1
Introdução a JavaScript

A LINGUAGEM HTML – HYPERTEXT MARKUP LANGUAGE, foi criada *exclusivamente* para definir a estrutura de uma página. Esforços para usar HTML como linguagem de formatação de página, visando uma melhor *apresentação* gráfica resultaram ineficazes[1]. De forma semelhante, HTML não é linguagem de *programação.* Não possui as estruturas essenciais para realizar operações e controle de fluxo. É uma linguagem declarativa criada para *estruturar* páginas de hipertexto através de *marcadores* que descrevem a *função* de blocos de texto.

Com HTML, é fácil criar interfaces do usuário sofisticadas, usando recursos de formulário como botões, caixas de seleção, etc. A coleção de componentes de formulário conta com dois tipos de botões que respondem a eventos do usuário. Um dos botões, ao ser apertado, provoca um evento permite enviar os dados coletados no formulário para um programa no servidor (CGI[2]) para processamento remoto. Não há processamento local.

Os componentes de formulário existem desde HTML 1.1 (1994) e com eles surgiram as primeiras "aplicações Web". Essas aplicações sempre tinham que se comunicar com o servidor para realizar qualquer operação. Para fazer uma simples conta era necessário enviar os dados para o servidor, rodar um programa na máquina remota e aguardar uma nova página retornada com o resultado. Isso era necessário porque não havia como fazer contas simples em HTML.

Plug-ins proprietários foram os primeiros a introduzir aplicações Web executando do lado do cliente. Depois vieram os applets Java, que tiveram mais sucesso que os *plug-ins* por não se limitarem a uma única plataforma. Os *plug-ins* e os applets usam o HTML apenas como base para aparecerem no browser. Utilizam uma interface própria que ocupa uma subjanela, toda a janela ou parte de uma janela do browser. Não aproveitam os recursos do HTML. É preciso desenvolver sua interface usando uma outra linguagem, o que torna o desenvolvimento bem mais complexo que a criação de formulários HTML.

---

[1] Várias versões do HTML continham descritores de apresentação. Eles foram considerados obsoletos pela última recomendação do W3C: HTML 4. A linguagem CSS (folhas de estilo em cascata) é a atual recomendação do W3C para a formatação de páginas HTML

[2] Common Gateway Interface – especificação que define o formato de programas cuja execução é iniciada por um servidor Web.

A primeira tecnologia proposta como extensão verdadeira do HTML foi JavaScript. Como é extensão, faz parte da *página* e pode interagir com todos os seus componentes, como formulários e imagens, inclusive com os *plug-ins* e applets. É a melhor solução para realizar tarefas simples que não compensam o esforço de desenvolver um applet ou *plug-in*. Em geral, apresenta um desempenho melhor que esses componentes de browser. Hoje, com mais recursos e mais estável, JavaScript tem ocupado lugares antes ocupados por applets e *plug-ins* em vários serviços on-line.

O que é JavaScript?

JAVASCRIPT É UMA LINGUAGEM de programação interpretada criada em 1995 por Brendan Eich da Netscape como uma *extensão* do HTML para o browser Navigator 2.0. Hoje existem implementações JavaScript nos browsers dos principais fabricantes. Mas o uso de JavaScript não tem se limitado aos browsers. Tem sido usado, em menor escala, como linguagem de suporte a tecnologias de *gateway* para servidores HTTP e até como linguagem de roteiro de propósito geral. Embora ainda seja mantida e estendida pela Netscape, parte da linguagem JavaScript já é padrão proposto pela ECMA – organização européia para padrões em comunicações, que visa transformá-la em padrão Web[3].

JavaScript do lado do browser (*client-side* JavaScript) tem evoluído e alcançado uma estabilidade razoável como um padrão da Web. É hoje (1998), suportada pelas principais versões de browser que povoam a Web e é a linguagem de programação mais popular do mundo, com presença em 35 milhões de páginas Web[4].

