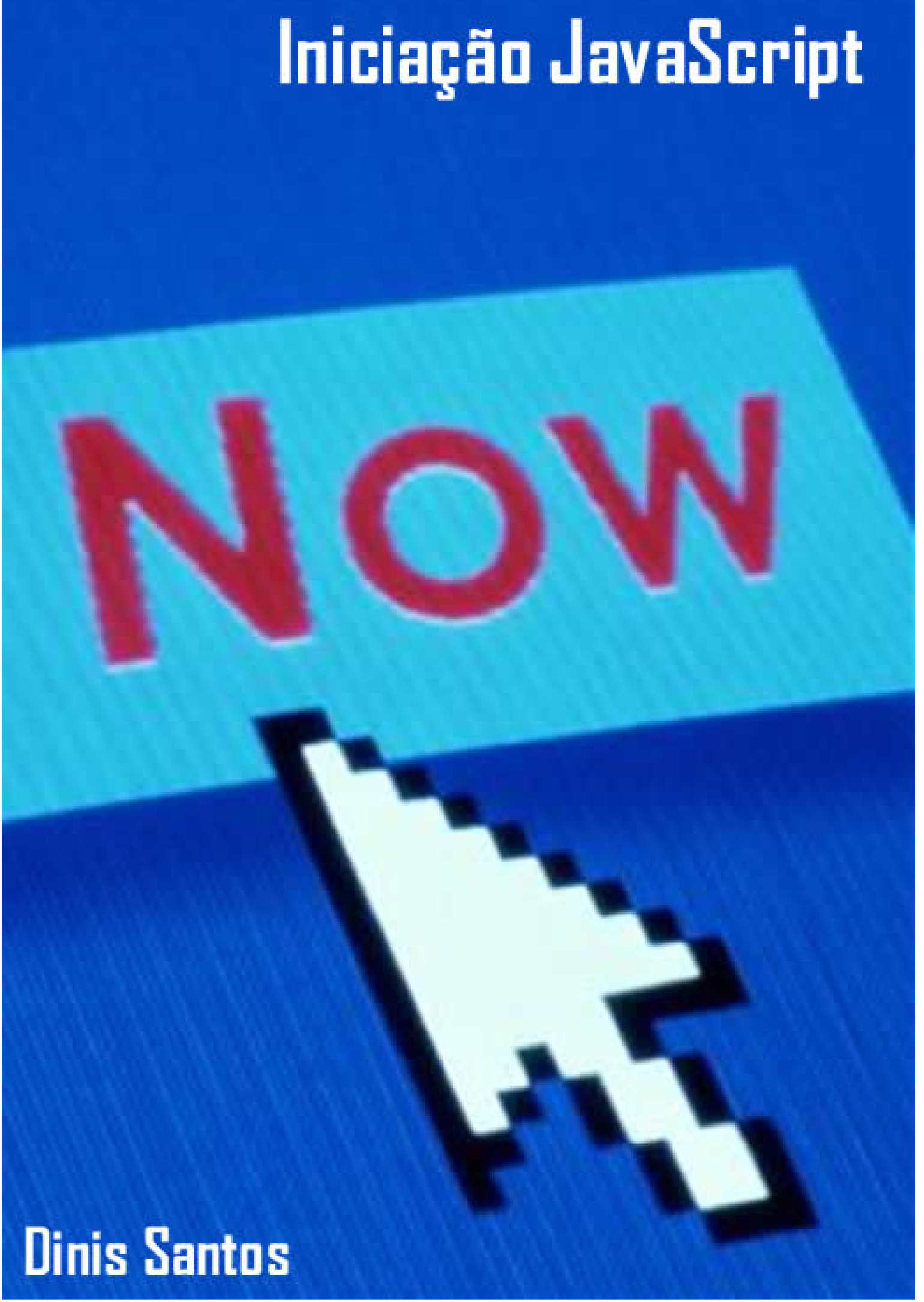
Iniciação JavaScript



Dinis Santos

Índice

Iniciação ao JavaScript

1. Introdução

O JavaScript é uma linguagem de programação simples criada para dar mais interactividade e maior funcionalidade às páginas da Web. Tendo sido inicialmente desenvolvida pela Netscape, a linguagem JavaScript acabou por dar origem à especificação técnica ECMAScript, que é um padrão oficial reconhecido pela indústria. Apesar de esta linguagem ser mais conhecida pelo nome de JavaScript, e de a versão produzida pela Microsoft ter recebido o nome de JScript, a verdade é que se tratam de implementações que sendo fiéis à norma ECMAScript lhe acrescentaram novas funcionalidades úteis, mas respeitando sempre as especificações oficiais.

