EVANDRO MANARA MILETTO
SILVIA DE CASTRO BERTAGNOLLI

DESENVOLVIMENTO DE SOFTWARE II

INTRODUÇÃO AO DESENVOLVIMENTO WEB COM HTML, CSS, JAVASCRIPT E PHP

EIXO INFORMAÇÃO E COMUNICAÇÃO

série tekne

bookman

D451 Desenvolvimento de software II [recurso eletrônico] : introdução ao desenvolvimento web com HTML, CSS, JavaScript e PHP / Organizadores, Evandro Manara Miletto, Silvia de Castro Bertagnolli. – Dados eletrônicos. – Porto Alegre : Bookman, 2014.

Editado também como livro impresso em 2014.
ISBN 978-85-8260-196-9

1. Informática – Desenvolvimento de software. 2. HTML. 3. CSS. 4. JavaScript. 5. PHP. I. Miletto, Evandro Manara. II. Bertagnolli, Silvia de Castro.

CDU 004.41

Catalogação na publicação: Ana Paula M. Magnus – CRB 10/2052

Júlia Marques Carvalho da Silva
Evandro Manara Miletto

» capítulo 5

Comportamento com JavaScript

O desenvolvimento de sites, como já visto, é realizado por um conjunto de linguagens de programação. Uma delas é o JavaScript, por meio da qual é possível desenvolver pequenos trechos de programação. O JavaScript tem a função de controlar o comportamento da página, permitindo, por exemplo, validar formulários, alterar textos, ocultar e mostrar objetos, alterar estilos, executar pequenas operações e manipulações junto ao navegador. Neste capítulo, você será apresentado ao JavaScript, às suas principais características e a eventos que possibilitarão controlar o comportamento das suas páginas Web.

Objetivos de aprendizagem

» Incluir código JavaScript em páginas HTML.

» Reconhecer a sintaxe da linguagem JavaScript.

» Desenvolver pequenos trechos de código em JavaScript.

» Identificar os eventos dos elementos HTML que podem acionar funções JavaScript.

» Validar informações enviadas por um formulário.

» Modificar o estilo e conteúdo de elementos HTML por meio do JavaScript.

» **DICA**
Ao tentar executar um arquivo JavaScript (extensão .js) em seu navegador, ele apenas exibirá o conteúdo desse arquivo, em vez de executar suas funções ou eventos.

Introdução

Linguagens para desenvolvimento de sites podem ser divididas em duas categorias básicas: as que rodam no **lado cliente** e as que rodam do **lado servidor**. As linguagens do lado cliente vistas até o momento (HTML, CSS e agora o JavaScript) são aquelas executadas utilizando apenas o navegador do computador do usuário. Uma vez carregadas, não necessitam de novas requisições ao servidor Web.

» CURIOSIDADE

O JavaScript, que surgiu em 1995, é uma linguagem de programação interpretada que é executada diretamente em navegadores Web.

JavaScript é uma das linguagens mais populares da Web e se caracteriza por ter tipagem dinâmica, por ser baseada em objetos, orientada a eventos (p.ex., movimentos do mouse, pressionar botão, arrastar e soltar, etc.) e realizar avaliação em tempo de execução. JavaScript é padronizada pela ECMA International (*European Computer Manufacturers Association*) nas especificações ECMA-262[3] e é baseada em ECMA Script.

» **NO SITE**
A ECMA International é uma associação, fundada em 1961, dedicada à padronização das Tecnologias da Informação e Comunicação. Para saber mais, acesse o ambiente virtual de aprendizagem Tekne: **www.bookman.com.br/tekne**.

Inclusão do JavaScript em páginas Web

Antes de aprender a linguagem JavaScript, é importante saber como incluí-la nas páginas HTML. Há duas formas de introduzir um código JavaScript no HTML, ambas utilizando a TAG `<script>`. Uma delas é demonstrada no exemplo a seguir.

EXEMPLO

```
<html>
  <head>
     <title>Primeira página com JS</title>
  </head>
  <body>
   Texto no HTML.<br/>
   <script type="text/javascript">
     document.write("Parte gerada via JavaScript");
   </script>
  </body>
</html>
```

Figura 5.1 Janela com diferentes formas de escrita.
Fonte: dos autores.

Note que, na TAG `<script>`, o atributo `type` indica a linguagem de programação que será descrita na sequência. Quando o trecho de código é adicionado dessa forma, a TAG pode ser incluída em qualquer trecho do HTML. Entretanto, faz mais sentido tê-la dentro da TAG `body`, principalmente se houver saída de tela, como ocorreu no caso exemplificado.

Porém, a forma mais comum de adicionar o código JavaScript no HTML é separando-os em dois arquivos: a marcação fica no HTML, e a programação, no JavaScript. A ligação entre eles se dá também pela TAG `<script>`. Nesse caso, será indicado o endereço onde está o arquivo JavaScript, conforme explanado no exemplo a seguir.

» EXEMPLO

```
<html>
  <head>
   <title>Segunda página com JS</title>
  </head>
  <script src="meuscript.js"></script>
  <body>
   Texto no HTML.<br/>
  </body>
</html>
```

O arquivo meuscript.js é um arquivo texto comum, no qual é inserido qualquer código JavaScript. Por isso, os arquivos podem ser abertos e programados em qualquer editor de texto, como o Bloco de Notas, nativo do sistema operacional. O arquivo de texto pode ser um programa estruturado ou separado por funções e classes. Ao longo deste capítulo, será explicado melhor como as funções e classes funcionam.

» Criação dos primeiros códigos

Para quem já programou em outras linguagens, é fácil compreender e programar a sintaxe do JavaScript. Observe o exemplo a seguir com uma função que recebe dois números e retorna a soma deles.

