidden**:

Propriedade	Descrição
form	Indica o formulário que possui o campo de entrada oculta.
name	Por meio dessa propriedade você pode definir o nome de um campo de entrada oculta, ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
type	Indica qual o tipo de elemento de formulário do qual o campo de entrada oculta faz parte.
value	O campo de entrada oculta tem seu valor padrão definido ou visualizado com essa propriedade.

8.2.11.5. Objeto Password

Um campo de senha é representado pelo objeto **Password**, gerado pela tag `<input type="password">`. O que diferencia o campo de senha de um campo de texto normal é que seu conteúdo é mostrado para o usuário de forma “mascarada”, com símbolos como asteriscos (*) ou bolhas (.) substituindo os caracteres inseridos.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar o campo de senha que deseja.

As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Password**.

- **Propriedades do objeto Password**

A tabela a seguir mostra as principais propriedades do objeto **Password**:

Propriedade	Descrição
defaultValue	Por meio desse atributo você pode definir o valor padrão do campo de senha ou apenas visualizar o valor da propriedade.
disabled	Por meio dessa propriedade você pode definir se o campo de senha está desabilitado ou não, ou apenas visualizar o valor do atributo.
form	Indica o formulário que possui o campo de senha.
maxLength	Delimita o número máximo de caracteres permitido em um campo de senha, ou apenas visualiza o valor deste atributo.
name	Por meio dessa propriedade você pode definir o nome de um campo de senha, ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
readOnly	Por meio dessa propriedade você pode definir se um campo de senha é somente leitura ou não, ou apenas visualizar o valor do atributo.
size	Este atributo refere-se ao comprimento do campo de senha, dado em número de caracteres.
type	Indica qual o tipo de elemento de formulário do qual o campo de senha faz parte.
value	Esta propriedade informa o valor padrão ou o valor inserido pelo usuário no campo de senha. É possível definir ou visualizar esse valor.

- **Método do objeto Password**

Para selecionar o conteúdo de um campo de senha, utilize o método **select()**.

Sua sintaxe é a seguinte:

```
passwordObject.select()
```

8.2.11.6. Objeto Radio

Botões de rádio, ou botões de opção, são representados em um formulário HTML pelo objeto **Radio**, criado pela tag `<input type="radio">`. Diferente das caixas de verificação, botões de rádio permitem que o usuário selecione apenas uma opção para enviar no formulário. Grupos são definidos pela propriedade **name**, portanto, botões de rádio com o mesmo nome fazem parte do mesmo grupo.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar o botão de rádio que deseja.

 As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Radio**.

- **Propriedades do objeto Radio**

A tabela a seguir mostra as principais propriedades do objeto **Radio**:

Propriedade	Descrição
checked	O estado de um botão de rádio, ou seja, se este se encontra marcado ou não, é mostrado ou definido por esse atributo.
defaultChecked	Este atributo define o estado padrão do valor do atributo checked .
disabled	Por meio dessa propriedade você pode definir se o botão de rádio está desabilitado ou não, ou apenas visualizar o valor do atributo.
form	Indica o formulário que possui o botão de rádio.
name	Por meio dessa propriedade você pode definir o nome do botão de rádio, ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
type	Indica qual o tipo de elemento de formulário do qual o botão de rádio faz parte.
value	O conteúdo dessa propriedade não é mostrada na interface do usuário. Seu valor apenas é enviado para o servidor quando o formulário é enviado, e consiste no nome do botão de rádio quando este está marcado. Quando não existe marcação num botão de rádio, seu valor simplesmente não é informado ao servidor.

Veja a seguir um exemplo da utilização de algumas propriedades do objeto **Radio**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 function bodyPronto() {
8
9     var radios = document.querySelectorAll('[name="sexo"]');
10
11    for(var i = 0; i < radios.length, radio = radios[i]; i++){
12
13        radio.addEventListener("change", function(){
14
15            document.getElementById("imagem").setAttribute("src", this.value+".png");
16
17        }, false);
18
19    }
20
21 }
22 </script>
23 </head>
24 <body onLoad="bodyPronto()">
25     <form action="#">
26
27         <label><input type="radio" name="sexo" value="Female"> Feminino</label>
28         <label><input type="radio" name="sexo" value="Male"> Masculino</label>
29         <br/>
30         <img id="imagem">
31
32     </form>
33 </body>
34 </html>
```

8.2.11.7. Objeto Reset

Para retornar todas as informações inseridas em um formulário para seus estados padrão, utilize um botão Reset, criado pela tag **<input type="reset">**, e que gera um objeto **Reset**. Lembre-se também que, sempre que um botão Reset é clicado, ele dispara o evento **onreset**.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar o botão Reset que deseja.

! As propriedades e métodos dos objetos **Node e **Element** também podem ser usados pelo objeto **Reset**.**

- **Propriedades do objeto Reset**

A tabela a seguir mostra as principais propriedades do objeto **Reset**:

Propriedade	Description
disabled	Por meio dessa propriedade você pode definir se o botão Reset está desabilitado ou não, ou apenas visualizar o valor do atributo.
form	Indica o formulário que possui o botão Reset.
name	Por meio dessa propriedade você pode definir o nome do botão Reset, ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
type	Indica qual o tipo de elemento de formulário do qual o botão Reset faz parte.
value	O texto que é mostrado no botão é definido pelo valor deste atributo.

8.2.11.8. Objeto Submit

Para enviar um formulário para o servidor, utilize um botão Submit (enviar), criado pela tag `<input type="submit">`, e que gera um objeto **Submit**. Lembre-se também que, sempre que um botão Submit é clicado, ele dispara o evento **onsubmit**.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar o botão Submit que deseja.

! As propriedades e métodos dos objetos **Node e **Element** também podem ser usados pelo objeto **Submit**.**

- **Propriedades do objeto Submit**

A tabela a seguir mostra as principais propriedades do objeto Submit:

Propriedade	Descrição
disabled	Por meio dessa propriedade você pode definir se o botão Submit está desabilitado ou não, ou apenas visualizar o valor do atributo.
form	Indica o formulário que possui o botão Submit.
name	Por meio dessa propriedade você pode definir o nome do botão Submit, ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
type	Indica qual o tipo de elemento de formulário do qual o botão Submit faz parte.
value	O texto que é mostrado no botão é definido pelo valor deste atributo.

8.2.11.9. Objeto Text

Para adicionar campos de entrada de texto em um formulário, utilize a tag `<input type="text">`, o que gera um objeto Text.

Ao fazer uma pesquisa no array `elements[]` de um formulário, como já foi visto também nesse capítulo, ou mesmo usando `getElementById()`, você pode acessar a caixa de entrada de texto que deseja.

As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Text**.

- **Propriedades do objeto Text**

A tabela a seguir mostra as principais propriedades do objeto **Text**:

Propriedade	Descrição
defaultValue	Por meio desse atributo você pode definir o valor padrão do campo de entrada de texto ou apenas visualizar o valor da propriedade.
disabled	Por meio dessa propriedade você pode definir se a caixa de entrada de texto está desabilitada ou não, ou apenas visualizar o valor do atributo.
form	Indica o formulário que possui a caixa de entrada de texto.
maxLength	Delimita o número máximo de caracteres permitido em um campo de entrada de texto, ou apenas visualiza o valor deste atributo.
name	Por meio dessa propriedade você pode definir o nome da caixa de entrada de texto, ou apenas visualizar o valor do atributo. É com o valor desse atributo que se torna possível a identificação dos dados do formulário quando estes são enviados ao servidor.
readOnly	Por meio dessa propriedade você pode definir se um campo de entrada de texto é somente leitura ou não, ou apenas visualizar o valor do atributo.
size	Este atributo refere-se ao tamanho (comprimento) do campo de entrada de texto, dado em número de caracteres.
type	Indica qual o tipo de elemento de formulário do qual o campo de entrada de texto faz parte.
value	Esta propriedade informa o valor padrão ou o valor inserido pelo usuário no campo de entrada de texto. É possível definir ou visualizar esse valor.

- **Método do objeto Text**

Para selecionar o conteúdo de um campo de entrada de texto, utilize o método **select()**.

Sua sintaxe é a seguinte:

```
textObject.select()
```

8.2.12. Objeto Link

Para inserir um link para uma fonte externa em um documento HTML, utilize a tag **<link>**, que cria um objeto **Link**. Lembre-se que este elemento de link deve ser sempre colocado na seção de cabeçalho (head) do documento.

 As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Link**.

Veja a seguir um exemplo da utilização do objeto **Link**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <link href="http://fonts.googleapis.com/css?family=Akronim" rel="stylesheet" type="text/css">
7 <style>
8 h1 { font-family: 'Akronim', cursive; font-size:500%; color:rgba(255,102,0,.7); }
9 </style>
10 <script>
11 console.dir(document.getElementsByTagName("link").item(0));
12 </script>
13 </head>
14 <body>
15
16     <h1>JavaScript</h1>
17
18 </body>
19 </html>
```

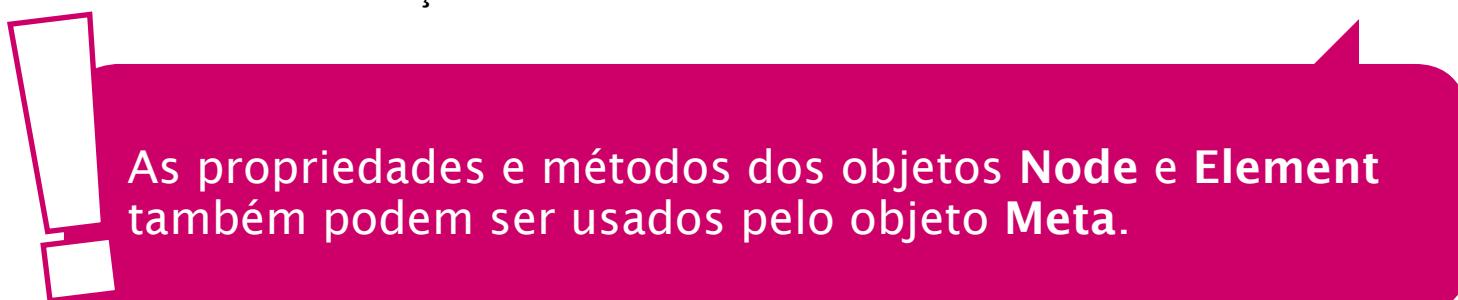
8.2.12.1. Propriedades do objeto Link

A tabela a seguir mostra as principais propriedades do objeto **Link**. Em todos os casos, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Description
charset	Este atributo é relativo à codificação de caracteres do documento referido no link.
href	Este atributo define o URL de destino de um link.
hreflang	Este atributo tem a função de especificar o código do idioma usado no documento referido no link.
media	Este atributo mostra o tipo de mídia do qual faz parte o documento referido no link.
rel	Este atributo mostra a relação entre o documento atual e o documento referido no link.
rev	Este atributo mostra a relação inversa de rel , ou seja, entre o documento referido no link e o documento atual.
type	Indica qual o tipo de elemento de formulário do qual o documento referido no link faz parte.

8.2.13. Objeto Meta

Você pode criar um metaelemento HTML com a tag `<meta>`, que gera um objeto **Meta**. Lembre-se que este metaelemento deve ser sempre colocado na seção de cabeçalho (`head`) do documento. A função desse metaelemento é fornecer informações sobre um documento HTML, como autor, descrição, palavras-chave, registro da última modificação, etc. Diversos serviços da Web fazem uso dessas informações para tornar a navegação mais interativa. Navegadores e ferramentas de busca podem personalizar o conteúdo mostrado ao usuário de acordo com as informações dos metadados.



