

# 10

## *Formulários*

OS COMPONENTES DE FORMULÁRIO SÃO OS OBJETOS HTML mais utilizados em JavaScript. Por componentes de formulário nos referimos a qualquer campo de entrada de dados dentro de um bloco HTML `<FORM>`, como botões, caixas de seleção, caixas de texto e botões de “rádio”. Eles são a principal forma de entrada de dados disponível no HTML.

Os objetos de formulário consistem de doze objetos, situados abaixo de *Form*, no modelo de objetos do documento JavaScript. São referências aos elementos HTML `<INPUT>`, `<SELECT>`, `<OPTION>` e `<TEXTAREA>`.

Uma das principais aplicações do JavaScript, é a validação de dados em aplicações Web. Verificar se os campos de um formulário foram preenchidos corretamente antes de enviar os dados a um programa no servidor é uma tarefa realizada eficientemente com JavaScript. Na maior parte das aplicações, JavaScript é capaz de fazer toda a verificação localmente, economizando conexões de rede desnecessárias.

Neste capítulo, apresentaremos cada um dos objetos JavaScript relacionados a componentes de formulário, como criá-los em HTML e como manipular os dados recebidos por eles.

### *Objeto Form*

O objeto *Form* é o mais alto da hierarquia dos componentes de formulários. Através dele se tem acesso aos componentes existentes dentro de um bloco HTML `<form>`, que podem ser botões, caixas de texto, caixas de seleção, etc.

Não é possível criar um objeto *Form* em JavaScript. Ele precisa existir no código HTML da página através de um bloco `<form> ... </form>`. Este bloco não é visível na página. Serve apenas para agrupar componentes de entrada de dados, torná-los visíveis e associar seus dados a um programa no servidor. A sintaxe do objeto *Form* em HTML está mostrada abaixo. Todos os atributos (em *itálico*) são opcionais para uso em JavaScript:

**<FORM**

```

NAME="nome_do_formulario (usado por JavaScript)"
ACTION="url para onde será enviado o formulário"
METHOD="método HTTP (pode ser GET ou POST)"
ENCTYPE="formato de codificação"
TARGET="janela alvo de exibição da resposta do formulário"
ONRESET="código JavaScript"
ONSUBMIT="código JavaScript" >
... corpo do formulário ...

```

**</FORM>**

Um bloco `<FORM>` deve estar dentro de um bloco `<BODY>`, em HTML. Para ter acesso às propriedades de um objeto *Form*, que é um reflexo de `<FORM>` em JavaScript, é necessário ter acesso ao documento que o contém. Um documento pode ter vários formulários. A propriedade `document`, portanto, possui um vetor com referências a todos os formulários do documento, na ordem em que eles aparecem no código. Este vetor está na propriedade `document.forms`. Para ter acesso ao primeiro formulário de um documento (se ele existir), pode-se usar:

```
objForm = document.forms[0]
```

Pode-se também dar um *nome* ao formulário. Todo componente de `<BODY>` que recebe um nome passa a ser uma propriedade de `document`. O nome é criado através do atributo `NAME` do HTML:

```
<form name="f1"> ... </form>
```

Criado o formulário em HTML, podemos ter acesso a seus métodos e propriedades através do operador ponto (.) usando seu nome ou posição no vetor `document.forms`:

```

x = document.forms[0].propriedade;
document.f1.método();

```

A maior parte das propriedades de *Form* são strings que permitem ler e alterar atributos do formulário definidos no elemento HTML `<FORM>`. A propriedade `elements` é a exceção. Ela contém um vetor com referências para todos os elementos contidos no formulário, na ordem em que aparecem no código.

Propriedade	Descrição
<code>elements</code>	<i>Array</i> . Vetor de elementos do formulário ( <i>read-only</i> ).
<code>elements.length</code>	<i>Number</i> . Número de elementos do formulário ( <i>read-only</i> ).
<code>name</code>	<i>String</i> . Contém o valor do atributo HTML <code>NAME</code> ( <i>read-only</i> ).
<code>action</code>	<i>String</i> . Contém o valor do atributo HTML <code>ACTION</code> .
<code>encoding</code>	<i>String</i> . Contém o valor do atributo HTML <code>ENCTYPE</code> .
<code>method</code>	<i>String</i> . Contém o valor do atributo HTML <code>METHOD</code> .
<code>target</code>	<i>String</i> . Contém o valor do atributo HTML <code>TARGET</code> .

## Elementos de um formulário

As propriedades do objeto *Form* incluem todos os elementos de formulário e imagens que estão *dentro* do bloco `<FORM>` na página HTML. Estes objetos podem ser referenciados pelos seus nomes – propriedades de *Form* criadas com o atributo `NAME` de cada componente, ou através da propriedade `elements` – vetor que contém *todos* os elementos contidos no bloco `<FORM>` na ordem em que aparecem no HTML. Por exemplo, os componentes

```
<form name="f1">
  <input type="text" name="campoTexto">
  <input type="button" name="botao">
</form>
```

podem ser acessados usando o vetor `elements` da forma:

```
txt = document.f1.elements[0]; // primeiro elemento (Text)
bot = document.f1.elements[1]; // segundo elemento (Button)
```

ou usando o seu nome, como propriedade de *Form*:

```
txt = document.f1.campoTexto; // primeiro elemento (Text)
bot = document.f1.botao; // segundo elemento (Button)
```

Usar `elements` em vez do nome de um componente exige uma atenção maior pois a reorganização do formulário irá afetar a ordem dos componentes. Componentes diferentes possuem propriedades diferentes. Se `elements[0]` tem uma propriedade que `elements[1]` não tem, a tentativa de utilizar a propriedade em `elements[1]` causará um erro.

Existem três propriedades que são comuns a *todos* os elementos. Podem ser usadas para identificar o tipo do componente, seu nome e para obter uma referência ao formulário que o contém. Essas propriedades são:

Propriedade	Descrição
<code>form</code>	<i>Form</i> . Referência ao formulário que contém este botão.
<code>name</code>	<i>String</i> . Contém o valor do atributo HTML <code>NAME</code> do componente.
<code>type</code>	<i>String</i> . Contém o tipo do elemento. Pode ter um dos seguintes valores: select-one, select-multiple, textarea, text, password, checkbox, radio, button, submit, reset, file, hidden

Uma forma de evitar o uso acidental de um tipo de componente em lugar de outro, é testando sua propriedade `type`. Antes de usar a propriedade de um componente obtido através de `elements`, teste-o para verificar se realmente se trata do componente desejado:

```
var info;
elemento = document.forms[0].elements[5];
if (elemento.type == "textarea")
  info = elemento.value;
```

```
else if (elemento.type == "select-one")
    info = elemento.options[elemento.selectedIndex].text;
```

O objeto `type` de cada objeto geralmente contém o string contido no atributo `TYPE` do elemento HTML em caixa-baixa. A única exceção é o objeto do tipo *Select* que pode ter dois valores: “*select-multiple*” ou “*select-one*”, identificando listas de seleção múltipla e simples.