» EXEMPLO

```
function soma(){
  var numero1 = 2;
  var numero2 = 5;
  var soma = 0;
  soma = numero1 + numero2;
  return soma;
}
```

Note que foram criadas três variáveis (`numero1`, `numero2` e `soma`). Para criá-las, foi utilizada a palavra reservada `var`; entretanto, seu uso não é obrigatório, ou seja, utilizando a variável pela primeira vez ela já estará disponível na linguagem. Na sequência, será solicitado que o `numero1` e o `numero2` sejam somados, e que seu resultado seja armazenado na variável `soma`. Esta será retornada, como resultado da execução da função.

O exemplo anterior (`soma`) mostra a criação de uma função. Geralmente, os códigos desenvolvidos em JavaScript são organizados dessa forma, em pequenas funções, as quais serão chamadas por eventos nos elementos da página HTML. Veja o exemplo a seguir:

» DICA
Uma variável pode ter diferentes tipos de dados.

» ATENÇÃO
Nem toda função precisa ter retorno.

» EXEMPLO

```
function mostraSituacao(media){
  if(media >= 7){
  alert("Aprovado");
  }
  else{
    alert("Reprovado");
  }
}
```

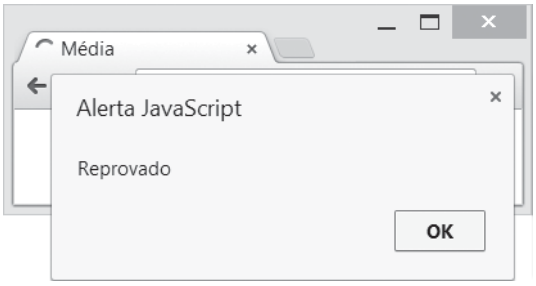


Figura 5.2 Janela pop-up resultante da função `alert()`.
Fonte: dos autores.

» IMPORTANTE
Antes de começar a programar, é importante verificar se há algum erro no código JavaScript, pois isso pode impedir a sua execução.

Perceba que a função `mostraSituacao` recebe um valor por parâmetro na variável `media`. Se o valor for maior ou igual a 7, a função apresentará uma tela de alerta com a mensagem "Aprovado". Caso contrário, a mensagem que surge é "Reprovado".

Agora é a sua vez!

1. Crie uma função `verificaFrete` que receberá um valor por parâmetro. Se esse valor for igual ou superior a 100, a função deve retornar 0. Se o valor for menor do que 100, deverá retornar 10% do valor informado.
2. Um laptop, se forem compradas menos de cinco unidades, custa R$ 1.200,00. Se forem adquiridas cinco ou mais unidades, o produto custará R$ 1.050,00. Escreva uma função que receba por parâmetro o número de laptops comprados, calculando e exibindo, em uma mensagem de alerta, o custo total da compra.

Exibição de informações ao usuário: escrevendo no HTML e usando as janelas de diálogo

Nos exemplos da seção anterior, foram apresentadas duas formas de exibir uma informação ao usuário usando o JavaScript: o `document.write` e as janelas de diálogo (`alert`).

`document.write`: permite que uma informação seja escrita dentro da página HTML. Essa informação pode estar diretamente no corpo da página, na TAG `body`, ou em uma área específica, em uma `div`.

Janelas de diálogo: oferecem outra forma de comunicar dados ao usuário. Existem três tipos de janelas: de alerta (`alert`), de confirmação (`confirm`) e de entrada de dados (`prompt`). Veja, no quadro a seguir, os exemplos de como utilizá-las.

EXEMPLO

```
function alerta(){
  alert("Alerta");
}
```

Figura 5.3 Janela de diálogo de alerta.
Fonte: dos autores.

```
function confirma(){
  resposta = confirm("Confirma");
  if(resposta==1){
    return true;
  }
  else {
    return false;
  }
}
```

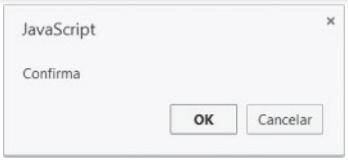


Figura 5.4 Janela de diálogo de confirmação.
Fonte: dos autores.

```
function entrada(){
  nome = prompt("Nome:");
  return nome;
}
```

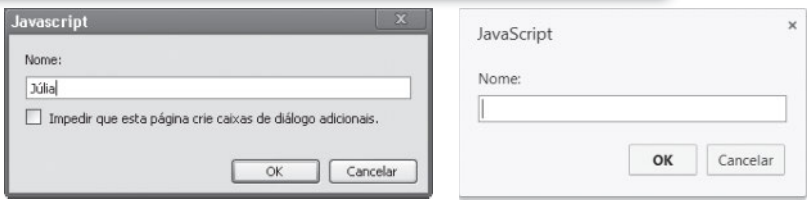


Figura 5.5 Janela de diálogo de entrada de dados.
Fonte: dos autores.

Podemos imaginar, agora, um exemplo que combine todas as janelas. Primeiro, seria solicitado o endereço de uma pessoa e, logo depois, que ela confirmasse o endereço. Se o endereço não for confirmado, o preenchimento será solicitado novamente. Ao confirmar, seria mostrado um alerta informando que a página HTML seria modificada, alterando o seu conteúdo. Veja a seguir como ficaria o código desse exemplo.

» EXEMPLO

```
function perguntaEndereco(){
  do {
    endereco = prompt("Insira o seu endereço:");
    confirma = confirm("Seu endereço é: " + endereco);
  } while(!confirma);
  alert("A página será alterada...");
  document.write("Seu endereço é "+endereco+".");
}
```

Fizemos, então, o código e o colocamos dentro de uma função. Também já sabemos como incluir um arquivo JavaScript em um HTML. Após ter digitado o código acima e criado a página HTML, talvez não tenha funcionado. Por quê? Porque apenas criamos a função, ainda não a "chamamos" dentro da página HTML. Para isso, é necessário que a função seja acionada em algum momento da página, ou seja, quando ocorrer um evento na página, a função iniciará.

