O guia amigo do seu cérebro

Use a Cabeça!

(Head First)

JavaScript

Aumente sua experiência como usuário com a interatividade das páginas da web

Aprenda conceitos importantes do JavaScript para o seu cérebro

Faça o código JavaScript funcionar num piscar de olhos

Destrinche o HTML com a ajuda do DOM

Pare de ter medo de manipular eventos – não machuca nada

Exercite sua mente com dezenas de quebra-cabeças e exercícios

O'REILLY

ALTA BOOKS

Michael Morrison

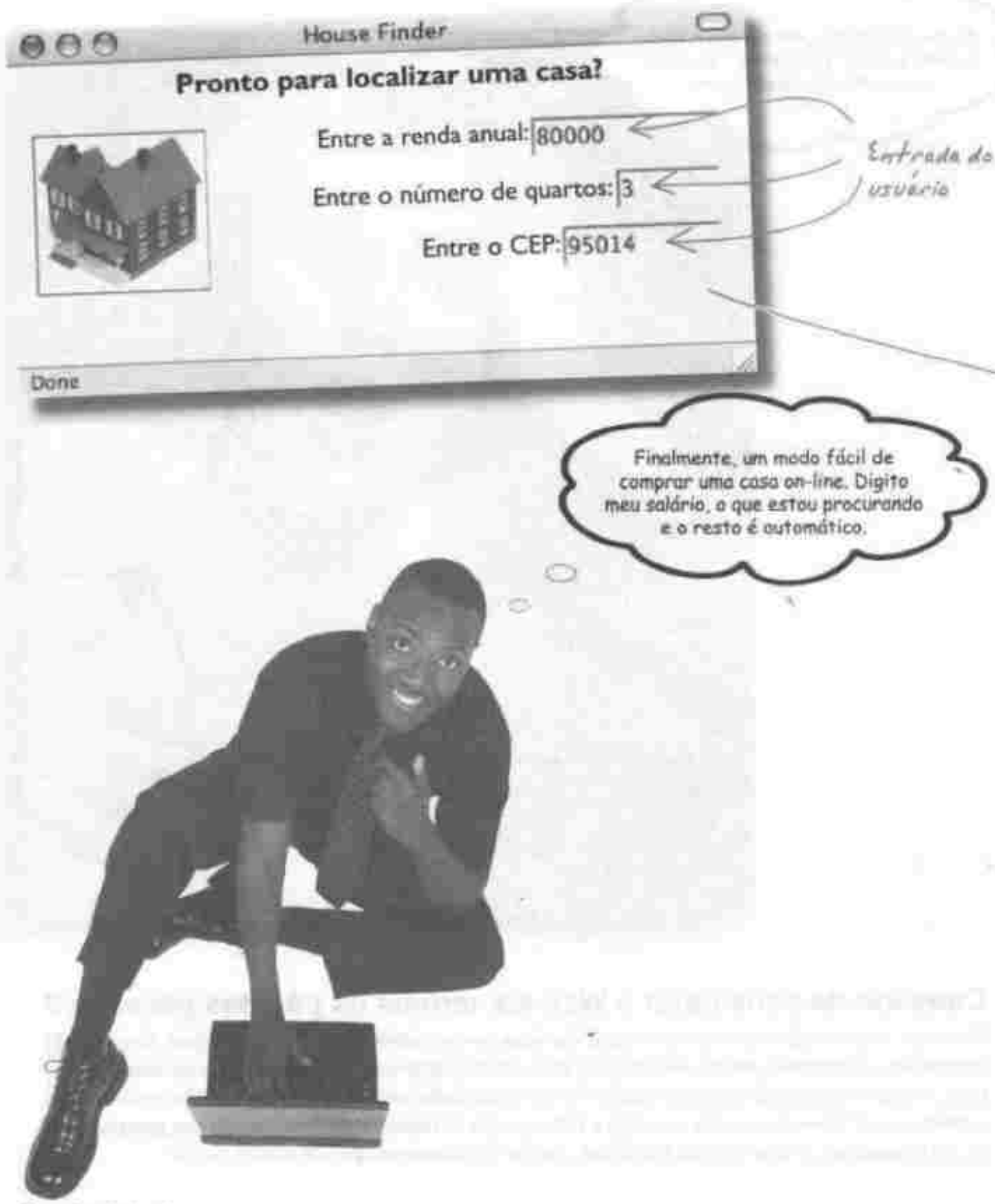
Como reagir ao mundo virtual

Cansado de considerar a Web em termos de páginas passivas?

Foi tempo, acabou. São chamadas de livros. E são boas para ler, aprender...muitas coisas boas. Mas não são **interativas**. E nem a Web sem um pouco de ajuda do JavaScript. Certamente, você pode enviar um formulário e talvez fazer um truque aqui e acolá com algum código HTML e CSS esperto, mas estará realmente apenas representando o *Weekend at Bernie's* ao apoiar uma página Web sem vida. A **interatividade** real requer **um pouco mais de inteligência** e **muito menos trabalho...**mas tem uma ***compensação muito maior***.

As pessoas (on-line) têm necessidades

Tudo bem, sabemos que a Web é virtual, mas as pessoas na Web são reais, com necessidades reais. Necessidades como pesquisar uma receita arrasadora de pão de carne, fazer o download de sua canção favorita do Meatloaf ou algo ainda tão grande quanto comprar uma casa nova. Felizmente, a Web diferencia ao priorizar suas necessidades!

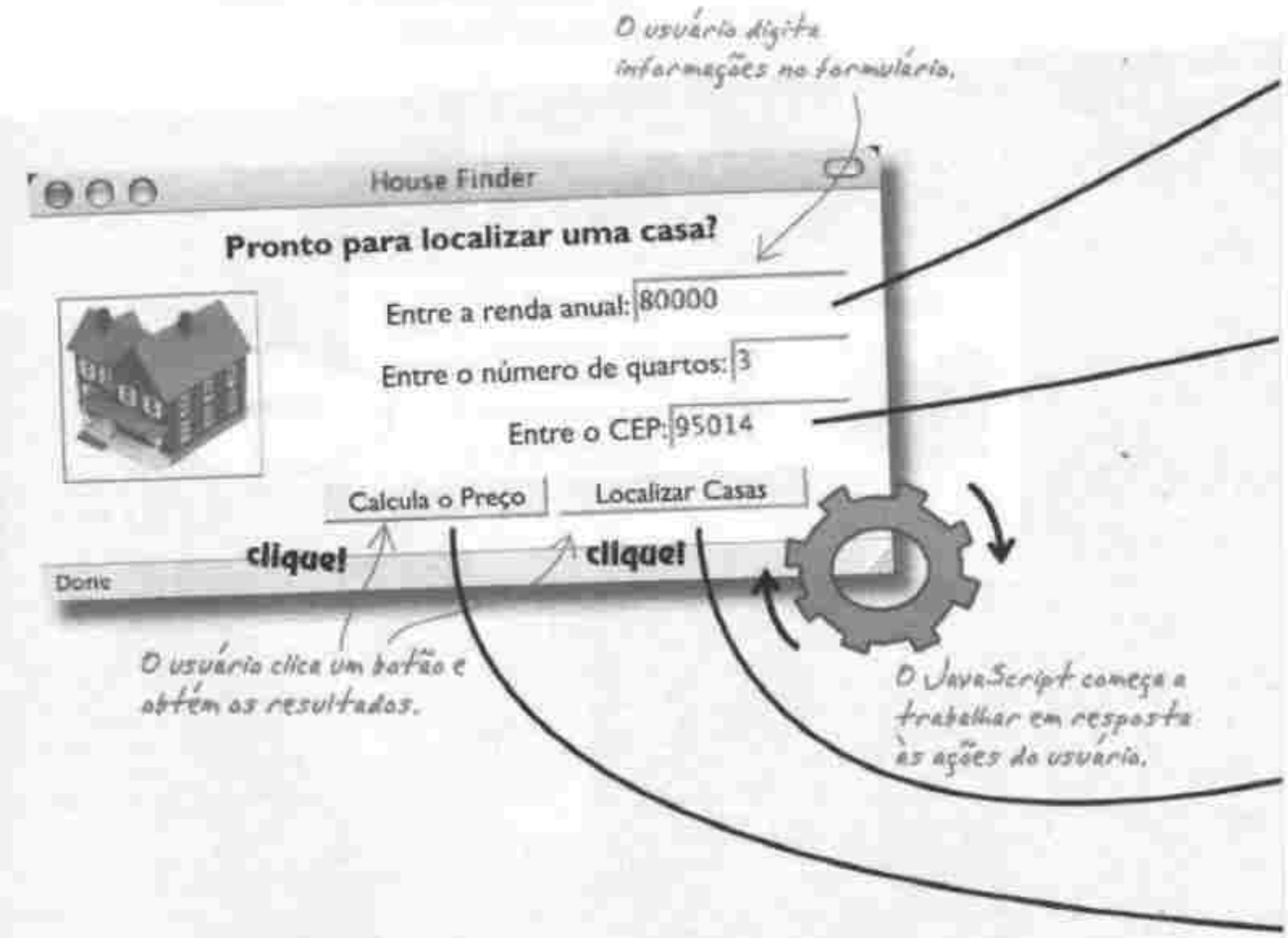


Como falar com uma parede... nada acontece

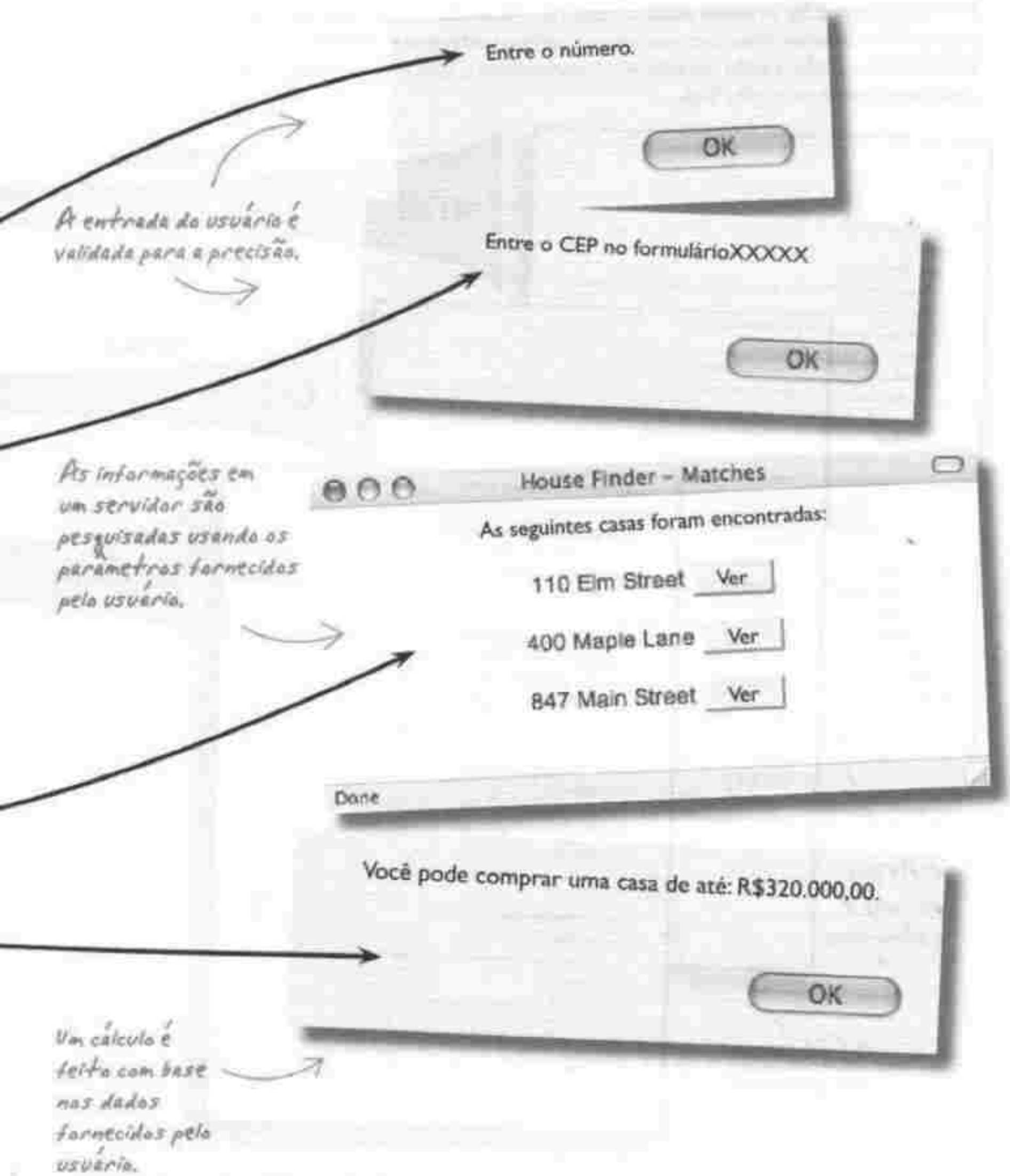
A Web nem sempre é tão responsiva quanto poderia ser. Na verdade, algumas vezes pode parecer totalmente fria e insensível, desinteressada do mundo externo e indiferente às necessidades de seus muitos usuários. Você poderia esperar que quando fornecesse dados assim iria gerar algum tipo de resposta...mas nada aconteceu. Não é pessoal; a Web estática não conhece nada melhor.

Mas o JavaScript dá uma resposta

O JavaScript move a chave que transforma uma página Web em uma experiência interativa. Ele capacita coisas que podem ouvir as suas necessidades, processar sua entrada e atender aos seus desejos mais profundos. Tudo bem, talvez seja um esforço, mas o JavaScript pode transformar uma página Web em um aplicativo interativo, em oposição a uma página estática e sem vida, e isso é bom!

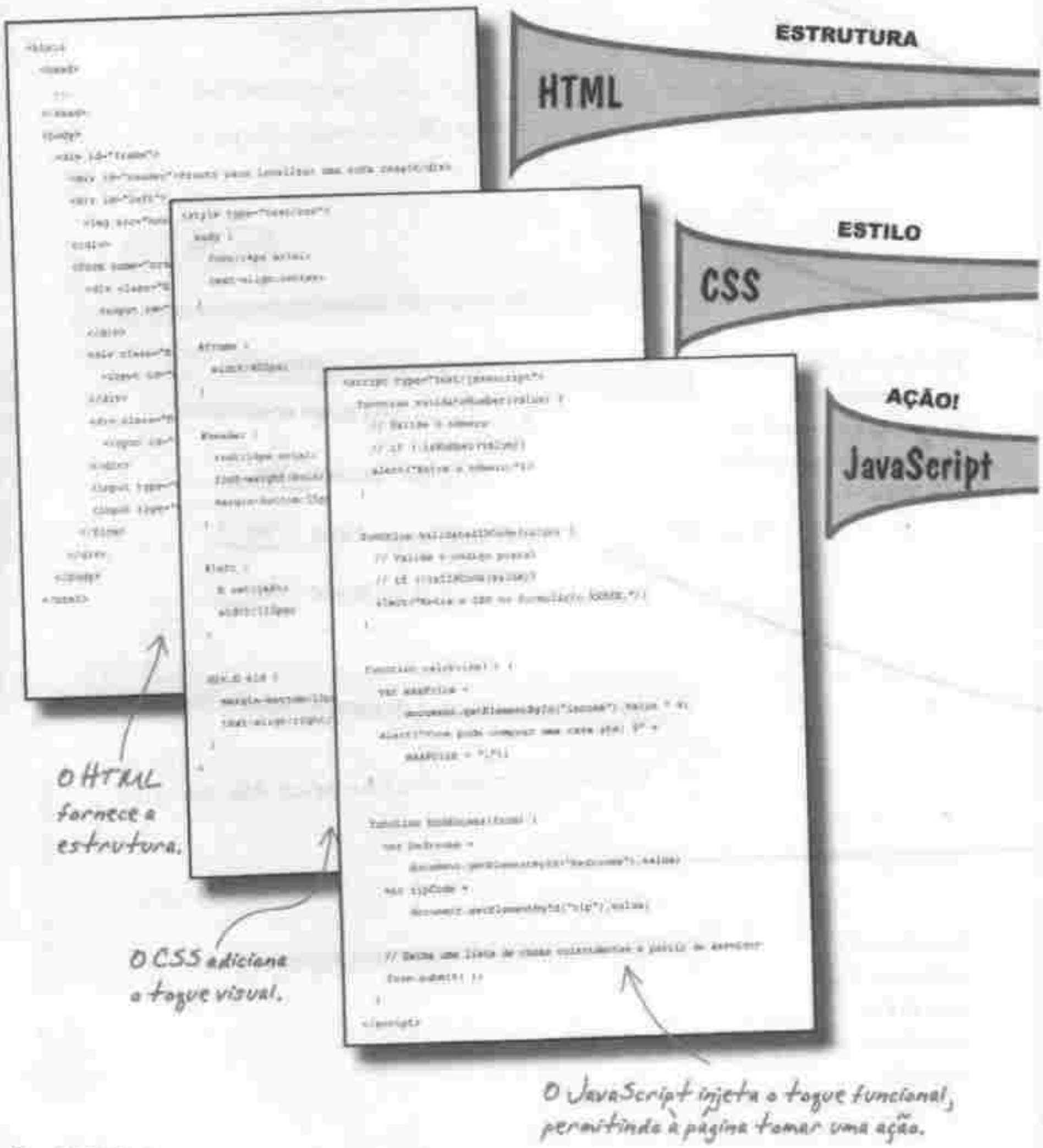


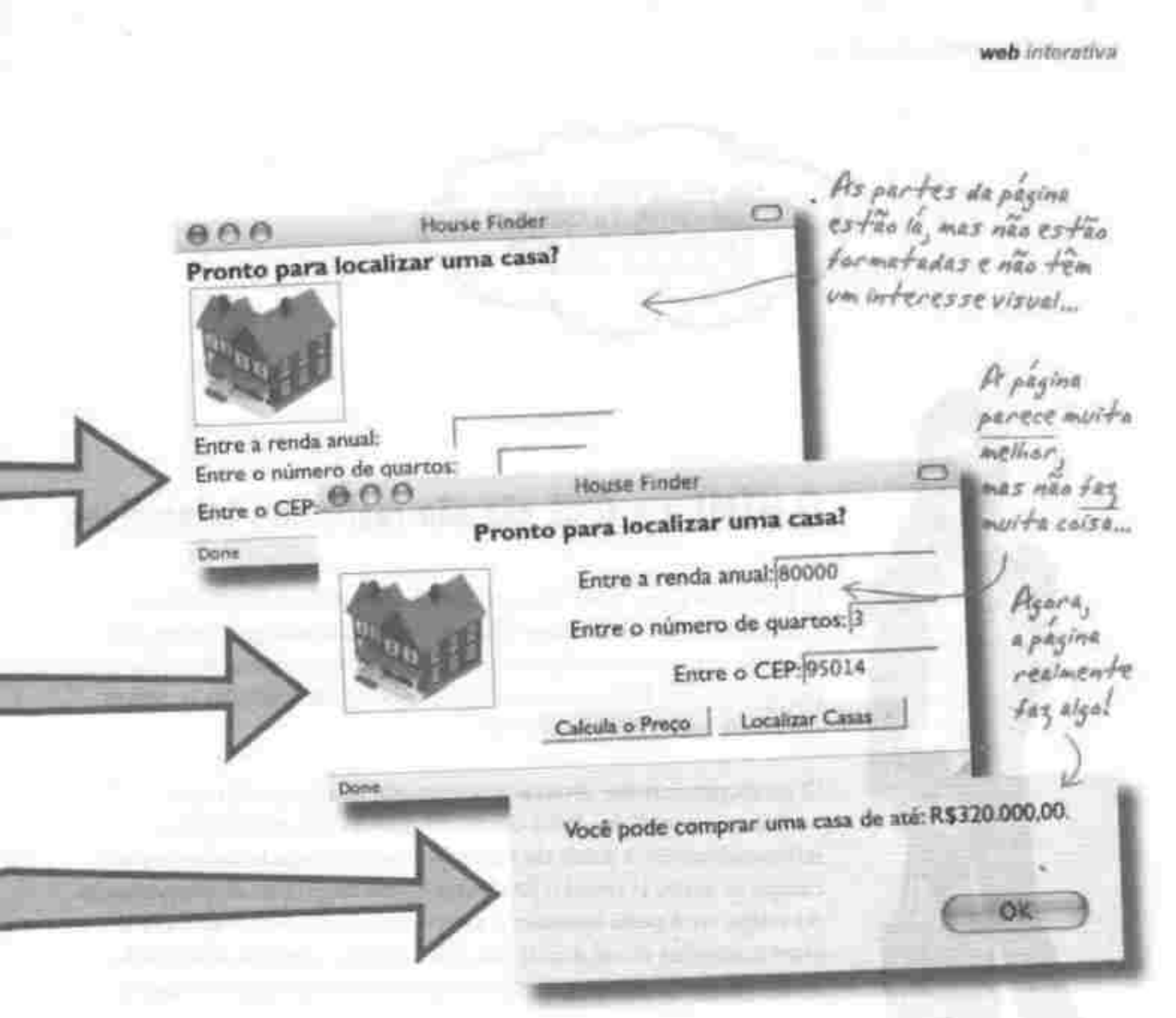
O JavaScript dá vida a uma página Web permitindo que responda à entrada do usuário.

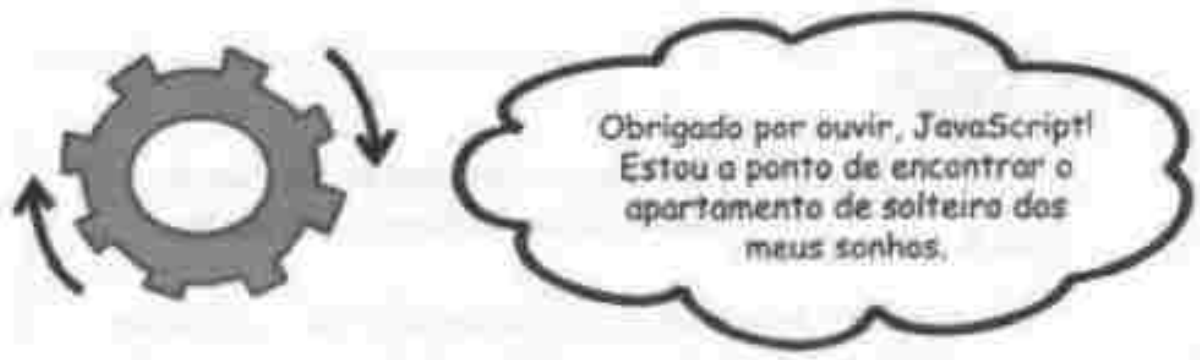

Entre o número.
OK
A entrada do usuário é validada para a precisão.
Entre o CEP no formulárioXXXXX
OK
As informações em um servidor são pesquisadas usando os parâmetros fornecidos pelo usuário.
House Finder - Matches
As seguintes casas foram encontradas:
110 Elm Street Ver
400 Maple Lane Ver
847 Main Street Ver
Done
Você pode comprar uma casa de até: R$320.000,00.
OK
Um cálculo é feito com base nos dados fornecidos pelo usuário.

Luzes, câmera, inter ação!

O JavaScript coloca o HTML e o CSS como uma das três peças da construção da página Web moderna. O HTML fornece a estrutura, o CSS adiciona o estilo e o JavaScript inicia e faz as coisas acontecerem. Para encontrar o caminho para uma página Web interativa, você tem que seguir a trilha da **estrutura** (HTML) até o **estilo** (CSS) e então, a **ação** (JavaScript). Parecido com o CSS, o código JavaScript geralmente reside na página Web.







O JavaScript entra em ação quando o usuário pede que a página execute uma tarefa.

Não se pode fazer o mesmo com o HTML e o CSS? A Web era muito legal **antes** do JavaScript, você sabe.

O HTML e o CSS não são realmente interativos

O problema é que o HTML e o CSS não são realmente interativos. Certamente existem truques CSS que você pode usar para manipular os estilos em situações muito específicas como, por exemplo, colocar o mouse sobre os links, mas suas opções serão bem limitadas se estiver usando apenas o HTML e o CSS.

O JavaScript permite detectar praticamente qualquer coisa que ocorre em uma página Web, como um usuário clicando botões, redimensionando a janela do navegador ou fornecendo dados em um campo de texto. E como o JavaScript é uma linguagem de programação do script, você pode aprender a escrever o código para responder a essas interações do usuário como, por exemplo, executar um cálculo, trocar dinamicamente as imagens na página ou até validar os dados.

Relaxe

Não se preocupe com os detalhes do JavaScript, pelo menos não ainda.

Embora o JavaScript seja capaz de fazer todos os tipos de coisas, sabemos que você está no início de sua jornada. Fique certo de que os eventos, funções e muitas outras partes do quebra-cabeça JavaScript irão juntar-se com o tempo. Além disso, provavelmente você estará mais à frente no jogo do que pensa.

HTML + CSS + JavaScript = Interatividade REAL

Aponte seu Lápis

Você já sabe mais do que pensa. Veja o código para a página Web House Finder e escreva o que acha que cada parte circulada do código JavaScript está fazendo. Não tem problemas se adivinhar.

```
<html>
  <head>
    <title>House Finder</title>
    <script type="text/javascript">
      function validateNumber(value) {
        // Valide o número
        // if (!isNumber(value))
        alert("Entre o número.");
      }

      function validateZIPCode(value) {
        // Valide o código postal
        // if (!isZIPCode(value))
        alert("Entre o CEP no formulário XXXXX.");
      }

      function calcPrice() {
        var maxPrice = document.getElementById("income").value * 4;
        alert("You can afford a house that costs up to $" + maxPrice +
".");
      }

      function findHouses(form) {
        var bedrooms = document.getElementById("bedrooms").value;
        var zipCode = document.getElementById("zip").value;

        // Exiba a lista de casas coincidentes a partir do servidor
        form.submit();
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div id="header">Ready to find a new house?</div>
      <div id="left">
        <img src="house.png" alt="House" />
      </div>
      <form name="orderform" action="..." method="POST">
        <div class="field">Entre a renda anual:
          <input id="income" type="text" size="12"
          onblur="validateNumber(this.value)" /></div>
        <div class="field">Entre o número de quartos:
          <input id="bedrooms" type="text" size="5"
          onblur="validateNumber(this.value)"/></div>
        <div class="field">Entre o CEP:
          <input id="zip" type="text" size="10"
          onblur="validateZIPCode(this.value)"/></div>
        <input type="button" value="Calcula o preço"
          onclick="calcPrice();" />
        <input type="button" value="Localizar casas"
          onclick="findHouses(this.form);" />
      </form>
    </div>
  </body>
</html>
```

Aponte seu Lápis
Solução

Você já sabe mais do que pensa. Veja o código para a página Web House Finder e escreva o que acha que cada parte circulada do código JavaScript está fazendo. Não tem problemas se adivinhar.

```
<html>
  <head>
    <title>House Finder</title>
    <script type="text/javascript">
      function validateNumber(value) {
        // Valide o número
        // if (!isNumber(value))
        alert("Entre o número.");
      }

      function validateZIPCode(value) {
        // Valide o código postal
        // if (!isZIPCode(value))
        alert("Entre o CEP no formulário XXXXX.");
      }

      function calcPrice() {
        var maxPrice = document.getElementById("income").value * 4;
        alert("Você pode comprar uma casa até $" + maxPrice + ".");
      }

      function findHouses(form) {
        var bedrooms = document.getElementById("bedrooms").value;
        var zipCode = document.getElementById("zip").value;

        // Exiba a lista de casas coincidentes a partir do servidor
        form.submit();
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div id="header">Pronto para localizar uma nova casa?</div>
      <div id="left">
        <img src="house.png" alt="House" />
      </div>
      <form name="orderform" action="_" method="POST">
        <div class="fl eld">Entre a renda anual:
          <input id="income" type="text" size="12"
          onblur="validateNumber(this.value)"/></div>
        <div class="fl eld">Entre o número de quartos:
          <input id="bedrooms" type="text" size="6"
          onblur="validateNumber(this.value)"/></div>
        <div class="fl eld">Entre o CEP:
          <input id="zip" type="text" size="10"
          onblur="validateZIPCode(this.value)"/></div>
        <input type="button" value="Calcula o preço"
          onclick="calcPrice();" />
        <input type="button" value="Localizar casas"
          onclick="fl ndHouses(this.form);" />
      </form>
    </div>
  </body>
</html>
```

Pede ao usuário que forneça um código postal no formato com cinco dígitos, XXXXX.

Calcula o preço máximo de casa como quatro vezes o salário do usuário.

Valida o campo income para assegurar que um número foi fornecido.

O valor do campo de entrada do código postal.

Calcula o preço máximo de casa quando o usuário clica o botão Calcular Price.

Use a tag <script> para informar ao navegador que você está escrevendo JavaScript

No momento, iremos colocar o JavaScript diretamente em uma página Web HTML, exatamente com vimos na última página. A primeira coisa que você tem que fazer é deixar que o navegador Web saiba que iremos fornecer-lhe o JavaScript, ao invés do HTML... e é onde a tag `<script>` entra.

Você pode adicionar uma tag `<script>` em qualquer lugar em seu HTML, mas geralmente é melhor colocá-la na `<head>` de nossa página Web, assim:

```
<html>
  <head>
    <title>House Finder</title>

    <script type="text/javascript">

      function validateNumber(value) {
        // Valide o número
        // if (!isNumber(value))
        alert("Entre o número.");
      }

    </script>
  </head>

  <body>
    <!-- All the rest of your HTML -->
  </body>
</html>
```

Esta tag do script informa que qualquer coisa depois dela é uma linguagem de script...

...e neste caso, o tipo de linguagem de script é JavaScript.

Você pode colocar a tag do script em uma página HTML normal, geralmente na seção head.

Tudo entre as tags do script de abertura e de fechamento é JavaScript... o navegador sabe tratar isso como uma linguagem de script, e não HTML.

A tag do script de fechamento informa ao navegador que o HTML normal está continuando agora.

Não existem Perguntas Idiotas

P: Então qualquer coisa que eu coloco dentro da tag <script> é JavaScript?

R: Não necessariamente... a tag `<script>` informa ao navegador que uma linguagem de script está chegando, mas não tem que ser JavaScript. A parte do tipo, `type="text/javascript"`, é o que permite ao navegador saber que você está para lhe fornecer o JavaScript especificamente.

P: Então existem outras linguagens de script que posso usar?

R: Absolutamente. A Microsoft tem algumas variedades como VBScript (uma versão de script do Visual Basic) e seu tipo Ajax, chamado ASP.NET AJAX. Falaremos mais sobre o Ajax no Capítulo 12 também. E há várias outras linguagens de script que você pode usar. Mas para nossas finalidades, sempre usaremos `text/javascript` neste livro.

P: Meus elementos <script> têm que estar na parte <head> de minha página HTML?

R: É uma boa pergunta. Você pode colocar os elementos `<script>` em qualquer lugar em sua página Web... mas é geralmente considerada uma prática ruim colocá-los em qualquer lugar, exceto em `<head>` de sua página Web. É como colocar o CSS no meio de uma página Web... geralmente é melhor separar o JavaScript, e a `<head>` de sua página é um lugar perfeito para fazer isso.

Seu navegador Web pode lidar com o HTML, CSS E JavaScript

Você já sabe que um navegador Web sabe como obter seu HTML e exibi-lo. E usou o CSS para informar ao navegador como mostrar as diferentes partes de seu HTML. Considere o JavaScript como apenas outro meio de se comunicar com o navegador... mas ao invés de dizer ao navegador como exibir algo (como no HTML ou no CSS), você dará ao navegador alguns comandos para seguir.

Não existem Perguntas Idiotas

P: Como os navegadores Web executam o código JavaScript?

R: Os navegadores Web têm uma parte especial do software dentro deles chamada interpretador JavaScript e seu serviço é executar o código JavaScript que aparece em uma página. É por isso que você pode ouvir o JavaScript sendo descrito como uma linguagem *interpretada*, em oposição a uma linguagem *compilada*. As linguagens compiladas como C++ ou C#, por exemplo, devem ser convertidas por uma ferramenta chamada compilador em um arquivo de programa executável. Não é necessário compilar os programas JavaScript porque o código JavaScript é interpretado diretamente pelo navegador.

P: Como informo a uma página Web para começar a executar o código JavaScript?

R: A maior parte do código JavaScript é executada quando algo ocorre na página como, por exemplo, a página sendo carregada ou o usuário clicando um botão. Um mecanismo JavaScript conhecido como "evento" permite a você inicializar uma parte do código JavaScript quando algo de interesse ocorre na página.

P: Considerando os problemas de segurança da Web, o JavaScript é seguro?

R: Sim, na maior parte. O JavaScript é designado totalmente para impedir que o código malicioso cause problemas. Por exemplo, o JavaScript não permite a você ler ou escrever arquivos no disco rígido do usuário. Esta limitação acaba com o potencial de muitos vírus e código malicioso parecido. Naturalmente, isso não significa que você não pode escrever um código JavaScript com erros que torne as páginas Web difíceis de usar. Significa apenas que é improvável que você coloque os usuários em sérios riscos com o JavaScript. E só para lembrar, os erros do navegador e hackers espertos descobriram modos de quebrar da segurança JavaScript no passado, e certamente ele não é imbatível.

P: E a tag `<script>` no código House Finder... é HTML ou JavaScript?

R: A própria tag `<script>` é HTML e sua finalidade é fornecer um meio de misturar o código do script com o código HTML. O código que aparece **dentro** da tag `<script>` é o código JavaScript. Como a tag `<script>` é para suportar diversas linguagens de script, você indica que o código é JavaScript usando seu atributo `type`.

P: Vi páginas Web que têm interatividade como, por exemplo, os formulários que verificam para assegurar que a data seja fornecida corretamente e parecem fazer isso sem o JavaScript. É possível?

R: Sim. É possível ter interatividade nas páginas Web sem o JavaScript, mas em muitos casos é ineficiente e desajeitado. Por exemplo, a validação dos dados nos formulários pode ser lidada no servidor Web quando você envia o formulário. Porém, isso significa que terá que enviar o formulário inteiro e, então, aguardar que o serviço faça a validação e retorne os resultados como uma página nova. Você pode também validar o formulário com lápis e papel! A interatividade JavaScript ocorre inteiramente no navegador sem carregar uma nova página, eliminando a transmissão desnecessária de dados para um servidor. Não só isso, mas grande parte do que o JavaScript tem a oferecer em termos de interatividade não pode ser feita de nenhuma outra maneira sem complementos do navegador de terceiros.

Identifique cada parte do código como sendo parte da linguagem JavaScript padrão ou uma parte personalizada do código criado por um programador para a página Web House Finder.

`alert`	JavaScript / Personalizado	`onblur`	JavaScript / Personalizado
`calcPrice`	JavaScript / Personalizado	`onclick`	JavaScript / Personalizado
`zipCode`	JavaScript / Personalizado	`findHouses`	JavaScript / Personalizado
`var`	JavaScript / Personalizado	`value`	JavaScript / Personalizado

Exercício Resolvido

Identifique cada parte do código como sendo parte da linguagem JavaScript padrão ou uma parte personalizada do código criado por um programador para a página Web House Finder.

```
<head>
  <title>House Finder</title>
  <script type="text/javascript">
    function validateNumber(value) {
      // Valide o número
      // if (!isNumber(value))
      alert("Entre o número.");
    }

    function validateZIPCode(value) {
      // Valide o código postal
      // if (!isZIPCode(value))
      alert("Entre o CEP no formulário XXXXX.");
    }

    function calcPrice() {
      var maxPrice = document.getElementById("income").
value * 4;
      alert("Você pode comprar uma casa até $" +
maxPrice
+ ".");
    }

    function findHouses(form) {
      var bedrooms = document.getElementById("bedrooms").
value;
      var zipCode = document.getElementById("zip").value;

      // Exiba uma lista de casas coincidentes a partir do
servidor
      form.submit();
    }
  </script>
</head>

<body>
  <div id="frame">
    <div id="header">Pronto para localizar uma nova casa?</div>
    <div id="left">
      <img src="house.png" alt="House" />
    </div>
    <form name="orderform" action="_" method="POST">
      <div class="fl eld">Entre a renda anual:
        <input id="income" type="text" size="12"
        onblur="validateNumber(this.value)"/></div>
      <div class="fl eld">Entre o número de quartos:
        <input id="bedrooms" type="text" size="6"
        onblur="validateNumber(this.value)"/></div>
      <div class="fl eld">Entre o CEP:
        <input id="zip" type="text" size="10"
        onblur="validateZIPCode(this.value)"/></div>
      <input type="button" value="Calcula o preço"
        onclick="calcPrice();" />
      <input type="button" value="Localizar casas"
        onclick="findHouses(this.form);" />
    </form>
  </div>
</body>
</html>
```

A caixa pop-up que indica um número inválido.

alert — JavaScript / Personalizado

Uma parte personalizada do código que calcula o preço da casa.

calcPrice — JavaScript / Personalizado

Separa o local de armazenamento para uma parte dos dados.

var — JavaScript / Personalizado

O código personalizado que encontra as casas coincidentes.

findHouses — JavaScript / Personalizado

zipCode — JavaScript / Personalizado

Um local de armazenamento usado para manter o código postal fornecido pelo usuário.

onblur — JavaScript / Personalizado

Indica que o usuário foi para o próximo campo de entrada.

value — JavaScript / Personalizado

O valor atual do campo de entrada do código postal.

onclick — JavaScript / Personalizado

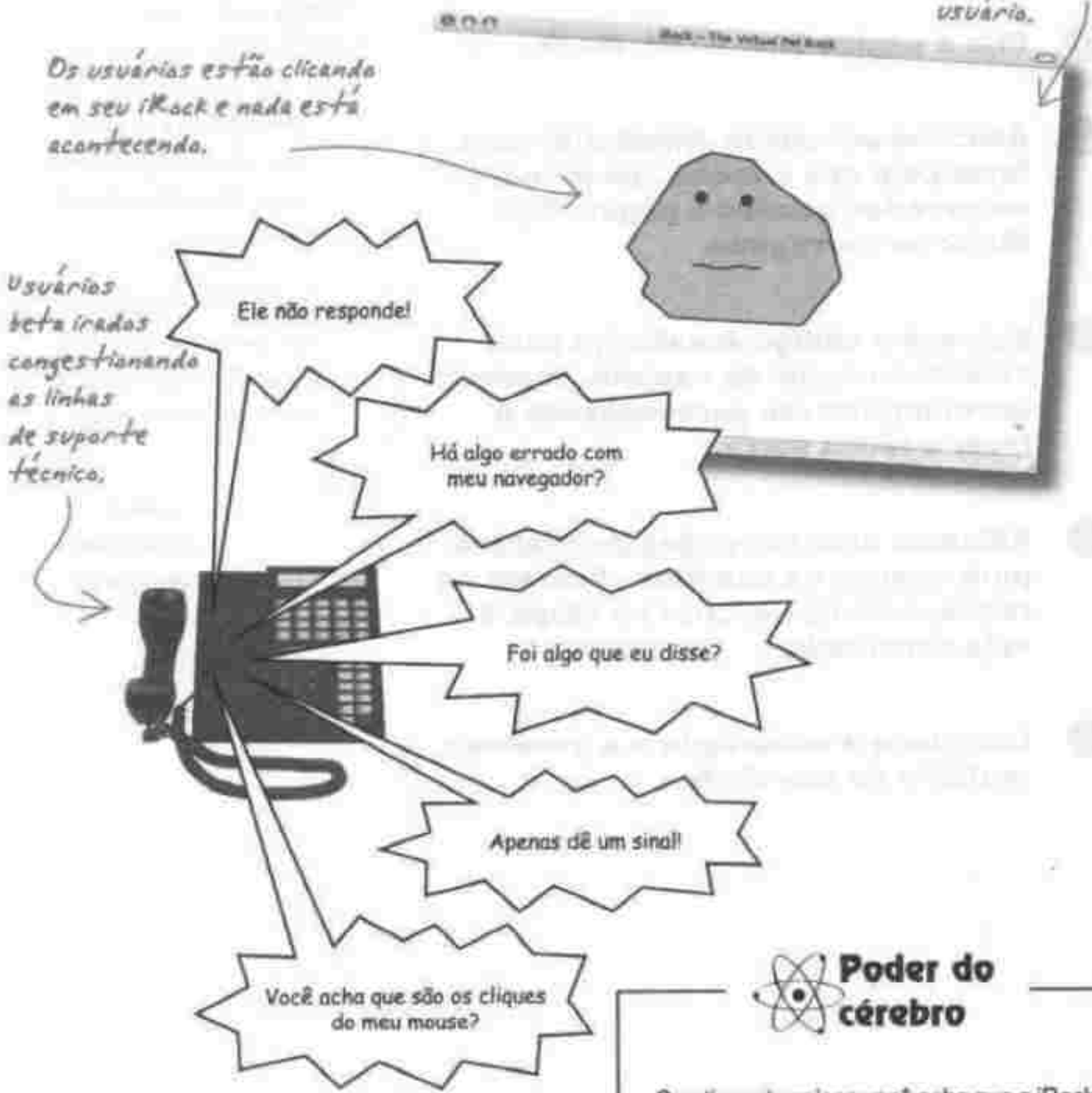
Indica que o botão Shop foi clicado.

O melhor amigo virtual do homem... precisa de SUA ajuda

Acabando com a tarefa bem-sucedida de escrever páginas HTML e CSS, você foi chamado ao escritório de seu chefe para ver sua última invenção on-line: o iRock. O bichinho de estimação virtual está causando agitação em todas as conferências de brinquedos, mas os usuários beta estão realmente infelizes com o bichinho on-line.

Aparentemente, os usuários estão clicando na rocha e esperando que algo legal aconteça... mas seu chefe nunca pensou nisso. Agora, cabe a você **tornar o iRock interativo** e conseguir a glória... ou descer ao inferno com o iRock.

Eis o iRock até o momento... apenas HTML e CSS. E isso significa nenhuma interação com o usuário.



Poder do cérebro

Que tipos de coisas você acha que o iRock deve ser capaz de fazer para interagir com seus usuários?

Como tornar o iRock interativo

Não só cabe a você tornar o iRock interativo, como também terá que aprender um pouco do JavaScript no percurso. Tudo bem, entretanto, terá sua rocha de estimação dizendo olá em pouco tempo.

Eis o que fará no resto do capítulo:

1. **Crie a página Web HTML iRock.**

   Você já sabe como fazer isto.

2. **Adicione um alerta JavaScript para fazer com que a rocha cumprimente os usuários quando a página Web iRock for carregada.**

   Um alerta é o modo do JavaScript exibir uma simples caixa de mensagem.

3. **Escreva o código JavaScript para solicitar o nome do usuário, imprimir um cumprimento personalizado e fazer a rocha sorrir.**

   Você está conectando algo que o usuário faz, como clicar na rocha de estimação virtual...

4. **Adicione uma sub-rotina de eventos para quando os usuários clicarem na rocha, o código escrito na etapa 3 seja executado.**

   ...com a atividade designada.

5. **Conquiste a admiração e a generosa gratidão de seu chefe.**

Crie a página Web iRock

Você não poderia encontrar uma página HTML mais simples do que o iRock. Continue e digite este HTML em seu editor favorito e grave-o como iRock.html. Você pode fazer download das imagens da rocha de estimação no site Web Head First Labs, em www.altabooks.com.br.

A página HTMAL da rocha de estimação é tão chata quanto a própria rocha... não é de se admirar que seu chefe precise de sua ajuda.

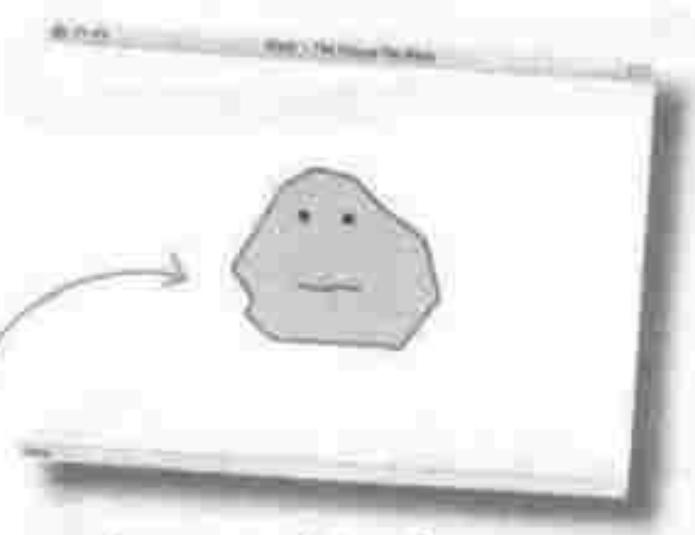
```
<html>
  <head>
    <title>iRock - A rocha virtual de estimação </title>
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      <img id="rockImg" src="rock.png" alt="iRock" />
    </div>
  </body>
</html>
```

Faça download do rock.png a partir dos exemplos on-line no site Web www.altabooks.com.br.

irock.html

'Test drive'

Antes de avançar mais, grave e teste sua página Web iRock em seu navegador Web. Certifique-se de que a sua esteja parecida com a nossa, pois você começará a adicionar alguma interatividade, no estilo JavaScript.



Em poucas páginas, você terá esta rocha chata sorrindo e conversando com seus usuários.

Não existem Perguntas Idiotas

P: Este CSS está na tag ?

R: Certamente. Boa pergunta.

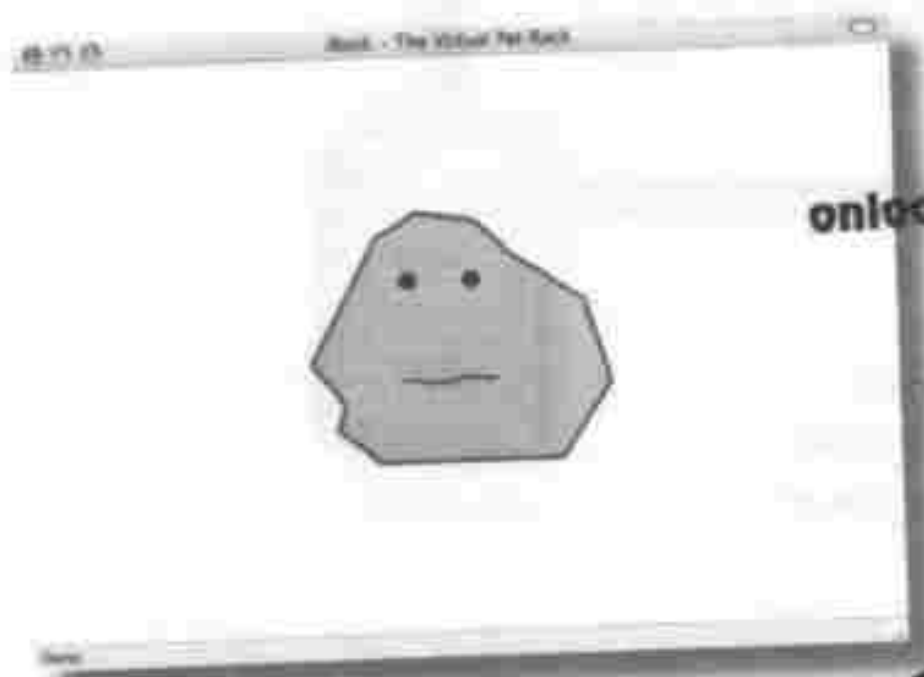
P: Pensei que fosse realmente uma má idéia colocar o CSS diretamente em uma página HTML. O que acontece?

R: Você leu o Use a Cabeça HTML com CSS e XHML, não leu? Sim, você está certo. Geralmente é melhor colocar seu CSS em uma tag `<style>` na `<head>` de sua página ou em uma folha de estilo externa. Mas seu chefe não é codificador e, além disso, torna este primeiro exemplo muito mais simples. Mas se quiser prosseguir e escrever sua própria folha de estilo externa para o iRock, achamos que seria muito legal!

Eventos JavaScript: como dar ao iRock uma voz

Para usar o JavaScript para cumprimentar o usuário quando a página é carregada pela primeira vez, teremos que resolver dois problemas principais relacionados ao JavaScript: saber quando a página termina de carregar e saber como exibir um cumprimento para que o usuário possa vê-lo.

O primeiro problema envolve responder a um evento (o evento de carregamento da página), ao passo que o segundo envolve usar um recurso JavaScript predefinido, a caixa de "alerta". Os *eventos* são notificações JavaScript que permitem saber quando algo de interesse ocorreu como, por exemplo, um carregamento da página (`onload`) ou um botão sendo clicado (`onclick`). Você pode responder aos eventos com seu próprio código JavaScript personalizado.



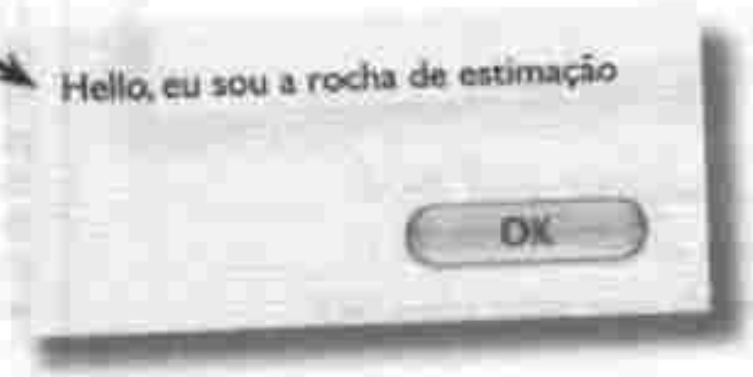
O evento onload é inicializado quando a página iRock termina de carregar no navegador.

O código para o evento onload é definido usando o atributo onload da tag <body> na página Web iRock.

Os eventos são notificações que você pode responder com o código JavaScript.

```
<body onload="alert('Hello, eu sou a rocha de estimação.');">
```

A função alert( ) informa ao navegador para exibir uma caixa de alerta que cumprimenta o usuário.



Como alertar o usuário com uma função

Um **alerta** JavaScript é uma janela pop-up, ou caixa, que você pode usar para exibir informações para o usuário. Exibir uma caixa de alerta envolve escrever o código para chamar a função `alert( )` JavaScript e transmitir-lhe o texto que desejar exibir. As *funções* são partes reutilizáveis do código JavaScript que executam tarefas comuns, como por exemplo, exibir informações em uma janela pop-up.

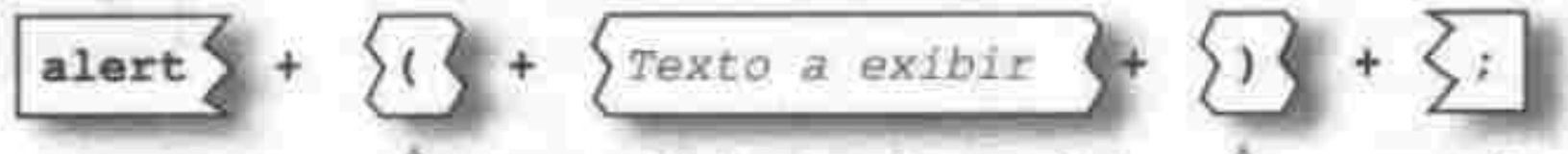
alert( )

Quando você vê parênteses imediatamente ao lado de um nome no código JavaScript, geralmente é uma função.

Um foco no alerta

alert é o nome da função predefinida que exibe uma caixa de alerta.

Este é o texto a ser exibido na caixa de alerta – coloque-o entre aspas.



Toda função JavaScript usa parênteses para conter as informações sendo transmitidas para a função – neste caso, o texto a ser exibido.

Um ponto-e-vírgula marca o final de uma linha do código JavaScript, um tipo de ponto no final de uma sentença.

Quando você reúne tudo, obtém uma linha completa de código JavaScript que chama uma função para exibir o texto de cumprimento em uma caixa de alerta:

```
alert('Hello, eu sou a rocha de estimação.');
```

As funções são partes reutilizáveis do código que executam tarefas comuns.

O texto a ser exibido é colocado entre apóstrofos ou aspas.

Relaxe

Não estresse os eventos.

Se todo esse negócio de eventos parecer uma tortura, não se preocupe porque os eventos continuarão a se desdobrar (e farão mais sentido) conforme avançar no livro.

Adicione o cumprimento iRock

Portanto, para cumprimentar os usuários quando eles carregam as imagens iRock, você precisa adicionar uma sub-rotina de evento `onload` e um cumprimento usando a função `alert( )` do JavaScript. Adicione esta linha de JavaScript à sua página `irock.html`:

```
<html>
  <head>
    <title>iRock - A rocha virtual de estimação </title>
  </head>

  <body onload="alert('Hello, eu sou a rocha de estimação.');">
    <div style="margin-top:150px; text-align:center">
      <img id="rockImg" src="rock.png" alt="iRock" />
    </div>
  </body>
</html>
```

Mesmo que o evento onload aplique-se à página inteira, você pode defini-lo como um atributo da tag <body> porque o corpo de uma página é a parte visível em um navegador.

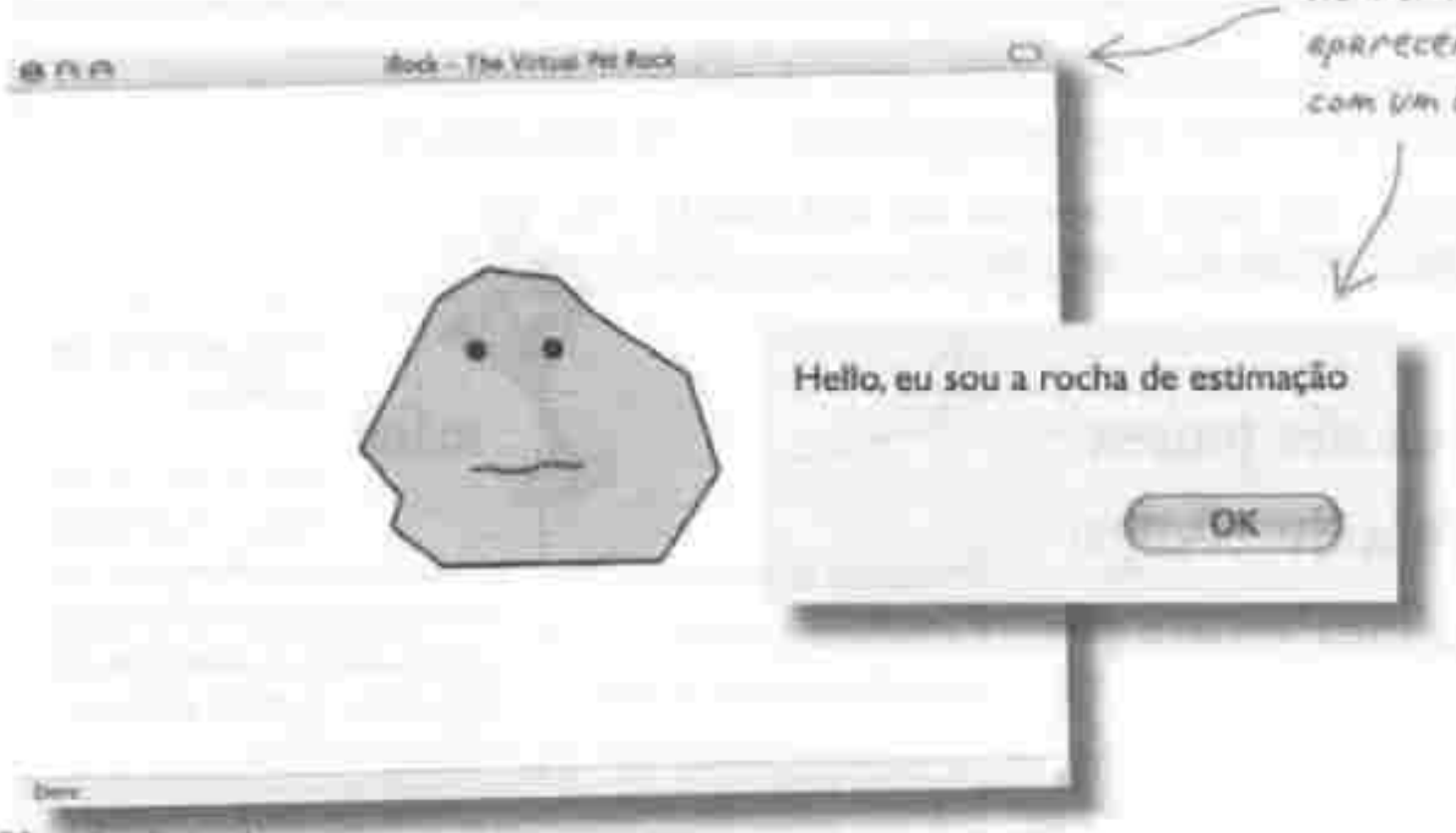
irock.html

Lembre-se, seu JavaScript vai diretamente em sua página Web. O navegador Web sabe como lidar com o JavaScript, exatamente como faz com o HTML e o CSS.

Faça um 'test drive' de sua rocha interativa

A página iRock agora é um pouco mais interativa graças a uma caixa de alerta cumprimentando exibida em resposta ao evento onload. Carregue o irock.html em seu navegador Web e veja o que acontece:

Assim que a página é carregada, uma caixa de alerta deverá aparecer em sua tela com um cumprimento.



Não existem Perguntas Idiotas

P: De onde vêm os eventos?

R: Embora os eventos sejam iniciados por um usuário, eles vêm basicamente do navegador. Por exemplo: uma "tecla pressionada" é um evento inicializado pelo usuário, mas o navegador deve integrar informações sobre o evento (tipo qual tecla foi pressionada) e, então, transmiti-las para uma função que foi designada a responder ao evento.

P: O que acontece com os eventos que não têm um código ligado a eles?

R: Se uma árvore cai e não há ninguém por perto para ouvir, ela fez barulho? O mesmo ocorre com os eventos. Se você não responder a um evento, o navegador prosseguirá e ninguém ficará sabendo. Em outras palavras, responder ou não a onload não tem significado no carregamento real da página.

P: Você não disse que o código JavaScript pertencia às tags <script>?

R: Geralmente, sim. Mas você também pode colocá-lo diretamente em uma sub-rotina de eventos, como fizemos com o evento onload. E quando precisar executar apenas uma linha do JavaScript, como para o iRock, essa será geralmente uma abordagem mais simples.

P: Existem outras funções predefinidas como a função alert( )?

R: Sim, muitas. alert( ) é apenas a ponta do iceberg quanto ao código JavaScript reutilizável e predefinido. Falaremos sobre as funções padrões quando virmos os recursos do JavaScript. No final do livro, você até estará criando suas próprias funções personalizadas.

P: Por que o código onload do iRock mistura aspas e apóstrofos?

R: O HTML e o JavaScript requerem que você feche uma seqüência de texto antes de iniciar outra... a menos que use um delimitador diferente (aspa ou apóstrofo). Portanto, nos casos onde o código JavaScript aparece em um atributo HTML (texto dentro de texto), você tem que misturar as aspas e os apóstrofos para solucionar esse problema. Não importa quais você usa para o atributo ou para o texto JavaScript, mas seja qual for o escolhido – terá que ser consistente. Talvez um exemplo de aspas e apóstrofos na linguagem esclareça as coisas... segundo o iRock. "O usuário clicou e disse: 'Olá, alguém aí'".

Pontos de Bala

- Os eventos são usados para responder aos acontecimentos da página Web com o código JavaScript.
- O evento `onload` é inicializado quando uma página termina da carregar.
- Você responde ao evento `onload` definindo o atributo `onload` da tag `<body>`.
- As funções permitem colocar o código JavaScript em módulos reutilizáveis.
- Algumas funções requerem que você transmita-lhes informações para completarem sua tarefa.
- A função `alert( )` é uma função JavaScript predefinida que exibe uma mensagem de texto em uma pequena janela pop-up.

QUEM FAZ O QUÊ?

Ligar cada parte do código JavaScript ao que ela faz.

onload	Exibe uma mensagem de texto em uma janela pop-up.
( )	Termina uma linha de código JavaScript.
alert	Indica que a página Web acabou de carregar.
;	Coloca as informações transmitidas em uma função.

QUEM FAZ O QUÊ?

Ligar cada parte do código JavaScript ao que ela faz.

`onload`	Exibe uma mensagem de texto em uma janela pop-up.
`( )`	Termina uma linha de código JavaScript.
`alert`	Indica que a página Web acabou de carregar.
`;`	Coloca as informações transmitidas em uma função.

Agora, tornemos o iRock realmente interativo

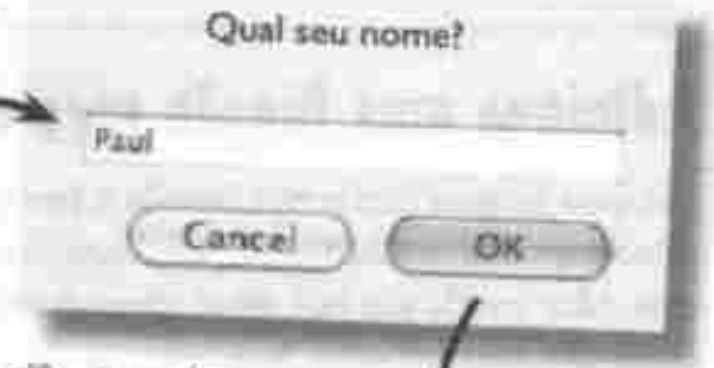
Você está fazendo um certo progresso em direção a um iRock mais interativo, mas ainda há mais a fazer antes da rocha de estimação virtual conquistar qualquer cliente... lembra de nossa lista de verificação?

1. ~~**Crie a página Web HTML iRock.**~~ ← Feito!
2. ~~**Adicione um alerta JavaScript para fazer com que a rocha cumprimente os usuários quando a página Web iRock for carregada.**~~ ← Também terminou com este!
3. **Escreva o código JavaScript para solicitar o nome do usuário, imprimir um cumprimento personalizado e fazer a rocha sorrir.**
4. **Adicione uma sub-rotina de eventos para quando os usuários clicarem na rocha, o código escrito na etapa 3 seja executado.**
5. **Conquiste a admiração e a generosa gratidão de seu chefe.**

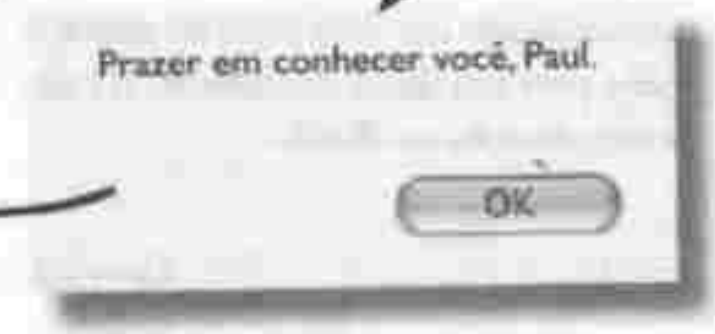
A interação é uma comunicação de mão DUPLA

Agora, nossa rocha diz oi, mas não deixa o usuário fazer muita coisa com ela. Realmente desejamos que a rocha *responda* aos usuários. Com a ajuda de um pequeno JavaScript, entretanto, o iRock poderá ser transformado em um bichinho de estimação simpático surpreendentemente sociável e totalmente amistoso, mudando sua expressão facial e cumprimentado o dono pelo nome...

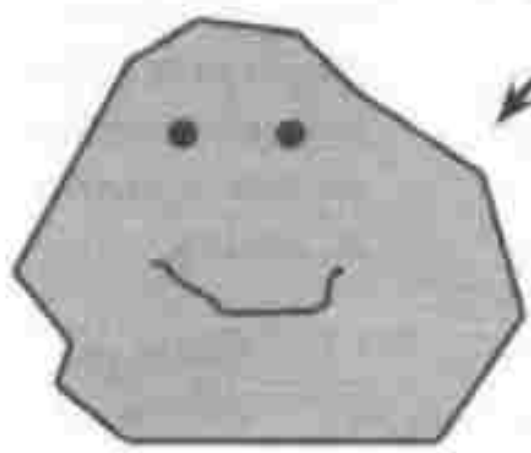
Quando o usuário clicar na rocha, ela perguntará seu nome.



Agora o iRock pode cumprimentar seu usuário pessoalmente.



O iRock também deve mostrar emoção sorrindo para o usuário.



Então, a satisfação do usuário aumenta (bem, essa é a idéia).

O JavaScript permite ao usuário interagir com o iRock, transformando as teclas pressionadas e os cliques do mouse (mais eventos) em brincadeiras entre um bichinho de estimação e seu dono. Uma amizade acionada pelo JavaScript nasceu!

Aponte seu Lápis

Adivinhe escrevendo o nome do evento JavaScript usado para responder a um clique do mouse.

.....

Aponte seu Lápis Solução

Adivinhe escrevendo o nome do evento JavaScript usado para responder a um clique do mouse.

onclick

O evento onclick é inicializado sempre que o usuário clica o mouse em um elemento na página – cada elemento da página Web pode ter seu próprio código de resposta onclick exclusivo.

Adicione uma função para obter o nome do usuário

Eis uma função JavaScript, assada e pronta para usar. Sempre que vir JavaScript Saindo do Forno, isso significa que se deve digitar o código, como está. Mas confie em nós, você aprenderá tudo sobre esse código em breve e escreverá suas próprias funções.

JavaScript saindo do forno

Este código é para uma função personalizada chamada `touchRock()`, que pede ao usuário que forneça seu nome e, então, exibe um cumprimento personalizado em uma caixa de alerta. A função também muda a imagem da rocha para um iRock sorridente. É tudo o que você precisa para acrescentar personalização ao iRock.

Exatamente como alert(), toda função no JavaScript tem um nome. O nome desta função é touchRock.

prompt() é uma função para exibir uma caixa e obter um valor a partir do usuário.

```
function touchRock() {
  var userName = prompt("Qual o seu nome?", "Enter your name here.");

  if (userName) {
    alert("Prazer em conhecer você, " + userName + ".");
    document.getElementById("rockImg").src = "rock_happy.png";
  }
}
```

Assim que tivermos um nome, cumprimentaremos o usuário pessoalmente...

... e mudaremos a imagem da rocha para uma sorridente.

Poder do cérebro

Você pode descobrir onde esta função deve ficar em sua página irock.html?

Ímãs de geladeira JavaScript

O código iRock amistoso não tem algumas partes principais do código. Você pode preencher as partes que faltam para compor a página?

Sugestão: Não tem certeza de suas respostas? Teste suas respostas digitando-as em sua página irock.html.

```
<html>
  <head>
    <title>iRock - A Rocha Virtual de Estimação </title>

    <............ type="text/javascript">
      function touchRock() {
        var userName = prompt("Qual o seu nome?", "Entre com seu nome aqui.");

        if (userName) {
          alert("Prazer em conhecer você, " + userName + ".");
          document.getElementById("rockImg").src = "rock_happy.png";
        }
      }
    </script>
  </head>

  <body <........... =" <........... {.................................};">
    <div style="margin-top:100px; text-align:center">

      <img id="rockImg" src="rock.png" alt="iRock" style="cursor:pointer"

        ........... ="................;" />
    </div>
  </body>
</html>
```

touchRock( )

alert

onload

script

onclick

'Hello, Eu sou a Rocha de Estimação.'

Solução dos Ímãs de geladeira JavaScript

O código iRock amistoso não tem algumas partes principais do código. Você pode preencher as partes que faltam para compor a página?

As funções JavaScript são colocadas em uma tag <script> especial que fica na <head> da página.

O atributo type da tag <script> é usado para identificar o tipo da linguagem de script, neste caso JavaScript.

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>

    <script type="text/javascript">
      function touchRock() {
        var userName = prompt("What is your name?", "Enter your name here.");

        if (userName) {
          alert("It is good to meet you, " + userName + ".");
          document.getElementById("rockImg").src = "rock_happy.png";
        }
    </script>
  </head>

  <body onload="alert('Hello, Eu sou a Rocha de Estimação.');">
    <div style="margin-top:100px; text-align:center">

      <img id="rockImg" src="rock.png" alt="iRock" style="cursor:pointer"
        onclick="touchRock()" />
    </div>
  </body>
</html>
```

O atributo do evento onload da tag <body> liga o cumprimento da caixa de alerta à página.

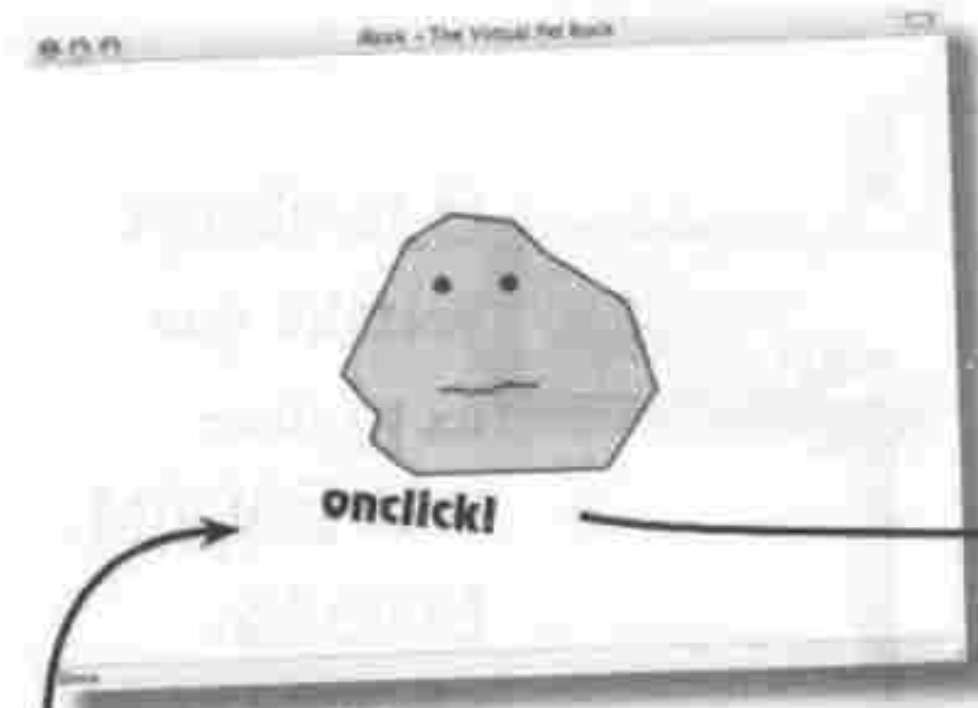
Mude a imagem da rocha para uma feliz.

O atributo do evento onclick da imagem da rocha faz com que a função touchRock() seja chamada quando a rocha é clicada.

Mude o cursor do mouse para uma mão ao passar sobre a rocha.

Reprodução instantânea: o que aconteceu?

Um pouco do JavaScript iniciou muitas mudanças, o que resultou em uma versão mais amável do iRock. Vejamos uma reprodução instantânea de quais alterações foram feitas e como elas comprimem a página.

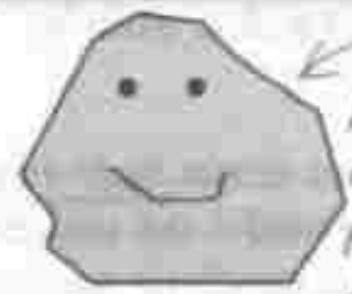


Clicar na imagem iRock faz com que um evento inicie uma função JavaScript personalizada.

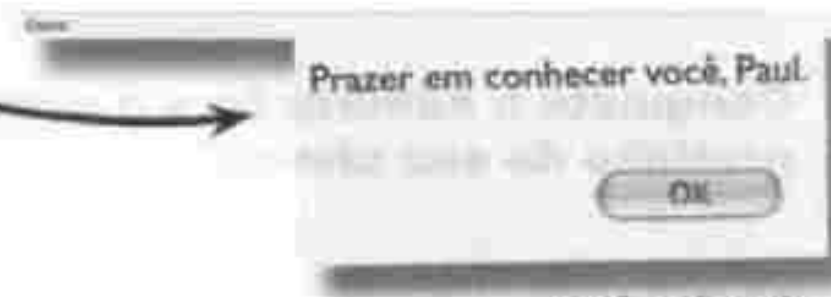
```
<img id="rockImg" src="rock.png" alt="iRock"
  style="cursor:pointer" onclick="touchRock();"
/>
```

```
function touchRock() {
  var userName = prompt("Qual o seu nome?",
                        "Entre com seu nome aqui.");
  if (userName) {
    alert("Prazer em conhecer você, " + userName + ".");
    document.getElementById("rockImg").src = "rock_happy.png";
  }
}
```

A função solicita do nome do usuário e então, cumprimenta-o pessoalmente.

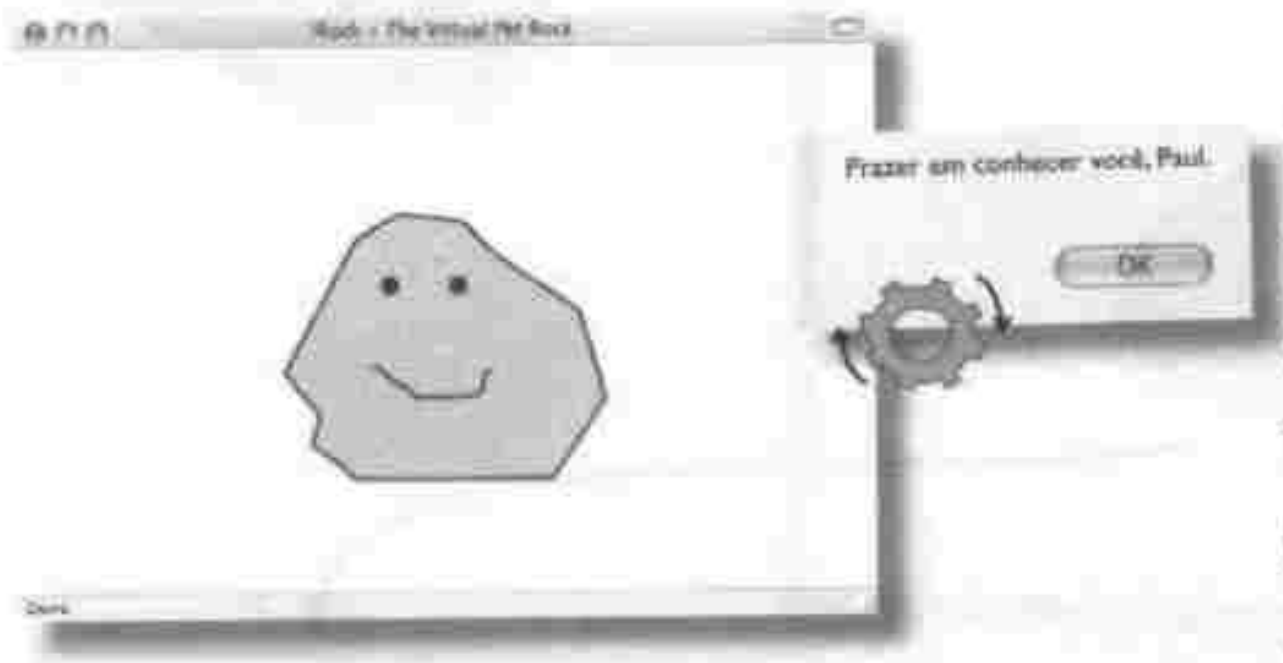


A imagem da rocha muda para um iRock sorridente.



'Test drive' do iRock 1.0

Certifique-se de que tenha deixado sua versão do `irock.html` como a da página 26 e que tenha feito o download de ambas as imagens de rocha em Head First Labs (*www.allabooks.com.br*). Então, abra sua página Web e dê à rocha um giro:

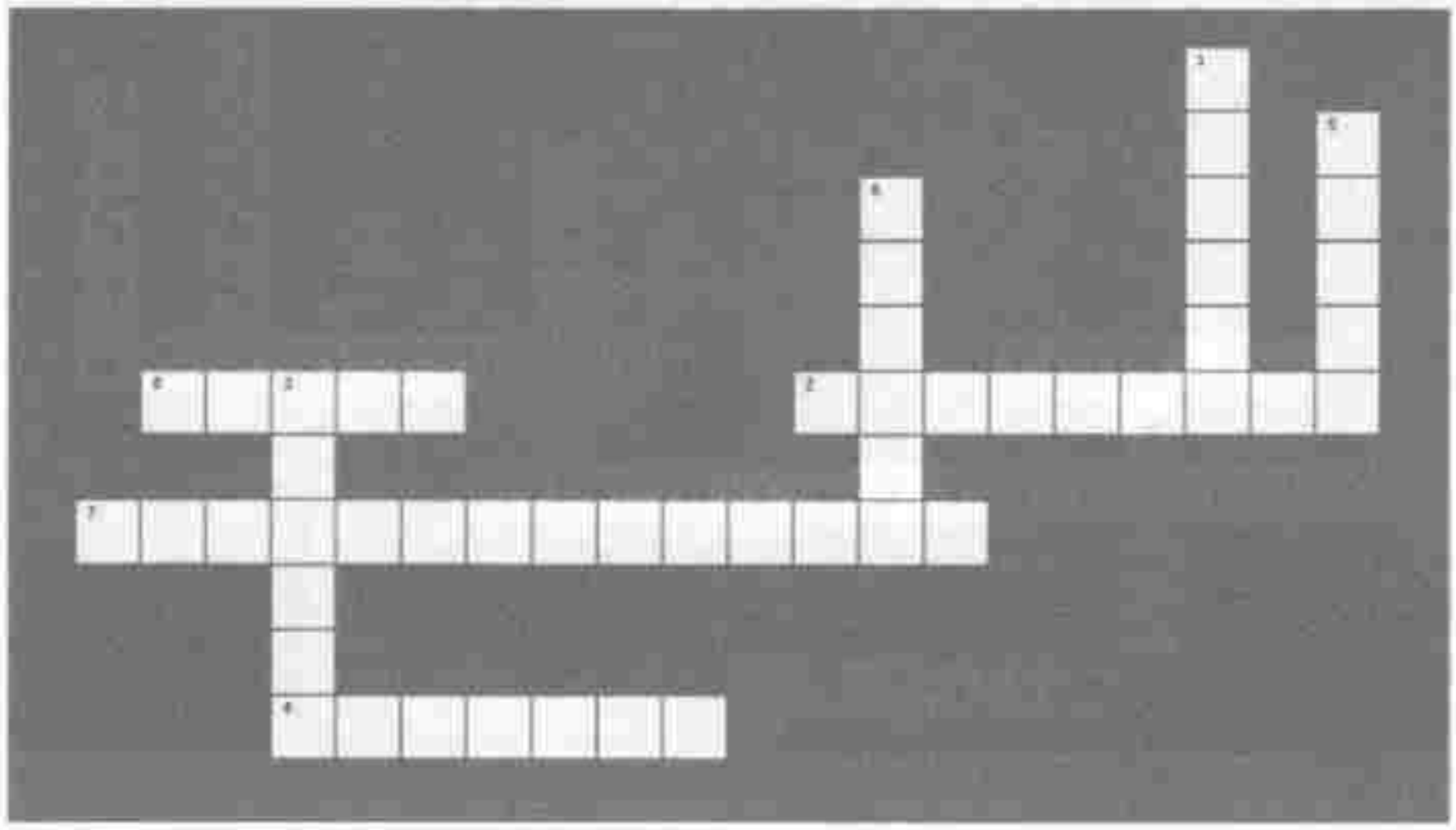


O JavaScript permite que as páginas Web FAÇAM COISAS, não apenas representem e se comuniquem.

1. **Crie a página Web HTML iRock.** ← Feito!
2. **Adicione um alerta JavaScript para fazer com que a rocha cumprimente os usuários quando a página Web iRock for carregada.** ← Também terminou com este!
3. **Escreva o código JavaScript para solicitar o nome do usuário, imprimir um cumprimento personalizado e fazer a rocha sorrir.** ← É aqui que usamos a função touchRock().
4. **Adicione uma sub-rotina de eventos para quando os usuários clicarem na rocha, o código escrito na etapa 3 seja executado.** ← Usamos a sub-rotina de evento onclick para isto.
5. **Conquiste a admiração e a generosa gratidão de seu chefe.** ← O chefe está feliz... um aumento e um monitor widescreen podem demorar muito?

Cruzadas do JavaScript

Reserve algum tempo para descansar e dê ao lado direito do seu cérebro algo para fazer. É sua palavra cruzada padrão; todas as palavras da solução são deste capítulo.



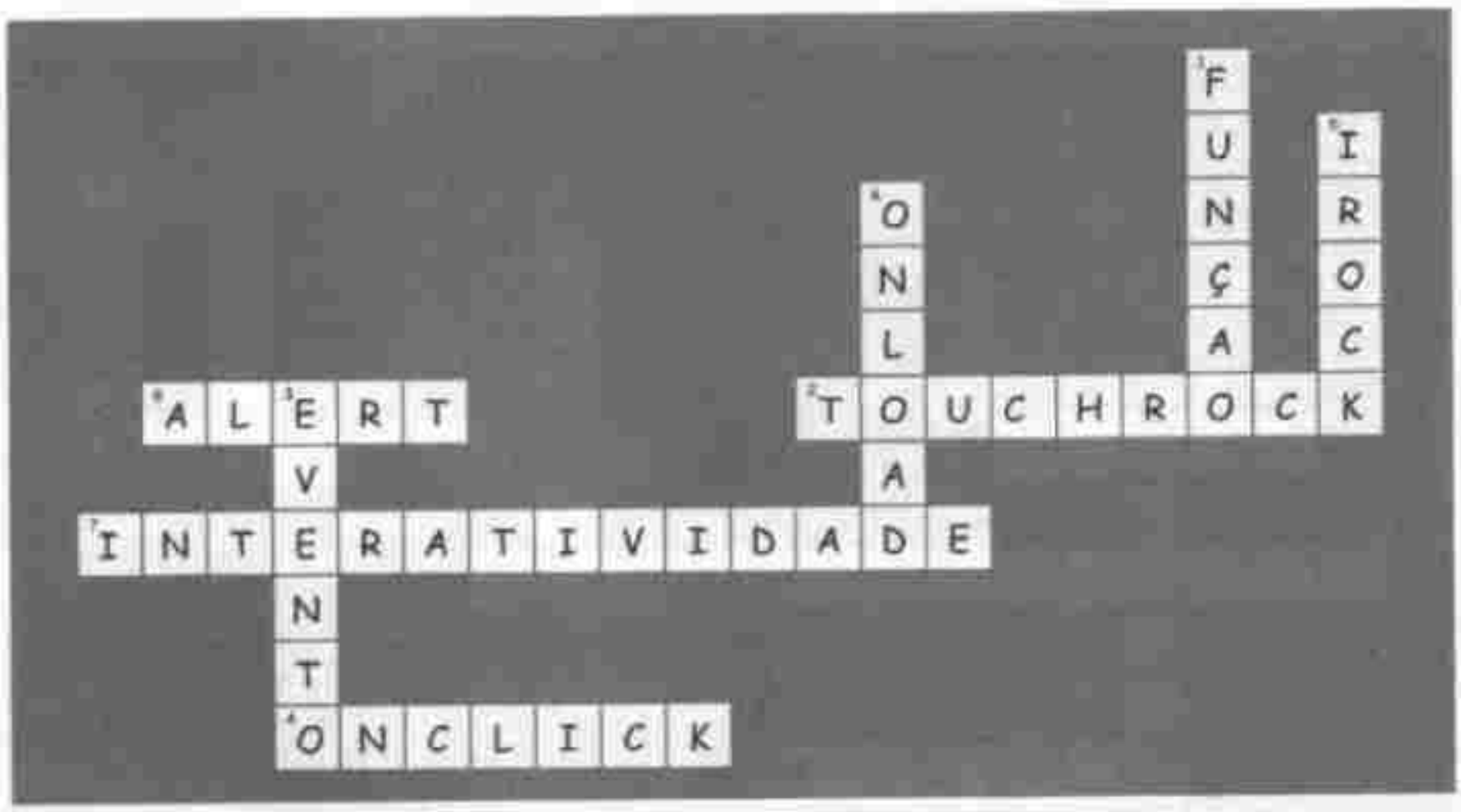
Horizontal

2. O nome de uma parte do código que fornece ao iRock um cumprimento personalizado.
4. Para responder a um clique do mouse, simplesmente defina algum código JavaScript para o atributo..... de um elemento HTML.
7. Sem isto, você pode simplesmente ficar com o HTML e o CSS.
8. Para exibir texto para o usuário, apenas chame a função .....

Vertical

1. Uma parte reutilizável do código JavaScript que executa uma tarefa comum.
3. Algo acabou de acontecer e o navegador está tentando permitir que você saiba.
5. "O brinquedo on-line bom da temporada".
6. Permite que você saiba que uma página Web acabou de carregar.

Solução da Cruzada do JavaScript



Tudo tem seu lugar

No mundo real, as pessoas normalmente ignoram a importância de se ter um lugar para armazenar todas as coisas. No JavaScript nem tanto. Você simplesmente não pode se dar ao luxo de ter um grande closet e uma garagem para três carros. No JavaScript, **tudo tem seu lugar**, e é sua função certificar-se disso. A questão é **dados** – como ***representá-los***, como ***armazená-los*** e como ***encontrá-los***, uma vez que os tenha colocado em algum lugar. Como um especialista em armazenamento JavaScript, você será capaz de pegar um espaço desordenado de dados JavaScript e impor sua vontade em um turbilhão de rótulos virtuais e depósitos de armazenamento.

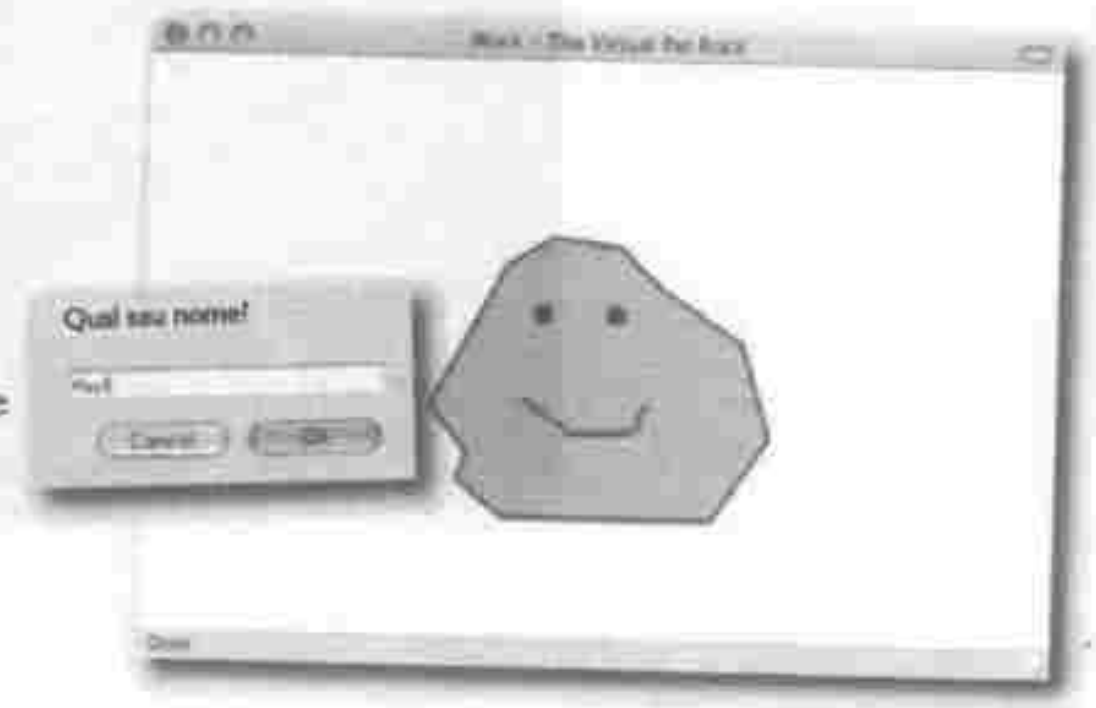
Seus scripts podem armazenar dados

Quase todo script precisa lidar com dados de uma forma ou de outra, e isso geralmente significa armazenar dados na memória. O interpretador de JavaScript que reside nos navegadores Web é o responsável por separar pequenas áreas de armazenamento para os dados JavaScript. No entanto, é sua função explicar claramente **quais são os dados** e **como pretende usá-los**.

As informações associadas a uma pesquisa de casas devem estar todas armazenadas dentro do script que executa os cálculos.

Os scripts usam dados armazenados para realizar cálculos e **lembrar** as informações sobre o usuário. Sem a capacidade de armazenar dados, você nunca encontraria aquela casa nova ou conheceria realmente o seu iRock.

O nome do usuário digitado na página do iRock é armazenado em outro lugar, fazendo com que o script possa exibi-lo em uma saudação personalizada.

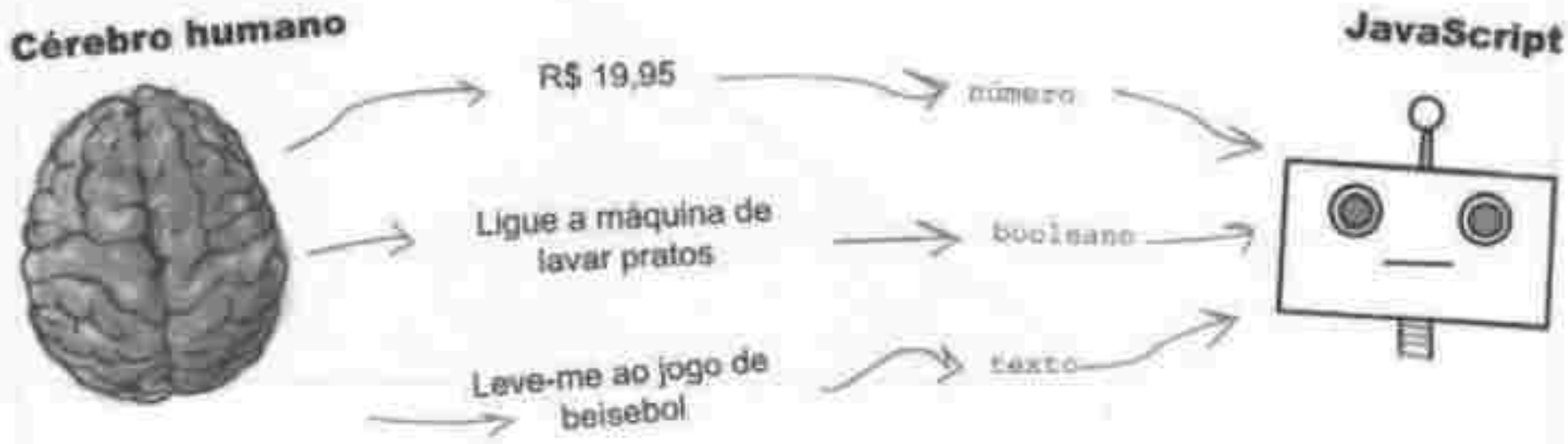


Poder do Cérebro

Pense nos diferentes fragmentos de informações do mundo real com os quais você lida todos os dias. Como eles se parecem? Como organizaria esses diferentes fragmentos de dados?

Os scripts pensam nos tipos de dados

Você organiza e agrupa dados do mundo real em tipos sem nem mesmo pensar nisso: nomes, números, sons, e assim por diante. O JavaScript também agrupa dados de script em tipos de dados. Os tipos de dados são a chave para mapear as informações do seu cérebro para o JavaScript.



O JavaScript usa três tipos básicos de dados: texto, número e booleano.

Número

Os números são usados para armazenar dados numéricos como pesos e quantidades de coisas. Os números em JavaScript também podem ser inteiros (2 quilos) ou decimais (2,5 quilos).

Booleano

Os dados booleanos estão sempre em um ou dois estados possíveis: verdadeiro ou falso (true ou false). Portanto, você pode usar um booleano para representar algo que possua duas definições possíveis, como o botão ligar/desligar de uma torradeira. Os booleanos aparecem a todo momento e você pode usá-los para ajudar na tomada de decisões. Falaremos mais sobre isso no Capítulo 4.

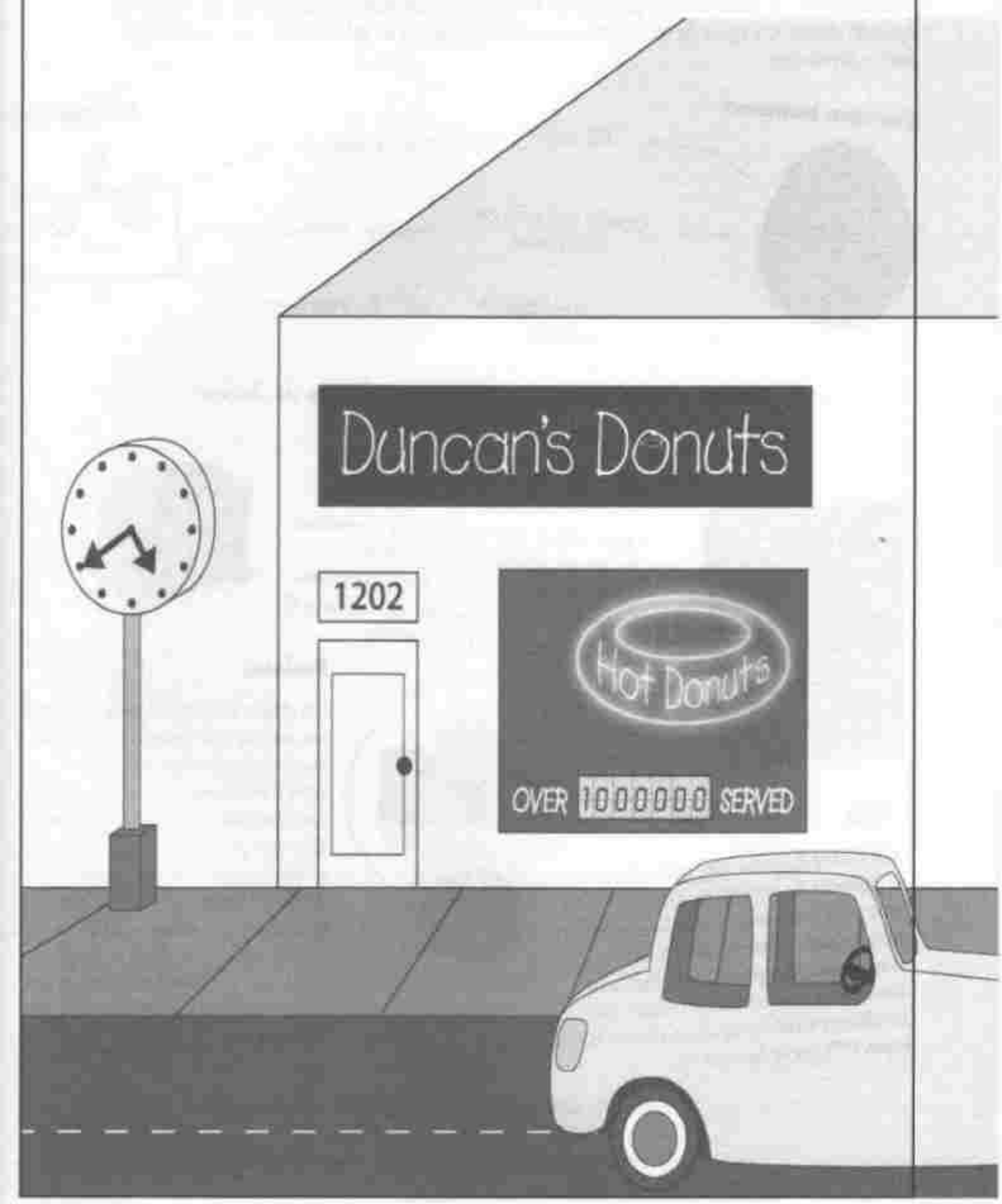
Texto

Os dados de texto são apenas uma seqüência de caracteres, como o nome de seu cereal favorito. Texto geralmente contém palavras ou frases, mas não precisa ter. Também conhecido como strings, o texto em JavaScript também aparece dentro de aspas ("") ou apóstrofo ('').

Os tipos de dados **afetam diretamente** a forma como você trabalha com dados no código JavaScript. Por exemplo, as caixas de alerta exibem texto, não números. Então, os números são convertidos despercebidamente em texto antes de serem exibidos.

Aponte seu lápis

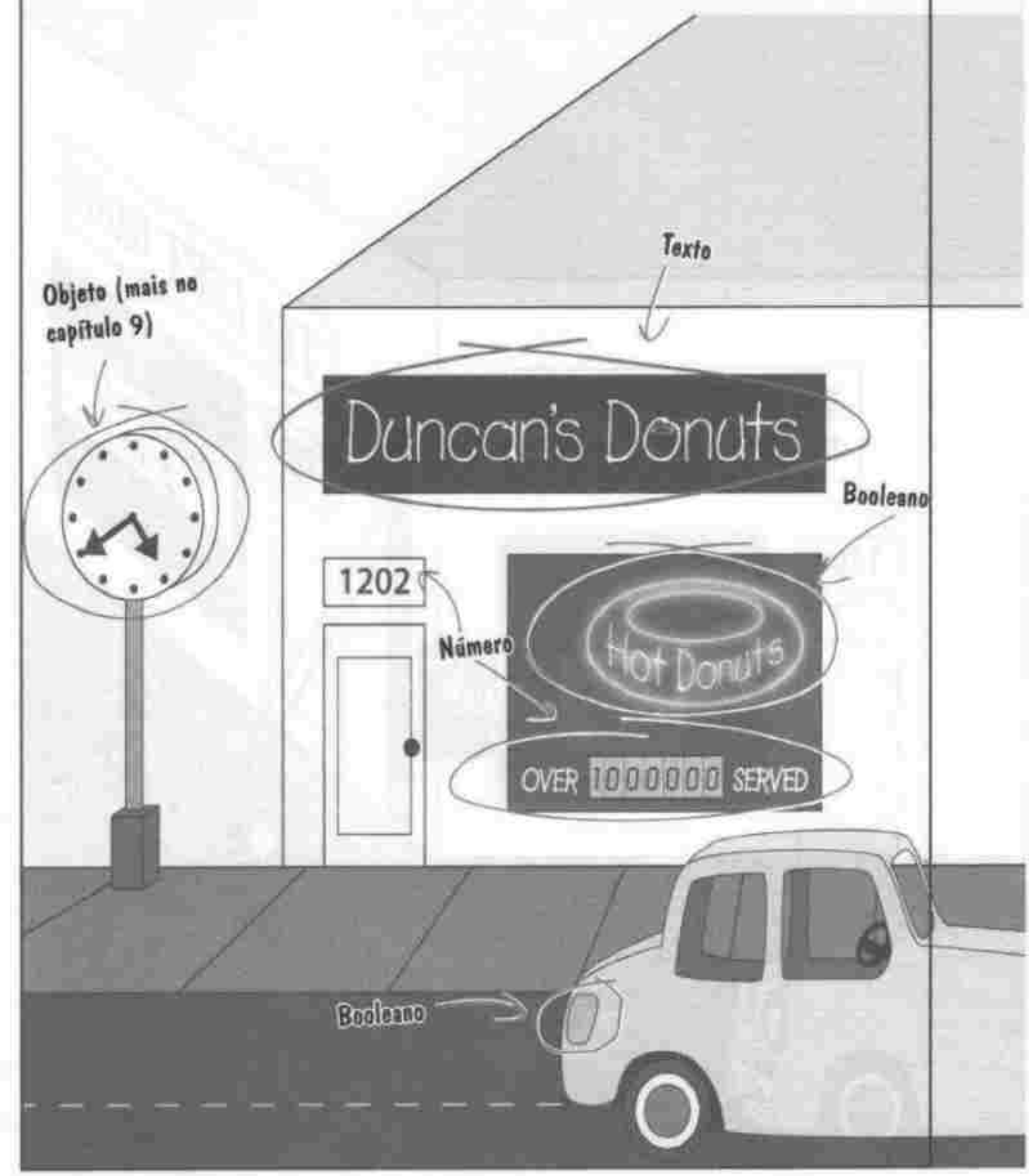
Procure tudo que possa ser representado por um tipo de dados JavaScript, e anote que tipo deveria ser.

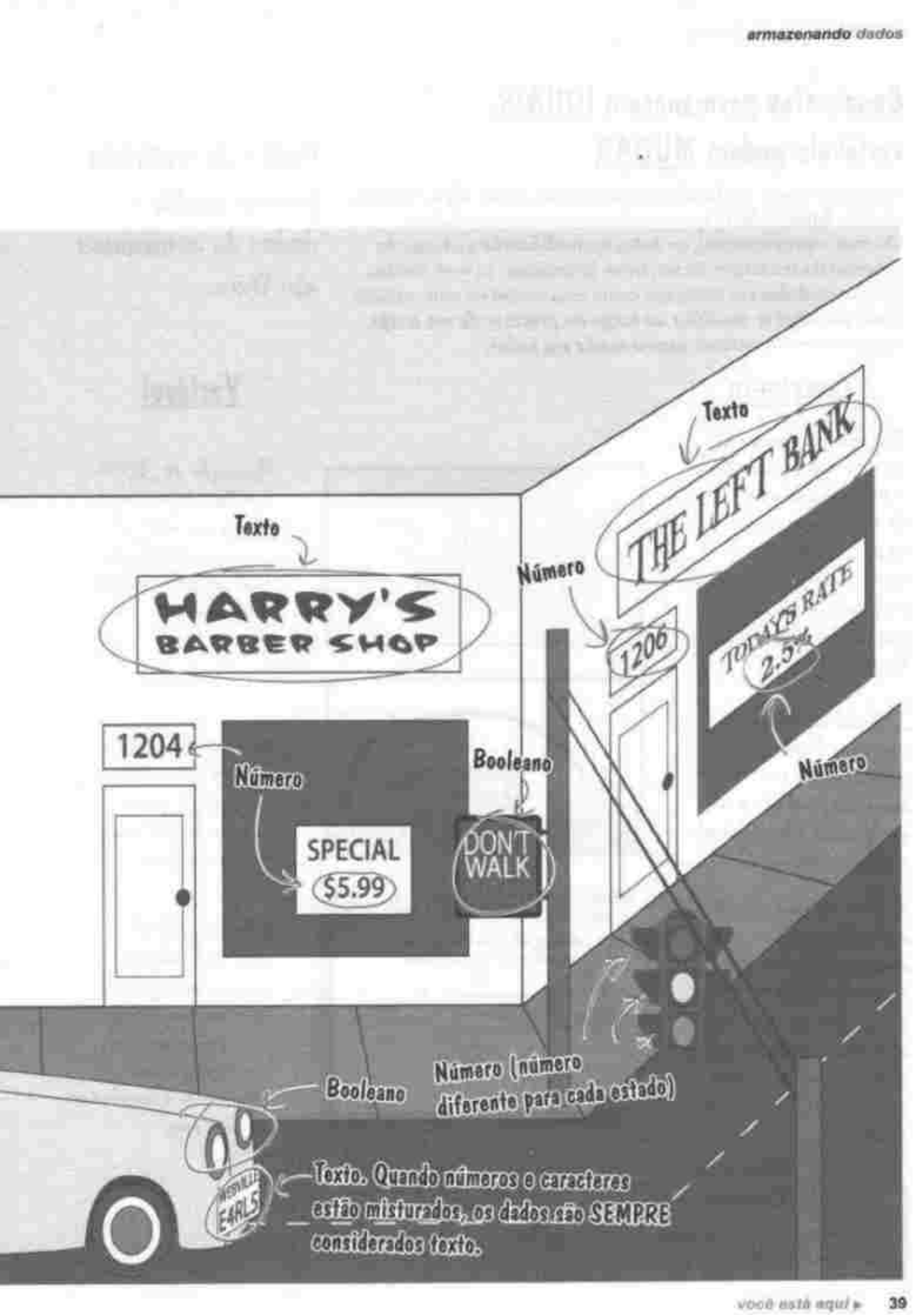


HARRY'S
BARBER SHOP
1204
SPECIAL
$5.99
DON'T
WALK
THE LEFT BANK
1206
TODAYS RATE
2.5%
EARLS

Aponte seu lápis
Solução

Sua tarefa era procurar tudo o que o JavaScript pudesse representar e descobrir o tipo que ele usaria.




Texto
THE LEFT BANK
Texto
HARRY'S
BARBER SHOP
Número
1206
TODAYS RATE
2.5%
Número
1204
Número
Booleano
SPECIAL
$5.99
DON'T
WALK
Booleano
Número (número
diferente para cada estado)
Texto. Quando números e caracteres
estão misturados, os dados são SEMPRE
considerados texto.

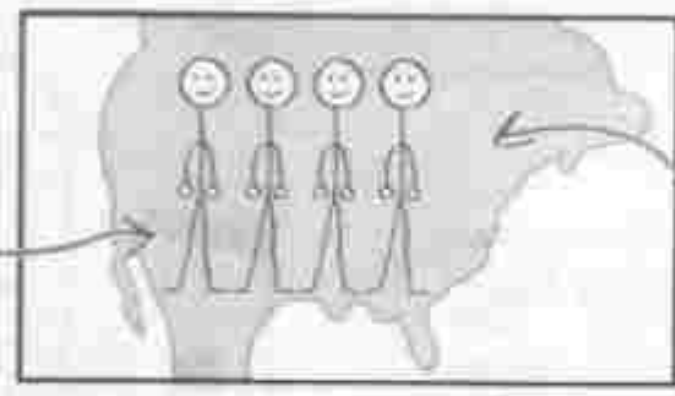
Constantes permanecem IGUAIS, variáveis podem MUDAR

Armazenar dados em JavaScript não significa apenas **tipo**, significa também **finalidade**. O que você deseja fazer com os dados? Ou mais especificamente, os dados se **modificarão** ao longo do processo do seu script? As respostas determinam se você codifica os tipos de dados em JavaScript como uma *variável* ou uma *constante*. ***Uma variável se modifica ao longo do processo de um script***, enquanto ***uma constante nunca muda seu valor.***

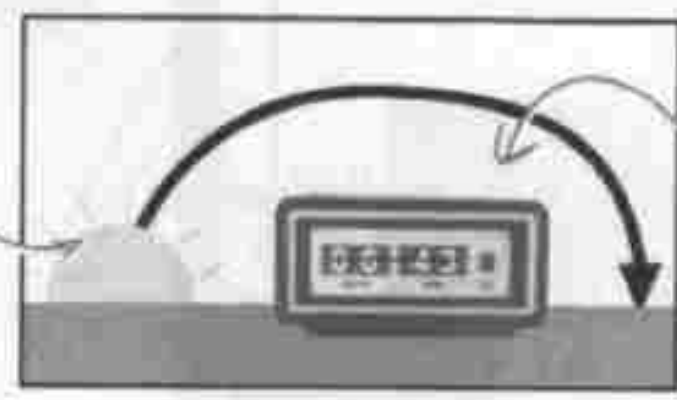
Dados de variáveis podem mudar – dados de constantes são fixos.

Constante

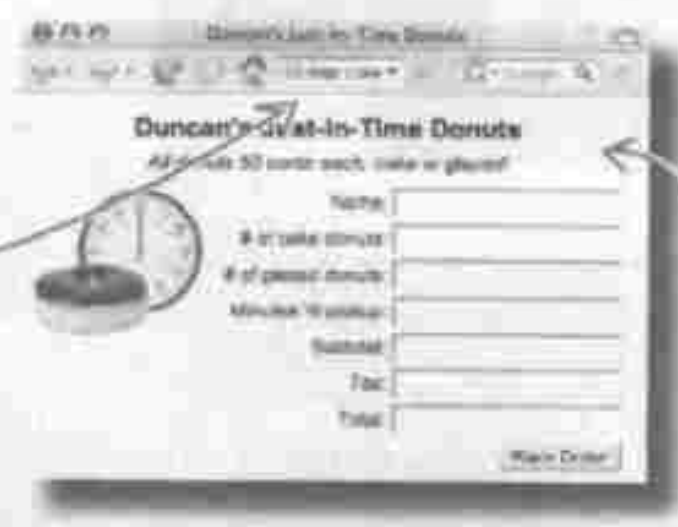
Terreno de 9 milhões de metros quadrados – uma constante (a menos que você espere o tempo suficiente para ver o deslocamento das placas tectônicas da Terra).



24 horas de um dia – uma constante, no que diz respeito aos seres humanos, embora a lua esteja nos abandonando lentamente.



A URL de uma página da Web é www.duncansdonuts.com – uma constante, a menos que os negócios da famosa rosquinha estejam enfrentando um período de retração econômica.



Variável

População de 300 milhões de pessoas – uma variável, já que a população dos EUA ainda continua crescendo.

Nascer do sol às 6:43h – uma variável, já que o nascer do sol muda todos os dias.

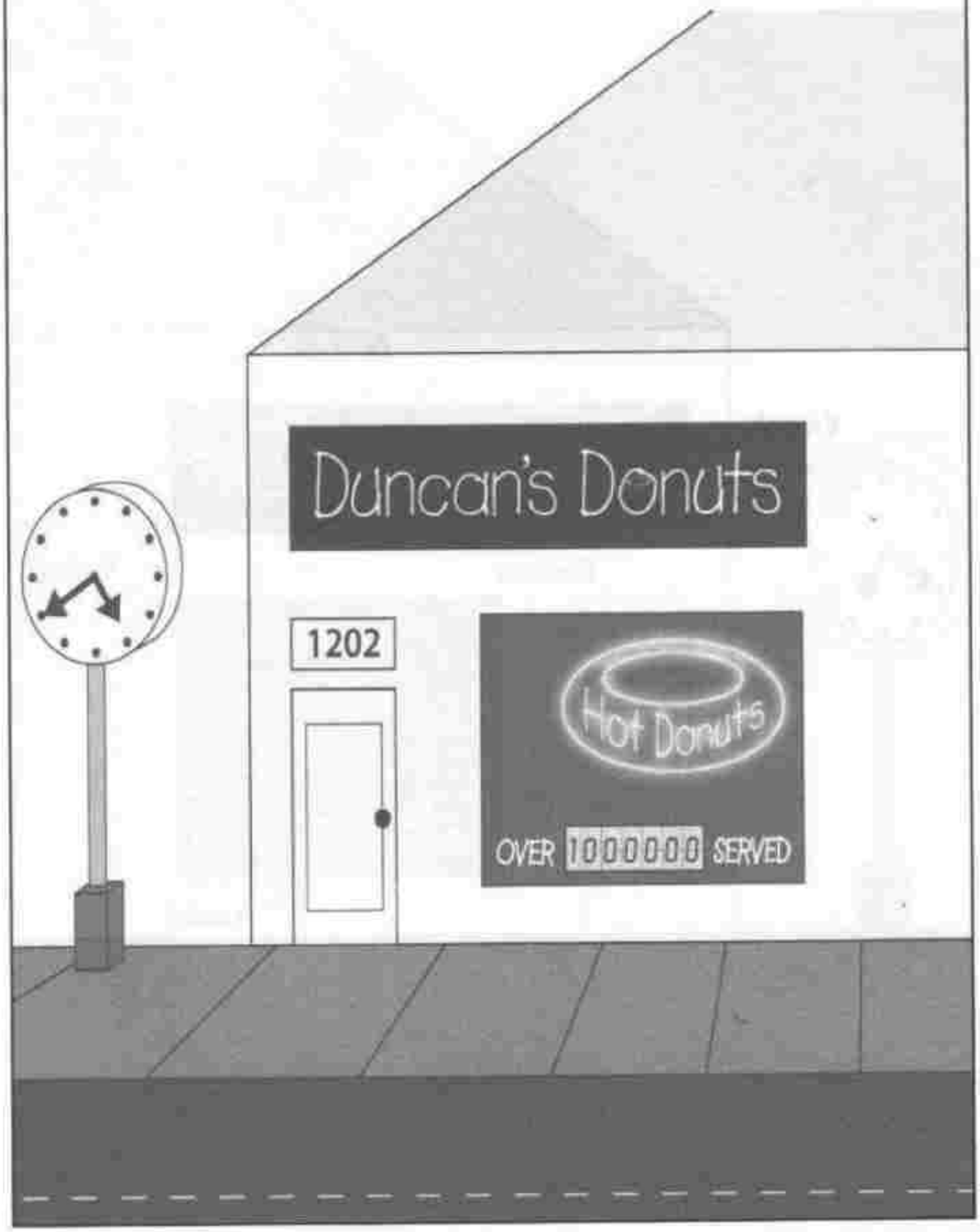
Total de 324 acessos à página – uma variável, já que os usuários estão constantemente visitando a página e alterando o contador de acessos.

Poder do Cérebro

Que outros tipos de informação poderiam envolver tanto as variáveis como as constantes?

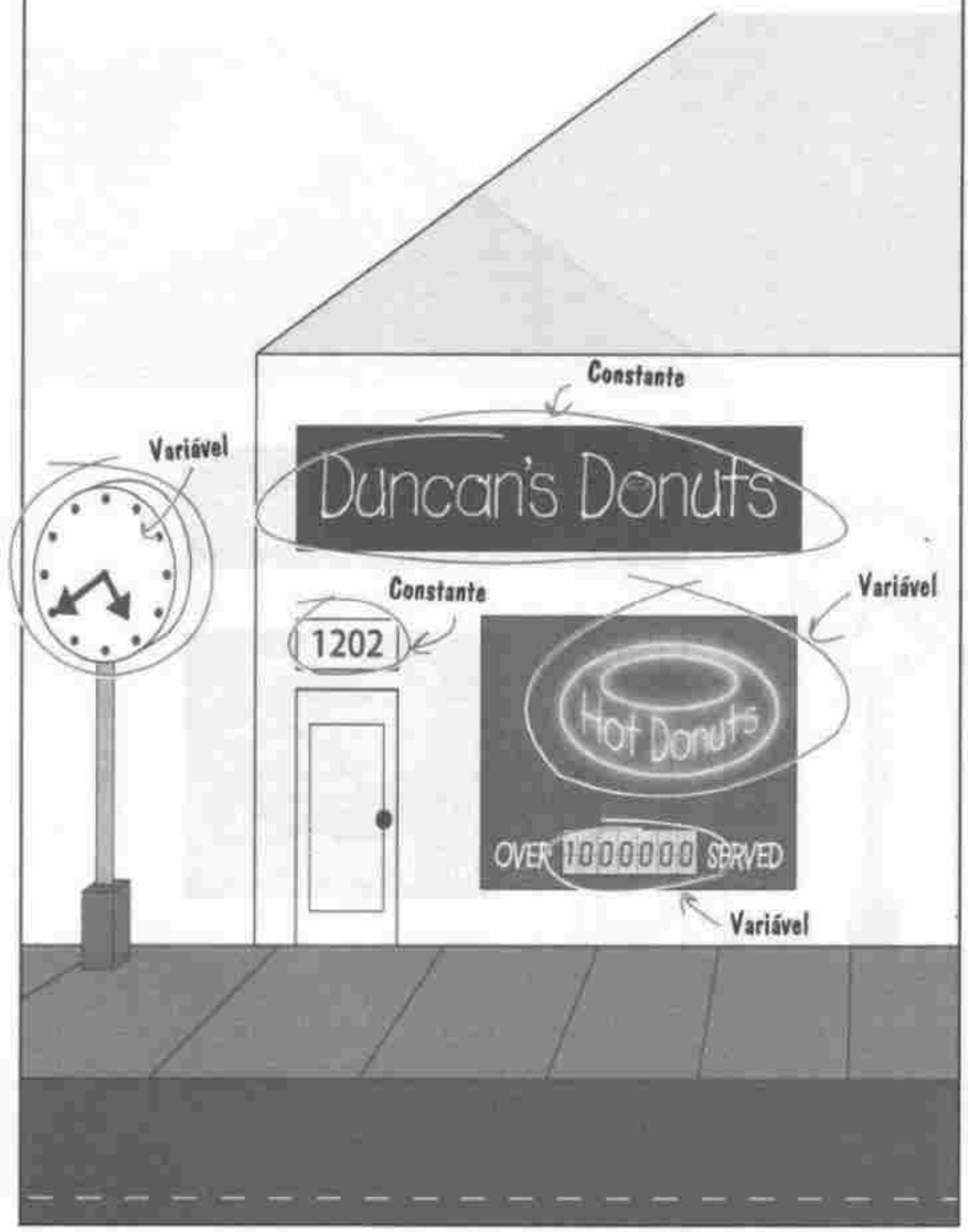
Aponte seu lápis

Faça um círculo em todas as informações do Duncan's Donuts e, em seguida, identifique cada item marcado com um círculo como sendo uma variável ou uma constante.



Aponte seu lápis

Solução Sua tarefa era encontrar todas as variáveis e constantes.



Conversa informal

Conversa dessa noite: **Variável e constante se enfrentam durante o armazenamento de dados.**

Variável

Quando o assunto é armazenar dados, ofereço o máximo em flexibilidade. Você pode mudar meu valor como quiser. Eu posso determinar um valor agora e um outro mais tarde – isso é o que eu chamo de liberdade.

Constante

E eu chamo isso de mudança de opinião! Digo: escolha um valor e se atenha a ele. É a minha persistência cruel que me torna tão valioso aos criadores de script... eles valorizam a capacidade de previsão dos dados que permanecem sempre no processo.

Com certeza, mas a sua resistência em mudar não funcionará em situações onde os dados têm que assumir valores diferentes ao longo do tempo. Por exemplo, a contagem regressiva do lançamento de um foguete tem que mudar no momento em que ele começa a contar de 10 a 1. Enfrente isso!

Ah, então *você* pensa que *é* a única opção de armazenamento de dados para aplicativos de missão crítica, né? Errou! Como você acha que aquele foguete chegou à plataforma de lançamento? Porque alguém foi inteligente o bastante para fazer com que a data de lançamento fosse uma constante. Mostre-me um prazo final que seja uma variável e eu lhe mostrarei um projeto atrasado.

Sim, claro, que seja. Como você qualifica variação como uma coisa ruim? Você não percebe que mudança pode ser uma coisa boa, especialmente quando é necessário armazenar as informações digitadas pelo usuário, realizar cálculos ou qualquer coisa do tipo?

Costumo dizer que quanto mais as coisas mudam, mais elas permanecem as mesmas. E realmente, por que mudar logo de cara? Fixe um bom valor no princípio e deixe-o em paz. Pense na satisfação de saber que um valor não poderá mudar nunca, acidentalmente ou não.

Acho que só precisamos concordar ou discordar.

Na verdade, eu discordei de você desde o início.

As variáveis iniciam sem um valor

Variável é um **local de armazenamento** na memória com um **nome específico**, como uma etiqueta em uma caixa usada para armazenar coisas. Crie uma variável usando uma palavra-chave JavaScript específica chamada var e o nome da nova variável. Uma **Palavra-chave** é a palavra reservada em JavaScript para realizar uma determinada tarefa, como criar uma variável.

O ponto-e-vírgula termina essa linha de código JavaScript.

A palavra-chave var indica que você está criando uma nova variável.

```
var + Variable name + ;
```

O nome da variável pode ser qualquer coisa que você queira, contanto que seja específico dentro de seu script.

Ao criar uma variável usando a palavra-chave var, essa variável se encontra inicialmente vazia... não possui nenhum valor. Tudo bem que uma variável inicie vazia, contanto que você não tente ler seu valor antes de atribuir a ela. Seria como tentar reproduzir uma música em seu tocador de MP3 antes de carregar as músicas para dentro dele.

Sim, essa é uma variável nova.

O fim da linha.

```
var pageHits;
```

O nome da variável é pageHits.

Vazia – pronta para armazenamento

Uma variável recém-criada reservou um espaço de armazenamento separado e está pronta para armazenar dados. E a chave de acesso e a manipulação dos dados que ela armazena é o seu nome. É por isso que é tão importante que o nome de cada variável seja **específico E significativo**. Por exemplo, o nome pageHits oferece uma dica muito boa de qual tipo de dados essa variável armazena. Nomear a variável de acesso a páginas como x ou gerkin teria sido bem menos descritivo.

Inicialize uma variável com "="

Você não precisa criar variáveis sem um valor inicial. Na verdade, é sempre uma boa idéia dar um valor à variável assim que você a cria. Isso se chama **inicializar** uma variável. É só uma questão de adicionar um pequeno fragmento de código extra à rotina de criação de uma variável normal:

Termina a linha de código.

var + Variable name + = + Initial value + ;

O sinal de igual conecta o nome da variável ao seu valor inicial.

Esse valor inicial é armazenado na variável.

Ao contrário de sua contrapartida vazia, uma variável inicializada está pronta para ser usada imediatamente... ela tem sempre um valor armazenado nela. É como comprar um tocador de MP3 pré-carregado – pronto para tocar.

Agora a variável contém dados numéricos.

Atribua o valor à variável.

Fim.

```
var population = 300;
```

Crie a variável.

Nomeie.

Especifique seu valor.

Ainda se lembra dos tipos de dados? Uma outra coisa que esta linha de script faz é atribuir automaticamente o tipo de dados da variável. Neste caso, o JavaScript cria a variável de população como um número, porque você deu a ela um valor inicial numérico de 300. Se alguma vez a variável for atribuída a um outro tipo, então o tipo da variável muda para refletir os novos dados. Na maioria das vezes, o JavaScript trata isso de forma automática; há casos em que será necessário ser explícito e até converter em um tipo de dados diferente... mas chegaremos lá mais adiante.

As constantes são resistentes à mudança

Inicializar uma variável tem a ver com definir seu **primeiro** valor – não há nada que impeça esse valor de ser mudado mais **tarde**. Para armazenar um fragmento de dados que não muda nunca, você precisa de uma constante. Constantes são criadas como variáveis inicializadas, mas use a palavra-chave const em vez de var, e o valor "inicial" se tornará um valor **permanente**... as constantes agem permanentemente!

PRESTE ATENÇÃO!

Nem todos os navegadores suportam a palavra-chave const.

A palavra-chave const é razoavelmente nova para o JavaScript, e nem todos os navegadores a suportam. Verifique os navegadores que serão usados antes de liberar o código JavaScript que usará const.

Cria uma constante que não pode ser mudada.

Atribui um valor à constante.

```
const + Constant name + = + Constant value + ;
```

O nome da constante.

O valor da constante – esse valor não pode mudar nunca.

Está encerrado.

A maior diferença entre criar uma constante e uma variável é que você precisa usar a palavra-chave const em vez de var. A sintaxe é a mesma ao inicializar uma variável. Em muitos casos, as constantes são nomeadas com todas as letras maiúsculas para que se **DESTAQUEM** das variáveis em seu código.

Esses dados nunca, nunca, nunca mudam... nunca!

```
const TAXRATE = .925 ;
```

O valor que a constante terá por toda a eternidade.

Esses dados não podem mudar.

O nome da constante com TODAS AS LETRAS MAIÚSCULAS ajuda a torná-la facilmente identificável se comparado a variáveis que usam letra maiúsculas.

Constantes são úteis para armazenar informações que você poderia codificar diretamente em um script, como "tax rate" (alíquota de imposto sobre vendas). Em vez de usar um número como 0,925, seu código ficaria muito mais fácil de entender se você usasse uma constante com um nome descritivo, como TAXRATE. E se alguma vez precisar mudar o valor da constante no script, faça a mudança em um único lugar – onde a constante é definida – em vez de tentar achá-la sempre que ela aparecer em seu script, o que poderia se tornar realmente complicado.

Espere aí! Eu pensei que constantes não pudessem mudar.

Constantes não podem mudar, pelo menos não sem um editor de texto.

É verdade que constantes não podem mudar enquanto um script estiver sendo **executado**... mas não há nada que o impeça de mudar o valor de uma constante onde ela é criada originalmente. Assim, a partir da perspectiva de seu script, com certeza uma constante é fixa, mas a partir da sua perspectiva, ela pode ser mudada retornando ao ponto onde você a criou. Então, uma constante de alíquota de imposto não pode mudar enquanto o script estiver sendo executado, mas você pode mudar a alíquota em seu código de inicialização, e o novo valor da constante será refletido no script daquele ponto em diante.

Exercício

Determinar se cada uma das informações a seguir é uma variável ou uma constante e, em seguida, escrever o código para criar cada uma delas e inicialize-as (caso seja apropriado).

A temperatura atual, que inicialmente é desconhecida

A unidade de conversão de idade de seres humanos em idade de cães (1 ano humano = 7 anos caninos)

A contagem regressiva de lançamento de um foguete (de 10 a 0)

O preço de um delicioso donut (50 centavos)

Solução do Exercício

Sua tarefa era determinar se cada uma das informações a seguir é uma variável ou uma constante e, em seguida, escrever o código para criar cada uma delas e inicializá-las quando fosse apropriado.

A temperatura atual, que inicialmente é desconhecida

A unidade de conversão de idade de seres humanos em idade de cães (1 ano humano = 7 anos caninos)

A contagem regressiva de lançamento de um foguete (de 10 a 0)

O preço de um delicioso donut (50 centavos)

var temp;

A temperatura muda toda hora e o valor é desconhecido, portanto uma variável vazia é o mais indicado.

const HUMANTODOG = 7;

Essa taxa de conversão não muda, portanto faz bastante sentido como uma constante.

var countdown = 10;

A contagem regressiva tem que ser contada de 10 a 1, portanto ela é uma variável e precisa ser inicializada para começar a contar (10).

var donutPrice = 0,50; ou const DONUTPRICE = 0,50;

Se o preço do donut mudar, faz sentido como uma variável que é inicializada ao preço atual.

...ou talvez o preço do donut seja fixo, e neste caso uma constante atribuída ao preço funcionará melhor.

Não existem perguntas idiotas

P: Se eu não especificar o tipo de dados, como o JavaScript saberá qual é o tipo?

R: Ao contrário de algumas linguagens de programação, o JavaScript não permite definir explicitamente o tipo de uma constante ou variável. Em vez disso, o tipo fica **implícito** quando você define o valor dos dados. Isso permite bastante flexibilidade às variáveis do JavaScript, já que seus tipos de dados podem mudar quando valores diferentes forem atribuídos a eles. Por exemplo, se você atribuir o número 17 a uma variável chamada x, a variável será um número. Mas se, ao contrário, atribuir o texto "dezessete" a x, o tipo de variável muda para string.

P: Se os tipos de dados dos dados JavaScript estiverem cuidando disso automaticamente, por que eu deveria me preocupar com os tipos de dados?

R: Porque existe um monte de situações onde você não pode contar somente com o tratamento automático dos tipos de dados do JavaScript. Por exemplo, talvez você tenha um número armazenado como texto que queira usar em um cálculo. Você precisa converter o tipo de texto em um tipo de número para realizar qualquer cálculo matemático com o número. O inverso é verdadeiro quando um número é exibido em uma caixa de alerta – ele precisa ser convertido primeiro em texto. O JavaScript realizará automaticamente a conversão do número em texto, mas é provável que ele não o converta exatamente como deseja.

P: Há algum problema em deixar uma variável não-inicializada, se eu não sei qual o valor de início?

R: Com certeza. A idéia por detrás da inicialização é tentar evitar problemas onde você acessa uma variável quando não houver um valor. Porém, também há vezes em que não existe nenhuma forma de saber o valor de uma variável ao criá-la pela primeira vez. Se isso acontecer, certifique-se de que a variável foi definida antes de tentar usá-la. E lembre-se de que você pode inicializar sempre uma variável para "nenhum" valor, assim como " " para texto, 0 para números ou false para booleanos. Isto ajuda a eliminar o risco de se acessar acidentalmente os dados não-inicializados.

P: Existe algum truque para se saber quando usar uma variável e quando usar uma constante?

R: Embora seja fácil dizer que constantes não podem mudar e variáveis podem, há algo mais do que isso. Em muitos casos, você começará usando variáveis para tudo, e só perceberá que existem oportunidades de transformar algumas dessas variáveis em constantes mais adiante. Mesmo assim, é raro poder transformar uma variável em uma constante. Muito provavelmente, você terá um fragmento fixo do texto ou do número usado em vários lugares, como uma saudação repetitiva ou uma taxa de conversão. Em vez de duplicar o texto ou o número várias vezes, crie uma constante para ele e use-a. Então, se você precisar ajustar ou mudar o valor, poderá fazê-lo em apenas um lugar em seu código.

P: O que acontece com os dados do script quando uma página da Web é recarregada?

R: Os dados do script retornam aos seus valores iniciais, como se o script nunca tivesse sido executado antes. Em outras palavras, atualizar uma página da Web causa o mesmo efeito sobre o script, como se ele estivesse sendo executado pela primeira vez.

Tipos de dados são estabelecidos quando os valores das variáveis e das constantes são definidos.

Pontos de Bala

- Geralmente, dados de script podem ser representados por um dos três tipos básicos de dados: **texto, número** ou **booleano**.
- Uma **variável** é um fragmento de dados que **pode mudar** durante o processo de um script.
- Uma **constante** é um fragmento de informação que **não pode ser mudado**.
- A palavra-chave var é usada para **criar variáveis**, enquanto const é usada para **criar constantes**.
- O **tipo de dados** de um fragmento de dados JavaScript é estabelecido quando você **atribui dados** a um certo valor, e o tipo das variáveis pode mudar.

O que contém um nome?

Variáveis, constantes e outros construtores de sintaxe do JavaScript são identificados nos scripts pelo uso de nomes conhecidos como **identificadores**. Os identificadores JavaScript são como nomes de pessoas no mundo real, exceto se eles não forem flexíveis (as pessoas podem ter o mesmo nome, mas as variáveis JavaScript não). Além de serem específicos dentro de um script, os identificadores devem obedecer a algumas leis de nomenclatura formuladas pelo JavaScript:

- Um identificador deve ter pelo menos 1 caractere.
- O primeiro caractere de um identificador precisa ser uma letra, um sublinhado (_) ou um cifrão ($).
- Cada caractere, após o primeiro caractere, pode ser uma letra, um sublinhado (_), um cifrão ($) ou um número.
- Espaços e caracteres especiais diferentes de _ e $ não são permitidos em nenhuma parte de um identificador.

Não vou tolerar transgressores da lei, quando o assunto for identificadores.

Ao criar um identificador JavaScript para uma variável ou uma constante, você está nomeando um fragmento de informações que normalmente possui **significado** dentro de um script. Por isso, não basta apenas obedecer às leis de nomenclatura dos identificadores. Você deve adicionar contexto aos nomes de seus fragmentos de dados, para que eles sejam imediatamente **identificáveis**.

Obviamente, existem momentos em que um simples x cumpre a tarefa – nem todo fragmento de dados em um script tem seu propósito facilmente descrito.

Xerife J.S. Justiça, um policial dedicado.

Identificadores devem ser descritivos, fazendo com que os dados sejam facilmente identificáveis, além de legais...

Variáveis legais e ilegais e nomes de constantes

Não legal: não pode começar com um número.

firstName

Legal: todas as letras, então está tudo bem.

top100

Legal: os números não aparecem no início, então estão em perfeitas condições.

ka_chow

Legal: letras e sublinhados são excelentes.

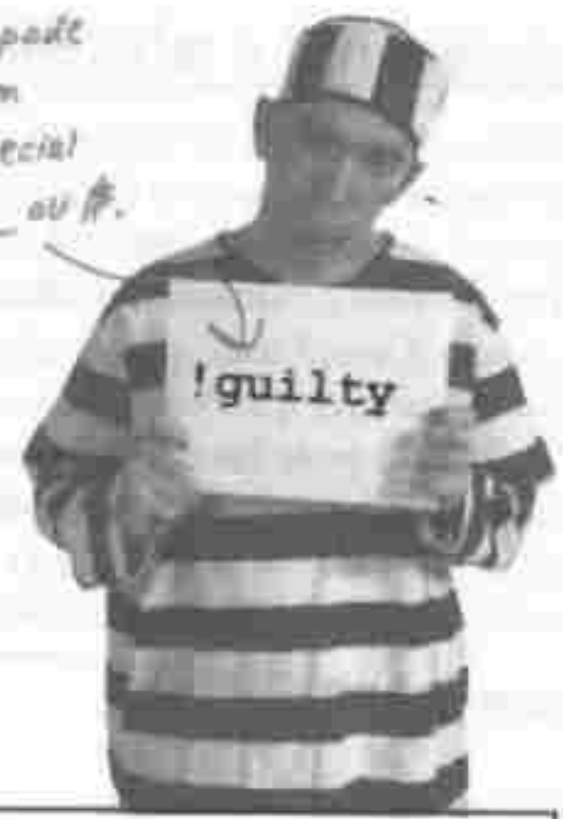
topSecret

Legal: não há nenhum problema em começar com sublinhado – algumas pessoas até usam essa técnica para nomear variáveis que possuem um significado especial.

Não legal: não pode começar com um caractere especial diferente de _ ou $.

$total

Legal: embora pareça um pouco estranho, começar com um cifrão é perfeitamente legal.

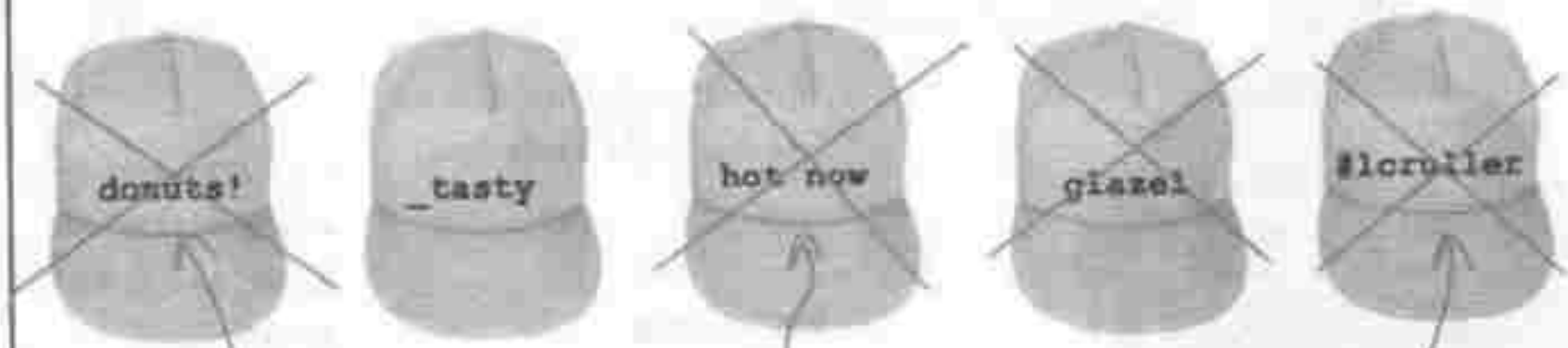


Exercício

Os magos da massa de donut do Duncan estão tentando decidir sobre o design de um boné promocional. Infelizmente, eles não perceberam que alguns dos designs transgridem as regras do JavaScript na nomeação de identificadores. Marque um X sobre os nomes nos bonés que não se adaptarão ao JavaScript.

Solução do Exercício

Sua tarefa era marcar um X sobre os bonés que possuem nomes de variáveis que não se adaptarão ao JavaScript.



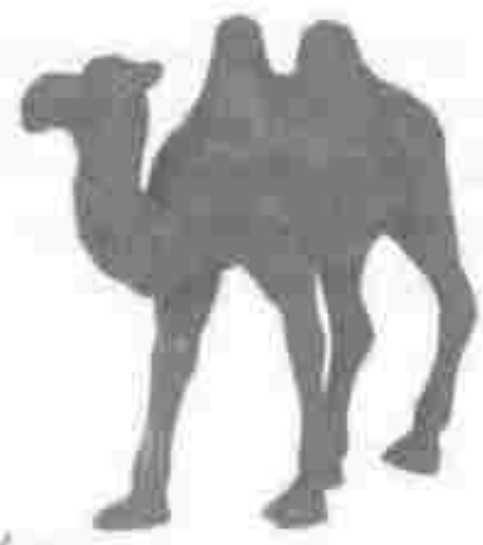
Pontos de exclamação não são permitidos em qualquer lugar de um identificador.

Desculpe, espaços também não são permitidos.

O símbolo da libra só vai invocar a fúria do Xerife Justiça.

Os nomes das variáveis usam letras maiúsculas e minúsculas (CamelCase)

Embora não exista nenhuma lei em JavaScript que regule uma forma de intitular os nomes de identificadores, a comunidade JavaScript possui alguns padrões **não oficiais**. Um desses padrões é usar **letras maiúsculas e minúsculas alternadas,** como uma corcunda de camelo (CamelCase), que significa misturar tipos de letra dentro de identificadores que consistem em mais de uma palavra (lembre-se de que não pode haver espaços em um nome de variável). Variáveis normalmente usam **letras minúsculas (camelCase)**, onde a primeira palavra é toda composta de letras minúsculas, mas as palavras adicionais possuem letras maiúsculas e minúsculas misturadas.



num_cake_donuts

Separar diversas palavras com um sublinhado em um identificador de variável não é ilegal, mas existe uma forma melhor.

A primeira letra de cada palavra está em maiúscula.

NumCakeDonuts

Melhor... esse estilo é conhecido como letras maiúsculas e minúsculas (CamelCase), mas ainda não é o totalmente correto para variáveis.

A primeira letra de cada palavra, exceto a primeira, está em maiúscula.

numCakeDonuts

Ah, aí está – letras minúsculas são perfeitas para nomear variáveis com diversas palavras.

letras minúsculas (camelCase) são usadas para nomear variáveis com diversasPalavras.

Imãs de geladeira JavaScript

Os imãs identificadores foram separados das variáveis e constantes identificadas nos donuts do Duncan. Correlacione o imã correto a cada variável/constante, e certifique-se de evitar os imãs com nomes ilegais. Pontos de bonificação: identificar cada tipo de dados.

O número de xícaras de café vendidas hoje

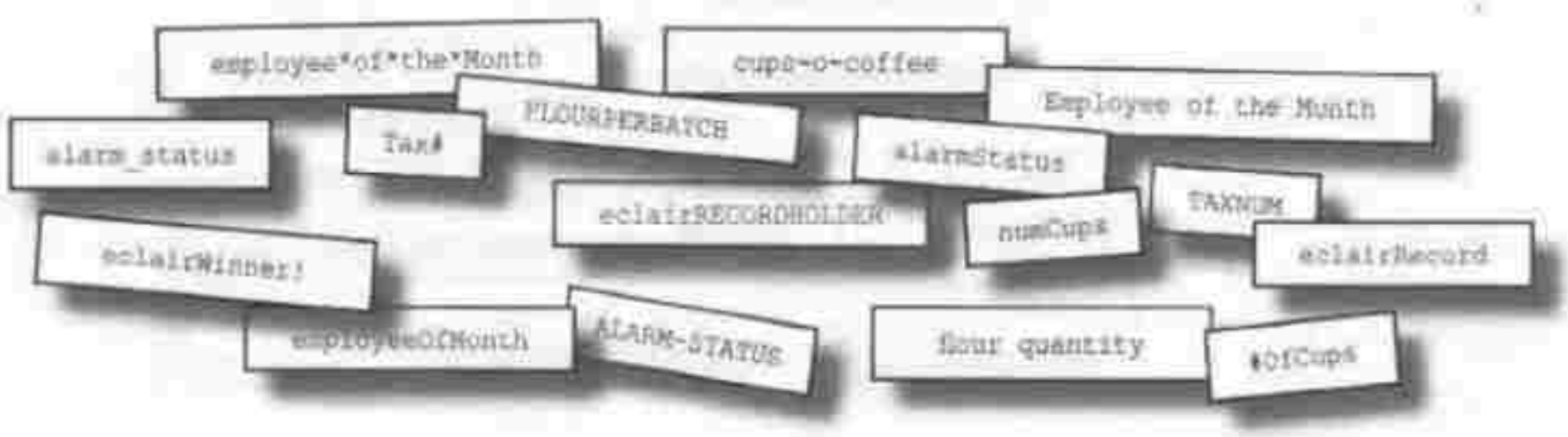
O nome do funcionário do mês

O montante de farinha que entra em uma única porção de donuts

O recordista dos biscoitinhos mais comidos em uma reunião

O status do sistema de alarme

O número do imposto sobre atividades empresariais usado para ajustar o imposto sobre vendas



Solução dos Imãs de geladeira Javascript

Os imãs identificadores foram separados das variáveis e constantes identificadas nos donuts do Duncan. Correlacione o imã correto a cada variável/constante, e certifique-se de evitar os imãs com nomes ilegais. Pontos de bonificação: identificar cada tipo de dados.

O número de xícaras de café vendidas hoje

numCups

Número

O nome do funcionário do mês

employeeOfMonth

Texto

O montante de farinha que entra em uma única porção de donuts

FLOURPERBATCH

Número

O recordista dos biscoitinhos mais comidos em uma reunião

eclairRecord

Texto

O status do sistema de alarme

alarmStatus

Booleano

O número do imposto sobre atividades empresariais usado para ajustar o imposto sobre vendas

TAXNUM

Número

Todos os nomes restantes são ilegais em JavaScript.

cups-o-coffee

Employee of the Month

alarm_status

eclairWinner!

eclairRECORDHOLDER

employee*of*the*month

Tax#

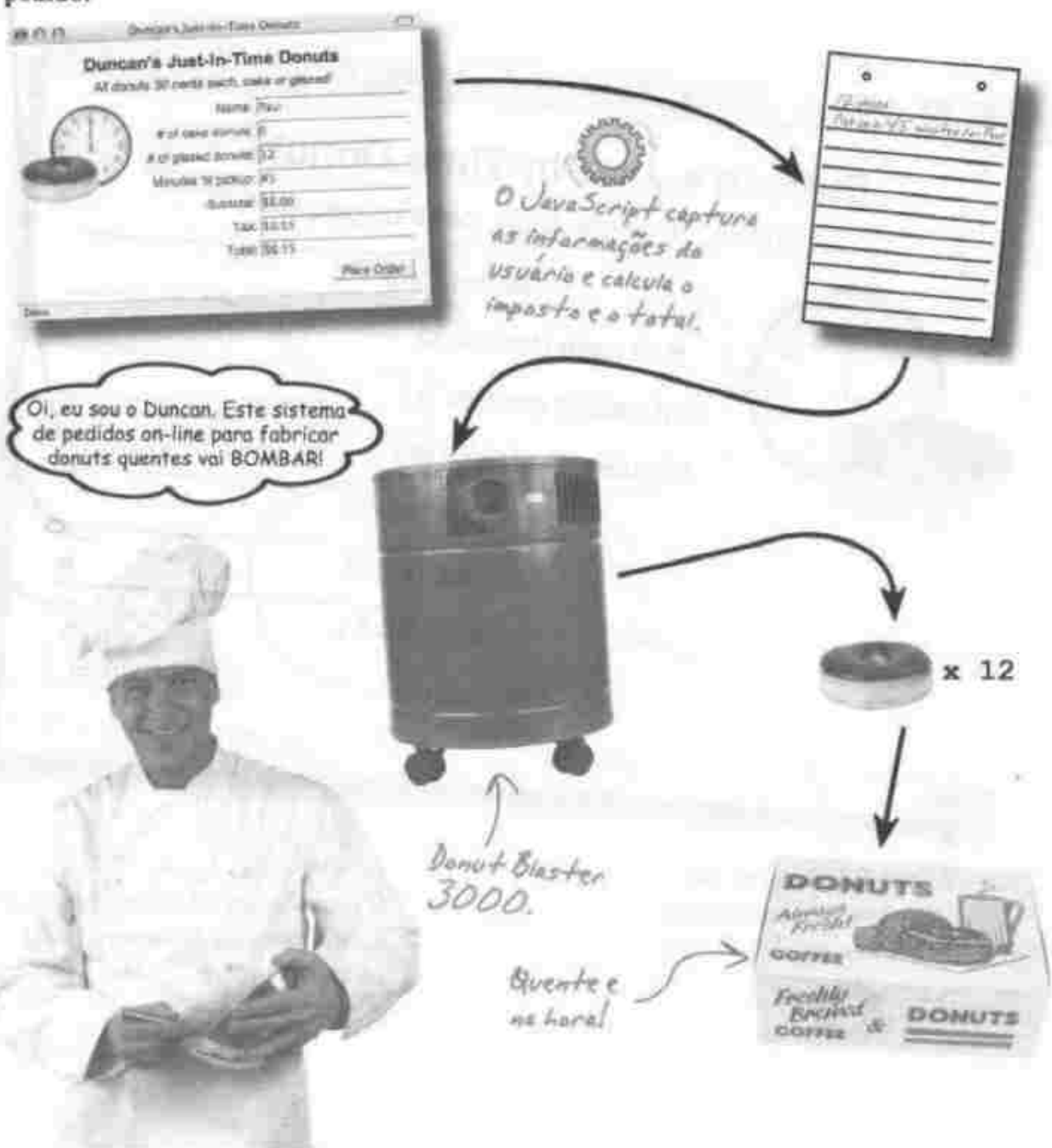
ALARM-STATUS

flour quantity

#OfCups

O próximo passo importante (em donuts)

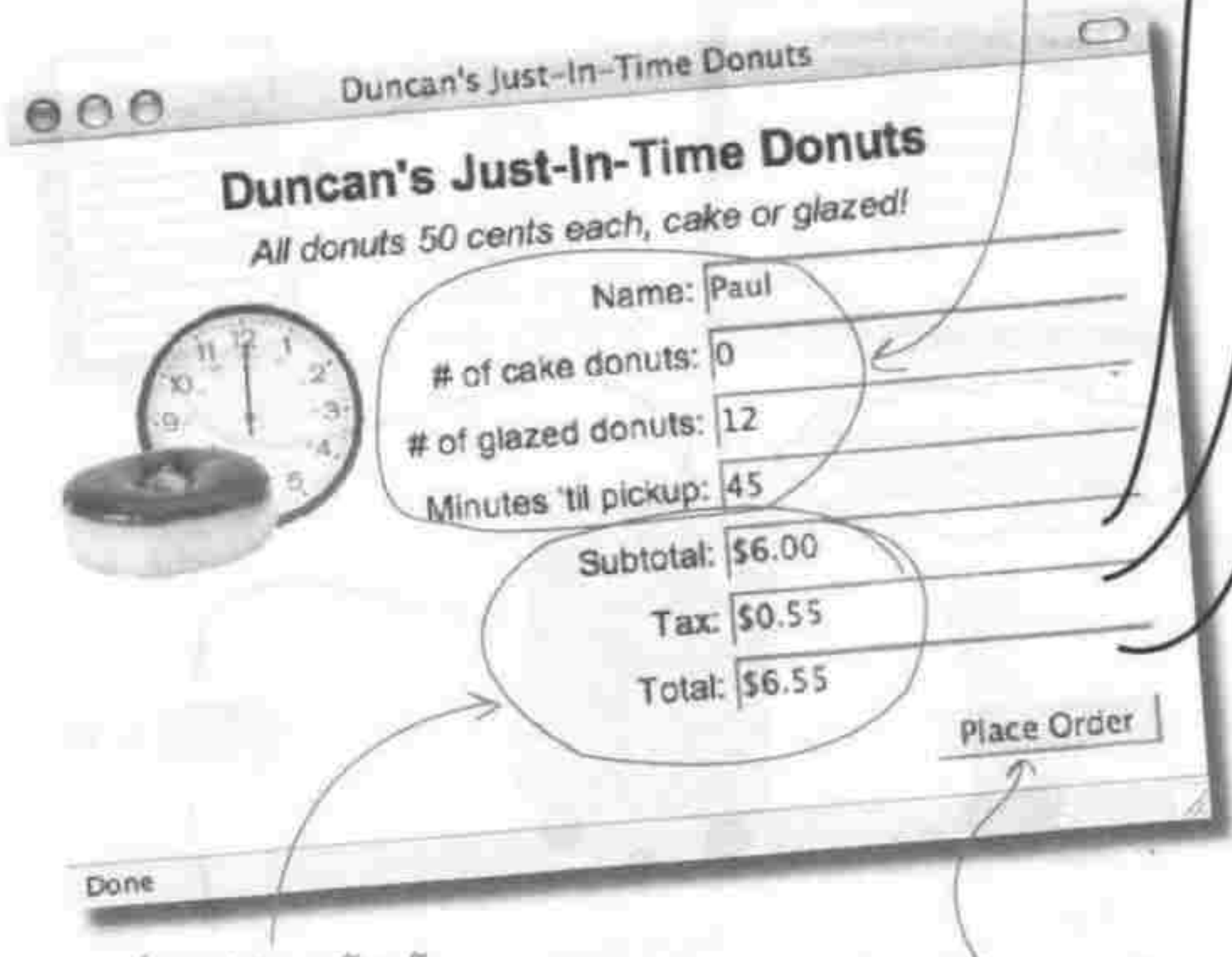
Você deve conhecer os donuts do Duncan, mas você não conheceu o Duncan ou ouviu falar de seu grande plano para agitar o mercado de donuts. Duncan quer levar o segmento das "Donuts Quentes" ao próximo nível... ele quer colocá-los on-line! A idéia é: **donuts com hora marcada**, onde você faz um pedido on-line digitando um horário específico para pegá-las, e terá um pedido **quente** de donuts esperando por você no tempo determinado. **Sua função é verificar se o usuário digita os dados exigidos, bem como calcular o imposto e o total do pedido.**



Projete uma página da Web para o Duncan's Donuts

Processar um pedido de donuts com hora marcada envolve a verificação (ou validação) dos dados exigidos no formulário de pedido e o cálculo do total do pedido baseado nesses dados. O subtotal e o total são calculados **dinamicamente** enquanto os dados são digitados, fazendo com que o usuário obtenha a **resposta imediata** sobre o preço total. O botão Place Order (Faça seu pedido) serve para enviar o pedido final, o que realmente não é um problema do JavaScript... não estamos preocupados com isso aqui.

Essas informações são necessárias para o pedido, e devem ser validadas pelo JavaScript.



Essas informações são calculadas dinamicamente com o uso do JavaScript.

O JavaScript não é necessário para o envio do formulário final ao servidor Web.

Calcula-se o subtotal multiplicando-se o número total de donuts pelo preço de cada um:

(Nº de donuts simples + Nº de donuts com cobertura) x preço por donuts

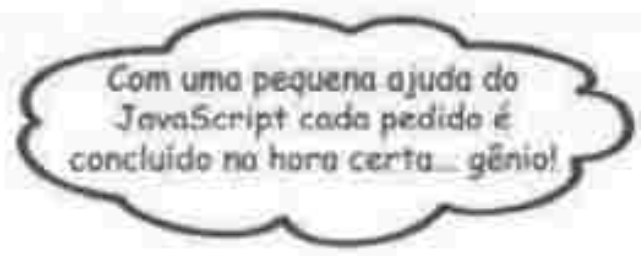
Calcula-se o imposto multiplicando-se o subtotal pela alíquota do imposto:

subtotal x alíquota do imposto

Calcula-se o total do pedido adicionando-se o subtotal e o imposto:

subtotal + imposto

Parece que o Duncan possui uma quantidade considerável de dados em seu formulário para verificação. Ele precisa manter não só os vários fragmentos de informações digitadas pelo usuário, como também existem vários fragmentos de dados que são calculados no código JavaScript.

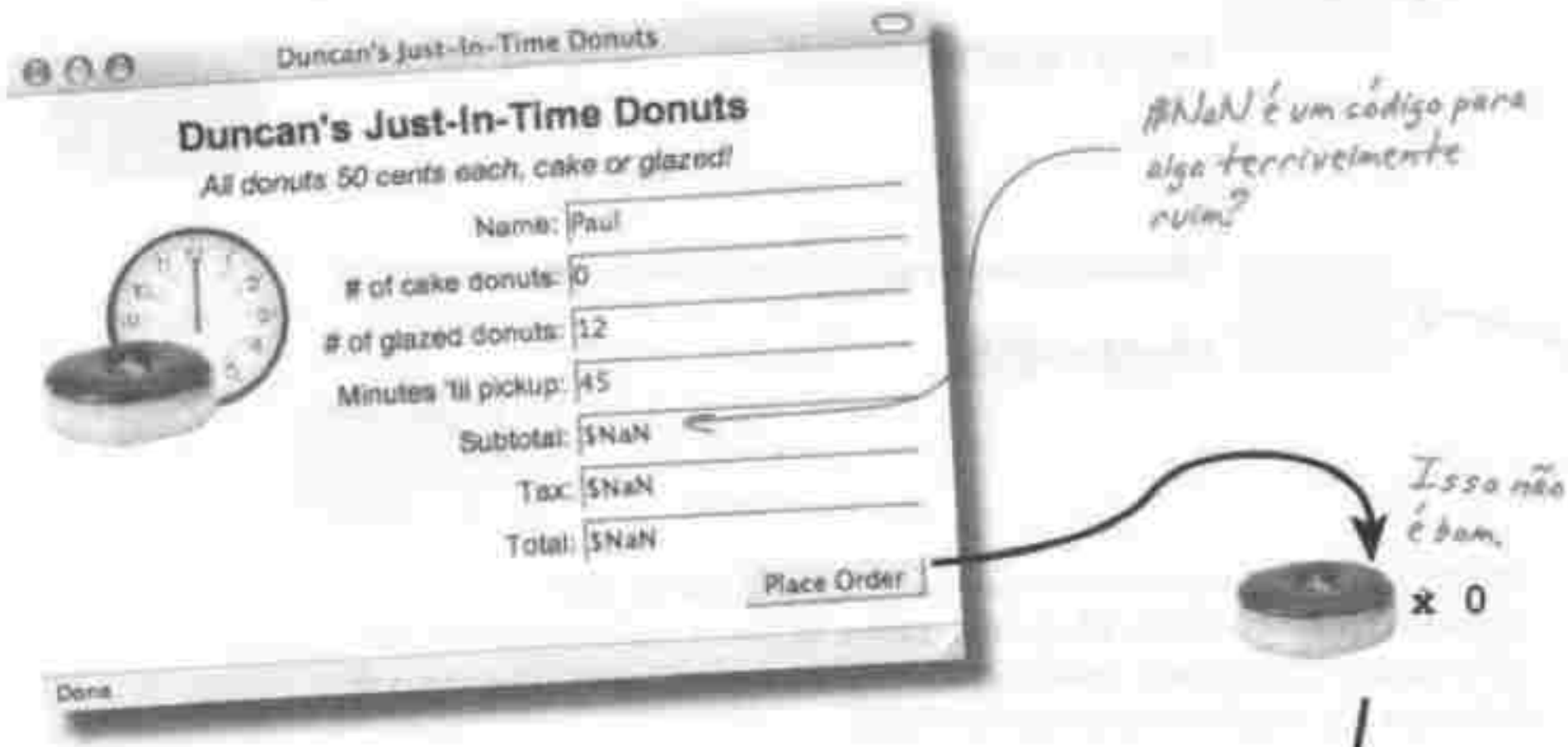


Poder do Cérebro

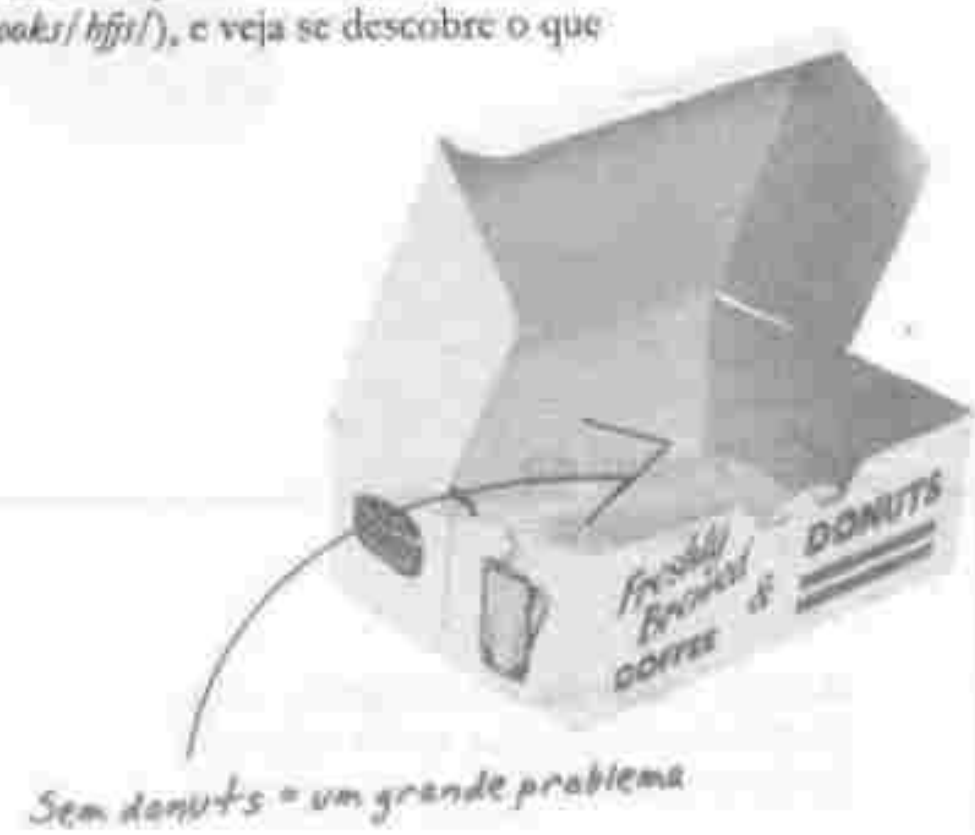
De que variáveis e constantes você precisará para realizar esses cálculos? Quais você nomearia?

Uma primeira olhada nos cálculos dos donuts

Duncan tentou escrever JavaScript para os cálculos, mas se defrontou com problemas. Tão logo o usuário digita uma quantidade de donuts, os cálculos dinâmicos perdem imediatamente o controle. Eles sugerem valores de $NaN que não fazem muito sentido. E o pior, os pedidos não são realizados e os clientes não estão exatamente vibrando com os "avanços" tecnológicos de Duncan.



É hora de analisar o código de script do donut e ver exatamente o que está acontecendo. Examine a próxima página (ou os exemplos de código que você pode baixar em *http://www.headfirstlabs.com/books/hfjs/*), e veja se descobre o que aconteceu.



Esse código é chamado para atualizar o pedido, calculando dinamicamente o subtotal e o total.

Já que os dados digitados pelo usuário pareceem estar certos, deve haver algo errado com as constantes.

Esse código envia o pedido para o servidor e confirma-o com o usuário.

O pedido é atualizado quando muda o número de donuts.

O pedido é enviado quando o botão Place Order (Faça seu pedido) é clicado.

```
<html>
  <head>
    <title>Duncan's Just-In-Time Donuts</title>
    <link rel="stylesheet" type="text/css" href="donuts.css" />
    <script type="text/javascript">
      function updateOrder() {
        const TAXRATE;
        const DONUTPRICE;
        var numCakeDonuts = document.getElementById("cakedonuts").value;
        var numGlazedDonuts = document.getElementById("glazeddonuts").value;
        var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
        var tax = subTotal * TAXRATE;
        var total = subTotal + tax;
        document.getElementById("subtotal").value = "$" + subTotal.
toFixed(2);
        document.getElementById("tax").value = "$" + tax.toFixed(2);
        document.getElementById("total").value = "$" + total.toFixed(2);
      }
      function placeOrder() {
        // Submit order to server...
        form.submit();
      }
    </script>
  </head>
  <body>
    <div id="frame">
      ...
      <form name="orderform" action="donuts.php" method="POST">
        ...
        <div class="field">
        # of cake donuts: <input type="text" id="cakedonuts"
name="cakedonuts"
          value="" onchange="updateOrder();" />
        </div>
        <div class="field">
        # of glazed donuts: <input type="text" id="glazeddonuts"
          name="glazeddonuts" value="" onchange="updateOrder();" />
        </div>
        ...
        <div class="field">
        <input type="button" value="Place Order"
        onclick="placeOrder(this.form);" />
        </div>
      </form>
    </div>
  </body>
</html>
```

Aponte seu lápis

Escreva o que você acha que aconteceu de errado com o código de script do Duncan's Donut com hora marcada.

.....

.....

Aponte seu lápis

Solução

Escreva o que você acha que aconteceu de errado com o código de script do Duncan's Donut com hora marcada.

As duas constantes, TAXRATE e DONUTPRICE, não foram inicializadas, o que significa que os cálculos que dependem delas não puderam ser concluídos.

Tudo bem, entendo que uma constante possui sempre o mesmo valor, mas se é esse o caso, então como ela pode ser inicializada?

Você não deve jamais deixar de inicializar uma constante. Você pode deixar de inicializar uma constante sem nunca dar um valor a ela, mas **não é uma idéia muito boa**. Quando você não inicializa uma constante ao criá-la, essa constante acaba na terra de ninguém – não possui nenhum valor e, pior, **nenhum valor pode ser dado a ela**. Uma constante não inicializada é essencialmente um **erro de codificação**, embora os navegadores geralmente não o informe.

Inicialize sempre as constantes ao criá-las.

Inicialize seus dados... ou senão

Quando você não inicializa um fragmento de dados, ele é considerado *indefinido* (undefined), o que é uma forma elegante de dizer que ele não possui valor. Isso não significa que ele não vale nada, apenas significa que ele não contém nenhuma informação... ainda. O problema aparece quando você tenta usar variáveis ou constantes que não tenham sido inicializadas.

Não inicializada

Em JavaScript, você multiplica os números usando * em vez de x

Inicializada

```
const DONUTPRICE;
var numCakeDonuts = 0;
var numGlazedDonuts = 12;
var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
```

0 12 ?

subtotal = (0 + 12) * 0

Isso é um grande problema.

A constante DONUTPRICE **não é inicializada**, o que significa que **não possui valor**. Na verdade, o JavaScript possui um valor especial só para este estado de "não-valor": undefined. É mais ou menos como se a sua secretária eletrônica informasse: "nenhuma mensagem", quando você não tivesse nenhuma mensagem – "nenhuma mensagem" tecnicamente ainda é uma mensagem, mas sua finalidade é representar a **falta de mensagens**. A mesma coisa é undefined – indica **falta de dados**.

Um fragmento de dados é indefinido quando ele não possui valor.

Você não tem mensagens.

Não há dados aqui, portanto é indefinido.

DONUTPRICE

NaN NÃO é um número

Assim como undefined representa uma **condição** especial de dados, existe um outro valor importante usado para indicar um caso especial de variáveis JavaScript: NaN. NaN significa **Not a Number**, e é o que a variável subTotal é atribuída, já que não há informações suficientes para realizar o cálculo. Em outras palavras, você tratou um valor inexistente como um número... e obteve NaN.

Um número

Não é um número!

```
subtotal = (0 + 12) * ? = NaN
```

Já que essa informação está indefinida, o cálculo não pode ser realizado.

NaN é um valor que não é um número, mesmo que você esteja esperando que o valor seja um número.

Então, resolver o problema do NaN requer a inicialização da constante DONUTPRICE ao criá-la:

```
const DONUTPRICE = 0.50;
```

Não existem perguntas idiotas

P: O que significa que os identificadores têm que ser específicos dentro de um script?

R: A questão toda dos identificadores é servir como um nome específico que pode ser usado para identificar um fragmento de informações em um script. No mundo real, é comum que as pessoas tenham o mesmo nome... porém, mais uma vez, as pessoas têm capacidade para lidar com esses "conflitos de nomes" e descobrir quem é quem. O JavaScript não é equipado para lidar com ambigüidade, então ele precisa que você diferencie cuidadosamente os diversos fragmentos de informações usando nomes distintos. Isso pode ser feito verificando se os identificadores dentro do seu código de script são específicos.

P: Cada identificador que eu criei tem que ser específico, ou específico apenas em um determinado script?

R: A singularidade do identificador só é realmente importante dentro de um **único** script e, em alguns casos, somente dentro de certas partes de um único script. Porém, lembre-se de que scripts para grandes aplicações Web podem se tornar muito grandes, dispersos por muitos arquivos. Neste caso, torna-se mais desafiante assegurar a singularidade entre todos os identificadores. A boa notícia é que não é terrivelmente difícil manter a singularidade do identificador em seus próprios scripts, contanto que você seja o mais descritivo possível ao nomeá-los.

P: Ainda não entendi bem quando usar letras maiúsculas (Camelcase) e quando usar letras minúsculas (camelCase). O que está acontecendo?

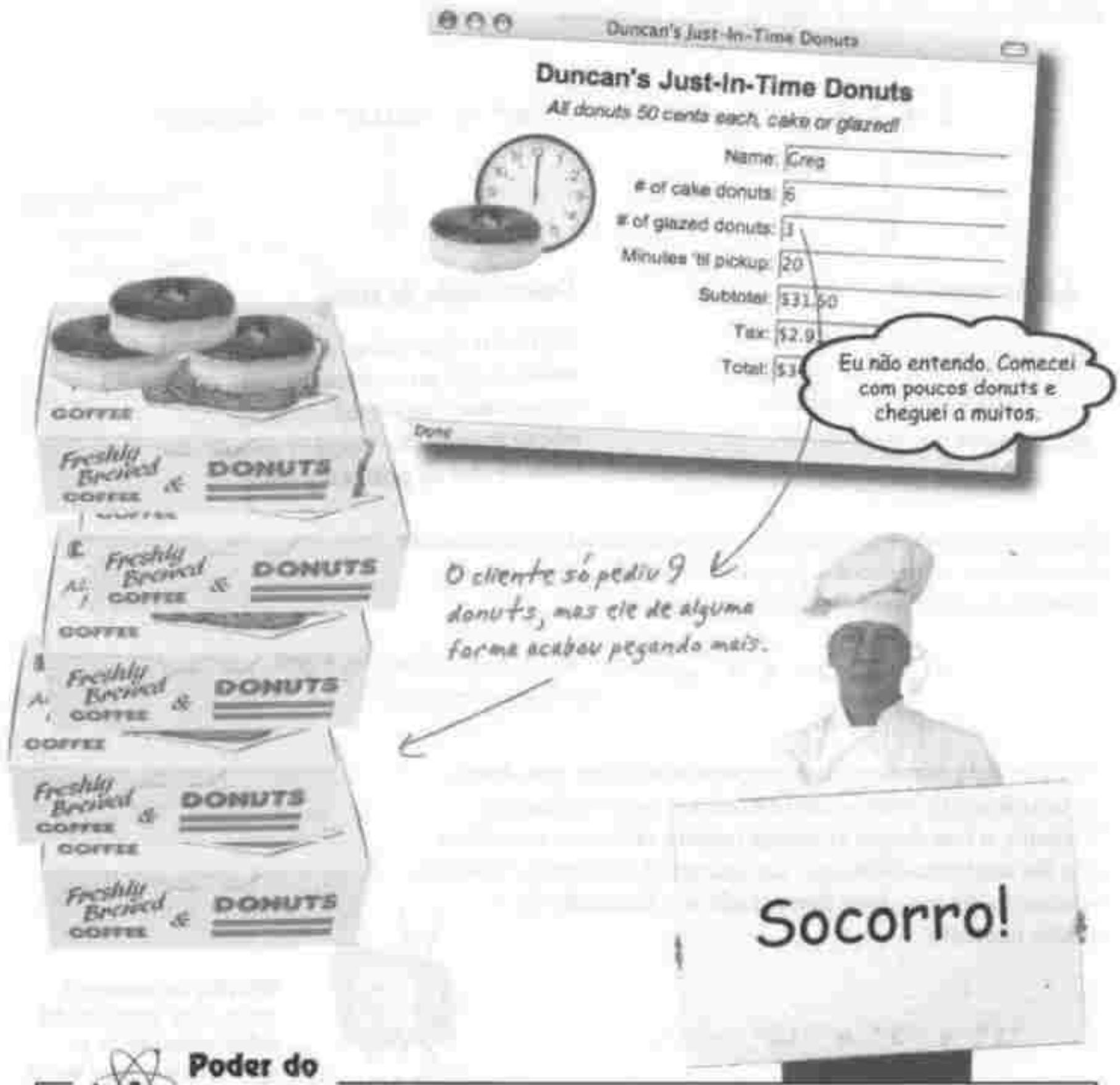
R: Camel case (com a primeira palavra em letra maiúscula) só se aplica à nomeação de objetos em JavaScript, sobre a qual conversaremos no Capítulo 9. Letras minúsculas se aplicam a variáveis e funções, e é igual a Camel case, exceto que a primeira letra do identificador é em letra minúscula. Então, camel case significa que você nomearia um objeto Donut, enquanto o uso de letras minúsculas (camelCase) significa que você nomearia uma função getDonut( ) e uma variável numDonuts. Não existe um nome bonitinho para constantes – eles são todos em letras maiúsculas.

P: Dados de texto e booleanos são considerados NaN?

R: Teoricamente sim, já que definitivamente eles não são números. Mas na realidade, não. A finalidade de NaN é indicar que um número não é o que você pensa que ele é. Em outras palavras, NaN não é bem uma descrição de dados JavaScript, como é um indicador de erro para tipos de dados de número. Normalmente, você só encontra NaN quando realiza cálculos que prevêem números, mas que por alguma razão são determinados dados não-numéricos para se trabalhar.

Enquanto isso, de volta ao Duncan's...

De volta ao Duncan's Donuts, as coisas têm ido de mal a pior. Em vez de caixas vazias, agora existem donuts por toda parte – de alguma forma, cada pedido está sendo super calculado. Duncan está chocado com as reclamações de sobrecarga de donuts e fraude de massa.



Poder do Cérebro

O que poderia ter dado errado em relação à manipulação da quantidade de informações sobre donuts?

Você pode adicionar mais do que número

Em JavaScript, **contexto** é tudo. Especificamente, é importante o **tipo** de dados que você está manipulando em um determinado fragmento de código, não apenas o que você está **fazendo** com os dados. Mesmo algo tão simples como adicionar dois fragmentos de informações pode produzir resultados muito diferentes dependendo do **tipo** de dados envolvido.

```
1 + 2 = 3
```

Adição numérica

dicionar dois números realiza o que provavelmente você espera – produz o resultado da adição **matemática** dos dois valores.

```
"do" + "nuts" = "donuts"
```

Palavra elegante para manter essas coisas unidas.

Concatenação de string

Adicionar duas strings também realiza o que provavelmente você espera, mas é muito diferente da adição matemática – aqui as strings estão ligadas de **ponta a ponta**.

Sabendo que essas strings de texto são adicionadas diferentemente dos números, o que você acha que acontece quando se faz uma tentativa de adicionar dois números textuais?

```
"1" + "2" = ?
```

Adição, concatenação, o que está acontecendo?

O JavaScript realmente não se importa com o que está dentro de uma string de texto – tudo é caractere para o JavaScript. Portanto, o fato de que as strings contêm caracteres numéricos não faz nenhuma diferença... a concatenação de string é realizada, resultando em um efeito **inesperado** se a **intenção** era a adição numérica.

```
"1" + "2" = "12"
```

Como são strings e não números, eles são adicionados com o uso da concatenação de string.

O resultado é uma string que não se parece nem um pouco com a adição matemática.

PRESTE ATENÇÃO!

Verifique sempre se o que você está adicionando é o que pensa que está adicionando.

Concatenar strings acidentalmente, quando você tem intenção de adicionar números, é um erro comum em JavaScript. Certifique-se de converter strings em números antes de adicioná-las, se a sua intenção for adição numérica.

parseInt() e parseFloat(): convertem texto em número

Apesar do problema da adição/concatenação, existem situações legítimas onde você precisa realizar uma operação matemática com um número que armazenou como string. Nesses casos, você precisa **converter** a string em um número **antes** de realizar qualquer operação numérica com ela. O JavaScript oferece duas funções úteis para aplicar este tipo de conversão:

`parseInt()`

Fornece uma string para essa função e ela converte a string em um inteiro.

`parseFloat()`

Fornece uma string para essa função e ela converte a string em um número (decimal) de ponto flutuante.

Cada uma dessas funções embutidas aceita uma string e retorna um número, após concluir a conversão:

parseInt() transforma "1" em 1.

```
parseInt("1") + parseInt("2") = 3
```

Dessa vez, o resultado é a adição matemática de 1 e 2.

A string "2" é convertida no número 2.

Lembre-se de que as funções parseInt() e parseFloat() **não garantem** que vão funcionar sempre. Elas são tão úteis quanto as informações que você fornece a elas. Elas se esforçarão ao máximo para converter as strings em números, mas a idéia é que você deveria fornecer strings que **só** contivessem caracteres numéricos.

```
parseFloat("$31.50") = NaN
```

Esse código é um problema, porque o caractere $ confunde a função.

Surpresa, surpresa! O resultado não é um número (Not a Number).

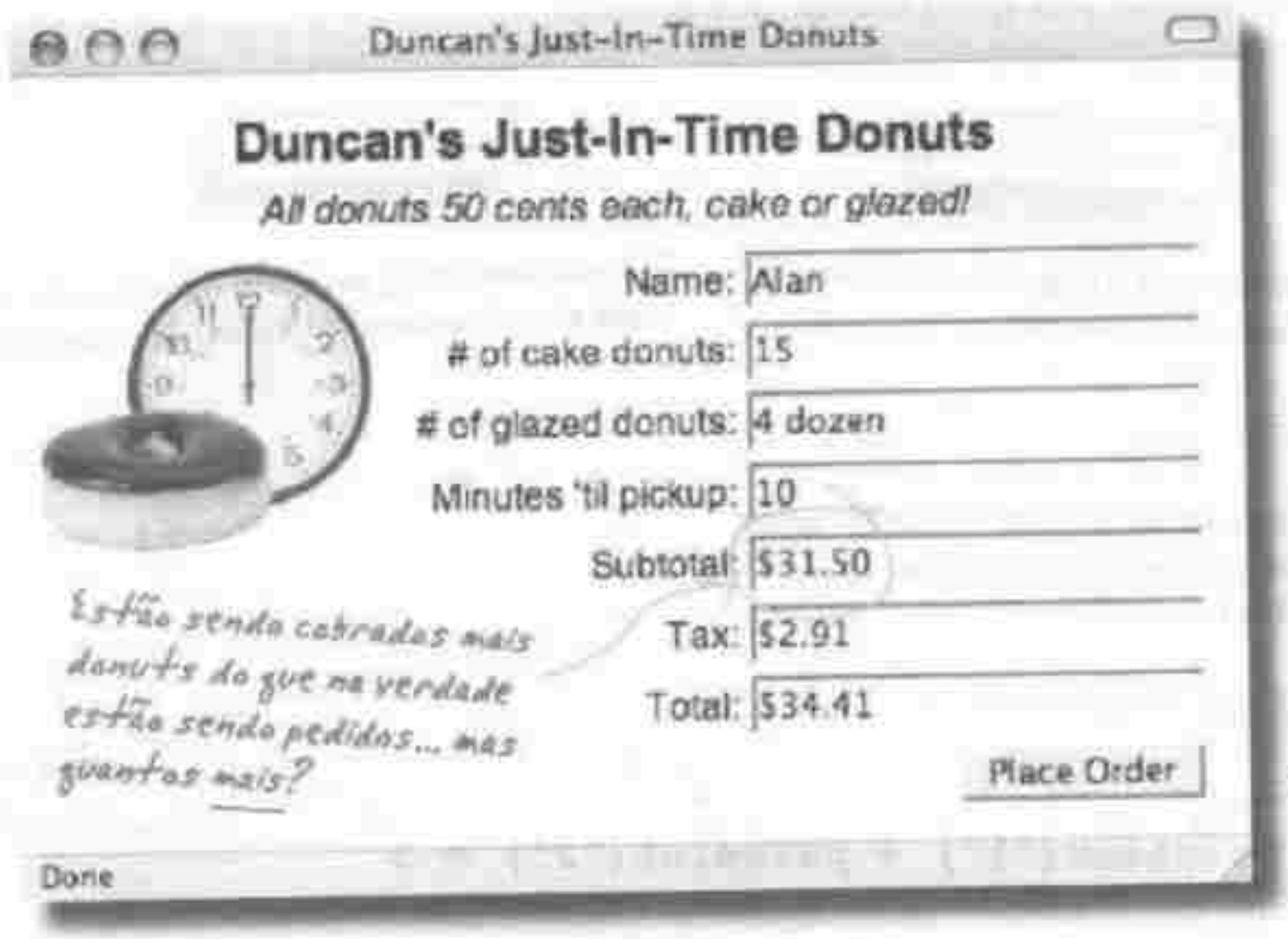
Relaxe

Não se preocupe se essa coisa de função ainda é um pouco confusa.

Você conhecerá os verdadeiros fatos sobre as funções mais adiante – por ora, tudo o que realmente precisa saber é que as funções permitem que você passe informações para elas e que depois devolvem algo em troca.

Por que donuts extras estão sendo pedidos?

Examinemos cuidadosamente o formulário de pedido de donuts com hora marcada. Somos capazes de descobrir o porquê de tantos donuts serem acidentalmente encomendados...



Podemos dividir o subtotal pelo preço de cada donut... e a resposta é quantos donuts foram pedidos.

O subtotal do pedido.

```
$31,50 / $0,50 = 63 donuts
```

O preço por donut.

O número total de donuts realmente pedidos... hmmm.

Lembra "1" + "2" = "12"? Parece semelhante, não?

Isto se parece muito com o problema da adição de string numérica, especialmente se levar em consideração aqueles dados de formulário que são sempre armazenados como strings, independentemente do que seja. Embora os números sejam digitados nos campos do formulário, do ponto de vista do JavaScript, eles realmente são apenas texto. Assim, só precisamos converter as strings em números reais para evitar que uma adição numérica seja mal interpretada como uma concatenação de string.

Aponte seu lápis

Usando os fragmentos de dados abaixo para extrair os conteúdos dos campos de quantidade de donuts do formulário, escreva as linhas de código ausentes na função updateOrder() do Duncan, de modo que as quantidades de donut sejam convertidas de strings para números.

```
document.getElementById("cakedonuts").value
```

Esse código extrai o número de donuts com cobertura digitado no formulário.

Esse código obtém o número de donuts simples digitado pelo usuário no formulário.

```
document.getElementById("glazeddonuts").value
```

```
function updateOrder() {
  const TAXRATE = 0.0925;
  const DONUTPRICE = 0.50;
  var numCakeDonuts =


  var numGlazedDonuts =


  if (isNaN(numCakeDonuts))
    numCakeDonuts = 0;
  if (isNaN(numGlazedDonuts))
    numGlazedDonuts = 0;
  var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
  var tax = subTotal * TAXRATE;
  var total = subTotal + tax;
  document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
  document.getElementById("tax").value = "$" + tax.toFixed(2);
  document.getElementById("total").value = "$" + total.toFixed(2);
}
```

Aponte seu lápis
Solução

Usando os fragmentos de dados abaixo para extrair os conteúdos dos campos de quantidade de donuts do formulário, escreva as linhas de código ausentes na função updateOrder() do Duncan, de modo que as quantidades de donut sejam convertidas de strings para números.

```
document.getElementById("cakedonuts").value
```

```
document.getElementById("glazeddonuts").value
```

Já que ambos os números são inteiros, parseInt() é usada na conversão.

```
function updateOrder() {
  const TAXRATE = 0.0925;
  const DONUTPRICE = 0.50;
  var numCakeDonuts =
    parseInt(document.getElementById("cakedonuts").value);
  var numGlazedDonuts =
    parseInt(document.getElementById("glazeddonuts").value);
  if (isNaN(numCakeDonuts))
    numCakeDonuts = 0;
  if (isNaN(numGlazedDonuts))
    numGlazedDonuts = 0;
  var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
  var tax = subTotal * TAXRATE;
  var total = subTotal + tax;
  document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
  document.getElementById("tax").value = "$" + tax.toFixed(2);
  document.getElementById("total").value = "$" + total.toFixed(2);
}
```

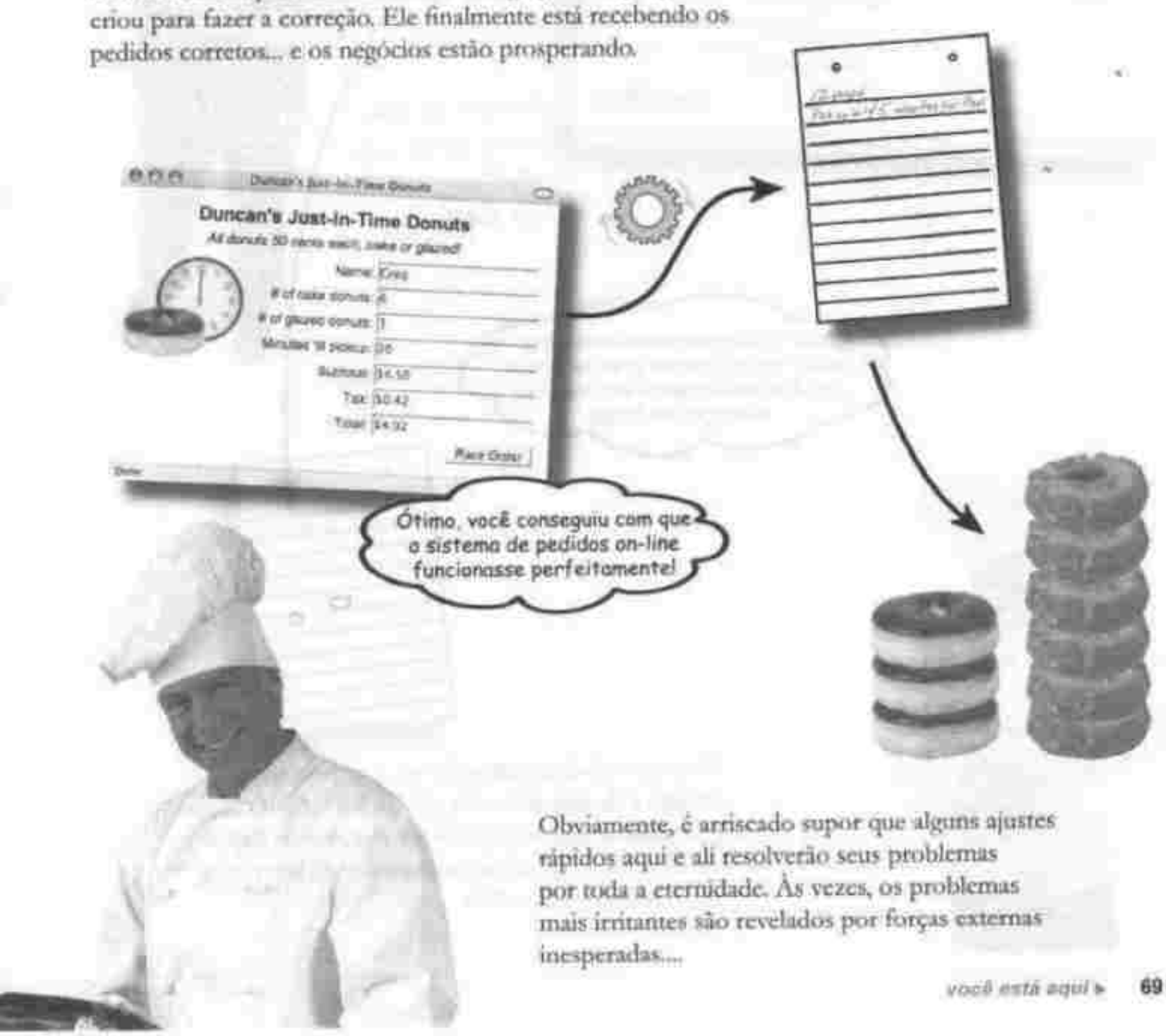
A função toFixed() arredonda os valores em dólar para duas casas decimais.

Pontos de Bala

- Embora não seja uma condição expressa do JavaScript, nomear as **constantes** com **LETRAS MAIÚSCULAS** e as **variáveis** em **letras minúsculas (camelCase)** é uma boa convenção de codificação.
- Inicialize sempre as constantes ao criá-las, e inicialize as variáveis sempre que for possível.
- Se uma variável não for inicializada, ela permanece **indefinida** (undefined) até que em algum momento seja atribuído um valor a ela.
- NaN significa **Not a Number** (não é um número), e é usado para indicar que um fragmento de dados não é um número, quando o que se espera é que ele seja.
- A concatenação de string é muito diferente da adição matemática, embora ambas usem o conhecido sinal de adição (+).
- As funções embutidas parseInt() e parseFloat() são usadas para converter strings em números.

Você descobrirá o problema...

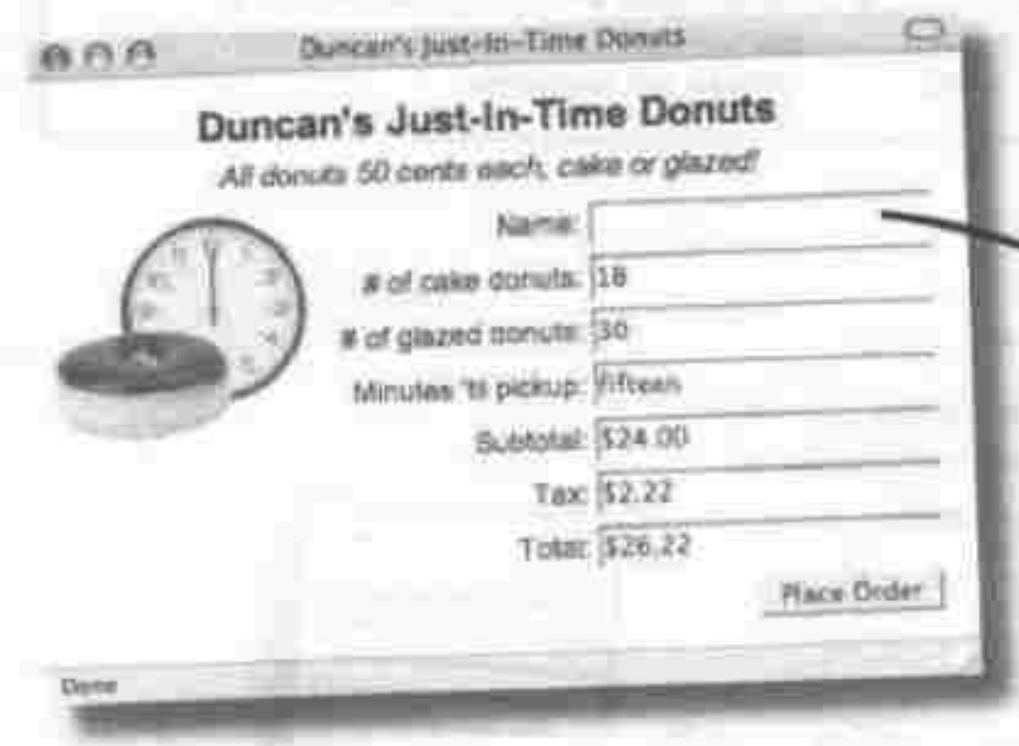
Duncan está impressionado com o código JavaScript que você criou para fazer a correção. Ele finalmente está recebendo os pedidos corretos... e os negócios estão prosperando.



Obviamente, é arriscado supor que alguns ajustes rápidos aqui e ali resolverão seus problemas por toda a eternidade. Às vezes, os problemas mais irritantes são revelados por forças externas inesperadas...

Duncan descobre espionagem de donuts

Duncan adquiriu um novo problema: um concorrente astucioso chamado Frankie. Frankie dirige um comércio de hotdog (cachorro-quente) do outro lado da rua, e que agora está oferecendo café da manhã. O problema é que Frankie está jogando sujo e enviando pedidos falsos de donut sem nomes. Então agora nós temos pedidos sem clientes – e isso não é bom.



Embora nenhum nome tenha sido digitado, o pedido foi aceito.

Não estou preocupado com meus concorrentes, apenas preciso tornar o código mais esperto em relação à aceitação de dados.

Duncan está desperdiçando seu precioso tempo, energia e donuts preenchendo pedidos falsificados... e ele precisa que você verifique se todos os dados do formulário foram digitados antes de permitir a aprovação de um pedido.

Use getElementById() para extrair dados do formulário

Para verificar a validade dos dados de formulário, você precisa de uma forma para extraí-los da sua página da Web. A chave para acessar um elemento de página da Web com JavaScript é o atributo id da tag HTML:

```
<input type="text" id="cakedonuts" name="cakedonuts" />
```

Elemento de informação HTML da quantidade de donuts simples.

O atributo id é o que você usa para acessar o campo de formulário em código JavaScript.

# of cake donuts: 18

O JavaScript permite recuperar um elemento de página da Web com seu ID usando uma função chamada getElementById(). Essa função não extrai diretamente os dados de um elemento, mas em vez disso fornece o próprio campo HTML como um objeto JavaScript. Você então acessa os dados através da propriedade value do campo.

Tecnicamente, getElementById() é um método no objeto document e não uma função.

```
document.getElementById()
```

Forneça o ID de um elemento em uma página da Web a esse método e ele lhe dará o mesmo elemento de volta, que pode então ser usado para acessar os dados da Web.

O método getElementById() pertence ao objeto document.

Relaxe

Não molhe objetos, propriedades e métodos por ora.

O JavaScript oferece suporte a um tipo de dados avançado chamado objeto que permite fazer algumas coisas realmente legais. Na verdade, a própria linguagem JavaScript é apenas um punhado de objetos. Conversaremos mais sobre objetos ainda neste livro – por ora, saiba que um método é muito parecido com uma função e uma propriedade é muito parecida com uma variável.

```
document.getElementById("cakedonuts")
```

O ID é a chave para acessar um elemento.

```
document.getElementById("cakedonuts").value
```

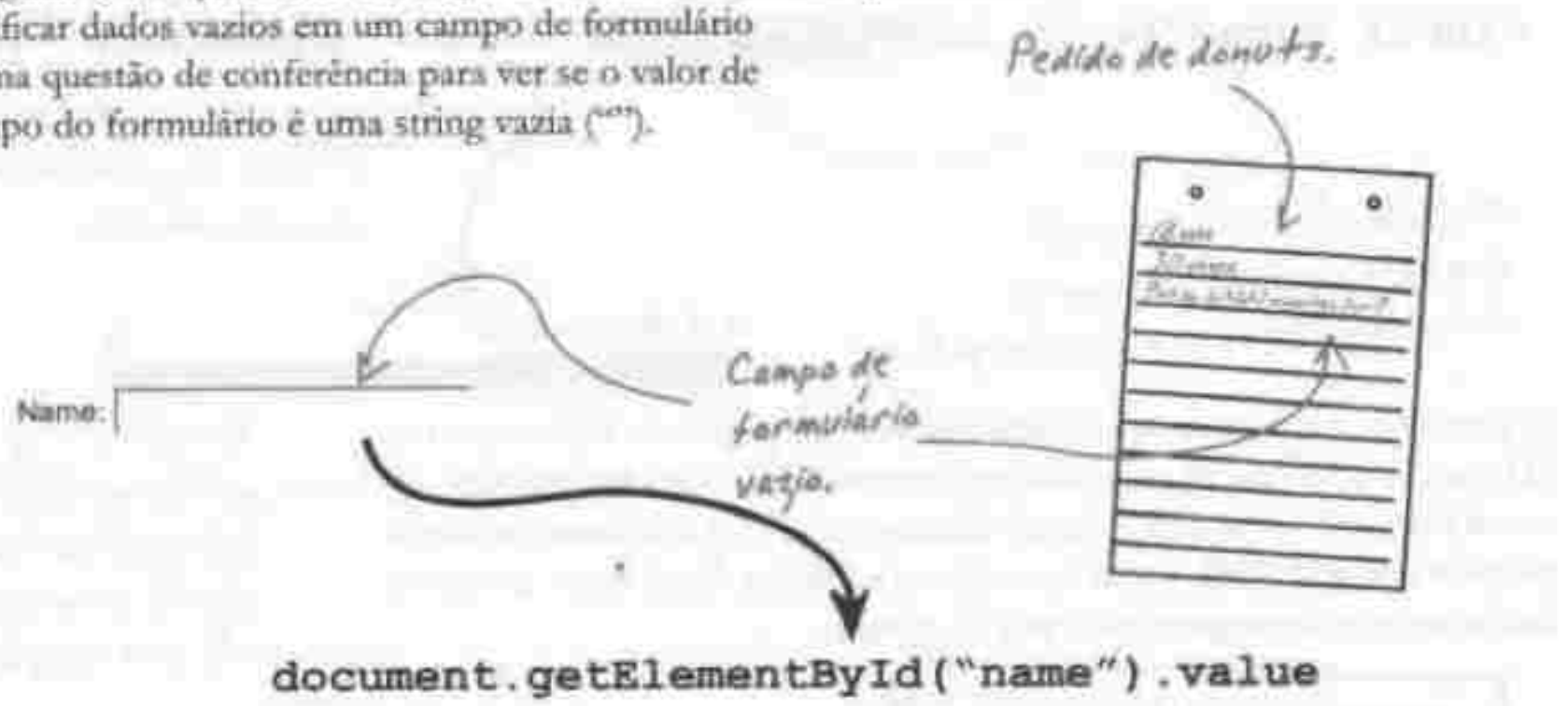
A propriedade do valor lhe dá acesso aos dados.

# of cake donuts: 15

Com este código em mãos, agora você está pronto para verificar os dados do formulário do Duncan para certificar-se de que os campos não estão vazios, antes de aceitar um pedido.

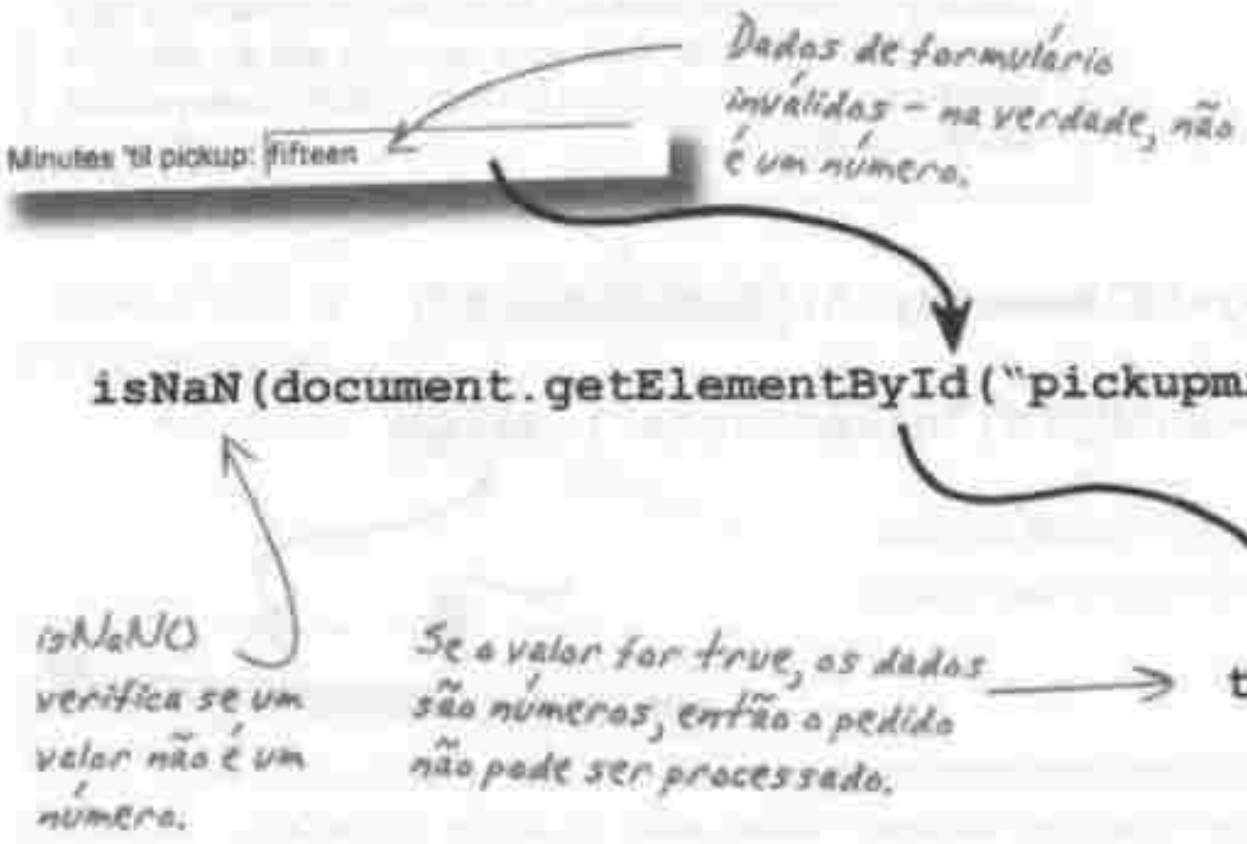
Valide os dados do formulário Web

Você precisa conferir para ter certeza de que um nome foi digitado no formulário. Não digitar o número de minutos até a coleta também pode ser um problema, já que a idéia central é oferecer donuts quentes com hora marcada. Então, você precisa assegurar-se de que ambos os fragmentos de dados foram preenchidos e validados. Verificar dados vazios em um campo de formulário é uma questão de conferência para ver se o valor de campo do formulário é uma string vazia ("").



Se o valor de campo do nome for uma string vazia, então você sabe que o pedido precisa ser interrompido e o usuário deve ser solicitado a digitar o nome dele. A mesma coisa acontece com o campo de minutos, exceto se também for útil dar um passo adiante e procurar ver se os dados naquele campo são números. A função isNaN() embutida é o que torna essa verificação possível – você passa um valor para ela e ela diz se o valor não é um número (true) ou se é um número (false).

Uma string vazia é um indício de que o campo do formulário não possui dados.



Imãs de geladeira JavaScript

A função placeOrder() é onde ocorre a validação das informações de nome e dos minutos para coleta. Use os imãs para terminar de escrever o código que verifica a existência das informações do nome e dos minutos para coleta, e verifique se os minutos para coleta digitados são números. Você precisará usar cada imã e alguns imãs mais de uma vez.

"if" é usado para testar uma condição e depois realizar uma ação de maneira apropriada – se isso, então faça alguma coisa.

Esse é um teste de igualdade – uma coisa é igual à outra?

Isso significa que uma de duas condições pode resultar na ação – se isso OU aquilo, então faça alguma coisa.

```
function placeOrder() {

  if (..................................................................... "".....)
    alert("Você precisa fornecer um nome antes de enviar o pedido.");

  else if (..................................................................... .....||

                                                                                  .....)
    .....................................................................................
    alert("Você deve fornecer o número de minutos até a coleta" +
      "antes de enviar o pedido.");
  else
    // Envie o pedido para o servidor
    form.submit();
}
```

"pickupminutes"

"name"

.

)

document

(

" "

isNaN

value

getElementById

Solução dos Imãs de geladeira Javascript

A função placeOrder() é onde ocorre a validação das informações de nome e dos minutos para coleta. Use os imãs para terminar de escrever o código que verifica a existência das informações de nome e dos minutos para coleta, e verifique se os minutos para coleta digitados são números. Todos os imãs são usados e alguns são usados várias vezes.

Isso diz: se o valor do nome estiver vazio, então dispare um alerta... senão, faça alguma coisa diferente.

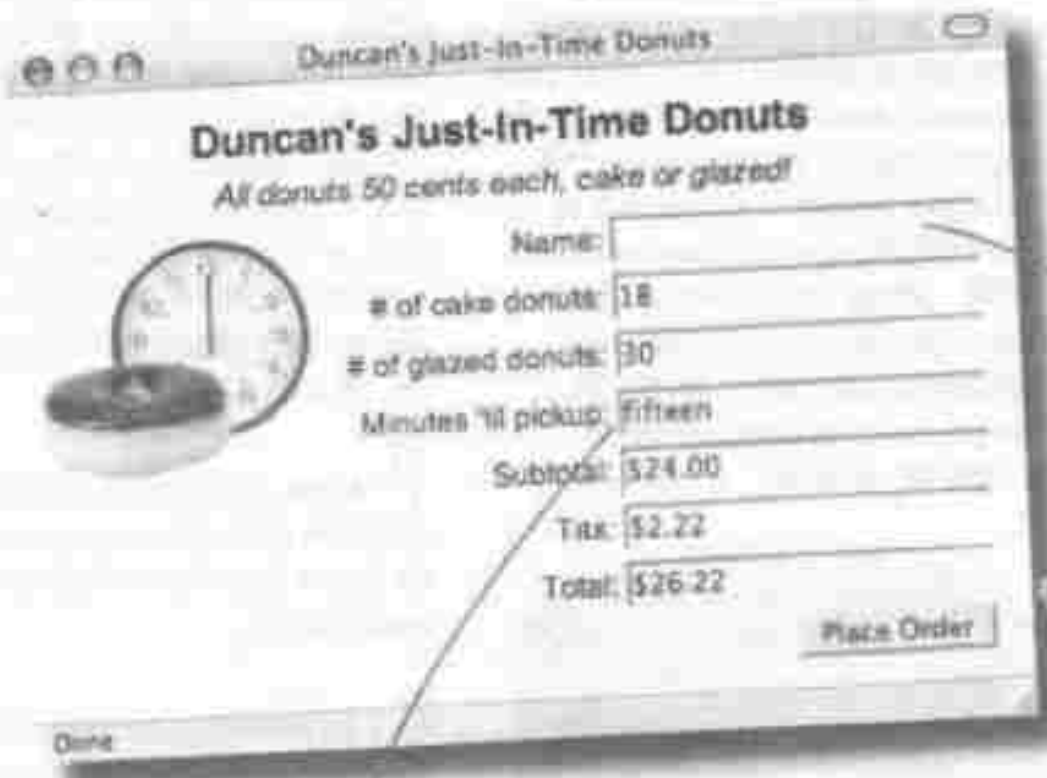
Isso verifica se o valor do campo de nome é igual a.

Aqui, estamos dizendo: se o valor estiver vazio, OU se o valor não é um número.