## Métodos

Os métodos do objeto *Form* são usados para submeter os dados ao programa no servidor ou reinicializar o formulário com os seus valores originais. Estes métodos podem habilitar outros botões (tipo *Button*) ou qualquer objeto do HTML que capture eventos a implementar a mesma funcionalidade dos botões `<SUBMIT>` ou `<RESET>`:

Método	Ação
<code>reset()</code>	Reinicializa o formulário
<code>submit()</code>	Envia o formulário
<code>focus()</code>	Seleciona o formulário
<code>blur()</code>	Tira a seleção do formulário

Por exemplo, o código abaixo irá enviar os dados do primeiro formulário de uma página para o servidor:

```
<script>
    document.forms[0].submit();
</script>
(...)
```

## Eventos

Os eventos relacionados ao objeto *Form* são a reinicialização do formulário e o envio dos dados ao servidor, desencadeados por botões *Reset* e *Submit* ou por instruções JavaScript. São interceptados pelo código contido nos atributos HTML abaixo:

- `ONSUBMIT` – antes de enviar o formulário ao servidor
- `ONRESET` – antes de inicializar o formulário com os valores *default*

Os eventos acima são sempre tratados *antes* das ações correspondentes acontecerem. O código em `ONSUBMIT`, por exemplo, será interpretado antes que o envio dos dados seja realizado. Se o código JavaScript dentro do `ONSUBMIT` produzir o valor `false`, os dados não serão enviados. Isto permite que os dados sejam verificados antes de enviados.

Estes eventos apresentam várias incompatibilidades entre browsers. O `ONSUBMIT` por exemplo chega a ser inútil tanto em versões do Netscape quanto no Internet Explorer<sup>1</sup>.

## Objetos *Button*, *Reset* e *Submit*

*Button*, *Reset* e *Submit* são todos botões criados em HTML. Têm a mesma aparência na tela, porém efeitos diferentes quando apertados. *Submit* e *Reset* têm eventos já definidos pelo HTML para enviar e limpar o formulário ao qual pertencem. *Button* é inoperante a menos que possua um atributo de eventos `ONCLICK`, com o código que será interpretado quando o usuário apertá-lo. A sintaxe do objeto *Button* em HTML está mostrada abaixo. Apenas os atributos em negrito são obrigatórios:

```
<INPUT TYPE="button"
      NAME="nome_do_botão"
      VALUE="rótulo do botão"
      ONCLICK="Código JavaScript"
      ONFOCUS="Código JavaScript"
      ONBLUR="Código JavaScript">
```



Objetos *Submit* e *Reset* são idênticos. A única diferença é o valor do atributo `TYPE`:

```
<INPUT TYPE="submit" ... >      <!-- Objeto Submit -->
<INPUT TYPE="reset"  ... >      <!-- Objeto Reset  -->
```



Os botões *Submit* e *Reset* também respondem ao evento de se apertar o botão caso tenham um atributo `ONCLICK`. Mas é importante lembrar que *elas já têm eventos* atribuídos pelo HTML, que sempre ocorrerão *depois* dos eventos JavaScript. Por exemplo, qualquer alteração no formulário realizada por um programa no `ONCLICK` de um objeto *Reset*, será anulada pelo evento nativo do *Reset* que reinicializa os campos do formulário aos seus valores originais.

O atributo `VALUE` é opcional em todos os botões. Permite alterar o texto que aparece dentro do mesmo. Em objetos *Reset* e *Submit*, `VALUE` possui um valor *default*. Em objetos *Button*, o *default* para `VALUE` é um string vazio, portanto, a menos que este atributo seja definido, o botão


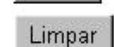
### Button

	<code>&lt;input type=button&gt;</code>
	<code>&lt;input type=button value="Apertar"&gt;</code>

### Submit

	<code>&lt;input type=submit&gt;</code>
	<code>&lt;input type=submit value="Enviar"&gt;</code>

### Reset

	<code>&lt;input type=reset&gt;</code>
	<code>&lt;input type=reset value="Limpar"&gt;</code>

<sup>1</sup> A documentação JavaScript 1.1 também de referência, aparece como evento de `Form` `TYPE=submit`.

aparecerá vazio. A figura ao lado mostra a aparência dos botões em um browser Netscape Navigator rodando em Windows 95 com e sem o atributo `VALUE`.

Qualquer objeto `<INPUT>` é um elemento de formulário e precisa estar dentro de um bloco `<FORM>...</FORM>`, para que seja visível na tela<sup>2</sup>, possa receber dados e ser manipulado como um objeto JavaScript. Os botões podem ser acessados através do vetor `elements`, na ordem em que aparecem no código, ou através do seu nome, especificado pelo atributo `NAME`:

```
<form>
  <input type=button name=b1>    <!--objeto tipo Button -->
  <input type=submit name=b2>    <!--objeto tipo Submit -->
  <input type=reset name=b3>     <!--objeto tipo Reset -->
</form>
```

Se os três botões acima estiverem dentro do primeiro formulário de uma página HTML, o segundo botão pode ser referenciado da forma:

```
but2 = document.forms[0].b2          // ou ...
but2 = document.forms[0].elements[1]
```

Os botões possuem quatro propriedades. Três refletem atributos do HTML e uma é um ponteiro para o formulário que contém o botão. A propriedade `value` é a única que pode ser alterada dinamicamente.

Propriedade	Descrição
<code>form</code>	<i>Form.</i> Referência ao formulário que contém este botão.
<code>value</code>	<i>String.</i> Contém o valor do atributo HTML <code>VALUE</code> que especifica o texto que aparece no botão. Pode ser lida ou alterada.
<code>name</code>	<i>String.</i> Contém o valor do atributo HTML <code>NAME</code> . ( <i>read-only</i> )
<code>type</code>	<i>String.</i> Contém o valor do atributo HTML <code>TYPE</code> . ( <i>read-only</i> )

Com a propriedade `form`, um botão pode subir a hierarquia e ter acesso a outros elementos do formulário no qual está contido. Por exemplo, no código abaixo, o primeiro botão altera o rótulo do segundo botão, ao ser apertado. Para ter acesso a ele, obtém uma referência ao formulário em que está contido através de sua propriedade `form`:

```
<form>
  <input type=button name=b1 value="Editar"
        onclick="this.form.b2.value='Alterar'">
  <input type=submit name=b2 value="Criar">
</form>
```

<sup>2</sup> O Internet Explorer mostra na tela componentes que não estão dentro de `<form>`, mas eles não têm utilidade alguma pois não podem enviar dados ao servidor nem serem manipulados em JavaScript.

Os métodos dos objetos *Button*, *Submit* e *Reset* estão associados a eventos. Permitem simular o evento que ocorre ao clique do mouse, ao ativar e desativar um botão.

Método	Ação
<code>click()</code>	Realiza as tarefas programadas para o clique do botão (executa o código JavaScript contido no atributo <code>ONCLICK</code> sem que o botão precise ser apertado).
<code>focus()</code>	Ativa o botão.
<code>blur()</code>	Desativa o botão.

## Eventos

Os eventos suportados por botões são três. Os atributos HTML abaixo respondem a eventos de botão interpretando o código JavaScript contido neles:

- `ONCLICK` – quando o usuário aperta o botão
- `ONFOCUS` – quando o usuário seleciona o botão.
- `ONBLUR` – quando o usuário seleciona outro componente.

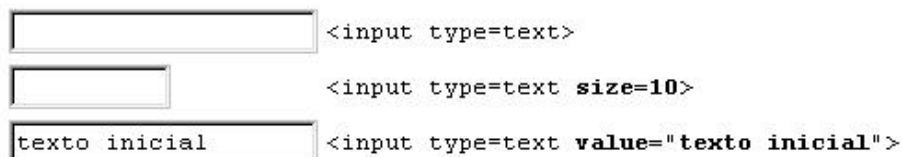
## Objetos *Password*, *Text* e *Textarea*

Os objetos do tipo *Password*, *Text* e *Textarea* são usados para entrada de texto. Possuem as mesmas propriedades. Objetos *Text* e *Password* definem caixas de texto de uma única linha enquanto que os objetos *Textarea* entendem quebras de linha.