» NO SITE
Para saber quais elementos são suportados por cada evento, acesse o ambiente virtual de aprendizagem Tekne.

» Eventos

Existem vários eventos que podem ocorrer dentro de uma página HTML. Podemos clicar com o mouse sobre algum elemento, podemos digitar informações em um formulário por meio do teclado e podemos até mesmo sair da página. Todas essas ações são chamadas de eventos, e elas podem ocorrer dentro das diversas TAGs do HTML.

Já temos a função `perguntaEndereco`, e ela será salva no arquivo perguntaendereco.js. Neste exemplo, ela será chamada tão logo a página HTML seja carregada. Para isso, será utilizado o evento `onLoad` na TAG `body`, ou seja, a função `perguntaEndereco` será chamada ao carregar o corpo da página.

» EXEMPLO

```
<html>
  <head>
    <title>Pergunta o Endereço</title>
  </head>
  <script src="perguntaendereco.js"></script>
  <body onLoad="perguntaEndereco();">
  </body>
</html>
```

Os **eventos** são ações realizadas dentro de uma página HTML. Por esse motivo, são vinculados às TAGs HTML. Dependendo da ação, o evento pode, ou não, ser chamado. Há eventos que são realizados com o mouse (ver Quadro 5.1), outros, com o teclado (ver Quadro 5.2) e, ainda outros, vinculados a objetos (ver Quadro 5.3) ou formulários (ver Quadro 5.4).

Quadro 5.1 » Eventos de mouse

Propriedade	Ocorre quando...
onClick	O usuário clica em um elemento.
onDblClick	O usuário clica duas vezes em um elemento.
onMouseDown	O usuário pressiona o botão do mouse sobre um elemento.
onMouseMove	O ponteiro do mouse está em movimento sobre o elemento.
onMouseOver	O ponteiro do mouse está sobre o elemento.
onMouseOut	O usuário move o ponteiro do mouse para fora do elemento.
onMouseUp	O usuário solta o botão do mouse sobre um elemento.

Quadro 5.2 » Eventos de teclado

Propriedade	Ocorre quando...
onKeyDown	O usuário está pressionando uma tecla.
onKeyPress	O usuário pressiona uma tecla.
onKeyUp	O usuário solta a tecla (previamente pressionada).

Quadro 5.3 » Eventos de objetos ou frames

Propriedade	Ocorre quando...
onAbort	Há a interrupção no carregamento de uma imagem.
onError	Uma imagem não carrega adequadamente.
onLoad	Um objeto foi carregado.
onResize	Um documento visualizado é redimensionado.
onScroll	Um documento visualizado é rolado pela barra de rolagem do navegador.
onUnload	Uma página deixa de ser exibida.

Quadro 5.4 » Eventos de formulários

Propriedade	Ocorre quando...
onBlur	Um campo do formulário perde o foco.
onChange	O conteúdo de um campo do formulário é alterado.
onFocus	Um campo do formulário ganha foco.
onReset	Um formulário é restaurado, isto é, tem seus campos limpos.
onSelect	Um usuário seleciona algum texto dentro de um campo do formulário.
onSubmit	Um formulário é enviado.

A seguir, apresentamos modelos de como usar eventos com diferentes elementos. Primeiro, apresentamos a página HTML e, em seguida, o código JavaScript. Nosso primeiro exemplo é o de troca de imagens. Quando o mouse passa sobre a imagem inicial, ela é trocada, retornando à ilustração inicial quando o mouse não está mais sobre ela.

» EXEMPLO

trocaImagem.html

```
<html>
  <head>
    <title>Troca de Imagens</title>
  </head>
  <script src="trocaImagem.js" language="JavaScript"></script>
  <body>
  Passe o mouse sobre a imagem...<br />
  <img src="carrinho_vazio.jpg" onMouseOver="mostraCheio(this);"
onMouseOut="mostraVazio(this);">
  </body>
</html>
```

Os eventos `onMouseOver` e `onMouseOut` chamam as funções `mostraQuadro` e `mostraTesoura`, respectivamente. Elas passam por parâmetro o objeto `this`, o qual significa que deve ser enviado o próprio elemento, isto é, a TAG `image`. Com ela, será possível trocar o endereço da imagem carregada, conforme demonstrado abaixo.

EXEMPLO

trocaImagem.js

```
function mostraCheio(imagem){
  imagem.src="carrinho_adicionar.jpg";
}
function mostraVazio(imagem){
  imagem.src="carrinho_vazio.jpg";
}
```

Figura 5.6 Janelas resultantes dos eventos `onMouseOver` e `onMouseOut`.
Fonte: dos autores.

No próximo exemplo, há um formulário com o campo `<textarea>`. Com isso, todo caractere digitado será forçado para permanecer em caixa alta.

» EXEMPLO

trocaMaiuscula.js

```
function maiuscula(texto){
  texto.value = texto.value.toUpperCase();
}
```

trocaMaiuscula.html

```
<html>
  <head>
    <title>Texto</title>
  </head>
  <script src="trocaMaiuscula.js"></script>
  <body>
    <form>
      Letra maiúscula:<br />
      <textarea name="letra" onKeyUp="maiuscula(this);"></textarea><br />
    </form>
  </body>
</html>
```

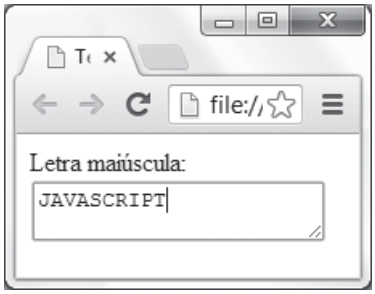


Figura 5.7 Janela resultante da função `toUpperCase()`.
Fonte: dos autores.

