Apostila de Javascript

Autor: Anderson Henrique R Maciel

Capítulo 1

Introdução ao JavaScript

Esta página contém alguns exemplos do que o JavaScript pode fazer.

JavaScript pode alterar o conteúdo HTML

Um dos muitos métodos JavaScript HTML é getElementById ( ).

Este exemplo usa o método para "encontrar" um elemento HTML (com id = "demo") e muda o conteúdo do elemento (innerHTML) para "Olá JavaScript":

```
document.getElementById("demo").innerHTML = "Olá JavaScript";
```

O JavaScript aceita as citações duplas e simples:

```
document.getElementById('demo').innerHTML = 'Olá JavaScript';
```

JavaScript pode alterar os atributos HTML

Este exemplo altera uma imagem HTML alterando o atributo src (source) de uma tag <img>:

<button onclick = "document.getElementById ('myImage'). src = 'pic_bulbon.gif'"> Ligue a luz </ button>

<img id = "myImage" src = "pic_bulboff.gif" style = "width: 100px">

<button onclick = "document.getElementById ('myImage'). src = 'pic_bulboff.gif'"> Desligue a luz </ button>

JavaScript pode mudar estilos HTML (CSS)

Alterar o estilo de um elemento HTML é uma variante de alteração de um atributo HTML:

```
document.getElementById("demo").style.fontSize = "35px";
ou
document.getElementById('demo').style.fontSize = '35px';
```

JavaScript pode ocultar elementos HTML

Ocultar elementos HTML pode ser feito alterando o estilo de exibição:

<p id = "demo"> JavaScript pode ocultar elementos HTML. </ p>

<button type = "button" onclick = "document.getElementById ('demo'). style.display = 'none'"> Clique em Mim! </ button>

JavaScript pode mostrar elementos HTML

Mostrar elementos HTML ocultos também pode ser feito alterando o estilo de exibição:

<p id = "demo" style = "display: none"> Olá JavaScript! </ p>

<button type = "button" onclick = "document.getElementById ('demo'). style.display = 'block'"> Clique em Mim! </ button>

Capítulo 2

JavaScript Onde

A tag <script>

Em HTML, o código JavaScript deve ser inserido entre as tags <script> e </ script>.

<script>

document.getElementById ("demo"). innerHTML = "Meu primeiro JavaScript";

</ script>

Exemplos de JavaScript anteriores podem usar um atributo de tipo: <script type = "text / javascript">.

O atributo de tipo não é necessário. O JavaScript é a linguagem de script padrão em HTML.

Funções e eventos do JavaScript

Uma função JavaScript é um bloco de código JavaScript, que pode ser executado quando "chamado" para.

Por exemplo, uma função pode ser chamada quando ocorre um evento, como quando o usuário clica em um botão.

Você aprenderá muito mais sobre funções e eventos em capítulos posteriores.

JavaScript em <head> ou <body>

Neste exemplo, uma função JavaScript é colocada na seção <head> de uma página HTML.

A função é invocada (chamada) quando um botão é clicado:

```
<!DOCTYPE html>
<html>

<head>
<script>
function minhaFuncao() {
    document.getElementById("demo").innerHTML = "Paragrafo
alterado.";
}
</script>
</head>

<body>

<h1>Uma página Web.</h1>
<p id="demo">Um parágrafo</p>
<button type="button" onclick="minhaFuncao()">

Experimente

</button>

</body>
</html>
```

JavaScript em <body>

Neste exemplo, uma função JavaScript é colocada na seção <body> de uma página HTML.

A função é invocada (chamada) quando um botão é clicado:

```
<!DOCTYPE html>
<html>
<body>

<h1>Uma página Web.</h1>
<p id="demo">Um parágrafo</p>
<button type="button" onclick="minhaFuncao()">Experimente</button>

<script>
function minhaFuncao() {
   document.getElementById("demo").innerHTML = "Paragrafo alterado.";
}
```

```
</script>

</body>
</html>
```

A colocação de scripts na parte inferior do elemento <body> melhora a velocidade da exibição, pois a compilação do script diminui a exibição.

JavaScript externo

Scripts também podem ser colocados em arquivos externos:

Arquivo externo: myScript.js

```
function minhaFuncao() {
   document.getElementById("demo").innerHTML = "Paragrafo alterado.";
}
```

Os scripts externos são práticos quando o mesmo código é usado em muitas páginas da web diferentes.

Os arquivos de JavaScript possuem a extensão de arquivo .js.

Para usar um script externo, coloque o nome do arquivo de script no atributo src (source) de uma tag <script>:

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

Você pode colocar uma referência de script externa em <head> ou <body> como quiser.

O script funcionará como se estivesse localizado exatamente onde a marca <script> está localizada.

Os scripts externos não podem conter tags <script>.

Vantagens de JavaScript Externo

A colocação de scripts em arquivos externos possui algumas vantagens:

- Separa HTML e código
- Isso torna o HTML e o JavaScript mais fáceis de ler e manter
- Arquivos de JavaScript em cache podem acelerar cargas de página

Para adicionar vários arquivos de script a uma página - use várias tags de script:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

Referências externas

Os scripts externos podem ser referenciados com um URL completo ou com um caminho relativo à página da web atual.

Este exemplo usa um URL completo para vincular a um script:

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

Este exemplo usa um script localizado em uma pasta especificada no site atual:

```
<script src="/js/myScript1.js"></script>
```

Este exemplo linka para um script localizado na mesma pasta que a página atual:

```
<script src="myScript1.js"></script>
```

Capítulo 3

Saída de JavaScript

Possibilidades de exibição de JavaScript

O JavaScript pode "exibir" dados de diferentes maneiras:

- Escrevendo em um elemento HTML, usando innerHTML.
- Escrevendo na saída HTML usando document.write ().
- Escrevendo em uma caixa de alerta, usando window.alert ().
- Escrevendo no console do navegador, usando console.log ().

Usando innerHTML

Para acessar um elemento HTML, o JavaScript pode usar o método document.getElementById (id).

O atributo id define o elemento HTML. A propriedade innerHTML define o conteúdo HTML:

```
<!DOCTYPE html>
<html>
<body>

<h1>Minha Primeira Página Web</h1>
<p>Meu Primeiro Parágrafo</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
```

```
</body>
</html>
```

Alterar a propriedade innerHTML de um elemento HTML é uma maneira comum de exibir dados em HTML.

Usando document.write ()

Para fins de teste, é conveniente usar document.write ():

```
<!DOCTYPE html>
<html>
<body>

<h1>Minha Primeira Página Web</h1>
<p>Meu Primeiro Parágrafo</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

Usando document.write () depois de um documento HTML estar totalmente carregado, irá excluir todos os HTML existentes:

```
<!DOCTYPE html>
<html>
<body>

<h1>Minha Primeira Página Web</h1>
<p>Meu Primeiro Parágrafo</p>

<button onclick="document.write(5 + 6)">Experimente</button>

</body>
</html>
```

O método document.write () só deve ser usado para testes.

Usando window.alert ()

Você pode usar uma caixa de alerta para exibir dados:

```
<!DOCTYPE html>
<html>
<body>

<h1>Minha Primeira Página Web</h1>
<p>Meu Primeiro Parágrafo</p>
```

```
<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

Usando console.log ()

Para fins de depuração, você pode usar o método console.log () para exibir dados.

Você aprenderá mais sobre a depuração em um capítulo posterior.

```
<!DOCTYPE html>
<html>
<body>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

Capítulo 4

Sintaxe do JavaScript

A sintaxe JavaScript é o conjunto de regras, como os programas JavaScript são construídos.

Programas de JavaScript

Um programa de computador é uma lista de "instruções" a serem "executadas" pelo computador.

Em uma linguagem de programação, estas instruções do programa são chamadas de declarações.

O JavaScript é uma linguagem de programação.

Declarações de JavaScript são separadas por ponto e vírgula:

```
var x, y, z;
x = 5;
y = 6;
z = x + y;
```

Em HTML, os programas JavaScript são executados pelo navegador da Web.

Declarações de JavaScript

Declarações de JavaScript são compostas por:

Valores, Operadores, Expressões, Palavras-chave e Comentários.

Valores de JavaScript

A sintaxe JavaScript define dois tipos de valores: valores fixos e valores variáveis.

Valores fixos são chamados literais. Os valores variáveis são chamados de variáveis.

Literais de JavaScript

As regras mais importantes para escrever valores fixos são:

Os números são escritos com ou sem decimais:

```
10.50
1001
```

As strings são texto, escritas em citações duplas ou simples:

```
"John Doe"
'John Doe'
```

Variáveis de JavaScript

Em uma linguagem de programação, as variáveis são usadas para armazenar valores de dados.

JavaScript usa a palavra-chave var para declarar variáveis.

Um sinal de igual é usado para atribuir valores a variáveis.

Neste exemplo, x é definido como uma variável. Então, x é atribuído (dado) o valor 6:

```
var x;
x = 6;
```

Operadores de JavaScript

O JavaScript usa operadores aritméticos (+ - * /) para calcular valores:

```
(5 + 6) * 10
```

O JavaScript usa um operador de atribuição (=) para atribuir valores às variáveis:

```
var x, y;
x = 5;
y = 6;
```

Expressões de JavaScript

Uma expressão é uma combinação de valores, variáveis e operadores, que calcula para um valor.

O cálculo é chamado de avaliação.

Por exemplo, 5 * 10 avalia para 50:

```
5 * 10
```

Expressões também podem conter valores variáveis:

```
x * 10
```

Os valores podem ser de vários tipos, como números e strings.

Por exemplo, "John" + "" + "Doe", avalia para "John Doe":

```
"John" + " " + "Doe"
```

Palavras-chave de JavaScript

As palavras-chave JavaScript são usadas para identificar as ações a serem executadas.

A palavra-chave var fala ao navegador para criar variáveis:

```
var x, y;
x = 5 + 6;
y = x * 10;
```

Comentários de JavaScript

Nem todas as declarações de JavaScript são "executadas".

Código após barras duplas // ou entre / * e * / é tratado como um comentário.

Os comentários são ignorados e não serão executados:

```
var x = 5;   // Será executado

// var x = 6;   Não será executado
```

Você aprenderá mais sobre comentários em um capítulo posterior.

Identificadores de JavaScript

Os identificadores são nomes.

Em JavaScript, identificadores são usados para nomear variáveis (e palavras-chave, funções e rótulos).

As regras para nomes legais são as mesmas na maioria das linguagens de programação.

Em JavaScript, o primeiro caractere deve ser uma letra, um sublinhado (_) ou um sinal de dólar ($).

Os caracteres subsequentes podem ser letras, dígitos, sublinhados ou sinais de dólar.

Os números não são permitidos como o primeiro caractere.

Desta forma, o JavaScript pode distinguir facilmente identificadores de números.

JavaScript é Case Sensitive

Todos os identificadores de JavaScript diferenciam maiúsculas de minúsculas

As variáveis lastName e lastname são duas variáveis diferentes.

```
var lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

O JavaScript não interpreta VAR ou Var como a palavra-chave var.

JavaScript e Caso Camelo

Historicamente, os programadores usaram diferentes maneiras de juntar várias palavras em um nome de variável:

Hífens:

First-name, last-name, master-card, inter-cidade.

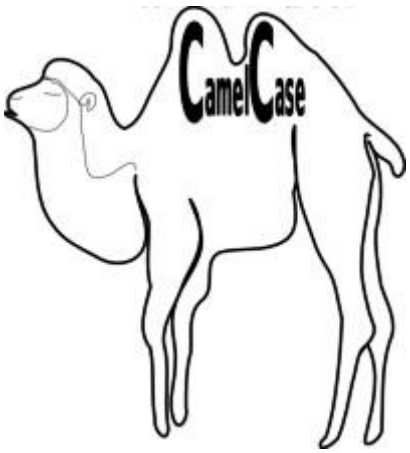
Hífens não são permitidos em JavaScript. É reservado para subtrações.

Underline:

first_name, last_name, master_card, inter_city.

Caso de camelo superior (Pascal Case):

FirstName, LastName, MasterCard, InterCity.



Caso de camelo inferior:

Os programadores de JavaScript tendem a usar o caso de camelo que começa com uma letra minúscula:

firstName, lastName, masterCard, interCity.

Conjunto de caracteres JavaScript

O JavaScript usa o conjunto de caracteres Unicode.

Unicode cobre (quase) todos os caracteres, pontuações e símbolos no mundo.