Veja a seguir um exemplo da utilização do objeto **Meta**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 console.dir(document.getElementsByTagName("meta").item(0));
8 </script>
9 </head>
10 <body>
11
12
13
14 </body>
15 </html>
```

8.2.13.1. Propriedades do objeto Meta

A tabela a seguir mostra as principais propriedades do objeto **Meta**. Em todos os casos, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Descrição
content	Este atributo refere-se ao conteúdo da metainformação. Ele depende diretamente dos valores das propriedades httpEquiv e name .
httpEquiv	Este atributo está ligado à informação da propriedade content , e fornece a ela um cabeçalho HTTP.
name	Este atributo é referente ao nome da informação no atributo content .
scheme	Define o formato a ser usado para a interpretação que o valor de content deve receber.

8.2.14. Objeto Object

Objetos diversos podem ser adicionados a uma página HTML, como imagens, elementos de áudio e vídeo, applets Java, arquivos PDF, ActiveX, Flash, etc. Eles podem ser adicionados a um documento usando a tag <object>, que cria um objeto **Object**.

 As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Object**.

8.2.14.1. Propriedades do objeto Object

A tabela a seguir mostra as principais propriedades do objeto **Object**:

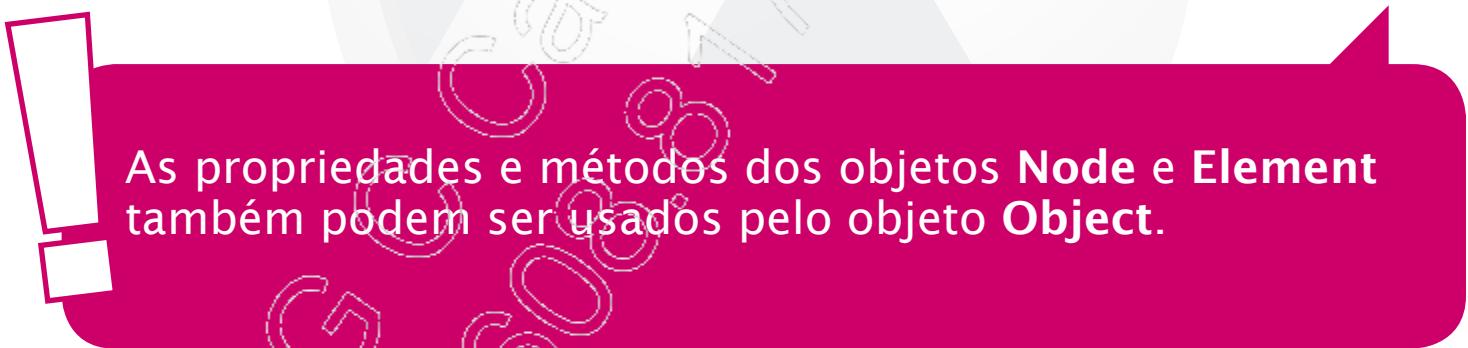
Propriedade	Descrição
align	Este atributo especifica, de acordo com o elemento ao redor, o alinhamento horizontal e vertical do objeto.
archive	Este atributo mostra uma string que pode ser usada para implementar, em um objeto, sua própria funcionalidade de arquivo.
border	Este atributo refere-se à borda ao redor do objeto.
code	O URL do arquivo que contém a classe Java compilada é definido ou mostrado.
codeBase	Este atributo refere-se ao URL do componente, que pode ser definido ou mostrado.
form	Indica qual é o formulário pai do objeto.
height	Este atributo refere-se à altura do objeto.
hspace	Este atributo refere-se à margem horizontal do objeto.
name	Este atributo refere-se ao nome do objeto.

Propriedade	Descrição
standby	Quando o objeto está sendo carregado, é possível por meio deste atributo definir uma mensagem a ser mostrada, ou apenas verificar seu valor.
vspace	Este atributo refere-se à margem vertical do objeto.
width	Este atributo refere-se à largura do objeto.

8.2.15. Objeto Option

Para criar uma lista drop-down com diversas opções em um formulário HTML, onde apenas uma opção é selecionável, utilize a tag `<option>`, que gera um objeto **Option** assim que é criada.

Ao fazer uma pesquisa no array `elements[]` de um formulário, como já foi visto também nesse capítulo, ou mesmo usando `getElementById()`, você pode acessar um objeto **Option** que deseja.



As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Object**.

8.2.15.1. Propriedades do objeto Option

A tabela a seguir mostra as principais propriedades do objeto **Option**:

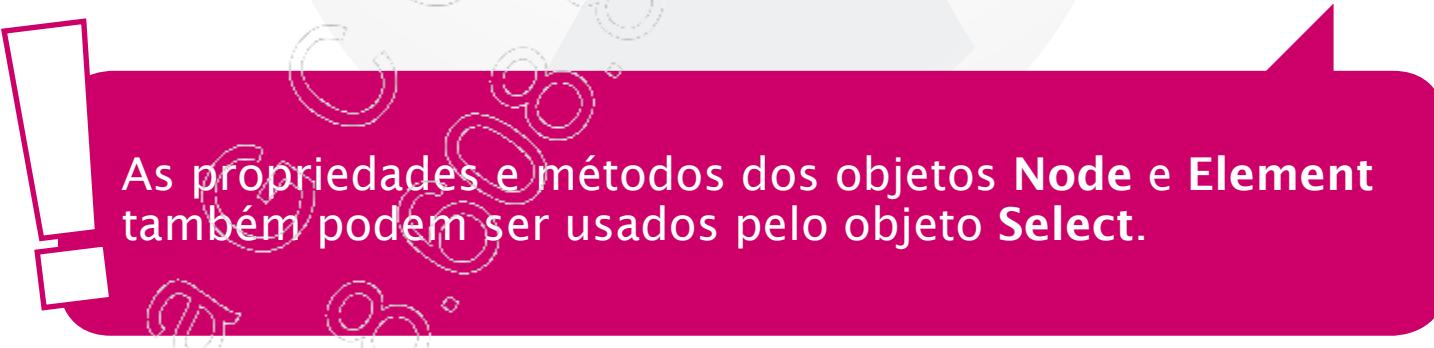
Propriedade	Descrição
defaultSelected	Este atributo define qual será o valor padrão do atributo selected .
disabled	Por meio dessa propriedade você pode definir se uma opção está desabilitada ou não, ou apenas visualizar o valor do atributo.

Propriedade	Descrição
form	Indica o formulário que possui a opção.
index	A posição que uma opção ocupa na lista de drop-down é definida ou mostrada por essa propriedade.
selected	O valor do atributo selecionado é definido ou mostrado através desta propriedade.
text	Cada elemento de opção tem seu texto definido ou mostrado por essa propriedade.
value	Este atributo refere-se ao valor que será enviado ao servidor.

8.2.16. Objeto Select

Já o objeto **Select** é gerado sempre que criamos a tag <select>, e este se refere a uma lista drop-down com diversas opções disponíveis, onde você pode fazer mais de uma escolha de opção.

Ao fazer uma pesquisa no array **elements[]** de um formulário, como já foi visto também nesse capítulo, ou mesmo usando **getElementById()**, você pode acessar um objeto **Select** que deseja.



As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Select**.

8.2.16.1. Coleção do objeto Select

Para ter acesso a um array contendo todas as opções de uma lista drop-down, você deve acessar a coleção **options**. Lembre-se que para cada tag <option> você encontra um elemento array referente, iniciando pelo 0.

Sua sintaxe é:

```
selectObject.options
```

8.2.16.2. Propriedades do objeto Select

A tabela a seguir mostra as principais propriedades do objeto **Select**:

Propriedade	Descrição
disabled	Por meio dessa propriedade você pode definir se a lista drop-down está desabilitada ou não, ou apenas visualizar o valor do atributo.
form	Indica o formulário que possui a lista drop-down.
length	O número de opções que existem em uma lista drop-down é mostrada por este atributo.
multiple	É nesta propriedade que você define ou confere o valor da quantidade possível de opções que podem ser selecionadas na lista drop-down: apenas uma, ou múltiplas opções.
name	Este atributo mostra ou permite que você defina o nome de uma lista drop-down.
selectedIndex	Na lista drop-down, a opção selecionada tem seu valor de índice definido ou mostrado por este atributo.
size	O tamanho da lista drop-down, ou seja, o número de opções visíveis na lista, é definido ou mostrado por essa propriedade.
type	Indica qual o tipo de elemento de formulário do qual a lista drop-down faz parte.

8.2.16.3. Métodos do objeto Select

A tabela a seguir mostra os principais métodos do objeto **Select**:

Método	Descrição
add()	Este método faz com que uma opção seja acrescentada em uma lista drop-down.
remove()	Este método faz com que uma opção seja removida de uma lista drop-down.

JavaScript

Veja a seguir um exemplo da utilização de alguns métodos do objeto **Select**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7 function ocultarTodosVeiculos(){
8     var optionsVeiculos = document.forms[0].Veiculos.children;
9     for(var i = 0; i < optionsVeiculos.length, option = optionsVeiculos[i]; i++){
10         option.style.display = "none";
11     }
12 }
13 function bodyPronto(){
14     ocultarTodosVeiculos();
15     document.forms[0].elements[0].addEventListener("change", function(){
16         ocultarTodosVeiculos();
17         var optionsVeiculos = document.forms[0].Veiculos.children;
18         for(var i = 0; i < optionsVeiculos.length, option = optionsVeiculos[i]; i++){
19             var Attr = option.attributes.getNamedItem('idmarca');
20             if(Attr) if(Attr.value == this.value) option.style.display = 'block';
21         }
22     }, false);
23 }
24 </script>
25 </head>
26 <body onLoad="bodyPronto()">
27     <form>
28         <select name="Marcas">
29             <option selected>-</option><option value="1">Porsche</option>
30             <option value="2">Ford</option><option value="3">Chevrolet</option>
31         </select>
32         <select name="Veiculos">
33             <option selected>-</option>
34             <option idmarca="1" value="1">Cayman</option>
35             <option idmarca="1" value="2">Carrera</option>
36             <option idmarca="1" value="3">Boxster</option>
37             <option idmarca="2" value="4">GT</option>
38             <option idmarca="2" value="5">Focus</option>
39             <option idmarca="2" value="6">Mustang</option>
40             <option idmarca="3" value="7">Camaro</option>
41             <option idmarca="3" value="8">Corvette</option>
42             <option idmarca="3" value="9">Impala</option>
43         </select>
44     </form>
45 </body>
46 </html>
```

8.2.17. Objeto Style

Uma declaração de estilo é representada pelo objeto **Style**. É possível acessar este objeto de duas maneiras: a partir do documento, ou a partir dos elementos onde este determinado estilo está aplicado.

Sua sintaxe é:

```
document.getElementById("id").style.property="value"
```

As propriedades do objeto **Style** são diversas, e podem ser divididas em algumas categorias para facilitar a compreensão. São elas:

- **Background**: essas propriedades definem os parâmetros do plano de fundo, como cor, imagem, posição de imagem e padrão de repetição;
- **Border/Outline**: essas propriedades definem tipo, cor e estilo das bordas, entre outras opções;
- **Generated Content**: essas propriedades administram o conteúdo gerado em uma página;
- **List**: essas propriedades trabalham com imagem, posicionamento e estilo de listas;
- **Misc**: apresenta apenas a propriedade **cssText**, que define ou retorna como uma string o conteúdo de uma declaração de estilo;
- **Margin/Padding**: essa propriedade gerencia as margens e o espaçamento interno de elementos;
- **Positioning/Layout**: essas propriedades definem posicionamento e layout de elementos, além de visibilidade, tipo do ponteiro do mouse, alinhamento de objetos etc.;
- **Printing**: essas propriedades controlam as opções de impressão, como geração de quebra de página, por exemplo;
- **Table**: Essas propriedades tratam de tabelas, o espaçamento de suas bordas, seu posicionamento, entre outras opções;

- **Text:** essas propriedades gerenciam o layout do texto, definindo características como cor, tamanho, fonte, alinhamento, entre outras opções.

8.2.18. Objeto Table

Para adicionar uma tabela a um documento HTML, você deve criar a tag `<table>`, gerando um objeto **Table**.



As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Table**.

8.2.18.1. Coleções do objeto Table

A tabela a seguir mostra as coleções do objeto **Table**:

Coleção	Descrição
cells	Esta coleção mostra os elementos <code><td></code> ou <code><th></code> contidos em uma tabela.
rows	Esta coleção mostra os elementos <code><tr></code> contidos em uma tabela.
tBodies	Esta coleção mostra os elementos <code><tbody></code> contidos em uma tabela.

8.2.18.2. Propriedades do objeto Table

A tabela a seguir mostra as principais propriedades do objeto **Table**:

Propriedade	Descrição
caption	Este atributo refere-se ao subtítulo da tabela.
cellPadding	O espaço entre a borda da célula e seu conteúdo é definido ou mostrado por esta propriedade.