```
function placeOrder() {
  if (document.getElementById("name").value == "")
    alert("Você precisa fornecer um nome antes de enviar o pedido.");
  else if (document.getElementById("pickupminutes").value == "" ||
           isNaN(document.getElementById("pickupminutes").value)
    alert("Você deve fornecer o número de minutos até a coleta" +
      "antes de enviar o pedido.");
  else
    // Envie o pedido para o servidor
    form.submit();
}
```

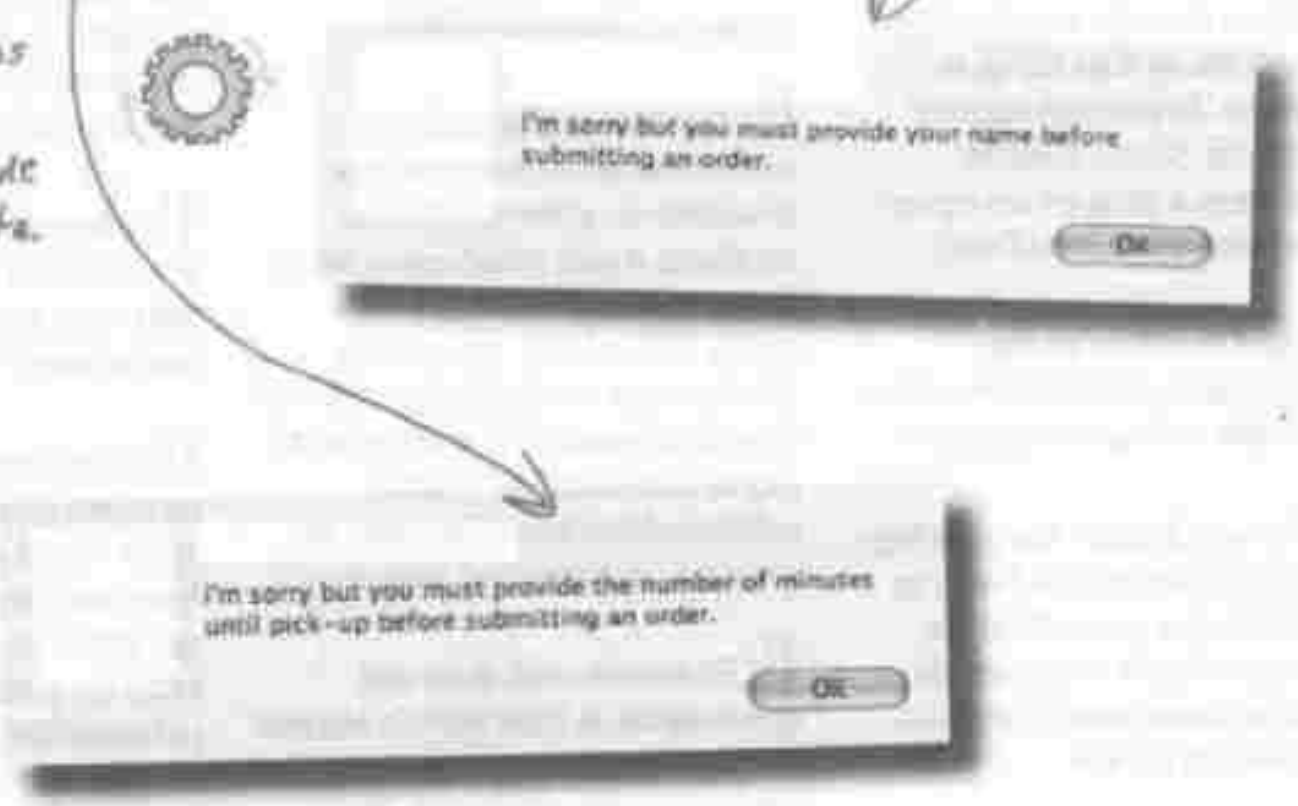
Você salvou o Duncan's Donuts... mais uma vez!

O novo e aperfeiçoado formulário de donuts na hora marcada com validação de dados colocou um fim na espionagem de Frankie, e também tornou a página mais robusta para os clientes reais. Usar JavaScript para proteger a integridade dos dados digitados pelo usuário é um ganho mútuo, especialmente no implacável setor de café da manhã!



Agora, deixar o campo do nome vazio tem como resultado um aviso, ao invés de permitir que o pedido seja aprovado.

Dados não-numéricos não são mais um problema no campo de minutos para coleta.



Não existem perguntas idiotas

P: Como o sinal de adição (+) sabe que é para adicionar ou concatenar?

R: Como muitas coisas no JavaScript, a funcionalidade é determinada pelo contexto. Isso significa que o sinal de adição examina duas coisas "adicionadas" e decide se adiciona ou as concatena numericamente como texto, com base em seus tipos de dados. Você já sabe que "adicionar" duas palavras significa uni-las de ponta a ponta. Mas problemas podem ocorrer ao supor erroneamente que você está trabalhando com um tipo de dados, quando na verdade é um outro. Esse é um dos motivos pelo qual é sempre uma boa idéia conferir para certificar-se de que você fornece dados numéricos ao planejar uma adição numérica, e texto por texto.

P: O que acontece se você tentar adicionar uma string a um número?

R: Como no JavaScript, a conversão de número para string é automática. A mistura dos dois tipos de dados em uma adição resulta sempre em uma concatenação de string. Então, primeiro o número se converte em uma string e, em seguida, as duas strings se concatenam. Se pretende adicionar dois números, precisa converter explicitamente a string em um número usando parseInt() ou parseFloat().

P: O que acontece se usar parseInt() **para converter uma string que contém um número decimal?**

R: Não se preocupe, não pega fogo. Tudo isso que acontece é devido ao JavaScript supor que você não se importa com a parte fracionária do número, então ele retorna apenas a parte inteira do número.

P: Como o atributo HTML id **liga os elementos Web ao código JavaScript?**

R: Pense no atributo id como um portal através do qual o código JavaScript acessa o conteúdo HTML. Quando as pessoas dizem que o código JavaScript é executado em uma página da Web, eles não querem dizer literalmente a página da Web em si – eles querem dizer o navegador. Na realidade, o código JavaScript está razoavelmente protegido do código HTML, e só se pode acessá-lo através de mecanismos muito específicos. Um desses mecanismos envolve o atributo id, que permite ao JavaScript recuperar um elemento HTML. Marcar um elemento Web com um ID permite que o elemento seja encontrado pelo código JavaScript, abrindo todas as possibilidades de se programar um script.

P: Isso é muito vago. Como, especificamente, o código JavaScript acessa um elemento HTML?

R: O método getElementById() do objeto document é a chave para se acessar um elemento HTML do JavaScript, e esse método usa o atributo id do elemento para encontrá-lo na página. Os IDs HTML são como os identificadores JavaScript que devem ser específicos dentro de uma determinada página. Caso contrário, o método getElementById()teria dificuldades em saber qual elemento Web retornar.

P: Eu sei que você disse que conversaremos mais sobre o assunto no Capítulo 9, mas os objetos já apareceram algumas vezes. Quais são?

R: Vamos nos adiantar aqui, mas não conte para ninguém. Objetos são um tipo de dados JavaScript avançado que pode combinar funções, constantes e variáveis em uma entidade lógica. Um método é apenas uma função que faz parte de um objeto, enquanto uma propriedade é uma variável ou constante em um objeto. Em nível prático, o JavaScript usa objetos para representar quase tudo – a janela do navegador é um objeto, como o documento de página da Web. Por isso, o método getElementById() deve ser chamado através do objeto document – ele é parte do objeto que representa a página da Web inteira. OK, de volta ao Capítulo 2...

P: Ainda não entendi a diferença entre um elemento de página da Web e seu valor. O que está acontecendo?

R: Os elementos de página da Web são expostos ao JavaScript como objetos, o que significa que eles possuem propriedades e métodos que você pode usar para manipulá-los. Uma dessas propriedades é value, que contém o valor armazenado no elemento. Como exemplo, o valor de um campo de formulário são os dados digitados dentro do campo.

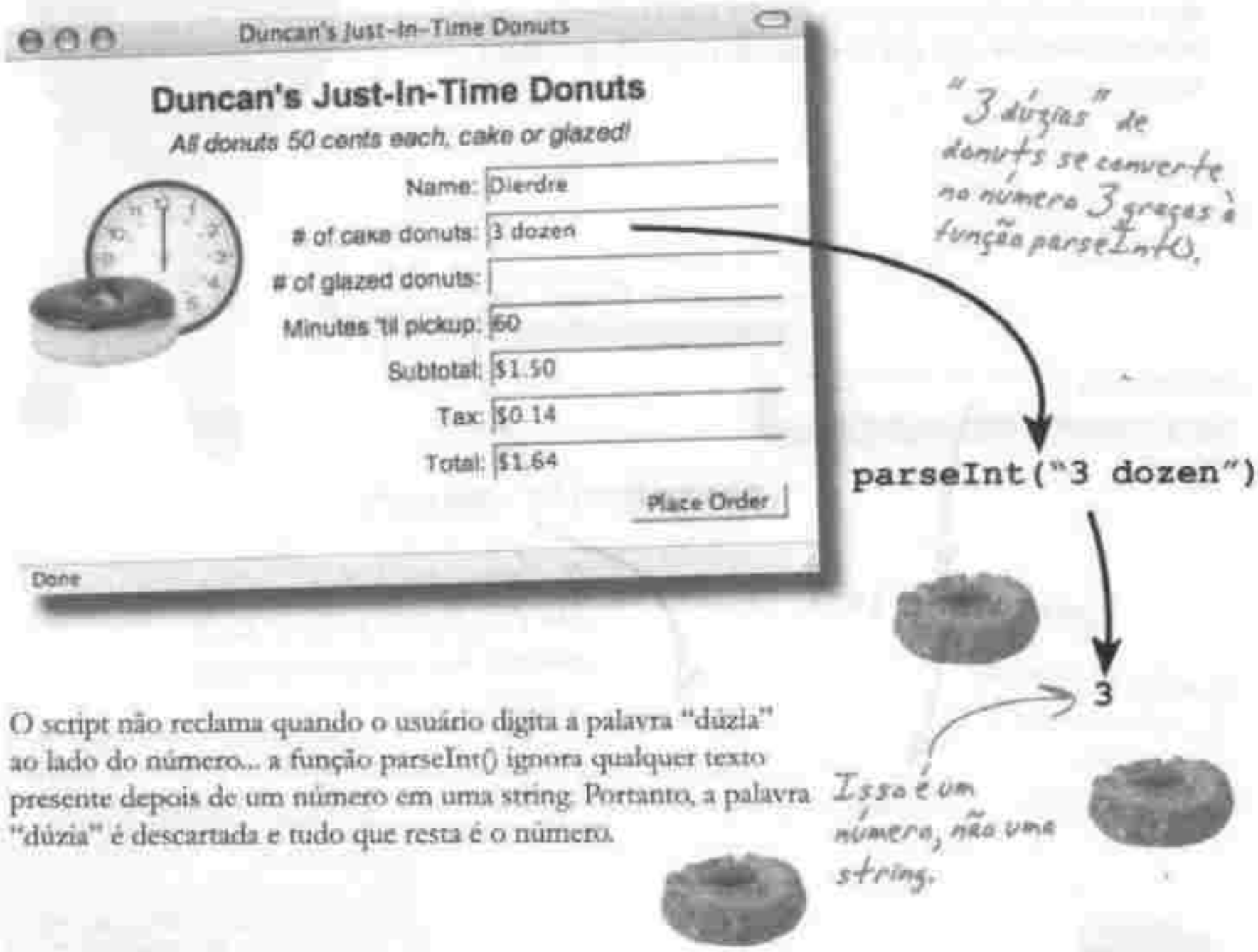
P: Por que é necessário saber se um valor *não* é um número? Não faria mais sentido verificar se ele *é* um número?

R: Boa pergunta. Isso se resume no motivo pelo qual você se preocupa se um valor é um número ou não. Em muitos casos, a hipótese é a de que esteja lidando com um número, então faz sentido verificar a exceção (o inesperado). Verificando NaN, você é capaz de tornar o código de script de tratamento de número mais robusto e, com sorte, minorar um cálculo estranho que envolve um não-número.

Empenhe-se pelas informações claras do usuário

Agora que o Duncan não está mais apagando fogo, ele realmente precisa melhorar a experiência do usuário com o formulário de donuts com hora marcada. Da mesma forma que o símbolo de "donuts quentes" é **intuitivo** para as pessoas que passam na frente da loja, é necessário que o formulário on-line também seja. Duncan sabe que normalmente os donuts são pedidos e servidos em dúzias. Poucas pessoas pedem 12 ou 24 donuts – elas pedem 1 ou 2 dúzias de donuts. Ele acha que o formulário de donuts deve permitir que os usuários digitem os dados da forma mais natural possível.

O problema é que o script atual não leva em consideração que o usuário digite a palavra "dúzia", ao especificar a quantidade de donuts.



O script não reclama quando o usuário digita a palavra "dúzia" ao lado do número... a função parseInt() ignora qualquer texto presente depois de um número em uma string. Portanto, a palavra "dúzia" é descartada e tudo que resta é o número.

Poder do Cérebro

Existe a possibilidade de o script do donut permitir que os usuários digitem um número ou um número e a palavra "dúzia" para pedir por dúzias? Como?

É possível pesquisar a palavra "dúzia" no texto digitado pelo usuário?

Se o usuário quiser uma "dúzia," multiplique por 12!
A opção "peça por dúzia" pode ser adicionada ao script do donut, ao verificar as informações do usuário para a palavra "dúzia" antes de calcular o subtotal. **Se** a palavra "dúzia" aparecer, multiplique o número por 12. Caso contrário, use o número no estado em que se encontra, já que ele se refere a cada donut.

# of cake donuts: 3 dozen

# of cake donuts: 18

```
parseInt("3 dozen")
```

O número digitado é multiplicado por 12 já que a palavra dúzia aparece nos dados informados.

```
parseInt("18")
```

O número digitado é o número exato de donuts encomendados.

```
3 * 12 = 36
```

```
18
```

JavaScript pronto para assar

A função personalizada parseDonuts() é responsável pelo processamento dos dados digitados referentes à quantidade de donuts. Ela, primeiro, converte os dados em número e, em seguida, verifica a aparição da palavra "dúzia" nos dados digitados. Se "dúzia" aparecer, o número de donuts é multiplicado por 12. Obtenha essa receita em *http://www.headfirstlabs.com/books/hfjs/*.

```
function parseDonuts(donutString) {
  numDonuts = parseInt(donutString);
  if (donutString.indexOf("dozen") != -1)
     numDonuts *= 12;
  return numDonuts;
}
```

Verifique se a palavra dúzia aparece nos dados digitados.

Multiplique o número de donuts por 12.

Como analisar dúzias de donuts

A função parseDonuts() é chamada na função updateOrder(), que é quando o subtotal e total são calculados a partir dos dados digitados pelo usuário.

Inicializa as duas constantes.

Pega o número de donuts do campo do formulário.

```
function updateOrder() {
  const TAXRATE = 0.0925;
  const DONUTPRICE = 0.50;
  var numCakeDonuts = parseDonuts(document.getElementById("cakedonuts").value);
  var numGlazedDonuts = parseDonuts(document.getElementById("glazeddonuts").value);
  if (isNaN(numCakeDonuts))
    numCakeDonuts = 0;
  if (isNaN(numGlazedDonuts))
    numGlazedDonuts = 0;
  var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
  var tax = subTotal * TAXRATE;
  var total = subTotal + tax;
  document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
  document.getElementById("tax").value = "$" + tax.toFixed(2);
  document.getElementById("total").value = "$" + total.toFixed(2);
}
```

Se o número de donuts digitados não for um número, defina-o como 0.

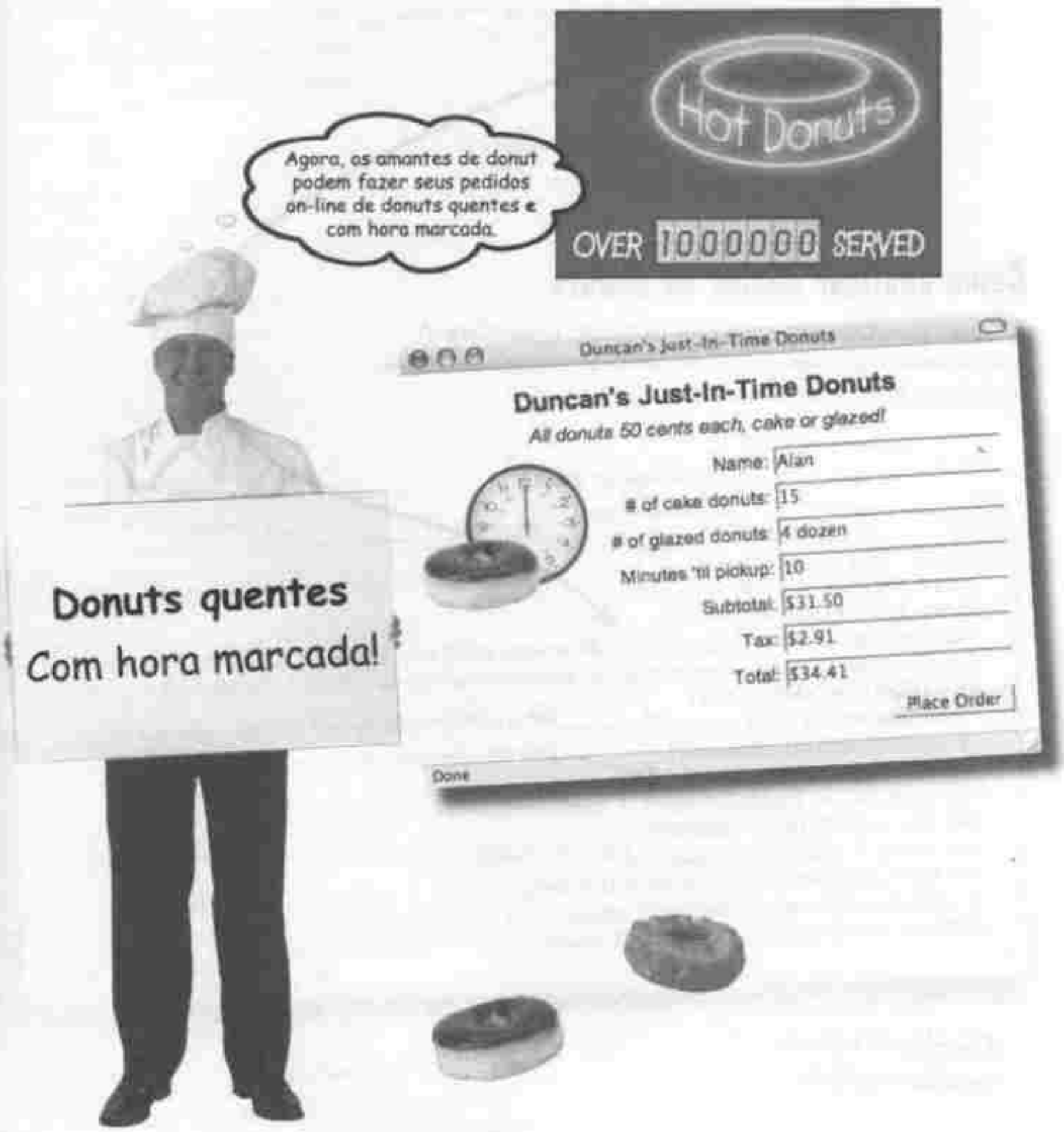
Calcula o subtotal, o imposto e o total.

Mostra os valores em dólar na página.

Arredonda os valores em dólar para duas casas decimais (centavos).

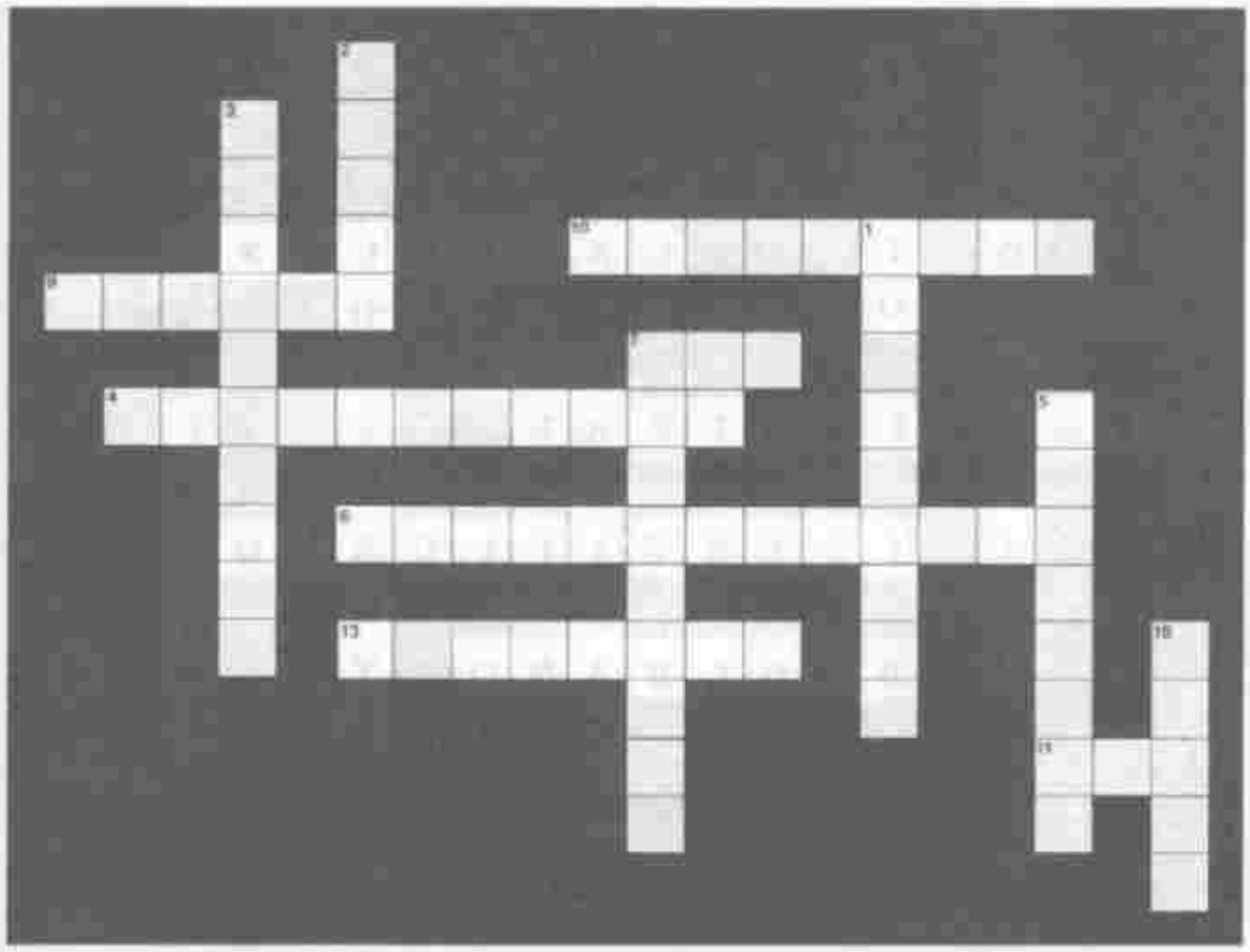
Donuts com hora marcada, um sucesso incrível!

A vida está ótima, agora que o Duncan e sua idéia de donut quente com hora marcada foi inteiramente concretizada em uma página acionada com JavaScript, que valida cuidadosamente os pedidos digitados pelo usuário.



Cruzadas do JavaScript

Nem sempre os dados são armazenados em código JavaScript. Às vezes eles ficam armazenados nas linhas horizontais e verticais das palavras-cruzadas, onde eles esperam pacientemente que você os revele.



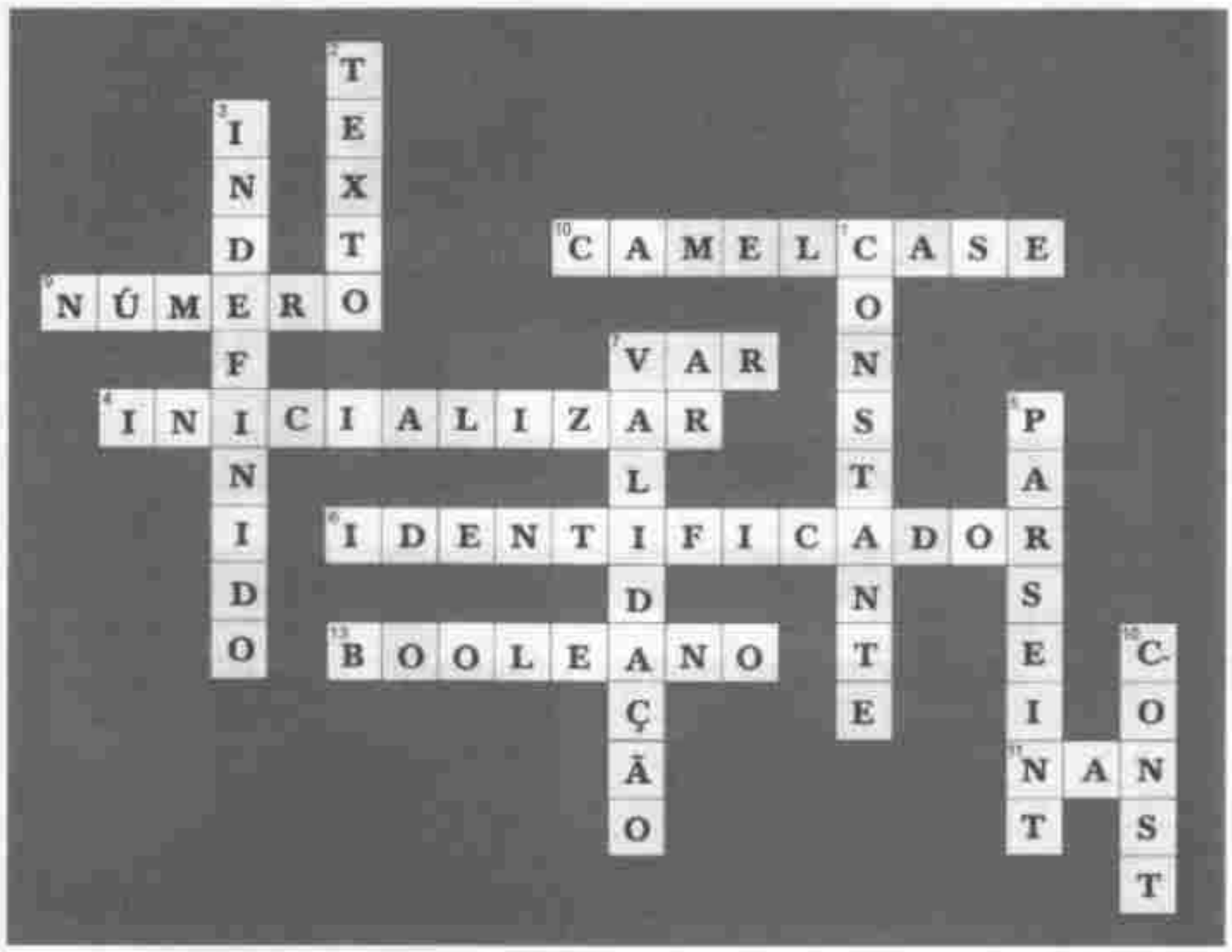
Horizontal

4. Quando você define o valor de um fragmento de dados ao criá-lo, você...
6. Nome específico usado para referenciar um fragmento de dados.
7. A palavra-chave do JavaScript usada para criar uma variável.
9. 3.14, 11 e 5280 são todos tipos de dados.
10. Convenção de codificação que envolve a nomeação de identificadores com letras misturadas, como em ThisIsMyName.
11. Não é um número.
12. Um fragmento de informação cujo valor pode mudar.
13. Um fragmento de dados com um valor on/off seria armazenado como este tipo de dados.

Vertical

1. Um fragmento de dados cujo valor não muda.
2. Tipo de dados usado para armazenar caracteres, palavras e frases.
3. Quando um valor não é atribuído a uma variável ou constante, os dados são considerados...
5. Função JavaScript embutida usada para converter uma string em um inteiro.
8. O processo de conferência para verificar se os dados digitados pelo usuário são precisos é chamado...
10. A palavra-chave do JavaScript usada para criar uma constante.

Solução das Cruzadas do JavaScript



Dobra da Página

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

O que todos nós queremos em nossos dados de script?

É um encontro de mentes!

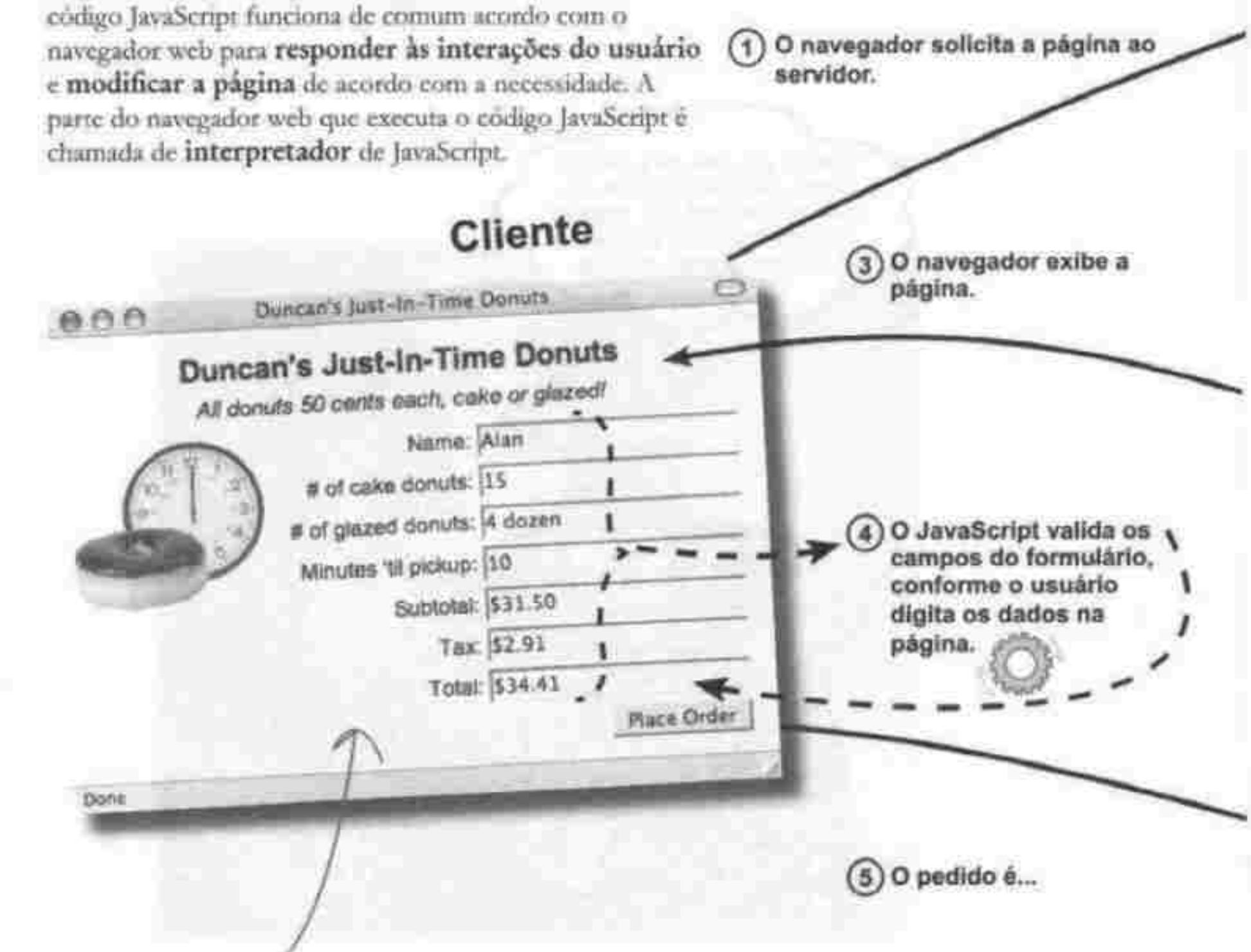
As informações digitas pelo usuário são o tipo de dados que você não deve confiar. É arriscado pressupor que os usuários digitarão dados e verificarão se estão OK. Uma solução com mais segurança no armazenamento envolve o uso de JavaScript.

Exploração do Navegador

Às vezes o JavaScript precisa saber o que está acontecendo no mundo à sua volta. Seus scripts podem começar como código em páginas da Web, mas no final eles vivem em um mundo criado pelo navegador ou cliente. Em muitos casos, os **scripts inteligentes** precisam conhecer mais sobre o mundo em que vivem. Nesse caso, eles podem se **comunicar com o navegador** para descobrir mais sobre ele. Seja descobrindo o tamanho da tela ou acessando o botão snooze do navegador, os scripts têm muito a ganhar cultivando seu relacionamento com o navegador.

Clientes, servidores e JavaScript

Ao clicar em um hyperlink ou digitar uma URL em seu navegador web, o navegador solicita a página em um servidor web, que então fornece a página de volta para ele ou para o **cliente** web. O JavaScript não insere a figura antes do navegador exibir a página. Então, o código JavaScript funciona de comum acordo com o navegador web para **responder às interações do usuário** e **modificar a página** de acordo com a necessidade. A parte do navegador web que executa o código JavaScript é chamada de **interpretador** de JavaScript.



O código JavaScript executa integralmente no cliente, sem exigir nada do servidor.

Quando uma página tiver sido fornecida ao navegador, o servidor estará completamente fora da equação. Praticamente tudo que o JavaScript fizer daqui para frente é restrito ao navegador. Isso torna as páginas **mais responsivas**, já que elas não têm que esperar que o servidor processe e retorne os dados. Esse processo faz com que o JavaScript seja conhecido como uma **linguagem de cliente**.

Solicitação de Página

```
GET / HTTP/1.1
Host: www.duncandonuts.com
Connection: close
Accept-Encoding: gzip
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg ...
Accept-Language: en-us
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.1.7) ...
```

O HTML diz onde está a página.

Página Apresentada

```
<html>
  <head>
    <title>Duncan's Just-In-Time Donuts</title>
    <link rel="stylesheet" type="text/css" href="donuts.css" />
    <script type="text/javascript">
      function updateTimer() {
        ...
      }
      function parseDonutString() {
        ...
      }
      function placeOrder() {
        ...
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div class="heading">Duncan's Just-In-Time Donuts</div>
      <div class="subheading">All donuts 50 cents each, cake or glazed</div>
      <div id="left">
        <image src="donutstime.gif" alt="Just-In-Time Donuts" />
      </div>
      <div id="right">
        ...
      </div>
    </div>
  </body>
</html>
```

O CSS torna a página mais bonita.

O JavaScript torna a página interativa, neste caso validando os dados digitados pelo usuário.

Servidor

② O servidor responde com uma página.

⑤ ...ao servidor.

⑤ ...submetido...

Poder do Cérebro

Existem outras tarefas que fazem mais sentido realizar no cliente do que no servidor?

O que um navegador pode fazer por você?

O seu navegador web **cliente** é responsável pela **execução do código JavaScript**, onde os scripts acessam o ambiente do cliente. Por exemplo, um script pode obter a largura e a altura da janela do navegador, bem como o histórico de páginas da Web visitadas. Outros recursos interessantes do navegador que estão abertos ao JavaScript incluem um mecanismo de controle de tempo que funciona como um alarme de relógio e o acesso a cookies, permitindo armazenar os dados que permanecem mesmo depois de sair da página ou fechar o navegador.

Métrica do navegador

A métrica do navegador inclui várias medidas associadas ao tamanho da janela do navegador, à página da Web visível e até mesmo as informações sobre o fornecedor do navegador e o número da versão.

Histórico do navegador

O histórico do navegador é a lista de páginas recém-visitadas. Você pode usar o JavaScript para acessar a lista de páginas e direcionar o navegador para uma delas, criando seus próprios controles de navegação.

Cookies

Cookies são como variáveis que se armazenam no disco rígido do usuário através do navegador, fazendo com que eles permaneçam após uma sessão da Web. Em outras palavras, você pode sair de uma página e voltar e os dados ainda estarão lá.

Temporizadores

Os temporizadores permitem acionar um fragmento de código JavaScript depois de transcorrido um tempo específico.

Esse recursos não são tudo o que o cliente tem a oferecer a seus scripts, mas eles devem lhe dar uma idéia de que há mais para o JavaScript do que existe dentro da página. De fato, existe uma série de situações onde é útil **olhar além da página** e obter uma ajudinha do navegador.

Não existem perguntas idiotas

P: Então, o JavaScript faz parte do cliente?

R: Sim. Os navegadores web que suportam JavaScript vêm com um interpretador, que é o responsável pela leitura do código JavaScript de uma página e que, em seguida, executa esse código.

P: Se o código JavaScript executa no cliente, como ele está relacionado ao servidor?

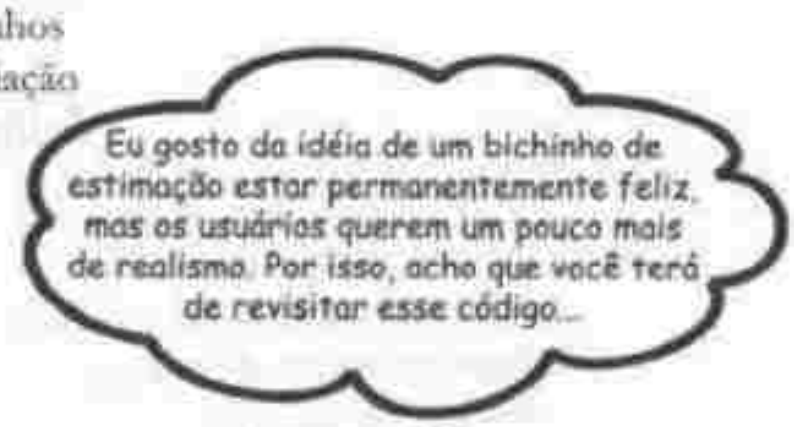
R: O código JavaScript normalmente não possui uma associação direta com o servidor web, já que ele executa exclusivamente no cliente. O JavaScript é comumente usado para interceptar dados da web conforme são retransmitidos do servidor para o navegador. No entanto, é possível escrever scripts que solicitem informações ao servidor e, em seguida, processem e exibam essas informações na página. Essa técnica é chamada Ajax, e conversaremos sobre como usá-la no Capítulo 12.

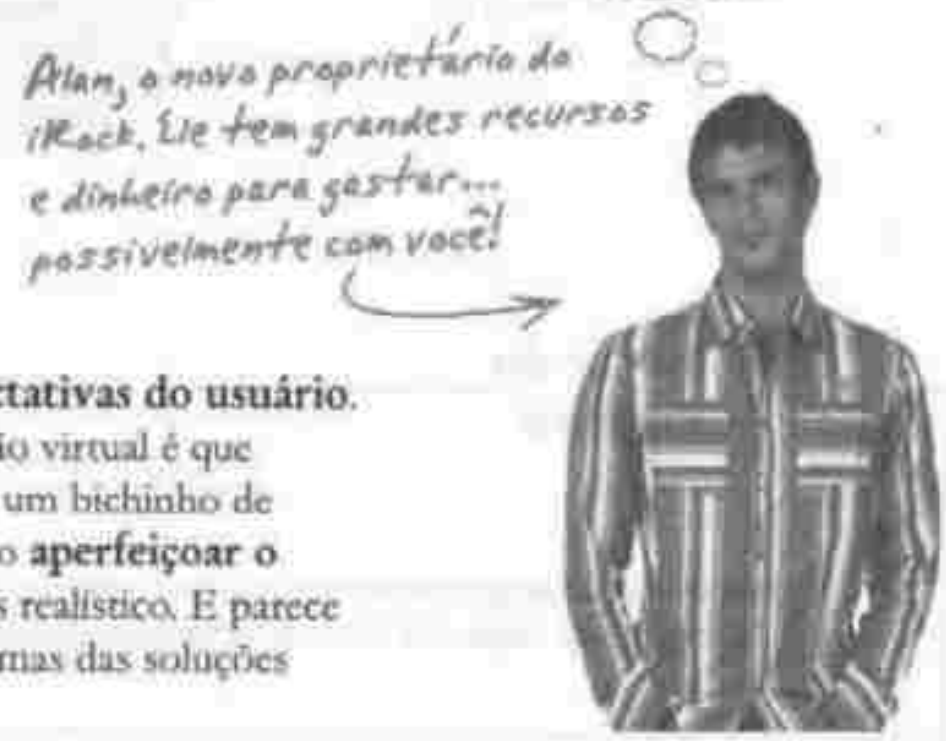
P: O JavaScript permite que você controle o cliente?

R: Sim e não. Embora os navegadores web permitam o acesso a certas partes do ambiente do cliente, eles não dão uma liberdade ilimitada ao JavaScript por razões de segurança. Por exemplo, a maioria dos navegadores não permite scripts para abrir e fechar janelas sem a aprovação do usuário.

O iRock está muito feliz

Lembra-se do iRock? O seu JavaScript fez tanto sucesso que Alan, um jovem empresário, comprou sua participação acionária. Mas ele o chamou de volta, porque existem alguns problemas... os usuários estão perturbados com o **persistente estado de felicidade** do iRock. Obviamente, todos nós queremos que nossos bichinhos de estimação sejam felizes, mas o iRock parece ter uma variação emocional rigorosamente limitada.





O problema do iRock tem a ver com as **expectativas do usuário**. A idéia por detrás de um bichinho de estimação virtual é que ele deveria se parecer o máximo possível com um bichinho de estimação real. O seu desafio é descobrir como **aperfeiçoar o comportamento** do iRock para torná-lo mais realístico. E parece que o navegador web cliente pode conter algumas das soluções para o problema do iRock...

O iRock precisa ser mais responsivo

Vamos levar em consideração alguns possíveis comportamentos do iRock, com o objetivo de tentar fazer com que a rocha seja mais **realística** e **cativante** para o usuário, além de mais **interativa**. Teoricamente, o iRock deveria se tornar mais **responsivo** ao usuário, à medida que sua variação de comportamento emocional aumentasse.

Faça o download do código mais recente para o iRock 2.0 em www.altabooks.com.br.

Rocha furiosa

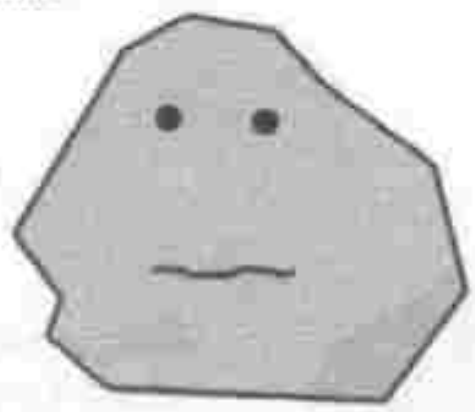
O iRock, de vez em quando, fica zangado sem nenhuma razão aparente fazendo com que o dono tenha que acalmá-lo.

Deprimida

O iRock chora toda vez que a página é fechada, exigindo que o usuário deixe o navegador aberto para evitar ele tenha um colapso.

Solitária

O iRock volta a se sentir solitário quando é deixado sozinho, exigindo que o usuário clique nele periodicamente dando a ele atenção.



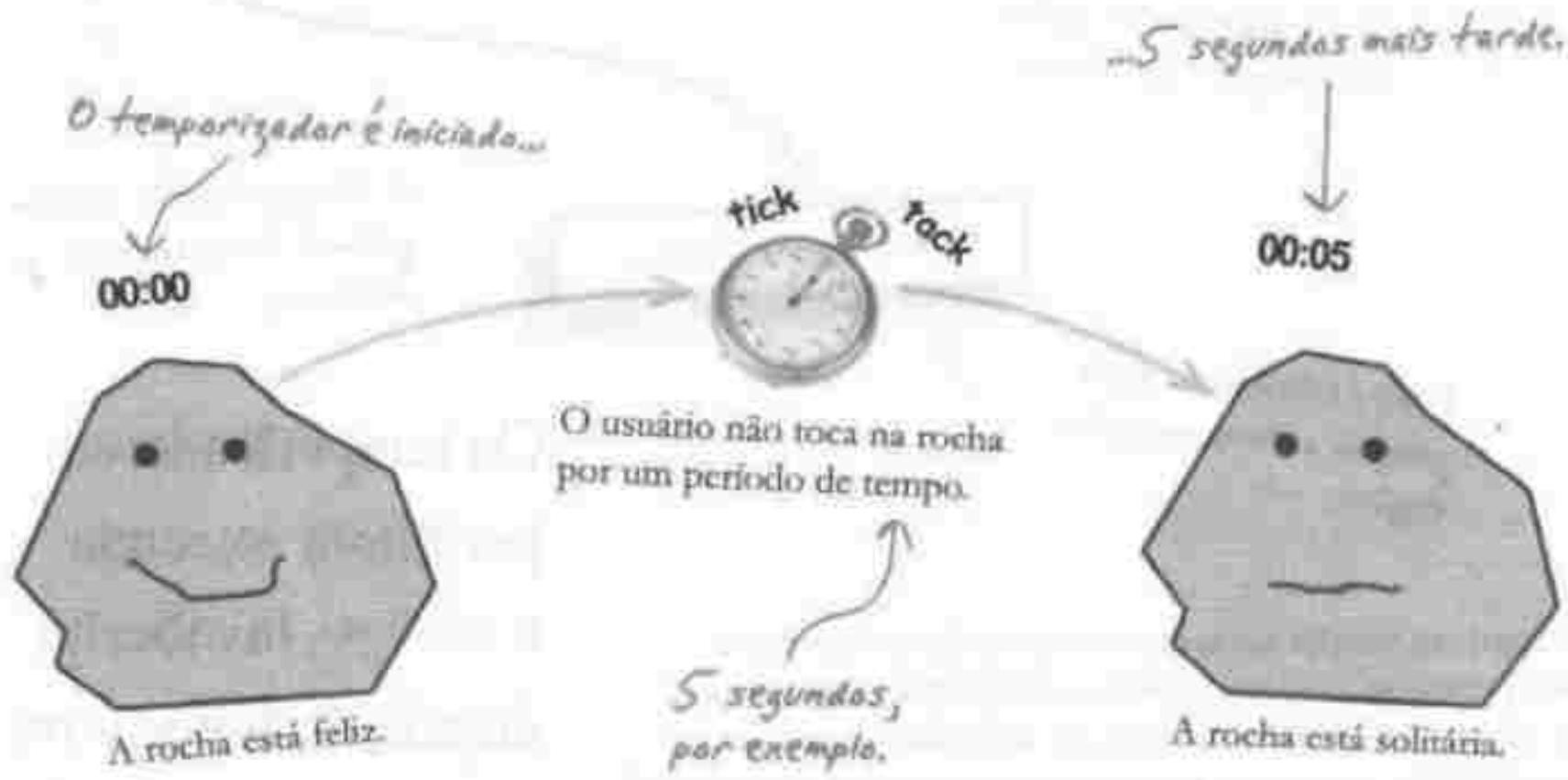
Poder do Cérebro

Quais desses comportamentos fazem sentido o iRock ter? Como você usaria o JavaScript para implementar esses comportamentos no script do iRock?

Eu gosto da idéia de uma rocha se sentir solitária depois de um tempo, porque é isso que os bichinhos de estimação fazem no mundo real. Podemos mudar o comportamento do iRock ao longo do tempo?

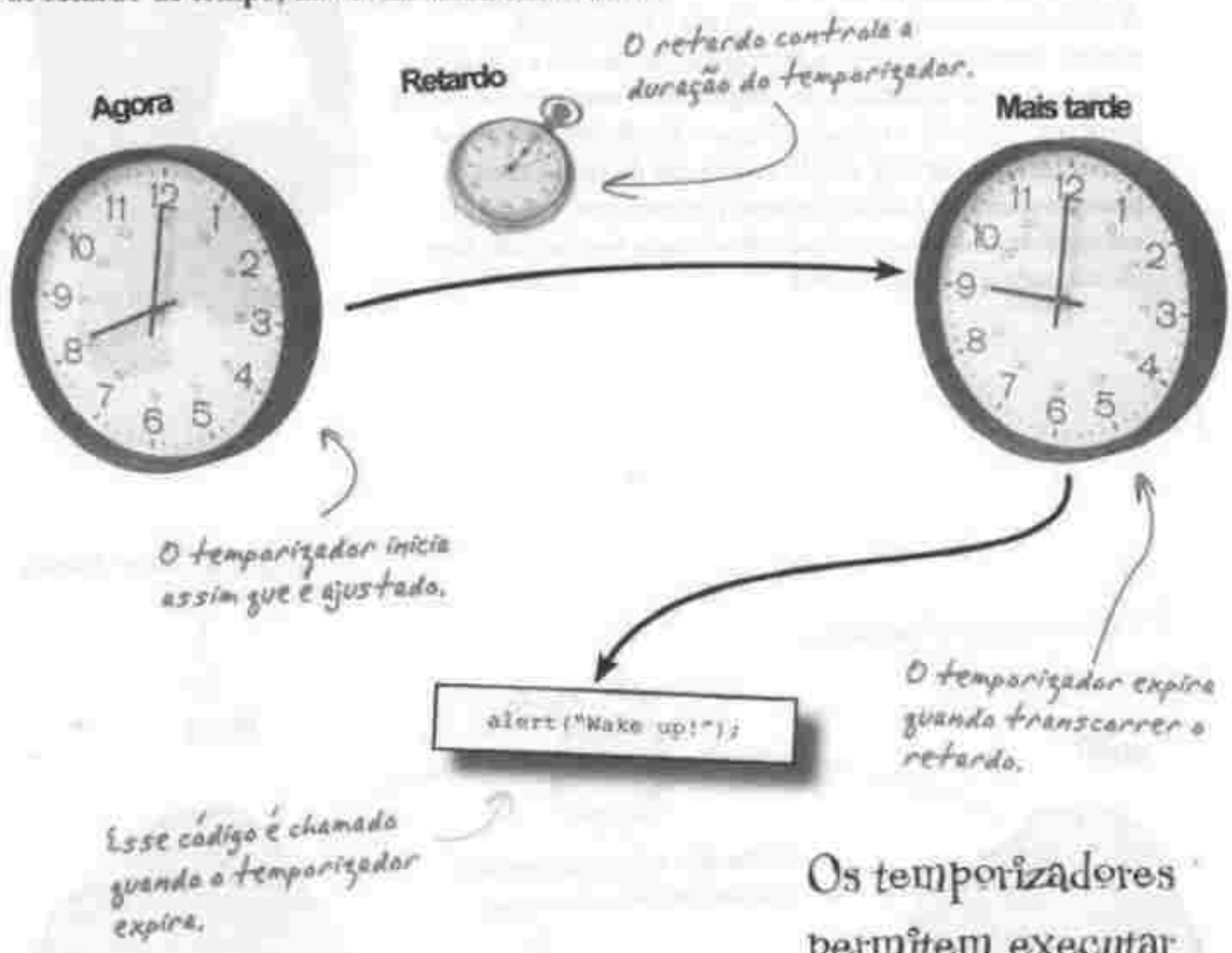
O JavaScript informa quando o usuário está fazendo alguma coisa... e quando não está.

A idéia de uma rocha se sentir solitária é interessante porque estimula o usuário a interagir com ela sem a culpa da sobrecarga, e o **usuário é recompensado** com uma **resposta positiva** do iRock. O desafio é usar o JavaScript de alguma forma para mudar esse estado emocional do iRock com o passar do tempo. A idéia é esperar por um certo tempo e, em seguida, mudar o estado do iRock, se o usuário não tiver clicado nele e o tempo passar.



Temporizadores associam ação ao tempo transcorrido

O JavaScript permite o ajuste de **temporizadores**. Um temporizador JavaScript funciona como o alarme de um relógio: você diz a ele quanto tempo esperar e quando expirar o tempo, um certo fragmento de código executará. No entanto, ao contrário de um alarme de relógio, os temporizadores JavaScript são acionados **quando tiver passado um certo período de tempo**, o oposto do acionamento **em um tempo determinado.** Isso não é problema, significa apenas que você precisa pensar em termos de **retardo de tempo**, em vez das horas exatas do dia.



O legal em relação aos temporizadores JavaScript é que eles permitem executar qualquer código que você queira ao expirarem. Algumas páginas da Web com dados que são modificados regularmente usam temporizadores para atualizá-los depois de um certo retardo, enquanto outros usam temporizadores para detectar se o usuário não interagiu com a página durante um tempo.

Os temporizadores permitem executar o código JavaScript depois de transcorrer um certo período de tempo.

Um temporizador decomposto

As duas peças-chave para se ajustar um temporizador no JavaScript são: 1) estabelecer o **retardo de tempo** e 2) informar ao temporizador **qual código executar** quando o retardo transcorrer. O temporizador começa a contar a partir do momento que você o ajustar.

O retardo de tempo é expresso em milésimos de segundo, que é a milésima parte de um segundo. Multiplique o número de segundos que você deseja por 1.000 para descobrir o número de milésimos de segundo. Por exemplo, 2 segundos são 2.000 milésimos de segundo.

Exibe uma mensagem

```
alert("wake up!");
```

O código que é executado quando o temporizador expirar pode ser qualquer código JavaScript que você deseje, inclusive uma única instrução, várias instruções (cada uma com uma terminação em ponto-e-vírgula) ou uma chamada para uma função embutida ou personalizada.

```
refresh(); setTimeout(refresh, 120000);
```

Atualiza a página da Web.

Ajusta um outro temporizador.

Quando os temporizadores JavaScript comuns expiram e o código do temporizador é executado, o temporizador é finalizado, e fim de papo. Esse tipo de temporizador é conhecido como temporizador de **disparo único** porque ele aciona um fragmento de código exatamente **uma única vez**. Também é possível criar um temporizador de **intervalo**, que ajusta **vários** intervalos em vez de um único retardo. Um temporizador de intervalo continua a chamar o código do temporizador **repetidamente**, após cada intervalo, até que você diga a ele para parar. Embora os temporizadores de intervalo com certeza tenham seu lugar, o temporizador de solidão do iRock é definitivamente um temporizador de disparo único.

O temporizador expira, eu me sinto sozinho, fim da história.

Exercício

Correlacione abaixo os períodos de tempo em milésimos de segundo aos seus equivalentes em segundos.

500ms	5 minutos
300.000ms	5 segundos
5.000ms	1/2 segundo

Solução do Exercício

Correlacione abaixo os períodos de tempo em milésimos de segundo aos seus equivalentes em segundos.

500ms	5 minutos
300.000ms	5 segundos
5.000ms	1/2 segundo

Ajuste um temporizador com setTimeout()

A função embutida JavaScript que possibilita os temporizadores (disparo único) é setTimeout(). Os dois fragmentos de informação que você precisará para a função são o **temporizador de retardo** e o **código para executar** quando o temporizador expirar (disponível em *http://www.headfirstlabs.com/books/hfjd/*), mas não necessariamente nessa ordem. Eis um exemplo:

A função setTimeout() ajusta um temporizador de disparo único.

O retardo do temporizador é de 600.000 milésimos de segundo, que são 600 segundos ou 10 minutos

```
setTimeout("alert('Wake up!');", 600000);
```

O código JavaScript é fornecido à setTimeout() como uma string de texto, e por isso deve estar entre aspas.

Quando o temporizador expira, uma caixa de alerta é exibida.

Nunca coloque vírgulas em um número JavaScript, mesmo que ele seja muito grande.

Essa chamada da função setTimeout() cria um temporizador que aguarda 10 minutos e, em seguida, exibe uma caixa de alerta.

Mais de perto: a função setTimeout()

Eis a forma geral que a função setTimeout() assume:

Inclui os dois argumentos de função.

```
setTimeout + ( + Timer code + , + Timer delay + );
```

O código a ser executado quando o temporizador expirar.

Separa os dois argumentos de função.

O retardo em milésimos de segundo.

Termina a instrução.

Ajustar um temporizador de intervalo é semelhante a ajustar um temporizador de disparo único, exceto se você chamar a função setInterval() no lugar de setTimeout(). O resultado final do ajuste de um temporizador de intervalo é o código que executa **repetidamente** a cada término de retardo de intervalo:

Minutos em milésimos de segundo.

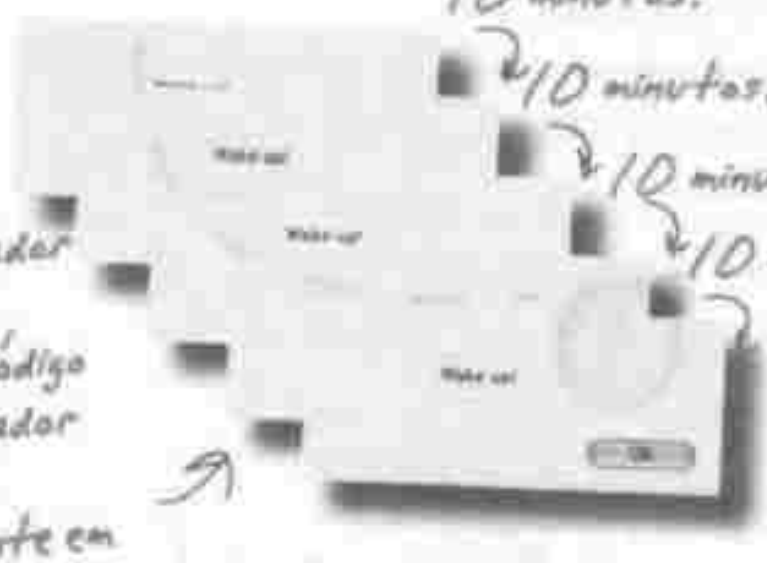
```
var timerID = setInterval("alert('Wake up!');", 600000);
```

Reserva o ID do temporizador.

Ajusta o temporizador recorrente.

PRESTE ATENÇÃO! **O retardo do temporizador pode ser especificado em milésimos de segundo.**

Um milésimo de segundo é uma fração de um segundo. Então, se você esquecer de usar os milésimos de segundo, acabará com alguns temporizadores absurdamente curtos (rápidos).



Um temporizador de intervalo resulta no código de temporizador que executa repetidamente em intervalos regulares.

Aponte seu lápis

Escreva um código para alterar, após 5 minutos, a imagem do iRock de feliz para solitário. Dica: O ID do elemento de imagem do iRock é rockImg, e o nome do arquivo da imagem solitária é rock.png.*

.....

.....

Antes de virar a página, tente esse código em sua versão da página irock.html para ver se funciona.

* Faça o download deste arquivo em www.altabooks.com.br

Aponte seu lápis
Solução

Escreva um código para alterar, após 5 minutos, a imagem do iRock de feliz para solitário. Dica: O ID do elemento de imagem do iRock é rockImg, e o nome do arquivo da imagem solitária é rock.png.

Aspas e apóstrofos são alternados para aninhar as funções adequadamente.

```
setTimeout( document.getElementById('rockImg').src
= 'rock.png', 5 * 60 * 1000);
```

O retardo de cinco minutos é calculado em milésimos de segundo, convertendo-se primeiro para minutos (x60) e, em seguida, para milésimos de segundo (x1000).

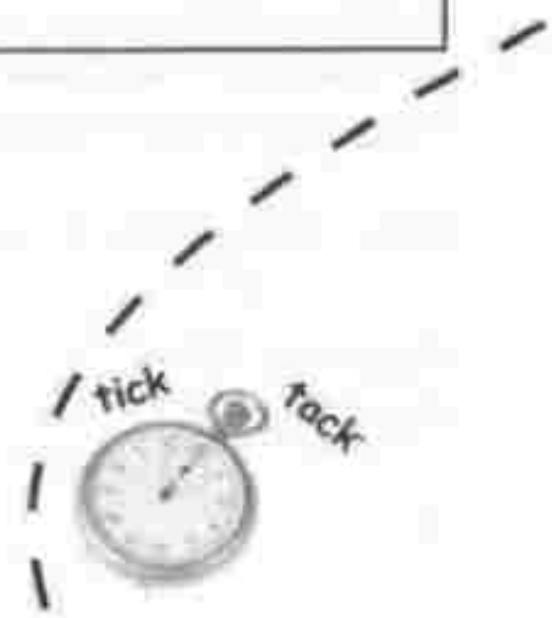
A imagem da rocha solitária.

A imagem do iRock é alterada, definindo-se um novo arquivo de imagem ao atributo src do elemento de imagem.

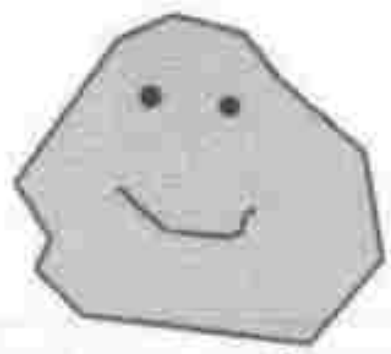
O ID do elemento de imagem.

Agora o iRock se sente solitário!

Certifique-se de ter feito as alterações detalhadas na sua irock.html acima, e dê uma ênfase ao iRock. Agora, o iRock demonstra solidão quando é deixado sozinho por cinco minutos (sem que o usuário clique nele). Esse retardo de tempo pode fazer com que a rocha pareça carente, mas a idéia é manter o usuário ocupado. Além disso, o bichinho de estimação certo se conhece nas horas incertas! Ou algo parecido...



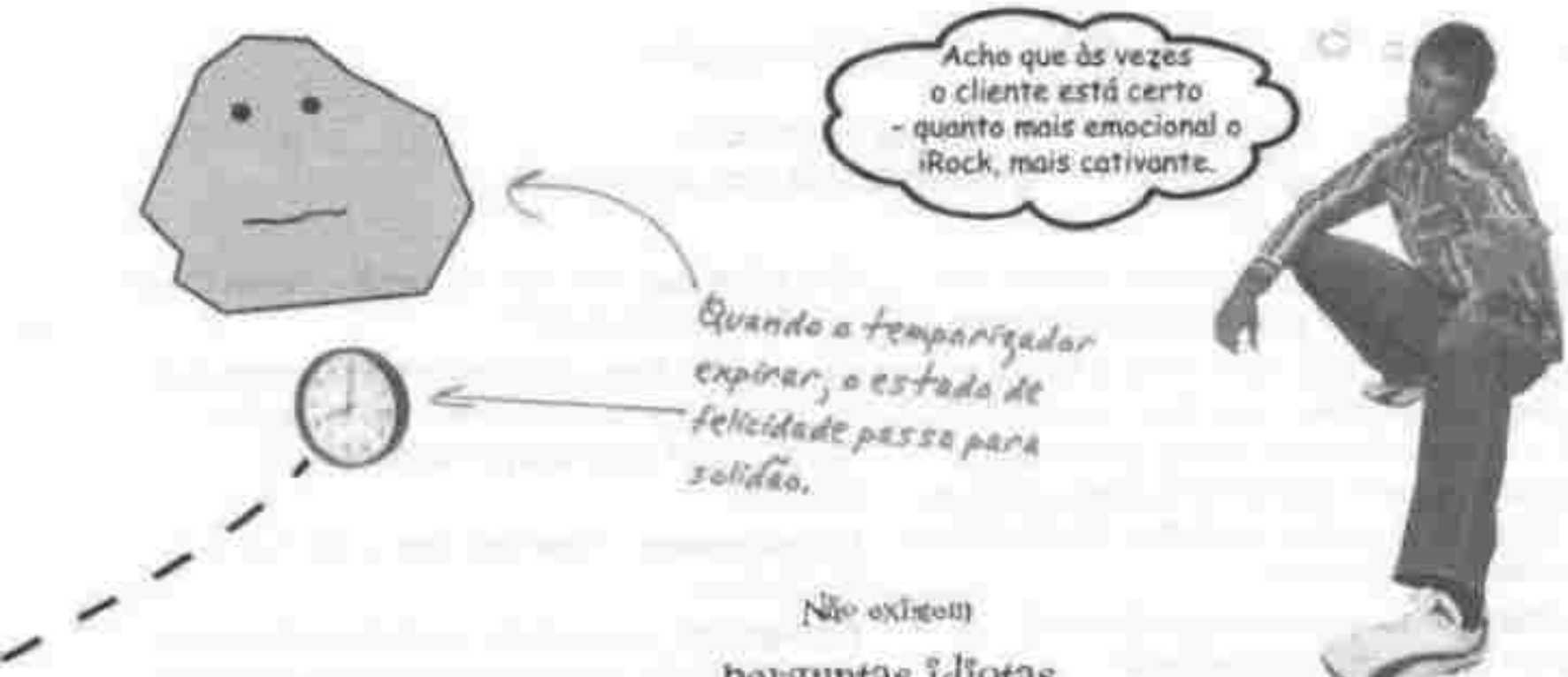
Um temporizador inicia literalmente a contagem regressiva do montante de "tempo feliz" que resta para o iRock.



A felicidade agora passa rápido para o iRock, o que é muito mais realístico.

Nerd

Você pode acelerar as alterações emocionais do iRock usando um retardo de tempo menor ao chamar a função setTimeout(). Essa é uma ótima maneira de testar o script sem ter que esperar pelo que vai acontecer.



Não existem

perguntas idiotas

P: Se a idéia é fazer com que o iRock retorne sempre ao estado solitário após 5 minutos, por que não usar um temporizador de intervalo?

R: A resposta tem a ver com que forma um temporizador de disparo único é usado. Embora a rocha seja capaz de se sentir solitária periodicamente, ela só fica sozinha *após* um período de felicidade. O temporizador é definido no clique inicial, a rocha se torna solitária quando o temporizador expirar 5 minutos depois e permanecer solitária até que ela seja clicada novamente. Isso não se parece com a regra de um temporizador de intervalo, é diferente do funcionamento de um temporizador de intervalo – ele acionaria a cada cinco minutos, independente do que o usuário faz.

P: O que acontece se o usuário fechar o navegador antes de o temporizador expirar?

R: Nada. O interpretador de JavaScript é desligado quando o navegador é fechado, e todo o código JavaScript pára, inclusive todos os temporizadores em andamento.

P: Como posso criar um temporizador que acione o código a uma certa hora do dia?

R: Como os temporizadores são baseados em retardo, e não em horas específicas, você precisa converter a hora do dia em um retardo. Isso pode ser feito subtraindo a hora atual da hora de acionamento desejada. Esse cálculo exige alguma ajuda do objeto de JavaScript Date, que você aprenderá mais no Capítulo 9.

P: Eu tenho uma página com dados que se modificam e gostaria que ela atualizasse a cada 15 minutos. Como faço isso?

R: Use a função setInterval() para definir um temporizador de intervalo de 15 minutos, que é 900.000 milésimos de segundo (15 x 60 x 1000). Você precisa do código do temporizador para atualizar a página, que é realizado chamando-se o método reload() do objeto location, assim: location.reload(); Agora, o temporizador aciona a página atualizada a cada 15 minutos. Obviamente, você também pode usar Ajax (Capítulo 12) para carregar os dados dinamicamente, em vez de atualizar a página.

P: Entendo que um temporizador de intervalo continua indefinidamente. Como faço parar um temporizador de intervalo?

R: Uma função chamada clearInterval() é usada para eliminar um temporizador de intervalo que tenha sido ajustado com setInterval(). A função clearInterval() exige que você passe o ID do temporizador de intervalo a ser eliminado, o qual é retornado pela função setInterval() ao criar o temporizador. Sim, as funções são capazes de retornar informações. Após reservar o valor de retorno de setInterval() em timerID, por exemplo, passe-o para o clearInterval() cancelar o temporizador, dessa forma : clearInterval(timerID).

O navegador exposto

Entrevista da semana: Confissões de um cliente web

Use a Cabeça: Obrigado por abrir mão de um dia atribulado e sentar para batermos um papo.

Navegador: Atribulado é a palavra certa. Como se eu não tivesse muito o que fazer com HTML e CSS, e todas as dores de cabeça processando páginas com aqueles dois personagens, agora eu tenho que pelejar com o JavaScript. É um animal completamente diferente.

Use a Cabeça: O que você quer dizer? O JavaScript é selvagem e indomável?

Navegador: Ah, não. Eu não quis dizer literalmente "animal". Eu quis dizer que o JavaScript traz seu próprio conjunto de problemas específicos com que eu tenho que me preocupar. Agora tenho essa função completamente nova de ler o código JavaScript, esperando ansiosamente que ele não esteja codificado inadequadamente e, em seguida, executá-lo enquanto fico de olho no HTML e no CSS ao mesmo tempo.

Use a Cabeça: Entendo. Como esses três se relacionam?

Navegador: Felizmente, esse é o menor dos meus problemas. Esses três normalmente funcionam bem juntos, embora ocasionalmente o JavaScript se empolgue e deturpe algum código HTML. O problema é que normalmente não há nada que eu possa fazer em relação a isso, porque a minha função é basicamente fazer o que eu já disse.

Use a Cabeça: Então, você se adapta a um estilo meio "bajulador"?

Navegador: Suponho que haja uma forma de expressar isso, porém é mais correto dizer que eu valorizo a consistência acima de qualquer coisa.Faço tudo de acordo com as regras. Minha função é pegar o código que o servidor me fornece e fazer exatamente como ele diz.

Use a Cabeça: Mesmo se você souber que está errado?

Navegador: Faço o possível para resolver os problemas quando os vejo, mas é um trabalho difícil. Além disso, é um assunto para um outro dia (Capítulo 11). Pensei que fôssemos falar sobre minha função como um cliente web.

Use a Cabeça: Ah, sim. Eu perdi o rumo. O que significa ser um cliente?

Navegador: Bem, significa principalmente que eu permaneço na extremidade da recepção, recebendo as páginas após solicitá-las ao servidor.

Use a Cabeça: O que isso tem a ver com o JavaScript?

Navegador: Na verdade, muitíssimo. Como eu lido com todas as tarefas não muito agradáveis, como a de exibir as páginas da web e processar as entradas de informações do usuário, o JavaScript está lá, ao meu lado, colocando seu nariz e alterando tudo. Mas isso não é de todo ruim. Há muitas coisas legais que o JavaScript pode fazer que eu não ousaria fazer sozinho.

Use a Cabeça: Tais como?

Navegador: Bem, eu nunca assumiria a responsabilidade de fazer algo especial quando um usuário colocasse o mouse sobre uma imagem ou redimensionasse minha janela, por exemplo. O JavaScript, por outro lado, tem tudo a ver com fazer coisas especiais. Não é nada demais um script alterar a aparência da página ou, ao contrário, reorganizar o conteúdo em resposta às alterações do cliente. E por mim, tudo bem, porque o código JavaScript executa página por página, portanto só impacta uma página específica ou um site da web.

Use a Cabeça: Você falou do JavaScript como se ele fosse uma outra entidade. O JavaScript não é você?

Navegador: Ambos. Com certeza, o JavaScript faz parte de mim, mas você pode pensar nele como uma entidade própria, porque ele só pode acessar o cliente (eu) através de uma interface limitada. Em outras palavras, eu não dou ao JavaScript um acesso incontrolado a tudo sobre mim. Isso seria um pouco irresponsável, já que não posso controlar quem escreve os scripts e me pede para executá-los.

Use a Cabeça: Entendi. Bem, obrigado por esclarecer algumas dessas coisas de cliente.

Navegador: Fico contente em ajudar.

Vários tamanhos de tela, várias reclamações

Alan mal tinha terminado de pagar a transformação emocional do iRock a você, quando uma nova onda de reclamações começaram a transbordar de proprietários frustrados do iRock. Parece que o **tamanho** do iRock não é muito consistente, com alguns usuários relatando a "síndrome da rocha que encolhe", enquanto outros experimentam um medo intenso das "formações parciais de uma rocha gigante". Você é o cara que o Alan confia, portanto é hora de ganhar algum dinheiro extra para consertar o iRock novamente.

Outros usuários estão vendo apenas uma parte de um iRock gigante.

Alguns usuários estão relatando sobre um iRock que é surpreendentemente pequeno.

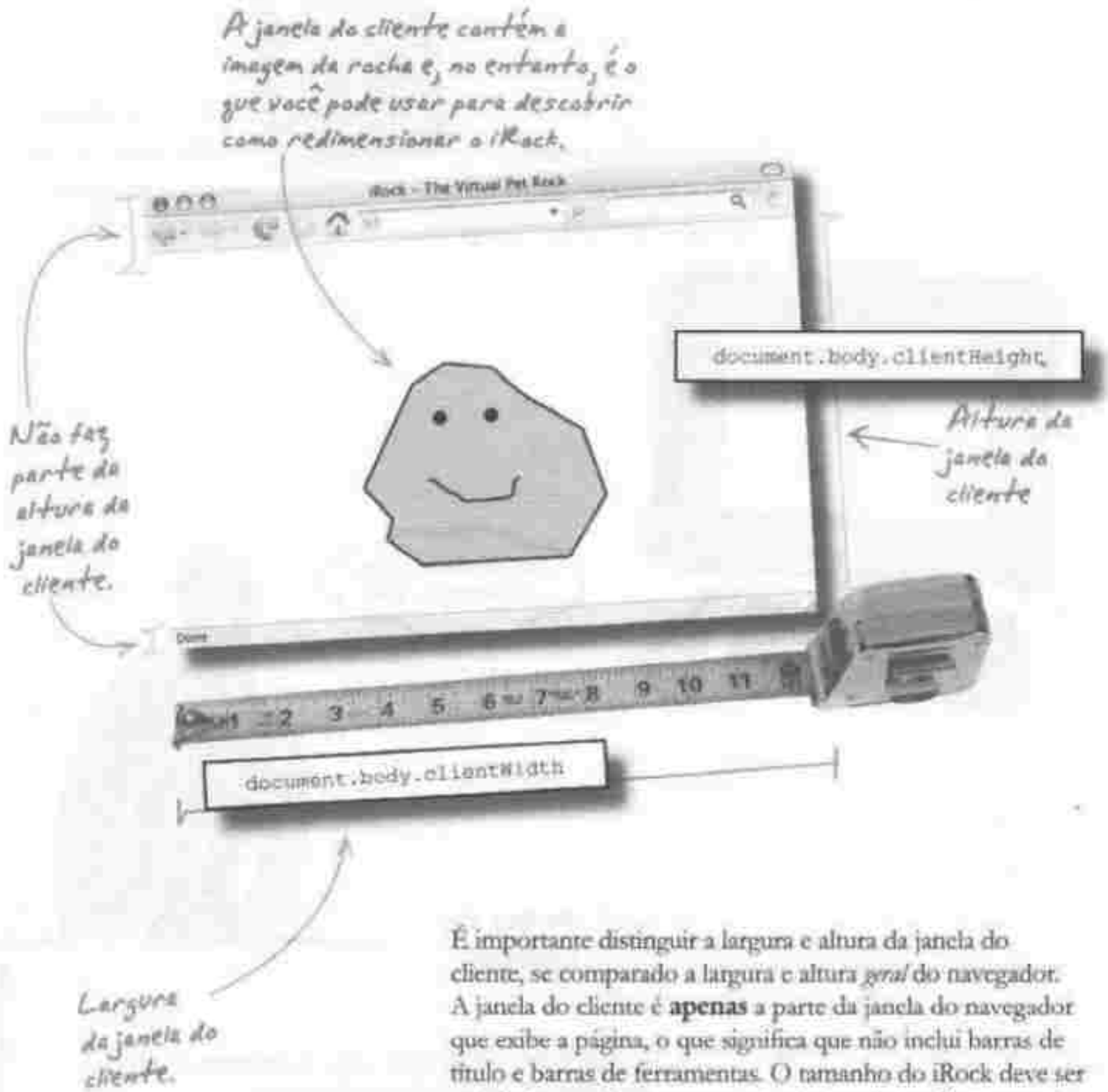
Poder do Cérebro

O que está acontecendo com os diversos tamanhos de rocha em navegadores diferentes?

Use o objeto document para obter a largura da janela do cliente

O problema do iRock tem a ver com o fato de que o tamanho dele não muda quando o tamanho da janela do navegador muda. Isso pode ser bom até você considerar a **variação substancial** no tamanho do navegador em todos os computadores capazes de navegar na Web, inclusive **dispositivos portáteis pequenos** e computadores desktop com **monitores gigantes**. Você precisa de uma forma para verificar o tamanho da janela do navegador, que pode então ser usada como uma medida para **redimensionar a imagem da rocha**.

A janela do cliente é apenas a parte da janela do navegador que exibe uma página da web.



É importante distinguir a largura e altura da janela do cliente, se comparado a largura e altura *geral* do navegador. A janela do cliente é **apenas** a parte da janela do navegador que exibe a página, o que significa que não inclui barras de título e barras de ferramentas. O tamanho do iRock deve ser calculado com base no tamanho da janela do **cliente**, e não no tamanho *geral* da janela do navegador.

Use as propriedades do objeto document para definir a largura da janela do cliente

O tamanho da janela do cliente está intimamente associado à página da web, que você acessa no JavaScript usando o objeto document. Esse é o mesmo objeto usado para acessar elementos na página com o método getElementById(). As propriedades de body.clientWidth e body.clientHeight do documento contêm a largura e altura da janela do cliente.

document

O objeto document representa a própria página da web.

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>
    <script type="text/javascript">
      var userName;
      function greetUser() {
        alert('Olá, eu sou a sua nova de estimação!');
      }
      function touchRock() {
        if (userName) {
          alert("Eu gosto de atenção, " + username +". Obrigada.");
        }
        else {
          userName = prompt("Qual é o seu nome?", "Digite seu nome aqui.");
          if (userName)
            alert("Bom conhecer você," + username + ".");
        }
        document.getElementById("rockImg").src = "rock_happy.png";
        setTimeout("document.getElementById('rockImg').src = 'rock.png'",
          5 * 60 * 1000);
      }
    </script>
  </head>

  <body onload="greetUser();">
    <div style="margin-top:100px; text-align:center">
      <img id="rockImg" src="rock.png" alt="iRock" style="cursor:pointer"
        onclick="touchRock();" />
    </div>
  </body>
</html>
```

document.body

O corpo do documento representa a parte visível da página, inclusive a altura e a largura do cliente.

Não existem perguntas idiotas

P: Só para esclarecer, qual a diferença entre um cliente web, um navegador, a janela de um cliente e a janela de um navegador?

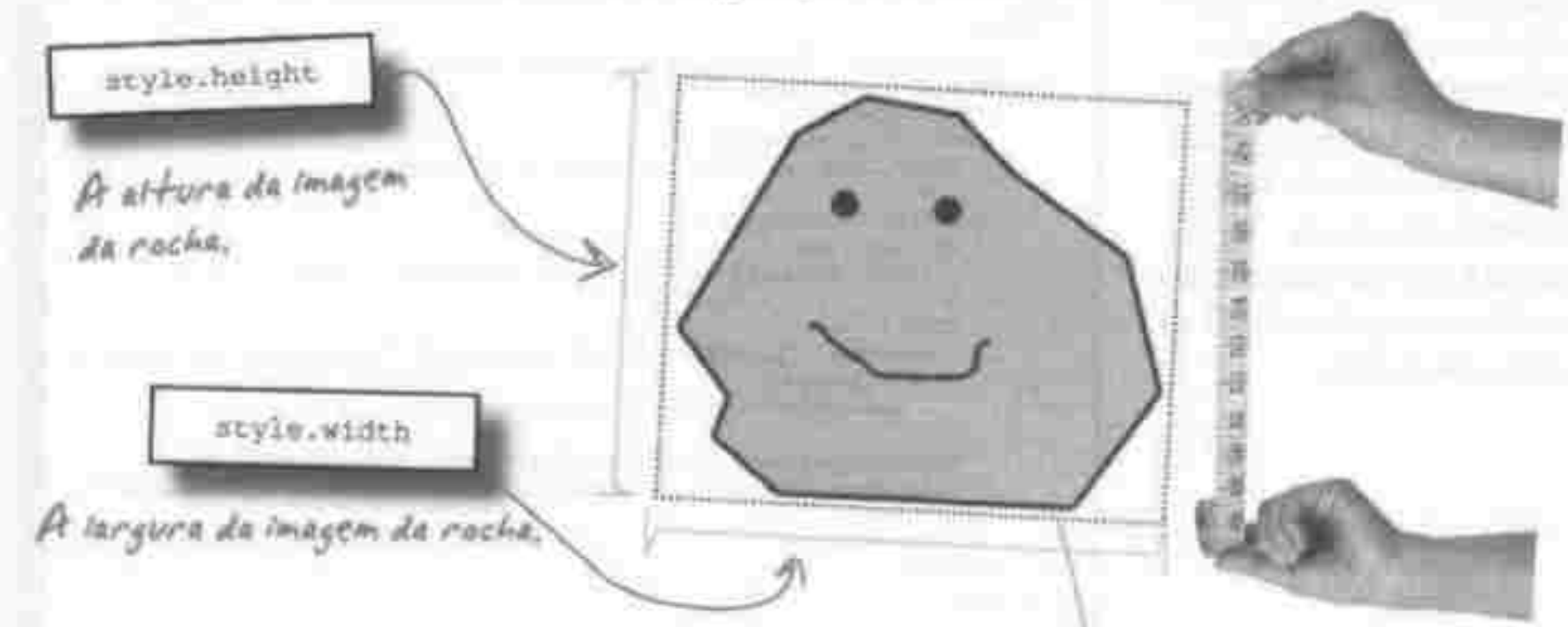
R: Sim, isso pode ser um pouco confuso. Em termos da Web em geral, refere-se a um navegador como um cliente web, porque está do lado do cliente de serviços de páginas da web. Dentro de um navegador, no entanto, o "cliente" assume um significado diferente porque ele se refere à área específica da janela do navegador onde a página aparece. Então, a janela do cliente é uma área dentro do navegador que não inclui outras coisas como as barras de título, barras de ferramentas, barras de rolagem etc.

P: Por que a janela do cliente é a medida preferida usada para redimensionar o iRock?

R: A janela do cliente oferece uma medida melhor para redimensionar a imagem da rocha porque ela reflete o montante real de espaço no qual a imagem é exibida. Isso elimina as variações que são difíceis de explicar, como os complementos de barras de ferramentas e as diferenças naturais das janelas de navegadores de diferentes fornecedores de plataformas e navegadores. Por exemplo, o Safari no Mac possui um tamanho de janela de navegador diferente do Firefox no Windows, mesmo que a parte exposta de cada um – a janela do cliente – seja a mesma.

Defina a altura e a largura da imagem do iRock

Conhecer o tamanho da janela do cliente não é muito útil no script do iRock sem ser capaz de **redimensionar a imagem da rocha**. Felizmente, você pode tentar ajustar o tamanho de uma imagem usando JavaScript com uma pequena ajuda do CSS (mesmo que CSS não seja o seu estilo). As propriedades width e height de um elemento de imagem não só permitem inicialmente determinar o tamanho da imagem, como também redimensionar dinamicamente a imagem quando necessário.



O código HTML da imagem da rocha é a chave para acessar as propriedades de estilo da imagem.

Existe um objeto style para cada elemento e página da web para que você possa acessar a largura e a altura de qualquer fragmento ou parte de uma página. Mas para acessar os estilos é necessário acessar primeiro o elemento de página da web, neste caso a imagem da rocha (se ainda não fez o download, pode obtê-lo em *http://www.headfirstlabs.com/books/hfjs/*). Isso requer o prático método getElementById() do objeto document:

```
<img id="rockImg" src="rock.png" alt="iRock" ... />
```

Esse código acessa a altura da imagem da rocha.

```
document.getElementById("rockImg").style.height
```

Para alterar o tamanho da imagem do iRock, defina um valor para a propriedade de largura e altura. Configurar cada propriedade **individualmente** dará certo, porque a outra propriedade **se ajustará automaticamente** para manter as mesmas **proporções** da imagem:

Defina a altura da imagem da rocha em 100 pixels.

```
document.getElementById("rockImg").style.height = "100px";
```

Você não precisa definir a largura... ela será dimensionada e se manterá proporcional à nova altura.

O iRock deve ser dimensionado em relação à página

Você ainda não tem uma estimativa de uso para alterar o tamanho da imagem da rocha, baseado no tamanho da janela do cliente. Como o tamanho da rocha pode mudar **na proporção** em relação ao tamanho da janela do cliente, precisamos definir o tamanho da rocha em **porcentagem** de tamanho da janela do cliente.

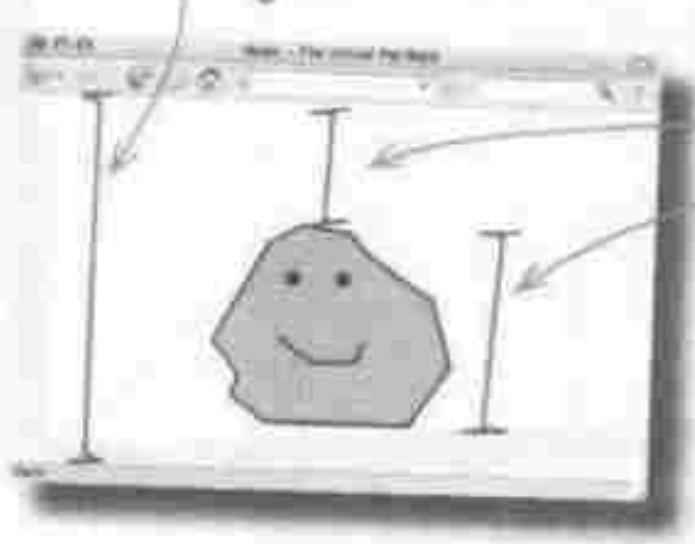
Mas você deve basear o tamanho da rocha na largura ou na altura da janela do cliente? Como os navegadores tendem a ser mais firmemente forçados à direção **vertical**, é mais seguro basear o tamanho da imagem da rocha na **altura** da janela do cliente.

```
(clientWindowHeight - 100) * 0.9 = rockImageHeight
```

A altura da janela do cliente.

100 pixels.

90% do que restou verticalmente.



Esse cálculo explica o espaço vertical da rocha na página (100 pixels) e, em seguida, dimensiona a imagem da rocha em 90% (0,9) do que restou. Às vezes, cálculos como este exigem uma pequena tentativa e erro para testá-los e descobrir qual funciona melhor. Você terá que tentar com o seu iRock para ver como ele funciona... mas, primeiro, há um código para ser escrito.

Aponte seu lápis

Escreva o código para a função resizeRock() que deve redimensionar a imagem da rocha com base no tamanho da janela do cliente. Adicione também o código ao manipulador de evento onload para chamar resizeRock() além de greetUser() que já estava sendo chamada.

```
function resizeRock() {
  ..................................................................
  ..................................................................
  ..................................................................
}
...
<body onload=
...          ......................................>
</body>
```

Aponte seu lápis
Solução

Sua função era escrever o código para a função resizeRock(), e também adicionar código ao manipulador de eventos onload para chamar resizeRock(). Lembre-se de verificar se greetUser() ainda está sendo chamada!

O tamanho da imagem da racha é calculada com base na altura da janela do cliente.

O ID da imagem da racha é usada para obter o elemento de imagem.

```
function resizeRock() {
  document.getElementById(rockImg).style.height =
  document.body.clientHeight - (100) * 0.9;
}
...
<body onload="resizeRock(); greetUser();">
...
</body>
```

Duas funções diferentes são chamadas quando a página é carregada pela primeira vez. Está perfeitamente correto ligar mais de um fragmento de código a um evento.

Diminua 100 pixels para justificar a posição vertical da racha.

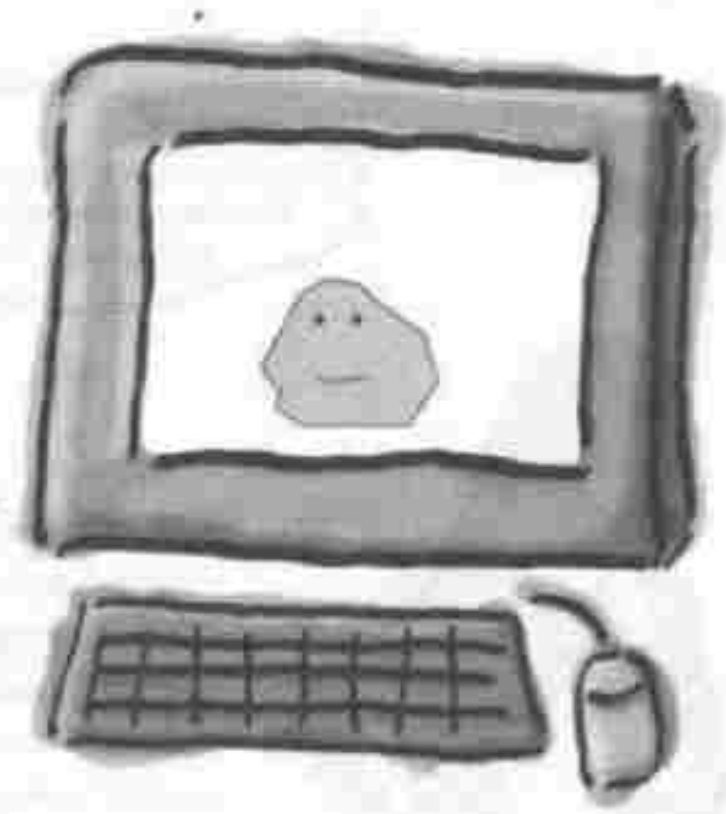
90% do tamanho restante da janela.

Pontos de Bala

- A função setTimeout() permite criar um temporizador de disparo único que aciona o código JavaScript, após ter passado um período de tempo.
- Para ajustar um temporizador que se repete em um certo intervalo, use setInterval() criando um temporizador de intervalo.
- Especifique sempre a duração em milésimos de segundo, que é a milésima parte de um segundo.
- Os elementos de página da web tem um objeto style usado para definir as propriedades de estilo, como width e height.
- A janela do cliente faz parte da janela do navegador que exibe nada mais do que a página da web.
- Você pode acessar a largura e a altura da janela do cliente usando as propriedades body.clientWidth e body.clientHeight do objeto document.

O seu iRock... evolui!

Com as alterações em seu código, o iRock evoluiu para se adaptar a cada ambiente único de navegação. Certifique-se de atualizar a sua iRock.html (disponível em *www.altabooks.com.br*) para coincidir com a página 104 e, em seguida, carregue-a em vários navegadores e diversos tamanhos de janela. Se quiser, tente também em seu reluzente iPhone!



Agora, o tamanho da imagem da rocha varia de acordo com o tamanho da janela do navegador.

Os usuários não vão mais informar problemas com seus bichinhos de estimação e o Alan está pronto para lhe fornecer uma tonelada de opções de compra de ações. Todo mundo está feliz... por enquanto.

Não existem

perguntas idiotas

P: Ainda não entendi a idéia dos 100 no cálculo do tamanho da imagem do iRock. Qual é a proposta?

R: O código HTML/CSS da página do iRock posiciona a imagem da rocha 100 pixels abaixo da página. Portanto, ela não está comprimida contra a parte superior da janela do cliente. O cálculo explica o posicionamento estético diminuindo a distância de 100 pixels antes de calcular a altura da rocha como uma porcentagem (90%) da altura da janela do cliente. Não há nada de mágico nos 100 pixels, isso só acontece para posicionar a rocha em um bom local na maior parte dos navegadores.

P: Posso mudar o tamanho de qualquer coisa que eu queira usando as propriedades de estilo width **e** height **do CSS?**

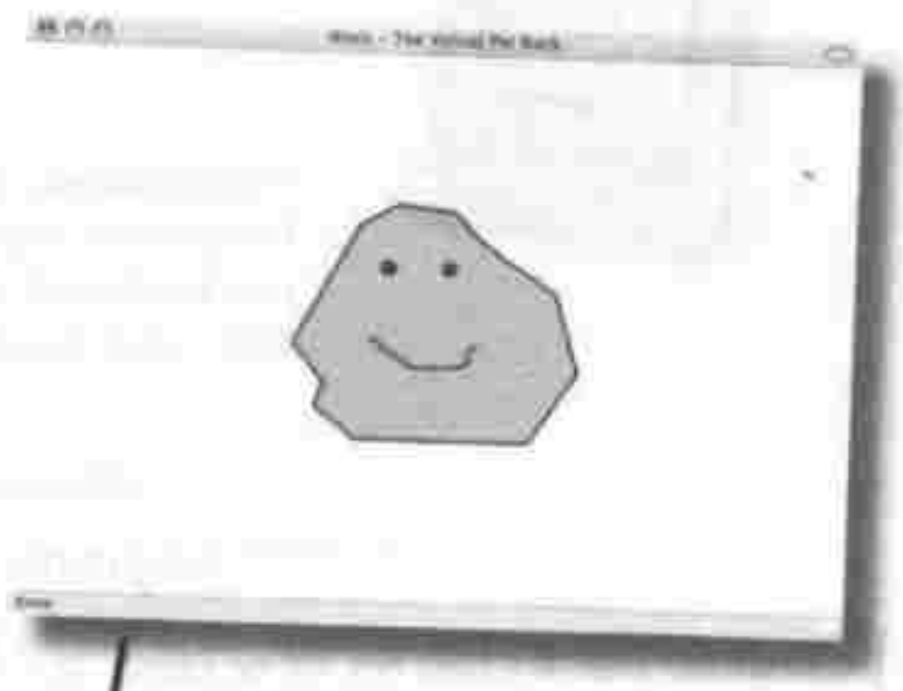
R: Praticamente. Felizmente, você está começando a ter uma pista do quão poderoso o JavaScript pode ser quando se trata de manipular o conteúdo de uma página da web. No caso do script do iRock, é a capacidade de consultar a janela do cliente para obter seu tamanho e, em seguida, usá-lo como base para alterar o tamanho de uma imagem.

P: Por que não alterar o tamanho da imagem do iRock no código JavaScript no topo da página, em vez de usar o evento onload**?**

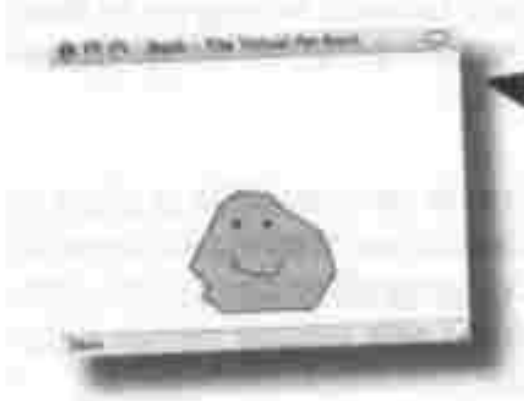
R: Esse problema tem a ver com o conteúdo de página da web que não é carregado até o evento onload ser disparado. Se o seu código JavaScript acessa elementos na página, como o código do iRock, você não pode executar nenhum código anterior ao evento onload.

O que acontece com o iRock quando o navegador é redimensionado? A imagem da rocha não permanece do mesmo tamanho?

Não, o tamanho da rocha não é dinâmico.
Alguns usuários estão envolvidos no redimensionamento da janela de seus navegadores, e o iRock não mudará de tamanho quando isso acontecer. Esses usuários não serão pessoas satisfeitas. Isso porque o tamanho da imagem da rocha só é alterado quando a página carregar a primeira vez no evento onload. Daí em diante, nada do que acontece resulta no redimensionamento de uma imagem. Infelizmente, estamos de volta ao início:

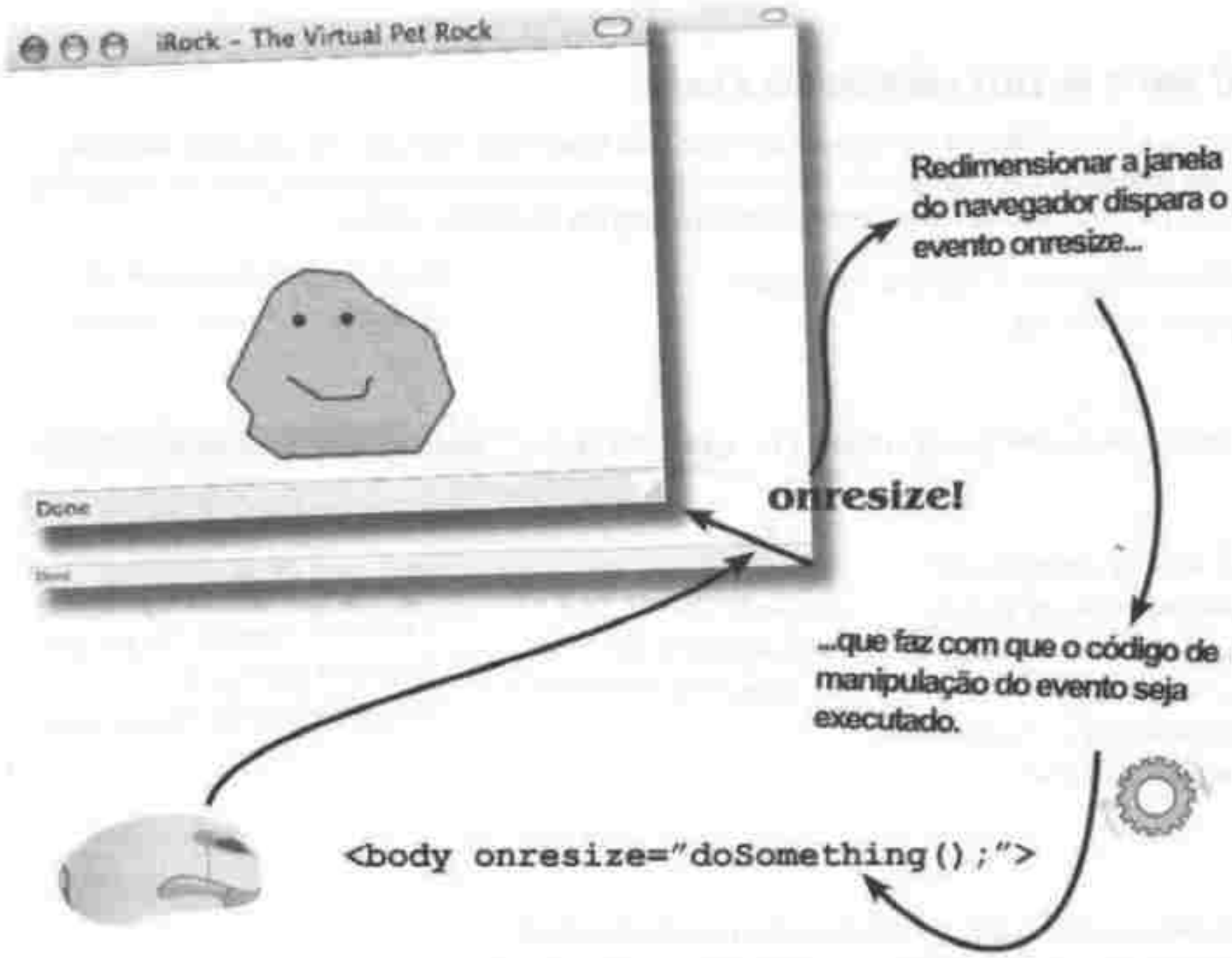


A imagem da rocha permanece do mesmo tamanho quando a janela do navegador for redimensionada.



onresize é acionado quando o navegador é redimensionado

Para que o tamanho da rocha mantenha seu tamanho proporcional à janela do cliente do navegador, o seu script precisa saber quando o usuário redimensionará a janela do navegador. Os redimensionamentos do navegador são comunicados usando-se um evento chamado onresize. O evento onresize é disparado sempre que a janela do navegador for redimensionada, e é justamente o que você precisa obter para redimensionar a imagem da rocha quando o tamanho da janela do navegador mudar.



Para responder um evento onresize, atribua o código JavaScript ao atributo onresize da tag <body>.

Aponte seu lápis

Uma dessas coisas não é parecida com a outra. Qual é... e por quê?

onload onresize onclick

.....

Aponte seu lápis
Solução

Uma dessas coisas não é parecida com a outra. Qual é... e por quê?

onload onresize onclick

Os eventos onresize e onclick são acionados pelo usuário, onload não é.

O evento onresize redimensiona a rocha

Agora é hora de obter a recompensa por criar uma função que redimensione a imagem da rocha. Para redimensionar a imagem da rocha em resposta ao redimensionamento da janela do navegador, você precisa chamar a função resizeRock() em resposta ao evento onresize:

Acionado quando a página é carregada pela primeira vez.

Acionado quando a janela do navegador é redimensionada.

```
<body onload="resizeRock(); greetUser();" onresize="resizeRock();">
```

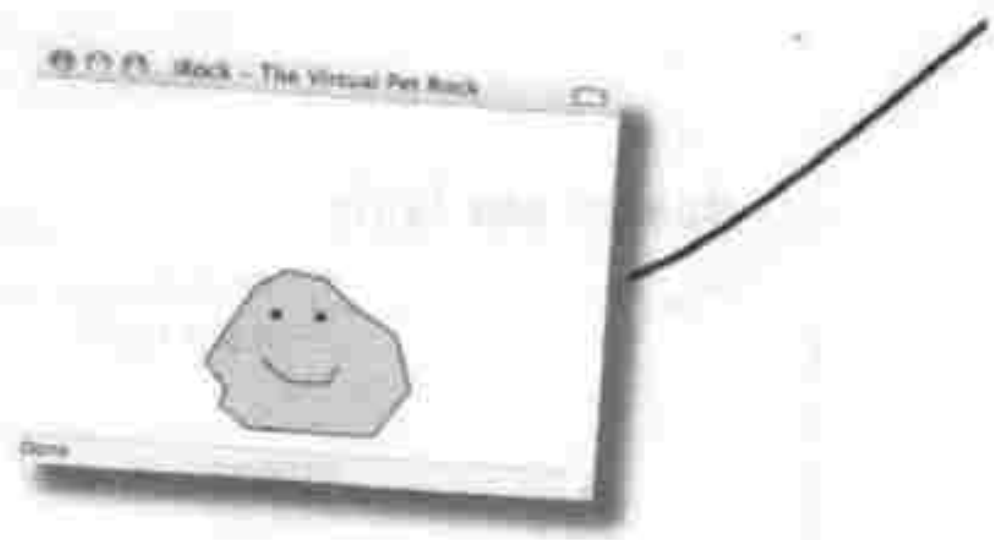
A função resizeRock() ainda é chamada quando a página é carregada pela primeira vez para definir inicialmente o tamanho da imagem da rocha.

Você pode chamar mais de um fragmento de código em resposta a um evento.

Agora, a função resizeRock() também é chamada em qualquer momento que a janela do navegador for redimensionada.

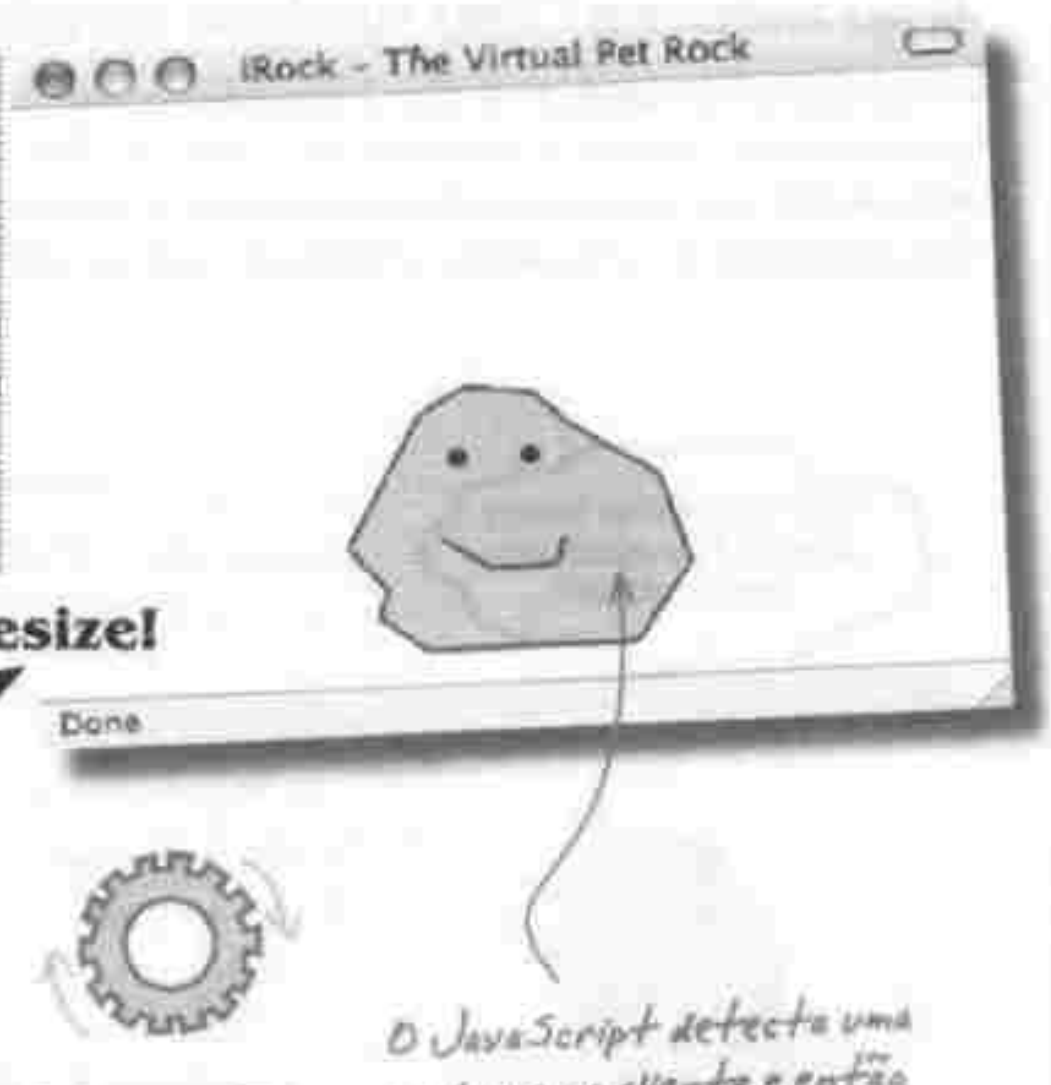
O tamanho da imagem do iRock se ajusta automaticamente sempre que o usuário alterar o tamanho da janela do navegador.

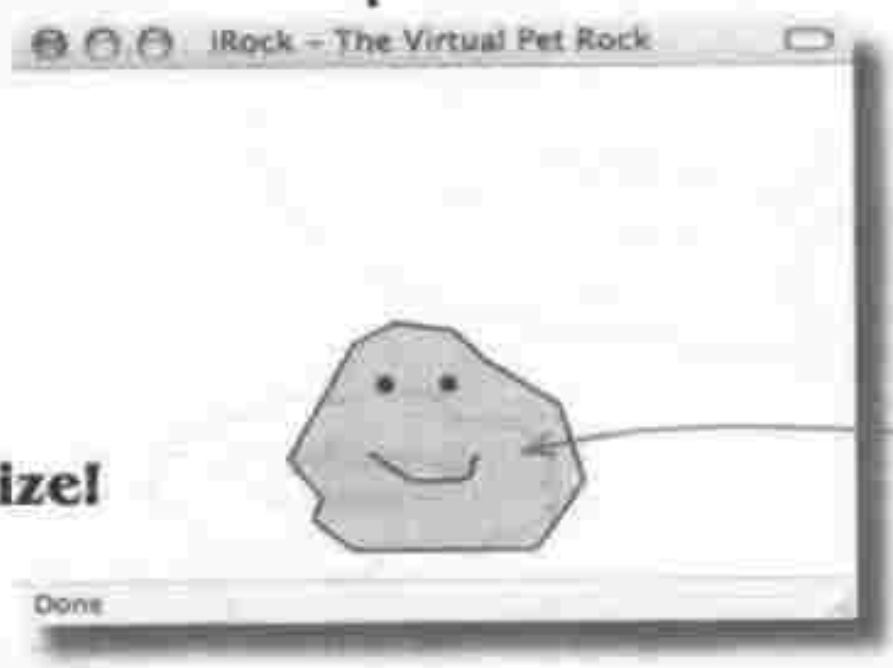
O evento onresize possibilita detecção e resposta às mudanças no tamanho da janela do navegador.



PRESTE ATENÇÃO!

Tenha cuidado ao redimensionar imagens no JavaScript.
Isso é especialmente importante ao transformar imagens pequenas em grandes. Às vezes, a qualidade da imagem pode ficar comprometida.



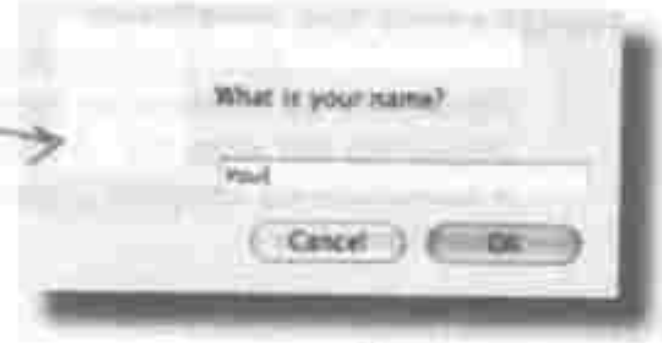


Querido, os clientes vão amar isso. Você tem mais alguma idéia legal?

Alan está amando os usuários que agora percebem que o iRock é imune às variações de tamanho do navegador. O tamanho do iRock não só se ajusta inicialmente para caber na janela do cliente do navegador, como se ajusta dinamicamente caso o usuário redimensione o navegador.

Já nos conhecemos? Como reconhecer o usuário

Os problemas de tamanho do iRock agora é coisa do passado... mas e quanto aos usuários que clicam no iRock mais de uma vez para não se sintam sozinhos? E quando eles voltam para a rocha depois de reiniciar o computador?



Como você não se lembra de mim? Não causei uma boa impressão?

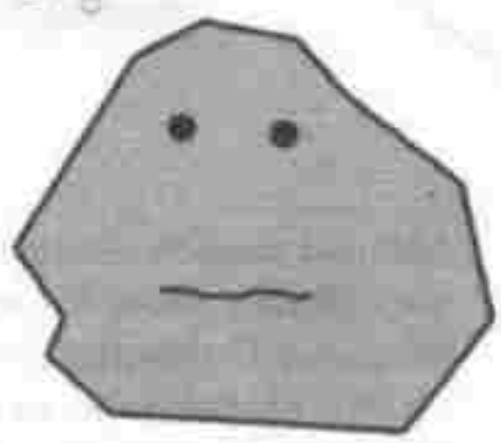
O usuário se encontra pela primeira vez com o iRock e digita seu nome.

O iRock responde com uma saudação pessoal – nasceu uma amizade!

O tempo passa e algumas coisas mudam...

Quem é Paul? Eu o conheço?

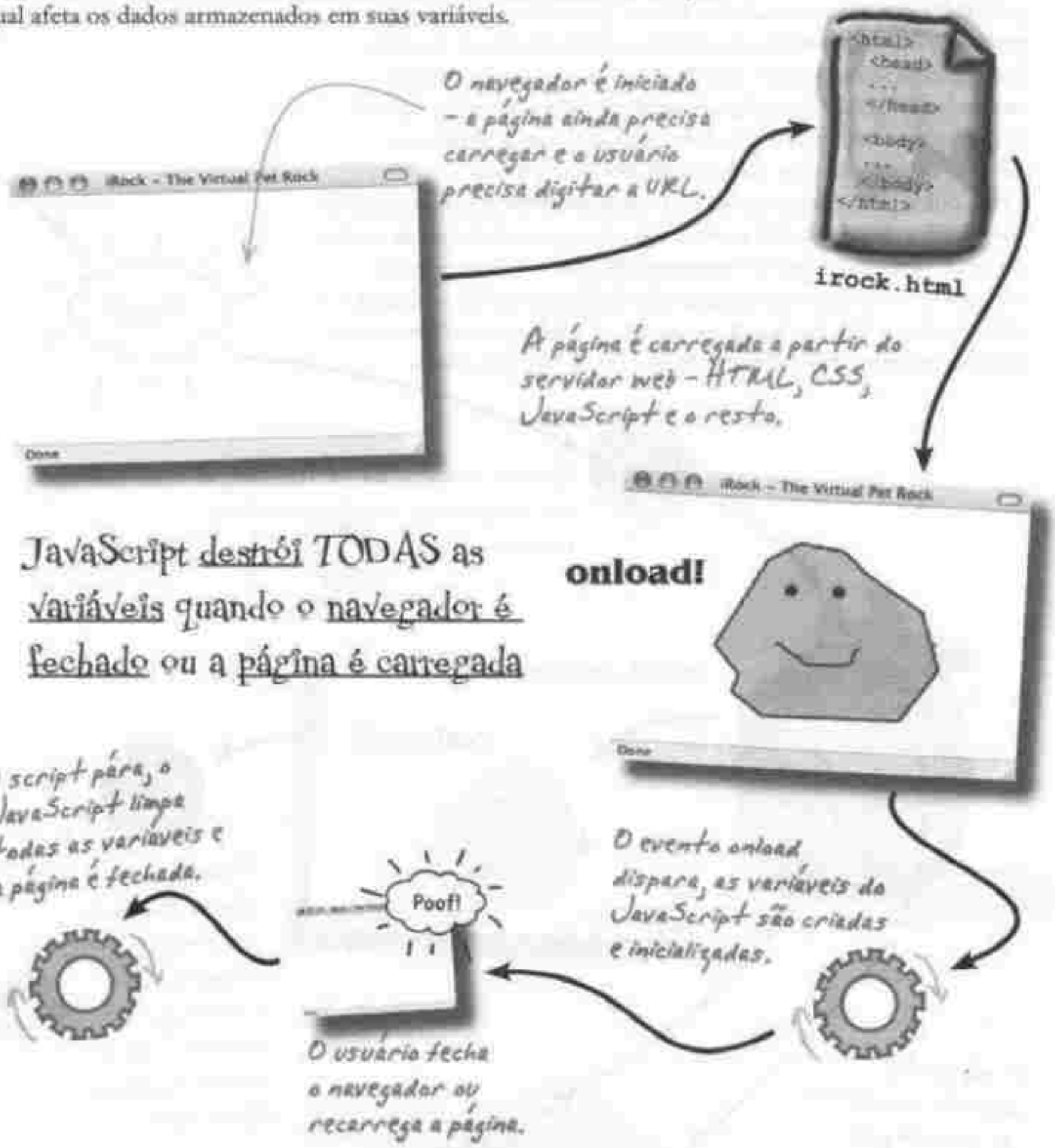
...o iRock não se lembra mais do usuário.



Embora o iRock tenha realmente se encontrado com seu dono em algum momento no passado, de algum modo ele esqueceu o nome do usuário...

Todo script tem vida útil

A perda de memória do iRock está relacionada à vida útil de um script, a qual afeta os dados armazenados em suas variáveis.



JavaScript destrói TODAS as variáveis quando o navegador é fechado ou a página é carregada

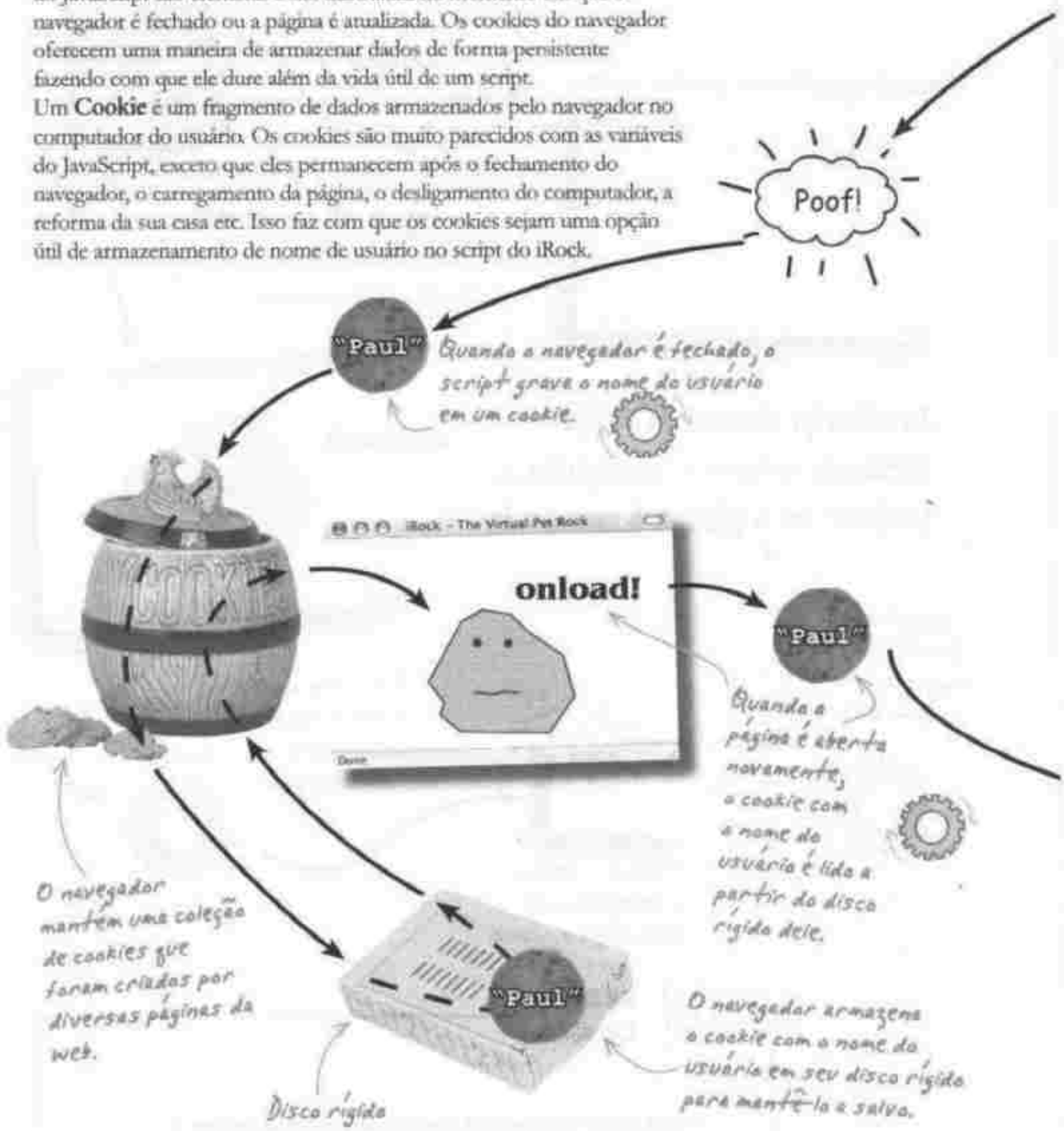
Poder do Cérebro

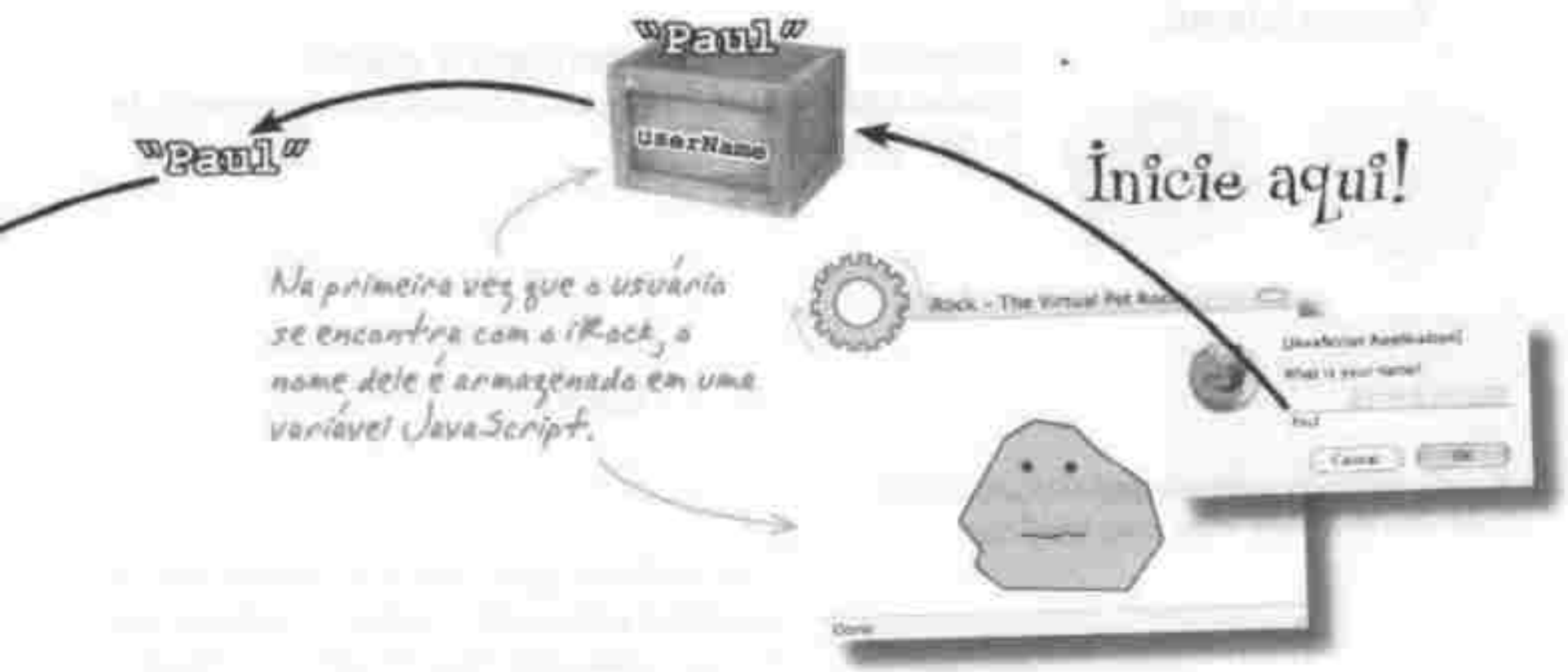
Então, como você tentaria ajustar o código para resolver o problema do iRock esquecer o nome do usuário?

Os cookies prolongam a vida útil do seu script

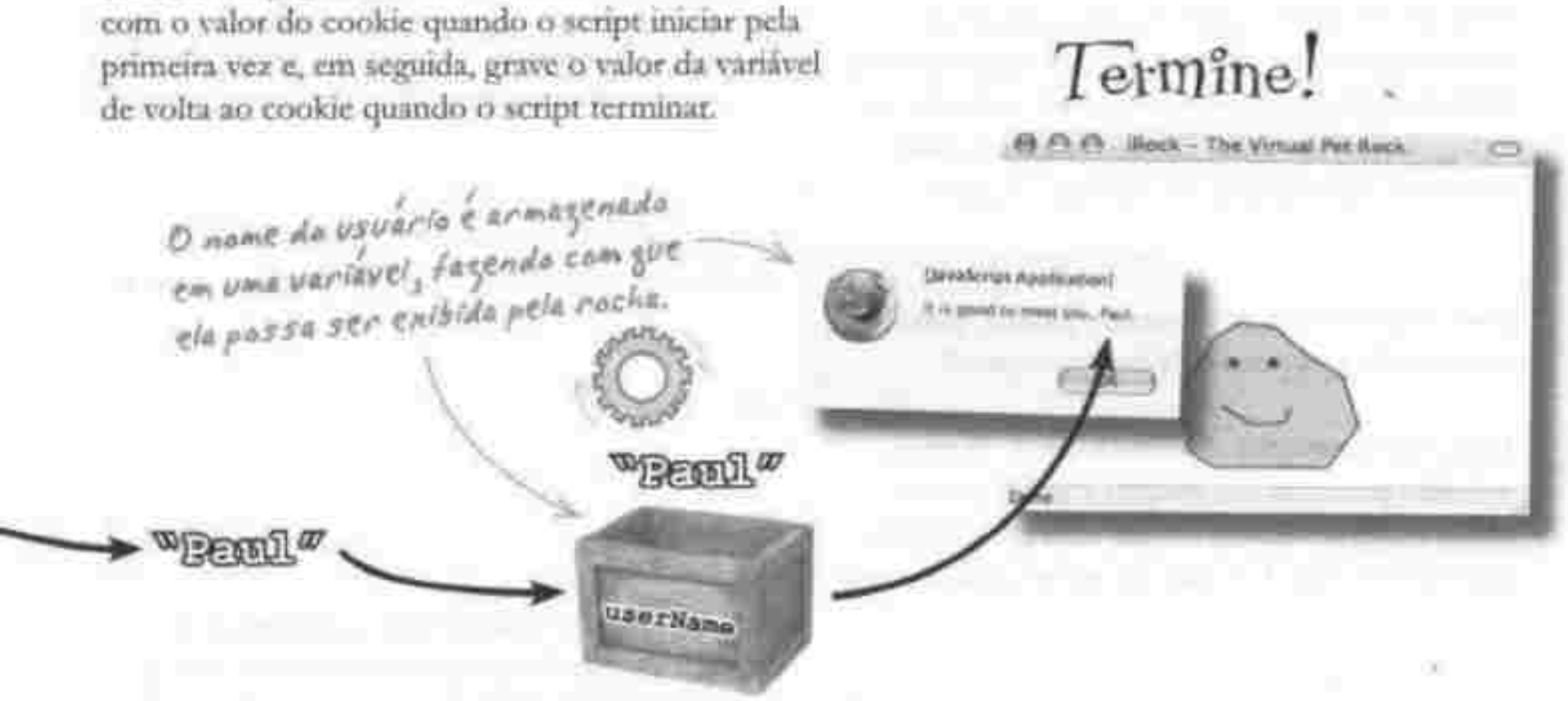
O problema que tivemos com o iRock tem um nome, e ele se chama **persistência**. Ou, na verdade, é a falta de persistência. Às vezes, você precisa de dados que nunca desaparecem. Infelizmente, as variáveis do JavaScript são efêmeras e são destruídas no momento em que o navegador é fechado ou a página é atualizada. Os cookies do navegador oferecem uma maneira de armazenar dados de forma persistente fazendo com que ele dure além da vida útil de um script.

Um **Cookie** é um fragmento de dados armazenados pelo navegador no computador do usuário. Os cookies são muito parecidos com as variáveis do JavaScript, exceto que eles permanecem após o fechamento do navegador, o carregamento da página, o desligamento do computador, a reforma da sua casa etc. Isso faz com que os cookies sejam uma opção útil de armazenamento de nome de usuário no script do iRock.





É bom usar uma variável JavaScript temporária enquanto o script está executando, mas você precisa armazená-la em um cookie de navegador caso queira que seu valor dure além de uma simples exibição de página da web. Inicialize a variável com o valor do cookie quando o script iniciar pela primeira vez e, em seguida, grave o valor da variável de volta ao cookie quando o script terminar.

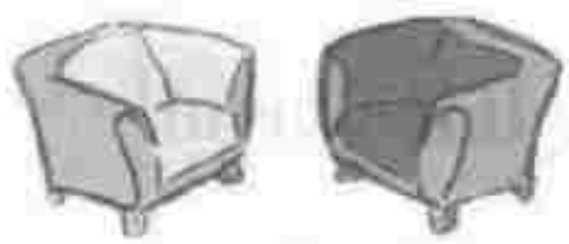


Aponte seu lápis

Escreva algum outro tipo de dados de página da web que você poderia armazenar de forma persistente usando cookies.

.....

Conversa informal



Conversa desta noite: **Variável e cookie examinam a importância do armazenamento de dados persistentess.**

Variável

Eu não entendo porque estamos conversando – você não tem nada a ver com JavaScript.

Cookie

Bem, você está quase certa. Eu realmente faço o que tenho a fazer sem a ajuda do JavaScript, mas também ainda tenho uma conexão importante com ele. Ofereço a maneira de armazenar dados de forma persistente. E, como você provavelmente já descobriu, o JavaScript não é bom em persistência.

Entendo para onde você está direcionado. Você acha que de algum modo eu sou uma opção inferior de armazenamento de dados, porque sou atacado toda vez que o navegador é fechado ou a página é recarregada. Ainda assim, sou *muito* acessível... ao contrário de algumas pessoas.

Inacessível? Estou sempre lá no navegador, pronto para ser chamado a qualquer momento.

Pode ser, mas você não vive em quartos apertados no meio de uma porção de outros cookies?

Sim... e?

Bem, se os rumores são verdadeiros, dá muito trabalho procurar um simples cookie... vocês todos estão armazenados em uma grande lista. Que sofrimento! É o que me refiro a inacessível.

Bem, sim, nós cookies somos armazenados em uma grande lista, mas temos nomes únicos. Não é assim tão difícil chegar até nós. Você só precisa saber como separar a lista para achar um determinado nome.

Aponte seu lápis
Solução

Escreva algum outro tipo de dados de página da web que você poderia armazenar de forma persistente usando cookies.

ID do usuário, conteúdo de carrinho de compras, localização geográfica, idioma

Variável

Certo, mas esse é o problema. Não há listas ou qualquer coisa envolvida no meu acesso. Apenas chame meu nome... e eu estarei lá![1]

Permanência é ótimo, mas isso não resolve os problemas do dia-a-dia. Se você pensar bem, nem todos aqueles dados precisam durar para sempre. Na verdade, é mais eficiente armazenar dados temporariamente e deixá-los ir embora ao terminar de usá-los. É onde eu entro. Eu sou a principal mídia de armazenamento temporário para dados de script.

Interessante, mas, para começar, como você acha que esses itens foram adicionados ao carrinho de compras? A maioria dos carrinhos de compras conta comigo para armazenar dados temporários ao longo das compras. Sou importante... talvez até mais importante. Mesmo que eu tenha tendência a esquecer coisas com um pouco mais de rapidez.

Acho que você está certo. Solucionamos problemas diferentes e nunca deveríamos competir. Embora eu tenha que admitir que ainda prefiro a minha facilidade de acesso à sua capacidade de armazenar coisas de forma persistente.

Que conversa?

Cookie

Você não precisa cantar. Eu entendi. Mas a verdadeira questão é que quando você armazena alguma coisa em mim, eu sempre me lembro. Não importa se o navegador estiver fechado ou se a página for carregada. Sou permanente... a menos que o usuário opte por limpar todos os cookies. Mas isso é uma outra questão.

Que seja... acho que você está subestimando a importância do armazenamento de dados permanentes. Você nunca se impressionou com a magia de retornar a um carrinho de compras dias depois de navegar, só para descobrir que tudo ainda está lá? Esse é o tipo de mágica que torno realidade!

Isso está começando a parecer como se pudéssemos nos complementar. Eu sempre lhe vi como uma inimiga.

Você não poderia deixar de dar uma alfinetada, não é? Vou me embora e descansarei tranqüilo, sabendo que tão logo essa página seja virada, você se esquecerá de toda essa conversa.

Não disse?

Por que você não armazena dados persistentes da web no servidor?

Você não precisa do servidor para pequenos fragmentos de informação, como um nome de usuário.

Ah, o servidor. Sim, o servidor é uma opção viável para armazenar dados de forma persistente, mas pode ser exagerado para pequenos fragmentos de informação. Armazenar dados no servidor requer trabalho de programação, junto com uma mídia de armazenamento, como um banco de dados. Programação de servidor e armazenamento de banco de dados é muita coisa para armazenar um fragmento de dados que você deseja persistir para um simples script de cliente, como o nome de usuário no script do iRock.

Cookies permitem armazenar dados de forma persistente no cliente sem envolver o servidor. Não só isso, mas os usuários possuem a capacidade de limpar os dados dos cookies no navegador, se ele quiser se ver livre das informações que as páginas da web armazenaram de forma persistente. Isso não é possível com os dados armazenados no servidor.

JavaScript

+

Cookie

=

Prático armazenamento de dados <u>persistentes</u> no cliente!

Os cookies têm um nome e armazenam um valor... e podem expirar

Um cookie armazena um simples fragmento de dados sob um único nome, como uma variável. Ao contrário de uma variável, contudo, um cookie pode ter uma data de validade. Quando essa data chega, o cookie é destruído. Então, na realidade os cookies não são verdadeiramente permanentes, eles só vivem **mais** do que as variáveis. Você pode criar um cookie sem uma data de validade, mas ele agirá como uma variável JavaScript – ele será apagado quando o navegador for fechado.

Os cookies são armazenados no computador de um usuário como uma longa string de texto associada ao site da web (ou domínio). Cada cookie é separado um do outro por um ponto-e-vírgula (;). Ponto-e-vírgula é a chave para a leitura da lista de cookies e extração de um único cookie.

Nome

O nome único do cookie.

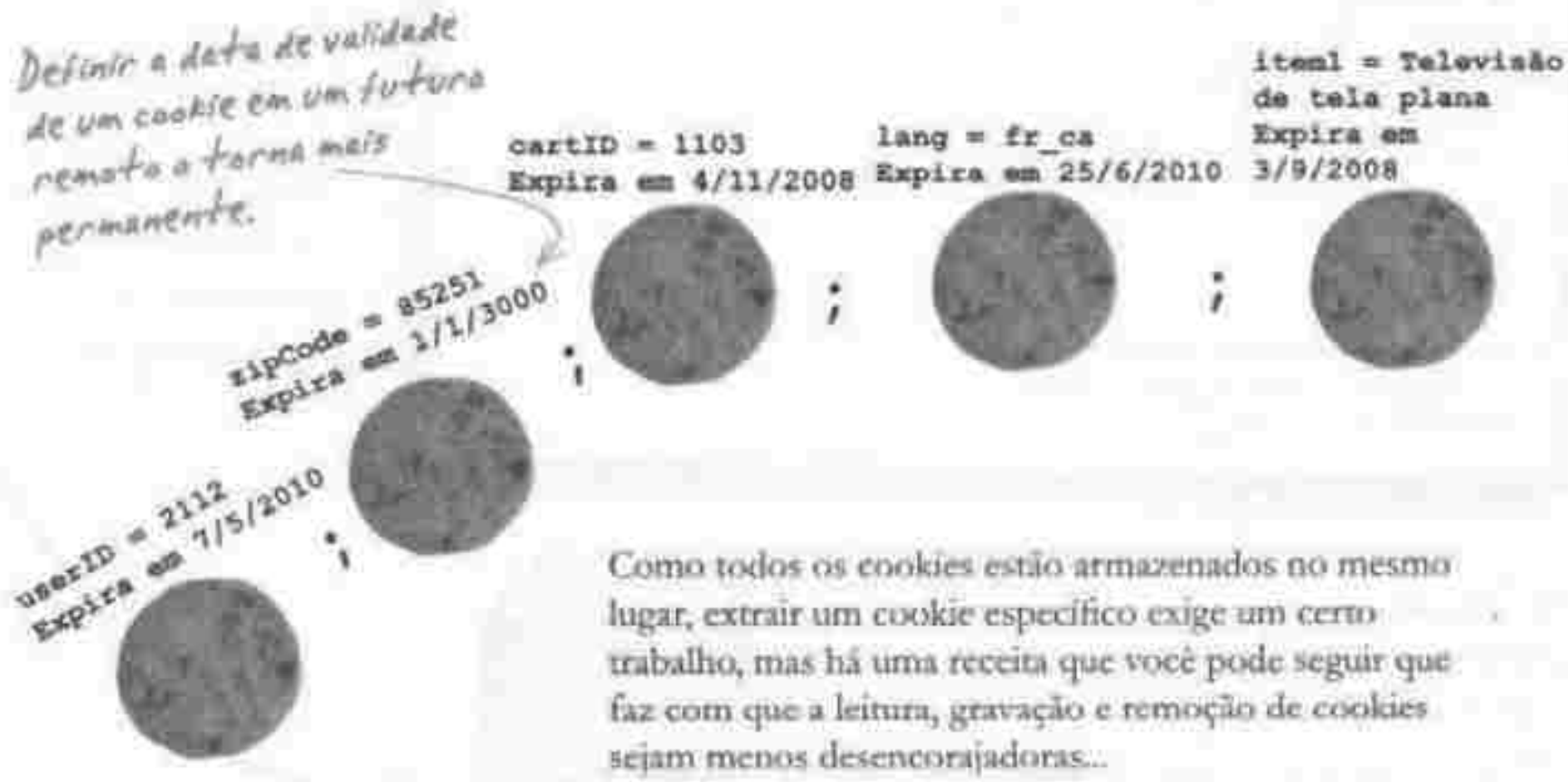
Valor

O valor armazenado no cookie.

Data de validade

A data em que o cookie expira... e morre.

Definir a data de validade de um cookie em um futuro remoto a torna mais permanente.



Como todos os cookies estão armazenados no mesmo lugar, extrair um cookie específico exige um certo trabalho, mas há uma receita que você pode seguir que faz com que a leitura, gravação e remoção de cookies sejam menos desencorajadoras...

JavaScript pronto para assar

Eis o código para três funções auxiliares de cookie que permite gravar, ler e excluir cookies com facilidade. Às vezes, a abordagem mais inteligente é velejar no trabalho dos outros. Pegue essa receita (disponível para download em *http://www.headfirstlabs.com/books/hfjs/*) e aproveite ao máximo as funções dos cookies feitos em casa.

Tudo bem se você não entende nada disso agora... você entenderá até o momento em que terminar este livro.

A data de validade é expressa como o número de dias que o cookie deve existir.

```
function writeCookie(name, value, days) {
  // Por padrão, não há expiração. Então, o cookie é temporário
    var expires = "";

  // Especificar um número de dias torna o cookie persistente
  if (days) {
    var date = new Date();
    date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
    expires = "; expires=" + date.toGMTString();
  }

  // Atribua nome, valor e data de validade ao cookie
  document.cookie = name + "=" + value + expires + "; path=/";
}

function readCookie(name) {
  // Busque o cookie especificado e retorne seu valor
  var searchName = name + "=";
  var cookies = document.cookie.split(';');
  for(var i=0; i < cookies.length; i++) {
    var c = cookies[i];
    while (c.charAt(0) == ' ')
      c = c.substring(1, c.length);
    if (c.indexOf(searchName) == 0)
      return c.substring(searchName.length, c.length);
  }
  return null;
}

function eraseCookie(name) {
  // Apague o cookie especificado
  writeCookie(name, "", -1);
}
```

Essa data de validade é calculada ao converter o número de dias em milésimos de segundo e, em seguida, adicioná-lo esse número ao momento atual.

A lista de cookies é dividida em cookies individuais separados por ponto-e-vírgula.

Você exclui um cookie, gravando-o sem nenhum valor e sem data de validade expirada (-1 dia).

cookie.js

Crie um arquivo vazio, nomeie-o cookie.js e adicione este código ao arquivo.

Os arquivos que só contêm código JavaScript normalmente são nomeados com uma extensão de arquivo .js.

Seu JavaScript pode residir FORA de sua página da web.

Quando o código JavaScript é armazenado em seu próprio arquivo, você precisa **importá-lo** para qualquer página da web na qual planeje usar o código. Então, no caso das funções de cookie em cookie.js, você precisará importá-los para a sua página iRock.html. Isso é feito com uma variação da tag <script>:

Lembre-se de encerrar a tag com </script>.

```
<script type="text/javascript" src="cookie.js"></script>
```

O tipo do código de script é sempre text/javascript para código JavaScript.

O nome do arquivo que contém o código de script normalmente termina com .js.

FAÇA ASSIM! Faça essa adição à sua página iRock.html e certifique-se de que o arquivo cookie.js está no mesmo diretório.

Quando você importa um código de script externo para a página, todo o código JavaScript no arquivo de script é inserido dentro da tag <script> no código HTML, como se você tivesse colocado o código diretamente na página da web. Sempre que tiver um código que possa ser usado em mais de uma página, é uma boa idéia colocá-lo em um arquivo externo e importar o arquivo para as páginas.

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>
    <script type="text/javascript" src="cookie.js"></script>
    <script type="text/javascript">
      var userName;

      function resizeRock() {
        document.getElementById("rockImg").style.height =
          (document.body.clientHeight - 100) * 0.9;
      }

      function greetUser() {
```

O código de script importado é colocado na página quando ela é recarregada.

Exercício

Escreva por que motivo é uma boa idéia organizar código reutilizável em um arquivo externo.

.....

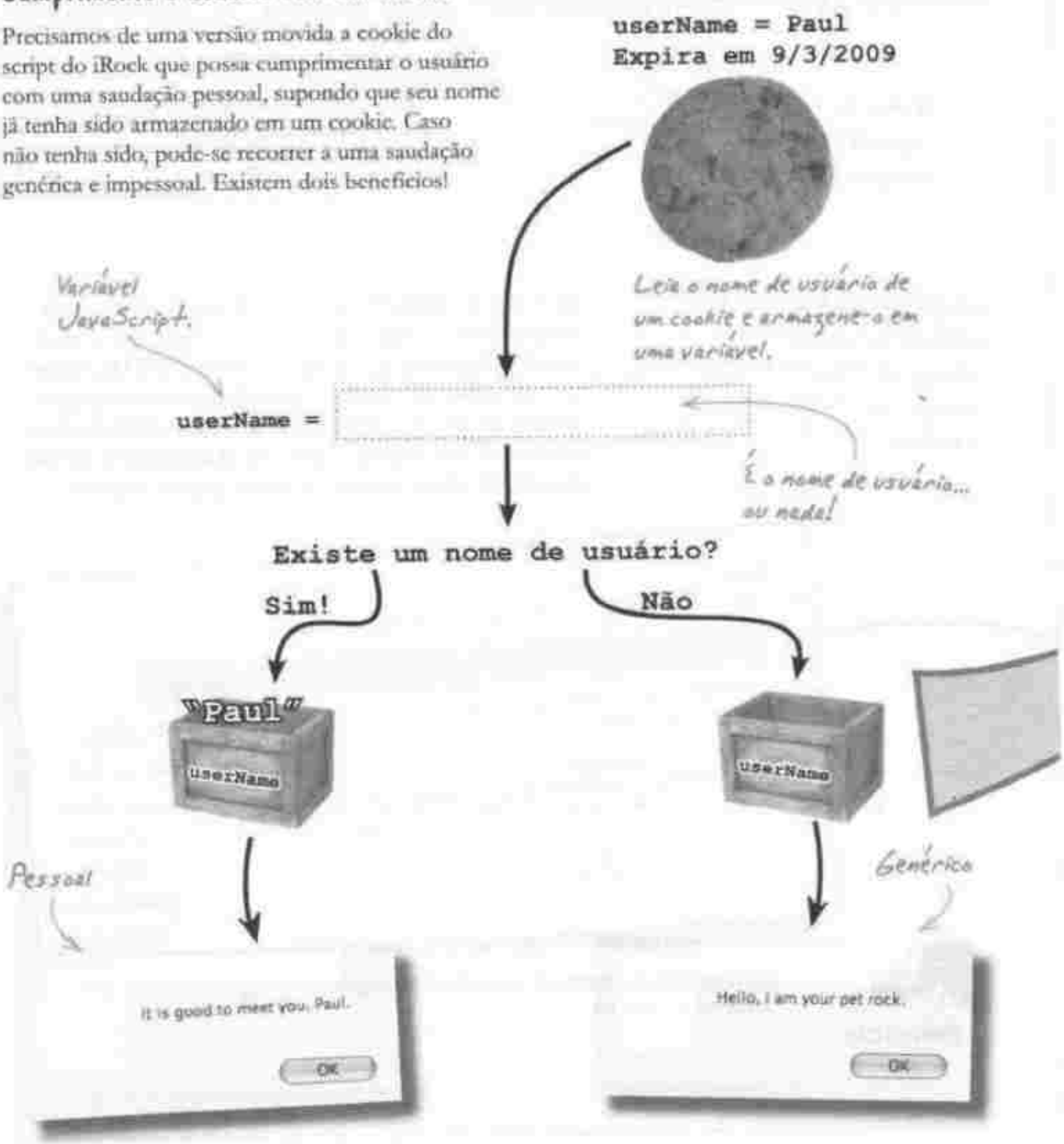
Escreva por que motivo é uma boa idéia organizar código reutilizável em um arquivo externo.

Solução do Exercício

Reutilização de código, mais fácil para manter já que ele está em um único lugar, melhor organização

Cumprimente o usuário com um cookie

Precisamos de uma versão movida a cookie do script do iRock que possa cumprimentar o usuário com uma saudação pessoal, supondo que seu nome já tenha sido armazenado em um cookie. Caso não tenha sido, pode-se recorrer a uma saudação genérica e impessoal. Existem dois benefícios!



O código de perto

A função greetUser() é responsável pela saudação do usuário quando a página é carregada pela primeira vez.

O cookie de nome de usuário do iRock precisa de um título descritivo para que não haja equívoco, caso você precise adicionar mais cookies ao script.

Não é uma adição... é concatenação de string!

```
function greetUser() {
  userName = readCookie("irock_username");
  if (userName)
    alert("Olá " + userName + ", senti sua falta.");
  else
    alert('Olá, eu sou sua rocha de estimação.');
}
```

O nome do usuário é lido a partir do cookie e armazenado na variável userName.

O nome do usuário está vazio, o que significa que o cookie não existia e que temos que escolher uma saudação genérica.

Se o nome de um usuário realmente não existia no cookie, mostre uma saudação pessoal.

greetUser() agora é acionado pelo cookie

O que realmente está acontecendo na função greetUser() é um dueto de dados interpretado por uma variável e um cookie. O nome do usuário é lido a partir do cookie e armazenado na variável. Mas você não pode contar que o cookie mantenha um nome... e se essa for a primeira vez que o script é executado e o usuário nunca tivesse digitado um nome? Eis o porquê de o código verificar se a variável realmente recebeu um nome do cookie – este é o teste que determina se a saudação é pessoal ou genérica.

Lembre-se de definir o cookie também

Ler o cookie do iRock é muito bom, mas você ainda tem que definir o cookie antes de qualquer coisa. A gravação do cookie deve ocorrer na função touchRock(), que é chamada quando o usuário clica na imagem da rocha. A função touchRock() imediatamente avisa o usuário para digitar um nome – agora ela precisa gravar esse nome em um cookie, após ele ser digitado.

O estilo do código de perto

Se não houver um nome de usuário, precisamos obtê-lo do usuário.

Primeiro, o mais importante – veja se há um nome de usuário.

Se houver um nome de usuário, agradeça-o pessoalmente pela atenção.

```
function touchRock() {
  if (userName) {
    alert("Eu gosto de atenção, " + userName + ". Obrigada.");
  }
  else {
    userName = prompt("Qual é o seu nome?", "Digite seu nome
aqui.");
    if (userName) {
      alert("Que bom conhecer você, " + userName + ".");
      writeCookie("irock_username", userName, 5 * 365);
    }
  }
  document.getElementById("rockImg").src = "rock_happy.png";
  setTimeout("document.getElementById('rockImg').src = 'rock.
png';",
    5 * 60 * 1000);
}
```

Confira para certificar-se de que o usuário de fato digitou um nome.

Existe um nome, então cumprimente o usuário e, em seguida, grave o nome em um cookie.

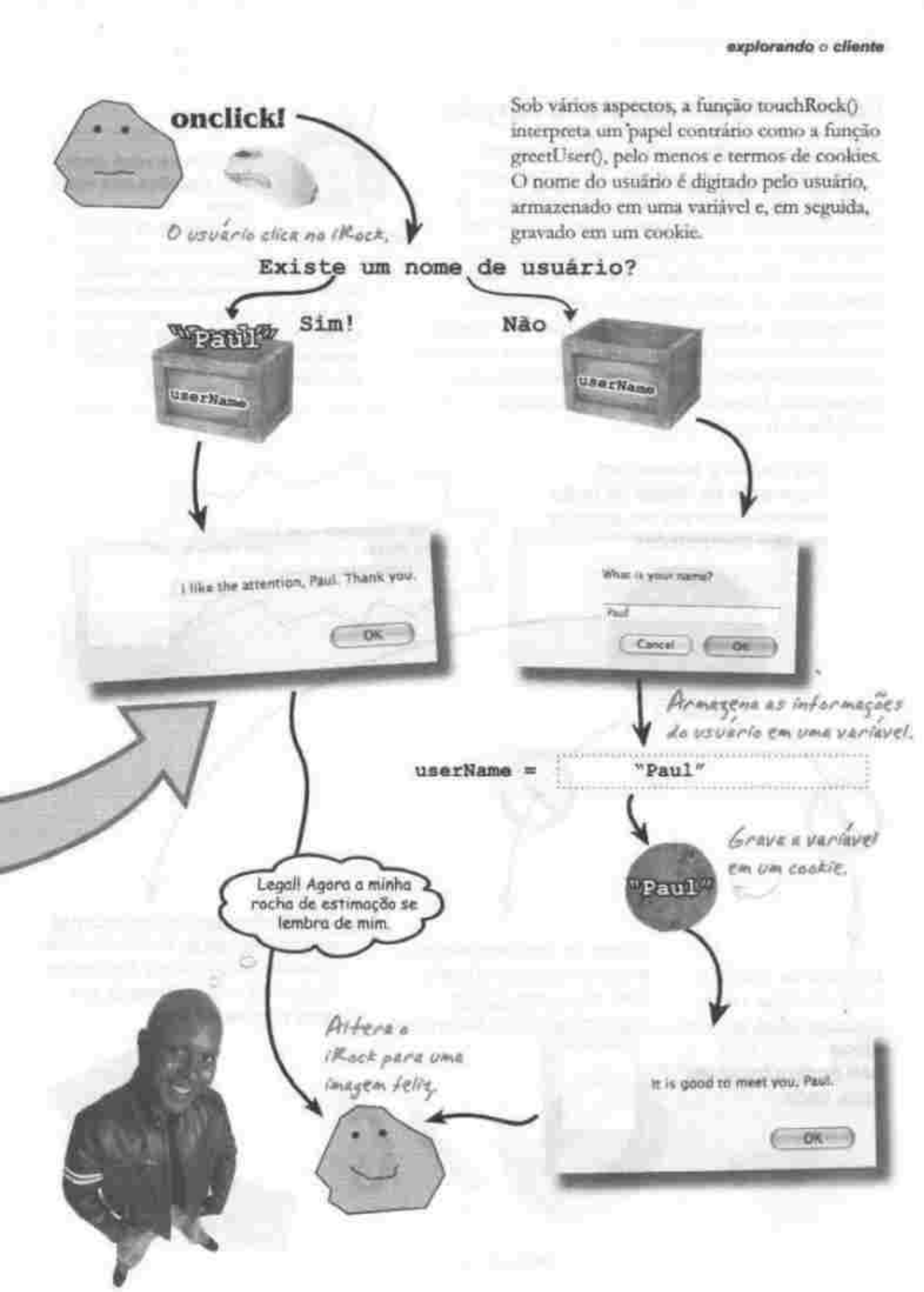
Esse é o mesmo nome de cookie usado para ler o cookie. Cookies não usam camelCase, eles são mais parecidos com IDs HTML.

O valor do nome de usuário é gravado a partir de uma variável.

Mantenha o cookie do nome do usuário por aproximadamente 5 anos.

Essa função é chamada quando a imagem da rocha for clicada.

```
<img id="rockImg" src="rock.png" onclick="touchRock();" />
```


onclick!
O usuário clica no iRock.
Existe um nome de usuário?
Sim!
"Paul"
userName
Não
userName
I like the attention, Paul. Thank you.
OK
What is your name?
Paul
Cancel
OK
Armazena as informações do usuário em uma variável.
userName = "Paul"
Grava a variável em um cookie.
"Paul"
Legal! Agora a minha rocha de estimação se lembra de mim.
Altera o iRock para uma imagem feliz.
It is good to meet you, Paul.
OK

Os cookies afetam a segurança do navegador

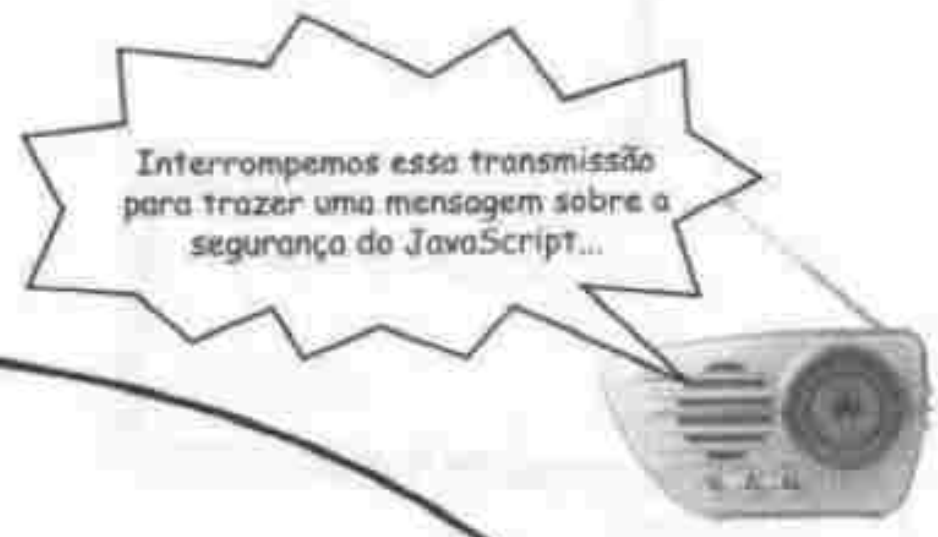
Embora a maioria dos usuários do iRock estejam impressionados com os cookies como a cura para a perda de memória, alguns questionam seus riscos de segurança. Essa é uma excelente questão, já que em muitos casos os dados pessoais ficam armazenados nos cookies. Mas a realidade é que os cookies **não** representam um risco à segurança... pelo menos não em termos de acesso a dados confidenciais que estejam armazenados em seu computador. No entanto, os cookies não são considerados seguros para armazenamento, portanto não é uma boa idéia armazenar dados **confidenciais** em um cookie.

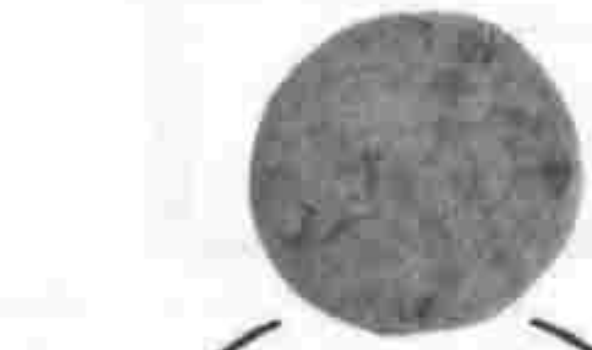
PRESTE ATENÇÃO!

Só porque você pode, não significa que você deve.

Embora você possa armazenar qualquer coisa em um cookie, eles não são incrivelmente seguros em termos de como armazenam dados. Então, não é uma boa idéia armazenar dados confidenciais em um cookie.

Um cookie é apenas um fragmento de dados de texto armazenados em um arquivo em seu computador.

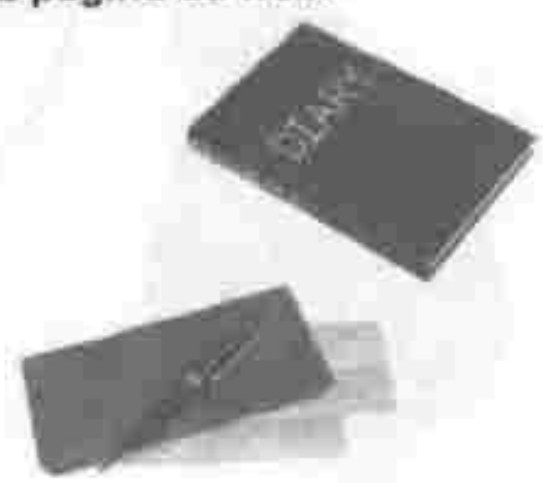




Embora os cookies normalmente sejam armazenados em um disco rígido, eles não podem tocar em mais nada.

Como os cookies não são programas executáveis, eles não conseguem disseminar vírus ou worms.

Os cookies podem armazenar dados pessoais, mas somente quando os usuários digitarem dados intencionalmente em uma página da web.



Não existem

perguntas idiotas

P: Os cookies são sempre armazenados no disco rígido do usuário?

R: Não. Mas o disco rígido é onde a grande maioria dos navegadores armazena cookies. Nem todo navegador tem acesso a um disco rígido. Por exemplo, alguns dispositivos móveis usam memória especial, em vez de discos rígidos para armazenamento de dados persistentes. Neste caso, o navegador usa a memória persistente para armazenar cookies. Mesmo assim, do ponto de vista do navegador (e seus scripts), os cookies lembram de seus valores independentemente de como eles são armazenados nos bastidores.

P: Como sei se o meu nome de cookie é único?

R: Nomes de cookies só precisam ser únicos dentro de uma determinada página da web, porque os cookies são armazenados de acordo com a página que os criaram, inclusive o site web da página. Isso significa que a página faz parte efetivamente do nome do cookie, pelo menos em termos de singularidade. Conclusão: certifique-se de que seus cookies são únicos dentro de uma determinada página ou site.

P: Os dados dos cookies são compartilhados entre navegadores diferentes?

R: Não. Cada navegador mantém seu próprio banco de dados de cookies. Portanto, o conjunto de cookies no Internet Explorer não estará visível para o Firefox ou o Opera.

P: Se os cookies são tão úteis, por que eu armazenaria dados no servidor?

R: Primeiramente, os cookies só servem para armazenar blocos de texto relativamente pequenos (menos de 4 Kb). Essa é uma das limitações importantes dos cookies. Ainda mais importante é o fato de que cookies não são particularmente eficientes, o que significa que você não precisaria ler e gravar constantemente milhares de dados neles. É aí que um verdadeiro banco de dados entra em cena, e bancos de dados normalmente residem no servidor. Então, embora os cookies sejam ótimos para armazenar pequenos fragmentos de dados que não precisam necessariamente de armazenamento no servidor em um banco de dados, eles não são uma solução para todas as necessidades de seus dados da web. E eles também não são exatamente ideais para armazenar dados confidenciais, já que foram criados sem a preocupação da segurança.

P: Existe alguma forma de criar cookies verdadeiramente permanentes?

R: Não. Goste ou não, cada cookie possui uma data de validade. A idéia por detrás de um cookie não é tanto a de um verdadeiro armazenamento de dados em longo prazo, tanto quanto ele é um meio de preservar dados em médio prazo. Em outras palavras, os cookies são ótimos para armazenar dados durante dias, semanas e meses. Se você estiver lidando com dados que precisam sobreviver por longos períodos de tempo, talvez precise armazená-los no servidor. Não é que um cookie não possa armazenar dados durante anos, é que é improvável que o usuário não vá fazer um upgrade no computador, reinstalar o navegador ou remover dados de cookies.

P: Chega de cookies... existe alguma desvantagem em armazenar código JavaScript em um arquivo externo?

R: Não, exatamente. No entanto, lembre-se de que o objetivo do código externo é facilitar o compartilhamento e manter o código quando ele precisa ser usado em mais de uma página da web. Se você estiver lidando com um código que só aparece em uma única página, realmente não se beneficiará muito movendo-o para um arquivo externo. Ou seja, a não ser que a página esteja particularmente desarrumada e você precise separar o código para sua própria saúde mental.

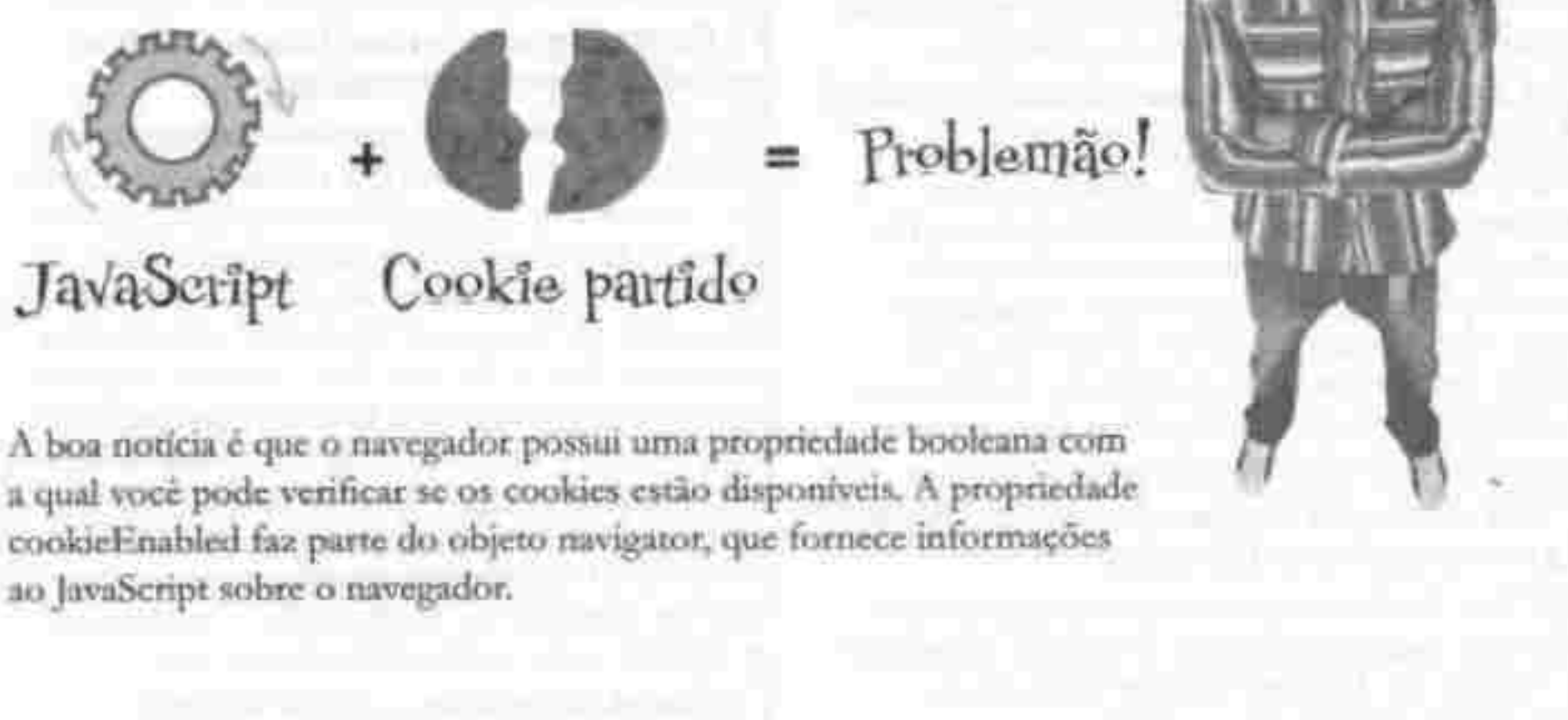
Pontos de Bala

- Um cookie é um fragmento de dados de texto armazenados pelo navegador no computador do usuário.
- Mover o código de script para um arquivo externo é uma maneira útil de tornar o código mais reutilizável e de fácil manutenção.
- Os cookies permitem que scripts armazenem dados que sobrevivem além de uma simples sessão da web. Todo cookie possui uma data de validade e, depois de expirada, o cookie é destruído pelo navegador.
- Cookies não conseguem acessar o disco rígido de um usuário ou disseminar vírus, mas eles são capazes de armazenar dados pessoais digitados em páginas da web.

Um mundo sem cookies

Cada usuário que vai embora é menos um cliente... isso é inaceitável!

Seja a preocupação com segurança ou navegadores limitados, poucos usuários do iRock não são capazes de se beneficiar do iRock movido a cookie, porque os cookies não estão disponíveis em seus navegadores. Isso apresenta um grande problema, porque o script do iRock parte do princípio de que todos possuem suporte a cookies. O iRock depende dos cookies para memorizar, mas é inaceitável não deixar ao menos os usuários de poucos cookies saberem o que eles estão perdendo em relação à experiência total do iRock.



A boa notícia é que o navegador possui uma propriedade booleana com a qual você pode verificar se os cookies estão disponíveis. A propriedade cookieEnabled faz parte do objeto navigator, que fornece informações ao JavaScript sobre o navegador.

Aponte seu lápis

Escreva o código que falta para verificar o suporte a cookies nas funções greetUser() e touchRock(). Além disso, adicione o código que falta em touchRock() para permitir que o usuário saiba que os cookies não estão disponíveis.

```
function greetUser() {

  ..............................................................
    userName = readCookie("irock_username");
  if (userName)
    alert("Olá " + userName + ", senti sua falta.");
  else
    alert('Olá, eu sou sua rocha de estimação.');
}

function touchRock() {
  if (userName) {
    alert("Eu gosto de atenção, " + userName + ". Obrigada.");
  }
  else {
    userName = prompt("Qual é o seu nome?", "Digite seu nome aqui.");
    if (userName) {
      alert("Que bom conhecer você, " + userName + ".");

      ..............................................................
        writeCookie("irock_username", userName, 5 * 365);
      else

      ..............................................................
    }
  }
  document.getElementById("rockImg").src = "rock_happy.png";
  setTimeout("document.getElementById('rockImg').src = 'rock.png';",
    5 * 60 * 1000);
}
```

Aponte seu lápis
Solução

Escreva o código que falta para verificar o suporte a cookies nas funções greetUser() e touchRock(). Além disso, adicione o código que falta em touchRock() para permitir que o usuário saiba que os cookies não estão disponíveis.

Se o suporte a cookie está disponível, leia o nome de usuário a partir do cookie do iRock.

```
function greetUser() {
  if (navigator.cookieEnabled)
  userName = readCookie("irock_username");
if (userName)
  alert("Olá " + userName + ", senti sua falta.");
else
  alert('Olá, eu sou sua rocha de estimação.');
}

function touchRock() {
  if (userName) {
    alert("Eu gosto de atenção, " + userName + ". Obrigada.");
  }
  else {
    userName = prompt("Qual é o seu nome?", "Digite seu nome aqui.");
    if (userName) {
      alert("Que bom conhecer você, " + userName + ".");
      if (navigator.cookieEnabled)
        writeCookie("irock_username", userName, 5 * 365);
      else
      alert("Sorry. Cookies aren't supported/enabled in your browser. I won't
    remember you later.");
  }
  document.getElementById("rockImg").src = "rock_happy.png";
  setTimeout("document.getElementById('rockImg').src = 'rock.png';",
    5 * 60 * 1000);
}
```

Se os cookies são suportados, grave o nome de usuário do cookie.

Informe ao usuário que a ausência de suporte a cookies limitará o iRock.

Faça com que as funções de sua página iRock.html tenham essa aparência... em seguida, faça o teste.

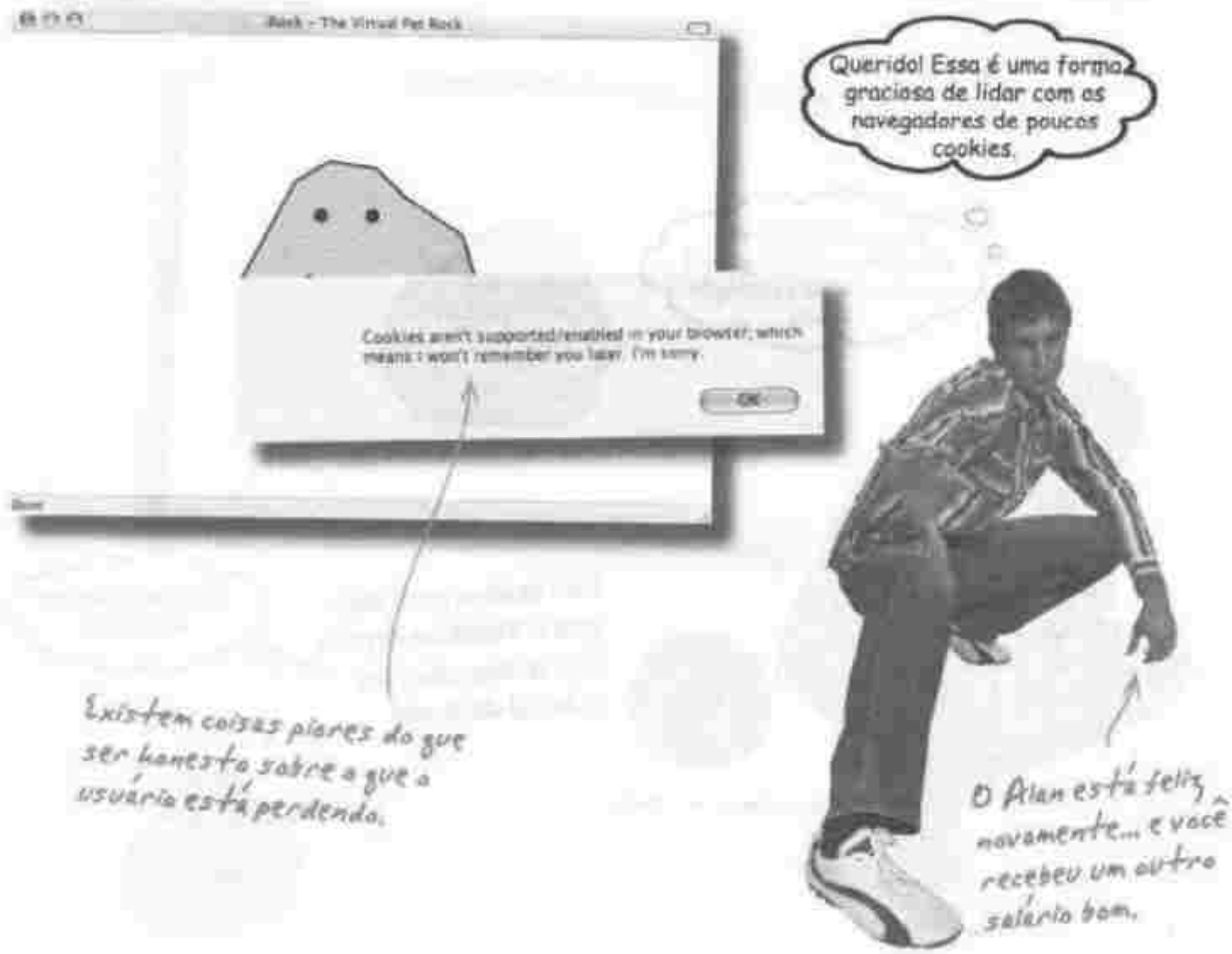
Não existem
perguntas idiotas

P: Você pode verificar o suporte a cookie de cliente com base no tipo ou na versão do navegador?

R: A detecção é uma ladeira escorregadia de script que, em última análise, leva a resultados duvidosos. Você não pode realmente confiar que os navegadores falem sobre eles mesmos quando se trata de informação de versão, o que torna a propriedade navigator.cookieEnabled a única maneira verdadeiramente confiável de verificar o suporte a cookie.

Fale com os usuários... é melhor do que nada

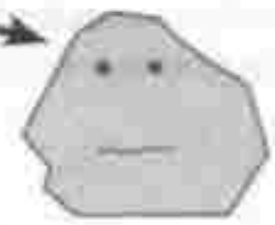
Embora não haja uma boa maneira de simular cookies quando eles não estão disponíveis, suspender as más notícias ao usuário vale muito em termos da satisfação dele.



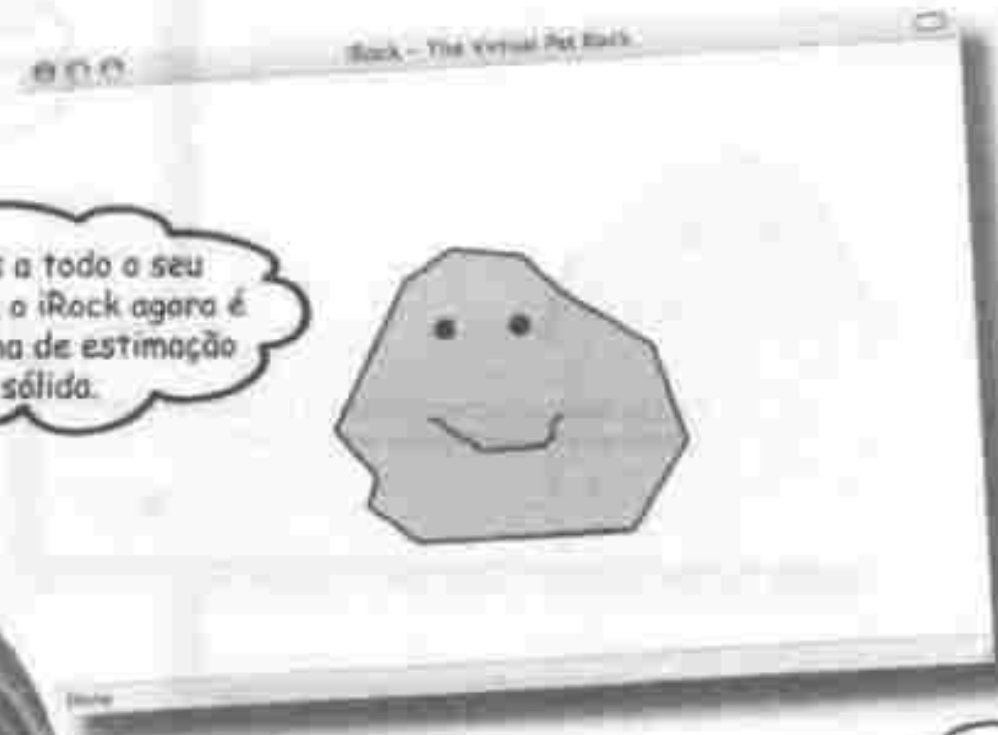
Um iRock digno do rei JavaScript

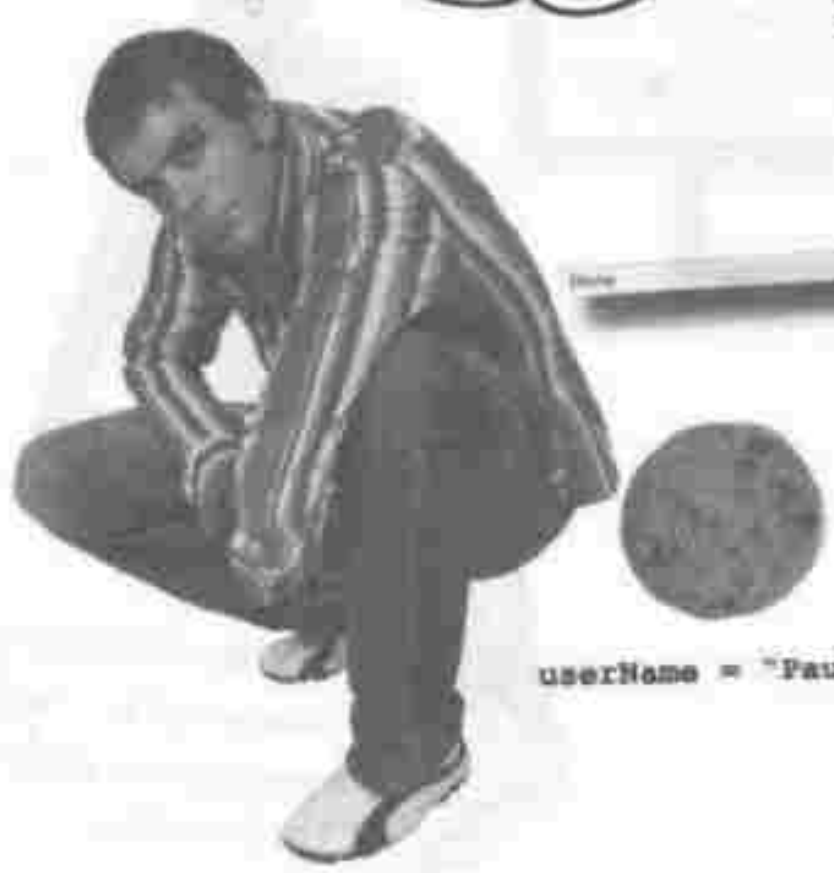
Você desgastou muito os sapatos do seu JavaScript passeando por todo o código necessário para fazer do iRock um sucesso. Mas com uma pequena ajuda do cliente, o iRock agora está emocionalmente mais real, perdeu suas inconsistências de tamanho e melhorou sua memória!

A métrica do navegador e as propriedades de estilo do CSS dá ao iRock a capacidade de se adaptar ao seu ambiente.



Temporizadores ampliam a limitada variação emocional do iRock.

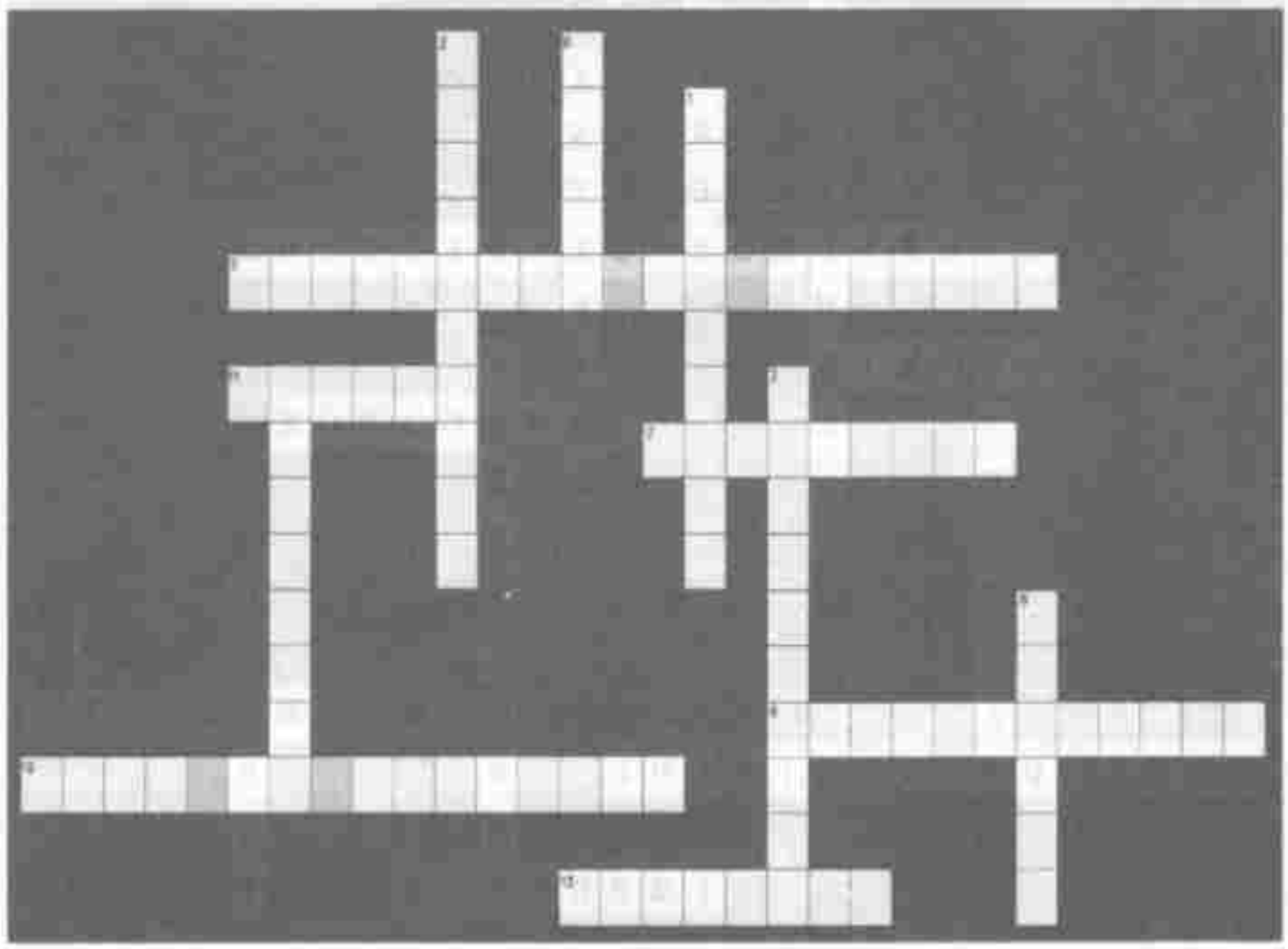




Os cookies permitem que o iRock se lembre dos dados além da vida útil do script.

Cruzadas do JavaScript

Dedique um tempo para relaxar e dar ao lado direito do seu cérebro algo para fazer. São suas clássicas palavras cruzadas; todas as palavras da solução encontram-se neste capítulo.



Horizontal

5. A milésima parte de um segundo.
7. Eu sou responsável pelo gerenciamento da lista de cookies.
8. Um mecanismo JavaScript que permite executar um código, depois de transcorrido um certo período de tempo.
10. Um cookie possui um nome, um valor e uma .....
11. Use um desses para armazenar um fragmento de informação no cliente que você pode precisar mais tarde.
12. O que você faz ao consultar o código JavaScript externo de uma página da Web.

Vertical

1. Esse tipo de temporizador executa um fragmento de código repetidamente.
2. Essa função permite criar um temporizador de disparo único.
3. Quando os dados permanecem após um script terminar a execução, é considerado .....
4. Quando a janela do navegador é redimensionada, o evento..... é acionado.
6. Um outro nome para um navegador Web.
9. Os cookies não são capazes de disseminá-los.

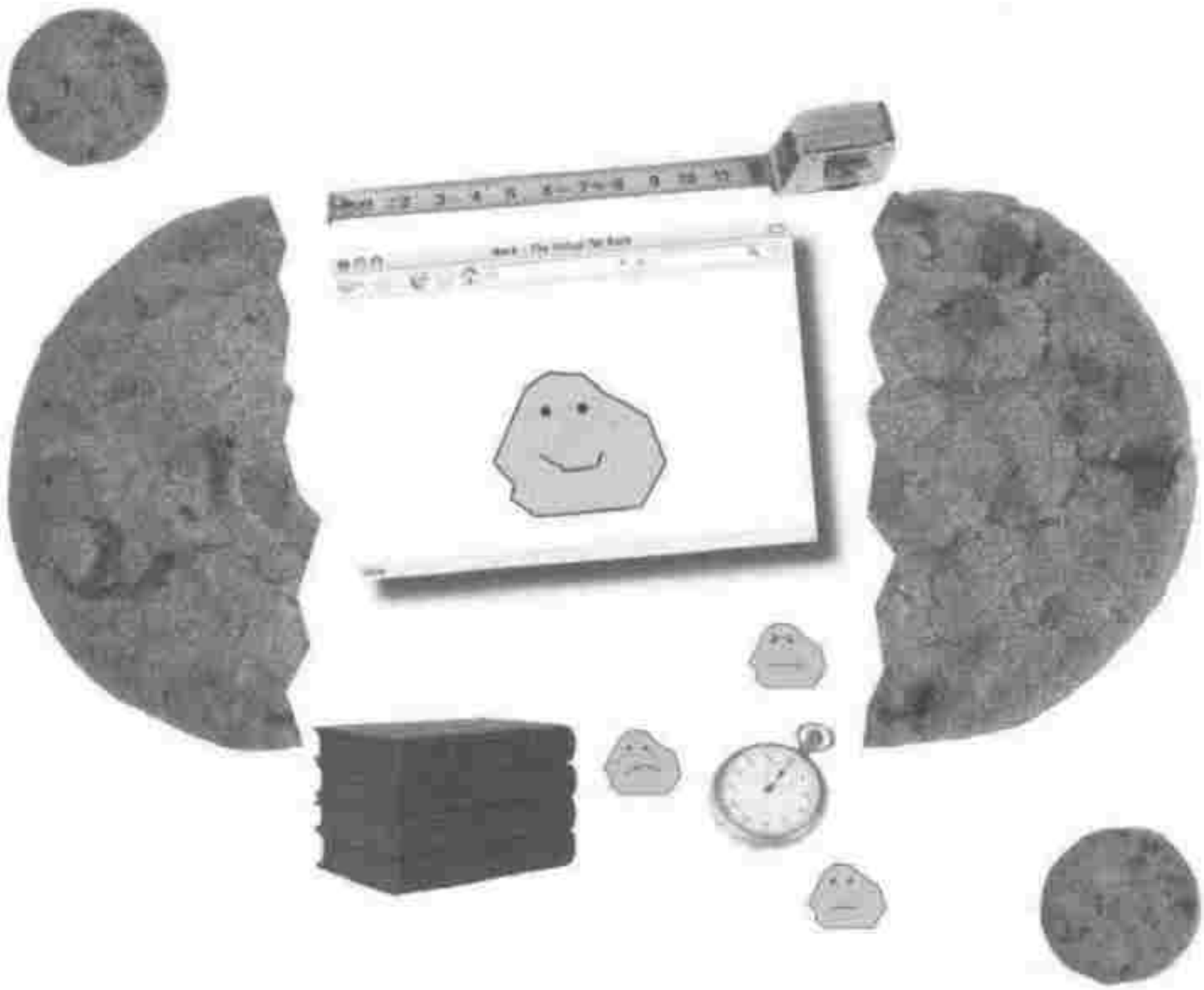
Solução das Cruzadas do JavaScript

Dobra da Página

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

Por que o JavaScript se preocupa com o cliente?

É um encontro de mentes!



O cliente é onde o código JavaScript é executado, o que significa que o JavaScript reside no navegador. Isso é muito positivo, porque significa que o servidor tem menos com o que se preocupar, como, por exemplo, armazenar cookies!

Se houver uma bifurcação na estrada, pegue-a

A vida é uma tomada de decisões. Parar ou seguir, dispensar ou guardar, declarar-se culpado ou ir a julgamento... sem habilidade para tomar decisões nada seria feito. Funciona da mesma maneira no JavaScript – **as decisões permitem que os scripts escolham entre diferentes possibilidades de resultado. *A tomada de decisão conduz a "história" de seus scripts***, e até mesmo os scripts mais rotineiros envolvem algum tipo de história. Você confia no que o usuário digitou e faz reserva em uma viagem de expedição Sasquatch[1] para ele? Ou verifica se ele só quer pegar um ônibus para Saskatchewan[2]? A escolha é sua!

Concorrente sortudo, venha até aqui!

No episódio de hoje do emocionante concurso *Quer fazer uma aposta?* um concorrente está prestes a ser escolhido...

Embora não haja dúvida de que você esteja sentindo um certo suspense na expectativa do talento de Eric em tomar decisões, a verdadeira pergunta é: que conhecimento tem o apresentador para anunciar Eric como um concorrente sortudo?

Escolhas têm tudo a ver com tomar decisões

Dã, está escrito lá no cartão dele! Exato, mas parta do pressuposto que o apresentador pode tomar **decisões** com base no nome que aparece no cartão. É porque ele é humano, e as pessoas se esmeram em processar informações e tomar decisões. Se o apresentador fosse um script, as coisas não seriam assim tão fáceis.

A pergunta é a seguinte: como um script usa um fragmento de informação como base para tomar uma medida? Conhecer o nome que aparece em um determinado cartão é apenas a metade da resposta. A outra metade envolve a capacidade de **avaliar** o nome no cartão e, em seguida, **escolher** o concorrente com o mesmo nome, neste caso o Eric.

"se" isso for verdadeiro... então faça algo

JavaScript é, na verdade, muito competente em processar informações e tomar decisões, e uma das maneiras disso acontecer é com a instrução if. A instrução if permite que você tome decisões simples, executando condicionalmente um fragmento de código JavaScript baseado em um teste true/false.

Se o teste terminar sendo true, faça algo.

```
Se (teste true/
false) Faça algo;
```

Se você der uma olhada no exemplo do show de perguntas através das lentes de uma instrução JavaScript if, você terá um código assim:

Confere se as duas coisas são iguais.

Eric

Existe uma correspondência!

```
if (chosenContestant == "Eric")
  alert("Eric, venha até aqui!");
```

O teste true/false é sempre colocado entre parênteses.

Esse é o fim da instrução if.

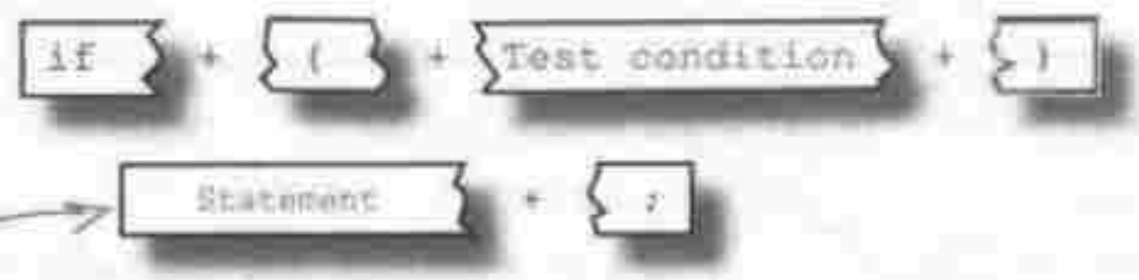
Nenhum ponto-e-vírgula, já que este não é de fato o fim da instrução if.

A instrução if é uma maneira excelente de executar condicionalmente um fragmento de código.

Uma instrução if avalia uma condição... e então, age

Toda instrução if adere ao mesmo formato. Você usava este formato quando adicionava cookies ao iRock, mas eis a decomposição:

Esse código TERÁ QUE, de alguma forma, avaliar true ou false.



Indente para facilitar a leitura do código. A instrução indentada faz parte de if.

Existem algumas coisas a serem lembradas sobre o formato da instrução `if`. Primeiro, você só pode executar um fragmento de código, e ele deve aparecer indentado acima de `if` e do teste de condição. Embora não seja rigorosamente exigido, é uma boa idéia indentar esse código para que você possa ver rapidamente que faz parte de sua instrução `if`. Eis os passos exigidos para levar a cabo uma decisão com uma instrução `if`:

1. Coloque o teste de condição true/false entre parênteses.
2. Indente alguns espaços na próxima linha do código.
3. Escreva o código que é executado se o teste de condição for true.

Exercício

Una as instruções if às ações que devem acompanhá-las.

```
if (hungry)                                numDonuts *= 12;
if (countDown == 0)                        userName = readCookie("irock_username");
if (donutString.indexOf("dozen") != -1)    awardPrize();
if (testScore > 90)                        goEat();
if (!guilty)                               alert("Houston, nós decolamos.");
if (winner)                                alert("Ela é inocente!");
if (navigator.cookieEnabled)               grade = "A";
```

Una as instruções if às ações que devem acompanhá-las.

Solução do Exercício

```
if (hungry)                                numDonuts *= 12;
if (countDown == 0)                        userName = readCookie("irock_username");
if (donutString.indexOf("dozen") != -1)    awardPrize();
if (testScore > 90)                        goEat();
if (!guilty)                               alert("Houston, nós decolamos.");
if (winner)                                alert("Ela é inocente!");
if (navigator.cookieEnabled)               grade = "A";
```

true se a string contém a palavra dozen

!guilty significa NOT guilty, portanto guilty deve ser false.

true se os cookies do navegador estiverem habilitados

Mas que droga! O que devo fazer SE existir mais de uma opção?

Faça isso... ou então.

Logo agora que você pensava que o JavaScript tinha abordado tudo, aparece algo fora do comum. Na verdade, escolher entre mais de um resultado não é tão fora do comum assim... chocolate ou baunilha, descafeinado ou normal, parece que muitas opções envolvem uma coisa ***ou*** outra. Eis porque a instrução `if` pode ser ajustada para permitir uma tomada de decisão e, em seguida, agir de uma ou duas maneiras possíveis... ou, às vezes, até mais do que isso.

Use if para escolher entre duas coisas

Uma guinada na instrução `if` permite que você escolha entre as duas possibilidades de resultado. De volta ao *Quer fazer uma aposta?*, Eric está lutando para tomar uma decisão como essa. Apresentado as duas alternativas, ele tem que escolher uma das duas.

O Caso A é a primeira opção de Eric...

...e o Caso B é a segunda opção.

```
if (teste true/false)
  Faça algo;
else
  Faça algo diferente;
```

Raramente, a vida é tão simples quanto escolher apenas **uma coisa**. A instrução `if` nos dá (e ao Eric) a capacidade de escolher o Caso A **ou** o Caso B. Mas como fica isso no JavaScript?