Para um olhar mais atento, por favor, estude nossa referência Unicode completa.

Capítulo 5

Declarações de JavaScript

Em HTML, as instruções de JavaScript são "instruções" para serem "executadas" pelo navegador da Web.

Declarações de JavaScript

Esta declaração diz ao navegador que escreva "Hello Dolly". dentro de um elemento HTML com id = "demo":

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

Programas de JavaScript

A maioria dos programas JavaScript contém muitas instruções de JavaScript.

As declarações são executadas, uma a uma, na mesma ordem em que estão escritas.

Neste exemplo x, y e z são dados valores, e finalmente z é exibido:

```
var x, y, z;
x = 5;
y = 6;
z = x + y;
document.getElementById("demo").innerHTML = z;
```

Programas de JavaScript (e instruções de JavaScript) geralmente são chamados de código JavaScript.

Ponto-e-vírgula;

Ponto-e-vírgula separam instruções de JavaScript.

Adicione um ponto-e-vírgula no final de cada declaração executável:

```
var a, b, c;
a = 5;
b = 6;
c = a + b;
```

Quando separados por ponto e vírgula, várias declarações em uma linha são permitidas:

```
a = 5; b = 6; c = a + b;
```

Na web, você pode ver exemplos sem pontos-e-vírgula. As declarações de conclusão com ponto-e-vírgula não são necessárias, mas altamente recomendadas.

Espaçamentos no JavaScript

O JavaScript ignora vários espaços. Você pode adicionar espaço em branco ao seu script para torná-lo mais legível.

As seguintes linhas são equivalentes:

```
var person = "Hege";
var person="Hege";
```

Uma boa prática é colocar espaços em torno de operadores (= + - * /):

```
var x = y + z;
```

Comprimento da linha JavaScript e intervalos de linha

Para uma melhor legibilidade, os programadores geralmente gostam de evitar linhas de código com mais de 80 caracteres.

Se uma declaração de JavaScript não se encaixa em uma linha, o melhor lugar para quebrar, é depois de um operador:

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

Blocos de código JavaScript

As instruções de JavaScript podem ser agrupadas em blocos de código, dentro de suportes curly {...}.

O propósito dos blocos de código é definir declarações a serem executadas em conjunto.

Um lugar onde você encontrará declarações agrupadas em blocos, está em funções de JavaScript:

```
function myFunction() {
    document.getElementById("demo1").innerHTML = "Hello Dolly!";
    document.getElementById("demo2").innerHTML = "How are you?";
}
```

Neste tutorial usamos 4 espaços de recuo para blocos de código. Você aprenderá mais sobre as funções mais adiante neste tutorial.

Palavras-chave de JavaScript

As instruções de JavaScript geralmente começam com uma palavra-chave para identificar a ação de JavaScript a ser executada.

Aqui está uma lista de algumas das palavras-chave que você aprenderá neste tutorial:

Palavra-chave	Descrição
break	encerra um switch ou um loop
continue	salta de um loop e começa no topo
debugger	pára a execução do JavaScript e chama (se disponível) a função de depuração
do ... while	executa um bloco de declarações e repete o bloco, enquanto uma condição é verdadeira
for	marca um bloco de declarações a serem executadas, desde que uma condição seja verdadeira
function	declara uma função
if ... else	marca um bloco de declarações a serem executadas, dependendo de uma condição
return	sai de uma função
switch	marca um bloco de declarações a serem executadas, dependendo de casos diferentes
try ... catch	implementa o tratamento de erros em um bloco de instruções
var	declara uma variável

As palavras-chave JavaScript são palavras reservadas. As palavras reservadas não podem ser usadas como nomes para variáveis.

Capítulo 6

Comentários de JavaScript

Os comentários de JavaScript podem ser usados para explicar o código JavaScript e torná-lo mais legível.

Os comentários de JavaScript também podem ser usados para evitar a execução, ao testar o código alternativo.

Comentários de linha única

Os comentários de linha única começam com //.

Qualquer texto entre // e o fim da linha será ignorado pelo JavaScript (não será executado).

Este exemplo usa um comentário de linha única antes de cada linha de código:

```
// Mudar título:
document.getElementById("myH").innerHTML = "Minha Primeira Página";
// Mudar parágrafo:
document.getElementById("myP").innerHTML = "Meu primeiro parágrafo.";
```

Este exemplo usa um comentário de linha única no final de cada linha para explicar o código:

```
var x = 5;      // Declare x, dê o valor de 5
var y = x + 2;  // Declare y, dê o valor de x + 2
```

Comentários de várias linhas

Comentários de várias linhas começam com / * e terminam com * /.

Qualquer texto entre / * e * / será ignorado pelo JavaScript.

Este exemplo usa um comentário de várias linhas (um bloco de comentários) para explicar o código:

```
/*
O código abaixo irá mudar
o título com id = "myH"
e o parágrafo com id = "myP"
na minha página web:
*/
document.getElementById("myH").innerHTML = "Minha Primeira Página";
document.getElementById("myP").innerHTML = "Meu primeiro parágrafo.";
```

É mais comum usar comentários de linha única.

Os comentários em bloco geralmente são usados para documentação formal.

Usando comentários para impedir a execução

O uso de comentários para evitar a execução do código é adequado para testes de código.

Adicionar // na frente de uma linha de código altera as linhas de código de uma linha executável para um comentário.

Este exemplo usa // para evitar a execução de uma das linhas de código:

```
//document.getElementById("myH").innerHTML = "Minha Primeira Página";
document.getElementById("myP").innerHTML = "Meu primeiro parágrafo.";
```

Este exemplo usa um bloco de comentários para evitar a execução de várias linhas:

```
/*
document.getElementById("myH").innerHTML = "Minha Primeira Página";
document.getElementById("myP").innerHTML = "Meu primeiro parágrafo.";
*/
```

Capítulo 7

Variáveis de JavaScript

As variáveis JavaScript são recipientes para armazenar valores de dados.

Neste exemplo, x, y e z, são variáveis:

```
var x = 5;
var y = 6;
var z = x + y;
```

A partir do exemplo acima, você pode esperar:

- x armazena o valor 5
- y armazena o valor 6
- z armazena o valor 11

Muito parecido com álgebra

Neste exemplo, price1, price2 e total, são variáveis:

```
var preco1 = 5;
var preco2 = 6;
var total = preco1 + preco2;
```

Na programação, como na álgebra, usamos variáveis (como preco1) para manter valores.

Na programação, assim como na álgebra, usamos variáveis em expressões (total = preco1 + preco2).

A partir do exemplo acima, você pode calcular o total para ser 11.

As variáveis JavaScript são recipientes para armazenar valores de dados.

Identificadores de JavaScript

Todas as variáveis JavaScript devem ser identificadas com nomes exclusivos.

Esses nomes exclusivos são chamados de identificadores.

Os identificadores podem ser nomes curtos (como x e y) ou mais nomes descritivos (idade, soma, totalVolume).

As regras gerais para a construção de nomes para variáveis (identificadores exclusivos) são:

- Os nomes podem conter letras, dígitos, sublinhados e sinais de dólar.
- Os nomes devem começar com uma letra
- Os nomes também podem começar com $ e _ (mas não vamos usá-lo neste tutorial)
- Os nomes são sensíveis a maiúsculas e minúsculas (y e Y são variáveis diferentes)
- As palavras reservadas (como palavras-chave JavaScript) não podem ser usadas como nomes

Os identificadores de JavaScript diferenciam maiúsculas de minúsculas.

O Operador de Atribuição

Em JavaScript, o sinal de igual (=) é um operador de "atribuição", não um operador "igual a".

Isso é diferente da álgebra. O seguinte não faz sentido na álgebra:

```
x = x + 5
```

Em JavaScript, no entanto, faz todo o sentido: atribui o valor de x + 5 a x.

(Calcula o valor de x + 5 e coloca o resultado em x. O valor de x é incrementado em 5.)

O operador "igual a" é escrito como == em JavaScript.

Tipos de dados JavaScript

As variáveis de JavaScript podem conter números como 100 e valores de texto como "John Doe".

Na programação, os valores de texto são chamados de strings de texto.

O JavaScript pode lidar com muitos tipos de dados, mas, por enquanto, apenas pense em números e strings.

Strings são escritas dentro de citações duplas ou soltas. Os números estão escritos sem aspas.

Se você colocar um número entre aspas, ele será tratado como uma seqüência de texto.

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Sim, sou eu!';
```

Declarando (Criando) Variáveis de JavaScript

Criar uma variável em JavaScript é chamado de "declarar" uma variável.

Você declara uma variável de JavaScript com a palavra-chave var:

```
var nomeCarro;
```

Após a declaração, a variável não tem valor. (Tecnicamente tem o valor de indefinido)

Para atribuir um valor à variável, use o sinal de igual:

```
nomeCarro = "Volvo";
```

Você também pode atribuir um valor à variável quando declara:

```
var nomeCarro = "Volvo";
```

No exemplo abaixo, criamos uma variável chamada nomeCarro e atribuímos o valor "Volvo" a ele.

Então, "exibimos" o valor dentro de um parágrafo HTML com id = "demo":

```
<p id="demo"></p>

<script>
var nomeCarro = "Volvo";
document.getElementById("demo").innerHTML = nomeCarro;
</script>
```

É uma boa prática de programação declarar todas as variáveis no início de um script.

Uma declaração, muitas variáveis

Você pode declarar várias variáveis em uma declaração.

Comece a declaração com var e separe as variáveis por vírgula:

```
var pessoa = "John Doe", nomeCarro = "Volvo", preco = 200;
```

Uma declaração pode abranger várias linhas:

```
var pessoa = "John Doe",
nomeCarro = "Volvo",
preco = 200;
```

Valor = indefinido

Em programas de computador, as variáveis são frequentemente declaradas sem valor. O valor pode ser algo que deve ser calculado, ou algo que será fornecido posteriormente, como a entrada do usuário.

Uma variável declarada sem valor terá o valor indefinido.

A variável nomeCarro terá o valor indefinido após a execução desta declaração:

```
var nomeCarro;
```

Re-declarando variáveis de JavaScript

Se você re-declarar uma variável JavaScript, não perderá seu valor.

A variável nomeCarro ainda terá o valor "Volvo" após a execução dessas instruções:

```
var nomeCarro = "Volvo";
var nomeCarro;
```

Aritmética do JavaScript

Tal como acontece com a álgebra, você pode fazer aritmética com variáveis JavaScript, usando operadores como = e +:

```
var x = 5 + 2 + 3;
```

Você também pode adicionar strings, mas as strings serão concatenadas:

```
var x = "John" + " " + "Doe";
```

Também experimente isso:

```
var x = "5" + 2 + 3;
```

Se você colocar um número entre aspas, o resto dos números serão tratados como strings e concatenados.

Agora tente isso:

```
var x = 2 + 3 + "5";
```

Capítulo 8

Operadores de JavaScript

Exemplo

Atribua valores a variáveis efetua a soma :

```
var x = 5;          // atribua o valor 5 a x
var y = 2;          // atribua o valor 2 a y
var z = x + y;      // atribua o valor 7 a z (x + y)
```

O operador de atribuição (=) atribui um valor a uma variável.

```
var x = 10;
```

O operador de adição (+) soma números:

```
var x = 5;
var y = 2;
var z = x + y;
```

O operador de multiplicação (*) multiplica números.

```
var x = 5;
var y = 2;
var z = x * y;
```

Operadores Aritméticos de JavaScript

Os operadores aritméticos são usados para executar aritmética em números:

Operadores	Descrição
+	Adição
-	Subtração
*	Multiplicação

/	Divisão
%	Módulo (resto)
++	Incremento
--	Decremento

Operadores aritméticos são totalmente descritos no capítulo aritmética de JS.

Operadores de atribuição de JavaScript

Os operadores de atribuição atribuem valores às variáveis JavaScript.

Operador	Exemplo	É igual
=	x = y	X = y
+=	x += y	x = x + y
-=	x -= y	x = x – y
***=**	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

O operador de atribuição de adição (+ =) adiciona um valor a uma variável.

```
var x = 10;
x += 5;
```

Os operadores de atribuição são totalmente descritos no capítulo de Atribuição JS.

Operadores de String de JavaScript

O operador + também pode ser usado para adicionar (concatenar) strings.