Outro exemplo é o de habilitar e desabilitar campos de um formulário. Observe as telas no quadro a seguir. Ao carregar, a página apresenta um formulário em que o campo texto não permite digitação de conteúdo. Entretanto, ao selecionar a opção "Habilita", o campo é liberado. E, ao selecionar a opção "Desabilita", o campo é limpo e não é permitida a digitação de texto algum.

EXEMPLO

habilita.html

```
<html>
  <head>
    <title>Habilita</title>
  </head>
  <script src="habilita.js"></script>
  <body>
    <form name="formulario">
      <input type="radio" name="habilita" value="sim"
             onClick="HabilitarCampo(1);">Habilita<br/>
      <input type="radio" name="habilita" value="nao"
             onClick="HabilitarCampo(0);" checked>Desabilita<br/>
      Campo: <input type="text" name="nome" disabled>
    </form>
  </body>
</html>
```

habilita.js

```
function HabilitarCampo(opcao){
  if(opcao){
    document.formulario.nome.disabled = false;
  }
  else{
    document.formulario.nome.value = "";
    document.formulario.nome.disabled = true;
  }
}
```

Figura 5.8 Janela exemplo da função `HabilitarCampo()`.
Fonte: dos autores.

Note que a função `HabilitarCampo` recebe por parâmetro a variável `opcao`. Ela indica se o campo `nome` deve ser habilitado para edição. Para obtê-lo, é necessário localizá-lo dentro da página (`document`) que contém um formulário (`formulario`) e então informar o nome do campo (`nome`). A partir daí, podemos acessar o atributo `disabled` e alterá-lo.

No próximo exemplo, apresentamos um cálculo do IMC. Para isso, o usuário informa seu peso e sua altura e, ao clicar no botão, o índice é calculado. Note que os valores informados nos campos do formulário são recebidos pelo JavaScript como `string`. Logo, é necessário convertê-los para ponto flutuante (`float`). Veja esse modelo no quadro a seguir.

» EXEMPLO

imc.js

```
function calculaIMC(){
  peso = parseFloat(document.formulario.peso.value);
  altura = parseFloat(document.formulario.altura.value);
  resultado = peso/(altura*altura);
  resultado = resultado.toFixed(2);
  alert("Seu IMC é: "+resultado);
}
```

imc.html

```
<html>
  <head>
    <title>IMC</title>
  </head>
  <script src="imc.js"></script>
  <body>
    <form name="formulario">
      Peso: <input type="text" name="peso"><br>
      Altura: <input type="text" name="altura"><br>
      <input type="button" value="Calcular" onclick="calculaIMC();">
    </form>
  </body>
</html>
```

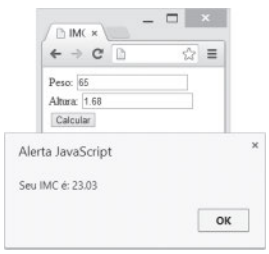
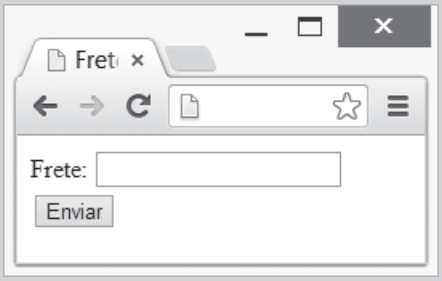


Figura 5.9 Janela com exemplo da aplicação do IMC.
Fonte: dos autores.

Agora é a sua vez!

Desenvolva o formulário abaixo de forma que, ao clicar no botão "Enviar", seja exibido o valor do frete em uma janela de alerta.



Validação dos campos dos formulários

Nos exemplos apresentados nas seções anteriores, foram utilizados formulários para a entrada de dados e execução de alguma ação, como um cálculo. Como nem sempre o usuário preenche o formulário corretamente, pode acontecer de o campo ficar vazio ou com uma informação diferente da solicitada. Assim, para que não haja campos em branco ou com informações equivocadas, é necessário proceder à validação.

Com a chegada do HTML 5 (embora ainda não oficializado) em 2010, novos tipos de campos foram disponibilizados para certificar que o usuário digite a informação solicitada. Há campos que verificam automaticamente se o valor é uma data, hora, email, número, etc. Entretanto, nem todos os navegadores os implementam, o que torna necessária uma validação mais específica.

No exemplo a seguir, é possível verificar se o campo foi preenchido. Note que será usada a codificação em HTML 5, buscando validar os campos duplamente: pelo HTML e pelo JavaScript. No campo `input`, será adicionado o atributo `required`, tornando obrigatório o seu preenchimento. Ao submeter o formulário, é necessário executar a função `valida`. Na função, deverá ser verificado se o valor preen-

chido no campo está preenchido. Nesse caso, a função retornará `true`, e, então, o formulário é submetido. Caso o campo não esteja preenchido, uma mensagem de alerta é exibida, o foco é direcionado para o campo que deve ser revisado pelo usuário e é impedida a continuidade da submissão, retornando `false`.

» EXEMPLO

busca.html

```
<html>
  <head>
    <title>Busca</title>
  </head>
  <script src="busca.js"></script>
  <body>
    <form name="formulario" action="busca.php" onSubmit="return valida();">
      Procurar: <input type="text" name="busca" required><br>
      <input type="submit">
    </form>
  </body>
</html>
```

busca.js

```
function valida(){
  if(document.formulario.busca.value == ""){
    alert("Preencha o campo BUSCA corretamente");
    document.formulario.busca.focus();
    return false;
  }
  return true;
}
```

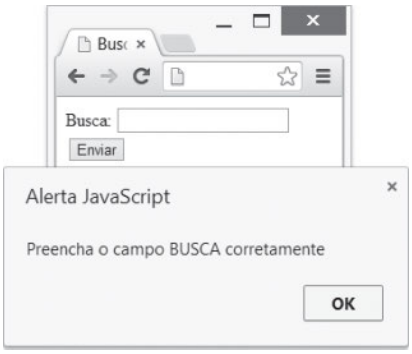


Figura 5.10 Janela com exemplo da aplicação da busca.
Fonte: dos autores.