Propriedade	Descrição
cellSpacing	O espaço entre células em uma tabela é definido ou mostrado por esta propriedade.
frame	As bordas exteriores de uma tabela que devem ser mostradas são definidas ou têm seu valor retornado com essa propriedade.
rules	As bordas interiores de uma tabela que devem ser mostradas são definidas ou têm seu valor retornado com essa propriedade.
summary	Este atributo refere-se a uma descrição dos dados contidos na tabela. Você pode visualizar seu valor ou defini-lo.
tFoot	Este elemento é usado para agrupar o conteúdo do rodapé de uma tabela.
tHead	Este elemento é usado para agrupar o conteúdo do cabeçalho de uma tabela.

8.2.18.3. Métodos do objeto Table

A tabela a seguir mostra os principais métodos do objeto **Table**:

Método	Descrição
createCaption()	Este método adiciona à tabela um elemento de subtítulo vazio.
createTFoot()	Este método adiciona à tabela um elemento tfoot vazio.
createTHead()	Este método adiciona à tabela um elemento thead vazio.
deleteCaption()	Este método remove da tabela o primeiro elemento de subtítulo.
deleteRow()	Este método remove da tabela uma linha.
deleteTFoot()	Este método remove da tabela um elemento tfoot .
deleteTHead()	Este método adiciona à tabela um elemento thead .
insertRow()	Insere um elemento tr a uma tabela.

8.2.19. Objeto Textarea

Para criar uma área de texto em um formulário HTML basta usar a tag **<textarea>**, gerando o objeto **Textarea**.

Utilizando **getElementById()** ou pelo índice do array de elementos do formulário, é possível você acessar o objeto **Textarea**.

! As propriedades e métodos dos objetos **Node e **Element** também podem ser usados pelo objeto **Textarea**.**

8.2.19.1. Propriedades do objeto Textarea

A tabela a seguir mostra as principais propriedades do objeto **Textarea**: Com exceção dos atributos **form** e **type**, que apenas mostram seus valores, as propriedades têm a função de definir o valor de cada um dos atributos ou apenas retorná-lo.

Propriedade	Descrição
cols	É o comprimento de uma área de texto.
defaultValue	É o valor padrão contido em uma área de texto.
disabled	Por meio dessa propriedade você pode definir se a área de texto está desabilitada ou não, ou apenas visualizar o valor do atributo.
form	Indica o formulário que possui a área de texto.
name	Por meio dessa propriedade você pode definir o nome da área de texto, ou apenas visualizar o valor do atributo.
readOnly	Por meio dessa propriedade você pode definir se uma área de texto é somente leitura ou não, ou apenas visualizar o valor do atributo.
rows	Esta propriedade refere-se à altura de uma área de texto, mostrada em número de linhas.
type	Indica qual o tipo de elemento de formulário do qual a área de texto faz parte.

Propriedade	Descrição
value	É o conteúdo apresentado em uma área de texto.

8.2.19.2. Métodos do objeto Textarea

O conteúdo completo de uma área de texto pode ser selecionado pelo método **select()** do objeto **Textarea**.

Sua sintaxe é:

```
textareaObject.select()
```

8.3. DOM Core

O acesso e a manipulação de objetos encontrados em um documento HTML é feita por meio do DOM Core, uma interface de programação, ou API (Application Programming Interface), que realiza essa função por meio da definição de objetos e interfaces. É desse modo que desenvolvedores acessam e manipulam os conteúdos HTML e XML, utilizando sua estrutura hierárquica em forma de cadeia, ou árvore, que o DOM Core apresenta.

Essa cadeia é uma representação dos documentos em forma de objetos, ordenados hierarquicamente, e que se chamam nós, com interfaces próprias podendo ser implementadas. Além disso, alguns desses nós podem possuir tipos diferentes de nós-filhos ligados a eles, enquanto outros não têm essa possibilidade.

JavaScript

O DOM Core apresenta 12 tipos de nós, cada um deles com características diferentes e com a possibilidade ou não de possuírem nós-filhos. Esses nós representam aspectos de um documento HTML como documentos, elementos, atributos, entre outros. Veja na tabela a seguir cada um desses 12 tipos de nós:

Tipo de nó	Descrição	Filhos
Element	Denota um elemento.	<ul style="list-style-type: none">• Element• Text• Comment• ProcessingInstruction• CDATASEction• EntityReference
Attr	Denota um atributo.	<ul style="list-style-type: none">• Text• EntityReference
Text	Denota conteúdo de texto. Pode ser tanto referente a um elemento quanto a um atributo.	Nenhum.
CDATASEction	Denota em um documento o texto que não será analisado por um parser, chamado de seção CDATA.	Nenhum.
EntityReference	Denota uma referência de entidade.	<ul style="list-style-type: none">• Element• ProcessingInstruction• Comment• Text• CDATASEction• EntityReference
Entity	Denota uma entidade.	<ul style="list-style-type: none">• Element• ProcessingInstruction• Comment• Text• CDATASEction• EntityReference
	Denota uma instrução de processamento.	Nenhum.
Comment	Denota um comentário.	Nenhum.

Tipo de nó	Descrição	Filhos
Document	Denota um documento completo (este é o nó básico).	<ul style="list-style-type: none"> • Element • ProcessingInstruction • Comment • DocumentType
DocumentType	Uma interface é fornecida para as entidades escolhidas para o documento.	Nenhum.
DocumentFragment	Desempenha o papel de um objeto Document que contém apenas parte de um documento, tornando-o mais leve.	<ul style="list-style-type: none"> • Element • ProcessingInstruction • Comment • Text • CDATASection • EntityReference
Notation	Simboliza uma notação que está declarada no DTD.	Nenhum.

Veja a seguir os principais objetos do DOM Core.

8.3.1. Objeto Node

Um nó em um documento HTML é representado pelo objeto **Node**. Um nó pode consistir em um documento, um elemento, um atributo, texto ou um comentário.

8.3.1.1. Propriedades do objeto Node

A tabela a seguir mostra as principais propriedades do objeto **Node**:

Propriedade	Descrição
attributes	Esta propriedade mostra uma coleção dos atributos de um nó.
baseURI	O URI base absoluta de um nó é mostrada.
childNodes	Esta propriedade mostra os nós-filhos de outro nó em uma NodeList.
firstChild	O primeiro nó-filho de um nó é mostrado.
lastChild	O último nó-filho de um nó é mostrado.
localName	Esta propriedade mostra a parte local do nome de um nó.
namespaceURI	O URI do espaço de nomes (namespaceURI) de um nó é mostrado.
nextSibling	Esta propriedade mostra o nó seguinte no mesmo nível da cadeia de nós.
nodeName	O nome do nó, dependendo do seu tipo, é mostrado.
nodeType	Esta propriedade mostra o tipo do nó.
nodeValue	O valor de um nó é mostrado, dependendo do seu tipo, além de também poder ser definido por meio desta propriedade.
	O elemento raiz de um nó é mostrado como um objeto Document .
parentNode	Um nó tem seu nó pai mostrado por esta propriedade.
prefix	O prefixo do espaço de nomes de um nó é mostrado ou definido por esta propriedade.
	Esta propriedade mostra o nó anterior no mesmo nível da cadeia de nós.
textContent	O conteúdo de texto de um nó e de seus descendentes é definido ou mostrado por esta propriedade.

Veja a seguir um exemplo da utilização de algumas propriedades do objeto **Node**:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7  <body>
8
9      <div id="A"></div>
10
11 <script>
12 var div = document.getElementById("A");
13
14 console.log('attributes', div.attributes);
15 console.log('baseURI', div.baseURI);
16 console.log('childNodes', div.childNodes);
17 console.log('firstChild', div.firstChild);
18 console.log('lastChild', div.lastChild);
19 console.log('localName', div.localName);
20 console.log('namespaceURI', div.namespaceURI);
21 console.log('nextSibling', div.nextSibling);
22 console.log('nodeName', div.nodeName);
23 console.log('ownerDocument', div.ownerDocument);
24 console.log('parentNode', div.parentNode);
25 console.log('prefix', div.prefix);
26 console.log('previousSibling', div.previousSibling);
27 console.log('textContent', div.textContent);
28 </script>
29 </body>
30 </html>
```

Ano 2008.

8.3.1.2. Métodos do objeto Node

A tabela a seguir mostra os principais métodos do objeto **Node**:

Método	Descrição
appendChild()	Um novo nó-filho será adicionado ao nó especificado como se fosse o último filho.
cloneNode()	Este método faz um nó ser clonado.
	Dois nós têm suas posições comparadas.
hasAttributes()	Quando um nó possui algum atributo, este método retorna o valor true .
hasChildNodes()	Quando um nó possui nós-filhos, este método retorna true .
insertBefore()	Com este método, você define um nó-filho existente e um novo nó-filho é criado antes dele.
isDefaultNamespace()	Quando o URI do espaço de nomes (namespaceURI) especificado é o padrão, este método retorna o valor true .
isEqualNode()	Quando dois nós são iguais, este método é verificado.
isSameNode()	Quando dois nós são o mesmo, este método é verificado.
isSupported()	Quando um nó suporta o recurso especificado, este método retorna o valor true .
lookupNamespaceURI()	Mostra o URI do espaço de nomes referente a um prefixo que você especificar.
lookupPrefix()	Mostra o prefixo referente a um URI do espaço de nomes que você especificar.
normalize()	Os nós de texto vazios são removidos e os adjacentes são unidos com este método.
removeChild()	Um nó-filho é removido.
replaceChild()	Um nó filho é substituído.

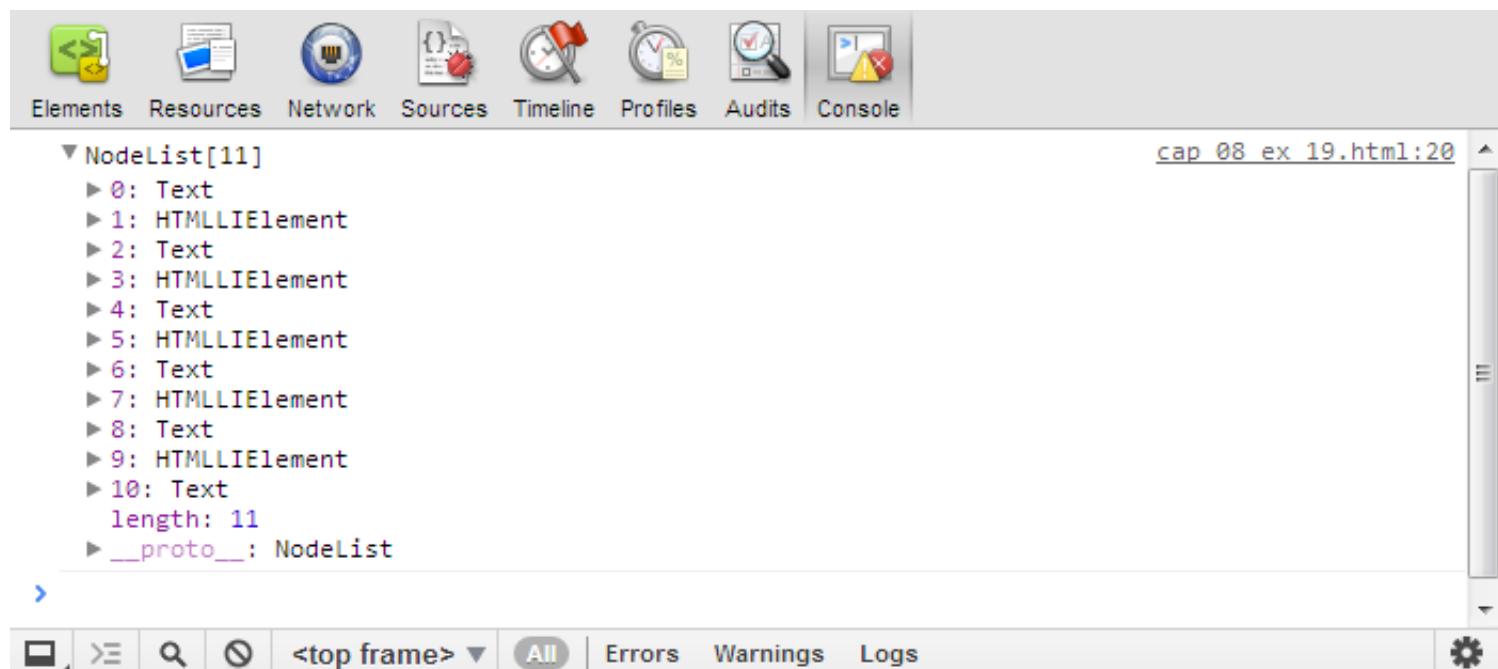
8.3.2. Objeto NodeList

É possível ter acesso a uma coleção de nós com uma ordem definida através do número de índice de um nó, que inicia-se em 0 (zero). Esta coleção é representada pelo objeto **NodeList**, que pode denotar, por exemplo, uma coleção de nós-filhos do objeto **Node**.

Veja a seguir um exemplo da utilização do objeto **NodeList**:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  </head>
7  <body>
8
9      <ul id="list">
10         <li>Item</li>
11         <li>Item</li>
12         <li>Item</li>
13         <li>Item</li>
14         <li>Item</li>
15     </ul>
16
17     <script>
18         var ul = document.getElementById("list");
19
20         console.dir(ul.childNodes);
21     </script>
22
23 </body>
</html>
```

Ao salvar e executar o código anterior, você visualizará a seguinte página:



8.3.2.1. Propriedade do objeto NodeList

O tamanho da coleção de nós, ou seja, o número de nós existentes nessa coleção, é mostrada pela propriedade **length**.