```
txt1 = "John";
txt2 = "Doe";
txt3 = txt1 + " " + txt2;
```

O resultado de txt3 será:

```
John Doe
```

O operador + = atribuição também pode ser usado para adicionar (concatenar) strings:

```
txt1 = "Que dia ";
txt1 += "muito bom";
```

O resultado de txt1 será:

Que dia muito bom

Quando usado em strings, o operador + é chamado de operador de concatenação.

Adicionando strings e números

Adicionando dois números, retornará a soma, mas adicionando um número e uma seqüência de caracteres retornará uma string:

```
x = 5 + 5;
y = "5" + 5;
z = "Olá" + 5;
```

O resultado de x, y e z será:

```
10
55
Olá5
```

Se você adicionar um número e uma string, o resultado será uma string!

Operadores de comparação de JavaScript

Operador	Descrição
==	Igual a
===	Valor igual e tipo igual
!=	Diferente de
!==	Valor diferente e tipo diferente
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
?	Ternário (retorna true ou false)

Os operadores de comparação estão completamente descritos no capítulo Comparações JS.

Operadores lógicos de JavaScript

Operador	Descrição
&&	E (and)
||	Ou (or)
!	Não (not)

Os operadores lógicos são totalmente descritos no capítulo Comparações JS.

Operadores de tipo JavaScript

Operador	Descrição
typeof	Retorna o tipo de variável
instanceof	Retorna true se um objeto for uma instância de um tipo de objeto

Os operadores de tipo são totalmente descritos no capítulo Conversão de tipo JS.

Operadores Bitwise

Os operadores de bit funcionam em números de 32 bits.

Qualquer operando numérico na operação é convertido em um número de 32 bits. O resultado é convertido de volta para um número de JavaScript.

Operador	Descrição	Exemplo	É igual	Resultado	Decimal
&	E (and)	5 & 1	0101 & 001	001	1
|	Ou (or)	5 | 1	0101 | 001	0101	5
~	Não (not)	~ 5	~0101	1010	10
^	Xor	5 ^ 1	0101 ^ 001	0100	4
<<	Desloca o bit para a esquerda	5 << 1	0101 << 001	1010	10
>>	Desloca o bit para a direita	5 >> 1	0101 >> 001	0010	2
>>>	Desloca o bit para a direita com sinal	5 >>> 1	0101 >>> 001	0010	2

Os exemplos acima usam exemplos não assinados de 4 bits. Mas o JavaScript usa números assinados de 32 bits. Por isso, em JavaScript, ~ 5 não retornará 10. Ele retornará -6. ~ 00000000000000000000000000000101 retornará 11111111111111111111111111111010

Os operadores Bitwise são descritos completamente no capítulo JS Bitwise.

Capítulo 9

Aritmética do JavaScript

Operadores Aritméticos de JavaScript

Operadores aritméticos executam aritmética em números (literais ou variáveis).

Operadores	Descrição
+	Adição
-	Subtração
*****	Multiplicação
/	Divisão
%	Módulo (resto)
++	Incremento
--	Decremento

Operações aritméticas

Uma operação aritmética típica opera em dois números.

Os dois números podem ser literais:

```
var x = 100 + 50;
```

Ou variáveis:

```
var x = a + b;
```

Ou expressão:

```
var x = (100 + 50) * a;
```

Operadores e Operandos

Os números (em uma operação aritmética) são chamados de operandos. A operação (a ser realizada entre os dois operandos) é definida por um operador.

Operando	Operador	Operando
100	+	50

O operador de adição (+) adiciona números:

```
var x = 5;
var y = 2;
var z = x + y;
```

O operador de subtração (-) subtrai números.

```
var x = 5;
var y = 2;
var z = x - y;
```

O operador de multiplicação (*) multiplica números.

```
var x = 5;
var y = 2;
var z = x * y;
```

O operador da divisão (/) divide os números.

```
var x = 5;
var y = 2;
var z = x / y;
```

O operador modular (%) retorna o restante da divisão.

```
var x = 5;
var y = 2;
var z = x % y;
```

O operador de incremento (++) incrementa os números (soma 1).

```
var x = 5;
x++;
var z = x;
```

O operador de decremento (-) diminui os números (subtrai 1).

```
var x = 5;
x--;
var z = x;
```

Precedência dos Operadores

O precedência do operador descreve a ordem em que as operações são realizadas em uma expressão aritmética.

```
var x = 100 + 50 * 3;
```

O resultado do exemplo acima é o mesmo que 150 * 3, ou é o mesmo que 100 + 150?

A adição ou a multiplicação são feitas primeiro?

Como na matemática escolar tradicional, a multiplicação é feita primeiro.

A multiplicação (*) e a divisão (/) têm maior precedência do que a adição (+) e a subtração (-).

E (como na matemática escolar), a precedência pode ser alterada usando parênteses:

```
var x = (100 + 50) * 3;
```

Ao usar parênteses, as operações dentro dos parênteses são calculadas primeiro.

Quando muitas operações têm a mesma precedência (como adição e subtração), elas são computadas da esquerda para a direita:

```
var x = 100 + 50 - 3;
```

Valores de precedência do operador JavaScript

Valor	Operador	Descrição	Exemplo
19	( )	Agrupamento de expressões	(3 + 4)
18	.	Membro	pessoa.nome
18	[ ]	Membro	pessoa[“nome”]
17	( )	Chamada de função	minhaFuncao( )
17	new	Instancia objeto	New Date( )
16	++	Pós-incremento	i++
16	--	Pós-decremento	i--
15	++	Pré-incremento	++i
15	--	Pré-decremento	--i
15	!	Lógico não (not)	!(x == y)
15	typeof	Tipo de dado	typeof x
14	*	Multiplicação	10 * 5
14	/	Divisão	10 / 5
14	%	Módulo (resto)	10 % 5
14	**	Exponenciação	10 ** 2
13	+	Adição	10 + 5
13	-	Subtração	10 – 5
11	<	Menor que	x < y
11	<=	Menor ou igual a	x <= y
11	>	Maior que	x > y
11	>=	Maior ou igual a	x >= y
10	==	Igual a	x == y
10	===	Idêntico	x === y
10	!=	Diferente de	x != y

10	!==	Não idêntico	x !== y
6	&&	Lógico E (and)	x && y
5	||	Lógico Ou (or)	x || y
3	=	Atribuição	x = y
3	+=	Atribui e soma	x += y
3	-=	Atribui e subtrai	x -= y
3	*=	Atribui e multiplica	x *= y
3	%=	Atribui e módulo	x %= y

As entradas vermelhas claras indicam tecnologia experimental ou proposta (ECMAScript 2016 ou ES7)

As expressões entre parênteses são totalmente calculadas antes que o valor seja usado no restante da expressão.

Capítulo 10

Operadores de atribuição de JavaScript

Os operadores de atribuição atribuem valores às variáveis JavaScript.

Operador	Exemplo	É igual
=	x = y	X = y
+=	x += y	x = x + y
-=	x -= y	x = x – y
***=**	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Exemplos de atribuição

O operador = atribuição atribui um valor a uma variável.

```
var x = 10;
```

O operador + = atribuição adiciona um valor a uma variável.

```
var x = 10;
x += 5;
```

O operador - = atribuição subtrai um valor de uma variável.

```
var x = 10;
x -= 5;
```

O operador * = atribuição multiplica uma variável.

```
var x = 10;
x *= 5;
```

O operador / = atribuição divide uma variável.

```
var x = 10;
x /= 5;
```

O operador de% = atribuição atribui um restante a uma variável.

```
var x = 10;
x %= 5;
```

Capítulo 11

Tipos de dados JavaScript

As variáveis de JavaScript podem conter muitos tipos de dados: números, strings, objetos e muito mais:

```
var tamanho = 16;                                   // Número
var ultimoNome = "Johnson";                         // String
var x = {primeiroNome:"John", ultimoNome:"Doe"};    // Objeto
```

O conceito de tipos de dados

Na programação, os tipos de dados são um conceito importante. Para poder operar em variáveis, é importante saber algo sobre o tipo.

Sem tipos de dados, um computador não pode resolver com segurança isso:

```
var x = 16 + "Volvo";
```

Faz algum sentido adicionar "Volvo" a dezesseis? Isso produzirá um erro ou produzirá um resultado?

JavaScript tratará o exemplo acima como:

```
var x = "16" + "Volvo";
```

Ao adicionar um número e uma string, o JavaScript tratará o número como uma string.

```
var x = 16 + "Volvo";
```

```
var x = "Volvo" + 16;
```

O JavaScript avalia as expressões da esquerda para a direita. Seqüências diferentes podem produzir resultados diferentes:

```
var x = 16 + 4 + "Volvo";
```

```
Resultado:
```

```
20Volvo
```

```
var x = "Volvo" + 16 + 4;
```

Resultado:

Volvo164

No primeiro exemplo, o JavaScript trata 16 e 4 como números, até chegar a "Volvo".

No segundo exemplo, uma vez que o primeiro operando é uma string, todos os operandos são tratados como strings.

Tipos de JavaScript são dinâmicos

O JavaScript possui tipos dinâmicos. Isso significa que a mesma variável pode ser usada para armazenar diferentes tipos de dados:

```
var x;               // Agora x é indefinido
var x = 5;           // Now x é um Número
var x = "John";      // Now x é uma String
```

Strings JavaScript

Uma string (ou uma string de texto) é uma série de caracteres como "John Doe". As strings são escritas com aspas. Você pode usar aspas simples ou duplas

```
var nomeCarro = "Volvo XC60";   // Usando aspas duplas
var nomeCarro = 'Volvo XC60';   // Usando aspas simples
```

Você pode usar aspas dentro de uma seqüência de caracteres, desde que não correspondam às citações que cercam a seqüência de caracteres:

```
var resposta = "Está tudo bem";             // Citação única dentro de aspas duplas

var resposta = "Ele se chama 'Johnny'";    // Citações simples entre aspas duplas

var resposta = 'Ele se chama "Johnny"';    // Cotações duplas dentro de citações simples
```

Você aprenderá mais sobre strings mais adiante neste tutorial.

Números JavaScript

O JavaScript tem apenas um tipo de números. Os números podem ser escritos com, ou sem decimais:

```
var x1 = 34.00;     // Escrito com decimais
var x2 = 34;        // Escrito sem decimais
```

Os números extra grandes ou extra pequenos podem ser escritos com notação científica (exponencial):

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

Você aprenderá mais sobre números mais tarde neste tutorial.

Booleanos JavaScript

Os booleanos só podem ter dois valores: verdadeiro ou falso (true ou false).

```
var x = 5;
var y = 5;
var z = 6;
(x == y)       // Retorna true
(x == z)       // Retorna false
```

Os booleanos são freqüentemente usados em testes condicionais.

Você aprenderá mais sobre o teste condicional mais adiante neste tutorial.

Arrays JavaScript

As matrizes de JavaScript são escritas com colchetes. Os itens da matriz são separados por vírgulas.

O código a seguir declara (cria) uma matriz chamada carros, contendo três itens (nomes do carro):

```
var carros = ["Saab", "Volvo", "BMW"];
```

Os índices de matrizes são baseados em zero, o que significa que o primeiro item é [0], o segundo é [1], e assim por diante.

Você aprenderá mais sobre arrays mais adiante neste tutorial.

Objetos JavaScript

Objetos de JavaScript são escritos com chaves curly.

As propriedades dos objetos são escritas como pares nome: valor, separados por vírgulas.

```
var pessoa = {primeiroNome:"John", ultimoNome:"Doe", idade:50,
corOlhos:"azul"};
```

O objeto (pessoa) no exemplo acima possui 4 propriedades: primeiroNome, ultimoNome, idade e corOlhos.

Você aprenderá mais sobre objetos mais tarde neste tutorial.

O Operador typeof

Você pode usar o mecanismo typeof do JavaScript para encontrar o tipo de uma variável JavaScript.

O operador typeof retorna o tipo de uma variável ou uma expressão:

```
typeof ""                  // Retorna "string"
typeof "John"              // Retorna "string"
typeof "John Doe"          // Retorna "string"

typeof 0                   // Retorna "number"
typeof 314                 // Retorna "number"
```

```
typeof 3.14                 // Retorna "number"
typeof (3)                  // Retorna "number"
typeof (3 + 4)              // Retorna "number"
```

Undefined

Em JavaScript, uma variável sem valor, tem o valor indefinido. O tipo de tipo também é indefinido.

```
var carro;                  // Valor é undefined, tipo é undefined
```

Qualquer variável pode ser esvaziada, definindo o valor como indefinido. O tipo também será indefinido.