JavaScript no servidor (*server-side* JavaScript) é uma linguagem que possui o mesmo núcleo que o JavaScript do lado do cliente, mas acrescenta estruturas exclusivas para interação com entidades do servidor. Não tem ainda a estabilidade necessária para ser considerada um padrão pois suas implementações são praticamente restritas à extensões Netscape, como a tecnologia LiveWire. O núcleo da linguagem JavaScript também está presente na tecnologia ASP (Active Server Pages) da Microsoft, mas LiveWire e ASP são incompatíveis entre si.

Este curso trata exclusivamente do *client-side* JavaScript, que roda no browser. No restante deste livro, chamaremos *client-side* JavaScript simplesmente de "JavaScript".

JavaScript é uma linguagem de programação *baseada* em objetos. Trata suas estruturas básicas, propriedades do browser e os elementos de uma página HTML como *objetos* (entidades com propriedades e comportamentos) e permite que sejam manipulados através de eventos do usuário programáveis, operadores e expressões. JavaScript oferece recursos interativos que faltam no HTML e permite a criação de páginas interativas e dinâmicas, que são interpretadas localmente pelo browser, sem precisar recorrer a execução remota de programas no servidor.

---

[3] http://www.ecma.ch/stand/ecma-262.htm[5]

[4] http://home.netscape.com/newsref/pr/newsrelease599.html

JavaScript não é Java

JavaScript freqüentemente é confundida com a linguagem Java, provavelmente devido à semelhança do nome. Há também algumas semelhanças na sintaxe. Tudo mais é diferente. O nome "script", que quer dizer roteiro, já indica que se trata de uma linguagem interpretada. Além do nome, podemos apontar diversas outras diferenças:

- *Interpretada*. Programas em Java são compilados para um código de máquina, que é executado em uma plataforma própria (que pode ser fornecida pelo browser). Programas em JavaScript são interpretados linha-por-linha enquanto o browser carrega a página ou executa uma rotina.
- *Simples.* Programas em Java são bem mais poderosos que programas JavaScript e não estão limitados à página HTML. Por outro lado, são bem mais complexos.
- *Pequena.* JavaScript 1.1, abordado neste livro, consiste de umas 300 funções, objetos, métodos, eventos e propriedades. A API do Java 2 possui mais de 20000 estruturas.
- *Baseada em objetos.* O modelo de objetos e as estruturas das duas linguagens são completamente diferentes. Java é uma linguagem *orientada* a objetos que possui estruturas como classes, herança, polimorfismo, etc. que não existem em JavaScript.
- *Extensão do HTML*. Nunca se coloca Java em uma página Web. Pode-se incluir uma *applet* em uma página, que é um tipo de aplicação que pode ter sido escrito em Java, ou não. O browser freqüentemente tem capacidade de executar um applet, mas não de interpretar o seu código Java. O código JavaScript geralmente vem embutido dentro de uma página HTML. Não existe JavaScript (client-side) sem HTML.

Quem suporta JavaScript?

Somente os browsers compatíveis com a linguagem JavaScript conseguem executar os roteiros (scripts). Entre os mais populares, isto inclui o Netscape Navigator versões 2 em diante, o Microsoft Internet Explorer versões 3 em diante e o OperaSoftware Opera 3.5 em diante. O JavaScript suportado por um browser pode não funcionar em outro. Os principais motivos são incompatibilidades entre versões e plataformas.

A primeira versão de JavaScript, chamada de JavaScript 1.0, foi primeiro suportada pelo Navigator 2.0. A Microsoft desenvolveu sua primeira implementação do JavaScript 1.0 no Internet Explorer 3.0, onde se chamava JScript. JavaScript 1.0 tinha muitos bugs e poucos recursos. Com o Navigator 3.0 foi lançado o JavaScript 1.1, que teve uma implementação semelhante em versões do Internet Explorer 3 e 4. A versão 1.2 do JavaScript, introduzida com o Netscape 4.05 e suportada em parte pelo Internet Explorer 4, acrescenta alguns recursos como expressões regulares e suporte a camadas.