```
if (chosenCase == "A")
openCase("A");

?
```

E agora? Como dizer o que fazer se chosenCase não é igual a A?

Você pode tomar várias decisões com if

Usar uma instrução `if` para tomar uma medida significa transformá-la em uma instrução `if/else`, o que lhe dará a opção de executar um fragmento de código diferente se o teste true/false falhar. É como dizer: **se** o teste for verdadeiro, execute o primeiro fragmento de código; e se **não** (***então***), execute um outro fragmento de código.

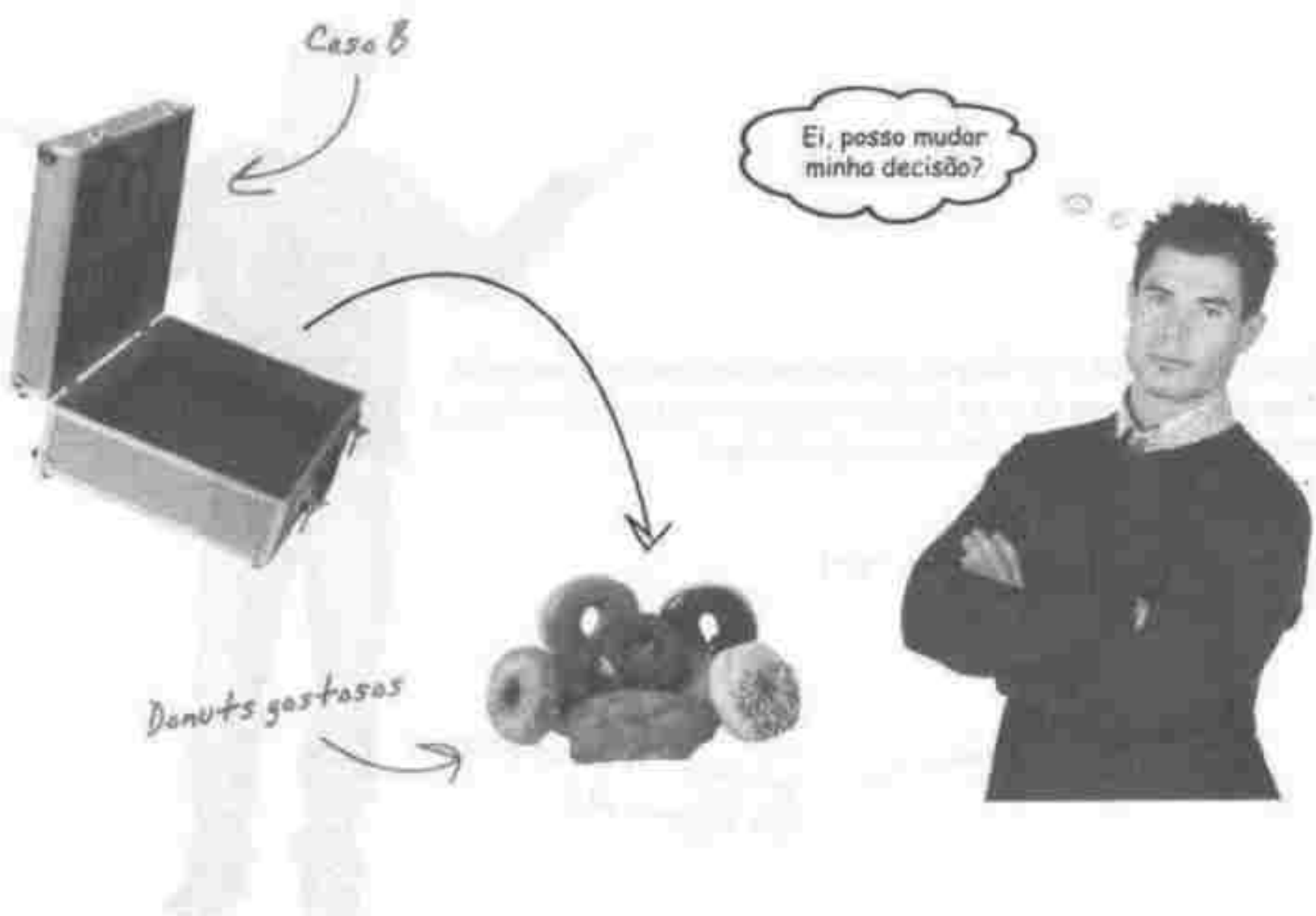
```
if (chosenCase == "A")
openCase("A");
else
openCase("B");
```

True

False

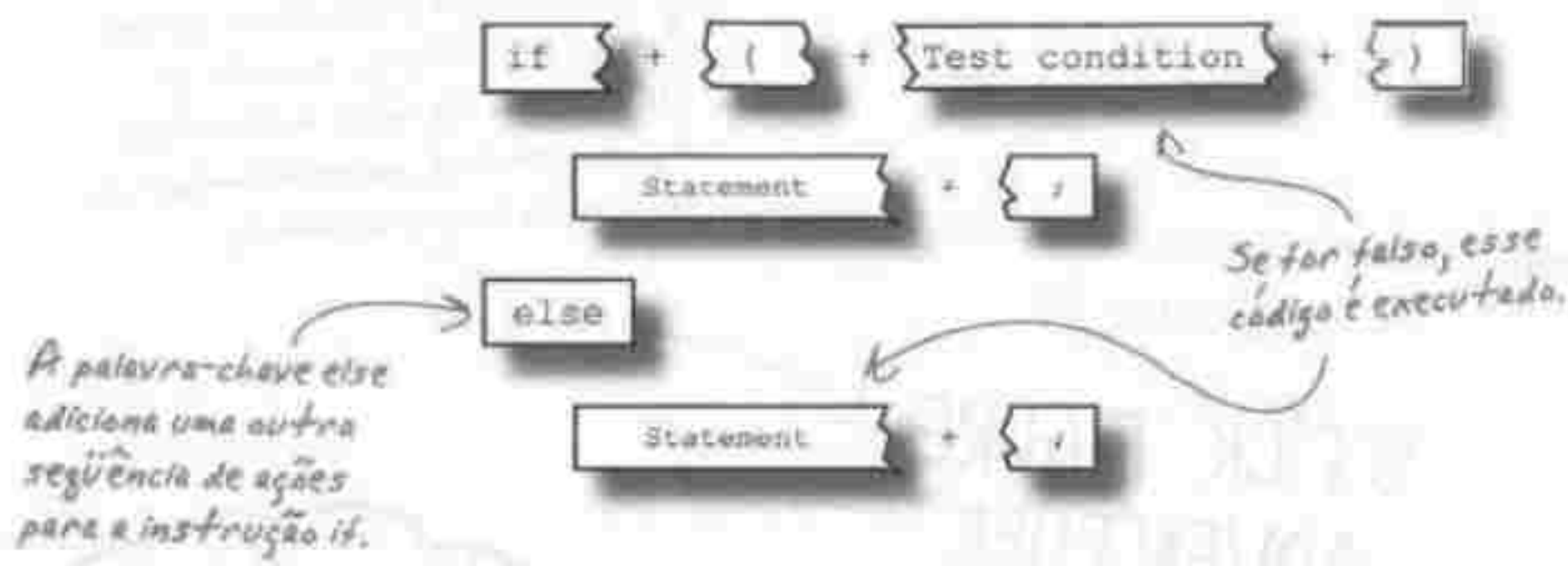
A instrução if/else consiste em duas possibilidades de resultados, uma para cada possibilidade de valor do teste de condição.

Eric escolheu o Caso B, o que significa que a variável `chosenCase` seja "B". Portanto, como o primeiro teste de condição é falso, isso disparará a instrução `if/else` para executar o código `else`. Para Eric, infelizmente, o Caso B contém donuts, e não a pilha de dinheiro que ele estava esperando.



Como adicionar else à sua instrução if

A formatação de uma instrução `if/else` é muito semelhante à instrução if. Associe else à palavra-chave de forma a executar junto com o outro fragmento de código, se o teste de condição for falso:



Para adicionar uma segunda seqüência de ações a uma instrução if, siga esses passos:

1. Coloque a palavra-chave else depois da primeira instrução de ação.
2. Indente alguns espaços na próxima linha de código para facilitar a leitura.
3. Escreva o código que é executado se o teste de condição for falso.

Não existem perguntas idiotas

P: Por que não existe um ponto-e-vírgula depois dos parênteses em uma instrução if?

R: A regra no JavaScript é que cada instrução deve terminar com um ponto-e-vírgula, e a instrução if não é uma exceção. No entanto, a instrução `if` não é apenas `if (Test Condition)`, ela também é o código que é executado se a condição for true. Esse código termina com um ponto-e-vírgula. Então, a instrução `if` termina com um ponto-e-vírgula, se você entender exatamente o que representa uma instrução .

P: O que acontece quando o teste de condição é false em uma instrução `if` que não possui uma cláusula `else`?

R: Absolutamente nada. Neste caso, o valor do teste de condição resulta literalmente na não-ação a ser tomada.

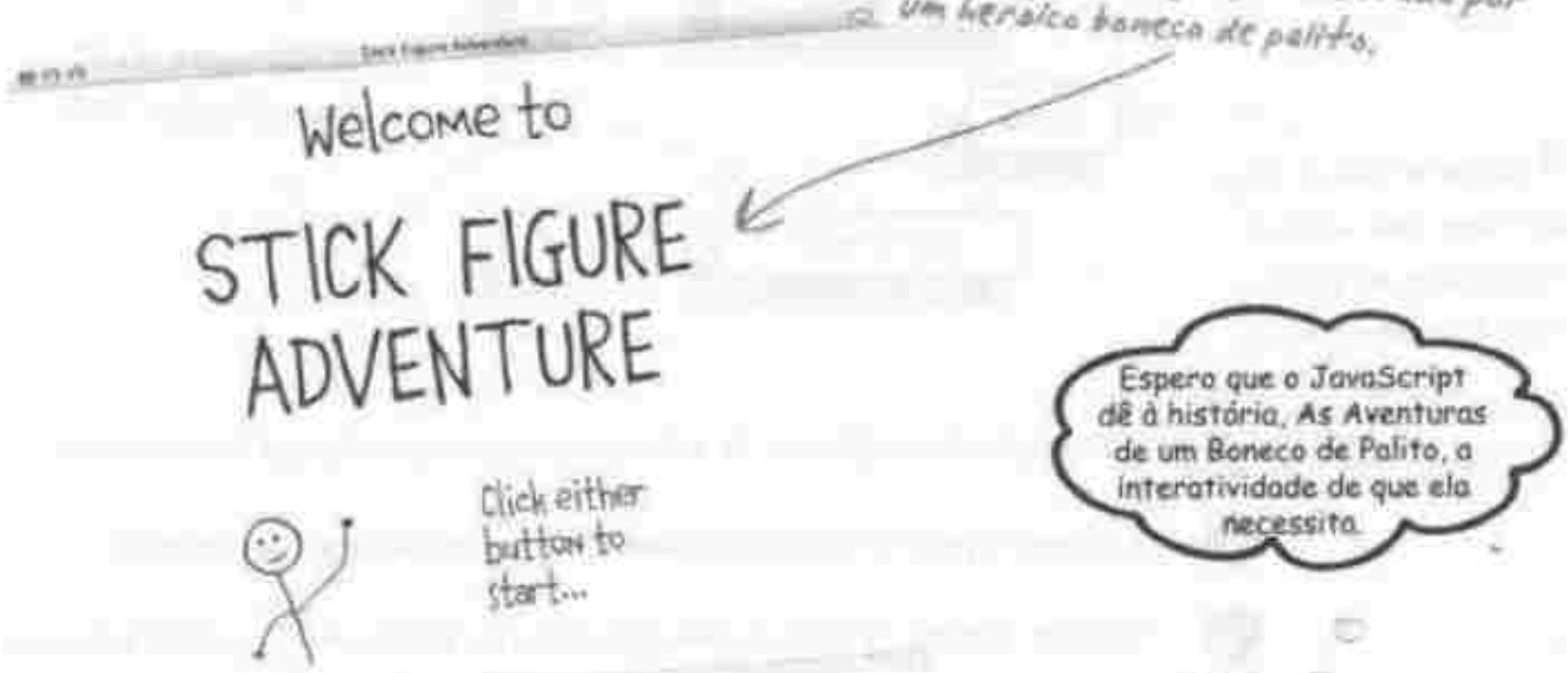
P: É possível usar mais de uma instrução `else` para escolher entre mais de duas possibilidades de resultado?

R: Sim. Certamente existe a possibilidade de estruturar a instrução `if/else` para dar suporte a mais de dois resultados, mas não é tão fácil quanto adicionar cláusulas `else` a mais. Você pode acabar aninhando instruções `if/else` inteiras uma dentro da outra, podendo se desorganizar rapidamente, se você tiver tomando uma decisão complexa com diversos resultados diferentes. A abordagem `if/else` não é incorreta, mas o JavaScript oferece uma estrutura melhor de tomada de decisão para esta situação, a instrução `switch/case`, que você aprenderá um pouco mais adiante neste capítulo.

Uma aventura de proporções épicas

Ellie está escrevendo uma aventura interativa chamada As Aventuras de um Boneco de Palito. O projeto dela envolve uma tomada de decisão em cada reviravolta na história, e ela espera que a tomada de decisão do JavaScript possa oferecer uma solução para o problema dela em colocar a aventura online para outras pessoas apreciarem.

A história As Aventuras de um Boneco de Palito apresenta ao usuário uma história interativa baseada em web, experimentada por um heróico boneco de palito.



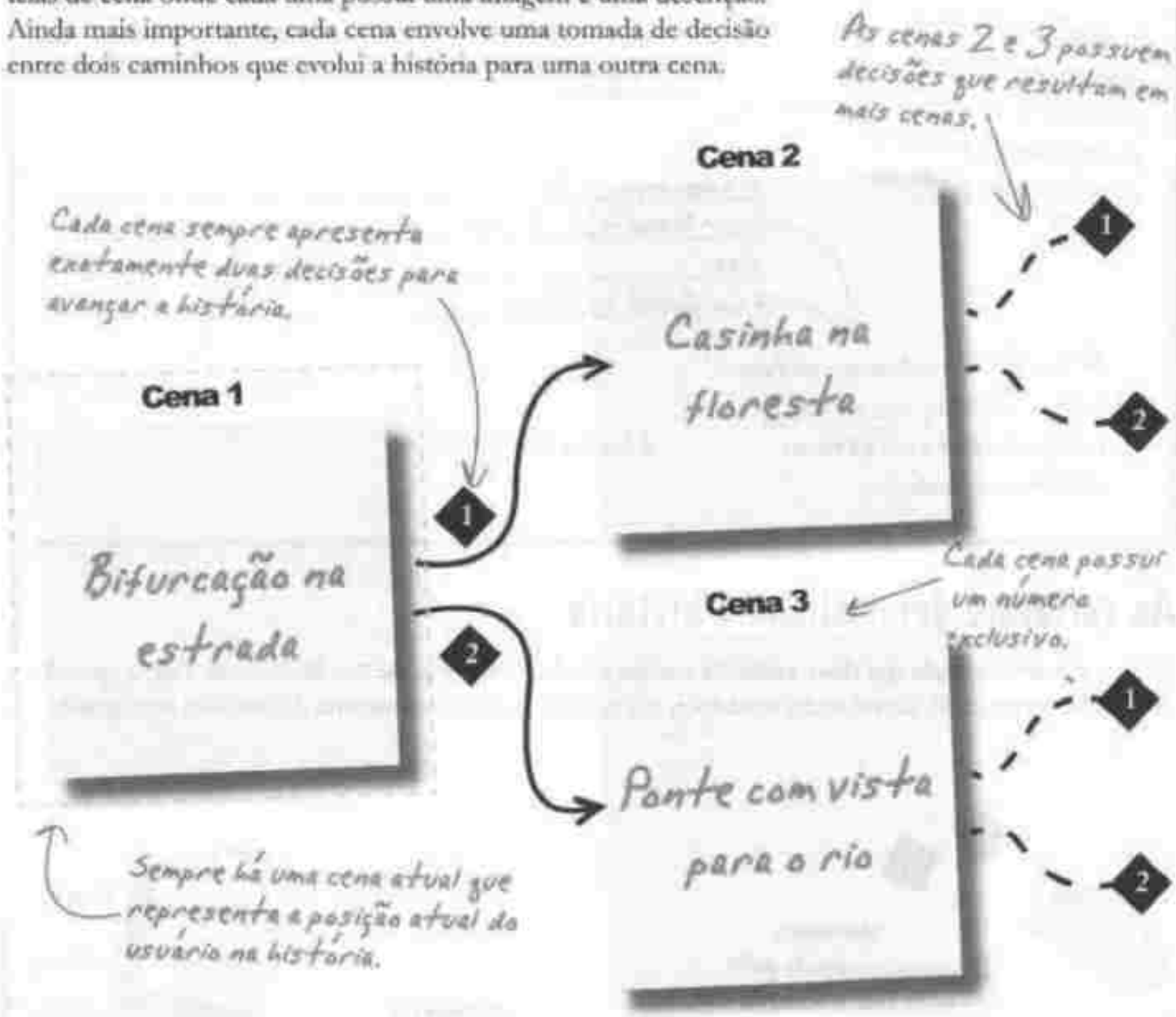
Espero que o JavaScript dê à história, As Aventuras de um Boneco de Palito, a interatividade de que ela necessita.

Ellie sonha com as reviravoltas e transformações no enredo, mas se preocupa um pouco em como torná-las realidade.

Ellie deseja que a história de As Aventuras de um Boneco de Palito esteja online para permitir ao usuário navegar através da história, escolhendo entre duas opções em cada etapa ao longo do caminho. Você pode acompanhar através dos arquivos relacionados, disponíveis para download em *http://www.headfirstlabs.com/books/hfjs/*.

A configuração da aventura

A história As Aventuras de um Boneco de Palito é uma seqüência de telas de cena onde cada uma possui uma imagem e uma descrição. Ainda mais importante, cada cena envolve uma tomada de decisão entre dois caminhos que evolui a história para uma outra cena.



Aponte seu lápis

Escreva o código para uma instrução `if/else` que decide por uma das três primeiras cenas em As Aventuras de um Boneco de Palito. Dica: uma variável chamada `decision` armazena a decisão do usuário, enquanto a variável `curScene` conterá a cena resultante.

.....

.....

.....

.....

Aponte seu lápis
Solução

Escreva o código para uma instrução `if/else` que decide por uma das três primeiras cenas em As Aventuras de um Boneco de Palito. Dica: uma variável chamada `decision` armazena a decisão do usuário, enquanto a variável `curScene` conterá a cena resultante.

A variável decision armazena a decisão do usuário em um determinado ponto da história, que pode ser 1 ou 2.

```
if (decision == 1)
  curScene = 2;
else
  curScene = 3;
```

A variável curScene contém a cena atual e, neste caso, avança a cena com base na decisão do usuário.

Muda para a Cena 3.

Muda para a Cena 2.

As variáveis determinam a história

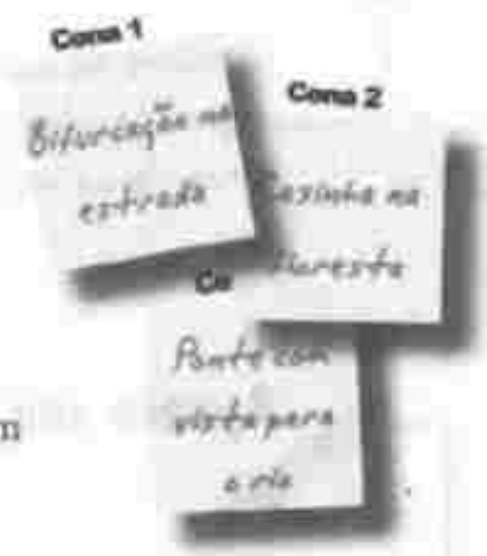
Vamos dar uma olhada nas duas variáveis usadas em As Aventuras de um Boneco de Palito, que são críticas na resposta às decisões do usuário e, em seguida, adiantar a história de maneira apropriada.

decision

Decisão escolhida pelo usuário, que é sempre 1 ou 2. Essa decisão determina a próxima cena na história.

curScene

Cena atual na história, que é sempre um número que corresponde à cena, como em Cena 1, Cena 2 etc.



As variáveis `decision` e `curScene` trabalham juntas para armazenar a decisão do usuário e, em seguida, usa a decisão como base para adiantar a história. Esse processo se repete de cena a cena, à medida que a história continua a se desenvolver, tudo graças à instrução `if/else`.

Mas está faltando parte da história

A instrução `if/else` funciona bem como um instrumento para tomada de decisão de parte das Aventuras de um Boneco de Palito, mas a história toda não foi contada. Cada cena envolve imagem **e** descrição em texto; imagem e texto de uma determinada cena são exibidos, conforme a história se desenvolve. Alterar o número da cena atual é suficiente para alterar a imagem da cena, mas não ajuda no texto de descrição da cena.

A nova cena atual.

Cena 2

Casinha na floresta

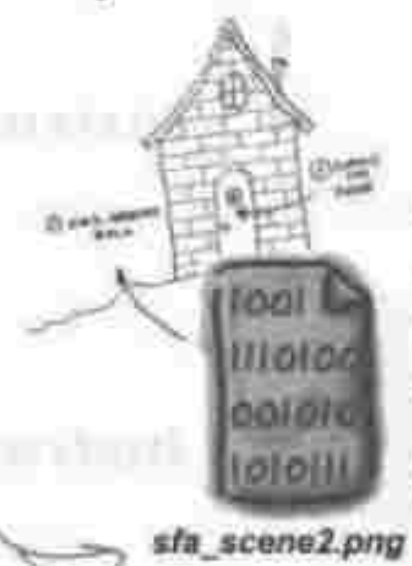
O usuário escolhe a Decisão 1, que leva à Cena 2.

Cena 1

Bifurcação na estrada

1

Imagem de cena



sfa_scene2.png

Descrição de cena

"Você chegou a uma linda casinha na floresta".

Uma imagem PNG é muito semelhante a uma imagem GIF, mas PNG é um padrão mais recente e amplamente aceito.

A descrição de cena não foi incluída, porque somente uma medida é tomada em resposta à decisão da cena.

O número de cena é suficiente para definir a imagem de cena, já que o nome de arquivo da imagem de cena contém o número da cena.

```
document.getElementById("sceneimg").src = "scene" + curScene +
".png";
```

Com capacidade para executar somente um único fragmento de código em cada parte da instrução `if/else`, você está limitado a apenas uma ação. Em outras palavras, você não pode exibir uma imagem **e** mostrar uma descrição de texto.

Poder do Cérebro

Como você faria mais de uma coisa em resposta a uma decisão?

Como compor as atividades de seu JavaScript

Ellie precisa ser capaz de fazer mais de uma coisa em cada desvio de código `if/else`. Ela precisa alterar o número de cena e o texto de descrição de cena, de modo que as duas linhas seguintes do código possam avançar ao longo da história:

Altera a imagem de cena para a cena recém-escolhida.

```
document.getElementById("sceneimg").src = "scene" +
curScene + ".png";
alert(message);
```

Exibe a mensagem de descrição de cena da nova cena para o usuário.

O desafio é fazer mais de uma coisa mesmo que o JavaScript só permita executar um simples fragmento de código. A solução é uma **instrução composta**, que lhe permite moldar um bloco de código, fazendo com que ele apareça no script como um simples fragmento de código. Pode-se criar uma instrução composta incluindo uma série de instruções dentro de chaves ({}).

Ahá! Uma instrução composta me permite tratar um bloco de código grande como se fosse um único fragmento de código.

```
doThis();
doThat();
doSomethingElse();
```

= 3 instruções

Combina as chaves de abertura e encerramento para incluir o código.

```
{
doThis();
doThat();
doSomethingElse();
}
```

= 1 instrução

Com uma instrução composta, é possível criar instruções `if/else` que façam mais de uma coisa em cada desvio de ação:

```
if (chosenDoor == "A") {
  prize = "donuts";
  alert("Você ganhou uma caixa de donuts!");
}
else {
  prize = "rocha de estimação";
  alert("Você ganhou uma rocha de estimação!");
```

Faz mais de uma coisa em cada desvio de ação de uma instrução if/else.

Não existem perguntas idiotas

P: Exatamente de que maneira As Aventuras de um Boneco de Palito usa as variáveis para conduzir a história?

R: Em um determinado momento, a variável `curScene` contém o número de cena atual. Cada cena mostra uma imagem de cena e uma descrição de cena, e também apresenta ao usuário uma opção que permite escolher entre uma ou duas cenas. A variável `decision` contém a decisão de cena do usuário, 1 ou 2. Quando a escolha é feita, o valor da variável `decision` é usado junto com o `curScene` para determinar a nova cena. Mais especificamente, a imagem de cena é alterada usando-se o valor em `curScene`, e a mensagem de descrição de cena é exibida usando-se uma caixa de alerta.

P: Por que é importante que uma instrução composta comprima várias instruções em uma?

R: É importante, porque muitas partes da linguagem JavaScript são estruturadas em torno da idéia de uma instrução simples. É como se uma empresa aérea lhe desse permissão para duas bagagens de mão. Nada o impede de enchê-las com uma porção de coisas, contanto que você se limite a duas. Então, instruções compostas são como uma parte da bagagem de mão em que é permitido enchê-la com várias instruções em um único "recipiente", o qual é **visto** como uma instrução única em relação ao resto do script.

P: Por que uma instrução composta termina com um ponto-e-vírgula?

R: O ponto-e-vírgula é reservado para instruções individuais, e não instruções compostas. Portanto, as instruções simples que aparecem dentro de uma instrução composta precisa ter o ponto-e-vírgula delimitador, mas a instrução composta não precisa.

P: Uma função é uma instrução composta?

R: Boa pergunta! E a resposta é sim. Você já deve ter notado que o código em uma função é cercado por chaves. Por enquanto, pense em uma função como uma instrução composta que você pode passar dados para dentro e fora dela.

Aponte seu lápis

Reescreva o código para a primeira decisão `if/else` em As Aventuras de um Boneco de Palito. Dessa vez, use as instruções compostas para definir tanto o número de cena como a mensagem de descrição de cena.

.....

.....

.....

.....

.....

.....

.....

.....

Aponte seu lápis
Solução

Reescreva o código para a primeira decisão if/else em As Aventuras de um Boneco de Palito. Dessa vez, use as instruções compostas para definir tanto o número de cena como a mensagem de descrição de cena.

O número de cena atual é ajustado com base na decisão tomada pelo usuário

A mensagem de descrição de cena é definida para corresponder à nova cena.

Inicia a instrução composta com uma chave de abertura.

É uma boa idéia indentar o código dentro de uma instrução composta.

```
if (decision == 1) {
  curScene = 2;
  message = "Você chegou a uma linda casinha na floresta";
}
else {
  curScene = 3;
  message = "Você está posicionada em uma ponte com vista para um rio tranqüilo";
}
```

Finaliza a instrução composta com uma chave de encerramento.

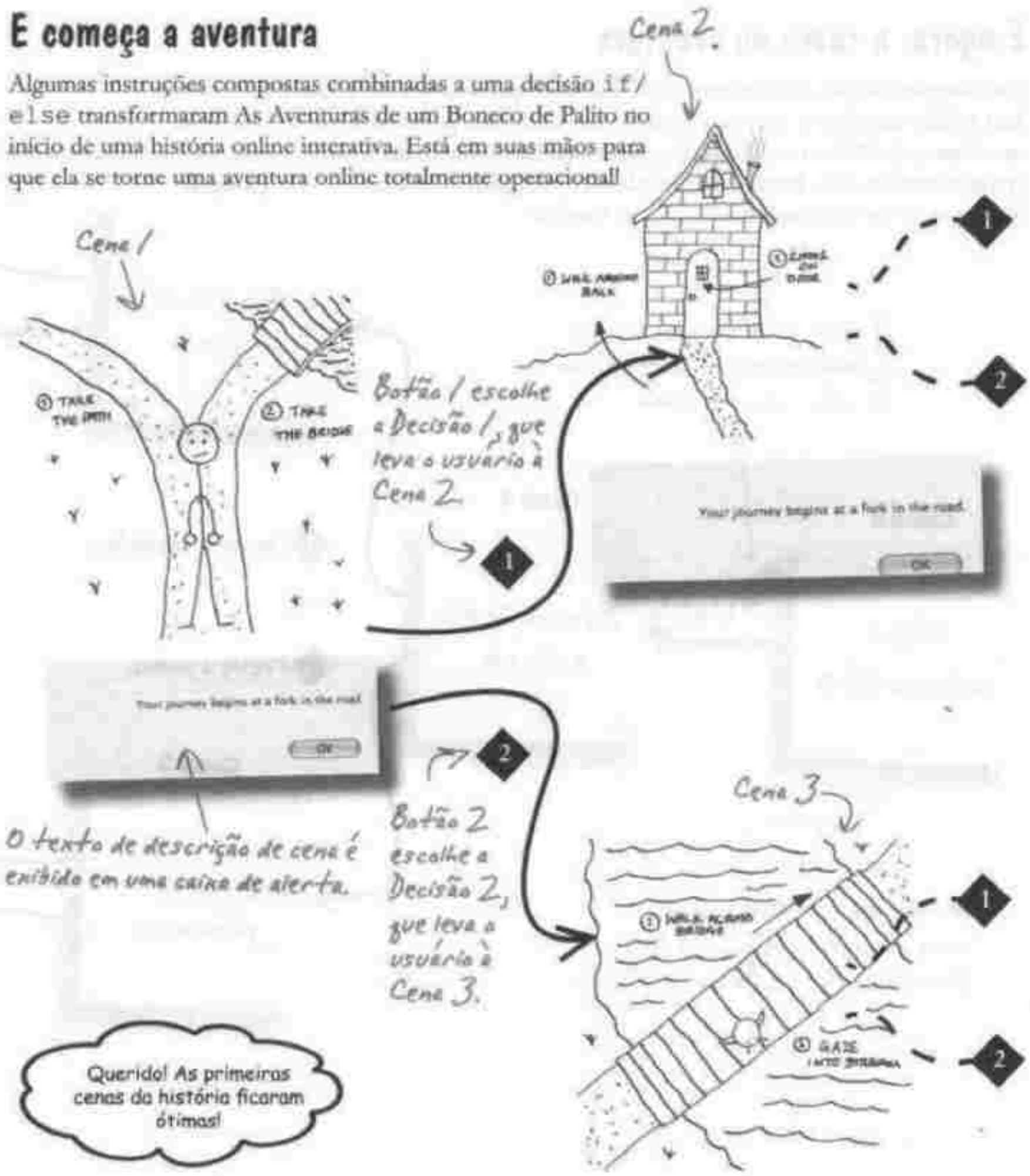
Uma mensagem de descrição de cena diferente é definida para Cena 3.

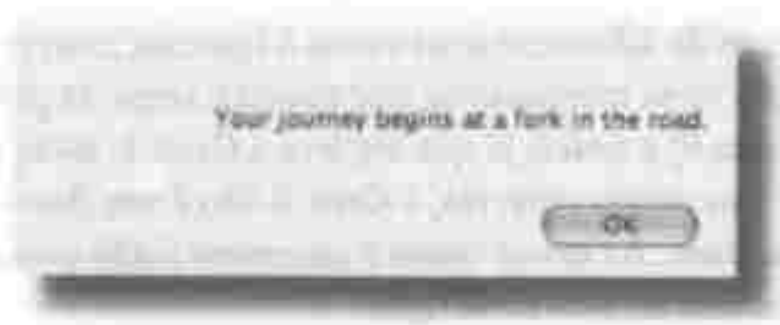
Pontos de Bala

- Use a instrução `if` para executar condicionalmente um fragmento de código JavaScript simples.
- O teste de condição em uma instrução `if` tem sempre que resultar em true ou false.
- Use a instrução `if/else` para executar condicionalmente um ou dois fragmentos JavaScript diferentes.
- Use uma instrução composta para executar várias partes do código JavaScript em vez de um fragmento de código simples.
- Crie uma instrução composta incluindo uma série de instruções individuais dentro de chaves ({}).
- Instruções compostas permitem que partes da ação das instruções `if` e `if/else` façam mais de uma coisa.

E começa a aventura

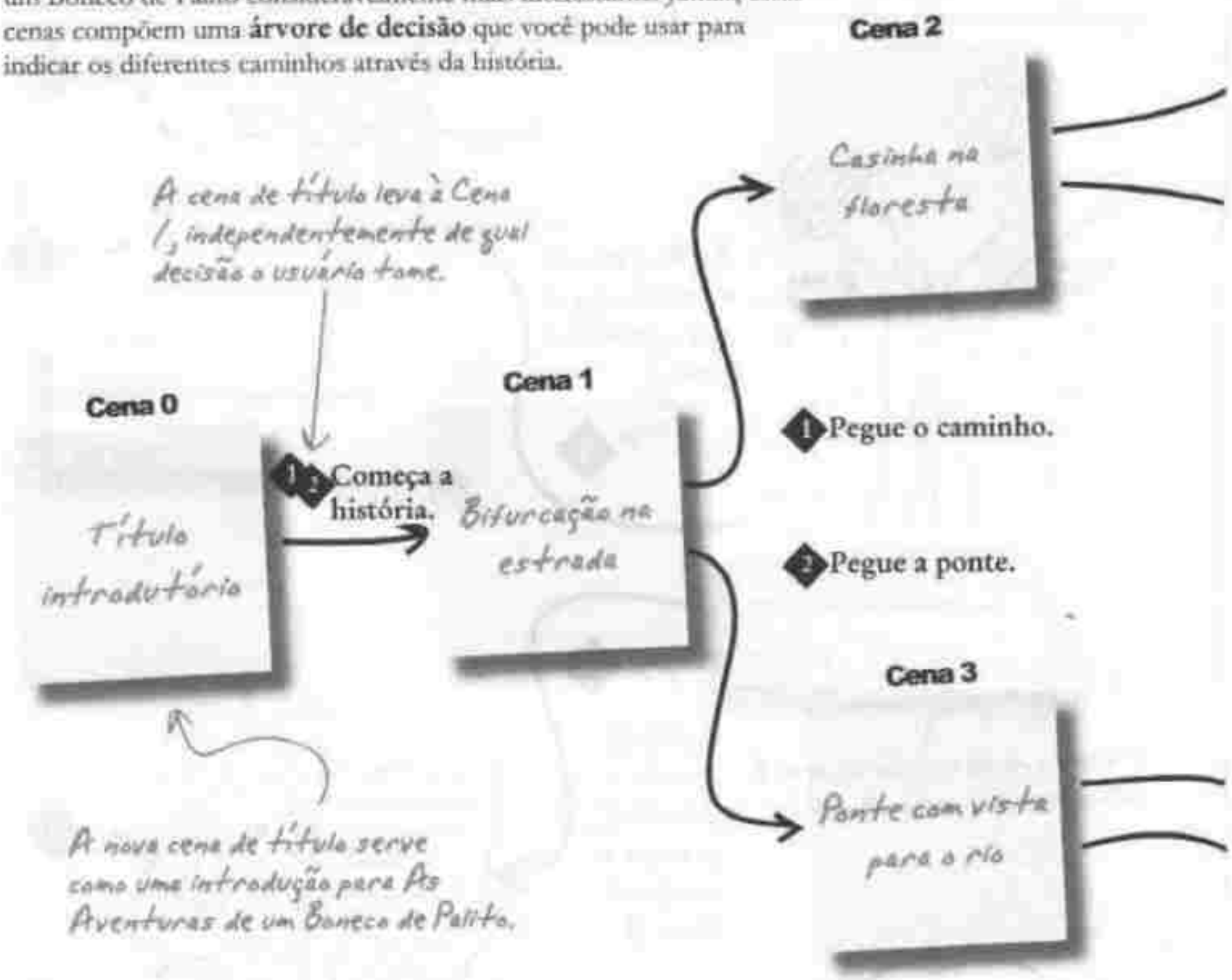
Algumas instruções compostas combinadas a uma decisão `if/else` transformaram As Aventuras de um Boneco de Palito no início de uma história online interativa. Está em suas mãos para que ela se torne uma aventura online totalmente operacional!



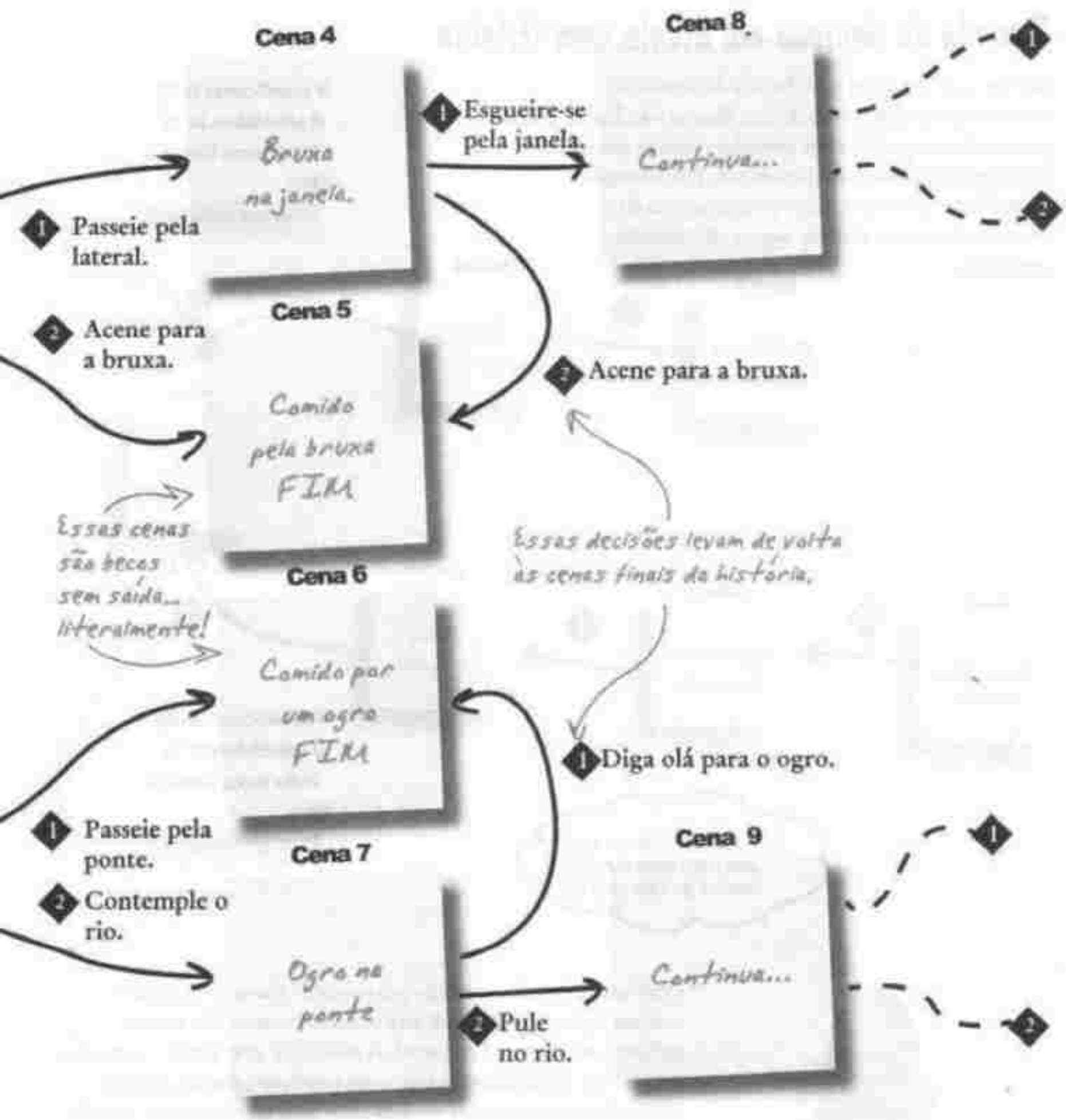


E agora, o resto da aventura

Uma simples decisão dificilmente tende a resultar em uma história interativa interessante. Mas a Ellie tem planos, inclusive várias cenas adicionais, para tornar As Aventuras de um Boneco de Palito consideravelmente mais interessante. Juntas, essas cenas compõem uma **árvore de decisão** que você pode usar para indicar os diferentes caminhos através da história.



Além de adicionar mais cenas à história, com novas reviravoltas e guinadas, Ellie também criou uma nova cena introdutória que aparece antes da primeira cena da história (Cena 1). A cena de título (Cena 0) é única, já que ela leva à Cena 1, independentemente de escolher a Decisão 1 ou a Decisão 2. Em outras palavras, a Cena 0 não é um desvio na história, ao contrário, apenas os créditos de abertura. As novas cenas e aberturas estão prontas para você fazer o download em *http://www.headfirstlabs.com/books/hfjs/*.

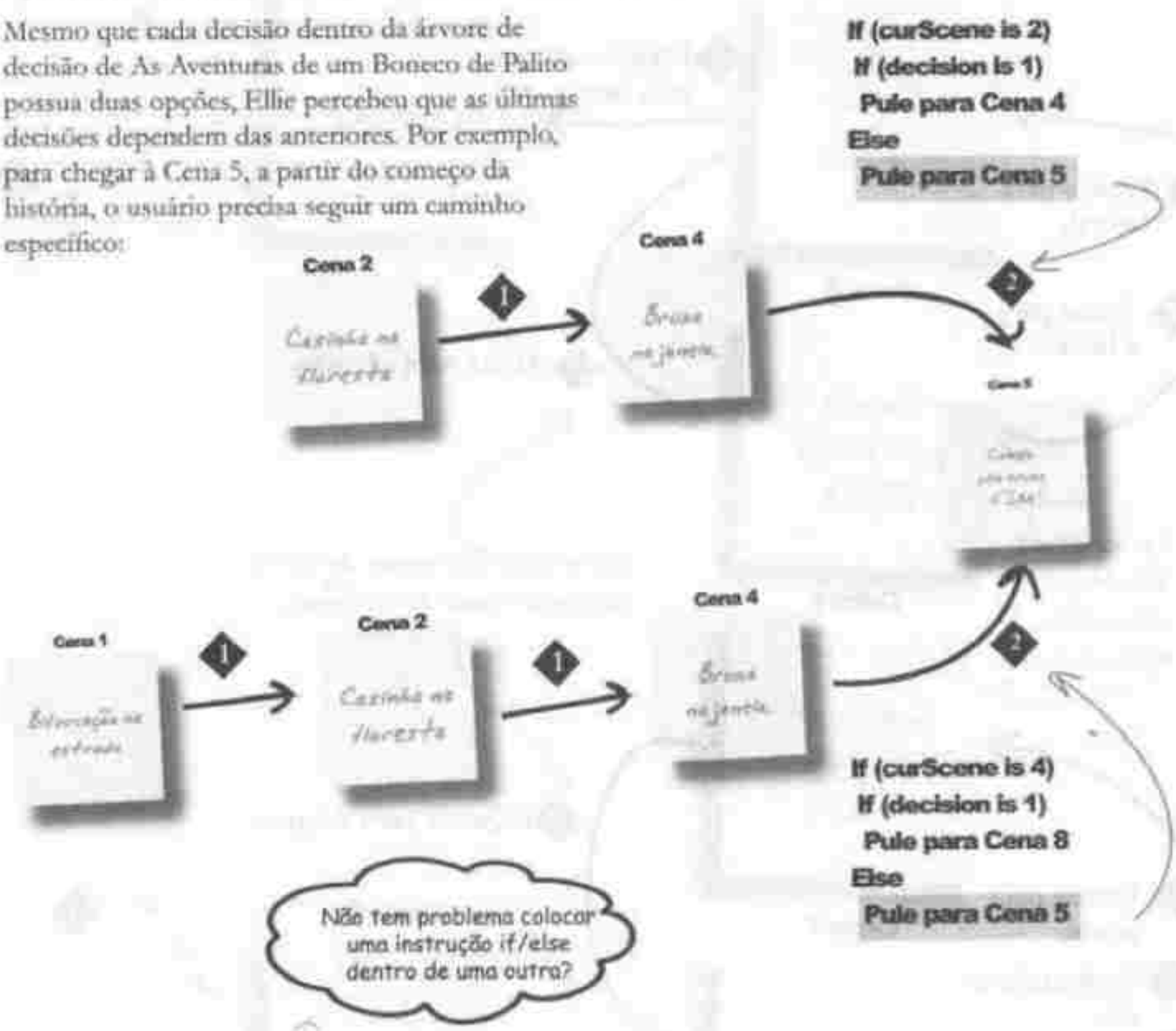


Poder do Cérebro

Com o quê a árvore de decisão de As Aventuras de um Boneco de Palito se pareceria usando-se apenas as instruções `if/else`?

Tomada de decisão em níveis com if/else

Mesmo que cada decisão dentro da árvore de decisão de As Aventuras de um Boneco de Palito possua duas opções, Ellie percebeu que as últimas decisões dependem das anteriores. Por exemplo, para chegar à Cena 5, a partir do começo da história, o usuário precisa seguir um caminho específico:



Conhecer a opção escolhida pelo usuário não é informação suficiente para decidir qual será a próxima cena. Ela precisa incluir como fator a cena atual. A solução é usar várias instruções `if/else` de tal maneira que você verifique primeiro a cena atual e depois tome uma medida com base na decisão do usuário. Mas essa abordagem de tomada de decisão em níveis envolve um `if` dentro de um `if`... uma proposta aparentemente estranha.

Ou seja, até que você leve em consideração que tomamos decisões em níveis o tempo todo. Você já respondeu à pergunta, "Gostaria de fritas como acompanhamento?" Essa pergunta naturalmente se segue a um pedido de salada. A pergunta se baseia na resposta que você já forneceu, como "Eu quero um cheeseburguer". Eis uma decisão em níveis, porque a última pergunta (fritas?) depende das respostas às perguntas anteriores (cheeseburguer ou salada?).

Um if pode estar dentro de um outro if

É perfeitamente legítimo colocar um if dentro de um if no JavaScript. Lembre-se, uma instrução if é sempre uma instrução, só que usada como parte da ação de um outro if. Em outras palavras, não há problema em complementar uma pergunta com uma outra pergunta. Um if dentro de um outro if é conhecido como uma instrução if **aninhada**.

```
if (order == "cheeseburger") {
  if (wantFries)
    order += " fries";
}
else if (order == "salad") {
  if (wantFruit)
    order += " fruit";
}
```

As instruções if aninhadas só entram em ação se as instruções if externas forem true.

Uma maneira mais rápida de dizer pedido = pedido + ... de modo que o resultado seja salada e fruta ou cheeseburger e fritas.

Aponte seu lápis

Escreva o código da tomada de decisão para a Cena 0 e Cena 1 de As Aventuras de um Boneco de Palito, certificando-se de usar as instruções if e if/else aninhadas quando necessário.

Aponte seu lápis
Solução

Escreva o código da tomada de decisão para a Cena 0 e Cena 1 de As Aventuras de um Boneco de Palito, certificando-se de usar as instruções `if` e `if/else` aninhadas quando necessário.

Cena 0 sempre leva à Cena 1, portanto não há necessidade de nenhuma instrução if aninhada.

Define a mensagem da descrição da Cena 1.

Se a cena atual não for a Cena 0, verificamos se é a Cena 1.

```
if (curScene == 0) {
  curScene = 1;
  message = Your journey begins at a fork in the road.;
}
else if (curScene == 1) {
  if (decision == 1) {
    curScene = 2;
    message = You have arrived at a cute little house in the woods.;
  }
  else {
    curScene = 3;
    message = You are standing on the bridge overlooking a peaceful stream.;
  }
}
```

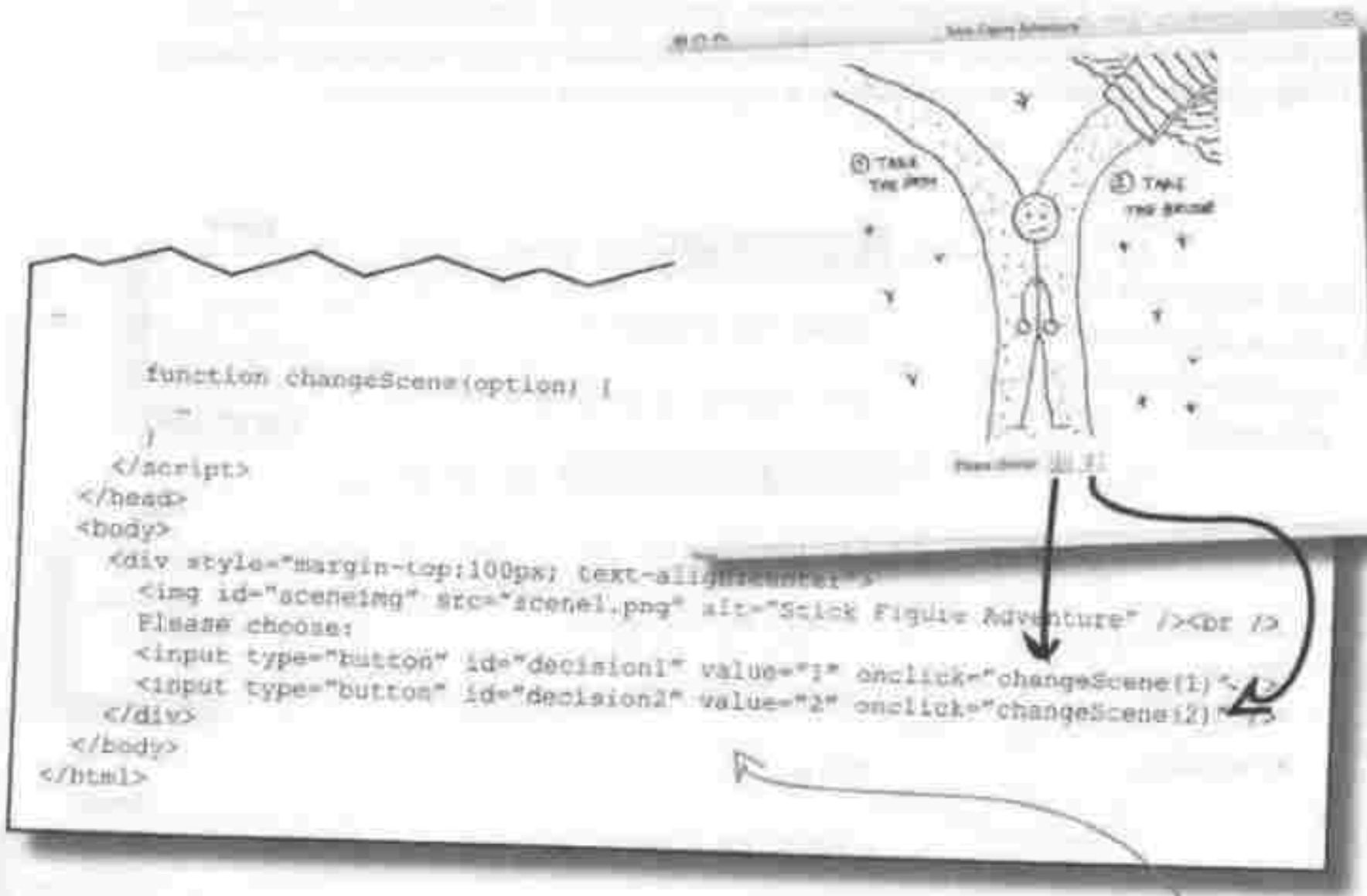
Uma instrução if/else manipula a decisão do usuário para a Cena 1.

Indentação ajuda a ver quais instruções estão aninhadas dentro de outras.

É extremamente importante combinar cuidadosamente as chaves de abertura e encerramento.

Suas funções controlam suas páginas

Um par de botões ("1" e "2") na página da web de As Aventuras de um Boneco de Palito é como os usuários percorrem a história. Quando eles decidem clicar em um dos botões, a função `changeScene()` é chamada para alterar a cena, com base no botão de decisão que foi clicado.



A página da web possui dois botões que são usados para determinar a próxima cena na história.

A função `changeScene()` recebe a decisão do usuário ("1" ou "2") como seu único argumento. Esse fragmento de informação é tudo que a função precisa para alterar a cena. Especificamente, a função `changeScene()` manipula:

1. A definição da variável `curScene` para o número da nova cena.
2. A definição da variável `message` para o texto de descrição da nova cena.
3. A alteração da imagem de cena com base no valor de `curScene`, e exibe a mensagem de texto de descrição de cena.

O pseudocódigo permite planejar sua aventura

Ellie tem uma idéia geral boa de como criar a função `changeScene()` para implementar a árvore de decisão de As Aventuras de um Boneco de Palito com código JavaScript. Porém, uma mera quantidade de decisões pode transformá-la em uma proposta confusa antes de começar a codificação. Às vezes, é mais prático escrever primeiro a árvore de decisão em pseudocódigo, que é uma maneira espontânea, mais legível e também muito informal de expressar o código de criação de scripts. Uma vez eliminado o pseudocódigo, o verdadeiro código JavaScript ficará muito mais claro e menos suscetível a erros de escrita.

Cada instrução if/else de nível superior representa uma cena diferente.

As instruções if/else tomam medidas com base na decisão do usuário.

```
If (curScene is 0)
  Jump to Scene 1
  Set Scene 1 message
Else If (curScene is 1)
  If (decision is 1)
    Jump to Scene 2
    Set Scene 2 message
  Else
    Jump to Scene 3
    Set Scene 3 message
Else If (curScene is 2)
...
```

Cena 0

Título introdutório

Cena 1

Bifurcação na estrada

Cena 2

Cabana na floresta

Não existem perguntas idiotas

P: O pseudocódigo se parece muito com o código JavaScript. Por que perder tempo?

R: Você não precisa perder tempo, mas a idéia é simplificar o processo de conversão de uma árvore de lógica complexa em código JavaScript e, ao mesmo tempo, minimizar o risco de cometer erros. Como o pseudocódigo não possui o mesmo nível de detalhes do código JavaScript, você pode concentrar seu trabalho mais na lógica de como uma cena leva à outra, do que na verificação da localização correta de cada chave e ponto-e-vírgula. Uma vez familiarizado com o pseudocódigo, traduzi-lo em código JavaScript é praticamente automático.

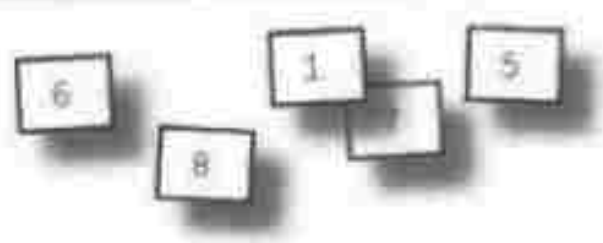
P: Você precisa usar chaves ao aninhar as instruções `if`?

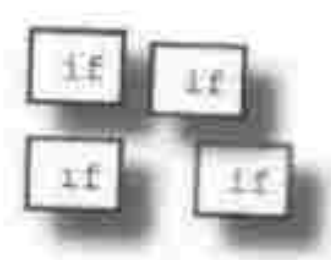
R: Não. Na verdade, se só estiver aninhando uma instrução `if` simples dentro de uma outra instrução `if` sem nenhum outro código, é mais simples omitir as chaves, já que *tecnicamente* você não precisa de uma instrução composta. No entanto, em uma distribuição complexa de instruções `if`, pode ser vantajoso usar chaves mesmo quando não for estritamente necessário, só para tornar a distribuição mais evidente.

Imãs de geladeira Javascript

Está faltando alguns fragmentos de código na função `changeScene()` de As Aventuras de um Boneco de Palito. Use os imãs para terminar o código que falta no diagrama de cena na página 152. Observe que nem todos os códigos de decisão de cena são mostrados – algumas cenas foram omitidas intencionalmente.

```
function changeScene(option) {
  var message = "";

  ........(curScene == 0) {
    curScene =      ;
    message = "Sua jornada começa em uma bifurcação na estrada.";
  }
...
  ..............(curScene == 3) {
    .....(option == 1) {
      curScene =      ;
      message = "Desculpe, tem um ogro que mora do outro lado da ponte e você" +
        "agora é o almoço dele.";
    }
    .....   {
      curScene =      ;
      message = "Seu contemplamento é interrompido pela chegada de um ogro
enorme.";
    }
  }
  ..............(curScene == 4) {
    if (option == 1) {
      curScene =      ;
    }
    ........{
      curScene =      ;
      message = "Desculpe, você agora faz parte do ensopado da bruxa.";
    }
  }
  ...

  document.getElementById("sceneimg").src = "scene" + curScene + ".png";
  alert(message);
}
```





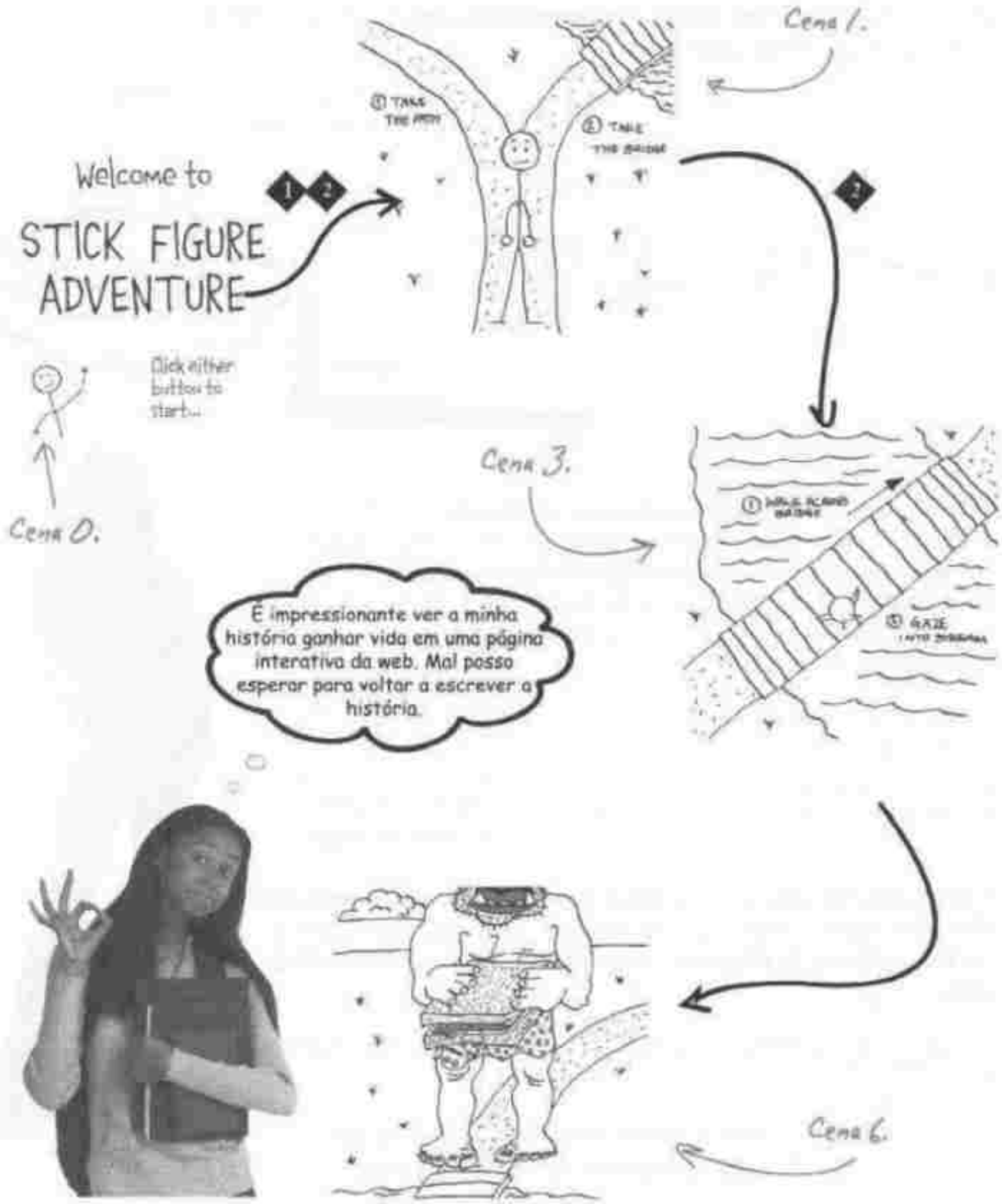
Solução dos Imãs de geladeira Javascript

Está faltando alguns fragmentos de código na função `changeScene()` de As Aventuras de um Boneco de Palito. Use os imãs para terminar o código que falta no diagrama de cena na página 152. Observe que nem todos os códigos de decisão de cena são mostrados – algumas cenas foram omitidas intencionalmente.

```
function changeScene(option) {
  var message = "";

  if (curScene == 0) {
    curScene = 1;
    message = "Sua jornada começa em uma bifurcação na estrada.";
  }
  else if (curScene == 3) {
    if (option == 1) {
      curScene = 6;
      message = "Desculpe, tem um ogro que mora do outro lado da ponte e você" +
        "agora é o almoço dele.";
    }
    else {
      curScene = 7;
      message = "Seu acampamento é interrompido pela chegada de um coro " +
enorme.";
    }
  }
  else if (curScene == 4) {
    if (option == 1) {
      curScene = 8;
    }
    else {
      curScene = 5;
      message = "Oh, cara, você agora faz parte do ensopado da bruxa.";
    }
  }
  ...

  document.getElementById("sceneimg").src = "scene" + curScene + ".png";
  alert(message);
}
```

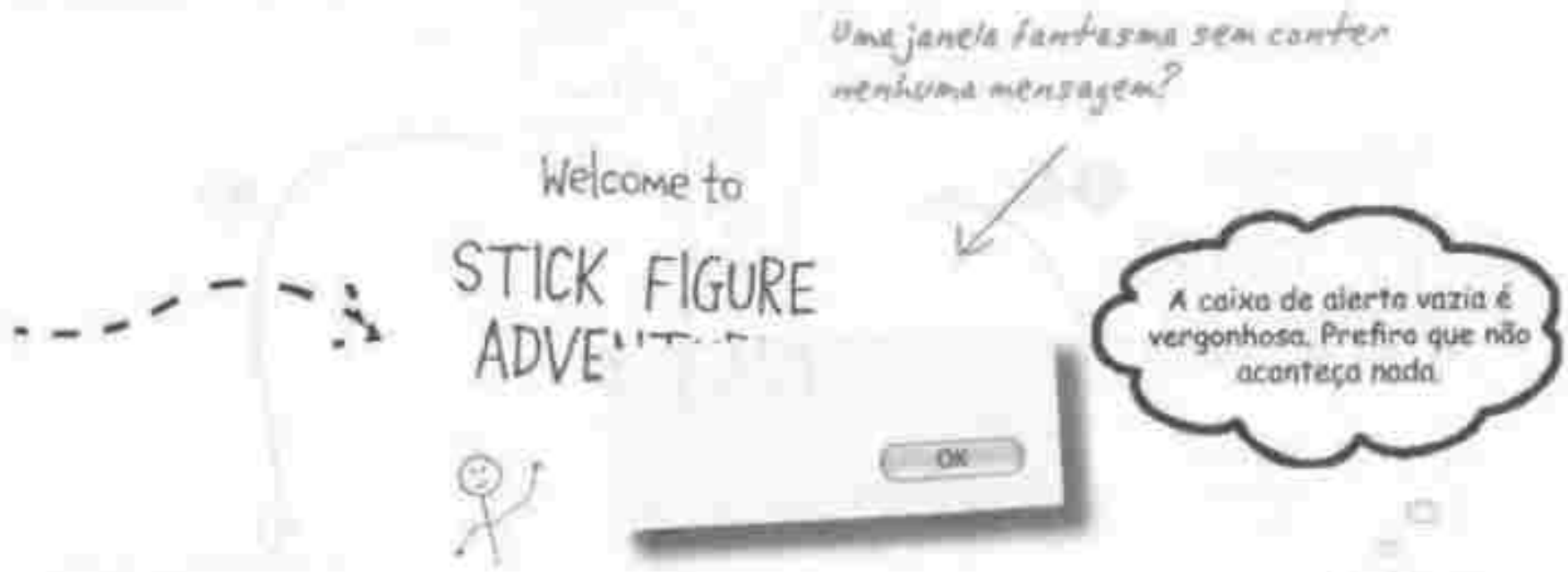
Continuando com As Aventuras de um Boneco de Palito

O script de As Aventuras de um Boneco de Palito agora reflete toda a árvore de decisão, permitindo que você navegue pela história ao longo de diversos caminhos. Eis um deles:



A disparidade de um boneco de palito

Infelizmente, Ellie já está encontrando problemas com As Aventuras de um Boneco de Palito. Após liberar a página para alguns amigos como teste, alguns deles informaram sobre uma janela estranha que exibe uma mensagem em branco. A "janela fantasma" é isolada quando uma nova aventura é iniciada, após o término da anterior. Então, o problema está de algum modo associado à mudança da Cena 0 para uma outra cena.



Existem somente duas cenas que possuem um caminho de volta à Cena 0: Cena 5 e Cena 6. Essas duas cenas levam de volta à Cena 0, porque elas representam a conclusão da história. Portanto, quando a história terminar faz sentido começar outra vez desde o início. Então, Ellie isolou o código na função `changeScene()` que manipula a alteração dessas duas cenas de volta à Cena 0:

```
else if (curScene == 5) {
  curScene = 0;
}
else if (curScene == 6) {
  curScene = 0;
}
```

As Cenas 5 e 6 definem a variável curScene, mas não fazem nada para a variável message.

Embora não haja nada que se destaque nesse código simples, dê mais uma olhada no código na parte superior da função `changeScene()`, que se encarrega da troca de imagem da cena e da exibição do texto de descrição de cena.

```
document.getElementById("sceneimg").src = "scene" + curScene + ".png";
alert(message);
```

Exibe o texto de descrição de cena, que está armazenado na variável message.

!= Psiu, não tenho nada para lhe dizer...

O problema com o código de As Aventuras de um Boneco de Palito é que ele sempre exibe uma caixa de alerta com a mensagem de descrição de cena, mesmo quando não há mensagem para exibir, como no caso da Cena 0, quando recomeça a aventura. Mas como você pode verificar se a variável `message` de fato contém o texto de descrição de cena?

A solução envolve a verificação de texto vazio ("") da variável `message` antes de exibir a caixa de alerta. Ou expressando de uma outra forma, a caixa de alerta só é exibida se a variável `message` **não for igual** a uma string vazia. Parece uma maneira, às avessas, de resolver o problema, mas lembre-se de que você está tentando propor um teste true/false para quando estiver tudo certo para exibir a caixa de alerta.

Assim como o operador de igualdade (==) que permite verificar se dois itens são idênticos, o operador de desigualdade (!=) verifica se dois itens são diferentes.

Isso é verdadeiro (true) para qualquer cena, exceto a Cena 6.

```
if (curScene != 6)
  alert("Ainda bem que você não foi comido pelo ogro.");
```

Aponte seu lápis

Reescreva o código que exibe a mensagem de descrição de cena de As Aventuras de um Boneco de Palito em uma caixa de alerta, mas dessa vez certifique-se de só exibir a caixa de alerta se a mensagem realmente possuir dados de texto.

.....

.....

.....

.....

Aponte seu lápis
Solução

Reescreva o código que exibe a mensagem de descrição de cena de As Aventuras de um Boneco de Palito em uma caixa de alerta, mas dessa vez certifique-se de só exibir a caixa de alerta se a mensagem realmente possuir dados de texto.

Isso é verdadeiro (true) somente se a mensagem contiver uma string com texto.

```
if (message != )
  alert(message);
```

Decisões qualificadas com operadores de comparação

Igualdade e desigualdade não são os únicos operadores de comparação que você provavelmente vai achar útil, ao criar condições de teste e tomar decisões em seu código JavaScript.

True se x for MAIOR QUE y, caso contrário false

Igualdade

```
x == y
```

True se x for IGUAL a y, caso contrário false

Desigualdade

```
x != y
```

True se x for DIFERENTE de y, caso contrário false

Menor que

```
x < y
```

True se x for MENOR QUE y, caso contrário false

Maior que

```
x > y
```

Negação

```
!x
```

False se x for true, true se x for false.

Menor que ou igual a

```
x <= y
```

True se x for MENOR QUE OU IGUAL A y, caso contrário false.

Maior que ou igual a

```
x >= y
```

True se x for MAIOR QUE OU IGUAL A y, caso contrário false

Os operadores JavaScript, como esses operadores de comparação, são usados para criar expressões que são blocos de código JavaScript associados de alguma forma a um único valor. Expressões representadas como operadores de comparação oferecem um resultado booleano (true/false), que as auxilia a construir a lógica da tomada de decisão usando instruções `if/else`.

PRESTE ATENÇÃO!

= e = = são completamente diferentes.

Quando você tiver intenção de comparar dois valores de igualdade, certifique-se de usar = =, e não =. Caso contrário, você acabará atribuindo um dos valores e possivelmente criando todos os tipos de bugs: novos e raros.

Não existem perguntas idiotas

P: Por que o operador de negação só usa um valor único?

R: Enquanto a maioria dos operadores de comparação exige dois operandos, o operador de negação exige apenas um. E o seu trabalho é muito simples: reverter o valor do operando. Então, `true` se torna `false` e `false` se torna `true`.

P: Eu já vi um operador de negação ser usado em um valor que não é uma comparação. Como isso funciona?

R: O código que usa o operador de negação em um valor de não-comparação está se aproveitando de um detalhe em relação a como o JavaScript determina a "fidelidade" de um valor. Se você usa um valor de não-comparação em uma situação onde se espera uma comparação, qualquer valor diferente de `null`, `0` ou `""` será automaticamente interpretado como `true`. Em outras palavras, a presença de dados é considerada um valor `true`, do ponto de vista da comparação. Então, quando ao vir o operador de negação ser usado em um valor de não-comparação, `null`, `0` e `""` são negados para `true`, enquanto todos os outros valores são negados para `false`.

P: Espere, o que é null?

R: `null` é um valor especial em JavaScript que representa a ausência de dados. Faz mais sentido no contexto de objetos, que será abordado nos Capítulos 9 e 10.

Exercício

Esse código é capaz de exibir uma mensagem positiva sobre As Aventuras de um Boneco de Palito. Que valores os quatro valores `a`, `b`, `c` e `d` devem ter para preencher a mensagem com sucesso?

```
var quote = "";

if (a != 10)
  quote += "Algum cara";
else
  quote += "Eu";
if (b == (a * 3)) {
  if (c < (b / 6))
    quote += "não me preocupo";
  else if (c >= (b / 5))
    quote += "não consigo me lembrar";
  else
    quote += "amo";
}
else {
  quote += "realmente odeia";
}
if (!d) {
  quote += "Boneco de Palito";
}
else {
  quote += "Rocha, papel, tesoura";
}

alert(quote + "Aventura!");
```

a = ...

b = ...

c = ...

d = ...

Solução do Exercício

Esse código é capaz de exibir uma mensagem positiva sobre As Aventuras de um Boneco de Palito. Que valores os quatro valores a, b, c e d devem ter para preencher a mensagem com sucesso?

```
var quote = "";

if (a != 10)
  quote += "Algum cara";
else
  quote += "Su";
if (b == (a * 3)) {
  if (c < (b / 6))
    quote += "não me preocupo";
  else if (c >= (b / 5))
    quote += "não consigo me lembrar";
  else
    quote += "amo";
}
else {
  quote += "realmente odeia";
}
if (!d) {
  quote += "Boneco de Palito";
}
else {
  quote += "Pedra, papel, tesoura";
}

alert(quote + "Aventura!");
```

a tem que ser igual a 10.

b tem que ser igual a 10 x 3.

c tem que ser igual 5.

a = 10

b = 30

c = 5

d = false

d tem que ser false para que !d seja true.

A mensagem positiva que estávamos procurando

Comentários, placeholders e documentação

As Aventuras de um Boneco de Palito é um bom exemplo de um script que possui blocos de código inacabados porque a história ainda está sendo desenvolvida. Por exemplo, as Cenas 8 e 9 são cenas de continuação que ainda aguardam algum trabalho criativo da Ellie. Pode ser útil sinalizar as áreas de código inacabado com notas em placeholders para que você não se esqueça de preencher os detalhes mais tarde. Os **comentários** JavaScript possibilitam a adição de notas ao código sem afetar a forma como o código é executado.

Um comentário começa com uma barra dupla.

```
// + Comment
```

O texto do comentário pode ser qualquer coisa que você queira ele é ignorado pelo interpretador de JavaScript.

Os comentários no JavaScript começam com //

Um comentário criado com // se estende das barras até o fim da linha. Para criar um comentário como um placeholder, siga as barras com uma observação que indica que haverá mais código.

Cena 8

Continua...

Essas linhas de código são ignoradas pelo interpretador.

```
else if (curScene == 8) {
  // CONTINUA
}
else if (curScene == 9) {
  // CONTINUA
}
```

Cena 9

Continua...

Os comentários não são apenas para os placeholders. Eles são mais usados para documentar o código, de modo que ele fique mais organizado e fácil de entender. Só porque **agora** você sabe como um fragmento de código funciona não significa que terá essa memória maravilhosa em relação a ele mais tarde. E existe sempre a chance de alguma outra pessoa herdar seu código, e certamente, ela se beneficiará das observações sobre como ele funciona.

```
// Inicializa a cena atual para
a Cena 0 (Intro)
var curScene = 0;
```

Esse comentário explica a inicialização da variável.

A inicialização da variável `curScene` em As Aventuras de um Boneco de Palito está mais clara, graças ao comentário detalhado. Um comentário semelhante poderia ser usado para esclarecer a inicialização da variável `message`.

```
// Limpa a mensagem da cena
var message = "";
```

Mais uma vez, um comentário esclarece o que está acontecendo no código que o precede.

Se você precisa de um comentário que ultrapasse mais de uma linha, você pode criar um comentário com várias linhas.

Os comentários com várias linhas sempre terminam com */.

```
*/ + Início do comentário
Mais comentário
Fim do comentário + */
```

Os comentários com várias linhas sempre começam com /*.

Os comentários de uma única linha começam com //, enquanto os comentários com várias linhas são delimitados por /* e */.

Um comentário com várias linhas pode ter a extensão que você quiser. Você só precisa começar com /* e terminar com */.

```
/* Todas essas linhas de código são um único
comentário grande. É sério, eu não estou
brincando. Sem brincadeira, isso ainda faz parte
do comentário. */
```

Espere aí! Os comentários fazem sentido, mas não entendo porque curScene e as variáveis message são criadas em lugares diferentes. Como é isso?

curScene é criada fora da função changeScene().

```
<script type="text/javascript">
  // Inicializa a cena atual para a Cena 0 (Intro)
  var curScene = 0;

  function changeScene(decision) {
    // Limpa a mensagem de cena
    var message = "";
    ...
  }
</script>
```

message é criada dentro da função changeScene().

Localização, localização, localização das variáveis

Assim como no ramo imobiliário, localização significa tudo para o JavaScript. Neste caso, o lugar onde as variáveis de As Aventuras de um Boneco de Palito são criadas tem muita importância. Em outras palavras, não é por acaso que `curScene` é criada fora da função `changeScene()`, enquanto `message` é criada dentro da função. O motivo é o **escopo**, que controla a vida útil de uma variável, bem como qual código pode acessá-la.

Escopo e contexto: Onde os dados residem

Em JavaScript, o escopo se refere ao **contexto de dados**, como onde os dados residem e como eles podem ser acessados. Dependendo de seu escopo, alguns dados podem ser vistos em qualquer lugar do script, enquanto outros dados são limitados a um bloco de código específico, como uma função. Eis um exemplo de duas variáveis que residem em lugares totalmente diferentes:

```
var x;

function doSomething(z) {
  var y;
  ...
}
```

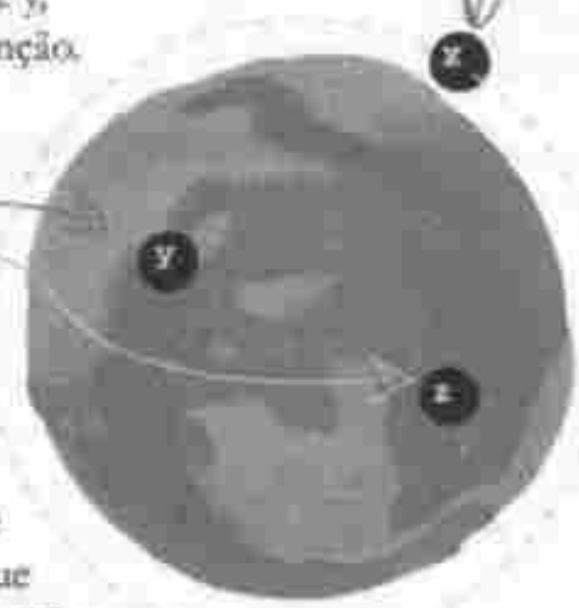
Variável global, visível em todo o script.

Variáveis locais, visíveis apenas dentro da função.

Nesse código, x é considerado uma variável **global**, já que ela é criada fora de qualquer função ou outro bloco de código e, portanto, pode ser vista por todo o script. Ainda mais importante, x permanecerá "viva" enquanto o script estiver executando. Ao contrário de x, y é uma variável **local** cuja visibilidade é confinada ao código dentro da função `doSomething()`. Além disso, y só existe enquanto a função `doSomething()` estiver executando – ela é criada quando a função inicia e depois é destruída quando a função termina.

Por enquanto tudo bem, mas onde isso deixa z, o argumento para a função `doSomething()`? Acontece que os argumentos de função agem como variáveis locais que já foram inicializadas. Portanto, z possui o mesmo escopo de y, significando que ele só pode ser acessado de dentro da função.

Variáveis globais são mantidas nas proximidades por toda a vida de um script.



Variáveis locais são criadas e destruídas conforme determinado por seu escopo.

A visibilidade dos dados fica disponível para quando precisar saber, o que significa que você deve limitar a acessibilidade sempre que possível. Isso ajuda a evitar que os dados sejam alterados inadvertidamente pelo código que não tem nada a ver acessá-lo. Em termos práticos, isso significa que as variáveis locais devem ser usadas sempre que possível.

Poder do Cérebro

Como as variáveis globais e locais se encaixam no código de As Aventuras de um Boneco de Palito?

Verifique o resultado da variável de sua aventura

Recordando as variáveis de As Aventuras de um Boneco de Palito, agora com o conhecimento sobre variáveis locais e globais, é possível ter uma idéia melhor de o porquê as variáveis são criadas em lugares diferentes.

O valor da variável message é redefinido a cada momento através da função changeScene(), portanto ela funciona bem como uma variável local.

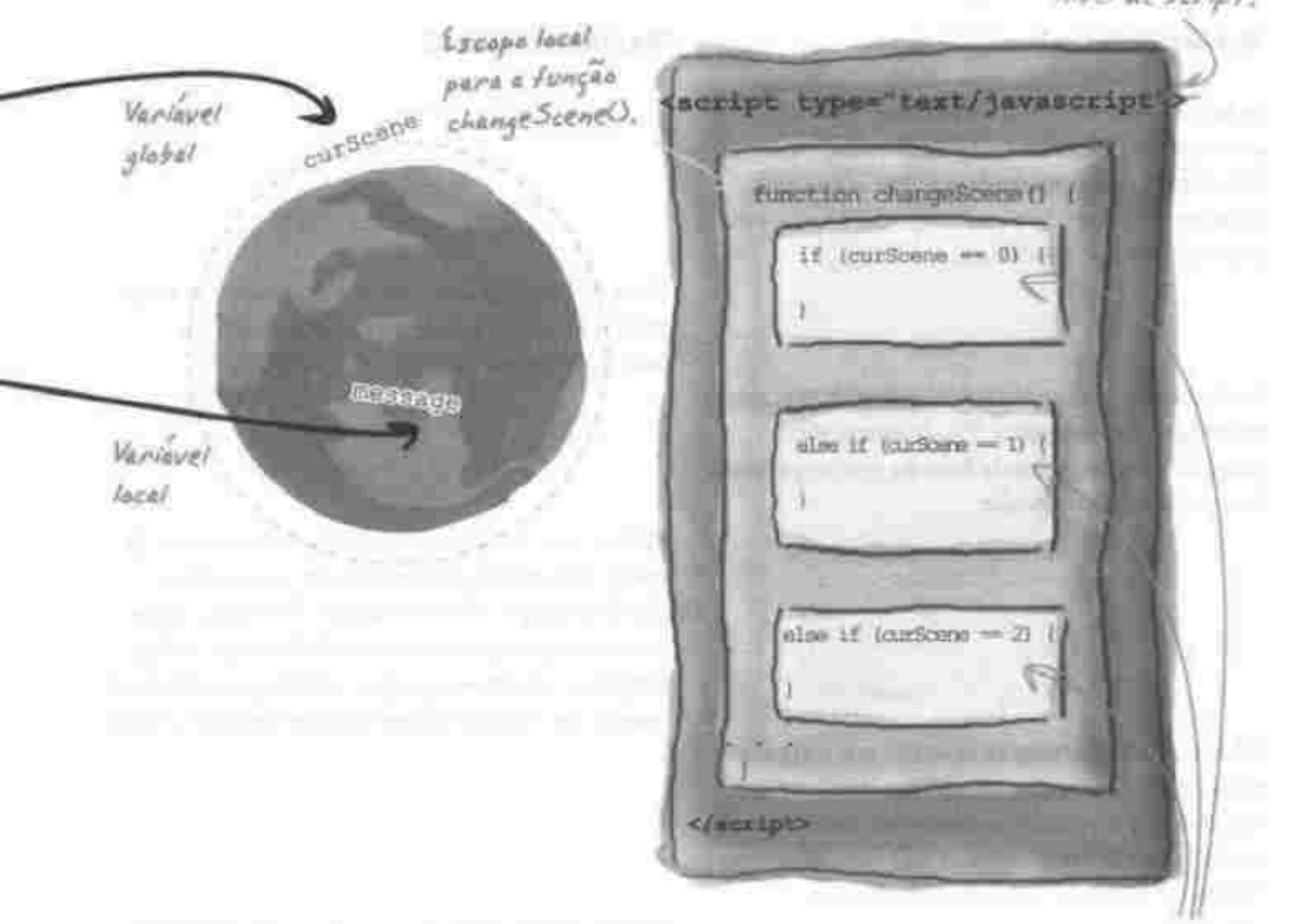
O valor da variável curScene deve ser mantido no meio das chamadas para a função changeScene(), então ela precisa ser uma variável global.

```
<script type="text/javascript">
  // Inicializa a cena atual para a Cena 0 (Intro)
  var curScene = 0;

  function changeScene(decision) {
    // limpa a mensagem de cena
    var message = "";

    if (curScene == 0) {
      curScene = 1;
      message = "Sua jornada começa em uma bifurcação na estrada.";
    }
    else if (curScene == 1) {
      if (decision == 1) {
        curScene = 2
        message = "Você chegou a uma linda...";
      }
      else {
        curScene = 3;
        message = "Você está posicionado em uma ponte...";
      }
    }
    else if (curScene == 2) {
      ...
    }
</script>
```

A questão aqui é a necessidade de preservar o valor de uma variável fora do escopo da função `changeScene()`. A variável `message` é excluída do início da função, portanto seu valor não precisa ser preservado fora da função. A variável `curScene`, por outro lado, é verificada em vários testes de condições `if/else`. Então, esse valor *tem que* persistir no meio das chamadas das funções. **Resultado, message pode ser criada localmente, mas, neste exemplo, curScene tem que ser global.**

Onde residem meus dados?

Se o escopo ainda intrigá-lo um pouco, isso talvez possa ajudar a lembrar as diferentes partes de um script como áreas independentes onde os dados podem residir. Por exemplo, o script de As Aventuras de um Boneco de Palito possui diversos escopos que você poderia usar para armazenar dados em outro lugar.



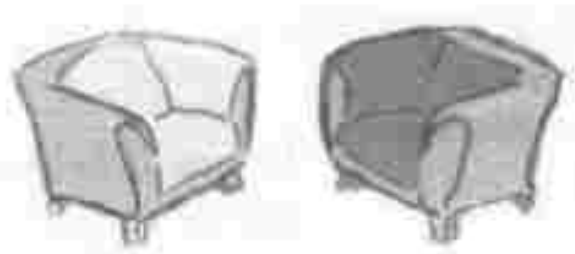
Só existe um único escopo global para criar variáveis globais, o resto é local. Qualquer coisa criada em nível global pode ser vista por todo o script, enquanto os dados locais só podem ser vistos e usados dentro de seu escopo limitado.

Não existem perguntas idiotas

P: Eu tenho alguns dados. Que tipo de variável devo usar para armazená-los: local ou global?

R: A maneira como você usa os dados determinará se ela precisa ser local ou global. Mas já que perguntou, a regra geral é tentar criar todas as variáveis como local e usar a global somente como recurso, caso a local não funcione.

Conversa informal



Conversa dessa noite: **A variável Local e a variável Global discutem sobre a importância da localização ao encontrar um lar para os dados.**

Variável Local

Acho útil se concentrar apenas no que está acontecendo à minha volta. Na verdade, eu nem tenho como lhe contar sobre o que está acontecendo fora da vizinhança onde vivo, e eu gosto muito que seja assim.

Embora isso pareça tentador, gosto da segurança de ambientes tranqüilos. Eu descanso fácil sabendo que ninguém de fora da minha pequena área pode me incomodar.

Ai! Não tenho certeza se acredito em toda essa coisa de reencarnação, mas posso lhe dizer que sou cada bit de armazenamento de dados tão prático quanto você. Apenas não quero me colocar lá fora para qualquer um me ver.

E quando um script precisa guardar alguma informação isolada de uma certa seção do código, ele me procura por causa do meu dom para a discrição.

E é por isso que as pessoas ainda nos acham úteis.

Variável Global

Cara, você precisa realmente expandir sua visão de mundo. Saia e viaje um pouco, verifique outras partes do universo do script.

Talvez, mas você tem conhecimento do fato de que a sua vidinha é inexpressiva no contexto geral do script. Você é criada e destruída várias vezes, sempre que o seu mundinho vai e volta, enquanto eu estou aqui por um longo período de tempo. Se o script estiver aqui, eu também estou.

É justo. E eu admitirei que me usam de maneira inapropriada uma vez ou outra, mas o meu lado positivo, sempre conservando o meu valor nos bons e maus momentos, é suficiente para compensar os problemas. Quando um script precisa de um fragmento de dados que lembre seu valor e que esteja disponível em qualquer lugar, ele me procura.

Que ótimo, mas algum dia eu assumirei a acessibilidade e a persistência em detrimento da privacidade.

Não existem perguntas idiotas

P: O que acontece se o código JavaScript existente é colocado dentro de um comentário?

R: Nada! Comentários são completamente ignorados pelo interpretador de JavaScript, portanto tudo que você coloca dentro de um comentário é desconsiderado quando o interpretador inicia a execução do código de script. Sabendo disso, os comentários podem ser usados como um meio de desabilitar temporariamente fragmentos de código para identificar um problema ou tentar abordagens diferentes de codificação.

P: Uma linha de código JavaScript pode ter um comentário de uma única linha no final?

R: Sim. E neste caso, o código ainda é executado, porque não faz parte do comentário. Um comentário de uma única linha não ocupa necessariamente uma linha inteira – o comentário começa a partir de // até o fim da linha. Então, se // antecede um fragmento de código, ainda assim ele será executado normalmente.

P: Por que os comentários não terminam com um ponto-e-vírgula?

R: Porque eles não são instruções JavaScript. Comentários são rótulos que não fazem nada além de descrever ou fornecer informações adicionais sobre o código, mais ou menos como as notas de rodapé em um livro. O mais importante a ser lembrado é que o interpretador JavaScript ignora todos os comentários – os comentários estão ali para o cérebro humano e não para o JavaScript.

P: O que significa "nível de script" em relação à criação de variáveis globais?

R: "Nível de script" é o nível superior do código de script, que se encontra exatamente dentro da tag `<script>`. O significado do "nível de script" é que ele se encontra fora de qualquer função ou outro bloco de código e, no entanto, tudo que for criado no "nível de script" é considerado global. Isso significa que tudo que for criado no "nível de script" vive por toda a vida do script e pode ser acessado por qualquer código dentro da página.

P: Se eu criar uma variável dentro de uma instrução composta, ela é uma variável local?

R: Sim. Uma instrução composta estabelece um novo nível de escopo, portanto tudo que for criado dentro de uma instrução composta possui um escopo local dentro dessa instrução. Tais variáveis podem ser consideradas como variáveis temporárias, já que elas são criadas e destruídas toda vez que o fluxo da execução do script entra e sai da instrução composta.

P: Escopo, fluxo, execução... essa coisa de variáveis local e global parece realmente complexo. É assim tão difícil quanto parece?

R: Nem tanto. O mais importante a ser lembrado é que as variáveis locais são perfeitas para armazenar informações temporárias que você não precisa se lembrar do lado de fora de uma função ou um outro bloco de código. Se você quer que os dados estejam disponíveis por toda a vida do script, então deve torná-los uma variável global. Por incrível que pareça, a maioria dos dados de script tende a ser mais temporário do que poderia pensar inicialmente, o que significa que, provavelmente, usará mais variáveis locais do que variáveis globais.

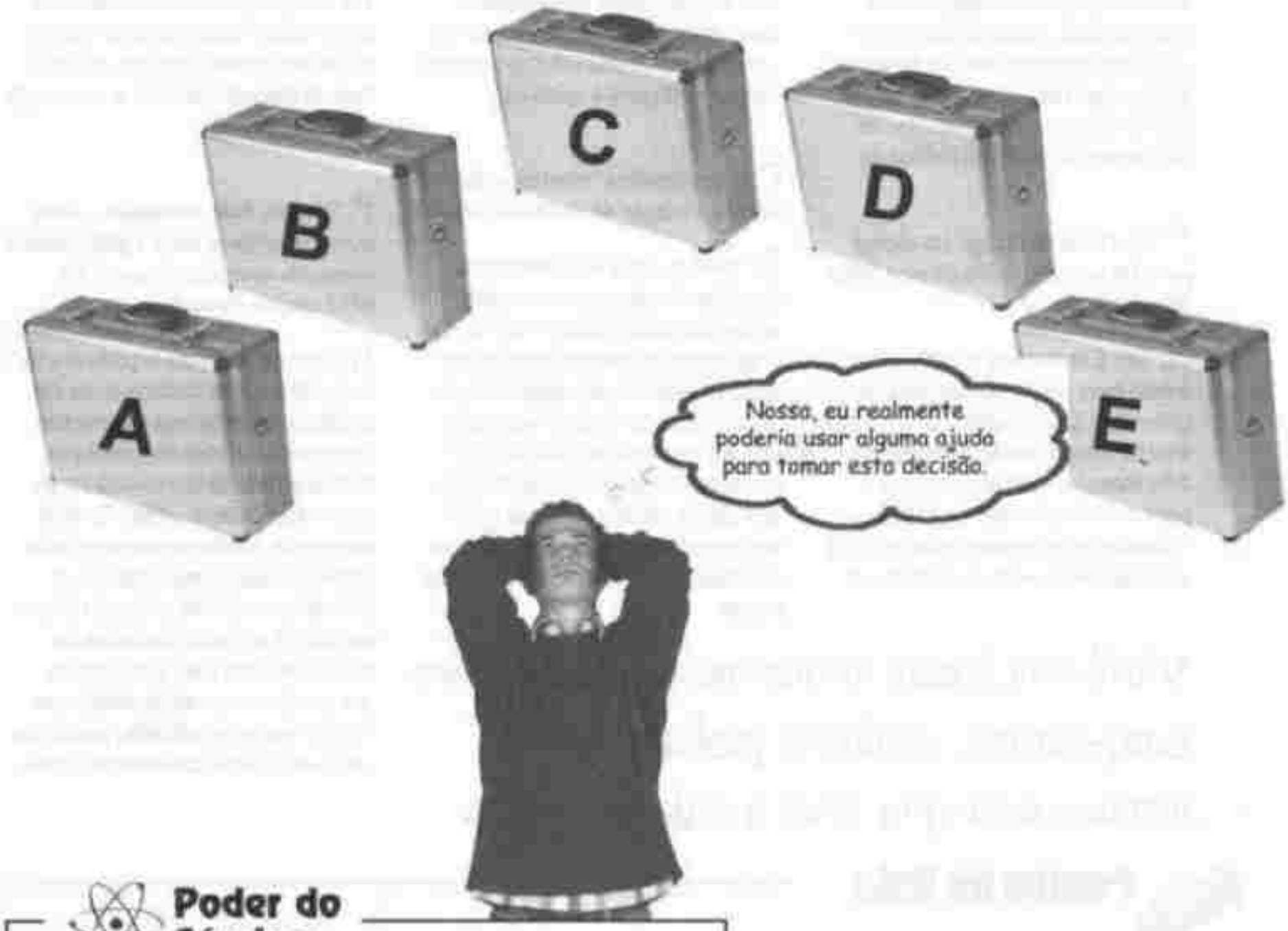
Variáveis locais armazenam informações temporárias. Variáveis globais são armazenadas por toda a vida do script.

Pontos de Bala

- **Comentários** são uma ótima maneira de lembrá-lo sobre a posterior adição de código.
- Não tenha receio de usar muitos comentários para documentar seu código, fazendo com que ele fique mais fácil de entender.
- Use uma barra dupla (//) para iniciar um **comentário de uma única linha**.
- **Comentários de várias linhas** começam com /* e terminam com */.
- Variáveis globais são criadas no nível de script, fora de qualquer função ou outro corpo de código e são mantidas nas proximidades durante a **vida do script**.
- Variáveis locais são criadas (e destruídas) dentro de um corpo de código, e só pode ser acessada dentro desse código.
- Variáveis locais são **preferidas** em detrimento das variáveis globais, porque seu acesso é rigidamente mais controlado.

Cinco opções

Você se lembra do Eric, nosso concorrente do show no início do capítulo? Parece que Eric acabou com seus donuts e avançou para uma última rodada de *Quer fazer uma aposta?* O problema é que ele agora enfrenta uma decisão extremamente desafiadora... ele tem que escolher entre uma das cinco opções.



Poder do Cérebro

Como você codificaria uma decisão JavaScript que envolvesse cinco opções diferentes?

Cinco opções

Boa idéia! Embora a instrução `if/else` seja direcionada para a tomada de decisão entre duas coisas, várias delas podem ser aninhadas para escolher entre tantas coisas que queira.

```
if (chosenCase == "A")
  openCase("A");
else if (chosenCase == "B")
  openCase("B");
else if (chosenCase == "C")
  openCase("C");
else if (chosenCase == "D")
  openCase("D");
else if (chosenCase == "E")
  openCase("E");
```

Esse código funciona, mas o último caso exige que cada teste de condição seja avaliado, que é ineficaz.

Aninhar if/else pode ficar complicado

As instruções `if/else` funcionam bem... mas elas não têm toda essa eficiência, primeiro porque elas não foram criadas para tomarem decisões que envolvam mais de duas possibilidades. Para entender quantos testes booleanos ocorrem no processo de escolha do último caso, o Caso E. Todos os cinco testes de condições são avaliados, o que é um pouco ineficaz.

Não seria um sonho se você pudesse escolher entre mais de duas coisas sem ter que usar todas aquelas instruções if/else ineficazes?

Instruções switch possuem vários cases

O JavaScript tem uma instrução para tomada de decisão apenas para decidir entre várias opções. Ao contrário da instrução `if/else`, que se adapta melhor à escolha entre **duas coisas**, a instrução `switch/case` permite escolher com mais eficácia entre qualquer **quantidade de coisas**. Vamos analisar o dilema de Eric através das lentes de uma instrução `switch/case`:

A instrução switch/case escolhe entre mais de duas coisas com eficiencia.

O "case" em switch/case não tem nada para fazer com as pastas de metal do Eric.

```
switch (chosenCase) {
case "A":
  openCase("A");
  break;
case "B":
  openCase("B");
  break;
case "C":
  openCase("C");
  break;
case "D":
  openCase("D");
  break;
case "E":
  openCase("E");
  break;
}
```

O valor da pasta escolhida pelo Eric faz parte da informação controladora na instrução switch/case.

O código de ação é colocado imediatamente abaixo de sua instrução case correspondente.

Cada possibilidade de opção é codificada usando uma instrução case.

A instrução break é necessária para terminar cada desvio de decisão para sair imediatamente de toda a instrução switch/case.

A instrução switch/case está estruturada como uma grande instrução composta.

Exercício

Fato ou ficção? A instrução `switch/case` consegue fazer qualquer coisa que a instrução `if/else` faz.

☐ Fato ☐ Ficção

Solução do Exercício

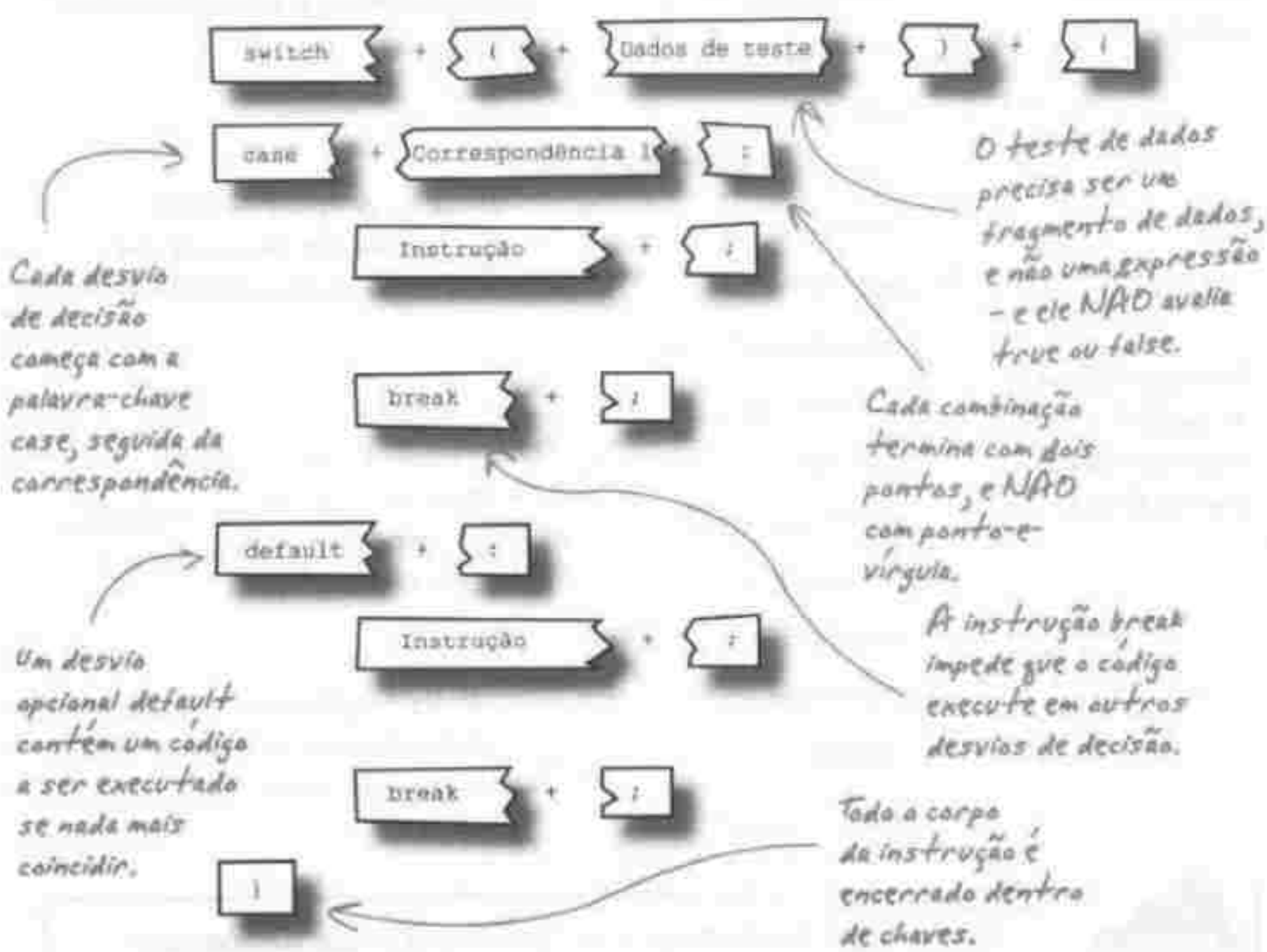
Fato ou ficção? A instrução `switch/case` consegue fazer qualquer coisa que a instrução `if/else` faz.

☐ Fato ☑ Ficção

Ficção. Ao contrário de `if/else`, o teste de dados que controla uma instrução `switch/case` não pode ser uma expressão – tem que ser simplesmente um fragmento de dados.

Dentro da instrução switch

Agora que você viu uma instrução `switch/case` em ação, vamos eliminá-la e examinaremos o formato geral da instrução.



Instruções switch case: escreva a sua

Criar uma instrução `switch/case` é admitidamente mais complicado do que criar uma instrução `if/else`, mas é muito mais eficiente lidar com mais de duas possibilidades de resultado. Eis o processo:

1. Inclua o teste de dados entre parênteses e abra a instrução composta ({).
2. Escreva o case correspondente seguido de dois pontos (:).
3. Escreva o código a ser executado se houver uma **correspondência**. Ela pode ter várias linhas de código – não há necessidade de uma instrução composta.
4. Adicione uma instrução break – lembre-se do ponto-e-vírgula (;).
5. Opcionalmente, inclua um desvio padrão para quando não houver correspondência.
6. Feche a instrução composta (}).

Fico imaginando se uma instrução switch-case poderia ser usada para tornar As Aventuras de um Boneco de Palito mais eficiente...

Não existem perguntas idiotas

P: Então, uma instrução `switch/case` toma uma decisão usando uma expressão true/false?

R: Correto. Ao contrário de uma instrução `if/else`, `switch/case` usa um fragmento de teste de dados para tomar decisões. É como ela aceita mais de dois resutados.

P: Então, cada `case` correspondente é apenas uma correspondência no teste de dados?

R: Sim. A idéia é que você use uma variável como teste de dados e, em seguida, use valores literais para aplicar em cada correspondência.

Interrompa por medida de segurança.

PRESTE ATENÇÃO! *Evite que o código seja executado acidentalmente terminando cada correspondência switch-case com uma instrução break.*

P: O que acontece se você não incluir todas as instruções `break` em uma `switch/case`?

R: Podem ocorrer resultados inesperados. As instruções `break` servem como divisores entre cada seção de código de ação em uma instrução `switch/case`. Sem elas, todo o código de ação executaria como um grande bloco, o que destruiria toda a proposta de tomar decisões diferentes. Quando uma correspondência é feita em uma instrução `switch/case`, o código abaixo do `case` correspondente é executado até que ele encontre uma instrução `break`. Somente então a instrução `switch/case` sai.

Switch exposto

Entrevista da semana: Os manda-chuvas e os tomadores de decisão

Use a Cabeça: Estou contente que você esteja disposta a conversar conosco. Então, diga-nos como você se descreveria em apenas uma palavra?

Switch: Seletiva.

Use a Cabeça: Está disposta a dar mais detalhes?

Switch: Eu possibilito a escolha entre várias coisas diferentes. Embora algumas situações envolvam decisões explícitas, existem muitas situações que exigem, digamos, mais sutileza. É onde eu entro.

Use a Cabeça: Mas dizem que If pode fazer a mesma coisa, às vezes com menos código.

Switch: Talvez seja verdade. E você pode cortar um pedaço de madeira com um martelo se você golpeá-lo várias vezes. Pessoalmente, prefiro usar uma serra. A realidade é que todos têm sua peculiaridade, e a minha é escolher de maneira eficiente entre várias coisas diferentes. Não tenho nenhuma objeção em relação a If, mas ela é uma ferramenta mais apropriada para uma outra tarefa.

Use a Cabeça: Você mencionou eficiência. Conte-nos como ela se inclui no que você faz.

Switch: Bem, sou estruturada para tomar uma decisão baseada no valor de um fragmento de dados, e tudo o que faço é comparar esse fragmento de dados com possíveis correspondências para determinar que código executar. É isso. Eu não me dou ao trabalho de tentar avaliar as expressões, e não exijo aninhamento ou qualquer coisa bonitinha como essa para escolher entre vários resultados. Se você quiser tomar uma decisão rápida com base em um fragmento de código, sou a pessoa certa!

Use a Cabeça: Fale-nos sobre o seu amigo Break. Soubemos que você não vive um só dia sem ele.

Switch: Isso é verdade. Sem Break, eu estaria em apuros, porque não saberia separar os diferentes fragmentos de código de ação. Break me informa quando uma seção de código termina a execução, e eu posso sair sem executar acidentalmente nenhum outro código.

Use a Cabeça: Entendi. E em relação ao Case, vocês também não são muito íntimos?

Switch: Claro que sim. Case e eu temos uma relação muito íntima, primeiro porque Case me informa todas as possibilidades de correspondências para um determinado fragmento de teste de dados. Sem Case, eu não teria base para tomar uma decisão.

Use a Cabeça: Então, entendo que Case expõe as diferentes possibilidades de correspondências, e você usa essas correspondências para determinar o que fazer. Mas o que acontece quando o teste de dados não possui correspondente?

Switch: Depende. Se nenhum código especial for adicionado para lidar com um cenário de "sem correspondência", então nada acontece. No entanto, o meu bom amigo Default possibilita a execução de um bloco especial de código só no evento em que nenhuma correspondência foi encontrada.

Use a Cabeça: Uau! Eu não percebia isso. Default se dá bem com Case?

Switch: Na verdade, muito bem. Eles não pisam um no calo do outro, porque não competem por atenção. Case manipula todas as correspondências, enquanto Default cuida das situações quando nada mais corresponde. Muito aqui entre nós, acho que Case está um pouco aliviado com a presença de Default porque ele fica nervoso quando nada coincide.

Use a Cabeça: Entendo. Bem, estamos próximos de acabar o nosso tempo. Alguma lembrança antes de você partir?

Switch: Claro. Lembre-se de que não existe nada pior do que a indecisão. Ninguém gosta de indecisos. Não é por existir inúmeras possibilidades que significa que você tem que jogar tudo para o alto e desistir. Chame-me e eu tentarei fazer o melhor possível para ajudá-lo a tomar uma decisão que funcione bem em seu script.

Aponte seu lápis

Converta as duas primeiras cenas do código de As Aventuras de um Boneco de Palito de modo a usar a instrução switch/case no lugar de if/else.

```
...
if (curScene == 0) {
  curScene = 1;
  message = "Sua jornada começa em uma bifurcação na estrada.";
}
else if (curScene == 1) {
  if (decision == 1) {
    curScene = 2
    message = "Você chegou à uma linda casinha na floresta.";
  }
  else {
    curScene = 3;
    message = "Você está posicionado em uma ponte com vista para um rio
tranquilo.";
  }
}
...
```

Eis a versão original do código que usa if/else.

Aponte seu lápis
Solução

Converta as duas primeiras cenas do código de As Aventuras de um Boneco de Palito de modo a usar a instrução switch/case no lugar de if/else.

```
...
if (curScene == 0) {
  curScene = 1;
  message = "Sua jornada começa em uma bifurcação na estrada.";
}
else if (curScene == 1) {
  if (decision == 1) {
    curScene = 2
    message = "Você chegou a uma linda casinha na floresta.";
  }
  else {
    curScene = 3;
    message = "Você está posicionado em uma ponte com vista para um rio
tranqüilo.";
  }
}
...
```

Eis a versão original do código que usa if/else.

Define o novo número de cena e o texto de mensagem de descrição de cena, como na versão if/else.

Cada par de case corresponde a um número de cena.

Dentro de cada case, ainda faz sentido aderir à if/else para manipular a decisão do usuário na história.

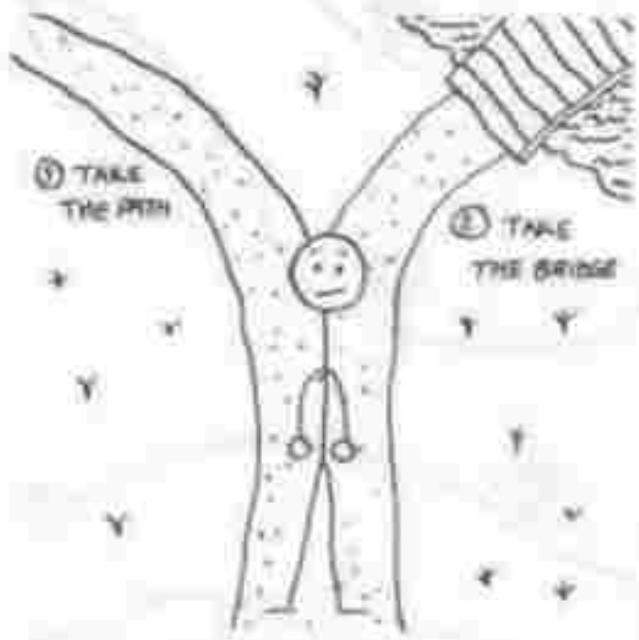
As cenas restantes seguem uma estrutura semelhante.

Fecha a instrução switch/case com uma }

```
switch (curScene) {
case 0:
  curScene = 1;
  message = "Sua jornada começa em uma bifurcação na estrada.";
  break;
case 1:
  if (decision == 1) {
    curScene = 2
    message = "Você chegou a uma linda casinha na floresta.";
  }
  else {
    curScene = 3;
    message = "Você está posicionado em uma ponte com vista para um rio tranqüilo.";
  }
  break;
...
}
```

Uma mudança em As Aventuras de um Boneco de Palito: test-drive

Após reformular completamente a lógica da tomada de decisão de As Aventuras de um Boneco de Palito, Ellie está se coçando para ver o resultado. As mudanças são imediatamente visíveis, à medida que você navega pela história...



Tudo parece exatamente igual

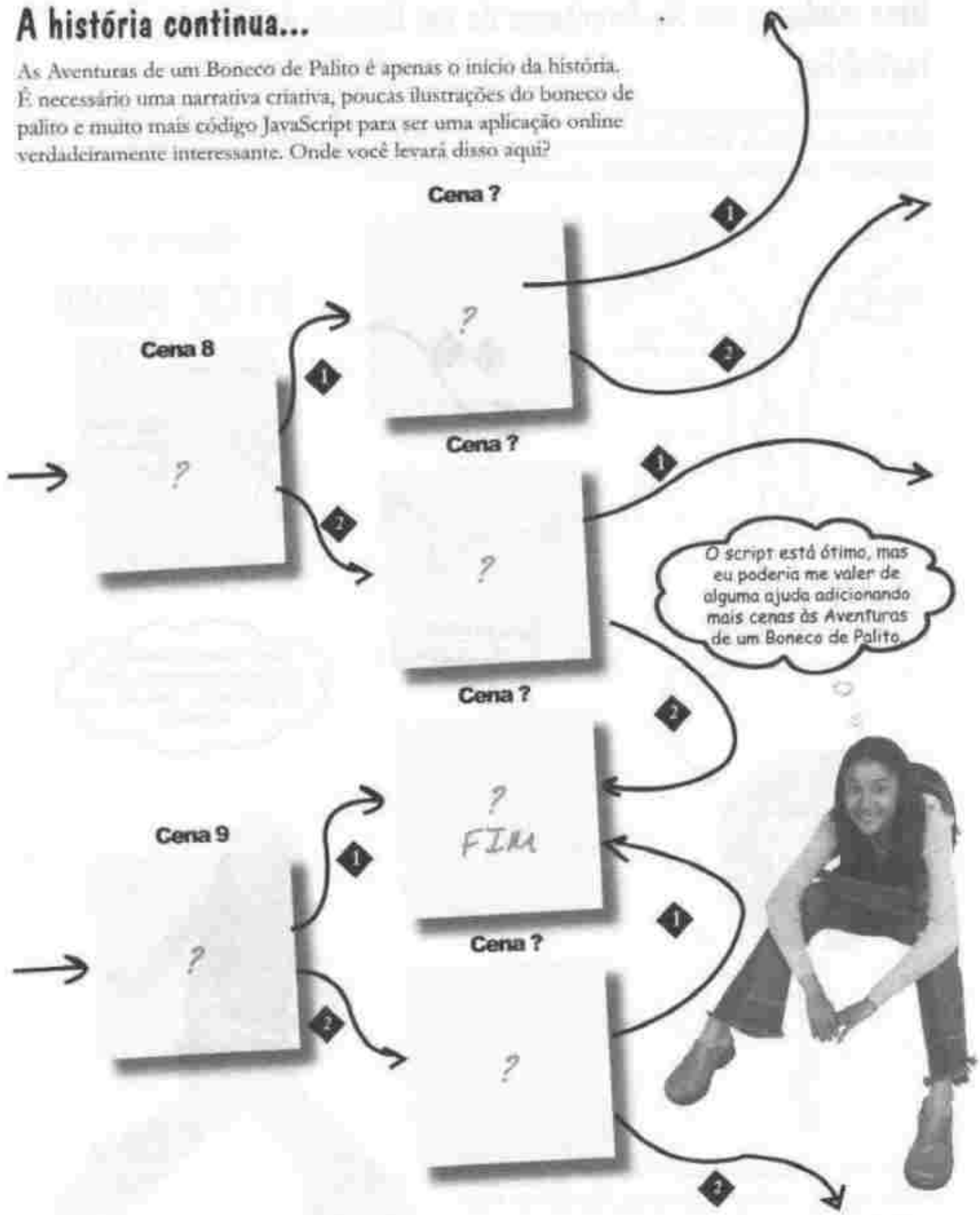
Pare aí mesmo!

Nem todas as melhorias no script podem ser vistas da superfície... isso não me incomoda.

Espere, não há nada diferente! É porque as alterações de switch/case em As Aventuras de um Boneco de Palito só afetam a estrutura do código, não a sua aparência. Eis um exemplo de como algumas alterações na codificação ocorrem puramente nos bastidores... literalmente!

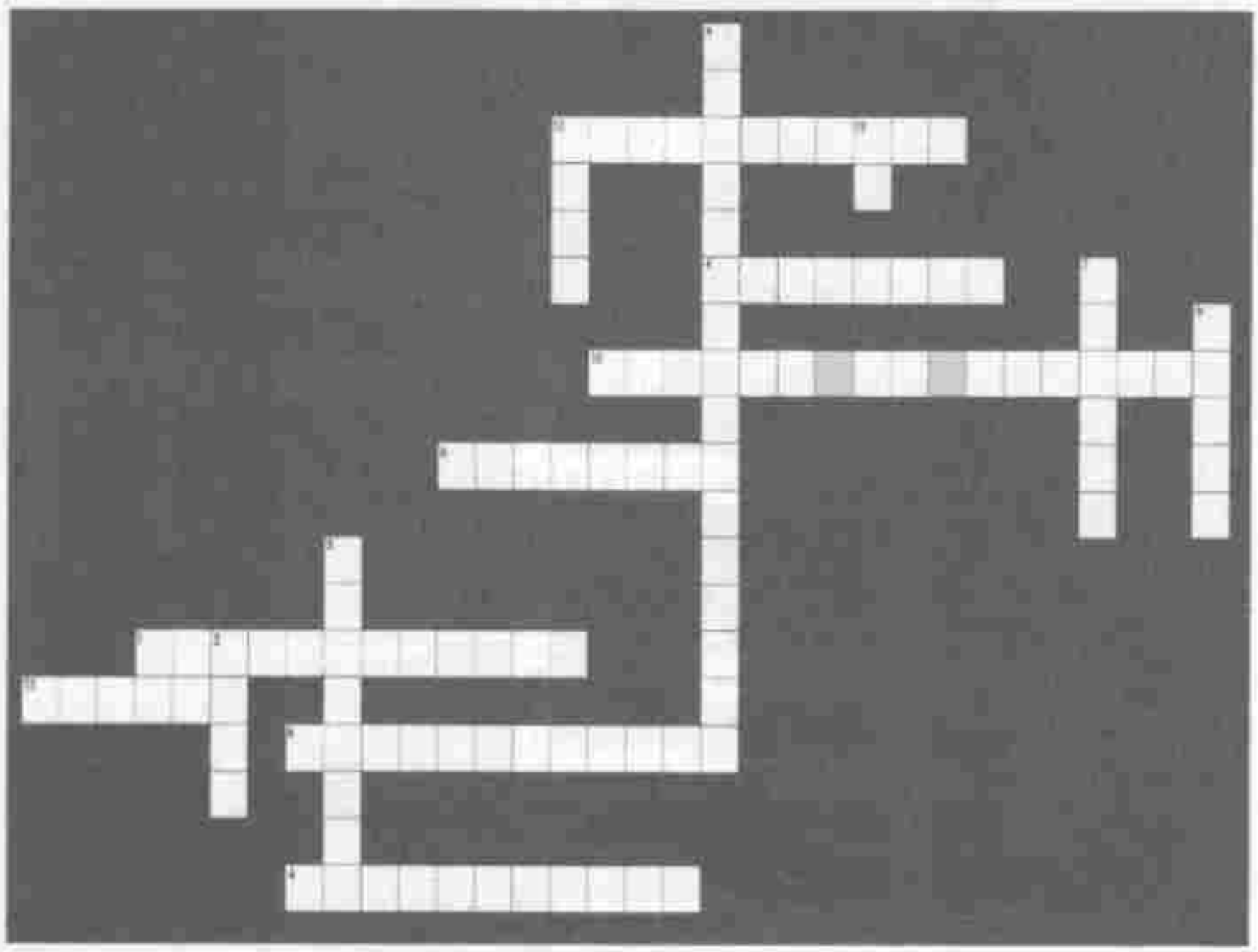
A história continua...

As Aventuras de um Boneco de Palito é apenas o início da história. É necessário uma narrativa criativa, poucas ilustrações do boneco de palito e muito mais código JavaScript para ser uma aplicação online verdadeiramente interessante. Onde você levará disso aqui?



Cruzadas do JavaScript

Eis uma decisão fácil. Essa é a hora de fazer uma pausa e arrebentar nas palavras-cruzadas? Claro!



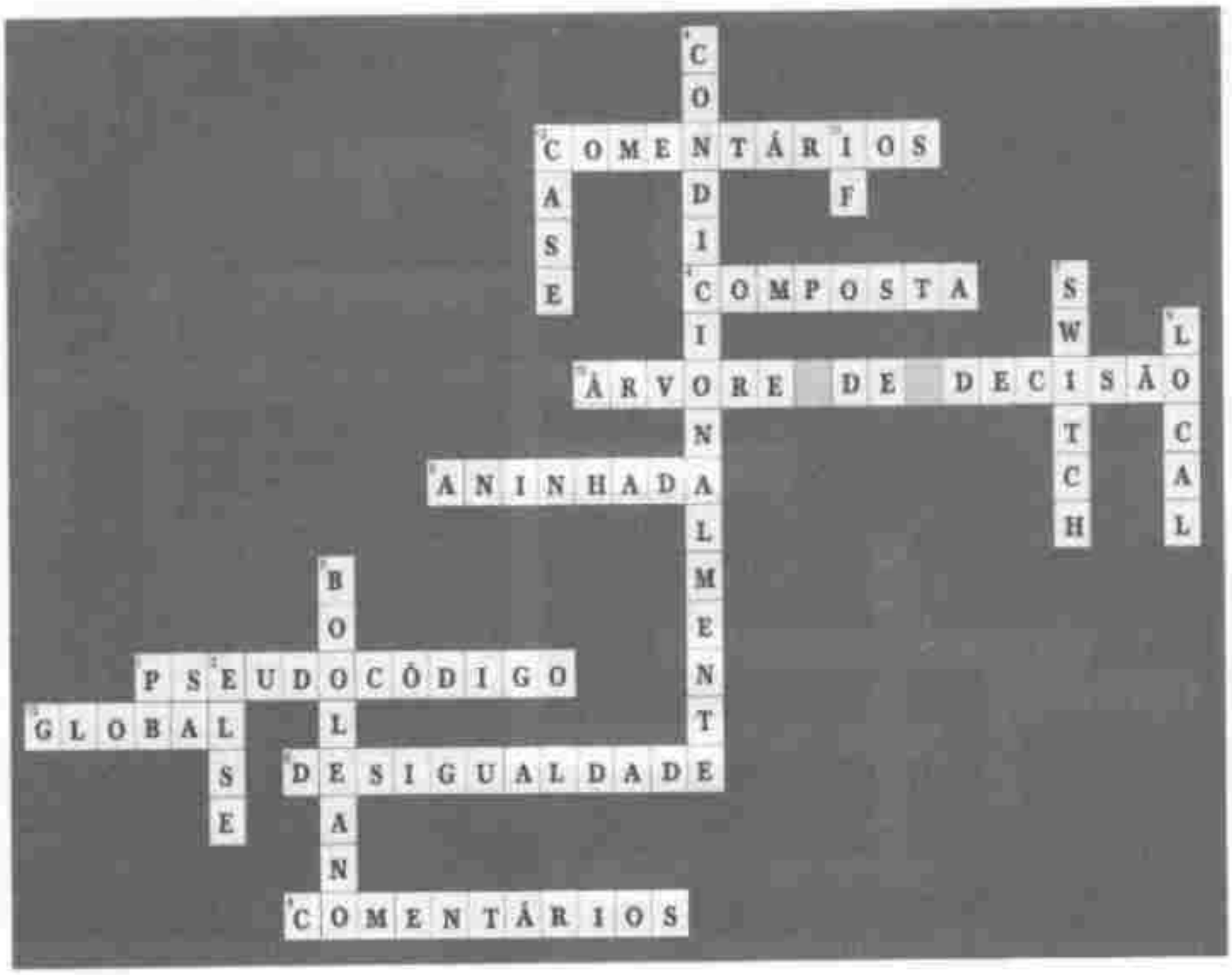
Horizontal

1. Escrevê-lo primeiro fica mais fácil para escrever um código JavaScript complexo.
4. Use-os para documentar seu código.
5. Esse tipo de instrução é composto de várias instruções.
6. Quando uma instrução if é colocada dentro de uma outra, dizem para ser .....
8. O operador != a testa para
10. Você pode usar uma dessas para ajudar a visualizar um grupo de decisões complexas.
13. Todo o script possui acesso a esse tipo de variável.

Vertical

2. Faça isso, ... faça aquilo.
3. Esses tipos de operadores retornam um resultado true/false.
4. Como o código é executado quando ele faz parte de uma instrução if-else.
7. Uma instrução que permite tomar uma decisão com base no valor de um fragmento de dados.
9. Uma variável que possui escopo limitado.
11. Essa instrução permite executar incondicionalmente um fragmento de código.
12. Cada desvio de decisão dentro de uma instrução switch possui um desses.

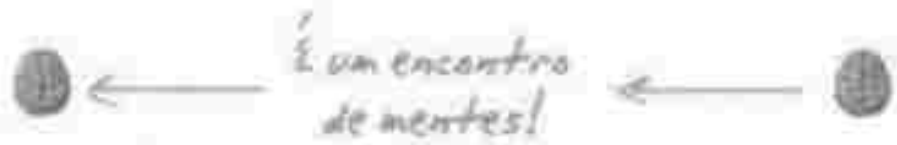
Solução das Cruzadas do JavaScript

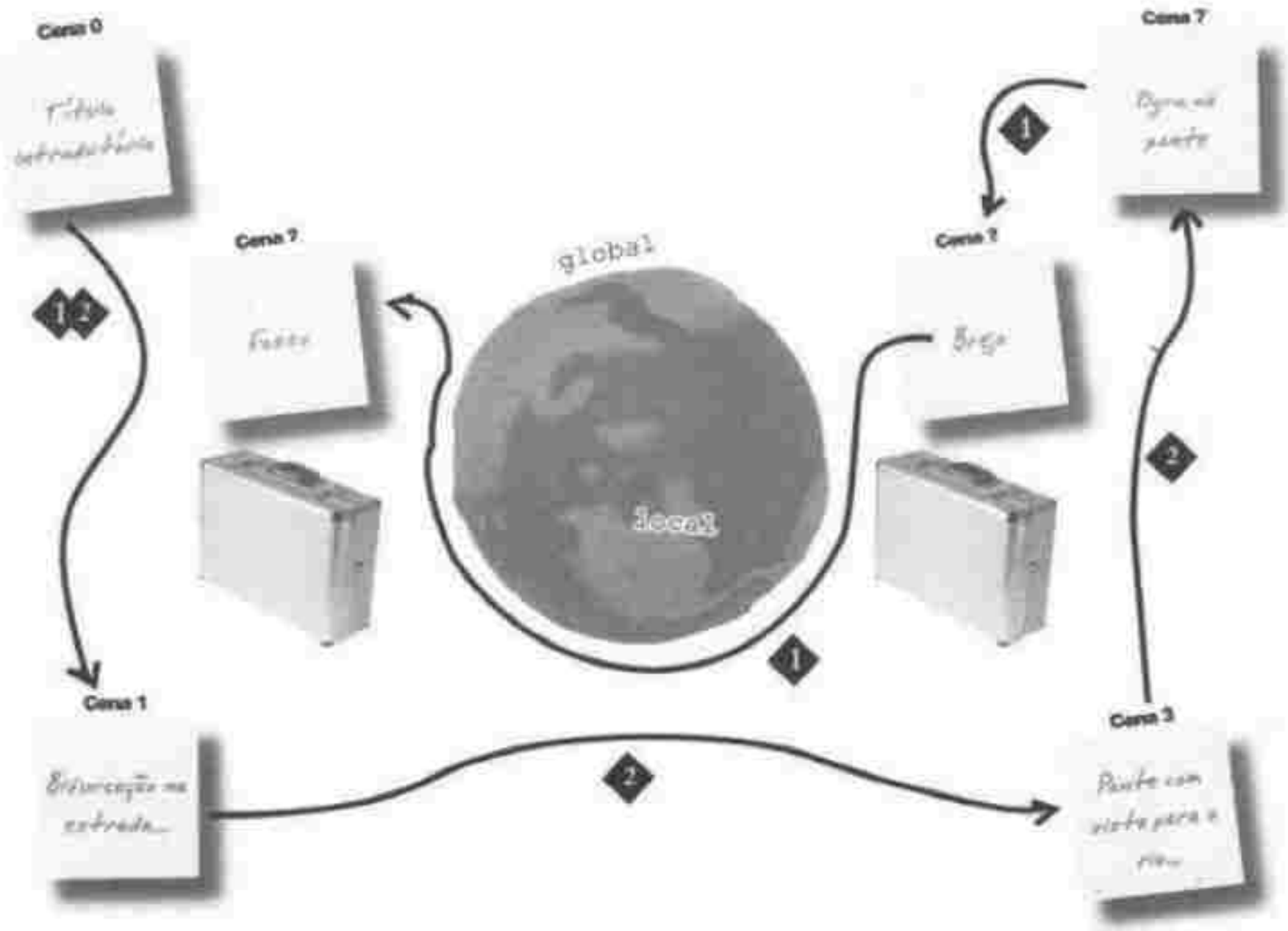


Dobra da Página

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

Quando if/else não for suficiente...





Embora a instrução if/else seja incrivelmente útil, ela tem suas limitações. Por exemplo, você não consegue alternar entre mais de duas coisas. Caso não acredite, tente você mesmo.

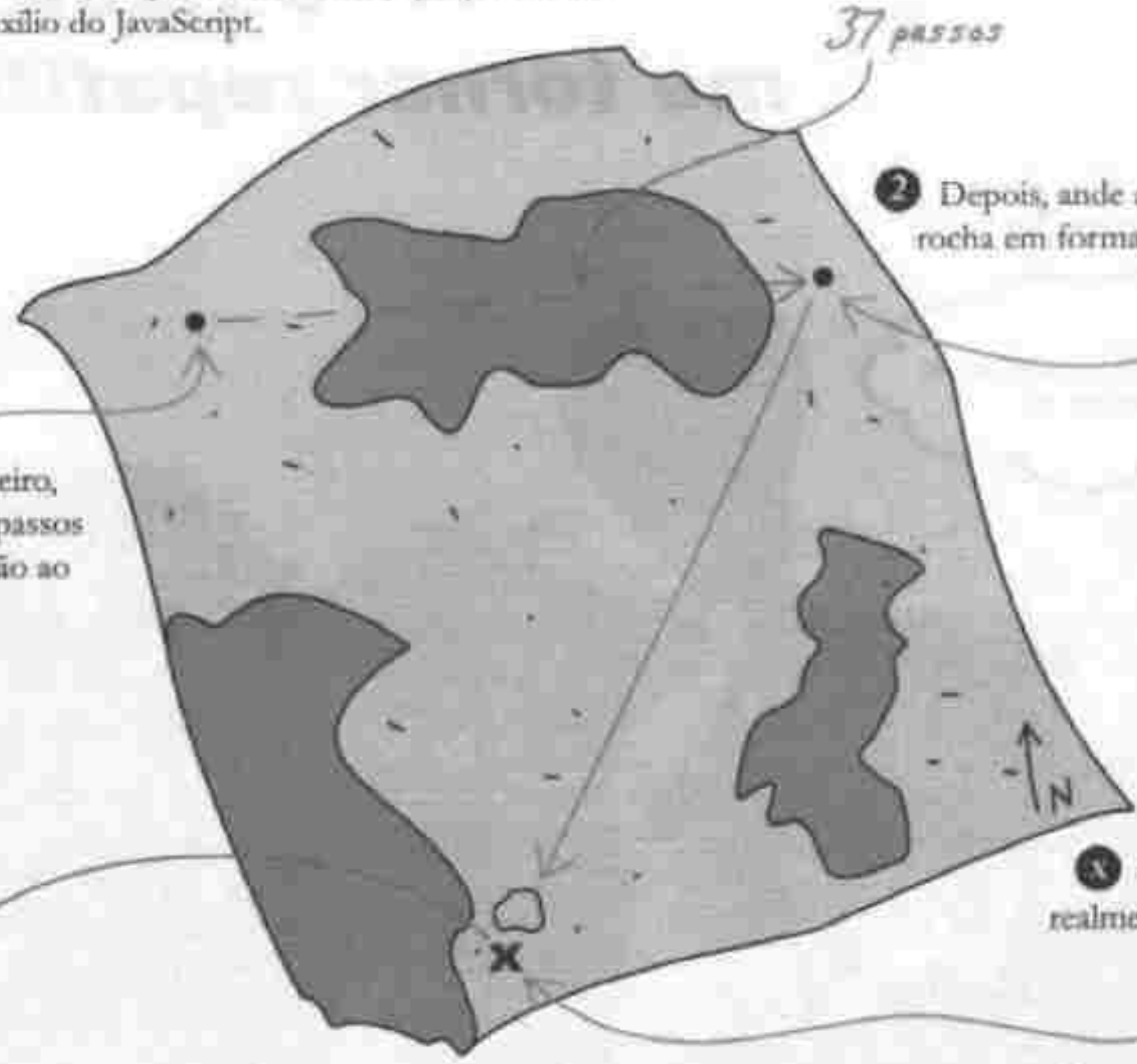
Switch

O risco de me tornar repetitivo

Dizem que a repetição é o tempero da vida. E é. Fazer algo novo e interessante é de fato excitante, mas são as pequenas coisas repetitivas que realmente possibilitam a superação dos obstáculos diários. A higiene compulsiva das mãos, um tique nervoso, clicar em Responder a todos em toda mensagem esquisita que você recebe! Tudo bem, talvez a repetição não seja lá grandes coisas no mundo real. No entanto, pode ser extremamente útil no mundo do JavaScript. Você ficaria surpreso com a freqüência que um script necessita para executar um fragmento de código várias vezes, e é aí que o poder do loop realmente transparece. Sem os loops, você estaria perdendo muito tempo cortando e colando um punhado de código inútil.

O X marca o local

É difícil demonstrar a fascinação por um tesouro enterrado. Eis o mapa de um tesouro que poderá ter algum auxílio do JavaScript.



❷ Depois, ande até ver uma rocha em forma de pipoca.

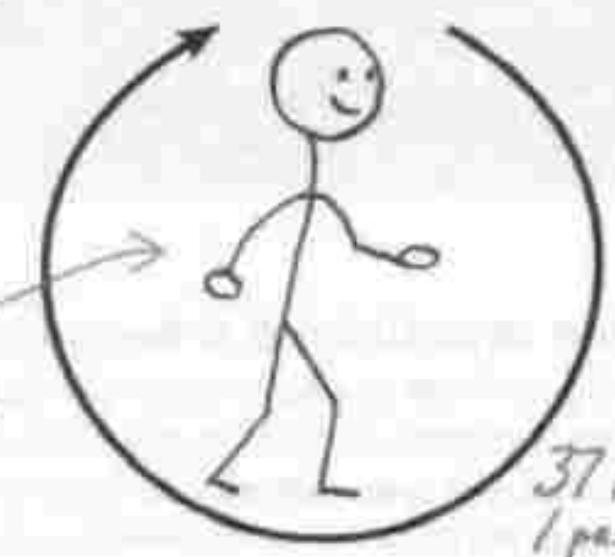
❶ Primeiro, ande 37 passos em direção ao leste.

Ⓧ O X marca realmente o local!

A primeira parte do mapa pode ser percorrida repetindo-se a ação (dando um passo) em um determinado número de vezes (37). Portanto, dar 37 passos é uma questão de repetir um único passo 37 vezes.

Ah, sim, é um tesouro!

Dar um passo é uma ação repetitiva.



❶ Primeiro, ande 37 passos a leste. =

37 passos é apenas 1 passo repetido 37 vezes.

37 ciclos

Então, a pergunta é... como o JavaScript torna possível a repetição?

Mais uma vez, déjà vu... loops for

A repetição no JavaScript é realizada com **loops** que permitem repetir o código. O loop `for`, em particular, é ótimo para repetir alguma coisa quando se sabe a quantidade de vezes. Por exemplo, os loops `for` são ótimos para contar tarefas, como uma contagem regressiva até zero ou somar algum valor.

Loops for permitem repetir o código em um determinado número de vezes.

Um loop `for` consiste em 4 partes diferentes:

❶ Inicialização
A inicialização ocorre uma única vez, no início de um loop `for`.

❷ Teste de condição
O teste de condição verifica se o loop deve continuar em um outro ciclo.

❸ Ação
A parte da ação do loop é o código que de fato se repete em cada ciclo.

❹ Atualização
A parte da atualização do loop atualiza quaisquer variáveis loop no fim de um ciclo.

Os passos 2, 3 e 4 ocorrem uma vez em cada ciclo completo de loop.

❶ ... ❷ ... ❸ ... ❹ ... ❷ ... ❸ ... ❹ ...

Um ciclo de loop

Um outro ciclo de loop

Poder do cérebro

Como os quatro passos de um loop `for` se referem ao exemplo do mapa do tesouro?

Caça ao tesouro com um loop for

Os loops `for` trabalham para trilhar o mapa do tesouro porque eles exigem que se saiba a quantidade de passos. Aplicar um loop `for` na primeira parte do mapa do tesouro ficará assim:

```
        ❶          ❷      ❹
for (var x = 0; x < 37; x++)
  takeStep();
      ❸
```

Incrementa x, da mesma forma que x = x + 1.

Nos loops do JavaScript, é comum iniciar a contagem em 0, embora ele possa ser facilmente modificado para começar em 1.

Decompondo o código para o loop `for`:

37 ciclos

❶ Inicialize a variável `x` do contador com 0.

❷ Verifique se `x` é menor que 37. Se for, prossiga com o Passo 3 e continue o ciclo completo de loop. Caso contrário, encerre o loop.

❸ Execute o código de ação do loop, que neste caso significa executar a função `takeStep()`.

❹ Aumente `x` e volte para o Passo 2, possivelmente para iniciar um outro ciclo de loop.

Após 37 ciclos completos de loop, ele termina com `x` igual a 37. Tudo isso, graças aos quatro fragmentos do quebra-cabeça loop `for` que funciona em conjunto para estabelecer uma repetição JavaScript.

Inicialização ❶
Inicia o loop com o contador, x, em 0.

`var x = 0`

Teste de condição ❷
Só realiza um outro ciclo de loop se o teste avaliar como true, ou seja, se x for menor que 37.

`x < 37`

37 ciclos

Ação ❸
Chama a função takeStep() para dar um passo.

`takeStep()`

Atualização ❹
Atualiza o contador de loop adicionando 1 a x.

`x++`

Aumenta x, da mesma forma que x = x + 1.

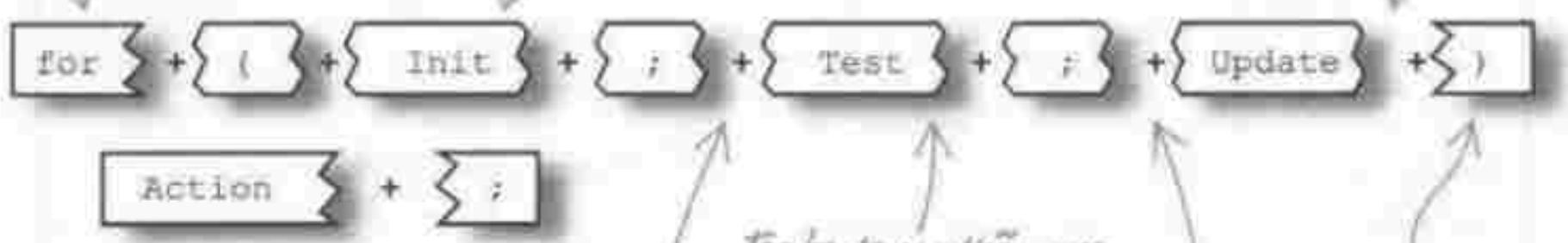
Disseque o loop for

Todos os loops `for` se atêm a um formato consistente que exige locais específicos para cada um dos quatro componentes. A boa notícia é que há muita flexibilidade na personalização de seus próprios loops usando este formato.

Todos os loops for iniciam com a palavra-chave for.

Código de inicialização que normalmente define a contador de loops.

Atualização do controle de loop, que normalmente aumenta ou diminui uma variável do contador.



Teste de condição, que no final resulta em um valor true/false.

Parênteses que abrangem as partes Init, Test e Update do loop.

Ação a ser repetida, que é uma instrução simples (ou composta).

Ponto-e-vírgula é necessário após o código de inicialização e de teste de condição.

Exercício

Termine o código que avisa ao usuário para digitar primeiro um número maior que 0 e, em seguida, use esse número como uma contagem inicial de um loop `for` que realiza a contagem regressiva de um rolo de filme antigo (4, 3, 2, 1, Rodando!). Além disso, confirme se o número é realmente maior que 0, antes de realizar a contagem regressiva.

Armazena o número em uma variável count.

Avisa ao usuário para digitar um número.

```
var count = prompt("Digite um número maior que 0:", "10");

.....

.....

.....

.....
```

Solução do Exercício

Termine o código que avisa ao usuário para digitar primeiro um número maior que 0 e, em seguida, use esse número como uma contagem inicial de um loop `for` que realiza a contagem regressiva de um rolo de filme antigo (4, 3, 2, 1, Rodando!). Além disso, confirme se o número é realmente maior que 0, antes de realizar a contagem regressiva.

Armazena o número em uma variável count.

Avisa ao usuário para digitar um número.

Contagem regressiva em 1.

Certifica-se de que count é maior que 0.

```
var count = prompt("Digite um número maior que 0:", "10");

if (count > 0) {

  for (var x = count; x > 0; x--)

    alert("Iniciando em..." + x);

    alert("Rodando!");

  }

else

  alert("O número não era maior que 0. Sem filme!");
```

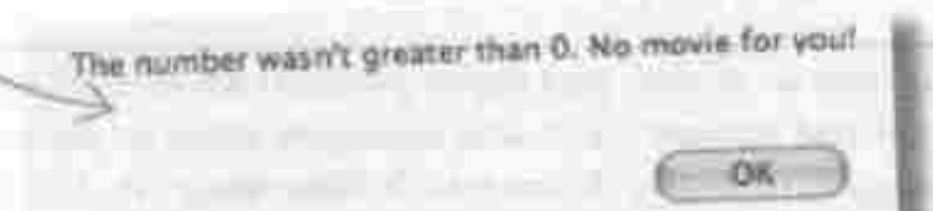
Diminui o contador toda vez que termina o loop.

Inicializa o contador de loop (x) com count.

Mostra a contagem atual.

Termina a contagem!

Dados inválidos.



Mandango: um localizador macho de poltronas de cinema

A contagem regressiva de rolos de filme não é apenas uma maneira que os loops JavaScript podem ser aplicados aos filmes. Como você deve saber, a maioria dos machos querem um lugar vazio entre os que estão ocupados quando assistem a um filme juntos. Esse conhecimento levou Seth e Jason a criar Mandango, o localizador macho de poltronas de cinema.

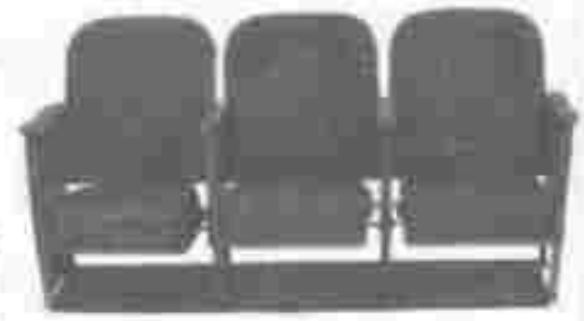
A idéia é permitir que os amigos machos comprem poltronas de cinema em grupos de três, assim haverá sempre uma poltrona entre eles. O problema é, Seth e Jason não descobriram como fazer isso funciona... ainda.

Primeiro, verifique a disponibilidade de poltronas

O desafio que os rapazes enfrentam é o de serem capazes de pesquisar cada poltrona em uma fileira, verificando uma seqüência de três poltronas disponíveis.

Não é legal.

Qualquer outra poltrona que não as três juntas disponíveis resulta em uma carência maior de "masculinidade".

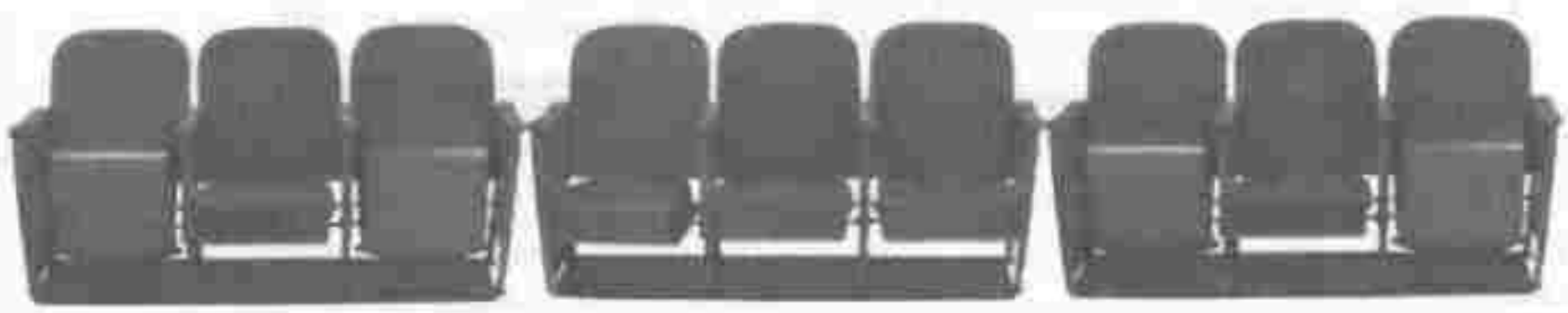


Legal!

Uma grande quantidade de machos na sala de projeção significa três poltronas disponíveis em uma fileira

Aponte seu Lápis

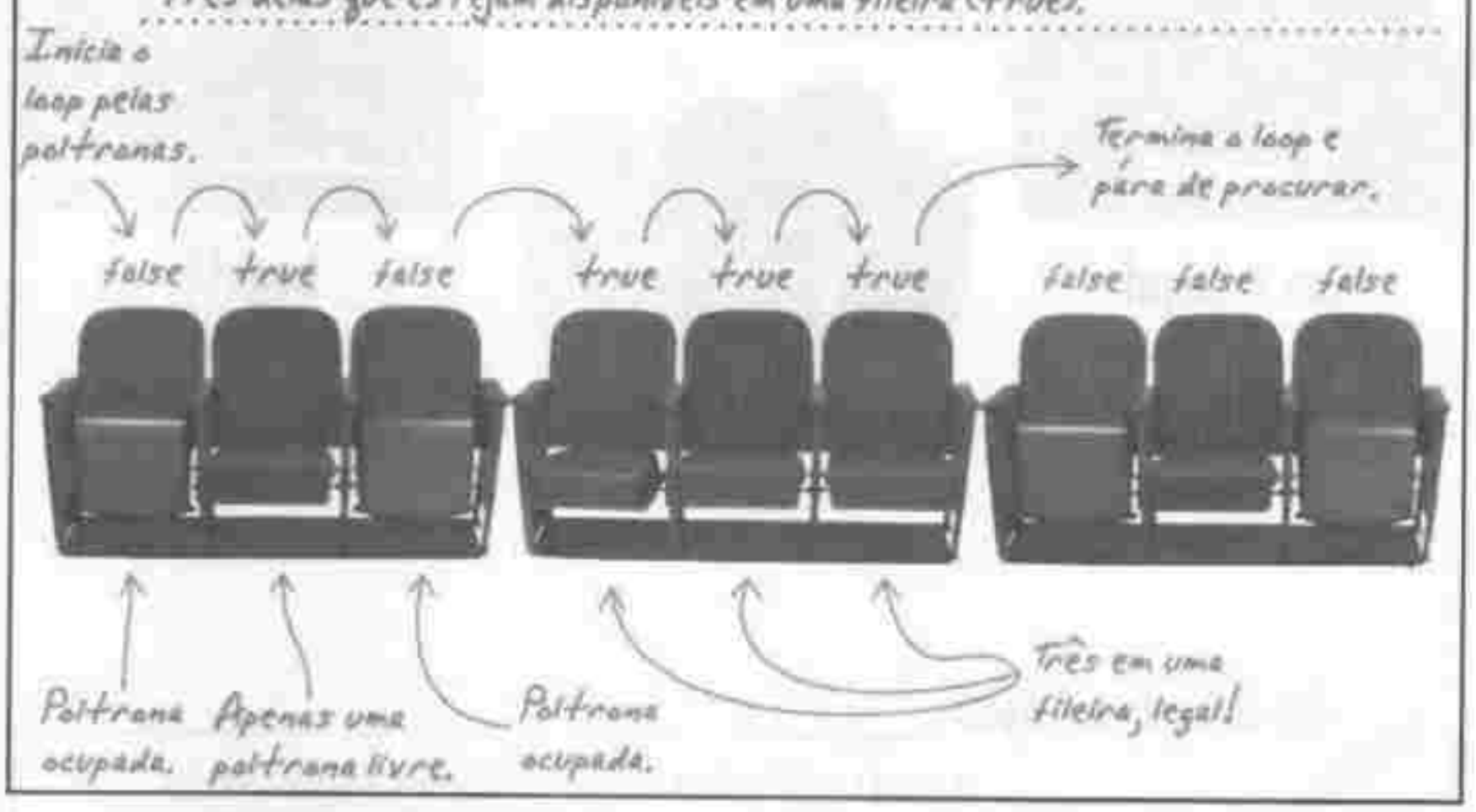
Usando a fileira de poltronas de cinema abaixo, escreva como você pesquisaria três poltronas disponíveis em uma fileira, usando o loop `for`. Certifique-se de formular como exatamente o loop funciona em relação às poltronas.



Aponte seu Lápis Solução

Usando a fileira de poltronas de cinema abaixo, escreva como você pesquisaria três poltronas disponíveis em uma fileira, usando o loop `for`. Certifique-se de formular como exatamente o loop funciona em relação às poltronas.

Se a disponibilidade de cada poltrona for representada por uma variável booleana, então você pode executar um loop pelas poltronas procurando por três delas que estejam disponíveis em uma fileira (true).



Execução de loop, HTML e disponibilidade de poltronas

O projeto geral do Mandango faz sentido, mas não é bem claro sobre como a disponibilidade de cada poltrona se converte em código HTML.

```
<img id="seat1" src="seat_unavail.png" alt="Unavailable" />
<img id="seat2" src="seat_avail.png" alt="Available" />
<img id="seat3" src="seat_unavail.png" alt="Unavailable" />
<img id="seat4" src="seat_avail.png" alt="Available" />
<img id="seat5" src="seat_avail.png" alt="Available" />
<img id="seat6" src="seat_avail.png" alt="Available" />
<img id="seat7" src="seat_unavail.png" alt="Unavailable" />
<img id="seat8" src="seat_avail.png" alt="Available" />
<img id="seat9" src="seat_unavail.png" alt="Unavailable" />
```

Cada poltrona de cinema é visualizada na página do Mandango como uma imagem.

O HTML completo e as imagens deste exemplo estão disponíveis em www.altabooks.com.br

Você não só precisa ser capaz de executar um loop pelos elementos de imagem HTML, como também precisa de uma forma de armazenar suas disponibilidades como variáveis booleanas no código JavaScript.

Poltronas de cinema como variáveis

Antes mesmo de pensar em executar um loop pelas poltronas pesquisando a disponibilidade, você precisa representar a disponibilidade de cada poltrona em código JavaScript. A disponibilidade de uma fileira de nove poltronas pode ser representada por nove variáveis booleanas.

```
var seat1 = false;
var seat2 = true;
var seat3 = false;
var seat4 = true;
var seat5 = true;
var seat6 = true;
var seat7 = false;
var seat8 = false;
var seat9 = false;
```

A disponibilidade de cada poltrona é armazenada como uma booleana.

True significa que uma poltrona está disponível.

False significa que uma poltrona não está disponível.

Então eu preciso de uma forma de usar o mesmo nome de variável para executar um loop pelos vários fragmentos de dados? Parece divertido... esqueça!

Agora, você está pronto para criar um loop `for` que executa um loop por essas nove poltronas, verificando se existem três disponíveis em uma fileira.

```
for (var i = 0; i < 10; i++) {
  if (seat1)
    ...
}
```

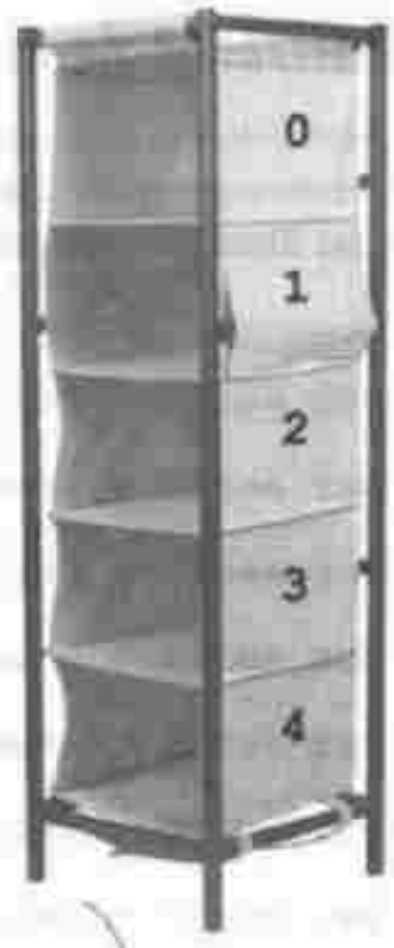
Você não pode mudar um nome de variável toda vez que termina o loop!

Espere, isso parece ser um problema. O loop `for` precisa ser capaz de verificar o valor de uma variável de poltrona diferente toda vez que termina o loop. Mas não há uma maneira de fazer isso, já que cada variável possui um nome diferente.

Poder do cérebro

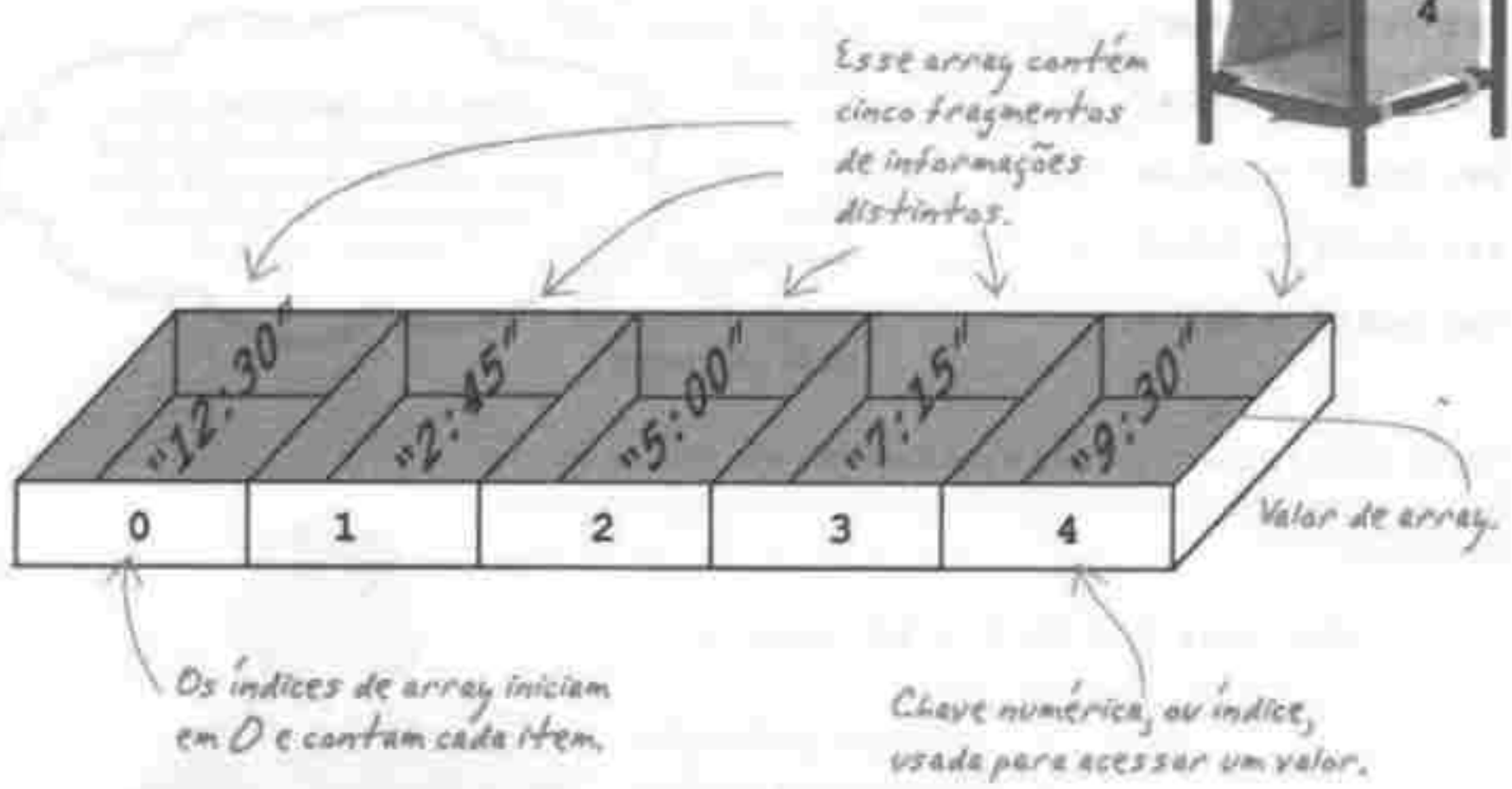
Se as variáveis individuais não funcionarem bem nos loops, como você poderia armazenar as informações de modo a executarem um loop?

Os arrays coletam vários fragmentos de dados



O JavaScript permite armazenar vários fragmentos de dados em uma única variável com um tipo especial de dados chamado **array**. Uma variável de array é como uma variável normal, já que ela só tem um nome, mas um array possui vários locais de armazenamento. Pense em um array como um cubículo de armazenamento em sua casa – é um equipamento individual com vários locais de armazenamento.

Cada item em um array consiste em dois fragmentos de informações: um **valor** e uma **chave** única que é usada para acessar o valor. As chaves são freqüentemente números que iniciam em zero e contam cada item. As chaves numéricas são conhecidas como **índices**, tornando-se um **array indexado**.



Criar um array é semelhante a criar uma variável normal, exceto se tiver que informar ao JavaScript que você precisa de um array, ao contrário de uma única unidade de armazenamento. De fato, você está dizendo ao JavaScript para criar um **objeto**.

```
var showTime = new Array();
```

Esse é um novo objeto do tipo Array.

O nome da variável de array.

Crie um novo objeto de array.

Relaxe

Não se preocupe com o fato de que um array é na verdade um objeto.

Para suas finalidades imediatas, não importa muito se um array é realmente um objeto. Você aprenderá muito sobre objetos nos Capítulos 9 e 10 e, nesse caso, essa coisa de objeto se resolverá naturalmente.

Os valores de Array são armazenados em chaves

Objeto ou não, uma vez criado um array você pode começar a adicionar e acessar os dados nele. A chave para se obter os dados armazenados em um array é, bem, a **chave**! A chave específica associada ao fragmento de dados é o que você usa para acessar esses dados. No caso de um array indexado, usa-se somente o índice do elemento de array que deseja acessar.

O valor a ser armazenado no array.

```
showTime[0] = "12:30";
```

O nome da variável de array.

O índice do valor de array dentro de colchetes.

Esse código define o primeiro valor do array showTime para uma hora do dia. Se não quiser definir cada valor de um array manualmente, um de cada vez, você pode inicializar o array inteiro quando for criá-lo.

Liste todos os valores de array separados por vírgulas.

```
var showTime = [ "12:30", "2:45", "5:00", "7:15", "9:30" ];
```

A primeira parte da criação do array começa igual.

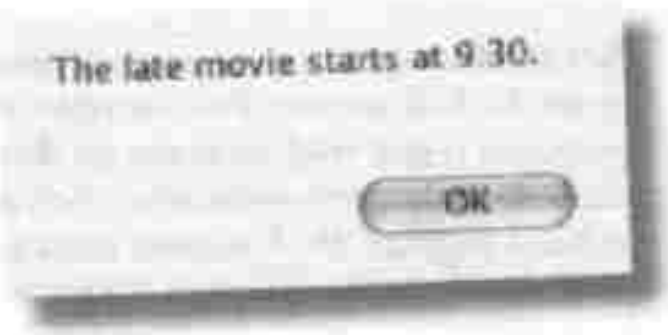
Certifique-se de incluir a lista de valores de array dentro de colchetes.

E lembre-se do ponto-e-vírgula.

Espere um minuto, esse código não envolve toda essa coisa de objeto. O que aconteceu? Esse código evita a criação formal de um objeto vazio prosseguindo e construindo um array (objeto) a partir de valores que ele contém. Liste todos os elementos contidos no array dentro de colchetes. Com o array preenchido com dados, você está pronto para usá-lo!

Pegue o último valor no array.

```
alert("O último filme começa às " + showTime[4] + ".");
```



Os arrays armazenam <u>vários</u> fragmentos de dados em um <u>único</u> lugar.

Arrays Expostos

Entrevista desta semana:

Dentro da mente de um armazenador de dados serial

Use a Cabeça: Prazer em conhecê-lo, Array. Ouvi dizer que você é ótimo em armazenar vários fragmentos de dados.

Array: É verdade. Volume é a minha essência. Você precisa de um lugar para armazenar 50 strings de texto ou 300 números? Sou o cara.

Use a Cabeça: Parece interessante. Mas as pessoas não podem armazenar quantidades de dados em variáveis normais?

Array: Claro, e as pessoas também podem ir trabalhar descalças, se quiserem. Olha, existe sempre mais de uma maneira de fazer as coisas. Neste caso, eu forneço uma maneira de armazenar vários fragmentos de informação melhor do que as variáveis normais.

Use a Cabeça: Bem, eu prefiro calçar sapatos para trabalhar. Mas no que exatamente você é melhor?

Array: Pense dessa forma. Se você está mantendo um diário e escreve alguma coisa todos os dias, como acompanha todas essas páginas após alguns anos?

Use a Cabeça: As páginas estão todas lá no diário. E daí?

Array: Você está partindo do pressuposto de que as páginas estão sendo organizadas junto com algum senso de conectividade. E se elas forem apenas um bando de bilhetes jogados em uma caixa de sapatos? Esse diário, de repente, ficaria muito mais difícil de se gerenciar.

Use a Cabeça: Certo, armazenar dados em um array é como manter as páginas de um diário em um livro?

Array: Eu organizo os dados de uma maneira muito fácil de acessar. Por exemplo, se eu pergunto o que você escreveu no diário no último dia 6 de junho, provavelmente você me diria para voltar à página 124. A mesma coisa acontece com os dados de array, exceto os números de páginas de um array que são chamados de chaves.

Use a Cabeça: Já ouvi falar em índices de array, mas não em chaves. O que é uma chave?

Array: Ah, desculpe. Chave é o termo geral usado para descrever um fragmento de informação usado para procurar um fragmento de dados. Índice é apenas um certo tipo de chave, uma chave numérica. Portanto, os números das páginas do diário não são somente chaves, elas também são índices. Se você estiver falando de procurar dados com números, chaves e índices específicos realmente são a mesma coisa.

Use a Cabeça: Entendi. Acho que a única coisa que ainda não entendi é o que tudo isso tem a ver com a execução de um loop?

Array: Bem, não necessariamente tudo. Sou bastante útil para armazenar dados sem que os loops entrem na jogada. No entanto, eu os torno incrivelmente úteis para que eles percorram o ciclo por um bocado de dados.

Use a Cabeça: Como assim?

Array: Lembre-se de que os loops usam contadores numéricos com freqüência para controlar a execução do loop, certo? Use o contador como índice dentro de um array e *voilà*! Agora você possui uma maneira de percorrer o ciclo por todos os dados que armazenei separadamente.

Use a Cabeça: Espere, você está dizendo que as pessoas podem usar um contador de loop como um índice de array para procurar dados?

Array: É exatamente isso que estou dizendo.

Use a Cabeça: Isso é extremamente eficiente!

Array: Pois é. Essa é a razão pela qual os scripts que precisam executar um loop pelos dados me acham indispensável. Com apenas algumas linhas de código, você pode executar um loop por um array de dados inteiro. É realmente muito legal.

Aponte seu Lápis

Escreva um código para criar um array de `poltronas` para o Mandango e, em seguida, execute um loop pelas poltronas no array, alertando o usuário sobre a disponibilidade de cada poltrona.

Não existem Perguntas Idiotas

P: Existe a possibilidade de um loop `for` nunca parar de executar um loop?

R: Ah, sim. O temor do loop infinito. Claro, existe a possibilidade de se criar um loop que nunca existiu, destinado a percorrer um ciclo continuamente até os limites de tempo e espaço... ou, pelo menos, até que você recarregue a página da web. Loops infinitos não são considerados uma coisa boa, porque eles evitam que o seu script faça algo mais – é o equivalente JavaScript de uma aplicação travada. Pense na tela azul do Windows, só que não tão ameaçadora.

Os loops infinitos ocorrem quando um contador de loop não atualiza de maneira adequada ou quando, ao contrário, nunca se altera provocando um resultado de valor `false` para o teste de condição do loop. Sabendo disso, você sempre deve verificar à risca, duas ou três vezes, o teste de condição e atualizar a lógica em seus loops `for`. Ah, e você saberá que tem um loop infinito em suas mãos quando seu script estiver inativo, sem aparentemente fazer nada.

P: É possível usar uma instrução composta como parte da ação de um loop `for`?

R: Sem dúvida! Na verdade em tudo, exceto no mais simples dos cenários de execução de loop, você precisará usar uma instrução composta. Isso porque os loops mais úteis acabam precisando executar um loop em mais de uma instrução.

P: Quando os testes de condição de loop resultam em `false`, a parte da ação do loop executa mais uma última vez?

R: Não. A parte da ação de um loop `for` só executa se o teste de condição resultar em `true`. Uma vez que o teste de condição resulte em `false`, o loop imediatamente sai sem nenhum outro código executando.

P: Os arrays indexados sempre começam sua indexação com 0?

R: Sim e não. Por padrão, todos os arrays indexados começam em 0. No entanto, você pode substituir esse comportamento e definir chaves numéricas para quaisquer valores de número que você queira, embora não seja convencional. Não faça isso, a menos que haja uma boa decisão de planejamento para não usar os índices baseados em zero... é um comportamento não-convencional e pode provocar confusão.

P: Os dados armazenados em um array têm sempre que ser do mesmo tipo?

R: Não, de modo algum. Para fins de execução de loop, é importante que os dados de array sejam do mesmo tipo, porque a idéia é executar um loop por um conjunto de dados semelhantes. Por exemplo, se quiser executar um loop por um array de notas para calcular uma média, não faria muito sentido que algumas ocorrências fossem booleanas – neste caso, elas todas devem ser números. Então, embora arrays possam conter valores de tipos diferentes, é uma boa idéia armazenar dados de mesmo tipo nos arrays, especialmente quando você está armazenando um conjunto de dados semelhantes.

Aponte seu Lápis
Solução

Escreva um código para criar um array de `poltronas` para o Mandango e, em seguida, execute um loop pelas poltronas no array, alertando o usuário sobre a disponibilidade de cada poltrona.

Já que os índices de array começam em 0, o contador de loop também inicia em 0.

Cria e inicializa o array de poltronas com valores booleanos.

Separa os valores de array com vírgulas.

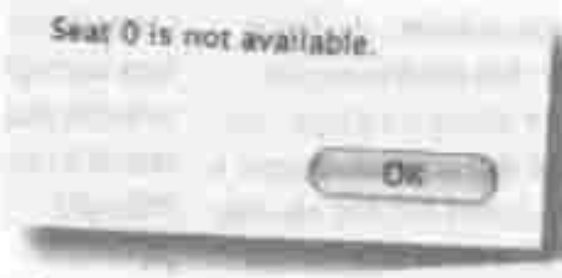
Aumenta o contador de loop desde que ele esteja contando.

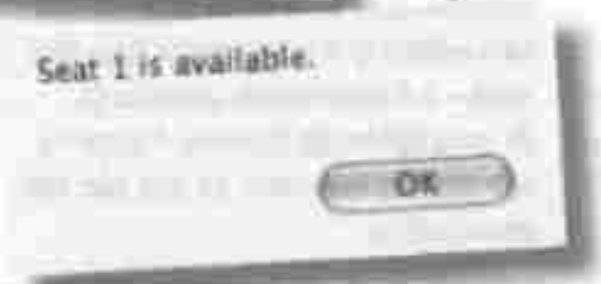
Embora você possa ter usado o número 9 aqui, a propriedade de extensão do objeto de array é ainda melhor. Ele é melhor porque funciona mesmo se o número de poltronas no array mudar em algum momento.

```
var seats = [ false, true, false, true, true, true, false, true, false ];

for (var i = 0; i < seats.length; i++) {

  if (seats[i])

    alert("Poltrona " + i + " disponivel.");

  else

    alert("Poltrona " + i + " indisponivel.");

}
```

Exibe um alerta diferente baseado na disponibilidade (true) ou não (false) da poltrona.

O contador de loop é usado como um índice de array para percorrer todos os valores de array.





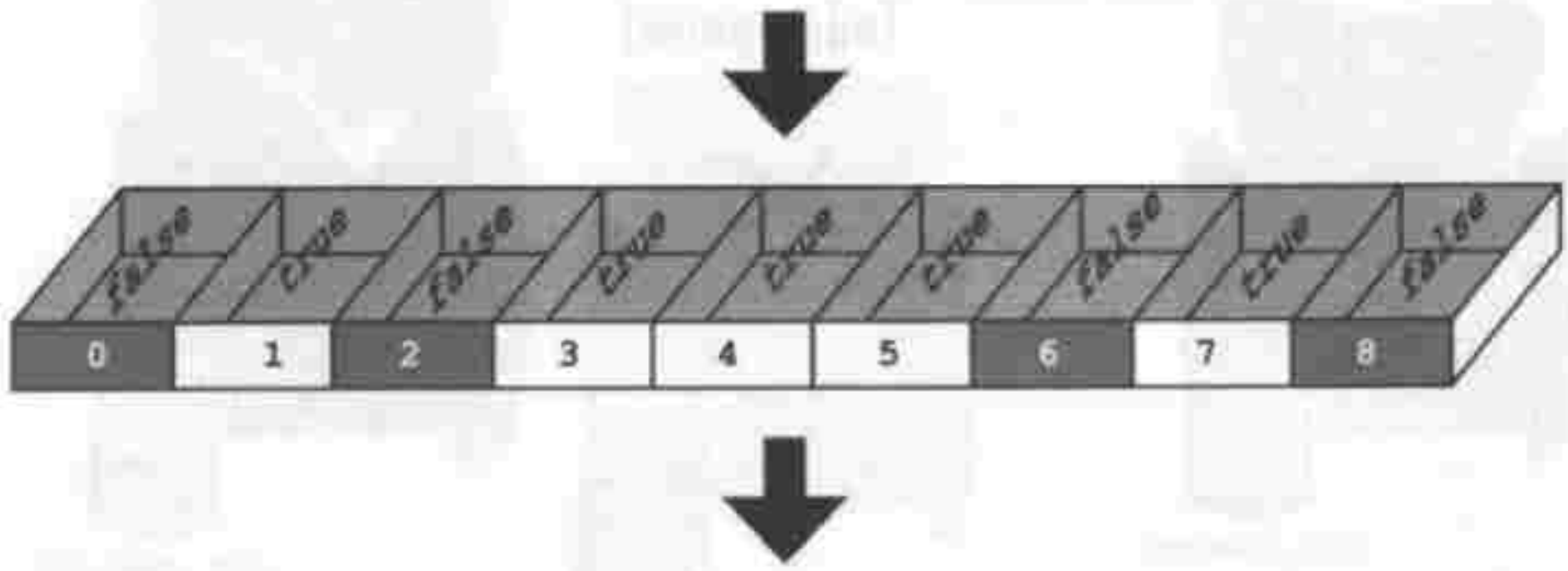
Pontos de Bala

- Loops for repetem um fragmento de código JavaScript um **certo número de vezes**.
- Os operadores de incremento (++) e diminuição (--) fornecem uma forma prática de atualizar os contadores de loop.
- Um array permite armazenar vários fragmentos de dados em um **único lugar**.
- Embora um array contenha vários fragmentos de informação, ele possui um único nome de variável.
- Arrays indexados são acessados através de chaves numéricas chamadas **índices**.
- Arrays indexados funcionam bem com loops porque eles permitem o uso de um contador de loop para executar um loop pelos dados de array.

Do JavaScript ao HTML

A disponibilidade de poltronas no Mandango é representada por um array booleano. Então, o próximo passo é converter esse array em imagens HTML (disponíveis em *http://www.headfirstlabs.com/books/hfjs/*) que reflitam a disponibilidade de poltronas na página da Web do Mandango.

```
var seats = [ false, true, false, true, true, true, false, true, false ];
```



Embora pareça perfeito, não há nenhum código que mapeie o array booleano para visualizar as imagens das poltronas na página da web. Agora, esse é o problema.

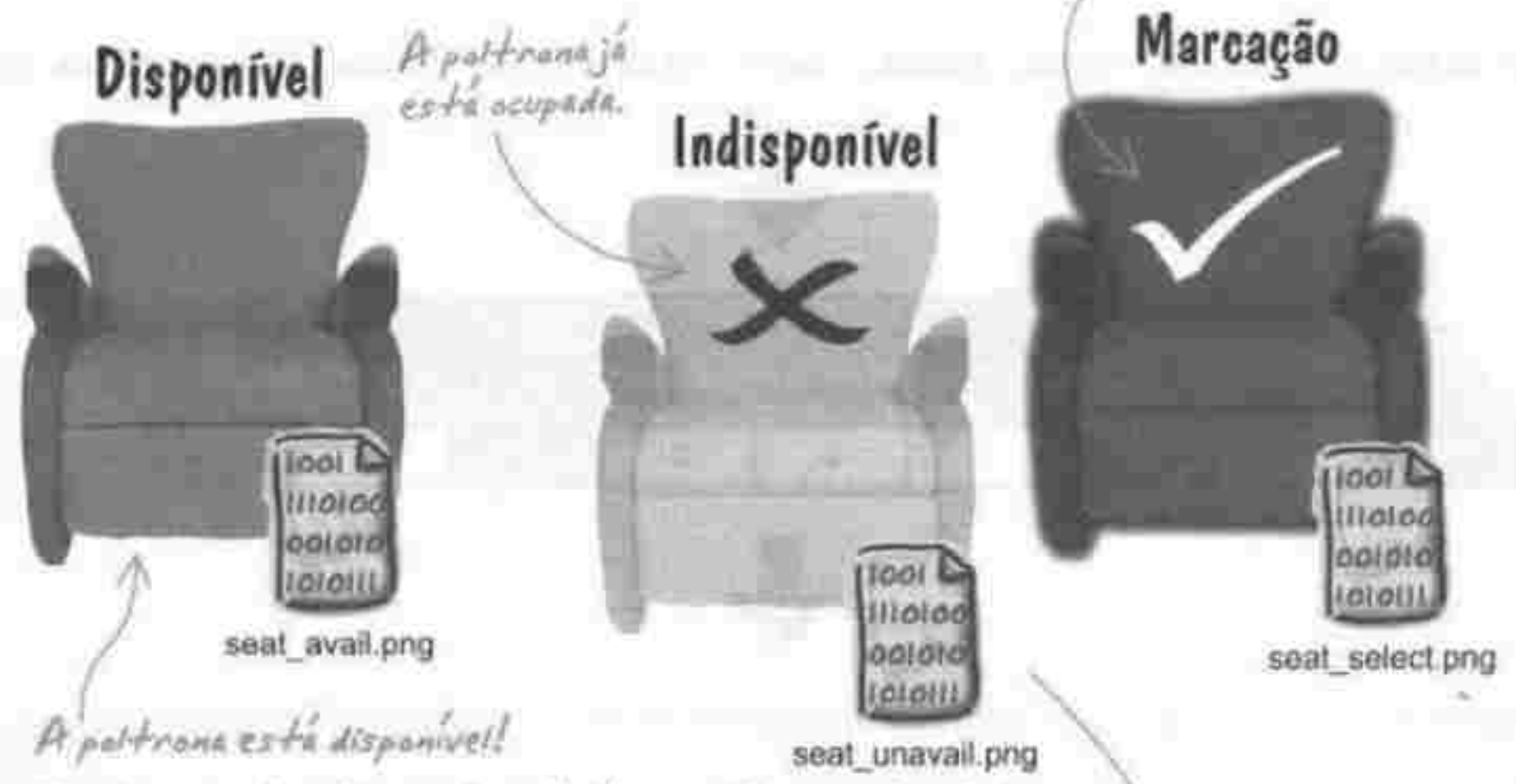
Poder do cérebro

Como você poderia fazer a conexão entre o array de disponibilidade de poltronas em JavaScript e as imagens das poltronas na página do Mandango?

Como visualizar as poltronas do Mandango

Para ligar o array JavaScript às imagens HTML, certifique-se primeiro de que as imagens estejam expostas de forma acessível, e depois determine quais imagens serão usadas para representar os diferentes status da poltrona. Vamos cuidar primeiro da última tarefa.

Essa imagem de poltrona se aplica à pesquisa de poltronas do Mandango, quando as marcações de poltronas estão realçadas.



Essas imagens de poltronas são atribuídas ao atributo src de cada imagem HTML de poltrona para definir as imagens que aparecem na página.

Esse ID é crítico no mapeamento de array para as imagens de poltrona – ele deve começar em 0 e terminar em 8, assim como os índices de array.

```
<img id="seat8" src="seat_unavail.png" alt="Unavailable" />
```

O desafio então se torna a execução de um loop através do array booleano ao definir a imagem da poltrona para cada tag HTML `<img>` na página. Os passos necessários para essa tarefa são surpreendentemente semelhantes a como executamos anteriormente um loop pelos arrays de poltrona. De fato, a única diferença concreta reside na ação do loop.

1. Inicializa a variável do contador `i` em `0`.
2. Verifica se `i` é menor que a extensão do array (9). Se for, prossegue com o Passo 3 e continua o ciclo até terminar o loop. Se não for, encerra o loop.
3. Executa o código de ação do loop, que neste caso **define a imagem da poltrona**.
4. Aumenta `i` e volta para o Passo 2, possivelmente para um outro ciclo de loop.

A função initSeats() de perto

A inicialização da poltrona no Mandango ocorre na função `initSeats()`, que executa o mapeamento do array JavaScript até as imagens HTML da poltrona usando um loop de inicialização de poltrona.

A inicialização realizada por initSeats() envolve a definição da aparência das imagens da poltrona de cinema na página, que é diferente da etapa de inicialização em um loop for.

O contador de loop começa em 0, já que o array indexado começa em 0. ❶

O teste de condição verifica se foi executado um loop em todas as poltronas. ❷

```
function initSeats() {
  // Inicializa a aparência de todas as poltronas
  for (var i = 0; i < seats.length; i++) {
    if (seats[i]) {
      // Define a poltrona como disponível
      document.getElementById("seat" + i).src = "seat_avail.png";
      document.getElementById("seat" + i).alt = "Available seat";
    }
    else {
      // Define a poltrona como indisponível
      document.getElementById("seat" + i).src = "seat_unavail.png";
      document.getElementById("seat" + i).alt = "Unavailable seat";
    }
  }
}
```

Aumenta o contador de loop. ❹

Se o valor de poltrona for true, define a poltrona como disponível. ❸

O ID da imagem da poltrona é criada a partir do contador de loop toda vez que termina o loop. ❺

Se o valor da poltrona for false, define a imagem HTML da poltrona como indisponível. ❻

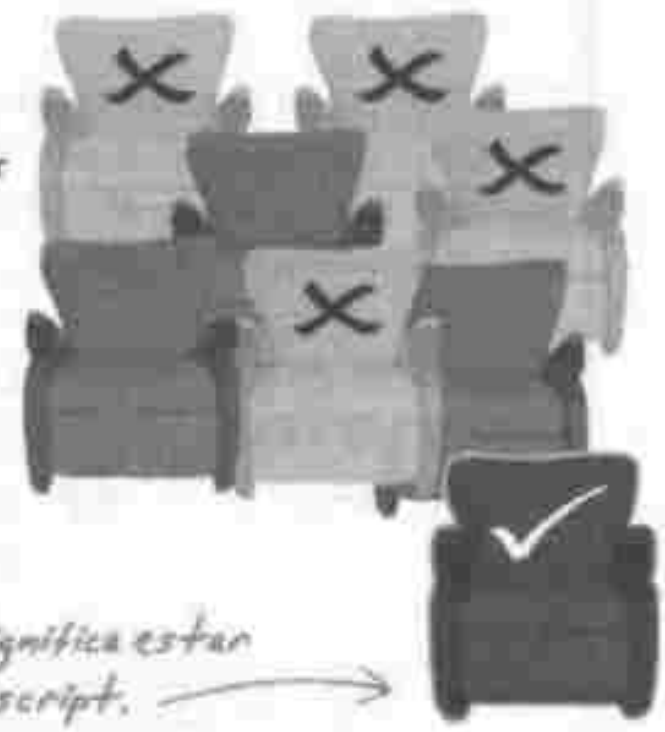
```
<body onload="initSeats();">
  <div style="margin-top:75px; text-align:center">
    <img id="seat0" src="" alt="" />
    <img id="seat1" src="" alt="" />
    <img id="seat2" src="" alt="" />
    <img id="seat3" src="" alt="" />
    <img id="seat4" src="" alt="" />
    <img id="seat5" src="" alt="" />
    <img id="seat6" src="" alt="" />
    <img id="seat7" src="" alt="" />
    <img id="seat8" src="" alt="" /><br />
  </div>
</body>
</html>
```

O ID dessa imagem de poltrona é seat6.

Os atributos src e alt são dinamicamente modificados a cada imagem de poltrona.

Pesquisa de poltronas não tão masculinas

Com as poltronas inicializadas, agora é possível prosseguir com a pesquisa de poltronas, que é realmente o objetivo do Mandango. Seth e Jason determinaram que seria melhor fazer com que o script procure primeiro por poltronas individuais antes de começar as futuras pesquisas das três poltronas. Isso simplifica a tarefa imediata, permitindo criar a aplicação de forma progressiva.



Já que eles querem pesquisar por uma poltrona individual disponível, a primeira coisa que o script vai precisar é de uma variável para rastrear a seleção da poltrona.

Variável global, que significa estar acessível em todo o script.

Essa variável armazena a marcação da poltrona e precisa estar por perto durante a vida do script, o que significa que ela deve ser uma variável global. Então, a função `findSeat()`, que manipula a tarefa de encontrar uma poltrona para o usuário, contará com a variável `selSeat` para armazenar o índice da poltrona marcada.

A variável selSeat faz sentido, mas qual valor indica uma poltrona desmarcada?

Seth traz uma boa pergunta. A variável `selSeat` armazena a marcação de poltrona, que se encontra no intervalo de 0 a 8, assim que ela tiver sido marcada. Mas você também precisa saber se um usuário ainda **não** escolheu uma poltrona. Um valor especial pode indicar esse estado de **não-marcação**, que pode ser observado como -1 (nenhuma poltrona marcada ainda). Então, `selSeat` realmente precisa começar inicializado como -1.

```
var selSeat = -1;
```

A variável selSeat agora é inicializada como -1, de modo que o script começa sem nenhuma poltrona marcada.

Com a variável de marcação de poltrona funcionando, estamos prontos para montar a função `findSeat()`. `findSeat()` pesquisará cada poltrona no array de poltronas, encontrará poltronas disponíveis e, em seguida, avisará o usuário para aceitar ou rejeitar cada poltrona disponível. Embora seja verdade que os machos não ficarão felizes com essa versão inicial do Mandango, é um passo na direção certa!

Ímãs de geladeira JavaScript

A função `findSeat()` do Mandango é onde o usuário procura por uma poltrona disponível e, em seguida, confirma ou nega quaisquer poltronas que sejam encontradas. Ajude Seth e Jason a terminar o código que falta com os ímãs.

```
function findSeat() {
  // Se a poltrona já estiver marcada, reinicializa todas as poltronas para removê-
las

  if (............>= 0) {

    ............= -1;

    ...............();


}

// Pesquisa por todas as disponibilidades de poltronas
for (var i = 0; i < seats.length; i++) {
  // Verifica se a poltrona atual está disponível

  if (..............) {
    // Define a marcação da poltrona e atualiza sua aparência

    ............= i;

    document.getElementById("seat" + i). ........= "seat_select.png";

    document.getElementById("seat" + i). ........= "Your seat";

    // Avisa o usuário para aceitar a poltrona

    var ............= confirm("Seat " + (i + 1) + " is available. Accept?");

    if (.............) {
      // O usuário rejeitou a poltrona, então desfaz a marcação e continua
procurando

      ............= -1;

      document.getElementById("seat" + i). ........= "seat_avail.png";

      document.getElementById("seat" + i). ........= "Available seat";
    }
  }
}
}
```

Ímãs de geladeira JavaScript Solução

A função `findSeat()` do Mandango é onde o usuário procura por uma poltrona disponível e, em seguida, confirma ou nega quaisquer poltronas que sejam encontradas. Ajude Seth e Jason a terminar o código que falta com os imãs.

```
function findSeat() {
  // Se a poltrona já estiver marcada, reinicializa todas as poltronas para removê-las
  if (selSeat != 0) {
    selSeat = -1;
    initSeats();
  }

  // Pesquisa por todas as disponibilidades de poltronas
  for (var i = 0; i < seats.length; i++) {
    // Verifica se a poltrona atual está disponível
    if (seats[i]) {
      // Define a marcação da poltrona e atualiza sua aparência
      selSeat = i;
      document.getElementById("seat" + i).src = "seat_select.png";
      document.getElementById("seat" + i).alt = "Your seat";

      // Avisa o usuário para aceitar a poltrona
      var accept = confirm("Seat " + (i + 1) + " is available. Accept?");
      if (!accept) {
        // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
        selSeat = -1;
        document.getElementById("seat" + i).src = "seat_avail.png";
        document.getElementById("seat" + i).alt = "Available seat";
      }
    }
  }
}
```

Se selSeat for diferente de -1, começa uma nova busca e redefine as poltronas.

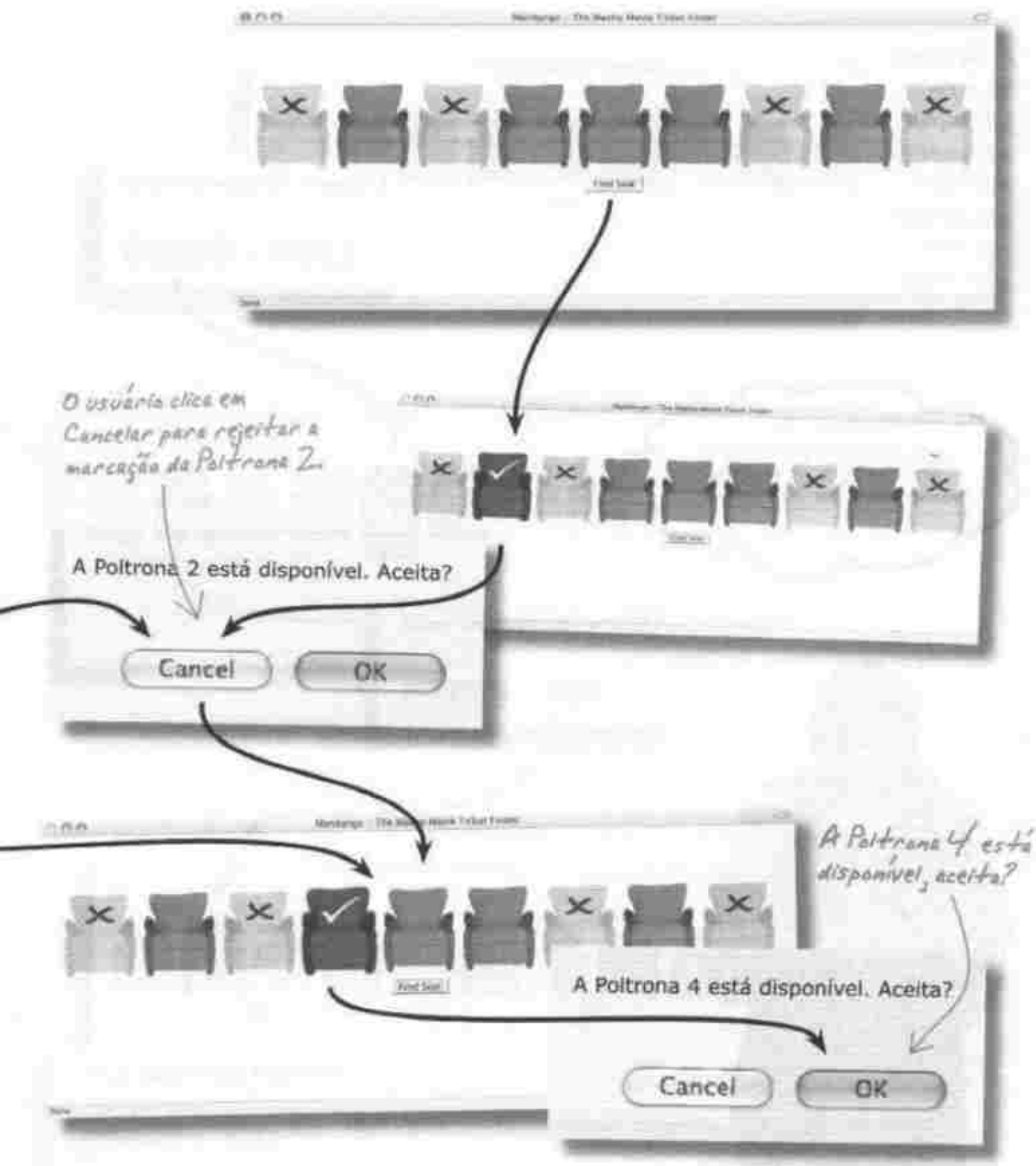
O usuário visualiza um número a mais de poltrona, já que a maioria dos usuários começa a numeração em 1 em vez de 0.

Se a poltrona estiver disponível, seats [i] será true.

Verifica se o usuário aceitou a poltrona disponível.

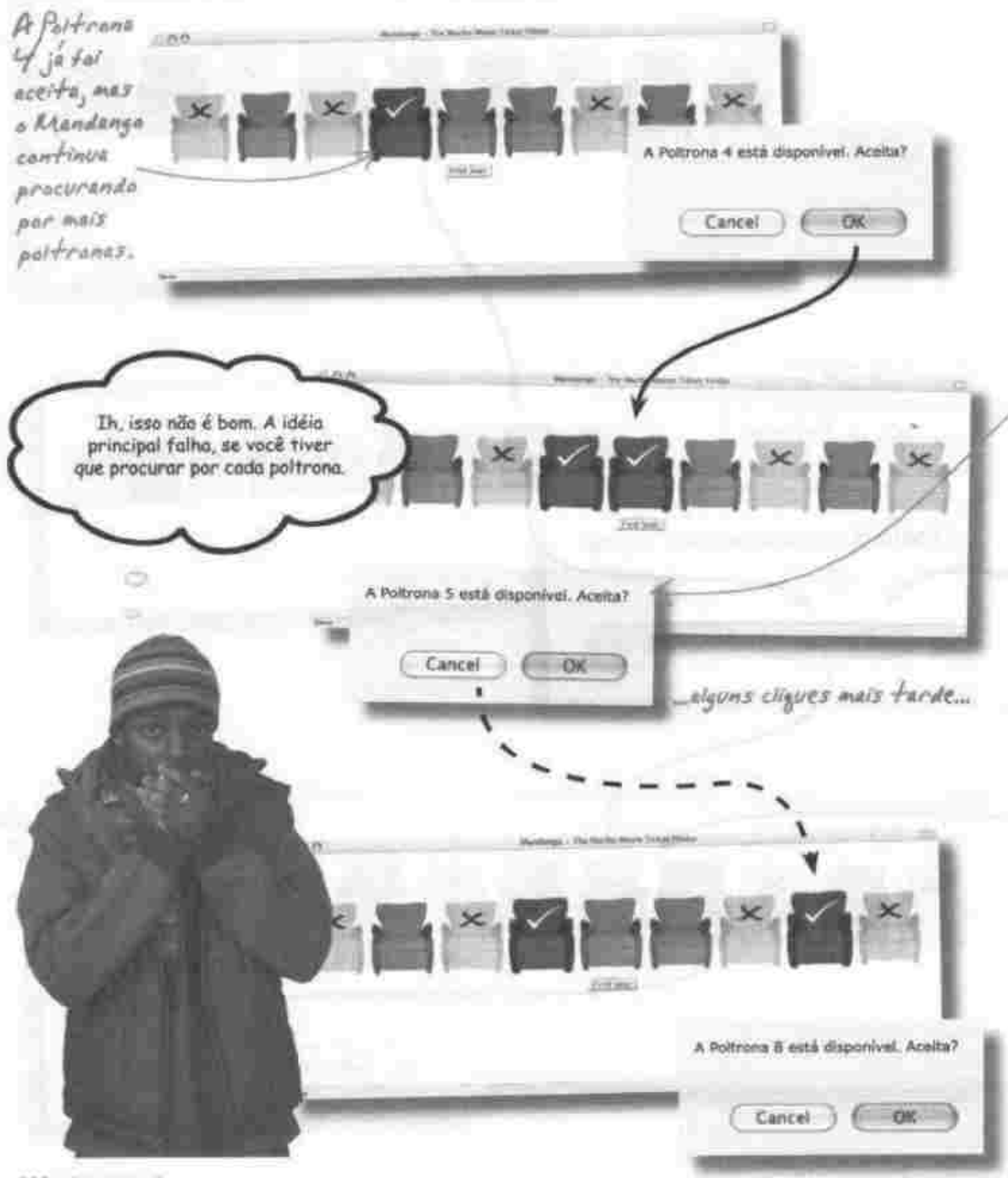
Test-drive: o localizador de poltronas individuais

A versão do Mandango de pesquisa por poltronas individuais usa o loop `for` e um array para permitir que o usuário pesquise poltronas individuais disponíveis. Não é muito para homens, mas é funcional...



Uma coisa boa demais: loops intermináveis

Embora a busca por poltronas individuais do Mandango tecnicamente funcione para descobrir uma poltrona individual que esteja disponível, há um problema com o loop que ele não sabe quando parar. Mesmo após o usuário aceitar uma poltrona, clicando em OK, o script se mantém em loop pelas outras poltronas disponíveis.



Os loops sempre precisam de uma condição de saída (ou duas!)

Já que a fanática busca por poltronas parece ser causada pelo loop que não termina nunca, Jason acha apropriado uma observação atenta ao loop for na função `findSeat()`.

A função confirm() avisa o usuário para responder uma pergunta sim/não, retornando true (sim) ou false (não).

```
for (var i = 0; i < seats.length; i++) {
  // Verifica se a poltrona atual está disponível
  if (seats[i]) {
    // Define a marcação da poltrona e atualiza selSeat
    selSeat = i;
    document.getElementById("seat" + i).src = "seat_select.png";
    document.getElementById("seat" + i).alt = "Your seat";

    // Avisa o usuário para aceitar a poltrona
    var accept = confirm("Seat " + (i + 1) + " is available. Accept?");
    if (!accept) {
      // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
      selSeat = -1;
      document.getElementById("seat" + i).src = "seat_avail.png";
      document.getElementById("seat" + i).alt = "Available seat";
    }
  }
}
```

?

Se o usuário aceitar uma poltrona disponível, nada acontece e o loop continua apenas o seu trabalho.

Esse é o código que é executado quando o usuário NÃO aceita uma poltrona disponível.

Então, quando o usuário clica em Cancelar para rejeitar a poltrona, a variável `selSeat` é definida como -1 (nenhuma marcação) e o loop continua. No entanto, não há nenhum código para quando o usuário aceitar uma poltrona. Isso é bom, desde que ele permita que a variável `selSeat` se lembre da poltrona atual, mas não há nada que evite o loop de continuar procurando por poltronas.

Poder do cérebro

O que precisa acontecer quando o usuário clica no botão OK para aceitar a poltrona atual?

Um "break" na ação

O problema com o código do Mandango é que você precisa tirar o loop do impasse, antes que o usuário aceite a poltrona. Um possível ajuste é enganar o loop `for` atribuindo ao contador um valor maior que a extensão do array.

```
i = seats.length + 1;
```

Isso finaliza o loop forçando a falha do teste de condição... mas há uma maneira melhor de forçar o término dos loops.

Embora esse código tenha uma alteração um pouco mais inteligente que conclui a tarefa, existe uma maneira melhor de enganar a condição de loop que não envolve embromação com o contador de loop. A instrução `break` foi criada especificamente para escapar de uma seção de código, inclusive do código de loop.

Sai imediatamente de um loop; não passa para Avançar, não cobra $200.

```
break;
```

Quando o loop encontra a instrução `break`, ele finaliza imediatamente, ignorando completamente o teste de condição. Então, a instrução `break` oferece uma maneira prática de sair imediatamente de um loop, sem fazer perguntas.

A instrução `continue` está intimamente relacionada à `break`, a qual tira o ciclo atual de loop do impasse, mas não sai dele. Em outras palavras, você pode usar `continue` para forçar o loop a saltar para o próximo ciclo.

Salta do ciclo atual de loop, continuando no próximo.

```
continue;
```

Tanto `break` como `continue` são extremamente úteis no controle de ajuste dos loops, mas break oferece uma solução para os problemas imediatos de Seth e Jason em relação ao looping do Mandango.

Saquei essa da instrução break. Ela vai se livrar de todos os loops desnecessários.

Não existem Perguntas Idiotas

P: O resto do código de ação em um loop `for` finaliza o ciclo atual quando a instrução `break` é usada?

R: Não. A instrução `break` força uma finalização imediata do loop, impedindo completamente o fluxo normal dele.

P: Por que embromar o contador do loop para forçar o loop a sair não é uma boa idéia?

R: Porque você não está usando o contador de loop com essa intenção, e dessa maneira corre o risco de introduzir bugs diferentes. Em vez de contar pelos elementos de array como é o esperado, você está forçando o contador a um valor artificial para além da faixa do array apenas para finalizar o loop. Em geral, você precisa confiar que a parte de atualização do loop é o único lugar onde o contador de loop se modifica. Existem sempre casos especiais que surgem onde truques são permitidos, mas este não é um deles – a instrução `break` manipula admiravelmente a saída do loop e sem nenhuma confusão em relação ao que está acontecendo.

Aponte seu Lápis

O loop `for` na função `findSeat()` do Mandango precisa de alguma ajuda quando um usuário aceita uma poltrona. Escreva as linhas que faltam no código que manipula a saída do loop, certificando-se de incluir um comentário para explicar como o código funciona.

```
// Pesquisa por todas as disponibilidades de poltronas
for (var i = 0; i < seats.length; i++) {
  // Verifica se a poltrona atual está disponível
  if (seats[i]) {
    // Define a marcação da poltrona e atualiza sua aparência
    selSeat = i;
    document.getElementById("seat" + i).src = "seat_select.png";
    document.getElementById("seat" + i).alt = "Your seat";

    // Avisa o usuário para aceitar a poltrona
    var accept = confirm("A Poltrona " + (i + 1) + "está disponível, aceita?");

    ..............................................................

    ..............................................................

    ..............................................................

    ..............................................................

    ..............................................................
    else {
      // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
      selSeat = -1;
      document.getElementById("seat" + i).src = "seat_avail.png";
      document.getElementById("seat" + i).alt = "Available seat";
    }
  }
}
```

Aponte seu Lápis
Solução

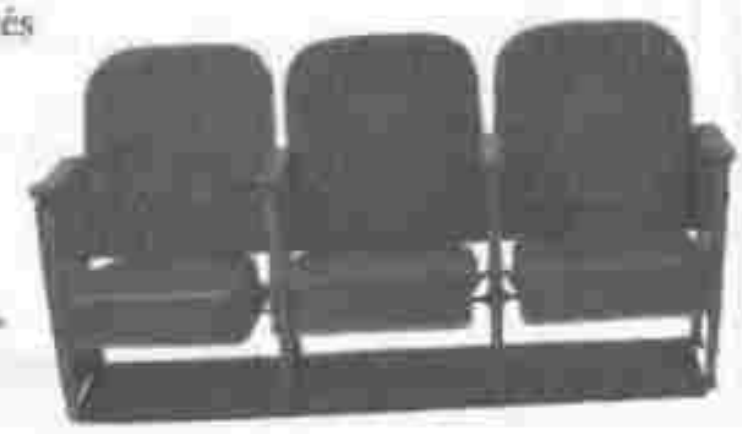
O loop `for` na função `findSeat()` do Mandango precisa de alguma ajuda quando um usuário aceita uma poltrona. Escreva as linhas que faltam no código que manipula a saída do loop, certificando-se de incluir um comentário para explicar como o código funciona.

```
// Pesquisa por todas as disponibilidades de poltronas
for (var i = 0; i < seats.length; i++) {
  // Verifica se a poltrona atual está disponível
  if (seats[i]) {
    // Define a marcação da poltrona e atualiza sua aparência
    selSeat = i;
    document.getElementById("seat" + i).src = "seat_select.png";
    document.getElementById("seat" + i).alt = "Your seat";

    // Avisa o usuário para aceitar a poltrona
    var accept = confirm("A Poltrona " + (i + 1) + "está disponível, aceita?");

    if (accept) {

      // The user accepted the seat, so we're done

      break;

    }
    else {
      // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
      selSeat = -1;
      document.getElementById("seat" + i).src = "seat_avail.png";
      document.getElementById("seat" + i).alt = "Available seat";
    }
  }
}
```

Colocando 'man' em Mandango

A intenção original do Mandango é permitir aos usuários a pesquisa de poltronas de cinema disponíveis em grupos de três. Agora, com a pesquisa por poltronas individuais funcionando, Seth e Jason estão prontos para voltar suas atenções para uma pesquisa verdadeiramente de macho. Eles precisam de uma maneira de verificar uma seqüência de três poltronas disponíveis.

Três poltronas disponíveis em uma fileira... bastante espaço!



Não existem Perguntas Idiotas

P: Ainda não entendi a diferença entre um operador booleano normal e um operador de lógica booleana. Qual é?

R: Bem, antes de tudo, eles todos são operadores booleanos, o que significa que eles sempre resultam em um valor booleano quando terminam suas tarefas. A única diferença é o tipo de dados em que eles operam. Os operadores booleanos normais funcionam em todos os tipos de dados, já que eles realizam comparações gerais, tais como "igual a", "diferente de", "maior que" etc. Os operadores de lógica booleana funcionam somente em dados booleanos e, portanto, realizam comparações lógicas como E, OU e NÃO. Então, os operadores de lógica booleana funcionam somente em informações true/false, enquanto os operadores booleanos normais funcionam em todos os tipos de dados.

P: Então, o operador NOT é um operador de lógica booleana?

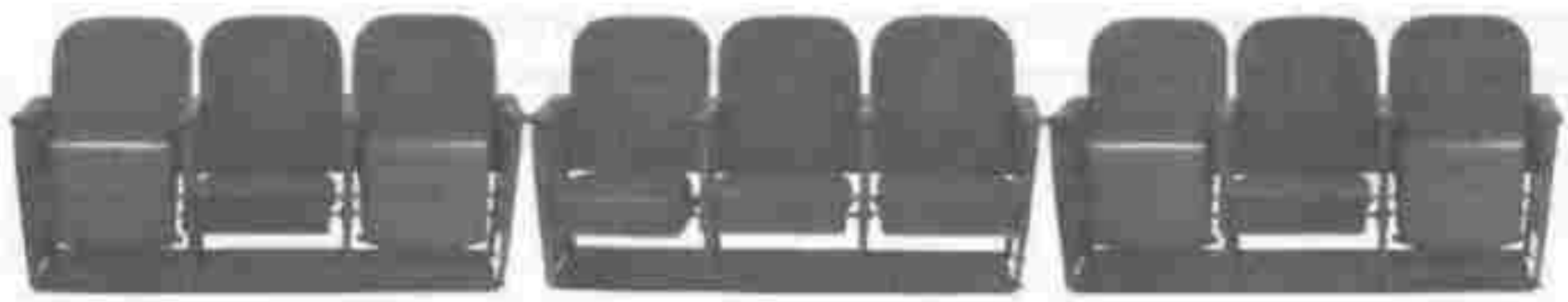
R: Sim. Ele opera somente em um valor booleano, então ele se qualifica como um operador de lógica booleana. Ele também é um operador *unário* já que ele opera em apenas um fragmento de dados.

P: Como os parênteses funcionam em relação aos operadores booleanos?

R: Os parênteses permitem alterar a ordem de avaliação padrão de todos os operadores, e não apenas os operadores booleanos. Agrupar um operador dentro de parênteses força a realização dessa operação antes das que estão em torno dela. Então, `largeDrink && largePopcorn` é forçada para ocorrer antes da operação `||` no código de balas gratuitas, porque ela aparece dentro de ().

Aponte seu Lápis

É a sexta vez que o loop `for` do Mandango circula (`i = 5`), e sua ajuda é necessária para determinar se há três poltronas consecutivas disponíveis, verificando a disponibilidade da poltrona e executando uma lógica booleana.



```
for (var i = 0; i < seats.length; i++) {
  // Verifica se a poltrona atual e as duas poltronas seguintes estão disponiveis
  if (seats[i] && seats[i + 1] && seats[i + 2]) {
    ...
  }
  ...
}
```

..... && ..... && ..... = ...

Aponte seu Lápis
Solução

É a sexta vez que o loop `for` do Mandango circula (`i` = 5), e sua ajuda é necessária para determinar se há três poltronas consecutivas disponíveis, verificando a disponibilidade da poltrona e executando uma lógica booleana.

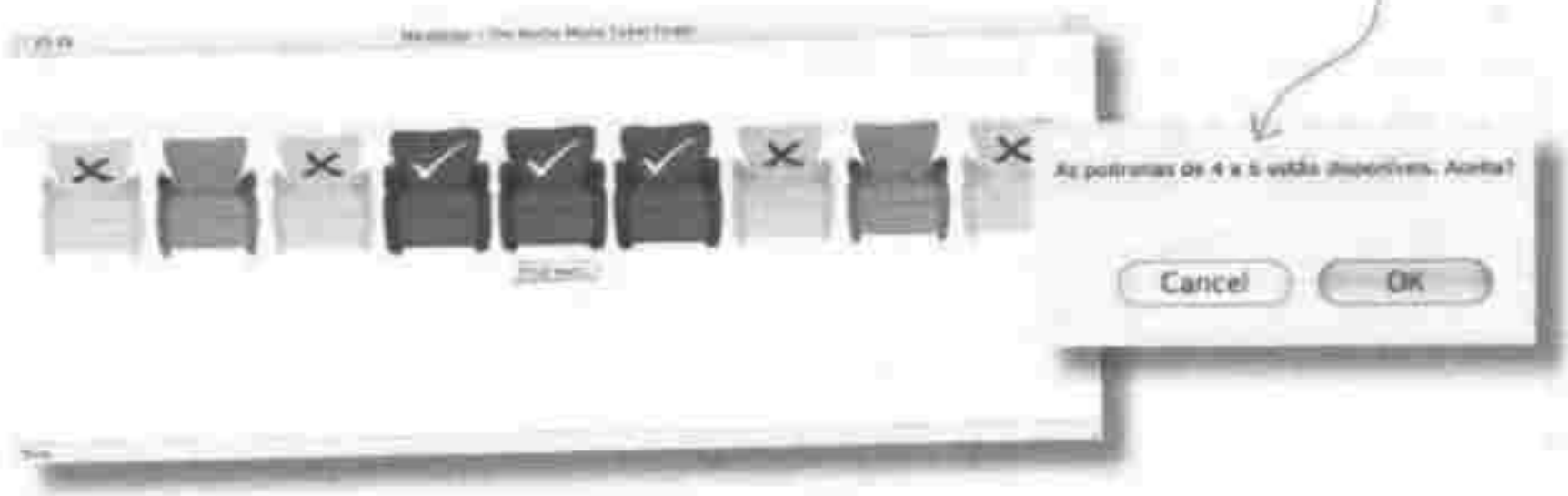
```
for (var i = 0; i < seats.length; i++) {
  // Verifica se a poltrona atual e as duas poltronas seguintes estão disponiveis
  if (seats[i] && seats[i + 1] && seats[i + 2]) {
    ...
  }
  ...
}
```

..... && ..... && ..... = ...

Finalmente, um localizador de poltronas masculinas

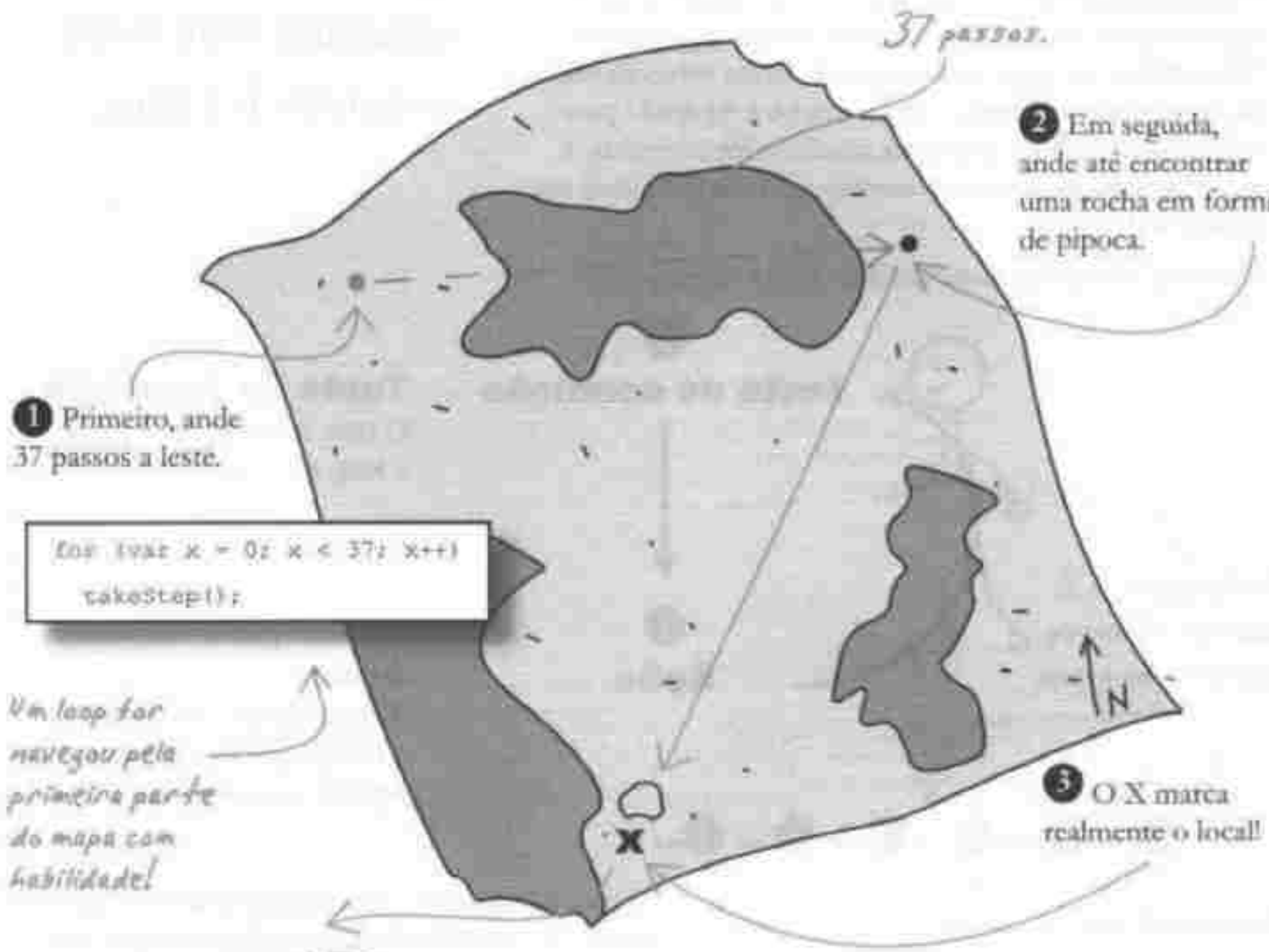
Agora, o Mandango pesquisa corretamente uma seqüência com três poltronas disponíveis, resultando em um serviço de ingressos para cinema que até mesmo o mais durão dos durões apreciará.

O usuário foi avisado para aceitar uma oferta de três poltronas.



De volta ao mapa do tesouro

Com o Mandango temporariamente em boa forma, podemos retornar à busca do tesouro escondido. Lembra do mapa do tesouro?



O tesouro ainda espera...

Um loop `for` funcionou bem na navegação da primeira parte do mapa. A segunda parte ainda permanece, e ela apresenta um desafio que não parece ser adequado às habilidades especiais de um loop `for`. É difícil configurar um contador de loop quando não há idéia de quantas repetições do loop são necessárias.

Poder do cérebro

Qual a diferença entre as duas partes da pesquisa sobre o mapa do tesouro? Como você criaria um loop para percorrer a segunda parte do mapa?

Todo esse papo de cinema me fez lembrar de pipoca... Ah, desculpe. Acho que algumas instruções if aninhadas poderiam eliminar essa pesquisa de três poltronas sem nenhum problema. Eis como eu faria!

O código de perto

Uma seqüência de três poltronas é verificada usando se instruções if aninhadas.

Se três poltronas em uma fileira forem encontradas, atribui a marcação à primeira.