```
carro = undefined;          // Valor é undefined, tipo é undefined
```

Valores vazios

Um valor vazio não tem nada a ver com indefinido.

Uma string vazia tem um valor legal e um tipo.

```
var carro = "";             // O valor é "", o typeof é "string"
```

Null

No JavaScript, null é "nada". É suposto ser algo que não existe. Infelizmente, em JavaScript, o tipo de dados de nulo é um objeto.

Você pode considerar um bug no JavaScript que typeof null é um objeto. Isso deve ser nulo.

Você pode esvaziar um objeto configurando-o como nulo:

```
var pessoa = {primeiroNome:"John", ultimoNome:"Doe", idade:50,
corOlhos:"azul"};
pessoa = null;          // Agora o valor é nulo, mas o tipo ainda é um
objeto
```

Você também pode esvaziar um objeto configurando-o para indefinido:

```
var pessoa = {primeiroNome:"John", ultimoNome:"Doe", idade:50,
corOlhos:"azul"};
pessoa = undefined;          // Agora, tanto o valor como o tipo são
indefinidos
```

Diferença entre indefinido e nulo

Indefinido e nulo são de valor igual, mas diferentes no tipo:

```
typeof undefined           // undefined
typeof null                // object

null === undefined         // false
null == undefined          // true
```

Dados primitivos

Um valor de dados primitivo é um único valor de dados simples sem propriedades e métodos adicionais.

O tipo de operador pode retornar um desses tipos primitivos:

- string
- number
- boolean
- undefined

```
typeof "John"              // Retorna "string"
typeof 3.14                // Retorna "number"
typeof true                // Retorna "boolean"
typeof false               // Retorna "boolean"
typeof x                   // Retorna "undefined" (se x não tem valor)
```

Dados complexos

O tipo de operador pode retornar um dos dois tipos complexos:

- function
- object

O operador typeof retorna objeto para objetos, arrays e null. O operador typeof não retorna objeto para funções.

```
typeof {name:'John', age:34} // Retorna "object"
typeof [1,2,3,4]             // Retorna "object" (não "array", veja a
nota abaixo)
typeof null                  // Retorna "object"
typeof function myFunc(){}   // Retorna "function"
```

O operador typeof retorna "objeto" para arrays porque os arrays no JavaScript são objetos.

Capítulo 12

Funções de JavaScript

Uma função JavaScript é um bloco de código projetado para executar uma tarefa específica. Uma função JavaScript é executada quando "algo" invoca (chama isso).

```
function minhaFuncao(p1, p2) {
    return p1 * p2;              // A função retorna o produto de p1 e
p2
}
```

Sintaxe da função JavaScript

Uma função JavaScript é definida com a palavra-chave function, seguida de um nome, seguido de parênteses ( ).

Os nomes de funções podem conter letras, dígitos, sublinhados e sinais de dólar (mesmas regras que variáveis).

Os parênteses podem incluir nomes de parâmetros separados por vírgulas: (parâmetro1, parâmetro2, ...)

O código a ser executado, pela função, é colocado dentro de chaves: { }

```
function nome(parâmetro1, parâmetro2, parâmetro3) {
    código a ser executado
}
```

Os parâmetros de função estão listados dentro dos parênteses ( ) na definição da função. Os argumentos de função são os valores recebidos pela função quando é invocado.

Dentro da função, os argumentos (os parâmetros) se comportam como variáveis locais.

Uma função é muito parecida com um procedimento ou uma sub-rotina, em outras linguagens de programação.

Invocação de função

O código dentro da função será executado quando "algo" invoca (chamadas) a função:

- Quando ocorre um evento (quando um usuário clica em um botão)
- Quando é invocado (chamado) do código JavaScript
- Automaticamente (auto invocado)

Você aprenderá muito mais sobre invocação de função mais tarde neste tutorial.

Retorno de Função

Quando o JavaScript alcança uma declaração de retorno, a função deixará de executar.

Se a função foi invocada a partir de uma declaração, o JavaScript "retornará" para executar o código após a declaração invocadora.

As funções geralmente compõem um valor de retorno. O valor de retorno é "retornado" de volta ao "chamador":

Calcule o produto de dois números e retorne o resultado:

```
var x = minhaFuncao(4, 3);    // A função é chamada, o valor de
retorno acabará em x

function minhaFuncao(a, b) {
    return a * b;            // A função retorna o produto de a e b
}
```

O resultado em x será: 12

Por que Funções?

Você pode reutilizar o código: defina o código uma vez e use-o muitas vezes. Você pode usar o mesmo código muitas vezes com diferentes argumentos, para produzir resultados diferentes.

Converta Fahrenheit para Celsius:

```
function paraCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = paraCelsius(77);
```

O operador ( ) invoca a função

Usando o exemplo acima, paraCelsius se refere ao objeto de função, e paraCelsius ( ) refere-se ao resultado da função.

O acesso a uma função sem ( ) retornará a definição da função em vez do resultado da função:

```
function paraCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = paraCelsius;
```

Funções usadas como valores variáveis

As funções podem ser usadas da mesma maneira que você usa variáveis, em todos os tipos de fórmulas, atribuições e cálculos.

Em vez de usar uma variável para armazenar o valor de retorno de uma função:

```
var x = paraCelsius(77);
var text = "A temperature é " + x + " Celsius";
```

Você pode usar a função diretamente, como um valor variável:

```
var text = "A temperature é " + paraCelsius(77) + " Celsius";
```

Você aprenderá muito mais sobre funções mais adiante neste tutorial.

Capítulo 13

Objetos JavaScript

Objetos, propriedades e métodos da vida real

Na vida real, um carro é um objeto.

Um carro tem propriedades como peso e cor, e métodos como ligar e desligar:

Objeto	Propriedades	Métodos
	carro.nome = Fiat carro.modelo = 500 carro.peso = 850kg carro.cor = branco	carro.ligar( ) carro.dirigir( ) carro.freiar( ) carro.parar( )

Todos os carros têm as mesmas propriedades, mas os valores das propriedades diferem de carro para carro.

Todos os carros têm os mesmos métodos, mas os métodos são realizados em momentos diferentes.

Objetos JavaScript

Você já aprendeu que as variáveis JavaScript são contêineres para valores de dados.

Este código atribui um valor simples (Fiat) a uma variável chamada carro:

```
var carro = "Fiat";
```

Os objetos também são variáveis. Mas objetos podem conter muitos valores. Este código atribui muitos valores (Fiat, 500, branco) a uma variável chamada carro:

```
var carro = {tipo:"Fiat", modelo:"500", cor:"branco"};
```

Os valores são escritos como nome: pares de valores (nome e valor separados por dois pontos).

Objetos JavaScript são recipientes para valores nomeados.

Propriedades do objeto

O nome: pares de valores (em objetos JavaScript) são chamados de propriedades.

```
var pessoa = {primeiroNome:"John", ultimoNome:"Doe", idade:50,
corOlhos:"azul"};
```

Propriedades	Valores das Propriedades
primeiroNome	John
ultimoNome	Doe
Idade	50
corOlhos	azul

Métodos do objeto

Os métodos são ações que podem ser realizadas em objetos. Os métodos são armazenados em propriedades como definições de função.

Propriedades	Valores das Propriedades
primeiroNome	John
ultimoNome	Doe
Idade	50
corOlhos	Azul
nomeCompleto	function( ){return this.primeiroNome + " " + this.ultimoNome;}

Objetos JavaScript são recipientes para valores nomeados chamados propriedades ou métodos.

Definição do objeto

Você define (e cria) um objeto JavaScript com um objeto literal:

```
var pessoa = {primeiroNome:"John", ultimoNome:"Doe", idade:50,
corOlhos:"azul"};
```

Espaços e quebras de linha não são importantes. Uma definição de objeto pode abranger várias linhas:

```
var pessoa = {
    primeiroNome:"John",
    ultimoNome:"Doe",
    idade:50,
    corOlhos:"azul"
};
```

Acessando propriedades do objeto

Você pode acessar as propriedades do objeto de duas maneiras:

objectName.propertyName

Ou

objectName["propertyName"]

Exemplos:

```
pessoa.ultimoNome
```

```
pessoa["ultimoNome"]
```

Acessando Métodos de Objetos

Você acessa um método de objeto com a seguinte sintaxe:

objectName.methodName()

Exemplo:

```
nome = pessoa.nomeCompleto();
```

Se você acessar um método sem ( ), ele retornará a definição da função:

```
nome = pessoa.nomeCompleto;
```

Um método é, na verdade, uma definição de função armazenada como um valor de propriedade.

Não declare strings, números e booleanos como objetos!

Quando uma variável JavaScript é declarada com a palavra-chave "new", a variável é criada como um objeto:

```
var x = new String();        // Declara x como um objeto String
var y = new Number();        // Declara y como um objeto Number
var z = new Boolean();       // Declara z como um objeto booleano
```

Evite String, Number e objetos booleanos. Eles complicam seu código e diminuem a velocidade de execução.

Você aprenderá mais sobre objetos mais tarde neste tutorial.

Capítulo 14

Escopo de JavaScript

O escopo determina a acessibilidade (visibilidade) das variáveis.

Escopo da função JavaScript

Em JavaScript existem dois tipos de escopo:

- Escopo local
- Escopo global

O JavaScript possui o escopo da função: cada função cria um novo escopo. O escopo determina a acessibilidade (visibilidade) dessas variáveis.

As variáveis definidas dentro de uma função não são acessíveis (visíveis) de fora da função.

Variáveis de JavaScript locais

Variáveis declaradas dentro de uma função JavaScript, tornam-se LOCAIS para a função. As variáveis locais têm um alcance local: só podem ser acessadas dentro da função.

```
// código aqui não pode usar nomeCarro

function myFunction() {
    var nomeCarro = "Volvo";

    // código aqui pode usar nomeCarro

}
```

Como as variáveis locais só são reconhecidas dentro de suas funções, variáveis com o mesmo nome podem ser usadas em diferentes funções.

As variáveis locais são criadas quando uma função é iniciada e excluídas quando a função é concluída.

Variáveis globais de JavaScript

Uma variável declarada fora de uma função, torna-se GLOBAL.

Uma variável global tem alcance global: todos os scripts e funções em uma página da Web podem acessá-lo.

```
var nomeCarro = " Volvo";

// código aqui pode usar nomeCarro

function myFunction() {

    // código aqui pode usar nomeCarro

}
```

Variáveis de JavaScript

Em JavaScript, objetos e funções também são variáveis.

O escopo determina a acessibilidade de variáveis, objetos e funções de diferentes partes do código.

Automaticamente global

Se você atribuir um valor a uma variável que não tenha sido declarada, ela se tornará automaticamente uma variável GLOBAL.

Este exemplo de código declarará uma variável global nomeCarro, mesmo que o valor seja atribuído dentro de uma função.