O código escrito em JavaScript destina-se a ser executado pelo web browser quando a página HTML que o contém é visualizada. Ele é uma parte integrante da página e permite que o browser seja capaz de tomar decisões quanto ao modo como o conteúdo é apresentado ao utilizador e como pode ser manipulado.

2. Inserir JavaScript numa página da Web

2.1 O elemento `<script>`

Os browsers capazes de executar código escrito em JavaScript reconhecem o elemento `<script>`. É dentro desse elemento que se coloca todo o código, como ilustra o exemplo seguinte:

```
<html>
   <head>
          <title>A Minha Página com JavaScript</title>

         <script type="text/javascript">
                    alert("Seja bem vindo(a) à minha página!");
          </script>
   </head>
   <body>
          Aqui colocamos o conteúdo da página em HTML
   </body>
</html>
```

Repare que no final da linha de código colocámos o carácter `;` o qual dá ao interpretador de JavaScript a indicação de que a instrução termina nesse local. O JavaScript não nos obriga a terminar as instruções deste modo, bastando que mudemos de linha para que ele perceba que a instrução chegou ao fim. No entanto isso torna mais difícil a localização dos erros e pode também contribuir para gerar mais erros. É conveniente que os principiantes terminem todas as instruções com o carácter `;` e, se preferirem, só deixem de o fazer quando se sentirem completamente à vontade com a linguagem.

Graças ao JavaScript podemos fazer com que os objectos gráficos apresentados na página (como por exemplo uma imagem, um botão ou uma ligação de hipertexto) respondam dinamicamente às acções do utilizador. Para que isso aconteça basta adicionar um novo atributo ao elemento responsável pela apresentação desse objecto e escrever o código que ao ser executado dará origem ao comportamento pretendido. O exemplo seguinte faz aparecer uma caixa de diálogo com um agradecimento sempre que o link for clicado:

```
<html>
<body>
 <a href="http://www.artifice.web.pt/" target="_blank"
     onclick="alert('Obrigado por visitar o Artífice da Web!')">
            visite o Artífice da Web</a>
</body>
</html>
```

Certamente já conhece bem o atributo `href="..."`, que serve para especificar o URL da página a que a ligação de hipertexto conduz, mas note que o atributo `onclick="..."` é bem diferente porque o seu conteúdo é constituído por código JavaScript, que neste caso faz aparecer a caixa de diálogo com a mensagem de agradecimento. (Se não conseguir compreender o modo como o texto contido no atributo `onclick` consegue fazer isto não se preocupe, esta técnica, entre outras, serão explicadas neste e nos restantes tutoriais desta colecção.)

3. Comentários e blocos de código

3.1 Comentários

Os comentários permitem-nos descrever o código JavaScript que produzimos tornando-o mais legível e mais fácil de manter. Se comentar adequadamente o código que produz, quando mais tarde precisar de o melhorar ou fazer alterações será mais fácil e rápido perceber o que fez antes. Se produz código para partilhar com outras pessoas então os comentários são ainda mais importantes para que os outros percebam aquilo que escreveu.

Em JavaScript podemos usar comentários com uma única linha e comentários com várias linhas. Os comentários com uma única linha começam com os caracteres //. Isto dá ao interpretador de JavaScript a indicação de que o resto da linha é um comentário, pelo que este ignora o resto da linha, continuando a interpretar o código na linha seguinte.

Um comentário que se estende por várias linhas começa com a sequência de caracteres /* e continua até ser econtrada a sequência de caracteres */, que marcam o fim do comentário. Ao encontrar a sequência /* o interpretador de JavaScript procura imediatamente a sequência de fecho */, continuando aí a interpretação do código e ignorando o que está no meio.

Aqui ficam alguns exemplos de comentários em JavaScript.

```
// Este é um comentário com uma única linha

/* Este comentário ocupa uma só linha mas podia ocupar mais */

/*
         Este comentário ocupa várias linhas. Tudo o que for
         escrito aqui dentro será ignorado pelo interpretador
         de JavaScript
*/
```

3.2 Blocos de código

Quando temos de executar funcionalidades não triviais é quase sempre preciso executar sequências de instruções compostas por várias linhas. Se essas sequências tiverem de ser executadas condicionalmente (veja por exemplo a descrição da instrução `if` mais à frente), ou se formarem uma função, então elas constituem um bloco e têm de ser agrupadas. Isso consegue-se colocando-as entre chavetas (`{ }`.)

```
{       // isto é um bloco de código
        var i = 0;
        var j = i * 3;
```

4. Variáveis

O que são as Variáveis?