No próximo exemplo, que exibe um formulário de busca, o campo deverá ser composto de pelo menos 3 caracteres. Assim, evita-se que palavras pequenas sejam informadas, retornando um grande volume como resultado. Logo, será alterada a função `valida`, para que também verifique quantos caracteres foram informados, impedindo o envio do formulário caso tenha menos de 3 caracteres.

EXEMPLO

busca.js

```
function valida(){
  if(document.formulario.busca.value == ""){
    alert("Preencha o campo BUSCA corretamente");
    document. formulario.busca.focus();
    return false;
  }
  if(document.formulario.busca.value.length < 3){
    alert("Informe pelo menos 3 letras!");
    document.formulario.busca.focus();
    return false;
  }
  return true;
}
```

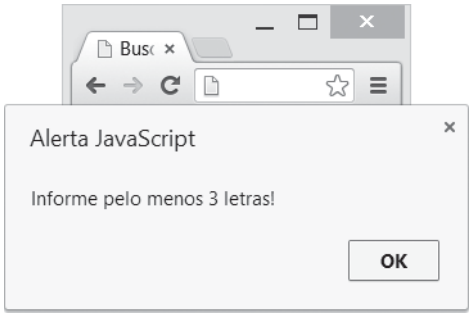


Figura 5.11 Janela com exemplo da aplicação da busca completo.
Fonte: dos autores.

Validação de campos com expressões regulares

Certamente você já se deparou com formulários como o da Figura 5.12. Ele pede ao usuário que forneça uma informação em determinado formato. Pode ser que, enquanto seja preenchido, o próprio formulário autocomplete alguns campos

Figura 5.12 Exemplo de validação de data.
Fonte: dos autores.

(com pontos ou barras) ou não permita certos caracteres (p.ex., um campo de CPF não permite a digitação de letras e símbolos). Esse tipo de validação envolve algoritmos mais complexos, que verificam, a cada digitação, a necessidade de validar e modificar a informação.

Uma forma de simplificar essa validação é utilizando expressões regulares. Elas permitem identificar a ocorrência de um conjunto de caracteres por meio de uma expressão representada por letras, números, caracteres e símbolos.

Para facilitar a compreensão, apresentamos uma função mais simples, de validação de data. O formato desejado é o seguinte: 01/01/2013, representando o dia 1° de janeiro de 2013. Se o usuário não preencher o campo corretamente, a mensagem "Preencha o campo DATA corretamente" é apresentada. Se o campo foi preenchido, mas não no formato esperado, uma mensagem equivalente é exibida. Por fim, se a data estiver correta, a mensagem "Data válida" é mostrada e, então, o formulário pode ser enviado. Observe o código desse exemplo a seguir.

» ATENÇÃO
Para verificar se o conteúdo de uma variável é um número, pode ser utilizada a expressão:
Números inteiros: /^\d+$/;
Números decimais (com vírgula): /^[+-]?((\d+|\d{1,3}(\.\d{3})+)(\,\d*)?|\,\d+)$/;

» EXEMPLO

data.html

```
<html>
  <head>
    <title>Data</title>
  </head>
  <script src="data.js"></script>
  <body>
      <form name="formulario" method="post" onSubmit="return validaData();">
      Data (DD/MM/AAAA): <input type="text" name="data" maxlength="10"><br>
      <input type="submit" value="Validar">
    </form>
  </body>
</html>
```

EXEMPLO

data.js

```
function validaData(){
  if(document.formulario.data.value == "" ||
     document.formulario.data.value.length != 10){
    alert("Preencha o campo DATA corretamente");
    document.formulario.data.focus();
    return false;
  }
// O comando abaixo deve ser digitado continuamente em uma linha
  expReg = /^((0[1-9]|[12]\d)\/(0[1-9]|1[0-2])|30\/(0[13-9]|1[0-2])|31\/(0
[13578]|1[02]))\/\d{4}$/;
  if (document.formulario.data.value.match(expReg) &&
      document.formulario.data.value != ''){
    alert("Data válida");
    return true;
    }
  else{
    alert("Formato inválido de data");
    document.formulario.data.focus();
    return false;
  }
}
```

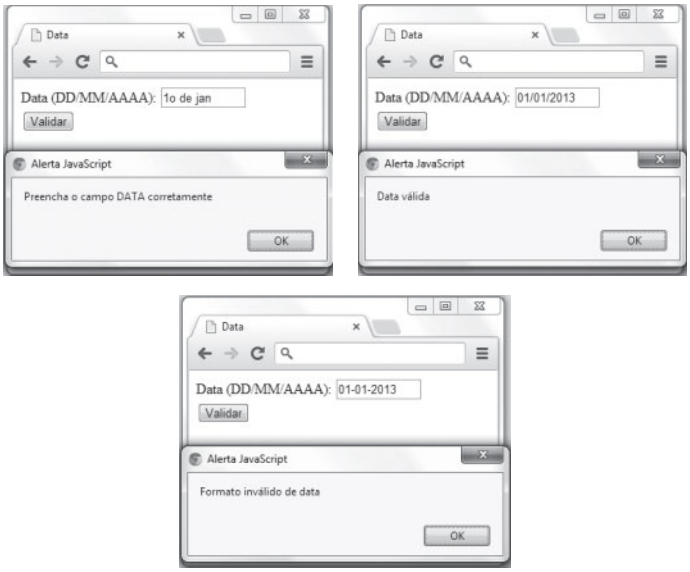
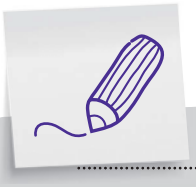


Figura 5.13 Exemplo de validação de data.
Fonte: dos autores.