Objetos *Text* e *Password* são criados com elementos HTML `<INPUT>` e têm a mesma aparência, mas o texto dos objetos *Password* não é exibido na tela. A sintaxe de um objeto *Text* em HTML é a seguinte:

```
<INPUT TYPE="text"
      NAME="nome_do_campo_de_texto"
      VALUE="texto inicial do campo de textos"
      SIZE="número de caracteres visíveis"
      MAXLENGTH="número máximo de caracteres permitido"
      ONBLUR="código JavaScript"
      ONFOCUS="código JavaScript"
      ONCHANGE="código JavaScript"
      ONSELECT="código JavaScript" >
```

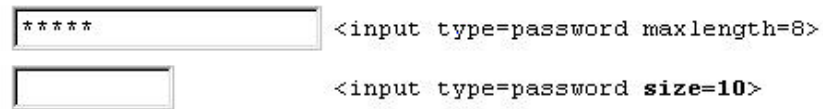
Todos os atributos, exceto o atributo `TYPE` são opcionais. Se `SIZE` não for definido, a caixa de texto terá 20 caracteres de largura. Se `MAXLENGTH` não for definido, não haverá limite para o número de caracteres digitado no campo de textos. A figura abaixo ilustra a aparência de objetos *Text* em um browser Netscape 4.5 rodando em Windows 95.



O objeto *Password* é criado da mesma forma, mas com um atributo `TYPE` diferente:

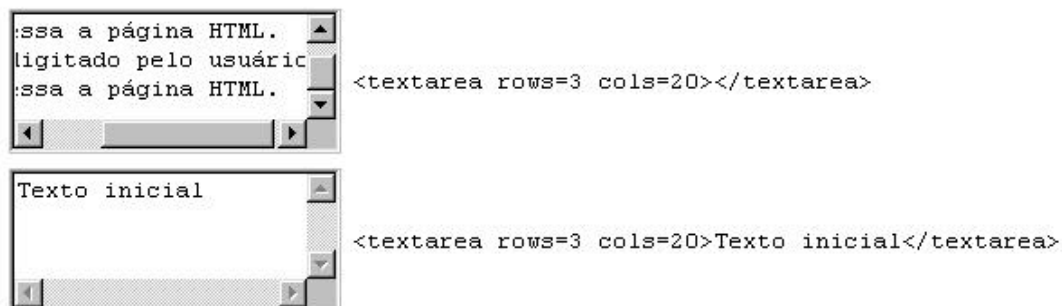
```
<INPUT TYPE="password" ... >
```

Os caracteres do texto digitado em objetos *Password* não aparecem na tela, como mostrado na figura abaixo (Windows95):



A sintaxe para criar um objeto *Textarea* em HTML é a seguinte. Os atributos em negrito são obrigatórios:

```
<TEXTAREA
  ROWS="número de linhas visíveis"
  COLS="número de colunas visíveis"
  NAME="nome_do_campo_de_texto"
  ONBLUR="handlerText"
  ONFOCUS="handlerText"
  ONCHANGE="handlerText"
  ONSELECT="handlerText" >
  Texto inicial
</TEXTAREA>
```



A figura abaixo mostra a aparência de objetos *Textarea*:

Como qualquer outro elemento de formulário, é preciso que os blocos `<TEXTAREA>` e descritores `<INPUT>` estejam dentro de um bloco `<FORM>...</FORM>` para que sejam visíveis na tela, possam receber dados e serem manipulados como objetos JavaScript. Os campos de texto podem ser acessados através do vetor `elements`, na ordem em que aparecem no código, ou através do seu nome, especificado pelo atributo `NAME`:

```
<form>
  <input type=text name=userid>
  <input type=password name=senha>
  <textarea rows=2 cols=10 name=codigos>
```



```

    </textarea>
</form>

```

Se os três campos acima estiverem dentro do primeiro formulário de uma página HTML, o segundo campo pode ser referenciado da forma:

```

texto2 = document.forms[0].senha           // ou ...
texto2 = document.forms[0].elements[1]

```

As propriedades dos campos de texto são as mesmas para *Text*, *Textarea* e *Password*. São todas *read-only* com exceção da propriedade `value`, que pode ser alterada, mudando o conteúdo dos campos de texto dinamicamente.

Propriedade	Descrição
<code>form</code>	<i>Form.</i> Referência ao formulário no qual este elemento está contido.
<code>type</code>	<i>String</i> Valor do atributo <code>TYPE</code> do HTML.
<code>name</code>	<i>String</i> Valor do atributo <code>NAME</code> do HTML.
<code>defaultValue</code>	<i>String</i> Valor default previamente definido no campo <code>VALUE</code> do HTML.
<code>value</code>	<i>String</i> Conteúdo do campo de texto. Valor que <i>pode ser redefinido</i> .

Um objeto *Textarea* pode ter várias linhas de texto e armazenar os caracteres “\n” passados através de sua propriedade `value`, o que não ocorre com objetos *Text* ou *Password*. Se o usuário digitar [ENTER], em um objeto *Textarea* o cursor pulará para a próxima linha. Isto é diferente do que acontece em um objeto *Text*, onde [ENTER] provoca um evento que interpreta o código disponível no atributo `ONCHANGE`.

Os métodos dos objetos *Text*, *Password* e *Textarea* são utilizados para selecionar os componentes e o seu conteúdo. Todos provocam eventos.

Método	Ação
<code>focus()</code>	Ativa o componente.
<code>blur()</code>	Desativa o componente.
<code>select()</code>	Seleciona o campo editável do componente (faz o cursor aparecer piscando dentro do campo de texto ou seleciona o texto nele contido).

## Eventos

Os eventos suportados por objetos *Text*, *TextArea* e *Password* são quatro. Os atributos HTML abaixo respondem aos eventos interpretando o código JavaScript contido neles:

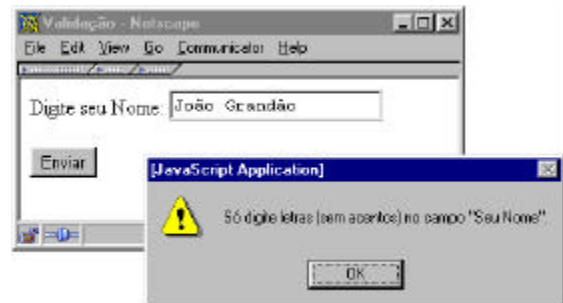
- `ONFOCUS` – quando o usuário seleciona o componente.
- `ONBLUR` – quando o usuário, que tinha o componente selecionado, seleciona outro componente.
- `ONCHANGE` – em *Textarea*, quando o usuário deixa o componente e o valor seu valor difere daquele existente antes da sua seleção; em *Text* e *Password* quando o usuário deixa o componente com valor diferente ou aperta a tecla [ENTER].

- **ONSELECT** – quando o usuário seleciona a área editável do componente.