8.3.2.2. Método do objeto NodeList

Para visualizar um nó em um índice específico de uma lista de nós, utilize o método **item()**.

8.3.3. Objeto NamedNodeMap

Ao contrário do objeto **NodeList**, que representa uma coleção ordenada de nós, o objeto **NamedNodeMap** denota uma coleção de nós sem ordem específica, cujos nós são acessados por meio de seus nomes. Um objeto **NamedNodeMap** pode ser, por exemplo, uma coleção dos atributos do objeto **Node**.

8.3.3.1. Propriedades do objeto NamedNodeMap

O tamanho da coleção de nós, ou seja, o número de nós existentes nessa coleção, é mostrada pela propriedade **length**.

8.3.3.2. Métodos do objeto NamedNodeMap

A tabela a seguir mostra os principais métodos do objeto **NamedNodeMap**:

Método	Descrição
getNamedItem()	Este método mostra um nó específico, definido pelo seu nome.
getNamedItemNS()	Este método mostra um nó específico, definido pelo seu nome e seu espaço de nomes.
item()	Este método mostra o nó que encontra-se em um índice definido no objeto NamedNodeMap .
removeNamedItem()	Este método remove um nó específico, definido pelo seu nome.
removeNamedItemNS()	Este método remove um nó específico, definido pelo seu nome e espaço de nomes.
setNamedItem()	Este método estabelece um nó específico, definido pelo seu nome.
setNamedItemNS()	Este método estabelece um nó específico, definido pelo seu nome e seu espaço de nomes.

8.3.4. Objeto Document

É por meio do objeto **Document** que você consegue ter acesso a todos os dados de um documento. Este objeto representa a raiz de uma árvore de documento, ou seja, funciona como a base para todos os objetos **Node**, que possuem uma propriedade chamada **ownerDocument**, por onde ficam associados ao documento que os criou. O objeto **Document** possui métodos que criam objetos como nós de elemento, nós de texto, atributos, comentários, entre outros, visto que estes não podem existir fora de um documento.

As propriedades e métodos do objeto **Node** também podem ser usados pelo objeto **Document**.

8.3.4.1. Propriedades do objeto Document

A tabela a seguir mostra as principais propriedades do objeto **Document**:

Propriedade	Descrição
doctype	A Document Type Declaration associada ao documento é mostrada.
documentElement	O Document Element de um documento, ou seja, o documento HTML, é mostrado.
documentURI	Esta propriedade refere-se à localização do documento. É possível definir seu valor ou apenas mostrá-lo.
domConfig	A configuração encontrada quando você chama normalizeDocument() é mostrada.
implementation	O objeto DOMImplementation, que manipula objetos, é mostrado.
inputEncoding	Esta propriedade mostra a codificação do conjunto de caracteres que é utilizada no documento HTML.

Propriedade	Descrição
strictErrorChecking	Define a imposição da checagem de erros, se ela é realizada ou não, ou apenas mostra o valor adotado.
xmlEncoding	A codificação XML do documento XML é mostrada.
xmlStandalone	Esta propriedade mostra se o documento XML é autônomo ou não, ou permite a definição deste valor.
xmlVersion	A versão XML do documento XML é mostrada.

8.3.4.2. Métodos do objeto Document

A tabela a seguir mostra os principais métodos do objeto **Document**:

Método	Descrição
adoptNode(nó)	Este método retorna um nó que é adotado de outro documento.
createAttribute()	Este método gera um nó de atributo.
createAttributeNS(URI,nome)	Este método gera um atributo com nome e URI de espaço de nomes (namespaceURI) definidos.
createCDATASection()	Este método funciona apenas com DOM XML. Com ele, um nó CDATA é criado com um texto específico.
createComment()	Um nó de comentário é criado, e você definir um texto específico.
createDocumentFragment()	Um nó DocumentFragment vazio é criado com este método.
createElement()	Um nó de elemento é criado com este método.
createElementNS()	Um elemento com o espaço de nomes (namespace) definido é criado com este método.

Método	Descrição
createEntityReference()	Este método funciona apenas com DOM XML. Com ele, um nó EntityReference é criado.
createTextNode()	Este método gera um nó de texto.
getElementById()	O elemento que contém o atributo ID como o valor específico é mostrado por este método.
getElementsByTagName()	Todos os elementos, com seus respectivos nomes de tag, são mostrados em uma NodeList.
getElementsByTagNameNS()	Todos os elementos, com seus respectivos nomes de tag e URI do espaço de nomes (namespaceURI), são mostrados em uma NodeList.
importNode()	Este método faz com que um nó seja importado de outro documento.
normalizeDocument()	Nós de texto que estejam vazios são removidos e unidos a nós adjacentes quando este método é usado.
renameNode()	Define um nó para ser renomeado.

8.3.5. Objeto Element

O objeto **Element** pode possuir atributos, que fazem parte do tipo de nó **Attr**, e pode também ter nós-filhos do tipo **Element**, **Text**, **Comment**, **CDATASection**, **ProcessingInstruction** e **EntityReference**, ou seja, este objeto representa um elemento encontrado em um documento HTML.

As propriedades e métodos do objeto **Node** também podem ser usados pelo objeto **Element**.

8.3.5.1. Propriedades do objeto Element

A tabela a seguir mostra as principais propriedades do objeto Element:

Propriedade	Descrição
schemaTypeInfo	A informação sobre o tipo do elemento é mostrada.
tagName	O nome da tag do elemento é mostrada.

8.3.5.2. Métodos do objeto Element

A tabela a seguir mostra os principais métodos do objeto Element:

Método	Descrição
getAttribute()	Este método mostra o valor específico do atributo.
getAttributeNS()	Este método funciona apenas com XML DOM, e mostra o valor específico do atributo juntamente com seu espaço de nomes (namespace).
getAttributeNode()	Este método mostra o nó específico do atributo.
getAttributeNodeNS()	Este método funciona apenas com XML DOM, e mostra o nó específico do atributo juntamente com seu espaço de nomes (namespace).
getElementsByTagName()	Este método mostra todos os elementos filhos em uma coleção, assim como seus nomes de tag específicos.
getElementsByTagNameNS()	Este método funciona apenas com DOM XML, e mostra todos os elementos filhos em uma coleção, assim como seus nomes de tag específicos e espaço de nomes (namespace).

Método	Descrição
hasAttribute()	Este método mostra se o elemento possui o atributo especificado, retornando o valor true , e caso contrário, false .
hasAttributeNS()	Este método mostra se o elemento possui o atributo especificado por nome e espaço de nomes (namespace), retornando o valor true , e caso contrário, false .
removeAttribute()	Este método remove o atributo especificado.
removeAttributeNS()	Este método funciona apenas com DOM XML, e remove o atributo especificado por nome e espaço de nome (namespace).
removeAttributeNode()	Este método remove e mostra um nó de atributo especificado.
setAttribute()	Este método permite a alteração ou definição de um atributo específico com um valor específico.
setAttributeNS()	Este método funciona apenas com DOM XML, e remove o atributo especificado com seu espaço de nomes (namespace) também especificado.
setAttributeNode()	Este método permite a alteração ou definição de um nó de atributo específico.
setAttributeNodeNS()	Este método funciona apenas com DOM XML, e permite a alteração ou definição de um nó de atributo específico com um espaço de nomes (namespace) específico.

8.3.6. Objeto Attr

Um atributo encontrado em uma página HTML sempre pertence a um objeto **Element**, e é representado pelo objeto **Attr**.



As propriedades e métodos dos objetos **Node** e **Element** também podem ser usados pelo objeto **Attr**.

8.3.6.1. Propriedades do objeto Attr

A tabela a seguir mostra as principais propriedades do objeto Attr:

Propriedade	Descrição
isId	Caso o atributo for do tipo ID, esta propriedade retorna true, e retorna false para outros tipos de atributo.
name	Esta propriedade mostra o nome do atributo.
ownerElement	O elemento ao qual o atributo pertence é mostrado por esta propriedade.
specified	Caso o atributo tenha sido especificado, esta propriedade retorna true, e retorna false para casos contrários.
value	O valor do atributo é definido ou mostrado por esta propriedade.

8.4. DOM Storage

O armazenamento de dados permanentes de lado cliente é feito em JavaScript por um sistema chamado Armazenamento Web, ou DOM Storage. Apesar da semelhança desse modelo com o sistema de cookies, existem algumas diferenças importantes entre eles.

8.4.1. Local Storage

Esse método de armazenamento destaca-se por permitir que dados sejam armazenados por mais de uma única sessão. Sendo assim, sempre que algum dado é armazenado desse modo, o objeto global **localStorage** permite o acesso a essas informações. A principal diferença entre os objetos de armazenamento **localStorage** e **sessionStorage** é que, no primeiro, os dados permanecem guardados após o fim da duração de uma janela ou guia, ao contrário do objeto **sessionStorage**. Com exceção desse caso, os dois objetos têm praticamente a mesma funcionalidade.

A área de armazenamento local (**localStorage**) é apropriada para utilização em cenários de transação múltiplas, exatamente pelo fato de ser compartilhada por mais de uma janela ou guia.

8.4.2. A interface Storage

Para realizar suas ações, como armazenar, recuperar e apagar itens, a interface Storage faz uso de certos métodos ou propriedades, conforme você verá a seguir. A interface **Storage** é implementada pelos objetos **localStorage** e também **sessionStorage**.

- **length**: Mostra o tamanho, ou seja, o número de pares de valores e chaves na lista;
- **key(indice)**: Mostra a chave contida em um índice definido;
- **getItem(chave)**: Mostra o item relativo a uma chave específica;
- **setItem(chave, valor)**: Define um item em um par de valor e chave;
- **removeItem(chave)**: O item relativo a uma chave definida é removido;
- **clear()**: Todos os pares de valor e chave são apagados.