As variáveis são objectos que servem para guardar informação. Elas permitem-nos dar nomes a cada um dos fragmentos de informação com que temos de lidar. Se esses nomes forem bem escolhidos fica fácil saber onde é que se deve guardar um determinado pedaço de informação e onde é que se pode ir buscar a informação que se guardou antes. Para evitar erros e aumentar a produtividade é importante escolher nomes que descrevam aquilo que que cada variável guarda. Assim, se escrevermos um programa que divide dois números é acertado chamar `dividendo`, `divisor` e `quociente` aos números envolvidos na operação. Escolhas como por exemplo `n1`, `n2` e `n3`, apesar de funcionarem, provocam confusão e dão origem a erros difíceis de detectar porque tornam o código mais difícil de ler.

É importante que saibamos quais as regras que temos de respeitar quando escolhemos um nome para uma variável:

- Todos os nomes têm de começar com uma letra ou com o carácter `_`.
- Os restantes caracteres que compoem o nome podem igualmente conter números. Nunca se esqueça que para o JavaScript **letra grande e letra pequena são coisas diferentes** e que, por exemplo, as variáveis `variavel1`, `Variavel1` e `vaRiavel1` são três objectos distintos.

4.1 Declaração de Variáveis

Ao acto de criar uma variável dá-se o nome de declaração. As variáveis que são declaradas fora de qualquer função (mais à frente iremos ver exemplos de declarações de variáveis e o que são funções) são designadas por variáveis globais. Aqui o termo global significa que a variável em causa pode ser utilizada em qualquer parte do script; ela está permanentemente acessível. Quando uma variável é declarada dentro de uma função ela será uma variável local porque só pode ser utilizada dentro dessa função.

Se tentarmos aceder a uma variável local fora da função em que ela foi declarada será gerado um erro porque a variável só existe no universo da função em que foi declarada; ela não faz parte do mundo exterior a essa função e como tal não pode ser aí utilizada.

A seguir temos alguns exemplos de declaração de variáveis:

```
dividendo = 12;
divisor = 3;
sabor = "Doce";
pi = 3.14159;
```

Nestes exemplos todas as variáveis declaradas serão variáveis globais. Se quisermos declarar variáveis cuja existência se limite a uma pequena secção do código teremos de usar a declaração `var`, assim: `var dividendo = 12;`

Se usarmos esta declaração fora de qualquer função então, porque a variável é declarada na base da estrutura de código, ela será global.

Temos assim que a declaração `var` serve para limitar o contexto em que a variável existe e que:

- As variáveis declaradas sem a declaração `var` são variáveis globais;
- As variáveis declaradas usando a declaração `var` existem apenas no contexto em que foram definidas.

Antes de começar a escrever código em JavaScript é importante planear o modo como este será organizado. Deve-se começar por identificar os dados que vão ser utilizados. A seguir escolhem-se os nomes das variáveis que vão guardar esses dados e só depois é que se começa a escrever o código propriamente dito.

4.2 Os valores das Variáveis

A linguagem JavaScript é capaz de reconhecer três tipos de dados:

- Números, como por exemplo `12` ou `3.14159`
- Texto (variáveis de tipo String), como por exemplo: `"Seja Bem Vindo(a)!"`
- Valores lógicos (`true` ou `false`)
- `null`, que é uma palavra especial que significa que a variável em causa não guarda qualquer valor, está vazia.

Como iremos ter oportunidade de aprender neste e nos tutoriais seguintes, usando apenas estes tipos de dados podemos executar muitas acções úteis nas nossas páginas da Web.

4.3 Conversões de Valores

A linguagem JavaScript exige muito pouco trabalho ao programador para definir o tipo de dados que uma variável deve guardar. É o próprio interpretador de JavaScript que em função dos dados que recebe decide se estes representam um número, texto (string), um valor lógico, ou nada (null). Assim, se escrever:

```
var resposta = 42;
```

o interpretador decidirá guardar internamente a variável `resposta` como um número inteiro, mas se escrevermos:

```
var resposta = 42;
resposta = "O JavaScript aprende-se muito depressa.";
```

ao chegar à segunda linha de código o interpretador mudará de ideias e a variável `resposta` deixará de ser guardada internamente como um número inteiro para passar a ser guardada como uma String (texto). Esta conversão no tipo da variável acontece de forma automática e o programador não precisa de fazer nada para que ela aconteça.