As implementações JavaScript em browsers de fabricantes diferentes são conflitantes. O uso de recursos exclusivos de um fabricante provocará erros quando a página for carregada por outro browser. Há várias formas de usar o próprio JavaScript para atenuar esse problema.

Para garantir uma maior segurança, todos os scripts devem sempre ser testados nos os browsers, versões *e plataformas* utilizadas pelo público-alvo de um site ou página.

O que se pode fazer com JavaScript?

Com JavaScript pode-se fazer diversas coisas que antes não era possível apenas com a limitada linguagem HTML como:

- Realizar operações matemáticas e computação.
- Gerar documentos com aparência definida na hora da visualização, com base em informações do cliente como versões do browser, *cookies* e outras propriedades.
- Abrir janelas do browser, trocar informações entre janelas, manipular com propriedades do browser como o histórico, barra de estado, plug-ins e applets.
- Interagir com o conteúdo do documento, alterando propriedades da página, dos elementos HTML e tratando toda a página como uma estrutura de objetos.
- Interagir com o usuário através do tratamento de eventos.

Como programar com JavaScript?

Nas seções seguintes, daremos início à apresentação da linguagem JavaScript. Para editar código HTML ou JavaScript, não é preciso mais que um simples editor de texto, como o Bloco de Notas (Windows) ou Vi (Unix). Pode-se também usar um editor HTML. Alguns editores colocam cores ou dão destaque ao código JavaScript. Outros até permitem a geração de código ou a verificação de sintaxe.

O editor HTML pode ser qualquer um, mas deve *expor o código HTML*. Editores que escondem e dificultam o acesso ao código HTML devem ser evitados. É preciso que o editor tenha pelo menos uma janela de edição de código. O ideal é usar um *editor de código* como o Allaire HomeSite, Sausage HotDog (para Windows), HotMetal (para Unix, Mac e Windows) e BBEdit (para Mac). Este livro é compatível com qualquer editor de texto ou de código.

Existem ferramentas que facilitam o desenvolvimento de JavaScript, porém elas não serão exploradas aqui. As mais conhecidas são a Microsoft Script Debugger, que funciona embutido no Microsoft Internet Explorer (é uma extensão com distribuição separada) e o Netscape Visual JavaScript. Ambos os produtos podem ser descarregados dos sites de seus respectivos fabricantes.

Os arquivos utilizados neste capítulo estão no subdiretório `cap1/`, do disquete distribuído com este livro. Para acompanhar e repetir os exemplos das seções seguintes, abra um arquivo HTML qualquer (ou use a página em branco `cap1/intro.html` disponível no disquete) e repita os exemplos, testando-os no seu browser. Qualquer editor de código ou de texto pode ser usado.

Formas de usar JavaScript

Há três[5] maneiras de incluir JavaScript em uma página Web:

- *Dentro de blocos HTML `<SCRIPT> ... </SCRIPT>` em várias partes da página:* para definir funções usadas pela página, gerar HTML em novas páginas ou alterar o procedimento normal de interpretação do HTML da página pelo browser.
- *Em um arquivo externo, importado pela página:* para definir funções que serão usadas por várias páginas de um *site.*
- *Dentro de descritores HTML sensíveis a eventos:* para tratar eventos do usuário em links, botões e componentes de entrada de dados, durante a exibição da página.

As três formas podem ser usadas em uma mesma página.