```
for (var i = 0; i < seats.length; i++) {
  // Verifica se a poltrona atual e mais as duas poltronas seguintes estão disponíveis
  if (seats[i]) {
    if (seats[i + 1]) {
      if (seats[i + 2]) {
        // Define a marcação da poltrona e atualiza sua aparência
        selSeat = i;
        document.getElementById("seat" + i).src = "seat_select.png";
        document.getElementById("seat" + i).alt = "Your seat";
        document.getElementById("seat" + (i + 1)).src = "seat_select.png";
        document.getElementById("seat" + (i + 1)).alt = "Your seat";
        document.getElementById("seat" + (i + 2)).src = "seat_select.png";
        document.getElementById("seat" + (i + 2)).alt = "Your seat";

        // Avisa o usuário para aceitar a poltrona
        var accept = confirm("Seats " + (i + 1) + " through " + (i + 3) + " are available. Accept?");
        if (accept) {
          // O usuário aceitou a poltrona, então damos por encerrado
          break;
        }
        else {
          // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
          selSeat = -1;
          document.getElementById("seat" + i).src = "seat_avail.png";
          document.getElementById("seat" + i).alt = "Available seat";
          document.getElementById("seat" + (i + 1)).src = "seat_avail.png";
          document.getElementById("seat" + (i + 1)).alt = "Available seat";
          document.getElementById("seat" + (i + 2)).src = "seat_avail.png";
          document.getElementById("seat" + (i + 2)).alt = "Available seat";
        }
      }
    }
  }
}
```

Altera todas as três poltronas para imagem marcada, de forma que o usuário possa visualizar quais poltronas estão disponíveis.

Se o usuário rejeitar as poltronas, atribua suas imagens de volta à disponível.

* Lembrete: Esse código e todos os códigos e imagens de exemplo do Mandango estão disponíveis em www.altabooks.com.br.

Não seria um sonho se houvesse uma maneira de combinar essas ifs aninhadas em algo um pouco mais elegante?

Uma solução lógica, elegante e bem-bolada com &&

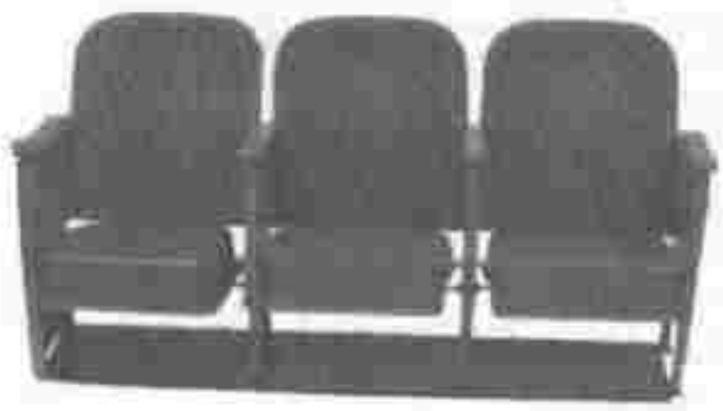
Existe uma maneira melhor de manipular a verificação das três poltronas no Mandango. A versão da `if` aninhada funciona, mas há espaço para melhorias e a mudança envolve essencialmente a criação de um código mais elegante.

Apesar das objeções de Seth, há momentos que valem a pena fazer mudanças em seu código, fazendo com que ele fique mais "elegante", que é uma outra forma de dizer que o código está limpo, eficiente e fácil de entender e manter. No caso das instruções `if` aninhadas, seria mais elegante uni-las dentro de uma única instrução `if`... mas como?

O operador booleano AND compara dois valores booleanos para verificar se ambos são true.

```
if (seats[i] && seats[i + 1] && seats[i + 2]) {
  ...
}
```

true true true



O operador booleano AND (`&&`) compara dois valores booleanos para verificar se ambos são `true`. Neste código do Mandango, dois operadores AND são usados juntos para verificar se os três valores de poltronas são `true`. Se forem, você sabe que há uma seqüência com três poltronas disponíveis. Resolvido o problema... e com um pequeno toque de elegância!

A lógica do operador booleano revelada

Você deve ter visto vários operadores de comparação, como == e <. A maioria dos operadores de comparação que já viu compara dois valores e produz um resultado booleano. Os operadores de **lógica** booleana também produzem um resultado booleano, mas eles operam apenas nos valores booleanos – eles realizam comparações de lógica booleana.

Você já deve ter visto esse!

AND

`a && b`

True se a AND b forem true, se não false.

OR

`a || b`

True se a OR b forem true, se não false.

NOT

`!a`

False se a for true, true se a for false.

Os operadores de lógica booleana podem ser combinados juntos para criar comparações lógicas mais interessantes, normalmente visando tomar decisões complexas.

Os parênteses permitem agrupar expressões de lógica booleana.

```
if ((largeDrink && largePopcorn) || coupon)
   freeCandy();
```

Você pode ganhar balas de graça comprando um combo OU comprando um cupom.

Neste exemplo, um operador AND é usado para selecionar uma bebida grande e uma pipoca grande... combo! Você ganha balas de graça com um combo. Ou, há um outro caminho para as balas gratuitas graças ao operador OR – um cupom. Então, você pode ganhar balas de graça pedindo uma bebida grande E uma pipoca grande, OU apresentando um cupom. Esse tipo de decisão seria extremamente difícil de se tomar sem a ajuda de operadores de lógica booleana.

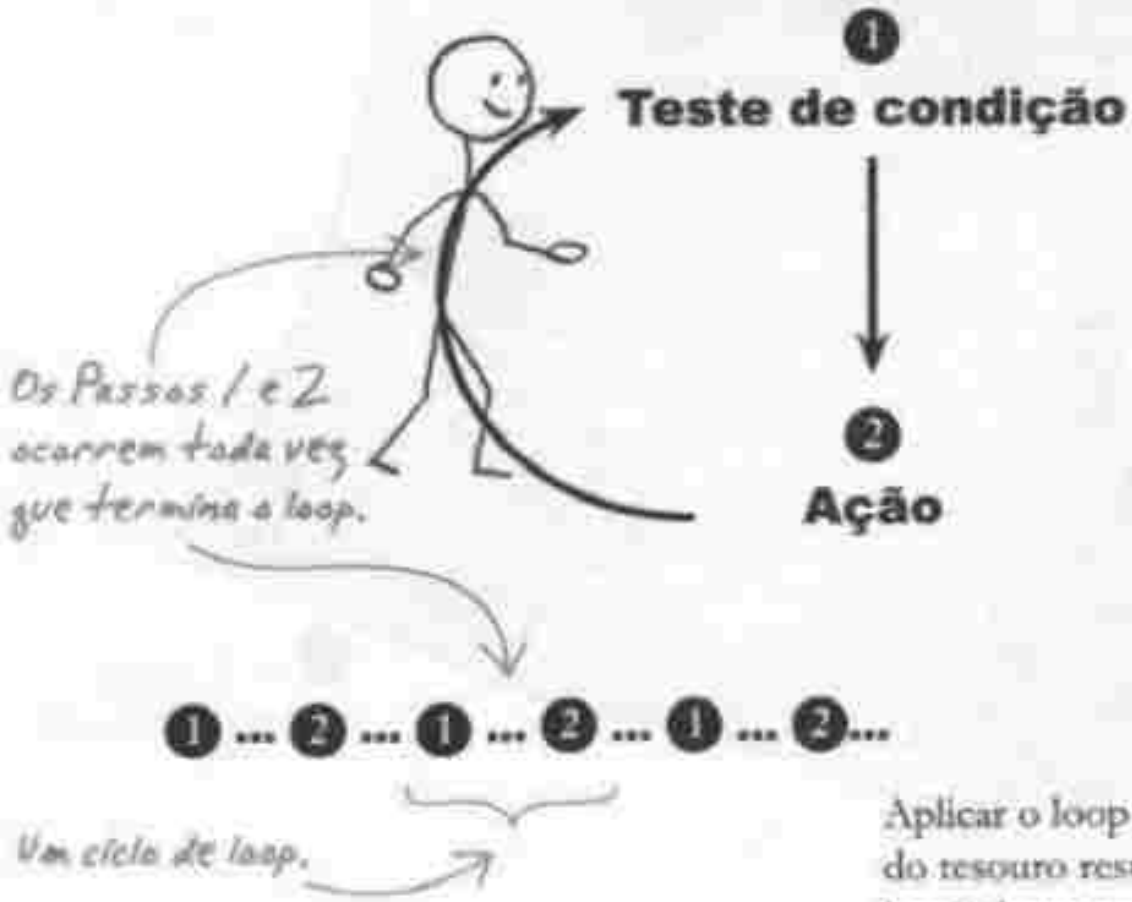
Operadores de lógica booleana podem ser combinados para tomar decisões complexas.

Execução de um loop for por "enquanto"... até cumprir uma condição

loops while permitem repetir o código enquanto uma certa condição for true.

Embora seja possível criar um loop `for` que percorra a segunda parte do mapa do tesouro, existe uma opção melhor. Ao contrário do loop, que é estruturado em torno da noção de um contador de loop, o loop `while` é equipado para a execução **enquanto** uma certa condição for cumprida. E essa condição não tem necessariamente nada a ver com um contador de loop.

Um loop `while` consiste em duas partes diferentes:



Teste de condição
O teste de condição verifica se o loop deve realizar um ciclo.

Ação
A parte da ação do loop é o código que na verdade é repetido em cada ciclo.

Teste de condição ❶
Só realiza um outro ciclo de loop se o teste resultar em true, ou seja, se a rocha ainda não estiver visível.

`!rockVisible`

Ação ❷
Chame a função `takeStep()` para dar um passo.

`takeStep()`

Aplicar o loop `while` na segunda parte do mapa do tesouro resulta em alguns códigos simples, por incrível que pareça, pelo menos quando comparado a um loop `for`:

```
❶
while (!rockVisible)

  takeStep();
❷
```

Eis como funcionam as diferentes partes desse loop `while`:

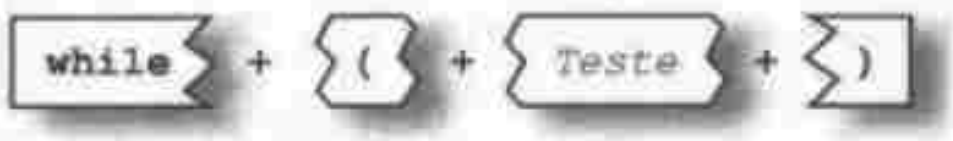
❶ Verifica se a rocha **não** está visível. Se não estiver, avança para o Passo 2 e realiza um ciclo pelo loop. Caso contrário, encerra o loop.

❷ Executa o código de ação do loop, que neste caso significa executar a função `takeStep()`.

O loop while decomposto

Muito mais simples na estrutura que os loops `for`, os loops `while` ainda precisam seguir uma fórmula previsível:

O teste de condição, que precisa avaliar se true ou false.



A ação a ser repetida, que é uma instrução simples (ou composta).

Preste atenção! Já que os loops while não possuem um fragmento de código embutido que atualiza o loop, certifique-se de que existe código dentro do loop que de alguma forma afeta o teste de condição. Caso contrário, você se arrisca criando um loop infinito.

Seja cauteloso com os testes de condição do loop while.

Exercício

A partir do exercício do rolo de filme, reescreva o código do loop que avisa ao usuário para digitar um número maior que 0 e, em seguida, usa esse número como início da contagem de um loop que realiza a contagem regressiva de um rolo de filme antigo (4, 3, 2, 1, Rodando!). Dessa vez, use um loop `while` em vez de um loop `for`.

```
var count = prompt("Digite um número maior que 0:", "10");

if (count > 0) {

.............................................................

.............................................................

.............................................................

.............................................................

.............................................................

.............................................................

}

else

   alert("O número não era maior que 0. Sem filme!");
```

Solução do Exercício

A partir do exercício do rolo de filme, reescreva o código do loop que avisa ao usuário para digitar um número maior que 0 e, em seguida, usa esse número como início da contagem de um loop que realiza a contagem regressiva de um rolo de filme antigo (4, 3, 2, 1, Rodando!). Dessa vez, use um loop `while` em vez de um loop `for`.

Armazena o número na variável count.

Avisa o usuário para digitar um número.

```
var count = prompt("Digite um número maior que 0:", "10");

if (count > 0) {
  var x = count;

  while (x > 0) {

    alert("Começar em..." + x);

    x--;

  }

  alert("Rodando!");

}
else

    alert("O número não era maior que 0. Sem filme!");
```

Contagem regressiva até 0.

Certifica-se de que count é maior que 0.

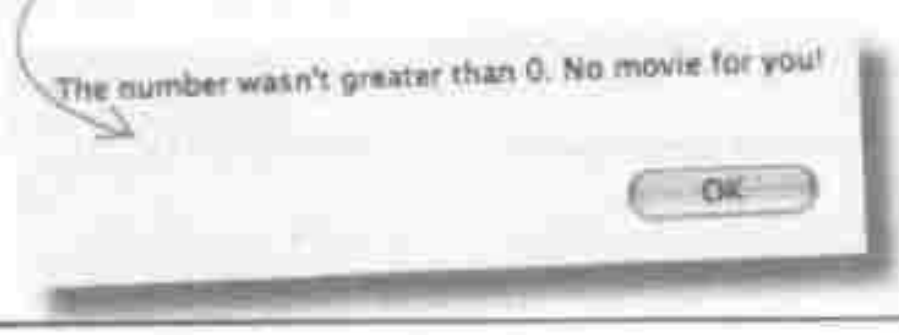
Ainda existe um contador, mas ele é criado fora do loop while.

Diminui o contador como parte da ação do loop, igual a: x=x-1.

A ação do loop é uma instrução composta.

Dados inválidos.

Terminou a contagem!



Pontos de Bala

- A instrução `break` escapa **imediatamente** de um loop, ignorando qualquer código de loop restante.
- O loop `while` executa um fragmento de código **enquanto** um certo teste de condição se mantém true.
- Os operadores de lógica booleana permitem criar uma lógica eficiente true/false para tomada de decisões.
- Evite um loop infinito certificando-se de que o teste de condição é de algum modo afetado pelo código dentro do loop `while`.

Use o loop certo para a tarefa

O exercício da contagem regressiva de um rolo de filme revelou que os loops `for` e `while` são freqüentemente capazes de resolver os mesmos problemas. De fato, qualquer loop `for` pode ser reconstruído como um loop `while` usando-se a seguinte fórmula:

```
Init + ;

while + ( + Teste + ) + {
        Ação + ;
        Atualização + ;
}
```

Um loop while pode fazer tudo que um loop for faz, e vice-versa.

Portanto, é tecnicamente possível codificar os mesmos loops usando `for` ou `while`. Mesmo assim, você achará, na maioria dos casos, que um dos loops parece funcionar claramente melhor em termos de código, o que faz sentido. Isso talvez tenha algo a ver com elegância?

A inicialização acontece fora do loop.

A atualização acontece dentro da ação.

```
for (var i = 0; i < 10; i++)
  alert(i);
```

=

```
var i = 0;
while (i < 10) {
  alert(i);
  i++;
}
```

Esses loops são bem apropriados para os problemas que eles resolvem.

Sem inicialização.

Sem atualização.

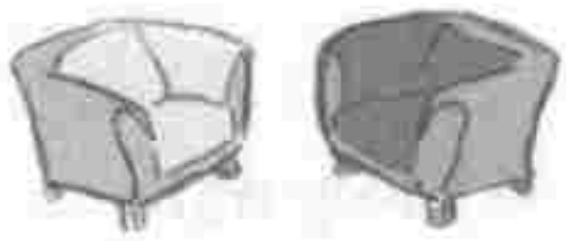
```
while (!finished)
  doIt();
```

=

```
for (; !finished; )
  doIt();
```

Escolher um loop `for` versus um loop `while` tem tudo a ver com usar a ferramenta certa para a tarefa. Em outras palavras, a mecânica do loop deve se encaixar no problema que está prestes a acontecer.

Conversa informal



Conversa desta noite: Loop for e loop while tentam incansavelmente se repetir

Loop for:

Ah, aqui estamos... apenas uma dupla de colegas repetitivos se encontrando.

Não sou tão complicado assim, apenas adiciono um pouco mais de estrutura para criar certos tipos de loops. Quando as pessoas querem executar um loop usando algum tipo de contador numérico, eles encontram satisfação na maneira fácil que eu tenho que inicializar e atualizar o contador que me controla.

Parece-me um tanto vago, embora suponho que ele poderia funcionar. Eu gostaria de ser mais exato, entende, vigiar bem de perto o que me faz funcionar. Eis porque faço um esforço específico para me inicializar antes mesmo de eu começar a executar o loop. Eu também me mantenho atualizado no fim de cada loop, apenas para me certificar de que continuarei executando como esperado. Acho que sou um pouco compulsivo sobre me certificar de que eu me repito com uma exatidão rigorosa.

Tenho consciência de que existem várias maneiras diferentes de estruturar os loops. Apenas gosto de conduzir com pulso firme.

Loop while:

Sim. Se bem que eu tenho que dizer que não sou tão louco pelas diferentes etapas que envolvem fazer você funcionar. Parece-me um tanto complicado.

Isso é verdade, mas executar um loop não diz respeito somente à contagem, entende? Existem todos os tipos de loops excelentes que nem envolvem números. Às vezes, você só precisa da simplicidade de dizer, "Ei, continue fazendo isso por algum tempo". Esse é o meu tipo de loop.

Enquanto eu aplaudo seu trabalho ético, você percebe que é igualmente possível executar um loop de forma confiável e previsível sem toda essa coisa formal de inicialização e atualização? Além disso, eu sempre repito o código em situações onde não há necessidade de inicializar nada, e a atualização acontece lá no código de ação. Por isso, me contento em executar sem formalidade, apenas me concentrando na execução do loop.

Loop for:

Isso é verdade. A boa notícia é que nós concluímos nosso trabalho da nossa maneira. E eu até consigo enxergar onde o meu estilo é um pouco demais para um loop com controles lógicos simples.

Você pode dizer isso novamente!

Sem problemas, eu entendo. Agradeço o bate-papo.

Loop while:

Suponho que isso seja realmente uma questão de estilo, e cada loop possui o seu próprio. Você gosta de manter todos os controles do loop em seu lugar, enquanto sou um pouco mais informal em relação a como sou controlado.

É assim que se fala! Afinal de contas, acho que há espaço suficiente para nós dois.

Eu acho que há espaço suficiente... ih, suponho que meus instintos se manifestaram por um momento. Desculpe.

Não existem Perguntas Idiotas

P: O loop `while` parece muito simples. Deixei escapar alguma coisa?

R: De jeito nenhum. Apenas lembre-se de que simples não significa necessariamente fraco ou limitado. Em outras palavras, você pode se surpreender com a grande eficácia dos loops `while`. Obviamente, o loop `while` consiste somente em um teste de condição e um fragmento de código de ação, mas em muitos casos isso é tudo que você precisa para executar um loop realmente eficiente. Especialmente se considerar que o teste de condição pode se tornar muito mais interessante graças aos operadores de lógica booleana. Não só isso, mas a parte da ação do loop `while` pode conter a mesma quantidade de código que precisar se usar uma instrução composta.

P: O que aconteceria se eu criasse um loop while que iniciasse `while (true)`..., funcionaria?

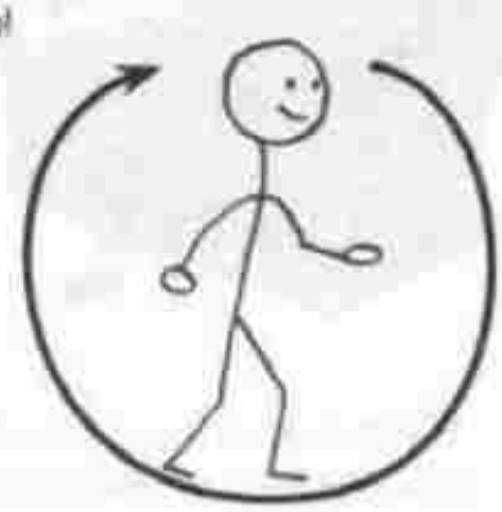
R: Sim, funcionaria... talvez muito bem. O problema é que você criou apenas um loop infinito, porque o teste de condição é permanentemente `true`. Um loop `while` continua se repetindo até que o teste de condição avalie como `false`, e neste caso essa hora nunca chega. É assustador pensar em quantos loops infinitos estão executando exatamente neste momento em que você está lendo isso, fadados a se repetirem para sempre, eternamente... ei, caia fora disso!

P: É possível que um código de ação do loop (o código entre parênteses) nunca seja chamado?

R: Sim. Tanto o loop `for` quanto o loop `while` exigem que seus testes de condição sejam `true`, antes de iniciar a execução do código de ação. Portanto, se por alguma razão o teste de condição falhar logo no início, o código de ação não executará e o loop sairá antes mesmo de começar.

P: Os loops podem ser aninhados um dentro do outro?

R: Ah, sim! Os loops aninhados permitem mais de um nível de repetição. Isso parece estranho agora, mas é muito legal. Exploraremos os loops aninhados mais adiante quando o Mandango se desenvolver para pesquisar um cinema inteiro!



O tesouro no fim do loop

Com o uso de um loop `for` seguido de um loop `while`, o mapa do tesouro pode ser integralmente percorrido, levando ao tesouro no local marcado com um X.

37 passos.

❷ Em seguida, ande até encontrar uma rocha em forma de pipoca.

❶ Primeiro, ande exatos 37 passos a leste.

```
for (var x = 0; x < 37; x++)
  takeStep();
```

```
while (!rockVisible)
  takeStep();
```

Um loop while navegou a segunda parte do mapa sem falhar!

Um loop for navegou pela primeira parte do mapa com habilidade!

N

X

❸ O X marca realmente o local!

O baú do tesouro se abre para revelar...

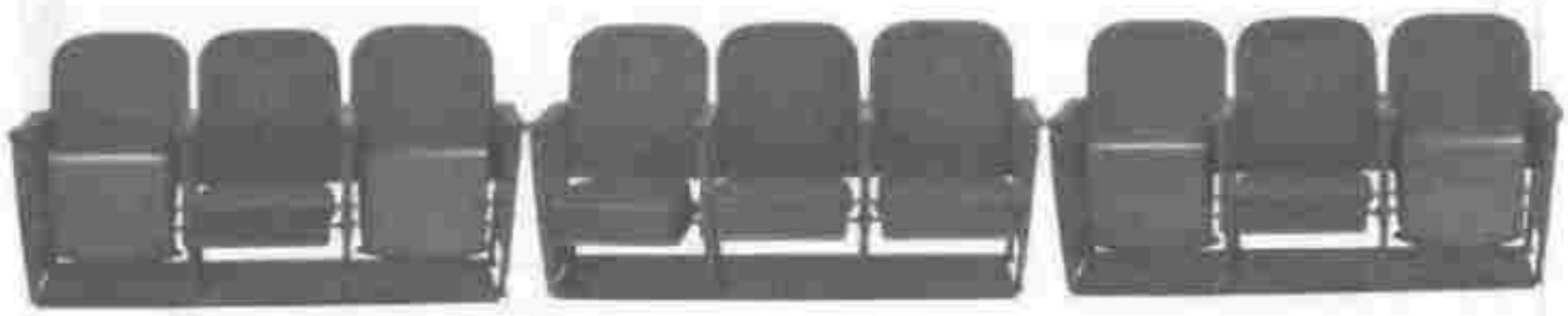
Ingressos de cinema!

Isso poderia ser um sinal? O seu recente conhecimento sobre a execução do loop `while` e os ingressos de cinema podem levá-lo de volta somente a uma coisa... Mandango!

Aponte seu Lápis

Reescreva o loop na função `findSeats()` do Mandango, de modo a usar um loop `while` em vez de um loop `for`. Adicione uma nova variável de controle de loop, `finished`, que seja usada como um meio de sair do loop pelo teste de condição, ao contrário do uso de `break`.

```
.....................................................................
.....................................................................
.....................................................................
  // Verifica se a poltrona atual e mais as duas poltronas seguintes estão disponíveis
  if (seats[i] && seats[i + 1] && seats[i + 2]) {
    // Coloca a marcação das poltronas e atualiza a aparência delas
    ...
    // Avisa o usuário para aceitar as poltronas
    var accept = confirm("As Poltronas de" + (i + 1) + "a" + (i + 3) +
      "estão disponíveis, aceita?");
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
    else {
      // O usuário rejeitou as poltronas, então destaca a marcação e continua procurando
      ...
    }
  }
  // Aumenta o contador do loop
.....................................................................
}
```



Aponte seu Lápis
Solução

Reescreva o loop na função `findSeats()` do Mandango, de modo a usar um loop `while` em vez de um loop `for`. Adicione uma nova variável de controle de loop, `finished`, que seja usada como um meio de sair do loop pelo teste de condição, ao contrário do uso de `break`.

Inicializa o contador de loop e a variável finished.

Executa um loop, já que o contador de loop é menor que o número de poltronas & finished não é true.

Esse loop é um pouco mais híbrido do que temos visto até agora, porque ele é dependente tanto de uma expressão de contagem quanto de uma expressão de lógica booleana. Codificar loops híbridos normalmente fica mais simples usando while.

```
var i = 0, finished = false;

while (i < seats.length && !finished) {

  // Verifica se a poltrona atual e mais as duas poltronas seguintes estão disponíveis
  if (seats[i] && seats[i + 1] && seats[i + 2]) {
    // Define a marcação das poltronas e atualiza a aparência delas
    ...

    // Avisa o usuário para aceitar as poltronas
    var accept = confirm("As Poltronas de" + (i + 1) + "a" + (i + 3) +
      "estão disponíveis, aceita?");

    if (accept) {
      // O usuário aceitou as poltronas, então damos por encerrado

      finished = true;

    }

    else {
      // O usuário rejeitou as poltronas, então desfaz a marcação e continua procurando
      ...

    }
  }
  // Aumenta o contador do loop
  i++;

}
```

Aumenta o contador de loop.

Define finished como true para sair do loop. Como isso afeta o teste de condição, não há necessidade de interromper aqui.

Parece que o Mandango está muito satisfeito com a execução do loop e tudo o mais, mas não existem muitos cinemas com apenas uma única fileira de poltronas. Precisamos descobrir como lidar com mais fileiras...

Modelagem de dados de poltronas de cinema

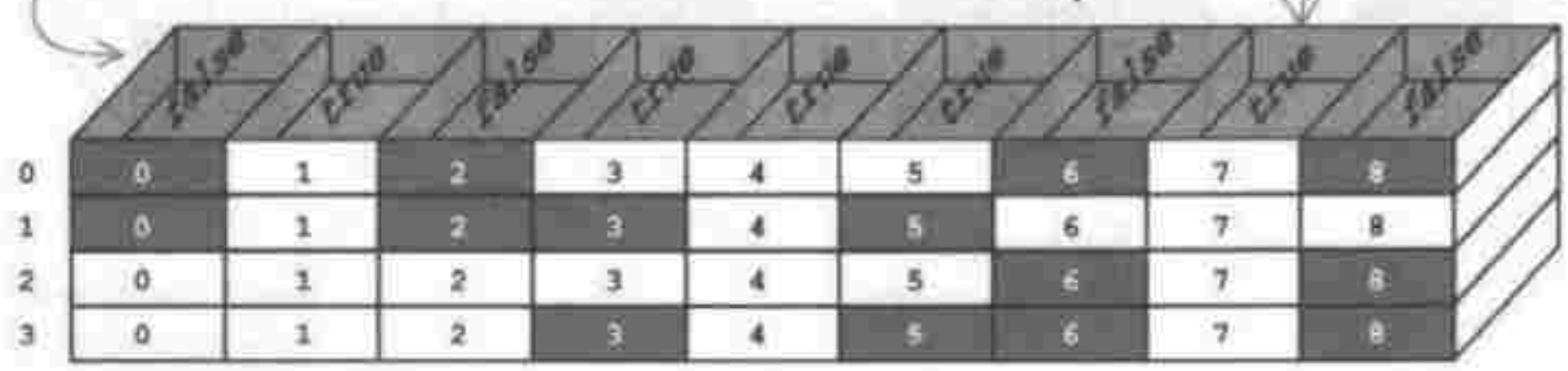
Jason está certo. O Mandango realmente precisa ser capaz de lidar com mais fileiras de poltronas para se tornar verdadeiramente funcional. Até aqui, uma única fileira de poltronas fazia sentido, porque ela mapeava claramente um array de booleanos que representa a disponibilidade de poltronas. Para expandir a ideia de várias fileiras de poltronas é necessário a expansão do array, e isso requer uma outra dimensão. Isso mesmo, estamos falando de um array bidimensional!

Esse cinema possui quatro fileiras de poltronas com nove poltronas em cada fileira. Sim, é acanhegante!

Cada item no array bidimensional continua a ser um booleano.

Precisamos de um array que tenha um tamanho de 9 x 4 para corresponder às poltronas atuais, que são quatro fileiras com nove poltronas em cada.

Agora, existe uma outra dimensão de índices de array.



Um array de um array: arrays bidimensionais

Você não precisa de óculos especiais ou qualquer coisa para criar um array bidimensional. De fato, criar um array bidimensional é semelhante a criar um array normal (unidimensional), exceto se criar vários sub-arrays como elementos do array. São esses sub-arrays que adicionam a segunda dimensão, resultando em uma **tabela** de dados que possui fileiras e colunas.

Depois, cria sub-arrays que servem como elementos no array externo. São duas dimensões!

```
var seats = new Array(new Array(9), new Array(9), new Array(9), new Array(9));
```

Primeiro, cria um array para abrigar os sub-arrays. É uma dimensão!

Quatro sub-arrays resultam em quatro fileiras de dados de array.

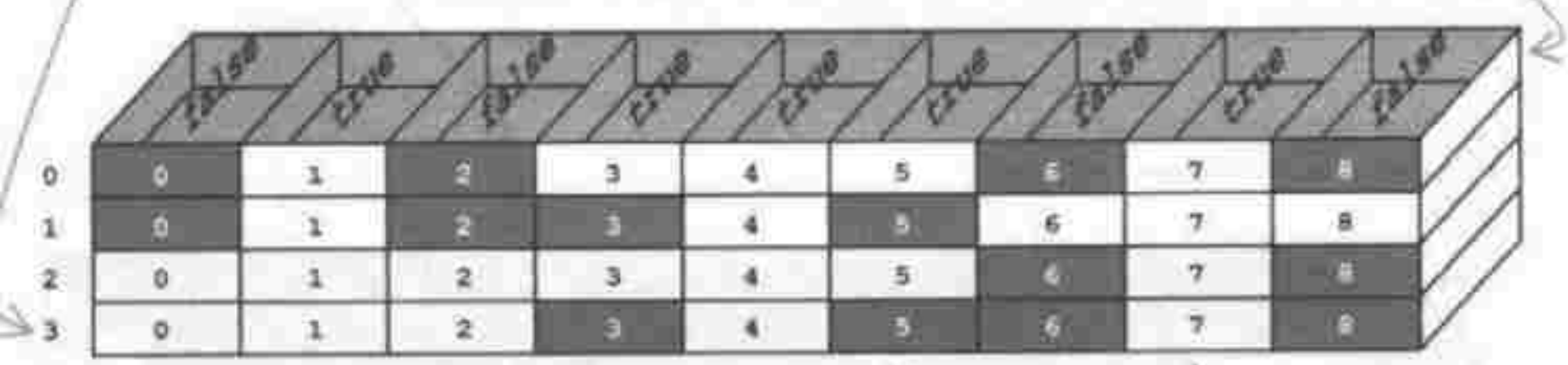
No caso do Mandango, nós já sabemos os valores iniciais dos elementos de array, então faz sentido usarmos uma abordagem **diferente** para criar o array bidimensional, uma que envolva uma literal de array. Ela **cria** o array e ao mesmo tempo **inicializa-o** – uma situação de ganho mútuo!

Chaves duplas indicam um array bidimensional.

```
var seats = [[ false, true, false, true, true, true, false, true, false ],
  [ false, true, false, false, true, false, true, true, true ],
  [ true, true, true, true, true, true, false, true, false ],
  [ true, true, true, false, true, false, false, true, false ]];
```

Cada sub-array possui seu próprio índice de array, neste caso entre 0 e 3.

A primeira lista de valores booleanos está na primeira fileira do array bidimensional.



False – a poltrona já está ocupada.

True – a poltrona está disponível.

Duas chaves para acessar dados de array bidimensional

Acessar dados em um array bidimensional não é diferente de acessar um array unidimensional, exceto se tiver que fornecer um fragmento de informação adicional: o índice do array extra. Mais especificamente, você determina os índices da **fileira** e da **coluna** onde os dados se encontram no array. Por exemplo, para pegar o valor da quarta poltrona na segunda fileira de poltronas, use este código:

Arrays bidimensionais permitem armazenar dados em tabela.

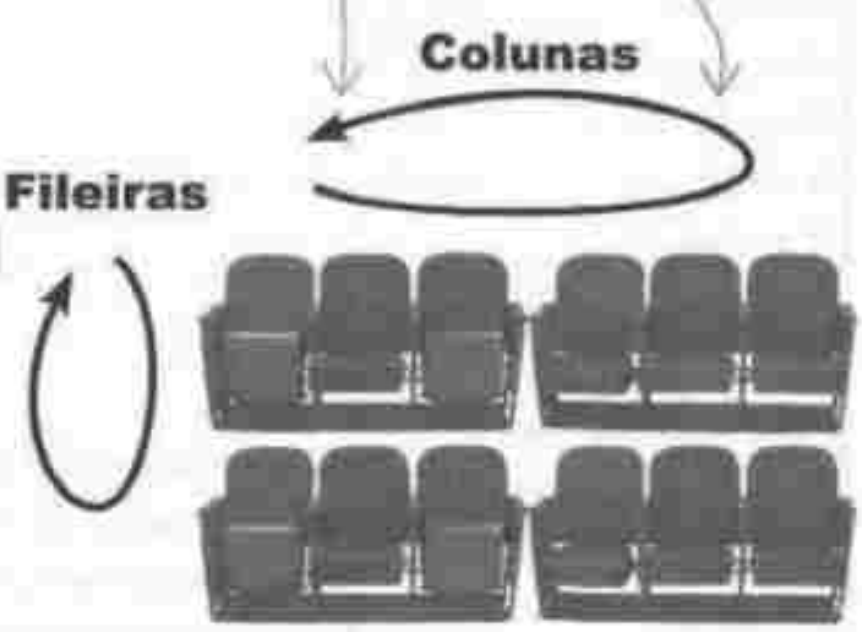
O índice da segunda fileira no array é 1 (começa em 0).

```
alert(seats[1][3]);
```

O índice do quarto elemento em uma fileira é 3 (começa em 0).

Duas dimensões.

Uma dimensão.



Executar um loop por um array com mais de uma dimensão envolve o aninhamento de um loop em cada dimensão. Portanto, executar um loop por um array bidimensional envolve um total de dois loops, um dentro do outro. O loop externo circula pelas fileiras de dados de array, enquanto o loop interno circula dentro de uma fileira.

Loops aninhados permitem a repetição através das duas dimensões dos dados.

Aponte seu Lápis

Escreva o código para executar um loop pelas poltronas no array de `poltronas` bidimensionais, alertando o usuário sobre a disponibilidade de cada poltrona.

Aponte seu Lápis
Solução

Escreva o código para executar um loop pelas poltronas no array de `poltronas` bidimensionais, alertando o usuário sobre a disponibilidade de cada poltrona.

O loop externo circula pelas fileiras de poltronas usando o contador de loop i.

O loop interno circula pelas poltronas em uma fileira usando o contador de loop j.

Esse é o comprimento do sub-array localizado na fileira i.

Ele realiza dois loops aninhados pelas duas dimensões de dados de array.

```
for (var i = 0; i < seats.length; i++) {
  for (var j = 0; j < seats[i].length; j++) {
    if (seats[i][j])
      alert("A Poltrona " + i + " na fileira " + j + " esta disponivel.");
    else
      alert("A Poltrona " + i + " na fileira " + j + " não esta disponivel.");
  }
}
```

Para acessar uma poltrona individual, use tanto a fileira (i) como a coluna (j) dos dados de array.

Exibe um alerta diferente com base na disponibilidade (true) ou na indisponibilidade (false) das poltronas.

As mensagens de disponibilidade de poltrona mostram a fileira e a coluna de cada poltrona.

Não existem Perguntas Idiotas

P: Os arrays podem ter mais de duas dimensões?

R: Sim, embora em algum ponto possa ficar complicado visualizar os dados. Três dimensões podem ser úteis para modelar dados do mundo real, como as coordenadas x-y-z de um ponto no espaço. Além disso, as dimensões adicionais provavelmente são transferidas para situações muito isoladas. Ao adicionar uma outra dimensão, pense em termos de substituir elementos de array individuais por sub-arrays.

P: Posteriormente, posso adicionar mais dados a um array, se eu inicializá-lo com os dados na criação?

R: Com certeza. Você é livre para adicionar mais dados a um array sempre que quiser, atribuindo novos dados a `seats[4]` e também pode chamar o método `push()` do objeto `Array` para adicionar um novo item ao final do array.

P: Os arrays bidimensionais precisam conter o mesmo número de fileiras?

R: Não, não exatamente. Lembre-se apenas de que, se as fileiras não contiverem o mesmo número de elementos, você está preparando a receita de um desastre na execução de um loop, porque os loops aninhados normalmente são criados para circular pelo comprimento uniforme de um sub-array. Aí sim, é possível variar o comprimento de fileiras de arrays bidimensionais, mas é uma proposta arriscada que é mais seguro evitar.

Pontos de Bala

- Arrays bidimensionais permitem armazenar fileiras e colunas de dados em estruturas de **tabela**.
- Ao acessar um fragmento de dados individual em um array bidimensional, você precisa especificar tanto a **fileira quanto a coluna** do índice.
- **Loops aninhados** podem ser usados para se repetirem pelos dados em um array bidimensional.
- Como nos arrays normais, os arrays bidimensionais podem ser criados e inicializados a partir das literais do objeto de array.

Mandango em 2-D

Embora você já tenha passado pelos fragmentos e partes do código, migrar o Mandango de uma simples fileira de poltronas para um cinema inteiro de poltronas envolve um trabalho dobrado em grande parte do código de script para explicar os dados bidimensionais.

Poder do cérebro

De que diferentes formas os arrays bidimensionais afetam o Mandango quando ele migra para trabalhar com dados de todo um cinema de poltronas? Como você visualizaria o código de script?

O Mandango em 2-D de perto

mandango.html

Não precisa visualizar... eis o código completo do Mandango em 2-D!

```
<html>
  <head>
    <title>Mandango - The Macho Movie Ticket Finder</title>

    <script type="text/javascript">
      var seats = [[ false, true, false, true, true, true, false, true, false ],
                   [ false, true, false, false, true, false, true, true, true ],
                   [ true, true, true, true, true, true, false, true, false ],
                   [ true, true, true, false, true, false, false, true, false ]];
      var selSeat = -1;

      function initSeats() {
        // Inicializa a aparência de todas as poltronas
        for (var i = 0; i < seats.length; i++) {
          for (var j = 0; j < seats[i].length; j++) {
            if (seats[i][j]) {
              // Define a poltrona como disponível
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
            }
            else {
              // Define a poltrona como indisponível
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_unavail.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Unavailable seat";
            }
          }
        }
      }

      function findSeats() {
        // Se a poltrona já estiver marcada, reinicializa todas as poltronas para removê-las
        if (selSeat >= 0) {
          selSeat = -1;
          initSeats();
        }

        // Pesquisa por todas as disponibilidades de poltronas
        var i = 0, finished = false;
        while (i < seats.length && !finished) {
          for (var j = 0; j < seats[i].length; j++) {
            // Verifica se a poltrona atual e mais as duas poltronas seguintes estão disponíveis
            if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
              // Define a marcação da poltrona e atualiza sua aparência
              selSeat = i * seats[i].length + j;
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
              document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
              document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

              // Avisa o usuário para aceitar a poltrona
              var accept = confirm("As Poltronas de " + (j + 1) + " a " + (j + 3) +
                " na Fileira " + (i + 1) + " estão disponíveis, aceita?");
              if (accept) {
                // O usuário aceitou a poltrona, então damos por encerrado (sai do loop interno)
                finished = true;
                break;
              }
              else {
```

Não precisa visualizar... eis o código completo do Mandango em 2-D!

O array bidimensional das variáveis booleanas de disponibilidade de poltronas é criado.

Reinicializa as poltronas se o usuário estiver começando uma nova pesquisa por poltronas, clicando novamente no botão Encontrar Poltronas.

Beneficiando-se do melhor de ambos os mundos, um loop while é usado para circular pelas fileiras, enquanto um loop for circula por poltronas individuais em uma fileira.

```
            // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
            selSeat = -1;
            document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
            document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
            document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
            document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
            document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
            document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
        }
    }
}

// Aumenta o contador de loop externo
i++;
}
}
</script>
</head>

<body onload="initSeats();">
  <div style="margin-top:20px; text-align:center">
    <img id="seat0" src="" alt="" />
    <img id="seat1" src="" alt="" />
    <img id="seat2" src="" alt="" />
    <img id="seat3" src="" alt="" />
    <img id="seat4" src="" alt="" />
    <img id="seat5" src="" alt="" />
    <img id="seat6" src="" alt="" />
    <img id="seat7" src="" alt="" />
    <img id="seat8" src="" alt="" /><br />
    <img id="seat9" src="" alt="" />
    <img id="seat10" src="" alt="" />
    <img id="seat11" src="" alt="" />
    <img id="seat12" src="" alt="" />
    <img id="seat13" src="" alt="" />
    <img id="seat14" src="" alt="" />
    <img id="seat15" src="" alt="" />
    <img id="seat16" src="" alt="" />
    <img id="seat17" src="" alt="" /><br />
    <img id="seat18" src="" alt="" />
    <img id="seat19" src="" alt="" />
    <img id="seat20" src="" alt="" />
    <img id="seat21" src="" alt="" />
    <img id="seat22" src="" alt="" />
    <img id="seat23" src="" alt="" />
    <img id="seat24" src="" alt="" />
    <img id="seat25" src="" alt="" />
    <img id="seat26" src="" alt="" /><br />
    <img id="seat27" src="" alt="" />
    <img id="seat28" src="" alt="" />
    <img id="seat29" src="" alt="" />
    <img id="seat30" src="" alt="" />
    <img id="seat31" src="" alt="" />
    <img id="seat32" src="" alt="" />
    <img id="seat33" src="" alt="" />
    <img id="seat34" src="" alt="" />
    <img id="seat35" src="" alt="" /><br />
    <input type="button" id="findseats" value="Find Seats" onclick="findSeats();" />
  </div>
</body>
</html>
```

A função initSeats() é chamada quando a página é carregada pela primeira vez.

Os contadores de loop de fileira e coluna são necessários para modificar as imagens das poltronas e alterar o texto.

Quatro fileiras com nove poltronas cada exige 36 imagens HTML... uau!

A função findSeats() é chamada quando o usuário clica no botão Procurar poltronas.

Relaxe

Não se intimide pelo tamanho deste código.

Ele usa as mesmas técnicas de array bidimensional, mas agora está incluído no contexto do Mandango, com todo o código e imagens HTML (disponível para download em www.altabooks.com.br).

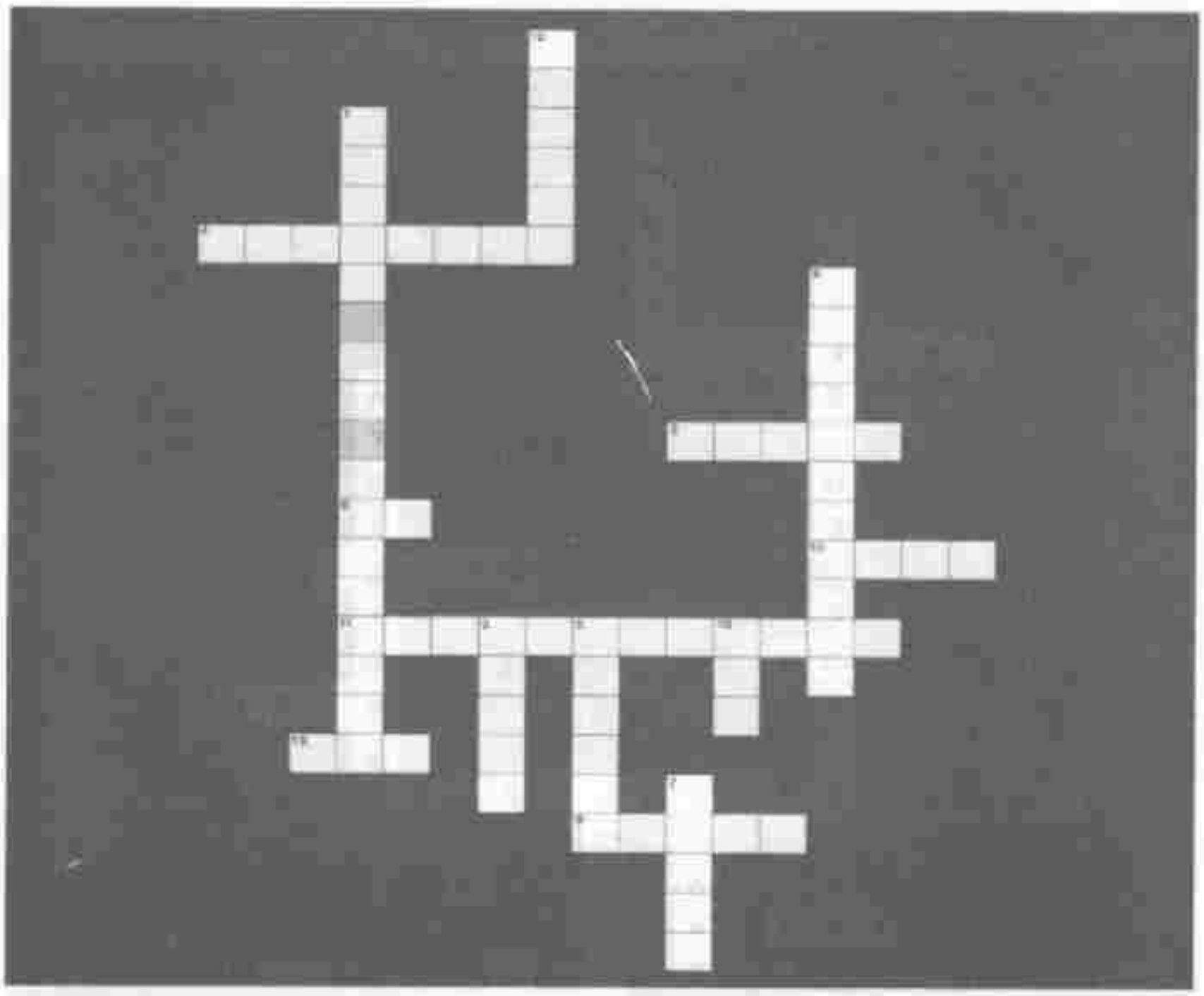
Um cinema inteiro de poltronas masculinas

Com duas dimensões para trabalhar, Seth e Jason estão aptos a levar o Mandango ao próximo nível e oferecer suporte à pesquisa de poltronas em cinemas amplos... numa guinada de macho! Os caras estão eufóricos.

O Mandango agora oferece aos cinéfilos machos uma alternativa de três poltronas em uma fileira dentro de várias opções em um cinema.

Cruzadas do JavaScript

Todo esse papo sobre poltronas provavelmente lhe deu coceira para assistir a um filme no cinema. Antes de sair, tente fazer um pequeno alongamento mental com essas palavras cruzadas.



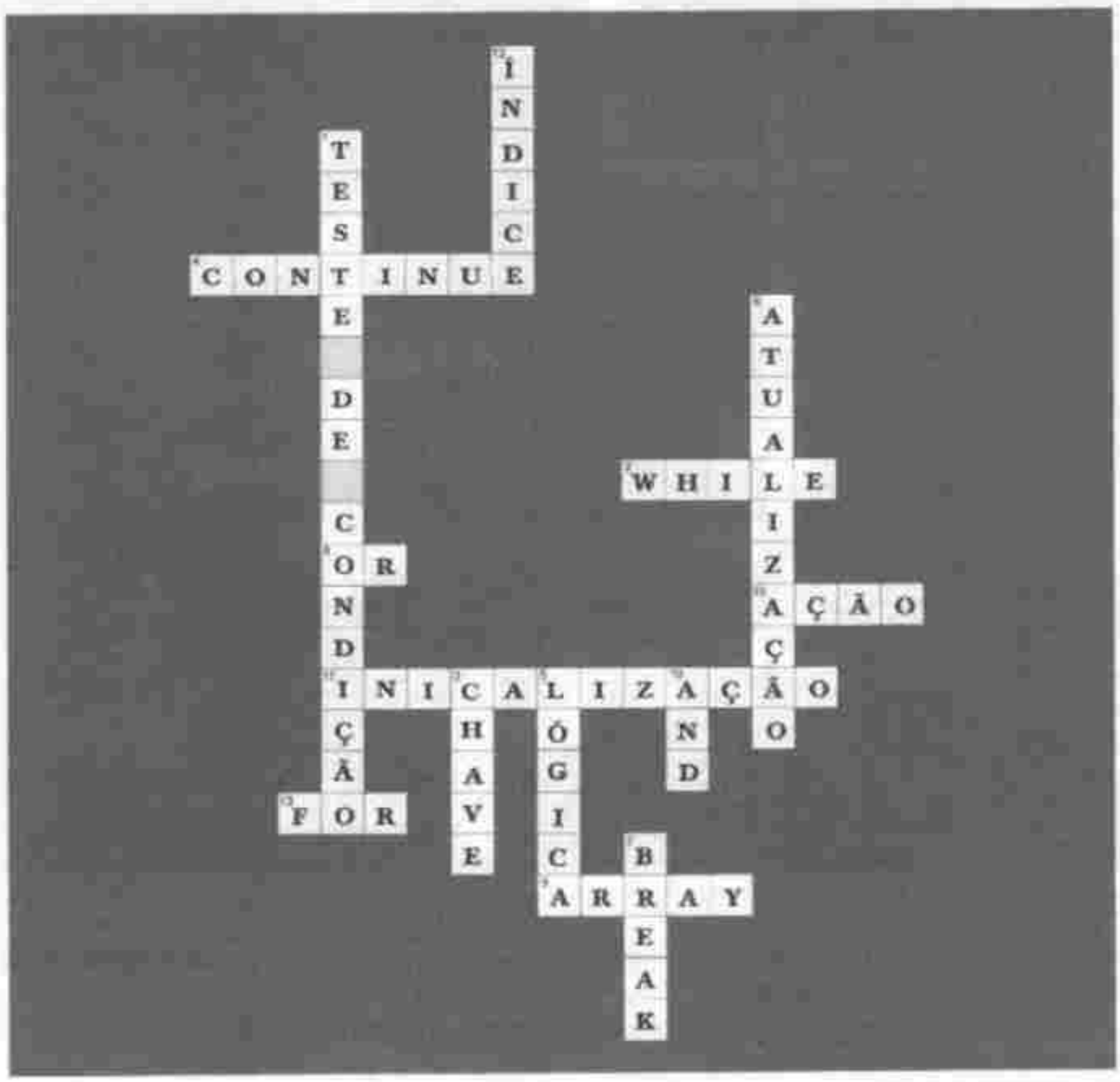
Horizontal

2. Esse tipo de loop continua a executar o código, enquanto um teste de condição for true.
4. Use essa instrução para saltar do ciclo atual de loop, mas continuar executando o loop.
8. Se a for true ou b for true, então a .. b são true; se não a .. b são false.
9. Um tipo de dados que permite armazenar vários fragmentos de dados em uma variável simples.
10. A parte de um loop que contém o código para executar repetidamente.
11. A parte de um loop que prepara o loop para começar.
13. Um tipo de loop que é perfeitamente apropriado para contagem.

Vertical

1. A parte de um loop que precisa ter um resultado booleano.
3. Uma... é usada para acessar um valor em um array.
5. Operadores de ..... booleana opera em valores booleanos e retorna um resultado booleano.
6. A parte de um loop que é responsável pela mudança do estado de quaisquer controles de loop.
7. Se você quiser terminar um loop imediatamente, use essa instrução.
10. Se a for true e b for true, então a ... b são true; se não a ... b são false.
12. Acessar um valor de array usando um número requer um_____.

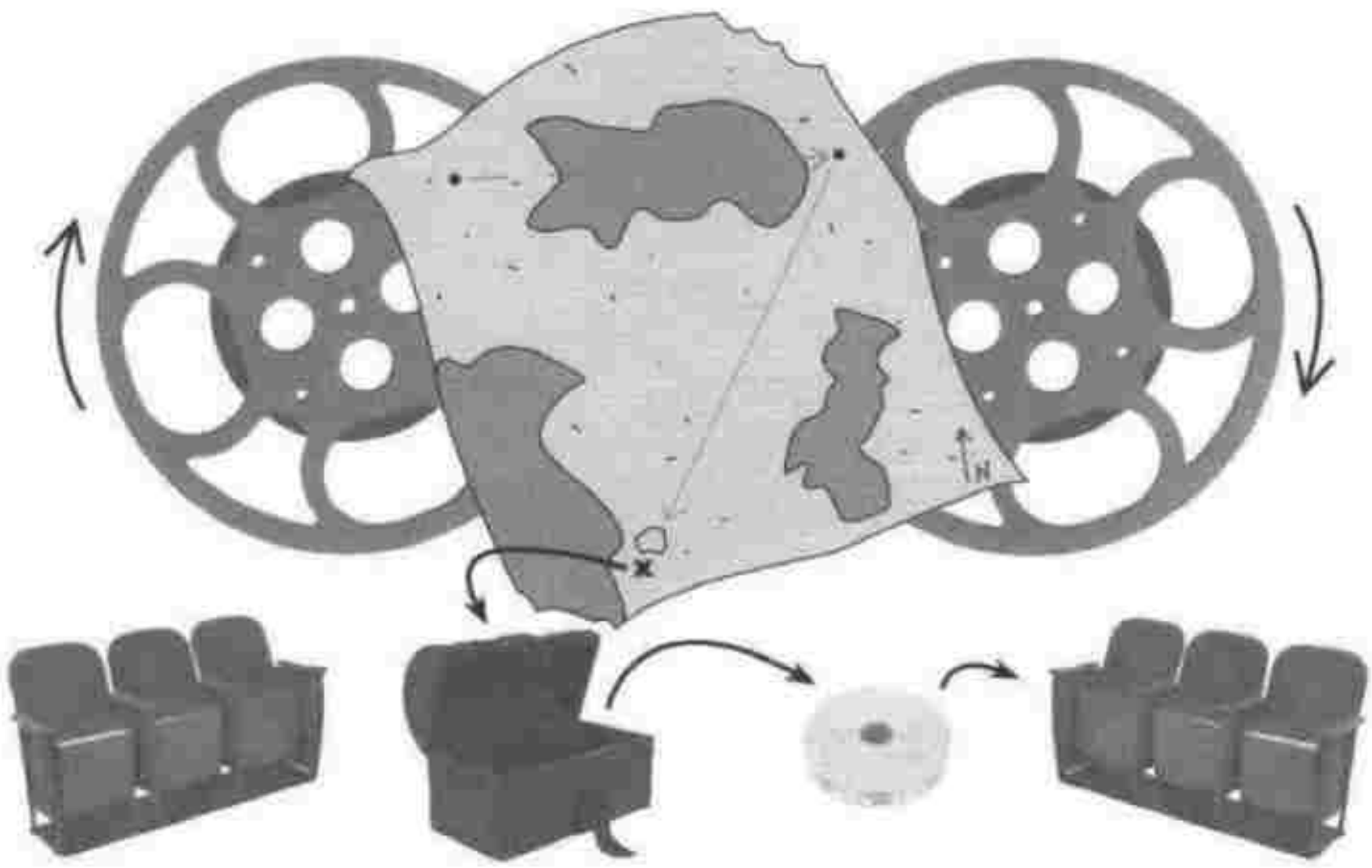
Solução da Cruzada do JavaScript



DOBRA DA PÁGINA

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

O que os loops e os filmes tem em comum?



Alguns filmes são conhecidos por terem enredos circulares difíceis de entender. Outros usam movimentos e muita ação para atrair o público. No final, um filme é apenas um filme.

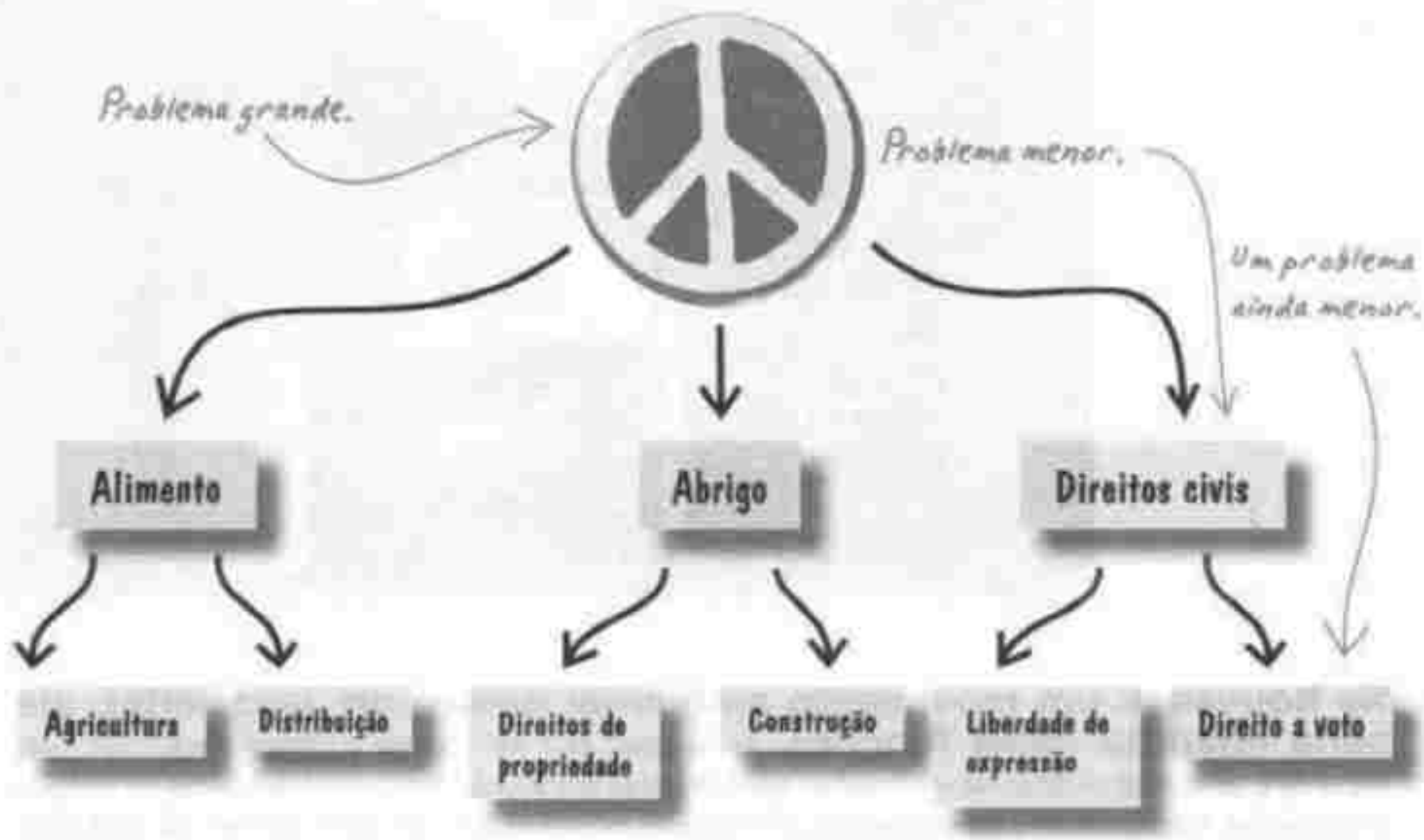
Reduza, Reutilize, Recicle

Se houvesse um movimento ambiental dentro do JavaScript, ele seria liderado pelas funções. As funções permitem tornar o código JavaScript mais eficiente e, sim, mais reutilizável. As funções são baseadas em tarefas, boas na organização de códigos e excelentes solucionadores de problemas. Parecem os requisitos para um bom reinício! Na realidade, todos, exceto os mais simples dos scritps, estão sujeitos ao benefício de uma reorganização funcional. Embora seja difícil avaliar o efeito do carbono da função comum, digamos apenas que elas fazem a parte delas tornando os scripts os mais ecologicamente amigáveis.

A mãe de todos os problemas

Quando refletimos friamente sobre isso, a criação de scripts para a web tem tudo a ver com solução de problemas. Não importa se é um grande problema, com idéias e planejamento suficientes, há sempre uma solução. Mas e em relação aos **problemas realmente enormes?**

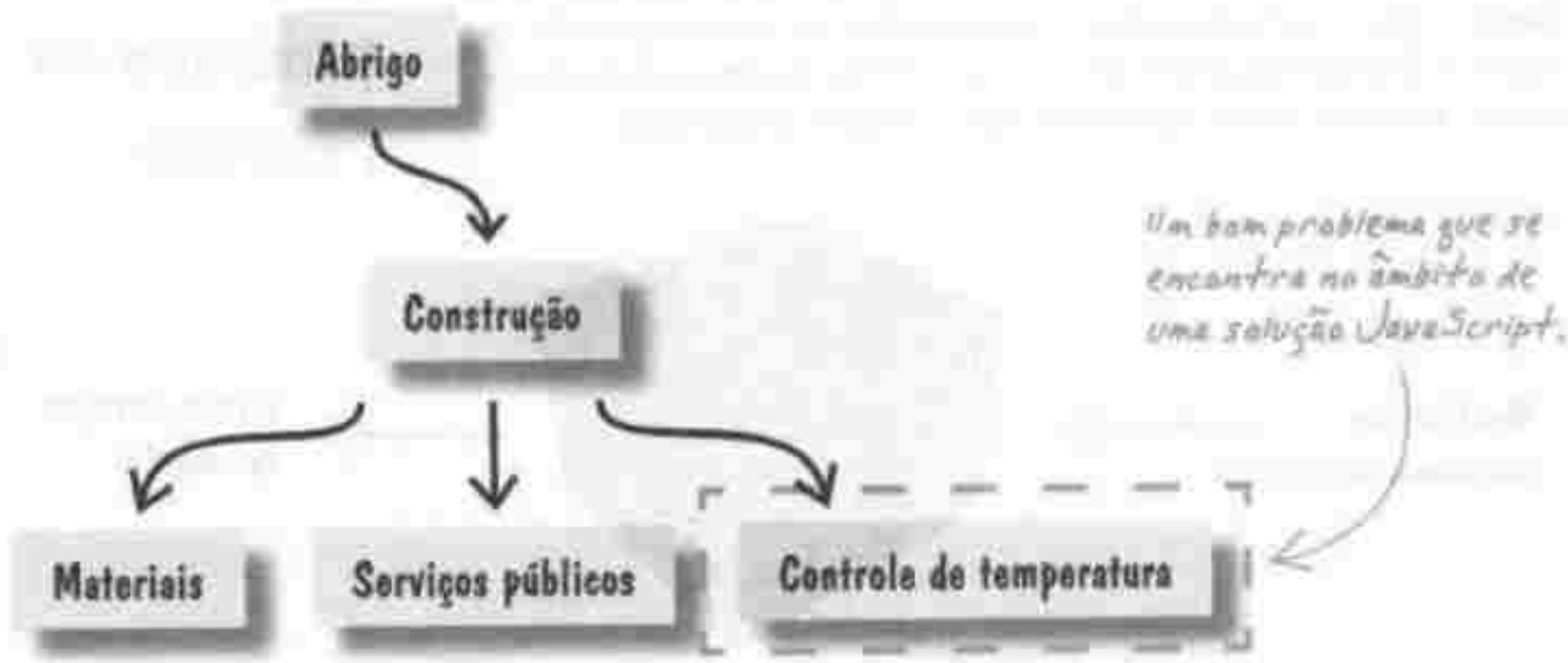
O truque para solucionar grandes problemas é decompô-lo em problemas menores e mais gerenciáveis. E se esses problemas ainda forem muito grandes, então decomponha-os mais uma vez.



E esse processo continua muitas vezes...

Solucione grandes problemas solucionando os pequenos

Continuando a decompor o problema da paz mundial em pequenos problemas, com tempo, você chega até um problema pequeno o suficiente para o JavaScript gerenciar.



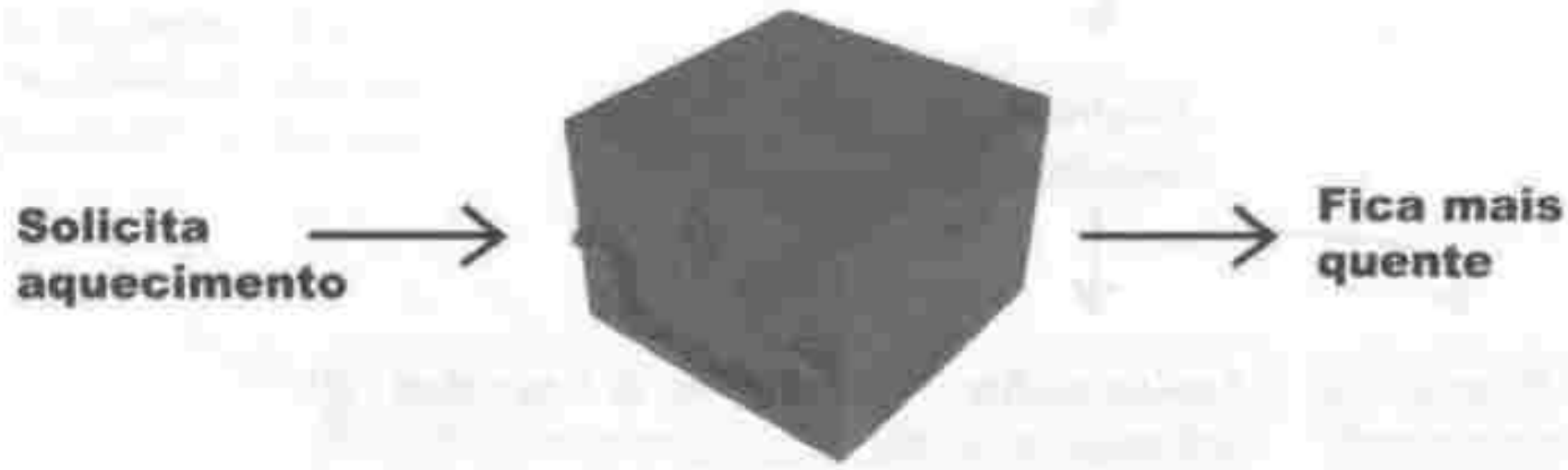
Pense em um equivalente JavaScript para o problema do controle de temperatura, que envolveria a criação de um script equivalente a um termostato, usado para controlar a temperatura de um meio ambiente. O termostato mais básico teria simplesmente um botão "aquecimento".

Observe como o termostato não revela nada sobre como o aquecimento é realizado. Você pressiona o botão Aquecimento e fica aquecido. Resolvido o problema do controle de temperatura!

Funções como solucionadores de problemas

O botão Aquecimento no termostato é o equivalente de uma **função** no JavaScript. A idéia é semelhante ao termostato do mundo real – alguém solicita aquecimento, a função fornece. Os detalhes do aquecimento são manipulados dentro da função, e não são importantes para o código chamar a função. Desta forma, você pode pensar em uma função como uma "caixa preta" – informações que podem proceder de dentro e de fora dela, mas o que acontece dentro é responsabilidade da caixa e, portanto, não é importante para o código fora da caixa.

Funções transformam grandes problemas em pequenos.



Converter o botão Aquecimento em código JavaScript envolve a chamada de uma função chamada `heat()`...

Solicita aquecimento → `heat();` → **Fica mais quente**

```
function heat() {
  // Produz algum aquecimento, de certa forma
  shovelCoal();
  lightFire();
  harnessSun();
}
```

Qualquer pessoa que queira aquecimento só precisa saber como chamar a função heat().

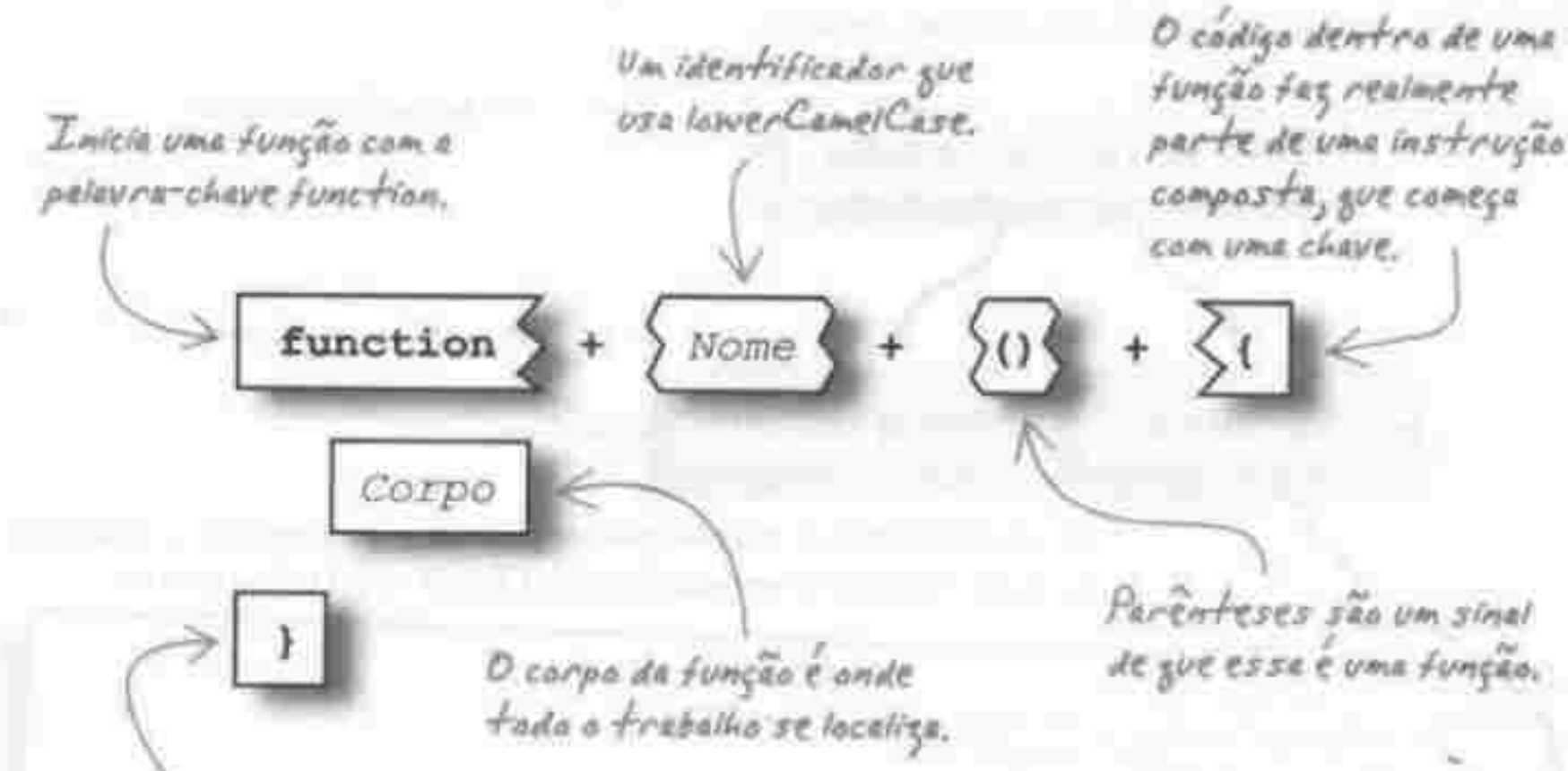
O aquecimento atual é produzido pela chamada dessas três outras funções.

A pessoa que escreve a função heat() é a única pessoa que precise se preocupar em como a função esquenta.

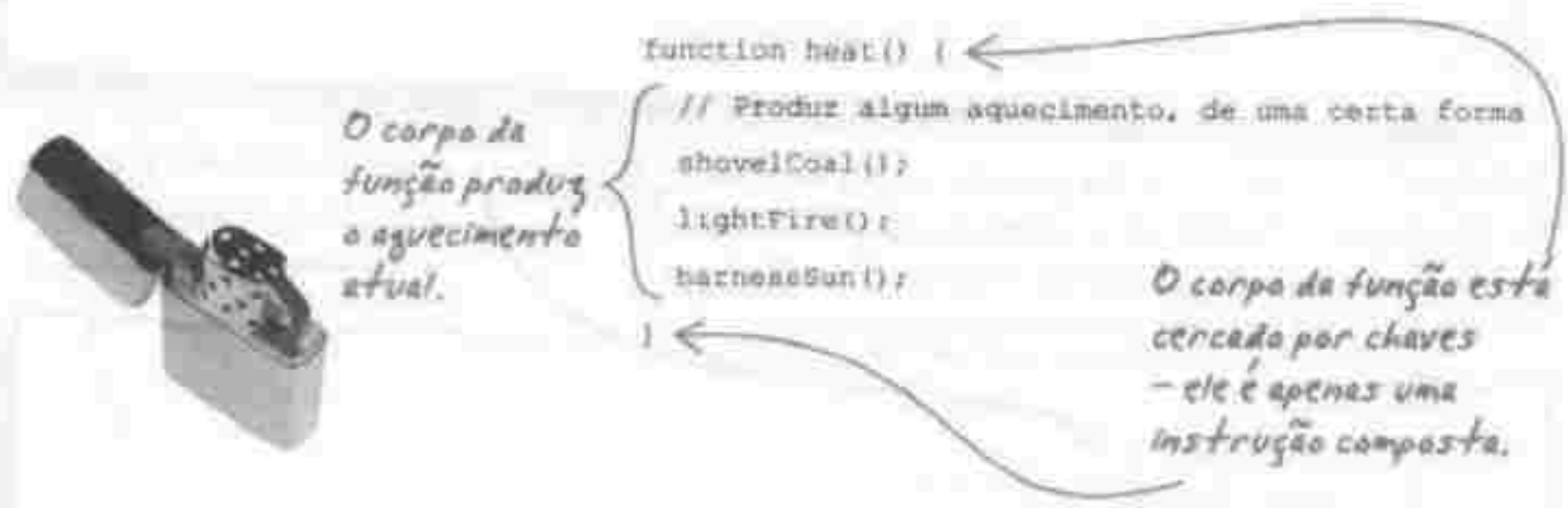
Não é tão importante **como** a função `heat()` produz o aquecimento. O que interessa é que ela serve como uma solução independente para o problema. Se precisar de algum aquecimento, chame a função `heat()`. Os detalhes da solução do problema são deixados para o mecanismo interno da função.

Elementos básicos de uma função

Ao escolher criar uma função, você se torna o solucionador do problema. Criar uma função exige que se use uma sintaxe coerente que ligue o nome da função ao código que a executará. A seguir, a sintaxe para a função mais básica do JavaScript:



Dar uma outra olhada no código em relação à função `heat()`, ajuda a colocar a sintaxe de função em perspectiva:



Poder do cérebro

Até agora, quando foi que você viu as funções solucionarem os problemas?

Uma função que você já conhecia

Você não precisa observar mais atentamente o script do localizador macho de poltronas Mandango (arquivos completos disponíveis em *http://www.headfirstlabs.com/books/bfjs/*) para encontrar um bom exemplo de função que soluciona problemas – neste caso, o problema de inicializar os dados de poltronas de cinema. Eis como o problema do Mandango foi decomposto:

O sub-problema da inicialização de poltronas era pequeno o suficiente para ser resolvido com uma função, a função `initSeats()`:

```
function initSeats() {
  // Inicializa a aparência de todas as poltronas
  for (var i = 0; i < seats.length; i++) {
    for (var j = 0; j < seats[i].length; j++) {
      if (seats[i][j]) {
        // Define a poltrona como disponível
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
      }
      else {
        // Define a poltrona como indisponível
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_unavail.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Unavailable seat";
      }
    }
  }
}
```

A função `initSeats()` faz parte da página da web do Mandango. A função não é chamada até que ela seja ligada ao manipulador de eventos `onload`. Isso faz com que a função execute quando a página carrega.

initSeats() não é o único solucionador de problemas no Mandango.

```
<body
onload="initSeats();">
    <div style="height:25px"></div>
    <div style="text-align:center">
      <img id="seat0" src="" alt="" />
      ...
      <img id="seat35" src="" alt="" />><br />
      <input type="button" id="findseats" value="Find Seats" onclick="findSeats();" />
    </div>
  </body>
</html>
```

Não existem Perguntas Idiotas

P: Como funciona mesmo a convenção de nomenclatura para funções?

R: lowerCamelCase é a convenção para nomenclatura de identificadores JavaScript onde a primeira palavra em um identificador é toda em letras minúsculas, mas as palavras adicionais são em letras maiúsculas e minúsculas, misturadas. Então, uma função que classifica um filme poderia ser chamada de `rateMovie()`, enquanto uma função que coloca para fora um cara que insiste em falar em seu telefone celular durante um filme poderia ser chamada de `removeInappropriateGuy()`.

P: As funções sempre se relacionam à transformação de grandes problemas em pequenos?

R: Não necessariamente. Existem situações onde as funções são úteis simplesmente como uma parte do trabalho de codificação. Em outras palavras, talvez seja o único problema a ser resolvido por várias funções trabalhando juntas. Neste caso, a justificativa da divisão do código em funções é ajudar a dividir o trabalho e dar a cada função seu próprio sentido. Mais ou menos como quando são dados cargos diferentes às pessoas, de forma que elas possam se concentrar em um tipo específico de tarefa. Tais funções podem ou não solucionar problemas específicos, mas elas definitivamente melhoram a estrutura dos scripts dividindo o trabalho.

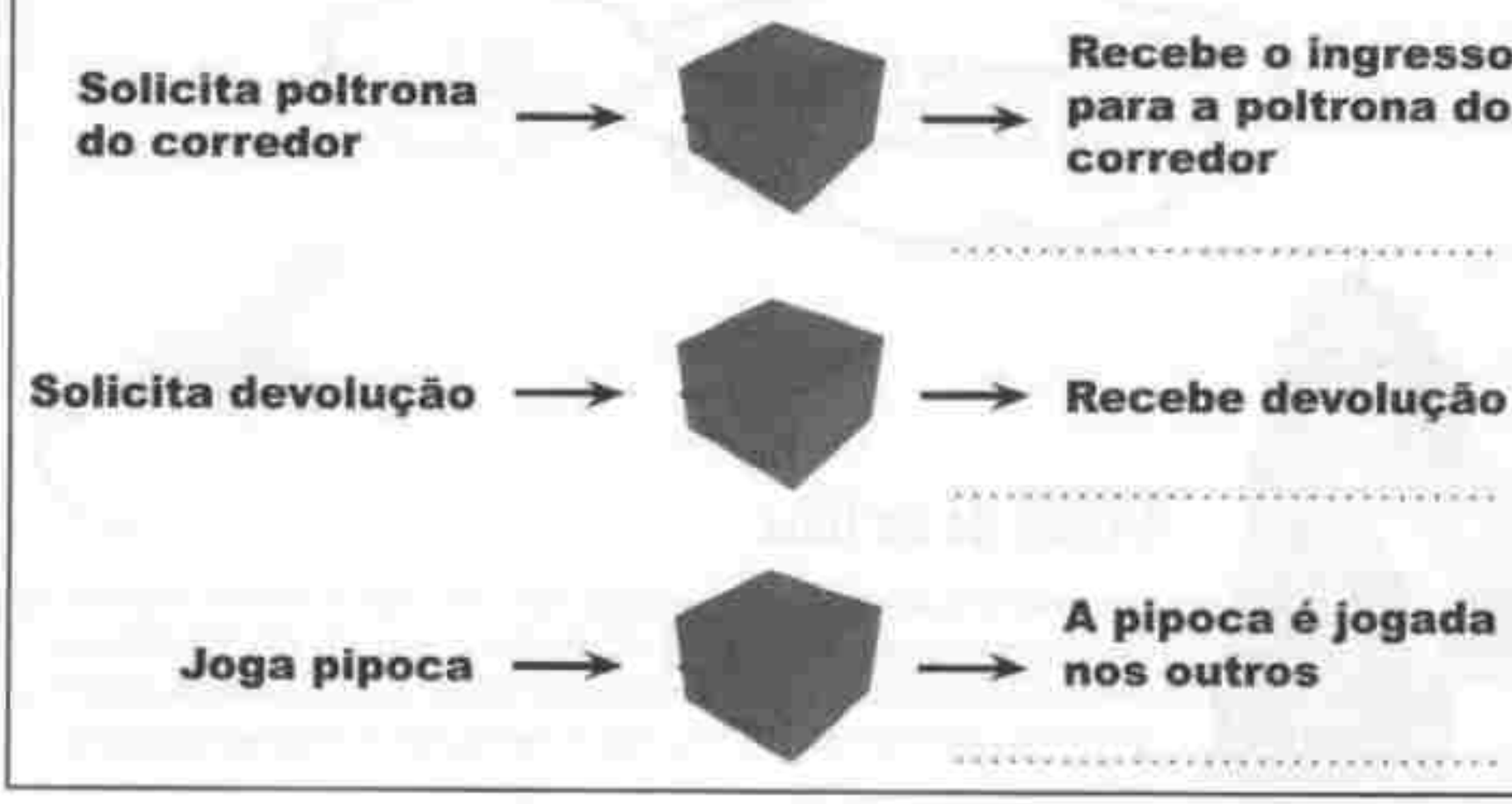
P: Como você sabe quando um bloco de código deve ser colocado em uma função?

R: Infelizmente, não há uma fórmula mágica para saber quando faz mais sentido colocar um código em uma função. Mas existem alguns sinais que você pode procurar como pistas. Um deles é quando você se pega reproduzindo um fragmento de código. A reprodução de código nem sempre é uma boa idéia, porque é preciso mantê-lo em mais de um lugar. Então, um código reproduzido é um grande alvo a ser colocado em uma função. Um outro sinal é uma situação onde um fragmento de código torna-se incontrolável, e você é capaz de decifrar várias partes lógicas para ele. Essa é uma boa hora de aplicar a idéia de "divisão de trabalho" ao código, e considerar sua decomposição em múltiplas funções.

P: Pensei que eu lembrasse das funções que aceitavam argumentos e, em seguida, passavam os dados de volta. Estou deixando escapar alguma coisa?

R: Não, absolutamente. Certamente, existem funções que aceitam e retornam dados. Na verdade, você está prestes a ver a função `beat()` se transformar em uma delas, se tiver paciência.

O nome de uma função é extremamente importante em transmitir imediatamente o que ela faz. Tente nomear essa funções, certificando-se de usar letras minúsculas e maiúsculas.



Solução do Exercício

O nome de uma função é extremamente importante em transmitir imediatamente o que ela faz. Tente nomear essa funções, certificando-se de usar letras minúsculas e maiúsculas.

Solicita poltrona do corredor → [ ] → **Recebe o ingresso para a poltrona do corredor**

`requestAisleSeat()`

Solicita devolução → [ ] → **Recebe devolução**

`getRefund()`

Joga pipoca → [ ] → **A pipoca é jogada nos outros**

`throwPopcorn()`

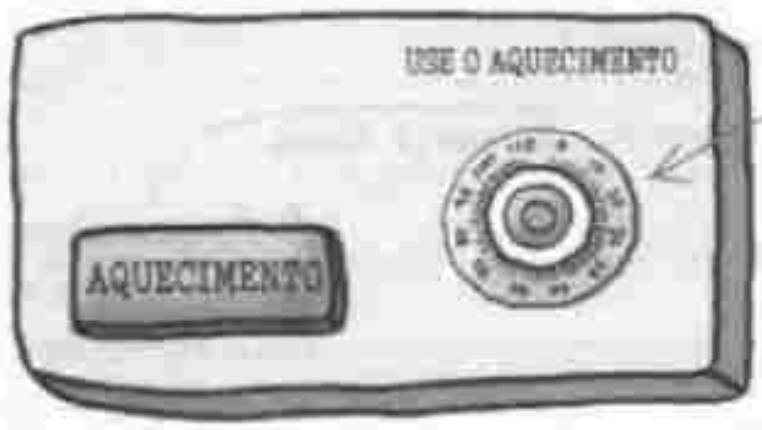
Existem muitos outros nomes de função que funcionariam aqui. Esses são apenas alguns bons exemplos de nomes resumidos em lower camel case.

Difícil de se lidar

Enquanto isso, a iniciativa de viabilizar a paz mundial através do controle de temperatura topou com uma dificuldade. Parece que o botão Aquecimento funciona muito bem, ou talvez seja apenas um problema da função `heat()` que precisa de mais dados. De qualquer forma, algo precisa ser corrigido.

Como criar um termostato melhor com mais dados

Então, de volta ao termostato... ele não sabe quando parar de aquecer, porque em nenhum momento definimos uma temperatura a ser atingida. Simplesmente, não há informação suficiente para solucionar o problema de maneira eficaz, o que significa que alguém pressiona o botão Aquecimento e se aquece... continuamente!



O botão Temperatura permite ao usuário definir uma temperatura a ser atingida.

Agora, o termostato aperfeiçoado aceita a temperatura a ser atingida como uma informação que ele pode usar para realizar o processo de "aquecimento".

A temperatura a ser atingida agora serve como uma informação dentro da função.

```
heat(targetTemp);
```

Exercício

Escreva o código de uma função `heat()` aperfeiçoada que aceita uma temperatura a ser atingida e a usa para gerar calor somente enquanto a temperatura atual for menor que a temperatura a ser atingida. Dica: chame a função hipotética `getTemp()` para obter a temperatura atual.

```
function heat(targetTemp){
.....................................
.....................................
.....................................
.....................................
.....................................
.....................................
.....................................
.....................................
.....................................
```

Acrescentamos uma linha para você iniciar.

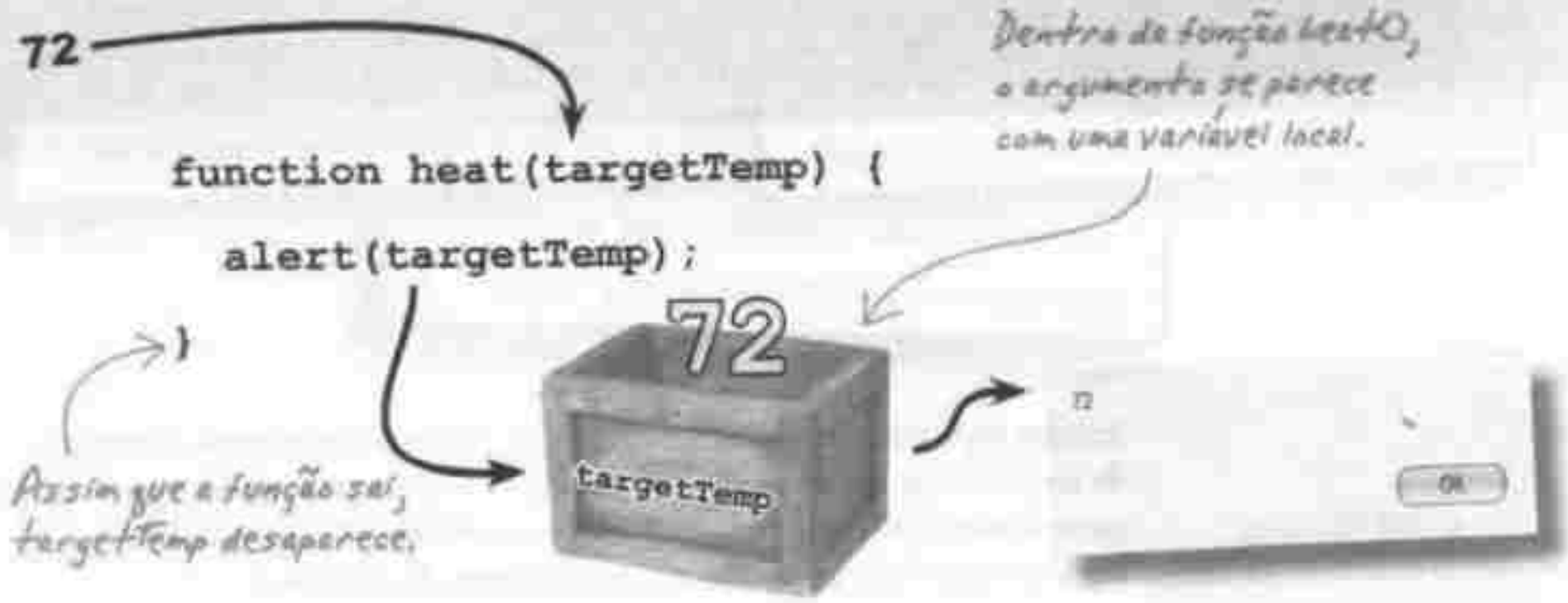
Argumentos de função como dados

Quando os dados são passados para uma função como um argumento, eles agem como uma variável local inicializada dentro da função. Como exemplo, aqui a função `heat()` está sendo fornecida com uma temperatura a ser atingida que é passada para a função como um argumento:

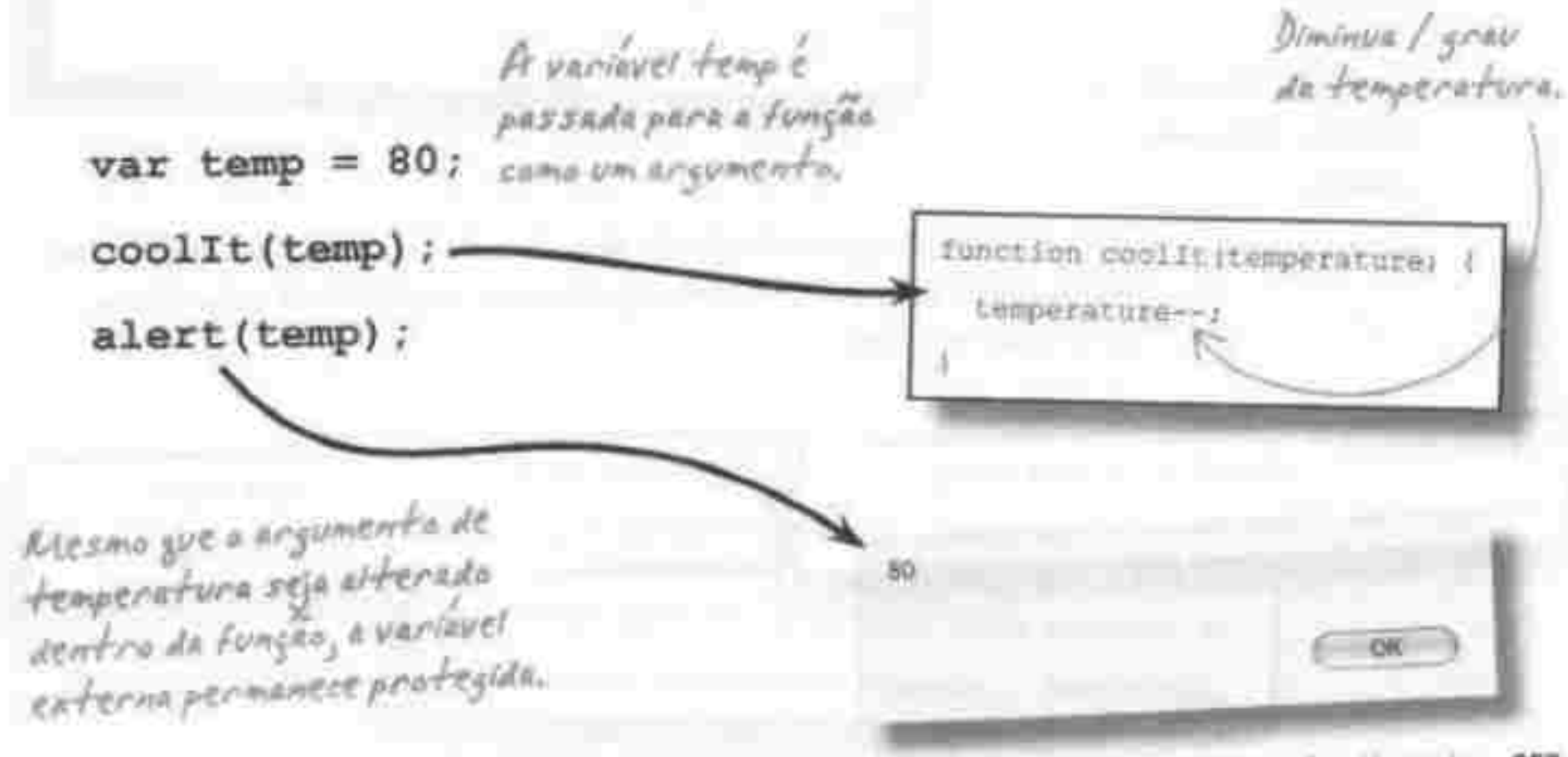
```
heat(72);
```

A temperatura a ser atingida é passada para a função como um número literal.

Dentro da função `heat()`, o argumento `targetTemp` está acessível como se fosse uma variável local inicializada como 72. Substituir o código normal da função `heat()` por um alerta revela o valor do argumento.



Embora os argumentos de função ajam de forma bem parecida com as variáveis locais dentro de uma função, alterar um valor de argumento **dentro** de uma função **não** afeta nada **fora** da função. Essa regra não se aplica a objetos que são passados como argumentos – vamos nos aprofundar em objetos nos Capítulos 9 e 10.



Funções eliminam código duplicado

Além de decompor os problemas, de forma que eles possam ser solucionados com mais facilidade, as funções servem como excelente forma de eliminar código duplicado, generalizando as tarefas. Uma **tarefa generalizada** pode ser usada para eliminar códigos semelhantes que aparecem em mais de um lugar. Mesmo que o código não seja idêntico, em muitos casos você pode generalizá-lo como código idêntico que pode ser colocado em uma função. Então, você chama a função em vez de duplicar o código semelhante.

O cálculo da porcentagem de desconto é desnecessariamente duplicado.

A seguir, três fragmentos de código diferentes que envolvem tarefas semelhantes que poderiam ser generalizados em uma tarefa única e **reutilizável**:

```
// Ingresso para matinê é 10% mais barato
matineeTicket = adultTicket * (1 - 0.10);
```

```
// Ingresso para adulto é 15% mais barato
seniorTicket = adultTicket * (1 - 0.15);
```

```
// Ingresso para crianças é 20% mais barato
childTicket = adultTicket * (1 - 0.20);
```

Tarefas específicas envolvem o cálculo de preços de três tipos diferentes de ingressos de cinema com desconto. Mas essas tarefas podem ser generalizadas em uma tarefa que envolve o cálculo de preço de um ingresso com base em qualquer porcentagem de desconto.

```
function discountPrice(price, percentage) {
  return (price * (1 - (percentage / 100)));
}
```

Funções também podem retornar dados.

Com uma função generalizada de desconto de ingresso sob controle, os outros três fragmentos de código podem ser reescritos de uma forma muito mais eficiente:

```
// Ingresso para matinê é 10% mais barato
matineeTicket = discountPrice(adultTicket, 10);
```

```
// Ingresso para adulto é 15% mais barato
seniorTicket = discountPrice(adultTicket, 15);
```

```
// Ingresso para crianças é 20% mais barato
childTicket = discountPrice(adultTicket, 20);
```

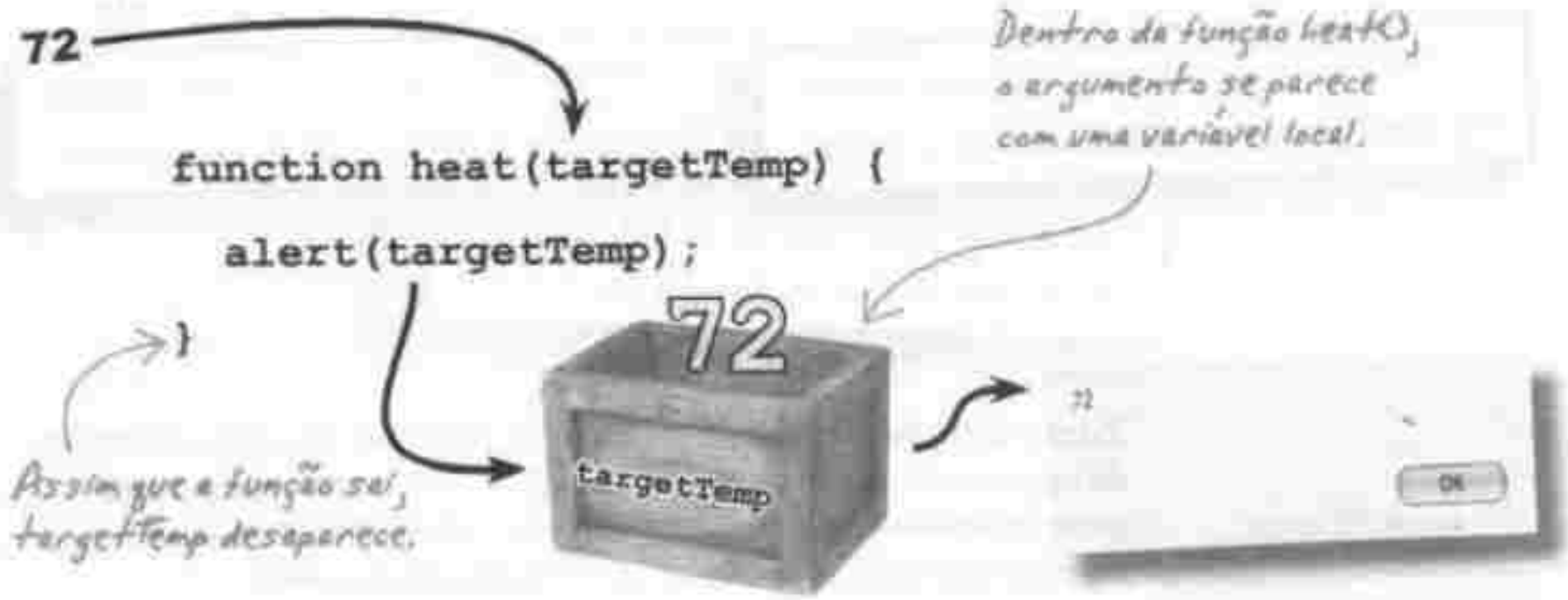
Argumentos de função como dados

Quando os dados são passados para uma função como um argumento, eles agem como uma variável local inicializada dentro da função. Como exemplo, aqui a função `heat()` está sendo fornecida com uma temperatura a ser atingida que é passada para a função como um argumento:

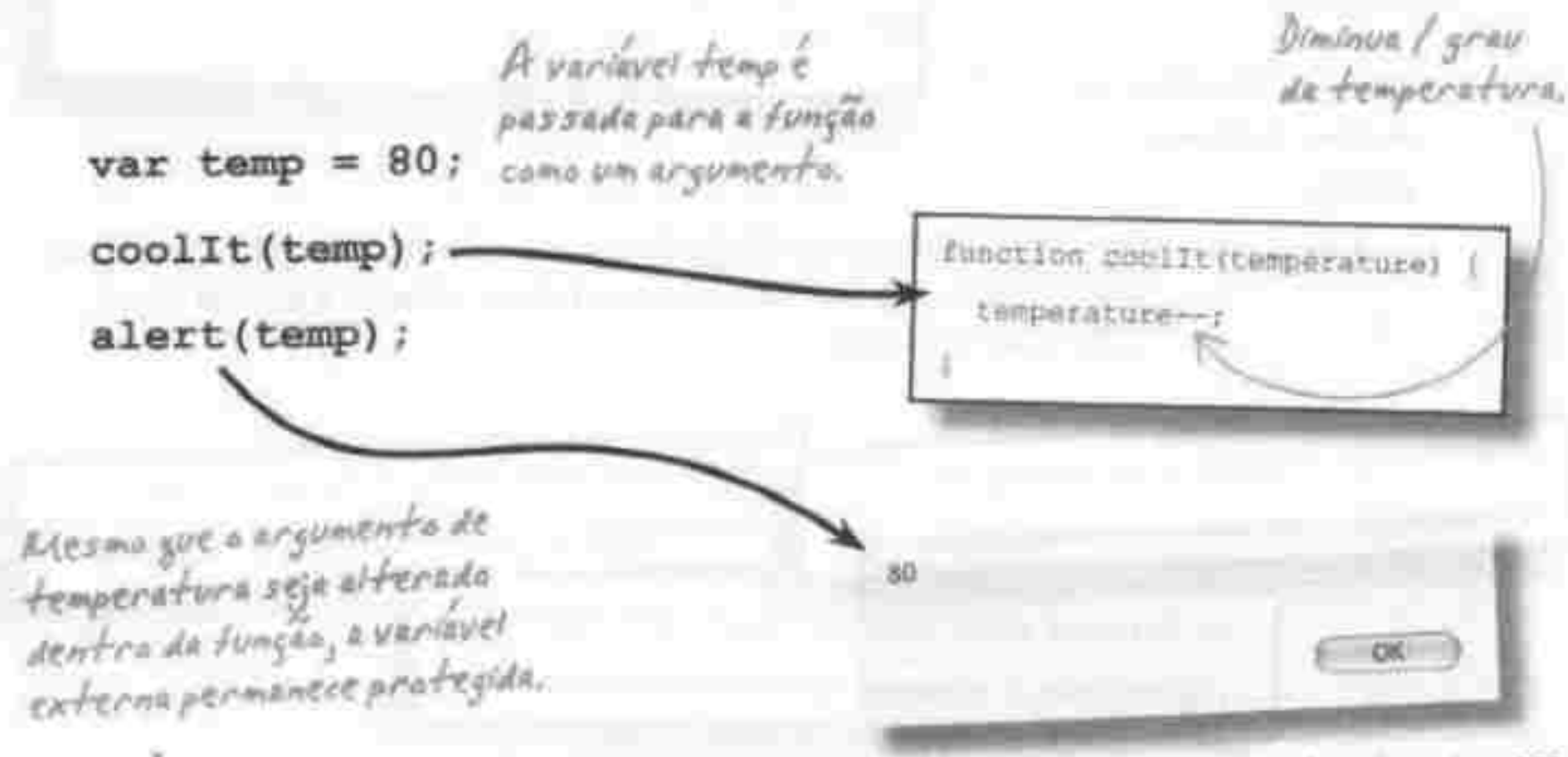
```
heat(72);
```

A temperatura a ser atingida é passada para a função como um número literal.

Dentro da função `heat()`, o argumento `targetTemp` está acessível como se fosse uma variável local inicializada como 72. Substituir o código normal da função `heat()` por um alerta revela o valor do argumento.



Embora os argumentos de função ajam de forma bem parecida com as variáveis locais dentro de uma função, alterar um valor de argumento **dentro** de uma função **não** afeta nada **fora** da função. Essa regra não se aplica a objetos que são passados como argumentos – vamos nos aprofundar em objetos nos Capítulos 9 e 10.



Funções eliminam código duplicado

Além de decompor os problemas, de forma que eles possam ser solucionados com mais facilidade, as funções servem como excelente forma de eliminar código duplicado, generalizando as tarefas. Uma **tarefa generalizada** pode ser usada para eliminar códigos semelhantes que aparecem em mais de um lugar. Mesmo que o código não seja idêntico, em muitos casos você pode generalizá-lo como código idêntico que pode ser colocado em uma função. Então, você chama a função em vez de duplicar o código semelhante.

O cálculo da porcentagem de desconto é desnecessariamente duplicado.

A seguir, três fragmentos de código diferentes que envolvem tarefas semelhantes que poderiam ser generalizados em uma tarefa única e **reutilizável**:

```
// Ingresso para matinê é 10% mais barato
matineeTicket = adultTicket * (1 - 0.10);
```

```
// Ingresso para adulto é 15% mais barato
seniorTicket = adultTicket * (1 - 0.15);
```

```
// Ingresso para crianças é 20% mais barato
childTicket = adultTicket * (1 - 0.20);
```

Tarefas específicas envolvem o cálculo de preços de três tipos diferentes de ingressos de cinema com desconto. Mas essas tarefas podem ser generalizadas em uma tarefa que envolve o cálculo de preço de um ingresso com base em qualquer porcentagem de desconto.

```
function discountPrice(price, percentage) {
  return (price * (1 - (percentage / 100)));
}
```

Funções também podem retornar dados.

Com uma função generalizada de desconto de ingresso sob controle, os outros três fragmentos de código podem ser reescritos de uma forma muito mais eficiente:

```
// Ingresso para matinê é 10% mais barato
matineeTicket = discountPrice(adultTicket, 10);
```

```
// Ingresso para adulto é 15% mais barato
seniorTicket = discountPrice(adultTicket, 15);
```

```
// Ingresso para crianças é 20% mais barato
childTicket = discountPrice(adultTicket, 20);
```

SEJA o Perito em Eficiência

A função findSeats() do localizador macho de poltronas Mandango encontra-se abaixo com todo o seu esplendor anterior. Usando o seu conhecimento recente de eficiência, circule o código semelhante que poderia ser reescrito como uma função generalizada e reutilizável.

```
function findSeats() {
  // Se as poltronas já estiverem marcadas, reinicializa todas as poltronas para removê-las
  if (selSeat >= 0) {
    selSeat = -1;
    initSeats();
  }

  // Pesquisa por todas as disponibilidades de poltronas
  var i = 0, finished = false;
  while (i < seats.length && !finished) {
    for (var j = 0; j < seats[i].length; j++) {
      // Verifica se a poltrona atual e as duas seguintes estão disponíveis
      if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
        // Define a marcação da poltrona e atualiza sua aparência
        selSeat = i * seats[i].length + j;
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
        document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
        document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

        // Avisa o usuário para aceitar a poltrona
        var accept = confirm("As poltronas de " + (j + 1) + " a " + (j + 3) +
          " na fileira " + (i + 1) + " estão disponíveis, aceita?");
        if (accept) {
          // O usuário aceitou a poltrona, então damos por encerrado (sai do loop interno)
          finished = true;
          break;
        }
        else {
          // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
          selSeat = -1;
          document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
        }
      }
    }

    // Aumenta o contador de loop externo
    i++;
  }
}
```

SEJA o Perito em Eficiencia Solução

A função findSeats() do localizador macho de poltronas Mandango encontra-se abaixo com todo o seu esplendor anterior. Usando o seu conhecimento recente de eficiencia, circule o código semelhante que poderia ser reescrito como uma função generalizada e reutilizável.

```
function findSeats() {
  // Se as poltronas já estiverem marcadas, reinicializa todas as poltronas para removê-las
  if (selSeat >= 0) {
    selSeat = -1;
    initSeats();
  }

  // Pesquisa por todas as disponibilidades de poltronas
  var i = 0, finished = false;
  while (i < seats.length && !finished) {
    for (var j = 0; j < seats[i].length; j++) {
      // Verifica se a poltrona atual e as duas seguintes estão disponíveis
      if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
        // Define a marcação da poltrona e atualiza sua aparência
        selSeat = i * seats[i].length + j;
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
        document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
        document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

        // Avisa o usuário para aceitar a poltrona
        var accept = confirm("As Poltronas de " + (j + 1) + " a " + (j + 3) +
          " na Fileira " + (i + 1) + " estão disponíveis, aceita?");
        if (accept) {
          // O usuário aceitou a poltrona, então damos por encerrado (sai do loop interno)
          finished = true;
          break;
        }
        else {
          // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
          selSeat = -1;
          document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
        }
      }
    }
    // Aumenta o contador de loop externo
    i++;
  }
}
```

Já que estes seis fragmentos de código executam a mesma tarefa geral, eles podem ser transformados em uma única função.

Código duplicado. Podemos extrair alguns atributos daqui...

A propriedade length ainda funciona para obter o número de itens em um sub-array.

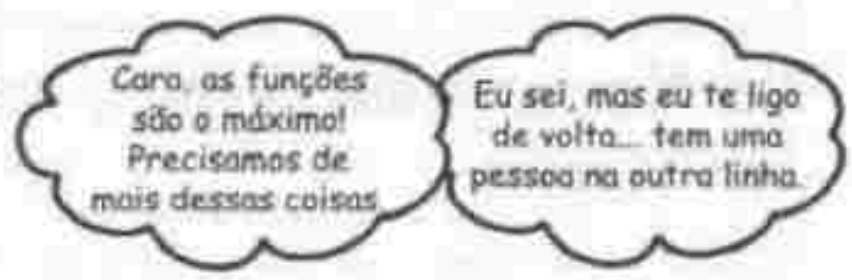
Como criar uma função de definição de poltronas

Agora que os caras do Mandango aprenderam essa coisa de eficiência, eles se entusiasmaram com a adição de uma função no Mandango que torna a marcação de poltrona mais eficiente (código disponível em *www.altabooks.com.br*). No entanto, para escrever a função `setSeat()`, eles realmente precisam descobrir quais argumentos são necessários. Você pode isolar os argumentos necessários examinando quais fragmentos de informação estão diferentes no código duplicado. Uma observação mais atenta nas partes duplicadas da função `findSeats()` revela esses argumentos:

Número da poltrona

Número da poltrona a ser definido. Ele não é um índice de array; apenas o número da poltrona, se você fosse começar a contar da esquerda para a direita e de cima para baixo, começando em 0.

Os atributos findSeats() são extraídos do código duplicado que será colocado na função.



Status

Status da poltrona: disponível, indisponível e marcado. Ele é usado para determinar qual imagem de poltrona exibir.

Descrição

Descrição do status das poltronas: "Poltrona disponível", "Poltrona indisponível" e "Sua poltrona". Ele é usado para definir o texto `alt` das imagens das poltronas.

Aponte seu Lápis

Escreva o código da função `setSeat()` do Mandango.

Aponte seu Lápis
Solução

Escreva o código da função `setSeat()` do Mandango.

Os três argumentos estão separados por vírgulas.

```
function setSeat(seatNum, status, description) {
  document.getElementById("seat" + seatNum).src = "seat_" + status + ".png";
  document.getElementById("seat" + seatNum).alt = description;
}
```

Os dados específicos usados no código original são substituídos agora pelos argumentos generalizados.

Um Mandango mais enxuto, mais organizado, com funções

Manifestar um código semelhante, duplicado, dentro da função `setSeat()` `function` simplifica o código da função `findSeats()` de forma considerável. Agora, existem seis chamadas para a função `setSeat()`, o que é uma melhoria significante em termos de reutilização de código.

O número, o status e a descrição da poltrona são passados para setSeat() em cada chamada de função.

```
function findSeats() {
  ...
  // Pesquisa por todas as disponibilidades de poltronas
  var i = 0, finished = false;
  while (i < seats.length && !finished) {
    for (var j = 0; j < seats[i].length; j++) {
      // Verifica se a poltrona atual e as duas seguintes estão disponíveis
      if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
        // Define a marcação da poltrona e atualiza sua aparência
        selSeat = i * seats[i].length + j;
        setSeat(i * seats[i].length + j, "marque", "sua poltrona");
        setSeat(i * seats[i].length + j + 1, "marque", "sua poltrona");
        setSeat(i * seats[i].length + j + 2, "marque", "sua poltrona");

        // Avisa o usuário para aceitar a poltrona
        var accept = confirm("As Poltronas de " + (j + 1) + " a " + (j + 3) +
          " na Fileira " + (i + 1) + " estão disponíveis, aceita?");
        if (accept) {
          // O usuário aceitou a poltrona, então damos por encerrado (sai do loop interno)
          finished = true;
          break;
        }
        else {
          // O usuário rejeitou a poltrona, então desfaz a marcação e continua procurando
          selSeat = -1;
          setSeat(i * seats[i].length + j, "avail", "Poltrona disponível");
          setSeat(i * seats[i].length + j + 1, "dispon", "Poltrona disponível");
          setSeat(i * seats[i].length + j + 2, "dispon", "Poltrona disponível");
        }
      }
    }

    // Aumenta o contador do loop externo
    i++;
  }
}
```

A nova função setSeat() é chamada seis vezes.

A função setSeat() torna o Mandango ainda melhor

Mas a função `setSeat()` não beneficia apenas `findSeats()`. Ela também ajuda a tornar a função `initSeats()` mais **eficiente**, porque essa função também possui um código de definição de poltronas semelhante.

```
function initSeats() {
  // Inicializa a aparência de todas as poltronas
  for (var i = 0; i < seats.length; i++) {
    for (var j = 0; j < seats[i].length; j++) {
      if (seats[i][j]) {
        // Define a poltrona como disponível
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
      }
      else {
        // Define a poltrona como indisponível
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_unavail.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Unavailable seat";
      }
    }
  }
}
```

Duas linhas de código complexas são transformadas em uma chamada de função relativamente óbvia.

```
function initSeats() {
  // Inicializa a aparência de todas as poltronas
  for (var i = 0; i < seats.length; i++) {
    for (var j = 0; j < seats[i].length; j++) {
      if (seats[i][j]) {
        // Define a poltrona como disponível
        setSeat(i * seats[i].length + j, "dispon", "Poltrona disponível");
      }
      else {
        // Define a poltrona como indisponível
        setSeat(i * seats[i].length + j, "indisp", "Poltrona
indisponível");
      }
    }
  }
}
```

Generalizando a tarefa de marcar uma poltrona, a função setSeat() funciona bem em um contexto completamente diferente.

Então, uma função razoavelmente simples que consiste em duas linhas de código, agora é usada oito vezes em todo o script do Mandango. Ela não só **simplifica** o código de script, como o torna mais fácil de manter, porque se precisar alterar a definição de uma poltrona, você só precisa alterar um único fragmento de código `setSeat()`, em contraste com os oito fragmentos separados. Nenhum codificador de JavaScript em seu juízo perfeito deseja alterar vários fragmentos de código, quando não há necessidade. Sustentabilidade... é bom.

Pontos de Bala

- As funções permitem transformar **problemas grandes em pequenos**, o que facilita a solução.
- As funções oferecem um mecanismo para separar as tarefas de script e, em seguida, completá-las com blocos de código **reutilizável**.
- As funções servem como uma ótima maneira de **eliminar código duplicado**, já que o código em uma função pode ser reutilizado quantas vezes você quiser.
- Os **argumentos** permitem passar dados para as funções como **informações** de uma determinada tarefa.

Não existem Perguntas Idiotas

P: Existe um limite para o número de argumentos que podem ser passados para uma função?

R: Não e sim. Não, não há um limite real em relação ao número de argumentos que podem ser passados para uma função, a menos que você inclua como fatores os limites da memória do computador. Se estiver passando muitos argumentos a ponto da memória se tornar um problema, você precisa parar e repensar sobre o que está fazendo, porque para causar um problema de memória é necessário muitos argumentos. O máximo da limitação tem a ver com um bom design, e que significa manter o número de argumentos em uma quantidade gerenciável perfeitamente ajustada para que as chamadas de função não sejam absurdamente complicadas. Normalmente, é uma boa idéia usar apenas um número reduzido de argumentos.

P: Eu aprendi que as funções transformam problemas grandes em pequenos, divide o trabalho de criação de scripts e elimina código duplicado. Qual é a opção certa?

R: Todas. As funções são boas em mais de um fator e, em muitos casos, as melhores funções cumprem várias metas de uma só vez. Não que seja impossível criar uma função que solucione um problema secundário, que execute uma tarefa de divisão de trabalho e que elimine código duplicado, tudo ao mesmo tempo. De fato, essas são três excelentes metas para se ter ao criar qualquer função. Mas se tiver que se concentrar em uma coisa, você vai querer errar no lado da divisão de trabalho, o que realmente significa dar a cada função um objetivo específico. Se cada função se sobressair em algo específico, seus scripts serão beneficiados consideravelmente.

P: Mais uma vez, para onde vão as funções no cabeçalho ou no corpo de uma página da web?

R: As funções devem aparecer dentro da tag `<script>` no topo da página, ou em um arquivo JavaScript externo que seja importado para o topo da página.

P: Se eu quiser mesmo que uma função altere o valor de um argumento, como faço?

R: Os argumentos de função não podem ser alterados diretamente, ou pelo menos as alterações não prosseguirão fora da função. Então, se quiser alterar um fragmento de dados que tenha sido passado como um argumento, você precisa retornar o valor alterado da função. Continue a leitura para descobrir como funciona o retorno de valores!

Inverno em janeiro: feedback com funções

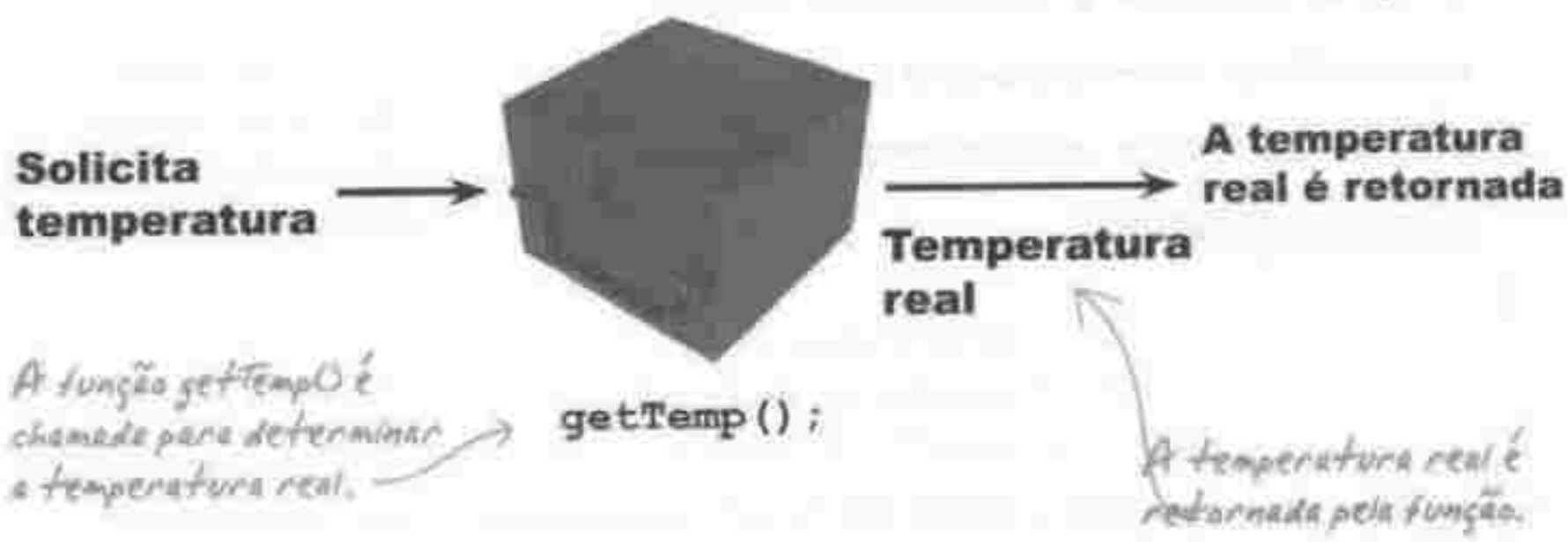
Embora o Mandango tenha progredido bastante graças às funções, elas não se saíram tão bem na frente de batalha da alteração de temperatura. Parece que o termostato JavaScript ainda não está funcionando adequadamente, resultando em alguns usuários com frio que agora estão ansiosos pelo antigo botão Aquecimento que nunca parou de esquentar.

A importância do feedback

O nosso termostato atual permite definir a temperatura graças aos argumentos de função, mas ele não informa a temperatura atual. A temperatura atual é importante, porque ela proporciona a base para determinar uma temperatura a ser atingida. Além disso, em muitos casos, termostatos diferentes informam temperaturas diferentes, mesmo dentro do mesmo espaço. Isso se resume à necessidade de feedback... você precisa conhecer a temperatura atual para definir a temperatura a ser atingida.

A exibição da temperatura atual permite que os usuários a conheçam, para que eles possam ajustar o aquecimento com mais precisão.

Agora, o termostato exibe periodicamente a temperatura atual como feedback para ajudar a determinar uma temperatura mais adequada.



Então, precisamos realmente de uma maneira de fazer com que as funções JavaScript retornem as informações de volta ao código que as chamou.

Poder do cérebro

Como você acha que uma função poderia ser persuadida a retornar dados?

Como retornar dados de funções

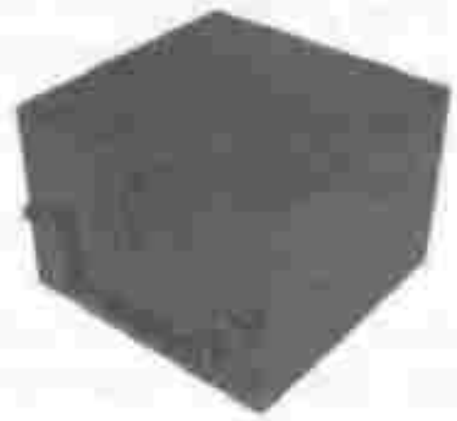
Retornar as informações de uma função envolve o uso da palavra-chave `return`, seguida dos dados a serem retornados. Esses dados são, então, retornados ao código que chamou a função.

A palavra-chave return indica que a função está retornando um valor.

O valor de retorno pode ser qualquer dados que você escolher.

Um valor de retorno permite retornar um fragmento de dados de uma função.

Uma instrução `return` pode ser colocada em qualquer lugar dentro de uma função; apenas saiba que a função sairá imediatamente ao encontrar `return`. Então, a instrução `return` não só retorna dados como também finaliza uma função. Como exemplo, a função `getTemp()` finaliza retornando a temperatura real, de acordo com a leitura do sensor.



```
function getTemp() {
  // Lê e converte a temperatura real
  var rawTemp = readSensor();
  var actualTemp = convertTemp(rawTemp);
  return actualTemp;
}
```

Os dados do sensor estão em um formato estranho e precisam ser convertidos em graus.

A temperatura real é retornada da função que usa a instrução return.

Se você se recordar atentamente, a função `getTemp()` já foi usada no código do termostato:

```
function heat(targetTemp) {
  while (getTemp() < targetTemp) {
    // Produz algum aquecimento, de uma certa forma
    ...
  }
}
```

A função getTemp() fornece o valor que é usado no teste de condição para o loop while do heat().

O valor de retorno da função `getTemp()` **substitui** a chamada de função `getTemp()` e se torna parte do teste de condição no loop `while`.

O valor de retorno de uma função substitui a chamada para a função.

Muitos valores de um retorno feliz

Já que a instrução `return` finaliza imediatamente uma função, você pode usá-la para controlar o fluxo de uma função, além de retornar dados. Não apenas isso, mas é muito comum as funções indicarem seu êxito usando valores de retorno. A função `heat()` apresenta a oportunidade de realizar ambos:

```
function heat(targetTemp) {
  if (getTemp() >= targetTemp)
    return false;
  while (getTemp() < targetTemp) {
    // Produz algum aquecimento, de uma certa forma
    ...
  }
  return true;
}
```

Lembra-se da variável actualTemp? Ela fornece o valor de retorno de getTemp().

Não há necessidade de produzir nenhum aquecimento, então retorna false e finaliza a função.

Esse código produz o aquecimento real que afeta a temperatura e, portanto, o valor de retorno de getTemp().

Terminado o aquecimento, retorna true para indicar êxito.

A função `heat()` demonstra como um valor de retorno booleano pode controlar o fluxo de uma função e também indicar êxito ou fracasso. Apenas por puro controle de fluxo, você pode usar a instrução return sem nenhum valor de retorno como um meio de sair do impasse de uma função. Por exemplo, eis uma outra versão da função `heat()` que não conta com o valor de retorno para indicar êxito ou fracasso.

```
function heat(targetTemp) {
  if (getTemp() >= targetTemp)
    return;
  while (getTemp() < targetTemp) {
    // Produz algum aquecimento, de certa forma
    ...
  }
}
```

A instrução return interrompe a função, já que não há necessidade de nenhum aquecimento.

A função sempre finaliza quando tem que finalizar, sem a ajuda de uma instrução return.

A instrução return pode ser usada por ela mesma para finalizar uma função.

Mestre dos padrões

Entrevista desta noite:

Os segredos de um artista que passa despercebido à função

Use a Cabeça: Ouvi dizer que você é escorregadio, capaz de sair de praticamente qualquer coisa.

Return: É verdade. Coloque-me em qualquer função e eu sairei dela num piscar de olhos. Não levarei nem mesmo um fragmento de dados comigo.

Use a Cabeça: Para onde você vai quando sai de uma função?

Return: Bem, lembre-se de que as funções são sempre chamadas por outro código, então retornar de uma função significa apenas retornar para o código que a chama. E neste caso, de retorno de dados, significa que os dados são retornados para o código que chamou a função.

Use a Cabeça: Como isso funciona?

Return: Isso ajuda se você pensar em uma chamada de função como uma expressão que possui um resultado. Se a função não retornar nenhum dado, o resultado da expressão é zero. Mas se a função retorna dados, e muitas delas retornam, então o resultado da expressão é esse fragmento de dados.

Use a Cabeça: Então, se a função é apenas uma expressão, isso significa que você pode atribuir o valor de retorno de uma função a uma variável?

Return: Não e sim. Não, a própria função não é uma expressão – é a chamada para a função que é a expressão. E sim, você pode e, em muitos casos, deve colocar uma chamada de função, fazendo com que o resultado seja atribuído a uma variável. É onde a expressão entra em cena – quando uma chamada de função é avaliada, ela é tratada como uma expressão onde o resultado é o valor de retorno da função.

Use a Cabeça: Entendo. Mas o que acontece à expressão quando você não retorna nada?

Return: Se você me usar sem nenhum retorno de dados, então a função retorna zero e a expressão fica vazia.

Use a Cabeça: Isso não é um problema?

Return: Não, não exatamente. Você precisa se lembrar de que as pessoas só se preocupam em fazer algo com os valores de retorno de função quando elas sabem que a função é capaz de retornar dados. Se uma função não pretende retornar nada, você não deve se preocupar em tentar fazer algo sem um valor de retorno.

Use a Cabeça: Te peguei! Então, de volta às suas habilidades de fuga, não é uma má idéia evitar que uma função finalize seu curso natural de execução?

Return: Não, e eis o motivo: só porque uma função possui a primeira e a última linha não significa que ela é criada para executar cada linha de código sempre do início ao fim. De fato, é perigoso até pensar em uma função com um início e um fim. A finalização "natural" de uma função poderia muito bem estar no meio do código, graças a um desenvolvedor esperto que me coloca no lugar certo.

Use a Cabeça: Não entendi. Você está dizendo que é normal para alguns códigos de função nunca serem chamados?

Return: Eu nunca digo nunca, mas direi que normalmente há mais de um caminho até uma função e eu, em muitos casos, ajudo a estabelecer esses caminhos. Se algo ocorrer, indicando que a função não deveria continuar executando, estou lá para proporcionar a ela uma saída precoce. Em outros cenários, uma função pode executar o tempo todo até sua última linha de código e sair sem nem mesmo me encontrar, ou ela pode finalizar comigo só para que eu possa retornar alguns dados.

Use a Cabeça: Ah, entendi. Você oferece opções, ambas em termos de retorno de dados e controle de fluxo de execução através de uma função.

Return: Ei, você está aprendendo!

Use a Cabeça: É, eu sou assim, bem rápido. Obrigado por dedicar seu tempo.

Return: Sem problema. Tenho que cair fora daqui!

Parece que o JavaScript foi colocado no meio de um escândalo sobre mudança de temperatura. O pessoal do Somos Contra o Aquecimento Rápido, ou SCAR, criou um script para promover suas mensagens sobre o aquecimento local. Mas o pessoal do Irritados, mas Cidadãos Responsáveis Cansados de Aumentar o Frio, ou ICRCAF, empenhado em abafar a mensagem do SCAR, sabotou o código de script. A sua tarefa é separar o código bom do ruim e desvendar a mensagem planejada do SCAR.

```
function showClimateMsg() {
  return;
  alert(constructMessage());
}

function constructClimateMsg() {
  var msg = "";

  msg += "Global "; // "Local ";

  if (getTemp() > 80)
    msg += "aquecimento ";
  else
    msg += "resfriamento ";

  if (true)
    msg += "não é";
  else
    msg += "é ";

  if (getTemp() <= 70)
    return msg + "uma farsa!";
  else
    return msg + "real!";

  return "Eu não acredito.";
}

function getTemp() {
  // Lê a temperatura real
  var actualTemp = readSensor();
  return 64;
}
```

Solução do Exercício

Parece que o JavaScript foi colocado no meio de um escândalo sobre mudança de temperatura. O pessoal do Somos Contra o Aquecimento Rápido, ou SCAR, criou um script para promover suas mensagens sobre o aquecimento local. Mas o pessoal do Irritados, mas Cidadãos Responsáveis Cansados de Aumentar o Frio, ou ICRCAF, empenhado em abafar a mensagem do SCAR, sabotou o código de script. A sua tarefa é separar o código bom do ruim e desvendar a mensagem planejada do SCAR.

```
function showClimateMsg() {
  ~~return;~~
  alert(constructMessage());
}

function constructClimateMsg() {
  var msg = "";

  msg += ~~"Global ";~~ // "Local ";

  if (getTemp() > 80)
    msg += "aquecimento ";
  else
    msg += "resfriamento ";

  ~~if (true)~~
    ~~msg += "não é";~~
  ~~else~~
    msg += "é ";

  if (getTemp() <= 70)
    return msg + "uma farsa!";
  else
    return msg + "real!";

  ~~return "Eu não acredito ";~~
}

function getTemp() {
  // Lê a temperatura real
  var actualTemp = readSensor();
  return ~~64;~~ actualTemp;
}
```

Esse retorno evita que o alerta apareça sempre.

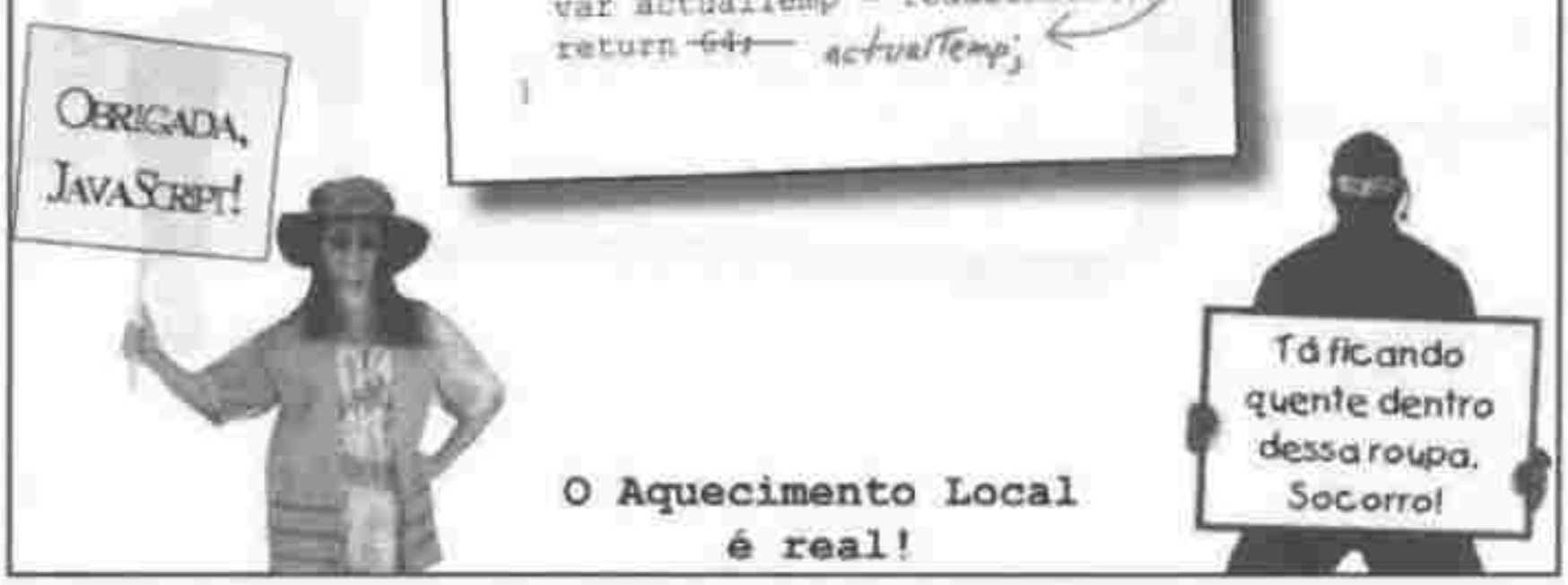
Este código é satisfatório, já que a temperatura real é controlar a mensagem.

A instrução if-else impede que a função chegue até este código, portanto não há razão para tê-la.

O texto planejado foi assinalado como comentado para mantê-lo fora da mensagem.

Uma instrução if que é permanentemente true não faz muito sentido.

Retorna a leitura da temperatura real do sensor.



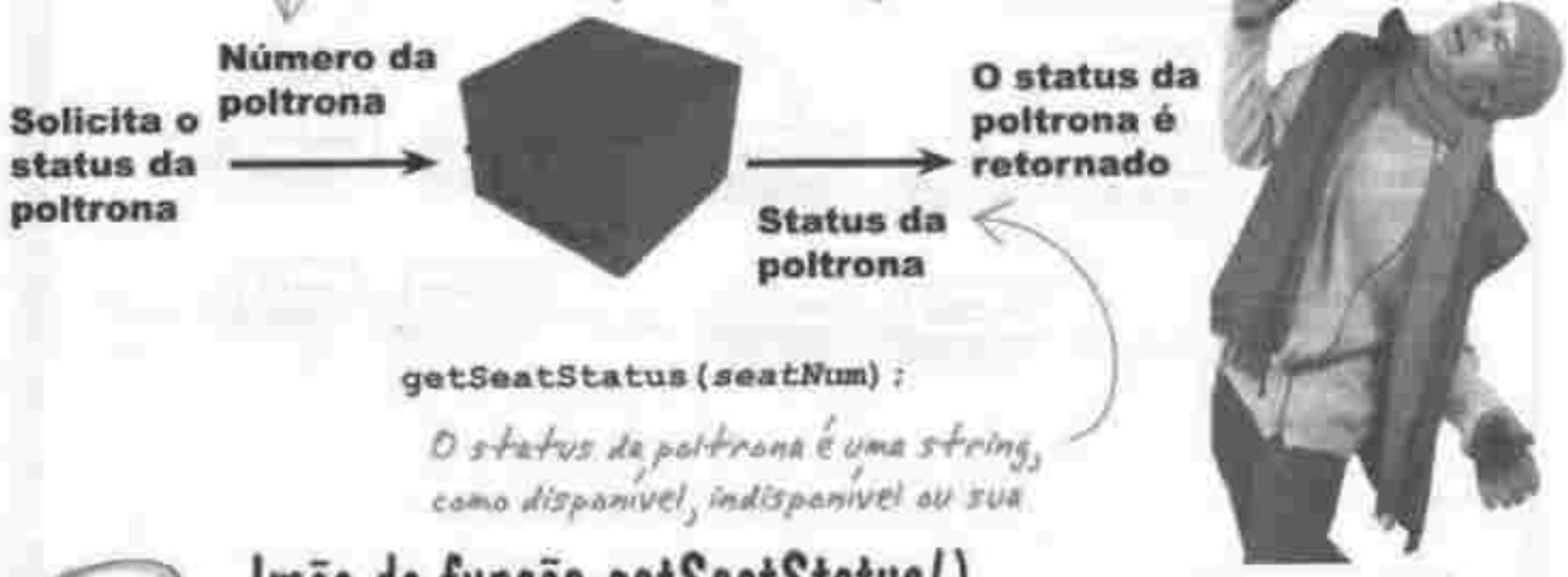
O Aquecimento Local é real!

Como obter o status de uma poltrona

De volta ao Mandango, Seth e Jason estão cansados de ouvir sobre mudança de temperatura e estão prontos para fazer mais algumas melhorias em seu código de script. Alguns usuários reportaram a dificuldade em distinguir as diversas cores de poltronas e gostariam de poder clicar e pesquisar qualquer disponibilidade de poltrona. Parece que o Mandango precisa de uma nova função.

Todo esse papo de temperatura está me cansando. Eu tenho que mandar consertar esse Mandango!

As poltronas são numeradas da esquerda para a direita, de cima para baixo, começando do zero.



`getSeatStatus(seatNum);`

O status da poltrona é uma string, como disponível, indisponível ou sua

Imãs da função getSeatStatus()

Está faltando um código importante na função `getSeatStatus()` do Mandango que ajuda a descobrir o status de uma determinada poltrona. A função verifica primeiro se a poltrona faz parte da seqüência de três poltronas marcadas. Se não fizer, ela procura a poltrona no array de poltronas para ver se ela está disponível ou indisponível. Use os imãs abaixo para terminar os fragmentos de código que faltam.

```
function getSeatStatus(seatNum) {

  if (.............!= -1 &&

    .............==.............[.............] == (.............+ 1) [.............== (.............+

2)) {

    return "sua";

  else if (.............Math.floor(............./.............[0].length)%.............%.............[0].
length) {

    return "disponível";

  else

    return "indisponível";

}
```

seats

selSeat

seatNum

Imãs de geladeira da função getSeatStatus() Solução

Está faltando um código importante na função `getSeatStatus()` do Mandango que ajuda a descobrir o status de uma determinada poltrona. A função verifica primeiro se a poltrona faz parte da seqüência de três poltronas marcadas. Se não fizer, ela procura a poltrona no array de poltronas para ver se ela está disponível ou indisponível. Use os imãs abaixo para terminar os fragmentos de código que faltam.

A variável global selSeat é -1 quando nenhuma poltrona está marcada, então verifique-a primeiro.

Estamos lidando com três poltronas em uma fileira, então você deve verificar essa poltrona e as duas seguintes.

```
function getSeatStatus(seatNum) {
  if (selSeat == -1 &&
      seatNum == selSeat &&
      seatNum == selSeat + 1 || seatNum == selSeat + 2) {
    return "sus";
  }
  else if (seats[Math.floor(seatNum / seats[0].length)][seatNum % seats[0].length] == 0) {
    return "disponivel";
  }
  else {
    return "indisponivel";
  }
}
```

Descobre a fileira de array da poltrona dividindo o número de poltronas na fileira e, em seguida, arredondando para um inteiro.

Descobre a coluna de array da poltrona pegando a diferença do número de poltronas dividido pelo número de poltronas na fileira.

Você poderia inserir 9 aqui no código, mas ele interromperia se você tivesse alterado o tamanho do array de poltronas.

Como exibir o status da poltrona

Obter o status da poltrona é útil, mas permitir que o usuário pesquise o status de qualquer poltrona exige um meio de mostrar esse status quando o usuário clicar em uma poltrona. A função `showSeatStatus()` fornece uma solução simples para este problema, delegando o trabalho pesado para a função `getSeatStatus()` que acabamos de escrever.

```
function showSeatStatus(seatNum) {
  alert("Essa poltrona está " + getSeatStatus(seatNum) + ".");
}
```

Passe o número da poltrona para getSeatStatus() para obter o status da poltrona.

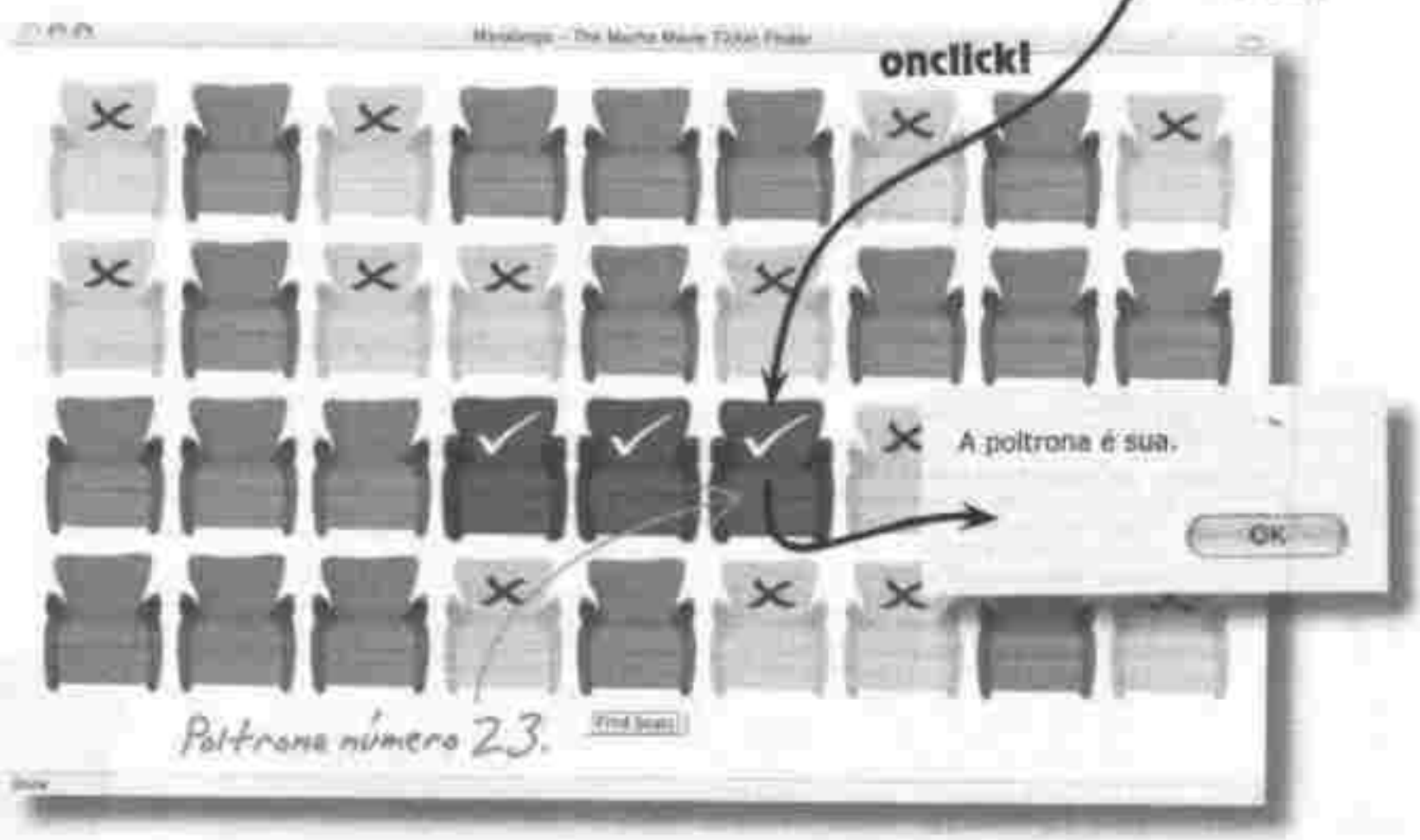
Concatene as strings para criar uma mensagem de status.

Você pode vincular a função a uma imagem

Conectar essa função a uma imagem de poltrona na página do Mandango permite que o usuário pesquise o status da poltrona clicando na imagem. Cada imagem precisa ter seu próprio evento `onclick` ligado à `showSeatStatus()`, assim:

A função showSeatStatus() é chamada quando o usuário clicar na imagem seat23.

```
<img id="seat23" src="" alt="" onclick="showSeatStatus(23);" />
```



Um clique é tudo o que precisa para ver o status de qualquer poltrona na sua caixa de alerta, que é útil para qualquer um que tenha problemas com as imagens das poltronas, e apenas quer clicar em uma poltrona individual para seu status.

Pontos de Bala

- A instrução `return` permite que as funções retornem dados de volta ao código que a chamou.
- Quando um fragmento de dados retorna de uma função, ele substitui o código que chamou a função.
- Uma função só pode retornar um único fragmento de dados.
- A instrução `return` pode ser usada sem quaisquer dados para simplesmente finalizar uma função mais cedo.

Código repetitivo nunca é bom

O script do Mandango está funcionando muito bem, mas os rapazes estão começando a se preocupar em manter o script a longo prazo. Em particular, Jason tem feito algumas pesquisas e aprendeu que as modernas aplicações da web muitas vezes se beneficiam da separação do código HTML, JavaScript e CSS.

```
<html>
  <head>
    <title>Mandango - O localizador macho de ingressos de cinema</title>
    <script type="text/javascript">
      ...
      function initSeats() {
        ...
      }
      function getSeatStatus(seatNum) {
        ...
      }
      function showSeatStatus(seatNum) {
        alert("Essa poltrona está " + getSeatStatus(seatNum) + ".");
      }
      function setSeat(seatNum, status, description) {
        document.getElementById("seat" + seatNum).src = "seat_" + status + ".png";
        document.getElementById("seat" + seatNum).alt = description;
      }
      function findSeats() {
        ...
      }
    </script>
  </head>
  <body onload="initSeats();">
    <div style="margin-top:25px; text-align:center">
      <img id="seat0" src="" alt="" onclick="showSeatStatus(0);" />
      <img id="seat1" src="" alt="" onclick="showSeatStatus(1);" />
      <img id="seat2" src="" alt="" onclick="showSeatStatus(2);" />
      <img id="seat3" src="" alt="" onclick="showSeatStatus(3);" />
      <img id="seat4" src="" alt="" onclick="showSeatStatus(4);" />
      <img id="seat5" src="" alt="" onclick="showSeatStatus(5);" />
      <img id="seat6" src="" alt="" onclick="showSeatStatus(6);" />
      <img id="seat7" src="" alt="" onclick="showSeatStatus(7);" />
      <img id="seat8" src="" alt="" onclick="showSeatStatus(8);" /><br />
      <img id="seat9" src="" alt="" onclick="showSeatStatus(9);" />
      <img id="seat10" src="" alt="" onclick="showSeatStatus(10);" />
      <img id="seat11" src="" alt="" onclick="showSeatStatus(11);" />
      <img id="seat12" src="" alt="" onclick="showSeatStatus(12);" />
      <img id="seat13" src="" alt="" onclick="showSeatStatus(13);" />
      <img id="seat14" src="" alt="" onclick="showSeatStatus(14);" />
      <img id="seat15" src="" alt="" onclick="showSeatStatus(15);" />
      <img id="seat16" src="" alt="" onclick="showSeatStatus(16);" />
      <img id="seat17" src="" alt="" onclick="showSeatStatus(17);" /><br />
      <img id="seat18" src="" alt="" onclick="showSeatStatus(18);" />
      <img id="seat19" src="" alt="" onclick="showSeatStatus(19);" />
      <img id="seat20" src="" alt="" onclick="showSeatStatus(20);" />
      <img id="seat21" src="" alt="" onclick="showSeatStatus(21);" />
      <img id="seat22" src="" alt="" onclick="showSeatStatus(22);" />
      <img id="seat23" src="" alt="" onclick="showSeatStatus(23);" />
      <img id="seat24" src="" alt="" onclick="showSeatStatus(24);" />
      <img id="seat25" src="" alt="" onclick="showSeatStatus(25);" />
      <img id="seat26" src="" alt="" onclick="showSeatStatus(26);" /><br />
      <img id="seat27" src="" alt="" onclick="showSeatStatus(27);" />
      <img id="seat28" src="" alt="" onclick="showSeatStatus(28);" />
      <img id="seat29" src="" alt="" onclick="showSeatStatus(29);" />
      <img id="seat30" src="" alt="" onclick="showSeatStatus(30);" />
      <img id="seat31" src="" alt="" onclick="showSeatStatus(31);" />
      <img id="seat32" src="" alt="" onclick="showSeatStatus(32);" />
      <img id="seat33" src="" alt="" onclick="showSeatStatus(33);" />
      <img id="seat34" src="" alt="" onclick="showSeatStatus(34);" />
      <img id="seat35" src="" alt="" onclick="showSeatStatus(35);" /><br />
      <input type="button" id="findseats" value="Find Seats" onclick="findSeats();" />
    </div>
  </body>
</html>
```

O código JavaScript e HTML misturados podem ser separados dos atributos HTML do manipulador de eventos.

O código JavaScript e HTML estão misturados em toda a página da web do Mandango.

Como separar a funcionalidade do conteúdo

Então, qual é o problema de misturar código? Obviamente ele funciona, certo? O problema tem mais a ver com a visualização de suas páginas da web movida a JavaScript não como páginas, mas como **aplicações**. E como qualquer boa aplicação, as aplicações JavaScript exigem um planejamento e design cuidadosos para o sucesso a longo prazo. Mais apropriadas, as aplicações são menos infestadas com bugs e mais fáceis de se manter quando há uma separação de conteúdo, apresentação e funcionalidade. Mandango apresenta uma fusão obscura de todas as três.

Separar conteúdo, apresentação e funcionalidade transforma um problema grande em pequenos problemas.

```
<html>
  ...
</html>
```

Conteúdo

Este é o código HTML na página, que oferece estrutura para a página se harmonizar fisicamente, bem como abrigar dados.

```
<style>
  ...
</style>
```

Apresentação

Esta é a parte da página em CSS, que melhora o aspecto do conteúdo e determina sua aparência, como fontes, cores e até o layout.

```
<script>
  ...
</script>
```

Funcionalidade

Este é o código JavaScript que aciona a página e a torna interativa. Você pode pensar nessa parte da página como a parte que executa.

Pense sobre a questão da separação do código dessa forma. Vamos dizer que Seth e Jason encontraram um script de gerenciamento de poltronas de cinema realmente ágil que eles gostariam de usar no lugar do código. Eles precisariam reestruturar o Mandango, fazendo com que ele usasse o novo código de script, mas eles teriam que correr o risco de estragar a estrutura da página, porque o código JavaScript está intimamente ligado ao código HTML. Seria muito melhor se o código HTML fosse isolado e a conexão JavaScript-para-HTML ocorresse puramente em JavaScript.

Separar a funcionalidade do conteúdo torna as aplicações da web mais fáceis de criar e manter.

Poder do cérebro

Como você empreenderia o uso de funções para separar a funcionalidade do conteúdo no Mandango?

Funções são apenas dados

Para separar efetivamente o código, você precisará entender como as funções são conectadas aos eventos; até agora, fizemos isso usando os atributos HTML. Existe uma outra maneira que muitas pessoas consideram de alto nível: misturar os códigos JavaScript e HTML. Essa é outra maneira de conectar os manipuladores de evento, o que requer uma visão diferente das funções.

Por incrível que pareça, realmente, as funções são apenas variáveis. É verdade. A distorção é que o corpo da função é o valor, enquanto o nome da função é o nome da variável. Eis a forma com que está acostumado a visualizar as funções:

A função é criada como sempre.

```
function showSeatStatus(seatNum) {
  alert("Essa poltrona está " + getSeatStatus(seatNum) + ".");
}
```

O código funciona bem, mas essa é a mesma função criada de uma maneira diferente.

O nome da função é o nome da variável.

```
var showSeatStatus = function(seatNum) {
  alert("Essa poltrona está " + getSeatStatus(seatNum) + ".");
};
```

O corpo da função é o valor da variável, também conhecido como literal de função, quando expressado dessa forma.

Esse código mostra como uma função pode ser criada usando-se a mesma sintaxe como uma variável, e até mesmo consistir nos mesmos fragmentos e partes: um identificador único (nome de função) e um valor (corpo de função). Quando um corpo de função aparece sozinho sem um nome, ele é conhecido como um **literal de função**.

O que torna essa revelação sobre funções tão interessante é que ela mostra que as funções podem ser manipuladas como as variáveis. Por exemplo, o que você acha que o código a seguir faz?

```
var myShowSeatStatus = showSeatStatus;
```

Atribui a função showSeatStatus() à variável myShowSeatStatus.

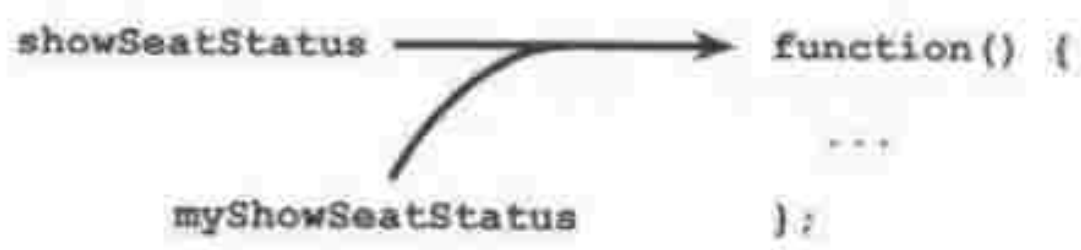
Como chamar ou referir-se às suas funções

Ao atribuir o nome de uma função a uma outra variável, você está dando acesso ao corpo da função a essa variável. Em outras palavras, você pode escrever o código para chamar a mesma função, dessa forma:

```
alert(myShowSeatStatus(23));
```

Chama a mesma função através da variável myShowSeatStatus.

O resultado final da chamada de `myShowSeatStatus()` é o mesmo que o da chamada de `showSeatStatus()`, porque ambas as funções, em última análise, **referem-se** ao mesmo código. Por esta razão, um nome de função também é conhecido como uma **referência de função**.



Uma função é na verdade apenas uma variável cujo valor é uma referencia ao corpo da função.

A diferença entre **referir-se** a uma função e **chamar** uma função tem a ver com a colocação de parênteses () após o nome da função. Referências de função aparecem de forma independente, enquanto as chamadas de função são sempre seguidas de parênteses e, em muitos casos, por argumentos de função.

Executa a função myShowSeatStatus(), que é o mesmo que showSeatStatus().

```
var myShowSeatStatus = showSeatStatus;
myShowSeatStatus(23);
```

Atribui uma referência de função a myShowSeatStatus.

Exercício

Analise o código a seguir e escreva o número que aparece na caixa de alerta.

```
function doThis(num) {
  return num++;
}

function doThat(num) {
  return num--;
}

var x = doThis(11);
var y = doThat;
var z = doThat(x);
x = y(z);
y = x;
alert(doThat(z - y));
```

Solução do Exercício

Analise o código a seguir e escreva o número que aparece na caixa de alerta.

```
function doThis(num) {
  return num++;
}

function doThat(num) {
 return num--;
}

var x = doThis(11);
var y = doThat;
var z = doThat(x);
x = y(z);
y = x;
alert(doThat(z - y));
```

x = 12
y = doThat
z = doThat(12) = 11
x = doThat(11) = 10
y = 10
alert(doThat(11 - 10))

Não existem Perguntas Idiotas

P: Separar o conteúdo é tão importante assim?

R: Sim e não. Para aplicações simples não é necessariamente errado misturar códigos HTML, CSS e JavaScript. Os benefícios da separação do código se tornam significantes em aplicações mais complexas que envolvem muitos códigos. É muito mais difícil compreender o quadro global em grandes aplicações, o que significa que é fácil se meter em apuros ao fazer alterações, especialmente quando tipos diferentes de código estão misturados. Separar o código de forma limpa faz você se sentir mais seguro para fazer alterações funcionais sem violar alguma coisa na estrutura ou na aparência da página. Isso permite também que pessoas de diferentes áreas de especialização trabalhem no mesmo projeto.

Por exemplo, os web designers podem trabalhar na estrutura e na apresentação de uma aplicação sem medo de criar erros no código funcional do JavaScript, que talvez eles não entendam.

P: Se uma função é apenas dados, como posso distinguir uma função de uma variável normal?

R: A diferença entre uma função e uma variável "normal" caracteriza-se pelo o que você faz com os dados. Os dados (código) associados às funções são capazes de serem executados. Você indica o que deseja para executar uma função colocando parênteses após o nome da função, inclusive os argumentos se a função exigi-los.

P: De que adianta as referências de função?

R: Ao contrário da variável normal, que armazena seus dados como um valor em uma área da memória, as funções armazenam uma referência ao seu código. Então, o valor de uma variável de função não é o código propriamente dito, mas uma referência à localização na memória onde o código está armazenado. Mais ou menos como o seu endereço de correspondência é uma referência à sua casa, não da casa propriamente dita.

As funções usam referências em vez de valores, porque é mais eficiente do que armazenar várias cópias do código de função. Então, quando você atribui uma função a um manipulador de evento, como atribui momentaneamente, você está apenas atribuindo uma referência ao código de função, não ao código propriamente dito.

OK, as referências de função parecem muito bem cuidadas, mas o que elas têm a ver com separar o conteúdo da funcionalidade?

*69 (recursos de callback) para funções

As referências de função estão intimamente ligadas a uma maneira especial de chamar funções que tem a ver com separar o conteúdo da funcionalidade. Você conhece a chamada de função a partir do seu próprio código do Mandango.

```
setSeat(i * seats[i].length + j, "marcada", "Sua poltrona");
```

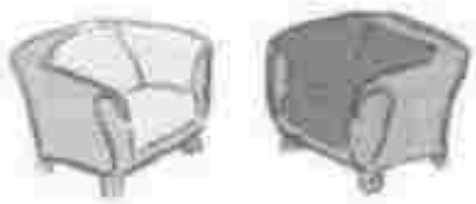
```
function setSeat(seatNum, status, description) {
  ...
}
```

Mas essa não é só uma forma de se chamar as funções nos scripts. Um outro tipo de função conhecida como função de callback pode ser chamada sem ter nada a ver com isso diretamente.

Poder do cérebro

Como você acha que o Mandango poderia obter vantagens das funções de callback?

Conversa informal



Conversa desta noite: Função normal e função de callback se confrontam

Função normal:

Então, você é o cara que eu sempre ouço falar, que não aceita chamadas locais. Qual a razão dessa atitude?

Você quer dizer como o navegador? Realmente incomum. Acho que você é um pouco teimoso em relação àqueles que conversam com o código regularmente. Puxa, seria mesmo uma perda. Esqueça! Quem se importa com o que acontece fora do script?

Talvez você tenha uma idéia. Eu gosto de saber quando a página carrega ou quando o usuário clica ou digita alguma coisa. Então, está dizendo que eu não saberia sobre essas coisas sem você?

Bem, fico contente em saber que afinal de contas nós realmente não somos tão diferentes.

Não me chame, eu chamo você.

Função de callback:

Não há nenhuma atitude, eu apenas sirvo a uma finalidade diferente. Prefiro só ser chamado de locais incomuns e distantes.

Ouça, não tem nada a ver com quem é melhor ou pior. Todos nós somos códigos de script, é que eu ofereço aos observadores externos um meio de acessar o código de script. Sem mim, você nunca saberia quando algo ocorre fora do script.

Na verdade, todos. Lembre-se de que a idéia central da criação de scripts é oferecer aos usuários da web uma experiência melhor. Se um script não tivesse nenhum meio de identificar eventos fora dele mesmo seria muito difícil melhorar a experiência do usuário.

Tudo bem. O navegador me chama e, em muitos casos, eu chamo você, já que responder aos acontecimentos externos sempre exige várias funções.

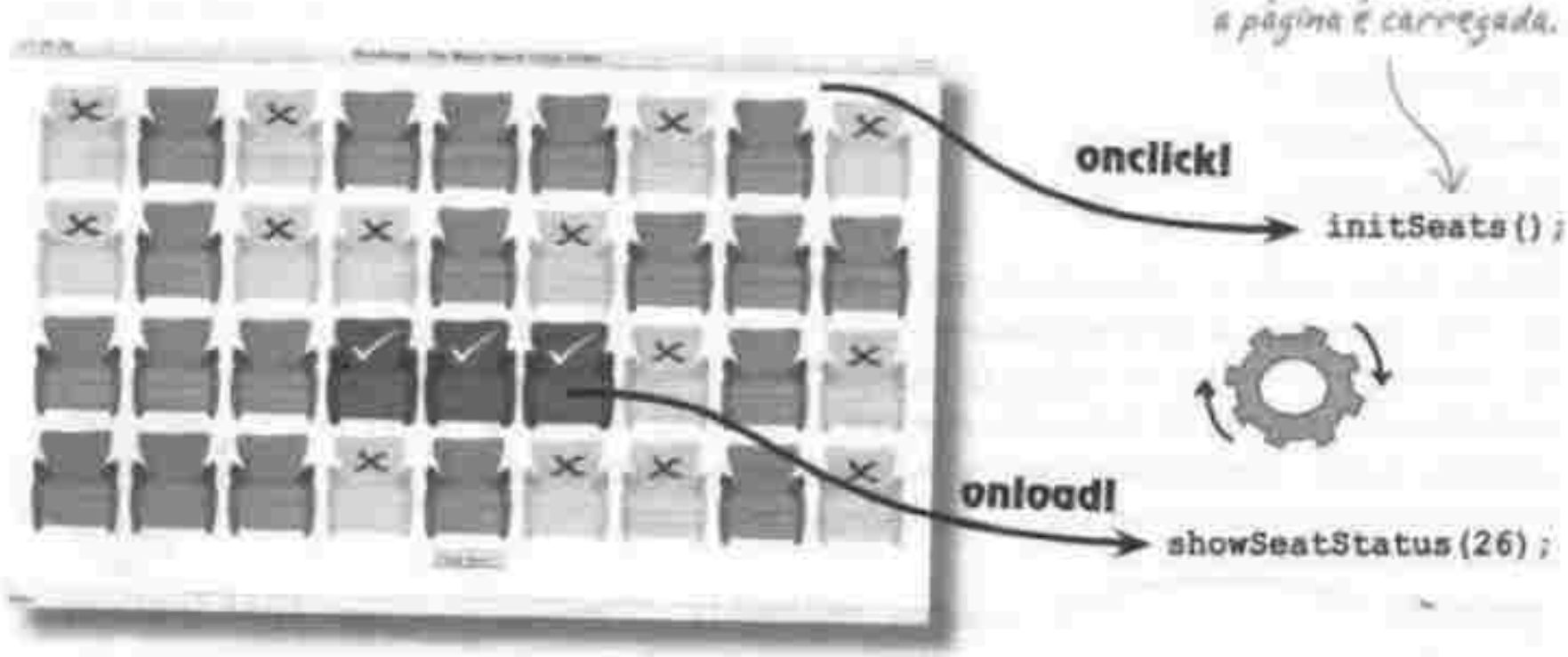
Sim. Acho que nos veremos por aí.

Boa sorte aí.

Eventos, callbacks e atributos HTML

Usamos as funções de callback o tempo todo, as quais são chamadas pelo navegador no lugar de seu próprio código. O uso mais comum de funções de callback é na manipulação de eventos. O Mandango já conta expressivamente com suas funções de callback. Na verdade, as funções de manipulação de evento são a base do problema que envolve a mistura de códigos HTML e JavaScript.

O navegador chama initSeats() quando a página é carregada.



Essas funções de callback são conectadas aos eventos no código HTML da página do Mandango.

O atributo HTML onload é usado para conectar a função initSeats() ao evento onload.

```
<body onload="initSeats();">

<img id="seat26" src="" alt="" onclick="showSeatStatus(26);" />
```

O atributo HTML onclick é usado para conectar a função showSeatStatus() ao evento onclick.

Essa técnica de ligar as funções de manipulação de evento aos eventos via atributos HTML funciona perfeitamente bem, mas ela tem o inconveniente de exigir que o código JavaScript seja misturado com o código HTML. As referências de função possibilitam isolar essa mistura e separar o HTML e o JavaScript...

Como conectar eventos usando referências de função

Em vez de usar um atributo HTML para conectar uma função de callback a um evento como um manipulador de evento, você pode atribuir uma referência de função diretamente no código JavaScript. Em outras palavras, não há a necessidade de se arriscar dentro do código HTML – apenas defina a função de callback usando uma referência de função, tudo de dentro do código JavaScript.

```
window.onload = initSeats;
```

Não há parênteses após o nome da função, porque você não quer executar a função, apenas referenciá-la.

O evento onload é uma propriedade do objeto window.

Uma referência à função initSeats() é atribuída à propriedade do evento onload.

Então, definir um manipulador de evento estritamente em código JavaScript envolve a atribuição de uma referência de função a uma propriedade de evento de um objeto. No caso desse código de manipulação de evento `onload`, a atribuição da referência de função faz com que a função `initSeats()` seja chamada quando o evento for desencadeado. Embora a chamada aconteça automaticamente quando o evento é disparado, o efeito é este:

onload! → `window.onload() ;` → `initSeats() ;`

O evento onload aciona um manipulador de evento através da propriedade window.onload...

...e já que a propriedade é definida como uma referência de função, a função initSeats() é chamada para manipular o evento.

O lado positivo de usar uma referência de função para atribuir uma função de manipulador de evento a um evento é que ela permite separar de forma limpa o código JavaScript do código HTML – não há necessidade de atribuir o código JavaScript aos atributos de evento HTML.

```
<body onload="initSeats();">   →   <body>
```

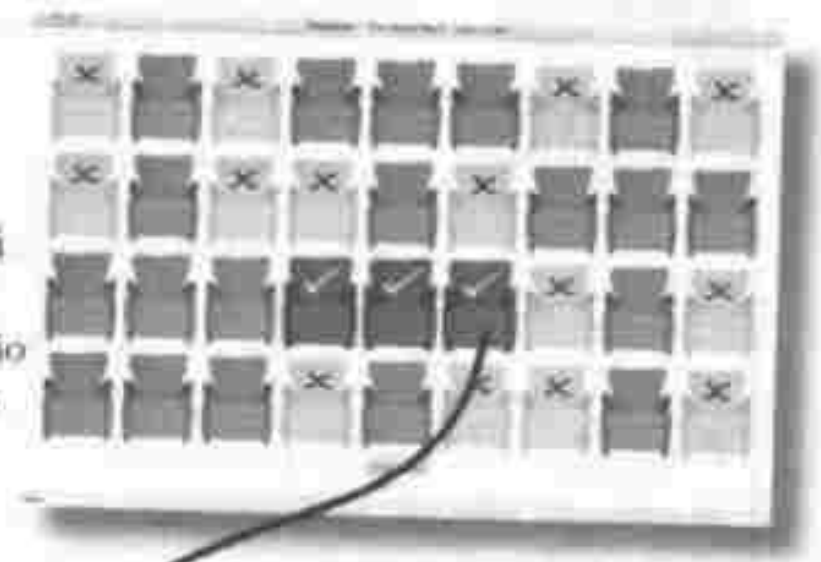
Agora, a tag `<body>` pode ser apenas a tag `<body>`, já que o manipulador de função está conectado estritamente ao código JavaScript. Só precisamos nos certificar de que o código de atribuição de evento será executado o mais breve possível, o que normalmente ocorre no topo da página.

Mas existe um problema. O que acontece se precisarmos passar um argumento para o manipulador de evento ajudá-lo a executar sua tarefa? Isso não é um problema com o `onload` no Mandango, mas o evento `onclick` precisa passar o número da poltrona que foi clicada. As referências de função não oferecem nenhum meio de passar os argumentos, então precisamos de uma outra opção...

As referências de função oferecem uma maneira conveniente de conectar as funções do manipulador de evento aos eventos.

Literais de função para o resgate

O evento `onclick` das imagens de poltronas no Mandango precisa chamar a função `showSeatStatus()` com um argumento (o número da poltrona) indicando a poltrona que foi clicada. Simplesmente atribuir uma referência à função não passará o argumento, o que representa um grande problema, mas há uma outra maneira. A solução é usar um literal de função como referência de função e, em seguida, chamar a função `showSeatStatus()` de dentro do literal de função.



O objeto de imagem da poltrona é recuperado, fazendo com que sua propriedade onclick possa ser acessada.

onclick!

```
document.getElementById("seat26").onclick = function(evt) {
  showSeatStatus(26);
};
```

O literal de função encapsula a chamada para showSeatStatus(), permitindo que um argumento seja passado para ele.

O literal de função é atribuído à propriedade do evento onclick como uma referência de função.

Um objeto de evento é passado automaticamente para um manipulador de evento como seu primeiro argumento.

O literal de função é usado estritamente como um encapsulador de chamada para a função `showSeatStatus()`, mas ele interpreta um papel importante permitindo-nos passar o número apropriado da poltrona à função. Você pode pensar no literal de função como uma função sem nome que manipula o evento. Por esta razão, os literais de função são muitas vezes chamados de **funções anônimas**.

Esse código revela como o JavaScript oferece um objeto de evento que é passado para os manipuladores de evento, neste caso através do argumento `evt`. O objeto de evento contém informações específicas para cada evento diferente. Neste caso, não precisamos saber nenhuma informação detalhada sobre o evento. Portanto, não tem problema não usar o argumento `evt`.

Literais de função permitem criar funções anônimas de manipuladores de evento.

Conecte a função `initSeats()` ao manipulador de evento `onload`, mas dessa vez use o literal de função em vez de uma referência de função.

.....

.....

.....

Solução do Exercício

Conecte a função `initSeats()` ao manipulador de evento `onload`, mas dessa vez use o literal de função em vez de uma referência de função.

```
window.onload = function(evt) {
  initSeats();
};
```

A função initSeats() é chamada dentro do literal de função do manipulador de evento onload.

O argumento evt é ignorado, já que o manipulador de evento onload não precisa do objeto de evento.

Onde está a conexão?

Ainda existe uma questão não resolvida relacionada à conexão do evento através dos literais de função. Sabemos que o manipulador de evento onload só pode ser atribuído no topo da página dentro da tag `<script>`, como no código de script normal. E isso funciona bem, porque o código ligado a `onload` não executa mesmo depois que a página é carregada (quando o evento `onload` é disparado), como se tivéssemos usado a antiga abordagem de atribuir `initSeats()` ao atributo HTML onload da tag `<body>`. Mas onde outros manipuladores de evento de literal de função são conectados?

A resposta retorna à função de callback do manipulador de evento, que serve como um ótimo lugar para conectar todos os eventos de uma página.

O manipulador de evento onload é um excelente lugar para inicializar todos os eventos.

```
window.onload = function() {
  // Conecta outros eventos aqui
  ...
  // Inicializa as aparências das poltronas
  initSeats();
};
```

Todos os outros eventos na página podem ser conectados dentro do manipulador de evento onload.

O código específico para o evento onload é sempre executado dentro do manipulador de evento.

O que este código resume é que o manipulador de evento `onload` se torna uma função de inicialização de evento onde todos os outros eventos em uma página são definidos. Então, o manipulador de evento `onload` não só desempenha as funções normais de inicialização da página, como inicializar as poltronas, como também conecta todas as outras funções de callback do manipulador de evento da aplicação.

Não existem Perguntas Idiotas

P: Por que as funções de callback são importantes?

R: As funções de callback são significativas porque elas permitem reagir a algo que ocorre fora do seu código. Em vez de chamar uma função do seu próprio código, você cria uma função de callback que está essencialmente de prontidão esperando que algo aconteça para que ela possa agir. Quando esse algo ocorre, é responsabilidade do navegador permitir que a função de callback saiba que ela pode executar. Tudo o que você faz é preparar o terreno conectando a função de callback a um acionador, normalmente um evento.

P: Existem funções de callback diferentes dos manipuladores de evento?

R: Sim. Exploraremos um outro uso comum de funções de callback no Capítulo 12, quando elas serão chamadas para processar dados enviados pelo servidor em uma solicitação de dados que utiliza Ajax.

P: Eu ainda estou confuso em relação aos literais de função. O que eles são? Por que são importantes?

R: Um literal de função é apenas um corpo de função sem nome, mais ou menos parecido com um fragmento de dados literal, como um número ou string. Os literais de função são importantes porque eles são ideais em situações onde você precisa de uma rápida função de callback isolada. Em outras palavras, a função só é chamada uma vez, e não pelo seu código. Então, você cria um literal de função e o atribui diretamente a uma propriedade de evento, em vez de criar uma função identificada e atribuir sua referência. Isso é muito mais do que uma questão de eficiência na codificação, aproveitando-se do fato de que não precisa de uma função oficialmente identificada em algumas situações. E lembre-se de que os literais de função só são realmente necessários quando precisar fazer mais do que simplesmente referir-se a uma função, como passar um argumento a uma função.

Aponte seu Lápis

Experimente escrever um pedido usando um formulário de pedidos HTML de Howard. Não se preocupe, você não será cobrado pelo anúncio de banner!

```
window.onload = function() {
  // Conecta o evento de botão Encontrar Poltrona
  document.getElementById("findseats")......... = ............;

  // Conecta os eventos de imagem de poltrona
  document.getElementById("seat0")......... = function(evt) {........................};
  document.getElementById("seat1")......... = function(evt) {........................};
  document.getElementById("seat2")......... = function(evt) {........................};

  ...

  // Inicializa as aparências das poltronas
  ..............
};
```

Aponte seu Lápis
Solução

Experimente escrever um pedido usando um formulário de pedidos HTML de Howard. Não se preocupe, você não será cobrado pelo anúncio de banner!

Todo manipulador de evento onload é um literal de função.

A função findSeats() está ligada ao evento onclick que usa a referência de uma função.

```
window.onload = function() {
  // Conecta o evento de botão Encontrar Poltrona
  document.getElementById("findseats").onclick = findSeats;

  // Conecta os eventos de imagem de poltrona
  document.getElementById("seat0").onclick = function(evt) {showSeatStatus(0);};
  document.getElementById("seat1").onclick = function(evt) {showSeatStatus(1);};
  document.getElementById("seat2").onclick = function(evt) {showSeatStatus(2);};
  ...

  // Inicializa as aparências das poltronas
  initSeats();
}
```

A função initSeats() é chamada para terminar as tarefas de onload.

A propriedade onclick de cada imagem de poltrona é acessada para definir os manipuladores de evento onclick.

showSeatStatus() é chamada de dentro de um literal de função, de modo que um argumento possa ser passado para ela.

Pontos de Bala

- As funções de callback são chamadas pelo navegador em resposta às situações que ocorrem **fora do script**.
- As referências de função podem ser usadas para **atribuir funções** como se elas fossem **variáveis**.
- As referências de função permitem conectar as funções do manipulador de evento no código JavaScript **sem alterar o código HTML**.
- Os literais de função são **funções sem nome** úteis em situações onde uma função identificada não é necessária.

Não existem Perguntas Idiotas

P: Por que o manipulador de evento `onload` no Mandango é criado como um literal de função?

R: Porque realmente não há nenhum motivo para criar uma função identificada para isso. A função só é chamada uma vez e em resposta ao evento `onload`. Poderíamos facilmente ter criado apenas uma função identificada e atribuído sua referência a `window.onload`, mas a conexão entre a função de callback e o evento fica mais óbvia quando a função está diretamente ligada ao evento usando um literal de função.

P: As outras funções de callback têm que estar conectadas ao manipulador de evento `onload`?

R: Sim. Talvez você pense que poderia conectá-las diretamente no interior da tag `<script>` no topo da página. Mas lembre-se de que o conteúdo da página ainda não terminou de carregar. Portanto, todas as chamadas de `getElementById()` fracassariam e os manipuladores de evento não seriam conectados. O manipulador `onload` garante que a página seja carregada.

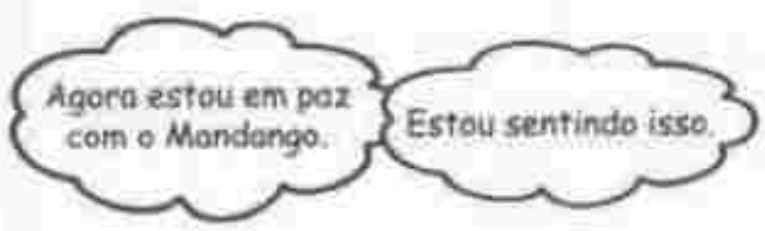
O shell de uma página HTML

Separar o código JavaScript do código HTML no Mandango revela o quanto a parte estrutural da página se torna verdadeiramente mínima. Isso torna o código HTML muito mais fácil de manter sem a preocupação de atropelar o códigos JavaScript que poderia interromper a aplicação.

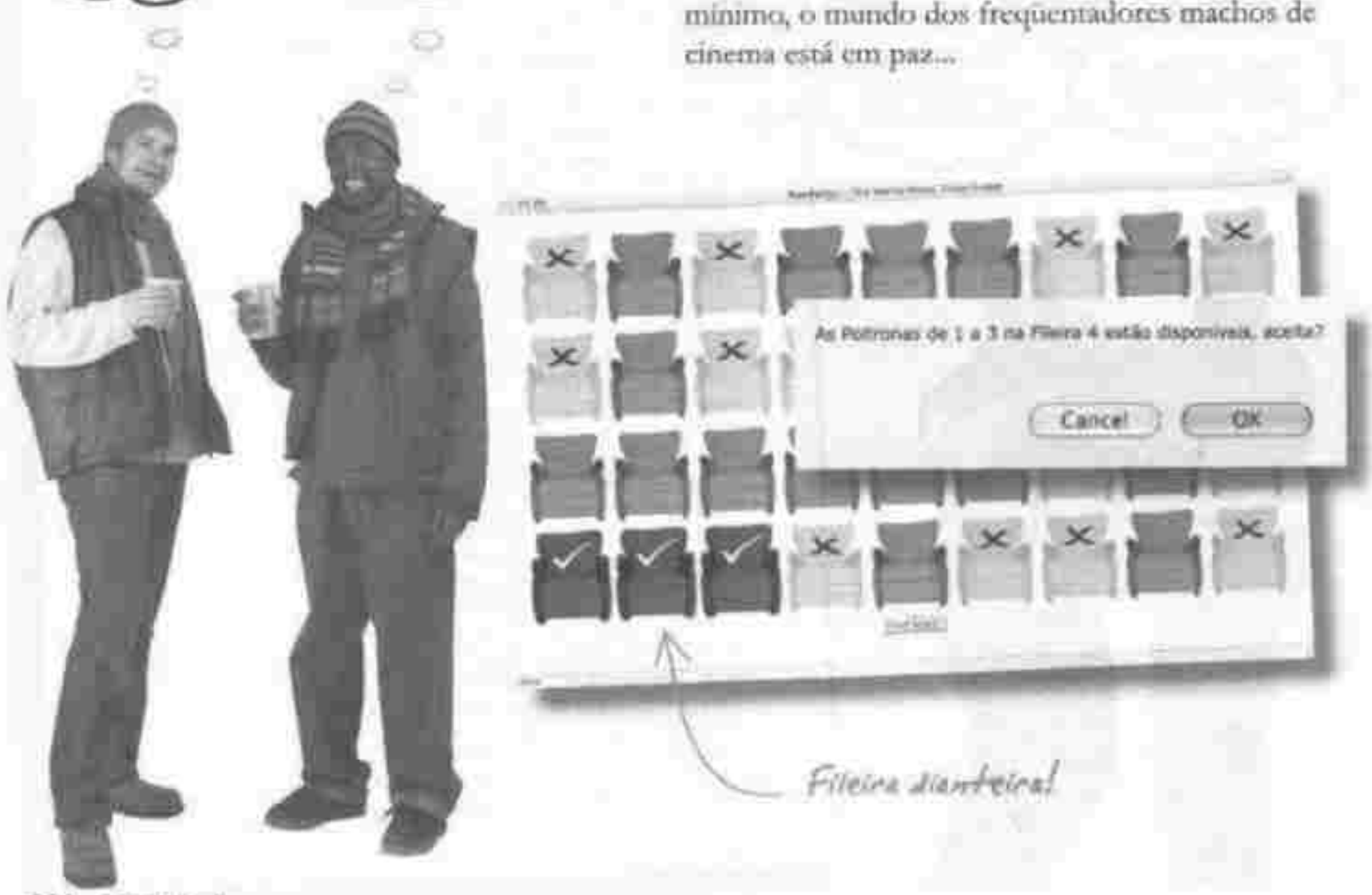
```
<body>
  <div style="margin-top:75px; text-align:center">
    <img id="seat0" src="" alt="" />
    <img id="seat1" src="" alt="" />
    <img id="seat2" src="" alt="" />
    <img id="seat3" src="" alt="" />
    <img id="seat4" src="" alt="" />
    <img id="seat5" src="" alt="" />
    <img id="seat6" src="" alt="" />
    <img id="seat7" src="" alt="" />
    <img id="seat8" src="" alt="" />
    <img id="seat9" src="" alt="" />
    <br />
    <img id="seat10" src="" alt="" />
    <img id="seat11" src="" alt="" />
    <img id="seat12" src="" alt="" />
    <img id="seat13" src="" alt="" />
    <img id="seat14" src="" alt="" />
    <img id="seat15" src="" alt="" />
    <img id="seat16" src="" alt="" />
    <img id="seat17" src="" alt="" />
    <img id="seat18" src="" alt="" />
    <img id="seat19" src="" alt="" />
    <br />
    <img id="seat20" src="" alt="" />
    <img id="seat21" src="" alt="" />
    <img id="seat22" src="" alt="" />
    <img id="seat23" src="" alt="" />
    <img id="seat24" src="" alt="" />
    <img id="seat25" src="" alt="" />
    <img id="seat26" src="" alt="" />
    <img id="seat27" src="" alt="" />
    <img id="seat28" src="" alt="" />
    <img id="seat29" src="" alt="" />
    <br />
    <img id="seat30" src="" alt="" />
    <img id="seat31" src="" alt="" />
    <img id="seat32" src="" alt="" />
    <img id="seat33" src="" alt="" />
    <img id="seat34" src="" alt="" />
    <img id="seat35" src="" alt="" />
    <img id="seat36" src="" alt="" />
    <img id="seat37" src="" alt="" />
    <img id="seat38" src="" alt="" />
    <img id="seat39" src="" alt="" />
    <br />
    <input type="button" id="findseats" value="Find Seats" />
  </div>
</body>
```

Um pequeno passo para o JavaScript...

Embora não tenhamos conseguido solucionar a paz mundial, demos um passo na direção certa usando o JavaScript para compreender a mudança de temperatura. Transformar grandes problemas em pequenos, concentrar-se na singularidade do objetivo e lutar pela reutilização do código são formas de melhorar os scripts através das funções.

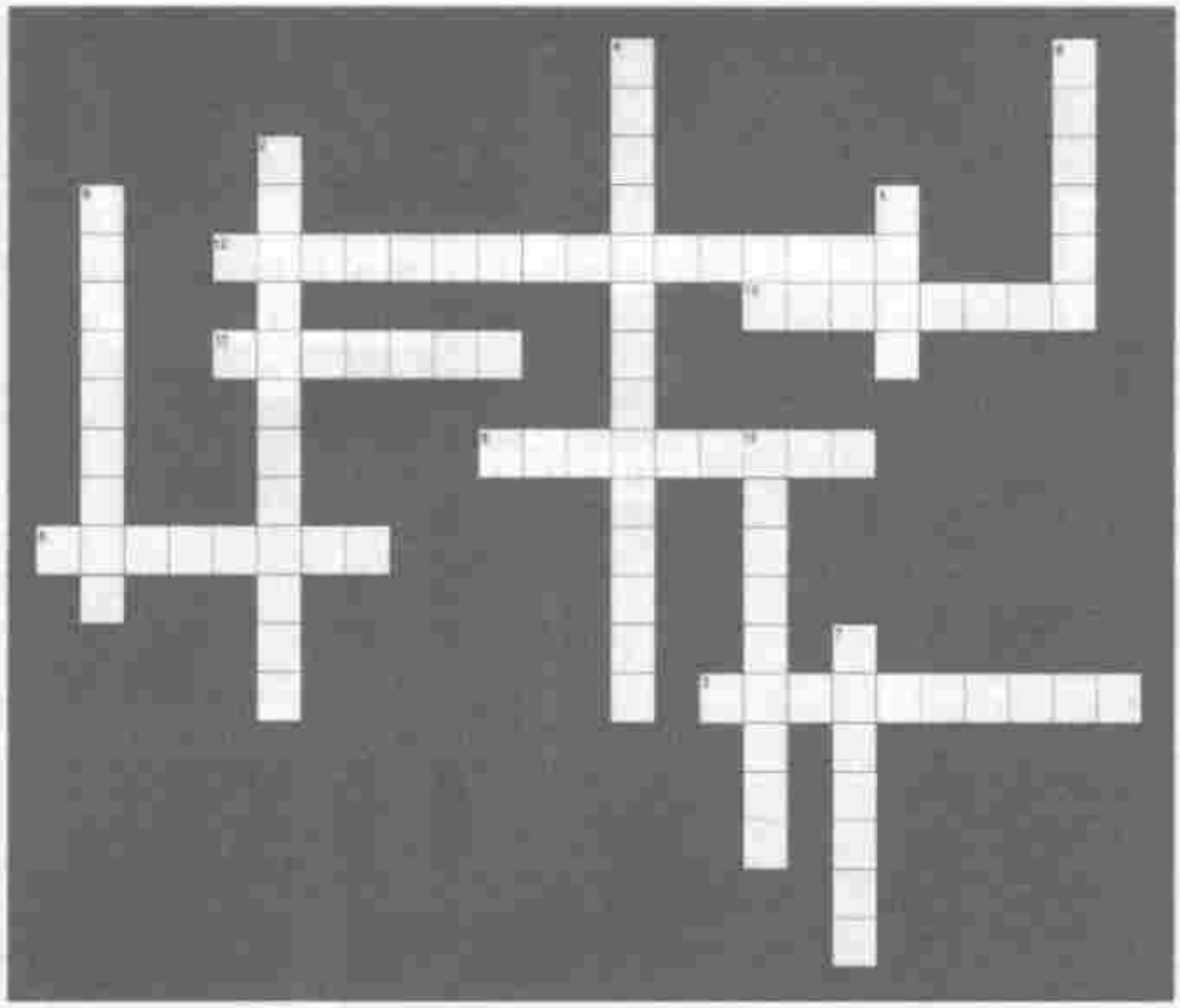


E, obviamente, Seth e Jason colocaram as mesmas técnicas de solução de problemas para funcionar criando uma versão do Mandango mais bem organizada e mais fácil de manter. No mínimo, o mundo dos freqüentadores machos de cinema está em paz...



Cruzadas do JavaScript

Muito Bem, chegou o momento novamente. Preencha as respostas da cruzadinha baseado nas declarações a seguir.



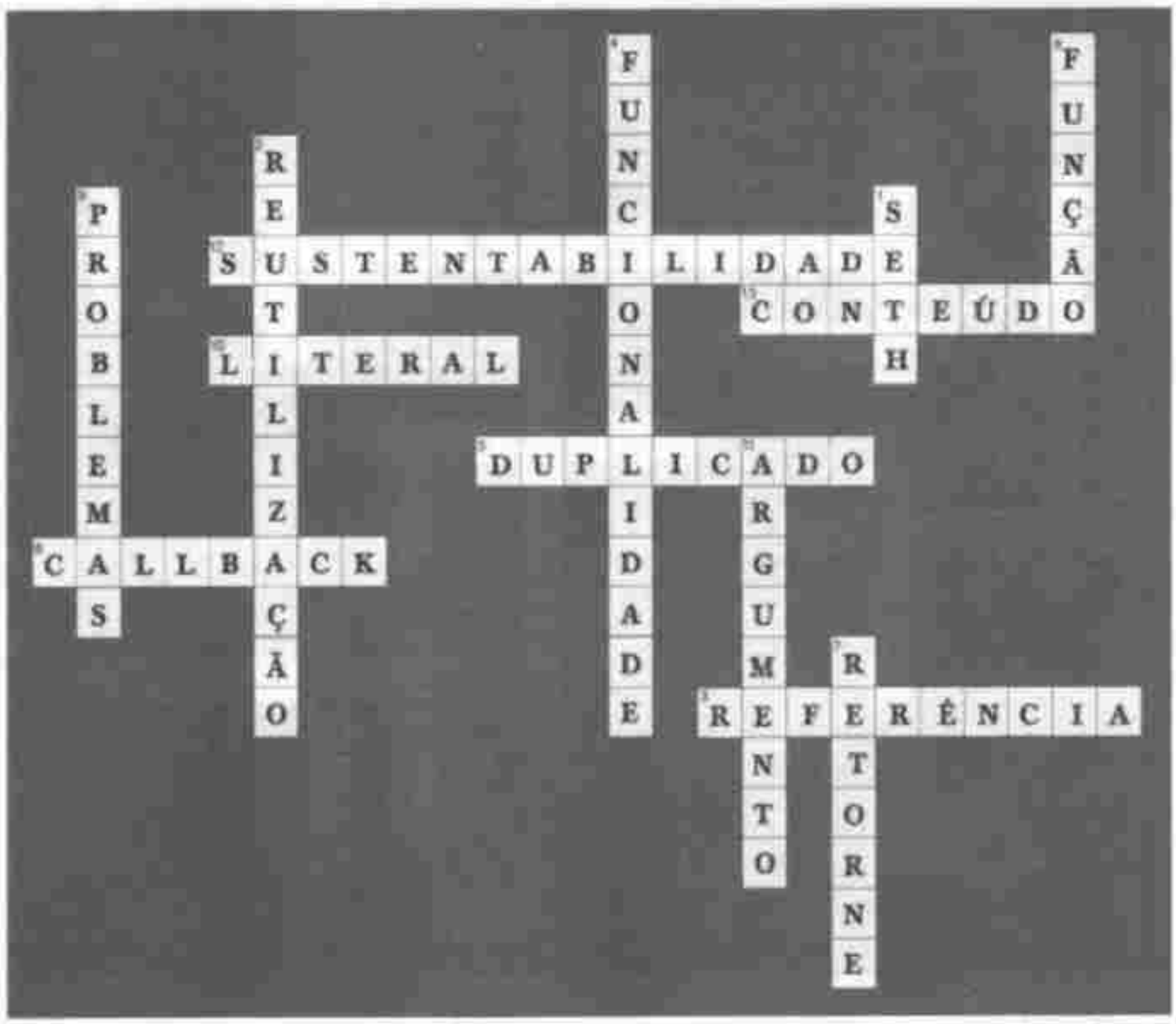
Horizontal

3. Ao atribuir uma função a uma variável, usa-se uma..... de função
5. Funções ajudam a eliminar esse tipo de código.
6. Nunca chame esse tipo de função.
10. Um corpo de função sem nome.
12. Quando o código é relativamente fácil de modificar, ele é considerado de boa .....
13. O código HTML representa essa parte de uma página da Web.

Vertical

1. ... agora está em paz com o Mandango.
2. As funções melhoram a ..... do código para que você não o copie desnecessariamente.
4. O JavaScript representa essa parte de uma página da Web.
7. Para passar os dados de volta a partir de uma função, apenas ..... -os.
8. Um fragmento de código JavaScript reutilizável.
9. As funções são excelentes para eliminá-los.
11. É como você passa os dados para uma função.

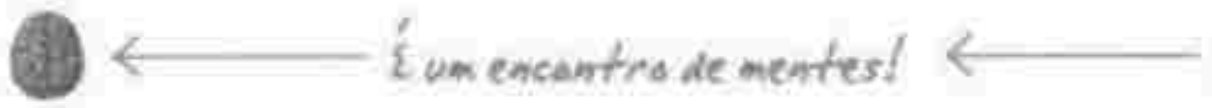
Solução da Cruzada do JavaScript

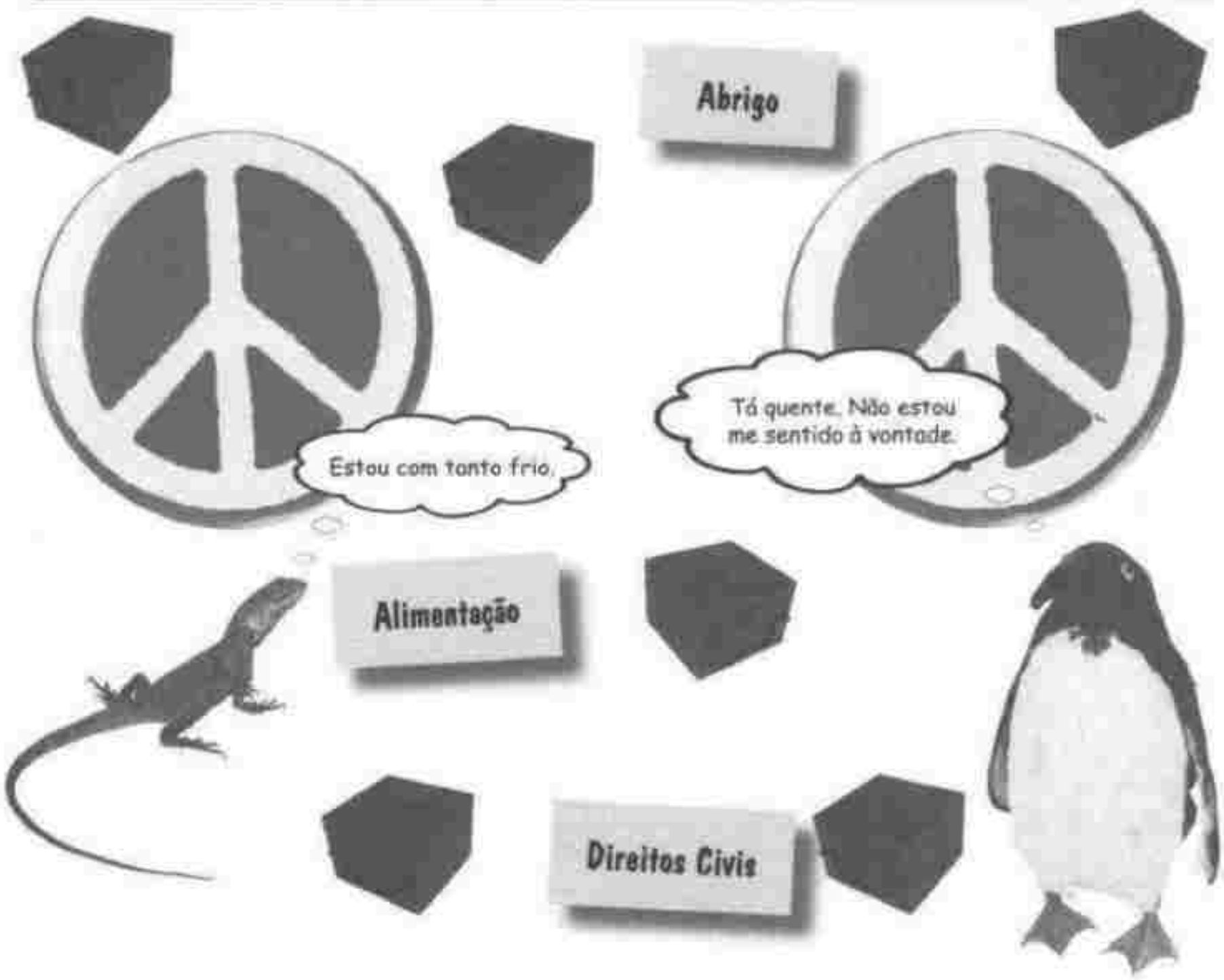


DOBRA DA PÁGINA

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

O que as funções agregam à vida de seu JavaScript?





A paz é sempre uma proposta delicada.
Mesmo com o código JavaScript,
somente o mais organizado dos
códigos leva à tranqüilidade e
ao sossego.

Não é fácil levar uma vida de
satisfação, pelo menos em termos de
JavaScript.

Como Fazer o Usuário Dizer Tudo

Você não precisa ser cortês ou furtivo para conseguir informações bem-sucedidas dos usuários com o JavaScript. Mas tem que ser cuidadoso. Os humanos têm a estranha tendência de cometer erros, o que significa que ***você não pode*** sempre contar com os dados fornecidos nos formulários on-line como sendo ***precisos***. Inicie o JavaScript. **Transmitindo os dados através do código certo do JavaScript** como se estivessem sendo digitados, você poderá **tornar seus aplicativos Web muito mais confiáveis** e também **retirar um pouco de carga do servidor**. Precisamos ***economizar a largura da banda preciosa para coisas importantes***, como, por exemplo, vídeos com retardo e imagens de bichinhos de estimação bonitinhos.

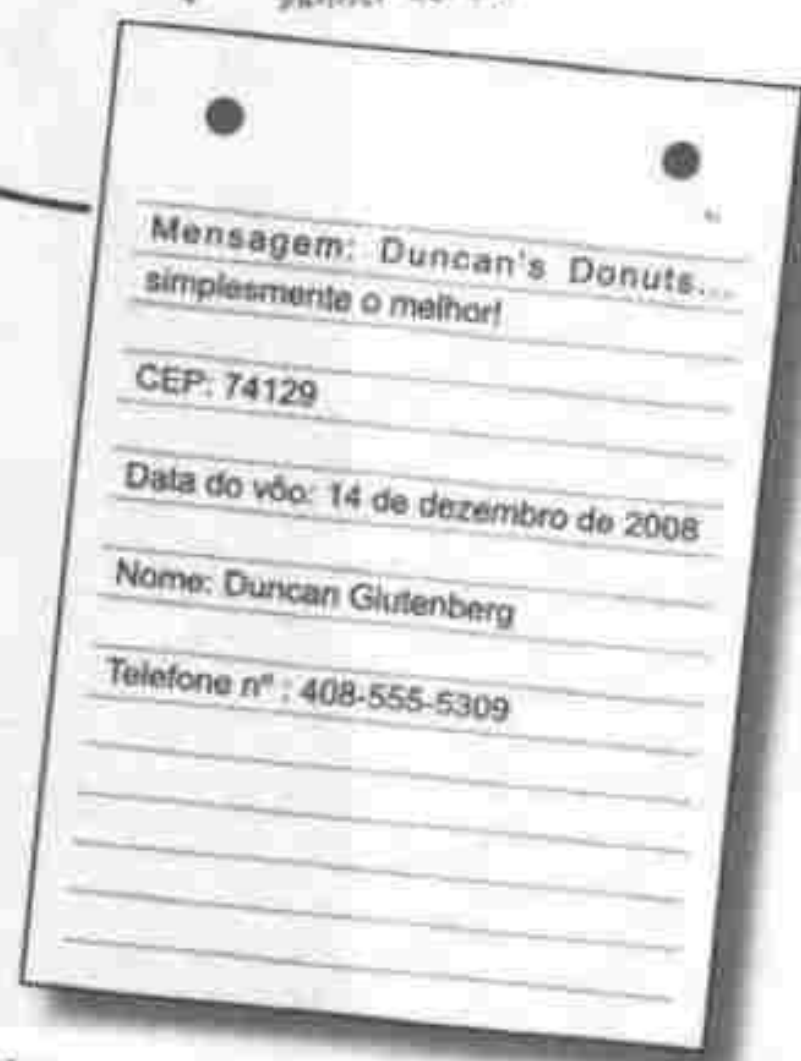
Bannerocity: mensagens aéreas amistosas

O aviador perito e adorável Howard transformou seu amor pelo vôo em um negócio de banner aéreo, Bannerocity. Howard deseja dar um significado inteiramente novo ao termo "anúncio de banner" pegando pedidos on-line para os banners aéreos. Além de dar início a seu novo negócio, Howard espera que o sistema de pedidos on-line libere seu tempo para que ele possa passar mais tempo divertindo-se com vôos amistosos.

Howard conseguiu um negócio impressionante com seu clássico avião da II Guerra Mundial.

Bannerocity... anúncios de banner no céu!

Atualmente, o formulário de pedidos em papel de Howard captura todas as informações necessárias para o pedido de banner aéreo.



É muito importante para o formulário de pedidos on-line Bannerocity capturar todas as informações de pedidos associadas a um pedido de banner aéreo. Howard calcula que o formulário de pedidos on-line deve incluir todos os campos no formulário de papel, mais um campo de e-mail, uma vez que os clientes estarão preenchendo o formulário on-line.

Mensagem

A mensagem a ser exibida no anúncio de banner aéreo.

Data do vôo

A data na qual o anúncio de banner é mostrado no ar.

Nome

O nome do cliente.

CEP

A área geográfica onde a mensagem será exibida. Howard voa sobre uma área postal específica ao exibir um anúncio.

Número de telefone

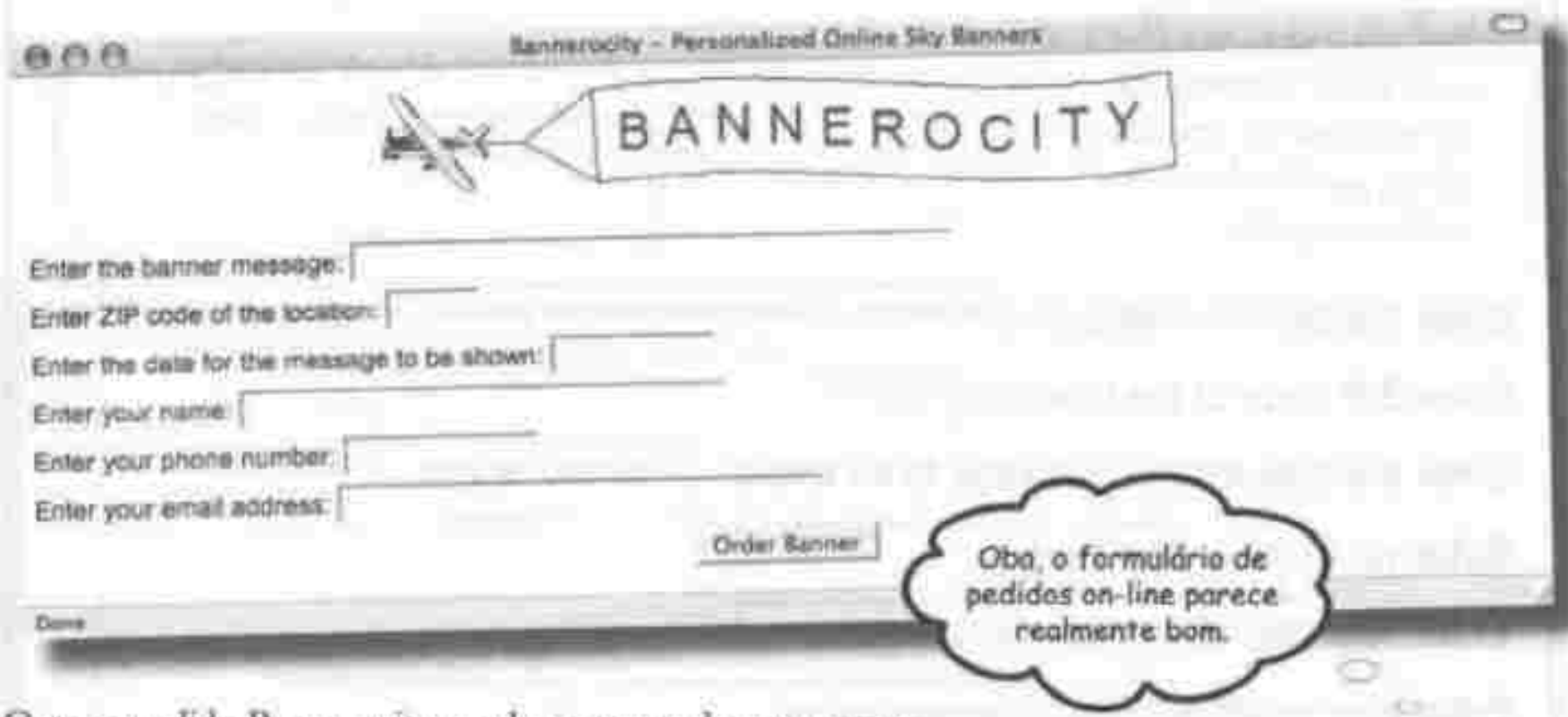
O número de telefone do cliente.

E-mail

O endereço de e-mail do cliente.

Formulário HTML Bannerocity

Com um pouco de ajuda do HTML, a primeira tentativa de Howard em um formulário de pedidos on-line para o Bannerocity parece ótima.



O novo pedido Bannerocity excelente tem todos os campos do formulário necessários e está pronto para pegar os pedidos sem usar nenhum código JavaScript. Qual é a desvantagem?

Faça download em www.altabooks.com.br e comece a trabalhar!

Howard tem estado no setor privado há algum tempo, mas ele não parece querer largar aquele uniforme bonito.

Aponte seu Lápis

Experimente escrever um pedido usando um formulário de pedidos HTML de Howard. Não se preocupe, você não será cobrado pelo anúncio de banner!

Enter the banner message:

Enter ZIP code of the location:

Enter the date for the message to be shown:

Enter your name:

Enter your phone number:

Enter your email address:

Aponte seu Lápis
Solução

Experimente escrever um pedido usando um formulário de pedidos HTML de Howard. Não se preocupe, você não será cobrado pelo anúncio de banner!

Você não sabia, mas Howard pode exibir apenas 32 caracteres em seu banner aéreo, portanto este é muito comprido.

Números demais no código postal – devem ser apenas cinco.

Enter the banner message: Mandango... o localizador macho de assentos de cinema para caras fortes!

Enter ZIP code of the location: 100012

Enter the date for the message to be shown: 11 de março de 2009

Enter your name: Seth Tinselman

Enter your phone number: (212) 555-5339

Enter your email address: setht@mandango

OK, o nome está bem.

Este é um endereço de e-mail inválido – está faltando a extensão do domínio, como, por exemplo .biz.

O número de telefone deve estar na forma ###-###-#### sem parênteses.

A data não está no formato AAAA/DD/AAAA que o formulário espera.

Quando o HTML não é suficiente

Como os formulários on-line são tão bons quanto os dados fornecidos, Howard percebe que precisará da ajuda do JavaScript para assegurar que os dados do formulário sejam confiáveis. E ele precisa ajudar a esclarecer para o usuário o que são exatamente "dados bons". Por exemplo, sem um tipo de dica da página Bannerocity, o usuário não tem nenhum indício de que há um limite de 32 caracteres na mensagem de anúncio de banner ou que a data deve ser fornecida como MM/DD/AAAA.

Desculpe-me, há um limite de 32 caracteres no texto de anúncio do banner.

Enter the banner message: Mandango... o localizador de assentos de cinema para caras fortes!

Com alguma ajuda do JavaScript, estes dados inválidos podem ser evitados.

Não, os formulários HTML não falam normalmente!

Há um pequeno problema. Todo código JavaScript inteligente de manipulação de dados no mundo não pode ajudar Howard até que ele descubra como ter acesso aos dados do formulário a partir do JavaScript...

Como acessar os dados do formulário

Para acessar os dados que foram fornecidos em um formulário, primeiro é necessário identificar com exclusividade cada campo em um formulário. Isso é lidado no código HTML, usando um dos dois atributos (ou ambos!).

O nome do atributo identifica com exclusividade um campo em um formulário.

O atributo id identifica com exclusividade qualquer elemento em uma página.