Esta liberdade que nos é dada pelo JavaScript destina-se apenas a simplificar a escrita do código. Quando é mal utilizada ela pode dar origem a código difícil de ler e a erros. As regras de boa programação dizem que ao definir uma variável o programador deve decidir qual o tipo de dados (número, texto ou valor lógico) que esta irá conter e não deverá escrever código que provoque uma conversão no tipo de dados que a variável guarda. Sempre que uma tal conversão for necessária deverá ser definida uma nova variável para guardar o resultado da conversão, mantendo inalterados os tipos das variáveis antigas. Na prática esta recomendação raramente é seguida.

5. Expressões Literais

5.1 Representação de valores

As expressões literais representam valores fixos. Elas são escritas directamente pelo programador ao produzir o script. Exemplos de expressões literais podem ser: `123` ou `"Isto é uma expressão literal"`.

As expressões literais podem ser usadas de diversas maneiras, como ilustra o excerto de código apresentado a seguir (o exemplo seguinte usa as instruções `if/else` que só são estudadas mais à frente):

```
var nome = "visitante";
var hora = 11;

if(hora < 12)
   document.write("Bom dia. Seja bem vindo senhor " + nome);
else
{  if(hora >= 13)
      document.write("Boa tarde. Seja bem vindo senhor " + nome);
   else
      document.write("Seja bem vindo! Almoça connosco?");
}
```

Na primeira linha usámos a expressão literal `"visitante"` para dar um valor inicial à variável `nome`. Na segunda linha usámos uma expressão literal numérica para dar um valor à variável `hora`. O resto do código usa as expressões literais `12` e `13` para determinar a parte do dia (manhã, tarde ou hora de almoço) e cumprimentar usando o texto (expressão literal) mais adequado.

5.2 Números inteiros

Os números inteiros podem ser expressos na forma decimal (base 10), hexadecimal (base 16) ou octal (base 8). Um número decimal consiste numa sequência de dígitos que nunca deve começar por `0` (zero). Se escrevermos um número com um zero no início isso significa que se trata de um número escrito na forma octal. Por outro lado, se no início escrevermos os caracteres `0x` (ou `0X`) isso significa que o número está escrito na forma hexadecimal. Os números escritos na forma decimal podem conter os dígitos (0-9), a forma octal aceita apenas dígitos de (0-7) e a forma hexadecimal aceita os dígitos (0-9) mais as letras a-f e A-F.

Exemplos de números inteiros são: `42, 052, 0X2A`, que representam todos o valor decimal `42`. No exemplo seguinte as variáveis `i, j, k` possuem todas o mesmo valor, apesar de serem usadas bases diferentes para as inicializar:

```
var i = 42;    // decimal
var j = 052;   // octal
var k = 0X2A;  // hexadecimal

// quando executar este código repare que as variáveis
// têm todas o mesmo valor

document.write("i = " + i);
document.write("<br/>");
document.write("j = " + j);
document.write("<br/>");
document.write("k = " + k);
```

5.3 Números com vírgula flutuante

Uma expressão literal com vírgula flutuante representa um número que não é inteiro mas que contém uma parte inteira e uma parte fraccionária. Os números `21.37` e `-0.0764` são exemplos disto. A representação que a máquina constrói para estes números baseia-se na notação científica. Por exemplo, o número -7645.4532 é igual a -7.64532 a multiplicar por 10 elevado a 3, e escreve-se como -7.6454532E3, em que E3 representa 10 elevado a 3. Um outro exemplo é o número 0.00045431, que é representado na forma 4.5431E-4, ou seja 4.5431 a multiplicar por 10 elevado a -4. Esta representação é construída automaticamente pela máquina, o programador pode escrever o número na forma que gostar mais.

5.4 Valores lógicos (booleanos)

Estas expressões podem assumir apenas dois valores: `true` (verdadeiro) e `false` (falso.)

5.5 Expressões de texto

Uma expressão de texto é composta zero ou mais caracteres colocados entre aspas ("), como por exemplo `"esta é uma expressão de texto"`, ou entre plicas ('), como por exemplo `'esta é outra expressão de texto'`. Se começarmos a expressão com aspas temos forçosamente de usar aspas para a terminar, e se a inciarmos com uma plica temos de usar outra plica para a terminar.