Para validar dados digitados em campos de texto, é preciso ler sua propriedade `value`, que é um objeto *String*. Qualquer operação que possa ser realizada sobre strings pode ser realizada sobre `value`. Por exemplo, *String* possui uma propriedade `length`, que informa quantos caracteres possui. Através dela pode-se verificar se um campo de textos está vazio usando:

```
if (document.forms[0].senha.length == 0) {
    alert("É preciso digitar uma senha");
}
```

Os métodos `select()` e `focus()` são úteis para rolar a tela e posicionar o cursor em um campo de textos específico, facilitando o acesso a campos que precisam ser corrigidos. O exemplo a seguir mostra uma aplicação desses métodos na validação de um campo de textos. Considere o seguinte formulário HTML que têm a aparência ao lado:



```
<FORM METHOD="POST" ACTION="/cgi-bin/mailForm.pl">
<P>Digite seu Nome:
<INPUT TYPE="TEXT" SIZE="20" NAME="nome">
<P><INPUT TYPE="button" VALUE="Enviar" ONCLICK="valida(this.form)">
</FORM>
```

O botão chama a função `valida()`, passando como argumento uma referência para o formulário, que verifica se o texto digitado no campo do `nome` contém apenas letras do alfabeto ASCII. Para comparar, usamos os métodos `toLowerCase()` e `charAt()` de *String*

```
<script>
function valida(f) {
    var valid = true;
    if (f.nome.value.length <= 0) { // verifica se string está vazio
        valid = false;
    } else { // verifica se usuario digitou caracteres ilegais
        for (i = 0; i < f.nome.value.length; i++) {
            ch = f.nome.value.charAt(i).toLowerCase();
            if (ch < 'a' || ch > 'z') valid = false;
        }
    }
    if (!valid) {
        alert("Só digite letras (sem acentos) no campo \"Seu Nome\".");
        f.nome.focus();
        f.nome.select();
    } else {
        f.submit(); // envia o formulário
    }
}
```

```

    }
  }
</script>

```

## Objeto Hidden

O objeto *Hidden* é um campo de entrada de dados invisível, que o usuário da página não tem acesso. Serve para que o *programador* passe informações ao servidor, ocultando-as no código HTML da página. É bastante útil na transferência de informações entre formulários distribuídos em mais de uma página. Vários desses campos foram usados no capítulo 8 para transferir dados digitados em um formulário para uma página gerada *on-the-fly*. Sua sintaxe é a seguinte:

```

<INPUT TYPE="hidden"
      NAME="nome_do_campo_oculto"
      VALUE="valor armazenado" >

```

Os atributos `NAME` e `VALUE` são opcionais se o campo oculto for manipulado em JavaScript<sup>3</sup>. São elementos de formulário e devem estar dentro de um bloco `<FORM>`. Os campos ocultos podem ser acessados através do vetor `elements`, na ordem em que aparecem no código, ou através do seu nome, especificado pelo atributo `NAME`:

```

<form>
  <input type=text name=userid>
  <input type=password name=senha>
  <input type=hidden name=email value="admin@root.where.org">
  <textarea rows=2 cols=10 name=codigos></textarea>
</form>

```

Se os quatro campos acima estiverem dentro do primeiro formulário de uma página HTML, o valor do campo oculto pode ser obtido de qualquer uma das forma abaixo:

```

valorOculto = document.forms[0].email.value      // ou ...
valorOculto = document.forms[0].elements[2].value

```

As propriedades do objeto *Hidden*, com exceção de `form`, são todas relacionadas aos atributos HTML correspondentes. A única que pode ser alterada é `value`.

Propriedade	Descrição
<code>form</code>	<i>Form</i> . Referência ao formulário no qual este elemento está contido.
<code>name</code>	<i>String</i> Valor do atributo <code>NAME</code> do HTML ( <i>read-only</i> ).
<code>type</code>	<i>String</i> Valor do atributo <code>TYPE</code> do HTML ( <i>read-only</i> ).
<code>value</code>	<i>String</i> Valor do atributo <code>VALUE</code> do HTML. Esta propriedade pode ser redefinida e usada como meio de passar informações entre páginas.

<sup>3</sup> Sem JavaScript, um campo hidden sem os atributos `NAME` e `VALUE` é inútil.

Não há novos métodos nem eventos associados ao objeto *Hidden*.

## Objeto *Checkbox* e *Radio*

Os objetos *Checkbox* e *Radio* representam dispositivos de entrada booleanos cuja informação relevante consiste em saber se uma opção foi selecionada ou não. As únicas diferenças entre os objetos *Checkbox* e *Radio* são a sua aparência na tela do browser e a quantidade de opções que podem conter para cada grupo de dispositivos.

Objetos *Radio* são organizados em grupos de descritores com o mesmo nome (atributo `NAME`). Cada componente aparece na tela como um botão ou caixa de dois estados: *ligado* ou *desligado*. Dentro de um grupo de componentes (todos com o *mesmo* atributo `NAME`), somente um deles poderá estar ligado ao mesmo tempo. A sintaxe de um objeto *Radio* em HTML é a seguinte:

```
<INPUT TYPE="radio"
      NAME="nome_do_componente"
      VALUE="valor (o valor que será enviado ao servidor)"
      CHECKED      <!-- previamente marcado -->
      ONBLUR="código JavaScript"
      ONFOCUS="código JavaScript"
      ONCLICK="código JavaScript" > Rótulo do componente
```

A figura abaixo mostra dois grupos de botões de rádio (em um browser Netscape rodando em Windows95). Observe que os atributos `NAME` distinguem um grupo do outro. O atributo `CHECKED` indica um botão previamente ligado mas que pode ser desligado pelo usuário ao clicar em outro botão.

### GRUPO I

<input type="radio"/>	pela manhã	<input type=radio name=turno value="Manhã"> pela manhã
<input type="radio"/>	à tarde	<input type=radio name=turno value="Tarde"> à tarde
<input checked="" type="radio"/>	à noite	<input type=radio name=turno value="Noite" checked> à noite

### GRUPO II

<input type="radio"/>	Masculino	<input type=radio name=sexo value="M"> Masculino
<input checked="" type="radio"/>	Feminino	<input type=radio name=sexo value="F"> Feminino

Um grupo não é tratado como um elemento de formulário, mas os nomes dos grupos são referências do tipo *Array* em JavaScript. Se os elementos acima fossem os únicos elementos do primeiro formulário de uma página, o primeiro elemento do segundo grupo poderia ser acessado de qualquer uma das formas abaixo:

```
fem = document.forms[0].elements[4];
fem = document.forms[0].sexo[1];
```

O código acima não inclui as informações que mais interessam, que são: 1) o usuário selecionou que opção? e 2) a opção *x* está ou não selecionada? Para responder à essas

questões, precisamos usar as propriedades do objeto *Radio*, que são as mesmas do objeto *Checkbox*, que veremos a seguir. Permitem manipular elementos individuais e grupos.

Cada *Checkbox* aparece na tela como um botão ou caixa que pode assumir dois estados: *ligado* ou *desligado*. Diferentemente dos objetos *Radio*, vários objetos *Checkbox* de um mesmo grupo podem estar ligados ao mesmo tempo, não havendo, portanto, necessidade de organizar tais objetos em um grupo. A sintaxe de um objeto *Checkbox* em HTML é praticamente idêntica à de *Radio*, mudando apenas o valor do atributo `TYPE`:

```
<INPUT TYPE="checkbox" ... > Rótulo do componente
```

A figura abaixo mostra um grupo de caixas de checagem (em um browser Netscape rodando em Windows95). O atributo `CHECKED` indica um botão previamente ligado mas que pode ser desligado pelo usuário ao clicar em outro botão.

```
☑ Café <input type=checkbox name=refeicoes value="Café" checked> Café
☐ Almoço <input type=checkbox name=refeicoes value="Almoço"> Almoço
☑ Jantar <input type=checkbox name=refeicoes value="Jantar"> Jantar
```

Assim como objetos *Radio*, elementos *Checkbox* podem ser manipulados em grupo, embora isto seja desnecessário, já que mais de um valor pode estar associado ao mesmo grupo, como mostrado acima. Se os elementos acima fossem os únicos elementos do primeiro formulário de uma página, o segundo elemento poderia ser acessado de qualquer uma das formas abaixo:

```
almoco = document.forms[0].elements[2];
almoco = document.forms[0].refeicoes[2];
```

Os *Checkboxes* acima poderiam também ser implementados da forma seguinte, sem organizar seus elementos em grupos, com os mesmos resultados:

```
<input type=checkbox name=cafe checked> Café<br>
<input type=checkbox name=almoco> Almoço<br>
<input type=checkbox name=jantar> Jantar
```

O acesso ao segundo elemento agora pode ser feito da forma:

```
almoco = document.forms[0].almoco;
```

As propriedades de *Checkbox* e *Radio* são as mesmas e estão listadas abaixo.