```
<input id="zipcode" name="zipcode" type="text" size="5" />
```

Ambos os atributos servem como identificadores para o campo de entrada.

Enter ZIP code of the location: 100012

A razão para as duas abordagens diferentes da identificação para os campos do formulário tem relação com o modo como os campos são acessados. A primeira abordagem usa a função `getElementById( )` que é usada para acessar qualquer elemento em uma página. Essa abordagem funciona bem, mas há uma mais simples que é específica para os campos do formulário.

Todo campo do formulário tem um objeto do formulário que pode ser transmitido a qualquer função que é chamada para validar os dados do formulário.

```
<input id="zipcode" name="zipcode" type="text" size="5" onclick="showIt(this.form)"/>
```

O bom do objeto `form` é que ele também é um array que mantém todos os campos no formulário. Mas os itens no array não são armazenados usando índices numéricos; ao contrário, são armazenados usando o seu identificar exclusivo como definido no atributo name! Portanto, se o objeto `form` for transmitido para uma função como um argumento denominado `theForm`, o valor fornecido no campo do código postal poderá ser acessado assim:

O valor do campo do código postal é exibido.

O nome do argumento do objeto do formulário.

```
function showIt(theForm) {
    alert(theForm["zipcode"].value);
}
```

100012

OK

O nome exclusivo do campo do formulário, como definido no atributo name da tag <input>.

Queremos o valor armazenado no campo, não o campo em si.

Esta abordagem para acessar os dados do formulário não é melhor nem pior que usar `getElementById( )`, a não ser que torna o código mais fácil de ler, uma vez que envolve uma menor quantidade. O objeto form fornece um atalho, portanto, você também pode usá-lo.

Não existem Perguntas Idiotas

P: Por que um campo do formulário individual tem acesso ao objeto `form`?

R: Algumas vezes não tem, mas lembre-se de que um campo do formulário é capaz de chamar uma função de validação que precisa acessar os dados nos outros campos do formulário. Neste caso, o objeto `form` disponível em cada campo torna-se a chave para acessar de modo conveniente os outros campos do formulário. Este objeto é geralmente transmitido à função de validação para que ela possa obter rapidamente qualquer campo necessário. O exemplo Bannerocity continua a depender muito do objeto do formulário para acessar os campos em seu formulário de pedidos.

P: Então, o valor é uma propriedade de um campo do formulário? Isso significa que cada campo do formulário é realmente um objeto?

R: Sim e sim. Cada campo do formulário é representado por um objeto para o código JavaScript e o objeto `form` fornece um modo rápido e fácil de acessar tal objeto para qualquer campo em um formulário. Assim que tiver um objeto do campo do formulário em mãos via `form["objectname"]`, poderá então acessar seu valor adicionando a propriedade `value`. Você aprenderá tudo sobre os objetos nos Capítulos 9 e 10.

Sei o **quanto** é importante acessar os dados do formulário no JavaScript para assegurar que os dados sejam bons. Mas como se sabe **quando** verificá-los?

Saber quando verificar os dados do formulário depende de escolher o evento correto de entrada do usuário para lidar.

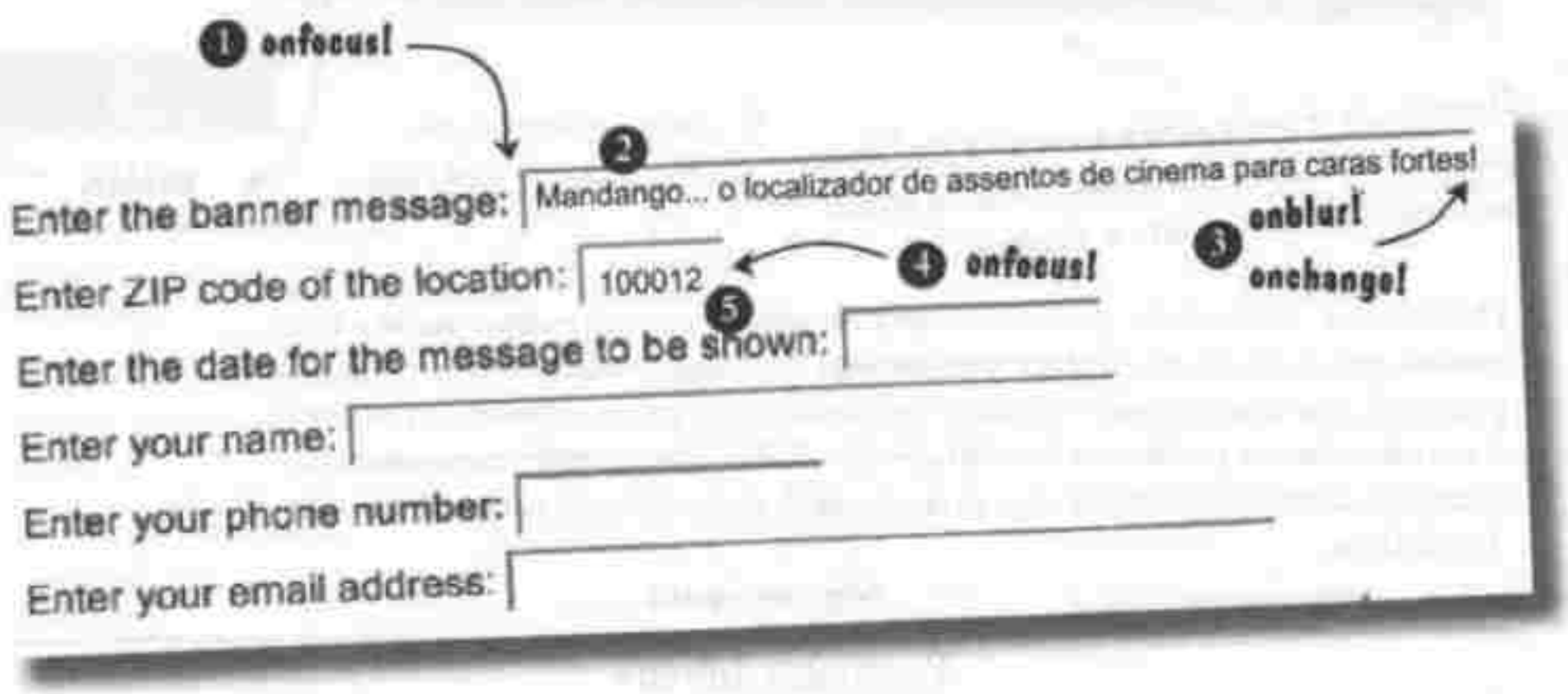
A resposta para o "quando" na validação dos dados envolve os eventos e compreender qual evento permite que você saiba quando o usuário forneceu dados em um determinado campo. Em outras palavras, o desafio é responder ao evento que é inicializado imediatamente depois dos dados terem sido fornecidos. Mas a pergunta ainda permanece... qual é o evento?

Os campos do formulário seguem uma cadeia de eventos

Quando os dados são fornecidos em um formulário, uma torrente de eventos é gerada. Você pode usar esses eventos como um ponto de entrada para validar os dados campo por campo. Mas fazer isso envolve dar uma olhada na sequência típica de entrada e compreender exatamente quais eventos são iniciados... e quando.

1. Selecione o campo de entrada (`onfocus`).
2. Forneça dados no campo.
3. Deixe o campo de entrada para ir para o próximo (`onblur/onchange`).
4. Selecione o próximo campo de entrada (`onfocus`).
5. Forneça dados no campo...

Fornecer dados em um formulário inicia uma cadeia de eventos JavaScript interessantes.



O evento `onfocus` é iniciado quando um campo é selecionado pela primeira vez para a entrada, ao passo que `onblur` é iniciado quando um campo perde a seleção de entrada. O evento `onchange` é parecido com `onblur`, exceto que será inicializado apenas se o campo perder a seleção de entrada **e** seu conteúdo tiver sido alterado.

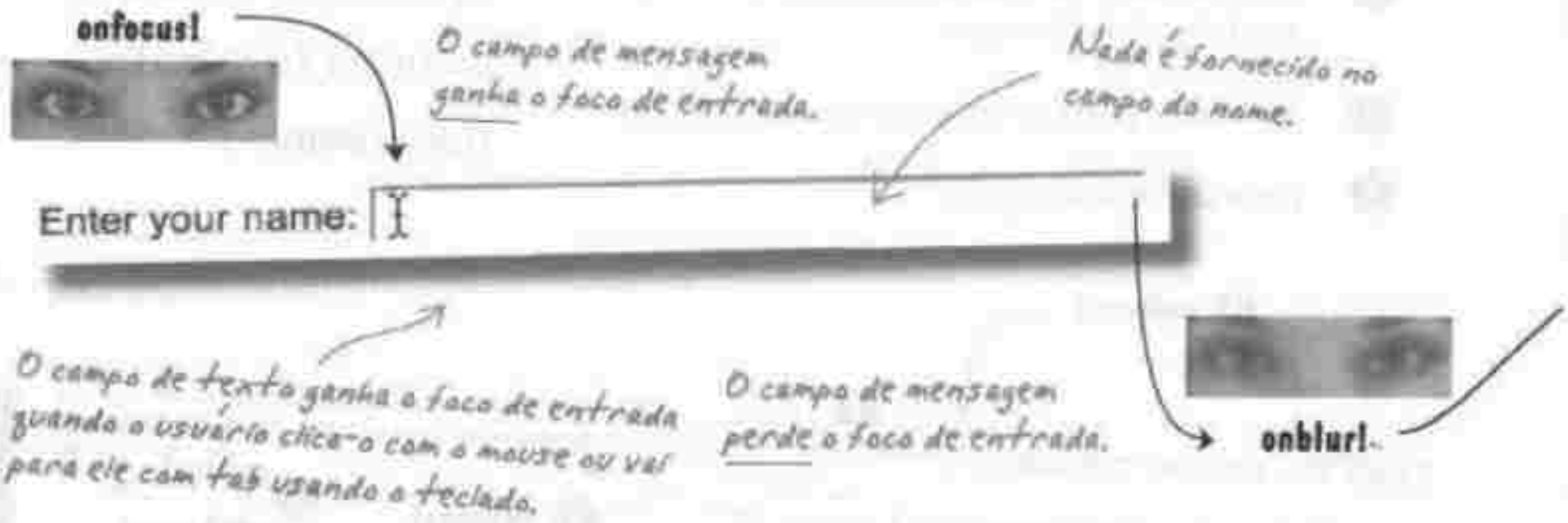
Poder do cérebro

Qual evento faz mais sentido para validar um campo de dados do formulário?

Como perder o foco com onblur

Embora haja um argumento a ser criado para usar o evento `onchange` para a validação dos dados, há um problema em particular no sentido de que não podemos usá-lo para validar um campo vazio. A razão é que nada existe quando um formulário é carregado pela primeira vez, mas como os dados do formulário não mudaram também, `onchange` não será inicializado mesmo que um usuário navegue os campos vazios do formulário. O evento `onblur` resolve esse problema sendo sempre inicializado quando uma seleção de entrada, ou o **foco**, deixa um campo.

O evento onblur é um inicializador perfeito para a validação dos dados.



Diferente de `onchange`, `onblur` é iniciado sempre que um campo perde o foco, mesmo que os dados não tenham sido tocados. Isso significa que `onblur` é muito poderoso, mas também que você deve ter cuidado com como e quando irá notificar o usuário sobre os problemas da validação dos dados. Um exemplo característico... a caixa de alerta, que pode ser uma proposta fácil, mas arriscada, para a notificação da validação.

Não existem Perguntas Idiotas

P: Alguns eventos não são gerados quando o usuário realmente fornece os dados do formulário?

R: Sim. Vários eventos são gerados em resposta a teclas pressionadas, como, por exemplo, `onkeypress`, `onkeyup`, `onkeydown` etc. Embora existam certamente situações nas quais faz sentido responder a esses eventos, geralmente eles não são boas idéias para validar os dados, pois o usuário normalmente ainda está no meio do fornecimento das informações quando esses eventos são inicializados. Validar os dados usando esses eventos seria um tanto arrogante, notificando o usuário sobre cada erro tipográfico e parte não terminada de dados quando eles estão sendo fornecidos. Provavelmente, é melhor aguardar até que os usuários deixem um campo, que é uma indicação de que terminaram de fornecer os dados. E isso é feito respondendo ao evento onblur.

P: `onblur` parece um nome estranho para um evento. O que significa?

R: A idéia é que `onblur` deve ser o correspondente de `onfocus`. Portanto, se `onfocus` é iniciado quando um elemento ou campo do formulário **ganha** o foco de entrada, então `onblur` é iniciado quando um campo **perde** o foco. Mesmo que a palavra "foco" neste contexto não esteja referindo-se exatamente à visão, a palavra "blur" é usada para indicar a falta de foco. É uma brincadeira do JavaScript com as palavras que acaba sendo um pouco confusa. Apenas se lembre que `onblur` é iniciado quando um campo perde o foco.

Você pode usar uma caixa de alerta para validar as mensagens

As caixas de alerta são de fato úteis para exibir rapidamente informações para o usuário e elas representam a forma mais simples de notificação para permitir que o usuário saiba que algo está errado com os dados do formulário. Apenas chame a função `alert( )` enquanto lida com o evento `onblur`, caso um problema seja detectado nos dados do formulário.

Verifique para saber se o campo do formulário está vazio.

Uma função de validação é chamada para validar os dados do nome.

```
function validateNonEmpty(inputField) {
  // Veja se o valor de entrada contém texto
  if (inputField.value.length == 0) {
    // Os dados são inválidos, portanto notifique o usuário
    alert("Please enter a value.");
    return false;
  }

  return true;
}
```

Instrui o usuário sobre como remediar o problema.

Please enter a value.

OK

Como o campo do nome está vazio, um alerta é exibido.

Aponte seu Lápis

Quantos eventos `onblur` são gerados pela seguinte seqüência de entrada? Quantos eventos `onchange`? Não se preocupe com `onfocus`.

Enter your name: Seth Tinselman

Enter your phone number:

Enter your email address: setht@mandango

O número de eventos `onblur`: .....

O número de eventos `onchange`: .....

Aponte seu Lápis
Solução

Quantos eventos `onblur` são gerados pela seguinte seqüência de entrada? Quantos eventos `onchange`? Não se preocupe com `onfocus`.

Enter your name: Seth Tinselman

Enter your phone number:

Enter your email address: setht@mandango

O número de eventos `onblur`: 3

O número de eventos `onchange`: 2

Conversa informal

Conversa desta noite: onblur e onchange conversam sobre quando reagir aos dados ruins do formulário

onblur:

Atualmente, parece que os scripts estão sempre preocupados com o que cabe ao usuário. Imagino que é onde você e eu entramos.

É algo que venho querendo conversar com você. O boato é que havia alguns dados vazios por aí e muitos dedos apontaram na sua direção.

É verdade. Ninguém está questionando sua confiabilidade sobre quando os dados mudaram. O problema é: o que acontecerá se um formulário iniciar com dados vazios que nunca mudarão?

onchange:

Está certo. Somos uma dupla, sempre é bom deixar alguém saber quando um elemento ou campo do formulário perde o foco, alguns dados mudaram... ou ambos!

Francamente, estou um pouco chocado com a acusação. Você sabe que eu nunca perco uma oportunidade de notificar um script sobre os dados que mudaram.

onblur:

É verdade, não faz nenhum sentido e nem para alguns usuários, mas eles fazem isso do mesmo modo.

Acalme-se, está tudo bem. Não é sua culpa. Veja, não é sua responsabilidade preocupar-se com os dados que nunca mudam. Lembre-se de que você tem o nome on**change**.

Não nos deixemos levar. Como eu estava dizendo, não é sua responsabilidade. Portanto, se um script estiver preocupado em validar um formulário para assegurar que os campos não fiquem vazios, realmente não deve usar você para iniciar o código de validação.

Ouça, tento não reagir tanto. Mesmo que você não possa ser ideal para iniciar o código de validação, isso não significa que os scripts não estejam interessados algumas vezes nos dados sendo ou não alterados. E o formulário que permite que as pessoas editem os dados que são armazenados? Faria muito sentido permitir apenas uma gravação se os dados realmente tivessem mudado.

Realmente! Portanto, não há nenhuma necessidade de se alarmar.

De nada. Tudo bem, eu adoraria conversar mais, porém tenho alguns dados para validar...até logo!

onchange:

Você está dizendo que um usuário é capaz de tentar enviar um formulário com campos em branco? Isso não faz nenhum sentido.

Tudo bem, então um formulário inicia em branco com campos vazios. O usuário pula a entrada de alguns dados e o envia com os campos ainda vazios... ah, cara, acho que estou começando e ter um ataque de pânico!

Mas e a situação sobre a qual acabamos de falar onde falho miseravelmente no script com dados vazios e o mundo começa a ruir?

Bem é um alívio, mesmo que não signifique que eu não posso servir para ninguém mais. Espere, acho que o pânico está voltando de novo...

Ei, é verdade. Então você está dizendo que ainda tenho uma finalidade?

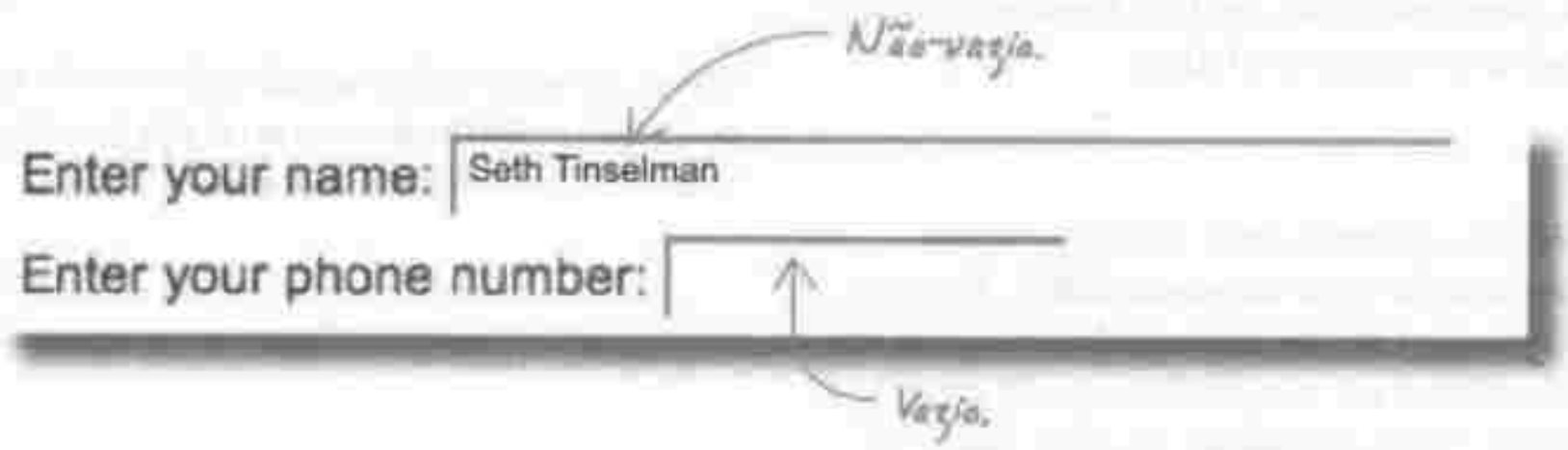
Obrigado. Isso é muito tranqüilizador.

Verificando... algo

De volta ao Bannerocity, Howard sabe que no mínimo precisa validar o formulário Bannerocity para assegurar que todos os campos tenham dados. Mas de um ponto de vista JavaScript, isso envolve ver as coisas de uma perspectiva estranha. Mais especificamente, ao invés de verificar para saber se um campo **tem algo**, você precisa verificar e ver se o campo **não tem nada**. Em outras palavras, "algo" é igual a "nada".

Algo = Nada

O motivo para este pensamento complicado é porque é mais fácil verificar um campo do formulário para ver se está vazio do que "não-vazio". Portanto, uma primeira linha de defesa para obter a validação dos dados envolve verificar para saber se um campo está preenchido.



A função de validação de Howard deve responder ao evento `onblur` de cada campo do formulário para executar a validação do preenchimento. Por exemplo:

O campo do formulário é identificado com exclusividade para que possa ser acessado a partir de outras partes do script.

```
<input id="phone" name="phone" type="text" size="12"
    onblur="validateNonEmpty(this);"/>
```

O objeto do campo do formulário é transmitido para a função usando a palavra-chave this.

A função validateNonEmpty() é chamada em resposta a onblur para verificar e ver se o campo está preenchido.

A palavra-chave `this` é usada neste código para fazer referência ao próprio campo do formulário. Transmitir o campo do formulário como um objeto para a função de validação dará à função uma oportunidade de acessar o valor do campo do formulário, assim como o objeto `form` que mantém todos os campos do formulário, o que algumas vezes é útil.

Valide os campos para assegurar que você "não tem nada"

Cada campo do formulário tem um código parecido que liga o evento `onblur` à função `validateNonEmpty( )`. Ligando o evento `onblur` de cada campo à função todos os dados do formulário serão validados.

```
<input id="name" name="name" type="text" size="32"
    onblur="validateNonEmpty(this);"/>
```

O campo do formulário name chama a função de validação para saber se há um name.

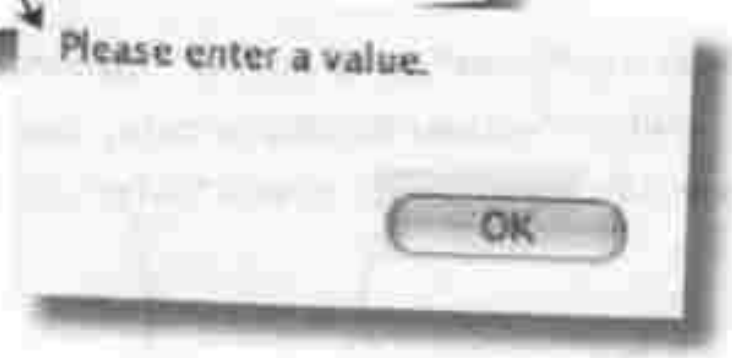
O campo do formulário phone number chama a função de validação para saber se há um número de telefone.

```
function validateNonEmpty(inputField) {
  // Veja se o valor de entrada contém texto
  A propriedade length mostra o número de caracteres em uma string.
  if (inputField.value.length == 0) {
    // Os dados são inválidos, portanto notifique o usuário
    alert("Please enter a value.");
    return false;
  }

  return true;
}
```

A propriedade length revela o número de caracteres em uma string.

Não há nenhum número de telefone, portanto a função exibe um alerta e retorna false.

Há um nome, portanto a função simplesmente retorna true...



Neste exemplo, os valores de retorno da função `validateNonEmpty( )` não são usados. Sua finalidade é comunicar ao código que chama o resultado da validação: `true` se os dados forem OK ou `false` se não. Um pouco mais tarde, veremos como esses valores de retorno são usados para assegurar que os dados do formulário sejam bons antes de enviar um formulário para o servidor para o processamento.

Uma função de validação preenchida verifica para assegurar que os campos não foram deixados vazios.

Poder do cérebro

Você pode pensar em alguma desvantagem ao usar uma caixa de alerta para notificar o usuário sobre os dados ruins do formulário?

Validação sem caixas de alerta desagradáveis

Não levou muito tempo para Howard perceber que as caixas de alerta não são ideais para notificar o usuário sobre os dados inválidos. Ele está tendo muitas reclamações sobre todas as caixas de alerta que surgem quando os usuários estão tentando fazer os pedidos Bannerocity. Todos ficamos um pouco condicionados a ser incomodados quando uma janela pop-up interrompe uma experiência on-line e a validação dos dados não é diferente, mesmo que as caixas de alerta, neste caso, estejam tentando ser úteis.

A solução de Howard é um "sistema de ajuda passivo", que não envolve caixas de alerta e, portanto, não interrompe o fluxo de entrada dos dados. Porém, essa abordagem passiva para notificar o usuário requer adicionar alguns elementos HTML novos ao formulário.

Um novo elemento HTML fornece um lugar para exibir as mensagens de ajuda da validação.

Enter your phone number: Please enter a value.

O novo elemento de ajuda HTML representa uma grande melhoria em relação às caixas de alerta, pois não fica no caminho e ainda transmite as mesmas informações para o usuário. E tudo que requer estruturalmente é o acréscimo de uma tag `<span>` HTML que é nomeada para coincidir com o campo do formulário ao lado. Essa nova tag `<span>` aparece no código do formulário da página Web logo abaixo do campo de entrada.

Um segundo argumento para validateNonEmpty(), agora, segue junto com o elemento do texto de ajuda.

```
<input id="phone" name="phone" type="text" size="12"
    onblur="validateNonEmpty(this, document.getElementById('phone_help'))" />
<span id="phone_help" class="help"></span>
```

A tag inicialmente está vazia, mas ela tem um ID que a associa ao campo do formulário phone number.

A classe de estilo é usada para formatar o texto de ajuda em uma fonte itálica vermelha, embora seja difícil ver o vermelho nesta página impressa!

Estes dois IDs têm que coincidir para que o texto de ajuda seja exibido para o campo de entrada.

Com o elemento span colocado mantendo o texto de ajuda, tudo que está faltando é o código que, de fato, exibe a mensagem de ajuda. E com base no segundo argumento para a função `validateNonEmpty( )`, há uma boa chance de que a função será responsável por assegurar que o texto de ajuda seja visto pelo usuário.

Um validador não-vazio mais sutil

A solução da mensagem de ajuda passiva engenhosa de Howard mostra-se em uma nova função `validateNonEmpty( )` melhorada que, agora, também lida com a tarefa de definir e limpar as mensagens de ajuda para um campo do formulário.

O objeto do texto de ajuda é transmitido para a função como o segundo argumento.

```
function validateNonEmpty(inputField, helpText) {
  // Veja se o valor de entrada contém texto
  if (inputField.value.length == 0) {
    // Os dados são inválidos, portanto defina a mensagem de ajuda
    if (helpText != null)
      helpText.innerHTML = "Please enter a value.";
    return false;
  }
  else {
    // Os dados são bons, portanto limpe a mensagem de ajuda
    if (helpText != null)
      helpText.innerHTML = "";
    return true;
  }
}
```

Primeiro, certifique-se de que o elemento do texto de ajuda exista (helpText != null), então defina sua propriedade innerHTML para a mensagem de ajuda.

Também é importante limpar o texto de ajuda assim que os dados tiverem sido fornecidos em um campo do formulário.

A validação dos dados no Bannerocity, agora, está muito melhor graças à nova abordagem de ajuda passiva, que ainda usa uma boa dose de JavaScript, mas de um modo muito mais claro, pelo menos em termos de aperfeiçoar a experiência do usuário.

É muito melhor... nenhuma caixa de alerta... menos intrusivo.

Enter ZIP code of the location: *Please enter a value.*
Enter the date for the message to be shown: *Please enter a value.*
Enter your name: Bob Trademan
Enter your phone number: *Please enter a value.*
Enter your email address: *Please enter a value.*

Os dados do nome estão presentes para que a mensagem de ajuda não seja mostrada.

Quando os dados estiverem faltando, agora o Bannerocity exibirá apenas uma mensagem de ajuda passiva.

Algo bom demais

Como resultado, a validação do preenchimento funciona bem, mas dados demais podem ser tão problemáticos quanto dados de menos. Verifique o último banner de Howard, que mostra um novo problema no formulário de pedidos Bannerocity.

Poder do cérebro

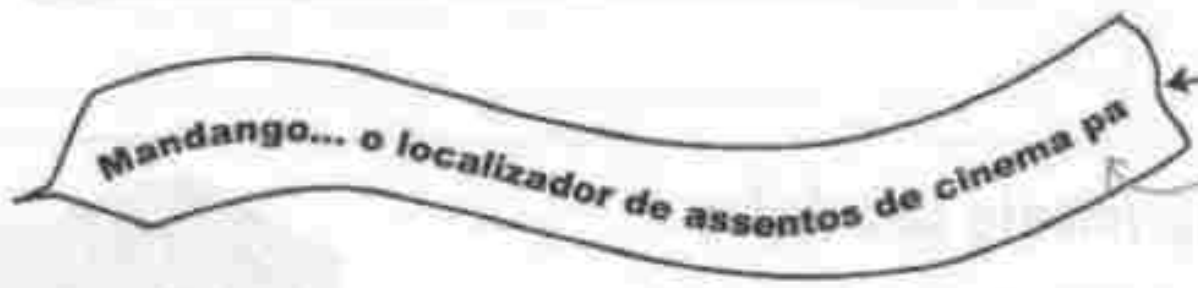
O que está errado com este banner e o que pode ser feito para resolver o problema?

O tamanho importa...

O problema com o Bannerocity é que o banner aéreo de Howard pode manter apenas 32 caracteres, mas o campo de mensagem no formulário de pedido não tem limite. Certamente, é bom que o usuário seja avisado se não puder fornecer uma mensagem, mas uma mensagem que é longa demais ainda passa sem nenhum problema. E isso é, na verdade, um grande problema para Howard!

O usuário forneceu texto demais, mas o Bannerocity não o informou que é um problema.

Enter the banner message: Mandango... o localizador de assentos de cinema para caras fortes!



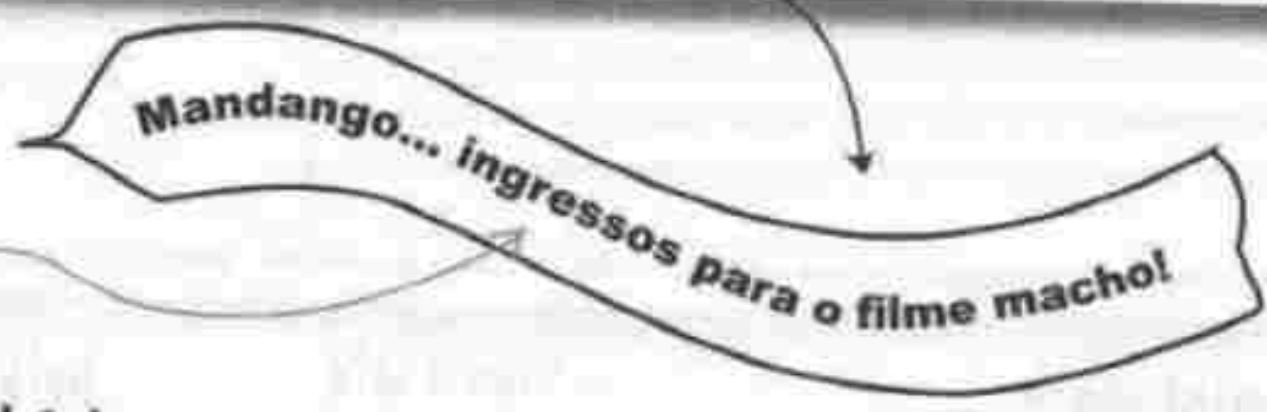
Como há mais texto do que o banner pode manter, o texto do anúncio é cortado... não é bom!

Tentar mostrar um texto ilimitado em um espaço limitado não funciona e acaba resultando em clientes infelizes... como Seth e Jason. A solução é validar o campo de mensagem para que ele tenha um comprimento máximo. Uma mensagem de ajuda mais personalizada para a nova validação também é uma boa idéia para assegurar que os usuários compreenderão o limite de tamanho da mensagem.

Este texto do banner está dentro do limite de 32 caracteres.

Enter the banner message: Mandango... ingressos para o filme macho!

O texto cabe bem no banner quando ele é limitado dentro do tamanho máximo.



Aponte seu Lápis

Escreva o pseudocódigo que mostra como uma nova função de validação do comprimento do Bannerocity funcionará, validando os comprimentos mínimo e máximo.

Aponte seu Lápis
Solução

Escreva o pseudocódigo que mostra como uma nova função de validação do comprimento do Bannerocity funcionará, validando os comprimentos mínimo e máximo.

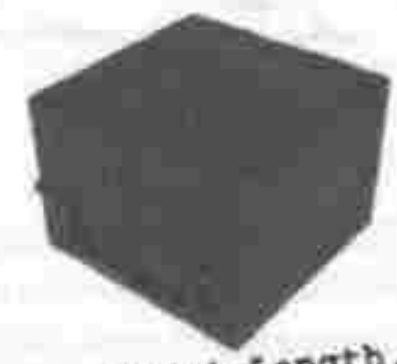
Os argumentos da função minLength e maxLength seriam definidos para 1 e 32 para o texto da banner Bannerocity.

```
If (fieldValue is shorter than minLength OR fieldValue is longer than maxLength)
    Show the help text
Else
    Clear the help text
```

Como validar o comprimento dos dados

O papel da nova função `validateLength( )` é verificar e ver se o valor em um campo do formulário segue certos comprimentos mínimo e máximo. No caso do Bannerocity, a função é basicamente usada para limitar o comprimento do campo de texto do banner, embora aplique um comprimento mínimo de um caractere também. É improvável que Howard encontre um cliente que queira anunciar uma letra L apenas, por exemplo, mas a idéia principal é assegurar que não haja mais de 32 caracteres e não menos que um.

Além dos comprimentos mínimo e máximo a serem aplicados por `validateLength( )`, a função também requer mais dois argumentos para o campo de entrada ser validado e o elemento do texto de ajuda usado para exibir uma mensagem de ajuda. Isso dá um total de quatro argumentos para a função.



```
validateLength(minLength,
  maxLength, inputField,
  helpText);
```

maxLength

A quantidade máxima do texto permitido no campo de entrada.

Enter the banner message: Mandengo... o localizador de assentos de cinema para cães fofos! Please enter a value 1 to 32 characters in length.

minLength

A quantidade mínima de texto requerido no campo de entrada

inputField

O campo de entrada cujo comprimento está sendo validado.

helpText

O elemento no qual o texto de ajuda é exibido.

```
<input id="message" name="message" type="text" size="32"
   onblur="validateLength(1, 32, this, document.getElementById('message_help'))" />
<span id="message_help" class="help"></span>
```

O objeto do campo de entrada da mensagem de banner.

A função `validateLength( )` obtém o valor do argumento `inputField` e verifica para assegurar que tenha, pelo menos, `minLength` de comprimento, porém não mais que `maxLength`. Se o valor for curto ou longo demais, uma mensagem de ajuda será exibida no elemento `helpText` na página.

Pontos de Bala

- Todo campo do formulário é acessível como um objeto JavaScript.
- Dentro de um objeto do campo do formulário, há uma propriedade chamada `form` que representa o **formulário inteiro** como um array de campos.
- O evento `onblur` é iniciado quando o foco de entrada deixa um campo do formulário e é uma ótima maneira de inicializar uma função de validação de dados.
- As caixas de alerta são um modo desajeitado e geralmente chato de notificar os usuários sobre os problemas de validação dos dados.
- Uma abordagem passiva para a ajuda de validação é muito mais fácil e menos confusa para os usuários.
- A propriedade `length` de uma string mostra o número de caracteres nela.

Aponte seu Lápis

Termine o código da função `validateLength( )`, prestando muita atenção nos argumentos sendo transmitidos.

```
function validateLength(minLength, maxLength, inputfield, helpText) {

// Pela se o valor da entrada contém pelo menos minLength, porém não mais que caracteres maxLength

  // Os dados são inválidos, portanto exiba a mensagem de ajuda

  // Os dados são bons, portanto limpe a mensagem de AJUDA

}
```

Aponte seu Lápis
Solução

Termine o código da função `validateLength( )`, prestando muita atenção nos argumentos sendo transmitidos.

```
function validateLength(minLength, maxLength, inputField, helpText) {
  // Se o valor do campo não pelo menos minLength, conte-lo e se ele ainda estiver
  // suficientemente longo e estiver [illegible] inputField.value.length > maxLength
  // Se nosso [illegible] [illegible] mostre a mensagem de ajuda
  // [illegible] no [illegible]
  helpText.innerHTML = "Please enter a value 1 to 32 characters in length.";
  return false;
  else {
    // Se nosso [illegible] [illegible] e comprimento de ajuda
    // [illegible] null
    helpText.innerHTML = null;
    return true;
  }
}
```

Verifique os comprimentos mínimo e máximo do valor do campo do formulário.

Defina a mensagem de ajuda para refletir o problema do comprimento do campo.

Limpe o texto de ajuda se o comprimento do campo for bom.

O problema da mensagem resolvido

Howard está aliviado pelo problema do comprimento do banner ter sido resolvido. Sem dinheiro para comprar um banner maior, ele não tinha nenhuma outra opção boa, portanto atacar o problema no nível JavaScript acabou sendo uma boa idéia. Pelo menos agora os usuários conhecem logo os limites dos banners Bannerocity antes de fazer os pedidos.

Agora, o texto de ajuda chama uma mensagem de banner que excede o limite.

Enter the banner message. Continue sua aventura com a Aventura do Bonequinho! Please enter a value 1 to 32 characters in length.

P: O que realmente está errado com a validação da caixa de alerta? A maioria das pessoas não percebe que uma caixa de alerta não é um anúncio pop-up?

R: Embora provavelmente seja verdade que a maioria das pessoas percebe que a caixa de alerta JavaScript não é um anúncio pop-up, ainda não elimina o fato de que elas são vistas como sendo altamente intrusivas. Qualquer coisa que requeira que o usuário pare o que está fazendo e clique em algo em outra janela é uma interrupção. Portanto, embora as caixas de alerta tenham um lugar na programação JavaScript, a validação dos dados não.

P: O uso de `this` no código `onblur` para os campos do formulário é ainda confuso. O campo do formulário é um objeto ou o próprio formulário é um objeto?

R: A resposta é ambos. Dentro do contexto de um elemento HTML, a palavra-chave `this` refere-se ao elemento como um objeto. Portanto, no caso de um campo do formulário, `this` é uma referência para o objeto do campo do formulário. Dentro de um dado objeto do campo do formulário, há uma propriedade chamada `form` que fornece acesso ao formulário inteiro como um objeto. Portanto, quando você vir `this.form` no código `onblur` para um campo do formulário, o que realmente está vendo é uma referência para o próprio formulário, como um objeto.

A finalidade de `this.form` no código Bannerocity é ter acesso ao elemento do texto de ajuda associado a um determinado campo de entrada. Lembre-se de que `this.form` é uma referência para o objeto `form`, que também é um array associativo contendo todos os campos do formulário. Portanto, você pode acessar rapidamente um campo denominado `my_field` usando a notação do array com o código `this.form["my_field"]`. Você poderia também usar `getElementById()`, mas a abordagem do formulário é um pouco mais concisa.

Não existem Perguntas Idiotas

P: Quando um elemento do texto de ajuda está associado dentro de um campo de entrada, qual é o significado dos atributos name e id de cada um?

R: O `id` de um elemento do texto de ajuda é baseado no `id/name` de seu campo de entrada associado, mas não é exatamente o mesmo. Mais especificamente, o ID do texto de ajuda usa o ID do campo de entrada com o texto `_help` adicionado ao final. O motivo dessa convenção de nomenclatura é criar uma conexão limpa e consistente entre um campo de entrada e o elemento que exibe o texto de ajuda para o campo. Na verdade, você pode nomear os IDs do elemento do texto de ajuda como quiser, contanto que sejam exclusivos e sejam devidamente transmitidos para as funções de validação.

P: Por que é necessário limpar a mensagem de ajuda quando os dados são validados como sendo bons em uma função de validação?

R: Lembre-se de que a finalidade do texto de ajuda é fornecer ao usuário ajuda quando há um problema. Se os dados fornecidos em um campo do formulário forem bons, não haverá nenhum problema e, portanto, nenhuma razão para exibir uma mensagem de ajuda. E como uma mensagem de ajuda já pode estar visível a partir de uma validação anterior no mesmo campo, a cautela é limpar a mensagem de ajuda sempre que um campo é validado com dados bons.

P: O que acontecerá se um elemento do texto de ajuda não for fornecido como um argumento para uma função de validação?

R: O script pesquisa e pesquisa o elemento ausente, superaquece a página e deixa uma bagunça queimada no navegador. Tudo bem, não é realmente assim. Segundo a construção, o sistema de ajuda passiva no Bannerocity desaparecerá silenciosamente se o argumento do texto de ajuda não for usado em uma função de validação. Portanto, o texto de ajuda para o campo de entrada simplesmente não é mostrado. Isso significa que o sistema do texto de ajuda é designado a ser inteiramente opcional. O que é bom nessa abordagem é que ela permite ao texto de ajuda ser usado tanto quanto for desejado; até com os campos individuais, você não é obrigado a adicionar um elemento do texto de ajuda para cada campo em um formulário.

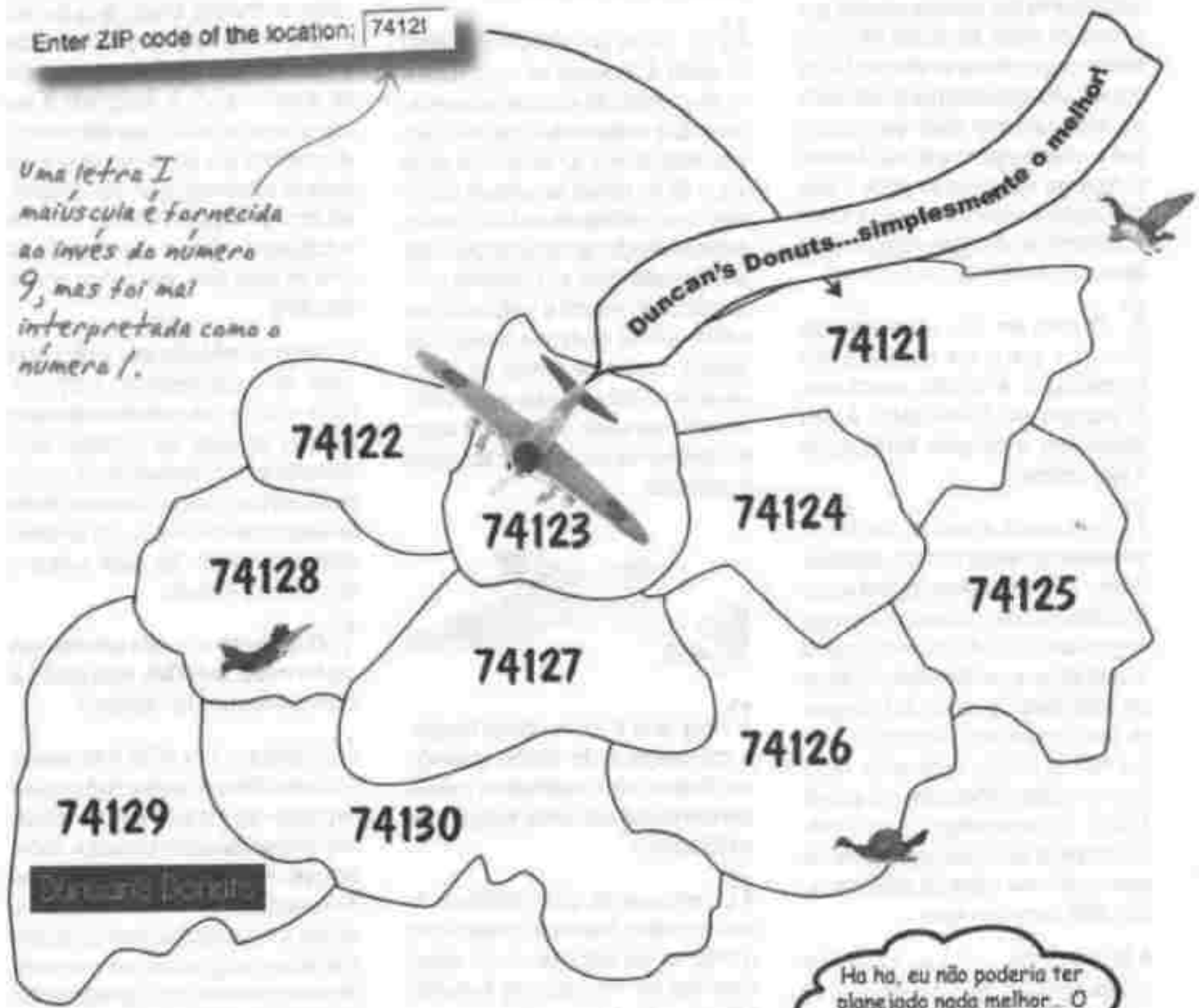
O código de validação que verifica para saber se o argumento `htmlText` não é nulo é o que permite ao elemento do texto de ajuda ser opcional. Se o elemento do texto de ajuda não for `null`, significará que o elemento existe e o texto de ajuda poderá ser exibido. Do contrário, simplesmente não faz nada porque o elemento está faltando.

P: O atributo `size` de um campo do formulário HTML não limita o comprimento do campo?

R: O atributo `size` HTML limita apenas o tamanho físico do campo do formulário na página – ele não tem nenhuma relação com quantos dados são fornecidos. Como exemplo, o campo do código postal no Bannerocity tem seu atributo `size` definido para 5, o que significa que o campo tem o tamanho na página para aceitar cerca de cinco caracteres de texto. É possível limitar o comprimento real do texto no HTML usando o atributo `maxlength`, mas não há nenhum `minlength` equivalente. Uma função de validação fornece uma maior flexibilidade ao controlar o comprimento dos caracteres que podem ser fornecidos em um campo, embora no caso de um código postal, realmente seria melhor não apenas procurar os cinco caracteres de texto, mas também assegurar que sejam cinco números. Talvez, seja algo que Howard deva considerar acrescentar ao Bannerocity...

Banner certo, lugar errado

O formulário on-line de Howard continua a causar problemas apesar de seus melhores esforços de validação. Desta vez, um código postal foi fornecido incorretamente, resultando em Howard voando várias horas sobre o lugar errado. Talvez pior que o tempo perdido de Howard seja o seu cliente infeliz, Duncan, que perdeu algumas vendas de donuts.



Nesta situação, o erro humano soma-se ao erro da entrada dos dados e cria uma bagunça realmente grande. O cliente digitou sem querer I ao invés do número 9, uma vez que as teclas estão próximas no teclado. Howard interpretou I como 1 e acaba anunciando o banner sobre o lugar errado.

Poder do cérebro

Como você validaria um código postal?

Como validar um código postal

O problema de Howard tem relação com um código postal não sendo fornecido devidamente. Simplificando, um código postal nos Estados Unidos consiste em exatamente cinco números. Portanto, validar um código postal pode ser tão simples quanto assegurar que o usuário forneça cinco números exatamente... nada mais, nada menos.

Aponte seu Lápis

Termine o código para a função `validateZIPCode( )` que valida os códigos postais para assegurar que eles tenham exatamente cinco caracteres de comprimento, assim como sejam numéricos.

```
function validateZIPCode(inputField, helpText) {
   // Primeira veja se o comprimento do valor de entrada é algo diferente de 5

   if (..........................) {
      // Os dados são inválidos, portanto defina a mensagem de ajuda
      if (helpText != null)
         helpText.innerHTML = "Please enter exactly five digits.";

      ..............
   }
   // Então, veja se o valor de entrada é um número

   else if (.........................) {
      // Os dados são inválidos, portanto defina a mensagem de ajuda
      if (helpText != null)
         helpText.innerHTML = "Please enter a number.";

      ..............
   }
   else {
      // Os dados são bons, portanto limpe a mensagem de ajuda
      if (helpText != null)
         helpText.innerHTML = "";

      ..............
   }
}
```

Aponte seu Lápis
Solução

Termine o código para a função `validateZIPCode( )` que valida os códigos postais para assegurar que eles tenham exatamente cinco caracteres de comprimento, assim como sejam numéricos.

Veja se o comprimento a string do código postal é algo diferente de 5.

```
function validateZIPCode(inputField, helpText) {
   // Primeira veja se o comprimento do valor de entrada é algo diferente de 5
   if (inputField.value.length != 5) {
      // Os dados são inválidos, portanto defina a mensagem de ajuda
      if (helpText != null)
         helpText.innerHTML = "Please enter exactly five digits.";
      return false;
   }
   // Então, veja se o valor de entrada é um número
   else if (isNaN(inputField.value)) {
      // Os dados são inválidos, portanto defina a mensagem de ajuda
      if (helpText != null)
         helpText.innerHTML = "Please enter a number.";
      return false;
   }
   else {
      // Os dados são bons, portanto limpe a mensagem de ajuda
      if (helpText != null)
         helpText.innerHTML = "";
      return true;
   }
}
```

Retorne false uma vez que o comprimento do código postal não é exatamente 5.

A função isNaN( ) verifica para saber se um valor é Not a Number.

Retorne false uma vez que o código postal não é um número.

Retorne true para indicar que o código postal é validado como sendo bom.

Nem sempre é seguro supor que os códigos postais são apenas numéricos.

Preste atenção! Se um formulário Web for capaz de receber códigos postais para os endereços fora dos Estados Unidos, então validar um código postal puramente numérico não será uma boa idéia. É porque muitos outros países contam com códigos postais que contêm uma mistura de letras e números. E mais, os códigos postais nos Estados Unidos realmente consistem em 9 dígitos na forma # # # # #-# # # #, neste caso o hífen tornaria os dados do código postal não numéricos.

As funções de validação Bannerocity são realmente ótimas, mas o que acontecerá se o usuário ignorar as mensagens de ajuda e clicar o botão Order Banner (Pedir Banner) com dados ruins? O formulário ainda será enviado para o servidor?

Os dados ruins nunca devem ter sucesso no servidor.

Nossa! Todo o código de validação dos dados no mundo não importaria se o usuário puder evitá-lo clicando um botão ou enviando o formulário apesar das boas intenções. A falha fatal do Bannerocity é que ele não sujeita os campos do formulário à validação antes de enviar o formulário e é por isso que os dados ruins do formulário são capazes, atualmente, de ser enviados para o servidor.

Nenhuma validação dos dados importará se o usuário tiver a opção de ignorá-la e enviar o formulário com dados ruins.

Um aplicativo realmente completo também validaria os dados no servidor apenas por segurança.

O botão Order Banner precisa de alguma validação antes de transmitir os dados do formulário para o servidor.

Bannerocity precisa de outra função e seu serviço é validar todos os campos do formulário antes de enviá-lo para o servidor para o processamento. A função `placeOrder( )` personalizada é ligada ao botão Order Banner e é chamada para fazer uma ronda final da validação antes de completar o pedido.

```
<input type="button" value="Order Banner" onclick="placeOrder(this.form);" />
```

Um foco na função placeOrder( )

A função espera que um objeto do formulário seja transmitido como seu único argumento para que possa acessar as campos individuais do formulário.

Cada campo do formulário e elemento do texto de ajuda é acessado através do objeto do formulário usando a notação do array.

```
function placeOrder(form) {
  if (validateLength(1, 32, form["message"], form["message_help"]) &&
    validateZIPCode(form["zipcode"], form["zipcode_help"]) &&
    validateNonEmpty(form["date"], form["date_help"]) &&
    validateNonEmpty(form["name"], form["name_help"]) &&
    validateNonEmpty(form["phone"], form["phone_help"]) &&
    validateNonEmpty(form["email"], form["email_help"])) {
    // Envie o pedido para o servidor
    form.submit( );
  } else {
    alert("I'm sorry but there is something wrong with the order information.");
  }
}
```

A maior parte da função é uma grande instrução if/else que chama as funções de validação em cada campo do formulário.

O método submit( ) será chamado para enviar o formulário para o servidor se as campos do formulário forem validados como sendo bons.

Um problema de validação ao fazer um pedido é muito importante para justificar uma caixa de alerta.

Não existem Perguntas Idiotas

P: Como a função `placeOrder( )` controla se o formulário é enviado ou não para o servidor?

R: A princípio, a instrução `if/else` na função é estruturada para que a condicional envolva uma validação de cada campo no formulário, que significa que se qualquer dado do formulário for inválido, a cláusula `else` será executada. A cláusula `else` contém apenas uma chamada para a função `alert( )`, portanto nade mais acontecerá se a função tiver sucesso nessa cláusula. Por outro lado, se os dados forem validados como bons, o método `submit( )` será chamado no objeto `form`, que envia o formulário para o servidor. Portanto, o envio do formulário para o servidor é controlado chamando ou não o método `submit( )` do formulário. Esse método é o equivalente JavaScript de um botão de envio HTML.

P: Achei que as caixas de alerta fossem ruins para a validação dos dados. Qual é o plano?

R: Em muitos casos elas são, mas o problema real aqui é quando é bom interromper o fluxo de uma página para exibir uma mensagem pop-up (alerta), solicitar que o usuário leia uma mensagem e clique em OK. Como o botão Order Banner é clicado apenas quando o usuário tem a intenção enviar um pedido, vale a pena assegurar que ele saiba que há um problema com os dados. Portanto neste caso, o problema é grave o bastante para que um alerta seja adequado. E não se esqueça de que as mensagens de ajuda passivas são ainda exibidas para ajudar a guiar o usuário em uma correção.

O tempo é tudo... na validação da data

Infelizmente, as correções de validação do código postal e de envio do formulário de Howard fornecem apenas uma sensação passageira de alívio porque agora há um problema inteiramente novo. Ele não voa mais sobre o lugar errado graças aos códigos postais validados, mas agora ele anuncia algumas vezes os banners na data errada, o que talvez seja ainda pior. Algo que está errado com as datas do vôo fornecidas...

Um erro tipográfico resultou em uma data contendo a letra o ao invés de um 9... parece que ninguém pode digitar um 9 corretamente.

Enter the date for the message to be shown: 05/1o/2008

Howard interpretou a letra o como um zero e voou no dia 10 ao invés do dia 19.

Continue na Aventura do Boneco de palito!

Ellie não é uma cliente feliz.

Poder do cérebro

Como Howard poderia validar o campo do formulário da data do vôo para que as datas fiquem de acordo com um padrão específico, como, por exemplo, MM/DD/AAAA?

Como validar uma data

Howard aparentemente não conseguirá apenas verificar para saber se o usuário forneceu dados no campo da data – ele terá que verificar, de fato, para saber se uma data válida foi fornecida. O segredo para validar uma data é decidir por um formato de data específico e, então, aplicá-lo. Um formato de data comum envolve especificar o mês com dois dígitos, então o dia com dois dígitos, depois o ano com quatro dígitos, separados por barras.

MM/DD/YYYY

O ano consiste em quatro caracteres... detestaríamos criar um problema do tipo Y3K!

O mês e o dia consistem em dois caracteres cada.

Cada parte da data é separada por uma barra.

Enter the date for the message to be shown: 05/1o/2008

Fechar um formato com uma data é a parte fácil... no código para validar uma parte dos dados com esse formato é onde as coisas ficam complicadas. Há algumas funções de string poderosas que possibilitam dividir uma string com base em certo caractere como, por exemplo, uma barra. Mas é um feito bem complexo separar uma string em partes e, então, analisar cada uma para assegurar que seja numérica e esteja de acordo com certo comprimento. É um tipo de desafio de validação do código postal levado ao extremo.

Vejamos as etapas de como uma função de validação da data poderia funcionar:

1. Divida o valor do campo do formulário em uma coleção de substrings, usando uma barra como a base para separar a string.
2. Analise a substring do mês para assegurar que tenha exatamente dois caracteres de comprimento e que seja um número.
3. Analise a substring do dia para assegurar que tenha exatamente dois caracteres de comprimento e que seja um número.
4. Analise a substring do ano para assegurar que tenha exatamente quatro caracteres de comprimento e que seja um número.
5. Ignore qualquer outro dado após a segunda barra.

Embora esta série de etapas não seja necessariamente aterrorizante da perspectiva da codificação, parece haver muito trabalho para validar um padrão relativamente simples.

1 "05/1o/2008"

"05" "1o" "2008"

2 "05" ← Dois caracteres e um número... OK.

3 "1o" ← Dois caracteres, mas não é um número...

"2008" problema!

4 Quatro caracteres e um número... OK.

5 " " ← Nenhum dado posterior.

Não seria um sonho se houvesse uma maneira melhor de validar os dados do que dividir as strings... Uma garota pode sonhar, não pode?

As expressões regulares não são "regulares"

O JavaScript tem uma ferramenta predefinida extremamente poderosa chamada **expressão regular** que é designada especificamente para coincidir os padrões no texto. Uma expressão regular permite criar um padrão e, então, aplicá-lo em uma string de texto, pesquisando uma coincidência de modo muito parecido com um suspeito em uma formação policial... mas com figuras mais cooperativas!

Uma expressão regular é usada para coincidir os padrões no texto.

O padrão descreve as propriedades físicas de uma pessoa que são, então, coincididas com a pessoa real. As expressões regulares permitem a você fazer o mesmo tipo de coincidência de padrão com as strings.

As expressões regulares definem os padrões a coincidir

Exatamente como um padrão para uma formação policial pode envolver a altura, estilo do cabelo e outros atributos físicos de uma pessoa, um padrão de texto envolve certa seqüência de caracteres como, por exemplo, cinco números em uma linha. Espere, parece um padrão familiar... um código postal, talvez?

Padrão = # # # # #

O padrão envolve uma seqüência de cinco dígitos numéricos exatamente.

Uma coincidência!

Contém uma letra.

Poucos dígitos.

"A3492" "5280" "37205"

"007JB" "741265"

Contém letras.

Dígitos demais.

Infelizmente, transformar um padrão de cinco dígitos para um código postal em uma expressão regular não é simples. É porque as expressões regulares contam com uma sintaxe muito compacta e um pouco enigmática para descrever os padrões do texto. Não é fácil ver imediatamente se essa expressão regular pode ser usada para coincidir com um código postal de cinco dígitos:

A string deve iniciar com o padrão definido, nada diferente de um dígito é permitido.

O único dígito deve repetir 5 vezes.

Quieto. Estou tentando ver um padrão.

Padrão = /^\d{5}$/

Todas as expressões regulares ficam entre barras.

Um único dígito numérico.

A string deve terminar com este padrão.

Relaxe

Não entre em pânico se esta coisa de expressão regular parecer um pouco opressiva.

Fará muito mais sentido quando virmos alguns exemplos práticos de validação.

Expressões regulares expostas

Criar uma expressão regular é como criar uma string literal, exceto que a expressão regular aparece entre barras (//), ao invés de aspas ou apóstrofos.

/ + Expressão + /

Duas barras são usadas para conter uma expressão regular.

As expressões regulares sempre iniciam e terminam com uma barra.

Dentro da expressão em si, uma coleção de símbolos especiais, conhecidos como **metacaracteres** são usados com letras e números para criar padrões de texto altamente descritivos. O bom é que não é necessário conhecer cada nuança da "linguagem" da expressão regular para criar expressões regulares práticas. A seguir, estão alguns metacaracteres da expressão regular mais usados:

Sim, é apenas um ponto.

.

Coincide com qualquer caractere, exceto uma nova linha.

O espaço em branco inclui os espaços, tabulações, novas linhas e retornos.

\s

Coincide com um caractere de espaço em branco.

A string sendo coincidida não pode ter nenhum texto antes do padrão.

^

A string deve iniciar com o padrão.

\d

Coincide com qualquer dígito numérico.

Muitos metacaracteres iniciam com uma barra invertida... muito diferente das barras usadas para conter uma expressão regular.

\w

Coincide com qualquer caractere alfanumérico (letra ou número).

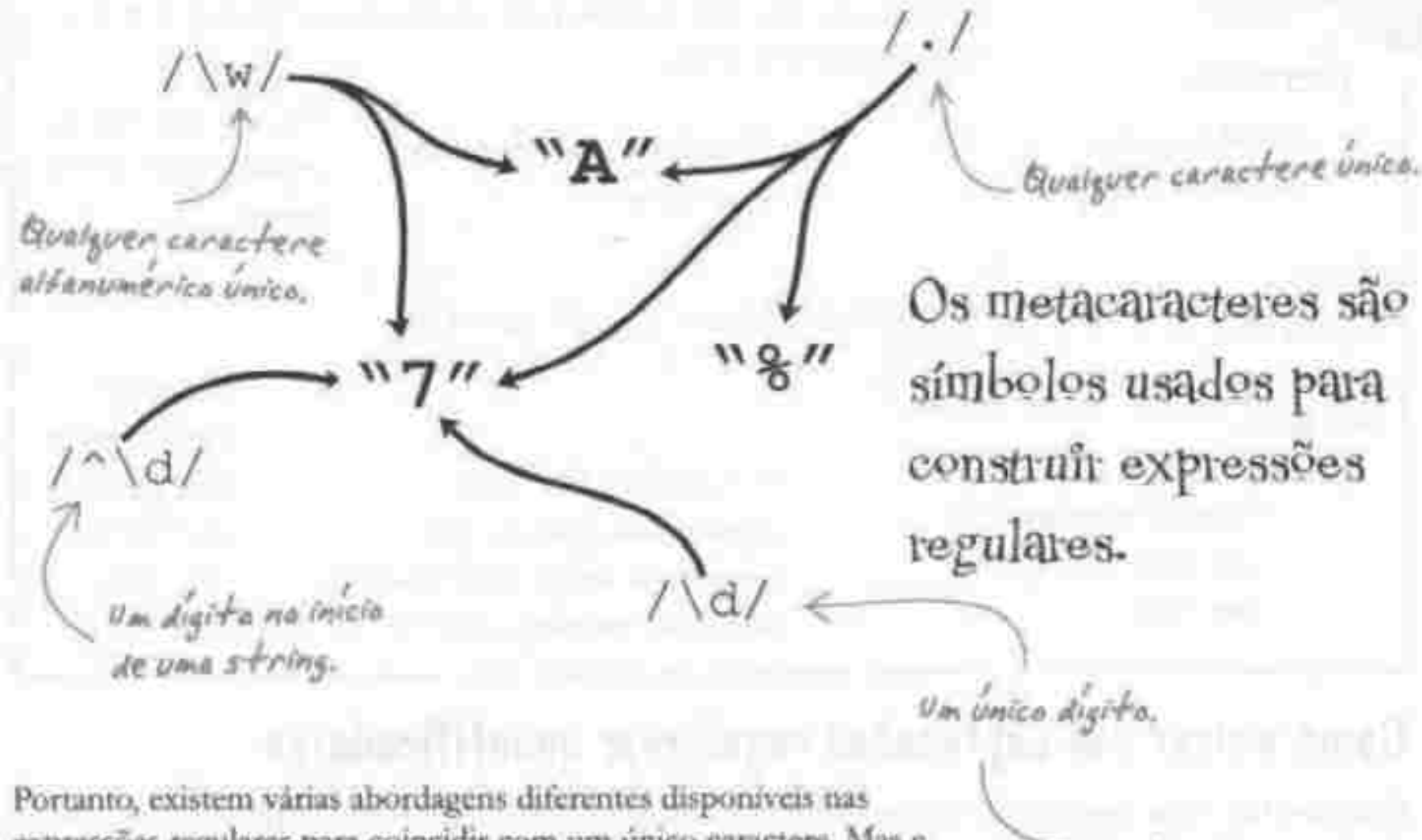
O padrão deve ser os últimos caracteres na string.

$

A string deve terminar com o padrão.

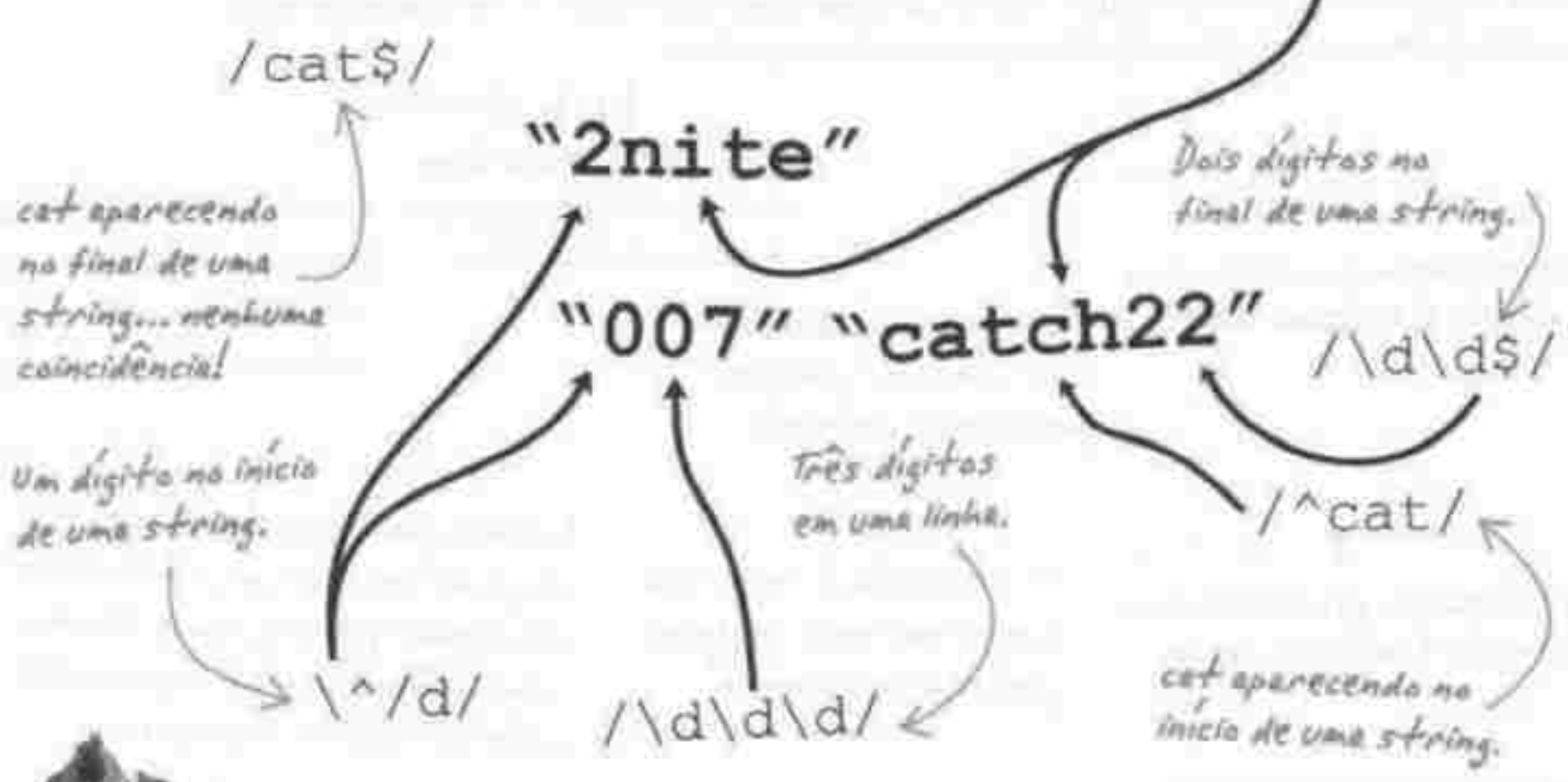
Embora estas descrições dos metacaracteres da expressão regular sejam precisas, eles são muito mais fáceis de entender quando examinados no contexto de um padrão real...

Os metacaracteres representam mais de um caractere literal



Os metacaracteres são símbolos usados para construir expressões regulares.

Portanto, existem várias abordagens diferentes disponíveis nas expressões regulares para coincidir com um único caractere. Mas e as strings que contêm mais de um caractere? A seguir, estão mais algumas situações práticas de coincidência de padrões:



Escreva uma expressão regular que coincida com um código postal completo dos Estados Unidos, com a forma # # # # #-# # # # e deva aparecer por si só.

. . . . . .

Exercício

Escreva uma expressão regular que coincida com um código postal completo dos Estados Unidos, com a forma # # # # #-# # # # e deva aparecer por si só.

A string deve iniciar com o padrão.

A string deve terminar com o padrão.

Feche a expressão regular com outra barra.

Inicie a expressão regular com uma barra.

```
/^\d\d\d\d\d-\d\d\d\d$/
```

Coincide exatamente com cinco dígitos em uma linha.

O hífen não tem nenhum significado especial aqui... é apenas um hífen separando os números no código postal.

Coincide exatamente com quatro dígitos em uma linha.

Como entrar nas expressões regulares: quantificadores

Qualquer texto que não seja um metacaractere é coincidido como está em uma expressão regular, significa que /howard/ coincide com o texto "howard" em qualquer parte de uma string. E mais, há algumas outras construções de expressão regular chamadas de **quantificadores** que aperfeiçoam mais os padrões. Os quantificadores são aplicados nos subpadrões que os precedem dentro de uma expressão regular e fornecem controle sobre quantas vezes um subpadrão aparece em um padrão.

{*n*}

O subpadrão precedente deve aparecer exatamente **n** vezes em uma linha.

Controla exatamente quantas vezes um subpadrão pode aparecer.

*****

O subpadrão precedente deve aparecer **0 ou mais vezes.**

O subpadrão é opcional e pode aparecer qualquer quantidade de vezes.

+

O subpadrão precedente deve aparecer **1 ou mais vezes.**

O subpadrão é requerido e pode aparecer qualquer quantidade de vezes.

Embora não sejam tecnicamente um quantificador, os parênteses são usados para agrupar tantos subpadrões quantas expressões matemáticas você agrupar.

?

O subpadrão precedente deve aparecer 0 ou 1 vez.

O subpadrão é opcional, mas poderá aparecer apenas uma vez, se aparecer.

( )

Agrupe os caracteres e/ou metacaracteres em um subpadrão.

Quantificação do padrão

Os quantificadores permitem que as expressões regulares sejam escritas de modo muito mais consistente do que com metacaracteres apenas. Ao invés de repetir explicitamente os subpadrões, um quantificador pode especificar exatamente quantas vezes um subpadrão pode aparecer. Como exemplo, o seguinte padrão coincide com um código postal usando quantificadores:

```
/^\d{5}-\d{4}$/
```

Com a ajuda do quantificador { }, não é mais necessário listar cada dígito.

Os quantificadores controlam o número de vezes em que um subpadrão aparece em uma expressão regular.

É possível ser muito criativo com os metacaracteres e os quantificadores para criar expressões regulares bem poderosas que coincidem com praticamente qualquer coisa que possa aparecer em uma string.

```
/\w*/
```

Coincide com qualquer quantidade de caracteres alfanuméricos, inclusive com uma string vazia!

```
/.+/
```

Qualquer caractere deve aparecer uma ou mais vezes... coincide com uma string preenchida.

```
/(Hot)? ?Donuts/
```

Coincide com Donuts ou Hot Donuts.

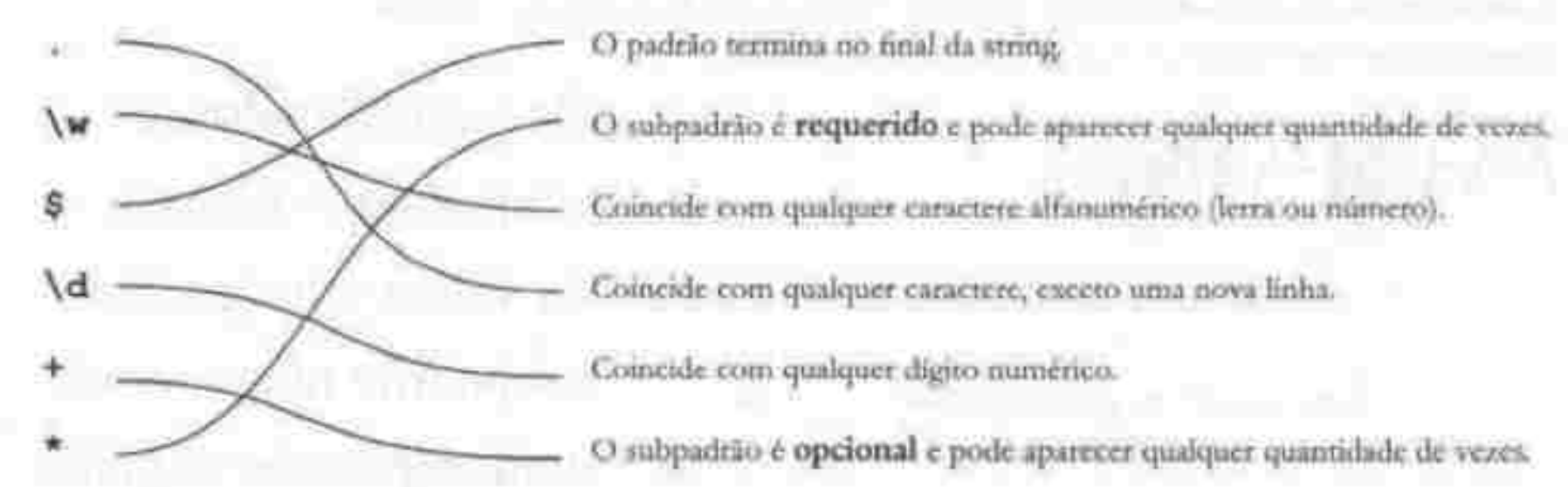
QUAL É MINHA FINALIDADE?

Coincida qualquer metacaractere ou quantificador da expressão regular com o que ele faz dentro de um padrão.

.	O padrão termina no final da string.
\w	O subpadrão é **requerido** e pode aparecer qualquer quantidade de vezes.
$	Coincide com qualquer caractere alfanumérico (letra ou número).
\d	Coincide com qualquer caractere, exceto uma nova linha.
+	Coincide com qualquer dígito numérico.
*	O subpadrão é **opcional** e pode aparecer qualquer quantidade de vezes.

QUAL É MINHA FINALIDADE?

Coincida qualquer metacaractere ou quantificador da expressão regular com o que ele faz dentro de um padrão.



Não existem Perguntas Idiotas

P: Uma expressão regular é uma string?

R: Não. Você pode considerar uma expressão regular mais como uma **descrição** de uma string ou pelo menos, uma descrição de parte de uma string. As expressões regulares estão intimamente ligadas às strings e são usadas para coincidir os padrões de texto que aparecem nas strings, mas não são strings em si.

P: As expressões regulares podem ser aplicadas em outros tipos de dados?

R: Não, as expressões regulares são designadas dentro de uma string de texto, portanto se aplicam apenas às strings. Mas isso não as impede de ser extremamente úteis ao executar tarefas complexas de coincidência de texto que seriam muito difíceis usando strings apenas.

P: O que acontecerá se você quiser coincidir com um metacaractere como, por exemplo, um cifrão?

R: Parecidos com as strings JavaScript, os caracteres com significado especial nas expressões regulares podem ter o escape aplicado com uma barra invertida. Portanto, para coincidir com um cifrão dentro de uma expressão regular, você especificaria o caractere $ como \$. Essa mesma regra aplica-se a qualquer outro caractere que tenha um significado especial dentro de uma expressão regular como, por exemplo, ^, * e +, para citar alguns. Qualquer caractere que não tenha um significado especial pode ser colocado diretamente em uma expressão regular sem uma formatação especial.

P: As expressões regulares têm alguma relação com a validação dos dados? Originalmente, não estávamos tentando validar um campo do formulário de data no Bannerocity?

R: Ah, paciência, jovem Jedi. Logo, as expressões regulares, você usará. Sim, a razão para este pequeno desvio nas expressões regulares é elaborar um modo de validar os dados com um formato complexo, como uma data ou um endereço de e-mail. O Bannerocity ainda precisa de muita ajuda na frente de formatação dos dados, portanto haverá muitas oportunidades para aplicar as expressões regulares. Apenas continue tentando um pouco mais.

P: Como as expressões regulares são usadas no JavaScript?

R: Estamos chegando lá... mesmo! As expressões regulares são representadas no JavaScript por um objeto, que suporta vários métodos para usar as expressões regulares para coincidir com os padrões dentro das strings.

Mestre dos padrões

Entrevista desta semana: A expressão regular misteriosa, porém poderosa

Use a Cabeça: Então, foi sobre você que ouvi falar da capacidade de pesquisar as strings e conhecer os padrões. É verdade?

Expressão regular: Sim, sou um tipo de divisor de código, capaz de ver uma string de texto e separar os padrões imediatamente. A CIA realmente poderia usar uma pessoa como eu... mas eles não retornaram meus telefonemas.

Use a Cabeça: Então, você tem um interesse em espionagem?

Expressão regular: Não, apenas adoro procurar padrões no texto. Na verdade, **sou** um padrão, qualquer padrão. Apenas me dê alguns parâmetros sobre o que está procurando e eu encontrarei ou, pelo menos, deixarei que você saiba se existe ou não em uma string.

Use a Cabeça: Parece ótimo, mas o método indexOf( ) do objeto String já não pode lidar com a pesquisa da string?

Expressão regular: Por favor, diga-me que você não foi até ele... aquele amador não sabe o básico sobre os padrões. Olhe, se precisar de um recurso de pesquisa muito simples que procura somente a palavra "defeituoso" em uma string, então indexOf( ) será sua resposta. Do contrário, perceberá rapidamente que indexOf( ) não faz nada sério ao analisar as strings.

Use a Cabeça: Mas uma pesquisa de string não é uma forma de coincidência de padrão?

Expressão regular: Sim e ir até a caixa de correio é uma forma de exercício, mas você não a vê nas Olimpíadas... ainda. O que quero dizer é que uma simples pesquisa de string é realmente a forma mais simplista de coincidência de padrões possível – o padrão é uma palavra ou frase estática. Agora, considere algo como uma data ou um URL do Web site. Esses são padrões verdadeiros porque, embora sigam formatos rígidos, as particularidades do que está sendo pesquisado não são estáticas.

Use a Cabeça: Acho que entendo o que você quer dizer. Um padrão é uma **descrição** do texto que pode aparecer em uma string, mas não necessariamente o texto em si?

Expressão regular: Exatamente. É como se eu pedisse a você para me avisar quando uma pessoa que é alta, tem cabelo curto e não usa óculos passa. Isso é uma descrição de uma pessoa, mas não uma pessoa real. Se um cara chamado Alan passar coincidindo com essa descrição, poderemos dizer que o padrão foi coincidido. Mas poderia haver muitas outras pessoas que também coincidem com essa descrição. Sem os padrões, não seríamos capazes de procurar uma pessoa com base em uma descrição – teríamos que procurar uma pessoa real. Portanto, a diferença entre pesquisar uma parte específica do texto usando indexOf( ) e coincidir com um padrão ao me usar é a diferença entre procurar Alan ou procurar uma pessoa que é alta, tem cabelo curto e não usa óculos.

Use a Cabeça: Isso faz sentido agora. Mas como a coincidência de padrões aplica-se à validação dos dados?

Expressão regular: Bem, validar os dados é basicamente assegurar que os dados encaixam-se em certo formato pré-concebido ou padrão. Portanto, meu trabalho é obter um padrão e ver se uma string de texto está de acordo com ele. Se estiver, então os dados serão considerados válidos. Do contrário, os dados serão ruins.

Use a Cabeça: Portanto, há uma expressão regular diferente para coincidir com os diferentes tipos de dados?

Expressão regular: Ah, sim. E é realmente onde grande parte do trabalho ocorre ao me usar para validar os dados – propor uma expressão regular que modela com sucesso um formato de dados.

Use a Cabeça: Isso é muito importante. Fico grato por explicar o papel que desempenha na validação dos dados.

Expressão regular: Tudo bem. Geralmente, tenho que explicar para mim mesmo... acho que é um padrão comportamental.

Como validar os dados com expressões regulares

Por mais sensacional que possa parecer criar expressões regulares puramente para ver os padrões no texto, temos uma necessidade premente das expressões regulares para ajudar a validar o campo de data no Bannerocity e fazer com que Howard volte a voar. Uma expressão regular no JavaScript é representada pelo objeto RegExp, que inclui um método chamado `test( )` que é a chave para usar as expressões regulares para a validação dos dados. O método `test( )` verifica a existência de um padrão dentro de uma string.

O método test( ) do objeto RegExp é usado para testar uma string para obter um padrão da expressão regular.

A expressão regular coincide com um código postal com 5 dígitos.

Este objeto literal cria automaticamente um objeto RegEx.

O valor de um campo de entrada, uma string, é transmitido para o método.

```
var regex = /^\d{5}$/;

if (regex.test(inputField.value)
    // o código postal é inválido!
```

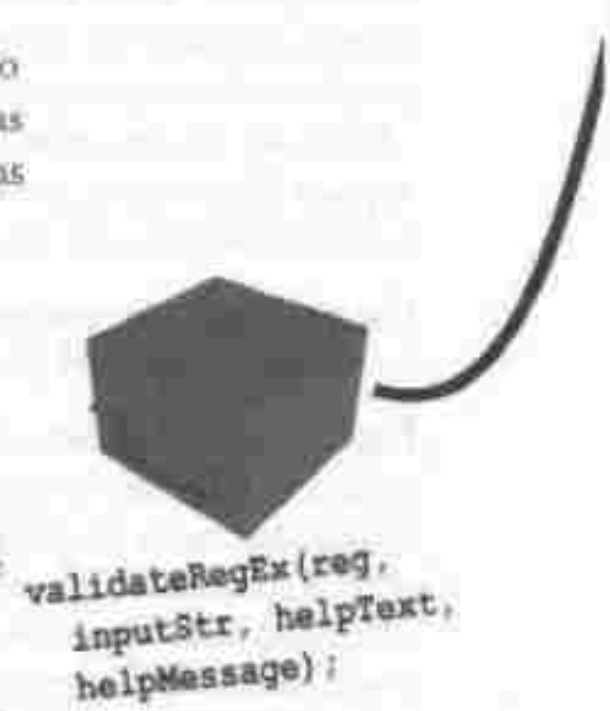
Se o método test( ) retornar false, o padrão falhou e os dados são inválidos.

O método test( ) é chamado no objeto da expressão regular.

O valor de retorno de test( ) será true se o padrão coincidir com a string ou false, do contrário.

Embora possamos fazer uma chamada para o método `test( )` dentro de cada função de validação diferente, há uma oportunidade para criar uma função de validação geral baseada em expressões regulares que funções mais específicas poderão chamar para fazer o trabalho genérico de validação. As seguintes etapas devem ser executadas pela função `validateRegEx( )` geral:

1. Execute um teste na expressão regular que é transmitida como um argumento, usando a string de entrada que também é transmitida.
2. Se o padrão coincidir, defina a mensagem de ajuda para o texto de ajuda que é transmitido como um argumento e então, retorne `false`.
3. Se o padrão não coincidir, limpe a mensagem de ajuda e retorne `true`.



Agora, tudo que está faltando é o código da função, que como resultado, não é tão ruim. Na verdade, grande parte desse código já apareceu nas outras funções de validação, portanto `validateRegEx( )` é tanto para a reutilização do código quanto para a criação de um validador geral de expressões regulares.

A expressão regular, a string de entrada, o texto da mensagem de ajuda e o elemento da mensagem de ajuda são transmitidos como argumentos.

A expressão regular é testada com a string de entrada.

```
function validateRegEx(regex, inputStr, helpText, helpMessage) {
  // Veja se os dados inputStr são validados como sendo bons
  if (!regex.test(inputStr)) {
    // Os dados são inválidos, portanto defina a mensagem de ajuda e retorne
    // false
    if (helpText != null)
      helpText.innerHTML = helpMessage;
    return false;
  }
  else {
    // Os dados são bons, portanto limpe a mensagem de ajuda e retorne true
    if (helpText != null)
      helpText.innerHTML = "";
    return true;
  }
}
```

Se o teste falhar, os dados serão inválidos, portanto o texto da mensagem de ajuda será exibido.

Se os dados forem verificados com sendo bons, a mensagem de ajuda será limpa

Aponte seu Lápis

Escreva o código para a função `validateDate( )`, que chama `validateNonEmpty( )` e `validateRegEx( )` para validar o campo do formulário de data no Bannerocity. Sugestão: A função aceita dois argumentos, o campo de entrada da data e seu elemento do texto de ajuda afim.

Aponte seu Lápis
Solução

Escreva o código para a função `validateDate( )`, que chama `validateNonEmpty( )` e `validateRegEx( )` para validar o campo do formulário de data no Bannerocity. Sugestão: A função aceita dois argumentos, o campo de entrada da data e seu elemento do texto de ajuda afim.

A função validateNonEmpty() é chamada primeiro para assegurar que o campo não está vazio.

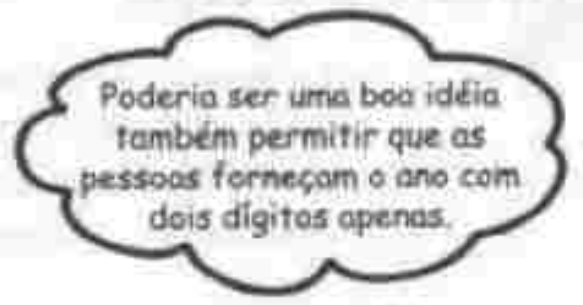
```
function validateDate(inputfield, helptext) {
  // Primeiro, veja se o valor da entrada carece disso
  if (!validateNonEmpty(inputfield, helptext))
    return false;

  // Então, veja se o valor de entrada é uma data
  return validateRegEx(/^[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}$/, inputfield.value, helptext,
    "Please enter a date (for example, 12/34/5678).");
}
```

A expressão regular é transmitida para a função validateRegEx().

Como as barras têm um significado especial nas expressões regulares, elas têm que ter o escape aplicado com uma barra invertida.

A expressão regular da data usa metacaracteres e quantificadores para aplicar o formato AAA/DD/AAAA.

É questionável se seus scripts ainda serão usados no ano 2100.



O ano 2100 está muito distante...

Saber que não mudaremos os séculos novamente por algum tempo, provavelmente estará tudo bem permitir que os usuários forneçam o ano como dois dígitos ao invés de quatro. Na realidade, é improvável que qualquer código JavaScript escrito hoje sobreviva 90 ou mais anos para que apresentem problemas. Howard considerou rapidamente uma abordagem cuidadosa para assegurar o Bannerocity no futuro ficando com os anos de quatro dígitos e, então, decidiu que pode conviver com um ajuste posterior, caso o código ainda esteja em uso na virada do próximo século.

Não existem Perguntas Idiotas

P: Por que é necessário chamar a função `validateNonEmpty()` em `validateDate()`? A expressão regular já não decompõe os dados vazios?

R: Sim, a expressão regular valida inerentemente a data para assegurar que ela não está vazia e a validação de preenchimento poderia ser removida, e a data ainda seria validada muito bem. Porém, verificando primeiro para saber se os dados foram fornecidos, a página torna-se mais clara para o usuário, oferecendo mensagens de ajuda específicas com base em um determinado problema de validação. Se nenhum dado tiver sido fornecido, uma mensagem diferente será exibida e, então, caso uma data inválida tenha sido fornecida. O resultado final é um sistema de ajuda passivo que se comporta como um guia para o usuário no preenchimento do formulário. Esta melhoria de utilização sutil parece ser uma troca valiosa para a pouca quantidade de código extra requerida.

P: E se eu realmente quiser assegurar o código de meu script no futuro? Será um problema?

R: Não, de forma alguma. Raramente é um problema tentar antecipar as futuras necessidades e escrever um código que possa adaptar-se a essas necessidades. No caso do Bannerocity, um campo de data com quatro dígitos é certamente mais seguro no futuro do que uma versão com dois. E lembre-se que se você realmente quisesse ser esperto, poderia permitir ao usuário fornecer apenas dois dígitos e, então, anexar o número do século a esses dígitos internamente. Portanto, o efeito no formulário será de um ano com dois dígitos, mas a data será realmente armazenada como um ano com quatro dígitos.

Como coincidir os mínimos e os máximos

O quantificador {} aceita um número que determina quantas vezes um subpadrão pode aparecer em uma string. Há outra versão desse quantificador que requer dois números, que especifica o número mínimo e máximo de vezes em que um subpadrão pode aparecer. Isso fornece uma maneira prática de ajustar a presença de um subpadrão.

Preste atenção! *Não é necessariamente seguro supor que todos os usuários sintam-se à vontade fornecendo datas como MM/DD/AAAA. Muitos lugares no mundo invertem os meses e os dias para que o formato seja DD/MM/AAAA.*

Nem todas as datas estão no formato MM/DD/AAAA.

`{min,max}`

O subpadrão anterior deve aparecer pelo menos **min** vezes em uma linha, porém não mais que **max** vezes.

Controle quantas vezes um subpadrão pode aparecer como uma faixa mínima e máxima.

```
/^\w{5,8}$/
```

Algumas senhas permitem entre cinco e oito caracteres alfanuméricos, o que é perfeito para o quantificador mín/máx { }.

Aponte seu Lápis

Reescreva a expressão regular usada na função `validateDate( )` para que ela permita anos com 2 e 4 dígitos.

.....

Aponte seu Lápis Solução

Reescreva a expressão regular usada na função `validateDate( )` para que ela permita anos com 2 e 4 dígitos.

```
/^\d{2}\/\d{2}\/\d{2,4}$/
```

A versão min/max do quantificador { }, define o número mínimo e máximo dos dígitos do ano permitidos na data.

Pontos de Bala

- A expressão regular coincide com um **padrão** de texto em uma string e fica entre barras.
- Além do texto normal, as expressões regulares são compostas de metacaracteres e quantificadores, que fornecem um controle cuidadoso sobre como um padrão de texto é coincidido.
- No JavaScript, as expressões regulares são suportadas pelo objeto `RegExp` predefinido, mas ele é raramente visto porque as expressões regulares são geralmente criadas como literais.
- O método `test( )` no objeto `RegExp` é usado para testar um padrão da expressão regular em uma string de texto.

Nenhuma história revisionista pode acrescentar ao JavaScript suporte para os primeiros dez séculos.

E como não foi suportado no passado, não há nenhuma razão para permitir que os usuários forneçam a parte do ano de uma data em centenas. Na verdade, não há nenhuma necessidade de permitir que os usuários peçam banners aéreos em nenhum ponto no passado, caso possamos possivelmente evitar. Portanto, eliminar os anos com 3 dígitos do código de validação é uma correção importante e ajudará a evitar que Howard enfrente um monte de novos problemas de entrada dos dados Bannerocity.

Como eliminar os anos com três dígitos com isto... ou aquilo

Outro metacaractere muito útil na caixa de ferramentas da expressão regular é o metacaractere de alternação, que parece e funciona de modo muito parecido com o operador OR no JavaScript. Diferente do operador OR JavaScript, o metacaractere de alternação envolve apenas uma barra vertical, mas permite um padrão para especificar uma lista de subpadrões alternativos. Em outras palavras, o padrão terá uma coincidência bem-sucedida se qualquer um dos subpadrões alternativos coincidir. É muito parecido com o operador OR lógico porque basicamente está dizendo: "este ou este ou este...".

O metacaractere alternativo fornece um modo prático de especificar as coincidências alternativas.

`este|aquele`

O padrão coincidirá se o subpadrão **este ou** o subpadrão **aquele** coincidir.

`/pequeno|médio|grande/`

Diversas possibilidades podem ser especificadas usando mais de um metacaractere alternativo.

`/(vermelha|azul) pílula/`

Uma escolha simples entre duas coisas; este padrão coincide com pílula vermelha e pílula azul.

Aponte seu Lápis

Reescreva a expressão regular usada na `função validateDate( )` mais uma vez e, dessa vez, certifique-se de que o ano possa apenas ter 2 ou 4 dígitos, e nada mais.

.....

Aponte seu Lápis
Solução

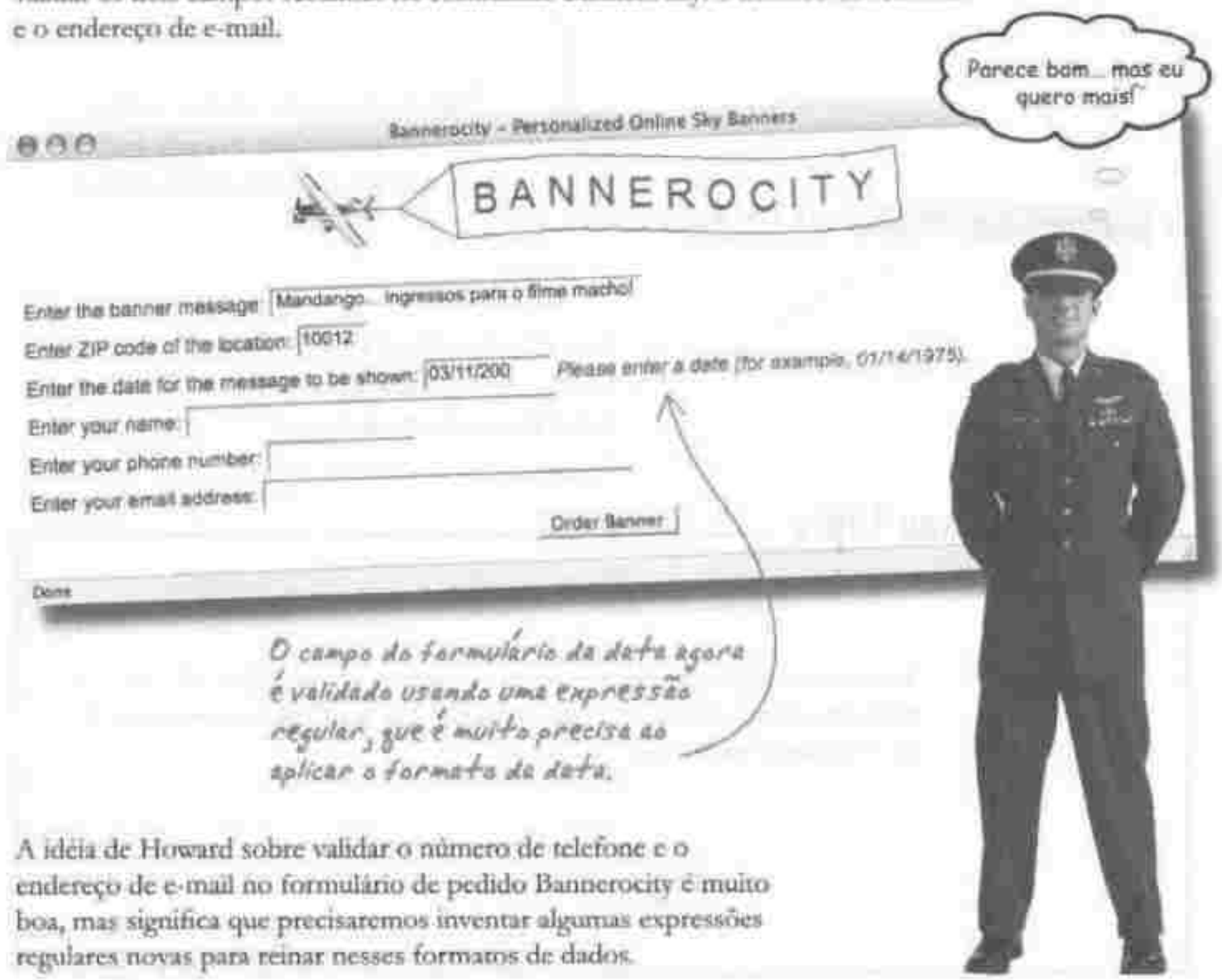
Reescreva a expressão regular usada na função `validateDate( )` mais uma vez e agora, certifique-se de que o ano possa apenas ter 2 ou 4 dígitos, e nada mais.

```
/^\d{2}\/\d{2}\/(\d{2}|\d{4})$/
```

O metacaractere de alternação (|) permite que o padrão aceite anos com 2 e 4 dígitos.

Não confie na sorte

Howard realmente gosta do novo validador de data robusto que conta com expressões regulares para obter uma coincidência de padrões precisa. Na verdade, ele gosta tanto do validador que deseja avançar e usar as expressões regulares para validar os dois campos restantes no formulário Bannerocity: o número de telefone e o endereço de e-mail.



O campo do formulário da data agora é validado usando uma expressão regular, que é muito precisa ao aplicar o formato da data.

A idéia de Howard sobre validar o número de telefone e o endereço de e-mail no formulário de pedido Bannerocity é muito boa, mas significa que precisaremos inventar algumas expressões regulares novas para reinar nesses formatos de dados.

Você pode me ouvir agora? Validação do número de telefone

De uma perspectiva da validação, os números de telefone não são terrivelmente difíceis de compreender, pois eles seguem um formato rígido. Naturalmente, sem as expressões regulares, eles ainda envolveriam uma boa quantidade de experimentação de strings, mas as expressões regulares tornam os números de telefone fáceis de validar. Os números de telefone nos Estados Unidos seguem o seguinte padrão:

Como Howard não planeja voar fora de sua área local, é seguro adotar um formato de número de telefone nos Estados Unidos.

Padrão = # # #-# # #-# # # #

Mudando os traços no padrão do número de telefone para barras e ajustando o número de dígitos, fica claro que o padrão do número de telefone é muito parecido com o padrão da data.

O padrão da data segue uma data como MM/DD/AAAA ou MM/DD/AA, usando o metacaractere \d e o quantificador { }:

```
/^\d{2}\/\d{2}\/\d{2,4}$/
```

O padrão do número de telefone é parecido com o padrão da data, exceto que ele usa hífens para separar um número diferente de dígitos.

```
/^\d{3}-\d{3}-\d{4}$/
```

A função `validatePhone( )` torna-se bem previsível graças à expressão regular do número de telefone e à função `validateRegEx( )`.

```
function validatePhone(inputField, helpText) {
  // Primeiro veja se o valor de entrada contém dados
  if (!validateNonEmpty(inputField, helpText))
    return false;

  // Então, veja se o valor de entrada é um número de telefone
  return validateRegEx(/^\d{3}-\d{3}-\d{4}$/,
    inputField.value, helpText,
    "Please enter a phone number (for example, 123-456-7890).");
}
```

Você recebeu uma correspondência: como validar o e-mail

Com a tarefa de validação do número de telefone fechada, o desafio final de Howard é validar o campo do endereço de e-mail no formulário Bannerocity. Como qualquer outro dado, o segredo para validar um endereço de e-mail é dividir o formato em um padrão consistente que possa ser modelado usando uma expressão regular.

2 ou 3 caracteres alfanuméricos.

Padrão = *NomeLocal@PrefixoDomínio.SufixoDomínio*

Alfanumérico.

Não parece muito ruim – um endereço de e-mail tem apenas três partes de texto alfanumérico com um símbolo (@) e um ponto incluído.

Todos esses endereços de e-mail seguem o padrão do e-mail. Nosso trabalho acaba aqui... ou não?

howard@bannerocity.com

sales@duncansdonuts.com

puzzler@youcube.ca

Criar uma expressão regular para coincidir com este padrão de e-mail é bem simples, considerando que tudo é muito previsível.

O endereço de e-mail deve iniciar com um ou mais caracteres alfanuméricos.

O ponto deve ter o escape aplicado, uma vez que é um caractere especial nas expressões regulares.

```
/^\w+@\w+\.\w{2,3}$/
```

Depois do símbolo @, um ou mais caracteres alfanuméricos aparecem.

O endereço de e-mail deve terminar com 2 ou 3 caracteres alfanuméricos.

Embora este padrão faça o serviço, algo parece faltar. Todos os endereços de e-mail realmente seguem tal formato previsível?

Poder do cérebro

Quais outras variações do padrão de e-mail são possíveis? Considere todos os endereços de e-mail diferentes que você já viu.

A exceção é a regra

Os endereços de e-mail são realmente mais complexos do que parecem ser à primeira vista. Existem algumas variações no formato básico do e-mail que devem ser consideradas ao formular um padrão de e-mail confiável para a validação dos dados. Eis alguns exemplos de endereços de e-mail perfeitamente válidos:

`cube_lovers@youcube.ca`

Sublinhado no nome local.

`aviator.howard@bannerocity.com`

Ponto no nome local.

`rocky@i-rock.mobi`

Hífen no prefixo do nome do domínio.

Quatro caracteres no sufixo do domínio.

`i-love-donuts@duncansdonuts.com`

Hífens no nome local.

Opa, precisamos realmente de um modo de coincidir os caracteres opcionais para validar os endereços de e-mail.

`seth+jason@mandango.us`

Sinal de mais no nome local.

`ruby@youcube.com.nz`

Sufixos extras do domínio, que são realmente apenas pontos extras no nome do domínio.

Os endereços de e-mail apresentam a necessidade de coincidir os caracteres opcionais em um padrão.

Como resultado, existem vários caracteres opcionais diferentes que podem estar espalhados nas partes de um endereço de e-mail que lidamos anteriormente como puramente alfanuméricos. Precisamos de uma maneira de incorporar tais caracteres opcionais em um padrão...

Como coincidir caracteres opcionais a partir de um conjunto

Outro recurso regular muito útil que afeta diretamente o padrão do endereço de e-mail é o de **classes de caracteres**, que permite criar subpadrões muito controlados dentro de um padrão. Mais especificamente, as classes de caracteres primam pelo estabelecimento de regras onde os caracteres opcionais favorecem muito um subpadrão. Você pode considerar uma classe de caractere como um conjunto de regras para coincidir com um único caractere.

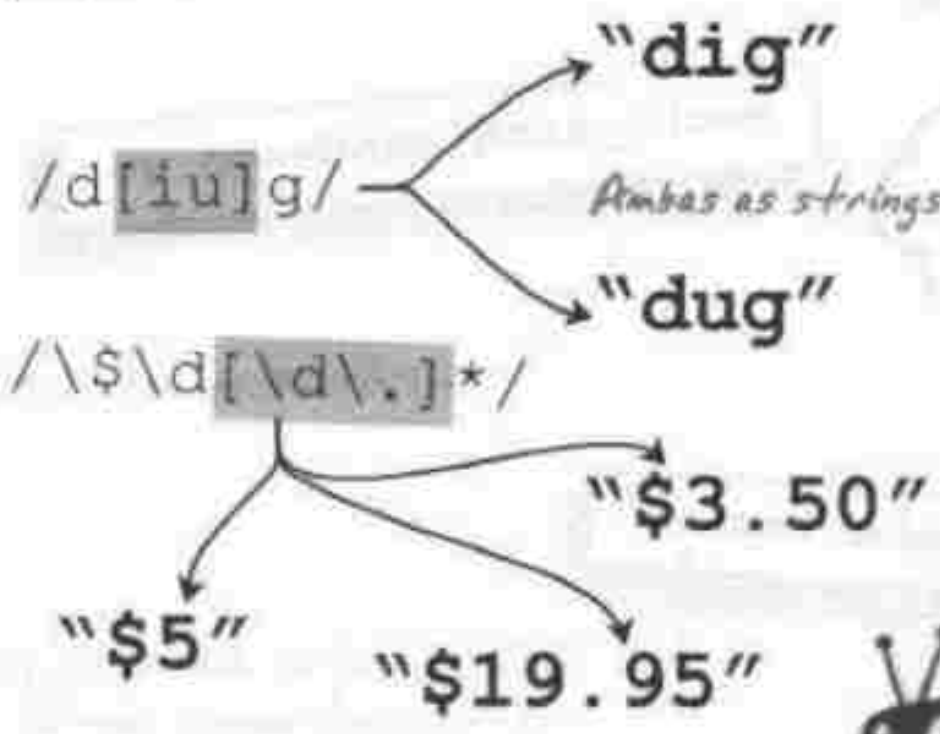
```
[ClasseCaractere]
```

ClasseCaractere é um conjunto de regras de expressão regular para coincidir com um único caractere.

As classes do caractere estão sempre entre colchetes.

Dentro de uma classe do caractere, todo caractere listado é considerado legal para a coincidência de caracteres, como um tipo de alternação entre os metacaracteres que permite construir uma lista de subpadrões alternativos. Porém, o resultado de uma classe de caractere é sempre uma coincidência para um único caractere, a menos que a classe de caractere seja seguida de um quantificador. Alguns exemplos ajudarão a colocar as classes do caractere em "perspectiva".

As classes do caractere oferecem um modo eficiente de controlar os caracteres opcionais em um padrão de expressão regular.



Ambas as strings são coincidências do padrão.

Todas estas strings financeiras coincidem com o padrão.

Não se esqueça de aplicar o escape nos caracteres especiais nas expressões regulares.

Preste atenção! *Os caracteres que têm um significado especial nas expressões regulares devem ter o escape aplicado para incluir o caractere real em uma expressão regular. Aplique o escape em um dos seguintes caracteres precedendo-o com uma barra invertida (\): [\^$.|?*+().*

As classes de caracteres são exatamente o que precisamos para colocar em forma o padrão do endereço de e-mail e adicionar a validação do e-mail ao Bannerocity...

Como construir um validador de e-mails

Agora, é possível criar um padrão muito mais robusto para os endereços de e-mail decompondo todos os possíveis caracteres opcionais que podem aparecer no nome local e no nome do domínio.

Esta parte do endereço de e-mail pode aparecer uma ou mais vezes.

Padrão = `NomeLocal@PrefixoDominio.SufixoDominio`

Qualquer alfanumérico, assim como ., -, _ e +.

Qualquer alfanumérico, assim como -.

O ponto é considerado parte do sufixo do domínio.

2, 3 ou 4 caracteres alfanuméricos, precedidos de um ponto.

Lembre-se de que há muitos modos diferentes de abordar a criação de padrões, inclusive o padrão do endereço de e-mail. Pode ser surpreendentemente difícil criar um padrão que enderece com sucesso cada pequena nuança de um determinado formato de dados. Já experimentamos uma vez como a construção geral do padrão é elaborada, traduzi-la em uma expressão regular real é bem simples.

Aponte seu Lápis

Termine o código que falta para a função `validateEmail( )`, que é usada para validar um endereço de e-mail no Bannerocity:

```
function validateEmail(inputField, helpText) {
  // Primeiro veja se o valor de entrada contém dados
  if (!.......................................................... inputField, helpText)
    return false;

  // Então veja se o valor de entrada é um endereço de e-mail
  return validateRegEx(.............................................................
    inputField.value, helpText,
    /.....................................................................................);
}
```

Aponte seu Lápis
Solução

Termine o código que falta para a função `validateEmail( )`, que é usada para validar um endereço de e-mail no Bannerocity.

A expressão regular de e-mail usa a maioria dos truques da expressão regular que aprendemos para validar um endereço de e-mail.

```
function validateEmail(inputField, helpText) {

  // Primeiro veja se o valor de entrada contém dados
    validateNonEmpty
  if (!.......................................... (inputField, helpText)

    return false;

  // Então veja se o valor de entrada é um endereço de e-mail
    /^[\w\.-_\+]+@[\w-]+(\.\w{2,4})+$/
  return validateRegEx(.........................................................
    inputField.value, helpText,
      "Please enter an email address (for example, johndoe@acme.com)."
  }.........................................................................)
}
```

A função validateNonEmpty( ) ainda é chamada inicialmente para verificar a falta de dados.

O nome local pode ser alfanumérico, assim como ., -, _, +, e deve estar no início da string.

Se a validação do e-mail falhar, uma mensagem de ajuda será exibida para esclarecer o formato de entrada com um exemplo.

O sufixo do nome do domínio pode ter de 2 a 4 caracteres alfanuméricos no final da string.

Um formulário Bannerocity a toda prova

A coleção de dados do pedido de banner aéreo no Bannerocity agora é pura perfeição graças a alguns esforços intensos de validação. Howard está tão entusiasmado que decidiu voar com um anúncio de banner próprio.

A validação dos dados é boa!

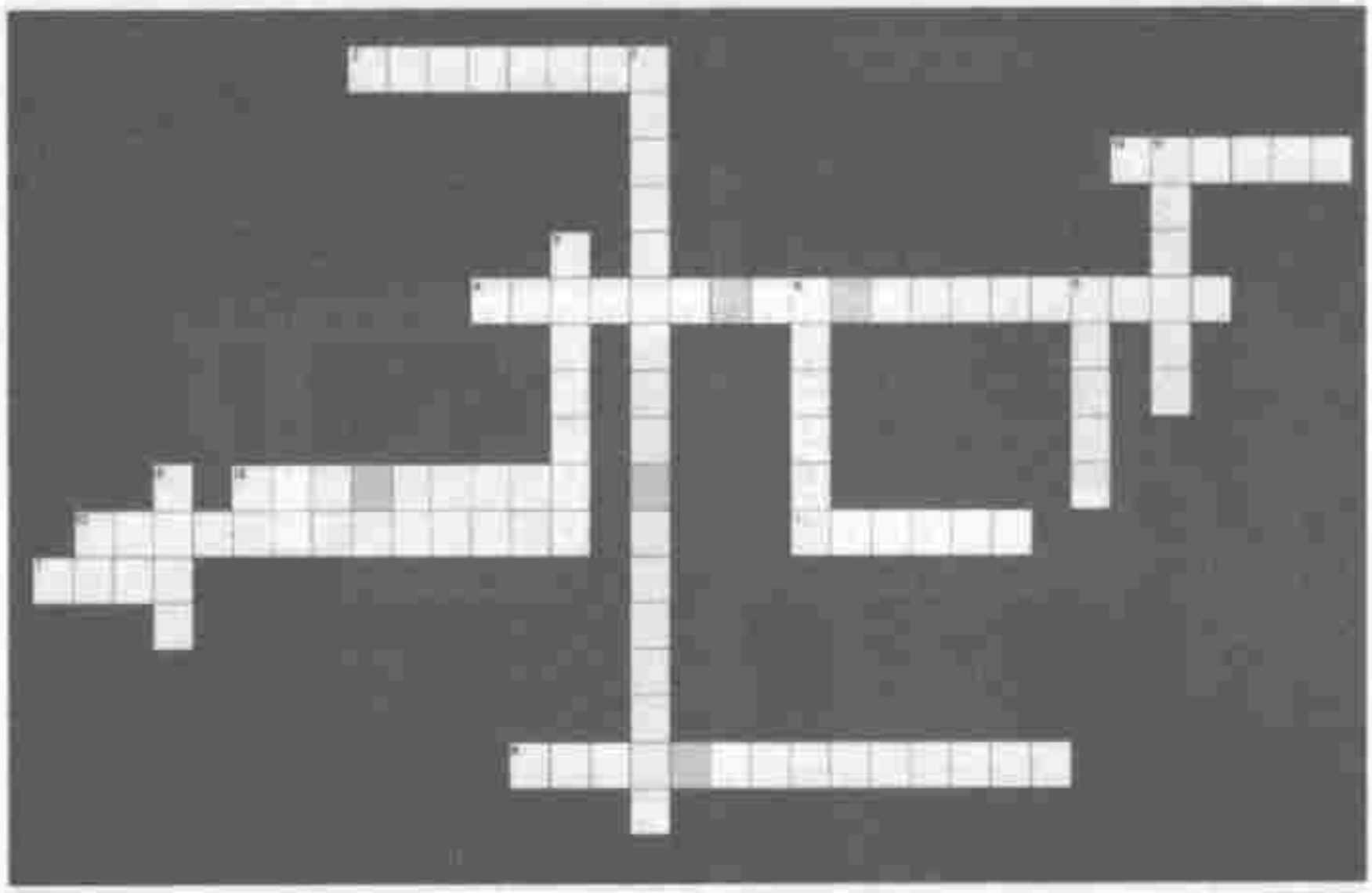
Howard está vibrando por voltar a fazer o que adora... voar!

Enter the banner message: Mandango...ingressos... lime macho!
Enter ZIP code of the location: 10012
Enter the date for the message to be shown: 03/11/2009
Enter your name: Seth Tinselman
Enter your phone number: (212) 555-5339 Please enter a phone number (for example, 123-456-7890).
Enter your email address: seth@mandango Please enter an email address (for example, johndoe@acme.com).
Order Banner

Os campos do número de telefone e do endereço de e-mail agora são validados segundo formatos de dado muito rígidos.

Cruzadas do JavaScript

Eis um padrão que você pode reconhecer... uma palavra cruzada! Nenhuma validação é requerida – apenas algumas respostas.



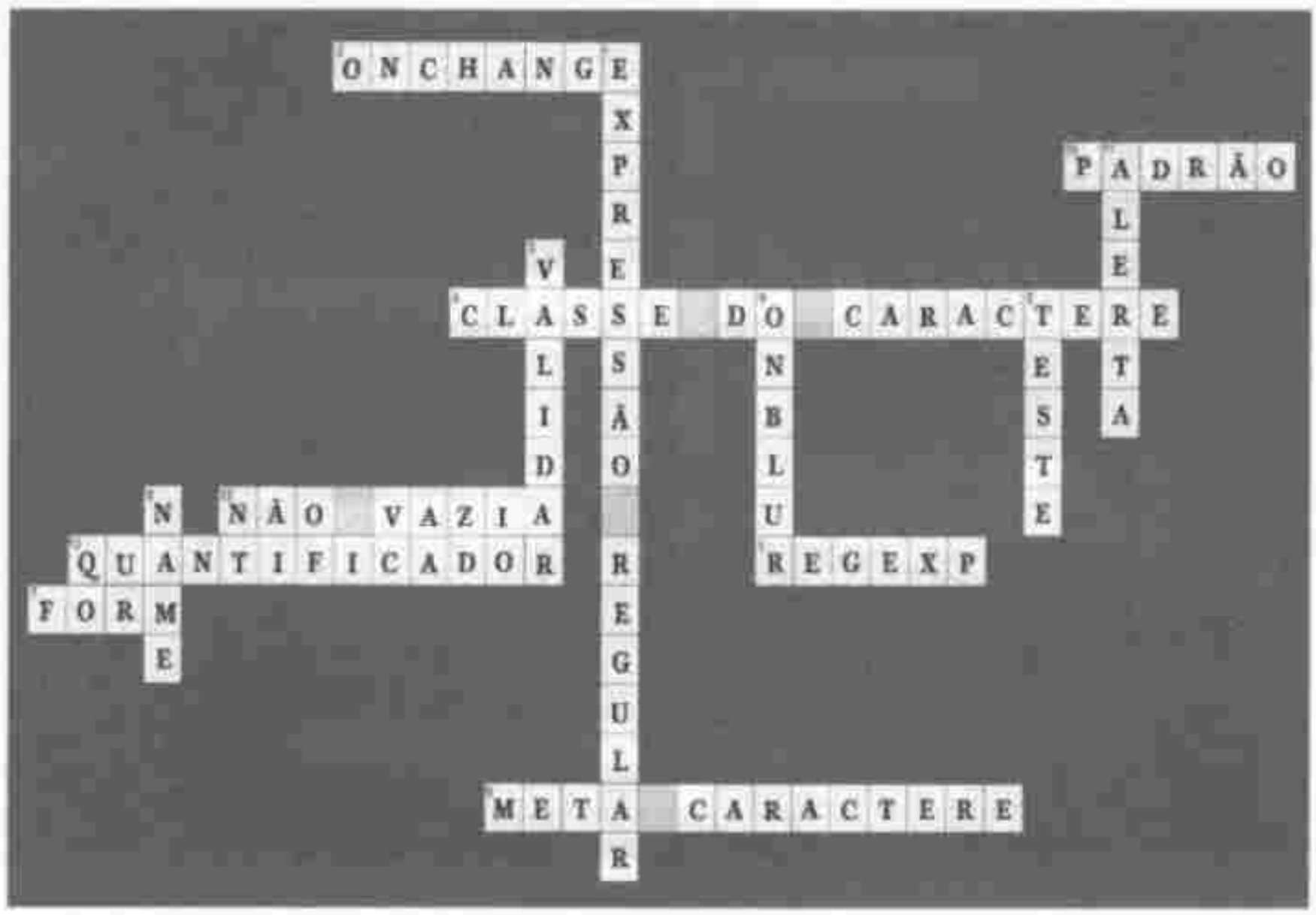
Horizontal

1. O objeto JavaScript que suporta as expressões regulares.
2. Inicializado quando os dados em um campo do formulário mudam.
4. Uma maneira prática de especificar caracteres opcionais em uma expressão regular.
7. Este objeto contém todos os campos individuais em um formulário.
9. Um caractere especial em uma expressão regular.
10. Uma descrição de um formato de dados.
12. Este tipo de validação verifica para assegurar que um campo do formulário tem dados.
13. Controla quantas vezes um subpadrão aparece em uma expressão regular.

Vertical

1. Usada para coincidir os padrões do texto.
3. Faça isto nos dados do formulário para assegurar que sejam legítimos.
5. O método usado para coincidir uma string com uma expressão regular.
6. Inicializado quando o usuário deixa um campo do formulário.
8. O atributo HTML que identifica com exclusividade um campo em um formulário.
11. Útil em muitos casos, mas geralmente não é a melhor maneira de notificar o usuário sobre os dados inválidos.

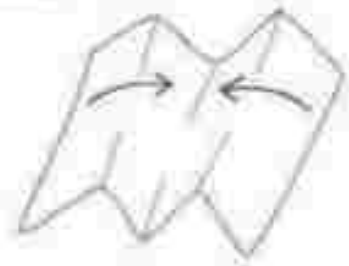
Solução da Cruzada do JavaScript



DOBRA DA PÁGINA

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

O que o JavaScript traz para os formulários Web?



"Mandango... o localizador de assentos de cinema para caras fortes!"

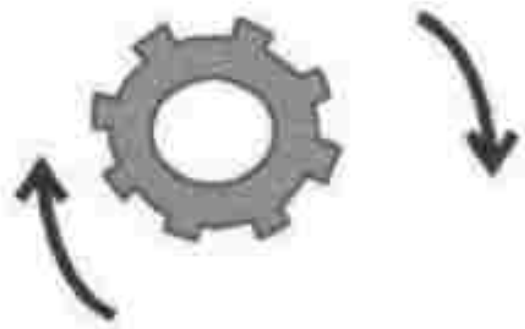
"... assentos de cinema para machos!"

105012

03/11/200

212-555-5339

seth@mandango



100012

11 de março de 2009

(212) 555-5339

seth@mandango.us

/^val(ley|ue|krie)/

/name|id$/

O JavaScript tem muito a oferecer para os formulários Web, portanto é difícil criar um argumento válido para qualquer coisa. A resposta quase certamente envolve dados em algum nível, mas como, especificamente?

Como repartir o HTML com o DOM

Ter controle do conteúdo da página Web é muito parecido com o ato de cozinhar. Bem, sem bagunça... e, infelizmente, também sem a recompensa comestível depois. Porém, você tem ***acesso total aos ingredientes HTML*** que entram em uma página Web e o mais importante, tem a capacidade de ***alterar*** a receita da página. Portanto, o **JavaScript torna possível manipular o código HTML em uma página Web** segundo a sua vontade, o que abre todos os tipos de oportunidades interessantes possibilitados por uma ***coleção de objetos padrão*** chamada de **DOM** (Document Object Model).

Funcional, mas desajeitada... problemas da interface

O script da aventura do Boneco de Palito no Capítulo 4 é um bom exemplo de tomada de decisão interativa com o JavaScript, mas a interface do usuário é um pouco difícil, especialmente seguindo os padrões Web modernos. As caixas de alerta tendem a ser chatas de navegar e os botões de opção obscuros não são muito claros, vendo como eles são simplesmente identificados como 1 e 2.

Os alertas podem ficar chatos e interrompem o fluxo do aplicativo.

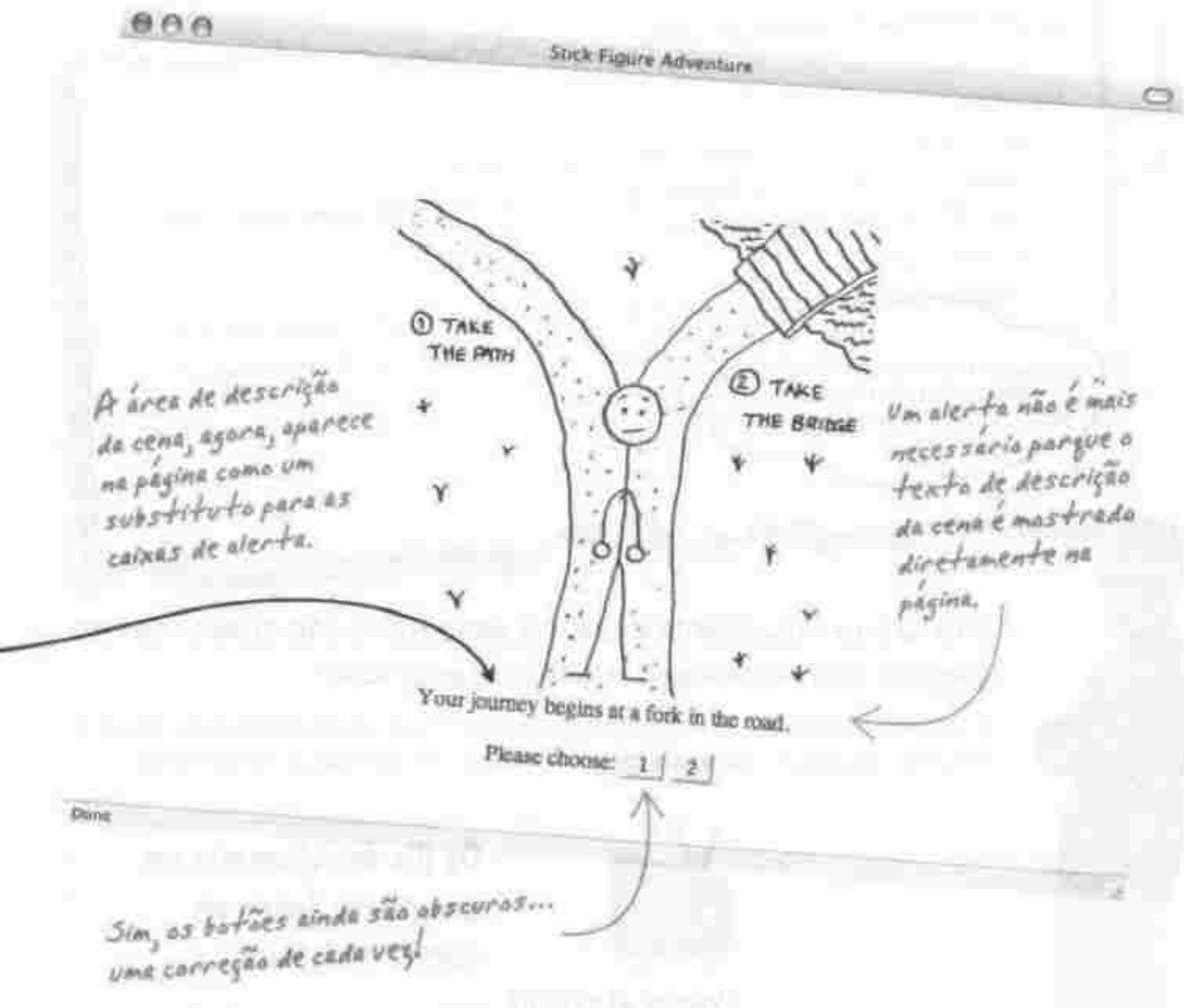
As opções de decisão são muito obscuras, fornecendo pouco contexto para tomar decisões.

Ellie percebe que é hora de corrigir os erros na interface do usuário na Aventura do Boneco de Palito...

Como descrever as cenas sem caixas de alerta

O problema com a abordagem da caixa de alerta para exibir as descrições da cena é que o texto desaparece assim que o usuário clica em OK. Poderia ser melhor se a descrição fosse exibida diretamente na página para evitar os alertas chatos e trazer a história para o corpo da página Web. É assim que Ellie deseja que a Aventura do Boneco de Palito fique:

O conjunto de arquivos mais recente para a Aventura do Boneco de Palito está pronto para você em www.altabooks.com.br.



Poder do cérebro

Qual você acha que seria a melhor maneira do JavaScript suportar a nova funcionalidade de descrição da cena?

Como criar espaço na página com div

Para exibir a descrição da cena na página, primeiro precisamos definir uma área física na página como um elemento HTML antes de usarmos com seriedade o código JavaScript. Como o texto de descrição da cena aparece como seu próprio parágrafo, uma tag funcionará bem para manter o texto da cena.

A tag >div> tem um ID que identifica com exclusividade o elemento para manter o texto de descrição da cena.

```
<body>
  <div style="margin-top:100px; text-align:center">
    <img id="sceneimg" src="scene0.png" alt="Stick Figure Adventure" />
    <div id="scenetext"></div><br />
    Please choose:
    <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
    <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
```

Vejo que a tag tem seu atributo id definido. Podemos usar esse ID para acessar a descrição da cena?

Um ID é precisamente como os elementos são acessados na página, inclusive a descrição da cena .

É verdade, o atributo `id` da tag `<div>` pode ser usado como a base para acessar o elemento na página a partir do código JavaScript. Na verdade, já fizemos isso...

Os IDs dos elementos em uma página devem ser sempre exclusivos.

Preste atenção! *Não se esqueça que todo o objetivo do atributo* `id` *é identificar com exclusividade os elementos em uma página. Por isso, eles devem sempre ser únicos dentro de uma determinada página.*

Como acessar os elementos HTML

O método `getElementById( )` do objeto `document` padrão que usamos muito. Ele permite alcançar uma página e acessar um elemento HTML... contanto que esse elemento tenha um ID exclusivo.

Isto deve coincidir com o atributo id do elemento HTML, neste caso um div.

```
var sceneDesc = document.getElementById("scenetext");
```

O elemento div é acessado usando seu atributo id.

```
<img id="sceneimg" src="scene0.png" ...
<div id="scenetext"></div><br />
Please choose:
```

Com o elemento de descrição da cena em mãos, estamos mais perto de manipular o conteúdo armazenado nele. Mas há um outro método que vale a pena investigar primeiro. É o método `getElementsByTagName( )`, que obtém todos os elementos em uma página de certo tipo, como `div` ou `img`. Esse método retorna um array contendo todos os elementos na página, na ordem na qual eles aparecem no HTML.

O nome da própria tag, sem <>.

```
var divs = document.getElementsByTagName("div");
```

Exercício

Escreva o código JavaScript para ter acesso à imagem laranja no seguinte código do corpo HTML, primeiro usando `getElementById( )` e então, usando `getElementsByTagName( )`.

```
<body>
  <p>Before starting, please choose an adventure stress level:</p>
  <img id="green" src="green.png" alt="Relaxing" /><br />
  <img id="blue" src="blue.png" alt="Irritating" /><br />
  <img id="yellow" src="yellow.png" alt="Frazzled" /><br />
  <img id="orange" src="orange.png" alt="Panicked" /><br />
  <img id="red" src="red.png" alt="Maddening" />
</body>
```

Usando `getElementById( )`: .....

Usando `getElementsByTagName( )`: .....

Exercício Solução

Escreva o código JavaScript para ter acesso à imagem laranja no seguinte código do corpo HTML, primeiro usando `getElementById( )` e então, usando `getElementsByTagName( )`.

```
<body>
  <p>Before starting, please choose an adventure stress level:</p>
  <img id="green" src="green.png" alt="Relaxing" /><br />
  <img id="blue" src="blue.png" alt="Irritating" /><br />
  <img id="yellow" src="yellow.png" alt="Frazzled" /><br />
  <img id="orange" src="orange.png" alt="Panicked" /><br />
  <img id="red" src="red.png" alt="Maddening" />
</body>
```

A imagem laranja é o quarto elemento no array, que tem um índice 3.

Usando `getElementById( )`: `document.getElementById("orange")`

Usando `getElementsByTagName( )`: `document.getElementsByTagName("img")[3]`

Como entrar em contato com seu HTML interno

Tudo bem, o motivo real de todo esse negócio de acesso do elemento HTML é para obter o conteúdo armazenado em um elemento. Você pode acessar os elementos HTML que são capazes de manter o conteúdo do texto, como, por exemplo, `div` e `p`, usando uma propriedade chamada `innerHTML`.

A propriedade innerHTML fornece acesso a todo conteúdo armazenado em um elemento.

Você está **sozinho** na floresta.

```
<p id="story">
  Você está
  <strong>sozinho</strong> na floresta.
</p>
```

O conteúdo HTML formatado também é armazenado na propriedade innerHTML.

innerHTML obtém todo o conteúdo de um elemento, inclusive qualquer tag HTML.

```
document.getElementById("story").innerHTML
```

Parece que você deve ser capaz de definir o conteúdo de um elemento HTML tão facilmente quanto pode obtê-lo. É possível?

innerHTML também pode ser usada para definir o conteúdo na página.

A propriedade `innerHTML` é realmente usada para **definir** o conteúdo HTML tanto quanto para **obtê-lo**. O conteúdo de um elemento pode ser definido para uma string de texto HTML atribuindo a string de texto à propriedade `innerHTML` do elemento. O novo conteúdo substituirá qualquer conteúdo que tenha pertencido anteriormente ao elemento.

```
document.getElementById("story").innerHTML =
    "Você não <strong>está</strong> sozinho!";
```

O conteúdo do elemento é definido ou, neste caso, substituído atribuindo uma string à propriedade innerHTML.

Você não **está** sozinho!

Aponte seu Lápis

Supondo que a mensagem de descrição da cena já está sendo definida corretamente com base nas decisões tomadas pelo usuário, escreva a linha de código que define o texto da mensagem para o elemento de descrição da cena na página Aventura do Boneco de Palito usando innerHTML.

Aponte seu Lápis Solução

Supondo que a mensagem de descrição da cena já está sendo definida corretamente com base nas decisões tomadas pelo usuário, escreva a linha de código que define o texto da mensagem para o elemento de descrição da cena na página Aventura do Boneco de Palito usando innerHTML.

O ID da mensagem da cena é sceneText.

```
document.getElementById("sceneText").innerHTML = message;
```

Uma aventura com menos interrupções

A área de descrição da cena que muda dinamicamente fornece à Aventura do Boneco de Palito uma experiência do usuário mais suave e divertida sem alertas desagradáveis.

Com exceção de adicionar `<div>` para a área de descrição da cena (mensagem) e do código para definir a propriedade `innerHTML`, as únicas outras alterações para o código da Aventura do Boneco de Palito envolvem adicionar uma variável `message` e então, definí-la em cada cena diferente...

Um foco no código da Aventura do Boneco de Palito

```
<html>
  <head>
    <title>Stick Figure Adventure</title>

    <script type="text/javascript">
      // Inicialize a cena atual na Cena 0 (Intro)
      var curScene = 0;

      function changeScene(decision) {
        // Limpe a mensagem da cena
        var message = "";

        switch (curScene) {
        case 0:
          curScene = 1;
          message = "Your journey begins at a fork in the road.";
          break;
        case 1:
          if (decision == 1) {
            curScene = 2
            message = "You have arrived at a cute little house in the woods.";
          }
          else {
            curScene = 3;
            message = "You are standing on the bridge overlooking a peaceful stream.";
          }
          break;
        ...
        }

        // Atualize a imagem da cena
        document.getElementById("sceneimg").src = "scene" + curScene + ".png";

        // Atualize o texto de descrição da cena
        document.getElementById("scenetext").innerHTML = message;
      }
    </script>
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      <img id="sceneimg" src="scene0.png" alt="Stick Figure Adventure" /><br />
      <div id="scenetext"></div><br />
      Please choose:
      <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
      <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
  </body>
</html>
```

A variável local da mensagem é criada para armazenar a mensagem de descrição da cena para a nova cena.

A variável message é definida para o texto de descrição único a cada cena.

O texto de descrição da cena é definido para o conteúdo da variável da mensagem usando a propriedade innerHTML.

Não existem Perguntas Idiotas

P: Posso usar `getElementById()` para acessar qualquer elemento em uma página?

R: Sim, mas apenas se o elemento tiver seu atributo `id` definido para um valor exclusivo. O atributo `id` é absolutamente essencial para usar o método `getElementById( )`.

P: `innerHTML` pode ser usada para definir o conteúdo de qualquer elemento HTML?

R: Não. Para definir o conteúdo "HTML interno" de um elemento, o elemento deve ser capaz de conter o conteúdo HTML. Portanto, na realidade, `innerHTML` é para definir o conteúdo de elementos como `div`, `span`, `p` e os outros elementos que agem como contêineres de conteúdo.

P: Quando você define o conteúdo de um elemento com `innerHTML`, o que acontece com o conteúdo?

R: A propriedade `innerHTML` sempre sobrescreve completamente qualquer conteúdo anterior quando o define. Portanto, não há nenhum conceito para anexar o conteúdo a `innerHTML`, embora você possa obter o efeito de anexação do conteúdo concatenando o novo conteúdo ao antigo e, então, atribuindo o resultado a `innerHTML`, como: `elem.innerHTML += " Esta sentença é anexada.`

Não tão rápido. Ouvi dizer que innerHTML não é nem mesmo um padrão Web. É verdade?

Bem, sim, mas os padrões Web realmente são uma preocupação?

É verdade, a `innerHTML` foi criada originalmente pela Microsoft como um recurso patenteado para o navegador Internet Explorer. Desde então, outros navegadores têm adotado a `innerHTML` e ela tornou-se um padrão **não oficial** para mudar rápido e facilmente o conteúdo dos elementos da página Web.

Mas o fato é que a `innerHTML` não é um padrão. Pode não parecer grande coisa, mas a idéia por detrás dos padrões é fazer com que as páginas Web e os aplicativos funcionem o máximo possível com muitos navegadores e plataformas. Além disso, há um modo compatível com os padrões de realizar a mesma tarefa que é basicamente mais flexível e poderoso, mesmo que não seja tão simples. Essa abordagem envolve o DOM, ou Document Object Model, uma coleção de objetos que fornece ao JavaScript um controle completo e total sobre a estrutura e o conteúdo das páginas Web.

Como ver a floresta e as árvores: Document Object Model (DOM)

O DOM oferece uma visão amistosa do script na estrutura e no conteúdo de uma página Web, que será importante se você quiser usar o JavaScript para alterar dinamicamente uma página. Através da lente do DOM, uma página parece uma hierarquia de elementos na forma de uma árvore. Cada folha na árvore é um **nó**, que se relaciona diretamente a cada elemento em uma página. Quando um nó aparece sob outro nó na árvore, ele é considerado um **filho** desse nó.

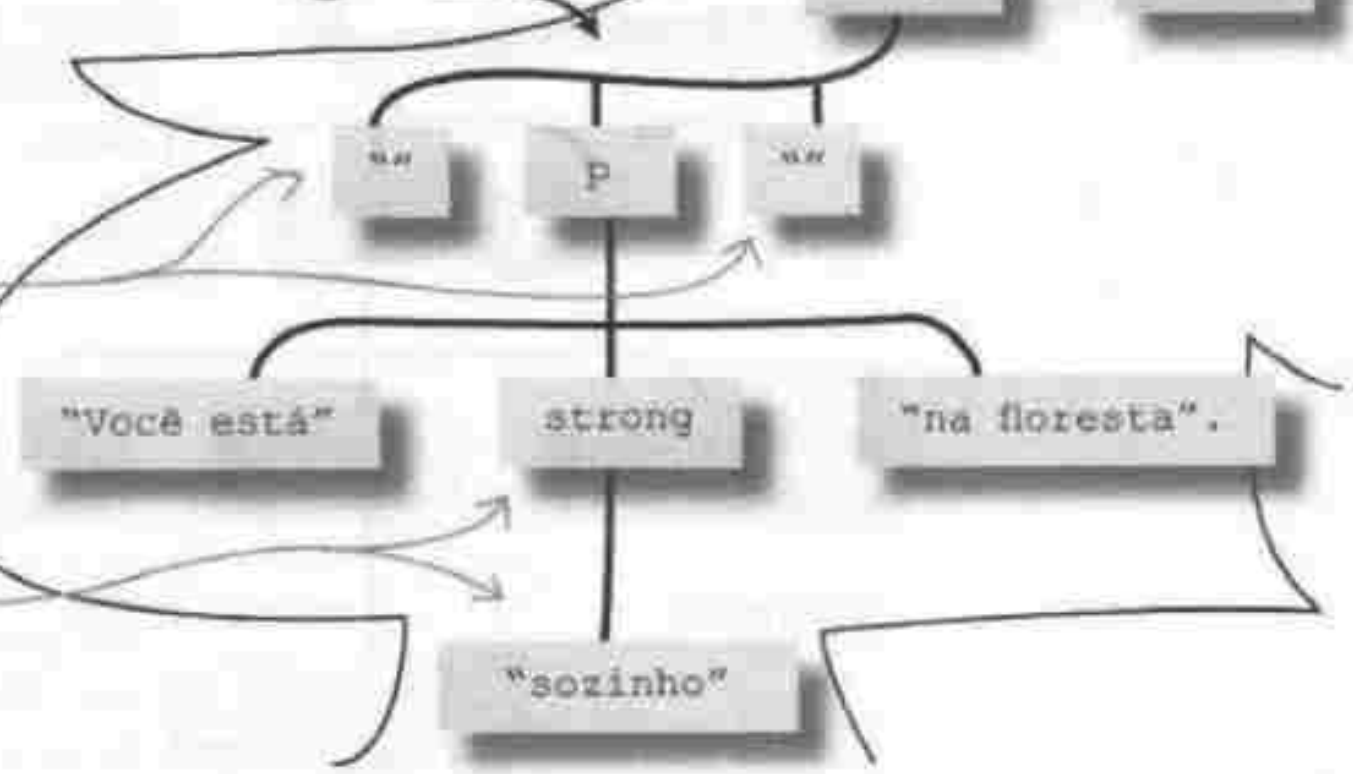
```
<html>
  <head></head>

  <body>
    <p id="story">
      Você está <strong>sozinho</strong> na floresta.
    </p>
  </body>
</html>
```

Sim, é uma árvore estranha, mas os nós de uma página lembram uma árvore.

No topo de cada árvore DOM, fica o nó Documento, que está acima do elemento HTML.

Estes são todos os nós.

O DOM "vê" uma página Web como uma árvore hierárquica de nós.



O espaço em branco em volta da tag <p> na página é interpretado como um texto vazio.

O texto forte sozinho aparece sob um nó da tag .

Sua página é uma coleção de nós DOM

Cada nó em uma árvore DOM é classificado segundo seu **tipo**. Os principais tipos de nó correspondem às partes estruturais de uma página, basicamente consistindo nos nós do elemento e os nós do texto.

DOCUMENTO
O nó superior em uma árvore HTML que representa o documento em si e aparece logo acima do elemento `html`.

ELEMENTO
Qualquer elemento HTML que corresponde a uma tag no código HTML.

Os nós DOM são classificados segundo seus tipos de nó.

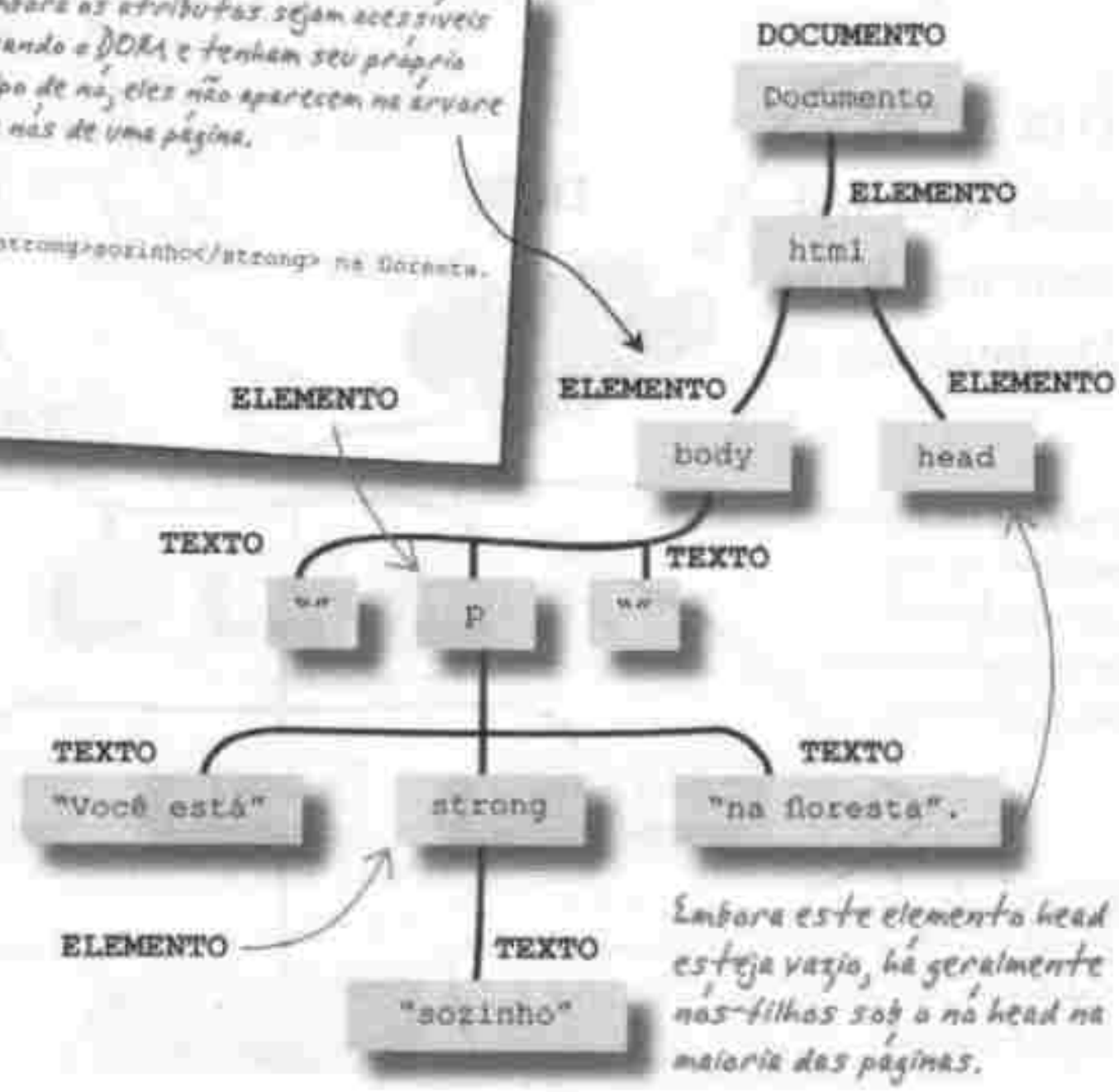
TEXTO
O conteúdo do texto para um elemento, sempre armazenado como um nó-filho sob um elemento.

ATRIBUTO
Um atributo de um elemento, acessível através de um nó do elemento, mas não presente diretamente na árvore DOM

Aplicar tipos de nó na árvore DOM para uma página Web ajuda a esclarecer exatamente como cada parte de uma página é percebida pelo DOM. O interesse particular é como os nós TEXTO sempre aparecem imediatamente sob um nó ELEMENTO como parte (ou todo) do conteúdo do nó.

```
<html>
  <head></head>
  <body>
    <p id="story">
      Você está <strong>sozinho</strong> na floresta.
    </p>
  </body>
</html>
```

Embora os atributos sejam acessíveis usando o DOM e tenham seu próprio tipo de nó, eles não aparecem na árvore de nós de uma página.

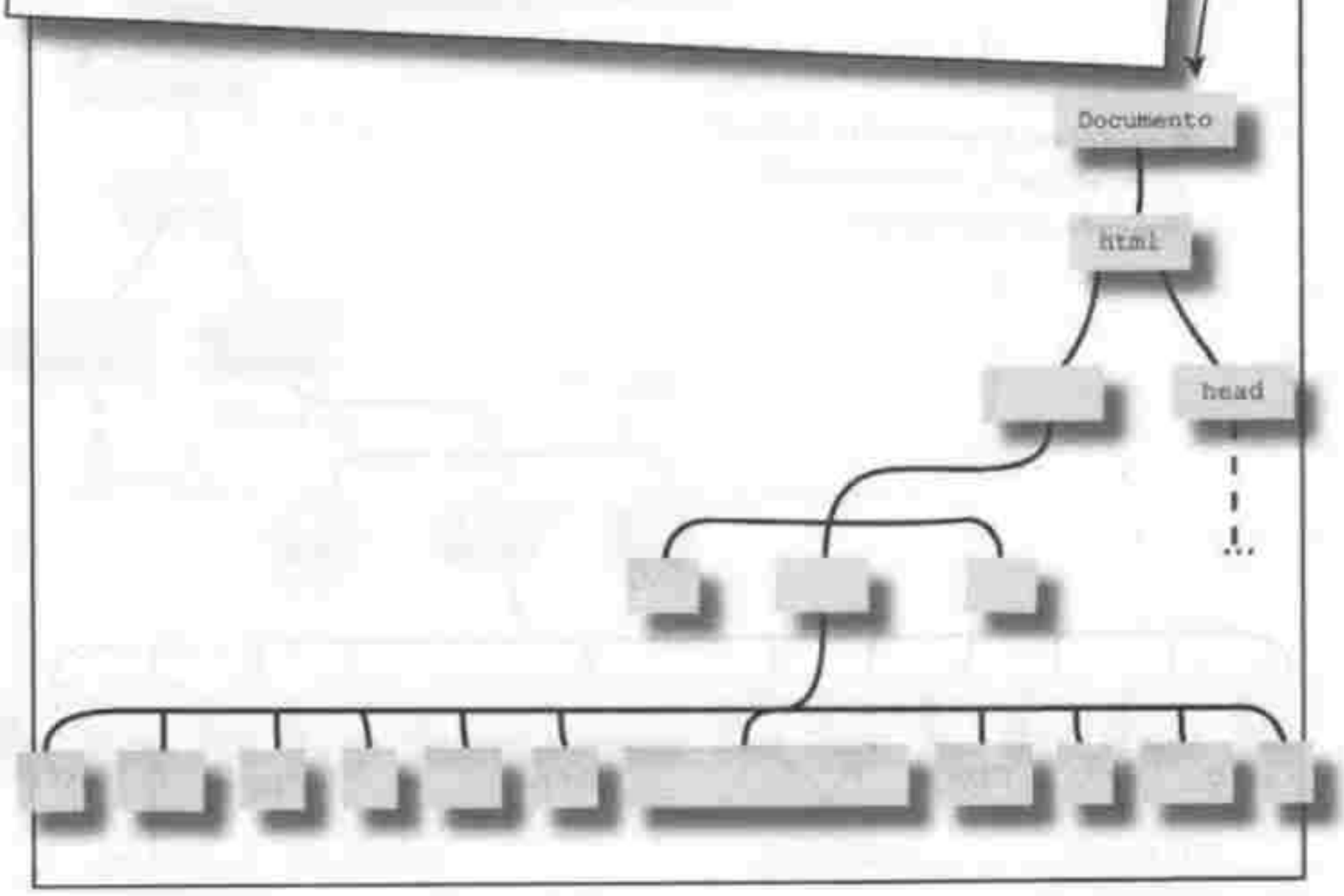


Embora este elemento head esteja vazio, há geralmente nós-filhos sob o nó head na maioria das páginas.

Aponte seu Lápis

Complete a representação da árvore DOM do código HTML da Aventura do Boneco de Palito escrevendo o nome de cada nó. Também anote o tipo de cada nó.

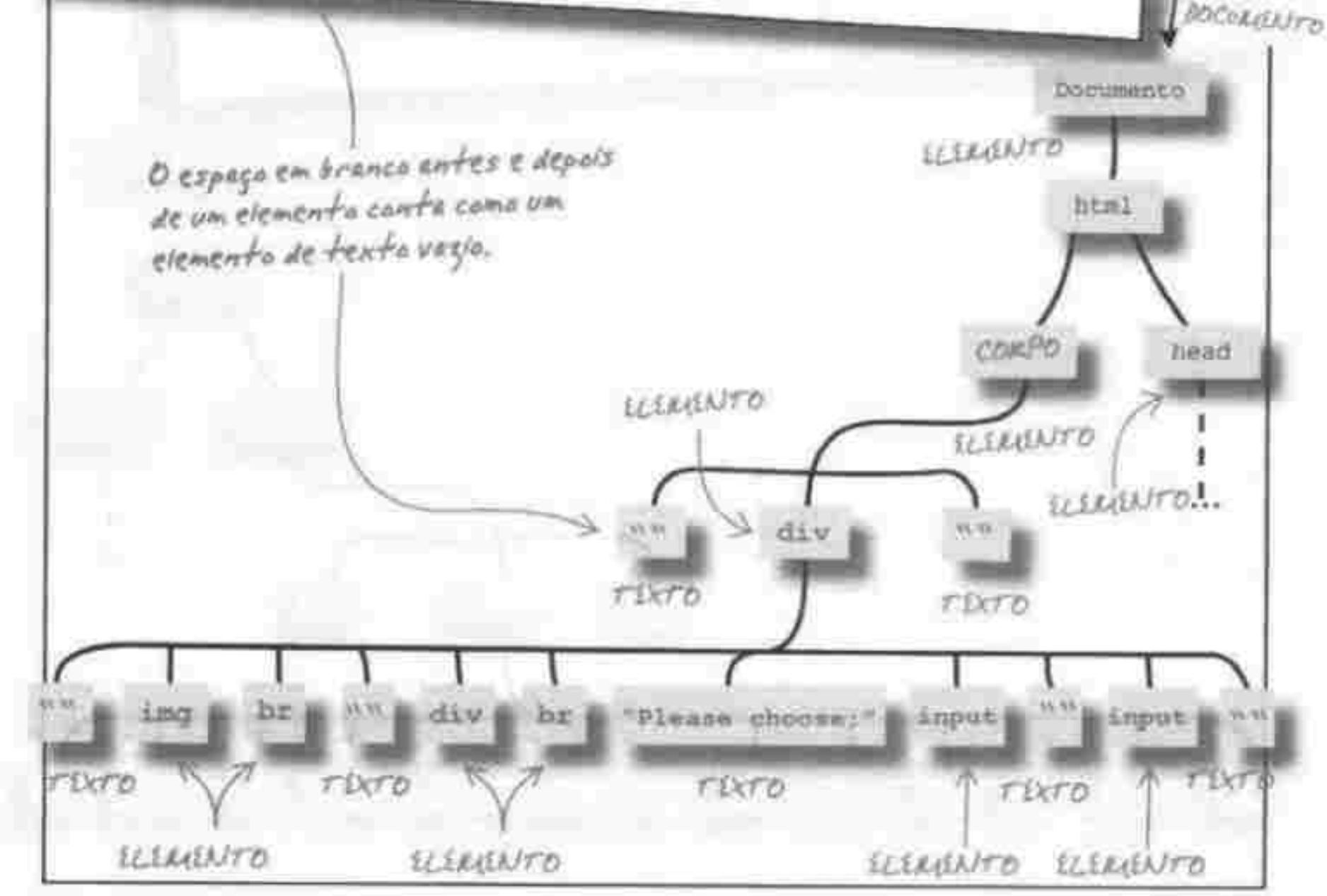
```
<html>
  <head>
    ...
  </head>

  <body>
    <div style="margin-top:150px; text-align:center">
      <img id="sceneimg" src="scene0.png" alt="Stick Figure Adventure" /><br />
      <div id="scenetext"></div><br />
      Please choose:
      <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
      <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
  </body>
</html>
```



Aponte seu Lápis
Solução

Complete a representação da árvore DOM do código HTML da Aventura do Boneco de Palito escrevendo o nome de cada nó. Também anote o tipo de cada nó.

```
<html>
  <head>
    ...
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      <img id="scenemg" src="scene0.png" alt="Stick Figure Adventure" /><br />
      <div id="scenetext"></div><br />
      Please choose:
      <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
      <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
  </body>
</html>
```



O espaço em branco antes e depois de um elemento conta como um elemento de texto vazio.

Como subir na árvore DOM com propriedades

A maioria das interações com o DOM começa com o objeto `document`, que é o nó mais superior na árvore de nós de um documento. O objeto do documento oferece métodos úteis como, por exemplo, `getElementById( )` e `getElementsByTagName( )`, e algumas propriedades também. Muitas propriedades do objeto `document` estão disponíveis a partir de **cada** nó em uma árvore. Alguns desses objetos ainda permitem a você navegar para outros nós. Isso significa que as propriedades do nó podem ser usadas para percorrer a árvore de nós.

As propriedades do nó são úteis para percorrer os nós na árvore DOM.

`nodeValue`
O valor armazenado em um nó, apenas para os nós de texto e de atributo (não de elementos).

`nodeType`
O tipo de um nó como, por exemplo, `DOCUMENT` ou `TEXT`, mas expressado como um número.

`childNodes`
Os arrays que contêm todos os nós-filhos sob um nó, na ordem na qual os nós aparecem no código HTML.

`lastChild`
O último nó-filho sob um nó.

`firstChild`
O primeiro nó-filho sob um nó.

Estas propriedades são a chave para ser capaz de manipular a árvore de documentos para acessar dados específicos do nó. Por exemplo, você pode usar as propriedades do nó com o método de acesso do nó `getElementById( )` para isolar rapidamente um nó específico.

A propriedade nodeValue sempre contém texto puro sem uma formatação adicional.

```
alert(document.getElementById("scenetext").nodeValue);
```

A propriedade nodeValue acessa o conteúdo texto armazenado no nó.

O texto de descrição da cena na Aventura do Boneco de Palito começa inicialmente vazio.

Tudo bem, então talvez não seja o melhor exemplo, vendo como o texto de descrição da cena div inicia vazio na Aventura do Boneco de Palito. Mas, finalmente, deve ser definido para algum texto muito atraente quando a história avançar, neste caso o código pareceria muito mais inteligente.

Exercício

O seguinte código irá se referir a um nó na árvore na página 356. Estude com cuidado o código e, então, circule o nó a que ele se refere.

```
document.getElementsByTagName("body")[0].childNodes[1].lastChild
```

Exercício Solução

O seguinte código irá se referir a um nó na árvore na página 356. Estude com cuidado o código e, então, circule o nó a que ele se refere.

```
document.getElementsByTagName("body")[0].childNodes[1].lastChild
```

Há apenas uma tag <body>, portanto ela tem que ser referida como o primeiro elemento no array retornado por getElementsByTagName( ).

O último filho do elemento div principal é um elemento de texto vazio.

O segundo nó-filho do elemento body é o elemento div.

O método getElementById( ) obtém um único elemento que é definido para um ID específico.

O método getElementsByTagName( ) obtém todos os elementos com certo nome de tag na página inteira, como <input>, por exemplo.

Não existem Perguntas Idiotas

P: Qual é a diferença entre `getElementById()` e `getElementsByTagName( )` na árvore DOM? Por que eu deveria escolher um ao invés do outro?

R: Os dois métodos oferecem abordagens diferentes que basicamente têm relação quer ou não seu objetivo seja isolar um **único** elemento ou um **grupo** de elementos parecidos. Para isolar um único elemento, você não pode tocar em `getElementById( )` - apenas mostre um ID no elemento e estará pronto para continuar.

Mas se quiser visar um grupo de nós, `getElementByTagName( )` será uma opção muito melhor. Por exemplo, se você quisesse ocultar todas as imagens em uma página usando o JavaScript, primeiro chamaria `getElementsByTagName( )` e transmitiria `"img"` para obter todos os nós da imagem na página. Então, mudaria a propriedade do estilo CSS `visibility` em cada um dos elementos da imagem para ocultá-los. Opa, estamos nos adiantando... voltaremos ao DOM e à CSS posteriormente neste capítulo. Agora, apenas compreenda que embora o `getElementsByTagName( )` não seja tão popular quanto o `getElementById( )`, ainda tem seu lugar em situações especiais.

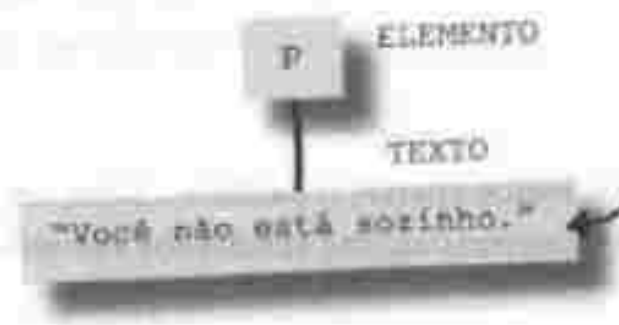
Então, as propriedades do nó tornam possível entrar no código HTML e acessar o conteúdo da página Web... mas elas podem ser usadas para mudar esse conteúdo?

As propriedades DOM permitem mudar o conteúdo da página Web e manter os padrões Web de acordo.

Como o DOM exibe tudo em um documento da página Web como um nó, mudar uma página envolve mudar seus nós. No caso do conteúdo de texto, o texto para um elemento como, por exemplo, div, span ou p sempre aparece como um nó ou nós-filhos, imediatamente sob o elemento (nó) na árvore. Se o texto estiver em um único nó de texto sem nenhum elemento HTML adicional, então o nó estará localizado no primeiro filho. Assim:

```
document.getElementById("story").firstChild.nodeValue
```

```
<p id="story">
   Você não está sozinho.
</p>
```



Você não está sozinho.

Poder do cérebro

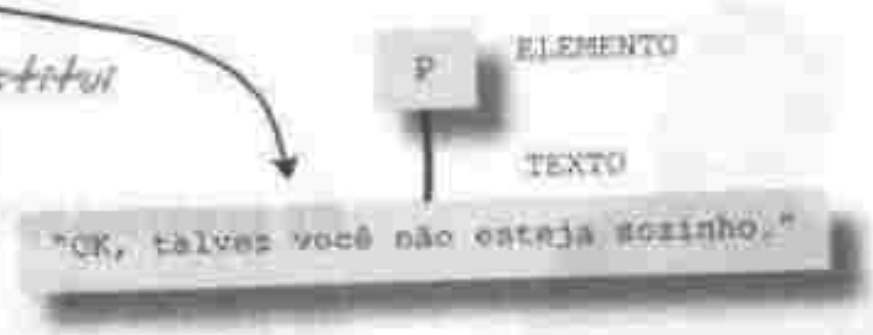
Como você mudaria o texto de um nó usando o DOM?

Como mudar o texto do nó com DOM

Se você puder supor com segurança que um nó tem apenas um filho que mantém seu conteúdo de texto, então será possível simplesmente atribuir o novo conteúdo de texto ao filho usando sua propriedade `nodeValue`. Essa abordagem funciona muito bem, novamente, mas apenas se houver um único nó-filho.

```
document.getElementById("story").firstChild.nodeValue = "OK, talvez você esteja sozinho.";
```

O novo conteúdo do texto substitui o conteúdo existente no filho.



Mas as coisas nem sempre são tão simples. O que acontecerá se um nó tiver mais de um filho? Como este código:

```
<p id="story">
  Você não <strong>está</strong> sozinho.
</p>
```

Este parágrafo divide-se em diversos nós-filhos.

Mudar o conteúdo no primeiro filho não é suficiente para mudar o conteúdo inteiro do parágrafo.

Como há uma tag no conteúdo do parágrafo, existem diversos nós-filhos.

Se substituirmos apenas o primeiro filho, os nós-filhos restantes ainda existirão e teremos alguns resultados estranhos como estes:

```
document.getElementById("story").firstChild.nodeValue = "OK, talvez você esteja sozinho.";
```

Apenas o primeiro filho é substituído, o que ainda deixa o conteúdo restante... e alguns resultados confusos.

Três etapas (seguras) para mudar o texto do nó

O problema em mudar o conteúdo de um nó alterando apenas o primeiro filho é que ele não se decompõe nas expectativas dos outros nós-filhos. Portanto, para mudar o conteúdo de um nó, realmente devemos limpar todos os seus filhos e, então, adicionar um novo filho que contenha o novo conteúdo.

1. Remova todos os nós-filhos.
2. Crie um novo nó de texto com base no novo conteúdo.
3. Anexe o nó do texto recém-criado como um nó filho.

Podemos fazer isso com três métodos DOM:

1. **`removeChild( )`**
   Remove um nó filho de um nó; transmita o nó-filho a ser removido.
2. **`createTextNode( )`**
   Cria um nó de texto a partir de uma string de texto.
3. **`appendChild( )`**
   Adiciona um nó como o último filho do nó; transmite o nó-filho a ser adicionado.

Para mudar o conteúdo do texto no exemplo "você não está sozinho", temos que realizar essas três etapas, primeiro removendo todos os nós-filhos, então criando um novo nó de texto e, finalmente, anexando o nó de texto ao parágrafo.

Primeiro, obtenha o elemento (nó) usando seu ID.

```
var node = document.getElementById("story");
while (node.firstChild)
  node.removeChild(node.firstChild);
node.appendChild(document.createTextNode("OK, talvez você esteja sozinho."));
```

Remove o primeiro nó-filho até que não haja mais nós-filhos.

Crie um nó de texto novo.

Depois de remover todos os nós-filhos, anexe o novo nó de texto ao nó-pai.

OK, talvez você esteja sozinho.

Blocos de construção DOM

Entrevista desta semana: O nó debate sobre o critério das árvores DOM

Use a Cabeça: Fiquei sabendo que você é a menor unidade de armazenamento em uma árvore DOM, um tipo de átomo para o conteúdo HTML. É verdade?

Nó: Não estou certo sobre como minha atomicidade, mas, sim, represento uma parte separada de informação em uma árvore DOM. Considere o DOM como uma divisão, em cada página Web, em partes minúsculas de informação... e eu sou essa parte!

Use a Cabeça: Por que isso importa? Quero dizer, é realmente importante ser capaz de dividir uma página Web em pequenas partes de dados?

Nó: Será importante apenas se você for acessar ou alterar as informações em uma página Web. Muitos scripts importam-se com isso, neste caso o DOM importa-se muito. Mas a razão real que importa é porque dá muito poder ser capaz de desmontar uma página Web em todos os seus pequenos fragmentos e partes.

Use a Cabeça: Você não corre o risco de perder uma parte ao separar uma página? Muitas pessoas separam algo apenas para ter os fragmentos para uma futura consideração e o que fazem sem seguida é danificar a coisa.

Nó: Não, não é um problema com o DOM, pois você não tem que separar literalmente nada para acessar uma página Web como uma árvore de nós. O DOM fornece a exibição de árvore independentemente de você planejar, de fato, fazer qualquer modificação na forma ou aparar os dados Web.

Use a Cabeça: É um alívio. Mas se eu realmente quiser fazer algum corte da página Web, é onde você entra?

Nó: Sim. Exceto que você não está limitado ao corte – está livre para adicionar à árvore de dados Web também.

Use a Cabeça: Oba, é muito surpreendente. Como isso funciona?

Nó: Bem, lembre-se de que cada parte da informação é modelada na árvore como um nó. Portanto, você pode passar por mim para acessar qualquer coisa em uma página. Ou pode criar partes inteiramente novas de dados Web ao me usar e, então, adicioná-las à árvore. O DOM é realmente bem flexível.

Use a Cabeça: É ótimo. Algo que ainda me confunde, porém, é como você se refere aos elementos. Vocês são realmente a mesma pessoa?

Nó: Sim, na verdade somos. Mas vou um pouco mais longe. Lembre-se de que um elemento é apenas outro modo de ver uma tag como, por exemplo, ou . Todo elemento em uma página é representado por um nó na árvore de documentos, portanto neste sentido, o elemento e eu somos os mesmos. Porém, também represento o conteúdo armazenado dentro de um elemento. Portanto, o texto armazenado em um também é seu próprio nó, armazenado logo abaixo do nó div na árvore.

Use a Cabeça: Parece um pouco confuso. Como posso saber a diferença entre os elementos e seu conteúdo?

Nó: Bem, primeiro de tudo, o conteúdo armazenado em um elemento, ou nó, sempre aparece como um filho do nó na árvore DOM. E, segundo, todos os nós são distinguíveis pelo tipo: um nó do elemento tem o tipo de nó ELEMENTO, ao passo que seu conteúdo de texto tem o tipo de nó TEXTO.

Use a Cabeça: Então, se eu quiser acessar o conteúdo de texto de um elemento, simplesmente procuro o tipo de nó TEXTO?

Nó: Você poderia; apenas lembre-se de que a propriedade nodeType de fato retorna um número para cada tipo de nó. Por exemplo, o tipo de nó TEXTO é 3, ao passo que ELEMENTO é 1. Mas mesmo isso não é realmente necessário porque tudo que tem a fazer é procurar os filhos de um nó do elemento para acessar seu conteúdo.

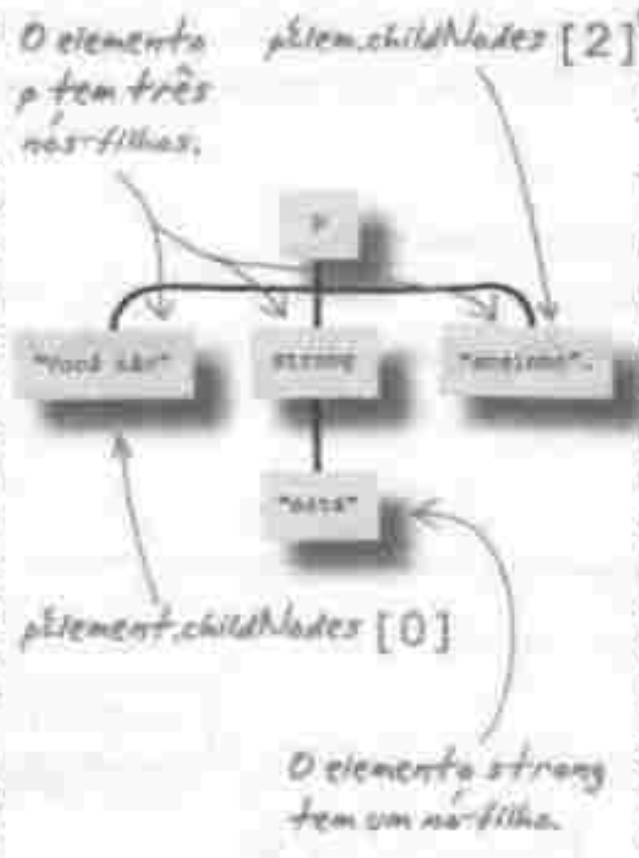
Use a Cabeça: Compreendo. Bem, obrigado pelo seu tempo e por mostrar as maravilhas da árvore DOM.

Nó: De nada. E se quiser alguma cirurgia na árvore, não se esqueça de me procurar!

Não existem Perguntas Idiotas

P: Ainda estou um pouco confuso com os nós-filhos e como eles são organizados. Por exemplo, como a propriedade `childNodes` funciona?

R: Quando um nó contém dados, o nó é considerado um nó-pai e os dados nele são percebidos pelo DOM como nós-filhos. Se os dados consistirem em qualquer coisa diferente dos dados brutos do texto, então eles serão divididos em vários nós-filhos. Os nós-filhos sob um nó-pai aparecem na propriedade `childNodes` do pai como um array e sua ordem no array coincide com a ordem na qual eles aparecem no próprio código HTML. Portanto, o primeiro nó-filho no array `childNodes` pode ser acessado usando `childNodes[0]`. O array pode também ser percorrido com um loop para acessar cada um dos nós-filhos.



P: No código que remove todos os nós-filhos de um nó, como funciona a condição de teste do loop `while`?

R: O teste do loop while é assim:

```
while(node.firstChild)
```

O que este teste está fazendo é a verificação para saber se o nó ainda contém um primeiro nó-filho. Se ainda houver um primeiro nó-filho, sua presença resultará em um valor `true` no contexto do loop `while` e o loop continuará em outra iteração. Se não houver nenhum primeiro nó-filho, isso significará que não há nenhum filho. E se este for o caso, o código `node.firstChild` resultará em `null`, que é convertido em `false` no contexto do loop `while`. Portanto, o que está realmente acontecendo é que o loop `while` verifica para saber se o primeiro nó-filho é `null`, que é uma evidência conclusiva de que não existe nenhum outro nó-filho à espreita.

Ímãs de geladeira JavaScript

A versão compatível com o DOM da Aventura do Boneco de Palito não tem várias partes do código importante. Use os ímãs abaixo para terminar o código que muda o texto do nó para o elemento de texto da cena. Os ímãs podem ser usados mais de uma vez.

```
// Atualize o texto de descrição da cena

var ..................... = document.getElementById("....................");

while (..................... = .....................)

..................... = ........................ (..................... = .....................)

..................... = ..........................(document.createTextNode(.................));
```

firstChild

appendChild

sceneText

removeChild

sceneText

message

Solução dos ímãs de geladeira JavaScript

A versão compatível com o DOM da Aventura do Boneco de Palito não tem várias partes do código importante. Use os ímãs abaixo para terminar o código que muda o texto do nó para o elemento de texto da cena. Os ímãs podem ser usados mais de uma vez.

Contanto que o nó de texto da cena tenha nós-filhos, continue no loop.

O loop continuará percorrendo, contanto que um primeiro filho exista.

Primeiro, obtenha o elemento de texto da cena usando seu ID.

A mensagem deve ser texto puro sem formatação ou tags HTML.

```
// Atualize o texto de descrição da cena

var ... = document.getElementById(...);

while (...) {
  ...
  ...
}
```

sceneText

sceneText

sceneText

firstChild

sceneText

removeChild

sceneText

firstChild

sceneText

appendChild

(document.createTextNode(...

message

Continue removendo o primeiro filho restante do nó de texto da cena até que não reste nenhum filho.

Agora que não há filhos, anexar o novo nó de texto irá assegurar que o conteúdo servirá como um substituto completo.

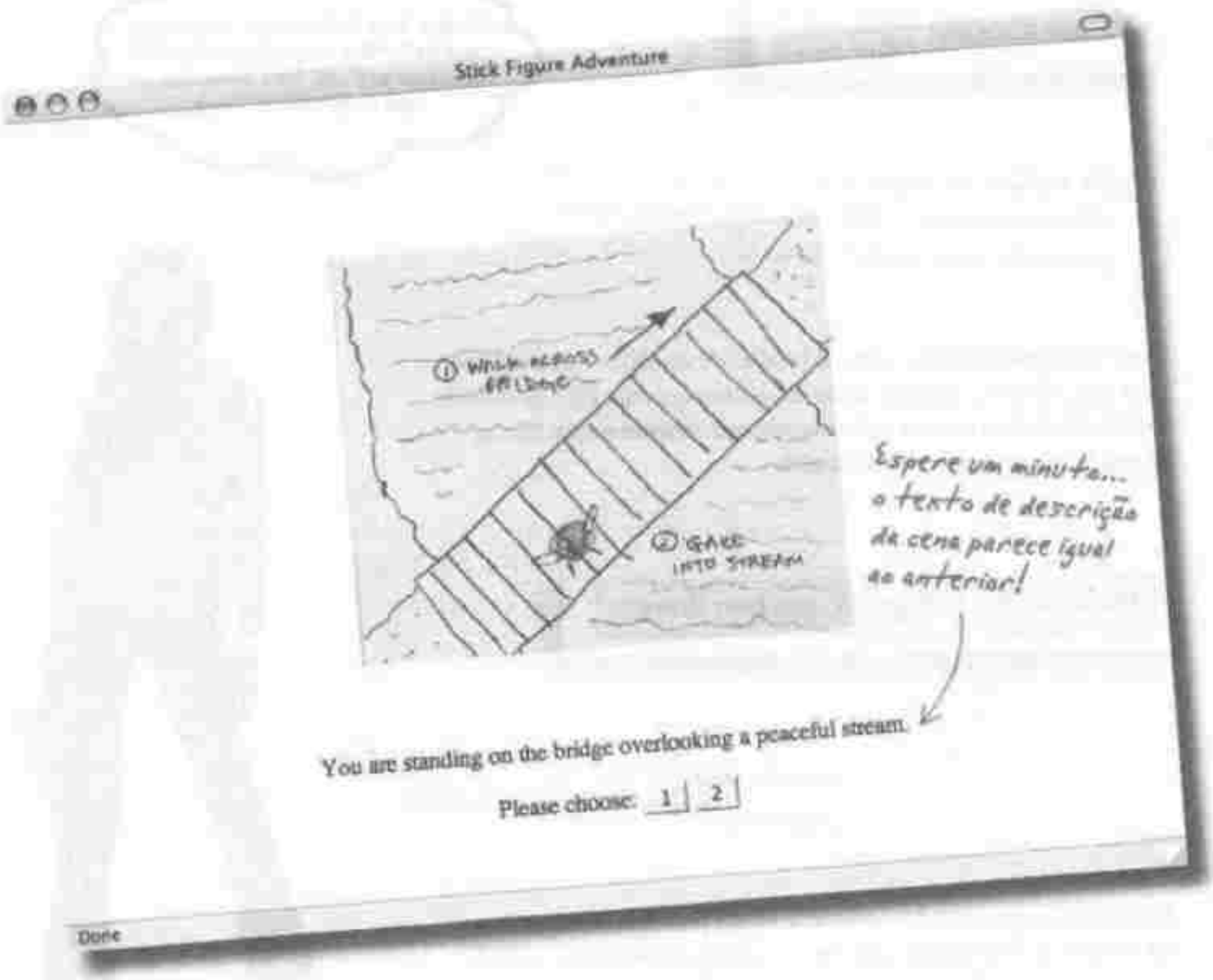
Com todos os nós-filhos removidos, um novo nó de texto é criado e adicionado como um filho do nó de texto da cena.

Pontos de Bala

- Embora não seja um padrão Web, a propriedade `innerHTML` fornece acesso a todo o conteúdo armazenado em um elemento.
- O Document Object Model, ou DOM, fornece um mecanismo **padronizado** de acessar e modificar os dados da página Web.
- O DOM vê uma página Web como uma **árvore** hierárquica de nós relacionados.
- A alternativa DOM de mudar o conteúdo da página Web com `innerHTML` envolve remover todos os nós-filhos de um elemento e, então, criar e anexar um novo nó-filho que contém o novo conteúdo.

Aventura compatível com padrões

Cara, parece divertido! A marca de qualquer boa aventura é a compatibilidade com os padrões... ou não. Mas pode ser uma boa coisa no contexto dos aplicativos Web modernos. E o mais importante, veja as mudanças drásticas que a abordagem DOM para alterar o texto de descrição da cena trouxe para a Aventura do Boneco de Palito...



Hmm. Tudo bem, então talvez a página não pareça diferente, mas internamente ela segue os últimos padrões Web graças ao seu uso do DOM. Nem tudo no código JavaScript pode ser apreciado visualmente e neste caso, nossa satisfação com a versão produzida pelo DOM da Aventura do Boneco de Palito terá que vir de dentro.

O DOM é um modo padrão de manipular a HTML que permite mais controle do que usar a propriedade innerHTML.

À procura de melhores opções

Então, agora, o texto dinâmico de descrição da cena foi revisado duas vezes, mas aqueles botões de opção obscuros ainda continuam na Aventura do Boneco de Palito. Certamente, algo poderia ser feito para tornar a navegação da história um pouco mais atraente e clara do que escolher entre os números 1 e 2!

Please choose: 1 2

Sei que eles terminaram o trabalho, mas os botões de opção realmente são inexpressivos. Eles deveriam ser muito mais descritivos.

Os botões de opção numéricos simplesmente não se encaixam – eles não dizem nada sobre a decisão que o usuário encara.

Uma melhoria razoável para os botões de opção seria mudá-los para que, de fato, reflitam as decisões disponíveis. Eles poderiam incluir texto nos botões informando exatamente o que são as duas opções em cada ponto, assim:

Walk across Bridge **Gaze into Stream**

Muito melhor! Agora, as opções estão desempenham o papel das decisões.

Pense, realmente não há nenhuma razão para termos que usar os botões do formulário para isso – qualquer elemento HTML que possa conter texto poderia funcionar possivelmente. Os estilos CSS poderiam ser usados para arrumá-los e fazer com que se pareçam mais com controles de entrada.

Poder do cérebro

Como você implementaria as opções baseadas em dados na Aventura do Boneco de Palito para que elas exibam o texto da opção específico para cada cena diferente?

Como construir opções melhores e mais claras

Como as opções de tomada de decisão novas e melhoradas na Aventura do Boneco de Palito são elementos HTML que contêm texto, o DOM pode ser usado para alterar dinamicamente o texto de decisão em cada cena. Isso significa que cada cena definirá seu texto de decisão junto com sua descrição. E também significa que a função `changeScene( )` precisará de duas variáveis novas para armazenar este texto de decisão, `decision1` e `decision2`.

Eis como poderíamos definir o texto de decisão da Cena 1 quando ele for para a Cena 3 na função `changeScene( )`:

```
curScene = 3;

message = "You are standing on the bridge overlooking a peaceful stream.";

decision1 = "Walk across Bridge";
decision2 = "Gaze into Stream";
```

As variáveis decision1 e decision2 são usadas para armazenar o texto de decisão da cena para uma determinada cena.

Aponte seu Lápis

As opções dinâmicas na Aventura do Boneco de Palito requerem uma nova abordagem para o modo como as opções são representadas no código HTML. Escreva o código para os novos elementos de texto que substituem os botões `<input>` existentes.

Sugestão: A classe de estilo CSS para os novos elementos é chamada "`decision`" e o conteúdo do primeiro elemento é inicialmente definido para "`Start Game`".

Please choose: 1 2

Reescreva o código para as opções dinâmicas!

```
<input type="button" id="decision1" value="1" onclick="changeScene(1)" />
<input type="button" id="decision2" value="2" onclick="changeScene(2)" />
```

Start Game

Aponte seu Lápis
Solução

As opções dinâmicas na Aventura do Boneco de Palito requerem uma nova abordagem para o modo como as opções são representadas no código HTML. Escreva o código para os novos elementos de texto que substituem os botões `<input>` existentes.

Sugestão: A classe de estilo CSS para os novos elementos é chamada "`decision`" e o conteúdo do primeiro elemento é inicialmente definido para "`Start Game`".

Reescreva o código para as opções dinâmicas!

Please choose: 1 2

```
<input type="button" id="decision1" value="1" onclick="changeScene(1)" />
<input type="button" id="decision2" value="2" onclick="changeScene(2)" />
```

Start Game

```
<div id="decision1" class="decision" onclick="changeScene(1)">Start Game</div>
.....................................................................
<div id="decision2" class="decision" onclick="changeScene(2)">Start Game</div>
.....................................................................
```

Substituir o texto do nó parece uma tarefa que seria útil ter em uma função.

Como repensar a substituição do texto do nó

Agora, tudo o que está faltando na Aventura do Boneco de Palito para o novo texto de decisão dinâmico é o código que realmente define o texto para os novos elementos `span`. Esse código está basicamente fazendo a mesma coisa que o código DOM que escrevemos anteriormente no capítulo, que muda dinamicamente o texto de descrição da cena. Na verdade, isso traz um problema porque, agora, precisamos executar a mesma tarefa exata em três elementos diferentes: a descrição da cena e as duas decisões da cena...

Como substituir o texto do nó com uma função

A função de substituição do texto do nó geral é algo útil e não apenas na Aventura do Boneco de Palito. Esse tipo de função opera de modo muito parecido com o código de substituição do texto de descrição da cena que vimos anteriormente, exceto que desta vez o código trabalhará nos argumentos da função.

O ID do nó cujo conteúdo será substituído.

```
function replaceNodeText(id, newText) {
  ...
}
```

O novo conteúdo do texto para substituir no nó.

A função `replaceNodeText( )` personalizada aceita dois argumentos: o ID do nó cujo conteúdo será substituído e o novo texto para colocar no nó. Use essa função para mudar o conteúdo de texto de qualquer elemento em uma página que possa manter texto. Na Aventura do Boneco de Palito, a função permite, agora, mudar dinamicamente o texto de descrição da cena e o texto para as duas decisões simultaneamente... mas, claro, você precisa escrevê-lo primeiro.

Substitua o texto de descrição da cena por uma nova mensagem.

Ao invés de duplicar o mesmo código três vezes, a função é chamada três vezes.

```
replaceNodeText("scenetext", message);
replaceNodeText("decision1", decision1);
replaceNodeText("decision2", decision2);
```

Mude o texto de decisão de cada uma das duas decisões.

Aponte seu Lápis

Escreva o código para a função `replaceNodeText( )`, a função geral para substituir o texto dentro de um nó que é referenciado pelo ID.

Não se esqueça de que a função aceita dois argumentos, `id` e `newText`.

.....

.....

.....

.....

.....

.....

.....

Aponte seu Lápis
Solução

Escreva o código para a função `replaceNodeText( )`, a função geral para substituir o texto dentro de um nó que é referenciado pelo ID.

Não se esqueça de que a função aceita dois argumentos, `id` e `newText`.

Obtenha o elemento usando seu ID exclusivo.

Remova todos os filhos do nó.

Crie um novo elemento-filho de texto usando o texto transmitido para a função.

A função createTextNode( ) está disponível apenas no objeto do documento e não tem nenhuma ligação direta com um determinado nó.

As opções dinâmicas são algo bom

As novas decisões de texto dinâmicas na Aventura do Boneco de Palito são muito mais claras do que seus correspondentes com botões obscuros.

As novas decisões dinâmicas permitem que os usuários saibam exatamente quais são suas opções em cada ponto na história.

Não existem Perguntas Idiotas

P: Por que os elementos `span` são usados para as decisões da Aventura do Boneco de Palito, em oposição aos elementos `div`?

R: Porque os elementos de decisão precisam aparecer lado a lado, o que significa que eles não podem ser elementos de bloco que iniciam em uma nova linha. Um `div` é um elemento de bloco, ao passo que `span` está em linha. Em linha é o que desejamos para as decisões, portanto `span` é o bilhete de entrada.

P: Quando crio um novo código com `createTextNode( )`, onde fica o nó?

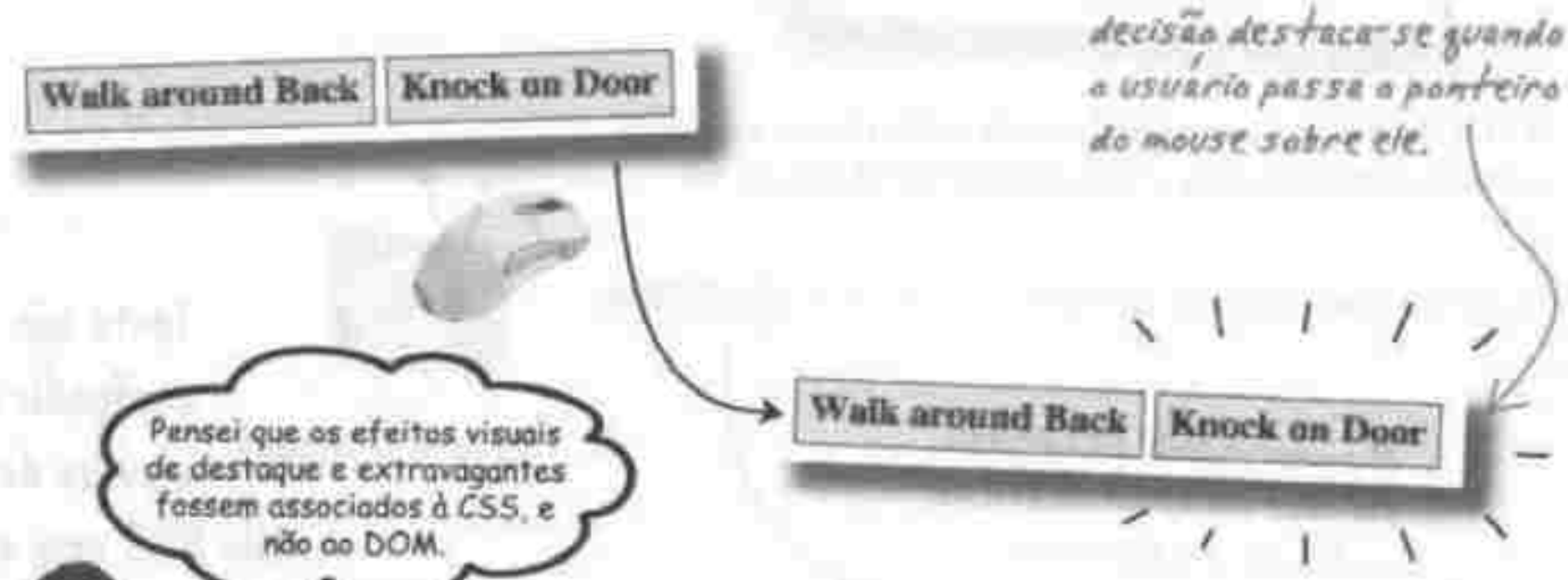
R: Em nenhum lugar. Quando um novo nó do texto é criado pela primeira vez, ele fica no limbo, pelo menos em relação à árvore DOM para uma determinada página. Fica assim até que você anexe o nó como um filho de outro nó que realmente é adicionado à árvore, que então o adiciona à página.

P: O conteúdo de um nó de texto criado com `createTextNode( )` tem que ser apenas texto?

R: Sim. O DOM não funciona como o `innerHTML`, onde você pode atribuir texto com tags misturadas nele. Quando o DOM fala sobre um "nó do texto", ele realmente quer dizer texto puro sem nenhuma outra tag ou formatação adicionada.

As opções interativas são ainda melhores

Portanto, as decisões de texto dinâmicas na Aventura do Boneco de Palito são uma melhoria sobre suas antecessoras obscuras, mas poderiam ser melhores ainda. Por exemplo, elas poderiam destacar-se quando o ponteiro do mouse passar sobre elas para deixar claro que são clicáveis.



O destaque está associado à CSS, mas o DOM ainda está envolvido diretamente.

Destacar um conteúdo da página Web é, na verdade, um problema da CSS porque envolve ajustar a cor do segundo plano de um elemento. Mas o DOM também entra na equação do destaque porque fornece um acesso programático aos estilos CSS dos elementos...

Uma questão de estilo: CSS e DOM

Os estilos CSS estão ligados aos elementos HTML e o DOM fornece acesso aos estilos através dos elementos (nós). Usando o DOM para ajustar os estilos CSS, é possível manipular dinamicamente a apresentação do conteúdo. Um modo dos estilos CSS serem exibidos através do DOM é na classe style de um elemento, que é onde um grupo (classe) de estilos é aplicado a um elemento.

```
<span id="decision1" class="decision" onclick="changeScene(1)">Start Game</span>
<span id="decision2" class="decision" onclick="changeScene(2)"></span>
```

```
<style type="text/css">
  span.decision {
    font-weight:bold;
    border:thin solid #000000;
    padding:5px;
    background-color:#DDDDDD;
  }
</style>
```

A classe style de decisão é o que dá às decisões seu apelo visual.

O DOM fornece acesso à classe de estilo de um elemento através da propriedade `className` do objeto de nó.

```
alert(document.getElementById("decision1").className);
```

A propriedade className fornece acesso à classe de estilo de um elemento.

A propriedade className de um nó fornece acesso à classe de estilo.

Preste atenção!

Tente não confundir as classes de estilo CSS com as classes JavaScript.

As classes de estilo CSS e as classes JavaScript são muito diferentes. Uma classe de estilo CSS é uma coleção de estilos que pode ser aplicada em um elemento na página, ao passo que uma classe JavaScript é um modelo para criar os objetos JavaScript. Descobriremos os detalhes das classes JavaScript e dos objetos no Capítulo 10.

Como trocar as classes do estilo

Para mudar a aparência de um elemento usando um estilo de classe completamente diferente, simplesmente mude o nome de sua classe de estilo para uma classe de estilo CSS diferente.

A classe de estilo decisioninverse inverte o esquema de cor do texto de decisão.

```
document.getElementById("decision1").className = "decisioninverse";
```

O mesmo elemento de decisão, classes de estilo diferentes!

Uma nova classe de estilo é aplicada no elemento de decisão usando className.

Start Game

```
<style type="text/css">
  span.decisioninverse {
    font-weight:bold;
    font-color:#FFFFFF;
    border:thin solid #000000;
    padding:5px;
    background-color:#000000;
  }
</style>
```

Mudar a classe de estilo de um elemento usando a propriedade `className` muda imediatamente a aparência do elemento para a nova classe de estilo. Esta técnica pode ser usada para fazer mudanças visuais drásticas nos elementos em uma página com um esforço de codificação relativamente pequeno.

Aponte seu Lápis

Usando os eventos do mouse `onmouseover` e `onmouseout`, adicione código aos elementos de decisão `<span>` na Aventura do Boneco de Palito para que eles mudem as classes de estilo para obter um efeito de destaque quando o mouse passar.

Sugestão: A classe de estilo "mouse sobre" é denominada `decisionhover`.

```
<span id="decision1" class="decision" onclick="changeScene(1)"
  ------------------------------------------------------------
  ------------------------------------------------------------>Start Game</span>
<span id="decision2" class="decision" onclick="changeScene(2)"
  ------------------------------------------------------------
  ------------------------------------------------------------></span>
```

Aponte seu Lápis
Solução

Usando os eventos do mouse `onmouseover` e `onmouseout`, adicione código aos elementos de decisão `<span>` na Aventura do Boneco de Palito para que eles mudem as classes de estilo para obter um efeito de destaque quando o mouse passar.

Sugestão: A classe de estilo "mouse sobre" é denominada `decisionhover`.

A classe de estilo decisionhover é definida em resposta ao evento onmouseover.

```
<span id="decision1" class="decision" onclick="changeScene(1)"
  onmouseover="this.className = 'decisionhover'"
  onmouseout="this.className = 'decision'">Start Game</span>
<span id="decision2" class="decision" onclick="changeScene(2)"
  onmouseover="this.className = 'decisionhover'"
  onmouseout="this.className = 'decision'"></span>
```

Este evento é inicializado quando o ponteiro do mouse move-se sobre o elemento span.

O estilo de decisão sem destaque é armazenado em resposta ao evento onmouseout.

Este evento é inicializado quando o ponteiro do mouse sai do elemento span.

Opções de classe

Aplicar as classes de estilo no código da Aventura do Boneco de Palito produz duas aparências diferentes para os elementos de decisão: normal e destacada.

Não existem Perguntas Idiotas

P: Não posso simplesmente usar o CSS para criar botões que se destacam quando o mouse move-se sobre eles?

R: Sim. E, em muitos casos, esse é um modo melhor de criar botões "mouse sobre" porque o CSS é mais suportado do que o JavaScript nos navegadores como, por exemplo, em alguns dispositivos móveis. Porém, a Aventura do Boneco de Palito é um aplicativo JavaScript e faz todos os tipos de coisas que são impossíveis de fazer no CSS apenas. Portanto, neste caso não é uma obrigação usar o JavaScript para os botões de decisão da cena.

```
<style type="text/css">
  span.decision {
    font-weight:bold;
    border:thin solid #000000;
    padding:5px;
    background-color:#DDDDDD;
  }
</style>
```

A única diferença entre as duas classes de estilo é a cor do segundo plano.