Blocos <SCRIPT> embutidos na página

A forma mais prática de usar JavaScript é embutindo o código na página dentro de um bloco delimitado pelos descritores HTML `<SCRIPT>` e `</SCRIPT>`. Pode haver vários blocos `<SCRIPT>` em em qualquer lugar da página.

```
<script>
   ... instruções JavaScript ...
</script>
```

O descritor `<SCRIPT>` possui um atributo `LANGUAGE` que informa o tipo e versão da linguagem utilizada. O atributo `LANGUAGE` é necessário para incluir blocos em outras linguagens como VBScript. É opcional para JavaScript:

```
<SCRIPT LANGUAGE="VBScript"> ... código em VBScript ... </SCRIPT>
<SCRIPT LANGUAGE="JavaScript"> ... código JavaScript ... </SCRIPT>
<SCRIPT> ... código JavaScript ... </SCRIPT>
```

A versão 1.1 de JavaScript possui estruturas inexistentes nas versões anteriores. Um browser Netscape 2.0 ou Internet Explorer 3.0 que tentar interpretar o código entre `<script>` e `</script>` pode provocar erros. O atributo `LANGUAGE` com o valor "JavaScript1.1" pode ser usado para identificar um bloco que só será usado por browsers que suportem JavaScript 1.1:

```
<BODY> <p>Última modificação:
<script language="JavaScript1.1"> <!--
    autor = "Cyrano de Bergerac";
    document.write("<b>" + document.lastModified + "</b>");
    document.write("<p>Autor: " + autor);
```

[5] A especificação da Netscape permite ainda incluir JavaScript dentro de qualquer atributo HTML para passar valores ou expressões e alterar características da página. É um recurso disponível apenas nos browsers Netscape.

```
//   --> </script>
</BODY>
```

Tudo o que está em negrito, na listagem acima, é JavaScript. O que não está em negrito é código HTML.

O código JavaScript foi colocado entre *comentários HTML* `<!--` e `-->`. Isto é usado para proteger contra browsers antigos, que não suportam JavaScript, e podem exibir o código na página em vez de executá-lo ou ignorá-lo. Browsers que suportam JavaScript ignoram os comentários HTML dentro de blocos `<SCRIPT>` e tentam interpretar o código. Browsers que suportam uma versão inferior a JavaScript 1.1 irão ignorar todo o bloco.

No código acima, **`autor`** é uma *variável* que recebe por atribuição o texto `"Cyrano de Bergerac"`; **`document`** é um *objeto* [6] JavaScript que representa a página da janela atual do browser. **`lastModified`** é uma *propriedade* [6] da página (texto contendo a data de última modificação do arquivo) e **`write()`** é um *método* [6] que escreve o texto passado como parâmetro na página representada por `document`.

O ponto (`.`) é usado para que se possa ter acesso a propriedades e métodos de um objeto. O sinal "`+`" é usado para concatenar caracteres e *strings*. As duas barras (`//`) representam um comentário JavaScript.

Ao se carregar a página HTML contendo o código acima em um browser, obtém-se uma página com a informação:

> Última modificação: **Quarta-feira, 2 de abril de 1998 13:11:47 –0300**
>
> Autor: Cyrano de Bergerac

Arquivos importados

Muitas vezes é necessário realizar um mesmo tipo de tarefa mais de uma vez. Para esse tipo de problema JavaScript permite que o programador crie *funções* que podem ser chamadas de outras partes da página várias vezes. As funções geralmente ficam em um bloco `<SCRIPT>` separado, antes de todos os outros blocos, para que sejam carregadas antes que a página seja exibida. Se várias páginas usam as mesmas *funções* JavaScript definidas pelo autor da página, uma boa opção é colocá-las em um arquivo externo e importá-lo nas páginas que precisarem delas. Este arquivo deve ter a extensão "`.js`" e conter apenas código JavaScript (não deve ter descritores HTML, como `<SCRIPT>`). Por exemplo, suponha que o arquivo `biblio.js` possua o seguinte código JavaScript[7]:

---

[6] O significado desses termos ficará mais claro nas seções posteriores.