Veja o exemplo a seguir:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 </head>
7 <body>
8
9     <h1 id="titulo_h1" contenteditable="true">Edite...</h1>
10    <p id="titulo_p" contenteditable="true">Edite este parágrafo...</p>
11
12 <script>
13 var editaveis = document.querySelectorAll('[contenteditable="true"]');
14
15 for(var i = 0; i < editaveis.length, elemento = editaveis[i]; i++){
16
17     elemento.addEventListener("blur", function(){
18
19         console.log(this);
20
21         localStorage.setItem(this.id, this.innerHTML);
22
23     }, true);
24
25     //Verificando se já tem dados no localStorage
26     if(localStorage.getItem(elemento.id)){
27         elemento.innerHTML = localStorage.getItem(elemento.id);
28     }
29 }
30
31 </script>
32 </body>
33 </html>
```

8.4.3. Eventos DOM Storage

O evento **DOM Storage** é ativado sempre que houver uma alteração da área de armazenamento. Veja a seguir os atributos relativos ao objeto do evento de armazenamento:

- **key**: é a chave que sofre a alteração;
- **oldValue**: é o valor antigo da chave que sofre alteração;
- **newValue**: é o valor novo da chave que sofre alteração;
- **url**: é o endereço URL da página onde encontra-se a chave que sofre alteração;
- **storageArea**: é o objeto Storage que sofreu alteração.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O DOM (Document Object Model) define a maneira padrão de se acessar e manipular documentos HTML. O DOM foi lançado em 1998 pela W3C (World Wide Web Consortium) com a especificação Nível 1, permitindo o acesso e a manipulação de cada elemento em uma página HTML;
- A estrutura de um documento HTML pode ser representada pelo DOM HTML. O DOM HTML pode ser retratado por meio de um diagrama em formato de árvore, do mesmo modo que uma árvore genealógica de uma família. Nesse diagrama é possível compreender os graus de “parentesco” entre elementos, além das ascendências e descendências;
- O DOM HTML é formado por diversos objetos, cada qual com suas respectivas propriedades e métodos, que permitem o acesso e a manipulação de inúmeras características de um documento HTML, como, por exemplo, a inserção de imagens mapeadas, formulários, tabelas, botões, etc.;
- O DOM Core permite o acesso e a manipulação de objetos encontrados em um documento HTML por meio de uma interface de programação, ou API, que realiza essa função definindo objetos e interfaces. O DOM Core apresenta 12 tipos de nós, cada um deles com características diferentes e com a possibilidade ou não de possuírem nós-filhos. Esses nós representam aspectos de um documento HTML como documentos, elementos, atributos, entre outros;
- O armazenamento de dados permanentes de lado cliente é feito em JavaScript por um sistema chamado Armazenamento Web, ou DOM Storage. Esse método de armazenamento destaca-se por permitir que dados sejam armazenados por mais de uma única sessão. Sendo assim, sempre que algum dado é armazenado desse modo, o objeto global **localStorage** permite o acesso a essas informações.

8

DOM Teste seus conhecimentos

Ana G. Cesar
778.6008-0600



IMPACTA
EDITORA

1. O objeto Document é parte de qual objeto?

- a) Window
- b) Current
- c) DOM
- d) Node
- e) Nenhuma das alternativas anteriores está correta.

2. Qual dos métodos não faz parte do objeto Document?

- a) close()
- b) open()
- c) write()
- d) writeln()
- e) console()

3. A propriedade dir do objeto HTMLElement é responsável por:

- a) Informa a direção do texto em um elemento ou permite que você a defina.
- b) Informa o atributo de classe de um elemento ou permite que você o defina.
- c) Mostra o atributo de estilo de um elemento ou permite que você o defina.
- d) Mostra a ordem de guias em um elemento ou permite que você a defina.
- e) Informa a chave de acesso para um elemento ou permite que você a defina.

4. Qual dos métodos não tem relação com a interface Storage?

- a) getKey()
- b) key()
- c) getItem()
- d) clear()
- e) removeItem()

5. A método getElementById() é responsável por:

- a) Mostrar todos os elementos, com seus respectivos nomes de tag.
- b) Mostrar o elemento que contém o atributo ID com o valor específico.
- c) Fazer com que um nó seja importado de outro documento.
- d) Mostrar todos os elementos, com seus respectivos nomes de tag e URI do espaço de nomes (namespaceURI).
- e) Nenhuma das alternativas anteriores está correta.

8

DOM Mãos à obra!

Ana G. Caezar
778.6000.
778.6000.



IMPACTA
EDITORA

Laboratório 1

A - Manipulando o DOM: Tirando um printScreen do document.

1. Crie um novo arquivo de texto e renomeie com a extensão **.html**;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD** e **BODY**;
4. Inclua dentro da tag **BODY** o seguinte HTML:

```
<html>
<head>
</head>
<body>
    <div id="images">
        
        
        
    </div>
    <button>Tirar PrintScreen da tela</button>
    <div id="tela"></div>
</body>
</html>
```

5. Inclua dentro da tag **HEAD** a tag **<script></script>**;

6. Dentro da tag **<script>**, faça o seguinte:

- Crie um evento **onClick()** no botão “Tirar PrintScreen da tela” e chame uma função **fnPrintScreen()**;
- Nesta função será necessário utilizar o objeto **XMLSerializer**;
- Utilizar o método do objeto **XMLSerializer** chamado **serializeToString()** que irá receber um parâmetro. O parâmetro deve ser o objeto **Document**.
- Deve-se utilizar também as funções **escape** e **unescape**;

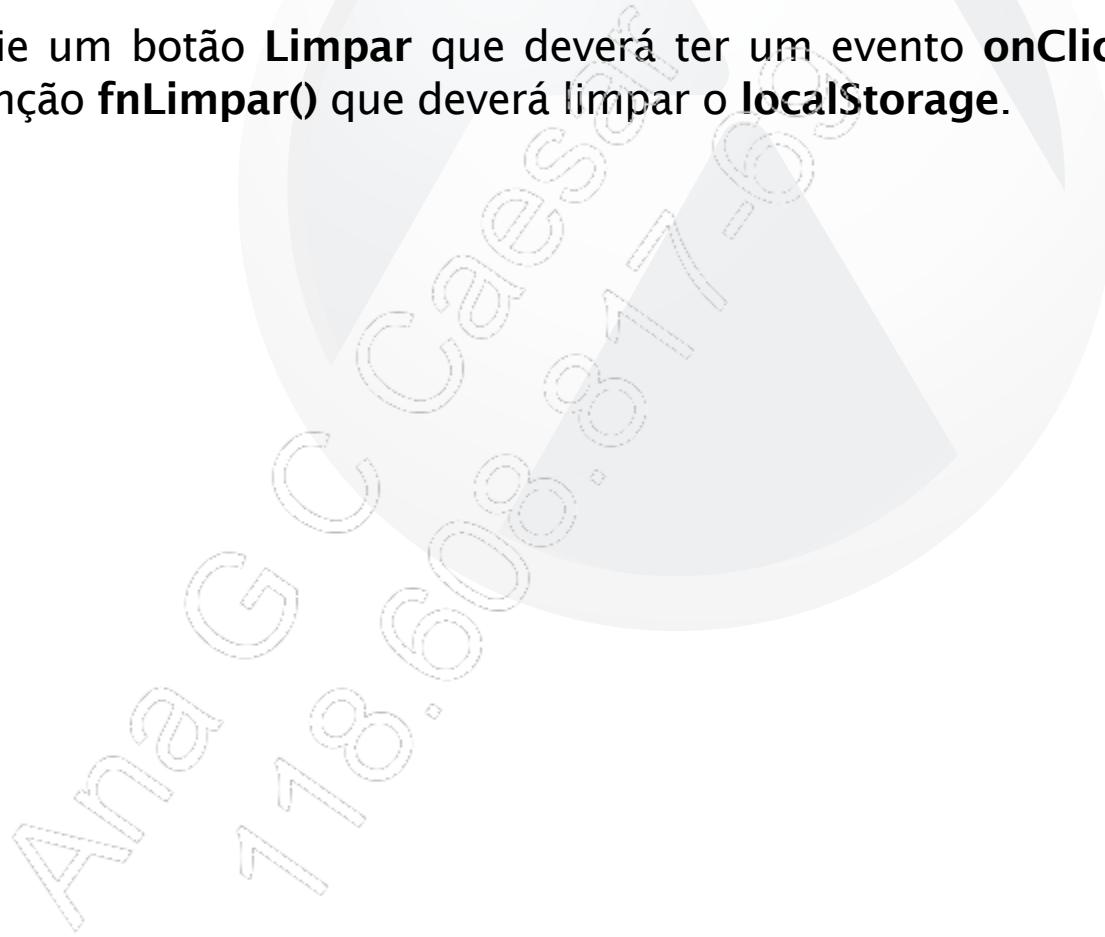
7. O resultado final é mostrar uma cópia do **Document** dentro da tag **DIV** na tela.

Laboratório 2

A - Utilizando LocalStorage.

Neste laboratório, utilize o Laboratório 1 do Capítulo 7 e incremente as seguintes condições:

- Todas as palavras inseridas para o sorteio deverão ser salvas no **localStorage** para, em caso de atualização da tela, não perder as informações;
- Crie um botão **Limpar** que deverá ter um evento **onClick()** chamando a função **fnLimpar()** que deverá limpar o **localStorage**.



AJAX

9

- ✓ O objeto XMLHttpRequest;
- ✓ Exemplos de uso.



IMPACTA
EDITORA

9.1. Introdução

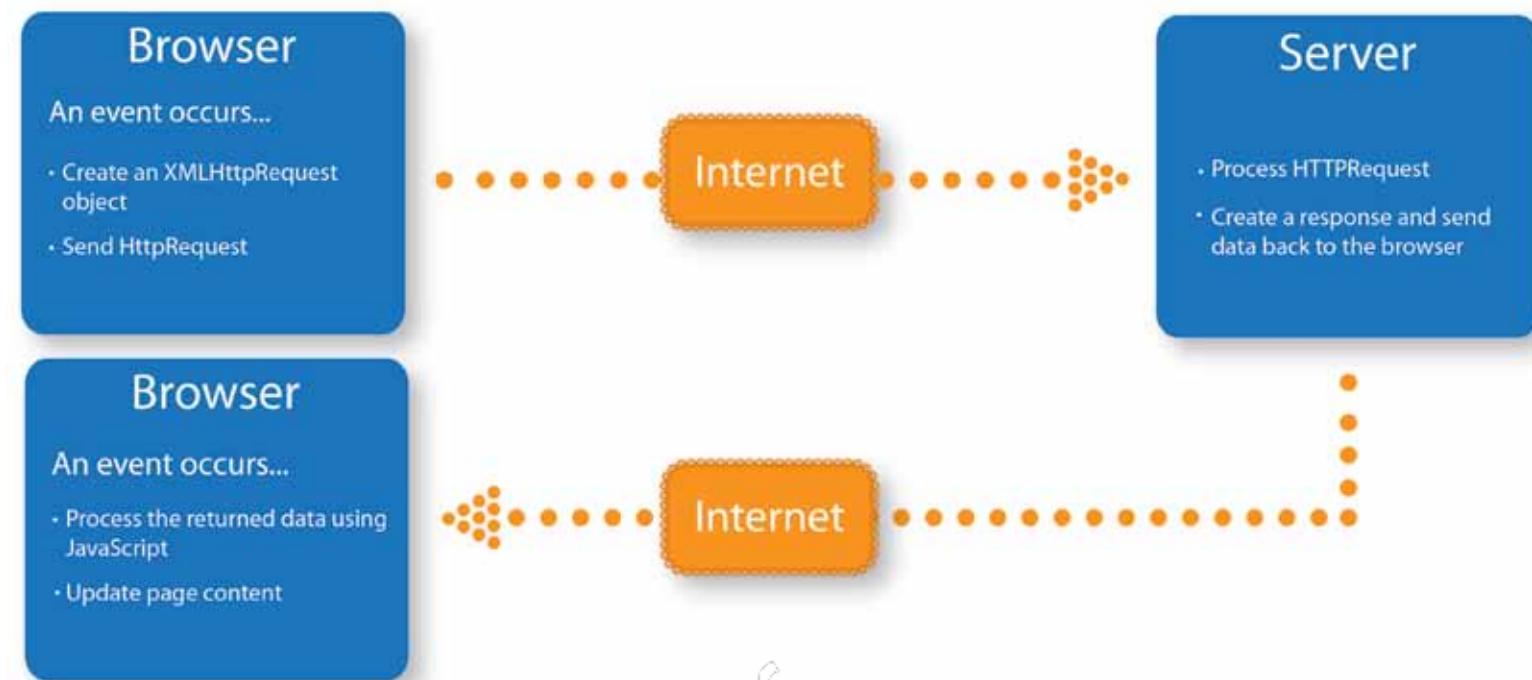
Antes do AJAX, como a maioria das páginas Web eram constituídas só por páginas HTML, elas precisavam ser recarregadas a partir do servidor a cada ação do usuário. Além de ineficaz, o fato de o conteúdo da página desaparecer e reaparecer era evidente e atrapalhava o usuário. Imagine que a cada alteração em uma página, todo o seu conteúdo era reenviado para que a página pudesse ser carregada. Seria muito mais fácil se só o conteúdo alterado pudesse ser reenviado, evitando, inclusive, sobrecarga no servidor e consumo de banda.

AJAX (acrônimo de Asynchronous JavaScript e XML) é uma forma diferenciada de utilizar os padrões existentes da Web. Unindo a funcionalidade de tecnologias como JavaScript e XML, permite criar páginas dinâmicas e rápidas através de mecanismos que estabelecem a troca de informações com o servidor sem a necessidade de atualizar e recarregar a página Web inteira, apenas partes dela.

AJAX possibilita um número reduzido de dados trafegados na rede, e o mais importante: durante as requisições, não há necessidade de aguardar a página Web ser recarregada a cada interação com o servidor. Essa funcionalidade ajuda a resolver um problema antes evidente para o usuário: a espera pelo envio e retorno da página por completo.

Apesar de seu nome sugerir, AJAX não exige a utilização de XML – JSON e TXT são frequentemente utilizados. Também não há necessidade de as solicitações serem assíncronas, embora assim sejam, na maioria das vezes. Google Maps, Gmail, YouTube são exemplos de aplicações cujas guias utilizam AJAX.

Para ter uma ideia melhor do funcionamento do AJAX, veja a seguir o esquema nesse simples diagrama:



9.2. O objeto XMLHttpRequest

Esta API, hoje disponível nas linguagens para script dos navegadores Web, foi utilizada primeiro pelo IE (Internet Explorer), desde sua versão 4.0. Antes do nome AJAX ser comumente usado, ela foi chamada também de XMLHTTP e só foi disseminada com a utilização pelo Google em 2005, no Gmail e no Google Maps.

XMLHttpRequest tem a finalidade de enviar as solicitações HTTP ou HTTPS diretamente a um servidor Web e depois carregar direto no script os dados obtidos como resposta. Tais dados devem, obrigatoriamente, ser recebidos em texto simples, JSON, XML ou HTML e podem ser utilizados para modificar o DOM do documento utilizado na janela do navegador, sem que seja necessário carregar um novo documento. Além disso, os dados obtidos com resposta podem ser avaliados pelo script no lado do cliente.

9.2.1. Criando o objeto

Os navegadores mais modernos possuem um objeto **XMLHttpRequest** nativo. Mas é bom verificar se o navegador que você está utilizando oferece suporte a este objeto.

Quando for necessário criá-lo, utilize a seguinte sintaxe:

```
variavel=new XMLHttpRequest();
```

No caso específico das versões mais antigas do IE (5 e 6), ou quando o navegador não oferecer suporte para o objeto **XMLHttpRequest**, há a necessidade de utilizar um objeto **ActiveXObject**:

```
variavel=new ActiveXObject("Microsoft.XMLHTTP");
```

Antes de criá-los, utilize o seguinte código, que verifica se o objeto que você deseja utilizar no script é reconhecido pelo navegador. Assim, ele utiliza ou **XMLHttpRequest**, ou **ActiveXObject**, dependendo do navegador.

```
var xmlhttp;
if (window.XMLHttpRequest)
    xmlhttp=new XMLHttpRequest();
else
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

9.2.2. Enviando uma requisição para o servidor

Após a criação da instância do objeto **XMLHttpRequest**, é necessário estabelecer comunicação com o servidor, podendo, posteriormente, requisitar um arquivo e/ou enviar dados.

Para isso, você pode utilizar alguns métodos, dentre os quais **open**, **send**, **setRequestHeader** e **onreadystatechange** são os mais importantes. Veja, nos próximos subtópicos, como eles funcionam.

9.2.2.1. Método open

Este método tem como finalidade passar ao servidor os dados sobre o endereço do arquivo que foi requisitado pelo navegador. Isto é, quando o navegador requisitar um endereço de arquivo ao servidor, é o método **open** que é responsável por enviar os dados do endereço ao servidor.

A sintaxe geral do método **open** é a seguinte:

```
xmlhttp.open(metodo, URL, assincronia, usuario, senha);
```

Em que:

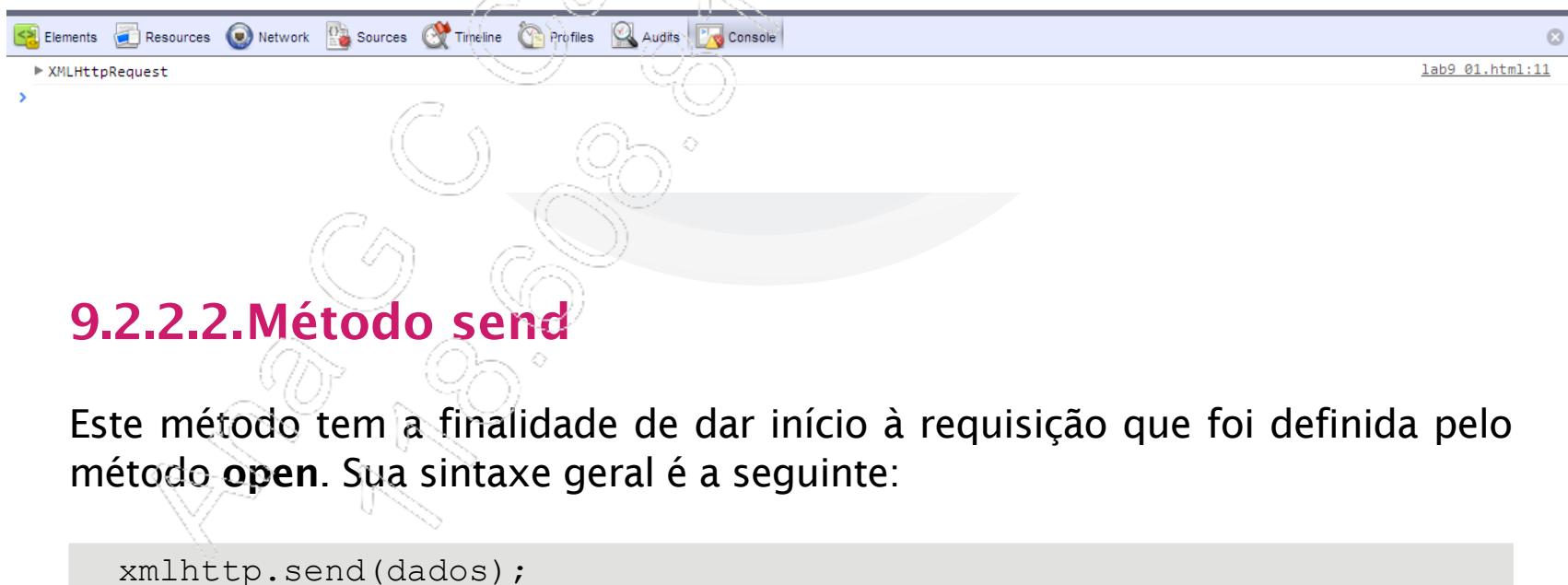
- **metodo**: Especifica qual o método responsável pelo envio de dados para o servidor. Os mais usados são os métodos GET e POST:
 - **GET**: Método que pode ser utilizado na maioria das vezes e é considerado mais simples e mais rápido que o método POST;
 - **POST**: Método aplicado para o envio de grande quantidade de dados ao servidor, já que este método não tem limitação de tamanho, bem como para o envio de entradas do usuário, oferecendo maior segurança e robustez. É utilizado, também, quando não é possível utilizar um arquivo em cache, de modo a ser necessário atualizar o arquivo ou o banco de dados no servidor.
- **URL**: Especifica o endereço do arquivo, independente de ser relativo ou absoluto, que será requisitado ao servidor;
- **assincronia**: Especifica, por meio de valores booleanos, se a requisição será assíncrona (valor **true**) ou síncrona (valor **false**). A utilização deste parâmetro é opcional e, no caso de sua omissão, a requisição será assíncrona;
- **usuario** e **senha**: Especifica nome de usuário e senha, a serem utilizados quando um arquivo requisitado for protegido e exigir que se forneça esses dados para acesso. Estes parâmetros são opcionais.

JavaScript

Veja a seguir um exemplo da utilização do método **open**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7     var xmlhttp;
8     if (window.XMLHttpRequest)
9         xmlhttp=new XMLHttpRequest(); //Navegadores atuais
10    else
11        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6
12
13    xmlhttp.open("GET","dados.txt");
14    console.log(xmlhttp);
15 </script>
16 </head>
17 <body>
18 </body>
19 </html>
```

A imagem a seguir mostra a execução do código anterior:



9.2.2.2. Método send

Este método tem a finalidade de dar início à requisição que foi definida pelo método **open**. Sua sintaxe geral é a seguinte:

```
xmlhttp.send(dados);
```

Em que o parâmetro **dados** especifica quais os dados que devem ser enviados ao servidor, independente do formato em que foram codificados, podendo ser uma string de consulta, string do DOM ou outro tipo.

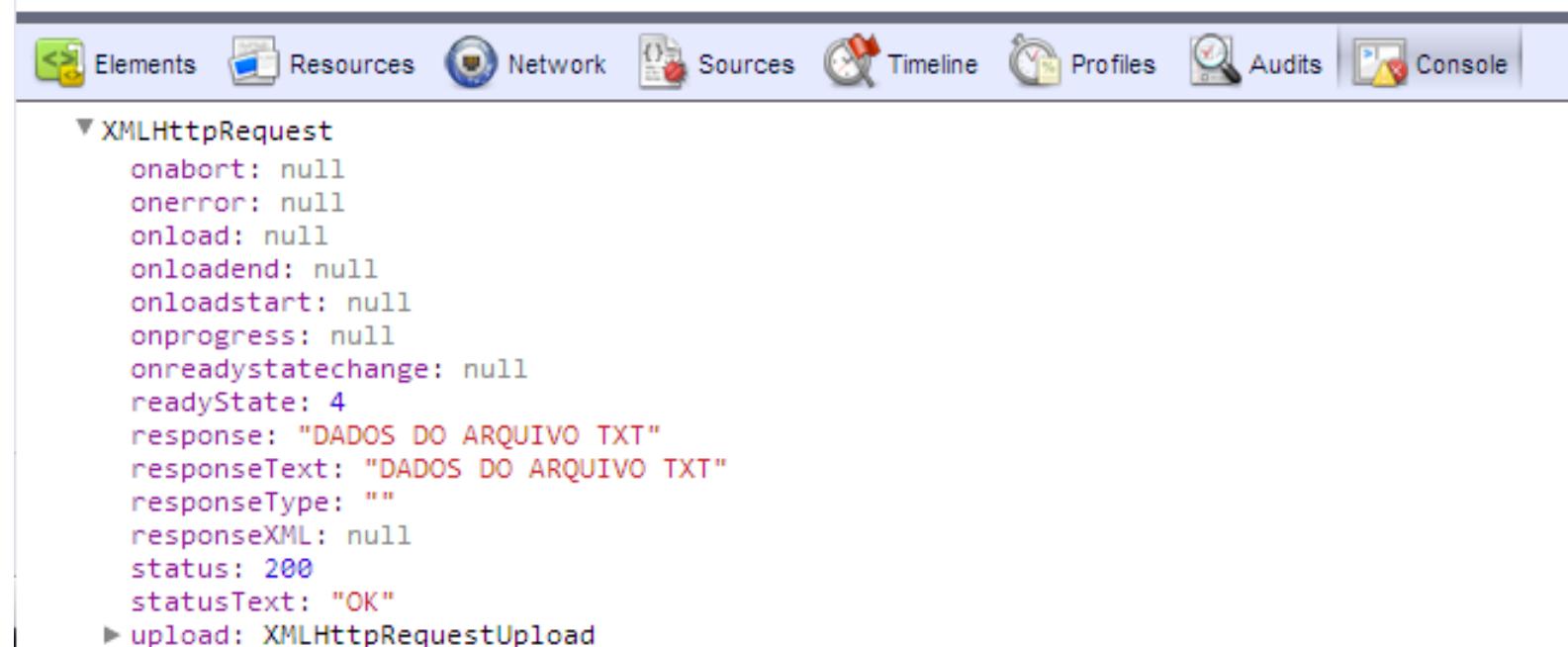
Quando não há dados a serem enviados para o servidor, o parâmetro **null** deve ser utilizado:

```
xmlhttp.send(null);
```

Veja a seguir um exemplo da utilização do método **send**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7     var xmlhttp;
8     if (window.XMLHttpRequest)
9         xmlhttp=new XMLHttpRequest(); //Navegadores atuais
10    else
11        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6
12
13    xmlhttp.open("GET", "dados.txt");
14    xmlhttp.send();
15    console.log(xmlhttp);
16 </script>
17 </head>
18 <body>
19 </body>
20 </html>
```

A imagem a seguir mostra a execução do código anterior:



9.2.2.3. Método setRequestHeader

Após o devido início da solicitação, este método pode ser chamado a fim de enviar, junto com a solicitação, os cabeçalhos HTTP. Tais cabeçalhos são metadados que descrevem a própria solicitação e são enviados pelo navegador, sempre que este requisita uma página ao servidor.