Em caso de navegadores que ainda não implementaram o HTML5, recomendamos o uso das expressões a seguir para validar as informações:

- Hora (24 horas): /^([0-1]\d|2[0-3]):[0-5]\d$/;
- Email: /^[\w!#$%&'*+\/=?^`{|}~-]+(\.[\w!#$%&'*+\/=?^`{|}~-]+)*@(([\w-]+\.)+[A-Za-z]{2,6}|\[\d{1,3} (\.\d{1,3}){3}\])$/;
- CPF: /^\d{3}\.\d{3}\.\d{3}\-\d{2}$/
- CNPJ: /^\d{2}\.\d{3}\.\d{3}\/\d{4}\-\d{2}$/
- Telefone (DDD e 8 ou 9 dígitos): /[1-9][1-9] [2-9]?[0-9]{4}-[0-9]{4}/
- CEP: /^[0-9]{5}-[0-9]{3}$/



Agora é a sua vez!

1. Construa um formulário contendo o campo de CEP e a quantidade de produto, com base na Figura 5.12.
   - Você deve verificar se o CEP contém dígitos inteiros (note que o mesmo foi separado em 2 campos). A validação deverá ser feita ao clicar no botão "Calcular frete".
   - O campo de quantidade de produtos deve aceitar, no máximo, 1 caractere (logo, o usuário poderá comprar até 9 unidades do mesmo produto). A validação deve ser feita quando o campo perder o foco.

Figura 5.14 Formulário de carrinho de compras.
Fonte: dos autores.

Agora é a sua vez!

2. Construa também formulários para cadastramento:
   a) Ao clicar no botão "Acessar", o email e a senha devem ser validados quanto ao seu correto preenchimento.
   b) Ao clicar no botão "Quero me cadastrar", o nome também deve ser informado.

   Os campos são validados da seguinte forma:
   • O email deve ser válido.
   • A senha deve conter entre 6 e 10 caracteres. Para isso, utilize o código /[a-zA-Z0-9]{6,10}/; incluindo, pelo menos, uma letra minúscula, uma maiúscula e um número. Neste caso, use a expressão /(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])/.
   • O nome deverá ser composto por, ao menos, um nome e um sobrenome, cujos tamanhos mínimos são de 2 caracteres cada.
3. Para finalizar a compra, elabore o formulário de pagamento por meio de cartão de crédito. Logo, ao clicar em "Confirmar pagamento", você deve certificar-se de que o comprador optou por uma das bandeiras dos cartões e preencheu todos os campos, seguindo as seguintes instruções:
   • O nome do titular deve conter apenas letras, sendo que, ao digitar cada uma delas, a letra é convertida em letra maiúscula. A quantidade máxima é de 19 caracteres, incluindo espaços em branco.
   • O número do cartão deve conter, exatamente, 16 números, sem espaços, e o código de segurança deve conter, exatamente, 3 números. Conforme a bandeira escolhida, o número deverá ser validado:
     - Visa: /^4[0-9]{12,15}$/
     - Mastercard: /^5[1-5]{1}[0-9]{14}$/
     - American Express: /^3(4|7){1}[0-9]{13}$/

Validação de campos com máscaras

Nas seções anteriores, foram apresentadas algumas formas de validar o conteúdo dos campos. Primeiro, foi visto como verificar se o campo está preenchido. Depois, como realizar a verificação utilizando as expressões regulares que ajudam a testar se o valor informado corresponde ao esperado.

Agora, você irá aprender como deixar os formulários mais seguros e facilitar seu preenchimento pelo usuário. Talvez você já tenha percebido, em alguns sites, a existência de **máscaras**. Trata-se de uma característica que permite a validação dos campos quando o usuário faz a digitação. Para implementar o uso de máscaras, serão usados os conhecimentos apresentados até aqui.

No exemplo a seguir, há o desenvolvimento de um formulário que contém um único campo de CEP. Nele, é permitido apenas a digitação de números inteiros e, na 6ª posição, é incluído o hífen a fim de separar os dígitos.

» EXEMPLO

cep.html

```
<html>
  <head>
    <title>CEP</title>
  </head>
  <script src="cep.js"></script>
  <body>
    <form action="#" method="post" onSubmit="return camposPreenchidos(this);">
      CEP: <input type="text" name="cep" onkeydown="cep(this);"
            onkeypress="cep(this);" onkeyup="cep(this);" maxlength="9" ><br />
      <input type="submit" name="botao" value="Enviar">
    </form>
  </body>
</html>
```

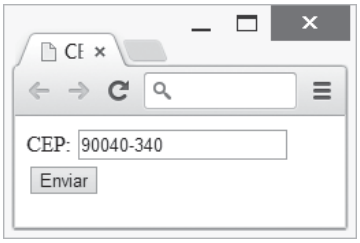


Figura 5.15 Exemplo de validação de CEP.
Fonte: dos autores.

cep.js

```
function cep(v){
  // Removendo os caracteres que não forem números
  v.value=v.value.replace(/\D/g,"");
  // Adicionando o hífen na 6ª posição
  v.value=v.value.replace(/^(\d{5})(\d)/,"$1-$2");
}
function camposPreenchidos(form){
  if(form.cep.value != "" && form.cep.value.length == 9){
    return true;
  }
  return false;
}
```

No arquivo JavaScript, foram criadas duas funções: `cep` e `camposPreenchidos`. A `cep` é chamada a cada interação do teclado com o campo `cep`, verificando o conteúdo digitado e completando com o hífen. Já a função `camposPreenchidos` é invocada quando se tentar submeter o formulário, impedindo que o campo `cep` seja enviado sem que esteja completo, ou seja, com os 9 caracteres.

Veja a seguir mais algumas funções JavaScript para a criação de máscaras.

» IMPORTANTE

Note que a função `cep` é chamada em três eventos de teclado. Trata-se de uma ação importante para o funcionamento correto do código.