[7] As palavras `return` e `function` são reservadas em JavaScript. No exemplo acima definem a função `soma(a, b)` que retorna a soma de números que recebe como parâmetro.

```
function soma(a, b) {
      return a + b;
}
```

Para carregar esta função e permitir que seja usada em outra página, usa-se o atributo `SRC` do descritor `<SCRIPT>`:

```
<script LANGUAGE=JavaScript SRC="biblio.js"></script>
(...)
<script>
  resultado = soma(5, 6);
  document.write("<P>A soma de 5 e 6 é " + resultado);
</script>
```

É preciso que o servidor Web esteja configurado para relacionar a extensão `.js` como o tipo MIME `application/x-javascript` para que a carga, de um arquivo externo seja possível.

Tratamento de eventos

A linguagem JavaScript introduziu no HTML como extensão 13 novos atributos[8], que permitem a captura de eventos disparados pelo usuário, como o arrasto de um mouse, o clique de um botão, etc. Quando ocorre um evento, um procedimento de manuseio do evento é chamado. O que cada procedimento irá fazer pode ser determinado pelo programador.

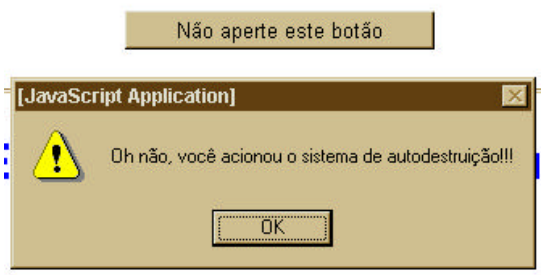
A linguagem HTML já possui três eventos *nativos* não programáveis, que são:

- clique sobre um **link** ou imagem mapeada
- clique em um botão **submit** (para envio de formulário ao CGI)
- clique sobre um botão **reset** (que limpa o formulário)

Em JavaScript 1.1, há 13 eventos adicionais programáveis através de atributos HTML especialmente criados para manuseio de eventos. No caso de conflito, eles têm precedência sobre os eventos do HTML. Os atributos de eventos se aplicam a elementos HTML específicos e e sempre começam com o prefixo "`ON`". Os valores recebidos por esses atributos é código JavaScript. Por exemplo:

```
<FORM>
<INPUT TYPE="button"
       ONCLICK="alert('Oh não, você acionou o sistema de
autodestruição!')"
       VALUE="Não aperte este botão">
</FORM>
```

mostra um trecho de código HTML que fará aparecer um botão na tela (figura).



[8] Refere-se ao JavaScript1.1/JScript 3.0

Tudo o que aparece entre as aspas duplas do atributo `ONCLICK` é JavaScript. `ONCLICK` é um atributo HTML, criado como extensão para dar suporte ao evento de clicar o botão.

O código JavaScript que está em negrito será interpretado quando o usuário apertar o botão com o mouse (`onclick`). A instrução `alert()` cria uma janela de alerta (acima) com a mensagem passada como parâmetro (entre parênteses e aspas no código em negrito). Observe que as aspas usadas dentro do método `alert()` são aspas simples já que aspas duplas já estão sendo usadas para representar o atributo HTML.

Código JavaScript também pode ser acionado através de eventos nativos do HTML, como links e botões de submissão de formulários usando uma URL "`javascript:`":

```
<a href="javascript:alert('Tem Certeza?')"> link </a>
```

O código acima fará com que o evento HTML (clicar no link) provoque a execução do código JavaScript após o *prompt* "`javascript:`". Este *prompt* também pode ser usado na barra de *Location* do browser. Oferece acesso direto ao interpretador.

Nem todos os elementos HTML suportam atributos de eventos. Também nem todas as operações JavaScript que são possíveis em blocos, como escrever na página, são possíveis após a carga completa da página, se acionados por um evento.