Para além dos caracteres normais, as expressões de texto podem conter os caracteres especiais apresentados na lista seguinte:

Carácter	Significado
\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	tab
\\	backslash

Cada um destes caracteres produz o mesmo resultado que se obtém carregando na tecla indicada na segunda coluna. Assim o carácter \b equivale a carregar na tecla backspace (apagar o carácter à esquerda). O carácter \n provoca uma mudança de linha tal como a tecla "enter". O carácter \ é usado como prefixo dos outros caracteres especiais, o que faz também dele um carácter especial. Por isso, para obtermos este carácter temos de temos escrevê-lo duas vezes (\\). Se o escrevermos uma única vez em lugar de o obtermos estaremos a tentar introduzir um outro carácter especial e o resultado será diferente do que pretendemos.

5.6 Caracteres de escape

Se o carácter que vem a seguir a \ não pertencer à lista anterior o seu efeito será nulo, mas há duas excepções: as aspas (") e a plica ('). Se pretendermos escrever aspas dentro de uma expressão de texto temos de colocar o carácter \ antes delas, como mostra o exemplo seguinte:

```
var texto = "Ele leu o \"Auto da Barca do Inferno\" de Gil Vicente.";
document.write(texto);
```

O resultado será:

Ele leu o "Auto da Barca do Inferno" de Gil Vicente.

Se em vez de aspas usarmos apenas plicas teremos:

```
var texto = 'Ele leu o \'Auto da Barca do Inferno\' de Gil Vicente.';
document.write(texto);
```

O resultado será:

Ele leu o 'Auto da Barca do Inferno' de Gil Vicente.

Porém, a melhor solução para este problema não é nenhuma das anteriores. Se usarmos plicas como caracteres delimitadores de uma string então passamos a poder usar as aspas como parte do conteúdo sem qualquer problema, como se mostra a seguir:

```
var texto = 'Ele leu o "Auto da Barca do Inferno" de Gil Vicente.';
document.write(texto);
```

Mas se quisermos colocar plicas no conteúdo a melhor forma de evitarmos os problemas consiste em usar aspas como caracterres delimitadores da string, como se mostra a seguir:

```
var texto = "Ele leu o 'Auto da Barca do Inferno' de Gil Vicente.";
document.write(texto);
```

6. Cadeias de variáveis (Array)

Uma cadeia de variáveis (objecto Array) é um objecto capaz de guardar muitos valores, tantos quanto a memória disponível na máquina permitir. Cada uma das variáveis que compoem o array possui um índice. Ilustremos isto com um exemplo:

```
var frutas_tropicais = new Array("Goiaba", "Manga", "Maracujá");
var frutas_nacionais = new Array(3);
frutas_nacionais[0] = "Maçã";
frutas_nacionais[1] = "Cereja";
frutas_nacionais[2] = "Laranja";
```

Ao declararmos a variável `frutas_tropicais` nós declarámos o Array e atribuímos-lhe os valores numa única operação. Já no segundo caso primeiro declarámos o Array e só depois definimos os valores que ele deve conter. Neste caso temos que a variável frutas_tropicais[3] possui o valor `"Maracujá"` e a variável frutas_nacionais[0] possui o valor `"Maçã"`.

Em JavaScript as variáveis não têm um tipo definido, por isso um array pode conter valores de tipos diferentes que podemos alterar sempre que necessário, como se mostra a seguir:

```
var sortido = new Array(8975, "Livro", false, -27.765, "Bolachas");
document.write("sortido = " + sortido);

sortido[0] = 0.0004763;
sortido[2] = true;
sortido[6] = "Caderno";

document.write("<br/>");
document.write("sortido = " + sortido);
```

Faça clique aqui para editar e executar este exemplo.

Se atribuirmos um valor a um elemento do array com um índice mais alto do que o seu comprimento, o sistema JavaScript resolve o problema aumentando o tamanho do array até chegar ao índice pretendido. É isso que acontece no exemplo anterior quando se chega à linha que tem `sortido[6] = "Caderno";` Os arrays são objectos, e entre as suas propriedades conta-se a propriedade `length`, que nos dá o número de elementos (variáveis) que ele contém num determinado momento. Assim, se ao exemplo anterior juntarmos uma linha com o seguinte código:

```
var numeroDeElementos = sortido.length;
```

a variável `numeroDeElementos` ficará com o valor 7 (repare que inserimos um elemento adicional com o índice 6, o que fez crescer o array). De forma análoga se usarmos `frutas_nacionais.length` iremos obter 3.

7. Operadores

A linguagem JavaScript possui muitos operadores de diversos tipos. Aqui iremos apenas abordar os aspectos mais básicos dos operadores disponíveis.

7.1 Operadores de atribuição de valor

Uma das coisas que os operadores podem fazer é fornecer um valor àquilo que estivar à sua esquerda. Se o que está à esquerda for uma variável então o valor dela passará a ser aquilo que o operador forneceu, se for outro operador o valor fornecido será usado como operando.