```
<style type="text/css">
  span.decisionhover {
    font-weight:bold;
    border:thin solid #000000;
    padding:5px;
    background-color:#EEEEEE;
  }
</style>
```

Normal

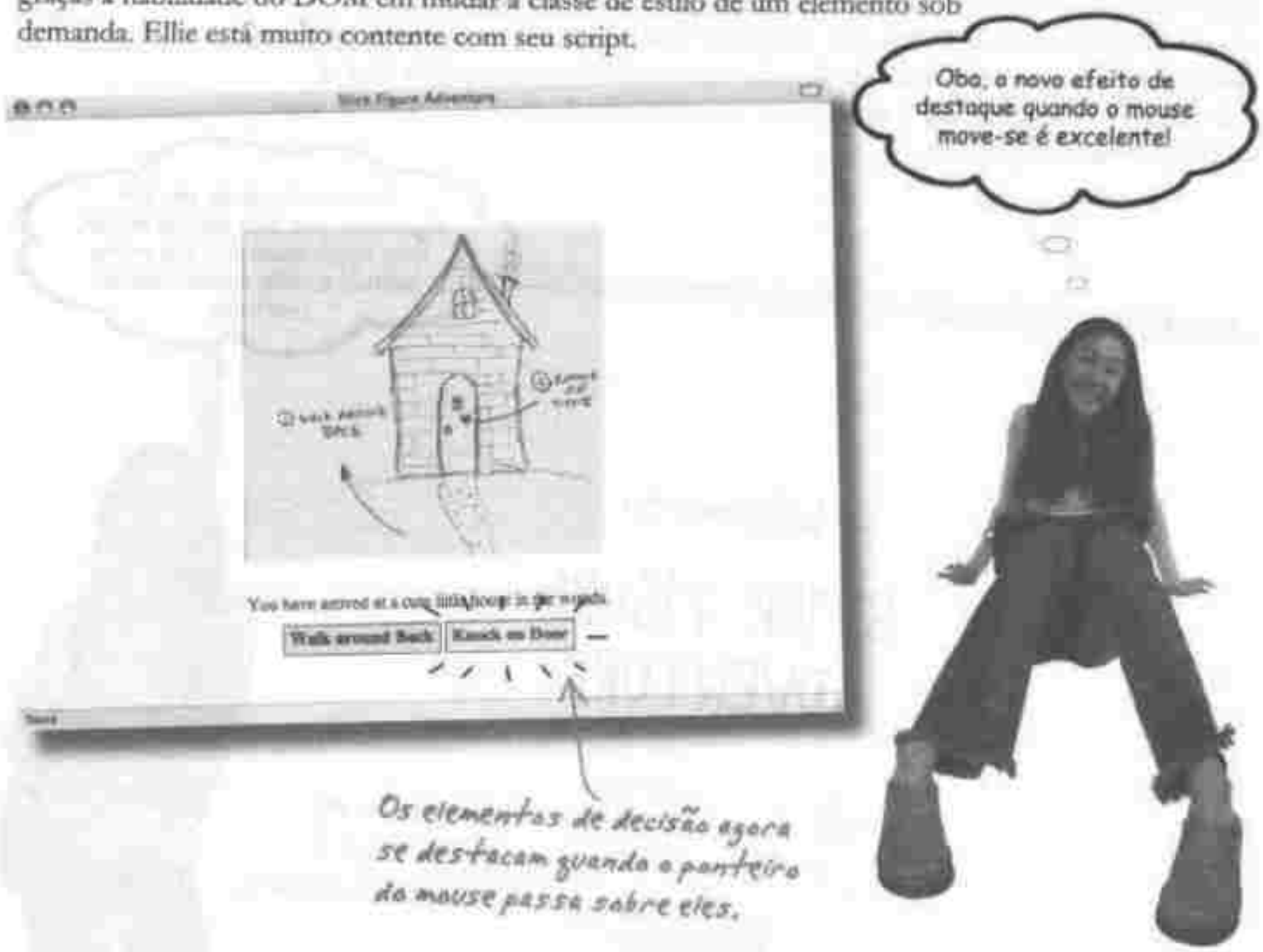
Walk around Back | Knock on Door

Destacado

Walk around Back | Knock on Door

Faça um 'test drive' das opções da aventura estilizada

A interface do usuário para a Aventura do Boneco de Palito agora está melhorada graças à habilidade do DOM em mudar a classe de estilo de um elemento sob demanda. Ellie está muito contente com seu script.



Não existem Perguntas Idiotas

P: Não me lembro dos eventos `onmouseover` e `onmouseout`. São eventos padrões?

R: Sim. Na verdade, existem muitos eventos JavaScript padrões que não exploramos. Mas o importante sobre os eventos é como você pode reagir a eles, mesmo quando não sabe necessariamente tudo sobre eles. No caso dos dois eventos do mouse, seus nomes são realmente tudo o que precisava saber para entender que um deles é inicializado quando o ponteiro do mouse move-se **sobre** um elemento e o outro inicializa-se quando o ponteiro do mouse **sai** de um elemento.

P: Por que não foi necessário usar `getElementById( )` no código que define a classe de estilo dos elementos de decisão?

R: Todo elemento no JavaScript é um objeto e no código HTML para um elemento temos acesso a esse objeto através da palavra-chave `this`. Portanto, no código da Aventura do Boneco de Palito, a palavra-chave `this` refere-se ao objeto-nó do elemento `span`. E esse é o mesmo objeto com a propriedade `className` que acessa sua classe de estilo. Portanto, mudar a classe de estilo envolve apenas definir `this.className`.

P: As classes de estilo são legais, mas eu realmente gostaria de mudar apenas uma propriedade do estilo. É possível?

R: Opa, que intuição! Há um problema chato na Aventura do Boneco de Palito que Ellie estava ansiosa para resolver. E acaba que envolve o uso do JavaScript e do DOM para manipular as propriedades do estilo individualmente...

As opções estão erradas: o botão vazio

Tem sido assim o tempo todo e até o momento Ellie enfrentou com sucesso. Mas é hora de prosseguir e endereçar a esquisitice associada às decisões vazias na Aventura do Boneco de Palito. Em algumas cenas, há apenas uma decisão viável, ainda que ambos os elementos de decisão sejam exibidos, como a tela aqui. É um pouco inquietante para o usuário ver um elemento de decisão interativo sem nenhuma informação nele.

Poder do cérebro

Quais as outras cenas têm o problema da opção vazia?
Quais as opções existem para corrigir isso?

Ajuste do estilo à la carte

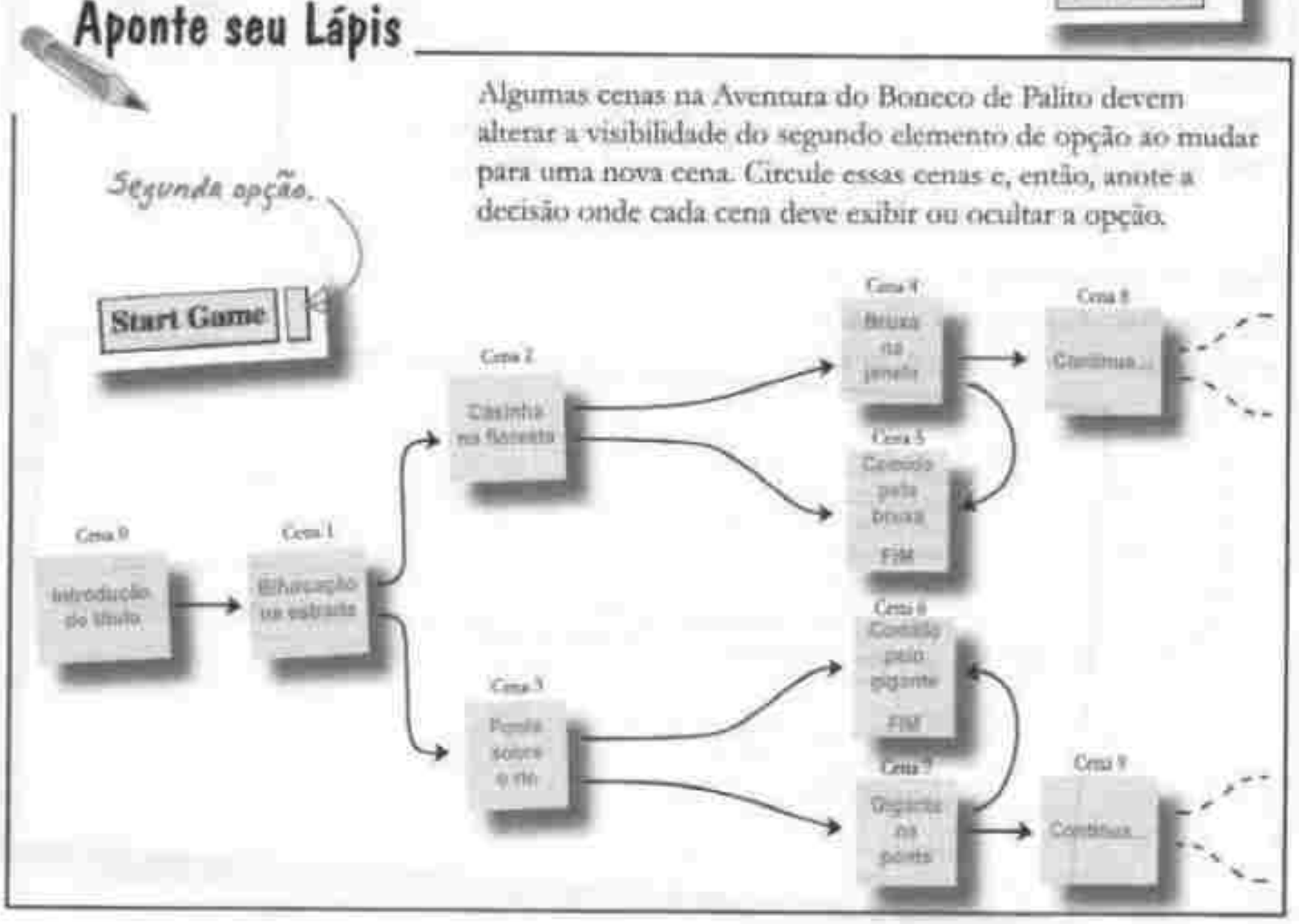
Algumas vezes, mudar a classe de estilo inteira de um elemento é demais. Para os momentos em que um pouco mais de granularidade é requerida, há o objeto `style`. O objeto `style` é acessível como uma propriedade de um objeto-nó e fornece acesso aos estilos individuais como propriedades. A propriedade do estilo `visibility` pode ser usada para mostrar e ocultar os elementos. No HTML da Aventura do Boneco de Palito, o segundo elemento de decisão pode ser ocultado inicialmente usando o seguinte código:

A propriedade de estilo de um nó fornece acesso às propriedades de estilo individuais.

```
<span id="decision2" class="decision" onclick="changeScene(2)"
  onmouseover="this.className = 'decisionhover'"
  onmouseout="this.className = 'decision'"
  style="visibility:hidden"></span>
```

Deste ponto em diante, mostrar e ocultar o elemento é apenas uma questão de definir a propriedade de estilo `visibility` para `visible` ou hidden.

```
document.getElementById("decision2").style.visibility = "visible";
document.getElementById("decision2").style.visibility = "hidden";
```

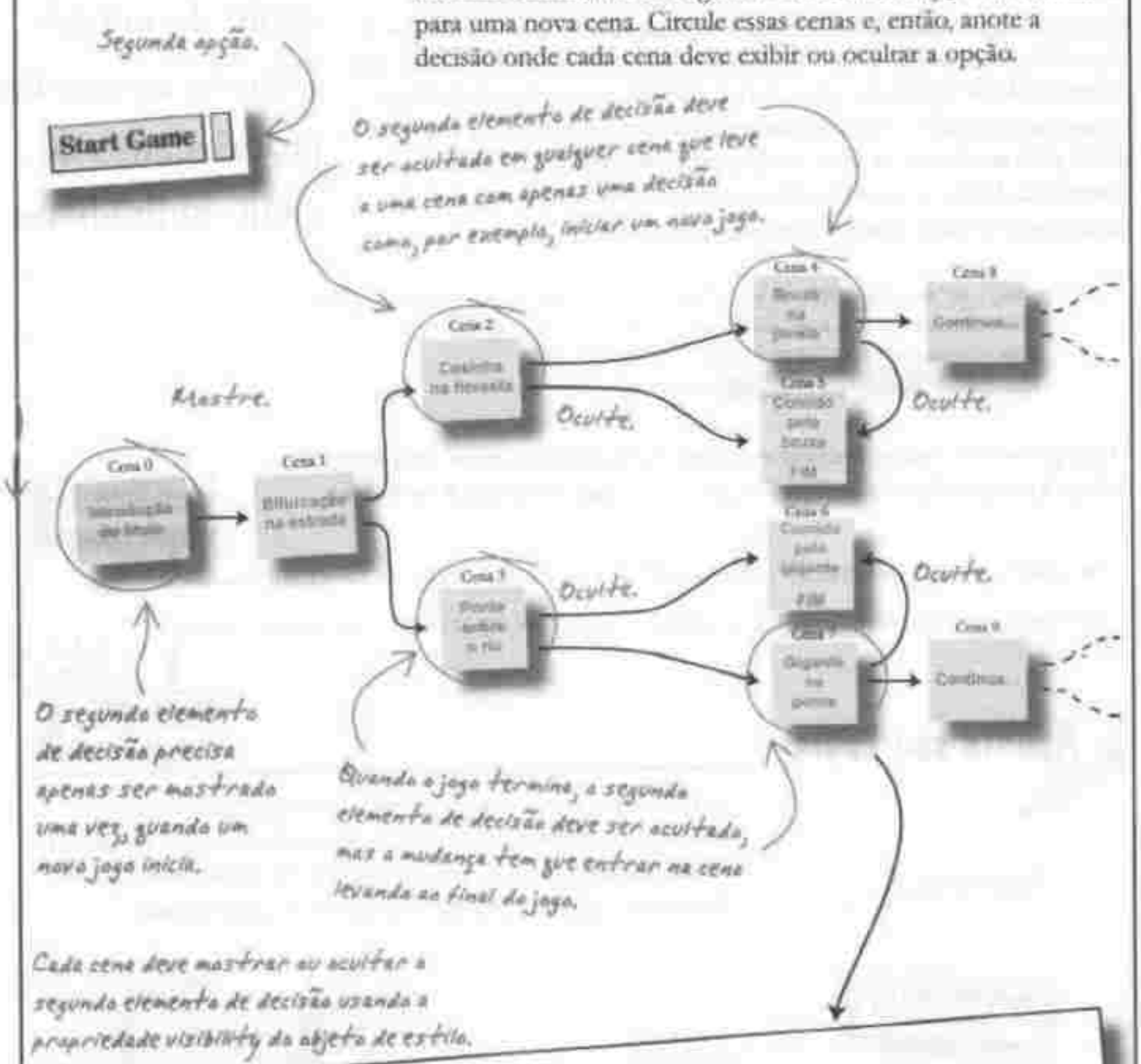


Aponte seu Lápis

Algumas cenas na Aventura do Boneco de Palito devem alterar a visibilidade do segundo elemento de opção ao mudar para uma nova cena. Circule essas cenas e, então, anote a decisão onde cada cena deve exibir ou ocultar a opção.

Aponte seu Lápis
Solução

Algumas cenas na Aventura do Boneco de Palito devem alterar a visibilidade do segundo elemento de opção ao mudar para uma nova cena. Circule essas cenas e, então, anote a decisão onde cada cena deve exibir ou ocultar a opção.



Cada cena deve mostrar ou ocultar o segundo elemento de decisão usando a propriedade visibility do objeto de estilo.

```
...
  case 7:
    if (decision == 1) {
      curScene = 6
      message = "Sorry, you became the troll's tasty lunch."
      decision1 = "Start over"
      decision2 = "";

      // Oculte a segunda decisão
      document.getElementById("decision2").style.visibility = "hidden";
    }
    else {
      curScene = 9;
      decision1 = "?";
      decision2 = "";
    }
    break;
...
```

Pontos de Bala

- A propriedade do nó `className` faz grandes mudanças de estilo alterando a **classe de estilo inteira** de um nó.
- A propriedade do nó `style` faz pequenas mudanças de estilo fornecendo acesso às **propriedades de estilo individuais** de um nó.
- Uma classe de estilo CSS não tem nenhuma relação com uma classe JavaScript – são coisas completamente diferentes.
- Os elementos em uma página podem ser exibidos ou ocultados dinamicamente usando a propriedade do estilo `visibility` do objeto do elemento.

A propriedade de estilo para exibir pode conseguir um efeito exibir/ocultar parecido definindo-a para display:none (ocultar) ou display:block (exibir).

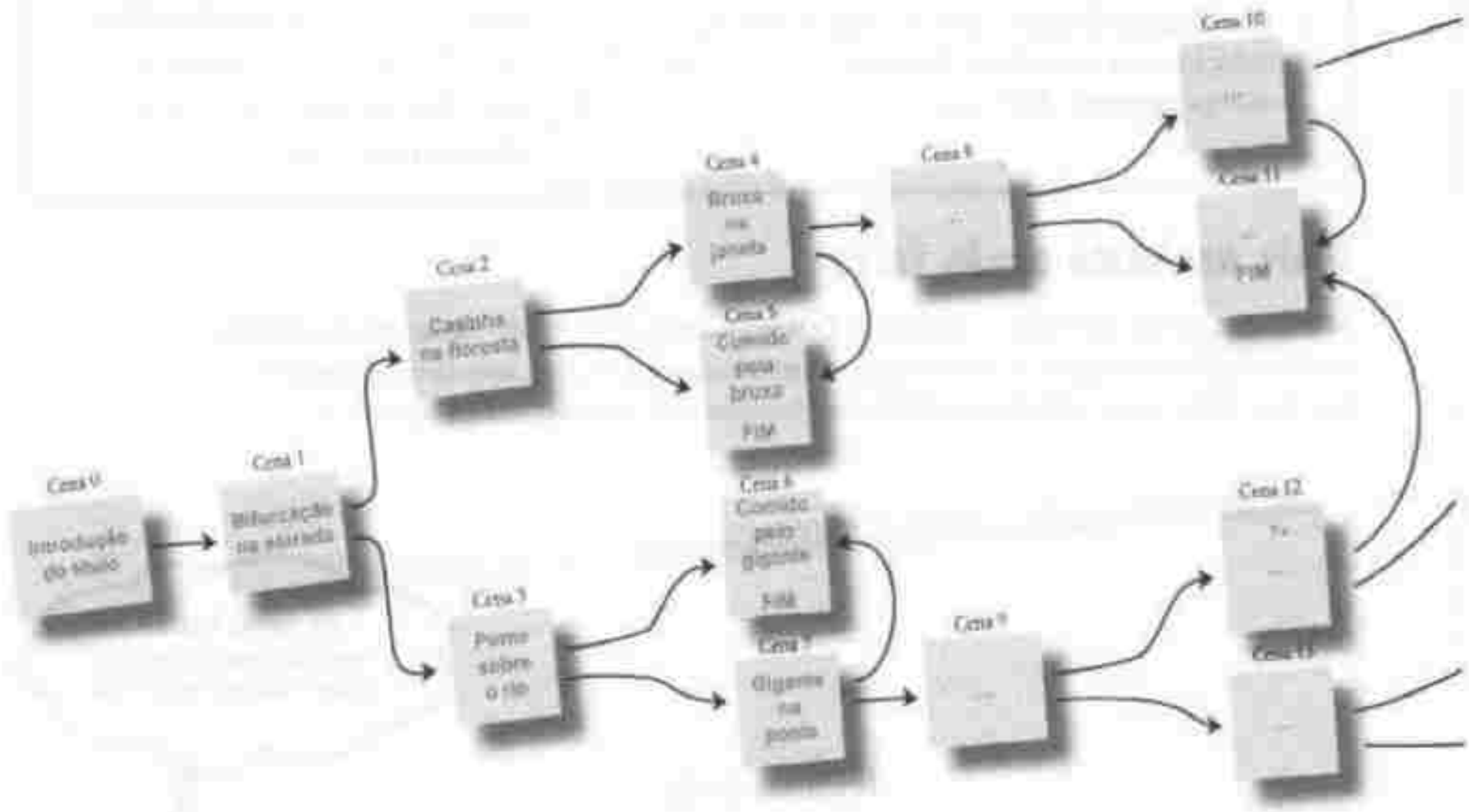
Mais nenhuma opção falsa

Manipular estilos individuais usando o DOM permite ao segundo elemento de decisão ser mostrado e ocultado seletivamente. O resultado final é uma interface do usuário que faz muito mais sentido agora que os elementos de decisão vazios acabaram.

O segundo elemento de decisão é agora ocultado quando não é necessário como, por exemplo, o título da cena.

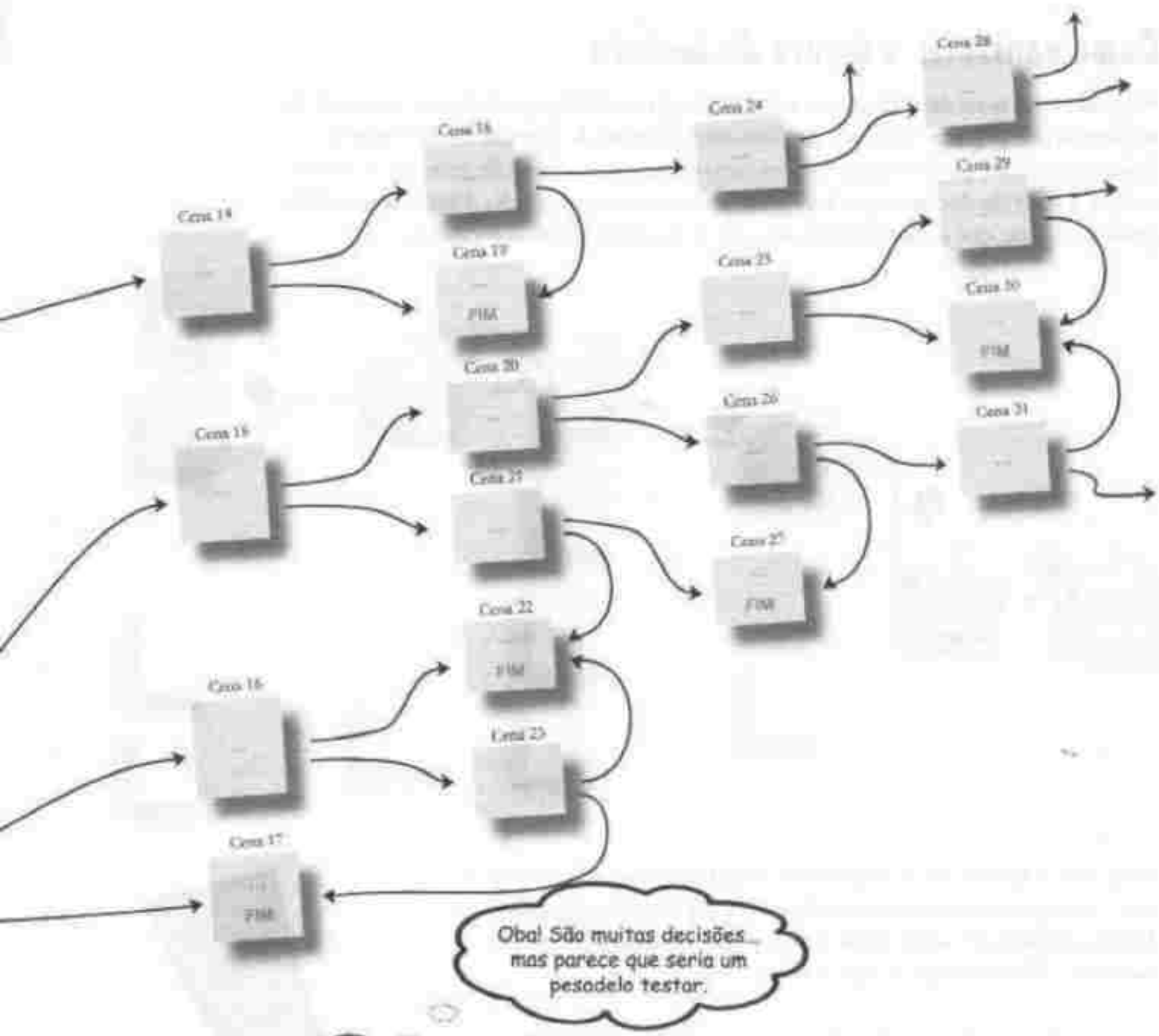
Mais opções, mais complexidade

Ellie prevê o crescimento da história Aventura do Boneco de Palito com grande velocidade para mostrar todos os tipos de cenas novas interessantes e decisões. Há modos em que o DOM pode decompor-se para ajudar a lidar com a complexidade de uma narrativa da Aventura do Boneco de Palito muito mais profunda.



Aventura mais profunda = Árvore de decisão maior!

* A última versão da Aventura do Boneco de Palito está on-line e esperando por sua assistência na codificação. Faça o download em www.altabooks.com.br, se já não tiver feito.



Certamente, as grandes histórias podem transformar-se em grandes problemas sem um meio de testar a árvore de decisão.

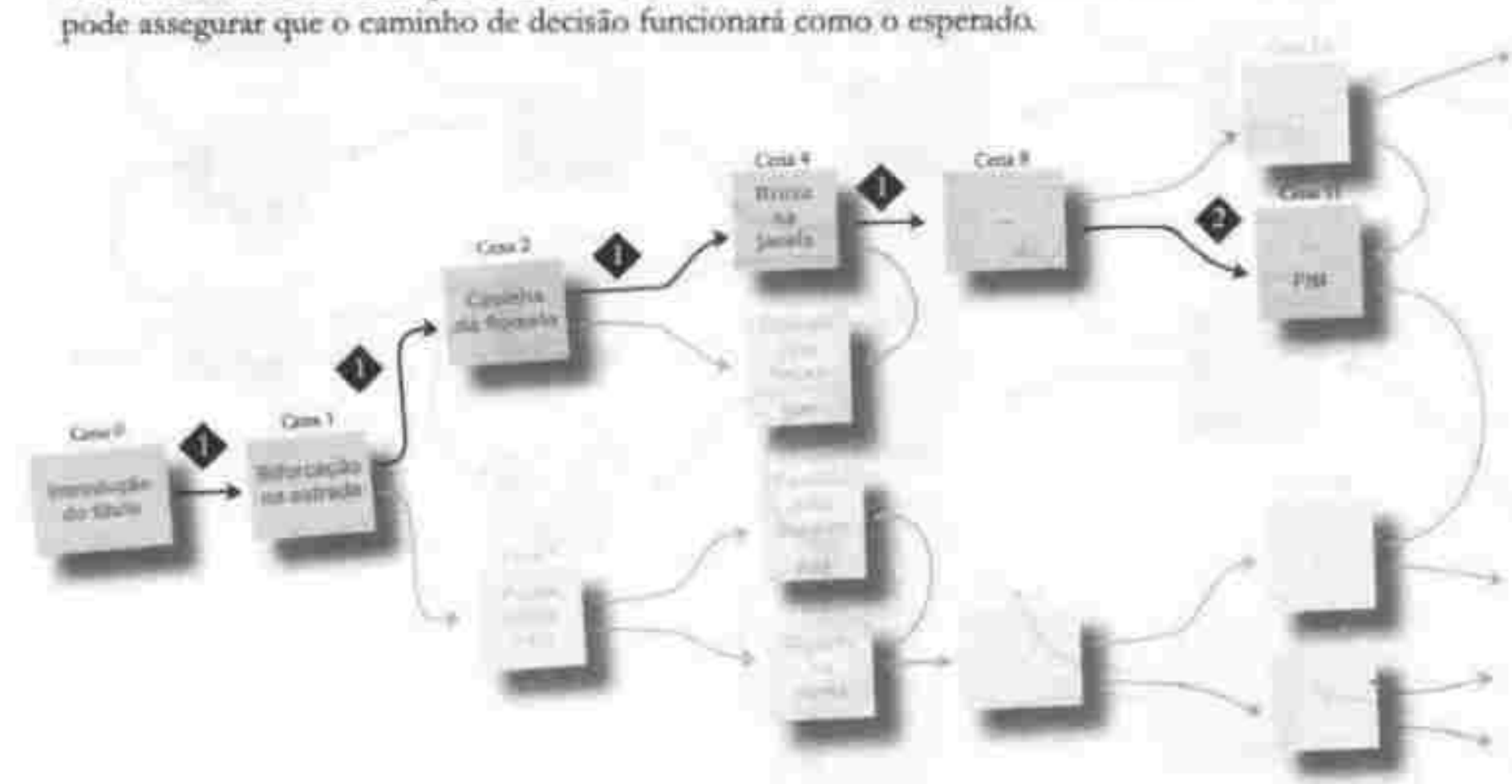
Quando a história continua a se desdobrar com muitas cenas e decisões, fica cada vez mais difícil testar sua lógica e assegurar que cada caminho de decisão levará ao lugar certo. A Aventura do Boneco de Palito precisa desesperadamente de um modo de analisar os caminhos na história.

Poder do cérebro

O que você acha que poderia ser o melhor modo de criar caminhos de teste em tal árvore de decisão monstruosa?

Como controlar a árvore de decisão

Parecido com o recurso histórico em um navegador Web, que mantém controle da seqüência das páginas visitadas, um recurso histórico de decisão na Aventura do Boneco de Palito pode ser usado para testar e depurar a linha da história. A idéia é revelar a série de decisões que leva a um determinado resultado. Fazendo isso, Ellie pode assegurar que o caminho de decisão funcionará como o esperado.



O histórico de decisão é construído como uma lista das opções e das cenas que ocorrem em um determinado caminho na história. Então, o histórico serve como um depurador da história que permite a Ellie rastrear as opções e as cenas.

Início Cena 0 – Introdução do título

1 Cena 1 – Bifurcação na estrada.

A decisão é escolhida...

1 Cena 2 – Casinha na floresta.

1 Cena 4 – Bruxa na janela.

...que leva à cena 4.

1 Cena 8 - ...

2 Cena 11 - ...

Fim

Cada cena percorrida é adicionada ao histórico de decisão, junto com a decisão tomada para chegar neste ponto.

Poder do cérebro

Quais os tipos de mudanças que são requeridos para a página Web da Aventura do Boneco de Palito suportar um recurso histórico de decisão?

Como construir o histórico de decisão em HTML

De uma perspectiva HTML, o código para o histórico de decisão não é terrivelmente complexo: um elemento `div` e um parágrafo de texto para cada decisão é tudo o que é necessário.

```
<div id="history">
  <p>Decision 1 -> Scene 1 : Fork in the road.</p>
  <p>Decision 1 -> Scene 2 : Little house in the woods.</p>
  <p>Decision 1 -> Scene 4 : Witch in window.</p>
  ...
</div>
```

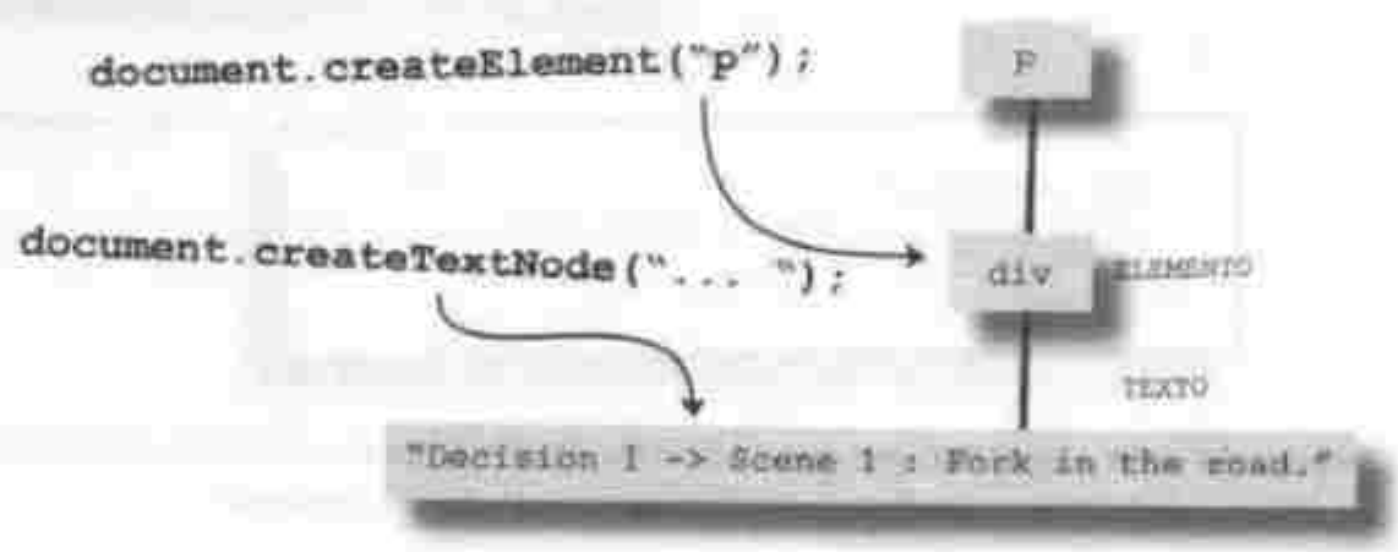
Cada elemento p contém uma decisão no histórico de decisão.

Um recurso histórico de decisão na Aventura Boneco de Palito pode ser uma ferramenta de depuração da história muito útil.

Tudo que resta é escrever algum JavaScript para usar o DOM para gerar o histórico de decisão como uma coleção de nós.

O DOM pode criar qualquer elemento HTML à vontade, inclusive parágrafos de texto.

Na verdade, você pode. E envolve outro método do objeto do documento, `createElement( )`, que pode ser usado para criar qualquer elemento HTML. A idéia é criar um novo elemento contêiner usando `createElement( )` e então, adicionar-lhe conteúdo de texto criando um nó-filho de texto com `createTextNode( )`. O resultado final é um novo desvio de nós inteiramente novo enxertado na árvore de nós de uma página.



Como construir o código HTML

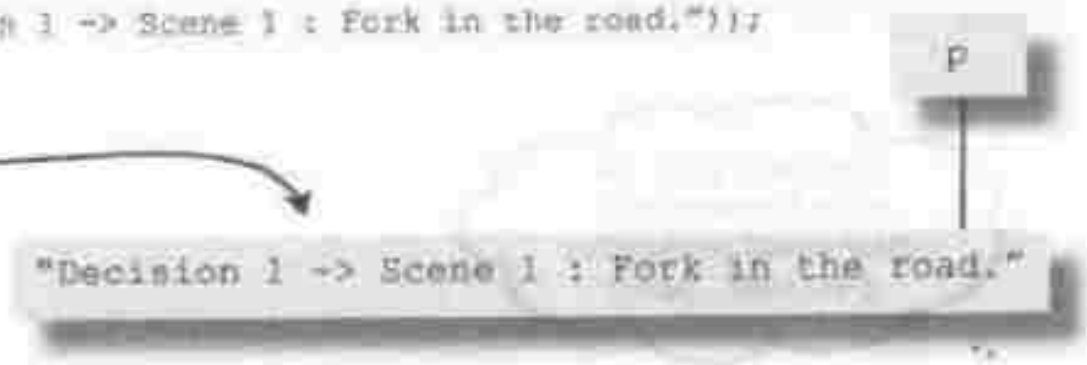
Criar um novo elemento com o método `createElement( )` apenas requer o nome da tag. Portanto, criar um elemento (p) do parágrafo simplesmente significa ter que chamar o método com um argumento "p", colocando o elemento resultante que é criado.

Iniciamos com um novo elemento p flutuando no espaço.

```
var decisionElem = document.createElement("p");
```

Neste ponto, há um novo elemento do parágrafo sem conteúdo e ainda não faz parte de nenhuma página também. Portanto, para adicionar o conteúdo de texto ao elemento, crie um nó de texto e, então, adicione-o como um filho do novo nó p.

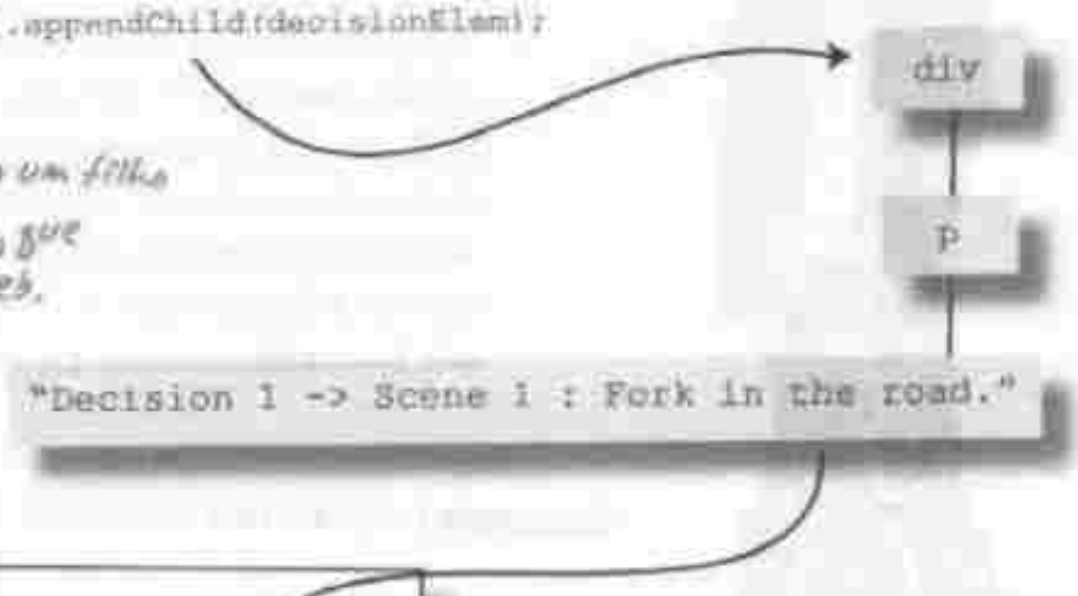
```
decisionElem.appendChild("Decision 1 -> Scene 1 : Fork in the road.");
```



O elemento p ainda está flutuando no espaço, mas agora tem algum conteúdo de texto graças a um novo nó-filho de texto.

A última etapa é adicionar o novo elemento do parágrafo à página como um filho do elemento `div` do histórico.

```
document.getElementById("history").appendChild(decisionElem);
```



O elemento p é adicionado como um filho de um elemento div existente, que mescla o parágrafo na página Web.

```
...
  <div id="history">
    <p>Decision 1 -> Scene 1 : Fork in the road.</p>
  </div>
...
```

Repetindo estas etapas sempre que cada cena é percorrida na Aventura do Boneco de Palito, um histórico de decisão poderá ser criado dinamicamente.

Pontos de Bala

- Qualquer elemento HTML pode ser criado usando o método `createElement( )` do objeto `document`.
- Para adicionar conteúdo de texto a um elemento, um nó-filho de texto deve ser criado e anexado ao elemento.
- Adicionando e removendo com cuidado os nós na árvore DOM, uma página Web pode ser desmontada e remontada à vontade.

Aponte seu Lápis

Adicione código à função `changeScene( )` para suportar o recurso histórico de decisão na Aventura do Boneco de Palito. Sugestão: Você precisará adicionar um novo elemento do parágrafo com um nó-filho de texto ao elemento histórico de decisão quando a cena atual não for a Cena 0 e limpar o histórico de decisão se a Cena for 0.

```
function changeScene(decision) {
...
// Atualize o histórico de decisão
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
}
```

Aponte seu Lápis Solução

Adicione código à função `changeScene( )` para suportar o recurso histórico de decisão na Aventura do Boneco de Palito. Sugestão: Você precisará adicionar um novo elemento do parágrafo com um nó-filho de texto ao elemento histórico de decisão quando a cena atual não for a Cena 0 e limpar o histórico de decisão se a Cena for 0.

Anexe o novo nó de texto ao novo elemento do parágrafo.

Obtenha o div do histórico usando seu ID.

A função changeScene( ) já tem uma variável local denominada decision, portanto essa variável deve receber outro nome.

```
function changeScene(decision) {
...
// Atualize o histórico de decisão
var history = document.getElementById("history");
if (curScene != 0) {
  // Adicione a última decisão ao histórico
  var decisionElem = document.createElement("p");
  decisionElem.appendChild(document.createTextNode("decision " + decision +
    " -> Scene " + curScene + " : " + message));
  history.appendChild(decisionElem);
  }
  else {
  // Limpe o histórico de decisão
  while (history.firstChild)
    history.removeChild(history.firstChild);
  }
}
```

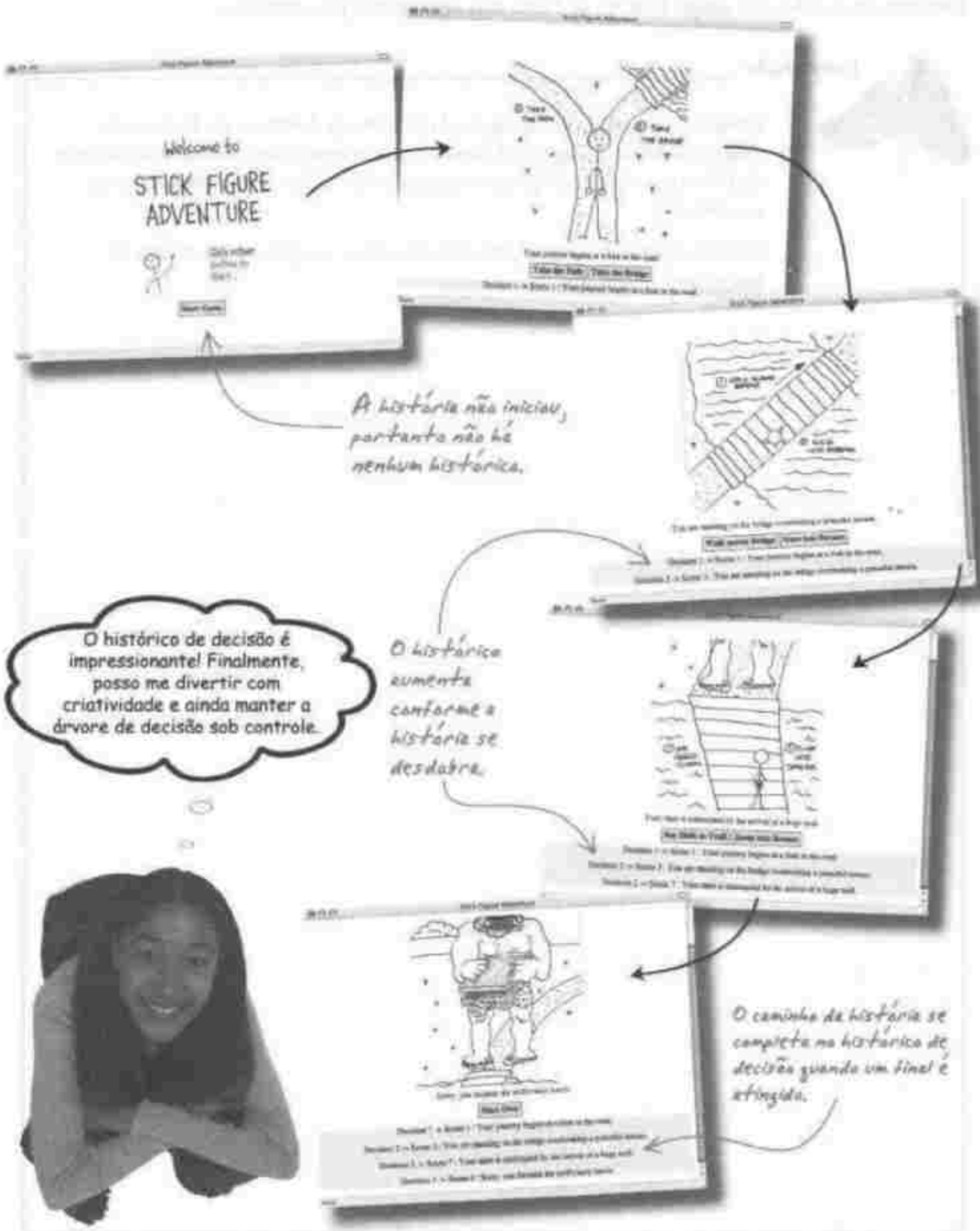
Limpe o div do histórico removendo todos os seus filhos.

Anexe o elemento do parágrafo a div para adicioná-lo à página.

Crie um novo nó de texto com as informações do histórico de decisão.

Como investigar a história da aventura

O recurso histórico de decisão na Aventura do Boneco de Palito agora torna possível controlar com cuidado a lógica da história quando ela se desdobra.



Uma viagem longa e estranha...

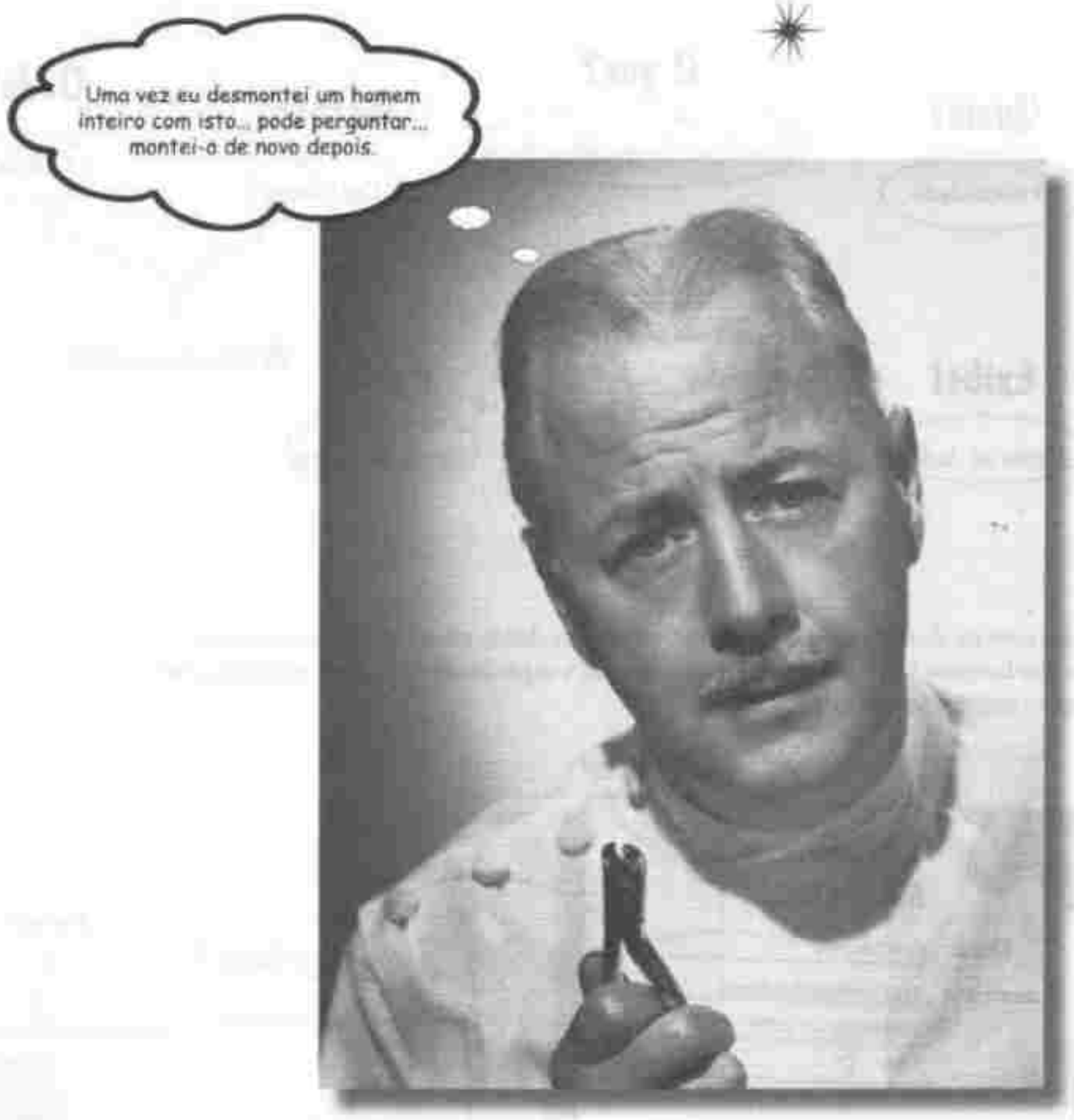
É hora de flexionar seu músculo criativo expandindo a história da Aventura do Boneco de Palito para algo que vale a pena fazer alguma depuração séria do histórico de decisão. Seu amigo Boneco de Palito está aguardando por aventura...

Exercício

Imagine sua própria continuação da história da Aventura do Boneco de Palito e adicione código para incorporá-la no aplicativo Aventura do Boneco de Palito para que você possa compartilhá-la on-line como uma aventura interativa.

Não há nenhuma solução para este exercício... simplesmente divirta-se imaginando aventuras!

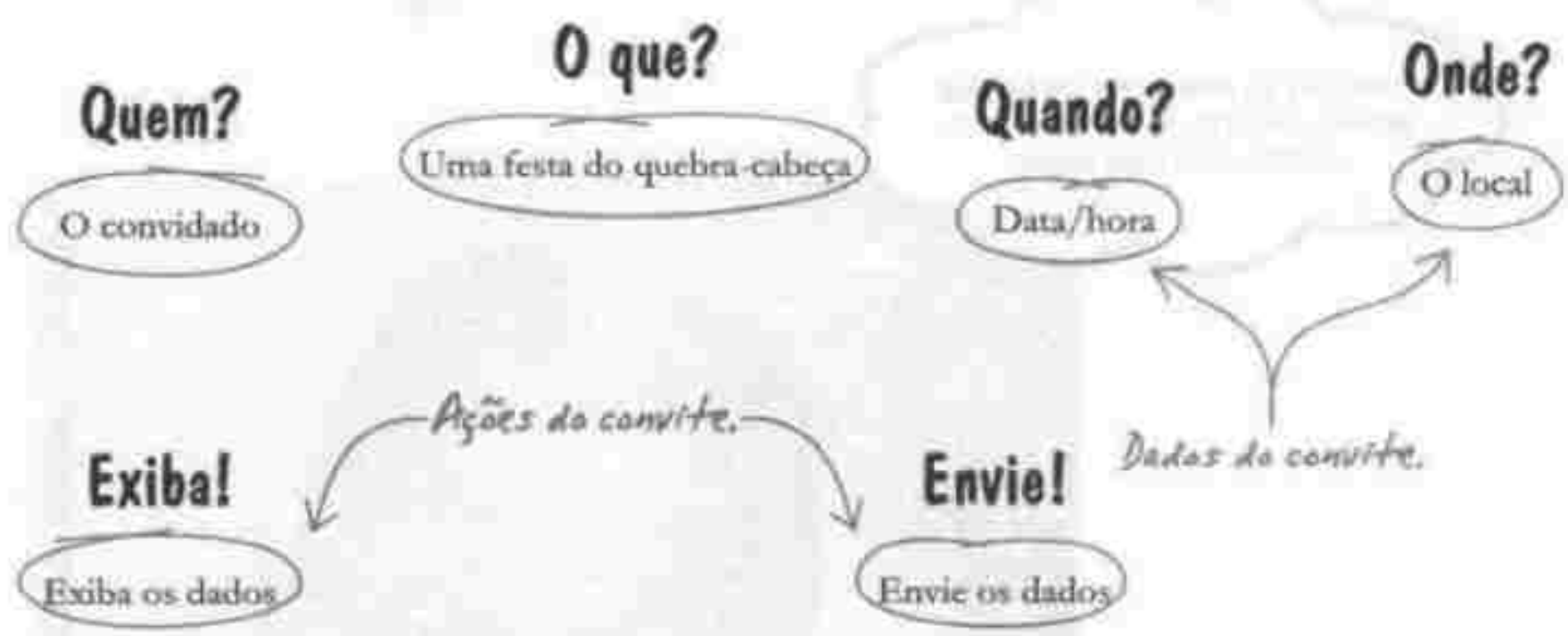
Objetos como Frankendados



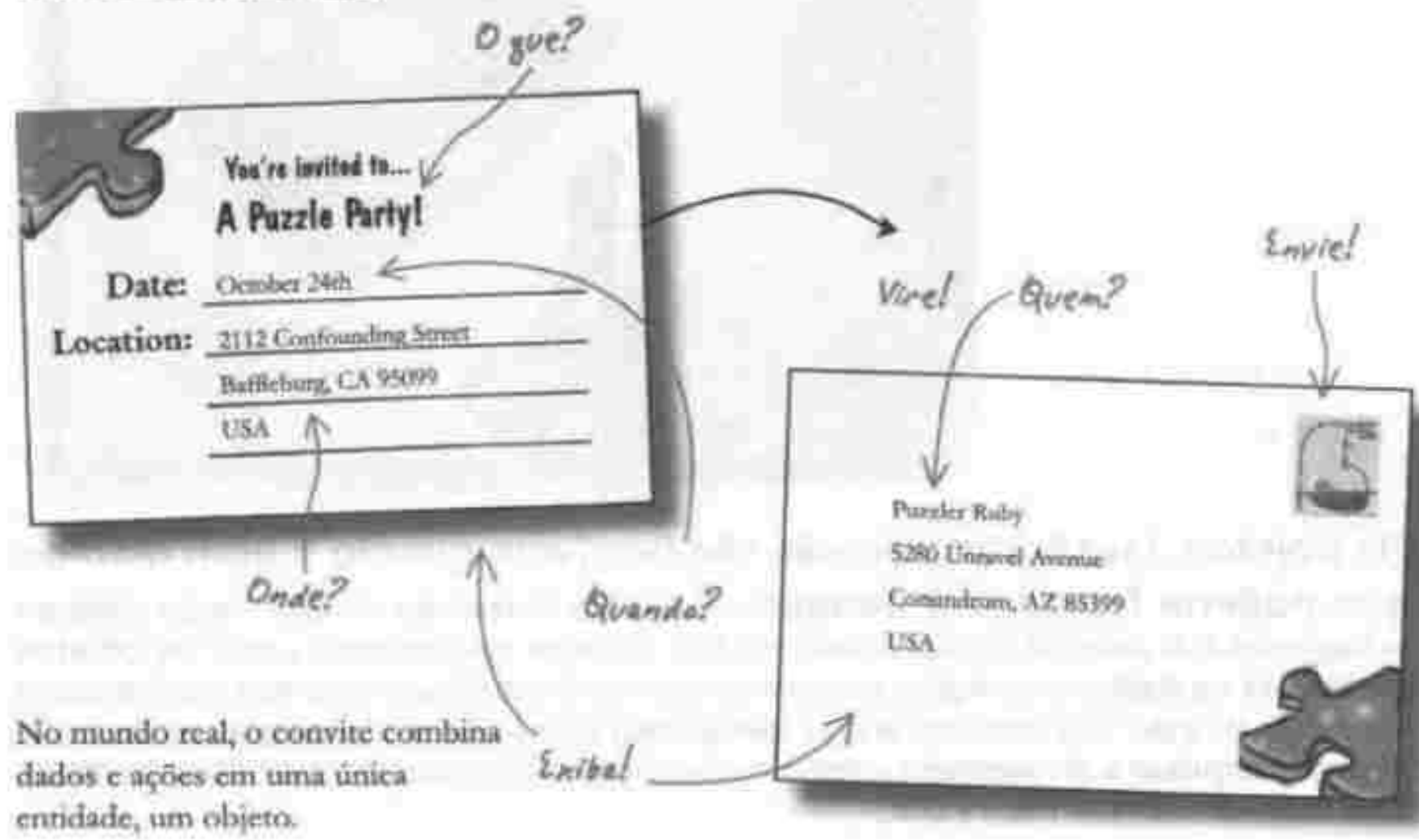
Os objetos JavaScript não são tão horríveis quanto o bom médico que poderia fazer você pensar. Mas eles são interessantes no sentido de que combinam os fragmentos e as partes da linguagem JavaScript para que sejam mais poderosos juntos. **Os objetos combinam os dados com ações** para criar um novo tipo de dados que é muito mais "ativo" do que os dados vistos até então. Você acaba com ***arrays que podem classificar a si mesmos, strings que podem pesquisar a si mesmas*** e scripts que podem crescer pêlos e uivar para a lua! Tudo bem, talvez a última opção não, mas você pegou a idéia...

Uma festa produzida pelo JavaScript

Há uma festa e você é responsável pelos convites. Portanto, a primeira pergunta é quais informações entram em um convite de festa perfeito?



Um convite de festa para o JavaScript exibiria os dados como variáveis e as ações como funções. O problema é: no mundo real, a capacidade de separar os dados e as ações realmente não existe.



No mundo real, o convite combina dados e ações em uma única entidade, um objeto.

Dados + ações = objeto

Nem sempre você tem que trabalhar com dados e ações como coisas separadas no JavaScript. Na verdade, os **objetos** JavaScript combinam os dois em uma estrutura de dados inteiramente única que **armazena** dados e **atua** neles. Essa funcionalidade permite ao JavaScript aplicar o pensamento real nos scripts. Portanto, você pode pensar em termos de "coisas", em oposição a separar os dados e as ações.

Ao ver o convite da festa, em termos de objeto JavaScript, pode conseguir isto:

Dados

```
var who;
var what;
var when;
var where;
```

+

Ações

```
function display(what, when, where) {
  ...
}
function deliver(who) {
  ...
}
```

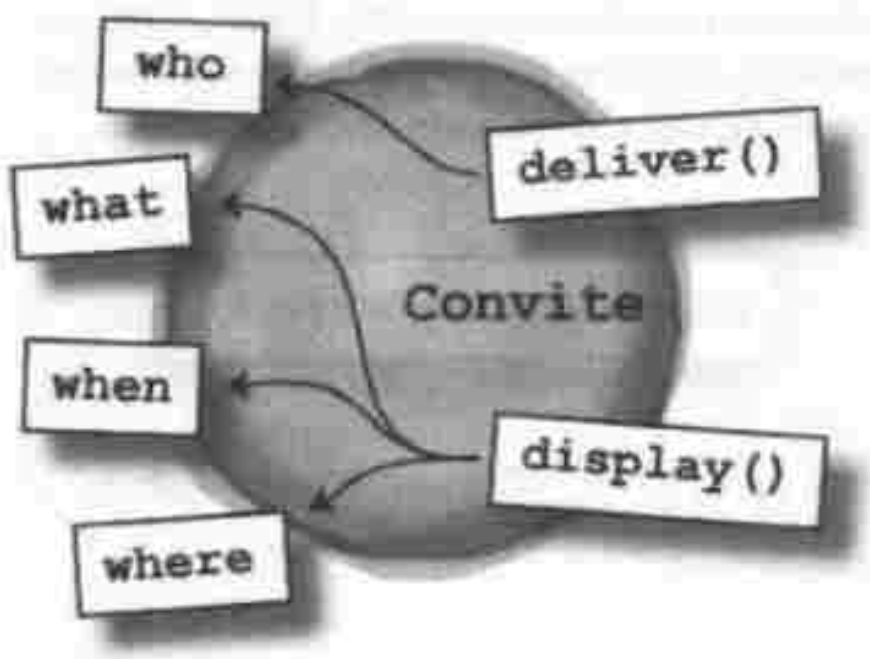
=

Objeto

```
var who;
var what;
var when;
var where;
function display() {
  ...
}
function deliver() {
  ...
}
```

Fora de um objeto, os dados têm que ser transmitidos para as funções como argumentos.

Dentro do objeto convite, os dados e as funções, agora, coexistem e têm ligações mais íntimas do que tinham fora dele. Mais especificamente, as funções colocadas em um objeto podem acessar as variáveis no objeto sem ter que transmitir as variáveis para as funções como argumentos.



Os objetos ligam as variáveis e as funções dentro de um conteiner de armazenamento.

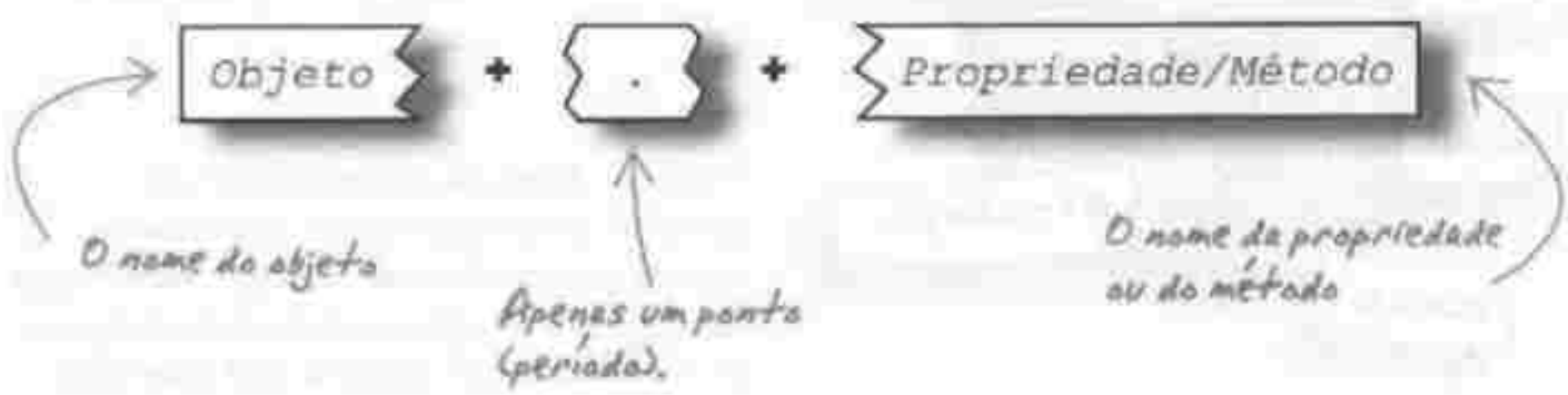
Os dados dentro do objeto convite são acessíveis pelas funções, mas ocultados do mundo exterior. Portanto, o objeto serve como um contêiner que armazena os dados **e** liga-os ao código que pode executar uma ação neles.

Um objeto possui seus dados

Quando as variáveis e as funções são colocadas dentro de um objeto, elas são referidas como **membros do objeto**. Mais especificamente, as variáveis são chamadas de **propriedades** do objeto e as funções são chamadas de **métodos** do objeto. Ainda armazenam dados e executam ações nos dados, apenas fazem isso no contexto de um objeto específico.

As propriedades e os métodos são os equivalentes do objeto para as variáveis e as funções.

As propriedades e os métodos são "possuídos" por um objeto, o que significa que são armazenados dentro do objeto de modo muito parecido com os dados armazenados em um array. Porém, diferente dos arrays, você geralmente acessa as propriedades e os métodos do objeto usando um operador especial chamado **operador ponto**.



O membro do objeto refere-se com um ponto

O operador ponto estabelece uma referência entre uma propriedade ou o método e o objeto a qual ele pertence. É como os primeiros nomes das pessoas que dizem a você quem **elas** são, mas seus últimos nomes informam a você a qual **família** elas pertencem. O mesmo é para os objetos – um nome da propriedade informa a você o que é a propriedade, ao passo que o **nome do objeto** informa a qual objeto a propriedade pertence. E o operador ponto conecta os dois.

O operador ponto refere-se a uma propriedade ou a um método de um objeto.

Agora, é possível, de fato, reunir os dados para um objeto convite JavaScript usando propriedades e o operador ponto:

Nome do objeto.

Um ponto!

Nome da propriedade.

```
invitation.who = "Puzzler Ruby";
invitation.what = "A puzzle party!";
invitation.when = "October 24th";
invitation.where = "2112 Confounding Street";
```

O operador ponto é usado para acessar cada propriedade.

Lembre-se de que como os dados e as ações são partes do mesmo objeto, você não tem que transmitir nada a um método para ele ser capaz de usar os dados. Isso torna bem simples tomar uma ação no objeto convite:

```
invitation.deliver();
```

Nome do método.

Nome do objeto.

Exercício

O convite da festa não tem uma propriedade RSVP que permite aos convidados responder se eles irão ou não à festa. Escreva o código para adicionar uma propriedade `rsvp` ao convite Puzzler Ruby (ela planeja comparecer) e, então, chame o método `sendRSVP( )` para enviar a resposta.

. . . . . .

. . . . . .

Solução do Exercício

O convite da festa não tem uma propriedade RSVP que permite aos convidados responder se eles irão ou não à festa. Escreva o código para adicionar uma propriedade `rsvp` ao convite Puzzler Ruby (ela planeja comparecer) e então, chame o método `sendRSVP( )` para enviar a resposta.

O operador ponto refere-se à propriedade e ao método do objeto convite.

```
invitation.rsvp = "attending";

invitation.sendRSVP( );
```

Isto também poderia ser uma propriedade booleana onde true significa que a pessoa está comparecendo, false significa que não.

Não existem

Perguntas Idiotas

P: O que é exatamente um objeto? Ele tem um tipo de dado?

R: Sim, os objetos têm um tipo de dado. Um objeto é uma coleção nomeada de propriedades e métodos. Ou mais exatamente, os objetos são um tipo de dados. Os outros tipos de dados aprendidos incluem: número, texto e booleano. Eles são conhecidos como tipos de dados *primitivos* porque representam uma única parte da informação. Os objetos são considerados tipos de dados *complexos* porque englobam diversas partes de dados. Você pode adicionar um "objeto" como um quarto tipo de dados à lista dos tipos primitivos já conhecida (número, string e booleano). Portanto, qualquer objeto criado ou qualquer objeto JavaScript predefinido usado tem um tipo de dados do objeto.

P: Eu não poderia simplesmente usar variáveis e funções globais ao invés das propriedades e métodos do objeto? As funções podem acessar as variáveis globais muito bem, certo?

R: Sim, podem. O problema é que não há nada para impedir que qualquer outro código acesse as variáveis globais também. Isso é problemático porque você sempre deseja experimentar e limitar a exposição dos dados apenas ao código que realmente precisa acessar os dados. Isso ajuda a evitar que os dados sejam mudados sem querer por outro código.

Infelizmente, o JavaScript não permite, atualmente, que você de fato impeça que uma propriedade do objeto seja acessada por um código externo. E há situações em que, especificamente, deseje que uma propriedade do objeto seja acessada diretamente. Porém, a idéia é que você coloque os dados em um objeto para associá-los logicamente ao objeto. Uma parte dos dados ligada a um objeto tem muito mais contexto e significado do que uma parte dos dados flutuando livremente em um script (uma variável global).

P: Já vi a notação do objeto com o operador ponto usada várias vezes. Realmente eu estava usando objetos todo este tempo?

R: Sim. Você achará que de fato é bem difícil usar o JavaScript sem usar objetos e que é por isso que o próprio JavaScript é realmente uma grande coleção de objetos. Por exemplo, a função `alert()` é tecnicamente um método do objeto `window`, que significa que pode ser chamada com `window.alert()`. O objeto `window` representa a janela do navegador e não tem que ser referido explicitamente como um objeto, por isso é que você pode ter sucesso com o `alert()` sozinho.

P: Tudo bem, isto é realmente confuso. Portanto, está dizendo que as funções são na verdade métodos?

R: Sim, embora possa ser confuso considerar as funções assim. Você já sabe que uma função é uma parte do código que pode ser chamada por outro código pelo nome. Um método é apenas uma função que foi colocada em um objeto. A confusão surge ao perceber que toda função, de fato, pertence a um objeto.

Portanto, `alert( )` é uma função e um método, o que explica por que pode ser chamada como uma função ou como um método – a maioria dos métodos tem que ser chamada como um método usando a notação da função. Na verdade, toda função JavaScript pertence a um objeto, assim tornando-a um método. E em muitos casos, esse objeto é o objeto `window` do navegador. Como esse objeto deverá ser o objeto padrão se nenhum objeto for especificado para uma chamada do método como, por exemplo, `alert( )`, será bom considerar esses métodos como funções. Sua propriedade pertencendo ao objeto `window` é acidental, pois não tem nenhuma conexão lógica com o objeto.

Pontos de Bala

- Os objetos são um tipo especial de **estrutura de dados** que combinam os dados com o **código que atua neles**.
- Em termos práticos, um objeto é realmente apenas variáveis e funções combinadas em uma **única estrutura**.
- Quando colocadas em um objeto, as variáveis ficam conhecidas como **propriedades**, ao passo que as funções ficam conhecidas como **métodos**.
- As propriedades e os métodos são referidos fornecendo o nome do **objeto** seguido de um **ponto** seguido do **nome** da propriedade ou do método.

Um blog para os amantes de quebra-cabeças de cubo

Na outra ponta do convite da festa está Ruby, uma apaixonada por quebra-cabeças de cubo que mal pode esperar para se reunir com seus outros amigos de quebra-cabeças. Mas Ruby tem mais coisas em mente do que apenas ir às festas – ela deseja criar um blog no qual deseja compartilhar seu amor por quebra-cabeças de cubo com o mundo. Ela está pronta para começar a compartilhar seu saber cubista no YouCube!

Ruby ouviu dizer que o JavaScript suporta objetos personalizados como um meio de criar um código mais robusto e basicamente mais gerenciável. Ela também ouviu falar que muitos blogs, possivelmente, ficam desatualizados porque os bloggers ficam cansados de mantê-los. Portanto, Ruby deseja iniciar seu blog com o pé direito construindo o YouCube como um script **baseado em objetos** usando objetos personalizados que a levarão longe no futuro dos quebra-cabeças.

YouCube orientado a objetos = Mais tempo do cubo!

Como desconstruir o YouCube

Ruby, atualmente, tem um diário escrito à mão e ela lê blogs o suficiente para saber que o dela precisará consistir em datas e texto, mas não consegue saber como armazená-los usando o JavaScript. Ela só sabe que continua escrevendo as entradas do diário de seu cubo (logo será um blog) à mão!

O YouCube escrito à mão.

08/14/2008
Got the new cube I ordered. It's a real pearl.

08/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.

08/16/2008
Managed to get a headache toiling over the new cube.
Gotta nap.

08/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could be a beast.

Cada entrada consiste em uma data combinada com uma string de texto.

A data da entrada.

O corpo do texto da entrada.

O quebra-cabeça de cubo favorito de Ruby.

Ruby precisa desesperadamente de um modo simples de armazenar e acessar os diversos pares de informação (data + texto). Certamente, parece muito para o que os objetos JavaScript têm a oferecer... combinar diversas partes de informação em uma única entidade.

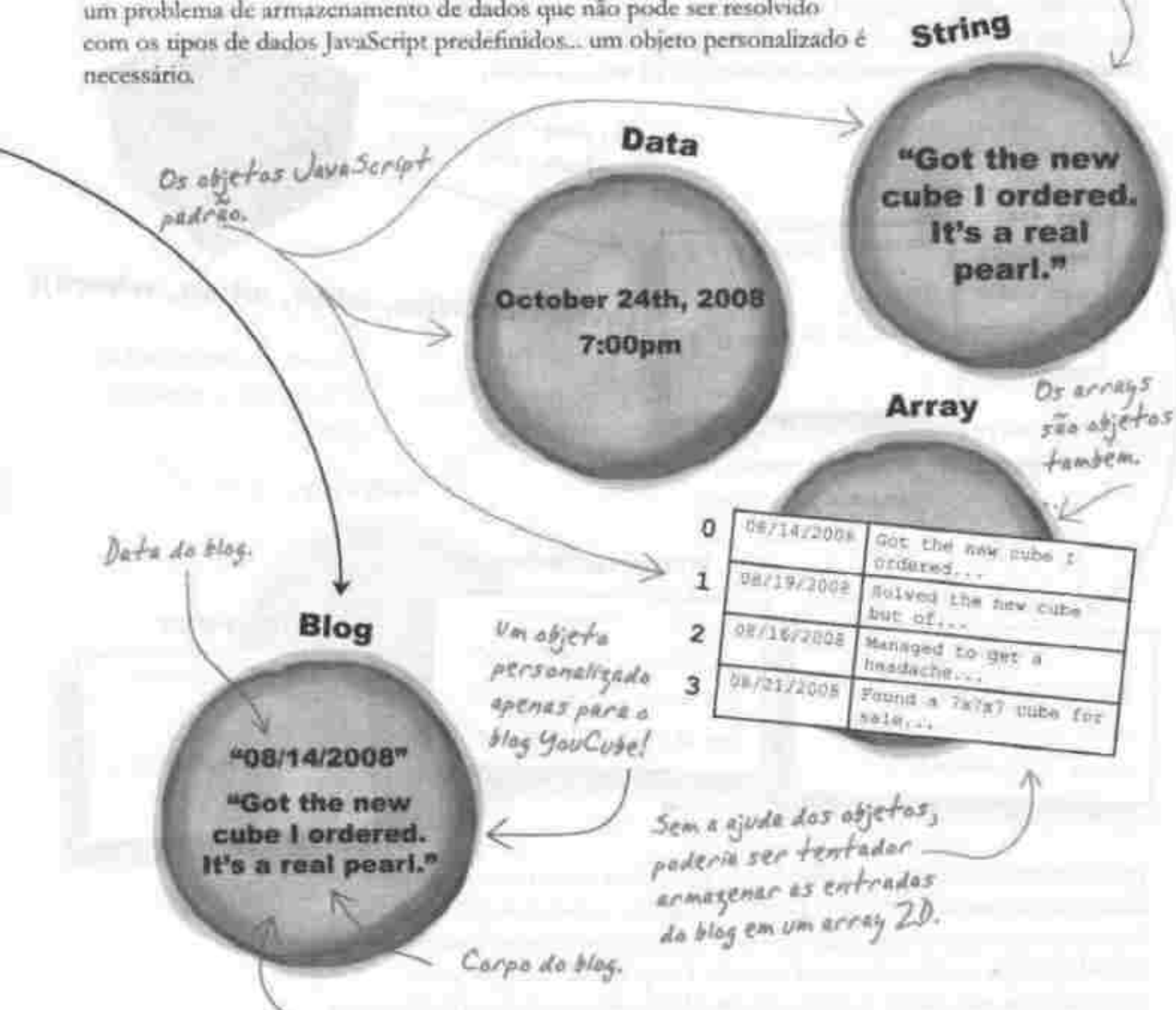
Um objeto personalizado permite que duas partes de dados do blog sejam combinadas em uma única entidade.

Data do blog + Corpo do blog = Objeto do blog

Os objetos personalizados estendem o JavaScript

A linguagem JavaScript inclui muitos objetos padrão úteis, vários dos quais exploraremos posteriormente neste capítulo. Por mais úteis que esses objetos possam ser, há vezes em que eles simplesmente não são suficientes. O blog YouCube é um bom exemplo desse limite, uma vez que envolve um problema de armazenamento de dados que não pode ser resolvido com os tipos de dados JavaScript predefinidos... um objeto personalizado é necessário.

É verdade, as strings são realmente objetos!



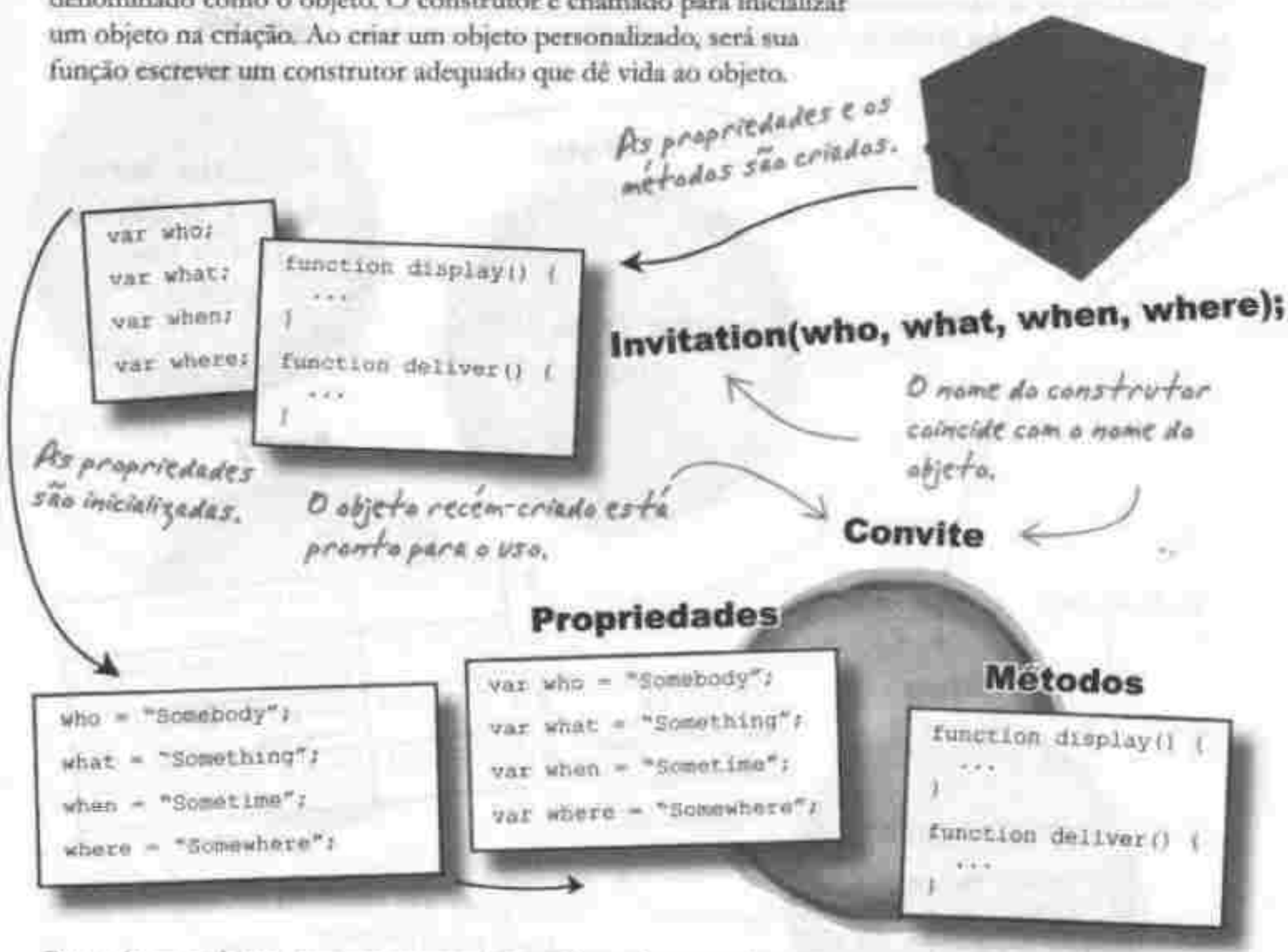
Os objetos personalizados permitem adicionar recursos ao JavaScript que servem às suas próprias necessidades específicas. No caso de Ruby, um objeto personalizado poderia modelar uma entrada do blog, usando as **propriedades** para representar uma data e o texto do corpo do blog. E mais, os **métodos** podem ser usados para adicionar um comportamento às entradas do blog, tornando mais simples criá-las e gerenciá-las.

Para dar vida a tal objeto personalizado, porém, primeiro temos que descobrir como os objetos personalizados são criados...

Construa seus próprios objetos personalizados

Como os objetos têm dados associados que devem ser inicializados quando um objeto é criado, um método especial chamado **construtor** é requerido para fazer com que o objeto comece a funcionar. Todo objeto personalizado requer seu próprio construtor, que é denominado como o objeto. O construtor é chamado para inicializar um objeto na criação. Ao criar um objeto personalizado, será sua função escrever um construtor adequado que dê vida ao objeto.

Um construtor é responsável por criar um objeto.



Para criar um objeto com um construtor, você usará o operador new, que inicia o processo de criação do objeto chamando o construtor do objeto. A parte do construtor para criar um objeto parece uma chamada para um método porque é realmente isso. Contudo, é importante usar sempre o operador new para iniciar a criação de um objeto, em oposição a simplesmente chamar o construtor de um objeto diretamente.

O operador new é usado para criar um novo objeto.

```
var invitation = new Invitation("Somebody", "Something", "Sometime", "Somewhere");
```

O novo objeto é armazenado em uma variável.

O construtor é chamado exatamente como um método.

As propriedades são definidas transmitindo argumentos para o construtor.

O que existe em um construtor?

Grande parte do serviço do construtor é estabelecer as propriedades de um objeto, junto com seus valores iniciais. Para criar uma propriedade em um construtor, você definirá a propriedade usando uma palavra-chave JavaScript chamada this. A palavra-chave this atribui a posse da propriedade ao objeto e também define seu valor inicial ao mesmo tempo. A palavra faz, literalmente, o que indica – você está criando uma propriedade que pertence a "este" (this) objeto, em oposição a ser apenas uma variável local dentro do construtor.

Os construtores têm sempre uma letra maiúscula inicial, pois são os nomes do objeto.

A palavra-chave this é o que distingue uma propriedade do objeto de uma variável normal.

```
function Invitation(who, what, when, where)
  this.who = who;
  this.what = what;
  this.when = when;
  this.where = where;
}
```

Os argumentos do construtor são atribuídos às novas propriedades.

O construtor é reunido, como qualquer outra função.

As propriedades do objeto são criadas e inicializadas em um construtor usando a notação do objeto (o operador ponto) e a palavra-chave `this`. Sem a palavra-chave `this`, o construtor não saberia que você está criando propriedades do objeto. O resultado desse construtor é a criação de quatro propriedades, que recebem os quatros valores transmitidos como argumentos para o construtor.

A palavra-chave this é a chave para criar as propriedades do objeto dentro de um construtor.

Aponte seu Lápis

Escreva um construtor para um objeto `Blog` que cria e inicializa as propriedades para a data e o texto do corpo de uma entrada do blog.

Aponte seu Lápis
Solução

Escreva um construtor para um objeto `Blog` que cria e inicializa as propriedades para a data e o texto do corpo de uma entrada do blog.

O construtor é nomeado como o objeto.

O texto do corpo e a data são transmitidos para o construtor como argumentos.

```
function Blog(body, date) {

  this.body = body;

  this.date = date;

}
```

A palavra-chave this refere-se às propriedades do objeto.

As propriedades são inicializadas usando os argumentos do construtor.

Como dar vida aos objetos do blog

O objeto `Blog`, certamente, está tomando forma, mas não foi realmente criado ainda. Por melhor que possa parecer na teoria, ele ainda é apenas uma hipótese a ser provada. Lembre-se de que o construtor estabelece a construção de um objeto, mas nada é criado fisicamente até que você use o operador `new`, que, então, constrói o objeto chamando o construtor. Portanto, continuemos e criemos um objeto `Blog` real.

Acompanhe os exemplos, disponíveis para download em www.altabooks.com.br.

Entrada do blog escrita à mão.

08/14/2008
Got the new cube I ordered. It's a real pearl.

```
var blogEntry = new Blog("Got the new cube I ordered...", "08/14/2008");
```

Objeto Blog do JavaScript.

Blog

"Got the new cube I ordered. It's a real pearl."

"08/14/2008"

```
function Blog(body, date) {
  ...
}
```

O construtor Blog( ) é chamado para criar o objeto.

O objeto é criado.

Não existem Perguntas Idiotas

P: Estou confuso com a criação do objeto. O operador `new` cria um objeto ou o construtor?

R: Ambos! O operador `new` é responsável por colocar a criação do objeto em movimento e grande parte de seu trabalho é assegurar que o construtor seja chamado. Simplesmente chamar um construtor como uma função sem usar o operador `new` não criaria um objeto e usar o operador `new` sem construtor não faria sentido.

P: Todo objeto personalizado requer um construtor?

R: Sim. O motivo é que o construtor é responsável por criar as propriedades do objeto, portanto, sem um construtor você não teria nenhuma propriedade. E sem nenhuma propriedade, não teria um objeto muito significativo.

Há uma exceção para esta regra sobre os construtores e ela se aplica ao criar um objeto puramente organizacional consistindo em uma coleção de métodos que não atuam nas propriedades do objeto. Neste caso, é tecnicamente possível fazer sem um construtor. Mas lembre-se de que tal objeto não é exatamente um exemplo brilhante de boas práticas de programação orientada a objetos porque é, na verdade, apenas uma coleção de funções afins. Ainda assim, o próprio JavaScript utiliza um objeto organizacional para agrupar as tarefas relacionadas à matemática, como você aprenderá posteriormente no capítulo.

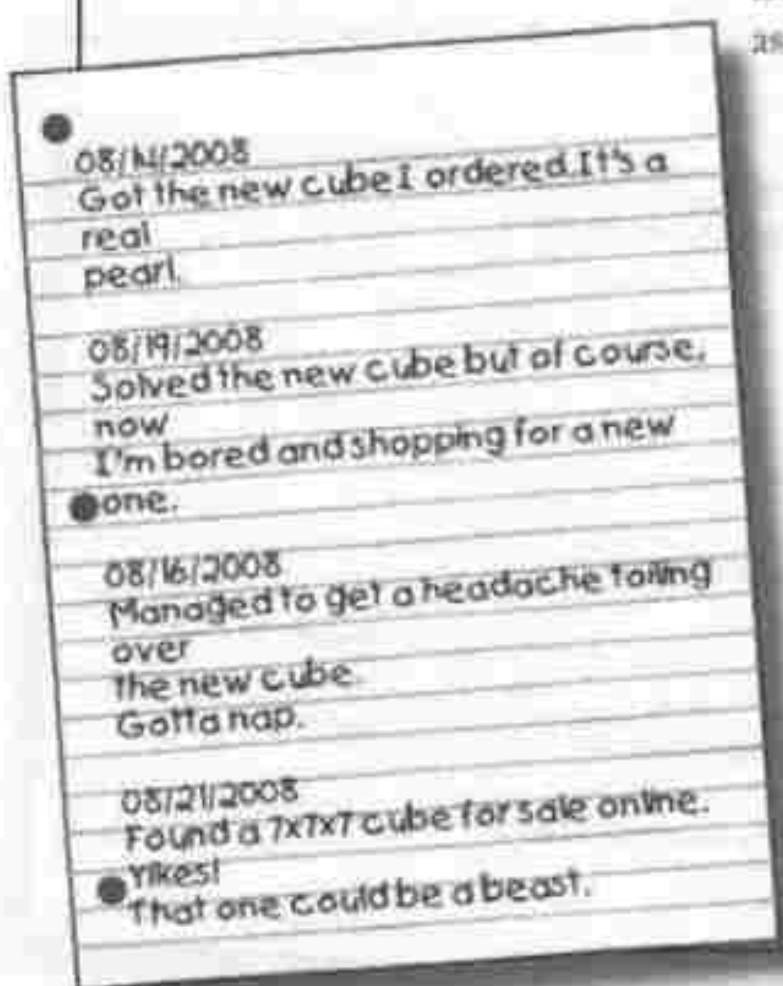
P: O que é exatamente `this`?

R: `this` é uma palavra-chave JavaScript usada para se referir a um objeto. Mais especificamente, `this` refere-se a um objeto de dentro desse **mesmo** objeto. Sim, isso parece muito estranho e um pouco esquizofrênico. Mas faz sentido, mais uma vez, envolver seu cérebro nisso. Para ver em termos reais, considere o que aconteceria se você perdesse seu relógio e alguém o encontrasse em um lugar cheio de pessoas. Quando essa pessoa levantasse o relógio, provavelmente você gritaria: "É meu relógio!". Você usou a palavra "meu" para se referir a si próprio. O mais importante, a palavra "meu" é usada para esclarecer que você é o dono do relógio. `this` funciona exatamente do mesmo modo – implica na **propriedade** do objeto. Portanto, `this.date` significa que a propriedade `date` pertence ao objeto no qual o código aparece.

Aponte seu Lápis

Crie um array de objetos `Blog` em uma variável denominada `blog` que é inicializada nas entradas do blog no blog YouCube. Sinta-se à vontade para escrever apenas as primeiras palavras de texto do corpo em cada entrada.

```
var blog =

[.......................................................

.......................................................

.......................................................

.......................................................];
```



Aponte seu Lápis
Solução

Crie um array de objetos `Blog` em uma variável denominada `blog` que é inicializada nas entradas do blog no blog YouCube. Sinta-se à vontade para escrever apenas as primeiras palavras de texto do corpo em cada entrada.

08/14/2008
Got the new cube I ordered. It's a real pearl.

08/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.

08/16/2008
Managed to get a headache toiling over the new cube.
Gotta nap.

08/21/2008
Found a 7x7x7 cube for sale online. Yikes!
That one could be a beast.

```
var blog =

  [ new Blog("Got the new cube I ordered...","08/14/2008"),

  new Blog("Solved the new cube but of course...","08/19/2008"),

  new Blog("Managed to get a headache toiling...","08/16/2008"),

  new Blog("Found a 7x7x7 cube for sale...", "08/21/2008")   ];
```

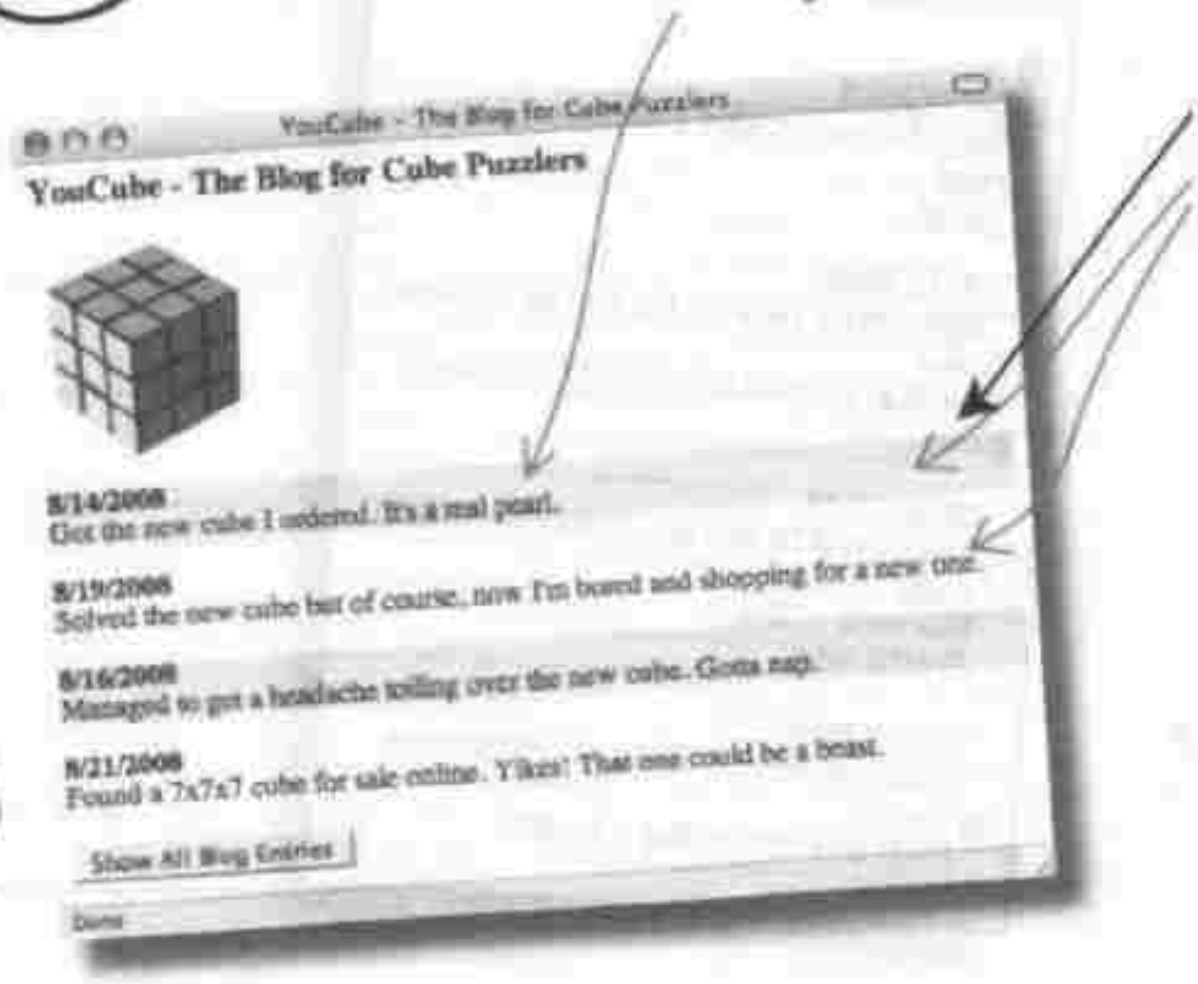
Cada entrada do blog é criada como um objeto Blog com seu próprio texto do corpo e data.

YouCube 1.0

Combinar o array de objetos `Blog` com algum código JavaScript para exibir os dados do blog produzirá uma versão inicial do YouCube. Ruby sabe que seu trabalho não está terminado, mas o blog está começando a funcionar e ela está contente com os primeiros resultados.

Gosto do modo como o objeto Blog combina a data e o texto do corpo do blog no YouCube.

Os dados armazenados em cada objeto Blog são bem exibidos na página YouCube.



Vejamos o código requerido para dar vida aos objetos `Blog` e tornar o YouCube 1.0 uma realidade...

Um foco no YouCube

```
<html>
  <head>
    <title>YouCube - The Blog for Cube Puzzlers</title>

    <script type="text/javascript">
      // O construtor do objeto Blog
      function Blog(body, date) {
        // Atribua as propriedades
        this.body = body;
        this.date = date;
      }

      // O array global das entradas do blog
      var blog = [ new Blog("Got the new cube I ordered...", "08/14/2008"),
                   new Blog("Solved the new cube but of course...", "08/19/2008"),
                   new Blog("Managed to get a headache toiling...", "08/16/2008"),
                   new Blog("Found a 7x7x7 cube for sale online...", "08/21/2008") ];

      // Exiba a lista de entradas do blog
      function showBlog(numEntries) {
        // Ajuste o número de entradas para mostrar o blog completo, se necessário
        if (!numEntries)
          numEntries = blog.length;

        // Exiba as entradas do blog
        var i = 0, blogText = "";
        while (i < blog.length && i < numEntries) {
          // Use um segundo plano cinza para uma entrada sim outra não do blog
          if (i % 2 == 0)
            blogText += "<p style='background-color:#EEEEEE'>";
          else
            blogText += "<p>";

          // Gere o código HTML formatado do blog
          blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>";

          i++;
        }

        // Defina o código HTML do blog na página
        document.getElementById("blog").innerHTML = blogText;
      }
    </script>
  </head>

  <body onload="showBlog(5);">
    <h3>YouCube - The Blog for Cube Puzzlers</h3>

    
    
```

Não existem Perguntas Idiotas

P: Por que o botão Show All Blog Entries é necessário em seu YouCube?

R: No estado atual do blog, o botão não é necessário, uma vez que há apenas quatro entradas do blog no total. Mas quando o blog crescer, será cada vez mais importante limitar o número de entradas mostradas inicialmente na página YouCube principal para evitar um excesso de visitantes. Portanto, o código do blog tem como padrão mostrar apenas as cinco primeiras entradas. O botão Show All Blog Entries (Exibir Todas as Entradas do Blog) anula esse padrão exibindo todas as entradas do blog.

P: Por que `innerHTML` é usada para mostrar as entradas do blog, ao invés dos métodos DOM?

R: Embora os métodos DOM sejam certamente preferidos em termos de compatibilidade com os padrões Web, eles são bem difíceis ao gerar dinamicamente o código HTML altamente formatado. A razão é que cada tag contêiner, como `<p>` e `<strong>`, tem que ser criada como um pai para os nós-filhos de seu conteúdo. `innerHTML` é uma grande conveniência neste caso e simplifica muito o código YouCube.

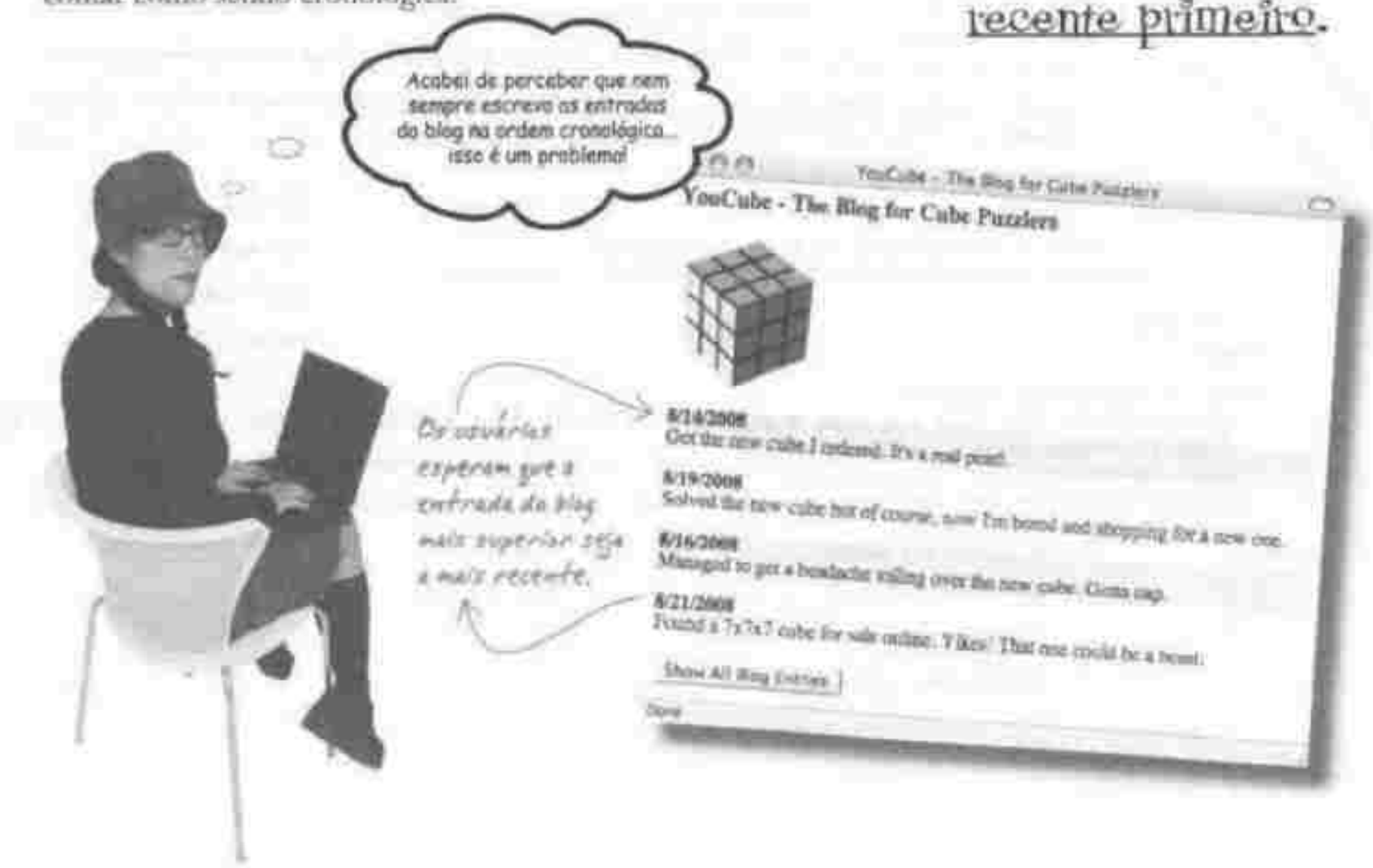
P: Por que o objeto `Blog` não tem nenhum método?

R: Ambição, isso é bom! A verdade é que há muitos outros aspectos do YouCube para trabalhar antes dos métodos `Blog` tornarem-se uma prioridade de fato. Mas não se preocupe, os métodos são definitivamente uma parte do plano de longo alcance para o YouCube. Os métodos são uma parte importante de qualquer objeto bem construído e o objeto `Blog` não é diferente.

Um blog desordenado

O YouCube 1.0 parece bom, mas tem falhas. Ruby notou que as entradas do blog estão na ordem errada – na verdade, elas devem aparecer com o envio mais recente primeiro. No momento, são exibidas na ordem em que são armazenadas, que não podemos contar como sendo cronológica.

A ordem das entradas do blog deve ser a mais recente primeiro.

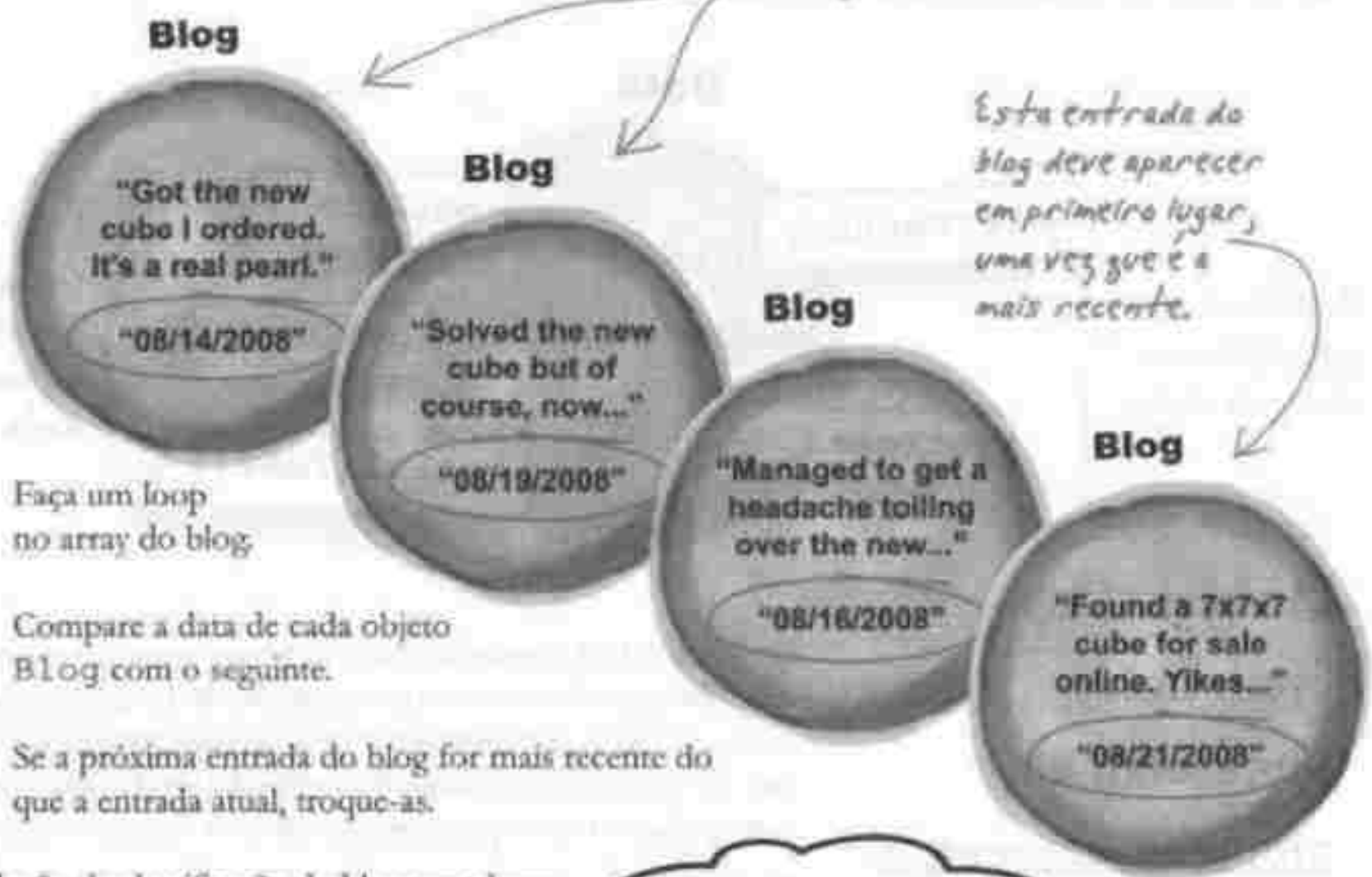


A necessidade da classificação

A solução de Ruby para o problema da ordem do blog é classificar o array do blog pela data. Como o JavaScript suporta o loop e as comparações, deve ser possível fazer um loop nas entradas do blog, comparar as datas entre si e classificá-las na ordem cronológica inversa (os envios mais recentes primeiro).

Troque estas duas entradas do blog, uma vez que a segunda é a mais recente.

Esta entrada do blog deve aparecer em primeiro lugar, uma vez que é a mais recente.



1. Faça um loop no array do blog.
2. Compare a data de cada objeto `Blog` com o seguinte.
3. Se a próxima entrada do blog for mais recente do que a entrada atual, troque-as.

Esta solução de classificação do blog tem algum mérito e parece que pode funcionar, supondo que podemos planejar os detalhes de comparar as datas do blog.

Espere um minuto! Se as datas forem armazenadas como strings, como se pode compará-las para saber qual é a mais recente?

Uma data armazenada em uma string não é realmente uma data.

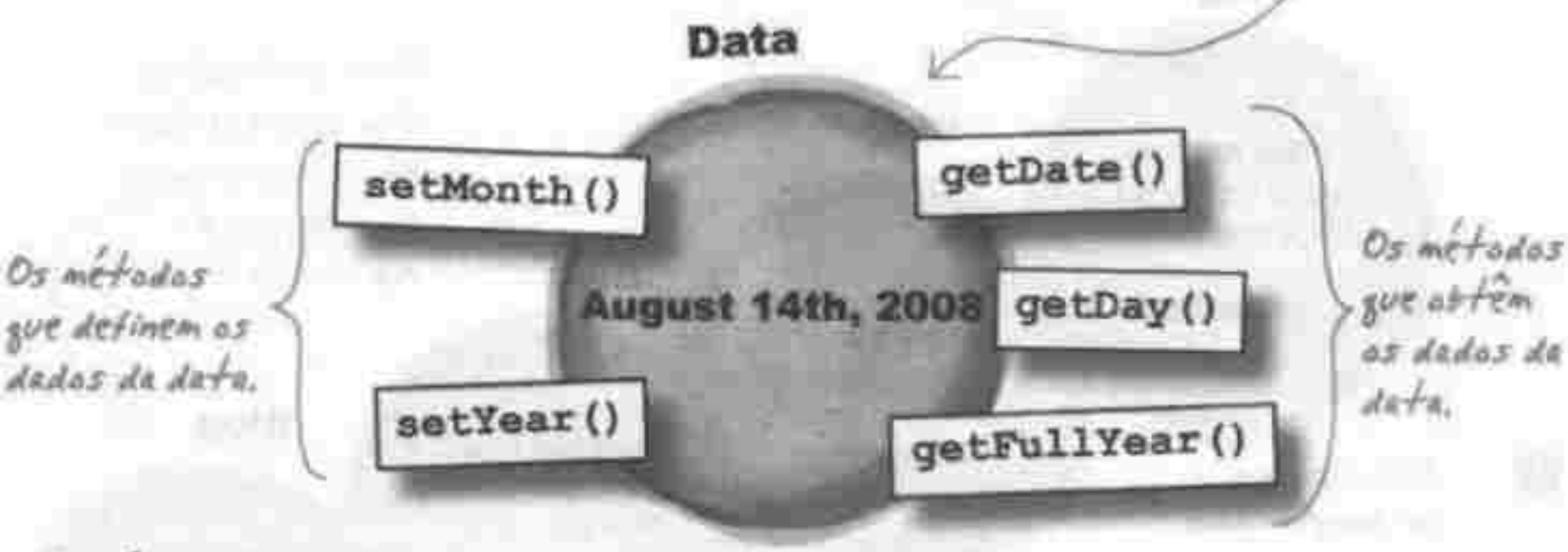
A estratégia de classificação do blog de Ruby teve dificuldades inesperadas devido ao fato de que uma data armazenada como uma string não tem nenhum conceito de tempo. Em outras palavras, não há nenhuma maneira de comparar as strings `"08/14/2008"` com `"08/19/2008"` para saber qual é a mais recente porque são apenas strings. Embora seja possível comparar as strings, tais comparações não compreendem o formato específico de uma data e, portanto, não são capazes de comparar os componentes do mês, dia e ano de uma data ao executar a comparação.

Portanto, antes de podermos pensar seriamente sobre como classificar as entradas do blog pela data, primeiro, precisamos repensar no modo como as datas são armazenadas no blog.

Um objeto JavaScript para a data

O que Ruby precisa é da capacidade de armazenar uma data de modo que possa ser comparada com outras datas. Em outras palavras, a data precisa **compreender** que é uma data e **comportar-se** de acordo. Espere um minuto, parece muito com um objeto! E como resultado, o JavaScript oferece um objeto Date predefinido que poderia ser muito bem ser o que o Ruby precisa.

A data da primeira entrada do blog.



Os métodos que definem os dados da data.

Os métodos que obtêm os dados da data.

O objeto Date predefinido representa um momento no tempo.

O objeto `Date` representa um momento específico no tempo, em milissegundos, e é uma parte padrão do JavaScript. Embora o objeto `Date` certamente use propriedades internamente, elas são invisíveis para você, o usuário do objeto. Você trabalha com o objeto `Date` puramente através de seus métodos.

Parecido com o objeto `Blog`, você cria um objeto `Date` usando o operador `new`. Eis um exemplo de como criar um objeto `Date` que representa a data e hora atuais:

Armazene o objeto Date recém-criado em uma variável.

```
var now = new Date();
```

Crie um objeto Date usando o operador new.

Este novo objeto Date representa a data/hora atuais.

Dentro do objeto Date, a hora é expressa em milissegundos.

Este objeto `Date` é criado e inicializado com a data e a hora atuais. Note que a sintaxe para criar um objeto `Date` é muito parecida com chamar uma função ou método e é por isso que você realmente está chamando o construtor do objeto `Date`. Você pode transmitir ao construtor `Date()` um argumento de string para especificar uma data diferente da atual. Por exemplo, este objeto `Date` representa a data da primeira entrada do blog YouCube:

A data é transmitida ao construtor como uma string de texto.

```
var blogDate = new Date("08/14/2008");
```

Como calcular a hora

Um dos recursos mais importantes dos objetos é como eles sabem, inerentemente, manipular a si mesmos. Por exemplo, considere o quanto capcioso seria calcular o número de dias entre duas datas por si mesmo. Você teria que converter de algum modo a data em um número de dias a partir de alguma referência conhecida, assegurando que iria decompor os anos bissextos. Ou poderia simplesmente deixar o objeto Data fazer o trabalho para você... verifique esta função que faz o levantamento pesado com alguns objetos `Date`:

A função aceita dois objetos Date como argumentos.

Converta de milissegundos em segundos, em minutos, em horas, em dias. Caramba!

```
function getDaysBetween(date1, date2) {
  var daysBetween = (date2 - date1) / (1000 * 60 * 60 * 24);
  return Math.round(daysBetween);
}
```

Simples, mas poderoso, este é o código que faz todo o trabalho!

Arredonde o resultado e retorne-o... round( ) é um método do objeto Math, com o qual lidaremos em um capítulo posterior.

Esta função mostra o poder do objeto `Date` em uma simples parte do código – uma subtração. Toda a complexidade associada ao cálculo da diferença entre as duas datas está convenientemente colocada dentro do objeto `Date`. Nossa única preocupação é o resultado da subtração, que é o número de milissegundos entre as duas datas. Converta os milissegundos em dias, arredonde o resultado e teremos uma pequena função útil que pode ser reutilizada sempre que precisarmos saber a diferença entre as duas datas.

```
getDaysBetween(data1, data2);
```

Exercício

Crie dois objetos `Date` para as duas primeiras entradas do blog YouCube. Então, chame a função `getDaysBetween( )`, transmitindo os dois objetos Date e exibindo o resultado em uma caixa de alerta.

Solução do Exercício

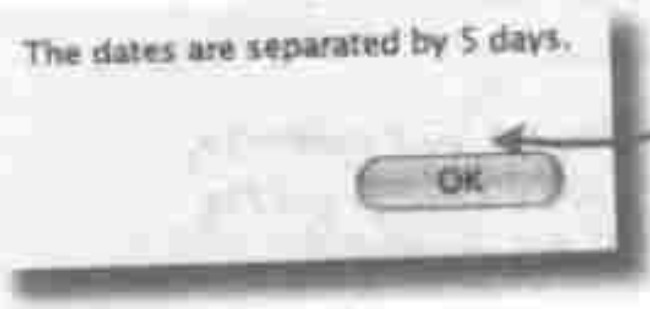
Crie dois objetos `Date` para as duas primeiras entradas do blog YouCube. Então, chame a função `getDaysBetween( )`, transmitindo os dois objetos `Date` e exibindo o resultado em uma caixa de alerta.

Crie objetos Date para as duas datas de entrada do blog.

```
var blogDate1 = new Date("08/14/2008");

var blogDate2 = new Date("08/19/2008");

alert("The dates are separated by " + getDaysBetween(blogDate1, blogDate2) + " days.");
```

A função retorna a diferença.

Transmita os dois objetos Date como argumentos para a função.



Como repensar as datas do blog

Embora seja ótimo que o JavaScript ofereça um objeto `Date` que torna possível manipular as datas de modo inteligente, o objeto `Blog` do YouCube atualmente ainda armazena as datas como strings, não como objetos `Date`. Para aproveitar os recursos disponibilizados pelo objeto `Date`, precisamos mudar o blog para que suas datas sejam objetos `Date`.

A pergunta é: a propriedade `date` do objeto `Blog` pode armazenar um objeto `Date` ao invés de uma string?

Um objeto em um objeto

O objeto `Blog` é um bom exemplo de como os objetos devem **conter**, com freqüência, outros objetos. As duas propriedades do objeto `Blog` já são, na verdade, objetos em si – ambas as propriedades são objetos `String`. Os objetos `String` realmente não se parecem objetos porque são criados como **literais do objeto** simplesmente colocando aspas em uma string de texto. Os objetos `Date` não são tão flexíveis e devem ser criados usando o operador `new`.

A literal de string cria automaticamente um objeto String.

Para criar uma propriedade date do blog como um objeto `Date`, temos que usar o operador `new` para criar uma nova `Date` ao criar o objeto `Blog`. Se isso parecer aterrorizante, talvez algum código diminua o medo.

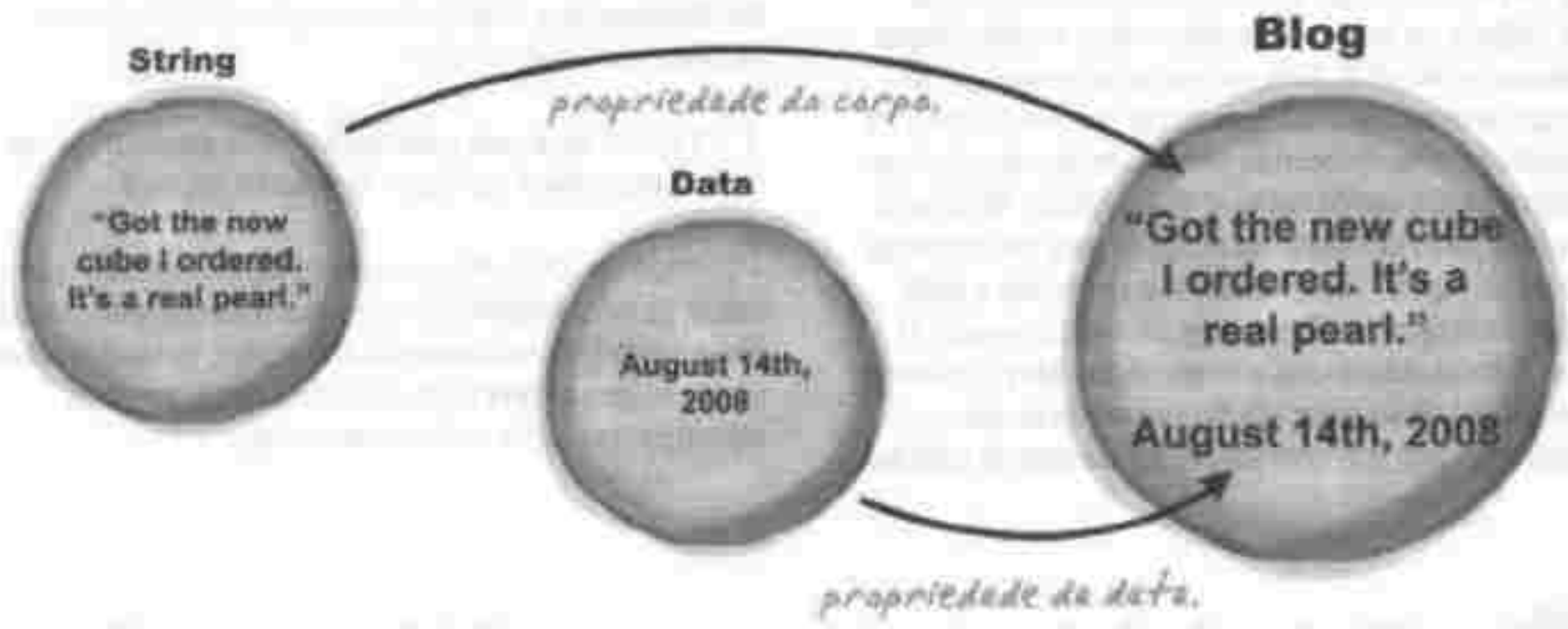
O objeto Blog é criado usando o operador new.

```
var blogEntry = new Blog("Nothing going on but the weather.",
    new Date("10/31/2008"));
```

O objeto Date é criado e transmitido para o construtor Blog(), também usando o operador new.

O código mostra como uma entrada do blog YouCube é, agora, criada como um objeto que contém dois outros objetos (um objeto `String` e um objeto `Date`). Naturalmente, ainda precisamos construir um array de objetos `Blog` para representar com sucesso todas as entradas do blog YouCube.

O operador new cria objetos com a ajuda de construtores.



Aponte seu Lápis

Reescreva o código para criar um array de objetos `Blog` do YouCube onde cada data é agora um objeto `Date`. Sinta-se à vontade para encurtar o texto do corpo.

Aponte seu Lápis
Solução

Reescreva o código para criar um array de objetos `Blog` do YouCube onde cada data é agora um objeto `Date`. Sinta-se à vontade para encurtar o texto do corpo.

Cada entrada do blog é ainda criada como um objeto Blog.

```
var blog = [ new Blog("Got the new cube I ordered...", new Date("08/14/2008")),
  new Blog("Solved the new cube but of course...", new Date("08/19/2008")),
  new Blog("Managed to get a headache toiling...", new Date("08/16/2008")),
  new Blog("Found a 7x7x7 cube for sale...", new Date("08/21/2008")) ];
```

As literais de string funcionam bem para o texto do corpo de cada entrada do blog.

A data para cada objeto Blog é criada como um objeto Date.

Não existem Perguntas Idiotas

P: Por que a data em um objeto `Date` é armazenada em milissegundos?

R: A princípio, saiba que o objeto `Date` representa um instante no tempo. Se pudesse clicar o botão Pause (Pausa) no universo, teria um momento congelado no tempo. Mas não teria nenhum modo de dizer às pessoas quando o momento ocorreu sem algum tipo de referência. Portanto, você decide sobre 1º de janeiro de 1970 como sendo o ponto de referência arbitrário para seu momento no tempo. Agora, você precisa de uma medida a partir deste deslocamento. Talvez seja 38 anos, 8 meses, 14 dias, 3 horas, 29 minutos e 11 segundos. Mas é um modo complicado de controlar um deslocamento do tempo. É muito mais fácil ficar com uma única unidade de medida, uma que seja capaz de representar as menores frações de tempo. Que tal milissegundo? Portanto, ao invés de todas aquelas unidades diferentes de tempo, agora terá 1.218.702.551.000 milissegundos. Sim, é o grupo inteiro de milissegundos, mas os números grandes não são um problema para o JavaScript.

P: Tenho que me preocupar em converter os milissegundos ao usar o objeto `Date`?

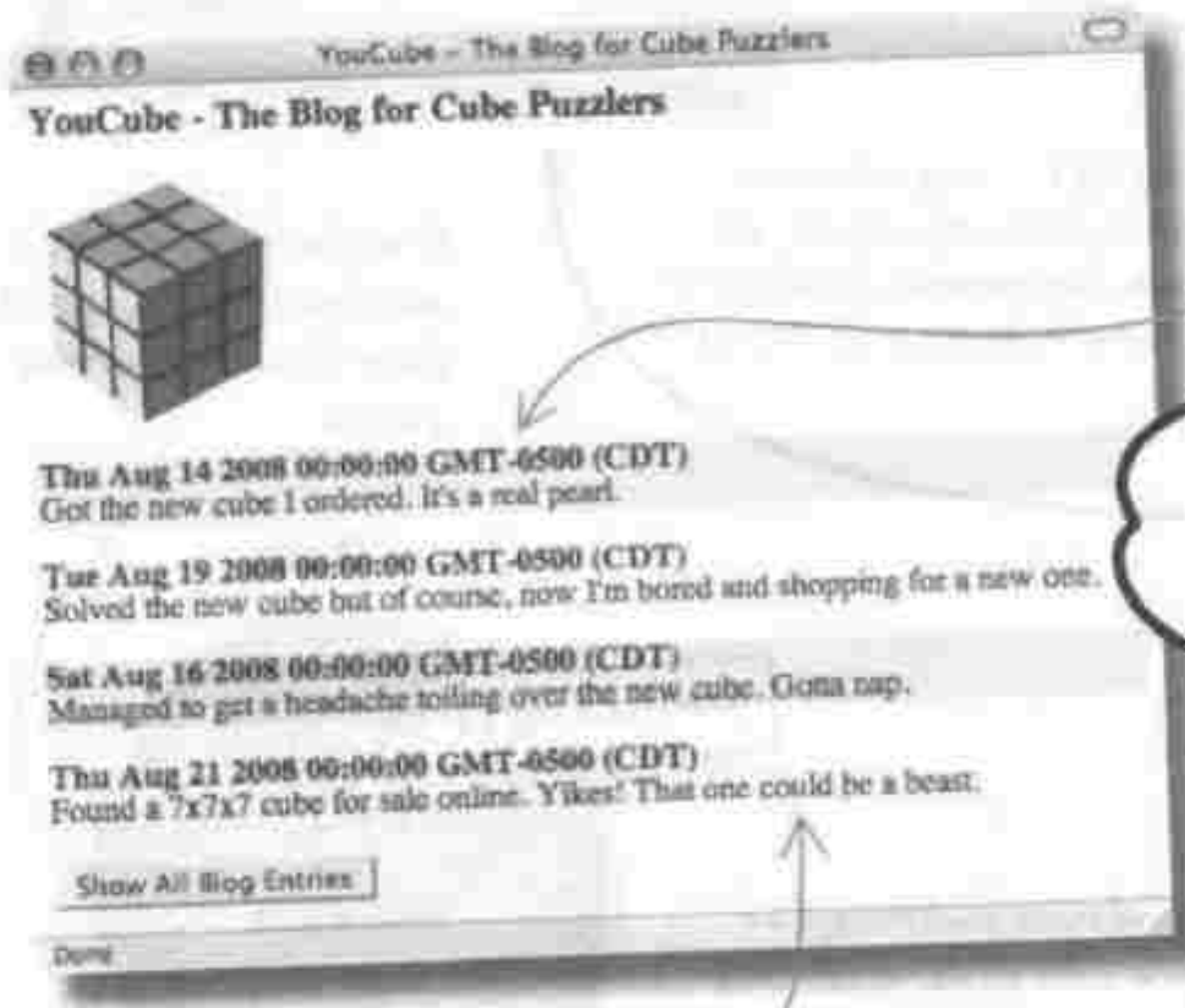
R: Depende. O objeto `Date` inclui vários métodos para extrair as partes significativas de uma data que evitam lidar diretamente com os milissegundos. Porém, se você precisar lidar com uma **diferença** entre duas datas, então os milissegundos quase certamente servirão.

Pontos de Bala

- Os objetos `Date` JavaScript padrões representam um **instante no tempo**, expressado em milissegundos.
- O objeto `Date` inclui vários métodos para acessar os diferentes fragmentos e partes de uma data e uma hora.
- O objeto `Date` é inteligente o bastante para saber como manipular as datas matematicamente, assim como comparar as datas entre si.
- Como a maioria dos objetos diferentes de `String`, você cria um objeto `Date` usando o operador `new`.

As datas não são úteis... para os humanos

Com a propriedade da data do objeto Blog convertida em um objeto Date, Ruby está pronta para voltar sua atenção para classificar as entradas do blog pela data. Bem, quase pronta. Parece que ela introduziu um novo problema, no sentido de que as datas das entradas do blog, agora, são extremamente obscuras. Ruby espera que os usuários realmente não se importem com o fuso horário de cada envio e ele apenas confundirá a experiência YouCube. Claramente, a introdução dos objetos `Date` no YouCube precisa ser examinada mais atentamente!



As datas do blog estão bem bagunçadas... as informações sobrecarregam-se!

A mudança do objeto Date faz sentido na hora, mas agora, as datas do blog parecem horríveis. Nem consigo lembrar como escrever o código para formatar as datas.

Não só as datas parecem ruins, como também as entradas do blog ainda aparecem na ordem errada... ugh!

Ruby está um pouco confusa sobre as datas obscuras do YouCube porque ela não se lembra como escrever nenhum código para exibi-las. Tudo que ela fez foi converter as strings da data nos objetos `Date`. As forças malignas do JavaScript estão conspirando para tornar suas datas feias?

Como converter os objetos em texto

Felizmente, não há nenhuma força maligna para responsabilizar pelas datas YouCube feias. Na verdade, são as forças muito naturais dos objetos JavaScript que são responsáveis pela formatação da data – as datas formatam a si mesmas! Funciona assim: todo objeto JavaScript tem um método chamado `toString()` que tenta fornecer uma representação de texto do objeto. A data obscura é a saída do método `toString()` **padrão** do objeto `Date`.

```
var blogDate = new Date("08/14/2008");
alert(blogDate.toString());
```

A parte esperta do método `toString()` é que ele entra em cena automaticamente quando um objeto é usado em um contexto onde uma string é esperada. Por exemplo, o código de alerta da data do blog poderia ser reescrito assim, e resultar na mesma saída exata:

```
alert(blogDate);
```

O método toString() mostra como um objeto Date também controla a hora.

A função alert() espera uma string, portanto o método toString() é chamado internamente para fornecer uma representação de string do objeto Date.

O método toString() é bem útil ao fornecer uma representação de string de uma data, mas também aparece em outros objetos também.

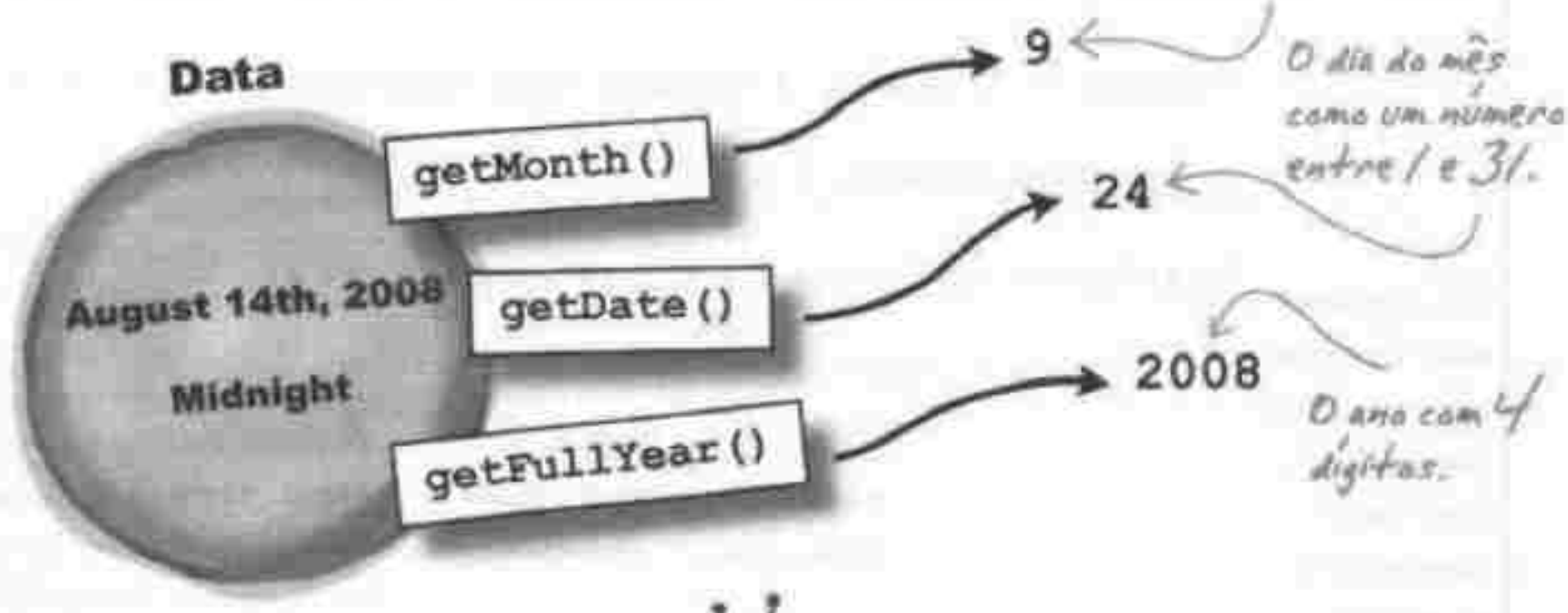
O método toString() fornece uma representação de string de um objeto.

Como a função `alert()` espera uma string, o objeto `Date` é esperto o bastante para saber que deve fornecer uma representa de string de si mesmo. Portanto, ele chama o método `toString()` para lidar com a tarefa.

Este negócio `toString()` não seria um problema, exceto pelo fato de que o YouCube realmente precisa que as datas sejam exibidas em um formato fácil de ler como, por exemplo, *MM/DD/AAAA*. No final, não parece que o YouCube será capaz de aproveitar a representação de string padrão do objeto `Date` disponibilizada por seu método `toString()`.

Como acessar as peças e as partes de uma data

Ruby precisa de um modo de personalizar o formato de uma data. A chave para personalizar a formatação de um objeto `Date` é acessar as partes individuais da data como, por exemplo, o mês, dia e ano. Então, poderemos remontar uma data em qualquer formato desejado. Felizmente, o objeto `Date` fornece métodos para acessar essas partes de informação.



O objeto `Date`, de fato, suporta muito mais métodos diferentes destes três, fornecendo todos os tipos de maneiras diferentes de acessar a data e a hora de um objeto `Date`. Porém, esses três métodos são tudo o que precisamos para colocar as datas do blog em forma.

Preste atenção!

Preste muita atenção nos valores retornados pelos métodos Date.

O método getMonth( ) retorna um mês como um número entre 0 (janeiro) e 11 (dezembro), ao passo que getDate( ) retorna o dia do mês na faixa de 1 a 31.

Aponte seu Lápis

Corrija o problema obscuro da data do blog YouCube reescrevendo o código que formata uma entrada do blog e armazena-a na variável `blogText`. Certifique-se de que a data do blog esteja formatada como *MM/DD/AAAA*. Eis a versão original do código:

```
blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>";
```

Aponte seu Lápis

Corrija o problema obscuro da data do blog YouCube reescrevendo o código que formata uma entrada do blog e armazena-a na variável `blogText`. Certifique-se de que a data do blog esteja formatada como *MM/DD/AAAA*. Eis a versão original do código:

```
blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>";

blogText += "<strong>" + blog[i].date.getMonth( ) + 1 + "/" +
blog[i].date.getDate( ) + "/" +
blog[i].date.getFullYear( ) + "</strong><br />" +
blog[i].body + "</p>";
```

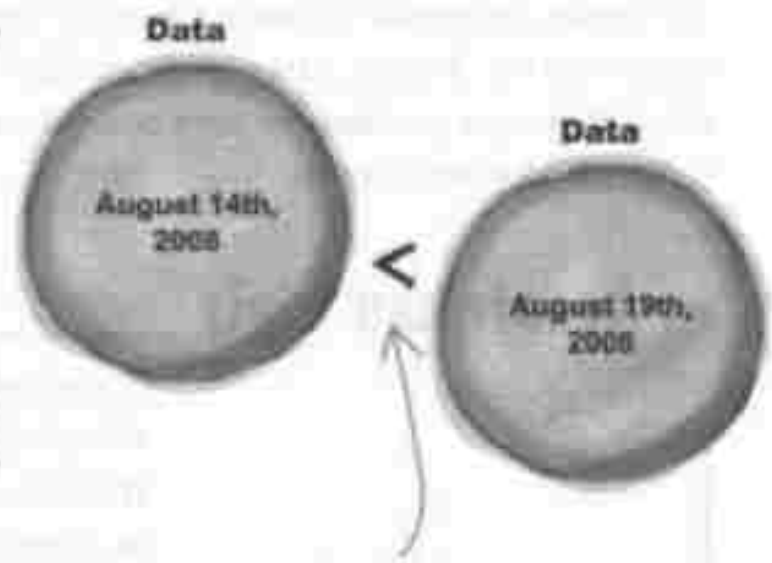
Conseguimos mais controle não contando com o objeto Date para formatar a si mesma.

Cada parte da data é extraída do objeto Date chamando os métodos.

A data do blog exibida, agora, está construída de modo personalizado no formato MM/DD/AAAA.

As datas facilitam a classificação

Agora que as datas do blog foram convertidas com sucesso em objetos `Date`, que são muito mais adequados para classificar do que as strings, é hora de rever a ordem do blog. O problema é que as entradas do blog são exibidas atualmente na mesma ordem em que são armazenadas no array `blog`, que não é necessariamente cronológica. A maioria dos blogs é exibida na ordem cronológica inversa, onde os envios mais recentes aparecem primeiro na lista de entradas do blog. Sabendo isso, é possível rever a estratégia de classificação do blog original:



Agora, temos os objetos Date que podem ser comparados entre si.

1. Faça um loop no array do blog.
2. Compare o objeto `Date` em cada objeto `Blog` com o seguinte.
3. Se a data da próxima entrada do blog for mais recente que a entrada atual, troque as entradas.

Embora a parte de comparação das datas desta estratégia certamente pareça muito menos assustadora com a ajuda do objeto `Date`, o resto do plano ainda envolve uma boa quantidade de codificação personalizada. Classificar uma seqüência de datas certamente parece um problema de programação comum que foi resolvido muitas vezes antes. Você odeia fazer sempre a mesma coisa...

Não seria um sonho se o JavaScript tivesse algum tipo de recurso de classificação predefinido que acabasse com o trabalho pesado de classificação de uma seqüência dos dados?

Arrays como objetos

Pode ser que um array seja capaz de classificar a si mesmo? Se uma data souber como se transformar em uma string, não será tão difícil pensar que um array poderia ser capaz de se classificar. Para que isso seja possível, porém, um array teria que ser um objeto para que a classificação pudesse ocorrer em um método. E, na verdade, é. Lembra deste código do script Mandango?

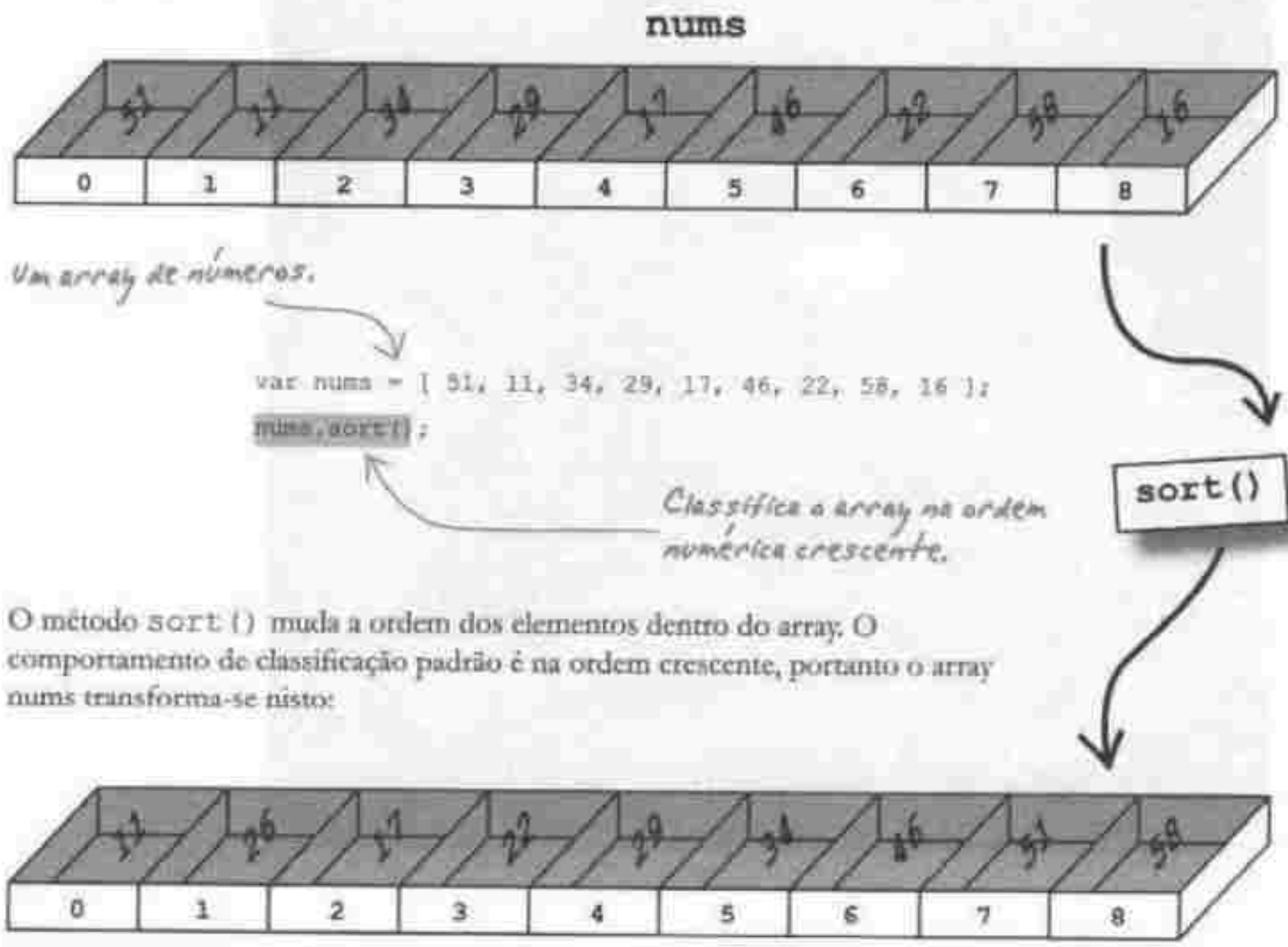
Um array é realmente apenas um objeto.

```
for (var i = 0; i < seats.length; i++) {
  ...
}
```

A variável seats é um array.

length é uma propriedade do objeto do array que informa como muitos elementos estão no array.

Portanto, o segredo está revelado, os arrays são objetos, mas isso significa que eles podem classificar a si mesmos? Não só os arrays têm propriedades, como `length`, como também têm métodos que atuam nos dados do array, dando-lhes vida. E sim, há um método denominado `sort()` que classifica os dados em um array. Vejamos como funciona:



Um array de números.

```
var nums = [ 51, 11, 34, 29, 17, 46, 22, 58, 16 ];
nums.sort();
```

Classifica o array na ordem numérica crescente.

O método `sort()` muda a ordem dos elementos dentro do array. O comportamento de classificação padrão é na ordem crescente, portanto o array nums transforma-se nisto:

Classificação personalizada de um array

O comportamento padrão do método `sort()` do objeto `Array` geralmente não é suficiente. A novidade é que o comportamento de classificação é determinado por uma **função de comparação** que `sort()` chama para lidar com a comparação de cada item classificado em um array. Você pode ajustar a ordem de classificação fornecendo sua própria versão dessa função de comparação. Eis um exemplo de como é, geralmente, a função:

```
function compare(x, y) {
  return x - y;
}
```

Os dois argumentos são dois itens do array que estão sendo comparados para a classificação.

O valor de retorno determina se x e y ficam onde estão no array ou se y é classificado na frente de x.

O valor de retorno da função `compare()` é um número que determina a ordem de classificação resultante de x em comparação a y.

compare(x, y)

- **< 0** — Classifica x na frente de y.
- **0** — Não classifica – deixa x e y como estão.
- **> 0** — Classifica y na frente de x.

Sua função `compare()` personalizada é introduzida na equação de classificação do array quando você chama o método `sort()` – simplesmente transmita uma referência para a função `compare()` para o método.

```
nums.sort(compare);
```

A classificação do array, agora, é controlada pela função compare() personalizada.

Aponte seu Lápis

Escreva o código para uma função de comparação personalizada denominada `compare()` que classifica as entradas do array do blog YouCube na ordem cronológica inversa (a mais recente primeiro). Sugestão: Os objetos `Blog` podem ser subtraídos uns dos outros simplesmente usando um sinal de menos.

Aponte seu Lápis
Solução

Escreva o código para uma função de comparação personalizada denominada `compare( )` que classifica as entradas do array do blog YouCube na ordem cronológica inversa (a mais recente primeiro). Sugestão: Os objetos `Blog` podem ser subtraídos uns dos outros simplesmente usando um sinal de menos.

Estes dois argumentos são objetos Blog, uma vez que o array contém objetos Blog.

```
function compare(blog1, blog2) {
    return blog2.date - blog1.date;
}
```

Subtrair a primeira data da segunda resulta em uma classificação cronológica invertida.

Estamos subtraindo as duas datas como números (milissegundos).

A classificação ficou simples com as literais da função

Quando você pensa sobre o papel da função de comparação de classificação do array, ela é realmente apenas usada pelo método `sort()` e nada mais. Como nunca é chamada por nenhum código de script YouCube, de fato não há nenhum motivo para ser uma função nomeada.

Ainda se lembra das literais da função do termostato no Capítulo 6? A função `compare( )` é uma excelente candidata para uma literal da função por causa do modo como é usada. Na verdade, a classificação do blog YouCube pode ser simplificada convertendo a função `compare( )` em uma literal da função que é transmitida diretamente para o método `sort( )`.

```
blog.sort(function(blog1, blog2) {
    return blog2.date - blog1.date;
});
```

A literal da função é transmitida diretamente para o método sort() do array.

Como devota dos quebra-cabeças, Ruby é eficiente. E, neste caso, isso se equipara a eliminar uma função nomeada desnecessária que é, realmente, apenas uma assistente do método `sort( )`. Ruby é tão inclinada à eficiência que ela não vê por que a função de comparação precisa ocupar três linhas de código. Embora a organização do código JavaScript não faça o código ser executado de modo diferente, neste caso a literal da função é simples o bastante para fazer sentido diminuí-la para uma única linha de código.

A literal da função é reduzida a uma única linha de código.

```
blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });
```

Não existem Perguntas Idiotas

P: Todo objeto tem um método `toString( )`?

R: Sim. Mesmo que você crie um objeto personalizado e não lhe forneça um método `toString( )`, o JavaScript informará, pelo menos, que é um objeto, caso você utilize-o em um contexto onde uma string é esperada. Certamente, a string não será muito significativa, mas caberá a você fornecer um método `toString( )` para os objetos personalizados se quiser que ele transmita um significado sobre o objeto.

P: Como a comparação da classificação funciona entre os objetos `Date`?

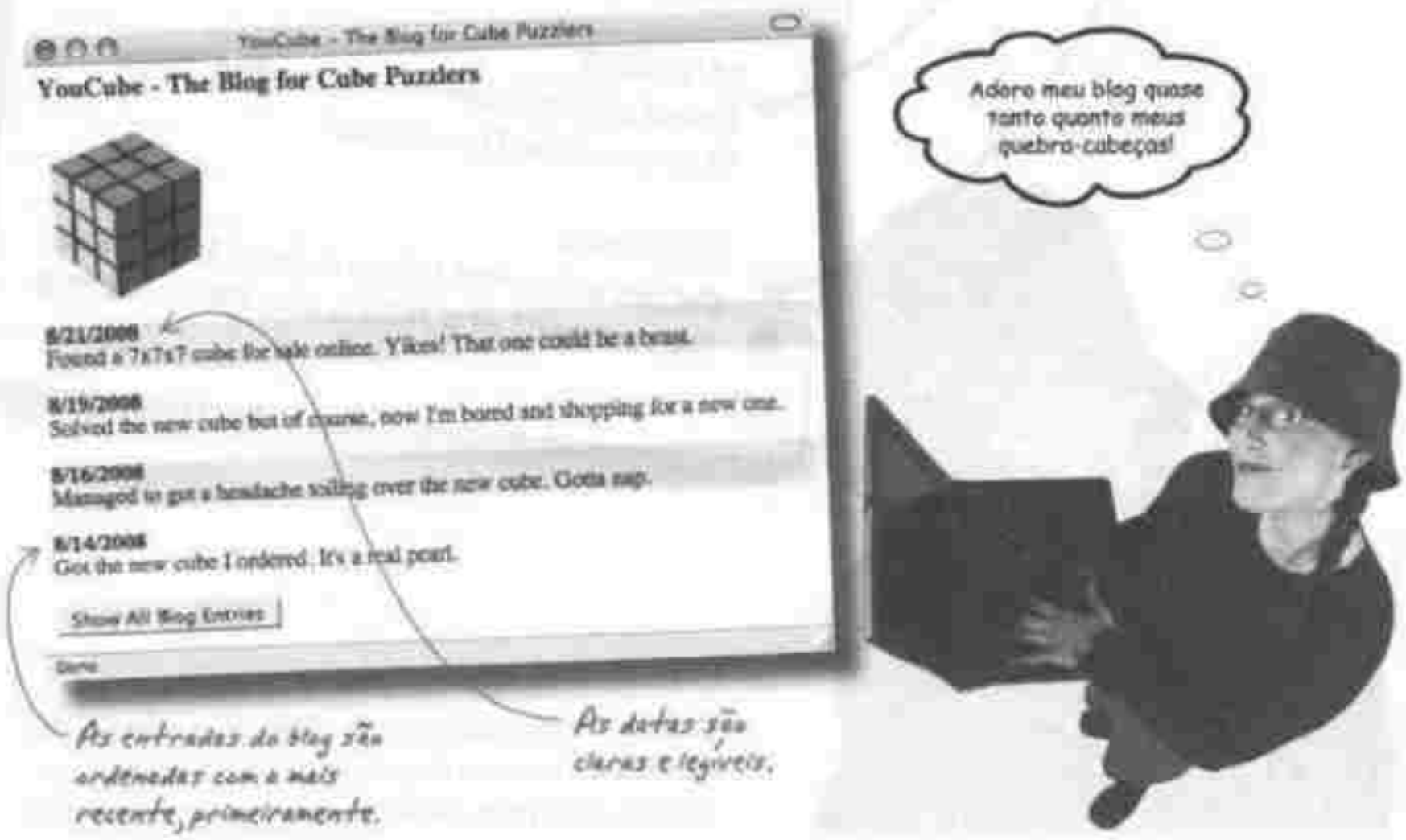
R: O objetivo de uma função de comparação de classificação é retornar um número cujo valor controle a classificação dos dois argumentos. No caso da comparação das datas, você deseja que a data mais recente seja classificada primeiramente. A data mais recente é a data maior, portanto subtrair a segunda data da primeira terá o resultado de classificar as datas recentes na frente das posteriores. Isso significa que a segunda data será classificada acima da primeira apenas se a segunda data for maior (o resultado é maior que 0).

P: Como o método `Array.sort( )` sabe usar uma função de comparação personalizada ou uma comparação padrão?

R: Esta decisão é tomada com base em um argumento ser ou não transmitido para a função `sort( )`. Se não houver nenhum argumento, uma comparação de classificação padrão será adotada. Se um argumento for fornecido, ele será interpretado como uma referência da função e usado como a base para comparar os itens na classificação. Portanto, a referência da função de comparação é um argumento **opcional**.

Ruby e seus cubos estão contentes

O blog YouCube, agora, está aproximando-se da visão de Ruby para um blog de quebra-cabeças do cubo que compartilha cada pensamento cubista seu com o universo.



A pesquisa seria ótima

O YouCube está se saindo muito bem, mas vários usuários têm solicitado um recurso de pesquisa que os permita procurar todos os envios do blog. Como Ruby pretende, possivelmente, ter muitos envios do blog, ela concorda que esse poderia ser um recurso muito útil, especialmente em uma longa jornada.

Uma string de pesquisa é fornecida e, então, usada para pesquisar todo o texto do corpo do blog.

Search the Blog | 7x7x7

onclick!

08/14/2008
Got the new cube I ordered. It's a real pearl.

08/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.

08/16/2008
Managed to get a headache toiling over the new cube.
Gotta nap.

08/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could be a beast.

08/29/2008
Met up with some fellow cubers to discuss the prospect of a 7x7x7 cube. Mixed feelings.

Um recurso de pesquisa permitiria aos usuários pesquisar todos os meus envios do blog simplesmente fornecendo um termo de pesquisa.

Ruby precisa apenas de um plano para saber como codificar o recurso de pesquisa no YouCube... os objetos poderiam, possivelmente, estar envolvidos?

Como pesquisar o array do blog

Um recurso de pesquisa para o YouCube envolve fazer um loop em cada entrada no array do blog procurando um texto coincidente em cada envio do blog.

Search the Blog | 7x7x7

Obtenha o texto de pesquisa com o usuário.

Faça um loop nas entradas.

Verifique a coincidência do texto em cada entrada do blog.

Saia do loop se houver uma coincidência.

Esta construção faz muito sentido, mas como pesquiso o texto em uma entrada do blog? Estou desorientada... mas sei que há um meio!

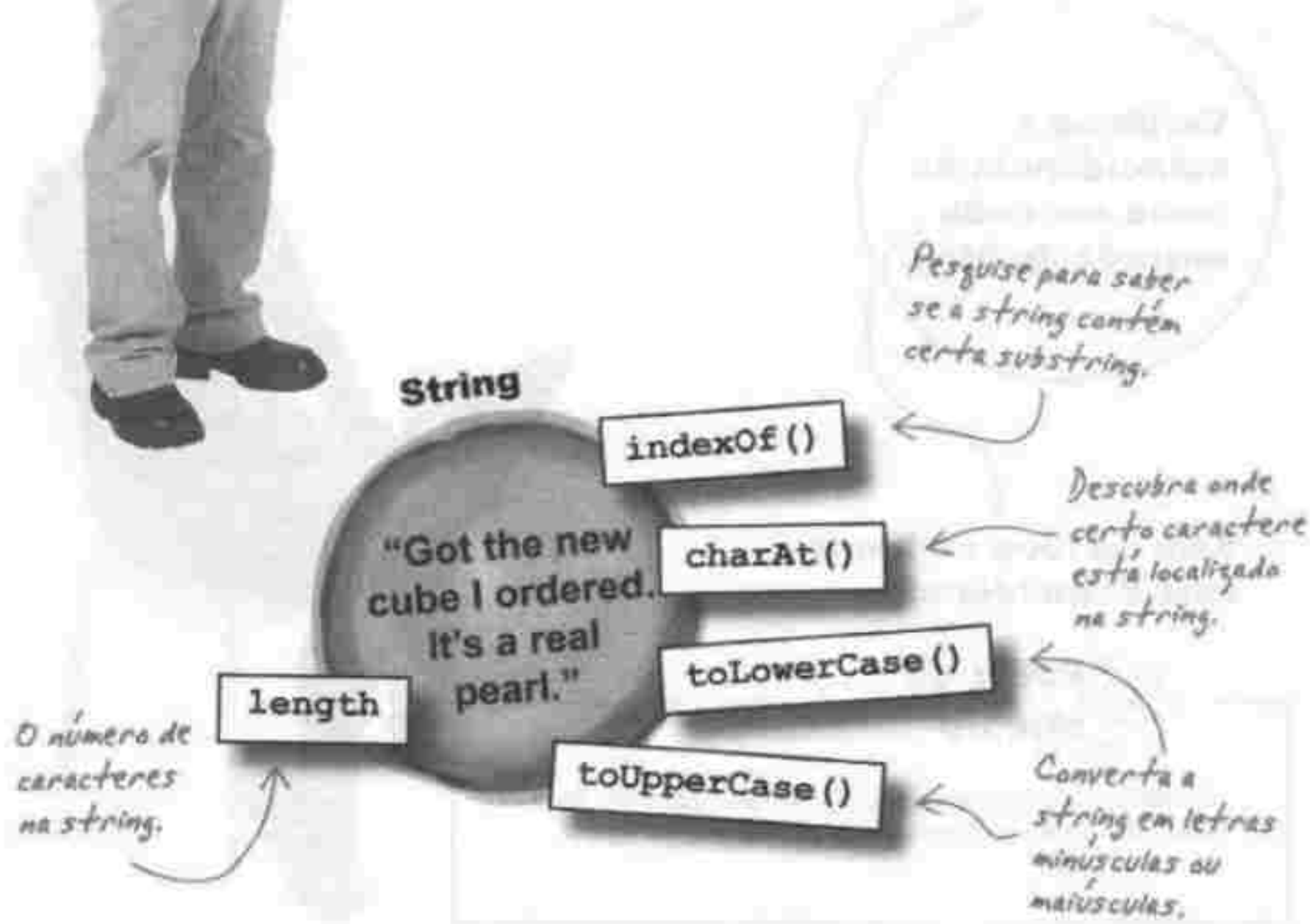
Poder do cérebro

Como você poderia pesquisar as entradas do blog YouCube para obter uma string coincidente de texto?

Já sabemos que uma string é, na verdade, um objeto. Portanto, é possível que a string possa simplesmente pesquisar a si mesma?

Uma string é um objeto pesquisável.

Talvez, você esteja começando a descobrir que os objetos estão em todo lugar no JavaScript. As strings são objetos e incluem muitos métodos úteis para interagir com os dados da string (texto). E, sim, um desses métodos permite pesquisar uma parte do texto dentro de uma string. Uma string dentro de uma string é algumas vezes referida como **substring**.



Como pesquisar as strings: indexOf( )

O método `indexOf( )` permite pesquisar uma string de texto, uma substring, dentro de um objeto String. A substring é transmitida como um argumento para o método `indexOf( )` – uma vez que você chama o método em um objeto String, não há nenhuma necessidade de transmitir nada mais. O método `indexOf( )` retornará o índice onde a substring está localizada ou –1 se não houver nenhuma coincidência.

```
var str = "Got the new cube I ordered. It's a real pearl.";
alert(str.indexOf("new"));
```

Para compreender de onde vem o número 8 neste exemplo, você tem que ver uma string de modo muito parecido com um array de caracteres individuais.

Quando `indexOf( )` for usado para pesquisar uma string que não existe, o resultado do método será `-1`.

```
var searchIndex = str.indexOf("used");
```

O resultado é -1, uma vez que a string de pesquisa não aparece no objeto String.

Abaixo, está uma das charadas favoritas de Ruby. Identifique o índice de cada ocorrência da substring "cub" na string da charada.

"Uma cubista cubou dois cubos e acabou com oito. Ela era cubana?"

Solução do Exercício

Abaixo, está uma das charadas favoritas de Ruby. Identifique o índice de cada ocorrência da substring "cub" na string da charada.

O início da string tem um índice 0.

A resposta não é óbvia?

```
"Uma cubista cubou dois cubos e acabou com oito. Ela era cubana?"
```

O índice da substring é 12.

O índice da substring é 23

Como pesquisar o array do blog

A pesquisa da string não é difícil demais graças ao método `indexOf( )` do objeto `String`, mas Ruby ainda tem um blog inteiro a pesquisar. Seu plano é fazer um loop no array de entradas do blog e usar o método `indexOf( )` para pesquisar uma substring no texto de corpo de cada entrada do blog. Se houver uma coincidência, ela desejará exibir a entrada do blog em uma caixa de alerta.

Antes de escrever uma função para lidar com a pesquisa real do blog, o blog YouCube precisará de um campo de texto para o texto de pesquisa, assim como um botão para iniciar uma pesquisa.

```
<input type="button" id="search" value="Search the Blog" onclick="searchBlog();" />
<input type="text" id="searchtext" name="searchtext" value="" />
```

O texto de pesquisa do blog é acessível através do ID do texto de pesquisa.

Texto da pesquisa!

O botão Search chama a função searchBlog( ) para pesquisar o blog.

Search the Blog | 7x7x7

Com os elementos de pesquisa HTML colocados, tudo que resta é reunir o código para a função `searchBlog( )`. Como a função usa um alerta para exibir os resultados da pesquisa, não há nenhuma necessidade de retornar informações dela. Também não há nenhuma necessidade de um argumento, pois a função lê diretamente o texto de pesquisa a partir do campo de texto HTML.

Ímãs de geladeira JavaScript

A função `searchBlog( )` do YouCube é responsável por fazer um loop no array de entradas do blog e pesquisar um texto coincidente em um corpo do blog. Ajude Ruby a terminar a função preenchendo o código que falta com os ímãs. Sugestão: O resultado da pesquisa coincidente deve ser exibido com sua data na forma *MM/DD/AAAA* dentro de colchetes, seguido do texto do corpo do blog.

```
function searchBlog( ) {
  var ------------------- = document.getElementById("------------------").value;
  for (var i = 0; i < ------------------.length; i++) {
    // Veja se a entrada do blog contém o texto de pesquisa
    if (blog[i]. ----------- .toLowerCase( ).indexOf(searchText.toLowerCase( )) != -1) {
      alert("[" + (blog[i] ---------- + ------------------ + ---------) + "/" +
        blog[i].date.getDate( ) + "/" + blog[i]. ----------- .getFullYear( ) + "] " +
        blog[i]. ----------- );
      break;
    }
  }
  // Se o texto da pesquisa não foi encontrado, exiba uma mensagem
  if (i == ------------------)
    alert("Sorry, there are no blog entries containing the search text.");
}
```

searchText

blog.length

date

searchText

body

1

getMonth( )

Solução dos ímãs de geladeira JavaScript

A função `searchBlog( )` do YouCube é responsável por fazer um loop no array de entradas do blog e pesquisar um texto coincidente em um corpo do blog. Ajude Ruby a terminar a função preenchendo o código que falta com os ímãs. Sugestão: O resultado da pesquisa coincidente deve ser exibido com sua data na forma *MM/DD/AAAA* dentro de colchetes, seguido do texto do corpo do blog.

Primeiro obtenha o texto de pesquisa no campo de texto HTML.

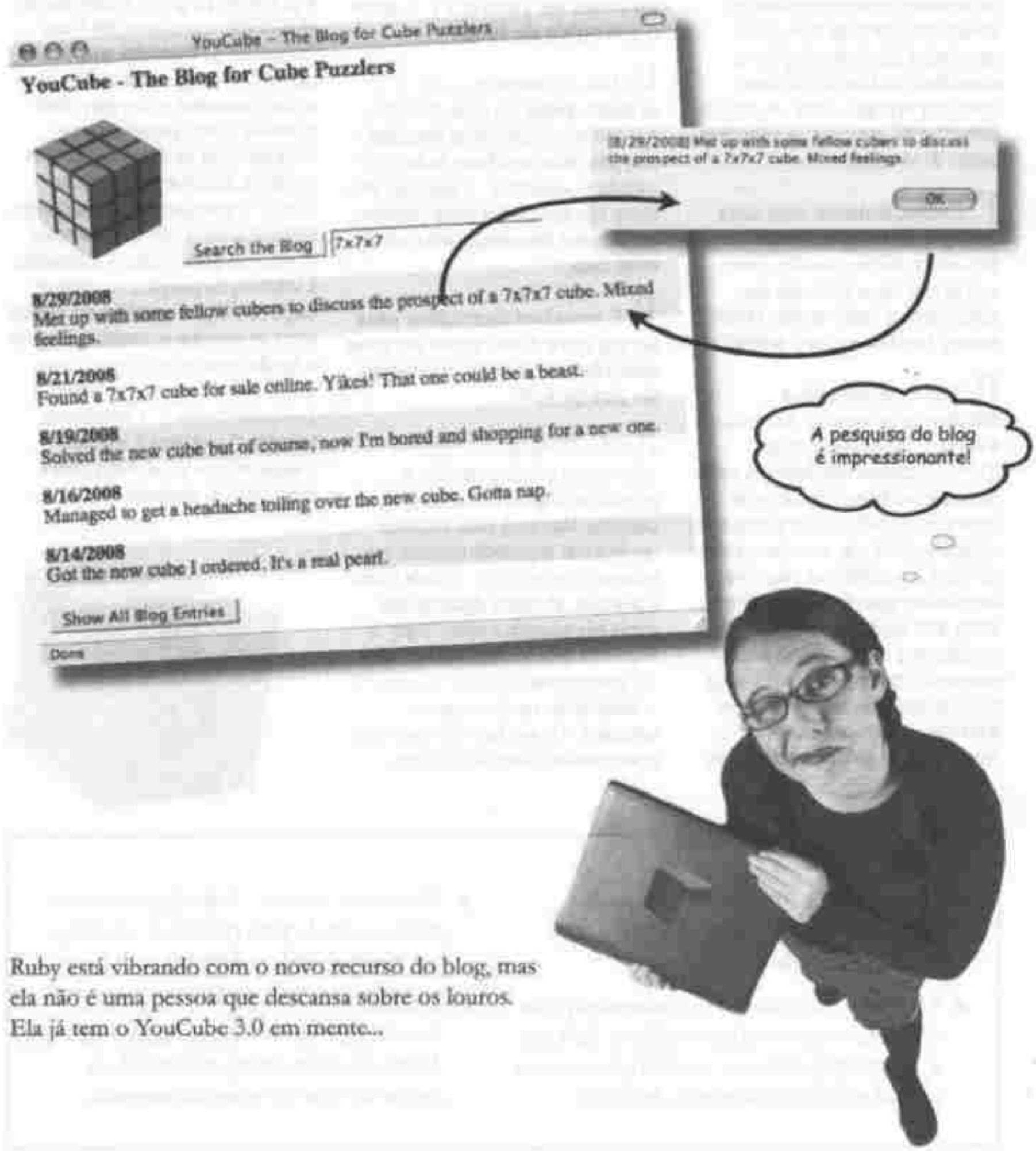
```
function searchBlog( ) {
  var searchText = document.getElementById("searchText").value;
  for (var i = 0; i < blog.length; i++) {
    // Veja se a entrada do blog contém o texto de pesquisa
    if (blog[i].body.toLowerCase( ).indexOf(searchText.toLowerCase( )) != -1) {
      alert("[" + (blog[i].date.getMonth( ) + 1) + "/" +
        blog[i].date.getDate( ) + "/" + blog[i].date.getFullYear( ) + "] " +
        blog[i].body);
      break;
    }
  }
  // Se o texto da pesquisa não foi encontrado, exiba uma mensagem
  if (i == blog.length) {
    alert("Sorry, there are no blog entries containing the search text.");
  }
}
```

Se i for igual ao comprimento do blog, significará que o loop for percorreu todas as entradas do blog sem encontrar uma coincidência.

A entrada do blog coincidente é formatada com MM/DD/AAAA entre colchetes, seguida do texto do corpo.

A pesquisa funciona agora também!

O YouCube 2.0, agora, está completo com um recurso de pesquisa que conta muito com as capacidades de pesquisa predefinidas no objeto `String`. É um ótimo exemplo de como os objetos tornam ativos os dados, neste caso transformando uma string de texto (dados puros) em uma entidade que tem um comportamento (pode pesquisar a si mesma). E talvez o mais importante, evitou que Ruby tivesse que inventar sua própria rotina de pesquisa, permitindo que ela se concentre na escrita de seu blog.



Ruby está vibrando com o novo recurso do blog, mas ela não é uma pessoa que descansa sobre os louros. Ela já tem o YouCube 3.0 em mente...

Não existem Perguntas Idiotas

P: Ainda não compreendo bem como toda string é realmente um objeto. É de fato verdade?

R: Sim. Toda string no JavaScript é um objeto. Se você colocar seu nome entre aspas no código JavaScript, como em "`Ruby`", acabará de criar um objeto. Embora possa parecer um excesso, o aspecto positivo do JavaScript tratar toda string como um objeto é que toda string tem a capacidade de fazer coisas úteis como, por exemplo, saber seu próprio comprimento, pesquisar as substrings dentro do si mesma etc.

P: Então, entendi que uma string é um objeto, mas também parece muito com um array com índices de caractere e tudo mais. Uma string também é um array?

R: Não. Uma string não é definitivamente um array. Contudo, é verdade que muitos métodos `String` operam nos dados da string como se eles fossem um array de caracteres individuais. Por exemplo, os caracteres dentro de uma string iniciam no índice 0 e contam um caractere de cada vez quando você percorre a string. Mas você não pode acessar um caractere dentro de uma string usando colchetes (`[ ]`), como pode com um array. Assim, embora ajude a considerar os caracteres dentro de um array como sendo parecidos com os elementos em um array, você não trabalha literalmente com um objeto `String` do mesmo modo como trabalha com um objeto `Array`.

P: A função `searchBlog( )` poderia ter usado `charAt( )` ao invés de `indexOf( )` para a pesquisa do blog?

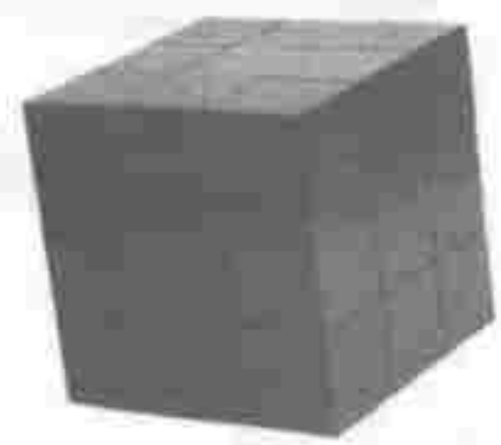
R: Não. O método `charAt( )` pesquisa apenas um único caractere, que não seria muito útil ao pesquisar o blog para obter uma frase de texto. O método `indexOf( )` pesquisa uma string, não apenas um único caractere e é a melhor ferramenta para o serviço neste caso.

P: É possível pesquisar uma string para obter mais de uma ocorrência de uma substring de pesquisa?

R: Sim. O método `indexOf( )` tem como padrão pesquisar a primeira ocorrência da substring de pesquisa. Mas você pode transmitir um segundo argumento opcional que informa a `indexOf( )` onde iniciar a pesquisa. Portanto, digamos que esteja pesquisando a string "cubo" e encontrou uma coincidência no índice 11. Você poderá chamar `indexOf( )` novamente com um segundo argumento 11, que fará com que inicie a pesquisa no índice 12. Portanto, a solução geral é transmitir o índice de pesquisa anterior para o método `indexOf( )` para continuar a pesquisar uma string.

P: Qual é a finalidade das duas chamadas para `toLowerCase( )` na função `searchBlog( )`?

R: Grande pergunta! A resposta tem relação com o problema das letras maiúsculas e minúsculas quando a pesquisa do texto é feita no blog. Se alguém pesquisar o blog para obter a palavra "cubo", provavelmente desejará todas as coincidências com a palavra, inclusive "cubo", "Cubo", "CUBO" e qualquer outra variação nas letras da palavra. Um modo simples de resolver esse problema é converter s substring de pesquisa e o texto do corpo do blog em uma letra em comum antes de executar a pesquisa. Embora a função `searchBlog( )` use `toLowerCase( )`, o método `toUpperCase( )` funcionaria também. O problema é remover inteiramente as letras maiúsculas e minúsculas da pesquisa.



Pontos de Bala

- O método `toString( )` é usado para converter qualquer objeto em uma representação de texto.
- Os arrays e as strings são realmente apenas objetos, contando com os objetos `Array` e `String` padrões no JavaScript para seus métodos e armazenamento de dados.
- O método `sort( )` do objeto `Array` pode ser usado para classificar um array em qualquer ordem desejada.
- O método `indexOf( )` no objeto `String` pesquisa uma string de texto dentro de outra string, retornando o índice do local da string de pesquisa.

Um YouCube aleatório

Na busca sem fim para manter os usuários interessados em seu blog, Ruby sugeriu mais um acréscimo ao YouCube que ela acha que seus colegas de quebra-cabeças gostarão. Ela deseja adicionar um botão Random (Aleatório) que permita aos visitantes exibir uma entrada do blog aleatoriamente.

08/11/2008
Got the new cube I ordered. It's a real

08/16/2008
Managed to get a headache toiling over the new cube. Gotta nap.

08/21/2008
Found a 7x7x7 cube for sale online. Yikes! That on
could be a beast.

08/29/2008
Met up with some fellow cubers to discuss the prospect of a 7x7x7 cube. Mixed feelings.

the new cube but of course, now I'm bored
and shopping for a new one.

Um recurso do blog aleatório adiciona um toque de diversão e mistério ao YouCube. Sou divertida e misteriosa!

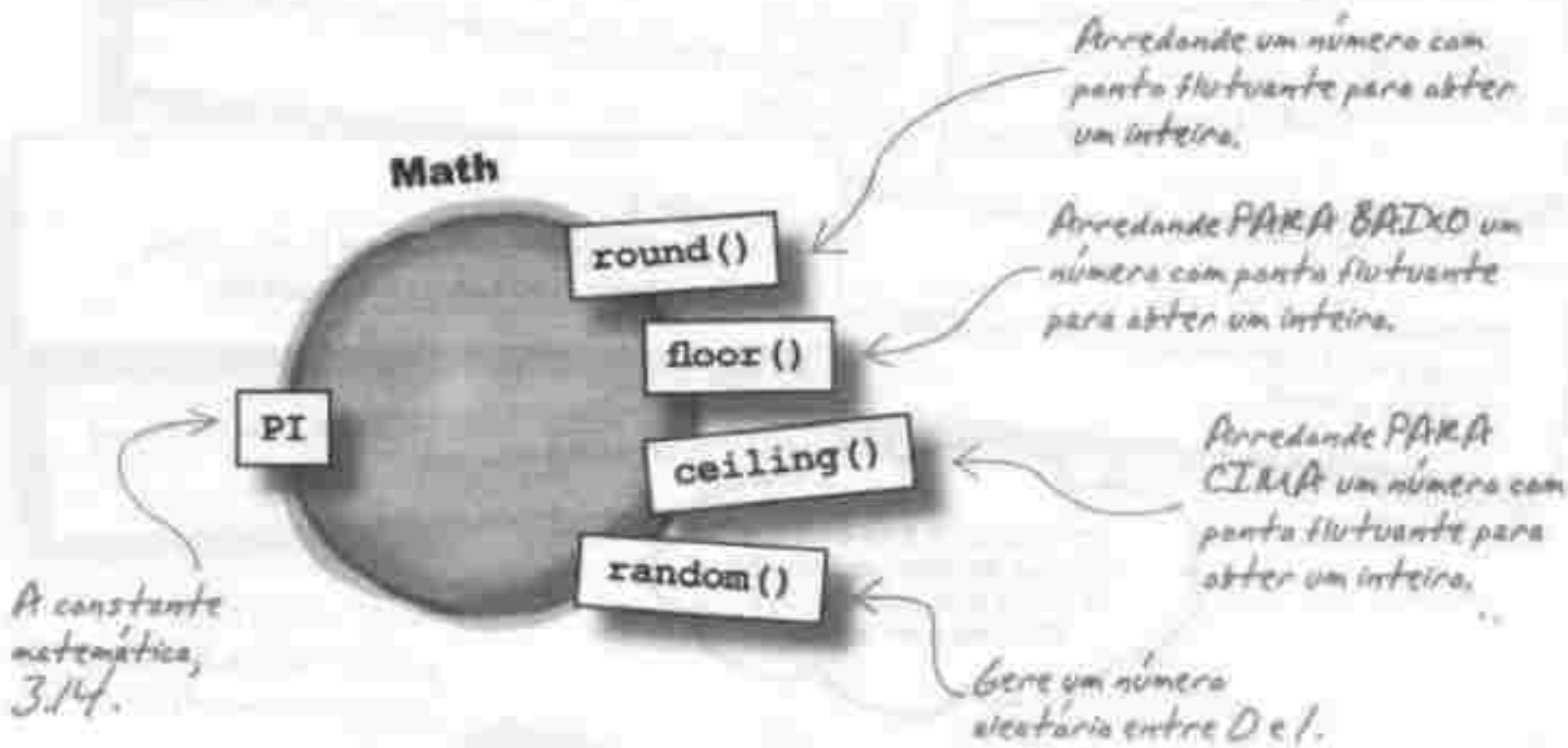
Ruby, a blogger de quebra-cabeça de cubo e a mulher dos mistérios.

Poder do cérebro

Você poderia escolher uma entrada do blog YouCube aleatoriamente?

O objeto Math é um objeto organizacional

Para ajudar Ruby a adicionar um recurso aleatório ao YouCube, precisamos desesperadamente de um modo de gerar números aleatórios. Isso envolve usar um objeto JavaScript predefinido que não é tão "vivo" quanto alguns dos outros objetos usados. O objeto `Math` padrão é onde os números aleatórios podem ser gerados, é um objeto único no sentido de que não tem nenhum dado que muda e nenhum método que atue nos dados internos.



O objeto Math é um **objeto organizacional** que significa que é apenas uma coleção de métodos e constantes utilitários relacionados à matemática. Não existe nenhuma variável, o que significa que o objeto `Math` não mantém nenhum estado – você não pode usá-lo para armazenar nada. Os únicos dados que ele contém são algumas constantes como, por exemplo, PI (3.14). Os métodos no objeto `Math`, porém, são bem úteis. Por exemplo, o método `random( )` gera um número com ponto flutuante aleatório entre 0 e 1.

O objeto Math é um objeto organizacional que hospeda métodos e constantes matemáticos.

Exercício

Escreva os resultados das seguintes chamadas para os métodos `Math`.

```
Math.round(Math.PI)        ..................................

Math.ceiling(Math.PI)      ..................................

Math.random( )             ..................................
```

→ Respostas na página 436.

Objeto Math em exposição

Entrevista desta semana:

Quando as funções matemáticas entram em conflito

Use a Cabeça: Tudo bem, realmente estou confuso. Você é um objeto, mas ouvi falar que não faz nada, exceto manter alguns métodos matemáticos e algumas constantes. Pensei que o objetivo dos objetos fosse tornar os dados ativos. Você sabe, integrar alguns dados e, então, ter métodos que façam coisas legais com eles.

Math: É isso que o bom senso JavaScript convencional leva pessoas a pensar, mas nem todos os objetos são para dar vida aos dados. É perfeitamente aceitável para um objeto desempenhar o papel de organizador, como eu.

Use a Cabeça: Mas todos esses métodos matemáticos não poderiam ter sido criados apenas como funções padrões?

Math: Sim, poderiam, mas você está esquecendo que a linguagem JavaScript é composta de objetos. Portanto, na realidade, não existe tal coisa como uma função "padrão".

Use a Cabeça: Mas eu posso criar uma função fora de um objeto e ela parece funcionar bem.

Math: Certo, mas na verdade todas as funções realmente são métodos porque pertencem a um objeto em algum lugar, mesmo que ele esteja oculto. Isso ajuda a explicar por que não há "funções padrões".

Use a Cabeça: Ah, compreendo. Está começando a fazer mais sentido por que você contém aqueles métodos matemáticos.

Math: E não se esqueça de que só porque eu não tenho dados internos que são manipulados por meus métodos, não significa que não desempenho um papel importante ao ser um objeto.

Use a Cabeça: O que você quer dizer?

Math: Bem, imagine um grupo de pessoas que compartilham um interesse em comum como, por exemplo, os quebra-cabeças de cubo. Em muitos casos, tais pessoas organizam-se para que possam interagir umas com as outras sobre seu interesse. Embora os métodos matemáticos não sejam exatamente tão sociais quanto as pessoas, eles aproveitam a organização que eu forneço.

Use a Cabeça: Porque estão todos relacionados a um interesse em comum, você quer dizer.

Math: Sim! E esse interesse é executar tarefas matemáticas como, por exemplo, arredondar números, executar operações trigonométricas e gerar números aleatórios.

Use a Cabeça: Você menciona a geração de números aleatórios. Ouvi dizer que seus números não são realmente aleatórios. Há alguma verdade neste boato?

Math: Tenho que confessar que não, eles não são realmente aleatórios. E nem a maioria dos números aleatórios gerados pelo computador. Meus números aleatórios são "pseudo-aleatórios", o que é bom o bastante para grande parte das situações.

Use a Cabeça: Pseudo-aleatórios, é como a pseudociência... ou o pseudocódigo?

Math: Ih, não e sim. Não, nada de pseudociência. E sim, um pouco como o pseudocódigo, uma vez que o pseudocódigo é para representar a idéia por trás do código sem realmente ser o código. No caso dos números pseudo-aleatórios, eles aproximam-se do caráter aleatório sem realmente serem aleatórios.

Use a Cabeça: Portanto, posso acreditar que os números pseudo-aleatórios são suficientemente aleatórios para a maioria das aplicações JavaScript?

Math: Sim, e é uma boa maneira de colocar isso: "suficientemente aleatórios". Provavelmente, você não gostaria de confiar nos números pseudo-aleatórios para as questões que envolvem a segurança nacional, mas funcionam muito bem para introduzir o caráter aleatório em cada script do cotidiano.

Use a Cabeça: Entendi. Bem, obrigado por seu tempo... e por sua honestidade em relação aos números aleatórios.

Math: Prazer... você sabe que não posso mentir.

Exercício Solução

Escreva os resultados das seguintes chamadas para os métodos Math.

O valor da constante PI é 3.14.

```
Math.round(Math.PI)      ..... 3
Math.ceiling(Math.PI)    ..... 4
Math.random( )           ..... ?
```

3.14 arredonda para 3.

O máximo (arredondar para cima) de 3.14 é 4.

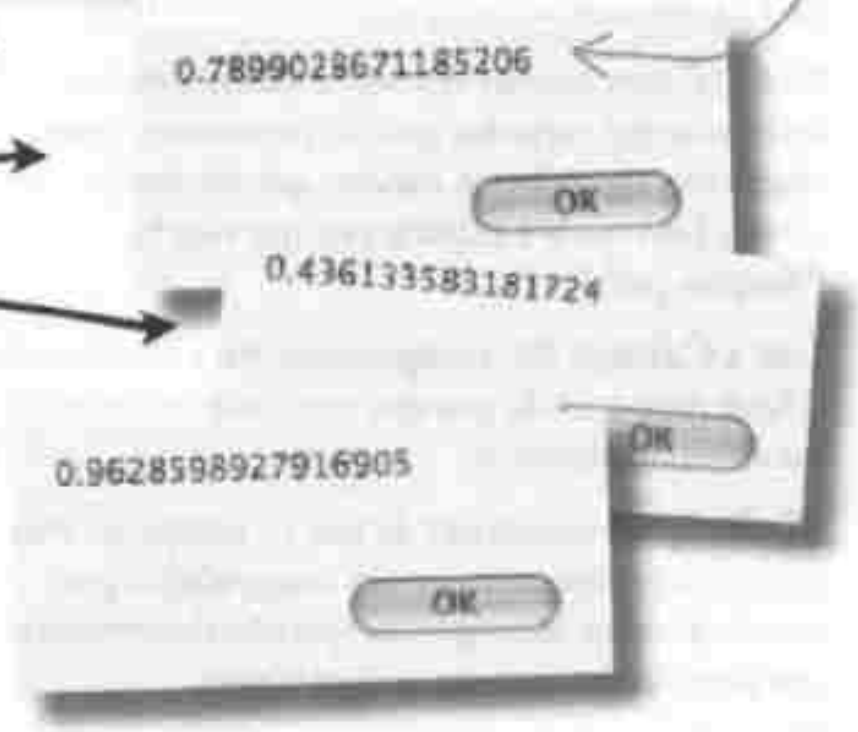
Desculpe, foi uma questão capciosa! Você não pode saber porque é aleatório.

Gere números aleatórios com Math.random

Pseudo-aleatórios ou não, os números aleatórios gerados pelo método `random( )` do objeto `Math` são extremamente úteis em aplicativos como, por exemplo, o YouCube, que precisam fazer uma seleção aleatória a partir de uma coleção de dados. O problema é que `random( )` retorna um número aleatório na faixa de 0 e 1, ao passo que Ruby precisa de um número aleatório que esteja na faixa de 0 até o final do array do blog. Em outras palavras, ela precisa gerar um índice aleatório do blog.

Cada um dos números aleatórios está na faixa de 0 a 1.

```
alert(Math.random());
alert(Math.random());
alert(Math.random());
```



Para gerar um número aleatório dentro de uma faixa diferente de 0 a 1, você tem que confiar no objeto `Math` um pouco mais e usar outro método. O método `floor( )` arredonda para baixo um número até o inteiro mais próximo e é perfeito para gerar inteiros aleatórios dentro de uma faixa dada de inteiros.

```
var oneToSix = Math.floor(Math.random() * 6) + 1;
```

0-5

1-6

Não existem Perguntas Idiotas

P: Por que não é necessário criar um objeto `Math` antes de usá-lo?

R: Ah, esta é uma pergunta perceptiva e toca em um conceito muito importante relacionado aos objetos. Como o objeto `Math` não contém, de fato, dados que podem mudar, também conhecidos como dados de instância, não há nenhuma necessidade de criar um objeto. Lembre-se de que o objeto `Math` é apenas uma coleção de métodos estáticos e constantes, portanto tudo que entra no objeto `Math` já existe – não há nada a criar. Isso fará muito mais sentido no Capítulo 10, quando você aprender sobre os detalhes das instâncias e das classes do objeto.

P: Qual é a diferença entre os métodos `round( )` e `floor( )` do objeto `Math`?

R: O método `round( )` arredonda para cima ou para baixo o número dependendo de sua parte decimal. Por exemplo, `Math.round(11.375)` resulta em 11, ao passo que `Math.round(11.625)` resulta em 12. O método `floor( )`, por outro lado, sempre arredonda para baixo, não importando qual é a parte decimal. Você pode simplesmente considerar o método `floor( )` como sempre cortando a parte decimal.

P: O que mais o objeto `Math` pode fazer?

R: Muitas coisas. Dois métodos úteis dos quais não precisamos ainda são `min( )` e `max( )`, que analisam dois números e retornam o menor ou o maior deles. `abs( )` é outro método `Math` muito útil – seu trabalho é retornar um número positivo, não importando o número fornecido.

Nerd

Se você estiver trabalhando em um aplicativo JavaScript que precisa desesperadamente de números aleatórios verdadeiros, veja http://random.org para aprender mais sobre como ir além do domínio dos números pseudo-aleatórios.

Aponte seu Lápis

Escreva o código para uma função `randomBlog( )` que seleciona uma entrada do blog aleatoriamente e, então, mostre-a em uma caixa de alerta. Sugestão: A entrada do blog na caixa de alerta pode ser formatada como o resultado da pesquisa em `searchBlog( )`.

Aponte seu Lápis
Solução

Escreva o código para uma função `randomBlog( )` que seleciona uma entrada do blog aleatoriamente e, então, mostre-a em uma caixa de alerta. Sugestão: A entrada do blog na caixa de alerta pode ser formatada como o resultado da pesquisa em `searchBlog( )`.

Use um número aleatório para selecionar uma entrada do blog.

Gere um número aleatório entre 0 e um menos o comprimento do blog.

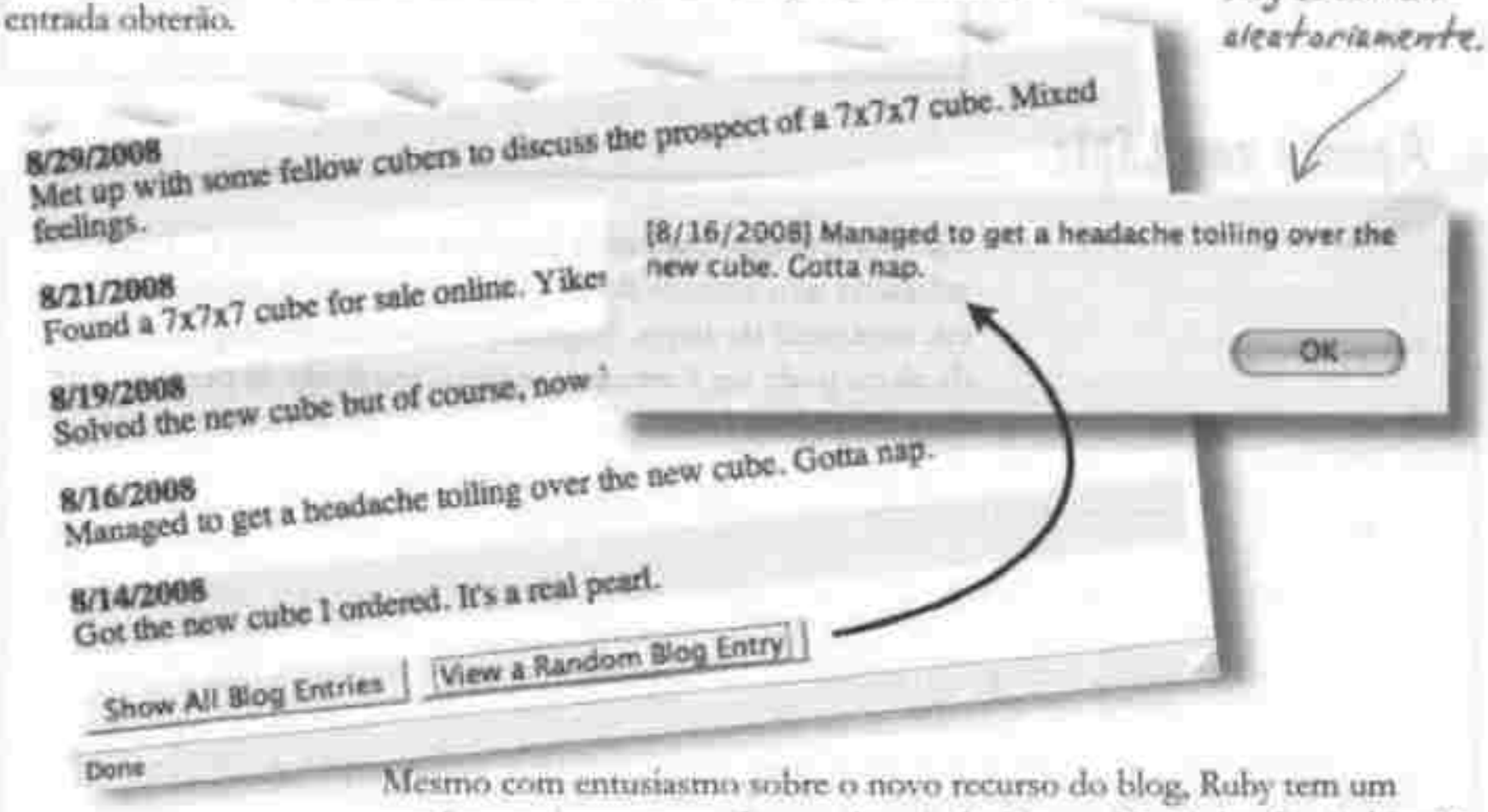
```
function randomBlog( ) {

// Selecione um número aleatório entre 0 e blog.length - 1

var i = Math.floor(Math.random( ) * blog.length);

alert("[" + (blog[i].date.getMonth( ) + 1) + "/" + blog[i].date.getDate( ) + "/" +

blog[i].date.getFullYear( ) + "] " + blog[i].body);

}
```

Formate a entrada do blog com uma data MM/DD/AAAA seguida do texto do corpo.

Aleatório, mas ainda falta

Agora, o blog de Ruby suporta um recurso de pesquisa do blog aleatório, com o qual ela está muito contente. Agora, os usuários podem exilir o blog YouCube com um sentimento saudável de curiosidade, uma vez que não sabem qual entrada obterão.

A entrada do blog escolhida aleatoriamente.



Mesmo com entusiasmo sobre o novo recurso do blog, Ruby tem um sentimento inoportuno de que algo ainda falta no YouCube. Seu "objeto" Blog é atualmente apenas algumas propriedades que contam com muitas funções separadas. Isso não parece uma construção de objeto muito boa...

Um objeto em busca de ações

Os instintos de Ruby sobre o objeto YouCube estão certíssimos. A parte comportamental do objeto é extremamente ausente e poderia usar alguma reestruturação crítica para que use métodos para lidar com as tarefas específicas do blog. Ruby precisa de métodos que acrescentem algumas **ações** ao objeto `Blog`!

Aponte seu Lápis

Leia todo o código YouCube e circule qualquer código que ache que poderia ser colocado nos métodos `Blog`; nomeie cada método.

```
function showBlog(numEntries) {
   // Primeiro classifique o blog na ordem cronológica inversa (a mais recente primeiro)
   blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

   // Ajuste o número de entradas para exibir o blog completo, se necessário
   if (!numEntries)
      numEntries = blog.length;

   // Exibe as entradas do blog
   var i = 0, blogText = "";
   while (i < blog.length && i < numEntries) {
      // Use um segundo plano cinza para uma entrada sim outra não do blog
      if (i % 2 == 0)
         blogText += "<p style='background-color:#EEEEEE'>";
      else
         blogText += "<p>";

      // Gere o código HTML formatado do blog
      blogText += "<strong>" + (blog[i].date.getMonth( ) + 1) + "/" +
         blog[i].date.getDate( ) + "/" +
         blog[i].date.getFullYear( ) + "</strong><br />" +
         blog[i].body + "</p>";
      i++;
   }

   // Defina o código HTML do blog na página
   document.getElementById("blog").innerHTML = blogText;
}

function searchBlog( ) {
   var searchText = document.getElementById("searchtext").value;
   for (var i = 0; i < blog.length; i++) {
      // Veja se a entrada do blog contém o texto de pesquisa
      if (blog[i].body.toLowerCase( ).indexOf(searchText.toLowerCase( )) != -1) {
         alert("[" + (blog[i].date.getMonth( ) + 1) + "/" + blog[i].date.getDate() + "/" +
            blog[i].date.getFullYear( ) + "] " + blog[i].body);
         break;
      }
   }

   // Se o texto de pesquisa não foi encontrado, exiba uma mensagem
   if (i == blog.length)
      alert("Sorry, there are no blog entries containing the search text.");
}

function randomBlog( ) {
   // Selecione um número aleatório entre 0 e blog.length - 1
   var i = Math.floor(Math.random() * blog.length);
   alert("[" + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
      blog[i].date.getFullYear() + "] " + blog[i].body);
}
```

Aponte seu Lápis

Estude o código YouCube e circule qualquer código que ache que poderia ser colocado nos métodos Blog; nomeie cada método.

```
function showBlog(numEntries) {
   // Primeiro classifique o blog na ordem cronológica inversa (a mais recente primeiro)
   blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

   // Ajuste o número de entradas para exibir o blog completo, se necessário
   if (!numEntries)
      numEntries = blog.length;

   // Exiba as entradas do blog
   var i = 0, blogText = "";
   while (i < blog.length && i < numEntries) {
      // Dê um segundo plano cinza para uma entrada sim outra não do blog
      if (i % 2 == 0)
         blogText += "<p style='background-color:#EEEEEE'>";
      else
         blogText += "<p>";

      // Gere o código HTML formatado do blog
      blogText += "<strong>" + (blog[i].date.getMonth( ) + 1) + "/" +
         blog[i].date.getDate( ) + "/" +
         blog[i].date.getFullYear( ) + "</strong><br />" +
         blog[i].body + "</p>";
      i++;
   }

   // Defina o código HTML do blog na página
   document.getElementById("blog").innerHTML = blogText;
}

function searchBlog( ) {
   var searchText = document.getElementById("searchtext").value;
   for (var i = 0; i < blog.length; i++) {
      // Veja se a entrada do blog contém o texto de pesquisa
      if (blog[i].body.toLowerCase( ).indexOf(searchText.toLowerCase( )) != -1) {
         alert("[" + (blog[i].date.getMonth( ) + 1) + "/" + blog[i].date.getDate() + "/" +
           blog[i].date.getFullYear( ) + "] " + blog[i].body);
         break;
      }
   }

   // Se o texto de pesquisa não foi encontrado, exiba uma mensagem
   if (i == blog.length)
      alert("Sorry, there are no blog entries containing the search text.");
}

function randomBlog( ) {
   // Selecione um número aleatório entre 0 e blog.length - 1
   var i = Math.floor(Math.random() * blog.length);
   alert("[" + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
      blog[i].date.getFullYear() + "] " + blog[i].body);
}
```

Converte a entrada do blog no HTML formatado - blog.toHTML( ).

Blog.toHTML( )
Converte uma entrada do blog no código HTML formatado, que retira muita responsabilidade do outro código que deseja exibir um blog com uma formatação limpa.

Blog.containsText( )
Não é muito código, mas ainda um método digno, uma vez que uma entrada do blog deve ser capaz de pesquisar seu próprio corpo para obter o texto.

Blog.toString( )
Converte uma entrada do blog em uma string, que faz sentido usar em qualquer situação onde a data é mostrada entre colchetes com o texto do corpo dentro.

Não existem Perguntas Idiotas

P: Como você sabe qual código de script deve entrar em um método?

R: Bem, primeiro você tem que se lembrar o que um método deve fazer, como o ideal, e isso é para tomar algum tipo de ação com base no estado (dados) de um objeto. Até certo ponto, descobrir os métodos para um objeto envolve descobrir o que o objeto está realmente fazendo ou precisa fazer. Então, concentre-se em capacitar os objetos para que façam as coisas por si mesmos.

Como exemplo, faz sentido para o objeto `Blog` transformar-se em uma string ou código HTML formatado, uma vez que essas duas ações requerem acesso aos dados internos do blog. Do mesmo modo, pesquisar o texto dentro de uma entrada do blog é uma ação que deve ser interna para o objeto `Blog` e, portanto, faz muito sentido como um método.

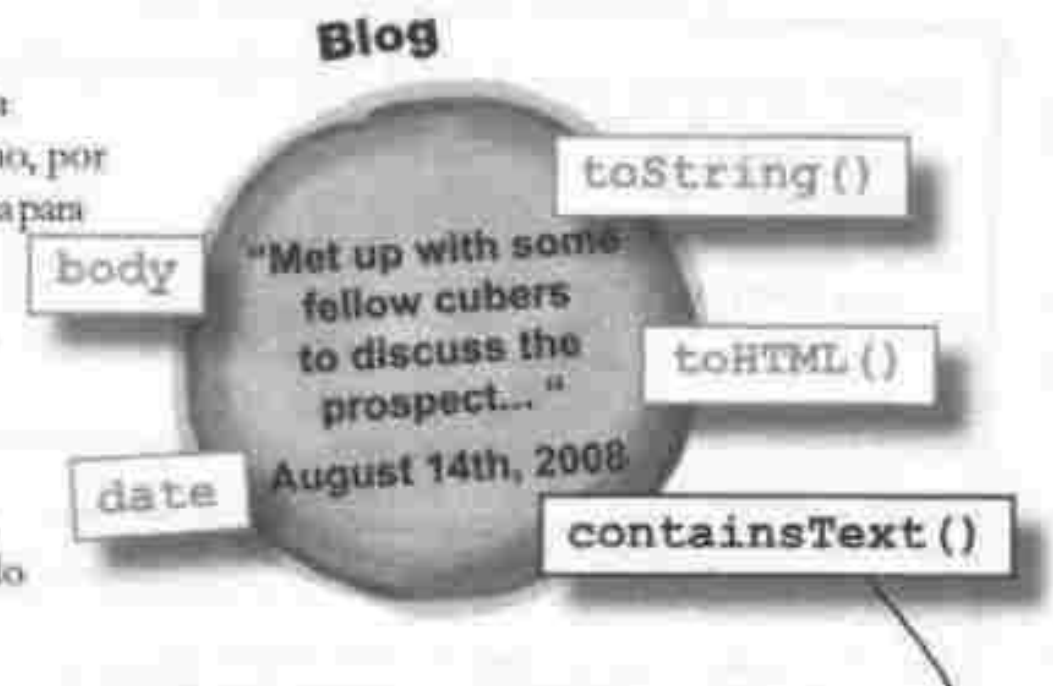
P: Portanto, há um exemplo de uma ação que o objeto Blog não deveria tomar?

R: As ações que estão muito fora do escopo do objeto `Blog` poderiam ser coisas como, por exemplo, exibir ou pesquisar a lista de entradas do blog. É porque o objeto `Blog` representa uma única entrada do blog. É por isso que o array do blog consiste em diversos objetos `Blog` **individuais**. Assim, cada objeto `Blog` individual não precisa se preocupar com uma coleção de objetos `Blog`. Ao contrário, um objeto `Blog` individual deve cuidar de seu próprio negócio, que envolve tomar uma ação com base unicamente em sua própria data e texto do corpo.

Transforme uma função em um método

Agora que algumas partes do código YouCube foram separadas, ajustando-se bem como métodos do objeto `Blog`, vejamos mais de perto como converter uma delas em um método `Blog`. O método é `containsText( )`, que assume a responsabilidade de pesquisar o corpo de uma entrada do blog para obter uma substring. Mover o código de pesquisa para um método envolve, basicamente, operar diretamente na propriedade `body` de um objeto `Blog`, em oposição a uma variável local na função `searchBlog( )`. Estas etapas ajudam a esclarecer o processo:

1. Declare o método, complete com uma lista de argumentos, se requerido, como, por exemplo, o argumento do texto de pesquisa para `containsText( )`.
2. Mova o código existente para o novo método.
3. Mude o código relevante para usar as propriedades do objeto como, por exemplo, `this.body` no método `containsText( )`.



Aponte seu Lápis

Escreva o código para o método `containsText( )` do objeto `Blog`, que é criado no construtor `Blog( )` atribuindo uma literal da função a `this.containsText`.

Aponte seu Lápis
Solução

Escreva o código para o método `containsText( )` do objeto `Blog`, que é criado no construtor `Blog( )` atribuindo uma literal da função a `this.containsText`.

O método é criado atribuindo uma literal da função a uma referência do método.

```
this.containsText = function(text) {
  return (this.body.toLowerCase( ).indexOf(text.toLowerCase( )) != -1);
};
```

A palavra-chave this é usada para criar um método do mesmo modo como é usada para criar as propriedades.

O código dentro do método acessa uma propriedade do objeto diretamente usando a palavra-chave this.

Revele o novo objeto do blog brilhante

Os outros dois métodos novos do blog juntam-se ao método `containsText( )` em uma nova versão do objeto `Blog` que tem propriedades **e** métodos.

Ei, dei uma forma muito boa!

Crie e inicialize as propriedades.

O método toString( ) retorna a entrada do blog formatada como uma string de texto.

O método toHTML retorna a entrada do blog como um código HTML formatado elegante.

O método containsText( ) retornará true se a texto do corpo contiver a string de pesquisa ou false, do contrário.

```
function Blog(body, date) {
  // Atribua as propriedades
  this.body = body;
  this.date = date;

  // Retorne uma representação de string da entrada do blog
  this.toString = function( ) {
    return "[" + (this.date.getMonth( ) + 1) + "/" + this.date.getDate( ) + "/" +
      this.date.getFullYear( ) + "] " + this.body;
  };

  // Retorne uma representação HTML formatada da entrada do blog
  this.toHTML = function(highlight) {
    // Use um segundo plano cinza como um destaque, se especificado
    var blogHTML = "";
    blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";

    // Gere o código HTML do blog formatado
    blogHTML += "<strong>" + (this.date.getMonth( ) + 1) + "/" +
      this.date.getDate( ) + "/" + this.date.getFullYear( ) + "</strong><br />" +
      this.body + "</p>";
    return blogHTML;
  };

  // Veja se o corpo do blog contém uma string de texto
  this.containsText = function(text) {
    return ((this.body.toLowerCase( ).indexOf(text.toLowerCase( )) != -1);
  };
}
```

O que os objetos realmente oferecem ao YouCube?

Foi somente na nova versão do objeto `Blog` (disponível em *www.altabooks.com.br*) ligada ao script YouCube que as vantagens reais da programação orientada a objeto foi mostrada. Agora que as várias tarefas importantes específicas do blog são delegadas aos métodos `Blog`, o código do script fica muito mais simples.

O novo objeto Blog simplifica o script YouCube.