```
myFunction();

// código aqui pode usar nomeCarro

function myFunction() {
    nomeCarro = "Volvo";
}
```

Modo estrito

Todos os navegadores modernos suportam o JavaScript em execução em "Modo estrito".

Você aprenderá mais sobre como usar o modo estrito em um capítulo posterior deste tutorial.

As variáveis globais não são criadas automaticamente em "Modo estrito".

Variáveis globais em HTML

Com JavaScript, o escopo global é o ambiente de JavaScript completo.

Em HTML, o escopo global é o objeto da janela. Todas as variáveis globais pertencem ao objeto da janela.

Atenção

NÃO crie variáveis globais, a menos que você pretenda. Suas variáveis globais (ou funções) podem substituir as variáveis (ou funções) da janela.

Qualquer função, incluindo o objeto da janela, pode substituir suas variáveis e funções globais.

O tempo de vida das variáveis no JavaScript

O tempo de vida de uma variável JavaScript começa quando é declarado. As variáveis locais são excluídas quando a função está concluída.

Em um navegador da Web, as variáveis globais são excluídas quando você fecha a janela do navegador (ou guia), mas permanece disponível para novas páginas carregadas na mesma janela.

Argumentos de função

Os argumentos de função (parâmetros) funcionam como variáveis locais dentro das funções.

Capítulo 15

Eventos de JavaScript

Os eventos HTML são "coisas" que acontecem aos elementos HTML.

Quando o JavaScript é usado em páginas HTML, o JavaScript pode "reagir" nesses eventos.

Eventos HTML

Um evento HTML pode ser algo que o navegador faz, ou algo que um usuário faz.

Aqui estão alguns exemplos de eventos HTML:

- Uma página da Web HTML acabou de carregar
- Um campo de entrada HTML foi alterado
- Foi clicado um botão HTML

Muitas vezes, quando os eventos acontecem, você pode querer fazer algo.

O JavaScript permite que você execute o código quando os eventos são detectados.

O HTML permite que os atributos do manipulador de eventos, com código JavaScript, sejam adicionados aos elementos HTML.

Com citações simples:

```
<elemento evento='algum JavaScript'>
```

Com aspas duplas:

```
<elemento evento="algum JavaScript">
```

No exemplo a seguir, um atributo onclick (com código), é adicionado a um elemento de botão:

```
<button onclick="document.getElementById('demo').innerHTML = Date()">O tempo é?</button>
```

No exemplo acima, o código JavaScript altera o conteúdo do elemento com id = "demo".

No próximo exemplo, o código altera o conteúdo de seu próprio elemento (usando this.innerHTML):

```
<button onclick="this.innerHTML = Date()">O tempo é?</button>
```

O código JavaScript muitas vezes é longo. É mais comum ver funções de chamada de atributos de eventos:

```
<button onclick="displayDate()">O tempo é?</button>
```

Eventos HTML comuns

Aqui está uma lista de alguns eventos HTML comuns:

Evento	Descrição
onchange	Altera um elemento HTML
onclick	O usuário clica em um elemento HTML
onmouseover	O usuário move o mouse sobre um elemento HTML
onmouseout	O usuário move o mouse para longe de um elemento HTML
onkeydown	O usuário pressiona uma tecla do teclado
onload	O navegador terminou de carregar a página

A lista é muito mais longa: W3Schools JavaScript Reference HTML DOM Events.

O que o JavaScript pode fazer?

Os manipuladores de eventos podem ser usados para manipular e verificar a entrada do usuário, as ações do usuário e as ações do navegador:

- Coisas que devem ser feitas toda vez que uma página carrega
- Coisas que devem ser feitas quando a página está fechada
- Ação que deve ser realizada quando um usuário clica em um botão
- Conteúdo que deve ser verificado quando um usuário insere dados
- E mais ...

Muitos métodos diferentes podem ser usados para permitir que o JavaScript funcione com os eventos:

- Os atributos de eventos HTML podem executar o código JavaScript diretamente
- Os atributos de eventos HTML podem chamar funções de JavaScript
- Você pode atribuir suas próprias funções de manipulador de eventos a elementos HTML
- Você pode impedir que eventos sejam enviados ou manipulados
- E mais ...

Você aprenderá muito mais sobre eventos e manipuladores de eventos nos capítulos HTML DOM.

Capítulo 16

Strings JavaScript

As strings JavaScript são usadas para armazenar e manipular texto.

Uma string JavaScript simplesmente armazena uma série de caracteres como "John Doe". Uma string pode ser qualquer texto dentro de aspas. Você pode usar aspas simples ou duplas:

```
var nomeCarro = "Volvo XC60";
var nomeCarro = 'Volvo XC60';
```

Você pode usar aspas dentro de uma seqüência de caracteres, desde que não correspondam às citações que cercam a seqüência de caracteres:

```
var answer = "Está tudo bem!";
var answer = "Ele se chama 'Johnny'";
var answer = 'Ele se chama "Johnny"';
```

Tamanho da String

O comprimento de uma string é encontrado utilizando a propriedade length:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Caracteres especiais

Como as strings devem ser escritas dentro de citações, o JavaScript não entenderá esta seqüência de caracteres:

```
var y = "Somos os chamados "Vikings" do norte."
```

A string será cortada em "Nós somos os chamados". A solução para evitar esse problema é usar o caractere \ escape.

O caractere de escape de barra invertida transforma caracteres especiais em caracteres de cadeia:

```
var y = " Somos os chamados \"Vikings\" do norte."
```

O caractere de escape (\) também pode ser usado para inserir outros caracteres especiais em uma string.

Estes são caracteres especiais comumente usados que podem ser inseridos em um texto com o sinal de barra invertida:

Código	Saída
\’	Aspas simples
\”	Aspas duplas
****	Barra invertida

Seis outros caracteres de escape são válidos em JavaScript:

Código	Saída
\b	Backspace
\f	Feed de formulário
\n	Quebra de linha
\r	Retorno de carro
\t	Tabulação horizontal
\v	Tabulação vertical

Os 6 caracteres de escape acima foram originalmente projetados para controlar máquinas de escrever, teletipos e máquinas de fax. Eles não fazem sentido no HTML.

Linhas de código de longa duração

Para uma melhor legibilidade, os programadores geralmente gostam de evitar linhas de código com mais de 80 caracteres.

Se uma declaração de JavaScript não se encaixa em uma linha, o melhor lugar para quebrar é depois de um operador:

```
document.getElementById("demo").innerHTML =
"Olá Dolly!";
```

Você também pode dividir uma linha de código dentro de uma string de texto com uma única barra invertida:

```
document.getElementById("demo").innerHTML = "Olá \
Dolly!";
```

O método \ não é o método preferido. Pode não ter suporte universal. Alguns navegadores não permitem espaços por trás do \ caracter.

Uma maneira mais segura de quebrar uma string, é usar a adição de seqüência de caracteres:

```
document.getElementById("demo").innerHTML = "Olá " +
"Dolly!";
```

Você não pode dividir uma linha de código com uma barra invertida:

```
document.getElementById("demo").innerHTML = \
"Olá Dolly!";
```

Strings podem ser objetos

Normalmente, as strings JavaScript são valores primitivos, criados a partir de literais:

var primeiroNome = "John";

Mas as strings de caracteres também podem ser definidas como objetos com a palavra-chave new:

var primeiroNome = new String ("John");

```
var x = "John";
var y = new String("John");

// typeof x retornará string
// typeof y retornará object
```

Não crie strings como objetos. Isso retarda a velocidade de execução. A nova palavra-chave complica o código. Isso pode produzir alguns resultados inesperados:

Ao usar o operador ==, as strings iguais são iguais:

```
var x = "John";
var y = new String("John");

// (x == y) é verdadeiro porque x e y têm valores iguais
```

Ao usar o operador ===, as strings iguais não são iguais, porque o operador === espera igualdade em ambos os tipos e valores.

```
var x = "John";
var y = new String("John");

// (x === y) é falso porque x e y têm diferentes tipos (string e
objeto)
```

Ou pior ainda. Os objetos não podem ser comparados:

```
var x = new String("John");
var y = new String("John");

// (x == y) é falso porque x e y são objetos diferentes
```

Observe a diferença entre (x == y) e (x === y). A comparação de dois objetos JavaScript sempre retornará falsa.

Capítulo 17

Métodos String no Javascript

Os métodos String ajudam você a trabalhar com strings.

Métodos e propriedades string

Valores primitivos, como "John Doe", não podem ter propriedades ou métodos (porque não são objetos).

Mas com JavaScript, métodos e propriedades também estão disponíveis para valores primitivos, porque o JavaScript trata valores primitivos como objetos ao executar métodos e propriedades.

Comprimento da string

A propriedade length retorna o comprimento de uma string:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Encontrando uma String em uma String

O método indexOf () retorna o índice de (a posição de) a primeira ocorrência de um texto especificado em uma string:

```
var str = "Por favor, localize onde" localizar "ocorre!";
var pos = str.indexOf("localizar");
```

O método lastIndexOf () retorna o índice da última ocorrência de um texto especificado em uma seqüência de caracteres:

```
var str = "Por favor, localize onde" localizar "ocorre!";
var pos = str.lastIndexOf("localizar");
```

Ambos os métodos indexOf () e lastIndexOf () retornam -1 se o texto não for encontrado.

O JavaScript conta as posições de zero. 0 é a primeira posição em uma string, 1 é a segunda, 2 é a terceira ...

Ambos os métodos aceitam um segundo parâmetro como a posição inicial para a pesquisa:

```
var str = "Por favor, localize onde" localizar "ocorre!";
var pos = str.indexOf("localizar", 15);
```

Procurando por uma String em uma String

O método search () procura uma string para um valor especificado e retorna a posição da partida:

```
var str = "Por favor, localize onde" localizar "ocorre!";
var pos = str.search("localizar");
```

Você notou?

Os dois métodos, indexOf () e search (), são iguais?

Eles aceitam os mesmos argumentos (parâmetros) e retornam o mesmo valor?

Os dois métodos NÃO são iguais. Estas são as diferenças:

- O método search () não pode ter um segundo argumento de posição inicial.
- O método indexOf () não pode ter valores de pesquisa poderosos (expressões regulares).

Você aprenderá mais sobre expressões regulares em um capítulo posterior.

Extraindo Partes de Strings

Existem 3 métodos para extrair uma parte de uma string:

- slice (começo, fim)
- substring (começo, fim)
- substr (início, comprimento)

O método slice ( )

slice () extrai uma parte de uma string e retorna a parte extraída em uma nova string.

O método leva 2 parâmetros: o índice inicial (posição) e o índice final (posição).

Este exemplo corta uma porção de uma string da posição 7 à posição 13:

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

O resultado de res será: Banana

Se um parâmetro for negativo, a posição será contada a partir do fim da string.

Este exemplo corta uma porção de uma string da posição -12 para a posição -6:

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(-12, -6);
```

O resultado de res será: Banana

Se você omitir o segundo parâmetro, o método cortará o restante da seqüência de caracteres:

```
var res = str.slice(7);
```

ou, contando desde o fim:

```
var res = str.slice(-12);
```

Posições negativas não funcionam no Internet Explorer 8 e anteriores.

O método substring ()

substring () é semelhante a slice ().

A diferença é que substring () não pode aceitar índices negativos.

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

O resultado de res será: Banana

Se você omitir o segundo parâmetro, a substring () irá cortar o restante da string.

O método substr ()

substr () é semelhante a slice ().

A diferença é que o segundo parâmetro especifica o comprimento da parte extraída.

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7, 6);
```

O resultado de res será: Banana

Se o primeiro parâmetro for negativo, a posição conta do final da string.

O segundo parâmetro não pode ser negativo, pois define o comprimento.

Se você omitir o segundo parâmetro, substr () irá cortar o restante da string.

Substituindo Conteúdo da String

O método replace () substitui um valor especificado por outro valor em uma string:

```
str = "Favor visite a Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

O método replace () não altera a string em que é chamado. Ele retorna uma nova string.

Por padrão, a função replace () substitui apenas a primeira ocorrência:

```
str = "Favor visite a Microsoft e Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

Por padrão, a função replace () é sensível a maiúsculas e minúsculas. Escrever MICROSOFT (com maiúsculas) não funcionará:

```
str = "Favor visite a Microsoft!";
var n = str.replace("MICROSOFT", "W3Schools");
```

Para substituir a insensibilidade a maiúsculas e minúsculas, use uma expressão regular com uma bandeira / i (insensível):

```
str = "Favor visite a Microsoft!";
var n = str.replace(/MICROSOFT/i, "W3Schools");
```

Observe que expressões regulares são escritas sem aspas.

Para substituir todas as correspondências, use uma expressão regular com um sinalizador / g (correspondência global):

```
str = "Favor visite a Microsoft e Microsoft!";
var n = str.replace(/Microsoft/g, "W3Schools");
```

Você aprenderá muito mais sobre expressões regulares no capítulo JavaScript Expressões regulares.

Convertendo para maiúsculas e minúsculas

Uma string é convertida em maiúsculas com toUpperCase ():

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 é text1 convertido em
maiúsculas
```

Uma string é convertida em minúsculas com toLowerCase ():

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2 é text1 convertido em
minúsculas
```

O método concat ()

concat () junta duas ou mais strings:

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

O método concat () pode ser usado em vez do operador mais. Essas duas linhas fazem o mesmo:

```
var text = "Hello" + " " + "World!";
var text = "Hello".concat(" ", "World!");
```

Todos os métodos de string retornam uma nova string. Eles não modificam a string original.

Dito formalmente: as strings são imutáveis: as strings não podem ser alteradas, apenas substituídas.

Extraindo caracteres de Strings

Existem 2 métodos seguros para extrair caracteres de seqüência de caracteres:

- charAt (posição)
- charCodeAt (posição)

O método charAt ()

O método charAt () retorna o caractere em um índice especificado (posição) em uma string:

```
var str = "HELLO WORLD";
str.charAt(0);            // retorna H
```

O método charCodeAt ()

O método charCodeAt () retorna o unicode do caractere em um índice especificado em uma string:

```
var str = "HELLO WORLD";

str.charCodeAt(0);         // retorna 72
```

Acessar um String como uma matriz é inseguro

Você pode ter visto um código como esse, acessando uma string como uma matriz:

```
var str = "HELLO WORLD";

str[0];                    // retorna H
```

Isso é inseguro e imprevisível:

- Não funciona em todos os navegadores (não no IE5, IE6, IE7)
- Isso faz com que as strings pareçam arrays (mas não são)
- str [0] = "H" não dá um erro (mas não funciona)

Se você quiser ler uma string como uma matriz, converta-a em uma matriz primeiro.

Convertendo uma String em uma matriz

Uma string pode ser convertida em uma matriz com o método split ():

```
var txt = "a,b,c,d,e";   // String
txt.split(",");          // Split com vírgulas
txt.split(" ");          // Split com espaços
txt.split("|");          // Split com pipe
```

Se o separador for omitido, a matriz retornada conterá toda a cadeia no índice [0].

Se o separador for "", a matriz retornada será uma matriz de caracteres únicos:

```
var txt = "Hello";       // String
txt.split("");           // Split em caracteres únicos
```

Referência de String Completa

Para uma referência completa, acesse a Referência de Cadeia de JavaScript Completa.

A referência contém descrições e exemplos de todas as propriedades e métodos de cadeia.

Capítulo 18

Números de JavaScript

O JavaScript tem apenas um tipo de número. Os números podem ser escritos com ou sem decimais.