Propriedade	Read/Write	Descrição
<code>type</code>	r	<i>String</i> . Conteúdo do atributo HTML <code>TYPE</code> ( <i>read-only</i> ).
<code>name</code>	r	<i>String</i> . Conteúdo do atributo HTML <code>NAME</code> ( <i>read-only</i> ).
<code>defaultChecked</code>	r	<i>Boolean</i> . Retorna <code>true</code> se o elemento HTML que representa o objeto contém o atributo <code>CHECKED</code> .
<code>checked</code>	r / w	<i>Boolean</i> . Retorna <code>true</code> se um <i>Checkbox</i> ou <i>Radio</i> está atualmente ligado. O valor desta propriedade pode ser

Propriedade	Read/Write	Descrição
		alterado dinamicamente em JavaScript para ligar ou desligar os componentes.
value	r / w	<i>String</i> . Conteúdo do atributo HTML <code>VALUE</code> . O valor desta propriedade pode ser alterado dinamicamente.
length	r	<i>Number</i> . Comprimento do vetor de objetos <i>Radio</i> ou <i>Checkbox</i> . Aplica-se apenas a grupos de elementos identificados pelo nome (não pode ser usado no vetor <code>elements</code> , que refere-se a objetos individuais). <i>Exemplo:</i> <code>document.forms[0].nomegrupo.length</code>

A validação de botões de rádio e caixas de checagem consiste em verificar se uma ou mais opções foram selecionadas. Para isto, basta testar a propriedade `checked` e verificar se ela é `true` nos campos existentes. Veja um exemplo:

```
opcao = null;
turnoArray = document.forms[0].turno; // vetor de botoes de radio
for (i = 0; i < turnoArray.length; i++) {
    if (turnoArray[i].checked) {
        opcao = turnoArray[i];
    }
}

if (opcao == null) {
    alert("É preciso escolher Manhã, Tarde ou Noite!");
}
```

Os métodos suportados pelos objetos *Radio* e *Checkbox*, como os outros elementos de formulário, provocam eventos. Estão listados na tabela abaixo:

Método	Ação
<code>click()</code>	Marca (seleciona) o componente.
<code>focus()</code>	Ativa o componente.
<code>blur()</code>	Desativa o componente.

## Eventos

Os eventos suportados são três. Os atributos HTML abaixo respondem aos eventos interpretando o código JavaScript contido neles:

- `ONCLICK` – quando o usuário liga ou desliga o componente
- `ONFOCUS` – quando o usuário seleciona o componente.

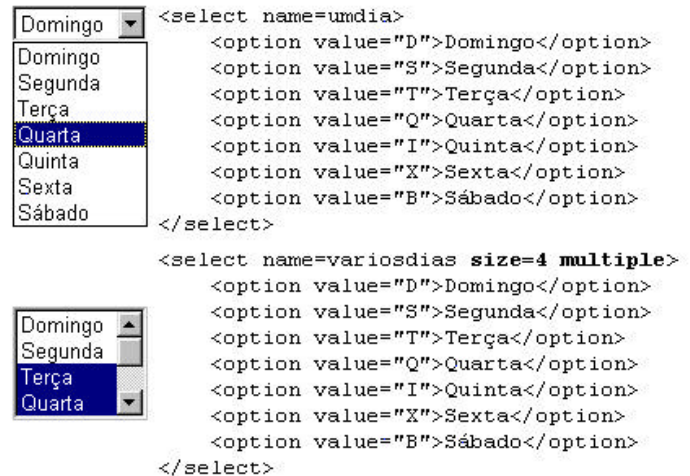
- **ONBLUR** – quando o usuário, que tinha o componente selecionado, seleciona outro componente.

## Objetos Select e Option

Caixas e listas de seleção como as mostradas nas figuras ao lado são representadas em JavaScript por objetos *Select*, que refletem o elemento HTML `<SELECT>`. Um objeto *Select* pode ter uma ou mais objetos *Option*, que refletem elementos `<OPTION>`.

Os objetos *Select* podem ter uma aparência e comportamento diferente dependendo se possuem ou não os atributos `SIZE` e `MULTIPLE`. A figura ao lado ilustra o efeito desses atributos, transformando uma caixa de seleção em uma lista que permite seleção de múltiplas opções.

Objetos *Select* só podem ser criados em HTML. Objetos *Option* podem tanto ser criados em HTML como em JavaScript através do construtor `Option()`. Desta forma, é possível ter listas que crescem (ou diminuem) dinamicamente. A sintaxe de um objeto *Select* em HTML está mostrada abaixo:



```
<SELECT
  NAME="nome_do_componente"
  SIZE="número de opções visíveis"
  MULTIPLE      <!-- Suporta seleção múltipla -->
  ONBLUR="código JavaScript"
  ONCHANGE="código JavaScript"
  ONFOCUS="código JavaScript" >
    <OPTION ...> Opção 1 </OPTION>
    ...
    <OPTION ...> Opção n </OPTION>
</SELECT>
```

Todos os atributos são opcionais. A existência do atributo `NAME` é obrigatória em formulários que terão dados enviados ao servidor. Os objetos *Option* não são elementos do formulário (não podem ser acessados através do vetor `elements`) mas são elementos do objeto *Select* e podem ser acessados pelo seu vetor `options`. A sintaxe de cada objeto *Option* está mostrada abaixo.

```

<OPTION
  VALUE="Valor da opção"
  SELECTED >
  Texto descrevendo a opção
</OPTION>

```

O atributo `VALUE` é opcional. Se os dados forem enviados ao servidor, o texto contido entre `<OPTION>` e `</OPTION>` é enviado somente se um atributo `VALUE` não tiver sido definido. Em JavaScript, ambos podem ser usados para armazenar informações ao mesmo tempo.

Um bloco `<SELECT>` é elemento de formulário e deve estar sempre dentro de um bloco `<FORM>`. Caixas e listas de seleção podem ser acessadas através do vetor `elements`, na ordem em que aparecem no código, ou através do seu nome, especificado pelo atributo `NAME`, por exemplo, considere o código HTML:

```

<form>
  <input type=text name=nome1>
  <select name=umdia>
    <option value="D">Domingo</option>
    <option value="S">Segunda</option>
    <option value="T">Terça</option>
    <option value="Q">Quarta</option>
  </select>
  <input type=text name=nome2>
</form>

```

No código acima há apenas três elementos de formulário. Se o bloco `<form>` acima for o primeiro de um documento, a caixa de seleção poderá ser referenciada da forma:

```

dia = document.forms[0].umdia      // ou ...
dia = document.forms[0].elements[1]

```

Os quatro elementos `<option>` são elementos do objeto *Select*. Para ter acesso ao terceiro objeto `<option>`, pode-se usar o vetor `options`, que é uma propriedade dos objetos *Select*. As duas formas abaixo são equivalentes e armazenam um objeto do tipo *Options* na variável `terca`:

```

terca = document.forms[0].umdia.options[2] // ou ...
terca = document.forms[0].elements[1].options[2]

```

Para ter acesso aos dados armazenados pelo objeto recuperado, é preciso usar propriedades do objeto *Option*. A propriedade `text` recupera o texto entre os descritores `<option>` e `</option>`. A propriedade `value` recupera o texto armazenado no atributo HTML `<VALUE>`:

```

textoVisivel = terca.text;
textoUtil = terca.value;

```



O código acima não obtém as informações que mais interessam, que são: 1) o usuário selecionou qual opção? e 2) a opção de índice *x* ou que contém o texto *y* está ou não selecionada? Para responder à essas questões, e poder realizar outras tarefas, como selecionar opções dinamicamente, precisamos conhecer as propriedades do objeto *Select* e dos seus objetos *Option*. Elas estão listadas nas tabelas abaixo.