Para uma requisição AJAX com a utilização do método **setRequestHeader**, ele deve ser declarado logo após o método **open** e, com isso, os metadados podem ser sobrescritos e personalizados:

```
xmlhttp.setRequestHeader(nome_cabecalho, valor)
```

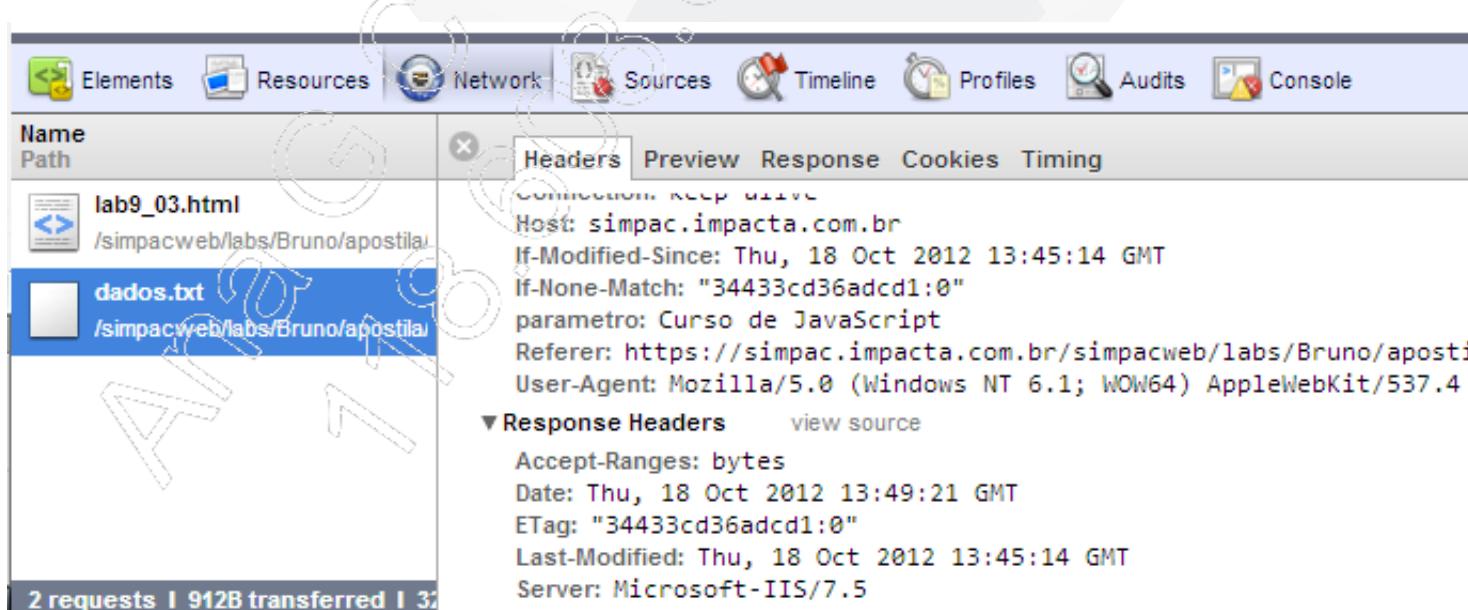
Em que:

- **nome_cabecalho**: Especifica o nome da string de texto do cabeçalho;
- **valor**: Especifica o valor da string.

Veja a seguir um exemplo da utilização do método **setRequestHeader**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7     var xmlhttp;
8     if (window.XMLHttpRequest)
9         xmlhttp=new XMLHttpRequest(); //Navegadores atuais
10    else
11        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6
12
13    xmlhttp.open("GET", "dados.txt");
14    xmlhttp.setRequestHeader('parametro', 'Curso de JavaScript');
15
16    xmlhttp.send();
17    console.log(xmlhttp);
18 </script>
19 </head>
20 <body>
21 </body>
22 </html>
```

A imagem a seguir mostra a execução do código anterior:



O método **setRequestHeader** deve ser chamado para cada cabeçalho a ser enviado junto com a solicitação do navegador. Quando o método **open** for chamado novamente para um usuário compatível com W3C, os cabeçalhos anexados serão excluídos.

9.2.2.4. Ação disparadora de evento `onreadystatechange`

É chamada de ação disparadora de evento qualquer ação praticada pelo usuário em relação a um elemento em uma interface gráfica. Em outros termos, a interação do usuário com a interface, como uma página Web, dispara o evento.

Mas não é somente a ação do usuário que é capaz de disparar um evento, ele pode ser disparado sem intervenção direta. A ação `onreadystatechange` é um exemplo: ela é disparada pelo servidor, ocorrendo toda vez que ele enviar para o cliente a informação de uma alteração ou atualização no status de sua comunicação com o cliente.

Para o objeto `XMLHttpRequest`, é a propriedade `readyState` que define o status da comunicação. Assim, quando houver mudança nesta propriedade, a ação `onreadystatechange` será disparada.

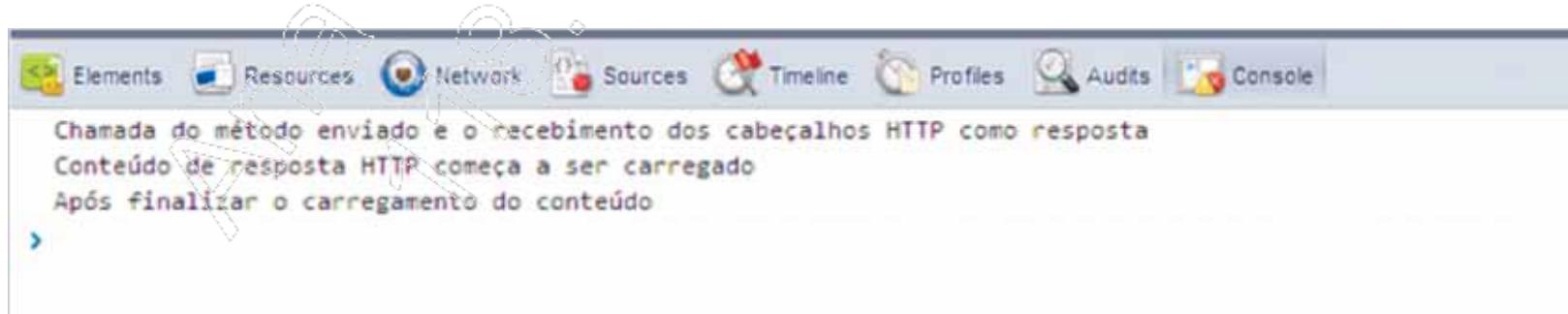
A propriedade `readyState` indica o status da comunicação por meio de atributos que retornam um valor numérico, da seguinte forma:

- Caso a requisição não tenha sido iniciada, o valor **0** é atribuído para a propriedade `readyState`;
- Após a chamada, com sucesso, de um método aberto, o valor **1** é atribuído à propriedade;
- Após a chamada do método enviado e o recebimento dos cabeçalhos HTTP como resposta, o valor **2** é atribuído;
- Quando o conteúdo de resposta HTTP começa a ser carregado, o valor **3** é atribuído;
- Após finalizar o carregamento do conteúdo, o valor **4** é atribuído à propriedade `readyState`.

Veja a seguir um exemplo da utilização da ação disparadora de evento **onreadystatechange**:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7     var xmlhttp;
8     if (window.XMLHttpRequest)
9         xmlhttp=new XMLHttpRequest(); //Navegadores atuais
10    else
11        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6
12
13 xmlhttp.open("GET","dados.txt");
14 xmlhttp.onreadystatechange = function(){
15     switch(xmlhttp.readyState){
16         case 0:
17             console.log('Propriedade ainda não foi iniciada');
18             break;
19         case 1:
20             console.log('Chamada de sucesso após objeto ser aberto');
21             break;
22         case 2:
23             console.log('Chamada do método enviado e o recebimento dos cabeçalhos HTTP como resposta');
24             break;
25         case 3:
26             console.log('Conteúdo de resposta HTTP começa a ser carregado');
27             break;
28         case 4:
29             console.log('Após finalizar o carregamento do conteúdo');
30             break;
31     }
32 }
33 xmlhttp.send();
34 </script>
35 </head>
36 <body>
37 </body>
38 </html>
```

A imagem a seguir mostra a execução do código anterior:



9.2.3. Respostas do servidor

Após enviar a requisição ao servidor, ele deve recebê-la e tratar o conteúdo recebido. Para obter a resposta do servidor, as propriedades **responseText** e **responseXML** são utilizadas.

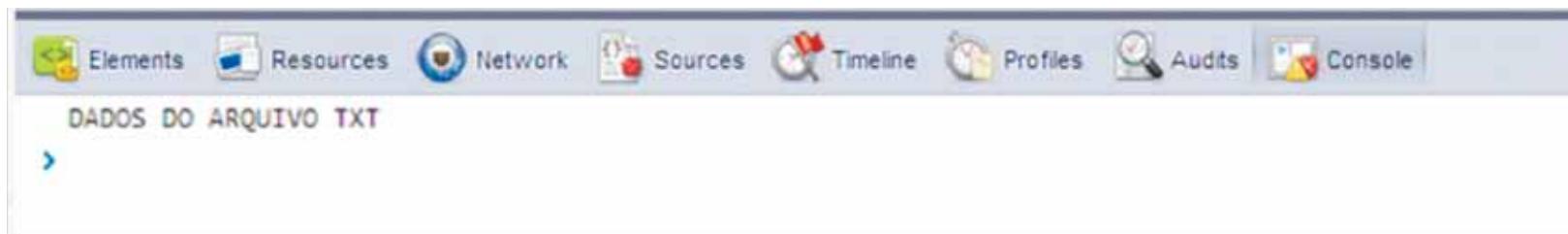
9.2.3.1. Propriedade **responseText**

Quando a resposta do servidor não é XML, utilize a propriedade **responseText**, cujo retorno é uma string.

Veja a seguir um exemplo da utilização da propriedade **responseText**:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7      var xmlhttp;
8      if (window.XMLHttpRequest)
9          xmlhttp=new XMLHttpRequest(); //Navegadores atuais
10     else
11         xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6
12
13     xmlhttp.open("GET","dados.txt");
14     xmlhttp.onreadystatechange = function() {
15         if(xmlhttp.readyState==4) console.log(xmlhttp.responseText);
16     }
17     xmlhttp.send();
18 </script>
19 </head>
20 <body>
21 </body>
22 </html>
```

A imagem a seguir mostra a execução do código anterior:



Após a propriedade **readyState** ter o valor 4 atribuído, isto é, somente após o carregamento total do conteúdo, é que a propriedade **responseText** estará disponível para manipulação no script.

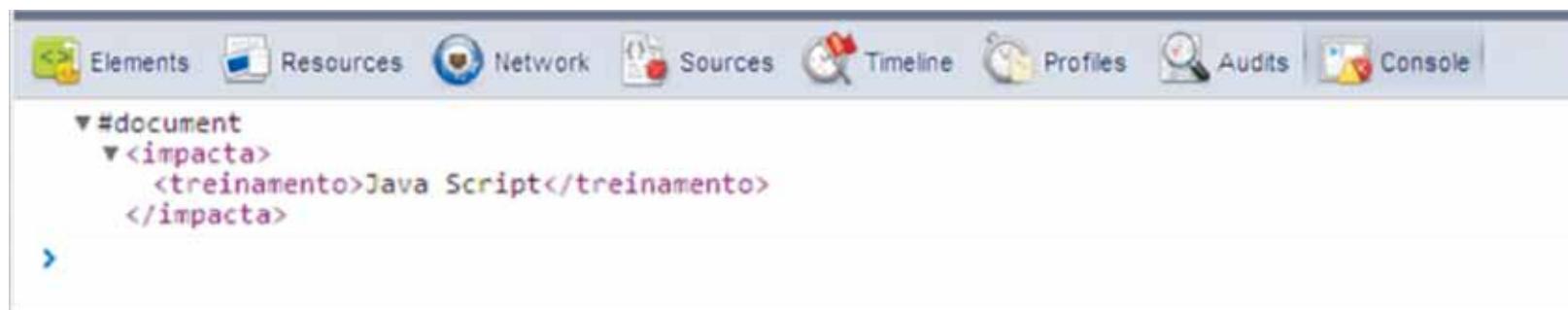
9.2.3.2. Propriedade **responseXML**

Quando a resposta do servidor é XML e um objeto XML deve ser interpretado, utilize a propriedade **responseXML**.

Veja a seguir um exemplo da utilização da propriedade **responseXML**:

```
1  <!doctype html>
2  <html lang="pt-br">
3  <head>
4  <meta charset="utf-8">
5  <title>JavaScript</title>
6  <script>
7      var xmlhttp;
8      if (window.XMLHttpRequest)
9          xmlhttp=new XMLHttpRequest(); //Navegadores atuais
10     else
11         xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6
12
13     xmlhttp.open("GET","dados.xml");
14     xmlhttp.onreadystatechange = function(){
15         if(xmlhttp.readyState==4) console.log(xmlhttp.responseXML);
16     }
17     xmlhttp.send();
18 </script>
19 </head>
20 <body>
21 </body>
22 </html>
```

A imagem a seguir mostra a execução do código anterior:



The screenshot shows the Network tab of the Chrome DevTools. The status bar at the bottom indicates "2 requests" and "1 loaded". One request is listed with the URL "http://localhost:8080/testeXML.php" and the method "GET". The response is labeled "text/xml" and has a size of "1.8 kB". The response body contains the XML code:

```
<#document>
<impacta>
  <treinamento>Java Script</treinamento>
</impacta>
```

O cabeçalho HTTP que define o tipo de MIME deve possuir o valor **text/xml** ou **application/xml** para que o servidor envie a resposta no formato XML.

9.3. Exemplos de uso

Veja agora, na prática, alguns exemplos que mostram requisições de AJAX em diferentes tipos de arquivo.

9.3.1. Solicitando HTML

Para solicitar um arquivo HTML usando AJAX, você deve usar o seguinte código:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7     var xmlhttp;
8     if (window.XMLHttpRequest)
9         xmlhttp=new XMLHttpRequest(); //Navegadores atuais
10    else
11        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6
12
13    xmlhttp.open("GET","exemplo.html");
14    xmlhttp.onreadystatechange = function(){
15        if(xmlhttp.readyState==4) document.getElementById('dados').innerHTML = xmlhttp.responseText;
16    }
17    xmlhttp.send();
18 </script>
19 </head>
20 <body>
21 <h1>Treinamento de JavaScript</h1>
22 <div id="dados"></div>
23 </body>
24 </html>
```

A imagem a seguir mostra a execução do código anterior:

Treinamento de JavaScript

40h - Impacta Tecnologia

Sao Paulo - SP

Brasil

9.3.2. Solicitando JSON

Para requisitar JSON por meio do AJAX, você deve usar o seguinte código:

- Exemplo.txt

```
{  
    "dados" : [  
        {"nome":"Bruno", "idade":26}  
    ]  
}
```

- Exemplo:

```
1 <!doctype html>  
2 <html lang="pt-br">  
3 <head>  
4 <meta charset="utf-8">  
5 <title>JavaScript</title>  
6 <script>  
7     var xmlhttp;  
8     if (window.XMLHttpRequest)  
9         xmlhttp=new XMLHttpRequest(); //Navegadores atuais  
10    else  
11        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6  
12  
13    xmlhttp.open("GET","exemplo.txt");  
14    xmlhttp.onreadystatechange= function(){  
15        if(xmlhttp.readyState==4) {  
16            var json = JSON.parse(xmlhttp.responseText);  
17            document.getElementById('nome').innerHTML = json.dados[0].nome;  
18            document.getElementById('idade').innerHTML = json.dados[0].idade;  
19        }  
20    }  
21    xmlhttp.send();  
22 </script>  
23 </head>  
24 <body>  
25 <div>  
26     <h1>Dados Cadastrais</h1>  
27     <label>Nome:</label><span id="nome"></span>  
28     <br />  
29     <label>Idade:</label><span id="idade"></span>  
30 </div>  
31 </body>  
32 </html>
```

- Resultado:

Dados Cadastrais

Nome:Bruno

Idade:26

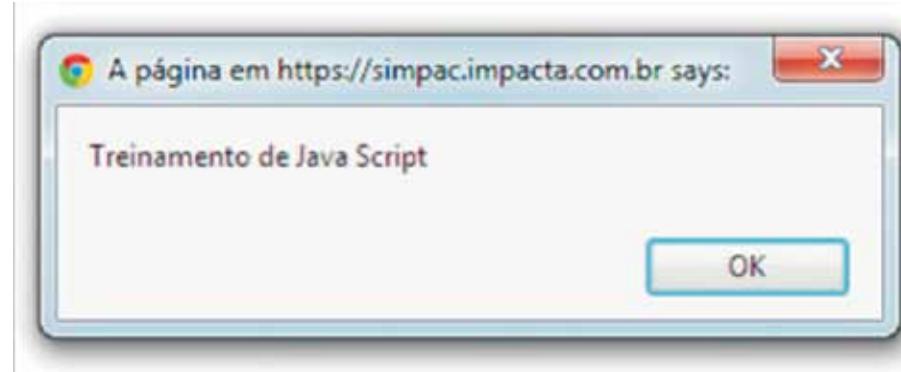
9.3.3. Solicitando JavaScript

Para solicitar JavaScript usando AJAX, utilize o seguinte código:

```
1 <!doctype html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="utf-8">
5 <title>JavaScript</title>
6 <script>
7     var xmlhttp;
8     if (window.XMLHttpRequest)
9         xmlhttp=new XMLHttpRequest(); //Navegadores atuais
10    else
11        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP"); //IE 5 e 6
12
13    xmlhttp.open("GET","exemplo.js");
14    xmlhttp.onreadystatechange = function(){
15        if(xmlhttp.readyState==4) eval(xmlhttp.responseText);
16    }
17    xmlhttp.send();
18 </script>
19 </head>
20 <body>
21 </body>
22 </html>
```

JavaScript

A imagem a seguir mostra a execução do código anterior:



Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- AJAX permite criar páginas dinâmicas e rápidas através de mecanismos que estabelecem a troca de informações com o servidor sem a necessidade de atualizar e recarregar a página Web inteira, apenas partes dela, reduzindo os dados trafegados na rede não havendo necessidade de aguardar a página Web ser recarregada a cada interação com o servidor;
- O objeto **XMLHttpRequest** envia as solicitações HTTP ou HTTPS diretamente a um servidor Web e depois carrega direto no script os dados obtidos como resposta. Os dados devem, obrigatoriamente, ser recebidos em texto simples, JSON, XML ou HTML;
- Os navegadores mais modernos possuem um objeto **XMLHttpRequest** nativo, mas é bom verificar se o navegador que você está utilizando oferece suporte a este objeto. Pode ser necessário criá-lo quando o navegador oferecer suporte ou, em caso contrário, há a necessidade de utilizar um objeto **ActiveXObject**;
- Após a criação da instância do objeto **XMLHttpRequest**, é necessário estabelecer comunicação com o servidor, podendo, posteriormente, requisitar um arquivo e/ou enviar dados. Para isso, você pode utilizar alguns métodos, dentre os quais **open**, **send**, **setRequestHeader** e **onreadystatechange** são os mais importantes;
- O método **open** passa ao servidor os dados sobre o endereço do arquivo que foi requisitado pelo navegador; o método **send** dá início à requisição que foi definida pelo método **open**; o método **setRequestHeader** é chamado a fim de enviar, junto com a solicitação, os cabeçalhos HTTP. Por fim, a ação **onreadystatechange** é disparada pelo servidor, ocorrendo toda vez que ele enviar para o cliente a informação de uma alteração ou atualização no status de sua comunicação com o cliente;
- Para obter a resposta do servidor, as propriedades **responseText** e **responseXML** são utilizadas. Quando a resposta do servidor não é XML, utilize a propriedade **respondeText**, cujo retorno é uma string e, quando é XML, utilize a propriedade **responseXML**.

9

AJAX

Teste seus conhecimentos

Ana G. Cesar
778.6008-0609



IMPACTA
EDITORA

1. Qual a sintaxe correta para criar um objeto XMLHttpRequest?

- a) variavel=new XMLHttpRequest();
- b) variavel=XMLHttpRequest();
- c) variavel=XMLHttpRequests();
- d) variavel=new XMLHttpRequests();
- e) variavel=new HttpXMLRequest();

2. O método open do objeto XMLHttpRequest tem por finalidade?

- a) Abrir uma conexão com o servidor.
- b) Enviar uma notificação para o servidor.
- c) Passar ao servidor os dados sobre o endereço do arquivo que foi requisitado pelo navegador.
- d) Passar ao servidor os dados sobre o tipo do arquivo que foi requisitado pelo navegador.
- e) Nenhuma das alternativas anteriores está correta.

3. O método send do objeto XMLHttpRequest tem por finalidade?

- a) Iniciar à requisição que foi definida pelo método open.
- b) Enviar uma requisição para o servidor.
- c) Enviar e receber uma requisição do servidor.
- d) Enviar uma requisição e posteriormente encerrar esta conexão com o servidor.
- e) Nenhuma das alternativas anteriores está correta.

4. Para obter as resposta do servidor a uma requisição, pode utilizar?

- a) responseJSON e responseXML
- b) responseText e responseXML
- c) responseText e responseJSON
- d) responseTXT e responseXML
- e) Nenhuma das alternativas anteriores está correta.

5. A resposta do servidor web a uma requisição XMLHttpRequest deve ser recebida obrigatoriamente por:

- a) JSON, XML, HTML ou texto simples.
- b) Texto simples, JSON ou HTML.
- c) Texto simples, HTML ou XML.
- d) JSON, XML ou HTML.
- e) Nenhuma das alternativas anteriores está correta.



9

AJAX

Mãos à obra!

Ana G. Caezar
778.6000.



IMPACTA
EDITORA

Laboratório 1

A - Lendo arquivo JSON com uma chamada XMLHttpRequest

1. Crie um novo arquivo de texto e renomeie com a extensão **.html**;
2. Abra o arquivo com o editor de texto;
3. Adicione as tags **HTML**, **HEAD**, **BODY**;
4. Inclua dentro da tag **BODY** o seguinte HTML:

```
<html>
<head>
</head>
<body>

<h2>Utilizando o XMLHttpRequest object</h2>
<div id="myDiv"></div>
<button type="button">Ler Arquivo</button>
</body>
</html>
```

Isso resulta em:

Utilizando o XMLHttpRequest object

Ler Arquivo

5. Inclua dentro da tag **HEAD** a tag **<script></script>**;
6. Inclua no botão **Ler Arquivo** um evento **onClick** e crie uma função que irá fazer a requisição **XMLHttpRequest**;
 - Nesta função, deve-se criar um objeto **XMLHttpRequest**.

- Crie no local onde foi criada esta página um arquivo TXT.
- Neste arquivo TXT, insira o seguinte conteúdo:

```
{  
    "times" : [  
        {"nome":"Corinthias","posicao":1},  
        {"nome":"Santos","posicao":2},  
        {"nome":"Palmeiras","posicao":20}  
    ]  
}
```

- Considere:

- readyState = 4;
- status = 200.

7. O resultado final é mostrar no console o conteúdo do JSON.

Referências bibliográficas

Livros:

Silva, Maurício Samy. JavaScript: Guia do programador. São Paulo: Novatec Editora, 2010.

Impacta Tecnologia Eletrônica LTDA. JavaScript. 1. ed. São Paulo: Impacta Tecnologia Eletrônica Ltda, 2007. 472 p.

Sites:

Google Developers. Disponível em: <<https://developers.google.com/chrome-developer-tools/docs/overview>> Acesso em: 25/04/2012

JavaScript. In: Wikipedia, The Free Encyclopedia. Flórida: Wikimedia Foundation, 2012. Disponível em: <<http://en.wikipedia.org/wiki/JavaScript>>. Acesso em: 25/04/2012.

Mozilla Developer Network. Disponível em: <<https://developer.mozilla.org/en-US/docs/JavaScript/Guide>> Acesso em: 25/04/2012

W3Schools. Disponível em: <<http://www.w3schools.com/js/default.asp>> Acesso em: 25/04/2012