```
var x = 3.14;    // Um número com decimais
var y = 3;       // Um número sem decimais
```

Os números extra grandes ou extra pequenos podem ser escritos com notação científica (expoente):

```
var x = 123e5;    // 12300000
var y = 123e-5;   // 0.00123
```

Os números de JavaScript são sempre um ponto flutuante de 64 bits

Ao contrário de muitas outras linguagens de programação, o JavaScript não define diferentes tipos de números, como números inteiros, curto, longo, ponto flutuante etc.

Os números de JavaScript são sempre armazenados como números de ponto flutuante de dupla precisão, seguindo o padrão internacional IEEE 754.

Este formato armazena números em 64 bits, onde o número (a fração) é armazenado nos bits 0 a 51, o expoente nos bits 52 a 62 e o bit de registro 63:

Valor (aka Fraction / Mantissa)	Expoente	Sinal
52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

Precisão

Inteiros (números sem um período ou notação de expoente) são precisos até 15 dígitos.

```
var x = 999999999999999;   // x será 999999999999999
var y = 9999999999999999;  // y será 10000000000000000
```

O número máximo de decimais é de 17, mas a aritmética de ponto flutuante nem sempre é 100% precisa:

```
var x = 0.2 + 0.1;         // x será 0.30000000000000004
```

Para resolver o problema acima, isto ajuda a multiplicar e dividir:

```
var x = (0.2 * 10 + 0.1 * 10) / 10;       // x será 0.3
```

Adicionando Números e Strings

ATENÇÃO !!

O JavaScript usa o operador + para adição e concatenação.

Os números são adicionados. As cordas são concatenadas.

```
var x = 10;
var y = 20;
var z = x + y;            // z será 30 (um número)
```

Se você adicionar duas strings, o resultado será uma concatenação de string:

```
var x = "10";
var y = "20";
var z = x + y;            // z será 1020 (uma string)
```

Se você adicionar um número e uma string, o resultado será uma concatenação de string:

```
var x = 10;
var y = "20";
var z = x + y;            // z será 1020 (uma string)
```

Se você adicionar uma string e um número, o resultado será uma concatenação de string:

```
var x = "10";
var y = 20;
var z = x + y;            // z será 1020 (uma string)
```

Um erro comum é esperar que este resultado seja 30:

```
var x = 10;
var y = 20;
var z = "O resultado é: " + x + y;
```

Um erro comum é esperar que esse resultado seja 102030:

```
var x = 10;
var y = 20;
var z = "30";
var resultado = x + y + z;
```

O compilador JavaScript funciona da esquerda para a direita.

Primeiro 10 + 20 é adicionado porque x e y são ambos os números.

Então 30 + "30" é concatenado porque z é uma string.

Strings Numéricas

Strings de caracteres JavaScript podem ter conteúdo numérico:

```
var x = 100;          // x é um número
var y = "100";        // y é uma string
```

O JavaScript tentará converter strings em números em todas as operações numéricas:

Isso funcionará:

```
var x = "100";
var y = "10";
var z = x / y;        // z será 10
```

Isso também funcionará:

```
var x = "100";
var y = "10";
var z = x * y;        // z será 1000
```

E isso vai funcionar:

```
var x = "100";
var y = "10";
var z = x - y;        // z será 90
```

Mas isso não vai funcionar:

```
var x = "100";
var y = "10";
var z = x + y;        // z will não será 110 (Será 10010)
```

No último exemplo, o JavaScript usa o operador + para concatenar as strings.

NaN - Não é um número

NaN é uma palavra reservada JavaScript que indica que um número não é um número legal.

Tentando fazer aritmética com uma seqüência não-numérica resultará em NaN (Não é um Número):

```
var x = 100 / "Apple";  // x será NaN (Não é um número)
```

No entanto, se a string contiver um valor numérico, o resultado será um número:

```
var x = 100 / "10";     // x será 10
```

Você pode usar a função JavaScript global isNaN () para descobrir se um valor é um número:

```
var x = 100 / "Apple";
isNaN(x);               // retorna true porque x Não é um número
```

Cuidado com a NaN. Se você usa NaN em uma operação matemática, o resultado também será NaN:

```
var x = NaN;
var y = 5;
var z = x + y;          // z será NaN
```

Ou o resultado pode ser uma concatenação:

```
var x = NaN;
var y = "5";
var z = x + y;          // z será NaN5
```

NaN é um número: typeof de NaN retorna número:

```
typeof NaN;             // retorna "número"
```

Infinito

Infinity (ou -Infinity) é o valor que o JavaScript retornará se você calcular um número fora do maior número possível.

```
var meuNumero = 2;
while (meuNumero != Infinity) {          // Execute até Infinity
    meuNumero = meuNumero * meuNumero;
}
```

Divisão de 0 (zero) também gera Infinity:

```
var x =  2 / 0;          // x será Infinity
var y = -2 / 0;          // y será -Infinity
```

Infinity é um número: typeof Infinity retorna número.

```
typeof Infinity;        // retorna "número"
```

Hexadecimal

O JavaScript interpreta as constantes numéricas como hexadecimais se forem precedidas de 0x.

```
var x = 0xFF;           // x será 255
```

Nunca escreva um número com um zero inicial (como 07).

Algumas versões de JavaScript interpretam os números como octal se eles estiverem escritos com um zero inicial.

Por padrão, o JavaScript exibe números como base de 10 decimais.

Mas você pode usar o método toString () para produzir números como base 16 (hexadecimal), base 8 (octal) ou base 2 (binário).

```
var meuNumero = 128;
meuNumero.toString(16);  // retorna 80
meuNumero.toString(8);   // retorna 200
meuNumero.toString(2);   // retorna 10000000
```

Números podem ser objetos

Normalmente, os números de JavaScript são valores primitivos criados a partir de literais:

var x = 123;

Mas os números também podem ser definidos como objetos com a palavra-chave new:

var y = novo número (123);

```
var x = 123;
var y = new Number(123);

// typeof x retorna number
// typeof y retorna object
```

Não crie objetos numéricos. Isso retarda a velocidade de execução.

A nova palavra-chave complica o código. Isso pode produzir alguns resultados inesperados:

Ao usar o operador ==, os números iguais são iguais:

```
var x = 500;
var y = new Number(500);

// (x == y) é true porque x e y tem valores iguais
```

Ao usar o operador ===, os números iguais não são iguais, porque o operador === espera igualdade em ambos os tipos e valores.

```
var x = 500;
var y = new Number(500);

// (x === y) é false porque x é y tem tipos diferentes
```

Ou pior ainda. Os objetos não podem ser comparados:

```
var x = new Number(500);
var y = new Number(500);

// (x == y) é false porque objetos não podem ser comparados
```

Observe a diferença entre (x == y) e (x === y).

A comparação de dois objetos JavaScript sempre retornará falsa.

Capítulo 19

Métodos Number no JavaScript

Os métodos numéricos ajudam você a trabalhar com números.

Métodos e Propriedades number

Valores primitivos (como 3.14 ou 2014), não podem ter propriedades e métodos (porque não são objetos).

Mas com JavaScript, métodos e propriedades também estão disponíveis para valores primitivos, porque o JavaScript trata valores primitivos como objetos ao executar métodos e propriedades.

O método toString ()

toString () retorna um número como uma string.

Todos os métodos de número podem ser usados em qualquer tipo de números (literais, variáveis ou expressões):

```
var x = 123;
x.toString();            // retorna 123 de variable x
(123).toString();        // retorna 123 de literal 123
(100 + 23).toString();   // retorna 123 da expressão 100 + 23
```

O método toExponential ()

toExponential () retorna uma string, com um número arredondado e escrito usando a notação exponencial.

Um parâmetro define o número de caracteres atrás do ponto decimal:

```
var x = 9.656;
x.toExponential(2);     // retorna 9.66e+0
x.toExponential(4);     // retorna 9.6560e+0
x.toExponential(6);     // retorna 9.656000e+0
```

O parâmetro é opcional. Se você não especificá-lo, o JavaScript não contornará o número.

O método toFixed ()

toFixed () retorna uma string, com o número escrito com um número especificado de decimais:

```
var x = 9.656;
x.toFixed(0);           // retorna 10
x.toFixed(2);           // retorna 9.66
x.toFixed(4);           // retorna 9.6560
x.toFixed(6);           // retorna 9.656000
```

toFixed (2) é perfeito para trabalhar com valores monetários.

O método toPrecision ()

toPrecision () retorna uma string, com um número escrito com um comprimento especificado:

```
var x = 9.656;
x.toPrecision();        // retorna 9.656
x.toPrecision(2);       // retorna 9.7
x.toPrecision(4);       // retorna 9.656
x.toPrecision(6);       // retorna 9.65600
```

O método valueOf ()

valueOf () retorna um número como um número.

```
var x = 123;
x.valueOf();            // retorna 123 de variable x
(123).valueOf();        // retorna 123 de literal 123
(100 + 23).valueOf();   // retorna 123 da expressão 100 + 23
```

Em JavaScript, um número pode ser um valor primitivo (typeof = número) ou um objeto (typeof = objeto).

O método valueOf () é usado internamente em JavaScript para converter objetos numéricos em valores primitivos.

Não há motivo para usá-lo em seu código.

Todos os tipos de dados JavaScript têm um método valueOf () e toString ().

Conversão de variáveis em números

Existem 3 métodos de JavaScript que podem ser usados para converter variáveis em números:

- O método Number ()
- O método parseInt ()
- O método parseFloat ()

Esses métodos não são métodos de número, mas métodos globais de JavaScript.

Métodos globais

Os métodos globais de JavaScript podem ser usados em todos os tipos de dados JavaScript.

Estes são os métodos mais relevantes, quando se trabalha com números:

Método	Descrição
Number( )	Retorna um número, convertido de seu argumento.
parseFloat( )	Analisa seu argumento e retorna um número de ponto flutuante
parseInt( )	Analisa seu argumento e retorna um inteiro

O Método Number ()

Number () pode ser usado para converter variáveis JavaScript em números:

```
Number(true);           // retorna 1
Number(false);          // retorna 0
Number("10");           // retorna 10
Number("  10");         // retorna 10
Number("10  ");         // retorna 10
Number("10 20");        // retorna NaN
Number("John");         // retorna NaN
```

Se o número não puder ser convertido, NaN (Não é um Número) é retornado.

O método Number () usado nas datas

Number () também pode converter uma data para um número:

```
Number(new Date(2017-09-30));    // retorna 1506729600000
```

O método Number () acima retorna o número de milissegundos desde 1.1.1970.

O método parseInt ()

parseInt () analisa uma string e retorna um número inteiro. Espaços são permitidos. Apenas o primeiro número é retornado:

```
parseInt("10");          // retorna 10
parseInt("10.33");       // retorna 10
parseInt("10 20 30");    // retorna 10
parseInt("10 years");    // retorna 10
parseInt("years 10");    // retorna NaN
```

Se o número não puder ser convertido, NaN (Não é um Número) é retornado.

O método parseFloat ()

ParseFloat () analisa uma string e retorna um número. Espaços são permitidos. Apenas o primeiro número é retornado:

```
parseFloat("10");        // retorna 10
parseFloat("10.33");     // retorna 10.33
parseFloat("10 20 30");  // retorna 10
parseFloat("10 years");  // retorna 10
parseFloat("years 10");  // retorna NaN
```

Se o número não puder ser convertido, NaN (Não é um Número) é retornado.

Propriedades de Number

Propriedade	Descrição
MAX_VALUE	Retorna o maior número possível em JavaScript
MIN_VALUE	Retorna o menor número possível em JavaScript
NEGATIVE_INFINITY	Representa o infinito negativo (retornado no estouro)
NaN	representa um valor "Não-numero"
POSITIVE_INFINITY	Representa o infinito (retornado no estouro)

Exemplo:

```
var x = Number.MAX_VALUE;
```

As propriedades de Number pertencem ao wrapper de objeto Number do JavaScript chamado Number.

Essas propriedades só podem ser acessadas como Number.MAX_VALUE.

Usando meuNumero.MAX_VALUE, onde meuNumero é uma variável, expressão ou valor, retornará indefinido:

```
var x = 6;
var y = x.MAX_VALUE;    // y se torna undefined
```

Referência de número de JavaScript completa

Para obter uma referência completa, acesse a Referência do número de JavaScript completo.

A referência contém descrições e exemplos de todas as propriedades e métodos do Número.

Capítulo 20

Objeto Javascript Math

O objeto Math JavaScript permite que você execute tarefas matemáticas em números.

Exemplo:

```
Math.PI;            // retorna 3.141592653589793
```

Math.round ()

Math.round (x) retorna o valor de x arredondado para o inteiro mais próximo:

```
Math.round(4.7);    // retorna 5
Math.round(4.4);    // retorna 4
```

Math.pow ()

Math.pow (x, y) retorna o valor de x para a potência de y:

```
Math.pow(8, 2);       // retorna 64
```

Math.sqrt ()

Math.sqrt (x) retorna a raiz quadrada de x:

```
Math.sqrt(64);       // retorna 8
```

Math.abs ()

Math.abs (x) retorna o valor absoluto (positivo) de x:

```
Math.abs(-4.7);      // retorna 4.7
```

Math.ceil ()

Math.ceil (x) retorna o valor de x arredondado para o inteiro acima mais próximo:

```
Math.ceil(4.4);      // retorna 5
```

Math.floor ()

Math.floor (x) retorna o valor de x arredondado para o inteiro abaixo mais próximo:

```
Math.floor(4.7);     // retorna 4
```

Math.sin ()

Math.sin (x) retorna o seno (um valor entre -1 e 1) do ângulo x (dado em radianos).

Se você quiser usar graus em vez de radianos, você deve converter graus em radianos:

Ângulo em radianos = Ângulo em graus x PI / 180.

```
Math.sin(90 * Math.PI / 180);     // retorna 1 (o seno de 90 graus)
```

Math.cos ()

Math.cos (x) retorna o coseno (um valor entre -1 e 1) do ângulo x (dado em radianos).

Se você quiser usar graus em vez de radianos, você deve converter graus em radianos:

Ângulo em radianos = Ângulo em graus x PI / 180.

```
Math.cos(0 * Math.PI / 180);     // retorna 1 (o cos de 0 graus)
```

Math.min () e Math.max ()

Math.min () e Math.max () podem ser usados para encontrar o valor mais baixo ou mais alto em uma lista de argumentos:

```
Math.min(0, 150, 30, 20, -8, -200);  // retorna -200
```

```
Math.max(0, 150, 30, 20, -8, -200);  // retorna 150
```

Math.random ()

Math.random () retorna um número aleatório entre 0 (inclusive) e 1 (exclusivo):

```
Math.random();      // retorna um número randômico
```

Você aprenderá mais sobre Math.random () no próximo capítulo deste tutorial.

Propriedades de Math (Constantes)

O JavaScript fornece 8 constantes matemáticas que podem ser acessadas com o objeto Math:

```
Math.E        // retorna o número de Euler
Math.PI       // retorna PI
Math.SQRT2    // retorna a raiz quadrada de 2
Math.SQRT1_2  // retorna a raiz quadrada de 1/2
Math.LN2      // retorna o logaritmo natural de 2
Math.LN10     // retorna o logaritmo natural de 10
Math.LOG2E    // retorna o logaritmo da base 2 de E
Math.LOG10E   // retorna o logaritmo da base 10 de E
```

Construtor de math

Ao contrário de outros objetos globais, o objeto Math não possui nenhum construtor. Métodos e propriedades são estáticos.

Todos os métodos e propriedades (constantes) podem ser usados sem criar um objeto Math primeiro.

Métodos de objetos Math

Método	Descrição
Abs(x)	Retorna o valor absoluto de x
Acos(x)	Retorna a arccoseno de x, em radianos
Asin(x)	Retorna a arcseno de x, em radianos
Atan(x)	Retorna o arctangente de x como um valor numérico entre -PI / 2 e PI / 2 radianos
Atan2(y,x)	Retorna o arctangente do quociente de seus argumentos
Ceil(x)	Retorna o valor de x arredondado para o inteiro acima mais próximo
Cos(x)	Retorna o coseno de x (x está em radianos)
Exp(x)	Retorna o valor de Ex
Floor(x)	Retorna o valor de x arredondado para o inteiro abaixo mais próximo
Log(x)	Retorna o logaritmo natural (base E) de x
Max(x,y,z,...,n)	Retorna o número com o valor mais alto
Min(x,y,z,...,n)	Retorna o número com o menor valor
Pow(x,y)	Retorna o valor de x para a potência de y
Random()	Retorna um número aleatório entre 0 e 1
Round(x)	Retorna o valor de x arredondado para o número inteiro mais próximo
Sin(x)	Retorna o seno de x (x está em radianos)
Sqrt(x)	Retorna a raiz quadrada de x
Tan(x)	Retorna a tangente de um ângulo

Referência matemática completa

Para uma referência completa, acesse nossa referência de objeto matemática completa.

A referência contém descrições e exemplos de todas as propriedades e métodos de Matemática.

Capítulo 21

JavaScript Random

Math.random ()

Math.random () retorna um número aleatório entre 0 (inclusive) e 1 (exclusivo):

```
Math.random();              // retorna um número randômico
```

Math.random () sempre retorna um número inferior a 1.

Inteiros aleatórios do JavaScript

Math.random () usado com Math.floor () pode ser usado para retornar inteiros aleatórios.

Exemplo:

```
Math.floor(Math.random() * 10);     // retorna um número entre 0 e 9
```

Ou

```
Math.floor(Math.random() * 11);      // retorna um número entre 0 e 10
```

Ou

```
Math.floor(Math.random() * 100);     // retorna um número entre 0 e 99
```

Ou

```
Math.floor(Math.random() * 101);     // retorna um número entre 0 e
100
```

Ou

```
Math.floor(Math.random() * 10) + 1;  // retorna um número entre 1 e 10
```

Ou

```
Math.floor(Math.random() * 100) + 1;  // retorna um número entre 1 e
100
```

Uma função aleatória adequada

Como você pode ver nos exemplos acima, pode ser uma boa idéia criar uma função aleatória apropriada para usar para todos os fins inteiros aleatórios.

Esta função JavaScript sempre retorna um número aleatório entre min (incluído) e máximo (excluído):

```
function getRndInteger(min, max) {
    return Math.floor(Math.random() * (max - min) ) + min;
}
```

Esta função JavaScript sempre retorna um número aleatório entre min e max (ambos incluídos):

```
function getRndInteger(min, max) {
    return Math.floor(Math.random() * (max - min + 1) ) + min;
}
```

Capítulo 22

Datas no JavaScript

O objeto Data permite trabalhar com datas (anos, meses, dias, horas, minutos, segundos e milissegundos)

Formatos de data de JavaScript

Uma data de JavaScript pode ser escrita como uma string:

Mon Jan 08 2018 00:34:17 GMT-0200 (Horário brasileiro de verão)

Ou como um número:

1515378483363

Datas escritas como números, especificam o número de milissegundos desde 1 de janeiro de 1970, 00:00:00.

Exibindo Datas

Neste tutorial usamos um script para exibir datas dentro de um elemento <p> com id = "demo":

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Date();
</script>
```

O script acima diz: atribua o valor de Data () ao conteúdo (innerHTML) do elemento com id = "demo".

Você aprenderá a exibir uma data, em um formato mais legível, na parte inferior desta página.

Criando objetos Date

O objeto Data nos permite trabalhar com datas.

Uma data consiste em um ano, um mês, um dia, uma hora, um minuto, um segundo e milissegundos.

Os objetos de data são criados com o construtor new Date ().

Existem 4 formas de iniciar uma data:

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Usando new Date (), cria um novo objeto de data com a data e hora atuais:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

Usando new Date (dateString), cria um novo objeto de data a partir da data e hora especificadas:

```
<script>
var d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>
```

As strings de data válidas (formatos de data) são descritas no próximo capítulo.

Usando new Date (number), cria um novo objeto de data como tempo zero mais o número.

O horário zero é 01 de janeiro de 1970 00:00:00 UTC. O número é especificado em milissegundos.

100000000000 milissegundos de 1 de janeiro de 1970, é aproximadamente 3 de março de 1973:

```
<script>
var d = new Date(100000000000);
document.getElementById("demo").innerHTML = d;
</script>
```

As datas JavaScript são calculadas em milissegundos a partir de 01 de janeiro de 1970 00:00:00 Hora Universal (UTC). Um dia (24 horas) contém 86,400,000 milissegundos.

Usando new Date(7 números), cria um novo objeto de data com a data e hora especificadas:

Os 7 números especificam o ano, mês, dia, hora, minuto, segundo e milissegundo, naquela ordem:

```
<script>
var d = new Date(99, 5, 24, 11, 33, 30, 0);
document.getElementById("demo").innerHTML = d;
</script>
```

As variantes do exemplo acima nos permitem omitir alguns dos últimos 4 parâmetros:

```
<script>
var d = new Date(99, 5, 24);
document.getElementById("demo").innerHTML = d;
</script>
```

O JavaScript conta dos meses de 0 a 11. Janeiro é 0. Dezembro é 11.

Métodos Date

Quando um objeto Date é criado, uma série de métodos permitem que você opere sobre ele.

Os métodos de data permitem que você obtenha e configure o ano, o mês, o dia, a hora, o minuto, o segundo e o milissegundo dos objetos, usando horário local ou UTC (universal ou GMT).

Os métodos de data são abordados em um capítulo posterior.