Os treze procedimentos de manuseio de eventos introduzidos por JavaScript são:

Atributo HTML	Quando o procedimento é executado	Descritores HTML onde é suportado
`onclick`	Quando um objeto é clicado pelo mouse	`<a>, <input>`
`onselect`	Quando um objeto é selecionado	`<input type=text>, <textarea>`
`onchange`	Quando uma seleção ou campo de texto tem seu conteúdo modificado	`<input type=text>, <textarea>, <select>`
`onfocus`	Quando um componente de formulário ou janela se torna ativa	`<textarea>, <body>, <form>, <input>, <select>, <option>`
`onblur`	Quando um componente de formulário ou janela se torna inativa	`<textarea>, <body>, <form>, <input>, <select>, <option>`
`onmouseover`	Quando o mouse está sobre um link	`<a>, <area>`
`onmouseout`	Quando o mouse deixa um link	`<a>, <area>`
`onsubmit`	Antes de enviar um formulário	`<input type=submit>`
`onreset`	Antes de limpar um formulário	`<form>`
`onload`	Após carregar uma página, janela ou frame	`<body>`
`onunload`	Ao deixar uma página, janela ou frame	`<body>`
`onerror`	Quando um erro ocorre durante a carga de uma imagem ou página	`<img>, <body>`
`onabort`	Quando a carga de uma imagem é abortada	`<img>`

Como procedimentos de eventos são atributos do HTML (e não do JavaScript), tanto faz escrevê-los em letras maiúsculas ou minúsculas. Usar `onclick`, `ONCLICK` ou `OnClick` não faz diferença. Já o texto *dentro das aspas* é JavaScript, que é uma linguagem que diferencia letras maiúsculas de minúsculas, portanto `alert` *não é* a mesma coisa que `ALERT`.

Introdução prática

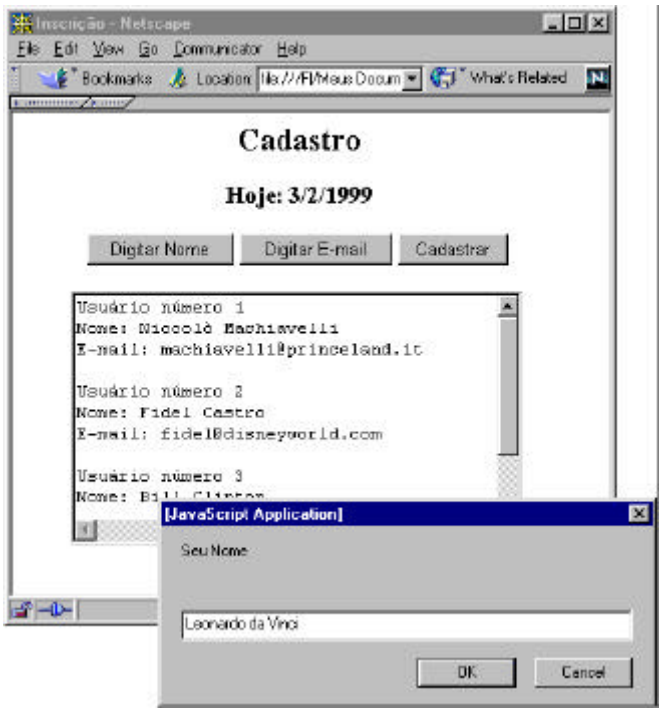
O objetivo desta seção é apresentar uma rápida introdução à linguagem JavaScript. Não são explorados assuntos relacionados à sintaxe da linguagem. O objetivo é dar uma visão geral da linguagem e facilitar a absorção das informações apresentadas nos capítulos posteriores.

A melhor forma de introduzir a linguagem é através de um exemplo. No exemplo a seguir, teremos contato com vários tópicos que veremos em detalhes nos capítulos a seguir, como: sintaxe de expressões, variáveis, objetos, métodos e propriedades, funções e eventos.

Exercício resolvido

Construa uma aplicação Web de entrada de dados que ofereça uma interface semelhante à da figura ao lado. O objetivo é construir uma lista de usuários com seus e-mails para posterior envio pela rede.