A primeira tabela lista as propriedades do objeto *Select*, que são:

Propriedade	Read/Write	Descrição
name	r	<i>String</i> . Equivalente ao atributo HTML NAME.
form	r	<i>String</i> . Referência ao formulário que contém este objeto.
type	r	<i>String</i> . Informa o tipo de lista: <i>select-one</i> , se o elemento HTML não tiver o atributo MULTIPLE, ou <i>select-multiple</i> , se tiver.
options	r	<i>Array</i> . Vetor de objetos <i>Option</i> contidos no objeto <i>Select</i> .
length	r	<i>Number</i> . Número de objetos do vetor options.
selectedIndex	r/w	<i>Number</i> . Índice da opção atualmente selecionada. Para listas múltiplas, contém o <i>primeiro</i> índice selecionado.
options.length	r	<i>Number</i> . Mesmo que length.
options.selectedIndex	r/w	<i>Number</i> . Mesma coisa que selectedIndex.

As propriedades *options* e *selectedIndex* permitem a manipulação dos objetos *Options* contidos no *Select*. As propriedades de *Option* são:

Propriedade	Read/Write	Descrição
index	r	<i>Number</i> . Contém o índice desta opção dentro do vetor options do objeto <i>Select</i> ao qual pertence.
defaultSelected	r	<i>Boolean</i> . Retorna true se o elemento HTML que representa o objeto contém o atributo SELECTED.
selected	r/w	<i>Boolean</i> . Retorna true se objeto está selecionado. Pode ser alterado para selecionar ou deselegionar o objeto dinamicamente.
value	r/w	<i>String</i> . Contém o conteúdo do atributo HTML VALUE (que contém os dados que serão enviados ao servidor).
text	r/w	<i>String</i> . O texto dentro de <option>...</option>, que aparece na lista de opções. Pode ser alterado. Este texto não será enviado ao servidor se existir um atributo VALUE.

A propriedade *selectedIndex* de *Select* contém o índice correspondente à opção atualmente selecionada. Muitas vezes, esta informação é suficiente para uma aplicação de validação de dados, por exemplo, que precisa verificar apenas que o usuário selecionou (ou não selecionou) uma determinada opção.

No trecho de código abaixo, é verificado o valor do índice selecionado de um objeto *Select*, cuja primeira opção não tem valor (apenas informa que o usuário deve selecionar uma opção). As opções válidas, portanto, são as opções selecionadas de índice maior ou igual a 1. Se o valor for zero, significa que o usuário não fez uma seleção válida:

```
if(document.forms[0].sele1.selectedIndex == 0) {
    alert("É preciso selecionar uma opção!");
}
```

Em outros casos, informações relevantes precisam ser recuperadas do objeto *Option* atualmente selecionado. Se o *Select* não permite seleção múltipla, o número armazenado na propriedade `selectedIndex` pode ser usado para recuperar o objeto correspondente no vetor `options`, e a partir daí obter os dados que estão na propriedade `value` ou `text`. Os exemplos abaixo operam sobre os objetos *Select* apresentados no início desta seção:

```
// obtenção do índice atualmente selecionado
indice = document.forms[0].umdia.selectedIndex;

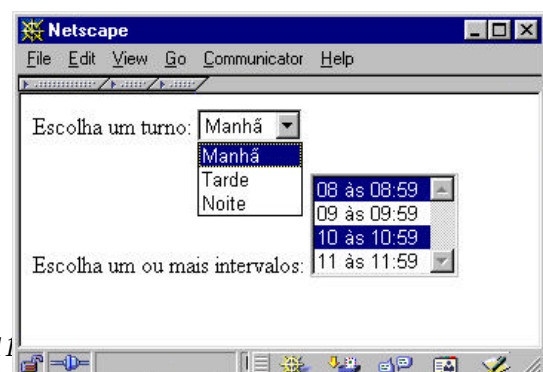
// valor enviado ao servidor: D, S, T, etc.
valorUtil = document.forms[0].umdia.options[indice].value;

// valor mostrado na lista de opções: Domingo, Segunda, Terça, etc.
valorVisivel = document.forms[0].umdia.options[indice].text;
```

Quando o objeto *Select* é uma lista de seleção múltipla, que pode ou não ter mais de um item selecionado, `selectedIndex` retorna apenas o índice do primeiro item selecionado. Neste caso, é preciso verificar uma a uma quais opções estão selecionadas através da propriedade `selected` de cada objeto *Options*. O código abaixo recupera o texto visível de cada uma das opções selecionadas de um objeto *Select* e as armazena em um vetor:

```
objSel = document.forms[1].variosdias;
opcoes = "";
if (objSel.type == "select-one") { // se for caixa de selecao
    opcoes = objSel.options[objSel.selectedIndex].text;
} else { // se for lista de multiplas selecoes
    for (i = 0; i < objSel.length; i++) {
        if (objSel.options[i].selected) {
            opcoes += objSel.options[i].text + "; "
        }
    }
}
opcoes = opcoes.split(";"); // armazena em vetor
```

É possível mudar o texto de uma opção dinamicamente alterando o valor da propriedade `text`. O novo texto ocupará o lugar do antigo, mas o espaço disponível não será aumentado se o texto for maior, e será truncado. No exemplo a seguir (ilustrado na figura ao lado), há dois



objetos *Select*. As opções disponíveis na segunda lista dependem da opção escolhida pelo usuário na primeira lista:

```
<head>
<script>
intervalos = new Array(3);      // vetor 2D c/ intervalos/turno
intervalos[0] = new Array("08 às 08:59", "09 às 09:59", "10 às 10:59", "11 às 11:59");
intervalos[1] = new Array("13 às 13:59", "14 às 14:59", "15 às 15:59", "16 às 16:59");
intervalos[2] = new Array("18 às 18:59", "19 às 19:59", "20 às 20:59", "21 às 21:59");

function setIntervalos(f) {
    idx = f.turno.options.selectedIndex;
    for (i = 0; i < f.horario.length; i++) {
        f.horario.options[i].text = intervalos[idx][i];
    }
}
</script> </head>
<body>
<form> <p>Escolha um turno:
<select name=turno onchange="setIntervalos(this.form)">
    <option>Manhã</option>
    <option>Tarde</option>
    <option>Noite</option>
</select>
<p>Escolha um ou mais intervalos:
<select name=horario size=4 multiple>
    <option> Intervalo 01 </option> <!-- Estes valores irão mudar -->
    <option> Intervalo 02 </option>
    <option> Intervalo 03 </option>
    <option> Intervalo 04 </option>
</select> </form>
</body>
```

É possível acrescentar e remover opções de uma lista *Select* criando e removendo objetos *Option*. Para criar um novo objeto *Option*, usa-se o construtor `Option()` (veja 1 no código abaixo). Depois, pode-se definir valores de interesse, através das propriedades `text`, `value` e `selected` (2). Finalmente, cada objeto *Option* precisa ser colocado na lista, ocupando o final do vetor `options` (3). A função abaixo acrescenta novas opções à lista do exemplo acima:

```
function mais(f) { // f é o formulário onde o Select está contido
    selObj = f.horario;
    novaOp = new Option(); // (1)
    novaOp.text = "Intervalo 0" + (selObj.length + 1); // (2)
    selObj.options[selObj.length] = novaOp; // (3)
}
```

Para remover uma opção de uma lista, basta encurtar o vetor `options`. A função abaixo redefine a propriedade `length` do vetor, encurtando-o cada vez que é chamada:

```
function menos(f) {
    selObj = f.horario;
    if (selObj.length > 0) {
        len = selObj.length - 1;
        selObj.length = len;      // length = length - 1
    }
}
```

O objeto *Select* possui dois métodos. Ambos provocam eventos e são usados para ativar ou desativar o componente:

Método	Ação
focus()	Ativa o componente.
blur()	Desativa o componente.