» EXEMPLO

mascaras.js

```
// Números
function numeros(v){
  // Remove os caracteres não numéricos
  v.value=v.value.replace(/\D/g,"");
}

// Data
function data(v){
  v.value=v.value.replace(/\D/g,"");
  //Adiciona a barra entre o dia e o mês
  v.value=v.value.replace(/^(\d{2})(\d)/,"$1/$2");
  //Adiciona a barra entre o mês e o ano
  v.value=v.value.replace(/(\d{2})(\d)/,"$1/$2");
}

//Telefone com DDD
function telefone(v){
  v.value=v.value.replace(/\D/g,"");
  //Adiciona parênteses no DDD
  v.value=v.value.replace(/^(\d\d)(\d)/g,"($1) $2");
  //Adiciona hífen no número do telefone
  v.value=v.value.replace(/(\d{4})(\d)/,"$1-$2");
}

// CPF
function cpf(v){
  v.value=v.value.replace(/\D/g,"");
  //Adiciona ponto após os três primeiros números
  v.value=v.value.replace(/^(\d{3})(\d)/,"$1.$2");
  //Adiciona ponto após os seis primeiros números
  v.value=v.value.replace(/(\d{3})(\d)/,"$1.$2");
  //Adiciona o hífen antes dos últimos 2 caracteres
  v.value=v.value.replace(/(\d{3})(\d{1,2})$/,"$1-$2");
}
```

Agora é a sua vez!

1. Construa um formulário contendo os campos de CPF, nome, sexo, data de nascimento e telefone. Você precisa validar cada campo, de modo que apenas os valores esperados sejam informados. Além disso, você deve impossibilitar a submissão do formulário se algum dos campos não estiver preenchido.
2. A seguir, desenvolva o formulário contendo os campos endereço, número, complemento, referência, CEP, bairro, cidade e estado. Você deve validar cada campo, de modo que apenas os valores esperados sejam informados. Além disso, você deve impossibilitar a submissão se algum dos campos não estiver preenchido.

Cookies

Os cookies são pequenos arquivos gerenciados pelos navegadores, capazes de armazenar conteúdos textuais das páginas Web. Esses conteúdos são guardados para utilização posterior pelas páginas que compõem um site. Com eles, pode-se personalizar o site por meio de ações como:

- Preencher um formulário com os dados informados no último acesso.
- Armazenar os itens selecionados em um carrinho de compras.
- Armazenar as preferências de layout do site.

Embora os cookies não sejam uma tecnologia recente e sejam utilizados amplamente, a sua implementação usando JavaScript é trabalhosa. Note na sequência a seguir.

EXEMPLO

cookies.js

```
function setcookie(nome, valor, expira) {
  //Cria um objeto da classe data (classe nativa do javascript)
  dia = new date();
  //Adiciona a quantidade de dias que o cookie estará válido à data atual
  dia.setdate(dia.getdate() + expira);
```

EXEMPLO

```
    //Prepara a string que armazenará o cookie
    var valor = escape(valor) + ((expira==null) ? "" : "; expires="
              +dia.toutcstring());
    //Adiciona a string ao cookie
    document.cookie = nome + "=" + valor;
}
function getcookie(nome) {
    //Obtém a string com os cookies
    var cookies = document.cookie;
    //Localiza onde está o cookie cujo nome foi passado por parâmetro
    var inicio = cookies.indexof(" " + nome + "=");
    //Se estiver no início
    if (inicio == -1) {
      //Posiciona no início do cookie
      inicio = cookies.indexof(nome + "=");
    }
    //Ou se não foi localizado (não existe)
    if (inicio == -1) {
      cookies = null;
    }
    else {
      //Localiza a posição inicial onde está o valor do cookie
      inicio = cookies.indexof("=", inicio) + 1;
      //Localiza a posição final onde está o valor do cookie
      var fim = cookies.indexof(";", inicio);
      if (fim == -1) {
        fim = cookies.length;
      }
    //Enfim, obtém o valor do cookie
    cookies = unescape(cookies.substring(inicio,fim));
    }
    //Retorna o valor
    return cookies;
}
```

Utilizando as funções do arquivo cookies.js, o processo de armazenar e recuperar cookies torna-se mais fácil. Observe a Figura 5.16 e o exemplo a seguir.

O formulário de login apresentado na Figura 5.16 (`login.html`) é feito a partir do fornecimento do email e da senha do usuário. Se estiverem corretos, a área restrita do site (aqui chamada de `restrita.html`), a qual necessita da identificação do usuário, pode ser acessada.

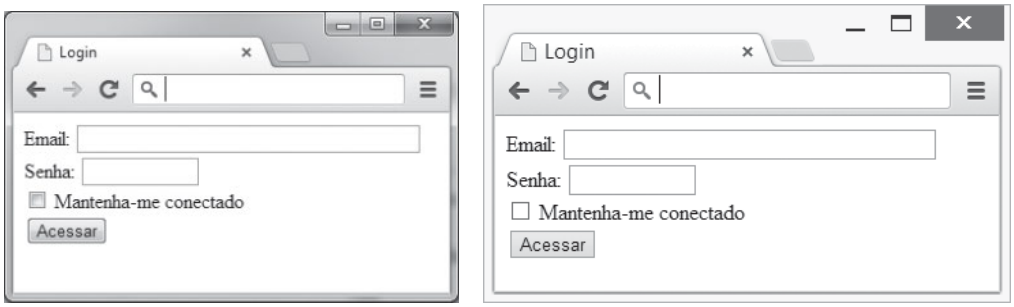


Figura 5.16 Exemplo de login.
Fonte: dos autores.

Entretanto, o exemplo tem características especiais, que provavelmente você já deve ter visto em alguns sites. Foi acrescentado um botão de "Mantenha-me conectado", que, se selecionado, permite o acesso direto à área restrita em um próximo acesso ao site. Essa funcionalidade é implementada com cookies, pois, ao selecioná-la, essa opção é gravada no navegador do usuário por um tempo determinado (10 dias).