Requisitos:



a) Quando o usuário apertar no botão “Digitar Nome”, deve aparecer uma janela de diálogo como mostrada na figura para que ele possa digitar um nome. Apertando no botão “Digitar E-mail”, uma janela semelhante deverá aparecer, para recuperar o e-mail.

b) Apertando o botão “Cadastrar”, os dados digitados mais recentemente devem ser “armazenados” no campo de texto no formato:

```
Usuário número <número>
Nome: <nome do usuário>
E-mail: <e-mail do usuário>
```

O número deve ser incrementado cada vez que um novo usuário for cadastrado. Cada novo usuário cadastrado não deve apagar os anteriores, mas aumentar a lista.

c) A data de hoje deve ser mostrada na página no formato ilustrado na figura.

Não é preciso construir o HTML. Use o arquivo `cap1/formcad.html` que já contém um esqueleto da aplicação (somente os descritores HTML). A solução será apresentada na página seguinte.

Solução

O arquivo sem JavaScript está listado abaixo. Os botões estão presentes mas não respondem a eventos. Não aparece a data de hoje.

```
<html>
<head>
   <title>Inscrição</title>
</head>
<body bgcolor=white>
  <form>
    <h2 align=center>Cadastro</h2>
    <div align=center>
      <p><input type=button value="Digitar Nome">
         <input type=button value="Digitar E-mail">
         <input type=button value="Cadastrar">
      <p><textarea rows=10 cols=40 name=Area></textarea>
    </div>
  </form>
</body>
</html>
```

A primeira alteração, para cumprir o requisito (a), consiste na programação dos eventos `ONCLICK` dos dois primeiros botões. É preciso coletar uma linha de texto do usuário e armazená-la em uma variável global. Para *declarar* uma variável globalmente accessível em JavaScript, usamos a palavra-chave `var` antes do nome escolhido. As declarações devem estar em um bloco `<SCRIPT>` que seja interpretado antes que um botão seja interpretado, então as colocamos no `<head>`:

```
<head>
   <title>Inscrição</title>
   <script>
      var nome
      var email
   </script>
</head>
```

As alterações estão mostradas em negrito.

O próximo passo é programar o evento. A instrução

```
prompt("texto da janela", "texto inicial do campo de entrada")
```

é um *método* JavaScript que abre uma janela de diálogo contendo um campo de entrada de dados (como o da figura). O usuário pode digitar texto no campo disponível e este será devolvido como valor de retorno, caso o OK seja apertado. Para colocar o valor na variável `nome`, podemos fazer:

```
nome = prompt("Digite o Nome", "")
```

Fazemos isto dentro do atributo `ONCLICK` de cada botão, para os dois valores, para que o comando só seja executado quando o usuário apertar o botão:

```
<input type=button value="Digitar Nome"
                  onclick="nome=prompt('Seu Nome', '')">
<input type=button value="Digitar E-mail"
                  onclick="email=prompt('Email', '')">
```

O segundo requisito requer instruções para o atributo `ONCLICK` do *terceiro* botão. Mas é necessário realizar diversas operações:

- incrementar um número (outra variável global)
- construir uma linha de texto (*string*) com os dados lidos
- imprimir os dados em um campo de textos

O ideal, neste caso, é criar uma função que realize as operações acima e chamar esta função a partir do atributo `ONCLICK` do terceiro botão. Acrescentamos a função no bloco `<SCRIPT>` do início da página e construímos o string concatenando as variáveis:

```
<script>
   var nome;
   var email;
   var num = 0;

   function escrever() {
      info = "Usuário número " + (++num) + "\n";
      info += "Nome: " + nome + "\n";
      info += "E-mail: " + email + "\n\n";
   }
</script>
```

O "+", como vimos antes, concatena strings. O valor da variável `num` é incrementado com "`++num`", que é equivalente à expressão "`num = num + 1`". A atribuição "`+=`" *acrescenta* o texto do lado direito à variável `info`, sem apagar o texto existente. "`\n`" representa uma quebra de linha. São armazenadas em `info` quatro linhas de informação, sendo a última em branco.