```
// Exiba a lista de entradas do blog
function showBlog(numEntries) {
   // Primeiro, classifique o blog na ordem cronológica inversa (a mais recente primeiro)
   blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

   // Ajuste o número de entradas para exibir o blog completo, se necessário
   if (!numEntries)
      numEntries = blog.length;

   // Exiba as entradas do blog
   var i = 0, blogListHTML = "";
   while (i < blog.length && i < numEntries) {
      blogListHTML += blog[i].toHTML(i % 2 == 0);
      i++;
   }

   // Defina o código HTML do blog na página
   document.getElementById("blog").innerHTML = blogListHTML;
}

// Pesquise a lista de entradas do blog para obter uma parte do texto
function searchBlog( ) {
   var searchText = document.getElementById("searchtext").value;
   for (var i = 0; i < blog.length; i++) {
      // Veja se a entrada do blog contém o texto da pesquisa
      if (blog[i].containsText(searchText)) {
         alert(blog[i]);
         break;
      }
   }

   // Se o texto de pesquisa não foi encontrado, exiba uma mensagem
   if (i == blog.length)
      alert("Sorry, there are no blog entries containing the search text.");
}

// Exiba uma entrada do blog escolhida aleatoriamente
function randomBlog( ) {
   // Selecione um número aleatório entre 0 e blog.length - 1
   var i = Math.floor(Math.random( ) * blog.length);
   alert(blog[i]);
}
```

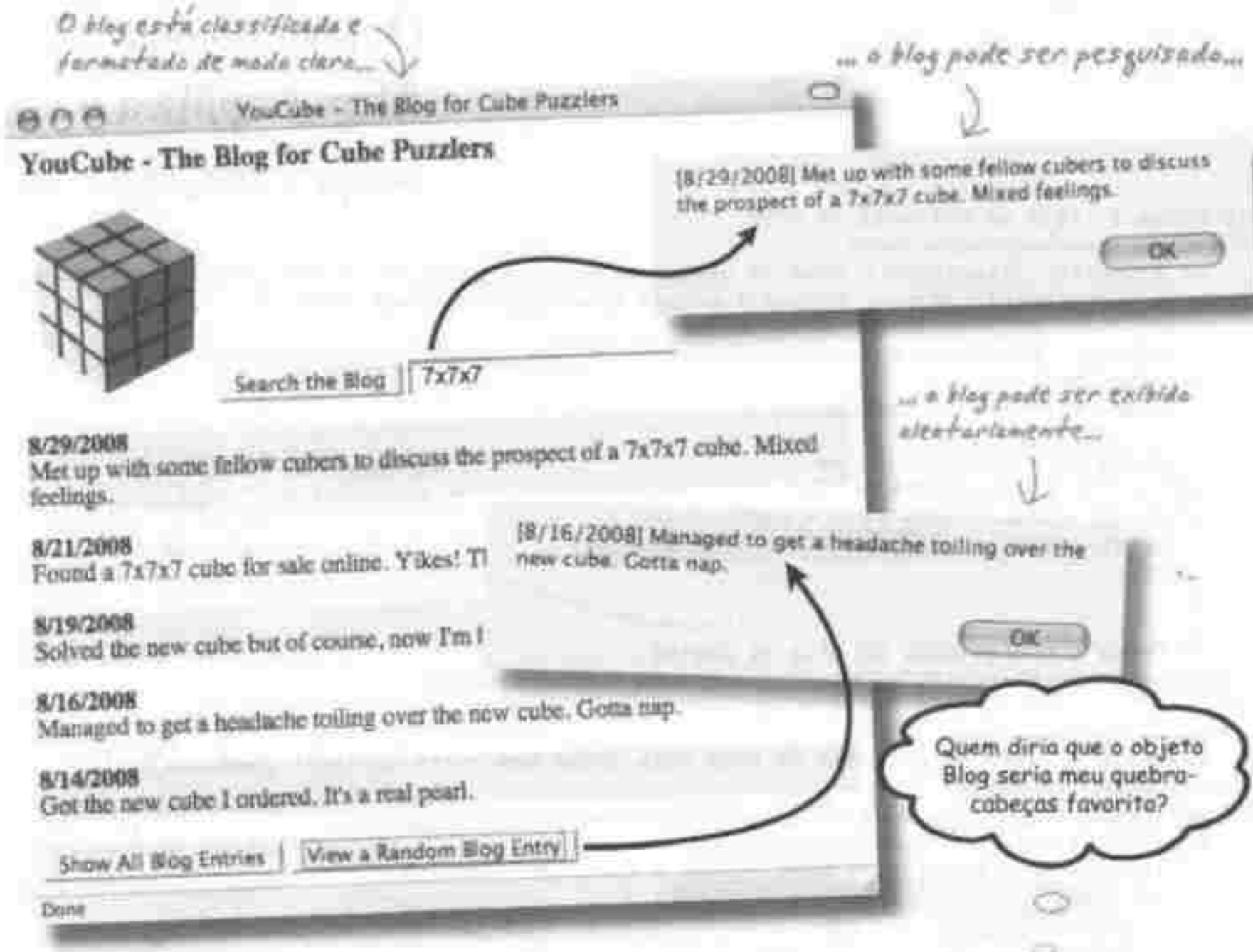
O método toHTML( ) é totalmente responsável por formatar uma entrada do blog como um código HTML.

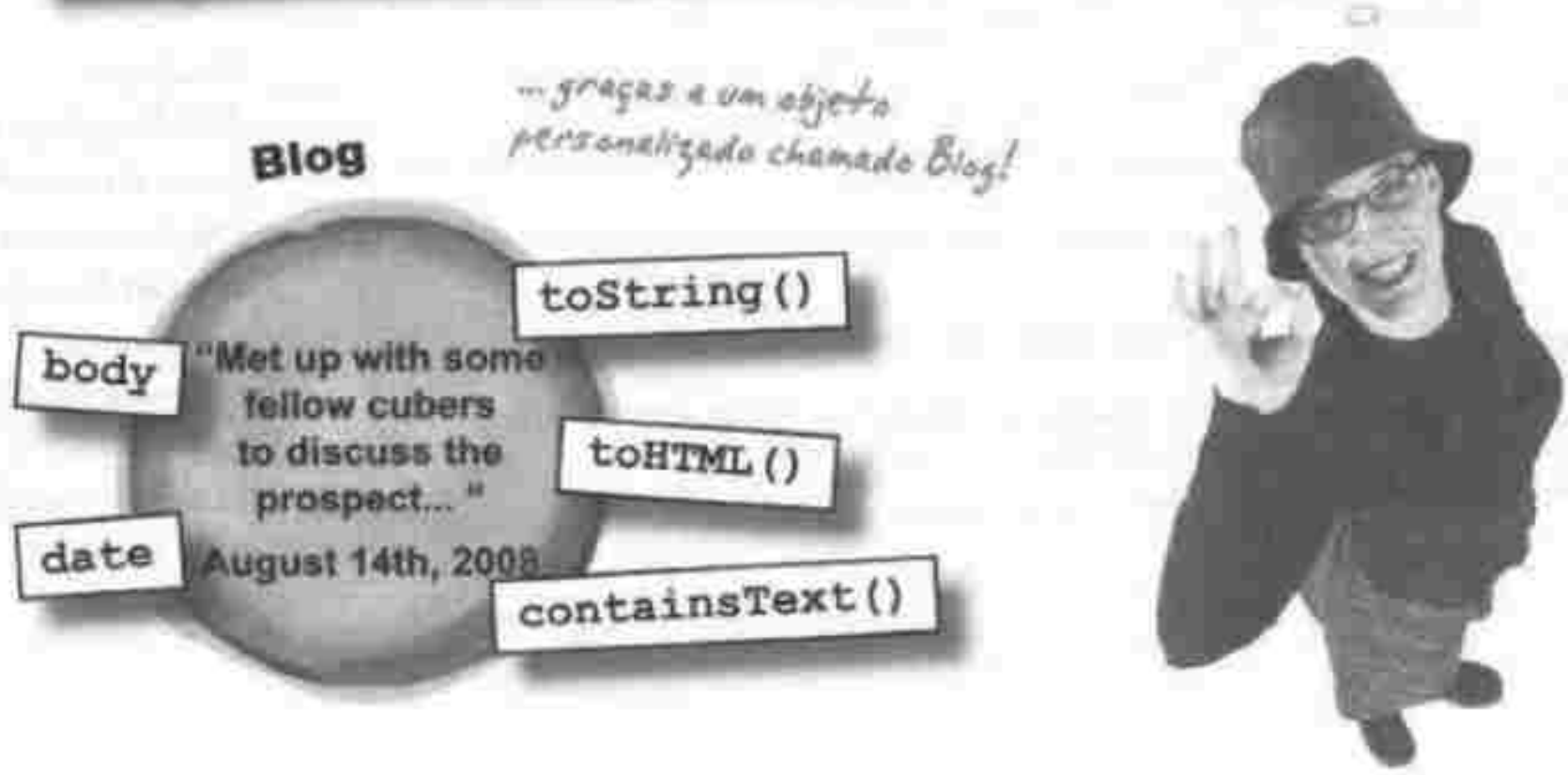
O método containsText( ) cuida da pesquisa de uma entrada do blog para obter uma substring.

O método toString( ) é um pouco mais sutil, sendo chamado automaticamente quando uma entrada do blog é usada em um contexto onde uma string é esperada.

YouCube 3.0!

Era um projeto, mas Ruby resolveu oficialmente que o YouCube 3.0 está bom o bastante para ela dar uma parada e voltar a fazer o quebra-cabeças. Ela também está entusiasmada em passar algum tempo preparando-se para aquela festa para a qual foi convidada...

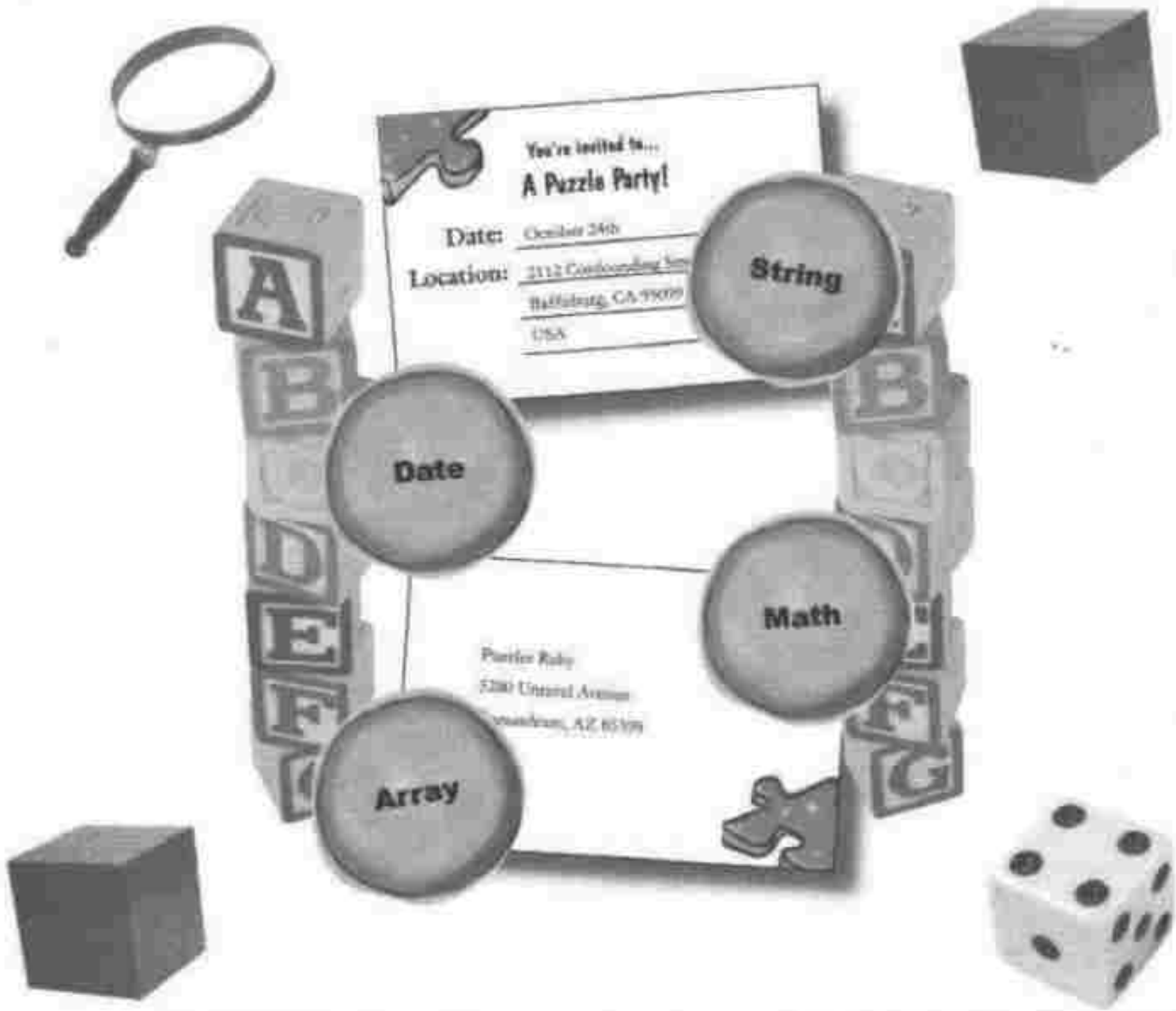




DOBRA DA PÁGINA

Dobre a página na vertical para alinhar os dois cérebros a resolver a charada.

O que os objetos JavaScript podem fazer em seus dados?



Pesquisar tudo o que quiser, mas é improvável que encontrará algo melhor que um objeto JavaScript para fazer coisas deste tipo e analisar os dados. Eles são até capazes de tornar aleatórios os números sem nenhum problema.

Como lidar com objetos personalizados

Faça agora, pule os intermediários, há uma garantia de retorno em dinheiro, será apenas um dólar... se pedir agora. Você pode fazer do seu jeito, garota!

Se fosse tão fácil, certamente teríamos feito. O JavaScript não tem uma garantia de retorno em dinheiro, mas você pode fazer do seu jeito. Os objetos personalizados são o equivalente JavaScript de uma grande dose tripla descafeinada, muito quente, sem creme extra salpicado e sem a espuma marmorizada manchada. Esta é uma xícara de café personalizada! E com alguns objetos JavaScript personalizados, você pode planejar algum código que faz exatamente o que deseja, enquanto aproveita as vantagens das propriedades e dos métodos. O resultado final é um código reutilizável orientado a objetos que estende com eficiência a linguagem JavaScript... só para você!

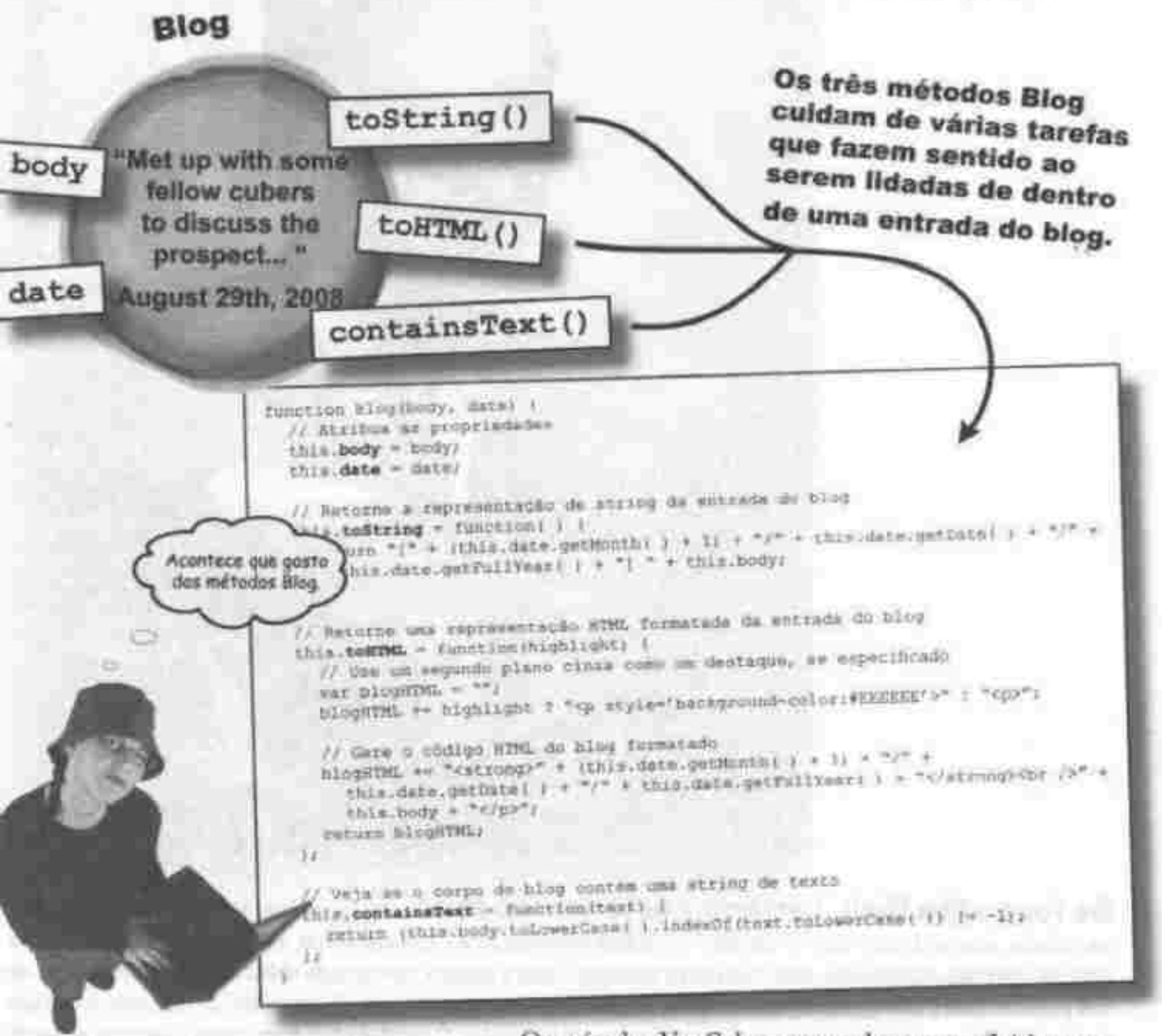
Como revisitar os métodos Blog do YouCube

Quando deixamos Ruby na última vez, ela estava bem entusiasmada por ter criado um blog acionado por objetos para escrever sobre seu interesse por quebra-cabeças de cubo. Embora Ruby tenha feito um trabalho decente ao criar o objeto `Blog` que conduz o blog YouCube, ela perdeu, sem querer, algumas oportunidades principais de aplicar os princípios orientados a objetos no YouCube. O mais importante, ela não explorou totalmente os diferentes modos do objeto `Blog` ser mais eficiente, mais organizado e, portanto, mais mantido no futuro.

O blog YouCube funciona, mas ainda não é um sinal brilhante da construção de programação orientada a objetos.

O último ajuste que Ruby fez no objeto `Blog` envolveu a criação de três métodos para lidar com várias tarefas específicas do blog.

A última versão dos arquivos pode ser baixada em www.altabooks.com.br.



Os métodos YouCube parecem bons superficialmente, mas há um problema sutil...

Sobrecarga de métodos

Parecidos com as propriedades do blog, os métodos no objeto `Blog` são criados dentro do construtor usando a palavra-chave `this`. Essa abordagem funciona, mas acaba criando uma nova cópia dos métodos para cada objeto `Blog` que é criado. Portanto, se o blog contiver seis entradas, haverá seis cópias dos três métodos `Blog`.

O objeto Blog cria, sem querer, mais métodos do que precisa, o que é desperdício e ineficiência.

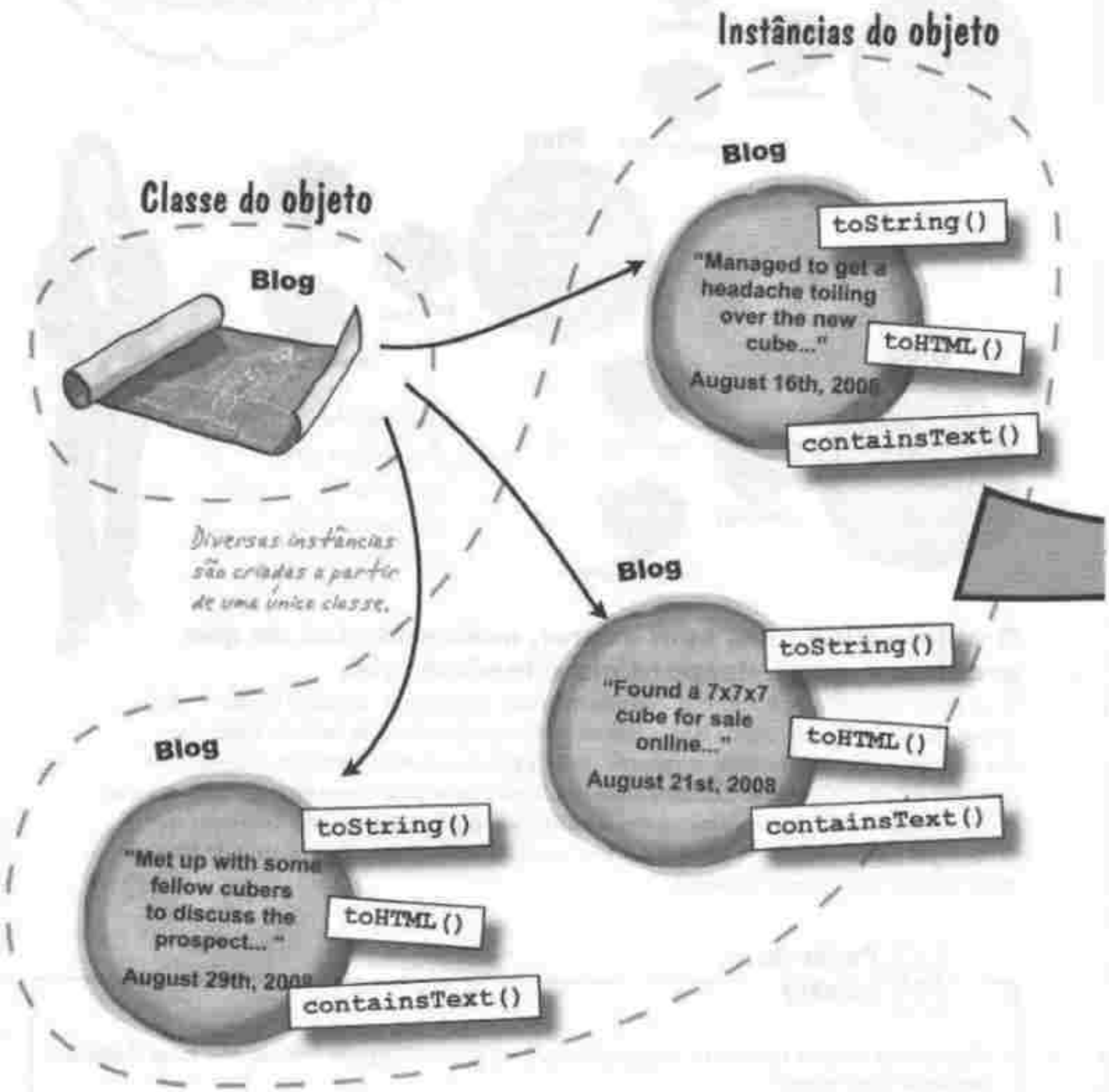
É verdade, o construtor `Blog( )` cria três métodos sempre que um novo objeto é criado, o que significa que todo objeto `Blog` tem sua própria cópia de cada método. Diferente das propriedades, que precisam armazenar dados exclusivos para cada objeto diferente, os métodos devem ser **compartilhados** entre os objetos. Seria uma construção muito melhor se todos os objetos `Blog` compartilhassem uma única cópia de cada método. Isso evita que o script fique lotado de métodos desnecessários quando muitas entradas (objetos!) do blog são adicionadas com o passar do tempo.

Poder do cérebro

Como você poderia reconstruir o objeto `Blog` para que o código do método não fosse duplicado em cada novo objeto?

Classes versus instâncias

O problema do método duplicado toca em um conceito extremamente importante relacionado aos objetos JavaScript: a diferença entre uma **classe** do objeto e uma **instância** do objeto. Uma classe é uma descrição do objeto, um modelo que resume do que é feito um objeto. Uma instância é um objeto real que foi criado a partir de uma classe. Em termos reais, uma classe é o projeto de uma casa, ao passo que um objeto é a casa em si. E semelhante aos objetos JavaScript, você pode construir muitas instâncias da casa a partir de uma única classe (projeto).



As instâncias são criadas a partir das classes

Uma classe descreve as propriedades e os métodos de um objeto, ao passo que uma instância coloca dados reais nas propriedades e dá vida a eles. Cada instância tem suas próprias cópias das propriedades, que é o que permite que as instâncias sejam diferentes com exclusividade umas das outras.

Uma classe do objeto é um modelo, ao passo que uma instância do objeto é a coisa criada a partir do modelo.

body	"Managed to get a..."
date	August 16th, 2008
toString	function() { ... }
toHTML	function() { ... }
containsText	function() { ... }

Os valores da propriedade geralmente variam entre as instâncias, portanto é importante que cada instância tenha sua própria cópia.

Propriedades

Métodos.

body	"Found a 7x7x7 cube..."
date	August 21st, 2008
toString	function() { ... }
toHTML	function() { ... }
containsText	function() { ... }

Os métodos são duplicados desnecessariamente em cada uma destas instâncias.

body	"Met up with some..."
date	August 29th, 2008
toString	function() { ... }
toHTML	function() { ... }
containsText	function() { ... }

Acesse as propriedades de uma instância com "this"

Todas as propriedades com as quais lidamos até então, eram **propriedades da instância**, significando que são possuídas por uma instância e o mais importante, cada instância tem sua própria cópia. Você pode identificar facilmente uma propriedade da instância porque ela é definida no construtor usando a palavra-chave `this`.

A palavra-chave this é usada para definir as propriedades e os métodos que são possuídos por uma instância.

```
function Blog(body, date) {
  this.body = body;
  this.date = date ;
  ...
}
```

Estas são as propriedades da instância porque são referidas usando a palavra-chave this.

Existem também métodos da instância, mas são um pouco mais capciosos, uma vez que podem ser possuídos por uma instância **ou** pela classe. Até o momento, apenas criamos métodos da instância que são definidos usando a palavra-chave `this`, que significa que são **possuídos por cada instância**. Isto explica por que o código do método é duplicado em cada instância.

```
function Blog(body, date) {
  ...
  this.toString = function() {
    ...
  }
  this.toHTML = function() {
    ...
  }
  this.containsText = function() {
    ...
  }
}
```

Cada instância de Blog tem sua própria cópia destes métodos.

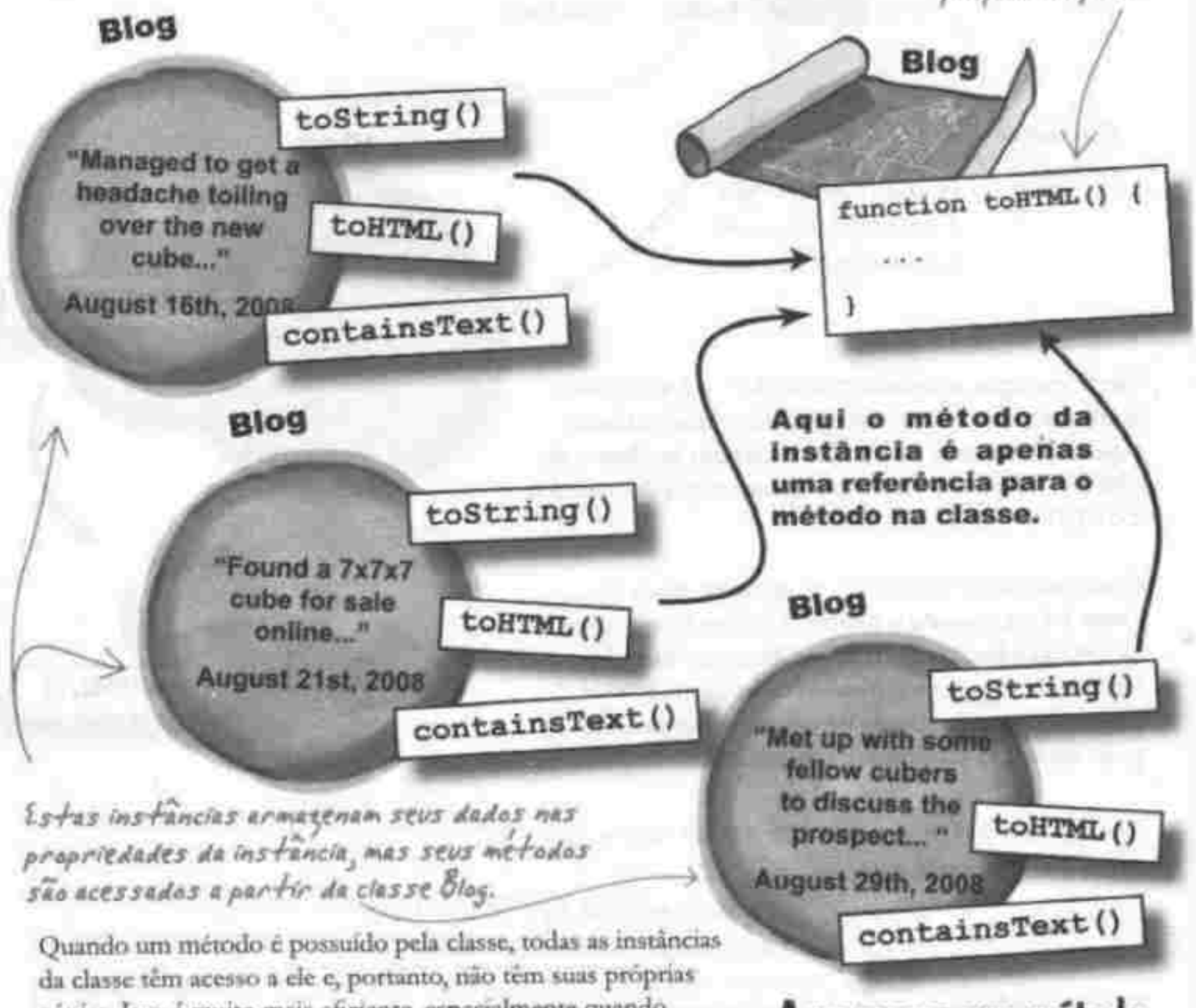
Estes são métodos da instância porque a palavra-chave this é usada para defini-los no construtor.

A boa notícia é que os objetos personalizados não são destinados a sempre desperdiçar o código do método duplicando-o em cada instância nova. A solução é criar métodos de tal modo que as instâncias compartilhem uma cópia comum do código do método.

Tenha uma vez, execute muitas: os métodos possuídos pela classe

Há outro tipo de método da instância que é possuído pela própria classe, que significa que há apenas uma cópia compartilhada por **todas** as instâncias. Essa instância **possuída pela classe** é muito mais eficiente do que armazenar uma cópia de um método em cada e toda instância.

O método é possuído pela classe Blog, portanto as instâncias não precisam de suas próprias cópias.



Estas instâncias armazenam seus dados nas propriedades da instância, mas seus métodos são acessados a partir da classe Blog.

Quando um método é possuído pela classe, todas as instâncias da classe têm acesso a ele e, portanto, não têm suas próprias cópias. Isso é muito mais eficiente, especialmente quando você considera quantas cópias do método poderia acabar ocupando espaço em um aplicativo que cria muitas instâncias do objeto. No caso do YouCube, três métodos (`toString()`, `toHTML()` e `containsText()`) seriam duplicados desnecessariamente para cada entrada do blog que é armazenada.

Naturalmente, ainda precisamos de um mecanismo para atribuir a propriedade de um método à classe, em oposição às instâncias individuais...

Armazenar um método em uma classe permite que todas as instâncias compartilhem uma cópia.

Use o protótipo para trabalhar no nível de uma classe

As classes JavaScript tornam-se possíveis no JavaScript graças a um objeto oculto chamado `prototype` que existe em cada objeto como uma propriedade. O objeto `prototype` permite definir propriedades e métodos que são possuídos no **nível da classe**, em oposição ao interior de uma instância. No caso dos métodos, o objeto `prototype` é como você estabelece que uma **classe** possui um método.

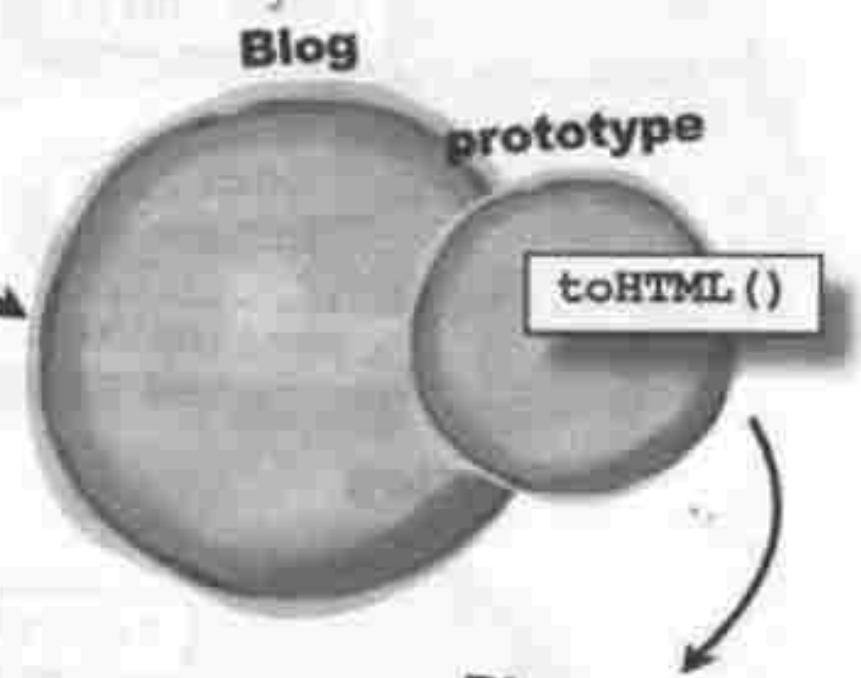
```
Blog.prototype.toHTML = function() {
  ...
}
```

O corpo da função, uma literal da função.

O nome da função.

O nome da classe.

O objeto prototype é acessado como uma propriedade da classe.



Neste exemplo, o método `toHTML( )` é adicionado à própria classe `Blog`, em oposição a uma instância específica da classe. Não importa quantas instâncias do objeto `Blog` criamos, há apenas uma cópia do método `toHTML( )`.

Como o método `toHTML( )` agora faz parte da classe `Blog`, quando o método é chamado, o código executado é localizado na classe. Porém, o método é tecnicamente ainda um método da instância porque ele pode ser chamado através de um objeto de instância e pode acessar as propriedades da instância.

```
var blogEntry1 = new Blog("Not much going on.", ...);
blogEntry1.toHTML();
```

Chamar o método toHTML() leva o código a ser executado na classe.

Se outra instância do objeto `Blog` for criada e o método `toHTML( )` for chamado nela, o mesmo código na classe será executado. Esta é a beleza de armazenar um método em uma classe – **armazene** uma vez, execute muitas!

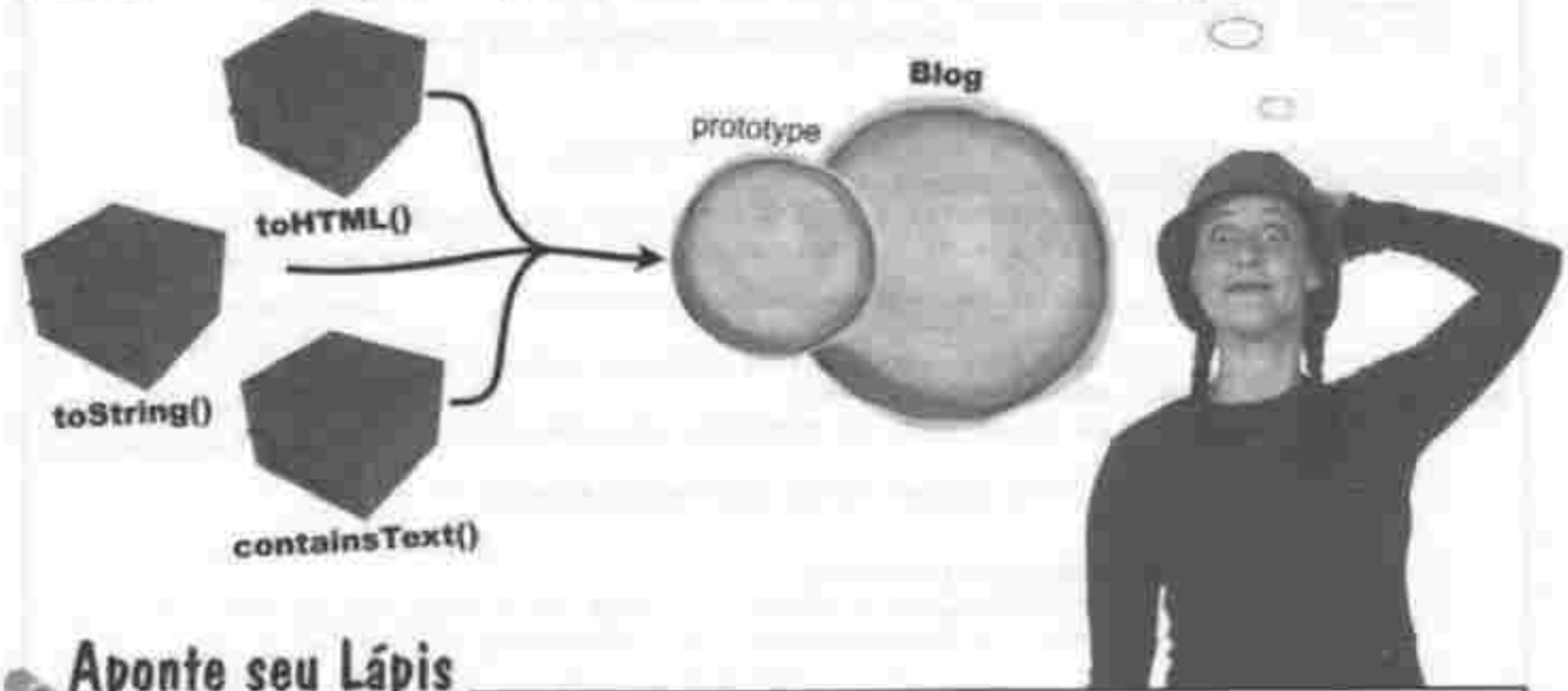
```
var blogEntry2 = new Blog("Still just hanging around.", ...);
blogEntry2.toHTML();
```

As outras instâncias ainda compartilham o mesmo método na classe.

Classes, protótipos e YouCube

Ruby está um pouco desarmada com toda essa conversa sobre classes e protótipos, mas ela tem uma sensação muito boa de que o YouCube poderia aproveitar a reconsideração do armazenamento do método `Blog` com a ajuda do objeto `prototype`.

Opa, usar o objeto prototype para armazenar os métodos do blog poderia tornar o código YouCube muito mais eficiente.



Aponte seu Lápis

O código `Blog`, agora, usa o objeto `prototype` para armazenar os métodos para que eles sejam possuídos pela classe. Anote o código e explique o que está ocorrendo.

```
function blog(body, date) {
   // Atribua as propriedades
   this.body = body;
   this.date = date;
}

// Retorne uma representação de string da entrada do blog
Blog.prototype.toString = function( ) {
   return "(" + (this.date.getMonth( ) + 1) + "/" + this.date.getDate( ) + "/" +
      this.date.getFullYear( ) + ") " + this.body;
};

// Retorne uma representação HTML formatada da entrada do blog
Blog.prototype.toHTML = function(highlight) {
   // Use um segundo plano cinza como um destaque, se especificado
   var blogHTML = "";
   blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";

   // Gere o código HTML do blog formatado
   blogHTML += "<strong>" + (this.date.getMonth( ) + 1) + "/" + this.date.getDate( ) + "/" +
      this.date.getFullYear( ) + "</strong><br />" + this.body + "</p>";
   return blogHTML;
};

// Veja se o corpo do blog contém uma string de texto
Blog.prototype.containsText = function(text) {
   return (this.body.toLowerCase( ).indexOf(text.toLowerCase( )) != -1);
};
```

Aponte seu Lápis
Solução

O código `Blog`, agora, usa o objeto `prototype` para armazenar os métodos para que eles sejam possuídos pela classe. Anote o código e explique o que está ocorrendo.

```
function Blog(body, date) {
   // Atribua as propriedades
   this.body = body;
   this.date = date;
}

// Retorne uma representação de string da entrada do blog
Blog.prototype.toString = function( ) {
   return "[" + (this.date.getMonth( ) + 1) + "/" + this.date.getDate( ) + "/" +
      this.date.getFullYear( ) + "] " + this.body;
};

// Retorne uma representação HTML formatada da entrada do blog
Blog.prototype.toHTML = function(highlight) {
   // Use um segundo plano cinza como um destaque, se especificado
   var blogHTML = "";
   blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";

   // Gere o código HTML do blog formatado
   blogHTML += "<strong>" + (this.date.getMonth( ) + 1) + "/" + this.date.getDate( ) + "/" +
      this.date.getFullYear( ) + "</strong><br />" + this.body + "</p>";
   return blogHTML;
};

// Veja se o corpo do blog contém uma string de texto
Blog.prototype.containsText = function(text) {
   return (this.body.toLowerCase( ).indexOf(text.toLowerCase( )) != -1);
};
```

Agora, o construtor concentra-se somente na criação e na inicialização das propriedades.

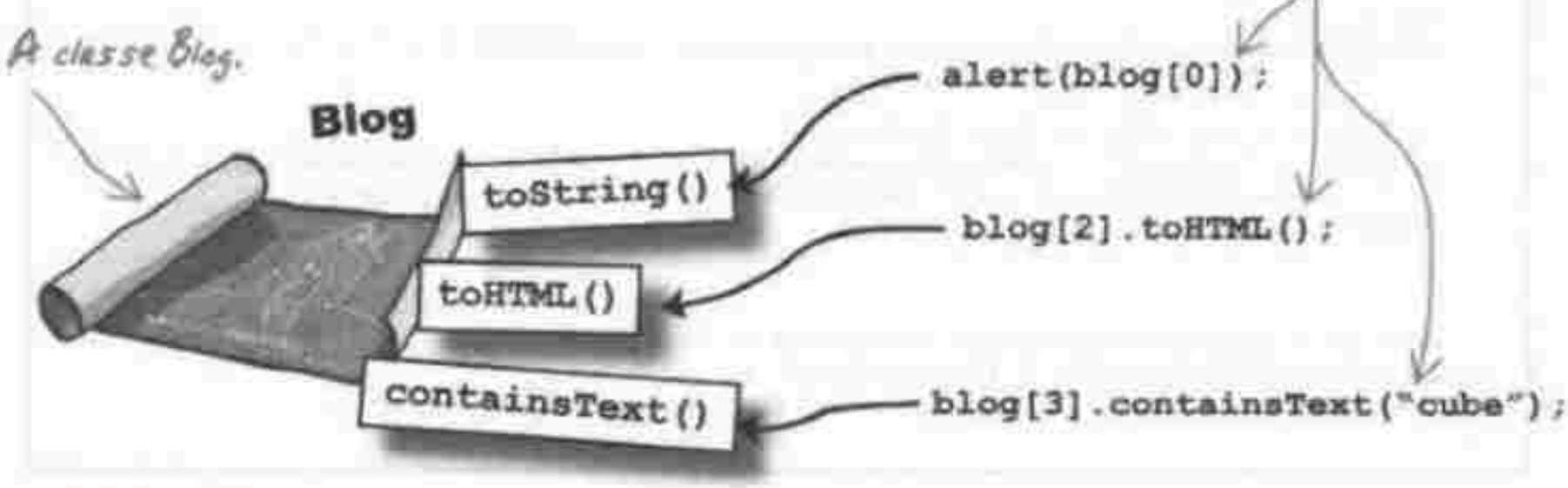
Como os métodos não estão sendo atribuídos a uma determinada instância Blog, a atribuição ocorre fora do construtor.

Cada método é definido no objeto prototype, ao invés de usar a palavra-chave this no construtor Blog().

Um YouCube mais eficiente

Agora, o blog YouCube usa os **métodos possuídos pela classe** para eliminar o código desperdiçado, graças ao protótipo do objeto. Não importa quantas instâncias `Blog` são criadas, apenas uma cópia dos métodos é criada, uma vez que agora eles residem na classe. O legal é que da perspectiva do script YouCube de usar o objeto `Blog`, nada mudou.

As instâncias do blog chamando os métodos na classe.



Pontos de Bala

- Uma classe é uma **descrição** de um objeto, ao passo que uma instância é um objeto **real** que foi criado com base nessa descrição.
- Uma classe organiza as propriedades e os métodos de um objeto, ao passo que uma instância coloca dados reais nas propriedades, dando aos métodos algo com o qual trabalhar.
- A palavra-chave `this` é usada para acessar uma instância de dentro de seu **próprio código**.
- O objeto `prototype` permite que os métodos sejam armazenados em uma **classe**, evitando que as instâncias dupliquem o código desnecessariamente.

Não existem Perguntas Idiotas

P: Ainda não compreendo muito bem toda a situação das classes versus instâncias. Qual é o caso?

R: A idéia por detrás de uma classe é facilitar a criação e a reutilização dos objetos. Você poderia criar objetos únicos como literais do objeto o dia inteiro e nunca ter problemas, exceto gastar muita energia desnecessariamente. É um desperdício porque você estaria duplicando seus esforços. Como um arquiteto que insiste em redesenhar os projetos da mesma casa sempre que deseja construí-la de novo.

Por que não criar um modelo que possa ser usado para criar quantas instâncias quiser, resultando em muito menos esforço? É onde entram as classes – crie uma classe e, então, use-a para criar quantas instâncias forem necessárias.

P: Tudo bem, então as classes são para facilitar criar os objetos que são parecidos entre si. Mas qual a relação de `this` e `prototype` com isso?

R: A palavra-chave `this` permite acessar uma instância de dentro de um de seus próprios métodos. Seu uso primário é acessar as propriedades da instância. Portanto, se você quiser acessar uma propriedade chamada x de dentro de um método, informará `this.x`. Se quisesse informar apenas x, o código não saberia que estava tentando acessar uma propriedade da instância; apenas pensaria que x era uma variável. É por isso que os construtores requerem que você use `this` ao criar e inicializar as propriedades.

`prototype` é algo diferente. Ele fornece o mecanismo para criar as classes. Diferente de algumas outras linguagens de programação como, por exemplo, C++ e Java, o JavaScript não suporta realmente as classes como uma parte concreta da linguagem. Ao contrário, o JavaScript usa protótipos para simular as classes. O resultado final é parecido, mas o JavaScript requer que você crie as "classes" manipulando o objeto `prototype`, que aparece como uma propriedade "oculta" de cada objeto JavaScript. Armazenando uma propriedade ou um método no objeto `prototype`, você torna-o acessível, com eficiência, como parte de uma classe, em oposição a apenas parte de uma instância.

P: Como os construtores encaixam-se na equação da classe?

R: Os construtores são uma parte muito importante do estabelecimento das classes JavaScript porque são responsáveis por criar as instâncias do objeto. Você pode considerar um construtor e um protótipo como representando as duas peças maiores do quebra-cabeça da classe JavaScript. Os construtores cuidam da configuração de tudo para as instâncias, enquanto que os protótipos lidam com tudo no nível da classe. Ambas as entidades trabalhando em conjunto fornecem a capacidade de fazer algumas coisas muito legais porque existem razões atraentes para posicionar alguns membros no nível da instância e outros no nível da classe. Continuaremos a explorar esta questão neste capítulo.

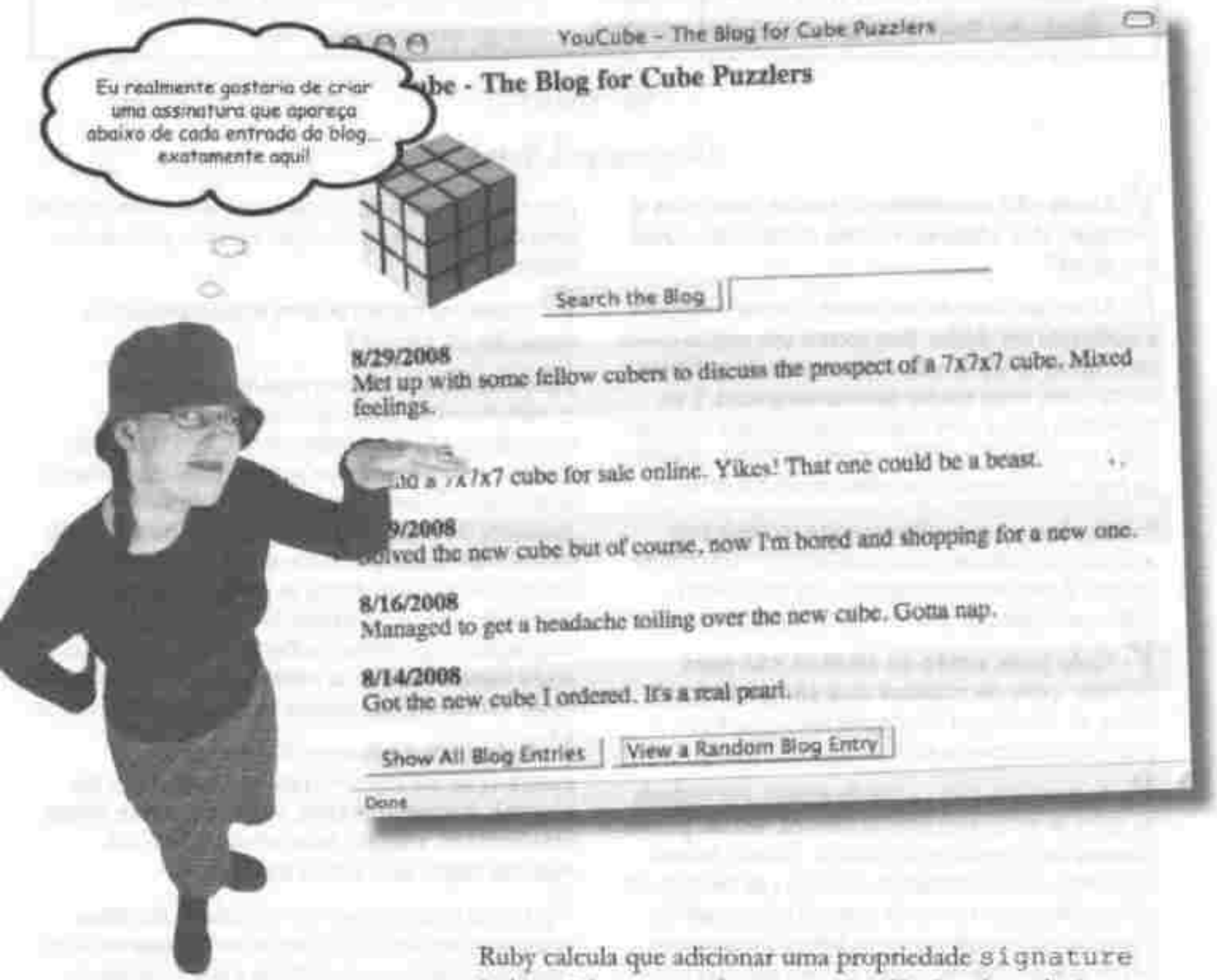
P: Estou um pouco confuso com a convenção de nomenclatura usada com os objetos. Algumas vezes, um objeto tem letras maiúsculas, outras, tem minúsculas. Há alguma regra que estou perdendo?

R: A única regra é que os nomes da classe têm letras maiúsculas, enquanto que os nomes da instância têm letras minúsculas. É porque uma instância é realmente apenas uma variável e os identificadores da variável usam letras minúsculas. A inconsistência tem relação, principalmente, com o fato de que estamos usando o termo "objeto" bem vagamente. Para ser preciso, porém, as classes, como `Blog`, devem ter letras maiúsculas, ao passo que as instâncias como `blogEntry` ou `blog[0]` devem ter letras minúsculas CamelCase.

Essa convenção de nomenclatura fará sentido se você lembrar dos outros objetos padrões que usamos. Você poderia armazenar a data/hora atuais em uma variável (instância) chamada `now`, que é criada a partir do objeto (classe) `Date`.

Como assinar o blog

Ruby está trabalhando na eficiência e nas melhorias organizacionais que a programação orientada a objetos ou OOP trouxe para o YouCube, mas ela está interessada em fazer mais do que apenas uma melhoria interna no código... ela deseja adicionar um recurso novo.



Ruby calcula que adicionar uma propriedade `signature` à classe `Blog` possa fazer o serviço. Então, ela poderá simplesmente definir a propriedade no construtor e exibi-la com cada entrada do blog... problema resolvido!

Poder do cérebro

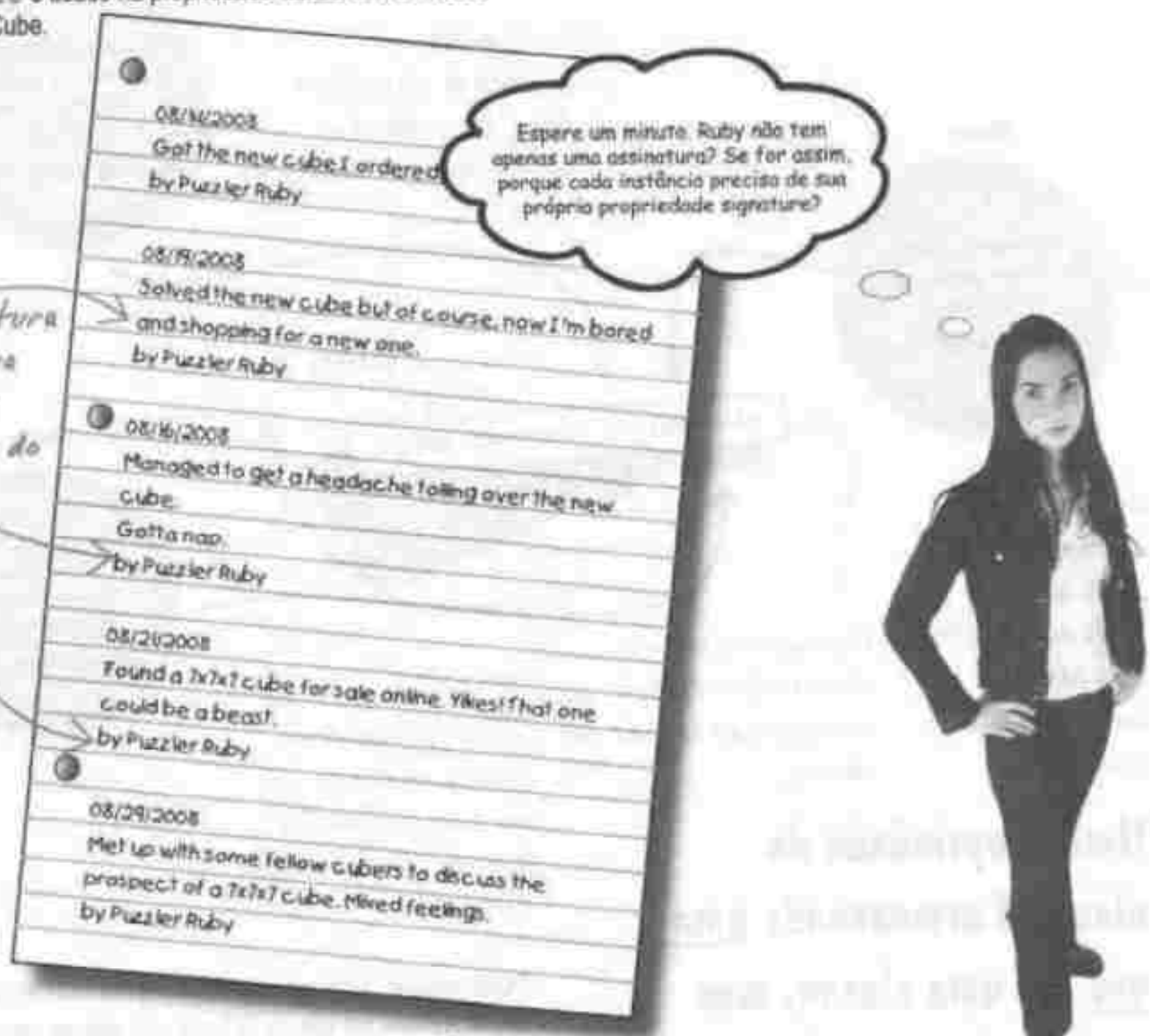
Ruby deve criar a assinatura como uma propriedade da instância? Você pode imaginar alguma razão para isso não ser uma boa idéia?

Não existem Perguntas Idiotas

P: Continuo vendo o termo orientado a objetos. O que significa?

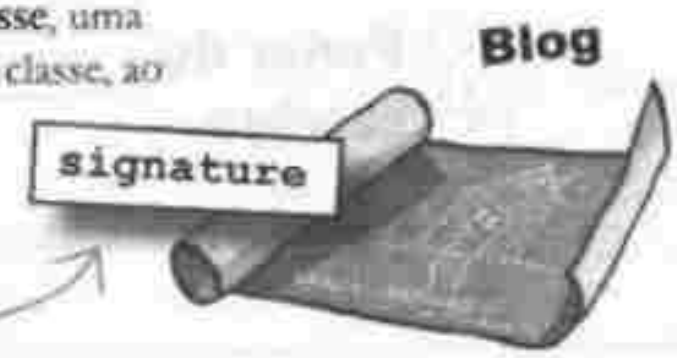
R: O termo **orientado a objetos** é muito usado (e abusado), terrivelmente, nos círculos da programação e pode significar coisas diferentes para pessoas diferentes. Em geral, a programação orientada a objetos (OOP) envolve construir um software dos objetos como, por exemplo, o modo como o objeto `Date` é usado na propriedade `date` das entradas do blog YouCube.

A maioria dos programadores associa a OOP com o uso dos objetos em todo um programa. Pelo menos na teoria, o verdadeiro programa orientado a objetos pode ser dividido em uma coleção de objetos que interagem entre si. Existem puristas orientados a objetos que argumentarão que o JavaScript não se qualifica como uma linguagem OOP. Economize energia e tente evitar esse debate. Existem argumentos válidos em ambos os lados, mas no final do dia, ninguém vence.



Talvez uma assinatura seja suficiente.

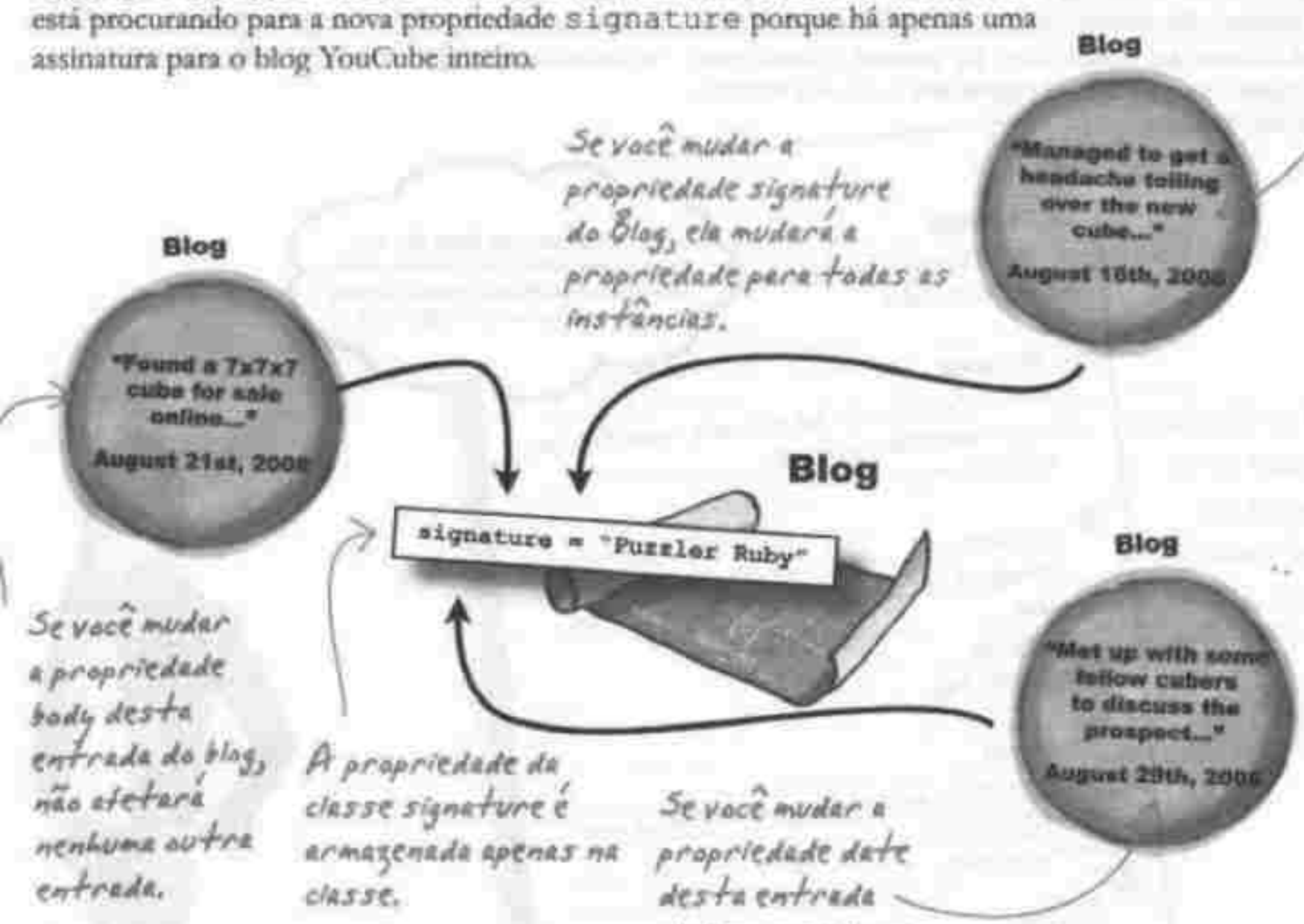
Sabendo que a assinatura do blog é igual para todas as instâncias, não há nenhuma necessidade de cada instância ter sua própria propriedade signature. O que Ruby precisa é de uma **propriedade da classe**, uma propriedade onde há apenas uma cópia que é armazenada na classe, ao invés de uma cópia diferente em cada instância individual.



A propriedade signature deve ser armazenada na classe Blog, não em cada instância do blog.

As propriedades da classe são compartilhadas também

As **propriedades da classe** são muito parecidas com os métodos de instância possuídos pela classe no sentido de que eles são possuídos pela classe com uma **única cópia** disponível para todas as instâncias. Em algumas maneiras, isso é mais importante para os dados, pois significa que a propriedade tem apenas **um valor** que é compartilhado por todas as instâncias. Isso é exatamente o que Ruby está procurando para a nova propriedade `signature` porque há apenas uma assinatura para o blog YouCube inteiro.



Uma propriedade da classe é armazenada <u>uma vez</u> em uma classe, mas acessível para todas as instâncias.

Mesmo que a propriedade `signature` seja armazenada na classe `Blog`, ela estará prontamente acessível para qualquer instância que deseja acessar a assinatura do autor do blog.

Poder do cérebro

Como você acha que poderia criar uma propriedade da classe?

Como criar as propriedades da classe com prototype

Para toda a conversa sobre onde uma propriedade da classe é armazenada e o impacto geral que tem na vida como sabemos, criar uma propriedade da classe é surpreendentemente comum. Na verdade, uma única linha de código é requerida:

A propriedade da classe é acessada usando a notação (ponto) do objeto.

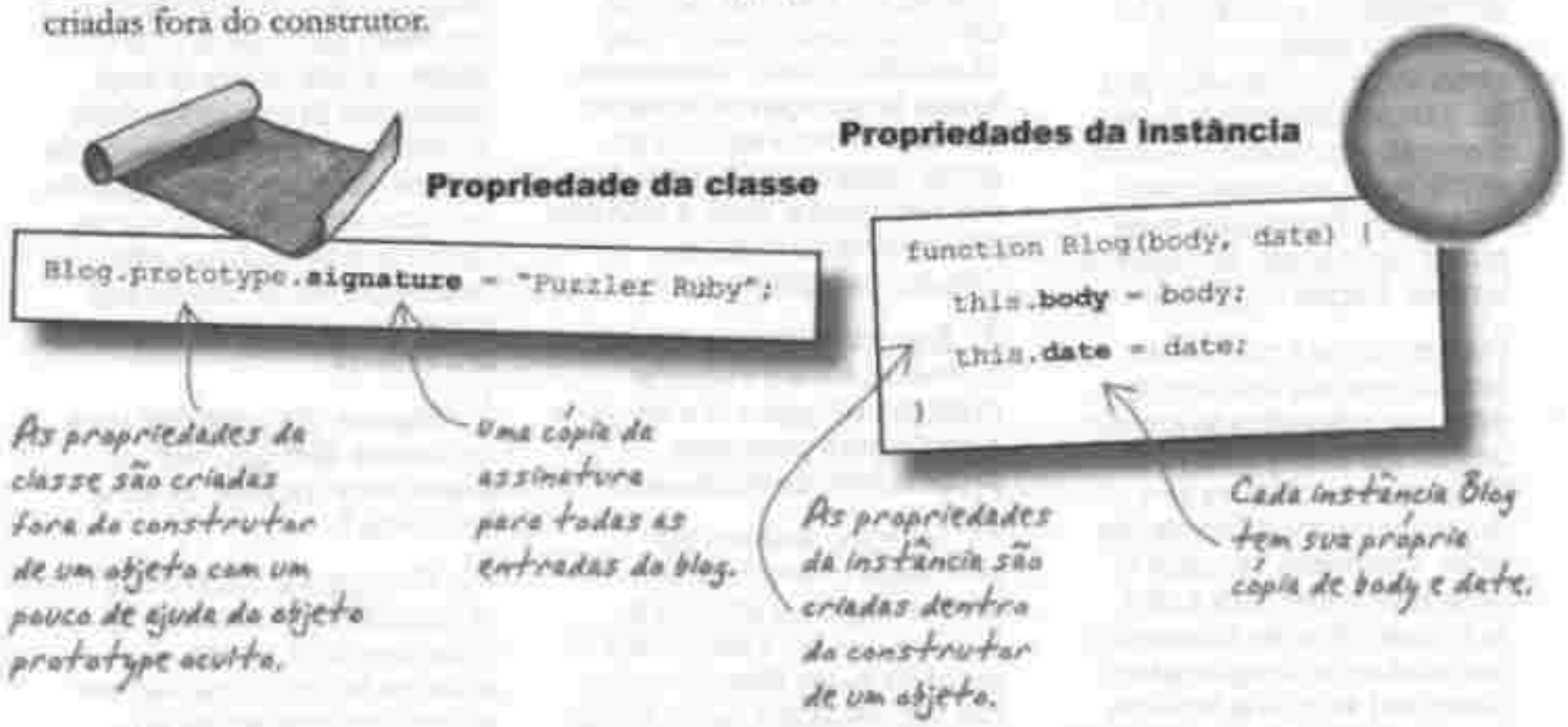
A classe Blog é especificada primeira, então o objeto prototype.

```
Blog.prototype.signature = "Puzzler Ruby";
```

Provavelmente você adivinhou, o objeto prototype é onde as propriedades da classe são armazenadas.

As propriedades da classe não têm que ser inicializadas, mas neste caso faz sentido porque a autora do blog já é conhecida.

Uma das coisas mais interessantes sobre este código é o que você não pode perceber totalmente vendo-o por si só – o código não aparece dentro de um construtor, como o código que cria as propriedades da instância. A razão é que os construtores são usados para dar vida às instâncias e, portanto, não são capazes de criar as propriedades da classe. Ao contrário, as propriedades da classe devem ser criadas fora do construtor.



Aponte seu Lápis

Escreva um código para exibir a propriedade `signature` em uma caixa de alerta. Sugestão: Suponha que o código esteja localizado dentro de um método `Blog`.

Aponte seu Lápis
Solução

Escreva um código para exibir a propriedade `signature` em uma caixa de alerta. Sugestão: Suponha que o código esteja localizado dentro de um método `Blog`.

```
alert(this.signature);
```

As propriedades da classe são acessadas como as propriedades da instância, usando a palavra-chave this!

Não existem Perguntas Idiotas

P: Por que você precisa armazenar a assinatura YouCube em uma propriedade? Ela não poderia ser fornecida como parte do texto do corpo para cada entrada?

R: Certamente, é possível incluir uma assinatura em cada entrada do blog como parte do texto do corpo, mas isso requer tempo e esforço desnecessários, supondo que há apenas uma pessoa enviando para o blog. Ficaria cansativo para Ruby assinar cada entrada do blog quando há uma maneira mais clara de lidar com a assinatura usando o JavaScript. E quem sabe se ela poderia digitar errado, sem querer, e tornar-se "Puzzled Ruby"? Não é bom!

Outra opção que é mais viável é simplesmente usar uma literal de string para a assinatura ao formatar uma entrada do blog como HTML. Essa abordagem funciona bem, mas esconde uma parte importante dos dados, a assinatura, no código de formatação do blog, onde é difícil de encontrar e manter. Colocando a assinatura em uma propriedade da classe, você irá torná-la facilmente acessível e, portanto, muito mais fácil para o blogger identificar e alterar.

P: Como mudaria criar uma entrada do blog se a assinatura fosse uma propriedade da instância?

R: Lembre-se de que cada instância de um objeto tem seu próprio conjunto de propriedades da instância que são inicializadas no construtor. Se a propriedade da assinatura fosse uma propriedade da instância, o construtor `Blog( )` precisaria defini-la em cada e toda instância. Isso não seria necessariamente uma grande confusão na codificação, uma vez que o construtor poderia definir a propriedade para a string de assinatura. Porém, internamente, haveria tantas cópias da assinatura quanto instâncias e isso seria um grande desperdício. Não apenas isso, mas seria possível mudar a assinatura para as diferentes instâncias independentemente das outras.

P: E se eu quisesse modificar o YouCube para suportar diversos bloggers, eu mudaria a assinatura para uma propriedade da instância?

R: Sim e isso seria uma boa idéia porque em uma situação com diversos bloggers, a propriedade `signature` tem a possibilidade de **precisar** manter diferentes valores para cada instância. A melhor maneira de lidar com isso provavelmente seria adicionar um argumento ao construtor `Blog( )` que permite que a string da assinatura seja transmitida. Então, use essa string para inicializar a propriedade da instância `signature`. Em outras palavras, lide com a propriedade `signature` exatamente como as outras propriedades da instância `Blog`.

P: As propriedades da classe parecem funcionar como as variáveis globais. Como são diferentes?

R: As propriedades da classe são, de fato, muito parecidas com as variáveis globais, uma vez que podem ser acessadas de praticamente qualquer lugar. As propriedades da classe também são criadas de modo semelhante às variáveis globais, pelo menos em termos de onde são criadas – a partir do nível do script principal, fora de outro código. Onde as propriedades da classe diferem das variáveis globais é em sua associação com uma classe e, portanto, com os objetos da instância. Isso significa que você sempre tem que acessar uma propriedade da classe em relação a uma instância.

P: Espere. As propriedades da classe *têm* que ser acessadas através de uma instância?

R: Mesmo que as propriedades da classe sejam criadas usando o objeto `prototype`, que os armazena em uma classe, elas têm que ser acessadas através das instâncias. Portanto, uma propriedade da classe é acessada exatamente como uma propriedade da instância, usando a palavra-chave `this` e a notação (ponto) do objeto. A diferença é realmente apenas onde a propriedade é armazenada – na classe (propriedade da classe) ou em uma instância específica (propriedade da instância).

Assinado e enviado

Com a propriedade da classe `signature` criada, inicializada e pronta para usar, Ruby está pronta para vê-la em ação. Vendo o código que formata uma entrada do blog para exibir em um navegador, o método `toHTML( )` é onde a assinatura entra na apresentação de cada entrada do blog.

O método toHTML( ), agora, formata a assinatura como parte de uma entrada do blog.

```
Blog.prototype.toHTML = function(highlight) {
  // Dar um segundo plano cinza como um destaque, se especificado
  var blogHTML = "";
  blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";

// Gere o código HTML do blog formatado
blogHTML += "<strong>" + (this.date.getMonth( ) + 1) + "/" + this.date.getDate( ) + "/" +
  this.date.getFullYear( ) + "</strong><br />" + this.body + "<br /><em>" + this.signature +
  "</em></p>";
return blogHTML;
}
```

A propriedade da classe signature é referida exatamente como uma propriedade da instância normal.

A assinatura de Ruby aparece como parte de cada entrada do blog.

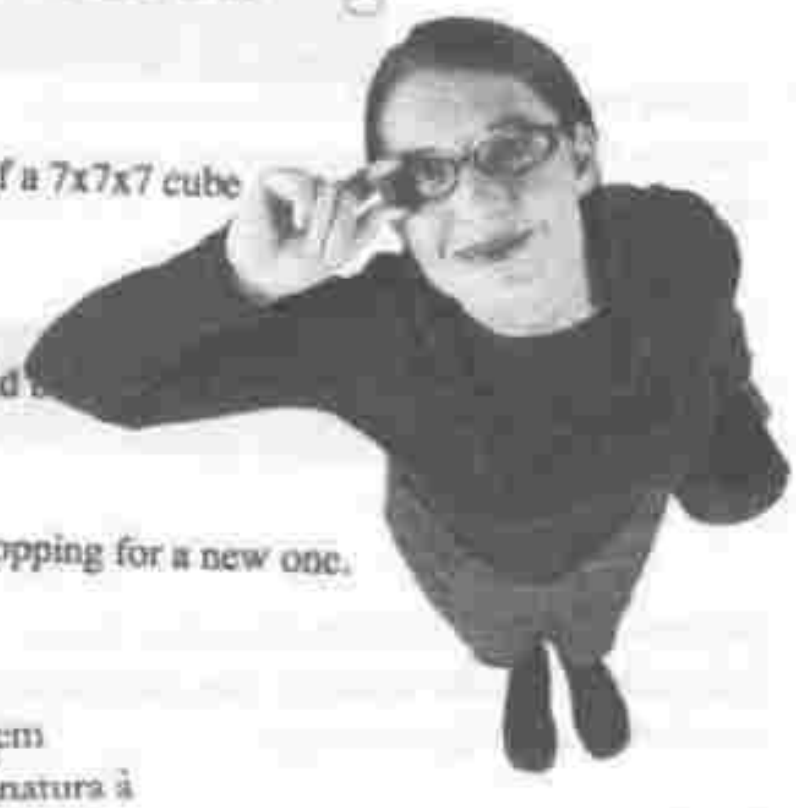
Agora, não há nenhum mistério sobre quem está escrevendo cada entrada do blog.

9/1/2008
Went ahead and ordered the scary 7x7x7 cube. Starting a mental exercise regimen to prepare.
by Puzzler Ruby

8/29/2008
Met up with some fellow cubers to discuss the prospect of a 7x7x7 cube feelings.
by Puzzler Ruby

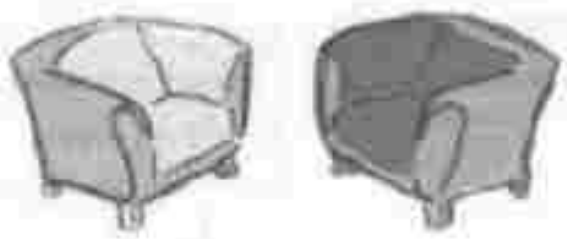
8/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could
by Puzzler Ruby

8/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.
by Puzzler Ruby



Ruby usou as técnicas OOP para estender mais a linguagem JavaScript adicionando uma propriedade da classe de assinatura à classe `Blog`. O mais importante, ela conseguiu acrescentar um toque mais pessoal ao blog YouCube.

Conversa informal



Conversa desta noite: As propriedades da instância e da classe conversam sobre a propriedade dos dados e as sociedades secretas

Propriedade da instância:

Então, você é o outro cara sobre quem ouvi falar. Tenho que dizer que não vejo por que você está aqui. Faço um trabalho excelente ao permitir que as instâncias do objeto sejam únicas e controlem seus próprios valores da propriedade.

Agora, isso é difícil de acreditar. Continue...

Então, você está dizendo que eu não seria melhor maneira de armazenar um cumprimento secreto?

Compreendo. Mas e a senha secreta? Posso armazenar isso?

Impressionante! Então, comecemos. Estou iniciando um clube secreto e nós duas estamos tendo nossas próprias senhas.

Boa! Então, o que é? Realmente. Estou curiosa...

Propriedade da classe:

Tenho certeza que sim e é algo admirável. Mas você sabia que algumas vezes as instâncias não desejam a chatice de acompanhar seus próprios dados?

Bem, há situações em que uma parte dos dados é comum a todas as instâncias, um tipo de clube secreto que tem um cumprimento secreto. Cada pessoa no clube conhece o cumprimento secreto, mas é um cumprimento de todo o clube. Se alguma moça no clube inventar seu próprio cumprimento secreto, irá alterar tudo. Então, alguma outra moça terá que cobrir isso com seu cumprimento e antes que você perceba, ninguém mais saberá o cumprimento secreto porque há muitos deles.

É isso aí. Sem ofensas, mas neste caso, os membros do clube só precisam de um cumprimento, mesmo que todas as pessoas precisam conhecê-lo.

Talvez. Se cada pessoa tiver sua própria senha secreta que seja pessoal, então sim, seria excelente para você armazenar as senhas secretas.

Mas você não conhece o cumprimento secreto... te peguei!

O código duplicado é inaceitável

Ruby está nisso de novo. Como ninguém descansa sobre os louros, ela decidiu ir além ao melhorar a eficiência do código YouCube. Ela descobriu um código de formatação da data duplicado que está convencida de que pode ser eliminado de algum modo com uma aplicação esperta dos princípios OOP.

Este código parece estar duplicado desnecessariamente. Como posso retirá-lo?

```
<html>
  <head>
    <title>YouCube - The Blog for Cube Puzzlers</title>

    <script type="text/javascript">
      // Construtor de objetos do blog
      function Blog(body, date) {
        // Atribua as propriedades
        this.body = body || "Nothing going on today.";
        this.date = date || new Date( );
      }

      // Retorne uma representação de string da entrada do blog
      Blog.prototype.toString = function( ) {
        return m/d + (this.date.getMonth( ) + 1) + "/" + this.date.getDate( ) + "/" +
        this.date.getFullYear( ) + "] " + this.body;
      }

      // Retorne uma representação HTML formatada da entrada do blog
      Blog.prototype.toHTML = function(highlight) {
        // Use um segundo plano cinza como um destaque, se especificado
        var blogHTML = "";
        blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";

        // Gere o código HTML do blog formatado
        blogHTML += "<strong>" + (this.date.getMonth( ) + 1) + "/" + this.date.getDate( ) + "/" +
        this.date.getFullYear( ) + "</strong><br />" + this.body + "<br />  " + this.signature
        +
        "</em></p>";
        return blogHTML;
      }

      // Veja se o corpo do blog contém uma string de texto
      Blog.prototype.containsText = function(text) {
        return (this.body.indexOf(text) != -1);
      }

      // Define a assinatura de todo o blog
      Blog.prototype.signature = "by Puzzler Ruby";
```

Este código de formatação da data é idêntico e, portanto, um pequeno desperdício.

Poder do cérebro

Como você poderia eliminar o código de formatação da data duplicado no YouCube?

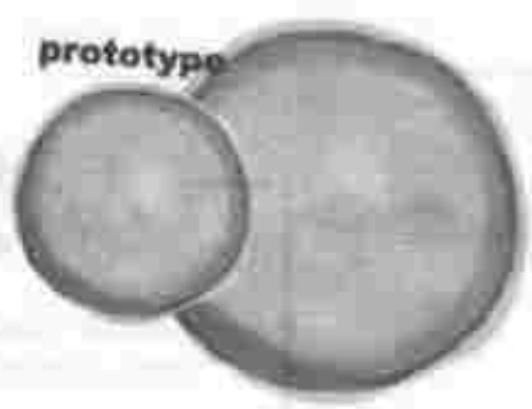
Um método de formatação de datas

Ruby acha que uma solução apropriada para o código de formatação da data duplicado é adicionar um método de formatação da data ao objeto `Blog`. Para reutilizar o código, ele deve ser colocado em uma função ou em um método e ela poderia também combinar com um método, uma vez que o objeto `Blog` é responsável por formatar as datas como parte da formatação de uma entrada do blog. Deveria?

Se formatar uma data é realmente um comportamento do objeto Date, faria mais sentido para o método ser um método Date? Há alguma maneira de adicionar um método a um objeto JavaScript padrão?

De volta ao objeto prototype

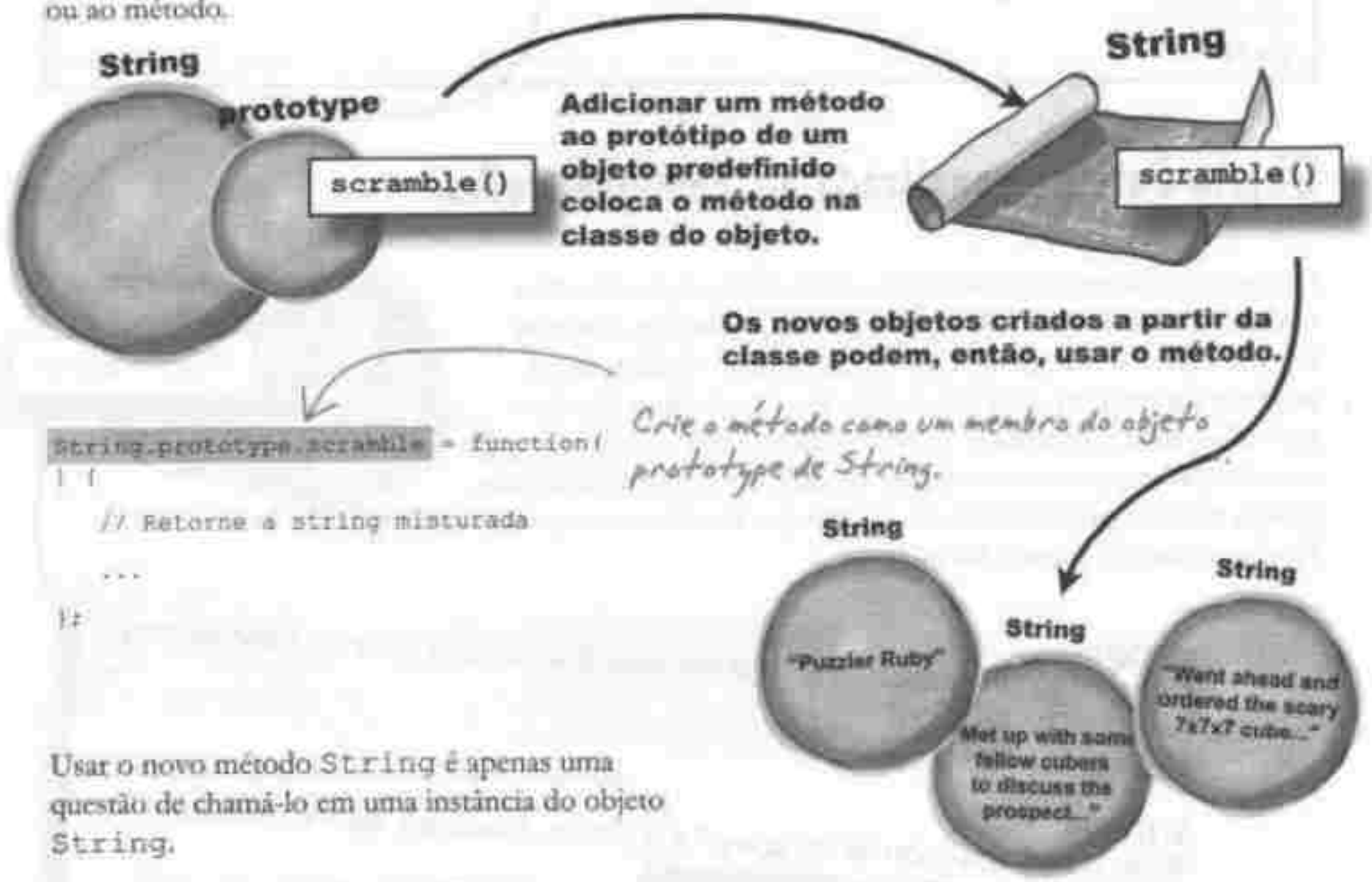
O que poderia ser mais poderoso do que pegar um objeto preexistente e torná-lo ainda melhor? Como resultado, há um modo de modificar um objeto padrão e acaba fornecendo a opção perfeita em termos de estender a linguagem JavaScript. O segredo para estender os objetos padrões ou qualquer objeto JavaScript para tanto, é o objeto `prototype`. Já usamos o objeto `prototype` para estender a classe `Blog` com as propriedades da classe e os métodos possuídos pela classe. Nada nos impede de fazer o mesmo tipo de extensão para as classes JavaScript predefinidas.



Como estender os objetos padrões

O segredo para estender um objeto está no objeto `prototype` e todo objeto no JavaScript tem um. Portanto, você pode estender qualquer objeto adicionando propriedades e métodos a seu objeto `prototype`, que estabelece as propriedades da classe e os métodos possuídos pela classe. No caso de um objeto JavaScript predefinido, adicionar uma propriedade ou um método ao seu objeto `prototype` significa que qualquer instância nova do objeto predefinido terá acesso à propriedade ou ao método.

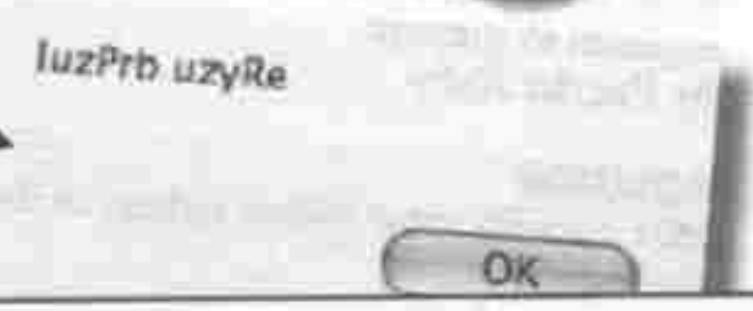
O objeto prototype permite estender os objetos JavaScript predefinidos.




```
String.prototype.scramble = function(
) {
  // Retorne a string misturada
  ...
};
```

Usar o novo método `String` é apenas uma questão de chamá-lo em uma instância do objeto `String`.

```
alert(this.signature.scramble());
```



Aponte seu Lápis

Escreva o código para um método chamado `shortFormat()` que seja uma extensão para o objeto `Date` padrão e cujo trabalho seja formatar uma data como *MM/DD/AAAA*.

.....

.....

.....

Aponte seu Lápis
Solução

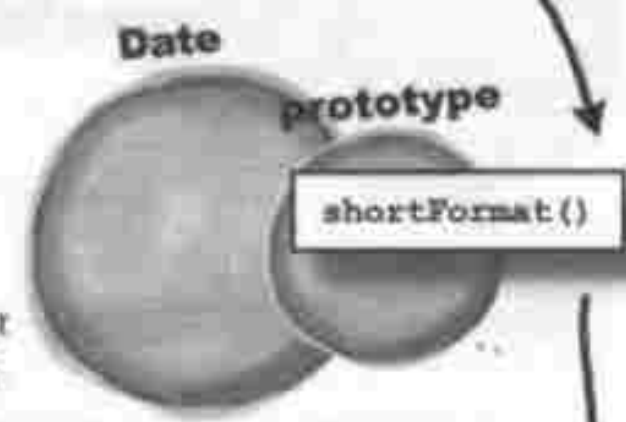
Escreva o código para um método chamado `shortFormat()` que seja uma extensão para o objeto `Date` padrão e cujo trabalho seja formatar uma data como *MM/DD/AAAA*.

O método é adicionado ao protótipo do objeto Date.

```
Date.prototype.shortFormat = function() {
  return (this.getMonth() + 1) + "/" + this.getDate() + "/" + this.getFullYear();
};
```

Objeto Date personalizado = YouCube melhor

O objeto `Date` personalizado torna o YouCube mais eficiente e estende os recursos do objeto predefinido. O YouCube também fica mais fácil de manter uma vez que o formato da data, agora, pode ser alterado em um local, porém afeta a aparência das datas em todo o blog. Certamente, as melhorias OOP no código do script nem sempre apresentam um toque imediato na forma de mudanças visuais, mas em geral resultam no código que simplesmente parece funcionar melhor a longa distância.



9/3/2008
Attended a rally outside of a local toy store that stopped carrying cube puzzles. Power to the puzzlers!
by Puzzler Ruby

9/1/2008
Went ahead and ordered the scary 7x7x7 cube. Starting a mental exercise regimen to prepare.
by Puzzler Ruby

8/29/2008
Met up with some fellow cubers to discuss the prospect of a 7x7x7 cube. Mixed feelings.
by Puzzler Ruby

As datas do blog agora estão formatadas usando um método personalizado do objeto Date.

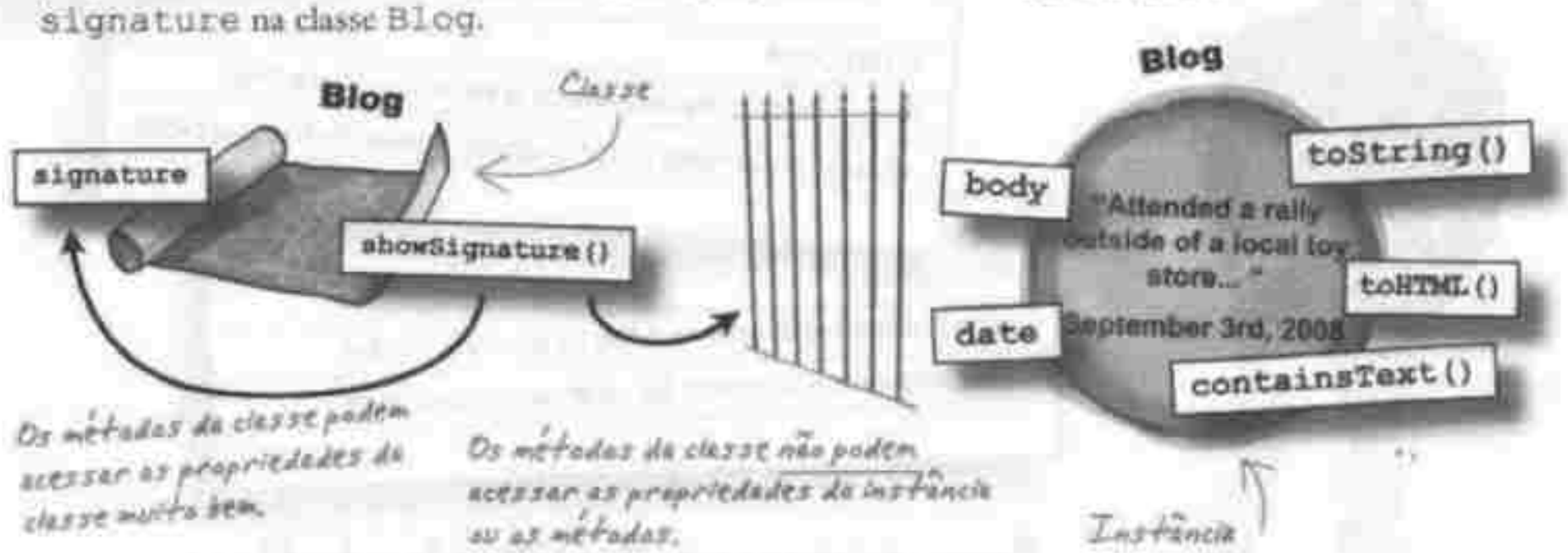
8/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could be a beast.
by Puzzler Ruby

8/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.
by Puzzler Ruby

Uma classe pode ter seu próprio método

O método `shortFormat( )` personalizado do objeto `Date` é um **método da instância possuído pela classe**, significando que tem acesso às propriedades da instância, mesmo que seja possuído pela classe. É isso que permite ao método formatar a data armazenada em uma determinada instância. Também é possível criar um **método da classe**, que é um método possuído por uma classe que **não pode** acessar as propriedades da instância. Porém, os métodos da classe podem acessar as propriedades da classe como, por exemplo, a propriedade `signature` na classe `Blog`.

Os métodos da classe são possuídos por uma classe e podem acessar apenas as propriedades da classe.



Criar um método da classe é uma questão de definir o método para a classe **sem** usar o objeto `prototype` – apenas atribua o método à classe usando o nome da classe e a notação do objeto.

```
Blog.showSignature = function() {
    alert("This blog created by " + Blog.prototype.signature + ".");
};
```

Para acessar uma propriedade da classe a partir de um método da classe, você tem que extrair a propriedade prototype.

Como a assinatura é uma propriedade da classe, o método da classe tem acesso a ela.

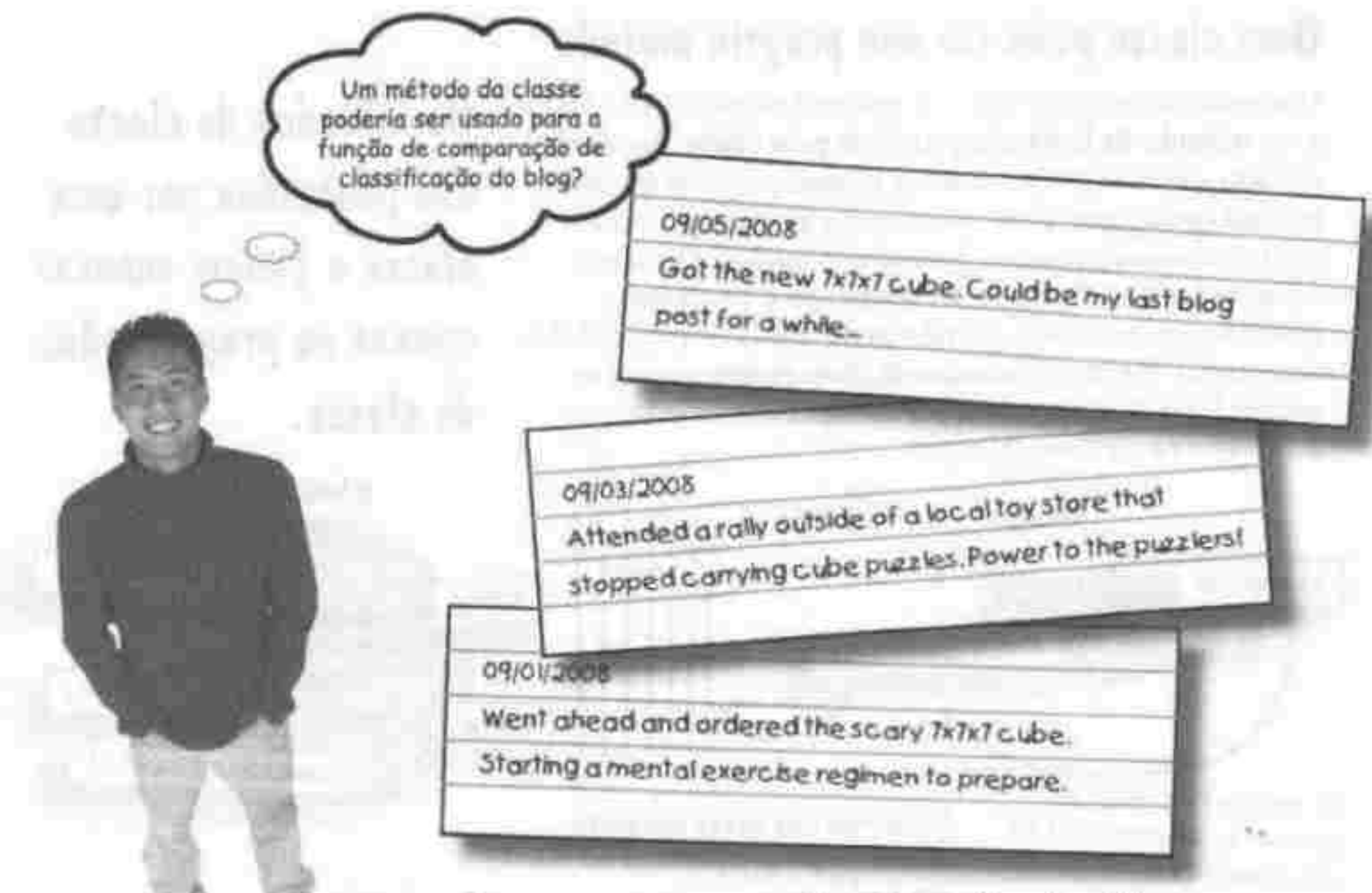
Como os métodos da classe não têm nenhuma associação a uma instância, você os chama fazendo uma referência apenas ao nome da classe. Isso não significa que uma instância não pode chamar um método da classe, apenas tem que fazer isso usando o nome da classe.

```
Blog.showSignature();
```

O nome da classe é a chave para chamar um método da classe.

Poder do cérebro

Você pode pensar em algum código no YouCube que faria sentido como um método da classe de `Blog`?



Como repensar o classificador de blogs

Esta é uma idéia intrigante porque a função de comparação da classificação está, definitivamente, desempenhando um papel que é específico do objeto `Blog`. Atualmente, essa função é criada como uma literal da função dentro da função `showBlog( )`, que é onde é necessária.

A classificação do blog é lidada dentro da função showBlog( ), que não é uma parte do objeto Blog.

```
function showBlog(numEntries) {
  // Primeiro, classifique o blog na ordem cronológica inversa (o mais recente primeiro)
  blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });
  ...
}
```

Poderia ser possível mover o código de comparação da classificação para um método da classe.

Um dos conceitos fundamentais da OOP é **encapsular** a funcionalidade de um objeto dentro do próprio objeto, significando que o código externo não deve estar fazendo o trabalho que um objeto pode fazer para si mesmo. Neste caso, a comparação das entradas do blog para classificar pode ser feita dentro do objeto, ao invés da função `showBlog( )`. Mas o código de comparação da classificação pode ser colocado em um método da classe da classe `Blog`? Para responder a esta pergunta, temos que descobrir se o método requer acesso aos dados ou aos métodos da instância. Isso seria um grande problema, uma vez que os métodos da classe não podem acessar nada em uma instância.

Examine a função de comparação da classificação

O único modo de saber se isso funcionará é decompor a função e ver o que está acontecendo. Esta é a literal da função de comparação da classificação com o código formatado mais como uma função normal:

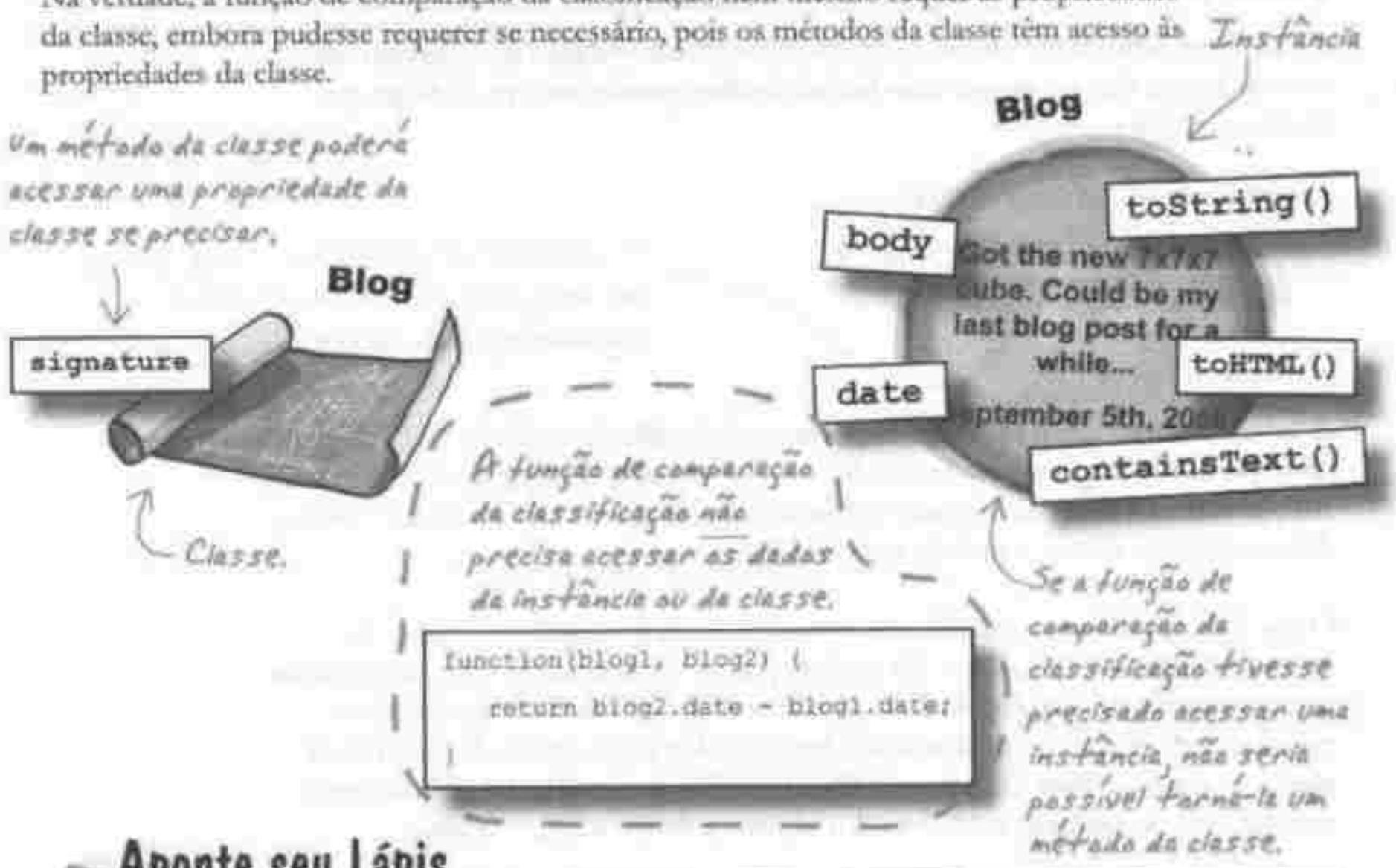
```
function(blog1, blog2) {
  return blog2.date - blog1.date;
}
```

Duas instâncias do blog são transmitidas para a função como argumentos.

A comparação da classificação envolve uma subtração dos dois argumentos.

Embora a função lide diretamente com as instâncias do blog, elas são transmitidas como argumentos. Isso é diferente de tentar acessar uma propriedade ou um método **dentro** de uma instância através da palavra-chave `this`, o que não é possível em um método da classe. Portanto, a função de comparação da classificação não precisa acessar nada dentro de uma instância, o que a torna uma candidata perfeita para um método da classe.

Na verdade, a função de comparação da classificação nem mesmo requer as propriedades da classe, embora pudesse requerer se necessário, pois os métodos da classe têm acesso às propriedades da classe.



Aponte seu Lápis

Reescreva a função de comparação da classificação do blog YouCube como um método da classe do objeto `Blog` que é denominado `blogSorter( )`.

.....

.....

Aponte seu Lápis
Solução

Reescreva a função de comparação da classificação do blog YouCube como um método da classe do objeto `Blog` que é denominado `blogSorter( )`.

```
Blog.blogSorter = function(blog1, blog2) {

  return blog2.date - blog1.date;

};
```

O método de comparação da classificação é, agora, um método da classe do objeto Blog chamado blogSorter( ).

Como chamar um método da classe

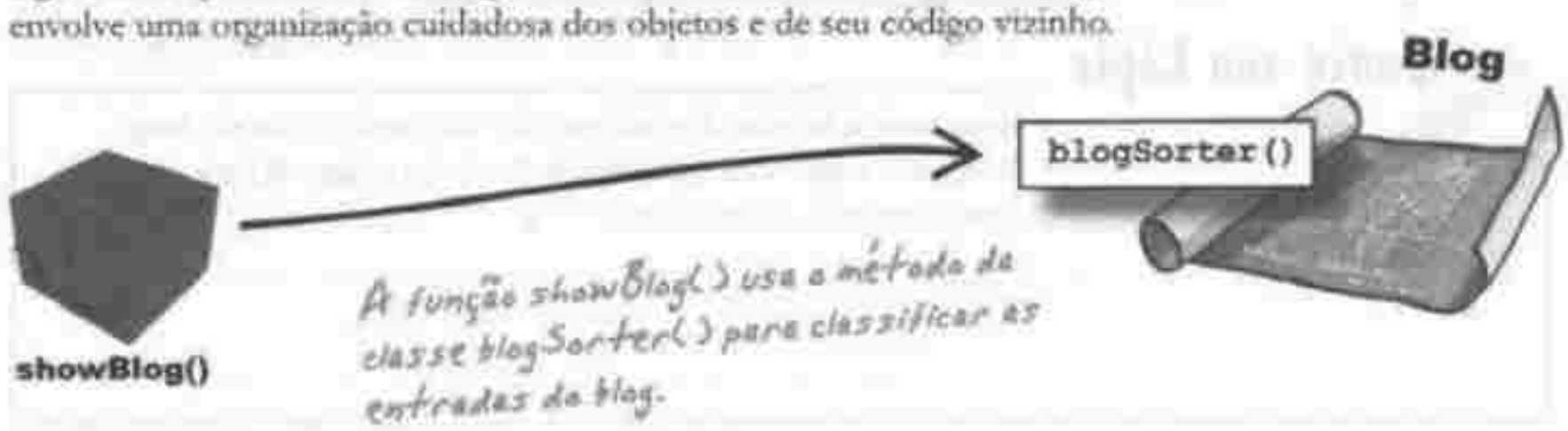
A vantagem de mover a função de comparação da classificação do blog para um método `Blog` fica um pouco mais fácil de avaliar quando você vê o código que chama o método.

```
function showBlog(numEntries) {
  // Primeiro, classifique o blog
  blog.sort(Blog.blogSorter);
  ...
}
```

Os detalhes de classificar o blog, agora, são delegados ao método da classe blogSorter( ) da classe Blog.

A beleza deste código é sutil, mas importante. A função `showBlog( )` não tem mais que se preocupar com o modo de como as entradas do blog são classificadas. Ao contrário, os detalhes de como as entradas do blog são classificadas são lidados dentro da classe `Blog`, a qual ela pertence logicamente.

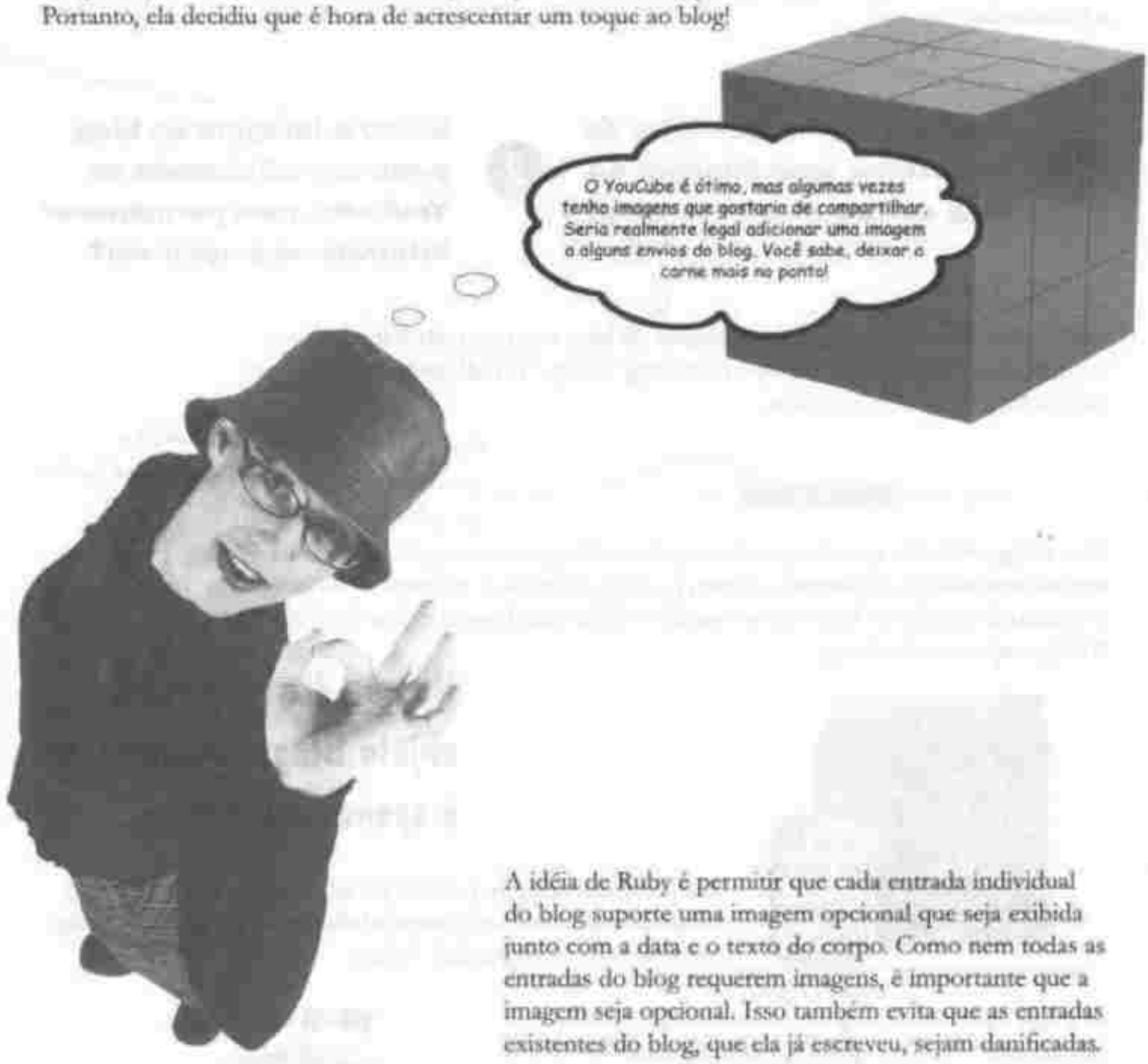
O que é ótimo é como a classificação é iniciada ainda fora da classe `Blog` na função `showBlog( )`, o que faz sentido porque a classificação afeta uma coleção inteira de instâncias do blog. Mas, as particularidades de como a classificação é executada em relação às entradas individuais do blog estão dentro do domínio de algo com o qual a classe `Blog` pode lidar. Uma boa construção OOP geralmente envolve uma organização cuidadosa dos objetos e de seu código vizinho.



A função showBlog( ) usa o método da classe blogSorter( ) para classificar as entradas do blog.

Uma imagem vale mais que mil palavras

Ruby continua a vibrar com as melhorias OOP para o YouCube, mas também sabe que os usuários não compartilharão, necessariamente, de seu entusiasmo porque os aperfeiçoamentos OOP têm ainda que impactar muito a experiência do usuário. Portanto, ela decidiu que é hora de acrescentar um toque ao blog!



A idéia de Ruby é permitir que cada entrada individual do blog suporte uma imagem opcional que seja exibida junto com a data e o texto do corpo. Como nem todas as entradas do blog requerem imagens, é importante que a imagem seja opcional. Isso também evita que as entradas existentes do blog, que ela já escreveu, sejam danificadas.

Poder do cérebro

Você poderia alterar o objeto `Blog` no YouCube para suportar imagens?

Como incorporar imagens no YouCube

Adicionar o suporte de imagem ao blog YouCube envolve descobrir como incorporar uma imagem no objeto `Blog` de tal modo que não interfira no modo como o objeto já funciona. Isso levanta duas questões importantes que conduzirão a construção:

1 Qual é a melhor maneira de armazenar uma imagem do blog dentro do objeto Blog?

2 Como a imagem do blog pode ser adicionada ao YouCube, mas permanecer inteiramente opcional?

Independentemente de como uma imagem do blog é armazenada, sabemos que no final das contas será fornecida em uma tag `<img>` HTML para que possa ser exibida na página Web YouCube.

```
<img src="cube777.png" />
```

Uma imagem do blog é descrita o suficiente usando um nome de arquivo da string.

Este código informa que, no que diz respeito ao blog, uma imagem é realmente apenas uma string. Certamente, a string, no final, refere-se a um arquivo de imagem armazenado em algum lugar em um servidor Web, mas da perspectiva do objeto `Blog`, é apenas uma string.

No que diz respeito ao objeto Blog, uma imagem é apenas uma string.

Portanto, uma imagem poderia ser adicionada ao objeto `Blog` simplesmente como uma propriedade que armazena uma string, parecido com a propriedade `body`.

cube777.png

A imagem do cubo com 7x7x7 é armazenada no arquivo denominado cube777.png.

As propriedades body e image são armazenadas no objeto Blog como strings.

Uma imagem do blog opcional

2 Então, a imagem do blog é armazenada no objeto `Blog` como uma propriedade de string denominada `image`, mas a pergunta ainda permanece como essa propriedade pode ser adicionada ao blog como um recurso puramente opcional. Esta pergunta, basicamente, tem que levar de volta ao construtor, que é onde o objeto é criado e inicializado. Certamente, algum código especial tem que ser colocado no construtor para lidar com o fato de que a propriedade é opcional.

image

?

```
function Blog(body, date) {
  // Atribua as propriedades
  this.body = body;
  this.date = date;
}
```

Não estou certo sobre isso. O que acontecerá se você não transmitir um argumento para um construtor? A propriedade não será simplesmente definida para null?

Os argumentos da função ausentes tornam-se nulos.

Quando um argumento não é transmitido para uma função, método ou construtor, ele assume um valor `null` em qualquer código que tenta usá-lo. E no caso de um construtor especificamente, isso significa que a propriedade associada a um argumento ausente é definida para `null`, o que não é necessariamente algo ruim. O truque real é, então, assegurar que o argumento opcional do construtor seja especificado no final da lista de argumentos para que possa ser retirado sem danificar os outros argumentos. Essa técnica, de fato, funciona para qualquer função ou método, mas é particularmente útil para o argumento `image` do construtor `Blog( )`.

Aponte seu Lápis

Reescreva o construtor `Blog( )` do YouCube para suportar uma nova propriedade da imagem para armazenar uma imagem de entrada do blog.

Aponte seu Lápis Solução

Reescreva o construtor `Blog( )` do YouCube para suportar uma nova propriedade da imagem para armazenar uma imagem de entrada do blog.

A propriedade image é adicionada e inicializada para o argumento image.

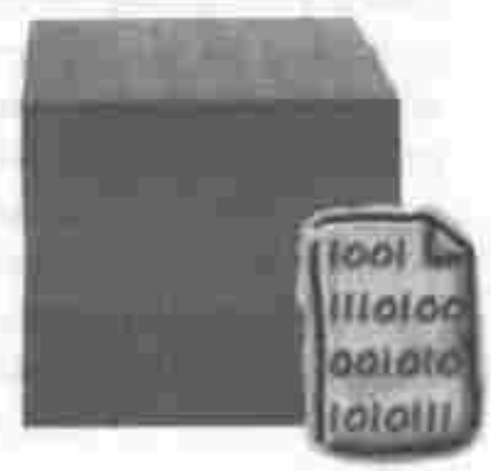
```
function Blog(body, date, image) {

// Atribua as propriedades

this.body = body;

this.date = date;

this.image = image;

}
```

Um argumento image é adicionado como o último argumento ao construtor.

Como adicionar imagens ao YouCube

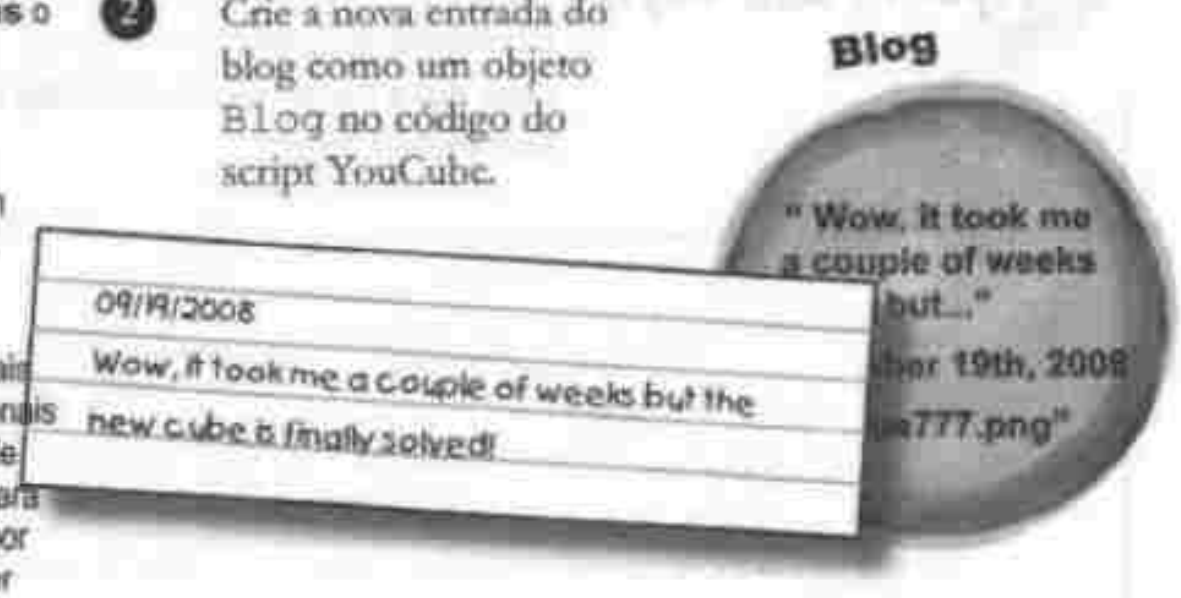
O construtor `Blog( )` novo e brilhante com suporte para as imagens não seria muito útil sem algumas entradas do blog que o utilizam. Para criar uma entrada do blog que suporta imagens, duas coisas devem ocorrer:

1. Coloque o arquivo de imagem do blog na mesma pasta do servidor Web como a página Web YouCube.



cube777.png

2. Crie a nova entrada do blog como um objeto `Blog` no código do script YouCube.



Não existem Perguntas Idiotas

P: É importante que o argumento `image` apareça por último na lista de argumentos do construtor `Blog( )`?

R: Sim, e a razão é que a imagem é considerada uma parte opcional de uma entrada do blog. O problema real aqui é como você irá transmitir os argumentos para as funções, especialmente quando pertencem aos argumentos opcionais. Se uma função tiver dois argumentos, você terá a opção de transmitir ambos os argumentos, transmitir apenas o primeiro argumento ou transmitir nenhum deles. Não há nenhum modo de transmitir **apenas** o segundo argumento.

Então, para os argumentos opcionais, considere-os como sendo capaz de retirá-los do final da lista de argumentos. Também tente considerar os argumentos em termos de importância, com os argumentos mais importantes aparecendo primeiro. Os argumentos menos importantes que têm mais probabilidade de serem considerados opcionais devem aparecer próximos ao final da lista de argumentos. Como o argumento `image` para `Blog( )` é opcional, ele deve aparecer por último na lista de argumentos onde pode ser retirado facilmente.

Completar estas etapas resultará no seguinte código, que cria com sucesso uma nova entrada do blog que transmite uma string de imagem no último argumento do construtor `Blog( )`:

```
new Blog("Wow, it took me a couple of weeks but the new cube is finally solved!",
    new Date("09/19/2008"), "cube777.pog")
```

A imagem do blog é transmitida no último argumento do construtor Blog( ).

Como exibir a imagem do blog

Agora que uma entrada do blog foi criada com uma imagem, há uma última parte do negócio para a melhoria da imagem no YouCube. Toda essa conversa de construtores e argumentos opcionais não significaria muito se o código que exibe uma entrada do blog, de fato, não se dividisse na nova propriedade `image`.

Este código está localizado no método `toHTML( )`. Já sabemos que esse método é responsável por formatar o blog como o código HTML, exceto agora que ele tem que levar em conta se a propriedade image tem ou não um valor significativo. O que realmente está ocorrendo é que existem duas maneiras diferentes de exibir uma entrada do blog, uma com uma imagem e outra sem, e a existência de uma imagem é o que determina de qual maneira o blog é exibido.

Uma entrada do blog deve, agora, ser exibida segundo a lógica deste pseudocódigo.

If (imagem existir)

Exiba a entrada do blog com a imagem

Else

Exiba a entrada do blog sem imagem

Aponte seu Lápis

O método `toHTML( )` do objeto `Blog` não tem uma parte do código que permitirá às imagens opcionais serem exibidas. Escreva a parte do código que falta e anote o que ela faz.

```
if (.......................) {
   blogHTML += "<strong>" + this.date.shortFormat( ) +
      "</strong><br /><table><tr><td><img src='" + this.image +
      "'/></td><td style='vertical-align:top'>" + this.body + "</td></tr></table><em>" +
      this.signature + "</em></p>";
}
else {
   blogHTML += "<strong>" + this.date.shortFormat( ) + "</strong><br />" + this.body +
      "<br /><em>" + this.signature + "</em></p>";
}
```

Aponte seu Lápis
Solução

O método `toHTML( )` do objeto `Blog` não tem uma parte do código que permitirá às imagens opcionais serem exibidas. Escreva a parte do código que falta e anote o que ela faz.

Se a propriedade image for definida para uma imagem real, a condição de teste if será avaliada como true e a imagem será exibida.

this.image

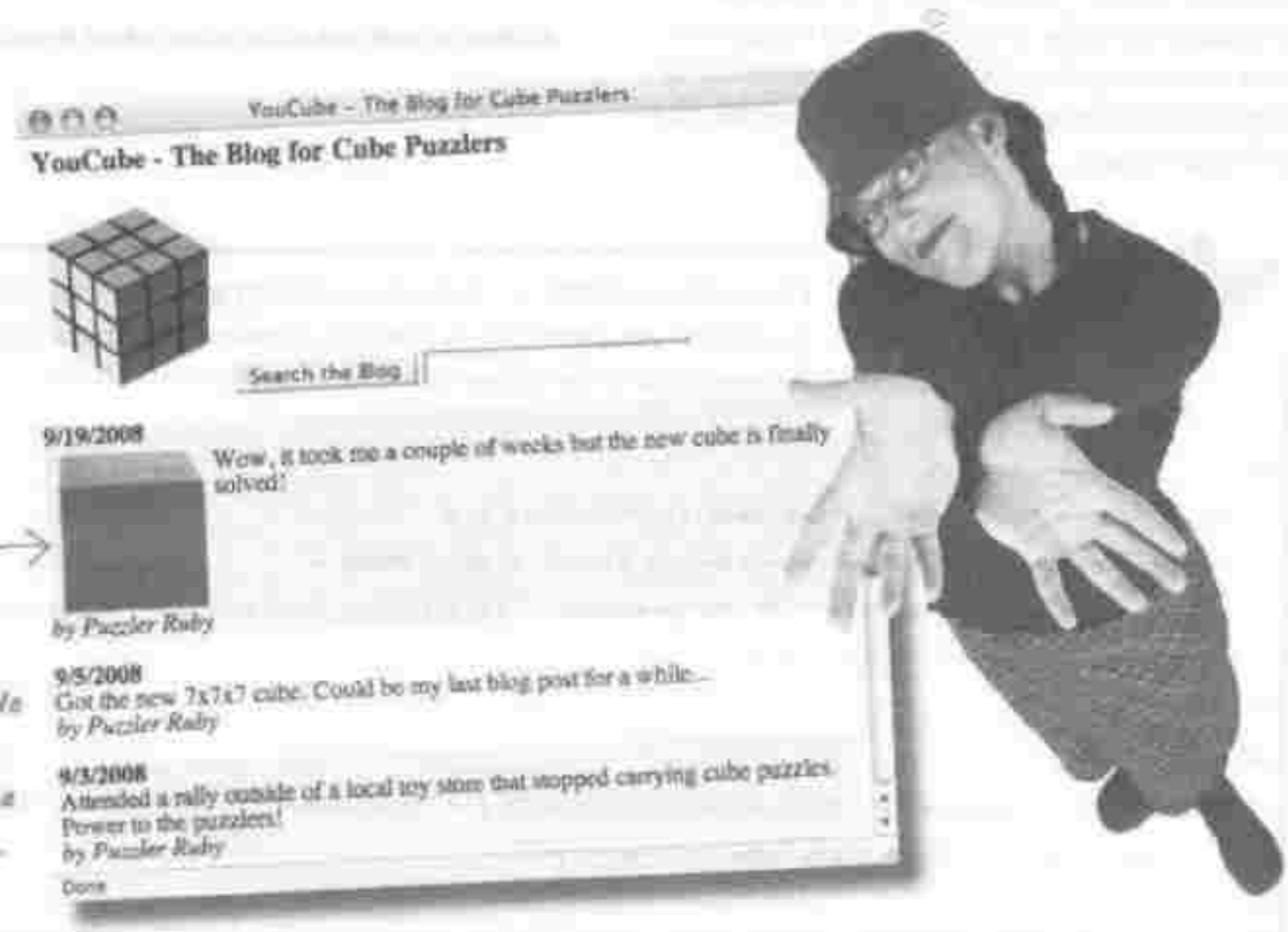
```
if (.......................) {
  blogHTML += "<strong>" + this.date.shortFormat( ) +
    "</strong><br /><table><tr><td><img src='" + this.image +
    "'/></td><td style='vertical-align:top'>" + this.body + "</td></tr></table><em>" +
    this.signature + "</em></p>";
}
else {
  blogHTML += "<strong>" + this.date.shortFormat( ) + "</strong><br />" + this.body +
    "<br /><em>" + this.signature + "</em></p>";
}
```

Do contrário, a entrada do blog será mostrada como normal sem imagem.

Um YouCube acionado por objetos

Ruby está em êxtase. Seu blog cresceu com grande velocidade graças aos objetos e agora, exibe um recurso de imagem novo e agradável que ela sabe que seus visitantes irão adorar.

Sei... está lindo.

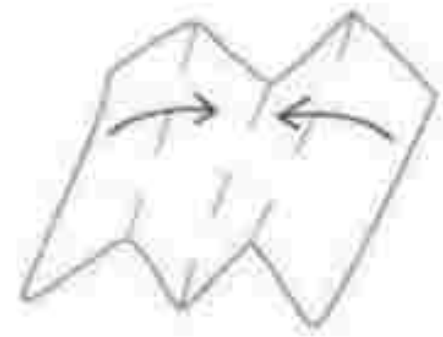


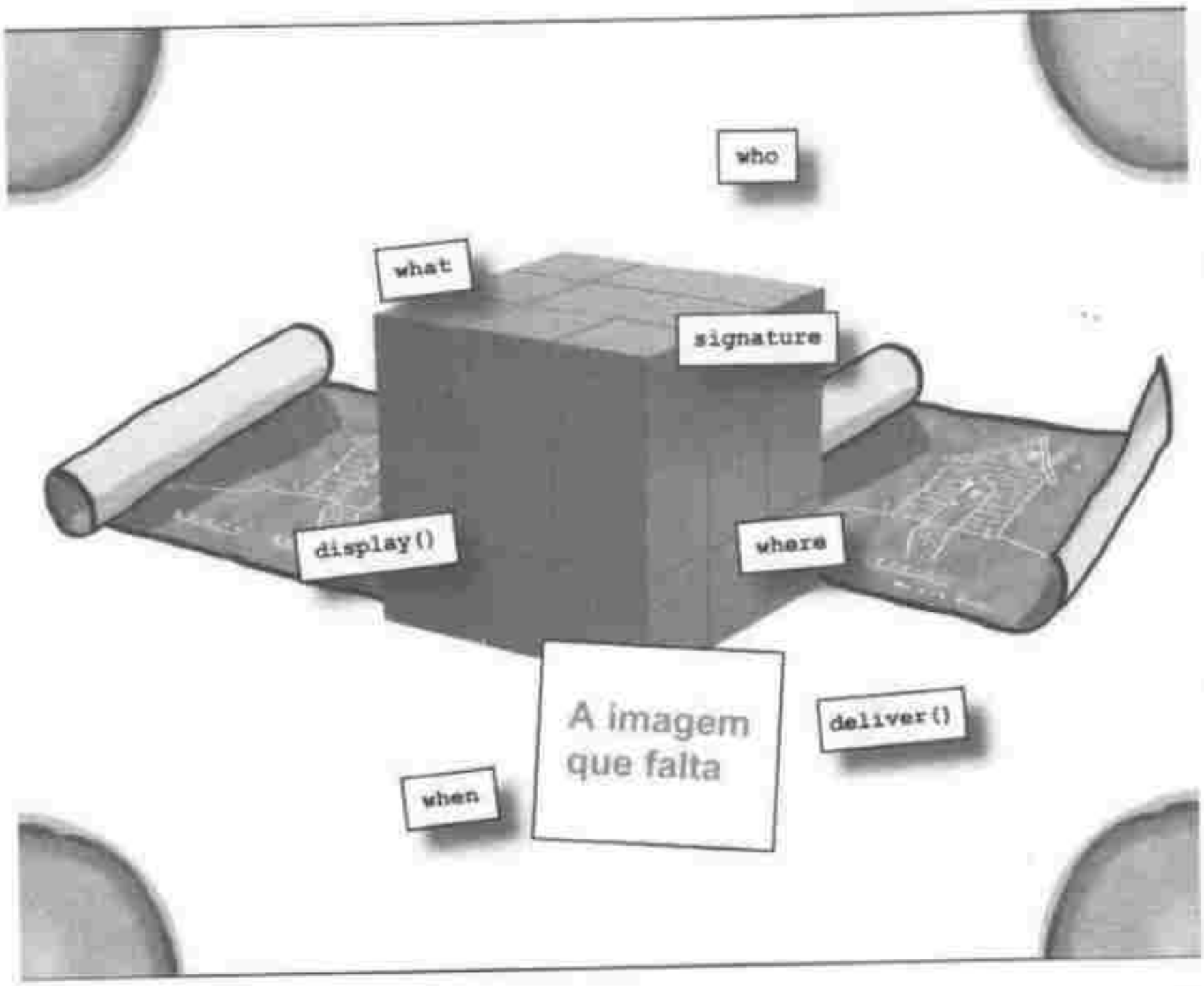
Uma entrada do blog com uma imagem.

DOBRA DA PÁGINA

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

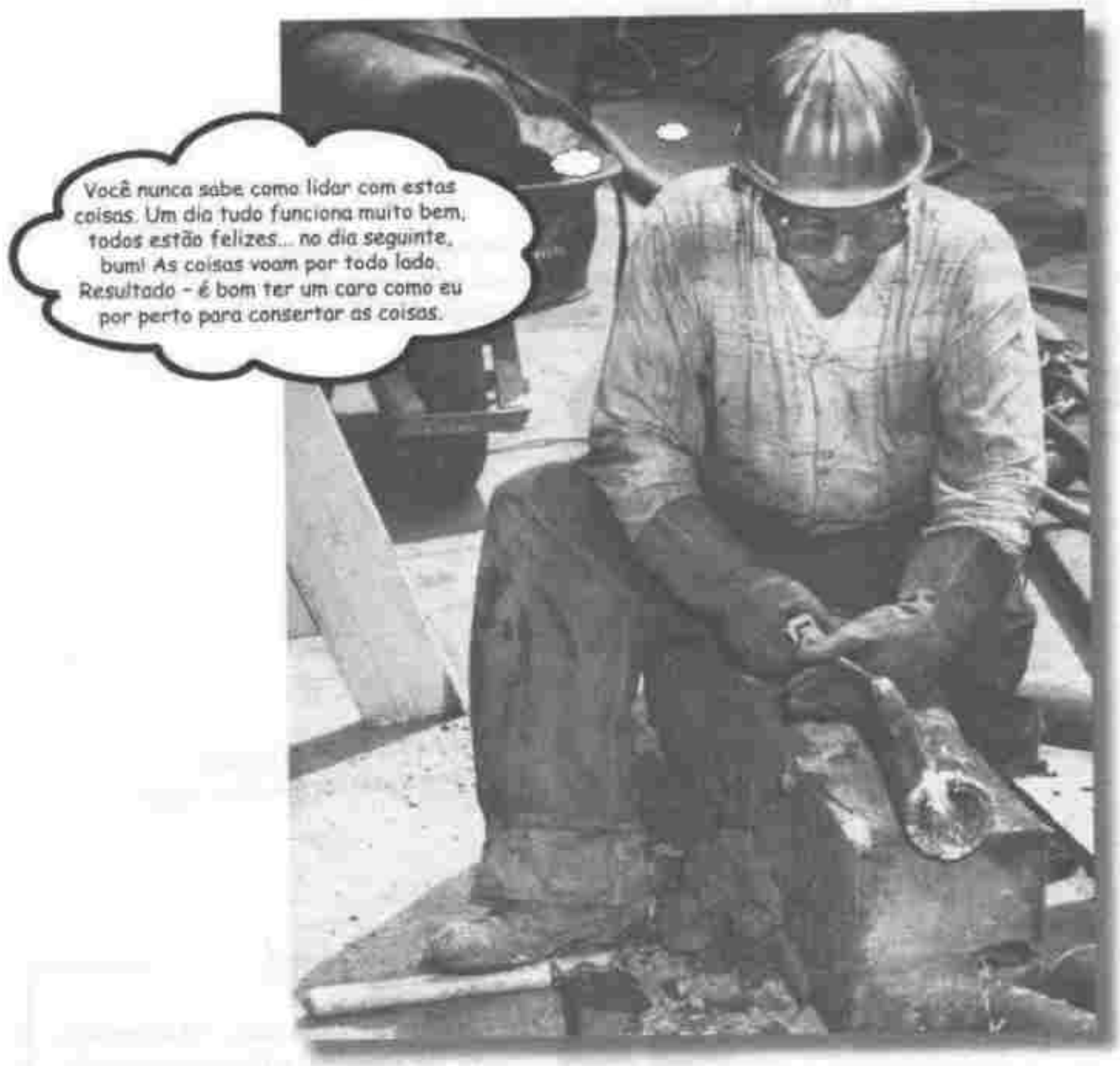
O que os objetos adicionam à maioria dos scripts?





Os objetos adicionam tantas coisas legais aos scripts que pode ser difícil escolher uma. É verdade, alguns objetos são "fora de classe" aos outros e fica ainda mais difícil, mas no final a resposta é clara.

Os bons scripts erraram



Mesmo os melhores planos JavaScript preparados falham algumas vezes. Quando isto acontece, e acontecerá, sua função será não entrar em pânico. Os melhores desenvolvedores JavaScript não são aqueles que nunca criam erros – essas pessoas são chamadas de mentirosas. Não, os melhores desenvolvedores JavaScript são aqueles capazes de **perseguir com sucesso e erradicar** os **erros** criados. O mais importante, os últimos exterminadores de erros JavaScript **desenvolvem bons hábitos de codificação** que minimizam os mais furtivos e sórdidos erros. ***Uma pequena prevenção pode contribuir muito***. Mas os erros acontecem e você precisará de um arsenal de armas para combatê-los...

Uma depuração real

É um fator chocante da vida dos lanches... uma barra de chocolate pode conter até 60 fragmentos de um inseto. Por mais assustador que possa ser esse pequeno petisco de informação, não há nenhuma razão para temer os erros no código JavaScript. O código JavaScript pode ser mais precisamente controlado do que o equipamento que processa os chocolates. Há até uma força-tarefa dedicada unicamente à remoção dos erros JavaScript.

ICE:

INVESTIGADORES DA CENA DO ERRO

São os Investigadores da Cena do Erro, ou ICE, como são referidos no ramo. Owen juntou-se ao ICE recentemente como um investigador JavaScript e está ansioso para provar e ajudar a livrar a Web dos erros JavaScript.

Entre Owen e o sucesso estão vários casos que precisam desesperadamente de sua atenção. Ele terá que dominar a magia da depuração JavaScript antes de poder subir de posição e tornar-se um detetive JavaScript completo.

Nerd

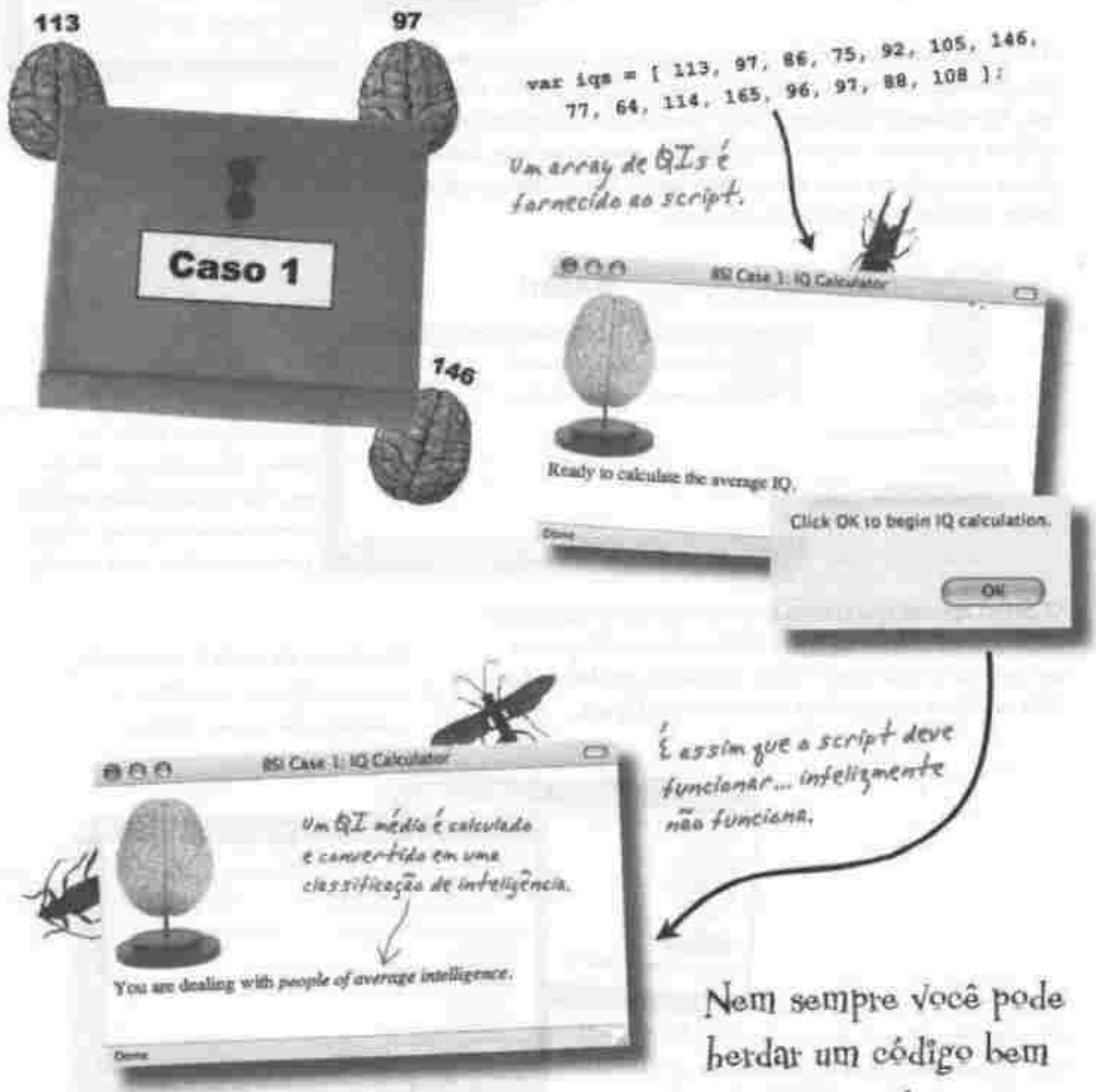
Segundo a Administração de Alimentos e Drogas dos Estados Unidos, até 60 "fragmentos" de inseto são permitidos em qualquer determinada barra de chocolate. Em oposição ao "mundo real", as pessoas no ICE têm uma política de tolerância zero para os erros JavaScript e você também deveria.

O caso da calculadora QI com erros

O primeiro caso que chegou à mesa de Owen é um script da calculadora QI que faz parte de uma página que calcula o QI a partir de um array de QIs e então, agrupa os usuários cujos resultados são parecidos. Portanto, o script recebe um array de números e depois, calcula uma média e indica a inteligência dessa média.

Os arquivos para Owen estão disponíveis em www.altabooks.com.br.

Owen foi informado que esse script tem muitos erros. Infelizmente, nenhuma outra informação foi fornecida além de "não funciona".



Nem sempre você pode herdar um código bem escrito para depurar.

Experimente navegadores diferentes

Owen calcula que executar o script problemático em alguns navegadores diferentes poderá ajudar a trazer alguma luz para o problema. Ele inicia com o Internet Explorer...

Clicar duas vezes no sinal amarelo no canto inferior esquerdo do IE abrirá uma janela de erro.

O Internet Explorer informa um erro quando a página é carregada pela primeira vez, mas Owen não está certo de que pode confiar nele. Uma olhada rápida no código para obter o script mostra que a variável `iqs` existe, mas o navegador IE mostra que não. Sabendo que os navegadores nem sempre são precisos ao informar erros, ele decide insistir com o navegador Safari...

A variável iqs é definida no código, portanto o erro do IE não parece fazer nenhum sentido.

Se você considerar as linhas de código que iniciam em /, o Safari apontará um erro em uma linha de código que parece boa inicialmente.

O Safari aponta que o erro está em uma linha de código inteiramente diferente, que imediatamente não parece ter nada de errado para Owen. Portanto, ele decide fazer uma tentativa em localizar o erro com o Opera...

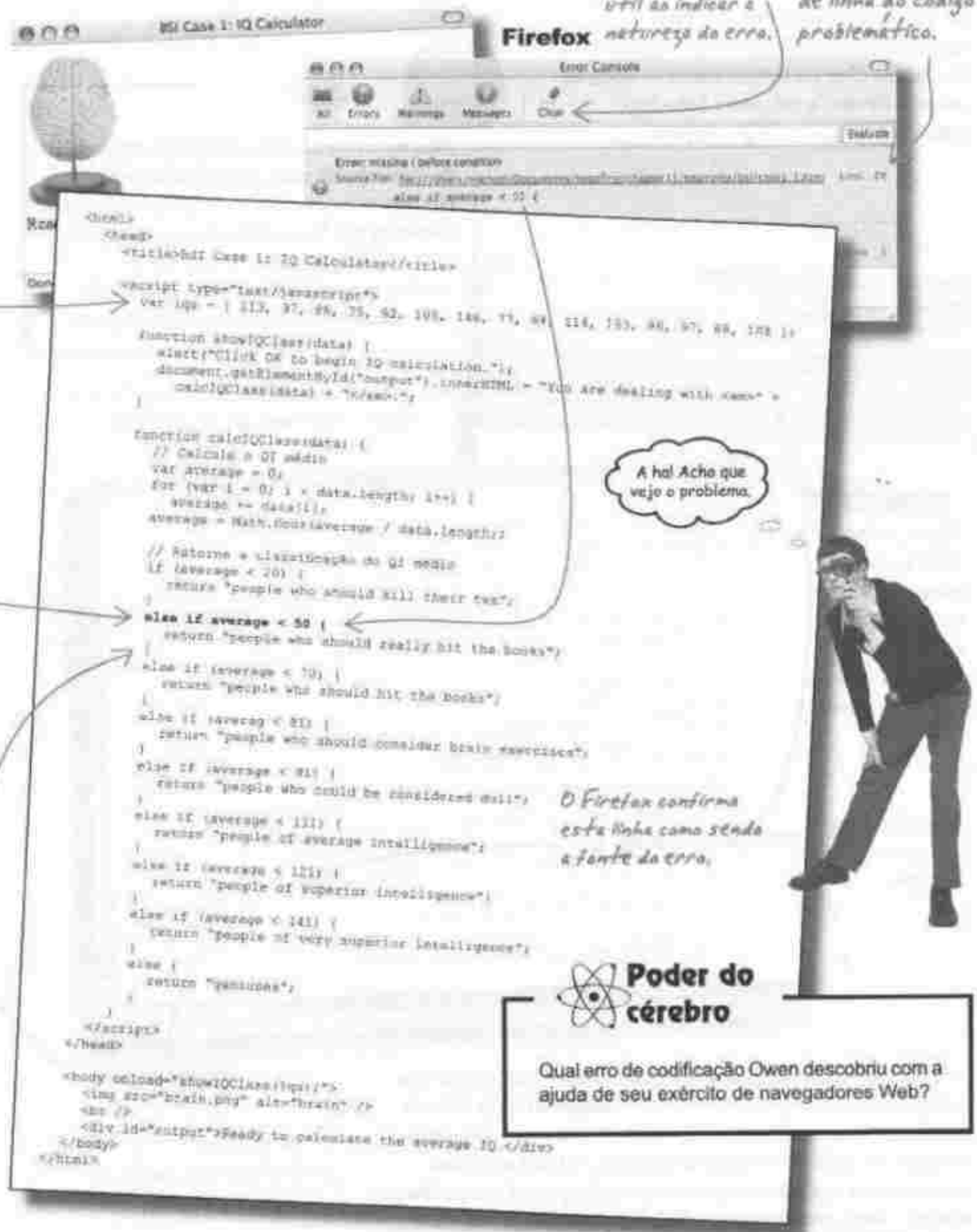
O número da linha é diferente, mas o código no erro Opera coincide com o erro Safari.

Os consoles de erro do navegador são uma ótima maneira de diagnosticar os problemas de codificação JavaScript.

Tem algo estranho aqui. O Opera menciona um número de linha diferente, mas está referindo-se claramente à mesma linha de código do Safari, o que é realmente uma boa notícia para Owen. Mas ele ainda não vê nada errado com o código. Portanto, decide tentar mais uma opção, o Firefox.

O Firefox é muito útil ao indicar a natureza do erro.

O Firefox adquiriu ainda outro número de linha do código problemático.



A ha! Acho que vejo o problema.

O Firefox confirma esta linha como sendo a fonte do erro.

Poder do cérebro

Qual erro de codificação Owen descobriu com a ajuda de seu exército de navegadores Web?

Firefox para o resgate

Dado o modo como o Firefox é específico ao descrever o erro, Owen decide aprofundar-se um pouco mais usando-o. Então, ele clica no link na janela de console do erro Firefox e vai para a linha logo anterior à linha de código suspeita.

O Firefox é amplamente considerado o melhor navegador de depuração, pelo menos por enquanto.

Nerd

Não só o Firefox tem excelentes capacidades de detecção de erros predefinidas, mas também tem um depurador plug-in chamado **Firebug** que leva a depuração a um nível inteiramente diferente. O depurador Firebug para o Firefox pode ser baixado gratuitamente em *http://www.getfirebug.com/*.

Não se preocupe com o segundo erro agora – veremos um de cada vez.

Seguir o link abrirá o código para a página Web, incluindo um destaque logo antes do código suspeito.

Este é o código que causa o problema.

Source of: file:///Users/michael/Documents/headfirst/examples/chapter11/bsi/ca...

```
<html>
  <head>
    <title>BSI Case 1: 2D Calculator</title>

    <script type="text/javascript">
      var cpr = [ 113, 97, 86, 15, 80, 105, 144, 77, 64, 214, 135, 96, 37, 98, 108 ];

      function showTOCIaes(data) {
        alert("Click OK to begin 2D calculation.");
        document.getElementById("output").innerHTML = "You are dealing with <ul>" +
          calcTOCIaes(data) + "</ul>";
      }

      function calcTOCIaes(data) {
        // Calcula a DT média
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        average = Math.floor(average / data.length);

        // Retorna a classificação da DT média
        if (average < 25) {
          return "people who should kill their cats";
        }
        else if average < 50 {
          return "people who should really hit the books";
        }
        else if (average < 75) {
          return "people who should hit the books";
        }
        else if (average < 85) {
          return "people who should consider brain exercises";
        }
        else if (average < 95) {
```

Line 25, Col 2

```
else if average < 50 {
```

A instrução if não tem parênteses em volta da condição de teste.

Analisando a mensagem de erro Firefox, Owen descobriu que o Safari está realmente certo sobre o número da linha (25). O Firefox destaca e menciona a linha 24, mas mostra corretamente o código na linha 25. O mais importante, o Firefox explica exatamente o que está errado com o código, que é um problema simples, ainda que seja muito fácil de deixar passar.

Não existem Perguntas Idiotas

P: Não consigo descobrir como exibir o console de erro em meu navegador. Como posso abri-lo?

R: Infelizmente, cada navegador é diferente e alguns o tornam um desafio para encontrar o console de erro JavaScript. Por exemplo, o Safari no Mac permite apenas acessar o console de erro a partir do menu Debug (Depurar), que está desativado por padrão. Você tem que enviar o seguinte comando (escrevê-lo em uma linha, sem retorno de linha) no aplicativo Terminal para mudar as definições e ativar o menu Debug:

```
defaults write com.apple.Safari
IncludeDebugMenu 1
```

Verifique a documentação de seu navegador específico para descobrir como abrir o console de erro e exibir os erros do script. O console de erro no Firefox é aberto selecionando Error Console (Console de Erro) no menu Tools (Ferramentas).

P: O que torna o Firefox tão especial?

R: Os desenvolvedores do Firefox fizeram um ótimo trabalho em suas capacidades de informar erros. Ele simplesmente supera os outros navegadores ao avaliar os erros do script e apontar a direção certa para encontrá-los. Isso não quer dizer que algum outro navegador poderá acompanhar e fazer um serviço melhor em algum momento no futuro, mas o Firefox tem se mostrado um navegador muito capaz para depurar as páginas que contêm o código JavaScript.

P: Sobre qual erro o Internet Explorer estava falando?

R: Realmente não há nenhuma maneira de saber com certeza. A razão é que o erro informado pelo Internet Explorer tem relação com o código do script não carregando a propriedade, que é resultado de um erro encontrado pelo interpretador JavaScript. Você sabe que o código não está sendo devidamente carregado porque a variável `iqs` está sendo informada como "indefinida", mesmo que o código mostre claramente que a variável `iqs` é criada. Portanto, o único modo de poder ser um problema é se algum outro erro estiver impedindo de alguma maneira que o script seja totalmente carregado.

E isso levanta a pergunta... há algum outro erro no script e o que "indefinido" realmente significa?

Como depurar de modo fácil

Owen está muito entusiasmado com a separação rápida do erro no script da calculadora QI. E com tal correção simples do código feita, ele calcula que será capaz de avançar em seu serviço e tornar-se um detetive ICE em pouco tempo.

Há uma chance de Owen ter conseguido uma dose prematura de confiança excessiva? Ele realmente precisa testar o script recém-corrigido antes de tirar o resto do dia de folga...

O relatório do erro nem sempre é a fonte do erro

Infelizmente, Owen não terminou com o caso da calculadora QI porque o Firefox ainda está reclamando, com exceção de que agora é um problema completamente diferente. Embora ele esteja tentado manter a mesma tática de confiar na avaliação do Firefox, implicitamente, desta vez Owen tem suas dúvidas sobre a validade do erro informado.

A chave desta função coincide com a chave mencionada pelo Firefox, portanto nada falta nas chaves da função.

Novamente, um erro de cada vez, portanto, continuemos a ignorar este agora.

Error Console

Error: missing) before condition

Source File: file:///Users/mcdonald/Documents/Heads%20First%20Web%20Dev/ch11/iqcalc/iqcalc.js?_=1243 Line: 24

```
else if average < 50 {
```

Error: showIQClass is not defined

Source File: file:///Users/mcdonald/Documents/Heads%20First%20Web%20Dev/ch11/iqcalc/iqcalc.html Line: 1

Esta é a chave sobre a qual o Firefox está reclamando

```
<html>
  <head>
    <title>HFWD Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var IQs = [ 113, 87, 98, 95, 92, 105, 146, 77, 54, 164, 160, 96, 97, 88, 108 ];

      function showIQClass(data) {
        alert("Click OK to begin IQ calculation.");
        document.getElementById("output").innerHTML = "You are dealing with <em>" +
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // Calcula o QI médio
        var average = 0;
        for (var i = 0; i < data.
          average += data[i];
        average = Math.floor(avera

        // Retorna a classificação
        if (average < 50) {
          return "people who shou
        }
        else if (average < 60) {
          return "people who shou
        }
        else if (average < 70) {
          return "people who shou
        }
        else if (average < 80) {
          return "people who shou
        }
        else if (average < 90) {
          return "people who could
        }
        else if (average < 110) {
          return "people of average intelligence";
        }
        else if (average < 120) {
          return "people of superior intelligence";
        }
        else if (average < 140) {
          return "people of very superior intelligence";
        }
        else {
          return "geniuses";
        }

      </script>
    </head>

    <body onload="showIQClass(IQs)">
      <image src="brain.png" alt="brain" />
      <br />
      <div id="output">Ready to calculate the average IQ.</div>
    </body>
  </html>
```

Acho que seu depurador mágico simplesmente confundiu tudo. Claramente, as chaves na função neste código são boas.

Nem sempre você pode confiar no navegador.

É verdade, as chaves da função são boas. Por melhor que possa ser em alguns casos, parece que o Firefox está desperdiçando seus esforços nesse erro. Porém, vale a pena considerar a menção de uma chave ausente como uma dica para estudar todas as chaves no código um pouco mais de perto.

Seja o interpretador JavaScript

Seu trabalho é ser o interpretador JavaScript e seguir a trilha das chaves no código para descobrir o que deu errado.

```
<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var iqs = [ 113, 87, 98, 75, 92, 105, 148, 77, 69, 114, 153, 96, 97, 88, 108 ];

      function showIQClass(data) {
        alert("Click OK to begin IQ calculation.");
        document.getElementById("output").innerHTML = "You are dealing with <em>" +
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // Calcule o QI médio
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        average = Math.floor(average / data.length);

        // Retorne a classificação do QI médio
        if (average < 20) {
          return "people who should";
        }
        else if (average < 50) {
          return "people who should";
        }
        else if (average < 70) {
          return "people who should";
        }
        else if (averag < 81) {
          return "people who shouldDeficient people";
        }
        else if (average < 91) {
          return "people who could";
        }
        else if (average < 111) {
          return "people of average intelligence";
        }
        else if (average < 121) {
          return "people of superior intelligence";
        }
        else if (average < 141) {
          return "people of very superior intelligence";
        }
        else {
          return "geniuses";
        }
      }
    </script>
  </head>

  <body onload="showIQClass(iqs);">
    <img src="brain.png" alt="brain" />
    <br />
    <div id="output">Ready to calculate the average IQ.</div>
  </body>
</html>
```

Seja o interpretador JavaScript Solução

Seu trabalho é ser o interpretador JavaScript e seguir a trilha das chaves no código para descobrir o que deu errado.

```
<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var iqs = [ 113, 87, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96, 97, 89, 108 ];

      function showIQClass(data) {
        alert("Click OK to begin IQ calculation.");
        document.getElementById("output").innerHTML = "You are dealing with <em>" +
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // Calcule o QI médio
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        average = Math.floor(average / data.length);

        // Retorne a classificação do QI médio
        if (average < 20)
          return "people who should";
        else if (average < 50)
          return "people who should";
        else if (average < 70)
          return "people who should";
        else if (averag < 81)
          return "people who shouldeficient people";
        else if (average < 91)
          return "people who could";
        else if (average < 111)
          return "people of average intelligence";
        else if (average < 121)
          return "people of superior intelligence";
        else if (average < 141)
          return "people of very superior intelligence";
        else
          return "geniuses";
      }
    </script>
  </head>

  <body onload="showIQClass(iqs);">
    <img src="brain.png" alt="brain" />
    <br />
    <div id="output">Ready to calculate the average IQ.</div>
  </body>
</html>
```

A chave de fechamento que falta deve ficar aqui, fechando apenas o acréscimo da variável average.

Esta chave não tem sua chave de fechamento!

Adicionar a chave que falta acaba com o erro.

Você poderia também simplesmente encerrar a chave de abertura depois do loop for, uma vez que ele está executando apenas uma única linha de código, embora as chaves ajudem a tornar claro qual código está sendo executado no loop.

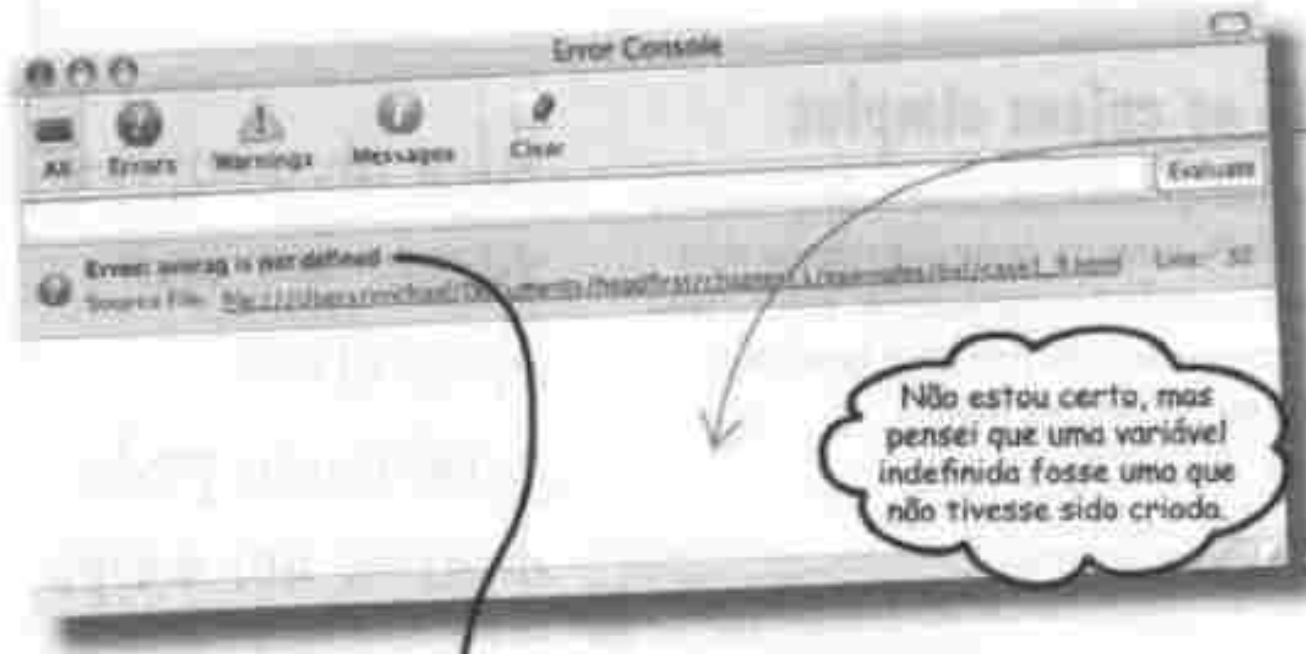
As chaves mal combinadas ou ausentes são um erro comum do JavaScript, que pode ser evitado com atenção aos detalhes.

As variáveis ficaram selvagem indefinida

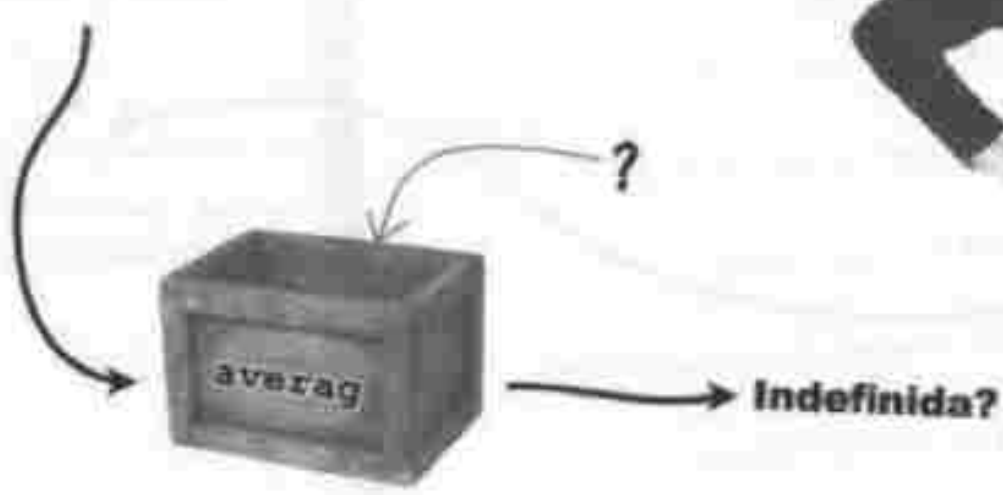
Owen parece não conseguir parar, pois o fluxo de erros da calculadora QI continua.

Agora, o Firefox está informando que uma variável "não está definida", que parece o tipo de erro falso que o Internet Explorer informou no início. Exceto que desta vez a variável indefinida é `averaq`, não `iqs`.

Note que o segundo erro agora sumiu. Algumas vezes, corrigir um erro resolve naturalmente mais de um erro.




```
else if (averag < 61) {
  return "people who should consider brain exercises";
}
```



Aponte seu Lápis

Escreva o que você acha que significa "indefinido" no contexto do último erro na investigação de Owen.

.....

.....

.....

Aponte seu Lápis
Solução

Escreva o que você acha que significa "indefinido" no contexto do último erro na investigação de Owen.

Indefinida refere-se a uma variável que não foi criada (usando var) ou que foi criada, mas não atribuída ainda a um valor. De qualquer modo, o problema é que a variável está sendo referenciada mesmo que não tenha nenhum valor.

Algumas vezes, são as coisas simples

Neste caso, "indefinido" definitivamente refere-se a uma variável que foi usada sem ter sido criada, embora neste caso tenha sido por acaso. A única razão para a variável ser indefinida é por causa de um erro tipográfico que resulta no interpretador JavaScript pensando que é uma variável inteiramente nova.

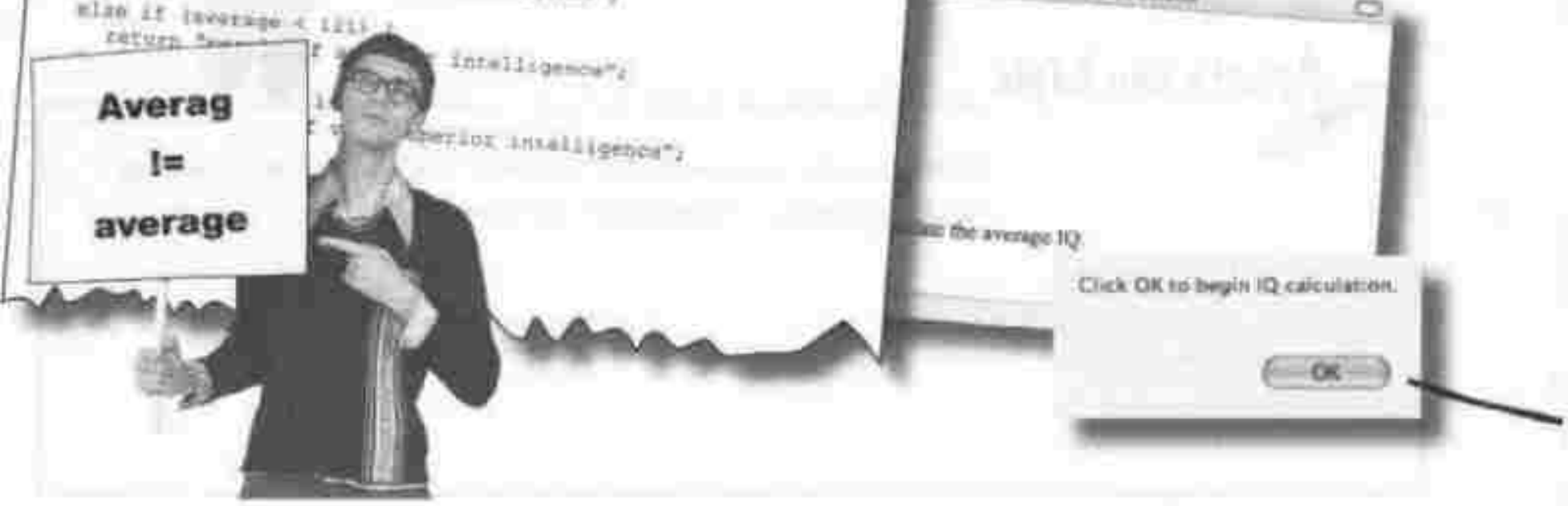
Algo tão simples quanto um erro tipográfico geralmente pode destruir um script.

```
function calcIQClass(data) {
  // Calcula o QI médio
  var average = 0;
  for (var i = 0; i < data.length; i++) {
    average += data[i];
  }
  average = Math.floor(average / data.length);

  // Retorna a classificação do QI médio
  if (average < 20) {
    return "people who should kill their tvs";
  }
  else if (average < 50) {
    return "people who should really hit the books";
  }
  else if (average < 70) {
    return "people who should hit the books";
  }
  else if (averag < 81) {
    return "people who should consider brain exercises";
  }
  else if (average < 91) {
    return "people who could be considered dull";
  }
  else if (average < 111) {
    return "people of average intelligence";
  }
  else if (average < 121) {
    return "[illegible] intelligence";
  }
  [illegible] superior intelligence";
```

Diagnosticar um erro tipográfico é a parte difícil... corrigi-lo é fácil.

Corrigir o nome da variável mal digitado resolve o problema da variável indefinida.

```
else if (average < 81) {
  return "people who should consider
    brain exercises";
}
```



Não existem Perguntas Idiotas

P: Há uma diferença entre "indefinido" e "não definido"?

R: Não. Significam exatamente a mesma coisa. Apenas alguns navegadores usam um termo e outros utilizam o outro. Considere os dois termos como sendo inteiramente alternativos.

P: Tudo bem, portanto há uma diferença entre "indefinido" e `null`?

R: Isso é um pouco mais capcioso. Sim, em um nível muito técnico, há uma diferença entre "indefinido" e `null`, mas não o suficiente para preocupar. Diferente de `null`, "indefinido" não é um valor que você deve considerar atribuir a uma variável. Há um tipo de dado `undefined` que as variáveis adotam automaticamente quando têm que ser atribuídas a um valor. Por outro lado, as variáveis nunca são definidas automaticamente a `null`. Porém, algumas vezes é uma boa idéia definir as variáveis do objeto para `null`, como uma etapa da inicialização para assegurar que está claro que um objeto não foi criado ainda.

Com exceção dos detalhes muito técnicos, o principal a saber sobre "indefinido" e `null` é que ambos convertem-se em `false` quando colocados em um contexto booleano como, por exemplo, a condição de teste de uma instrução `if`. É por isso que um código como `if` (algumObjeto) é geralmente usado para verificar se um objeto foi criado antes de tentar acessar seus membros.

P: Ainda não entendo bem como um erro tipográfico transformou, de algum modo, a variável `average` em indefinida. Como isso aconteceu?

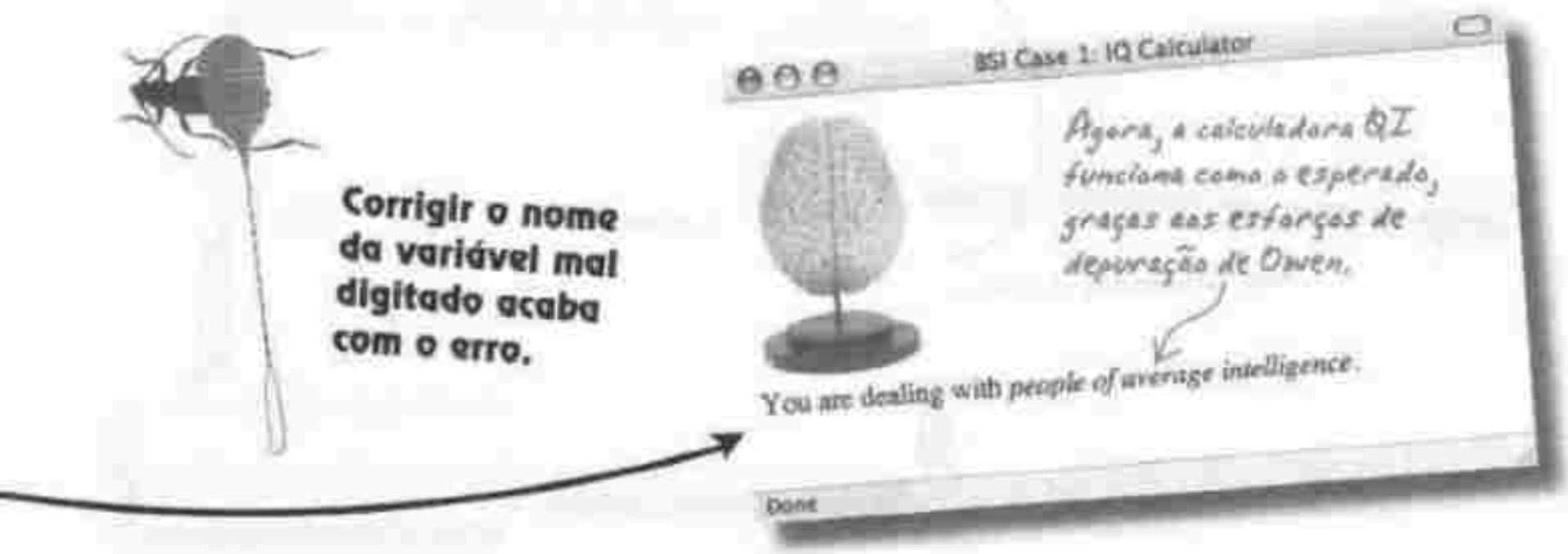
R: Mesmo que uma variável chamada `average` já tivesse sido criada e inicializada, o JavaScript não teria capacidade de fazer a conexão entre `averag` e `average` só porque elas têm **praticamente** o mesmo nome. A variável `averag` poderia facilmente ser nomeada como `shazbot` ou `lederhosen` no que diz respeito ao JavaScript. Isso quer dizer que o JavaScript a interpreta como uma variável inteiramente nova. E como essa nova variável ainda tem que ser atribuída a um valor, é um grande problema tentar comparar algo com ela em uma instrução `if`. É como tentar escrever uma crítica de cinema antes de ter decidido sobre qual filme assistir.

P: Você está brincando? Cometo erros tipográficos o tempo inteiro em meu processador de texto e ele não pára nada. Por que o JavaScript é tão sensível?

R: Respire fundo e coloque estas quatro palavras muito importantes e criteriosas na memória: ACOSTUME-SE COM ISTO. Não estamos escrevendo scripts para pessoas, estamos escrevendo para máquinas e as máquinas são qualquer coisa, menos bondosas, independentemente da linguagem na qual você está fazendo o script. Até um caractere no lugar errado pode irritar o script. Há alguma flexibilidade no espaço em branco em volta do código JavaScript como, por exemplo, os espaços e as novas linhas, mas o código em si deve ser muito exato.

Como esmiuçar os números da inteligência

Com o erro tipográfico sob controle, agora o script da calculadora QI funciona devidamente, ao calcular um QI médio a partir de um array e, então, exibindo o resultado como uma classificação de texto. Owen pode fechar este caso e gozar da glória de um trabalho bem feito... mas por quanto tempo?



Pontos de Bala

- Embora a maioria dos navegadores forneça um console de erro que oferece informações sobre os erros JavaScript, nem sempre eles podem ser confiados como sendo completamente precisos.
- Mesmo que os navegadores geralmente produzam informações incompletas do erro, normalmente fornecem **dicas** valiosas sobre onde começar a procurar o problema.
- As chaves em torno dos blocos de código são uma fonte comum de erros – tenha cuidado em sempre **coincidir** as chaves de abertura e de fechamento.
- Os erros tipográficos simples são fáceis de cometer, mas não necessariamente fáceis de encontrar – sempre verifique o nome dos identificadores.

O caso dos erros das chamadas da rádio

Owen mal teve tempo que comemorar seu primeiro caso fechado antes de outro caso chegar à sua mesa. Seu novo caso envolve um script que deve processar as chamadas para um concurso da estação de rádio, determinando um vencedor com base no número da chamada. O script deve manter uma contagem dos solicitantes e apenas declarar um vencedor depois de certo número de chamadas como, por exemplo, sete.

Como abrir a investigação

Antes de iniciar a página de chamadas da rádio em um navegador, Owen pensa que vale a pena dar uma olhada rápida no código (disponível em *http://www.headfirstlabs.com/books/hffs/*) e ter uma idéia sobre como é reunido. Talvez algo apareça como sendo obviamente errado, ou talvez ele pelo menos terá alguma compreensão de como o código **deve** funcionar.

O nome do solicitante e o número da chamada vencedora são transmitidos para a função checkWinner( ) junto com o objeto do formulário.

O contador de chamadas é inicializado em zero.

Avise o usuário e envie o formulário, caso o número da chamada seja igual ao número vencedor.

```
<html>
  <head>
    <title>Hot Case 2: Winning Caller</title>

    <script type="text/javascript">
      // O número total de chamadas
      var callNum = 0;

      function checkWinner(form, caller, winningNum) {
        // Aumente o número da chamada
        var callNum;
        ++callNum;

        // Verifique o vencedor
        if (callNum == winningNum) {
          alert(caller + ", caller number " + callNum + "... today's winner!");
          form.submit( );
        }
        else {
          // Redefina o campo caller para o próximo solicitante
          var callerField = document.getElementById('caller');
          callerField.value = "Next caller";
          callerField.focus( );
          callerField.select( );
        }
      }
    </script>
  </head>

  <body>
    <form name="callform" action="radiocall.php" method="POST">
      <img src="radio.png" alt="radio" />
      Caller name: <input id="caller" name="caller" type="text" />
      <input type="button" value="Call"
        onclick="checkWinner(this.form, document.getElementById('caller').value, 3)" />
    </form>
  </body>
</html>
```

Aumente o contador de chamadas.

Redefina o campo do nome do solicitante para o próximo solicitado, caso não haja um vencedor.

O método focus( ) define o foco de entrada para um elemento na página.

O método select( ) seleciona o valor armazenado em um elemento de texto.

Chame a função checkWinner( ) quando o botão Call for clicado.

O script do servidor para armazenar o solicitante vencedor.

Aponte seu Lápis

Ajude Owen a ter uma vantagem no caso circulando o número de erros que você acha que existem no código do script de chamadas da rádio.

Nenhum **Um** **Dois** **Três** **Quatro** **Cinco**

Aponte seu Lápis
Solução

Ajude Owen a ter uma vantagem no caso circulando o número de erros que você acha que existem no código do script de chamadas da rádio.

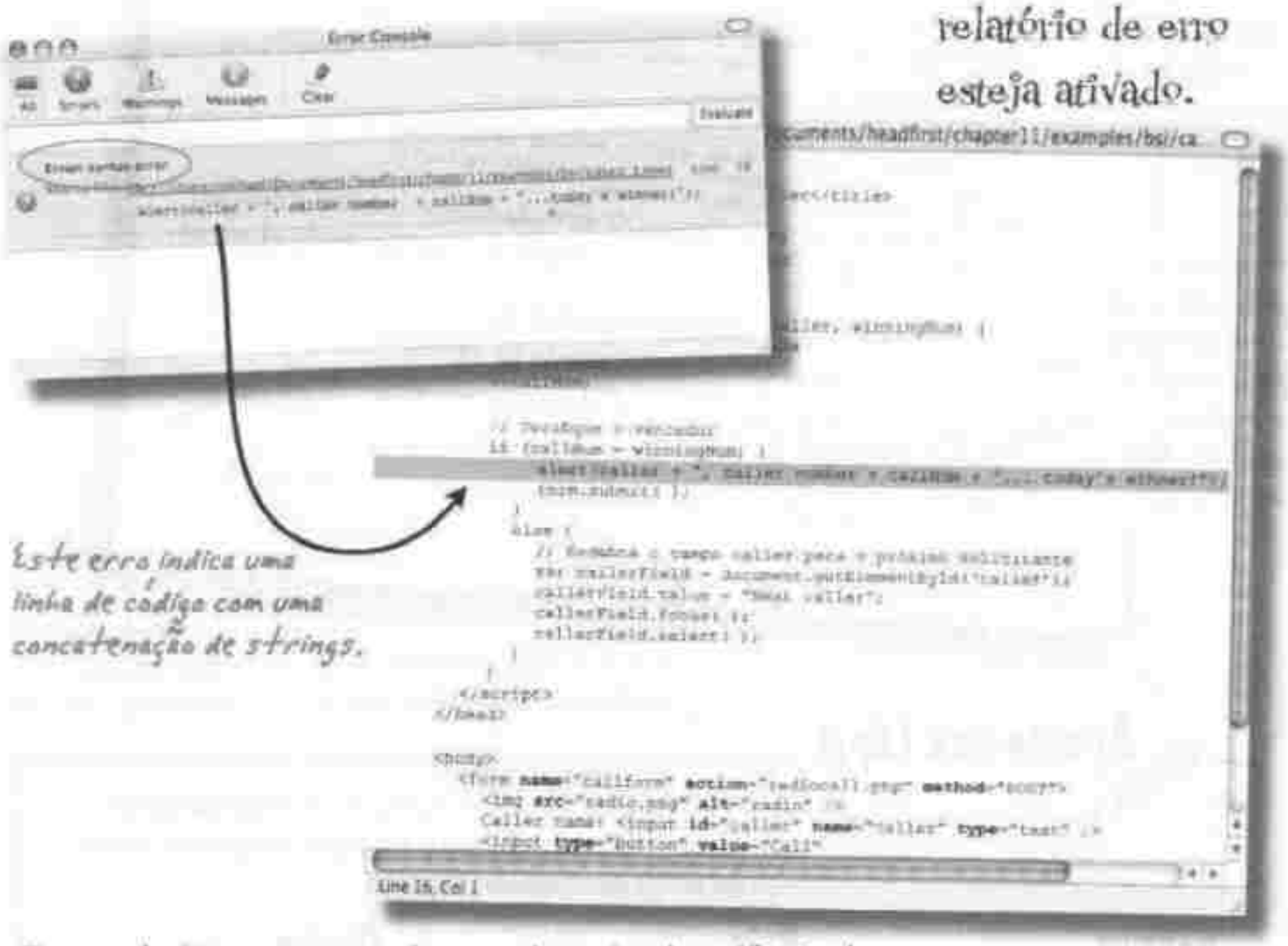
Nenhum Um Dois Três Quatro Cinco

Continuemos com o caso e ajudemos Owen a encontrar e resolver estes quatro erros...

Uma questão de validação da sintaxe (Erro #1)

Com uma idéia geral de como o código deve funcionar em mente, é hora de voltar ao Firefox e ver o que realmente acontece quando o script de chamadas da rádio é executado. De modo parecido com os outros erros vistos, o Firefox informa imediatamente um **erro de sintaxe**, que é um erro de codificação que viola as regras da linguagem JavaScript.

Os erros da sintaxe são sempre informados pelos navegadores, supondo que o relatório de erro esteja ativado.



Os erros de sintaxe sempre resultam em algum tipo de notificação do navegador, supondo que o navegador esteja definido para os erros do relatório. Isso fornece um começo muito importante ao controlar os erros.

Cuidado com essas coisas

O Firefox indicou uma linha de código com uma concatenação de strings, que é uma dica para analisar a linha de código com muito cuidado. O código chama a função `alert( )` com várias literais de string concatenadas com as variáveis `caller` e `callNum`.

```
if (callNum = winningNum)
  alert(caller + [redacted] + callNum + [redacted]);
```

Estas duas literais de string são concatenadas com duas variáveis.

É importante que as aspas sempre apareçam em pares, não é?

Colocar aspas em pares é fundamental no código JavaScript.

As aspas sempre devem aparecer em pares, do contrário o código JavaScript não seria capaz de dizer quando uma string termina e outra começa. No caso do código de chamadas da rádio, uma das literais da string na concatenação de strings não tem sua aspa posterior. Isto é definitivamente um erro de sintaxe, uma vez que confunde o JavaScript sobre onde a string termina.

Para corrigir o erro, simplesmente adicione a aspa que falta ao final da string:

```
if (callNum = winningNum)
  alert(caller + ", caller number " + callNum + "... today's winner!");
```

```
if (callNum = winningNum)
  alert(caller + ", caller number " + callNum + "... today's winner!");
```

Corrigir as aspas na string acaba com este erro.

Aspas, apóstrofos e consistência

As aspas que faltam são apenas metade da história ao controlar os erros relacionados a elas nas strings. Como o JavaScript e o HTML suportam igualmente aspas e apóstrofos para conter as strings (JavaScript) e os atributos (HTML), é fundamental ser consistente ao misturar os dois.

As aspas são usadas para conter todos os atributos HTML.

```
<input type="button" value="Call"
  onclick="checkwinner(this.form, document.getElementById('caller').value, 7)" />
```

Os apóstrofos são usados para conter as strings JavaScript dentro de um atributo.

Esta abordagem de usar aspas para os atributos HTML e apóstrofos para as strings JavaScript dentro dos atributos funciona perfeitamente bem e é uma boa idéia. Contudo, também é possível no HTML inverter os dois, como este código mostra:

O padrão moderno da página Web XHTML não permite que os apóstrofos sejam usados para conter atributos.

```
<input type='button' value='Call'
  onclick='checkwinner(this.form, document.getElementById("caller").value, 7)' />
```

Agora os apóstrofos são usados para os atributos HTML e as aspas são usadas para as strings JavaScript.

A idéia aqui é ficar com aspas para um tipo de código e apóstrofos para o outro. E como a versão moderna do HTML, o XHTML, requer aspas em torno dos atributos, faz sentido ficar com aspas em torno dos atributos e apóstrofos em torno das strings JavaScript dentro dos atributos.

As aspas e os apóstrofos devem ser alternados ao usar as strings JavaScript nos atributos HTML.

Mas surge um problema quando você precisa especificamente de uma aspa ou de um apóstrofo, mas já está comprometido com um deles como o caractere que contém a string ou o **limite** da string. Considere o seguinte código:

```
alert('It's so exciting!');
```

Este código funciona?

Poder do cérebro

O que acontece quando você precisa especificamente de um caractere de aspa ou apóstrofo em uma string que está contida usando o mesmo tipo de caractere?

Quando uma aspa não for uma aspa, use caracteres de escape

Os caracteres de escape são usados para especificar caracteres literais nas strings.

Um erro comum envolve usar uma aspa ou apóstrofo como um caractere dentro de uma string, mas sendo interpretado como um limite da string. Portanto, o código de alerta que acabamos de ver é um erro de sintaxe porque o interpretador JavaScript não pode descobrir quais apóstrofos são limites e quais são reais. Felizmente, há um modo fácil de declarar que um caractere é um caractere "real". É chamado de **caractere de escape** e envolve colocar uma barra invertida (\) na frente do caractere a ser usado literalmente.

```
alert('It's so exciting!');
```

Agora, o apóstrofo teve o escape aplicado e o JavaScript sabe, sem dúvida alguma, que realmente queremos que um caractere de apóstrofo na string, em oposição a declarar o final da string. Naturalmente, poderíamos ter evitado o escape mudando o limite da string para aspas.

```
alert("It's so exciting!");
```

O escape não é mais necessário.

Isso funciona bem, mas e este código:

```
alert("They said, "you've won!"");
```

A string contém aspas literais e um apóstrofo literal, portanto o escape é a única opção disponível. Em tal situação, é geralmente mais seguro aplicar o escape em todos os caracteres literais, mesmo que o apóstrofo possa passar sem ele.

```
alert("They said, \"you\'ve won!\"");
```

O escape não é necessário, mas ainda é uma boa idéia.

Exercício

Corrija as aspas e os apóstrofos nos seguintes fragmentos de código, usando os caracteres de escape sempre que possível.

```
var message = 'Hey, she's the winner!';
.......................................................................
var response = "She said, "I can't believe I won."";
.......................................................................
<input type="button" value="Winner" onclick="givePrize("Ruby");" />
.......................................................................
```

Solução do Exercício

Corrija as aspas e os apóstrofos nos seguintes fragmentos de código, usando os caracteres de escape sempre que possível.

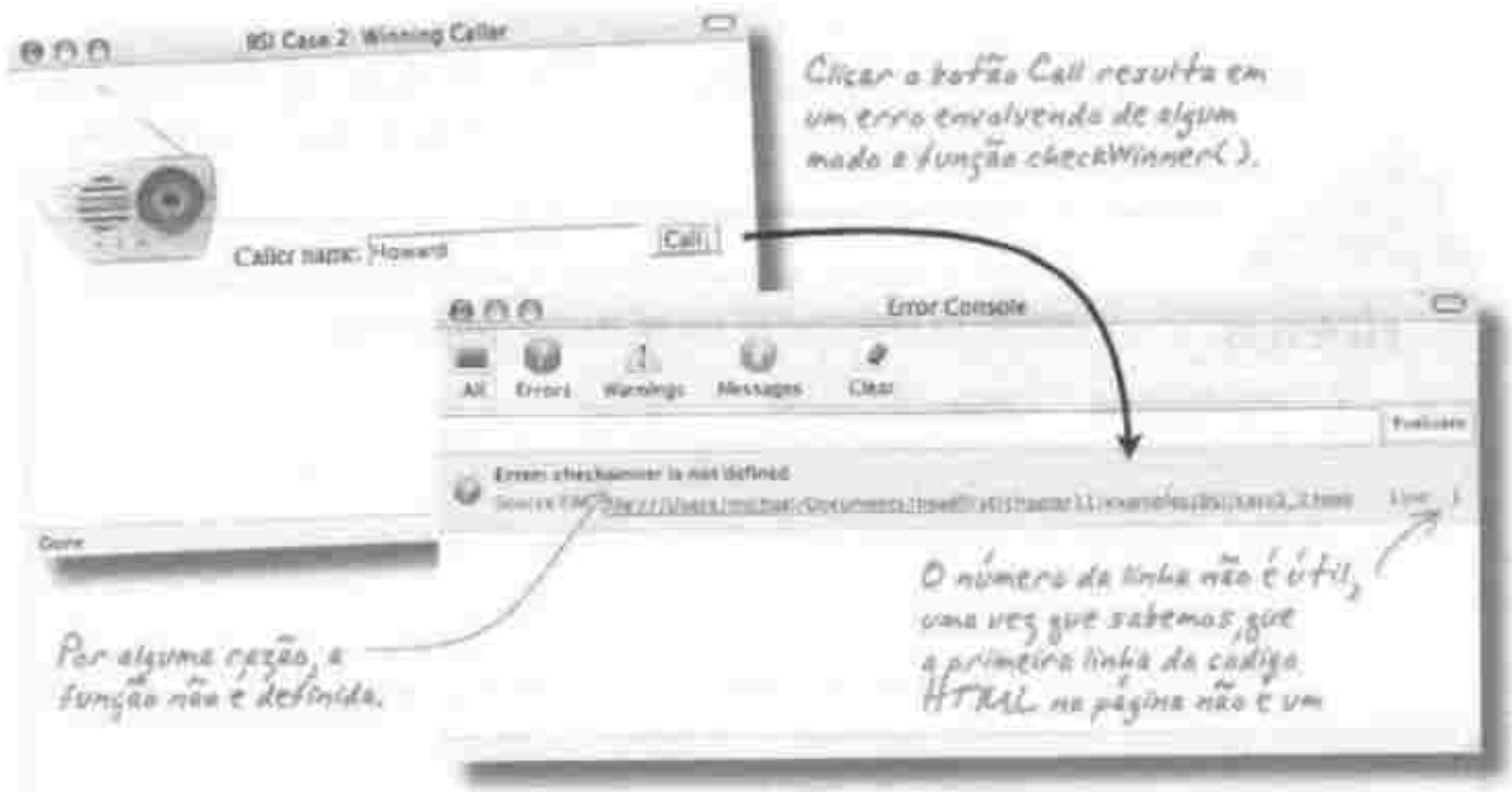
```
var message = 'Hey, she's the winner!';
var message = 'Hey, she\'s the winner!';

var response = "She said, "I can't believe I won.""
var response = "She said, \"I can\'t believe I won.\"";

<input type="button" value="Winner" onclick="givePrize("Ruby");" />
<input type="button" value="Winner" onclick="givePrize('Ruby');" />
```

O apóstrofo não tem que ter o escape aplicado, uma vez que aparece dentro de uma string cercada por aspas.

Os caracteres de escape não funcionarão neste caso, uma vez que é uma string JavaScript dentro de um atributo HTML. Misturar aspas e apóstrofos resolve o problema.

O indefinido não é apenas para as variáveis (Erro #2)

Um erro foi cuidado, mas Owen sabe que seu trabalho não terminou. Agora, o script de chamadas da rádio inicia bem sem erros, mas um clique no botão Call (Chamar) para fornecer um solicitante é tudo que é necessário para mostrar outro problema. E esse parece ter algo relacionado com a função `checkWinner( )`.



Suspeitas comuns: a lista de verificação

Com alguma experiência de depuração na manga, Owen decide executar sua lista de verificação recém-construída dos erros JavaScript comuns. Talvez esse erro coincida com um dos erros que ele já encontrou.

* Parênteses sem coincidência ou ausentes.
* Chaves sem coincidência ou ausentes.
* Identificador mal digitado com erro tipográfico.
* Aspa ou apóstrofo mal usado.

```
function checkWinner(form, caller, winningNum) {
    // Aumenta o número de chamada
    var callNum
    ++callNum;
    // Verifique um vencedor
    if (callNum == winningNum) {
      alert(caller + ", caller number " + callNum + "...
      form.submit();
    }
    else {
      // Redefina o campo caller para o próximo colocitante
      var callerField = document.getElementById("caller");
      callerField.value = "Next caller";
      callerField.focus();
      callerField.select();
    }
  }
  </script>
</head>
<body>
  <form name="callform" action="radiocall.php" method="POST">
    <img src="radio.png" alt="radio" />
    Caller name: <input id="caller" name="caller" type="text" />
    <input type="button" value="Call"
      onclick="checkWinner(this.form, document.getElementById("caller").value,
  </form>
</body>
</html>
```

A folha de cola para a detecção de erros de Owen.

A função checkWinner( ) é referida apenas duas vezes no código.

Hmmm...

Aponte seu Lápis

Ajude Owen a verificar o tipo de problema que você acha que está incomodando o script de chamadas da rádio.

- ☐ Chaves sem coincidência ou ausentes.
- ☐ Parênteses sem coincidência ou ausentes.
- ☐ Identificador mal digitado com erro tipográfico.
- ☐ Aspa ou apóstrofo mal usado.
- ☐ Algum tipo de problema inteiramente novo.

Aponte seu Lápis Solução

Ajude Owen a verificar o tipo de problema que você acha que está incomodando o script de chamadas da rádio.

- [ ] Chaves sem coincidência ou ausentes.
- [ ] Parêntesis sem coincidência ou ausentes.
- [x] Identificador mal digitado com erro tipográfico.
- [ ] Aspa ou apóstrofo mal usado.
- [ ] Algum tipo de problema inteiramente novo.

A função checkWinner() é chamada com w minúscula sem querer – checkwinner(). O erro tipográfico faz com que o JavaScript ache que checkwinner() é uma função inteiramente diferente que foi definida.

Corrigir o erro tipográfico do nome da função acaba com o Erro #2.

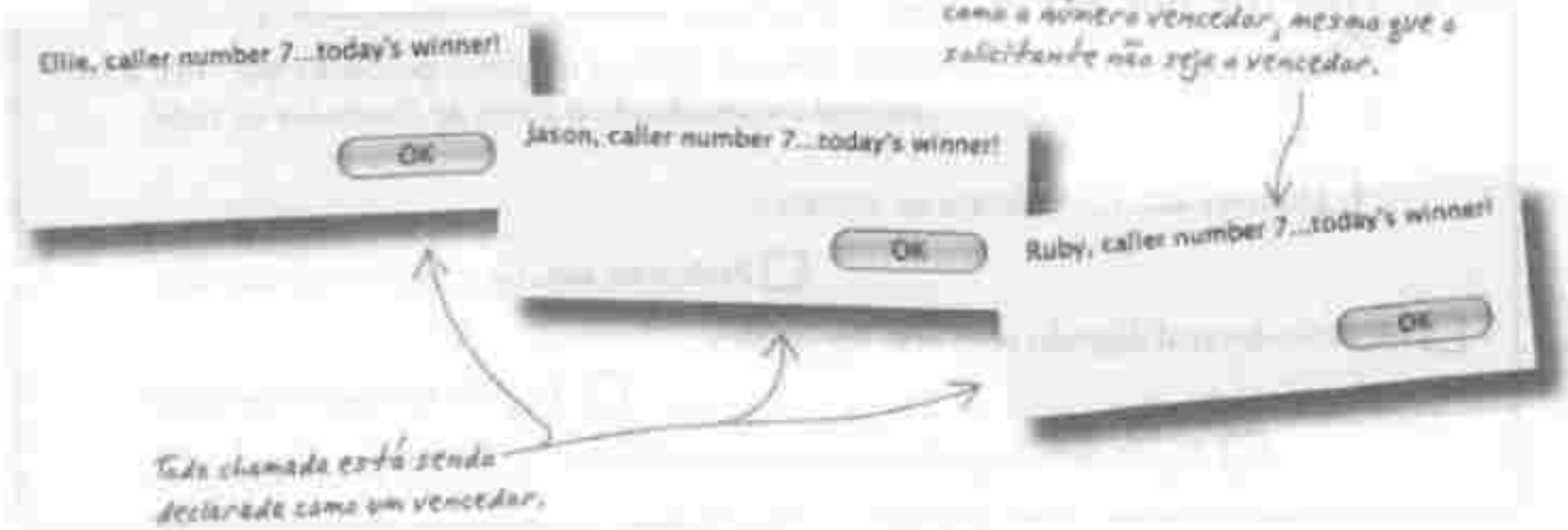
A correção simplesmente requer colocar W maiúsculo na chamada para checkWinner().

```
<input type="button" value="Call"
  onclick="checkWinner(this.form, document.getElementById('caller'). value, 7)" />
```

Todos somos vencedores (Erro #3)

Com o cuidado erro tipográfico "indefinido" e desagradável, o script de chamadas da rádio ainda está inicializando erros. A boa notícia é que o navegador não está mais informando nenhum problema. Mas o ruim é que **todo** solicitante agora é um vencedor – o script está declarando-os com o número correto do solicitante, mesmo quando não são. Serão muitos prêmios para entregar se Owen não sugerir uma correção!

A coisa realmente estranha é que o número da chamada está sendo mostrado como o número vencedor, mesmo que o solicitante não seja o vencedor.



Toda chamada está sendo declarada como um vencedor.

Depuração da caixa de alerta

Sabemos que o teste para o número vencedor ocorre comparando a variável `callNum` com o argumento `winningNum` para a função `checkWinner( )`. Mas este código parece bom... realmente precisamos de um modo de ver um pouco mais de perto o que está ocorrendo com a variável `callNum`.

```
...
  if (callNum == winningNum) {
...
```

Nada óbvio parece estar errado com este código.

Vale a pena tentar rastrear o valor de callNum para ver como ele está mudando a direção para este código.

Há um modo de ver o valor de uma variável em pontos diferentes em um script?

As caixas de alerta podem ser muito úteis para dar uma olhada rápida no valor de uma variável.

As caixas de alerta podem servir como uma janela de observação da depuração.

Como resultado, as caixas de alerta não são apenas para exibir informações pop-up para os usuários finais. Elas podem também ser úteis puramente no lado de desenvolvimento do código JavaScript como janelas de observação temporárias para ver as variáveis. Não só isso, mas os alertas podem ser usados para assegurar que certa seção do código está sendo chamada como o esperado. Neste caso, precisamos usar um alerta simplesmente para manter um olho na variável `callNum`.

A caixa de alerta fornece uma observação do valor da variável, neste caso o valor da variável callNum.

Como observar as variáveis com alerta

Uma **observação** é um termo de depuração que se refere a observar constantemente uma variável quando um programa é executado. Um alerta fornece uma observação primitiva que não é exatamente uma visão constante de uma variável, mas ainda pode ser muito útil. Um alerta pode ser usado como uma observação em qualquer lugar no código JavaScript, onde uma variável está em exame.

Algo não está certo... callNum deve ser definida para o número da chamada atual.

```
alert(callNum);
if (callNum = winningNum) {
  alert(caller + ", caller number " + callNum + "...
today's
winner!");
  form.submit();
}
```

Owen percebe que o script de chamadas da rádio está vendo `callNum` e `winningNum` como sendo iguais de algum modo, mesmo que `callNum` esteja aparecendo como `NaN`, logo antes da instrução `if`. Embora já seja confuso que `callNum` esteja aparecendo como `NaN`, ele decide mover o alerta para dentro da instrução `if` para ver se alguma coisa muda.

```
if (callNum = winningNum) {
  alert(callNum);
  alert(caller + ", caller number " + callNum + "... today's
winner!");
  form.submit();
}
```

A variável callNum agora está aparecendo definida para 7, logo depois da condição de teste if.

Poder do cérebro

Com a observação de alerta confirmando que `callNum` está sendo definida, de algum modo milagroso para 7 em uma única linha de código, qual você acha que é a causa do erro? O que Owen descobriu?

Depuração com alerta

Entrevista desta semana:

O alerta compartilha seu desprezo pelos erros

Use a Cabeça: Tenho que admitir, ouvi dizer que as coisas ficaram confusas para seu lado. As pessoas dizem que você pode ficar realmente chateado. Também ouvi que você poderia muito bem ser o melhor amigo de um depurador. Pode esclarecer quem é o alerta real?

Alerta: Estas pessoas estão loucas em primeiro lugar. Sou maravilhoso. Também sou muito simples – você me fornece algumas informações para exibir e boom, faço um movimento pop-up e exibo-as. É tudo. Onde está o problema?

Use a Cabeça: Acho que é a parte do "pop-up". Os pop-ups conseguiram uma reputação ruim nos últimos tempos com todos os anúncios ridículos que algumas vezes aparecem instantaneamente em todo lugar.

Alerta: Ah, entendi. Sim, posso ver onde isso poderia realmente ficar claro. Mas você não pode culpar um martelo pelo carpinteiro estúpido que não sabe como usá-lo. Entende o que quero dizer?

Use a Cabeça: Então, está me dizendo que qualquer coisa ruim que ouvi falar sobre você tem relação com você ser mal utilizado?

Alerta: É isso aí. Como disse, apenas faço o que pedem. Você diz para eu aparecer instantaneamente um monte de vezes com anúncios bobos, faço isso. Não estou dizendo que gosto, mas realmente não tenho escolha. Ei, achei que você fosse perguntar sobre minha contribuição para o mundo da depuração.

Use a Cabeça: Ah, desculpe-me. Sim, ouvi dizer coisas realmente boas sobre como você ajuda os desenvolvedores JavaScript a rastrear os erros em seu código. Como faz isso?

Alerta: É muito simples, realmente. Digamos que haja uma variável que ficou maluca, sendo definida para algum valor que não faz sentido. O programador está tendo uma experiência ruim, está supercafeinado, você sabe o que quero dizer, e precisa desesperadamente de um modo de dar uma olhada na variável em lugares diferentes no script para ver como ele está sendo alterado. Então, ele pede que eu apareça instantaneamente e mostre a variável.

Use a Cabeça: Mas como você é capaz de mostrar o valor alterado da variável em pontos diferentes em um script? Parece difícil.

Alerta: Absolutamente. Tudo que você faz é me chamar diversas vezes, com cada chamada em um ponto diferente no script.

Use a Cabeça: Entendo. Você alguma vez teve problemas ao ajudar como uma ferramenta de depuração?

Alerta: Bem, tenho que admitir que não sou tão bom ao exibir informações de depuração quando há uma parte do código sendo executada muitas vezes, como em um loop.

Use a Cabeça: Por quê?

Alerta: Bem, lembre-se de que sou uma janela pop-up, portanto tenho que ser clicado para desaparecer. Se eu estiver aparecendo muitas vezes, são muitos cliques.

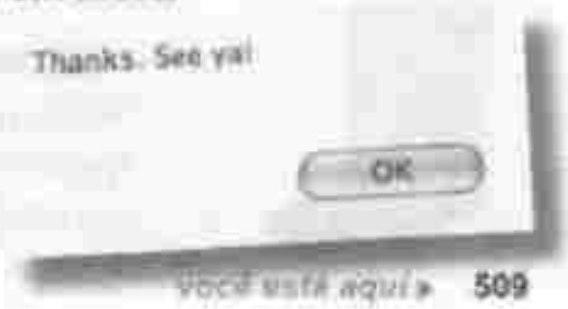
Use a Cabeça: Isso faz sentido. Também ouvi dizer que você pode ser útil mesmo quando não há dados para serem vistos.

Alerta: Ah, sim. Há muitas situações em que não ficam bem claro se ou quando uma parte do código está sendo chamada. Uma chamada rápida para mim dentro do código permitirá saber se o código está realmente sendo chamado. Sou transformado em um tipo de alarme apenas para permitir que você saiba se o código é chamado.

Use a Cabeça: Em todas essas situações de depuração, você está dizendo que você é apenas temporário?

Alerta: Ah, absolutamente. E não me importo. Não é como se eu não tivesse meu papel permanente também – apenas faço a depuração, da minha parte, como um pequeno serviço público.

Use a Cabeça: Bem, fico agradecido por seu tempo em explicar seu papel na detecção do erro. Espero vê-lo em breve.



Uma lógica ruim é legal, mas tem erros

Owen aperfeiçoou-se na **lógica do erro**, um erro que é perfeitamente legal segundo as regras do JavaScript, mas inteiramente errado em termos do que deve fazer. Neste caso, = é usado no lugar de = =, significando que `winningNum` está sendo **atribuído** a `callNum`, ao invés de ser **comparado** a ele. Sutil? Sim. Mas ainda altamente problemático.

O código que parecia bom anteriormente acabou tendo um erro sutil que foi difícil de descobrir.