Os operadores mais conhecidos são as quatro operações aritméticas básicas (adição subtracção, multiplicação e divisão.) Para estes a linguagem JavaScript define as seguintes versões curtas:

Versão curta	Significado
x+=y	x=x+y
x-=y	x=x-y
x*=y	x=x*y
x/=y	x=x/y

Repare que aqui o sinal = não representa a igualdade matemática. Ele serve apenas para indicar que a variável que está à sua esquerda deve passar a ter um valor igual ao valor da expressão que está à sua direita. Se tivermos x=5 e y=7 a expressão x=x+y não representa uma igualdade matemática mas sim a indicação que o valor de x deve passar a ser igual à soma do valor que tem actualmente com o valor de y. Neste caso x passaria a valer 12.

7.2 Operadores de comparação

Um operador de comparação compara os valores que lhe são fornecidos (que designamos por operandos) e retorna um valor lógico que indica se o resultado da comparação é verdadeiro ou falso. Os valores que recebe para analisar podem ser números ou variáveis de texto (string). Quando actuam sobre variáveis de texto, as comparações baseiam-se na forma como os caracteres estão ordenados sequencialmente. Esta ordenação baseia-se na ordem alfabética. A lista seguinte apresenta estes operadores.

Operador	**Descrição**	**Exemplo**
Igualdade (==)	Verifica se os dois operandos são iguais	x==y dá true se x igualar y
Desigualdade (!=)	Verifica se os operandos são desiguais	x!=y dá true se x não for igual a y
Maior do que (>)	Verifica se o operando da esquerda é maior do que o da direita	x>y dá true se x for maior do que y
Maior ou igual (>=)	Verifica se o operando da esquerda é maior ou igual ao da direita	x>=y dá true se x for maior ou igual a y
Menor do que (<)	Verifica se o operando da esquerda é menor do que o da direita	x<y dá true se x for menor do que y
Menor ou igual (<=)	verifica se o operando da esquerda é menor ou igual ao da direita	x<=y dá true se x for menor ou igual a y

7.3 Operadores aritméticos

Um operador aritmético recebe valores numéricos (tanto variáveis como expressões literais) e produz um valor numérico como resultado. Os operadores numéricos mais importantes são a adição (+), a subtracção (-), a multiplicação (*), a divisão (/) e o resto da divisão (%). O funcionamento destes operadores em JavaScript respeita todas as regras da álgebra.

Porque é muitas vezes necessário adicionar ou subtrair uma unidade a uma variável, a linguagem JavaScript define dois operadores especiais com esta finalidade. Assim, para adicionarmos uma unidade à variável `variavel1` podemos escrever `variavel1++`, e para subtrairmos uma unidade à `variavel2` escrevemos `variavel2--`. Por acção destes operadores no final do exemplo seguinte a variável `variavel1` terá o valor 4 e a variável `variavel2` terá o valor 6.

```
var variavel1 = 3;
variavel1++;
var variavel2 = 7;
variavel2--;
```

7.4 Operadores lógicos

Os operadores lógicos aceitam os valores lógicos true e false (verdadeiro e falso) como operandos e retornam valores lógicos como resultado. Os operadores lógicos base encontram-se listados a seguir (os restantes definem-se com base nestes três.)

Operador	**Utilização**	**Descrição**
e (&&)	b && c	Dá true se b for true e c for true.
ou (||)	b || c	Dá false se b for false e c for false. Dá true nos casos restantes.
negação (!)	!b	Dá true se b for false e dá false se b for true.

Os casos mais úteis e interessantes de uso destes operadores utilizam dois ou os três operadores ao mesmo tempo, como se mostra a seguir:

Se tivermos `x = 4` e `y = 7` a operação

```
((x + y + 2) == 13) && (((x + y) / 2) == 2)
```

dá false.

Se tivermos `x = 4` e `y = 7` a operação

```
((y - x + 9) == 12) || ((x * y) == 2)
```

dá true.

Se tivermos `x = 4` e `y = 7` a operação

```
!((x/2 + y) == 9) || ((x * (y/2)) == 2)
```

dá false.

8. Objectos

O objectivo da colecção de documentos de estudo (tutoriais) de que este faz parte é ensinar as tecnologias padrão definidas para criar páginas e aplicações para a Web. A utilização dos objectos da linguagem JavaScript é aqui tratada de forma rápida. O estudo aprofundado deste tópico será feito no tutorial Programação em JavaScript e no Tutorial de HTML Dinâmico.