## Eventos

Os eventos suportados por objetos *Select* são três. Os atributos HTML abaixo respondem aos eventos interpretando o código JavaScript contido neles:

- ONFOCUS – quando o usuário seleciona o componente.
- ONBLUR – quando o usuário, que tinha o componente selecionado, seleciona outro componente.
- ONCHANGE – quando o usuário seleciona uma opção diferente da que estava previamente selecionada no componente.

## Validação de formulários

Nesta seção mostraremos uma das aplicações mais frequentes de JavaScript: a verificação dos dados em um formulário antes do envio ao servidor. O exercício resolvido a seguir utiliza todas as categorias de componentes vistos neste capítulo para desenvolver um formulário de inscrição em um evento, a validadr as informações digitadas pelo usuário.

### Exercício Resolvido

Para este exercício, utilize o esqueleto contido no arquivo `validform.html` (disponível no diretório `cap10/`) que já contém todo o formulário montado em HTML (figura ao

**FORMULÁRIO DE RESERVA**

Preencha o formulário abaixo (um para cada participante):

Participante: Wiley E. Coyote

Empresa: Exterminadores de Papaléguas S/A

Endereço: Rua Ebonia, 235

CEP: 01423-002 Bairro: Jd. Paulist

Cidade: São Paulo Estado: SP

Telefone: (011) 81412 Fax: (011) 81623

E-mail: coyote@acme.com

**Dia 9/11/98 (2a feira)** - Seleccione os minicursos abaixo de acordo com a sua ordem de preferência.

Minicurso 1 - Migrando de C/C++ para Java	RS 170,00
Minicurso 2 - Usando Java em Ambientes Distribuídos	RS 170,00
Minicurso 3 - Desenvolvimento em Java com JavaBeans®	RS 170,00

☒ Dia 10/11/98 (3a feira) RS 160,00

☒ Dia 11/11/98 (4a feira) RS 110,00

☐ Dia 12/11/98 (5a feira) RS 160,00

☐ Participação em todo o Congresso RS 450,00  
Não esqueça de selecionar um minicurso acima!

Inscriver-se Congr

lado). Escreva uma rotina JavaScript que verifique os campos de texto, campos numéricos, caixas de checagem e listas de seleção para que estejam de acordo com as regras abaixo:

- Os únicos campos que podem permanecer vazios são `Empresa`, `Bairro` e `Fax`.
- Os campos `CEP` e `Telefone` só podem conter caracteres numéricos (0 a 9), traço “-”, espaço “ ” e parênteses “(” e “)”
- O campo `E-mail` deve necessariamente conter um caractere “@”
- Se o usuário escolher um minicurso (primeira lista de seleção) e marcar os três dias do congresso (três caixas de checagem), a opção “Participação em todo o congresso” (caixa de checagem) deverá ser selecionada, e as outras três, desmarcadas. Se o usuário marcar “Participação em todo o congresso”, as outras três opções devem ser desmarcadas.
- Se o usuário decidir participar de todo o evento, ele deve escolher um minicurso. Se escolher uma segunda ou terceira opção, deve necessariamente escolher uma primeira.
- Se tudo estiver OK, uma janela de alerta deverá ser apresentada ao usuário informando-o que os dados foram digitados corretamente.

A solução é apresentada na seção seguinte e pode ser encontrada no arquivo `validformsol.html`. Uma outra versão disponível no arquivo `job_form.html` gera uma nova página *on-the-fly* para confirmação e envio ao servidor (em vez de uma janela de alerta), caso os dados sejam digitados corretamente. É proposta como exercício.

## Solução

Toda a validação dos dados ocorre no método `validar(dados)`. O formulário é passado como argumento da função e passa a ser representado pela variável local `dados`. Dentro da função `validar()`, cinco outras funções são chamadas várias vezes. Cada uma realiza uma tarefa solicitada nos requisitos (a) a (e) do exercício, e retorna `true` se os dados estavam corretos. No final da função `validar()`, se nenhuma das chamadas de função retornou `false`, um diálogo de alerta é mostrado avisando que os dados estão corretos:

```
function validar(dados) {
  // requisito (a) - verifica campos vazios
  if (!checkVazios(dados.Nome, "Participante")) return;
  if (!checkVazios(dados.Endereco, "Endereço")) return;
  if (!checkVazios(dados.Cidade, "Cidade")) return;
  if (!checkVazios(dados.CEP, "CEP")) return;
  if (!checkVazios(dados.Telefone, "Telefone")) return;
  if (!checkVazios(dados.Email, "Email")) return;

  // requisito (b) - verifica campos numéricos
  if (!checkNumericos(dados.Telefone) ) return;
  if (!checkNumericos(dados.CEP) ) return;
```

```

//requisito (c) - verifica campo de email
    if (!checkEmail(dados.Email) ) return;

// requisito (d) - organiza selecoes
    checkDias(dados.Tudo, dados.Dia10, dados.Dia11, dados.Dia12, dados.Opcao_1);

// requisito (e) - garante selecao de minicursos
    if (!checkMinis(dados.Tudo, dados.Opcao_1, dados.Opcao_2, dados.Opcao_3))
        return;

// requisito (f) -se chegou até aqui, tudo eh true: diga OK!
    alert("Dados digitados corretamente!");
}

```

O primeiro requisito (a) deve verificar se seis campos não estão vazios. A função `checkVazios(elemento, "string")` verifica o comprimento do valor de cada elemento *Text*. Se o texto na propriedade `value` do objeto tiver comprimento inferior a um caractere, a função retorna `false`:

```

function checkVazios(elemento, nome) {
    if (elemento.value.length < 1) {
        alert("O campo " + nome + " não pode ser vazio!");
        elemento.focus();
        elemento.select();
        return false;
    } else return true;
}

```

O segundo requisito (b) é cumprido pela função `checkNumericos()` que verifica se os campos numéricos contém caracteres ilegais. Os caracteres legais são “ ” (espaço), “-”, “(”, “)” e os dígitos de 0 a 9. Os elementos `Telefone` e `CEP` são passados para a função `checkNumericos()` que faz a verificação. Se houver qualquer outro caractere que não os citados no campo de textos, o formulário não será enviado e a função retornará `false`:

```

function checkNumericos(elemento) {
    for (i = 0; i < elemento.value.length; i++) {
        ch = elemento.value.charAt(i);
        if ((ch < '0' || ch > '9') && ch != '-'
            && ch != ' ' && ch != '(' && ch != ')') {
            alert("O campo " + elemento.name +
                " só aceita números, parênteses, espaço e '-'");
            elemento.focus();
            elemento.select();
            return false;
        }
    }
    return true;
}

```

O requisito (c) pede para verificar se a informação de email contém o caractere “@”. Se o índice do substring do string `value` contendo o “@” for “-1”, o caractere não existe no

campo `value` do objeto, portanto não é um e-mail válido e a função `checkEmail()` retornará `false`:

```
function checkEmail(elemento) {
    if (elemento.value.lastIndexOf("@") == -1) {
        alert("Formato ilegal para E-mail");
        elemento.focus();
        elemento.select();
        return false;
    } else return true;
}
```