Por fim, para facilitar o entendimento, o acesso será liberado caso seja informado um email válido qualquer e utilizada a palavra "JavaScript" como senha. Veja no quadro a seguir o código desse exemplo.

» EXEMPLO

Login.html

```
<html>
  <head>
    <title>Login</title>
  </head>
  <script src="cookies.js"></script>
  <script src="login.js"></script>
  <body onLoad="verificarLogado()";>
    <form action="restrita.html" method="post"
          onSubmit="return camposPreenchidos(this);">
    Email: <input name="email" maxlength="100" size="40"><br />
    Senha: <input type="password" name="senha" maxlength="10" size="10"><br />
```

EXEMPLO

```
    <input type="checkbox" name="conectado" value="1">
    Mantenha-me conectado<br />
    <input type="submit" name="botao" value="Acessar">
    </form>
  </body>
</html>
```

Login.js

```
// Ao carregar a página, verificamos se o usuário já acessou o site anteriormente e optou por
// mantê-lo conectado.
function verificarLogado(){
  // Verifica a existência do login
  if(getCookie("email") != null){
    // Em caso afirmativo, direciona para a página restrita
    window.location.href = "restrita.html";
  }
}
// Ao submeter o formulário, verifica-se o preenchimento dos campos
function camposPreenchidos(form){
  // Expressão regular para verificar se o email é válido (mantenha a expressão em uma linha
  // única e contínua)
  erEmail = /^[\w!#$%&'*+\/=?^`{|}~-]+(\.[\w!#$%&'*+\/=?^`{|}~-]+)*
@(([\w-]+\.)+[A-Za-z]{2,6}|\[\d{1,3}(\.\d{1,3}){3}\])$/;
  // Verifica se o campo está vazio ou não corresponde ao email válido
  if(form.email.value == "" || !form.email.value.match(erEmail)){
    alert("Preencha o campo EMAIL corretamente");
    return false;
  }
  // Verifica se a senha está correta
  if(form.senha.value != "JavaScript"){
    alert("Preencha o campo SENHA corretamente");
    return false;
  }
  // Verifica se a opção de manter conectado foi selecionada
  if(form.conectado.checked){
    // Em caso afirmativo, cria um cookie com o email do usuário
    setCookie("email", form.email.value, 10);
  }
  return true;
}
```

Manipulação de estilos com JavaScript

No Capítulo 4, você aprendeu um pouco sobre CSS, a folha de estilos responsável pelo layout das páginas HTML. É possível alterar a cor, a fonte e o tamanho por meio da combinação de CSS com JavaScript. Ou seja, ao invocar um evento do JavaScript, é possível acessar um elemento e, então, solicitar que seu estilo seja modificado.

O exemplo da Fig. 5.10 referente ao assunto se baseia no formulário de busca que já foi visto antes. Entretanto, agora vai ser adicionado estilo. Para isso, quando o usuário deixar o campo de busca vazio ou com menos de 3 caracteres, o navegador irá exibir um alerta e também tornar o campo colorido a fim de destacá-lo. Veja no quadro a seguir como fica o código.

EXEMPLO

busca.html

```
<html>
  <head>
    <title>Busca</title>
  </head>
  <script src="busca.js"></script>
  <body>
    <form name="formulario" action="busca.php" onSubmit="return valida();">
      Procurar: <input type="text" name="busca" required><br>
      <input type="submit">
    </form>
  </body>
</html>
```

busca.js

```
function valida(){
  // Localize o elemento e, então, chame a propriedade style e qual elemento deseja
  // modificar, como cor da borda, cor de fundo, etc.
  document.formulario.busca.style.borderColor = "#FFFFFF";
  document.formulario.busca.style.backgroundColor = "#FFFFFF";
  if(document.formulario.busca.value == ""){
```

EXEMPLO

```
        alert("Preencha o campo BUSCA corretamente");
        document.formulario.busca.style.borderColor = "#FF4500";
        document.formulario.busca.style.backgroundColor = "#FFFFE0";
        document.formulario.busca.focus();
        return false;
      }
      if(document.formulario.busca.value.length < 3){
        alert("Informe pelo menos 3 letras!");
        document.formulario.busca.style.borderColor = "#FF4500";
        document.formulario.busca.style.backgroundColor = "#FFFFE0";
        document.formulario.busca.focus();
        return false;
      }
      return true;
    }
```

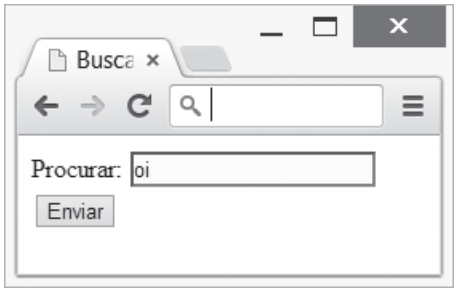


Figura 5.17 Exemplo de Busca
Fonte: dos autores.

Observe que praticamente todos as propriedades CSS podem ser modificadas via JavaScript. Geralmente, são mantidos os mesmos nomes e, quando houver hífen separando o nome (p.ex., border-color), basta removê-lo. Na Internet, é possível encontrar tabelas de correspondências entre as nomenclaturas em CSS e JavaScript.

LEITURAS RECOMENDADAS

MORRISON, M. *Use a cabeça! Javascript*. Rio de Janeiro: Alta Books, 2008

W3SCHOOLS. *JavaScript tutorial*. [S.l.: s.n., 2013]. Disponível em: <http://www.w3schools.com/js/>. Acesso em: 15 out. 2013.

YANK, K.; ADAMS, C. *Só JavaScript*: tudo o que você precisa saber sobre JavaScript e partir do zero. Porto Alegre: Bookman, 2009.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.