Falta ainda imprimir os resultados no campo de textos. Alteramos a função `escrever()` para que receba, como argumento, uma referência ao campo de textos (`quadro`), que será passado na chamada da função. Todo campo de textos `<TEXTAREA>` ou `<INPUT TYPE=TEXT>` em JavaScript tem uma propriedade `value`, que dá acesso à sua área editável. A

propriedade `value` é uma variável que pode receber ou conter texto. Dentro da função, concatenamos o texto (em `info`) com o texto já existente na caixa de texto (e visível na tela) em `value`:

```
function escrever(quadro) {
   info = "Usuário número " + (++num) + "\n";
   info += "Nome: " + nome + "\n";
   info += "E-mail: " + email + "\n\n";
   quadro.value += info;
}
```

Uma referência para o campo de textos (`<TEXTAREA>`) pode ser obtido a partir do formulário no qual está contido, através de seu nome. Os nomes dos componentes de um formulário são propriedades do formulário. *Dentro* de qualquer componente, pode-se obter uma referência ao formulário em que está contido usando `this.form`. A expressão:

```
this.form.Area
```

portanto, representa o campo de textos chamado `Area` (`<TEXTAREA NAME="Area">`), localizado *neste* formulário. A chamada acima é válida dentro de qualquer componente do formulário, como o terceiro botão. Assim, podemos chamar a função `escrever()` de dentro do botão e passar como argumento o campo de textos, que é representado pelo objeto de nome `Area` localizado neste formulário::

```
<input type=button value="Cadastrar"
                  onclick="escrever(this.form.Area)">
```

Agora, ao se apertar o botão, a função escrever será chamada. Dentro dela, a variável quadro receberá o valor em this.form.Area, como se tivesse ocorrido uma atribuição do tipo:

```
quadro = this.form.Area
```

O último requisito pede para que a página exiba a data de hoje na página. A exibição não depende de eventos do usuário. Deve ser uma transformação realizada somente na carga e exibição da página, portanto, incluímos o código em um segundo bloco `<SCRIPT>` no corpo da página.

Utilizamos a instrução `new Date()` para obter a data de hoje e passamos para uma variável `hoje`, que criamos. Não é preciso usar a palavra `var` para definir variáveis:

```
hoje = new Date(); // armazena a data de hoje
```

A instrução "`new`" é um operador utilizado para criar novos objetos. `Date()` é uma função especial, chamada de construtora. Ela constroi o novo objeto e define métodos e propriedades que podem ser invocados a partir do objeto. `hoje`, portanto, é um *objeto* que representa a data de hoje e tem métodos, definidos pela função construtora `Date()`. Uma data é um *tipo de dados* abstrato que possui várias propriedades. Só precisamos de três: dia, mês e ano. A única forma de obtê-las em JavaScript é invocando *métodos* sobre `hoje`. Os métodos

retornam propriedades específicas dos objetos onde são invocados. Usamos o operador "." para ter acesso a eles, assim como fizemos com as propriedades:

```
<div align=center>
<script>
   hoje = new Date()
   dia = hoje.getDate()
   mes = hoje.getMonth() + 1
   ano = hoje.getYear() + 1900
   document.write("<h3>Hoje: " + dia + "/" + mes + "/" + ano + "</h3>")
</script>
```

Tivemos que somar `1` ao valor retornado pelo método `getMonth()` porque ele retorna os meses contados a partir de `0` e terminando em `11`. Somamos `1900` ao valor retornado por `getYear()` porque o método retorna o número de anos desde `1900`. A última instrução, imprime os valores na página. Veja o código completo no arquivo `formcodsol.html`.

Exercícios

1.1 Faça com que a propriedade `window.status` (texto da barra de status do browser) seja redefinida com a string 'Um Link' quando o usuário passar o mouse sobre o link (use qualquer página HTML). Utilize os atributos eventos `onmouseover` e `onmouseout` em `<A HREF>`).

1.2 Altere o exercício resolvido para que os dados digitados sejam mostrados em uma janela de alerta (instrução `alert("string")`) em vez de serem mostrados no campo de texto.