Objectos definidos no padrão ECMAScript

A linguagem JavaScript é uma implementação do padrão ECMAScript. Esse padrão define as regras de sintaxe que temos estado a estudar e um conjunto mínimo de objectos que fazem do ECMAScript uma verdadeira linguagem de programação, mas não define os objectos que permitem manipular e interagir tanto com o browser como com as páginas da Web. Para ser verdadeiramente útil o JavaScrit tem de complementar o ECMAScript com objectos adicionais.

Document Object Model (DOM)

O W3C (*World Wide Web Consortium*) definiu o padrão DOM para padronizar a forma como os browsers e as aplicações da Web manipulam e interagem com as páginas da Web. Todos os browsers modernos implementam estes padrões. Apesar de essas implementações serem geralmente incompletas, elas são suficientes para que possamos programar quase tudo numa forma que funciona em todos os browsers dominantes (MSIE 5 e superior, Mozilla/Netscape 7 e Opera 7.)

Outros objectos úteis

Quando a linguagem JavaScript surgiu, os seus criadores definiram aqueles objectos que lhe pareceram importantes. De entre eles alguns foram incorporados pelo padrão ECMAScript, outros foram de alguma forma incorporados pelo DOM (geralmente com modificações), e outros não estão presentes em qualquer padrão oficial mas são suportados universalmente, o que faz deles padrões *de facto*. Com estes objectos podemos formar um grupo que podemos usar nas nossas aplicações com toda a confiança. O Tutorial de HTML Dinâmico irá tratar este assunto em profundidade.

8.1 Exemplos práticos com objectos

Dois dos objectos que ficam imediatamente disponíveis quando carrega um documento no browser são o objecto `document`, que nos permite mainpular e interagir com a página da Web, e o objecto `window`, que nos permite controlar a janela do browser que contém a página.

O objecto `window` possui vários métodos. Entre eles temos os métodos `close()`, `alert()`, `confirm()` e `prompt()`, com os quais podemos fechar a janela do browser, apresentar avisos ao utilizador e pedir-lhe para nos dar uma resposta ou escrever alguma coisa. O código:

```
window.alert("Esta é uma janela com um aviso");
```

faz aparecer uma janela com um aviso para o utilizador. A notação por pontos significa que estamos a chamar o método `alert()` pertencente ao objecto `window`. Neste caso podíamos ter escrito apenas `alert(mensagem)` e omitido a parte `window.` (o browser já sabe que o método `alert` pertence ao objecto `window`)

O objecto document contém uma representação da página HTML. Cada um dos elementos que compoem a página (formulários, parágrafos, imagens, links, etc) podem ser lidos e manipulados utilizando este objecto. Depois de uma página estar carregada, o código seguinte:

```
alert("A segunda imagem desta página foi carregada a partir de: "+
document.images[1].src);
```

mostra a origem (src) de uma imagem. Repare que com o objecto document temos de usar sempre a notação por pontos, não são aceites abreviações.

9. Definir uma Função

As funções permitem-nos agrupar várias linhas de código que realizam um determinado trabalho, dar-lhe um nome e pô-las a trabalhar chamando-as por esse nome.

O exemplo seguinte define uma função:

```
function dataActual()
{
          /*
               Cria um objecto com a data e hora actuais e mostra
               o seu valor na janela recorrendo ao método
               toLocaleString() do objecto Date
          */
          var d = new Date();
          document.write("A data e hora são: " + d.toLocaleString());
}

dataActual(); // esta linha faz executar a função
```

(nunca esqueça que em JavaScript as letras maiúsculas não são equivalentes às letras minúsculas, por isso tenha sempre muita atenção ao facto de que o nome que usa para chamar uma função tem de ser rigorosamente igual ao nome que lhe deu ao defini-la.)

No exemplo anterior usámos os caracteres { e } para delimitar um bloco de código. Tudo o que está dentro destes delimitadores faz parte da função e será executado sempre que esta for invocada escrevendo `dataActual()` no seu código. Como resultado será escrita na página a data e hora do momento em que a função foi chamada.

Também podemos passar argumentos para a função, como se mostra a seguir:

```
function cumprimentar(nome)
{
  var d = new Date();
  document.write("Olá " + nome + "<br/>A data e hora actuais são: "
                                                + d.toLocaleString());
}

cumprimentar('Zézinho'); // esta linha faz executar a função
```

como teremos oportunidade de ver quando aprofundarmos o nosso estudo, as funções têm uma importância fundamental na programação de scripts complexos

10. As instruções condicionais if...else

Uma instrução `if` permite-nos executar uma porção de código apenas se for verdadeira uma determinada condição. Se essa condição não for verdadeira essa porção de código não será executada, podendo ser ou não executado outro código alternativo, que será especificado através da palavra `else`.