O requisito (d) pede duas coisas: para deselegionar as três caixas de checagem intermediárias, se a caixa “Tudo” estiver selecionada e para marcar a caixa “Tudo” se o usuário tiver escolhido um minicurso e marcado todas as caixas intermediárias. A função `checkDias()` recebe como argumentos 4 objetos *Checkbox* (correspondentes às caixas intermediárias e a caixa “Tudo”) e um objeto *Select* (correspondente à seleção do primeiro minicurso) e faz todas as alterações.

```
function checkDias(ck, ck10, ck11, ck12, m1) {
    if (ck.checked) {          // caixa "Tudo" está marcada...
        ck10.checked = false;  // desmarque as intermediárias...
        ck11.checked = false;
        ck12.checked = false;
    }
    // todas as intermediarias ligadas e minicurso escolhido...
    if (ck10.checked && ck11.checked && ck12.checked && m1.selectedIndex != 0) {
        ck10.checked = false;
        ck11.checked = false;  // desmarque todas e...
        ck12.checked = false;
        ck.checked = true;     // ... marque a caixa "Tudo"
    }
}
```

Finalmente, a função `checkMinis()` cumpre o requisito (e) recebendo como argumentos o *Checkbox* que representa a opção de participar de todo o evento e os três componentes *Select*. Verifica se um minicurso foi selecionado (`m1`), quando o usuário selecionou `ckTudo` e garante que uma primeira opção de minicurso foi selecionada (`m1`) quando há uma segunda (`m2`) ou terceira (`m3`) seleção. O índice zero em um *Select* corresponde a uma opção não selecionada:

```
function checkMinis(ckTudo, m1, m2, m3) {
    if (m1.selectedIndex == 0 && ckTudo.checked) {
        alert("Por favor, selecione um minicurso!");
        m1.focus();
        return false;
    }
}
```

```

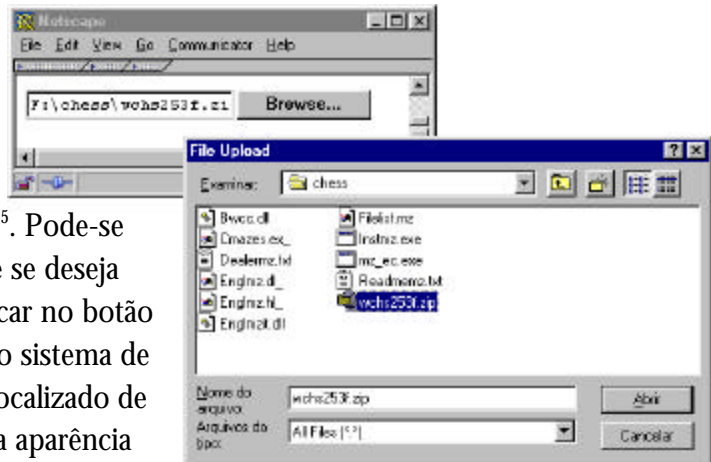
else if ((m2.selectedIndex != 0 && m1.selectedIndex == 0) ||
        (m3.selectedIndex != 0 && m1.selectedIndex == 0)) {
    alert("Por favor, selecione um minicurso como primeira opção!");
    m1.focus();
    return false;
} else return true;
}

```

Vários melhoramentos podem ser introduzidos nesta aplicação. Mudando a ordem de alguns testes na função `valida()`, por exemplo (que foram agrupados de acordo com os requisitos) pode tornar a interface mais eficiente. Algumas outras modificações são propostas em exercícios. A solução completa deste exercício está no arquivo `validformsol.html`.

## Objeto File

*File* (ou *FileUpload*<sup>4</sup>) representa um dispositivo de entrada que permite o envio de um arquivo no cliente ao servidor. Na tela do browser, aparece como uma caixa de texto e um botão com o rótulo “Browse...” ou “Procurar...”<sup>5</sup>. Pode-se digitar o caminho absoluto ao arquivo que se deseja enviar ao servidor na caixa de texto ou clicar no botão “Browse...” e fazer aparecer um diálogo do sistema de arquivos, que permite que o arquivo seja localizado de forma interativa. A figura ao lado mostra a aparência do objeto *File* no browser Netscape em Windows 95 e a janela de diálogo que aparece ao se apertar o botão “Browse...”.



O objeto *File* é criado em HTML através de um elemento `<INPUT>`. A sintaxe geral do componente está mostrada abaixo:

```

<INPUT TYPE="file"
      NAME="nome_do_componente"
      ONBLUR="código JavaScript"
      ONFOCUS="código JavaScript"
      ONCHANGE="código JavaScript" >

```

Um objeto *File* só poderá ser manipulado em JavaScript se seu descritor `<INPUT>` estiver dentro de um bloco `<FORM>...</FORM>`. Como qualquer outro componente, é um elemento do formulário e pode ser acessado através do vetor `elements`, ou através do seu

<sup>4</sup> Nomenclatura usada pela Netscape.

<sup>5</sup> Depende da versão e fabricante do browser. Este rótulo não pode ser mudado.



nome, especificado pelo atributo `NAME`. O trecho de código abaixo mostra como acessar um elemento *File* chamado `nomeFup` e que é o sétimo elemento do primeiro formulário de uma página:

```
fup = document.forms[0].nomeFup      // ou ...
fup = document.forms[0].elements[6]
```

Para que seja possível a transferência de arquivos ao servidor, o servidor deverá permitir o recebimento de dados. O browser também deve passar os dados ao servidor usando o método de requisição `POST` e no formato `multipart/form-data`, que pode ser definido através do atributo `ENCTYPE` de `<FORM>` ou na propriedade `encoding`, de *Form*:

```
<FORM ENCTYPE="multipart/form-data" ACTION="..." METHOD="POST">
  <INPUT TYPE="file">
</FORM>
```

Todas as propriedades dos objetos *File* são somente-leitura. Estão listadas na tabela abaixo:

Propriedade	Descrição
<code>form</code>	<i>Form</i> . Referência ao formulário que contém este componente.
<code>name</code>	<i>String</i> . Contém o valor do atributo HTML <code>NAME</code> . ( <i>read-only</i> )
<code>type</code>	<i>String</i> . Contém o valor do atributo HTML <code>TYPE</code> . ( <i>read-only</i> )
<code>value</code>	<i>String</i> . Contém o texto no campo de textos do objeto, que corresponde ao arquivo a ser enviado ao servidor. É <i>read-only</i> por questões de segurança.

*File* só possui os dois métodos listados abaixo. Ambos provocam eventos de ativação/desativação do componente.

Método	Ação
<code>focus()</code>	Ativa o botão.
<code>blur()</code>	Desativa o botão.

## Eventos

Os eventos suportados são dois. Os atributos HTML abaixo respondem aos eventos interpretando o código JavaScript contido neles:

- `ONFOCUS` – quando o usuário seleciona a caixa de textos ou o botão de *File*.
- `ONBLUR` – quando o usuário, que tinha o componente selecionado, seleciona outro componente.

## Exercícios

- 10.1 Após a validação dos dados no exercício resolvido, uma janela de alerta aparece na tela informando que os dados foram digitados corretamente. Acrescente uma nova função para substituir o alerta. Esta nova função deverá gerar uma nova página *on-the-fly* (veja o exercício resolvido do capítulo 8) com os dados que o usuário digitou. Deve dois botões na página de confirmação: um para voltar e alterar os dados e outro para enviar os dados ao servidor. Garanta que os dados sejam preservados em campos *Hidden* na nova página para que o programa no servidor possa usá-los.
- 10.2 No formulário do exercício resolvido, é preciso escolher três minicursos. Altere o código de forma que não seja possível o usuário definir o segundo ou terceiro sem antes definir o primeiro (na versão atual isto só ocorre quando o formulário está para ser enviado). Garanta também que: a) quando o usuário fizer uma seleção na primeira lista, ele não consiga fazer a mesma seleção nas outras listas, e b) que a terceira opção seja automaticamente escolhida assim que o usuário fizer duas seleções.