Exibindo Datas

Quando você exibe um objeto de data em HTML, ele é convertido automaticamente em uma string, com o método toString ().

```
<p id="demo"></p>

<script>
d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

É o mesmo que:

```
<p id="demo"></p>

<script>
d = new Date();
document.getElementById("demo").innerHTML = d.toString();
</script>
```

O método toUTCString () converte uma data em uma string UTC (um padrão de exibição de data).

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toUTCString();
</script>
```

O método toDateString () converte uma data em um formato mais legível:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toDateString();
</script>
```

Os objetos de data são estáticos. O tempo do computador está marcando, mas os objetos de data, uma vez criados, não são.

Fusos horários

Ao definir uma data, sem especificar o fuso horário, o JavaScript usará o fuso horário do navegador.

Ao obter uma data, sem especificar o fuso horário, o resultado é convertido no fuso horário do navegador.

Em outras palavras: se uma data / hora for criada em GMT (Horário Médio de Greenwich), a data / hora será convertida em CDT (Hora Central de US Daylight) se um usuário navegar no centro dos EUA.

Leia mais sobre fuso horário nos próximos capítulos.

Capítulo 23

Formatos de data no JavaScript

Existem geralmente 4 tipos de formatos de entrada de data de JavaScript:

Tipo	Exemplo
ISO Date	"2015-03-25" (O Padrão Internacional)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" ou "25 Mar 2015"
Full Date	" Quarta-feira março 25 2015"

O formato ISO segue um padrão rígido em JavaScript.

Os outros formatos não estão tão bem definidos e podem ser específicos do navegador.

Saída de Data no JavaScript

Independentemente do formato de entrada, o JavaScript irá (por padrão) datas de saída no formato de seqüência de caracteres de texto completo:

Ter 24 mar 2015 21:00:00 GMT-0300 (Hora oficial do Brasil)

Datas ISO no JavaScript

O ISO 8601 é o padrão internacional para a representação de datas e horários.

A sintaxe ISO 8601 (AAAA-MM-DD) também é o formato de data de JavaScript preferido:

Exemplo (data completa)

```
var d = new Date("2015-03-25");
```

A data computada será relativa ao seu fuso horário.

Dependendo do seu fuso horário, o resultado acima variará entre 24 de março e 25 de março.

Datas ISO (ano e mês)

As datas ISO podem ser escritas sem especificar o dia (AAAA-MM):

```
var d = new Date("2015-03");
```

Os fusos horários variam o resultado acima entre 28 de fevereiro e 01 de março.

Datas ISO (Somente Ano)

As datas ISO podem ser escritas sem mês e dia (AAAA):

```
var d = new Date("2015");
```

Os fusos horários variam o resultado acima entre 31 de dezembro de 2014 e 01 de janeiro de 2015.

Datas ISO (Data-Hora)

As datas ISO podem ser escritas com horas, minutos e segundos adicionados (AAAA-MM-DDTHH: MM: SSZ):

```
var d = new Date("2015-03-25T12:00:00Z");
```

A data e a hora são separadas por um T.

A hora UTC é definida com uma letra maiúscula Z.

Se você deseja modificar o tempo relativo a UTC, remova o Z e adicione + HH: MM ou -HH: MM em vez disso:

```
var d = new Date("2015-03-25T12:00:00-06:30");
```

UTC (Universal Time Coordinated) é o mesmo que GMT (Greenwich Mean Time).

Omitindo T ou Z em uma seqüência de data-hora pode dar resultados diferentes no navegador diferente.

Fusos horários

Ao definir uma data, sem especificar o fuso horário, o JavaScript usará o fuso horário do navegador.

Ao obter uma data, sem especificar o fuso horário, o resultado é convertido no fuso horário do navegador.

Em outras palavras: se uma data / hora for criada em GMT (Horário Médio de Greenwich), a data / hora será convertida em CDT (Hora Central de US Daylight) se um usuário navegar no centro dos EUA.

Datas curtas no JavaScript

As datas curtas são escritas com uma sintaxe "MM / DD / AAAA" como esta:

```
var d = new Date("03/25/2015");
```

AVISOS!

Em alguns navegadores, meses ou dias sem zeros iniciais podem produzir um erro:

```
var d = new Date("2015-3-25");
```

O comportamento de "YYYY / MM / DD" é indefinido.

Alguns navegadores tentarão adivinhar o formato. Alguns irão retornar a NaN.

```
var d = new Date("2015/03/25");
```

O comportamento de "DD-MM-AAAA" também é indefinido.

Alguns navegadores tentarão adivinhar o formato. Alguns irão retornar a NaN.

```
var d = new Date("25-03-2015");
```

Datas longas no JavaScript

As datas longas são gravadas com maior freqüência com uma sintaxe "MMM DD YYYY" como essa:

```
var d = new Date("Mar 25 2015");
```

Mês e dia podem estar em qualquer ordem:

```
var d = new Date("25 Mar 2015");
```

E, o mês pode ser escrito na íntegra (janeiro) ou abreviado (janeiro):

```
var d = new Date("January 25 2015");
```

```
var d = new Date("Jan 25 2015");
```

Aspas são ignoradas. Os nomes são insensíveis às maiúsculas e minúsculas:

```
var d = new Date("JANUARY, 25, 2015");
```

Datas completas no JavaScript

O JavaScript aceita strings de data em "formato JavaScript completo":

```
var d = new Date("Wed Mar 25 2015 09:56:24 GMT+0100 (W. Europe
Standard Time)");
```

O JavaScript ignorará os erros no nome do dia e nos parênteses de tempo:

```
var d = new Date("Fri Mar 25 2015 09:56:24 GMT+0100 (Tokyo Time)");
```

Capítulo 24

Métodos Date no Javascript

Os métodos de data permitem que você obtenha e defina valores de data (anos, meses, dias, horas, minutos, segundos, milissegundos)

Métodos de obtenção de data

Obter métodos são usados para obter uma parte de uma data. Aqui estão os mais comuns (em ordem alfabética):

Método	Descrição
getDate()	Obter o dia como um número (1-31)
getDay()	Obter o dia da semana como um número (0-6)
getFullYear()	Obter o ano de quatro dígitos (aaaa)
getHours()	Obter a hora (0-23)
getMilliseconds()	Obtenha os milissegundos (0-999)
getMinutes()	Obter os minutos (0-59)
getMonth()	Obter o mês (0-11)
getSeconds()	Obter os segundos (0-59)
getTime()	Obter o tempo (milissegundos desde 1 de janeiro de 1970)

O método getTime ()

getTime () retorna o número de milissegundos desde 1 de janeiro de 1970:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getTime();
</script>
```

O método getFullYear ()

getFullYear () retorna o ano de uma data como um número de quatro dígitos:

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getFullYear();
</script>
```

O método getDay ()

getDay () retorna o dia da semana como um número (0-6):

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.getDay();
</script>
```

Em JavaScript, o primeiro dia da semana (0) significa "Domingo", mesmo que alguns países do mundo considerem que o primeiro dia da semana seja "segunda-feira"

Você pode usar um array de nomes e getDay () para retornar o dia da semana como um nome:

```
<script>
var d = new Date();
var days =
["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday
"];
document.getElementById("demo").innerHTML = days[d.getDay()];
</script>
```

Métodos de definição de data

Os métodos de configuração são usados para definir uma parte de uma data. Aqui estão os mais comuns (em ordem alfabética):

Método	Descrição
setDate()	Defina o dia como um número (1-31)
setFullYear()	Defina o ano (opcionalmente mês e dia)
setHours()	Defina a hora (0-23)
setMilliseconds()	Defina os milissegundos (0-999)
setMinutes()	Defina os minutos (0-59)
setMonth()	Defina o mês (0-11)
setSeconds()	Defina os segundos (0-59)
setTime()	Defina o tempo (milissegundos desde 1 de janeiro de 1970)

O método setFullYear ()

setFullYear () define um objeto de data para uma data específica. Neste exemplo, até 14 de janeiro de 2020:

```
<script>
var d = new Date();
d.setFullYear(2020, 0, 14);
document.getElementById("demo").innerHTML = d;
</script>
```

O método setDate ()

setDate () define o dia do mês (1-31):

```
<script>
var d = new Date();
d.setDate(20);
document.getElementById("demo").innerHTML = d;
</script>
```

O método setDate () também pode ser usado para adicionar dias a uma data:

```
<script>
var d = new Date();
d.setDate(d.getDate() + 50);
document.getElementById("demo").innerHTML = d;
</script>
```

Se adicionar dias, muda o mês ou o ano, as alterações são tratadas automaticamente pelo objeto Date.

Entrada de data - Parâmetros de Data

Se você tiver uma seqüência de data válida, você pode usar o método Date.parse () para convertê-lo em milissegundos.

Date.parse () retorna o número de milissegundos entre a data eo 1 de janeiro de 1970:

```
<script>
var msec = Date.parse("March 21, 2012");
document.getElementById("demo").innerHTML = msec;
</script>
```

Você pode então usar o número de milissegundos para convertê-lo em um objeto de data:

```
<script>
var msec = Date.parse("March 21, 2012");
var d = new Date(msec);
document.getElementById("demo").innerHTML = d;
</script>
```

Compare datas

As datas podem ser facilmente comparadas.

O exemplo a seguir compara a data de hoje com 14 de janeiro de 2100:

```
var hoje, algumdia, texto;
hoje = new Date();
algumdia = new Date();
algumdia.setFullYear(2100, 0, 14);

if (algumdia > hoje) {
    texto = "Hoje é antes de 14 de janeiro de 2100.";
} else {
    texto = "Hoje é depois de 14 de janeiro de 2100.";
}
document.getElementById("demo").innerHTML = texto;
```

O JavaScript conta dos meses de 0 a 11. Janeiro é 0. Dezembro é 11.

Métodos de data UTC

Os métodos de data UTC são usados para atualizar datas UTC (datas do fuso horário universal):

Método	Descrição
getUTCDate()	Igual a getDate (), mas retorna a data UTC
getUTCDay()	Mesmo que getDay (), mas retorna o dia UTC
getUTCFullYear()	Igual ao getFullYear (), mas retorna o ano UTC
getUTCHours()	Igual a getHours (), mas retorna a hora UTC
getUTCMilliseconds()	Igual a getMilliseconds (), mas retorna os milissegundos do UTC
getUTCMinutes()	Igual a getMinutes (), mas retorna os minutos UTC
getUTCMonth()	O mesmo que getMonth (), mas retorna o mês UTC
getUTCSeconds()	Igual a getSeconds (), mas retorna os segundos UTC

Referência de data de JavaScript completa

Para uma referência completa, acesse nossa Referência de data de JavaScript completa.

A referência contém descrições e exemplos de todas as propriedades e métodos da Data.

Capítulo 25

JavaScript JSON (Javascript Object Notation)

O JSON é um formato para armazenar e transportar dados.

JSON é usado frequentemente quando os dados são enviados de um servidor para uma página da web.

O que é JSON?

- JSON significa JavaScript Object Notation
- JSON é um formato leve de intercâmbio de dados
- JSON é independente da linguagem *
- JSON é "auto-descrevente" e fácil de entender

* A sintaxe JSON é derivada da sintaxe de notação de objeto JavaScript, mas o formato JSON é apenas texto. O código para ler e gerar dados JSON pode ser escrito em qualquer linguagem de programação.

JSON Exemplo

Esta sintaxe JSON define um objeto de funcionários: uma matriz de 3 registros de funcionários (objetos):

```
{
"funcionarios":[
    {"primeiroNome":"John", "ultimoNome":"Doe"},
    {"primeiroNome":"Anna", "ultimoNome":"Smith"},
    {"primeiroNome":"Peter", "ultimoNome":"Jones"}
]
}
```

O formato JSON avalia os objetos JavaScript

O formato JSON é sintaticamente idêntico ao código para criar objetos JavaScript.

Por causa dessa similaridade, um programa de JavaScript pode facilmente converter dados JSON em objetos de JavaScript nativos.

Regras de sintaxe JSON

- Os dados estão nos pares nome / valor
- Os dados são separados por vírgulas
- Chaves servem para enlaçar objetos
- Os colchetes possuem arrays

Dados JSON - Um Nome e um Valor

Os dados JSON são escritos como pares nome / valor, assim como propriedades do objeto JavaScript.

Um par de nome / valor consiste em um nome de campo (com aspas duplas), seguido por dois pontos, seguido de um valor:

```
"primeiroNome":"John"
```

Os nomes JSON requerem aspas duplas. Os nomes de JavaScript não.

Objetos JSON

Os objetos JSON são escritos dentro de chaves.

Assim como em JavaScript, os objetos podem conter vários pares nome / valor:

```
{"primeiroNome":"John", "ultimoNome":"Doe"}
```

JSON Arrays

Arrays JSON são escritos dentro de colchetes.

Assim como em JavaScript, uma matriz pode conter objetos:

```
"funcionarios":[
    {"primeiroNome":"John", "ultimoNome":"Doe"},
    {"primeiroNome":"Anna", "ultimoNome":"Smith"},
    {"primeiroNome":"Peter", "ultimoNome":"Jones"}
]
```

No exemplo acima, o objeto "funcionários" é uma matriz. Contém três objetos.

Cada objeto é um registro de uma pessoa (com um primeiro nome e um sobrenome).

Convertendo um texto JSON para um objeto JavaScript

Um uso comum do JSON é ler dados de um servidor web e exibir os dados em uma página da Web.

Por simplicidade, isso pode ser demonstrado usando uma string como entrada.

Primeiro, crie uma string JavaScript contendo a sintaxe JSON:

```
var texto = '{ "funcionarios" : [' +
'{ "primeiroNome":"John" , "ultimoNome":"Doe" },' +
'{ "primeiroNome":"Anna" , "ultimoNome":"Smith" },' +
'{ "primeiroNome":"Peter" , "ultimoNome":"Jones" } ]}';
```

Em seguida, use a função embutida JavaScript JSON.parse () para converter a string em um objeto JavaScript:

```
var obj = JSON.parse(texto);
```

Finalmente, use o novo objeto JavaScript em sua página:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
obj.funcionarios[1].primeiroNome + " " +
obj.funcionarios[1].ultimoNome;
</script>
```