A ideia principal que está na base das instruções `if/else` pode ser resumida numa frase: "Se chegarmos antes da hora de almoço vamos dar um passeio e no caso contrário vamos para a mesa". O exemplo seguinte ilustra esta ideia:

```
var hora = 10;

if(hora < 12)
        document.write("Vamos passear");
else
        document.write("Vamos para a mesa");
```

Neste exemplo a hora é de antes do almoço e será apresentada uma janela que tem escrito `Vamos passear`. Se a hora fosse 12 ou mais seria mostrado o texto `Vamos para a mesa`.

Uma instrução `if` não precisa de ter associada a si uma instrução `else`. Quando isso acontece se a condição não se verificar não será executado qualquer código alternativo

11. Executar código repetidamente

Um dos recursos mais poderosos no arsenal de qualquer linguagem de programação é a capacidade para repetir a realização de tarefas de uma forma simples. Para isso definem-se ciclos de repetição dentro dos quais se coloca o código que se pretende executar repetidamente.

11.1 Ciclos for

Um ciclo `for` consiste num conjunto de três expressões contidas entre parêntesis, separadas pelo carácter `;` (ponto e vírgula) e pelo código a executar em cada um dos ciclos. Normalmente esse código estará contido entre chavetas para formar um bloco, mas se consistir numa única linha não é preciso usar chavetas.

A primeira das expressões do ciclo `for` declara a variável a usar como índice (funciona apenas como contador) e inicializa-a. A segunda expressão declara uma condição que deve ser testada sempre que se inicia um novo ciclo. Se essa condição der `false` como resultado o ciclo pára e o código definido abaixo não voltará a ser executado. A terceira expressão serve para actualizar o valor do índice no final de cada ciclo.

Ilustremos isto com um exemplo simples:

```
soma = 0;
for(var i = 0; i < 3; i++)
{      soma += i;
       document.write("O valor do índice é agora de " + i + "<br/>");
}
```

Este pedaço de código começa por definir uma variável (global) chamada `soma` atribuindo-lhe o valor zero. O ciclo `for` define uma variável de índice (`var i = 0`) e verifica se a condição `i < 3` é cumprida. Se o resultado da verificação for `true` será executado o código que se encontra entre chavetas mais abaixo, o qual adiciona `i` à variável `soma` e apresenta uma mensagem informando sobre o valor actual da variável `i`. Depois é executada a terceira instrução do ciclo (`i++`), a qual soma uma unidade ao valor do índice `i` e dá-se início a um novo ciclo. Este começa por testar de novo o respeito pela condição `i < 3`. Se o resultado for `true` volta a executar o código que está entre chavetas com o valor actualizado de `i`. Isto repete-se até que `i < 3` dê false, o que termina a execução do ciclo `for`.

O exemplo seguinte é mais elaborado e executa um ciclo que percorre todos os elementos de um array de nomes e destaca aqueles que começam com a letra H.

```
var nomes = new Array("Manuel", "Rita", "Joana", "Francisco", "Luís",
   "Bernardo", "Helder", "Patrícia", "Hugo", "António", "Nuno");
for(var i=0; i < nomes.length;i++)
{
       var nome = nomes[i]
       if(nome.charAt(0) == "H")
                   alert("O nome " + nome + " começa com a letra H");
}
```

Neste exemplo usámos o método `charAt()` do objecto String para verificar se o carácter inicial do nome (aquele está na posição zero) é igual à letra `H`.

11.2 Ciclos while

O ciclo `while` é muito parecido com o ciclo `for`. De facto tudo o que um faz pode ser feito com o outro, embora por questões de legibilidade (e de elegância do código) cada um deles possa ter áreas de aplicação que para as quais é mais indicado do que o outro.

O ciclo `while` avalia uma condição e se ela der `true` executa o bloco de código que vem imediatamente a seguir. Se der `false` salta para a frente do bloco de código que vem a seguir sem o executar.

Este exemplo usa um ciclo `while` para produzir o mesmo efeito que o exemplo que está antes daquele que acabámos de ver:

```
soma = 0;
i = 0;
while(i < 3)
{
  soma += i;
  document.write("O valor da variável i é agora de " + i + "<br/>");
  i++;
}
```

12. Notas finais

Esta introdução informal serviu para oferecer algumas noções básicas sobre a linguagem JavaScript e preparar o leitor para começar a usá-la para dar funcionalidades avançadas às suas páginas da Web.