```
...
if (callNum = winningNum) {
...
...
if (callNum == winningNum) {
...
```

O problema real com este erro é que ele não faz o navegador fracassar e gera um erro como os erros da sintaxe. O interpretador JavaScript não reclamou porque uma atribuição "retorna" o valor sendo atribuído, neste caso `winningNum`, que é, então, convertido automaticamente em `true` (não zero) na condição de teste `if`. Em outras palavras, o código é perfeitamente legal mesmo que não tenha feito o que queríamos.

Mudar = para = = destrói o Erro #3.

Os erros lógicos gostam de voar abaixo do radar.

O que torna os erros lógicos um problema para lidar é que eles geralmente não se mostram como erros da sintaxe. Embora um erro do script em um navegador possa parecer um pouco ridículo, realmente é uma sorte porque é um erro que foi detectado para você. Os erros lógicos não violam nenhuma regra da sintaxe JavaScript, portanto, geralmente, são mais difíceis de detectar.

Por outro lado, os erros lógicos algumas vezes resultam em um erro do script **enquanto** um script está em execução. Por exemplo, se um erro lógico resultar em uma variável não sendo inicializada, um erro "indefinido" aparecerá quando o script tentar referir-se à variável. Portanto, algumas vezes, um erro lógico poupa você do sofrimento de uma caçada cansativa ao erro.

Pontos de Bala

- Os erros da sintaxe envolvem o código que viola as regras da linguagem JavaScript e são geralmente capturados pelo interpretador JavaScript.
- As strings devem ficar cuidadosamente entre aspas ou apóstrofos coincidentes.
- As aspas e os apóstrofos devem ser misturados (mas ainda em pares coincidentes) nos atributos da sub-rotina de eventos HTML que contêm o código JavaScript.
- As caixas de alerta oferecem uma opção primitiva, porém útil para observar as variáveis em um script.
- É um erro comum codificar sem querer uma condição de teste com = quando você realmente deseja = =.

Não existem Perguntas Idiotas

P: Os caracteres de escape são apenas usados para aplicar escape nas aspas e nos apóstrofos?

R: Não, existem vários caracteres de escape suportados pelo JavaScript. Por exemplo, você pode usar `\t` para inserir uma tabulação em uma string. Do mesmo modo, uma nova linha é representada pelo caractere de escape `\n`. E uma barra invertida literal também deve ter o escape aplicado com `\\`.

Um lugar onde os caracteres de escape podem ser usados com eficiência é ao formatar o texto exibido em uma caixa de alerta. Você pode usar `\t` e `\n` para alinhar o texto usando tabulações e controlar como ele flui nas novas linhas.

P: Qual é o problema com o limite dos caracteres de escape nos atributos HTML?

R: O limite tem relação com o fato de que os atributos HTML não estão sujeitos às regras do JavaScript, pelo menos não quanto aos caracteres usados para limiar um valor do atributo. Portanto, embora seja bom aplicar o escape em um caractere dentro de uma string JavaScript que está contida em um atributo HTML, ele não pode ser o mesmo caractere usado para conter o atributo.

Se ainda estiver confuso, pense assim. O HTML vê um atributo simplesmente como um valor que deve aparecer entre aspas ou apóstrofos. Nada mais. Portanto, qualquer caractere de limite que você usar para iniciar o atributo, o HTML assumiria que o próximo encontrado será seu parceiro correspondente que fecha o valor do atributo. Isso acontece porque o HTML não processa o valor do atributo para os caracteres de escape do JavaScript.

Os caracteres de escape ainda funcionam dentro de um atributo HTML, contanto que não entrem em conflito com o caractere usado para conter o atributo. É porque o valor do atributo, conseqüentemente, é interpretado como o código JavaScript, supondo que estamos falando sobre um atributo da sub-rotina de eventos.

P: Não existem depuradores mais elegantes para o JavaScript que forneçam um controle detalhado sobre o processo de depuração?

R: Sim, existem alguns. E não é uma má idéia examiná-los mais e considerar experimentar um. Contudo, entenda que bons hábitos de codificação junto com as técnicas de depuração exploradas neste capítulo contribuirão muito em ajudar a criar scripts sem erro.

P: O que está acontecendo exatamente quando o código JavaScript tenta referir-se a uma variável ou função indefinida?

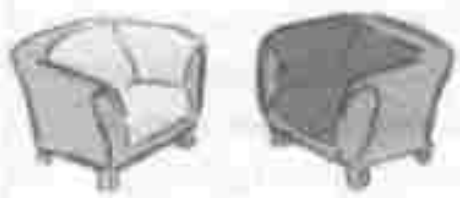
R: Lembre-se de que uma variável indefinida é uma variável que não foi criada ou que foi criada, mas não definida para nada. Em ambos os casos, o valor armazenado na variável é desconhecido, ou mais especificamente, é indefinido. Portanto, tentar ler esse valor e fazer algo significativo com ele não faz sentido, sendo por isso que o JavaScript gera um erro.

Uma situação parecida ocorre com as funções quando uma função é chamada, mas o interpretador JavaScript não pode encontrar uma função com esse nome. A função é indefinida, significando que chamá-la não faz nenhum sentido – não há nada a chamar. Novamente, o JavaScript considera-a um erro porque não há nenhuma maneira de executar de modo significativo o código.

P: Qual é o problema com a variável callNum tornando-se `NaN` antes da condição de teste `if` no script de chamadas da rádio?

R: Ainda não sabemos. Embora nos informe que algo ainda falta no código do script. Portanto, é importante continuar investigando mais erros...

Conversa informal



Conversa desta noite: O erro da sintaxe e o erro lógico compartilham seu amor pelo script ruim.

Erro da sintaxe:

Ei, ouvi falar de você. Ouvi que você é muito sorrateiro. Mas o que estou pensado é se você gosta realmente de um script mal escrito tanto quanto eu?

Discordo. Gosto de scripts que são uma destruição bem visivelmente. É onde eu sobressaio. Você espalha um pouco de mim em um script e tem a certeza de fazer um navegador sofrer.

Posso avaliar seu modo distorcido de ver as coisas, mas o problema é que você ainda permite que um script seja executado. Isso não é bom para mim. Gosto de pará-lo em seu caminho.

Seria, mas infelizmente somos muito limitados em termos de quanto dano podemos causar. Certamente, é divertido bagunçar uma página e impedir que ela funcione, mas detesto não poder ter acesso a nada mais. Cara, que diversão eu poderia ter com um disco rígido cheio de dados importantes!

Erro lógico:

Ah, sim. Não há nada melhor do que um script que parece bom na superfície, mas fora da vista estão todos os tipos de problemas estranhos.

Mas qual é a diversão? Todos sabem que um ataque furtivo é muito mais eficiente. Você sabe, acalmar fazendo pensar que tudo está bem e, então, começar a mostrar lentamente probleminhas aqui e ali. Se você fizer um trabalho bom o suficiente, as pessoas estarão perguntando se seu navegador está ou não funcionando corretamente.

Você conseguiu. É uma vergonha eu não ter descoberto um bom modo de me mostrar parando um script como você faz. Ou melhor ainda, danificar o navegador inteiro em uma grande nuvem de fumaça. Isso seria legal!

Ih cara, isso seria incrível. Você está certo de que não há um ponto para entrarmos lá?

Erro da sintaxe:

Não, o interpretador JavaScript bloqueou-nos com muita firmeza.

Não, como funciona?

Como você consegue fazer isso?

Tenho um muito bom que é onde as pessoas esquecem de terminar cada instrução JavaScript com um ponto-e-vírgula. É ótimo, pois o interpretador deixará passar se as instruções estiverem em linhas separadas. Mas, finalmente, algum programador arrogante tentará "otimizar" o código e combinará as instruções em uma única linha, e é quando eu apareço. Esse nunca fica velho – sempre consigo rir!

Adoro porque se o interpretador notar, então tenho que entrar em ação e chocá-lo com um erro. Ei, estou começando a perceber que devemos pensar em nos juntar – acho que poderíamos causar muito mais danos assim.

Estou logo atrás de você!

Erro lógico:

Bem, há ainda muita diversão. Eu falei sobre meu pequeno truque com = e = =?

É ótimo. O programador deseja digitar = = para comparar dois valores, mas sem querer digita = e faz uma atribuição. É hilário porque ficará horas sem ver o problema. E o interpretador JavaScript não é nada inteligente porque o código ainda é legal tecnicamente.

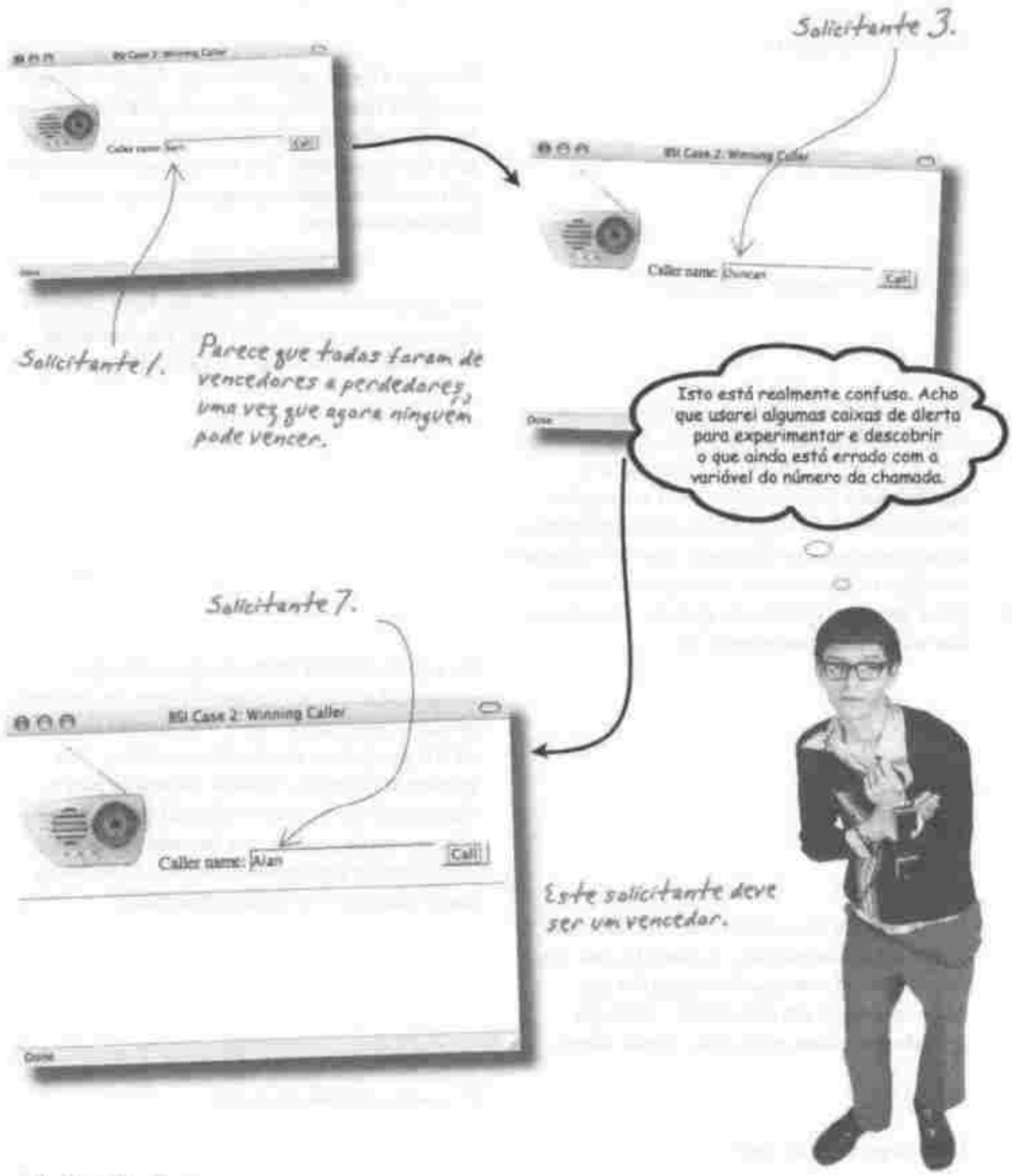
Ah, tenho muitas maneiras. É um bom caminho que percorro estando dentro da lei, mas ainda sendo capaz de levantar o problema.

Ei, isso me lembra mais um que tenho que compartilhar. Adoro quando as pessoas decidem mudar os argumentos para uma função depois de terem escrito a função. Nunca falha – elas esquecem de mudar todas as chamadas para a função, que devem ser atualizadas para coincidir com os novos argumentos. Se tudo der certo, o interpretador não notará e elas terão resultados inesperados com os argumentos ruins.

Concordo. Vamos começar!

Todos somos perdedores! (Erro #4)

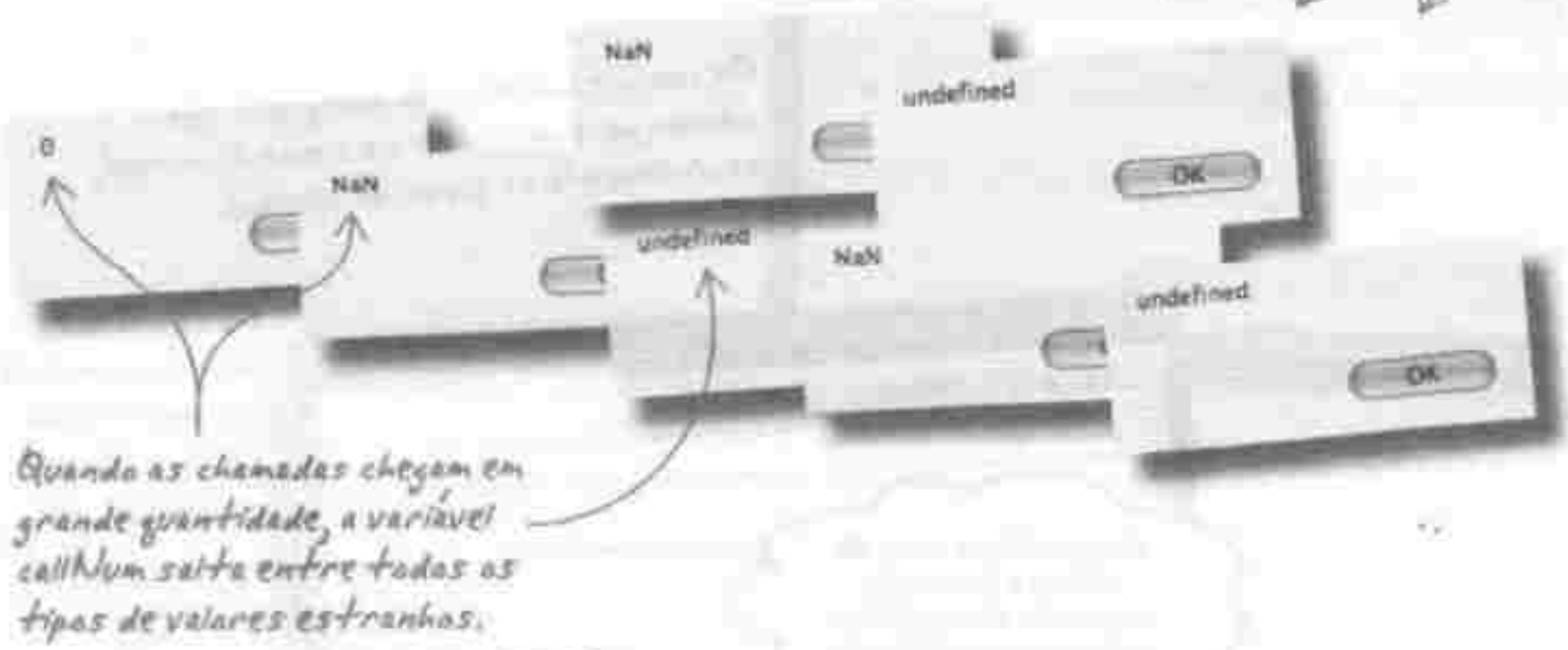
Owen está começando a perceber que este negócio de depuração não é tão fácil quanto pensava. Com o erro lógico da instrução `if` corrigido, agora o script nunca declara um vencedor. Portanto, fomos de todos sendo um vencedor para todos, agora, sendo um perdedor. A auto-estima de algumas pessoas definitivamente será afetada por esse erro se Owen não o corrigir rapidamente.



Dominado por alertas chatos

Owen tenta usar alertas para colocar uma observação em uma variável `callNum` e tentar descobrir o que está acontecendo. Porém, ele tem o problema de ficar entediado tentando prosseguir através de muitas caixas de alerta até o sétimo solicitante. Ele tentou usar vários alertas em diferentes partes do código, mas apenas conseguiu ficar sobrecarregado com tantas janelas pop-up inúteis e não sabe por onde começar...

A desvantagem dos alertas como janelas de observação é que eles podem ficar chatos quando usados no código repetitivo.

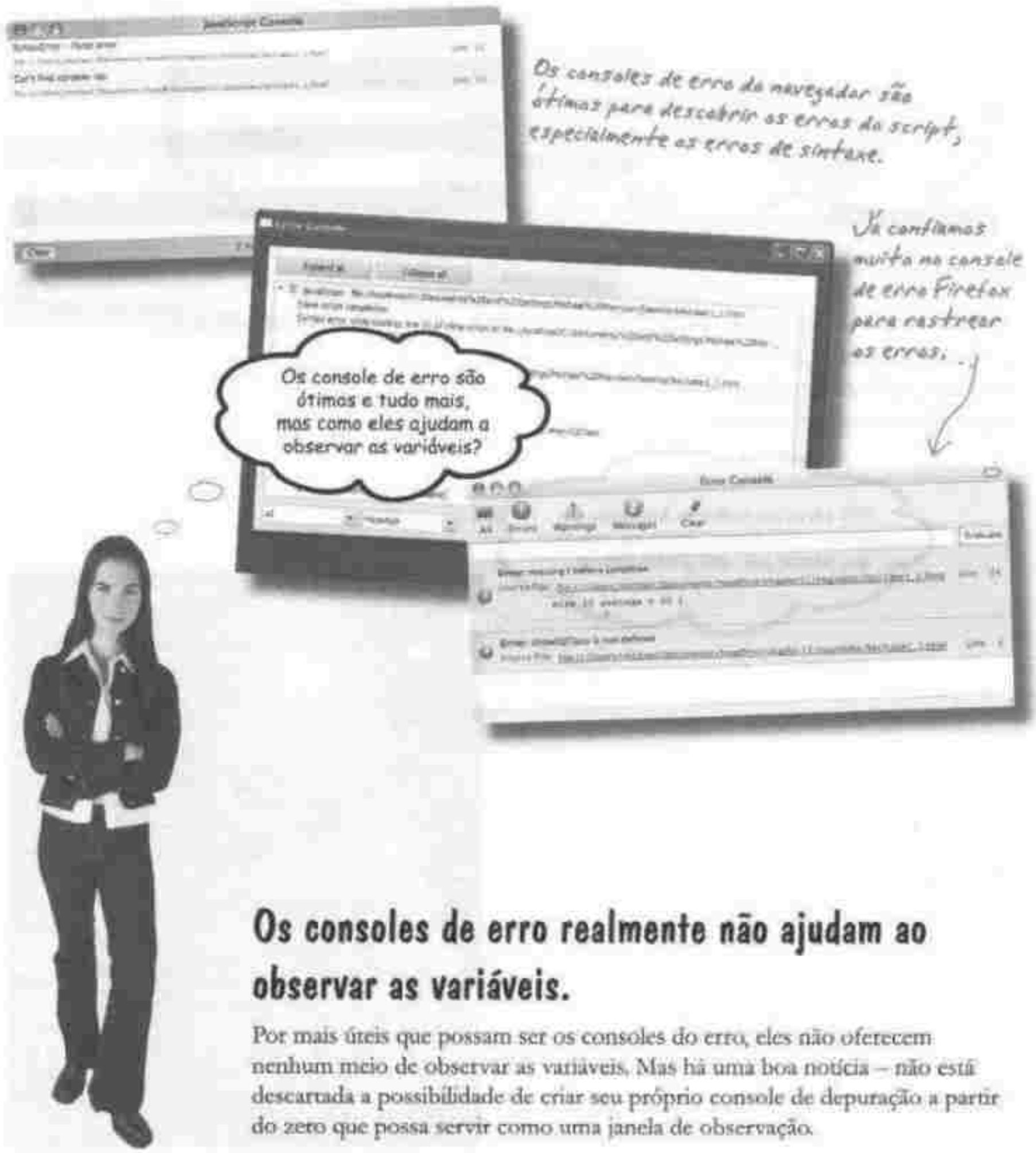


Quando as chamadas chegam em grande quantidade, a variável callNum salta entre todos os tipos de valores estranhos.

Não seria um sonho se houvesse um modo de observar as variáveis sem ter que passar por uma janela pop-up...

Os consoles de depuração do navegador podem ajudar

A maioria dos navegadores tem um console de depuração, na realidade um console de **erro**, que é responsável por exibir os erros que ocorrem em um script. Os consoles do erro são muito úteis para descobrir quando algo está errado em um script e, em muitos casos, ajudar a diagnosticar exatamente o que deu errado. O Firefox, em particular, mostrou ter um excelente console de erro.

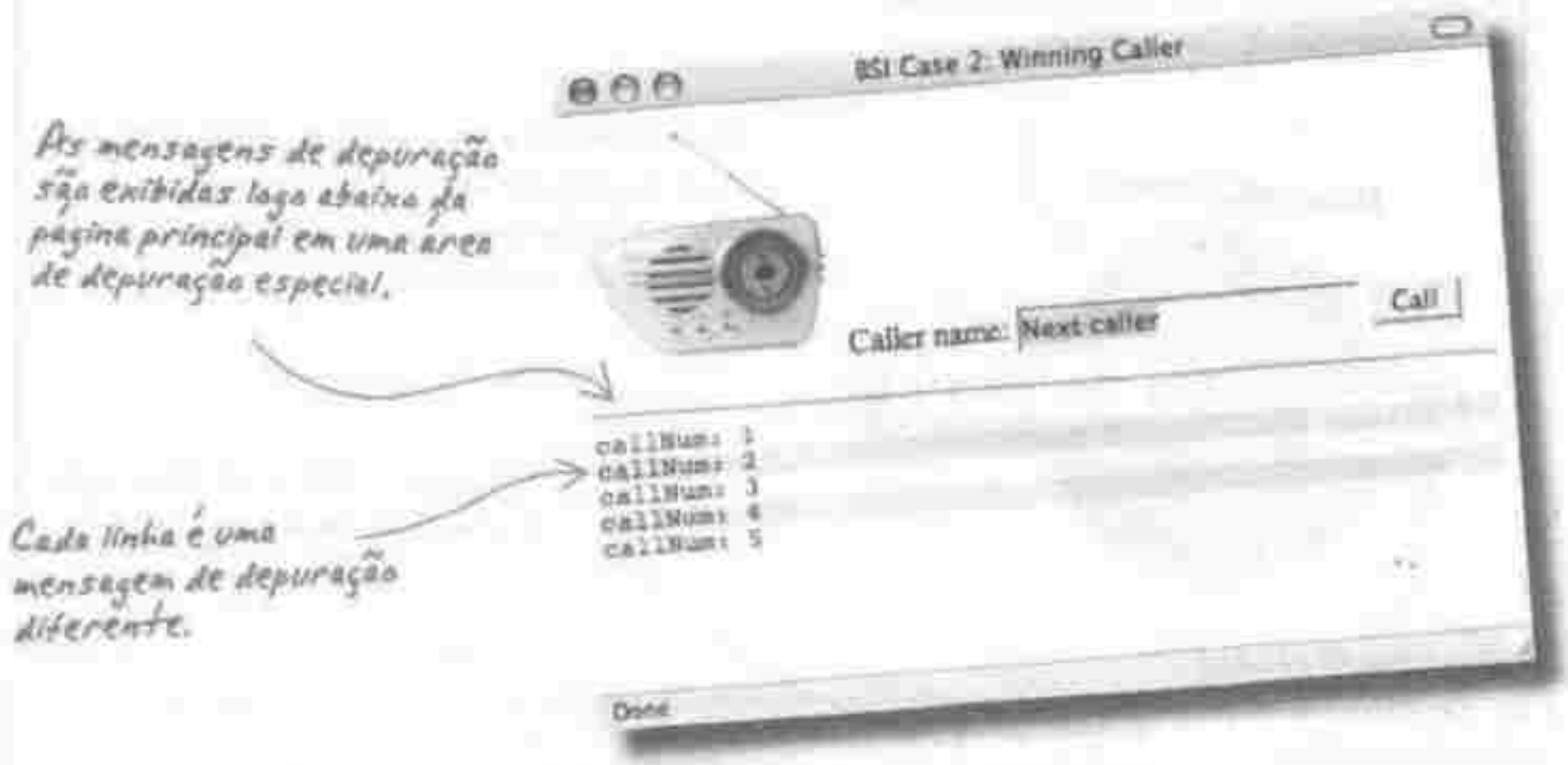


Os consoles de erro realmente não ajudam ao observar as variáveis.

Por mais úteis que possam ser os consoles do erro, eles não oferecem nenhum meio de observar as variáveis. Mas há uma boa notícia – não está descartada a possibilidade de criar seu próprio console de depuração a partir do zero que possa servir como uma janela de observação.

Construa um console personalizado para a depuração

A idéia de um console de depuração personalizado poderia parecer intimidadora à primeira vista, mas realmente tudo o que tem que fazer é exibir o texto quando solicitado. O segredo é que o console deve exibir as informações da depuração diretamente na página, não em uma caixa de alerta. Uma janela pop-up separada poderia ser usada, contanto que não requeira que o usuário clique em OK, mas é mais simples e eficiente mostrar as mensagens de depuração diretamente na página.



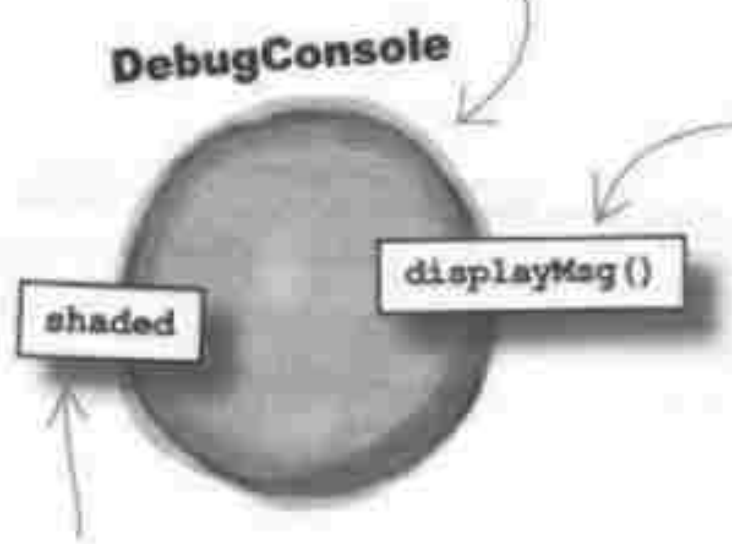
Aponte seu Lápis

Imagine o design de um console de depuração JavaScript que permita a Owen exibir as mensagens de depuração em uma lista dentro de uma área criada dinamicamente na página. Desenhe o que você acha que são os componentes necessários deste design e como se encaixam, incluindo um objeto JavaScript personalizado para o console de depuração.

Aponte seu Lápis
Solução

Imagine o design de um console de depuração JavaScript que permita a Owen exibir as mensagens de depuração em uma lista dentro de uma área criada dinamicamente na página. Desenhe o que você acha que são os componentes necessários deste design e como se encaixam, incluindo um objeto JavaScript personalizado para o console de depuração.

O console de depuração é desenhado como um objeto denominado DebugConsole com uma propriedade e um método.



Cada chamada para o método displayMsg() personalizado desenha uma mensagem de depuração em uma nova linha da área do console de depuração.

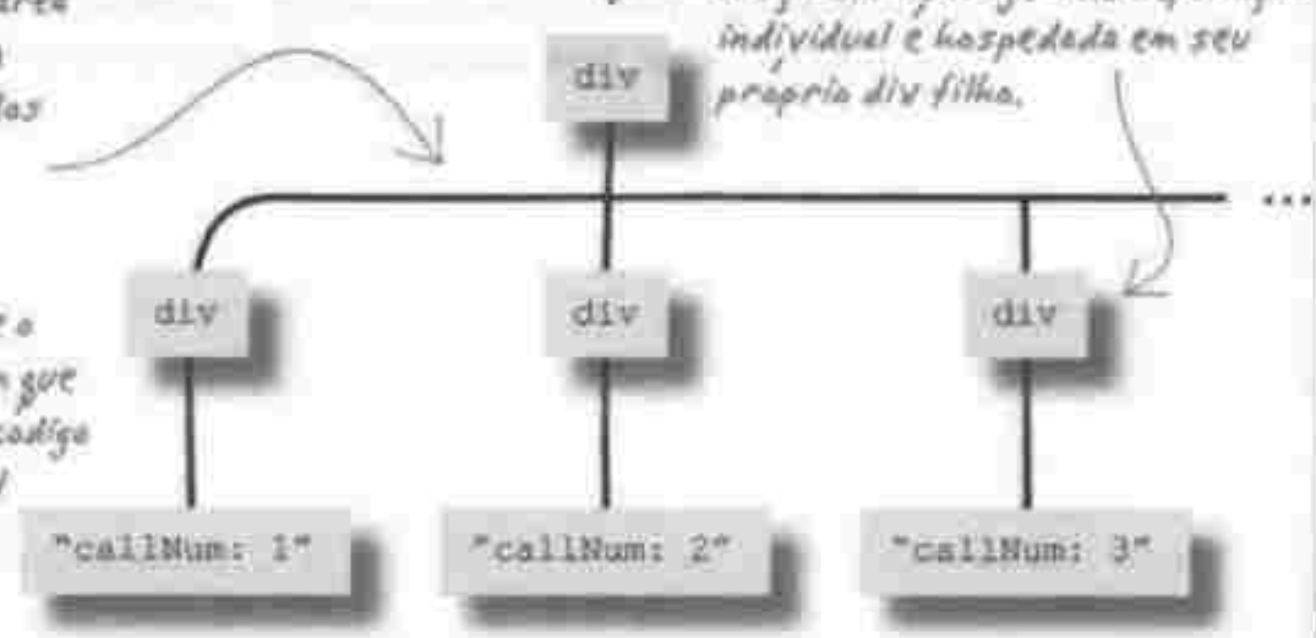
O próprio console de depuração é criado na página como div.

A propriedade shaded é um booleano que muda entre true e false para cada mensagem de depuração para alternar a cor do segundo plano.

```
callNum: 1
callNum: 2
callNum: 3
callNum: 4
callNum: 5
```

Os elementos HTML para a área de depuração na página são criados dinamicamente pelo console de depuração, significando que o usuário não tem que incluir nenhum código HTML especial para suportar o console de depuração.

Dentro do console de depuração div, cada mensagem de depuração individual é hospedada em seu próprio div filho.



Ímãs de geladeira JavaScript

O código para o console de depuração não tem algumas partes. Preencha as lacunas com os ímãs do código para terminar de construir o objeto `DebugConsole`.

```
function DebugConsole() {
  // Crie a área do console de depuração

  var consoleElem = document. ......................(.............);

  consoleElem.id = "debug";
  consoleElem.style.fontFamily = "monospace";
  consoleElem.style.color = "#333333";

  document.body. ..................(consoleElem);

  consoleElem. ..................(document. ......................("hr"));

  // Crie a propriedade da cor do segundo plano alternativa

  this. ............ = false;

}

DebugConsole.prototype.displayMsg = function(msg) {
  // Crie a mensagem
  var msgElement = document.createElement("div");

  msgElement.appendChild(document. ......................... (msg));

  msgElement.style.backgroundColor = this.shaded ? "#EEEEEE" : "#FFFFFF";

  var consoleElem = document.getElementById(..............);

  consoleElem.appendChild(.................);

  // Mude a propriedade da cor do segundo plano alternativa

  this.shaded = ...... this.shaded;

}
```

"div"

msgElement

"debug"

!

shaded

appendChild

createElement

createTextNode

Solução dos ímãs de geladeira JavaScript

O código para o console de depuração não tem algumas partes. Preencha as lacunas com os ímãs do código para terminar de construir o objeto `DebugConsole`.

```
function DebugConsole( ) {
  // Crie a área do console de depuração
  var consoleElem = document.[createElement].("div");

  consoleElem.id = "debug";
  consoleElem.style.fontFamily = "monospace";
  consoleElem.style.color = "#333333";

  document.body.[appendChild](consoleElem);

  consoleElem.[appendChild](document.[createElement]("hr"));

  // Crie a propriedade da cor do segundo plano alternativa

  this.[shaded] = false;

}

DebugConsole.prototype.displayMsg = function(msg) {
  // Crie a mensagem
  var msgElement = document.createElement("div");

  msgElement.appendChild(document.[createTextNode](msg));

  msgElement.style.backgroundColor = this.shaded ? "#EEEEEE" : "#FFFFFF";

  var consoleElem = document.getElementById("debug");

  consoleElem.appendChild(msgElement);

  // Mude a propriedade da cor do segundo plano alternativa

  this.shaded = [!] this.shaded;

}
```

O console de depuração div é anexado ao corpo do documento, o que significa que é adicionado ao final da página.

O primeiro elemento-filho no console de depuração é uma régua horizontal para separar as mensagens do console do resto da página.

A cor do segundo plano inicia como false, que resulta em uma cor branca inicialmente...

Uma mensagem é adicionada ao console de depuração como um div filho.

A cor do segundo plano alterna entre cada mensagem para que elas sejam mais fáceis de ler.

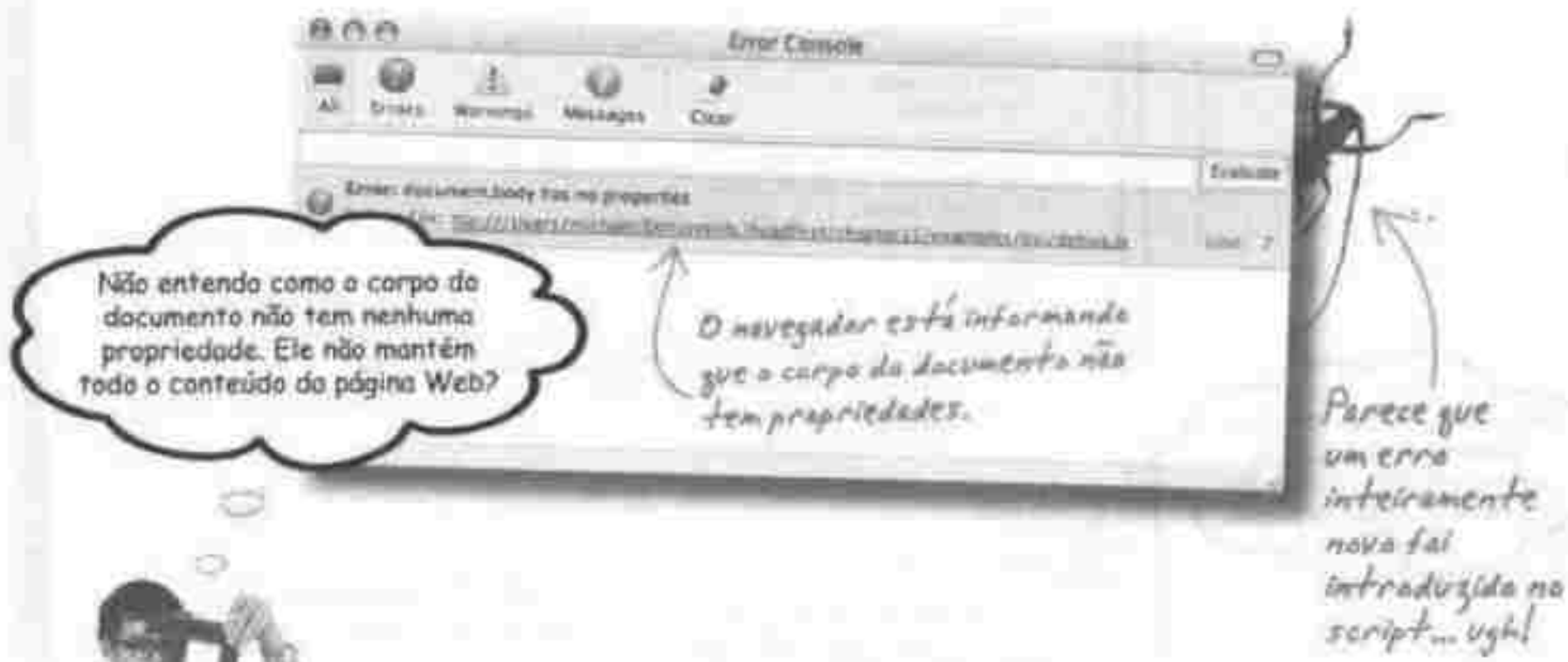
Depure seu depurador

Owen mal pode esperar para colocar à prova o novo console de depuração para descobrir o que ainda está errado com o script de chamadas da rádio. Portanto, ele importa o arquivo **debug.js** para a página e cria o objeto `DebugConsole` no início da página.

```
<script type="text/javascript">
  // Depure a variável global do console
  var console = new DebugConsole( );
  ...
```

Este código cria o objeto DebugConsole como uma variável global no início da página.

Infelizmente, as coisas não ocorrem como o planejado. Quando tenta pela primeira vez usar o console de depuração, Owen descobre que ele agravou seus problemas introduzindo um erro inteiramente novo no console de depuração novo.



A linha do código que gera o erro está apenas tentando anexar um nó-filho (`div`) ao corpo do documento, o que não deve ser um problema.

```
document.body.appendChild(console);
```

Algo mais está faltando, embora definitivamente pareça ter alguma relação com o novo objeto `DebugConsole`.

Poder do cérebro

O que poderia estar possivelmente errado com este código para fazer com que o corpo do documento acabe de algum modo vazio?

Como aguardar a página

O problema com o console de depuração tem relação com o tempo de como uma página é carregada e quando o código do script tem acesso ao corpo da página.

Os elementos HTML que residem atualmente na página não são carregados até o corpo ser... depois da cabeça.

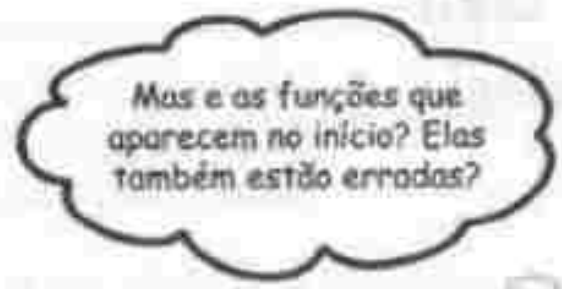
A cabeça da página é carregada antes do corpo, portanto nenhum conteúdo do corpo está disponível neste ponto.

```
<html>
  <head>
    <title>BBC Case 2: Winning Caller</title>

    <script type="text/javascript" src="debug.js"></script>

    <script type="text/javascript">
      // Variável global do console de depuração
      var console = new DebugConsole();

      // O número total de chamadas
      var callNum = 0;

      function checkWinner(form, caller, winningNum) {
        // Aumente o número da chamada
        var callNum;
        ++callNum;

        // Verifique o vencedor
        if (callNum == winningNum) {
          alert(caller + ", caller number " + callNum + "... today\'s winner!");
          form.submit();
        }
        else {
          // Redefina o campo caller para o próximo solicitante
          var callerField = document.getElementById('caller');
          callerField.value = 'Next caller';
          callerField.focus();
          callerField.select();
        }
      }
    </script>
  </head>

  <body>
    <form name="callform" action="radiocall.php" method="post">
      <img src="radio.png" alt="radio" />
      Caller name: <input id="caller" name="caller" type="text" />
      <input type="button" value="Call"
        onclick="checkWinner(this.form, document.getElementById('caller').value, 7)" />
    </form>
  </body>
</html>
```

Ah, então o código do script **executado** no início da página não pode acessar os elementos HTML na página.

O código JavaScript executado no início da página não tem acesso ao conteúdo da página Web.

Como o início de uma página é carregado antes do corpo, qualquer código executado diretamente na cabeça da página deve ter cuidado para não tentar acessar nenhum elemento HTML que esteja no corpo da página. Isso pode parecer uma restrição estranha, mas faz sentido quando você considera que grande parte do código não é geralmente **executada** na cabeça da página.



Nem todo o código no início de uma página é executado na cabeça da página.

Colocar o código em uma função que aparece no início de uma página não é o mesmo que **executar** o código no início da página – o código da função não é executado até que a função seja chamada. Mas o código que é colocado fora de uma função é executado imediatamente quando o cabeçalho é carregado. Este é o código que pode causar problemas.

No caso do objeto `DebugConsole`, ele não pode ser criado diretamente no início da página porque seu construtor depende muito do conteúdo no corpo da página.

Aponte seu Lápis

Escreva quando e onde você acha que o objeto `DebugConsole` deve ser criado para assegurar que pode acessar com segurança os elementos na página.

.....

.....

.....

.....

.....

Aponte seu Lápis
Solução

Escreva quando e onde você acha que o objeto `DebugConsole` deve ser criado para assegurar que pode acessar com segurança os elementos na página.

O navegador permite saber quando uma página acabou de carregar inicializando o evento onload. Portanto, o objeto DebugConsole deve ser criado em resposta ao evento onload. Porém, a variável do console deve ainda ser declarada na cabeça da página para que o objeto seja global – simplesmente não chamamos o construtor para, de fato, criar o objeto até o evento onload ser inicializado.

```
<body onload="console = new DebugConsole( );">
```

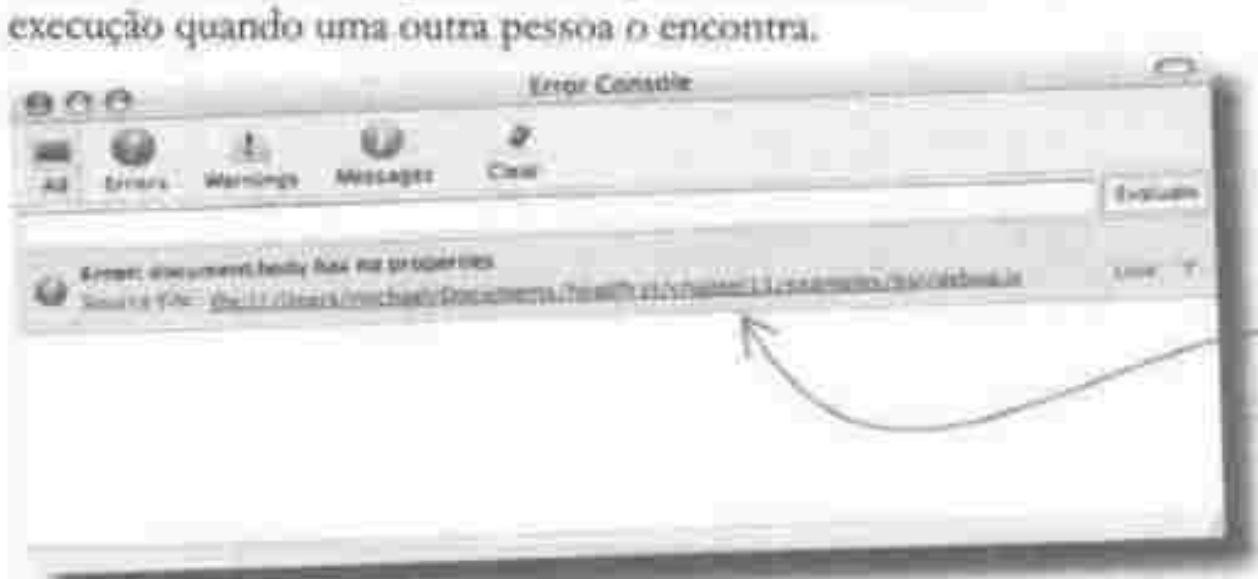
O objeto DebugConsole agora é construído em resposta ao evento onload.

Mover a criação do objeto DebugConsole matou o erro do console de depuração.

Os erros mais desagradáveis de todos: da execução

O problema do corpo do documento descarregado é um exemplo de **erro da execução** – um erro que mostra sua face apenas em certas condições enquanto um script está sendo realmente executado. Algumas vezes, os erros da execução vêm à tona apenas sob circunstâncias muito específicas como, por exemplo, certo tipo de entrada do usuário ou certo número de iterações do loop ocorrendo. Os erros da execução são, em geral, os mais difíceis de todos para encontrar porque são muito complicados de prever. Algumas vezes, é um desafio simplesmente reproduzir um erro da execução quando uma outra pessoa o encontra.

Os erros da execução ocorrem apenas por causa de condições específicas que ocorrem enquanto um script é executado.



O erro do console de depuração era um erro de execução causado por tentar acessar os dados antes deles terem sido carregados, que é um problema mostrado apenas quando um script é executado.

A tríade do erro JavaScript

Junto com os erros da execução, os dois outros erros que vimos anteriormente completam a tríade do erro JavaScript: erros de sintaxe, erros lógicos e erros da execução. Qualquer um deles é capaz de se manifestar em qualquer script, geralmente, ao mesmo tempo! Compreender suas diferenças é uma parte importante de ser capaz de encontrá-los e erradicá-los com sucesso.

Erro da execução

Um erro revelado apenas pelas condições da execução como, por exemplo, o usuário fornecendo certo tipo de dados em um formulário com o qual o script não pode lidar ou tentando acessar um objeto antes dele ter sido criado ou inicializado.

Erro da sintaxe

Um erro resultante de uma violação das regras da linguagem JavaScript, significando que o código é inadequado para ser executado no interpretador JavaScript.

Erro lógico

Um erro causado pela lógica ruim, geralmente envolvendo o código que deve fazer uma coisa, mas sem querer é codificado para fazer alguma outra coisa. Algum código com erros lógicos é executado exatamente como o pretendido, neste caso o programador compreendeu mal a tarefa com a qual começar.

```
<html>
  <head>
    <title>Ex: Case 3: Winning Caller</title>

    <script type="text/javascript" src="debug.js"></script>

    <script type="text/javascript">
      // Variável global do console de depuração
      var console = new DebugConsole();

      // O número total de chamadas
      var callNum = 0;

      function checkWinner(form, caller, winningNum) {
        // Aumente o número da chamada
        var callNum
        ++callNum;

        console.displayMsg("callNum: " + callNum);

        // Verifique os vencedores
        if (callNum == winningNum) {
          alert(caller + ", caller number " + callNum + "... today's winner!");
          form.submit();
        }
        else {
```

Exercício

Escreva o tipo de erro para cada uma das seguintes descrições do erro.

Parêntesis faltando em torno da condição de teste de uma instrução if. .....

Esquecer de inicializar uma variável do contador para 0. .....

Criar um loop que faz loops além do último elemento em um array. .....

Esquecer de fechar uma função com uma chave de fechamento. .....

Solução do Exercício

Escreva o tipo de erro para cada uma das seguintes descrições do erro.

Parêntesis faltando em torno da condição de teste de uma instrução if. ....Sintaxe....

Esquecer de inicializar uma variável do contador para 0. ....Lógico....

Criar um loop que faz loops além do último elemento em um array. ....Execução....

Esquecer de fechar uma função com uma chave de fechamento. ....Sintaxe....

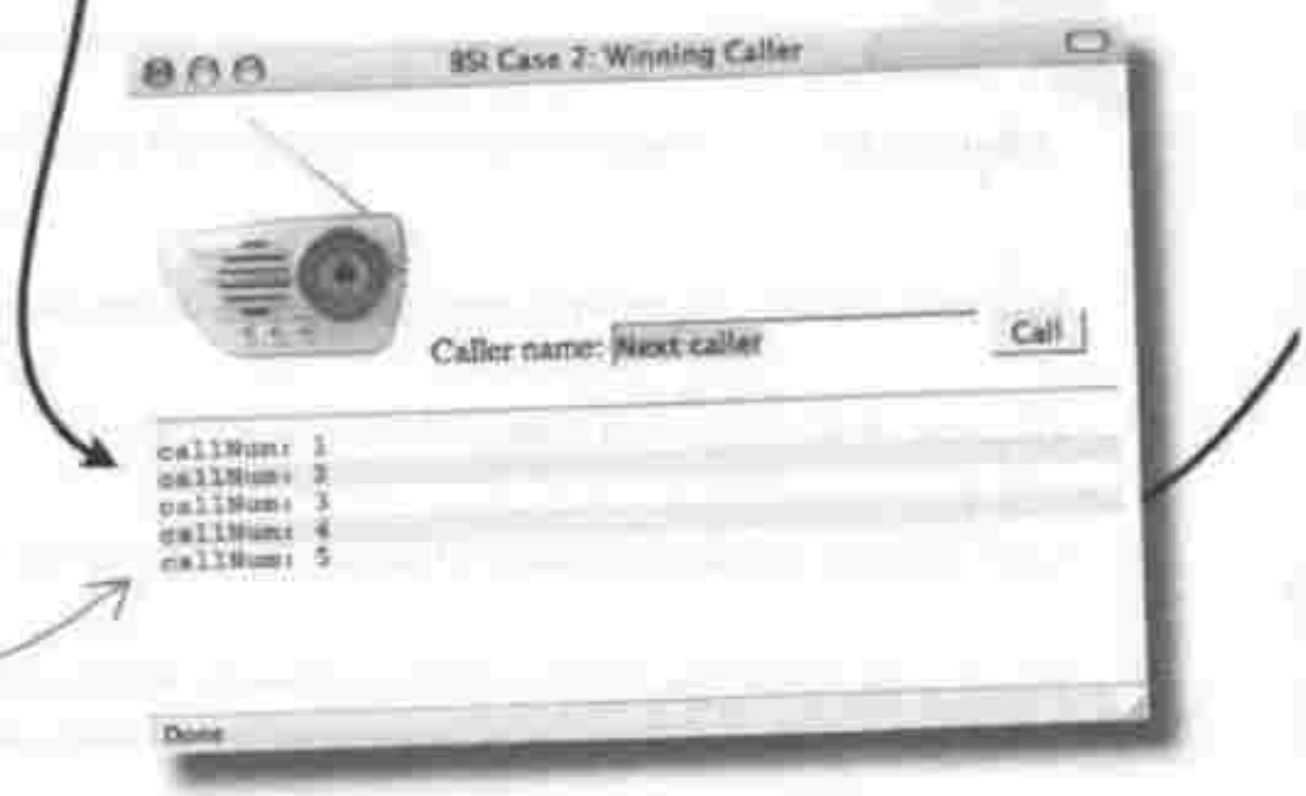
O número da chamada está aparecendo como "não é um número". Isso é muito estranho...

Não é um número — bah

Com o console de depuração finalmente funcionando, agora é possível dar uma olhada na variável `callNum` quando as chamadas chegarem sem ter que analisar através de todos aqueles alertas. E como resultado, um antigo problema que Owen ignorou finalmente voltou-se contra ele. A variável `callNum` está aparecendo como `NaN`, que significa que não é um número. Mas por que não é?

```
console.displayMsg("callNum: " + callNum);
```

Uma única linha de código define uma observação na chamada callNum



Pelo menos o console de depuração está funcionando!

Quando observar não é suficiente

Algumas vezes, observar uma variável cria mais perguntas do que respostas. Por que `callNum` não é um número? Por que não está sendo aumentada? Qual é a finalidade deste console de depuração se apenas confirma o que você já sabe... que há um problema. Portanto, como iremos descobrir especificamente o que é?

E agora?

Poderia ajudar tentar eliminar o código até o número da chamada mudar de valor.

Remover o código é uma ótima maneira de simplificar um script enquanto caçamos os erros.

Algumas vezes, menos é mais ao depurar o JavaScript. Neste caso, remover o código e observar para saber o que muda é uma idéia excelente. Mas simplesmente apagar o código não parece atraente, uma vez que a maioria permanecerá como está quando você tiver terminado de depurar. Realmente precisamos de um modo de **desativar** o código, em oposição a removê-lo de fato.

Comentários como desativadores temporários do código

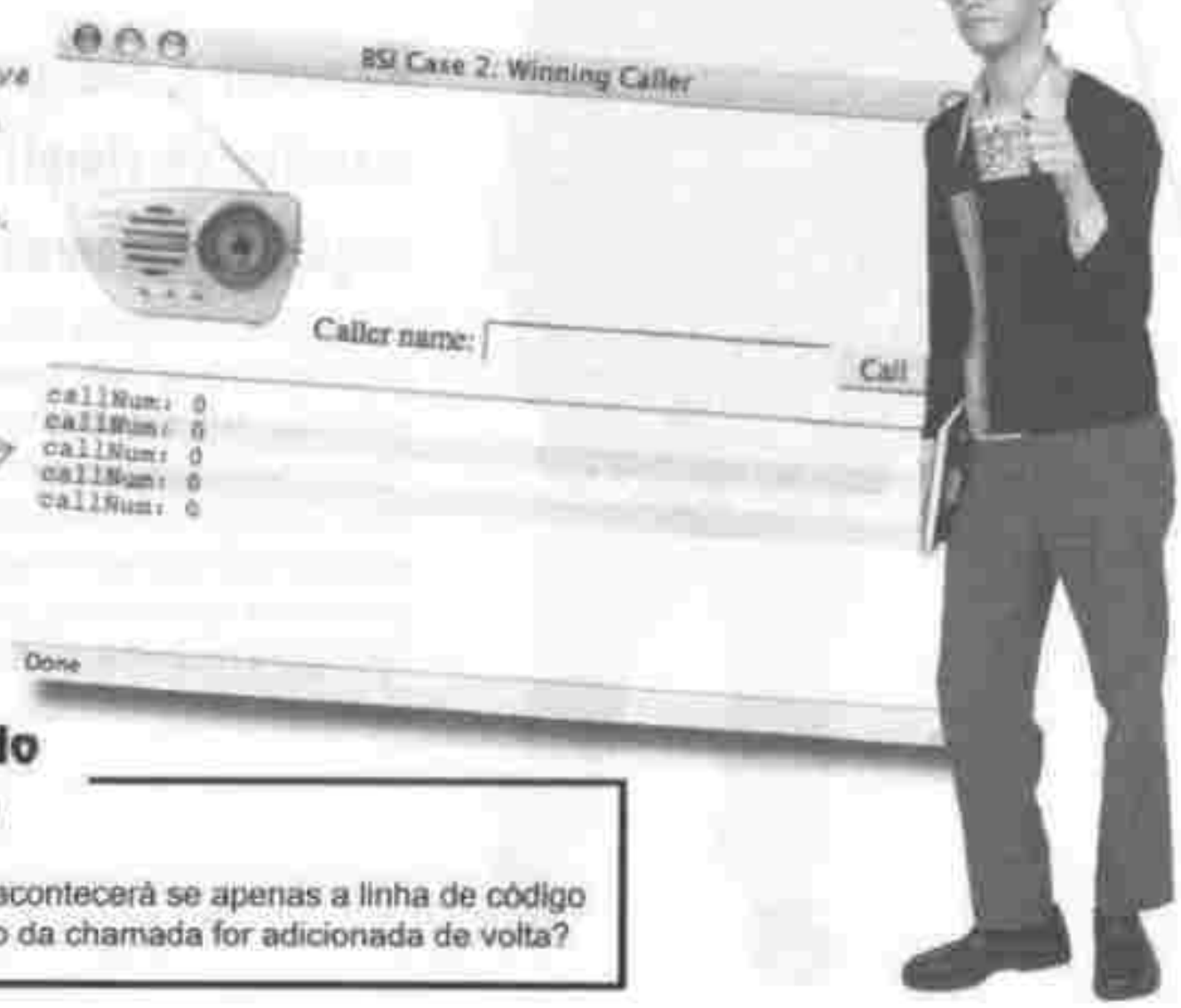
Ocultar o código executável dentro de comentários é um modo extremamente útil de desativar o código ao depurar. Isso permite ao código ser retirado seletivamente da execução de um script sem, de fato, apagar o código. Pense em comentar o código como um meio de subtrair as linhas ou as partes do código quando necessário para ajudar a isolar um erro.

Os comentários são extremamente úteis para desativar temporariamente o código.

```
function checkWinner(form, caller, winningNum) {
  console.displayMsg("callNum: " + callNum);

  /*
    // Aumente o número da chamada
    var callNum;
    ++callNum;

    // Verifique um vencedor
    if (callNum == winningNum) {
      alert(caller + ", caller number " + callNum + "... today's
winner!");
      form.submit();
    }
    else {
      // Redefina o campo caller para o próximo solicitante
      var callerField = document.getElementById('caller');
      callerField.value = "Next caller";
      callerField.focus();
      callerField.select();
    }
  */
}
```

Ei, o número da chamada agora está aparecendo como 0. Então, algo no código desativado está transformando-o em "não é um número".

Este comentário com diversas linhas desativa tudo na função, exceto o código que exibe a mensagem de depuração.



Agora, a variável callNum é 0, que significa que algo no código desativado está bagunçando-a.

Poder do cérebro

O que você acha que acontecerá se apenas a linha de código que aumenta o número da chamada for adicionada de volta?

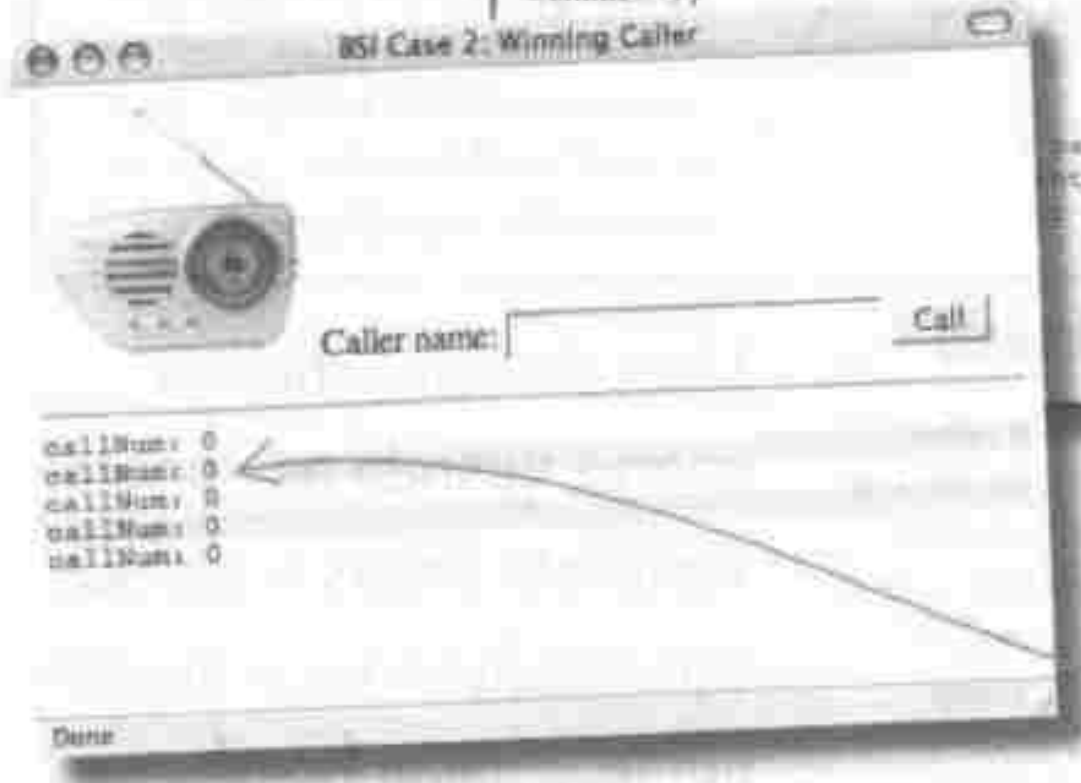
Problema resolvido... mais ou menos

Mudando para os comentários com uma linha, fica possível ser mais seletivo no código que é desativado. Se a linha do código que aumenta a variável callNum for adicionada de volta, a variável callNum começará a funcionar como deveria. Portanto, uma das linhas restantes do código desativado está causando o problema.

Os comentários com uma linha são usados para que linhas individuais possam ser ativadas e desativadas.

```
function checkWinner(form, caller, winningNum) {
  console.displayMsg("callNum: " + callNum);

  // Aumente o numero da chamada
//  var callNum;
  ++callNum;

  // Verifique um vencedor
//  if (callNum == winningNum) {
//    alert(caller + ", caller number " + callNum + "... today\'s
winner!");
```

Cancelar o comentário da linha de código que aumenta finalmente faz a variável callNum funcionar.



A variável callNum finalmente funciona como deveria, aumentando em cada chamada.

Aponte seu Lápis

Escreva o que está errado com o erro callNum no console de depuração, junto com sua correção.

Aponte seu Lápis
Solução

Escreva o que está errado com o erro callNum no console de depuração, junto com sua correção.

Outra variável chamada callNum é criada sem querer como uma variável local usando var dentro da função checkWinner( ). Portanto, a variável local callNum oculta a variável global callNum, criando um problema sutil de detectar. Como a variável local não é inicializada, aumentá-la e, então, compará-la com o número vencedor resultará em não é um número. A correção é apenas remover a linha dentro da função que cria a variável callNum local com var.

Acabar com a linha do código que cria a variável local elimina o Erro #4.

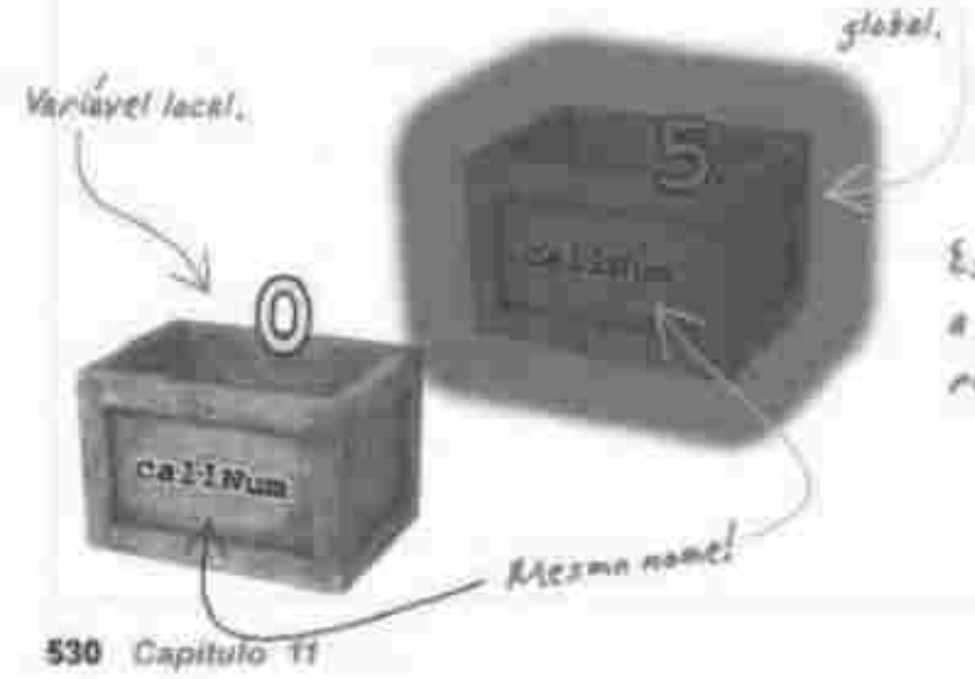
```
// Aumente o número da chamada
var callNum;
++callNum;
```

Removendo a linha do código que cria sem querer uma variável local chamada callNum, a função usa a variável callNum global, como foi pretendido originalmente.

Os perigos das variáveis shadow

O erro `callNum` no script de chamadas da rádio é um exemplo de **variável shadow**, que é uma variável que oculta sem querer outra variável com o mesmo nome. O problema surge quando uma variável local é criada com o mesmo nome de uma variável global. O JavaScript cria a variável local e fornece a precedência no bloco local do código. Portanto, qualquer alteração feita na variável local não vai para a variável global – a variável local lança, de fato, uma sombra (shadow) sobre a variável global, ocultando-a temporariamente do script.

Uma variável shadow ocorre quando as variáveis locais e globais são criadas com o mesmo nome... nada bom!



Código local

```
++callNum;
```

Este código aumenta a variável global, resultando em 6.

Código global

```
++callNum;
```

Este código aumenta a variável local, resultando em 1 – a variável global é encoberta e permanece inalterada.

Não existem Perguntas Idiotas

P: Ao comentar o código para controlar os erros, como sei quanto código desativar?

R: Este é um julgamento que você saberá melhor quando tiver mais experiência com a depuração JavaScript. Porém, nunca é errado enganar-se ao desativar mais, se não todo o código próximo de uma área problemática de um script. E se um problema realmente desagradável surgir, não hesite em desativar todo o código do script em uma página. E também não se esqueça de remover temporariamente qualquer tag de importação que coloca um código externo na página.

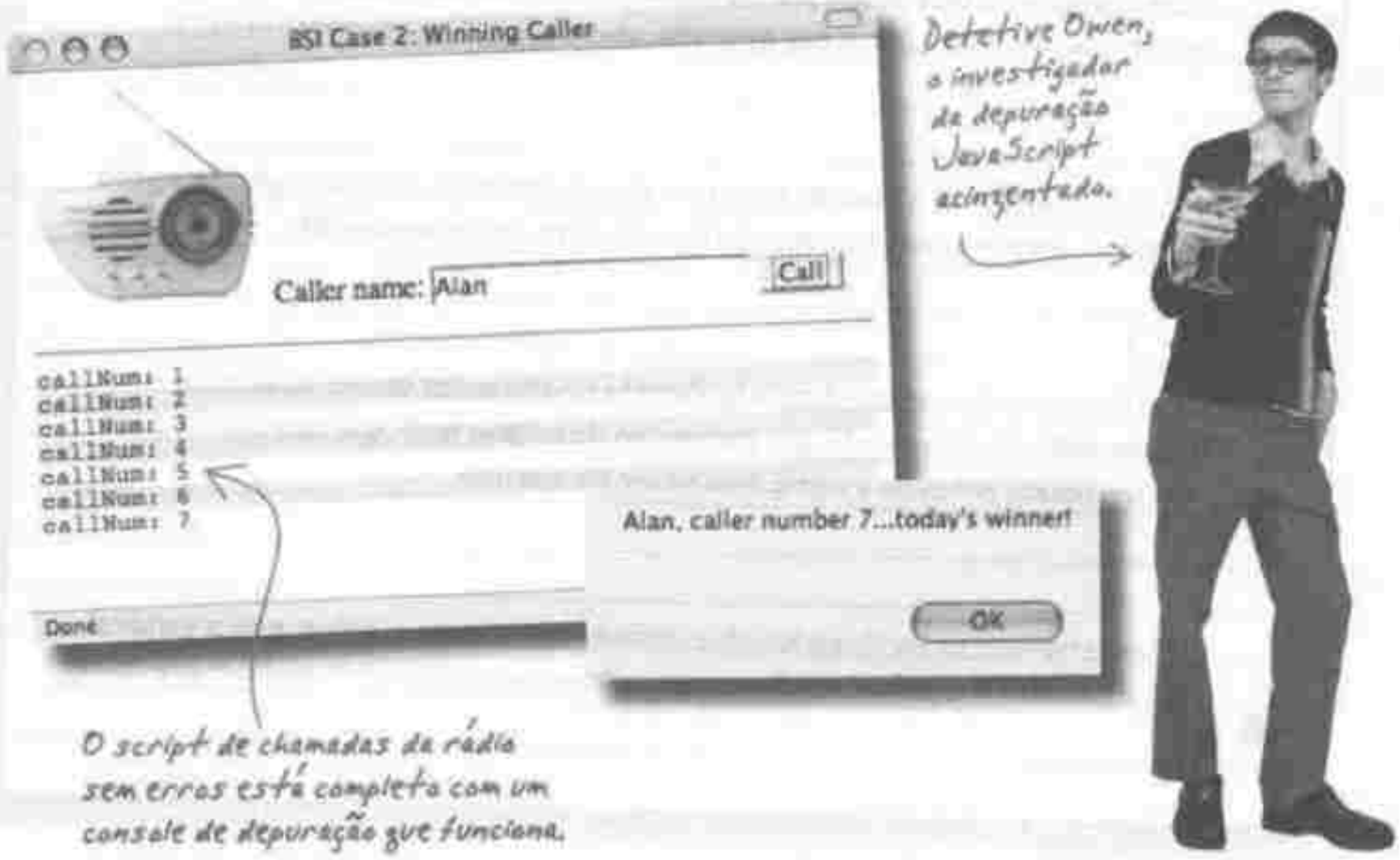
Há outra abordagem que poderá funcionar se você já isolou um erro em uma determinada seção do código do script. Essa abordagem envolve desativar uma linha de código de cada vez até que o erro desapareça. Portanto, ao invés de desativar tudo e ativar lentamente o código até o erro **aparecer**, você desativa lentamente o código, uma linha por vez, até que o erro **desapareça**. A primeira abordagem funcionará melhor se você não tiver noção sobre onde um erro está localizado, ao passo que a última funcionará melhor se tiver isolado o local do erro em algum grau.

P: E se eu quiser criar uma variável shadow? Tudo bem?

R: É como perguntar se você **pretende** quebrar sua perna, tudo bem? E a resposta é não. Só porque você deliberadamente causou problemas e sofrimento para si mesmo, não torna isso aceitável. Além disso, há muito dano para resolver ao depurar o código que deve funcionar perfeitamente, portanto não deve atirar até o fator de risco deliberadamente. Assim, a resposta real para a pergunta é que as variáveis shadow introduzem tanta confusão e incômodo no código JavaScript que elas devem ser evitadas em todas as situações.

Caso encerrado!

Com uma dose saudável de paciência e alguma ajuda de suas novas habilidades de depuração, Owen encerra o caso e embolsa uma promoção para ser um Detetive JavaScript nos Investigadores da Cena do Erro.



Lista de verificação para dar fim aos erros

- Certifique-se de que os parênteses estejam sempre em pares coincidentes.

- Certifique-se de que as chaves em torno dos blocos de código estejam sempre em pares coincidentes – o recuo cuidadoso do código ajuda a coincidir os pares de chaves.

- Tente evitar os erros tipográficos nos nomes do identificador – as variáveis e as funções poderão causar grandes problemas se seus nomes não forem usados com consistência.

- Seja consistente com o uso das aspas e dos apóstrofos, e misture os dois com cuidado nos atributos HTML, se necessário.

- Use caracteres de escape para codificar os caracteres que têm um significado especial nas strings, como, por exemplo, uma aspa (\") ou um apóstrofo (\').

- Nunca, jamais, use = quando quiser ==. Provavelmente, o JavaScript não o verá como um erro, mas seu código não funcionará como o pretendido.

- Certifique-se de que um objeto tenha sido criado antes de tentar acessá-lo – isso se aplica basicamente aos elementos da página Web, que não são criados até um pouco antes do evento onload ser inicializado.

- Nunca nomeie as variáveis locais e globais com o mesmo nome, pois a variável local encobrirá a global, resultando em algum comportamento muito imprevisível.

DOBRA DA PÁGINA

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

O que merecem todos os erros JavaScript?

É um encontro de mentes!

Dar a outra face é uma abordagem errada que pode sobrecarregar sua tolerância aos erros. A repulsa geral pelos erros causará uma debilitação em seu código, o que é um problema.

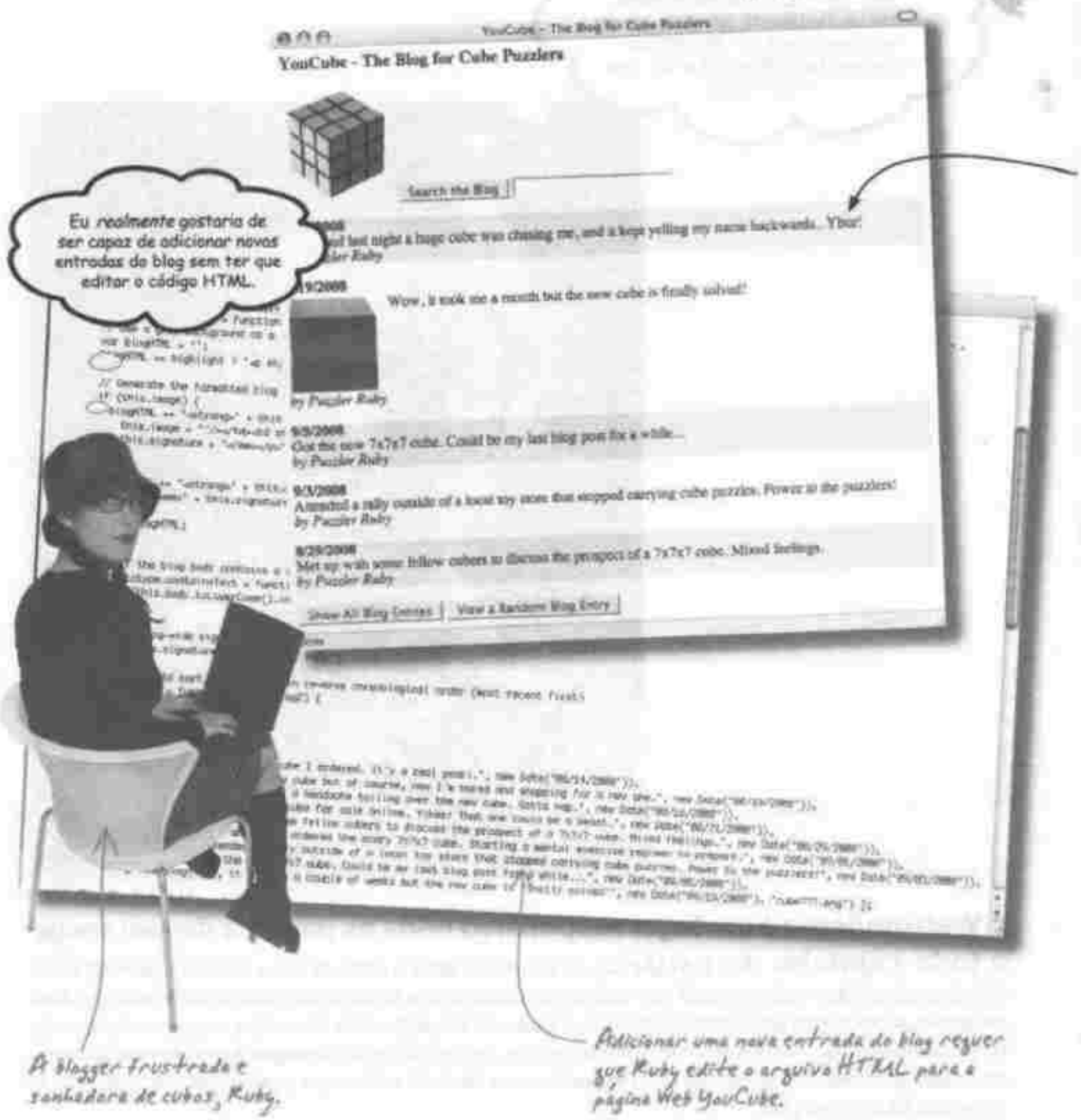
Aplicativos Web sensível ao toque

A Web moderna é um lugar responsivo onde as páginas devem reagir a todo capricho do usuário. Ou pelo menos este é o sonho de muitos usuários e desenvolvedores Web. O JavaScript desempenha um papel vital nesse sonho através de uma técnica de programação conhecida como **Ajax** que fornece um mecanismo para mudar drasticamente a "impressão" das páginas Web. Com o Ajax, as páginas Web agem de modo muito parecido com os aplicativos desenvolvidos, uma vez que são capazes de **carregar e gravar rapidamente os dados de modo dinâmico** enquanto **respondem ao usuário em tempo real** sem nenhuma renovação da página ou artifício do navegador.

Ansiando por dados dinâmicos

Lembra de Ruby, a apaixonada por quebra-cabeças de cubo e blogger? Ruby adora seu blog YouCube acionado pelo JavaScript, mas ela está frustrada por ter que editar o arquivo HTML para a página inteira só para adicionar algumas entradas novas. Ela gostaria de ser capaz de separar, de algum modo, as entradas do blog do código HTML que descreve a página do blog, liberando-a para se concentrar no conteúdo do blog em si.

Adicionar novas entrada do blog ao YouCube não deve requerer a edição da página Web.



Um YouCube baseado em dados

Os arquivos de Ruby para as páginas baseadas em dados estão disponíveis em www.altabooks.com.br.

Ruby está ciente de algo. Uma versão de seu blog que separa o conteúdo do blog da estrutura da página Web envolve dados dinâmicos, dados que são fornecidos em uma página dinamicamente quando a página é processada pelo navegador. As páginas Web, construídas a partir de dados dinâmicos, são conhecidas como páginas **baseadas em dados** porque a página realmente define apenas uma estrutura que é preenchida pelos dados. Em outras palavras, os dados são encarregados do conteúdo da página.

Os dados do blog são armazenados em um arquivo separado fisicamente que pode ser editado sem tocar na página Web.

Página Web

Dados do blog

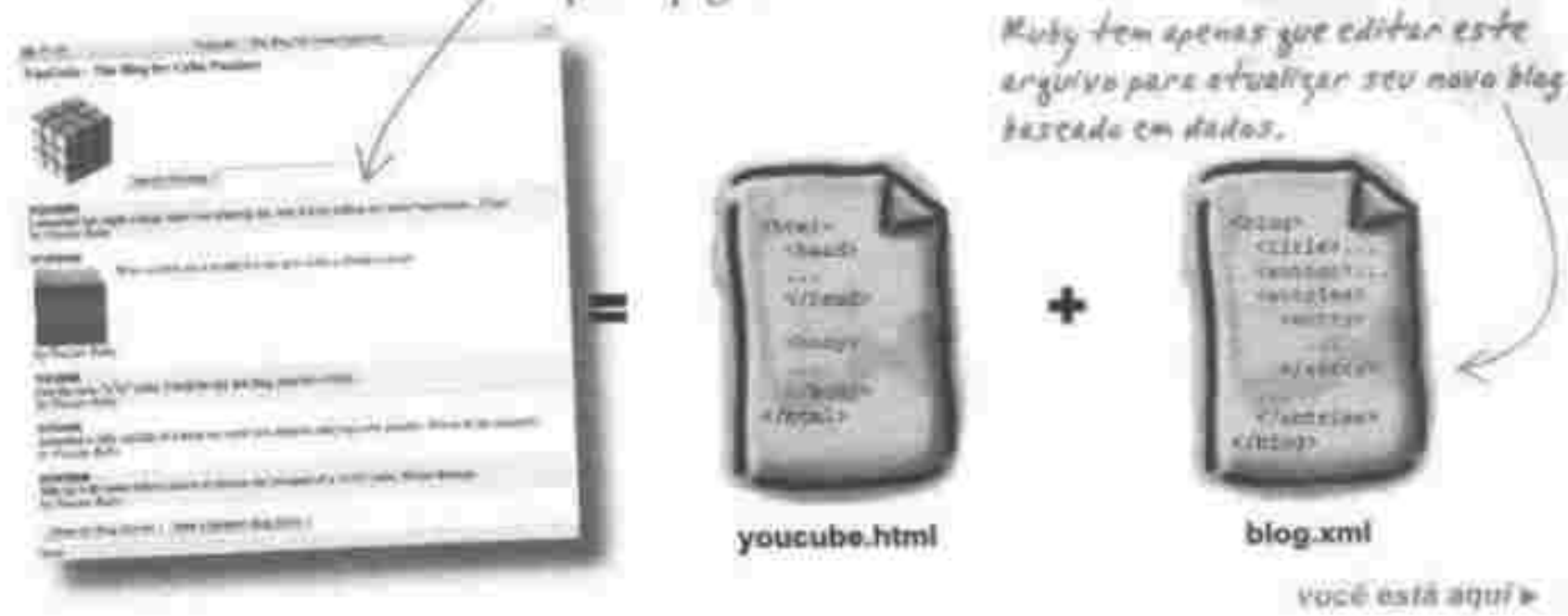
A página Web contém o código HTML para a estrutura da página Web mais o código JavaScript para incorporar os dados dinâmicos do blog na página.

O JavaScript é responsável por processar os dados do blog e combiná-los em uma página Web HTML final.

Com a ajuda do JavaScript, os dados brutos do blog são mesclados dinamicamente com o código HTML para gerar uma página YouCube final que parece idêntica à original. Mas essa página baseada em dados é montada a partir de partes separadas: a página estrutural e os dados do blog. Com os dados do blog separados em seu próprio arquivo, Ruby está livre para manipular o conteúdo do blog separado do código HTML, CSS e JavaScript para a página Web.

As entradas do blog são fornecidas à página do blog a partir de um arquivo separado.

Ruby tem apenas que editar este arquivo para atualizar seu novo blog baseado em dados.



Os dados dinâmicos parecem bem complicados. Aposto que eles requerem muito código JavaScript confuso, certo?

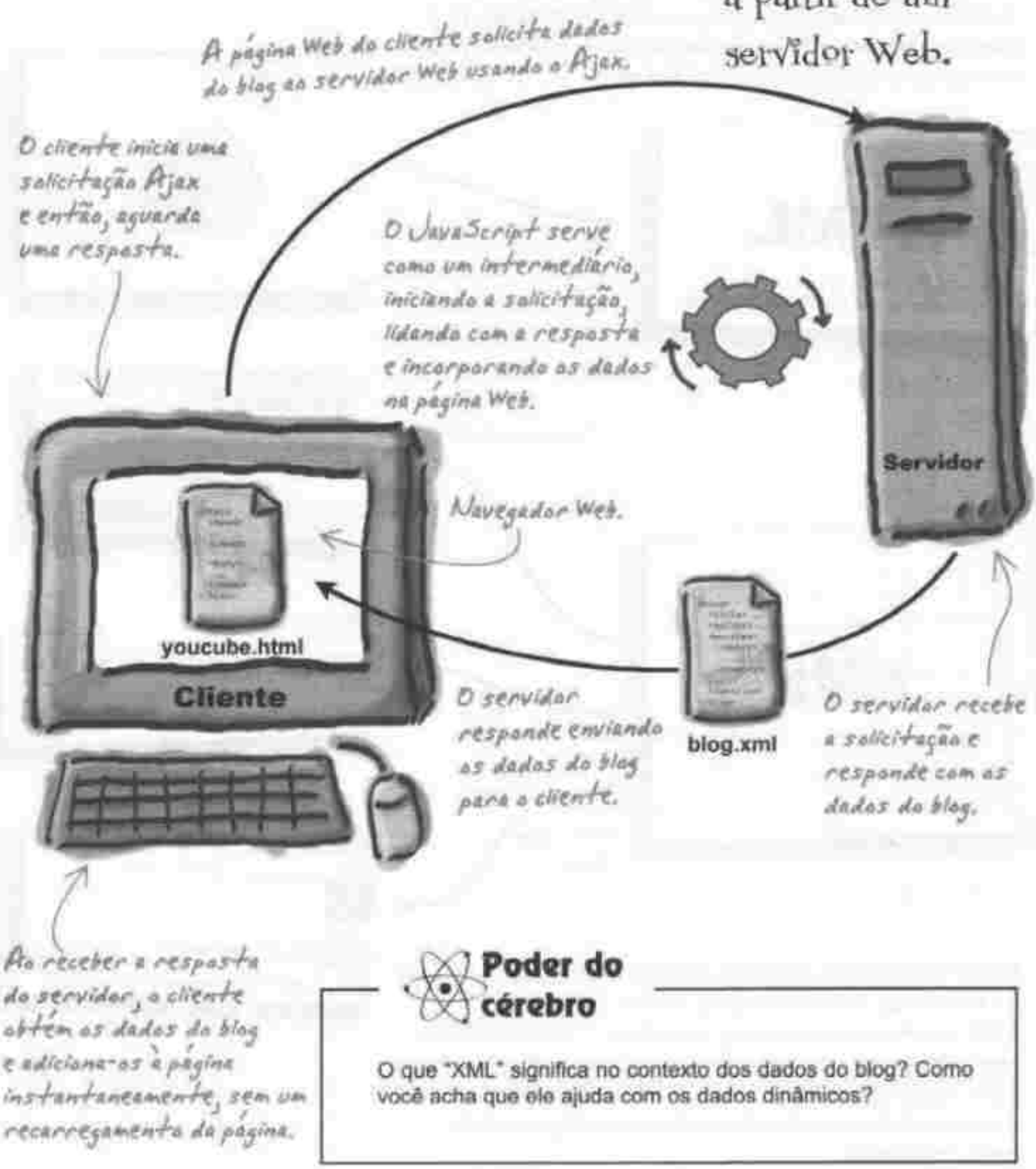
Os dados dinâmicos requerem um pouco mais de esforço de codificação direto, mas valem muito a pena no final.

Embora uma página conduzida por dados dinâmicos certamente requeira algum planejamento adicional e esforço direto, vale muito a pena, a longo prazo, com atualizações rápidas e fáceis das páginas. E mais, o JavaScript tem um suporte predefinido para os dados dinâmicos graças a uma técnica de programação elegante chamada Ajax.

O Ajax é só comunicação

O Ajax torna possível os dados dinâmicos permitindo "conversas" minúsculas entre o navegador Web do cliente e o servidor Web. Mais especificamente, um script é capaz de pedir ao servidor alguns dados como, por exemplo, uma coleção de entradas do blog, e o servidor envia-os usando o Ajax. Então, o script obtém os dados do blog e incorpora-os dinamicamente na página.

O Ajax permite a uma página Web receber dados dinamicamente a partir de um servidor Web.



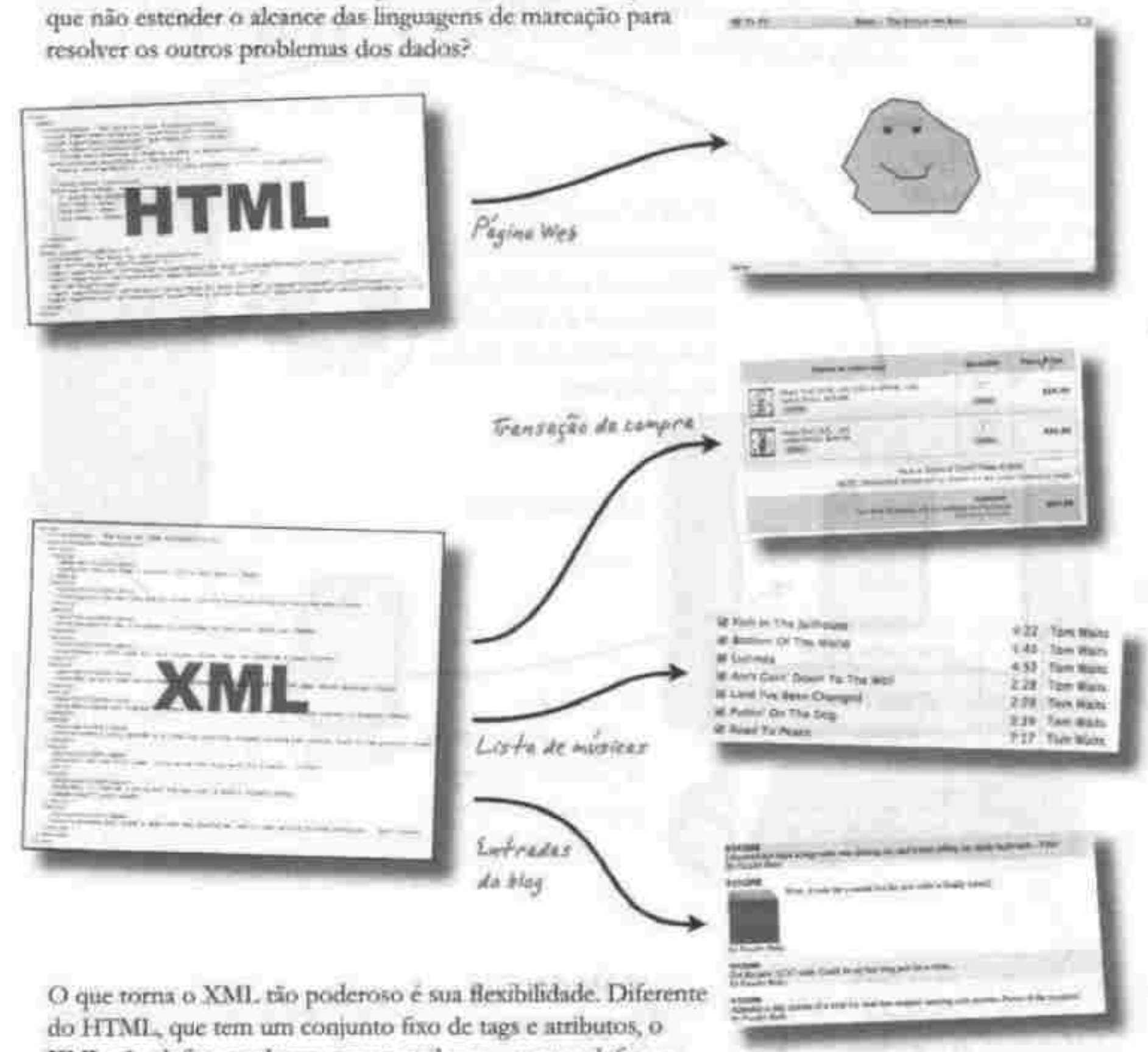
Poder do cérebro

O que "XML" significa no contexto dos dados do blog? Como você acha que ele ajuda com os dados dinâmicos?

Um HTML para tudo: XML

O "ML" em HTML significa **linguagem de marcação** e refere-se ao fato de que o HTML usa tags e atributos para criar hipertexto ("HT"). Assim como o HTML é usado para criar páginas Web de hipertexto, o XML é outra linguagem de marcação usada para criar, bem, qualquer coisa desejada. É isso que o "X" significa – qualquer coisa! A idéia é que há todos os tipos de dados que poderiam aproveitar o fato de serem armazenados como tags e atributos. Portanto, por que não estender o alcance das linguagens de marcação para resolver os outros problemas dos dados?

XML é uma linguagem de marcação usada para formatar qualquer tipo de dados.



O que torna o XML tão poderoso é sua flexibilidade. Diferente do HTML, que tem um conjunto fixo de tags e atributos, o XML não define nenhuma tag ou atributo – apenas define as regras sobre como as tags e os atributos são criados e usados. Cabe a cada aplicativo do XML, em particular, explicar as particularidades das tags e dos atributos que representam os dados específicos.

O XML permite colocar tag em SEUS dados do SEU jeito

O verdadeiro encanto do XML é que ele pode transformar qualquer pessoa em um criador de tags personalizado usando um pouco da alquimia das tags e dos atributos para inventar uma linguagem de marcação inteiramente personalizada para qualquer finalidade. Existem, certamente, muitas linguagens XML existentes que já foram criadas para resolver muitos problemas diferentes e não será uma má idéia usar uma delas se ela se adequar às suas necessidades. Mas criar uma linguagem de marcação personalizada própria é uma tentação difícil de resistir.

Parecido com o código HTML, este código XML consiste em uma hierarquia de elementos.

```
<movie>
  <title>Cleaning the Cube</title>
  <releaseDate>01/13/1989</releaseDate>
  <director>Graeme Clifford</director>
  <summary>A skateboarder investigates the death of his adopted brother.</summary>
</movie>
```

Cada aspecto do filme é armazenado em sua própria tag exclusiva.

Os detalhes do filme estão contidos na tag <movie>.

Mesmo que você nunca tenha visto esta linguagem de marcação XML de exemplo, que é inteiramente personalizada, as tags descritivas tornam possível decifrar os dados. Mais importante ainda, as tags são muito específicas para os dados que sendo armazenados – simplesmente faz sentido usar uma tag denominada `<director>` ao armazenar o diretor de um filme!

Exercício

Coincida as seguintes tags com suas descrições e, então, escreva ao lado de cada descrição se a tag é HTML ou XML.

..... `<itunes:author>`	O texto em negrito em uma página Web.
..... `<span>`	O título de um newsfeed on-line.
..... `<title>`	Um controle de entrada em uma página Web.
..... `<strong>`	O texto que é convertido em conversa para quem faz a chamada telefônica.
..... `<input>`	O artista de um podcast iTunes.
..... `<prompt>`	O conteúdo em linha em uma página Web.

Exercício Solução

Coincida as seguintes tags com suas descrições e, então, escreva ao lado de cada descrição se a tag é HTML ou XML.

XML <itunes:author>	O texto em negrito em uma página Web.
XML	O título de um newsfeed on-line.
XML <title>	Um controle de entrada em uma página Web.
XML	O texto que é convertido em conversa para quem faz a chamada telefônica.
XML <input>	O artista de um podcast iTunes.
XML <prompt>	O conteúdo em linha em uma página Web.

XML é apenas texto

Parecido com o HTML, os dados XML são **apenas texto**, o que significa que são armazenados em um arquivo de texto normal. Porém, os arquivos XML são nomeados com uma extensão de arquivo **.xml**, em oposição às extensões .html ou .htm usadas nos arquivos HTML.

Os dados XML são geralmente armazenados em arquivos com uma extensão de arquivo .xml.

XML + HTML = XHTML

Podem ter extensões de arquivo diferentes, mas o XML e o HTML têm uma conexão muito importante e ela é chamada de XHTML. O XHTML é uma versão moderna do HTML que segue regras mais rígidas do XML. Por exemplo, toda tag de abertura em uma página Web XHTML deve ter uma tag de fechamento. O HTML é irresponsável com sua sintaxe, significando que você pode escapar sem combinar as tags como, por exemplo, e . O XHTML não é tão bondoso e requer que tais tags sempre apareçam em pares coincidentes.

O XHTML é uma versão do HTML que segue as regras de sintaxe mais rígidas do XML.

HTML

```
Este é um parágrafo de texto no HTML.<p>
```

A tag geralmente é usada sozinha no código HTML para indicar o início ou o final de um parágrafo.

XHTML

```
<p>Este é um parágrafo de texto no XHTML.</p>
```

As tags com conteúdo sempre devem aparecer como pares coincidentes no XHTML.

Outra diferença importante entre HTML e XHTML envolve as tags vazias, como <br>, que devem ser codificadas com um espaço em branco e, então, uma barra no final para indicar que não há nenhuma tag de fechamento.

HTML

```
Esta é apenas uma frase.<br>
```

A tag da quebra de linha vazia é geralmente codificada no HTML sem uma barra.

XHTML

```
Esta é apenas uma frase.<br />
```

O espaço e a barra são requeridos em todas as tags vazias no XHTML.

Mais uma distinção importante entre HTML e XHTML é que o XHTML requer que todos os valores do atributo fiquem entre aspas.

HTML

```
<a href=home.html>Vá para casa</a>
```

O valor do atributo não está entre aspas, o que viola as regras do XHTML.

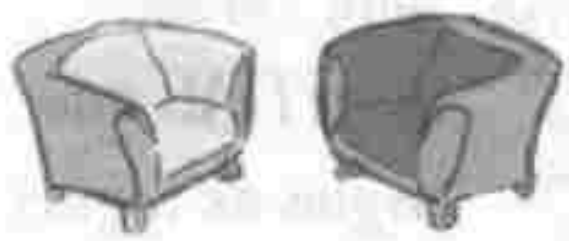
XHTML

```
<a href="home.html">Vá para casa</a>
```

Todos os valores do atributo XHTML devem aparecer entre aspas.

Embora o XHTML não atue diretamente nas necessidades imediatas de Ruby em termos de modelar os dados do blog no XML, ele esclarece algumas das regras de sintaxe mais importantes do XML, que se aplicam a todas as linguagens baseadas na XML, inclusive a linguagem de dados do blog personalizado de Ruby.

Conversa informal



Conversa desta noite: HTML e XML informam sobre os dados Web

A última versão do HTML foi reformulada usando a XML e é chamada de XHTML.

XHTML:

Você sabe, você deixou as coisas realmente confusas para mim. Aqui, sou a estrutura da Web e, agora, muitas pessoas estão confusas sobre mim, graças a você.

Mas você ainda não é bom sem mim porque os navegadores exibem apenas o código HTML. Eles nem sabem o que fazer com você.

Como é possível? Quem liga para dados sem aparência?

Tudo isso pode ser visto graças a mim – está tudo lá na Web.

Entendo. Então, você está sugerindo que realmente trabalhemos juntos? Isso é um grande alívio!

XML:

Não minha culpa que você tenha uma visão estreita, sempre pensando nas página Web. Eu amplíei a minha mente e fazendo isso, posso representar qualquer tipo de dado.

Ei, sou um cara misterioso. A verdade é que sou um homem sem face – todo substância e sem aparência. Preciso de você quando é hora de me mostrar.

Opa, você realmente não sai muito, sai? O resto do mundo opera nos dados que não podem ser vistos na maioria das vezes. Transações bancárias, apuração de votos, condições do tempo, você sabe.

É verdade, mas como você acha que tudo é armazenado antes de ter sucesso em um navegador Web? Não como parágrafos e tabelas, posso adiantar. Geralmente é armazenado usando-me porque forneço muita estrutura e contexto – eu torno os dados fáceis de serem processados.

Absolutamente! Não tenho nenhuma idéia de como ficam os dados. Na verdade, concentro-me no que eles **significam**. Contanto que as pessoas continuem usando os navegadores Web, continuarei a precisar de sua ajuda para exibir os dados que represento.

XML e dados do blog YouCube

O XHTML é um ótimo aplicativo do XML que está melhorando rapidamente a estrutura e a confiabilidade das páginas Web. Em relação ao blog YouCube, porém, Ruby precisa de uma linguagem XML personalizada que modele os dados específicos de seu blog. Isso requer avaliar os dados diferentes requeridos do blog e considerar como poderiam se encaixar no contexto das tags XML hierárquicas.

```
<blog>
  <title>      YouCube - The Blog For Cube Puzzlers
  <author>     Blog.prototype.signature = "by Puzzler Ruby";
  <entries>
    <entry>
      <date>   blog[0] = new Blog("Got the new cube I ordered. It's a real pearl.",
      <body>     new Date("08/14/2008"));
```

Aponte seu Lápis

Crie sua própria linguagem XML para armazenar um blog e use a linguagem para codificar uma entrada do blog. Os itens como título, data, autor e a própria entrada devem ser considerados.

<blog>

.....

.....

.....

.....

.....

.....

.....

.....

</blog>

Aponte seu Lápis
Solução

Invente sua própria linguagem XML para armazenar um blog e use a linguagem para codificar uma entrada do blog. Os itens como título, data, autor e a própria entrada devem ser considerados.

```
<blog>
  <title>YouCube - The Blog for Cube Puzzlers</title>
  <author>Puzzler Ruby</author>
  <entries>
    <entry>
      <date>11/14/2007</date>
      <body>Got the new cube I ordered. It's a real pearl.</body>
    </entry>
  </entries>
</blog>
```

O blog inteiro está contido na tag <blog>

A tag <title> hospeda o título do blog.

Adivinhe qual tag mantém o autor do blog?

A coleção de entrada do blog é armazenada na tag <entries>.

Cada entrada do blog é representada pela tag <entry>.

A data e o corpo de cada entrada do blog têm suas próprias tags respectivas.

Não existem Perguntas Idiotas

P: Por que não armazenar os dados do blog simplesmente como texto comum sem formatação?

R: Você poderia, mas então colocaria uma carga enorme no código do script para examinar os dados e tentar dividi-los em entradas do blog separadas com suas próprias datas e corpos. O XML adiciona uma estrutura previsível aos dados para que você possa distinguir facilmente os campos separados de dados como, por exemplo, as entradas exclusivas do blog com suas próprias datas e corpos, sem mencionar o título e o autor do próprio blog.

P: A tag `<entries>` realmente é necessária nos dados do blog XML?

R: Não é estritamente necessário, mas torna o formato dos dados mais estruturado e fácil de entender. Por exemplo, sem a tag `<entries>` nos dados do blog anteriores, seria impossível dizer que o formato do blog é capaz de suportar diversas tags `<entry>`, mas apenas uma tag `<title>` e `<author>`. A tag `<entries>` implica que há uma coleção de diversas entradas do blog, que fornece aos dados mais estrutura e torna mais óbvio o modo como os dados devem ser usados.

P: Qual a ligação entre XML e Ajax?

R: O Ajax já foi tido como sendo um acrônimo para **A**synchronous **J**avaScript **A**nd **X**ML, portanto XML esteve, em algum momento, diretamente ligado ao Ajax. Esse acrônimo agora é considerado passado, pois o papel do Ajax aumentou para nem sempre requerer o XML como parte da equação. Mas a realidade é que o XML ainda forma a base da maioria dos aplicativos Ajax porque fornece um ótimo mecanismo para modelar os dados.

Como descobriremos mais tarde no capítulo, há uma conexão entre o Ajax e o XML no modo como o JavaScript suporta o Ajax. O JavaScript não o impede de usar o XML como um formato de dados para executar as solicitações e as respostas Ajax, mas as torna muito mais fáceis ao lidar com tudo, exceto o mais comum dos dados. Portanto, embora os puristas Ajax possam reclamar que o XML e o Ajax não têm nenhuma ligação real entre si, em termos práticos eles geralmente andam de mãos dadas. O antigo acrônimo ainda ressoa a verdade grande parte das vezes, mesmo que tenha sido desaprovado. Iremos explorar a parte "assíncrona" do acrônimo um pouco mais tarde.

O XML sozinho não é dinâmico, mas ele se entrosa muito bem no Ajax e no DOM.

O XML é o formato de dados mais usado com o Ajax e, portanto, é o candidato lógico para representar os dados do blog que serão enviados do servidor para o cliente na versão baseada em dados do YouCube. É a natureza altamente estruturada do XML que o torna tão ideal para mover os dados.

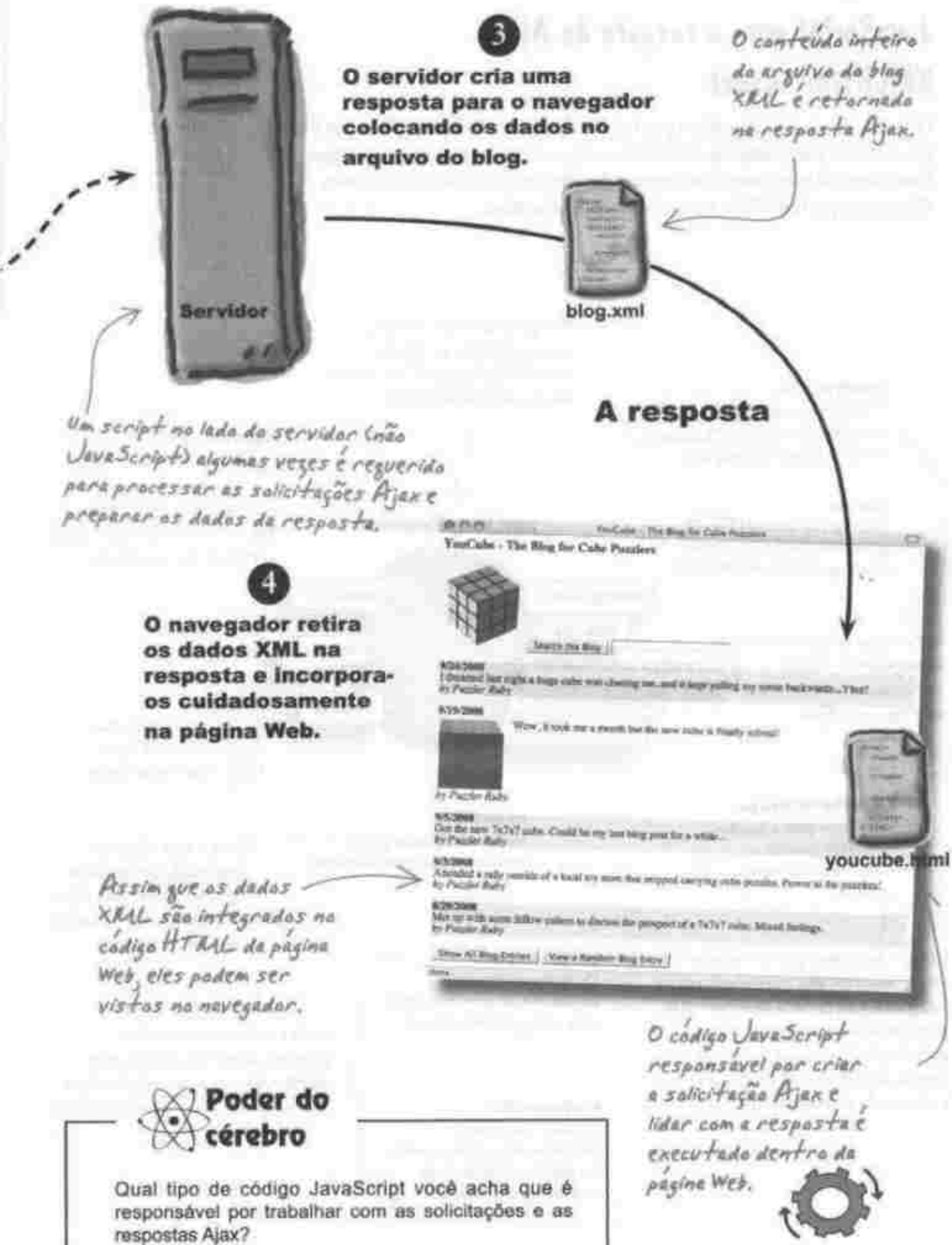
E a semelhança do XML com o HTML (XHTML) torna possível usar o DOM para acessar os dados XML como uma árvore de nós. Isso significa que você pode escrever o código JavaScript que percorre a árvore de nós XML, isolando com cuidado os dados desejados e, então, incorporando-os em uma página Web dinamicamente. São todas essas coisas e muito mais que tornam o XML uma ótima solução de armazenamento de dados para construir páginas dinâmicas baseadas em dados.

Como injetar o YouCube com o Ajax

Com um novo documento do blog XML brilhante em mãos, Ruby está pronta para carregá-lo dinamicamente na página YouCube com a ajuda do Ajax.

A solicitação

O Ajax gira em torno do conceito de **solicitações** e **respostas** como um meio de realizar a comunicação dos dados entre o navegador do cliente e o servidor.



Poder do cérebro

Qual tipo de código JavaScript você acha que é responsável por trabalhar com as solicitações e as respostas Ajax?

JavaScript para o resgate do Ajax: XMLHttpRequest

O JavaScript inclui um objeto predefinido chamado `XMLHttpRequest` que é usado para iniciar as solicitações Ajax e lidar com as respostas Ajax. Esse objeto é bem complexo, contendo vários métodos e propriedades diferentes que trabalham junto para suportar o Ajax.

Há alguns outros métodos e propriedades no objeto XMLHttpRequest, mas estes são os mais importantes.

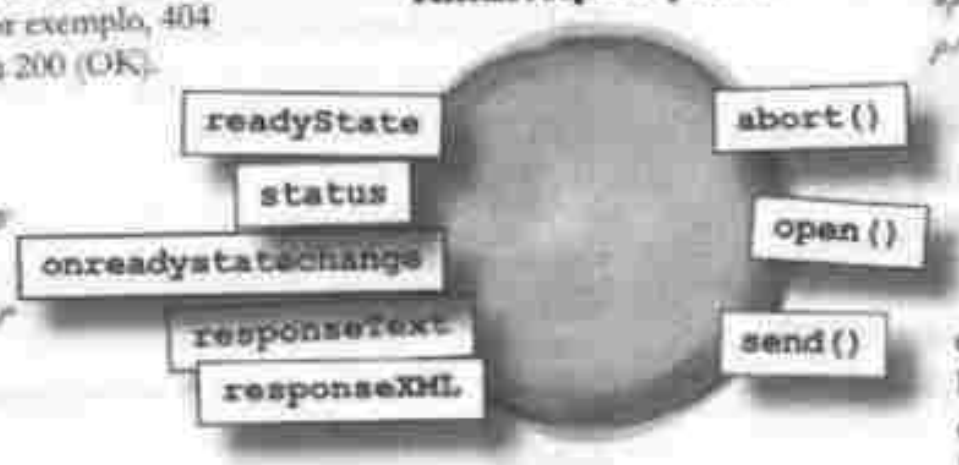
readyState
O estado numérico da solicitação: 0 (não inicializado), 1 (aberto), 2 (enviado), 3 (recebendo) ou 4 (carregado).

status
O código de status HTTP da solicitação como, por exemplo, 404 (não encontrado) ou 200 (OK).

Estas duas propriedades armazenam os dados retornados pelo servidor na resposta Ajax.

abort( )
Cancela a solicitação

Este método será usado apenas se a solicitação Ajax precisar ser cancelada.



open( )
Prepara uma solicitação especificando seu tipo e URL, entre outras coisas.

onreadystatechange
Uma referência para a função que é chamada quando o estado da solicitação muda.

send( )
Envia a solicitação para o servidor para o processamento.

Estes dois métodos trabalham juntos para ter uma solicitação Ajax pronta e, então, enviá-la para o servidor.

Esta propriedade é única no sentido de que mantém uma referência para a sub-rotina de evento personalizada que é chamada quando o estado da solicitação Ajax muda – é nesta função da sub-rotina de eventos que a resposta é processada.

responseText
Os dados da resposta retornados do servidor, como uma string de texto comum.

responseXML
Os dados da resposta retornados do servidor, como um objeto consistindo em uma árvore de nós XML.

Estas duas propriedades juntas podem ser usadas para determinar se a solicitação Ajax terminou com uma resposta válida.

O XMLHttpRequest é muito complexo

O `XMLHttpRequest` é incrivelmente poderoso e também surpreendentemente flexível. Mas com esse poder e flexibilidade vem a **complexidade**, significando que até a mais básica das solicitações Ajax requer uma boa quantidade de código JavaScript. Isso é graças, em parte, às inconsistências do navegador, mas também não ajuda que as diferentes opções disponíveis para ajustar o comportamento do objeto possam ser confusas, quando tudo o que você realmente precisa fazer é mover rapidamente alguns dados de modo dinâmico.

O objeto XMLHttpRequest é poderoso, mas também um pouco difícil de usar.

Como exemplo, considere que o seguinte código seja necessário apenas para criar um objeto `XMLHttpRequest` que funcionará em vários navegadores:

```
var request = null;
if (window.XMLHttpRequest) {
  try {
    request = new XMLHttpRequest();
  } catch(e) {
    request = null;
  }
// Agora, experimente a versão ActiveX (IE)
} else if (window.ActiveXObject) {
  try {
    request = new ActiveXObject("Msxml2.XMLHTTP");
  // Experimente o objeto ActiveX mais antigo para as versões mais antigas do IE
  } catch(e) {
    try {
      request = new ActiveXObject("Microsoft.XMLHTTP");
    } catch(e) {
      request = null;
    }
  }
}
```

O código tem que tentar algumas abordagens diferentes para criar o objeto XMLHttpRequest porque alguns navegadores (IE) suportam-no de modo diferente.

A instrução try-catch é um mecanismo de tratamento de erros JavaScript avançado que permite a um script lidar com elegância com os erros de execução.

Depois do objeto `XMLHttpRequest` ser criado, é hora de definir a função da sub-rotina da solicitação e, então, abrir a solicitação.

Exata é a função personalizada que é chamada quando o servidor responde à solicitação.

```
request.onreadystatechange = handler;
request.open(type, url, true); // sempre assincrona (true)
```

Abrir a solicitação a deixa pronta para ser enviada e também determina qual é o tipo de solicitação (GET ou POST).

Nerd

O problema ao criar um objeto `XMLHttpRequest` é que os navegadores têm que fornecer suas próprias implementações do objeto. A boa notícia é que os métodos e as propriedades são consistentes em todos os navegadores – é a criação do objeto que tem que decompor as diferenças do navegador.

Gets e posts

O **tipo** de uma solicitação Ajax é muito importante e reflete não apenas o que está sendo enviado para o servidor, mas também a **intenção** da solicitação. Um tipo de solicitação, também conhecido como **método** da solicitação, é `GET`, que é usado basicamente para recuperar os dados no servidor sem afetar nada nele. O outro tipo de solicitação, `POST`, geralmente envolve enviar dados para o servidor, depois do qual o estado do servidor geralmente muda de algum modo em resposta aos dados que foram enviados.

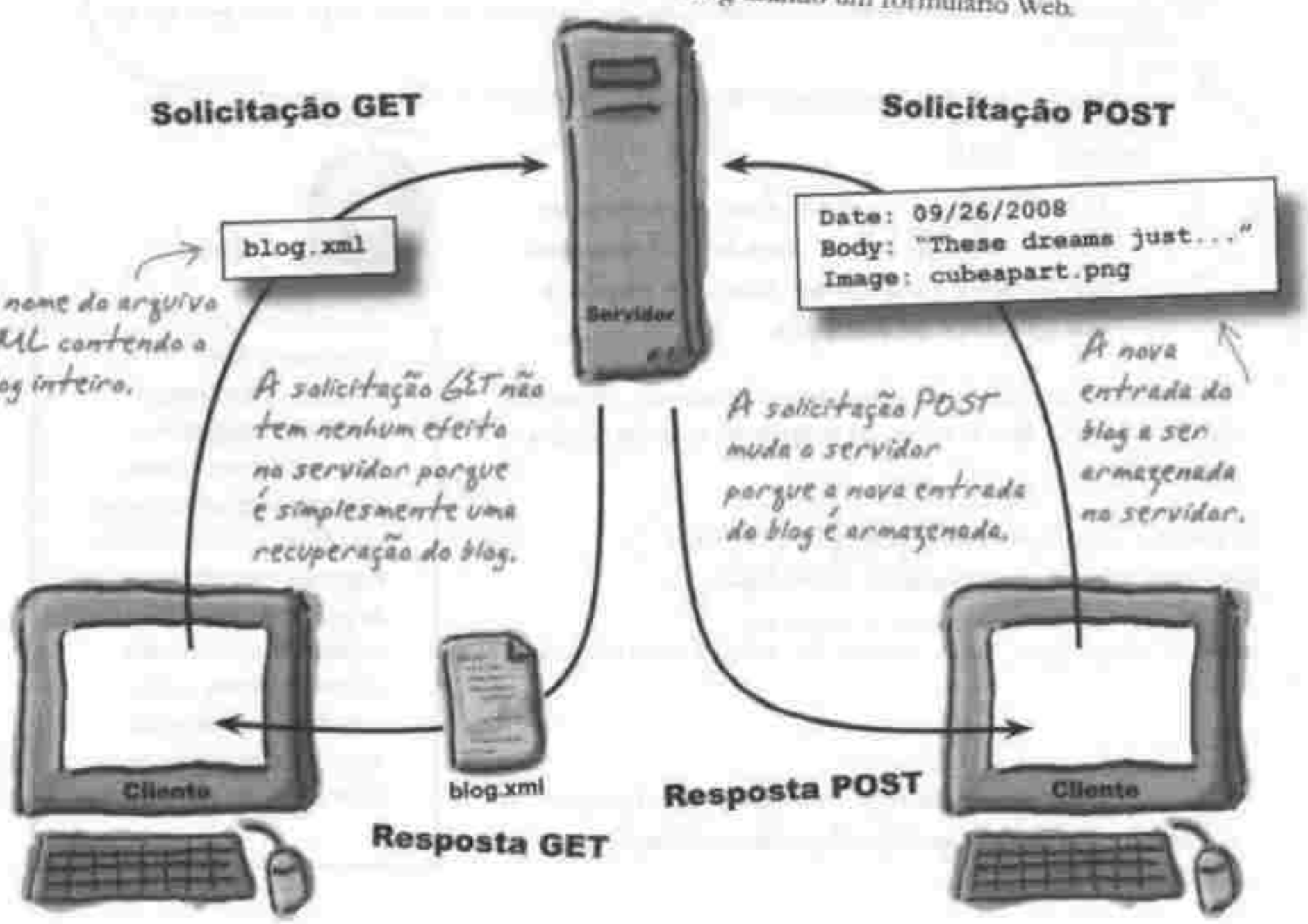
Os dois tipos de solicitações usadas com o Ajax são GET e POST, as mesmas usadas ao enviar os formulários HTML.

GET

Usado para a recuperação dos dados que não mudam nada no servidor. Pequenas quantidades de dados ainda poderão ser enviadas para o servidor no URL, se necessário. `GET` é perfeito para recuperar os dados do blog em um arquivo XML no servidor.

POST

Usado para enviar dados para o servidor que causam, de algum modo, uma mudança em seu estado como, por exemplo, salvar os dados em um banco de dados. Os dados ainda podem ser retornados em uma resposta. `POST` é ideal para uma tarefa como, por exemplo, adicionar dinamicamente uma nova entrada do blog ao blog usando um formulário Web.



Get ou Post? Uma solicitação com XMLHttpRequest

Depois de decidir sobre um tipo de solicitação e especificá-la ao abrir a solicitação, finalmente é hora de enviar a solicitação para o servidor para o processamento. O código específico para enviar uma solicitação varia de acordo com a solicitação ser GET ou POST.

A solicitação GET e o URL são especificados quando a solicitação é aberta.

```
request.open("GET", "blog.xml", true); // sempre assincrona (true)
request.send(null);
```

A solicitação é enviada sem dados, por isso o argumento para send() é null.

Os dados do blog XML são solicitados a partir do arquivo blog.xml no servidor através de uma solicitação GET.

Uma nova entrada do blog é enviada para o servidor em uma solicitação POST.

A solicitação envolve os dados sendo enviados para o servidor, portanto o tipo do dado deve ser definido.

A solicitação POST e o URL do servidor, neste caso um script do servidor, são especificados quando a solicitação é aberta.

```
request.open("POST", "addblogentry.php", true); // sempre assincrona (true)
request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded; charset=UTF-8");
request.send("09/26/2008&These dreams just...&cubeapart.png");
```

A solicitação é enviada com os dados transmitidos no argumento para o método send().

Relaxe

Não se estresse com todo esse negócio de GET e POST.

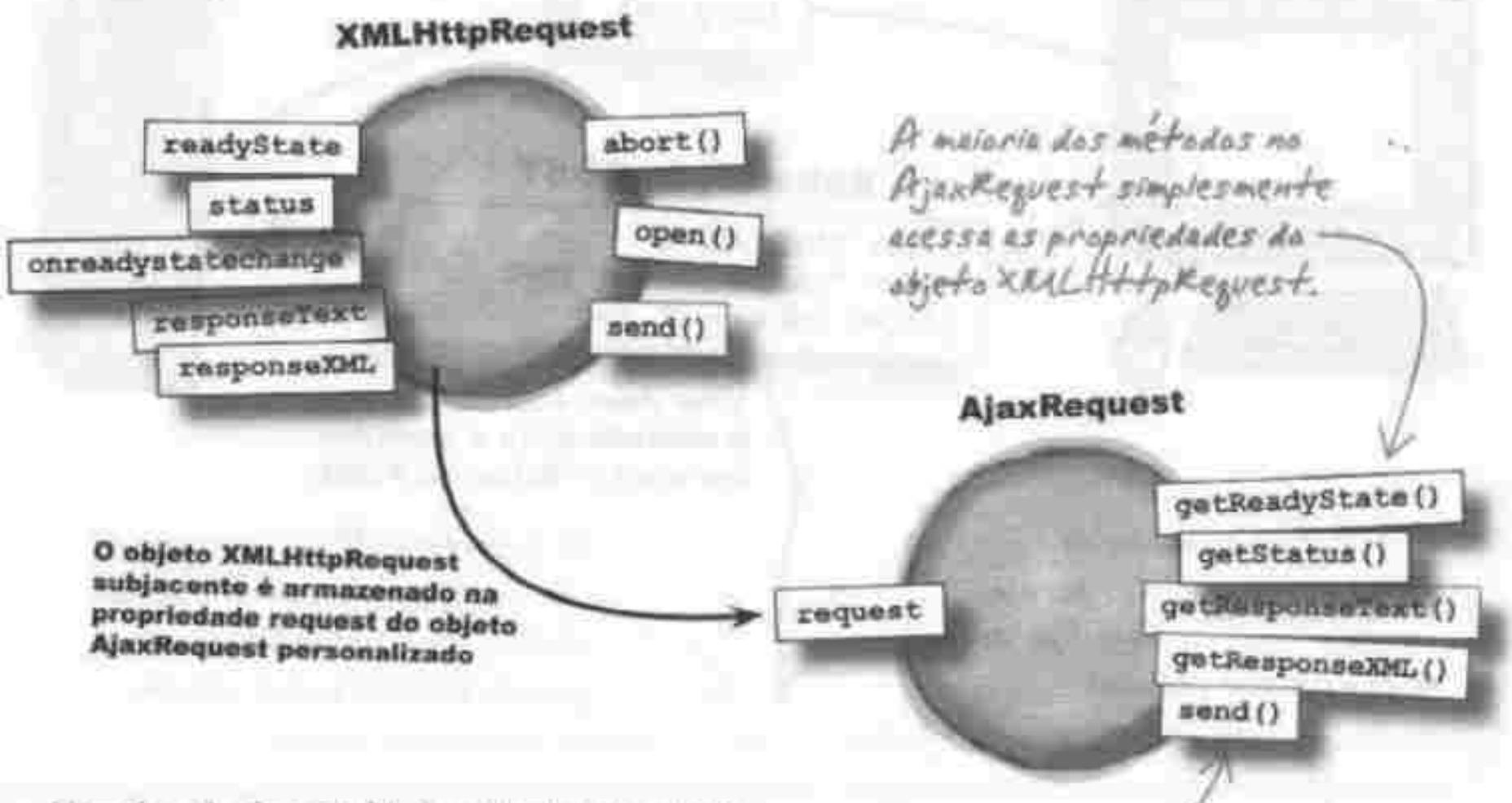
Se você não tiver muita experiência com o GET e o POST no HTML, não se preocupe. Eles farão mais sentido quando seus papéis no YouCube continuarem a ficar mais sólidos.

Torne o XMLHttpRequest menos difícil

Embora o objeto `XMLHttpRequest` seja incrivelmente poderoso, é preciso muito aprendizado, como sem dúvida alguma você já percebeu. Não apenas isso, mas ele requer certa quantidade de código "padronizado" que deve estar em cada aplicativo Ajax que o utiliza. Por isso, muitas bibliotecas de terceiros foram criadas para facilitar o uso do objeto `XMLHttpRequest`. Muitas dessas bibliotecas estendem os recursos do JavaScript, que é ótimo mas requer ainda mais aprendizagem.

O objeto AjaxRequest personalizado facilita o problema de fazer as solicitações Ajax.

Por isso, uma estratégia útil para o YouCube é criar um objeto personalizado mínimo que serve como um assistente conveniente para o `XMLHttpRequest`, permitindo-nos focar puramente em **fazer as coisas** com o Ajax, em oposição a lutar com o objeto `XMLHttpRequest` ou dominar alguma biblioteca de terceiros. Esse objeto personalizado, `AjaxRequest`, adota uma abordagem minimalista para tornar o objeto `XMLHttpRequest` mais útil.



Além do método `send()`, que veremos em um momento, é no construtor para o `AjaxRequest` que o Ajax é muito simplificado em comparação a usar o objeto `XMLHttpRequest` sozinho. É tudo o que é necessário para criar um objeto `AjaxRequest` capaz de iniciar as solicitações Ajax em qualquer navegador moderno:

```
var ajaxReq = new AjaxRequest();
```

O construtor para o AjaxRequest decompõe automaticamente todas as complexidades de criar o objeto XMLHttpRequest subjacente.

Ímãs de geladeira JavaScript

O objeto `AjaxRequest` personalizado integra o objeto `XMLHttpRequest` padrão, fornecendo uma interface muito mais simples para enviar as solicitações Ajax e lidar com suas respostas. O problema é, o método `send( )` do objeto `AjaxRequest` não tem algumas partes principais do código. Use os ímãs para terminar o código do método.

```
AjaxRequest.prototype.send = function(type, url, handler, postDataType, postData) {
  if (this.request != null) {
    // Encerre a solicitação anterior
    this.request.abort();

    // Adicione um parâmetro fictício para anular o cache do navegador
    url += "?dummy=" + new Date().getTime();

    try {

      this.request.onreadystatechange =..................;

      this.request.open(..........., ........., true); // always asynchronous (true)

      if (type.toLowerCase() == "get") {
        // Envie uma solicitação GET; sem dados envolvidos

        this.request.send(........);

      } else {
        // Envie uma solicitação POST; o último argumento é dados

        this.request.setRequestHeader("Content-Type",......................);

        this.request.send(..............);

      }
    } catch(e) {
      alert("Ajax error communicating with the server.\n" + "Details: " + e);
    }
  }
}
```

handler

url

null

type

postDataType

postData

Solução dos ímãs de geladeira JavaScript

O objeto `AjaxRequest` personalizado integra o objeto `XMLHttpRequest` padrão, fornecendo uma interface muito mais simples para enviar as solicitações Ajax e lidar com suas respostas. O problema é, o método `send( )` do objeto `AjaxRequest` não tem algumas partes principais do código. Use os ímãs para terminar o código do método.

O método send( ) envia uma solicitação Ajax com detalhes explicados em seus argumentos.

```
AjaxRequest.prototype.send = function(type, url, handler, postDataType, postData) {
  if (this.request != null) {
    // Encerre a solicitação anterior
    this.request.abort();

    // Adiciona um parâmetro fictício para anular o cache do navegador
    url += "?dummy=" + (new Date()).getTime();

    try {
      this.request.onreadystatechange = handler;

      this.request.open(type, url, true); // always asynchronous (true)

      if (type.toLowerCase() == "get") {
        // Envie uma solicitação GET: sem dados envolvidos

        this.request.send(null);

      } else {
        // Envie uma solicitação POST: o último argumento é dados

        this.request.setRequestHeader("Content-Type", postDataType);

        this.request.send(postData);

      }
    } catch(e) {
      alert("Ajax error communicating with the server.\n" + "Details: " + e);
    }
  }
```

A função de sub-rotina personalizada será chamada para lidar com a resposta do servidor à solicitação.

O argumento type para send( ) determina se a solicitação é GET ou POST.

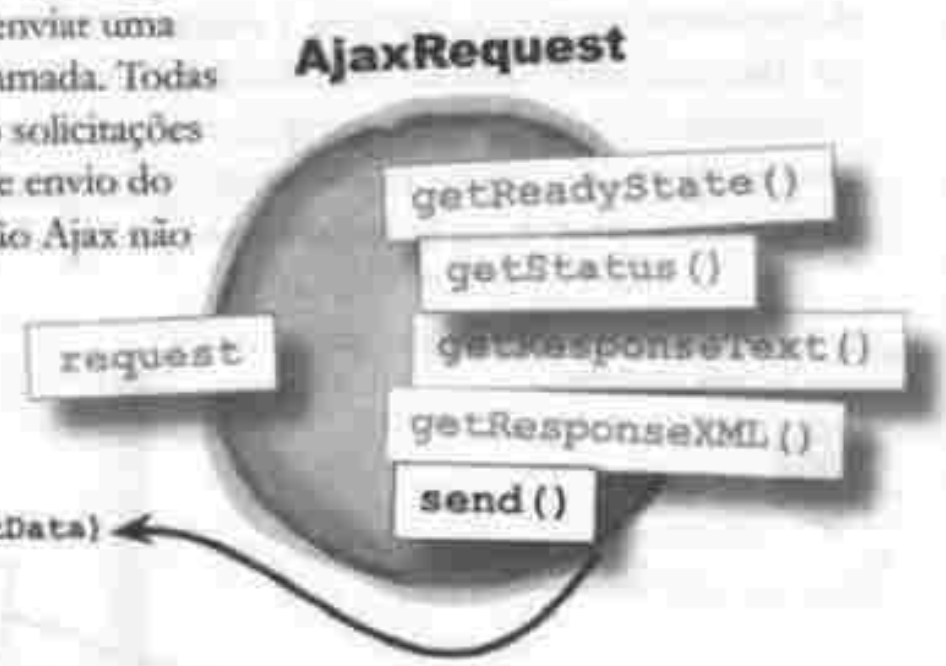
Os dados são enviados apenas para o servidor quando a solicitação é POST.

Este código é armazenado no arquivo JavaScript externo ajax.js junto com o construtor e outros métodos AjaxRequest.

ajax.js

Como dar sentido a uma solicitação Ajax

O objeto `AjaxRequest` personalizado consiste em um construtor e vários métodos, com um sendo particularmente útil. O método `send( )` é usado para preparar e enviar uma solicitação Ajax para um servidor em uma única chamada. Todas as solicitações Ajax enviadas usando `send( )` são solicitações GET ou POST, que correspondem às solicitações de envio do formulário HTML. A diferença é que uma solicitação Ajax não requer o carregamento completo de uma página.




```
send(type, url, handler, postDataType, postData)
```

type
O tipo da solicitação, GET ou POST.

url
O URL do servidor (blog.xml no caso do YouCube). Os dados poderão ser colocados nesse URL, se necessário.

postDataType
O tipo de dados sendo enviado (apenas para os POSTs, não requerido para os GETs).

handler
A função de callback usada para lidar com a resposta.

postData
Os dados a serem enviados (apenas para os POSTs, não requerido para os GETs). Os dados POST podem ser enviados em vários formatos diferentes.

Todas as solicitações Ajax envolvem estas mesmas partes da informação, embora as solicitações GET pulem os dois últimos argumentos, que são opcionais. Portanto, os três primeiros argumentos para `send( )` são os mais importantes e são suficientes para a as solicitações Ajax mais simples. Como exemplo, a seguinte chamada para `send( )` usa os três primeiros argumentos para solicitar os dados XML (GET) a partir de um arquivo chamado movies.xml no servidor:

Este código supõe que já criamos um objeto AjaxRequest e armazenamos na variável ajaxReq.

O tipo da solicitação.

```
ajaxReq.send("GET", "movies.xml", handleRequest);
```

O URL do arquivo de dados solicitado.

A função personalizada que será chamada para lidar com a resposta para a solicitação.

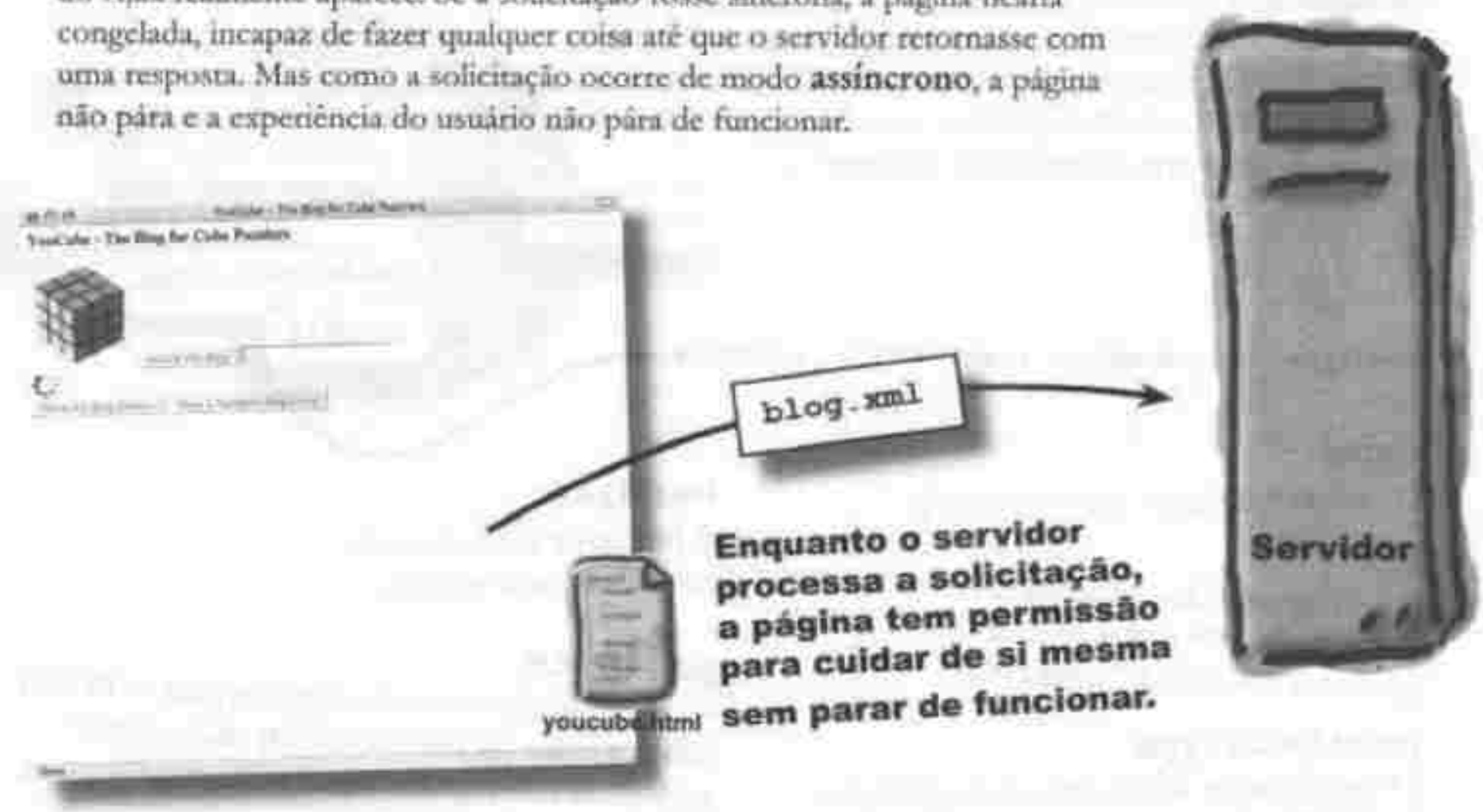
Relaxe

Não entre em pânico com o tratamento das solicitações.

Veremos os altos e baixos de como as solicitações Ajax são lidadas no código JavaScript personalizado em breve. No momento, apenas entenda que uma **função da sub-rotina de solicitação personalizada** deve ser definida para uma solicitação e que a função é chamada quando uma solicitação é completada.

É a vez da solicitação no servidor

Quando o método `send( )` é chamado em um objeto `AjaxRequest`, uma solicitação Ajax é enviada para o servidor e a página Web cuida de si, enquanto o servidor processa a solicitação. É onde a parte assíncrona do Ajax realmente aparece. Se a solicitação fosse síncrona, a página ficaria congelada, incapaz de fazer qualquer coisa até que o servidor retornasse com uma resposta. Mas como a solicitação ocorre de modo **assíncrono**, a página não pára e a experiência do usuário não pára de funcionar.



Só porque a página não fica congelada enquanto uma solicitação está sendo processada não significa necessariamente que o usuário pode de fato fazer algo produtivo. Tudo depende da página específica. No caso do YouCube, exibir com sucesso o blog depende inteiramente de obter os dados do blog de volta no servidor em uma resposta Ajax. Portanto, neste caso, a experiência do usuário está ligada à resposta Ajax.

Uma solicitação Ajax assíncrona ocorre sem congelar uma página enquanto ela espera que a solicitação seja processada por um servidor.

Pontos de Bala

- O objeto `XMLHttpRequest` é o objeto **padrão** para executar as solicitações Ajax, mas pode ser um pouco confuso de usar.
- O objeto `AjaxRequest` **personalizado** serve como um modo conveniente de usar o Ajax sem ter que lidar diretamente com o `XMLHttpRequest`.
- As solicitações Ajax têm dois tipos, `GET` ou `POST`, que são determinados pelos dados sendo enviados para o servidor, assim como os dados que afetam o servidor.
- O método `send( )` do objeto `AjaxRequest`.

Não existem Perguntas Idiotas

P: O objeto `AjaxRequest` é necessário para executar as solicitações Ajax?

R: Não. É perfeitamente correto usar o objeto `XMLHttpRequest` diretamente para enviar solicitações Ajax e lidar com suas respostas. Mas por que você usaria quando há um modo mais fácil graças ao objeto `AjaxRequest`? O objeto `AjaxRequest` não faz nada perturbador – é apenas um objeto conveniente que ajuda a simplificar a tarefa de usar o Ajax cuidando da "atividade" envolvida em montar as solicitações Ajax.

P: Como uma solicitação/resposta Ajax é diferente de uma solicitação/resposta HTTP?

R: As solicitações e as respostas HTTP são usadas pelos navegadores Web para recuperar as páginas Web HTML a partir dos servidores Web. As solicitações e respostas Ajax são muito parecidas com as solicitações e as respostas HTTP, exceto por algumas diferenças principais: as versões Ajax podem ocorrer a qualquer momento e não envolvem necessariamente a entrada de dados HTML. Na verdade, uma das grandes vantagens do Ajax é que ele pode ser usado para solicitar qualquer tipo de dado.

É ótimo que o Ajax possa lidar com qualquer tipo de dado, mas o tamanho dos dados também não importa em nada. O Ajax não está limitado a lidar com uma página inteira ou documento de dados por vez. Na verdade, ele é realmente equipado para enviar pequenas partes de dados. Fazendo isso, o Ajax permite que uma página modifique-se dinamicamente solicitando pequenas partes dos dados e, então, incorporando-os na página. E tudo isso ocorre sem a página nem mesmo ser recarregada.

P: Então, o Ajax torna possível montar dinamicamente uma página Web em partes?

R: Sim! Essa é a principal idéia por trás do Ajax. Porém, é mais do que apenas montar uma página a partir de partes. Também é **sincronizar** quando essa montagem ocorre. As solicitações e as respostas Ajax ocorrem em tempo real, geralmente sem interromper a utilização de uma página. Em outras palavras, os usuários não ficam aguardando que uma página inteira seja recarregada quando tudo o que precisa para ser atualizada é uma pequena seção da página. Essa seção da página pode ser carregada em "segundo plano" enquanto alguém continua a ler e a interagir com as outras partes da página.

P: Qual a relação de `GET` e `POST` com tudo isto?

R: `GET` e `POST` determinam as particularidades de como uma solicitação Ajax é lidada pelo servidor. Porém, não são diferentes em termos de serem capazes de solicitar dinamicamente qualquer tipo de dados a qualquer momento – todas as vantagens do Ajax aplicam-se a ambos os tipos de solicitação. A principal distinção entre `GET` e `POST` tem relação com o servidor passar ou não por uma mudança de estado com base nos dados como, por exemplo, armazená-los em um banco de dados. Se acontecer, um `POST` será solicitado. Do contrário, será o `GET`.

QUAL O MEU PROPÓSITO

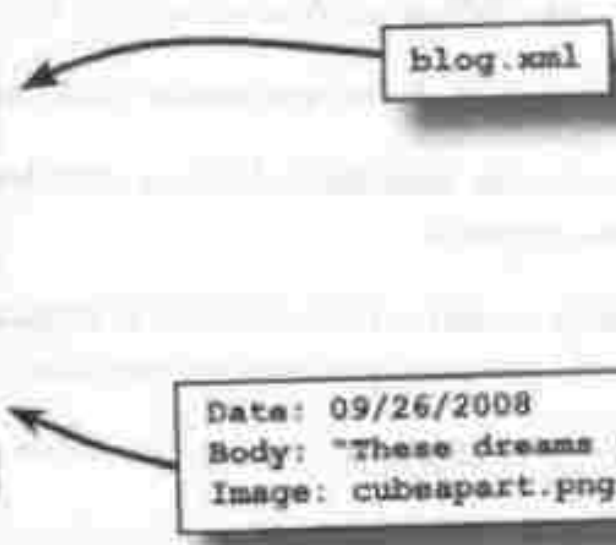
Combine cada parte do código relacionada ao Ajax ao que ela faz.

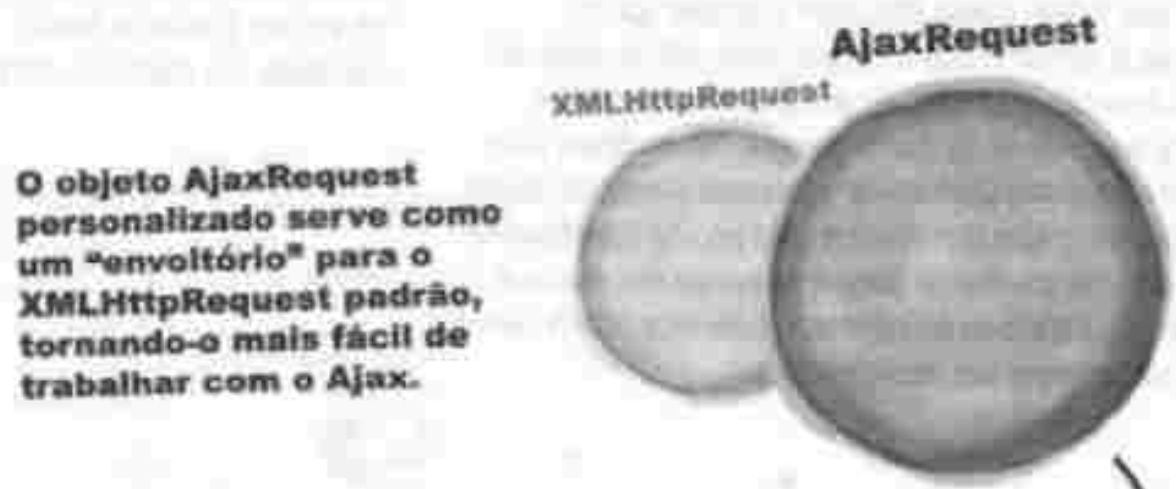
`XMLHttpRequest`	Recupera os dados sem mudar nada no servidor.
`GET`	Envia uma solicitação Ajax ao servidor, resultando em uma resposta.
`send( )`	Envia os dados para o servidor, resultando de algum modo em uma mudança no servidor.
`AjaxRequest`	O objeto JavaScript padrão que torna o Ajax possível.
`POST`	O objeto personalizado usado para simplificar as solicitações e as respostas Ajax.

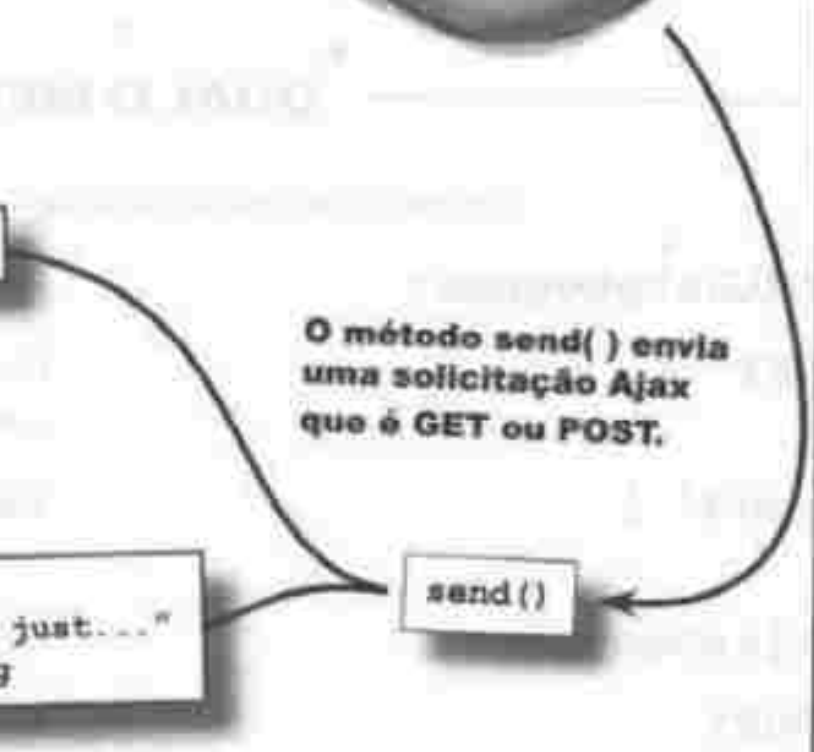
QUAL O MEU PROPÓSITO

Combine cada parte do código relacionada ao Ajax ao que ela faz.

`XMLHttpRequest`	Recupera os dados sem mudar nada no servidor.
`GET`	Envia uma solicitação Ajax ao servidor, resultando em uma resposta.
`send( )`	Envia os dados para o servidor, resultando de algum modo em uma mudança no servidor.
`AjaxRequest`	O objeto JavaScript padrão que torna o Ajax possível.
`POST`	O objeto personalizado usado para simplificar as solicitações e as respostas Ajax.







As páginas interativas iniciam com um objeto de solicitação

Independentemente de como o Ajax está sendo usado ou qual tipo de dados ele está tentando acessar, qualquer comunicação Ajax de dados inicia com uma solicitação. Portanto, a primeira tarefa de Ruby ao transformar o YouCube em um aplicativo baseado em dados é enviar uma solicitação Ajax para o arquivo XML que contém os dados do blog.

1. Crie um objeto `AjaxRequest`.
2. Envie uma solicitação GET para recuperar o arquivo blog.xml no servidor.
3. Lide com a solicitação...?

Ruby ainda não está certa sobre a etapa 3, mas ela pode se concentrar nas duas primeiras etapas agora.

Aponte seu Lápis

Escreve o código para criar um objeto `AjaxRequest` e, então, use-o para enviar uma solicitação para os dados do blog XML.

.....

.....

Aponte seu Lápis
Solução

Escreve o código para criar um objeto `AjaxRequest` e, então, use-o para enviar uma solicitação para os dados do blog XML.

```
1  var ajaxReq = new AjaxRequest( );

2  ajaxReq.send("GET", "blog.xml", handleRequest);
```

A solicitação Ajax é uma solicitação GET, uma vez que tudo que estamos fazendo é recuperar os dados no servidor.

O arquivo XML é especificado como o URL da solicitação

Nenhum deles significa muito até lidarmos com a resposta na função handleRequest( ) personalizada.

Chame quando tiver terminado

Assim que uma solicitação Ajax é enviada, o papel do navegador muda – ele não está aguardando uma resposta do servidor. Mas como as solicitações Ajax são geralmente executadas de modo **assíncrono**, o usuário pode continuar a interagir com a página enquanto o navegador aguarda a resposta internamente. Em outras palavras, a solicitação Ajax não pára a página enquanto a solicitação está sendo processada no servidor. Assim que a solicitação termina de ser processada no servidor, sua resposta é lidada no código JavaScript usando uma função de callback, a **sub-rotina da solicitação**.

O script do cliente lida com a resposta para uma solicitação Ajax usando uma função de callback personalizada.

A resposta é enviada do servidor para o navegador, que então conta com uma função de callback personalizada para lidar com a solicitação.

1. ~~Crie um objeto AjaxRequest.~~
2. ~~Envie uma solicitação GET para recuperar o arquivo blog.xml no servidor.~~
3. Lide com a solicitação.

A função de callback handleRequest( ) é inteiramente personalizada e deve ser fornecida pelo script.

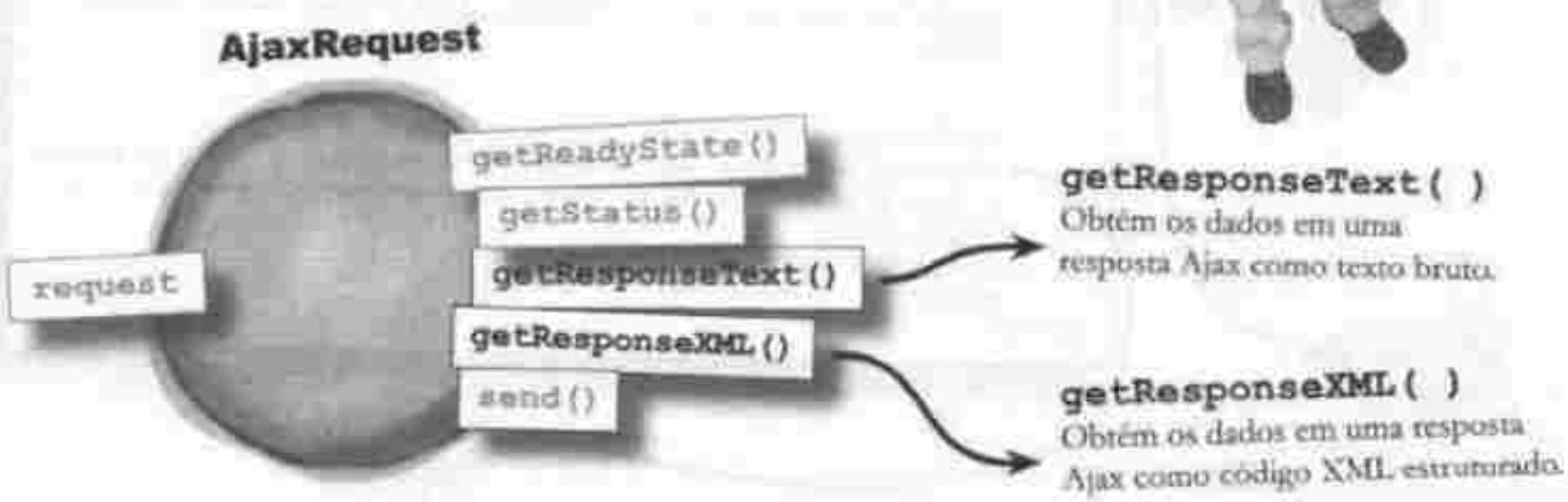
Como lidar com uma resposta... uniformemente

A função de callback da sub-rotina de solicitação personalizada, `handleRequest( )`, neste caso, é chamada assim que uma solicitação Ajax termina. Além de sinalizar que uma solicitação terminou com sucesso, o serviço desta função é tomar uma ação com base nos dados de resposta retornados pelo servidor.

Entendo que a sub-rotina de solicitação é chamada para cuidar da resposta Ajax, mas como ela acessa os dados da resposta?

Os métodos do objeto AjaxRequest têm acesso aos dados de resposta Ajax.

A função de sub-rotina da solicitação fornece acesso aos dados retornados em uma resposta Ajax através de dois métodos do objeto AjaxRequest, `getResponseText( )` e `getResponseXML( )`.



Apenas um desses métodos tem acesso aos dados viáveis para qualquer determinada resposta, significando que o formato dos dados determina qual método deve ser usado. Portanto, `getResponseXML( )` deverá ser usado se os dados da resposta forem XML, neste caso `getResponseText( )` não retornará dados significativos. O mesmo ocorrerá no inverso se os dados forem texto bruto, em oposição ao código XML estruturado.

Poder do cérebro

Sabendo que o código XML é estruturado de modo muito parecido com o código HTML, você poderia acessar os dados do blog XML na sub-rotina de solicitação?

DOM para seu resgate

As habilidades de quebra-cabeças de Ruby são certamente recompensadas porque ela está certa sobre a idéia de usar o DOM para processar os dados da resposta XML. O DOM é para manipular os dados HTML como uma árvore de nós. Mas não há nada específico do HTML sobre o DOM, que significa que também pode ser usado para trabalhar com o XML como uma árvore de nós. Ruby apenas tem que considerar os dados do blog XML de seu YouCube em termos de **nós**.

Ruby precisa ser capaz de extrair o conteúdo dos nós dos dados XML, que é uma tarefa DOM repetitiva mais adequada para uma função. Não fará nenhum sentido adicionar muito código duplicado ao YouCube se puder ser evitado.

Um foco na função getText( )

A função `getText( )` personalizada lida com o trabalho chato de repetir um elemento (nó) na árvore DOM e extrair todo seu conteúdo.

```
function getText(elem) {
  var text = "";
  if (elem) {
    if (elem.childNodes) {
      for (var i = 0; i < elem.childNodes.length; i++) {
        var child = elem.childNodes[i];
        if (child.nodeValue)
          text += child.nodeValue;
        else {
          if (child.childNodes)
            if (child.childNodes[0].nodeValue)
              text += child.childNodes[0].nodeValue;
        }
      }
    }
  }
  return text;
}
```

O argumento do elemento contém o elemento cujo conteúdo será extraído.

Faça um loop em todos os nós-filhos do elemento.

Adicione o conteúdo do filho à variável text.

Se houver nós-filhos em um nó-filho, simplesmente obtenha o conteúdo do texto do primeiro e prossiga.

Retorne a variável text, que contém o conteúdo-filho de todo elemento.

Aponte seu Lápis

Sabendo que os dados de resposta XML já foram armazenados em uma variável chamada `xmlData`, escreva o código para definir a assinatura do blog YouCube para o nome armazenado na tag XML `<author>`.

Aponte seu Lápis
Solução

Sabendo que os dados de resposta XML já foram armazenados em uma variável chamada `xmlData`, escreva o código para definir a assinatura do blog YouCube para o nome armazenado na tag XML `<author>`.

```
Blog.prototype.signature = "by " + getText(xmlData.getElementsByTagName("author")[0]);
```

Deve haver apenas uma tag <author> nos dados XML, portanto simplesmente obtenha a primeira.

Use a função auxiliar getText() personalizada para extrair o conteúdo do texto da tag <author>.

A assinatura é uma propriedade da classe, portanto deve ser definida através do objeto prototype do Blog.

Como lidar com as respostas Ajax

Entrevista desta semana:

A sub-rotina de solicitação Ajax handleRequest( ) confessa

Use a Cabeça: Ouvimos dizer que você é muito boa ao responder às solicitações Ajax. Em que isto implica?

handleRequest( ): Quando uma solicitação Ajax chega, começo a lidar com a resposta, que geralmente contém dados enviados do servidor. Meu trabalho é primeiro assegurar que a solicitação tenha sido bem executada no servidor e se foi, então, abro caminho nos dados e cuido de sua integração na página Web, se necessário.

Use a Cabeça: Então, você é realmente chamada quando uma solicitação termina?

handleRequest( ): Ah, sim. Na verdade, sou chamada várias vezes no processo de solicitação, mas na maioria das vezes as pessoas estão interessadas apenas quando faço algo bem no final.

Use a Cabeça: Entendo. E como você sabe quando é o final?

handleRequest( ): Bem, o objeto AjaxRequest tem alguns métodos que posso chamar para verificar o estado e o status da solicitação para assegurar que tenha terminado sem nenhum problema.

Use a Cabeça: Assim que isso aconteca, como você sabe o que fazer?

handleRequest( ): Bem, realmente não é comigo. Lembre-se, sou uma função personalizada, portanto sou diferente em cada aplicativo.

Use a Cabeça: Por que é assim?

handleRequest( ): Porque aplicativos diferentes usam os dados da resposta de modos diferentes – é inteiramente específico do aplicativo. E eu também.

Use a Cabeça: Espere um minuto, você quer dizer que alguém reescreve você novamente para cada aplicativo?

handleRequest( ): Correto. Apenas faz sentido porque um aplicativo do carrinho de compras processaria suas respostas Ajax de modo muito diferente de um blog, por exemplo. O Ajax assegura que eu seja chamada assim que o servidor termina com a solicitação e a partir desse momento, tudo é completamente personalizado.

Use a Cabeça: Então, parte da construção de uma página Web acionada pelo Ajax é criar uma sub-rotina de solicitação personalizada?

handleRequest( ): Absolutamente. É onde grande parte do trabalho real ocorre em um aplicativo Ajax.

Use a Cabeça: Isso é muito esclarecedor. Agradeço.

handleRequest( ): Sempre fico feliz em responder.

SEJA o anotador JavaScript

Sua função é fingir ser o anotador JavaScript e adicionar muitas anotações para explicar exatamente o que está ocorrendo na função handleRequest( ). 7 é um número mágico – existem 7 coisas que levam a uma solicitação bem-sucedida?

```
function handleRequest() {
  if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
    // Armazene os dados da resposta XML
    var xmlData = ajaxReq.getResponseXML().getElementsByTagName("blog")[0];

    // Defina a assinatura de todo o blog
    Blog.prototype.signature = "by " + getText(xmlData.getElementsByTagName("author")[0]);

    // Crie o array de objetos de entrada Blog
    var entries = xmlData.getElementsByTagName("entry");
    for (var i = 0; i < entries.length; i++) {
      // Crie a entrada do blog
      blog.push(new Blog(getText(entries[i].getElementsByTagName("body")[0]),
        new Date(getText(entries[i].getElementsByTagName("date")[0])),
        getText(entries[i].getElementsByTagName("image")[0])));
    }

    // Exiba o blog
    showBlog(5);
  }
}
```

SEJA o anotador JavaScript Solução

Sua função é fingir ser o anotador JavaScript e adicionar muitas anotações para explicar exatamente o que está ocorrendo na função handleRequest( ). 7 é um número mágico – existem 7 coisas que levam a uma solicitação bem-sucedida?

Defina a assinatura do blog para o conteúdo na tag <author>.

Certifique-se de que a solicitação Ajax seja completada com sucesso verificando seu estado e seu status.

Há apenas uma tag <blog> nos dados XML, portanto obtenha o primeiro item no array retornado por getElementsByTagName( ).

```
function handleRequest() {
  if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
    // Armazene os dados da resposta XML
    var xmlData = ajaxReq.getResponseXML().getElementsByTagName("blog")[0];

    // Defina a assinatura de todo o blog
    Blog.prototype.signature = "by " + getText(xmlData.getElementsByTagName("author")[0]);

    // Crie o array de objetos de entrada Blog
    var entries = xmlData.getElementsByTagName("entry");
    for (var i = 0; i < entries.length; i++) {
      // Crie a entrada do blog
      blog.push(new Blog(getText(entries[i].getElementsByTagName("body")[0]),
        new Date(getText(entries[i].getElementsByTagName("date")[0])),
        getText(entries[i].getElementsByTagName("image")[0])));
    }

    // Exiba o blog
    showBlog(5);
  }
}
```

Crie um novo objeto do blog para a entrada e adicione-o ao final do array usando o método push( ) do objeto Array.

Obtenha todos os elementos da entrada que mantêm as entradas do blog. Não é necessário obter o elemento entries quando podemos ir diretamente para cada entrada.

Chame a função showBlog( ) para exibir as cinco últimas entrada do blog na página.

1. ~~Crie um objeto AjaxRequest.~~
2. ~~Envie uma solicitação GET para recuperar o arquivo blog.xml no servidor.~~
3. ~~Lide com a solicitação.~~ *Feito!*

O YouCube é conduzido por seus dados

Ruby está vibrando com a renovação Ajax do YouCube (ela está economizando uma grande quantidade de tempo), mas ela tem uma incômoda preocupação relacionada ao que acontece na página enquanto os dados do blog estão sendo carregados.

As últimas versões dos arquivos YouCube estão disponíveis em www.altabooks.com.br.

Agora, o blog é conduzido pelos dados XML...

...Mas alguns usuários estão confusos com a página em branco que aparece enquanto os dados estão carregando.

Aponte seu Lápis

Escreva a linha de código que falta na função `loadBlog()` que exibe uma imagem de "espera" denominada wait.gif enquanto os dados do blog estão sendo carregados.

Sugestão: Use o `div` principal do blog, com um ID "`blog`".

```
function loadBlog() {
  ..................................................................
  ajaxReq.send("GET", "blog.xml", handleRequest);
}
```

Aponte seu Lápis
Solução

Escreva a linha de código que falta na função `loadBlog( )` que exibe uma imagem de "espera" denominada wait.gif enquanto os dados do blog estão sendo carregados.

Sugestão: Use o `div` principal do blog, com um ID "`blog`".

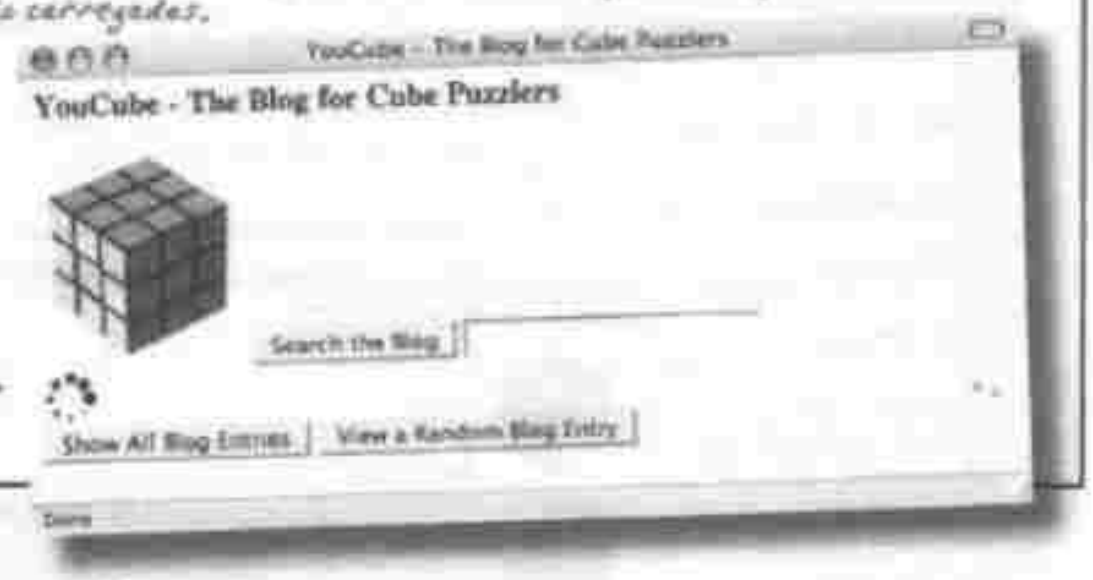
```
function loadBlog() {
  document.getElementById("blog").innerHTML = "<img src='wait.gif' alt='Loading...' />";

  ajaxReq.send("GET", "blog.xml", handleRequest);
}
```

innerHTML é mais fácil de usar do que o DOM neste caso, uma vez que estamos adicionando uma tag de imagem com alguns atributos.

A imagem de espera substitui completamente o conteúdo do blog enquanto os dados do blog estão sendo carregados.

A imagem wait.gif animada é usada no lugar das entradas do blog para deixar que o usuário saiba que os dados do blog estão sendo carregados.

wait.gif



Não existem Perguntas Idiotas

P: A última entrada do blogYouCube tinha uma tag `<strong>` HTML. Como isso é possível no código XML?

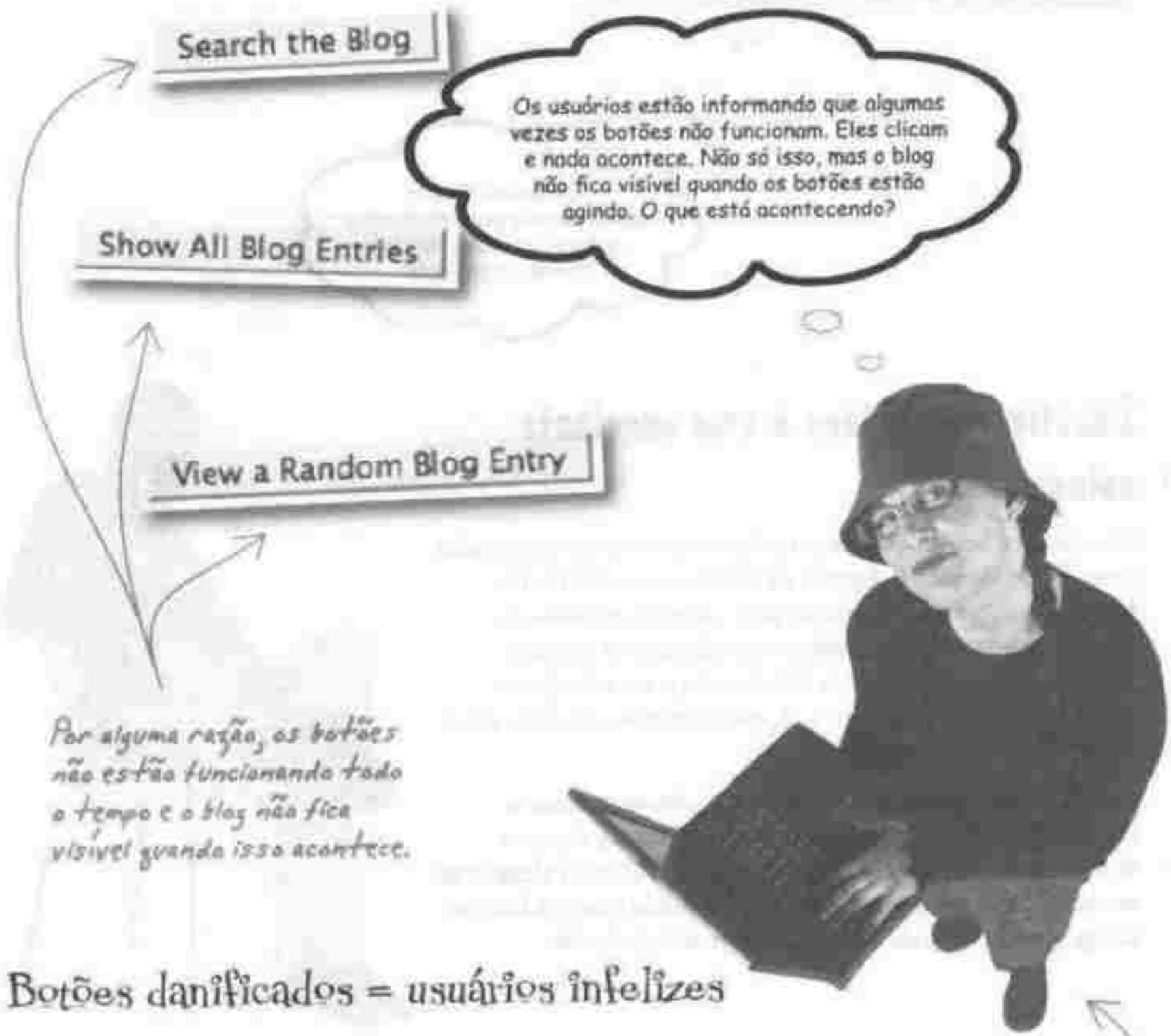
R: Lembre-se de que o código XML pode ser usado para representar qualquer tipo de dados. Neste caso, sabendo que o corpo de uma entrada do blog está sendo injetado em uma página Web, é tecnicamente possível incluir tags HTML que afetem como o corpo aparece na página. Em outras palavras, o conteúdo do corpo de uma determinada entrada do blog pode conter tags HTML que são transmitidas como nós de formatação especial no código XML. Porém, é uma perspectiva bem capciosa, uma vez que teríamos que reconstruir os nós de formatação HTML no código HTML para a página ao injetar os dados XML na página. Ao invés de percorrer esse caminho, o código YouCube escolhe apenas retirar o conteúdo do texto de qualquer tag HTML, deixando a formatação para trás. Ruby ainda está livre para adicionar tags de formatação HTML ao conteúdo do blog, possivelmente para uma futura versão do YouCube, mas elas são ignoradas para a formatação. Seu texto permanece, que é uma boa coisa.

P: Como funciona o estado e o status de pronto de uma resposta Ajax?

R: Estas duas propriedades, basicamente, vêm do objeto `XMLHttpRequest` e seu trabalho é controlar o **estado** da solicitação como, por exemplo, (0) inicializado ou (4) carregado, assim como o **status** da solicitação como, 404 (não encontrado) ou 200 (OK). Certamente é possível controlar de perto essas propriedades, mas não é necessário. Tudo que você precisa saber é que uma solicitação Ajax completou com sucesso, caso o estado seja 4 (carregado) e o status seja 200 (OK). É por isso que a função `handleRequest( )` entrará apenas em ação se ambas as condições forem satisfeitas.

Botões disfuncionais

Embora a revisão Ajax do YouCube tenha ocorrido basicamente de modo interno, fora da visão dos usuários YouCube, aparentemente há um problema da interface do usuário que aparece. Mais especificamente, parece que os botões na página são estão funcionando bem como deveriam.



Poder do cérebro

Por que os botões do blog não estão funcionando? Quando você acha que está ocorrendo o problema no processo de carregamento da página?

Ruby está calma, mas ele realmente precisa chegar à raiz deste problema.

Os botões precisam de dados

O problema com os botões YouCube é que eles são aplicáveis apenas quando os dados do blog estão disponíveis. E como os dados do blog agora são carregados a partir de um arquivo XML externo, haverá um período de tempo, geralmente muito breve, no qual as páginas não têm dados. Durante esse período, os botões não terão sentido e apenas estarão confundindo os usuários.

Desativar os botões é uma excelente solução.

Desativar os botões enquanto os dados do blog são carregados é um modo simples e elegante de resolver o problema do botão. Como a solicitação Ajax para carregar os dados do blog é enviada quando a página é carregada pela primeira vez, os botões podem iniciar desativados e ser ativados na função `handleRequest( )`, que é quando sabemos que a solicitação Ajax terminou.

Para realmente executar a desativação, precisamos usar o atributo `disabled` da tag `<input>`. Essa tag deve ser definida para "`disabled`" no código HTML para **desativar** um botão. Em oposição, ela deve ser definida para `false` no código JavaScript para **ativar** um elemento do botão.

```
<input type="button" value="Search the Blog"
  disabled="disabled" />
```

Search the Blog

Search the Blog

```
buttonElem.disabled = false;
```

Ímãs de geladeira JavaScript

Use os ímãs para terminar o código na página YouCube para que os botões do blog sejam desativados até que os dados do blog terminem de ser carregados. Você precisará usar alguns desses ímãs mais de uma vez.

```
<html>
  <head>
    <title>YouCube - The Blog for Cube Puzzlers</title>
    <script type="text/javascript" src="ajax.js"> </script>
    <script type="text/javascript" src="date.js"> </script>

    <script type="text/javascript">
      ...
      function handleRequest() {
        if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
          ...
          // Ative os botões do blog

          document.getElementById(..............). ..................=............;

          document.getElementById(..............). ..................=............;

          document.getElementById(..................). ..................=............;


          ...
        }
      }
      ...
    </script>
  </head>
  <body onload="loadBlog();">
    <h1>YouCube - The Blog for Cube Puzzlers</h1>
    <img src="cube.png" alt="YouCube" />
    <input type="button" id="search" value="Search the Blog"

                                  ..................onclick="searchBlog();" />
    <input type="text" id="searchtext" name="searchtext" value="" />
    <div id="blog"></div>
    <input type="button" id="showall" value="Show All Blog Entries"

                                  ..................onclick="showBlog();" />
    <input type="button" id="viewrandom" value="View a Random Blog Entry"

                                  ..................onclick="randomBlog();" />
  </body>
</html>
```

true

"search"

"viewrandom"

disabled

false

"showall"

"disabled"

Solução dos Ímãs de geladeira JavaScript

Use os ímãs para terminar o código na página YouCube para que os botões do blog sejam desativados até que os dados do blog terminem de ser carregados. Você precisará usar alguns desses ímãs mais de uma vez.

```
<html>
  <head>
    <title>YouCube - The Blog for Cube Puzzlers</title>
    <script type="text/javascript" src="ajax.js"> </script>
    <script type="text/javascript" src="date.js"> </script>
    <script type="text/javascript">
      ...
      function handleRequest() {
        if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
          ...
          // Ativa os botões do blog
          document.getElementById("search").disabled = false;
          document.getElementById("showall").disabled = false;
          document.getElementById("viewrandom").disabled = false;

          ...
        }
      }
      ...
    </script>
  </head>
  <body onload="loadBlog();">
    <h1>YouCube - The Blog for Cube Puzzlers</h1>
    <image src="cube.png" alt="YouCube" />
    <input type="button" id="search" value="Search the Blog"
      disabled="disabled" onclick="searchBlog();" />
    <input type="text" id="searchtext" name="searchtext" value="" />
    <div id="blog"></div>
    <input type="button" id="showall" value="Show All Blog Entries"
      disabled="disabled" onclick="showBlog();" />
    <input type="button" id="viewrandom" value="View a Random Blog Entry"
      disabled="disabled" onclick="randomBlog();" />
  </body>
</html>
```

Como economizar tempo com os acréscimos do blog baseados na Web

Agora, o YouCube é conduzido por dados dinâmicos, mas Ruby ainda tem que aproveitar totalmente a recompensa. A verdadeira vantagem dos dados dinâmicos no YouCube não será completamente compreendida até que ela tenha a capacidade de usar uma **interface baseada na Web** para adicionar as entradas do blog. Ao invés de editar um arquivo XML para adicionar ao blog, ela deseja ser capaz de simplesmente fornecer uma nova entrada em uma página Web e salvá-la no servidor.

Fico cansada de editar os arquivos e, então, enviá-los com FTP para o servidor só para atualizar meu blog. Gostaria de atualizar o YouCube a partir de meu próprio navegador!

Editar código + Fazer upload dos arquivos = Nenhuma diversão!

Ruby imagina uma página Web só para ela, que lhe permita enviar uma nova entrada do blog simplesmente preenchendo um formulário. Ela poderia sempre estar a um clique de atualizar seu blog e tudo que precisaria é de um navegador. Nenhum editor de texto, nenhum cliente FTP, apenas seu entusiasmo pelo quebra-cabeça de cubo.

YouCube – Adding to the Blog for Cube Puzzlers

YouCube - Adding to the Blog for Cube Puzzlers

Date: 10/04/2008

Body: I'm really looking forward to this puzzle party at the end of the month.

Image (optional):

Add the New Blog Entry

Adicionar uma nova entrada do blog é tão fácil quanto preencher os campos e clicar um botão!

Done

A nova página do blog para adicionar usa os campos do formulário para as três partes dos dados de entrada do blog.

Poder do cérebro

Como o Ajax poderia ser usado para adicionar as entradas do blog XML através de uma interface do usuário da página Web?

Como escrever os dados do blog

Ao considerar o acréscimo do blog em termos de Ajax, é possível imaginar uma solicitação POST Ajax que envia os novos dados de entrada do blog para o servidor, depois disso, o servidor grava os dados no arquivo `blog.xml` como uma nova entrada do blog. A resposta Ajax realmente não precisa fazer nada neste caso, uma vez que não há nada para retornar.

O JavaScript não é a resposta para gravar em um arquivo no servidor.

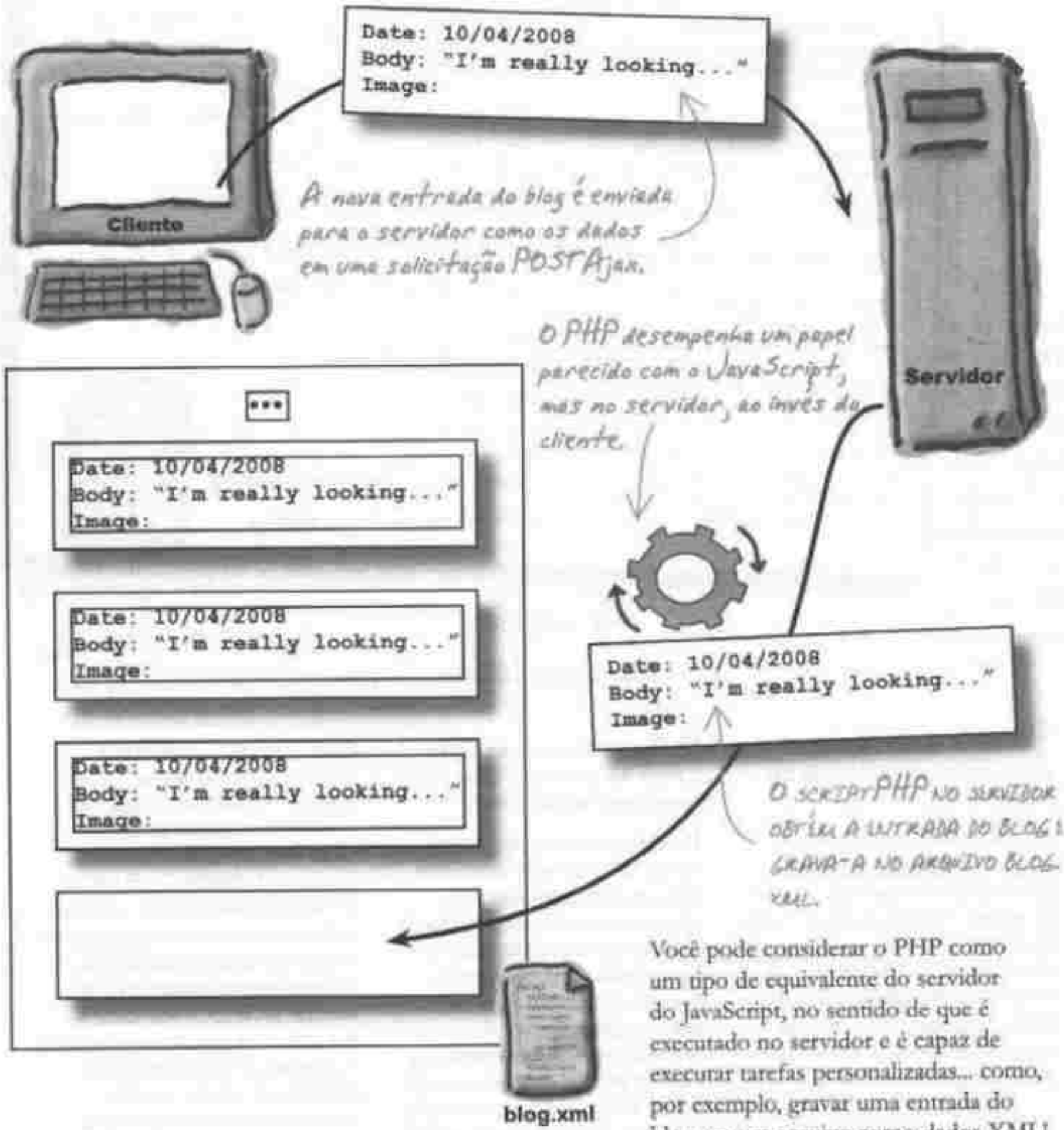
O JavaScript não é uma opção para gravar no arquivo blog.xml no servidor. Na verdade, você não pode nem executar o código JavaScript no servidor. É porque o JavaScript é uma tecnologia do cliente designada a ser executada unicamente nos navegadores Web. Neste caso, em particular, o JavaScript não ajuda porque precisamos gravar um arquivo no servidor. É um problema comum, sendo por isso que as tecnologias no lado do servidor são geralmente usadas junto com o JavaScript.

O que precisamos é de uma tecnologia parecida com o JavaScript, mas unicamente para fazer as coisas no servidor. Existem várias opções, mas uma é lembrada, pois não é complicada demais e funciona surpreendentemente bem com os dados XML...

PHP para o resgate... desta vez

Uma linguagem de script chamada PHP oferece tudo que precisamos para gravar os dados do blog em um arquivo XML no servidor. A tarefa real envolvida é ler o arquivo XML, então adicionar a nova entrada do blog às entradas existentes e depois gravar todas as entradas do blog de volta no arquivo original. Mas tudo tem sua origem no recebimento dos novos dados de entrada do blog no servidor como uma solicitação Ajax a partir do navegador do cliente.

O PHP é uma tecnologia de script que pode executar tarefas no servidor.



Você pode considerar o PHP como um tipo de equivalente do servidor do JavaScript, no sentido de que é executado no servidor e é capaz de executar tarefas personalizadas... como, por exemplo, gravar uma entrada do blog em um arquivo como dados XML!

PHP Saindo do Forno

No lado do servidor do YouCube, um script PHP lida com os detalhes de adicionar uma nova entrada do blog aos dados do blog XML que são armazenados no arquivo blog.xml.

Verifique para saber se o arquivo do blog existe.

Carregue os dados XML brutos na variável $rawBlog.

```
<?php
$filename = "blog.xml";

if (file_exists($filename)) {
  // Carregue as entradas do blog a partir do arquivo XML
  $rawBlog = file_get_contents($filename);
}
else {
  // Crie um documento XML vazio
  $rawBlog = "<?xml version=\"1.0\" encoding=\"utf-8\" ?>";
  $rawBlog .= "<blog><title>YouCube - The Blog for Cube Puzzlers</
title>";
  $rawBlog .= "<author>Puzzler Ruby</author><entries></entries></blog>";
}
$xml = new SimpleXmlElement($rawBlog);

// Adicione a nova entrada do blog como um nó-filho
$entry = $xml->entries->addChild("entry");
$entry->addChild("date", $_REQUEST["date"]);
$entry->addChild("body", stripslashes($_REQUEST["body"]));
if ($_REQUEST["image"] != "")
  $entry->addChild("image", $_REQUEST["image"]);

// Grave o blog inteiro no arquivo
$file = fopen($filename, "w");
fwrite($file, $xml->asXML());
fclose($file);
?>
```

Se o arquivo do blog não existir, crie um documento do blog XML vazio.

Adicione a nova entrada do blog como um nó-filho dentro da estrutura de dados XML.

Converta os dados brutos do blog em uma estrutura de dados XML, que é muito parecida com uma árvore DOM no JavaScript.

Sobrescreva o arquivo do blog com os novos dados do blog.

addblogentry.php

Este script PHP é armazenado no arquivo addblogentry.php.

Não existem Perguntas Idiotas

P: Tenho que usar o PHP para gravar os arquivos no servidor?

R: Não, absolutamente. Existem todos os tipos de tecnologia para gravar os scripts do servidor. Você tem os servlets Perl (CGI) e Java para citar alguns e eles podem fazer as mesmas coisas que o PHP. Portanto, caso se sinta mais confortável com uma dessas outras tecnologias, use-a para criar o componente do lado do servidor de seus aplicativos Ajax.

P: Posso ter sucesso usando o Ajax sem ter que usar um programa no servidor?

R: Em alguns casos sim, mas na maioria não. Lembre-se de que tudo, exceto as solicitações Ajax mais simples, envolve o servidor recebendo dados do cliente e, então, fazendo algo com eles como, por exemplo, pesquisar algo em um banco de dados ou gravar algo em um arquivo ou banco de dados. A página do blog YouCube principal é um bom exemplo de uma solicitação Ajax que é simples o bastante para não requerer nenhum script do servidor. A maioria dos aplicativos Ajax não tem tanta sorte, portanto na maioria dos casos, você precisará fazer alguma codificação no servidor. O problema real é se o servidor pode ou não apenas retornar um arquivo inteiro, como é o caso com o blog.xml, ou se ele tem que processar de alguma maneira os dados e fazer algo com eles no servidor como, por exemplo, gravá-los. A boa notícia é que os tipos de scripts requeridos no servidor para muitos aplicativos Ajax são bem simples e podem ser calculados sem um domínio da tecnologia de script do servidor.

O PHP tem necessidades também

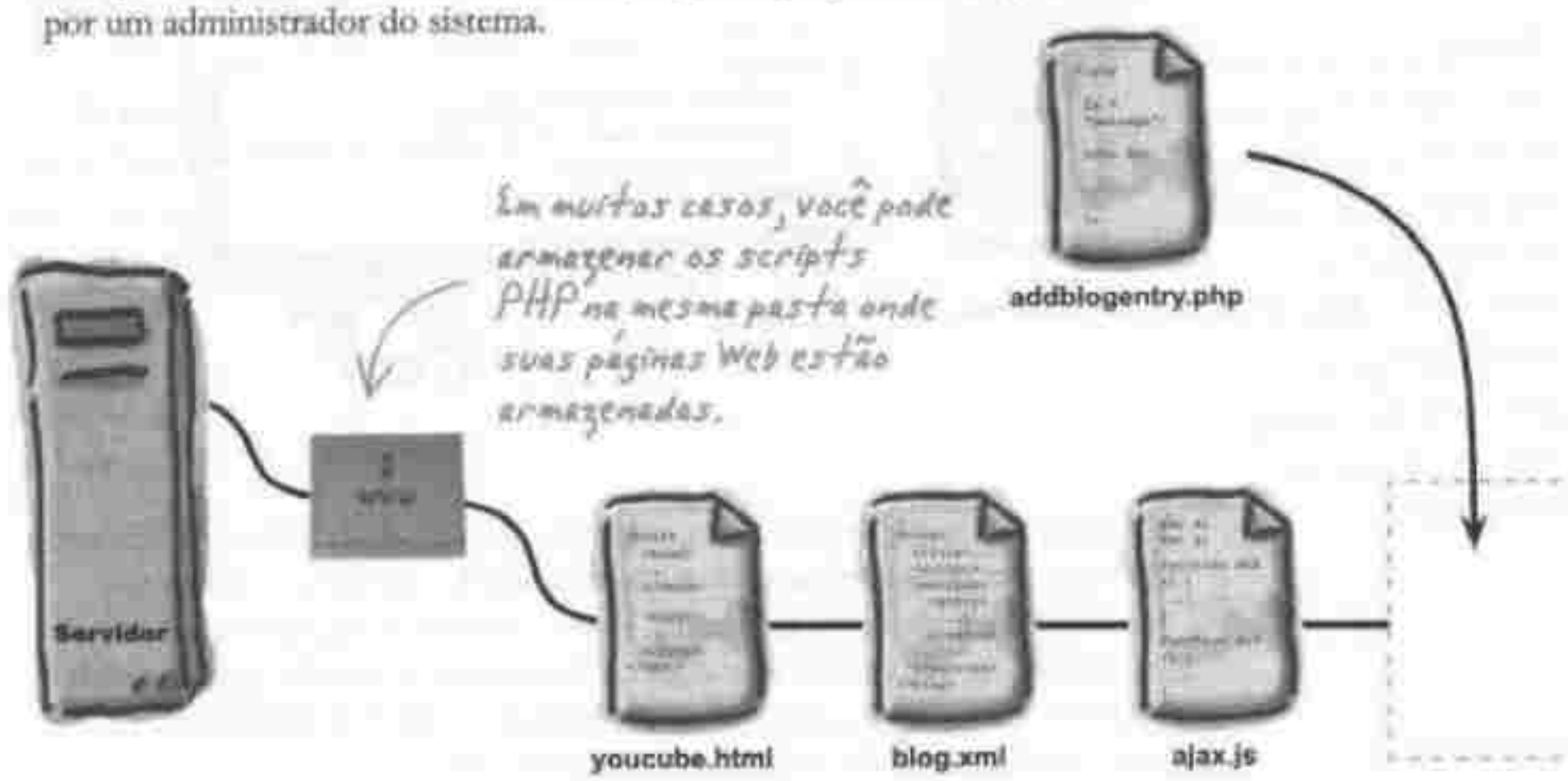
Diferente do JavaScript, que é suportado inerentemente nos navegadores modernos, o suporte PHP não é sempre uma conclusão prévia no servidor. Portanto, antes de enviar arquivos PHP para seu servidor Web, provavelmente vale a pena verificar com seu administrador do sistema ou serviço de host Web para saber se o PHP é suportado. Se não for, você precisará fazer tudo que puder para adicioná-lo ou possivelmente encontrar um servidor Web diferente. O script PHP para o YouCube simplesmente não funcionará, a menos que o PHP seja suportado no servidor.

Executar um script PHP pode requerer alguns ajustes em seu servidor Web.

Certifique-se de que servidor Web suporte o PHP.

Mesmo que não, você poderá ser capaz de instalá-la sozinha ou persuadir um administrador a instalá-lo. Definitivamente, você precisará dele para o Ajax!

O suporte para o PHP em seu servidor Web é o primeiro obstáculo. O segundo é descobrir onde colocar os arquivos PHP no servidor. Em muitos casos, é bom colocar os arquivos PHP na mesma pasta de suas páginas Web HTML e arquivos JavaScript externos. Contudo, algumas instalações PHP são mais exigentes e requerem que os scripts PHP sejam armazenados em uma pasta especial. Novamente, é uma pergunta que pode ser respondida por um administrador do sistema.



Como fornecer dados ao script PHP

Com o PHP funcionando no servidor e o arquivo do script PHP no lugar, podemos examinar mais de perto do que o script PHP precisa para gravar os dados em um arquivo XML no servidor. Isso ajudará a chegar a uma construção para a solicitação Ajax que forneça ao servidor exatamente o que ele precisa para executar a tarefa.

O script PHP está esperando dados para uma nova entrada do blog, que sabemos que consiste em pelo menos duas partes de informação e potencialmente três.

Os dados são fornecidos ao script PHP através de uma solicitação Ajax.

Date

A data da entrada do blog

Body

O texto do corpo da entrada do blog

Image

Uma imagem opcional para a entrada do blog

```
Date: 10/04/2008
Body: "I'm really looking..."
Image:
```

O código JavaScript do cliente deve colocar os dados em um formato que possa ser enviado para o servidor como parte de uma solicitação Ajax.

O trabalho do servidor é receber a solicitação Ajax e fornecer seus dados ao script PHP para o processamento.

Estas informações devem ser reunidas de algum modo e enviadas para o servidor como uma solicitação Ajax, onde serão processadas e gravadas no arquivo blog.xml.

blog.xml

```
<entry>
  <date>10/04/2008</date>
  <body>I'm really looking...</body>
  <image></image>
</entry>
```

O script PHP cuida da conversão da entrada do blog no XML e da gravação no arquivo blog.xml

Neste ponto, a nova entrada do blog foi adicionada ao arquivo blog.xml e aparecerá no blog YouCube automaticamente na próxima vez em que a página do blog for carregada ou renovada.

O desafio é, então, sugerir uma construção para a página Web de acréscimo do blog que apresente primeiro uma interface do usuário para fornecer uma nova entrada do blog e, então, reúna essas informações e envia-as para o servidor em uma solicitação Ajax. A boa notícia é que não precisaremos realmente fazer nada em resposta à solicitação, a não ser talvez confirmar que a nova entrada do blog foi gravada com sucesso.

Aponte seu Lápis

Projete a construção da página Web de acréscimo da entrada do blog YouCube, mostrando exatamente como a solicitação Ajax e a resposta decompõem-se no fluxo de dados.

Aponte seu Lápis
Solução

Projete a construção da página Web de acréscimo da entrada do blog YouCube, mostrando exatamente como a solicitação Ajax e a resposta decompõem-se no fluxo de dados.

A solicitação Ajax é uma solicitação POST que consiste nas seguintes partes de dados:

* Data do blog
* Corpo do blog
* Imagem do blog (conteúdo opcional)

A página Web de acréscimo do blog tem campos do formulário para fornecer novos dados de entrada do blog.

YouCube - Adding to the Blog for Cube Puzzlers

YouCube - Adding to the Blog for Cube Puzzlers

Date: 10/04/2008
Body: I'm really looking forward to this puzzle party at the end of the month.
Image (optional):
Add the New Blog Entry

onclick!

Done

A data do blog é enviada para o servidor como dados na solicitação POST.

```
Date: 10/04/2008
Body: "I'm really looking..."
Image:
```

Clicar o botão Add faz com que a solicitação Ajax de acréscimo do blog seja enviada.

The new blog entry was successfully added.

OK

blog.xml

O cliente notifica o usuário que a nova entrada do blog foi adicionada com sucesso.

A resposta Ajax não retorna nenhum dado porque o cliente não precisa de nada de volta.

Servidor

O servidor grava a nova entrada do blog como dados XML no arquivo blog.xml.

Como se preparar: Como enviar dados do blog para o servidor

Uma solicitação POST Ajax é um pouco mais complicada do que uma solicitação GET porque requer enviar dados para o servidor. Embora a solicitação POST suporte diferentes maneiras de reunir os dados para o servidor, a técnica confiável de **codificar com o URL** os campos dos dados funciona muito bem. Essa técnica é a mesma que os navegadores usam para transmitir campos de dados para um servidor no URL de uma página Web e é diferenciada pelos caracteres 'E comercial' (&) que são usados para separar cada parte dos dados.

```
Date: 10/04/2008
Body: "I'm really looking..."
Image:
```

```
"date=10/04/2008&body=I'm really looking forward... &image="
```

Uma parte individual dos dados consiste em um par de nome/valor.

Cada parte dos dados é separada das outras por &.

Este formato de dados requer que cada parte dos dados tenha seu nome e valor separados por um sinal de igual (=) e, então, cada par de nome/valor seja separado dos outros dados por um E comercial (&). O formato é chamado de URL codificado e tem seu próprio tipo de dados que é definido como o tipo de dados da solicitação POST Ajax.

```
"aplicativo/x-www-formulario-urlcodificado; charset=UTF-8"
```

Este é o tipo de dados oficial do URL com dados codificados e deve ser especificado como parte da solicitação POST.

Com os dados de entrada do blog formatados no formato URL codificado e o tipo de dados da solicitação POST, estamos prontos para reunir o código da solicitação e enviar os dados para o servidor para que eles possam ser gravados no arquivo blog.xml.

Exercício

Reúna as seguintes partes dos dados no formato com URL codificado, adequado para uma solicitação POST.

```
releaseDate: 01/13/1989

title: Gleaming the Cube        director: Graeme Clifford
```

Solução do Exercício

Reúna as seguintes partes dos dados no formato com URL codificado, adequado para uma solicitação POST.

releaseDate: 01/13/1989

title: Gleaming the Cube **director: Graeme Clifford**

"title=Gleaming the Cube&releaseDate=01/13/1989&director=Graeme Clifford"

Não existem Perguntas Idiotas

P: Se o script de acréscimo do blog YouCube não requer nenhum dado do servidor na solicitação Ajax, por que se importar em lidar com a solicitação?

R: A razão é que saber que uma solicitação completou ainda é muito importante. Portanto, mesmo que não precisemos que o servidor retorne nenhum dado em resposta à solicitação, ainda precisamos muito saber se e quando a solicitação terminou com sucesso. É isso que permite ao script saber quando exibir o alerta que confirma o novo acréscimo da entrada do blog.

P: Uma solicitação GET também poderia ser usada no script de acréscimo do blog?

R: Sim, tecnicamente poderia. Ainda é possível enviar dados para o servidor em uma solicitação GET, mas você tem que especificá-los diretamente no URL da solicitação. Realmente não é um problema – o problema é que o GET não é para ser usado em situações onde o estado do servidor está mudando. E neste caso, o estado do servidor está mudando definitivamente devido a gravar uma nova entrada do blog no arquivo blog.xml. Portanto, uma solicitação POST será a abordagem certa se não houver nenhuma outra razão, pois ela indica claramente a intenção da comunicação com o servidor.

P: Como leva tempo para o servidor processar a solicitação Ajax e gravar a entrada do blog, haverá algum problema se o botão Add for clicado novamente antes da solicitação terminar?

R: Sim, será um problema. Cada clique do botão Add cancela a solicitação Ajax atual e envia uma nova. Embora possa ser o objetivo de alguém ao clicá-lo duas vezes, a interface do usuário seria muito mais clara se a opção para clicar o botão simplesmente fosse removida enquanto a solicitação está sendo processada. Portanto, o código para adicionar uma nova entrada do blog deve desativar o botão Add enquanto a solicitação Ajax está ocorrendo e, então, ativá-lo de novo assim que a solicitação tiver terminado. Pequenos toques como este na interface do usuário de um aplicativo JavaScript pode contribuir muito ao torná-lo mais claro e fácil de usar, resultando em usuários mais contentes.

P: O que acontece com os espaços nos dados do blog que são formatados em uma string com URL codificado? Algumas vezes, parece ser um problema com os URLs.

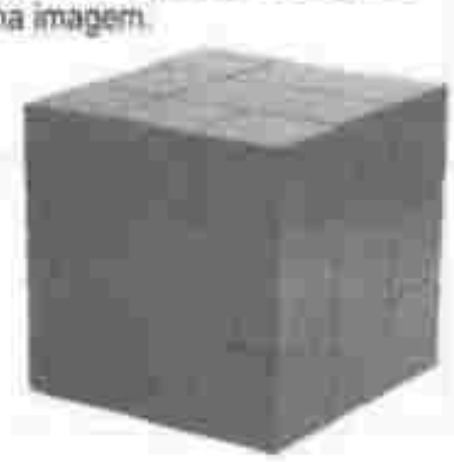
R: Os espaços não apresentam um problema neste caso porque o Ajax lida automaticamente com o processamento dos dados e assegura que cheguem ao servidor em um formato adequado.

P: Como a imagem é opcional no blog, ela sempre tem que ser transmitida para o servidor ao adicionar uma nova entrada do blog?

R: Não, não tem que ser. Mas lembre-se que não há nada de errado em enviar uma parte vazia dos dados onde não há nenhum valor após o sinal de igual na string com URL codificado, como:

"date=...&body=...&image="

Neste exemplo, o campo de dados da imagem ainda é enviado para o servidor, mesmo que de fato não contenha nenhum dado. É onde o script PHP no servidor brilha, pois ele é esperto o bastante para saber que o campo da imagem está vazio e, portanto, a nova entrada do blog não tem uma imagem.



Aponte seu Lápis

Escreva as linhas de código que faltam para terminar as funções `addBlogEntry( )` e `handleRequest( )` no script de acréscimo do blog YouCube.

```
function addBlogEntry( ) {
  // Desative o botão Add e defina o status para ocupado

  ..............................................................

  ..............................................................

  // Envie os novos dados de entrada do blog como uma solicitação Ajax
  ajaxReq.send("POST", "addblogentry.php", handleRequest,
    "application/x-www-form-urlencoded; charset=UTF-8",
                                                          );
    ..............................................................

    ..............................................................

    ..............................................................
}

function handleRequest( ) {
  if (ajaxReq.getReadyState( ) == 4 && ajaxReq.getStatus( ) == 200) {
    // Ative o botão Add e limpe o status

    ..............................................................

    ..............................................................

    // Confirme o acréscimo da entrada do blog
    alert("The new blog entry was successfully added.");
  }
}
```

Aponte seu Lápis
Solução

Escreva as linhas de código que faltam para terminar as funções `addBlogEntry( )` e `handleRequest( )` no script de acréscimo do blog YouCube.

O botão Add é desativado enquanto uma entrada do blog está sendo gravada no servidor.

A área de status da página exibe uma mensagem de ocupado para que o usuário saiba que está aguardando algo.

```
function addBlogEntry( ) {
  // Desative o botão Add e defina o status para ocupado
  document.getElementById("add").disabled = true;
  document.getElementById("status").innerHTML = "Adding...";
  // Envie os novos dados de entrada do blog como uma solicitação Ajax
  ajaxReq.send("POST", "addblogentry.php", handleRequest,
    "application/x-www-form-urlencoded; charset=UTF-8",
    "date=" + document.getElementById("date").value +
    "&body=" + document.getElementById("body").value +
    "&image=" + document.getElementById("image").value );
}
function handleRequest( ) {
  if (ajaxReq.getReadyState( ) == 4 && ajaxReq.getStatus( ) == 200) {
    // Ative o botão Add e limpe o status
    document.getElementById("add").disabled = false;
    document.getElementById("status").innerHTML = "";
    // Confirme o acréscimo da entrada do blog
    alert("The new blog entry was successfully added.");
  }
}
```

Esta é uma solicitação POST.

Um script PHP do servidor é usado para processar a entrada do blog e grava-la no arquivo do blog no servidor.

Reúne os dados POST da solicitação a partir dos campos do formulário da data, corpo e imagem.

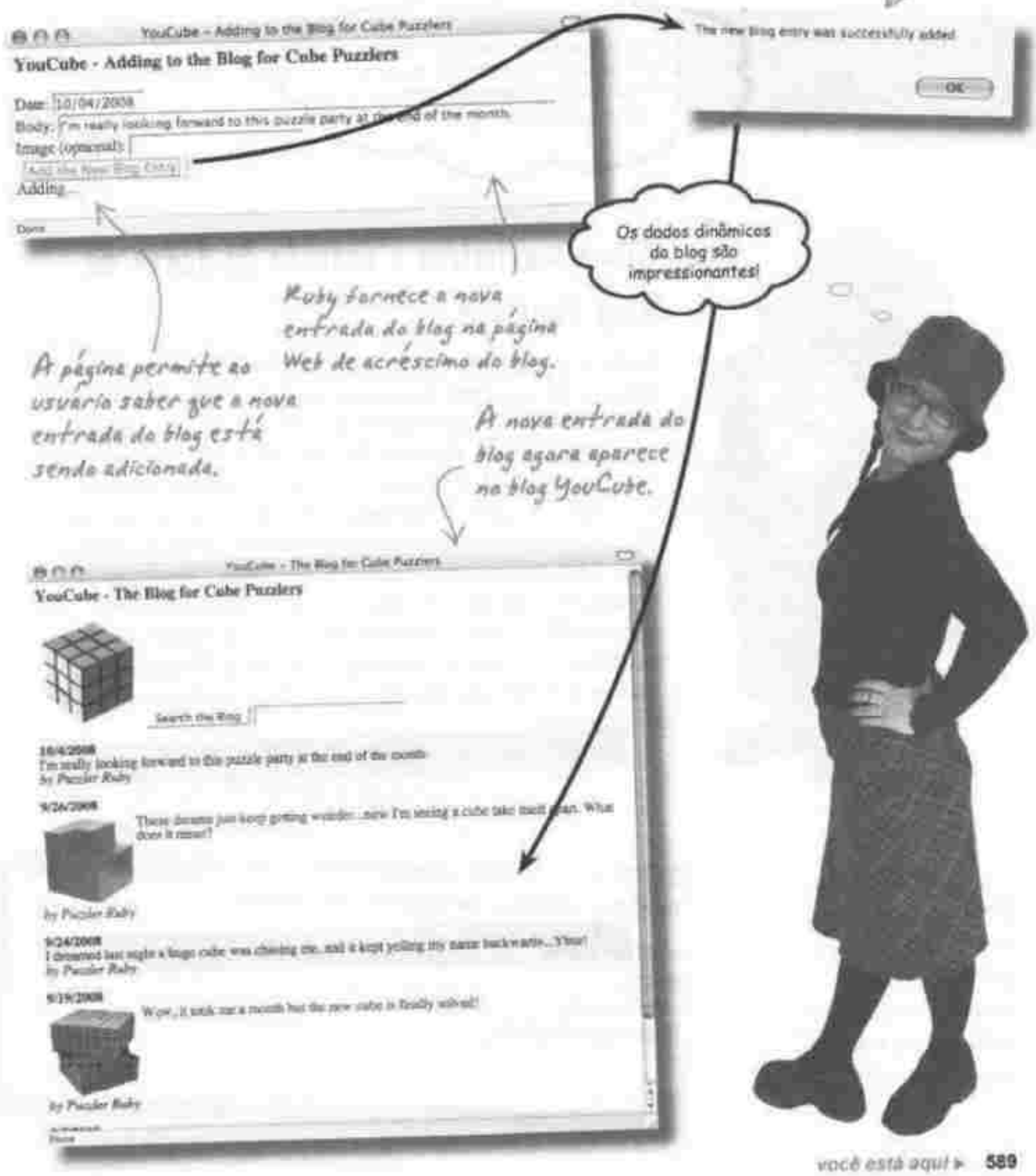
Verifique para assegurar que a solicitação de gravação do blog terminou com sucesso.

Ative o botão Add e limpe a área de status agora que a entrada do blog terminou de gravar.

O blog ficou fácil

Ruby não pode acreditar em quanta diferença faz ser capaz de atualizar seu blog sem ter que abrir um arquivo, editar o código e fazer upload do arquivo para o servidor. Não só seu blog agora está realmente baseado em dados, como também ela está se sentindo muito mais inspirada em escrever alguns dados novos do blog!

A página confirma se a entrada do blog foi adicionada com sucesso.



Como tornar o YouCube mais, ah, útil

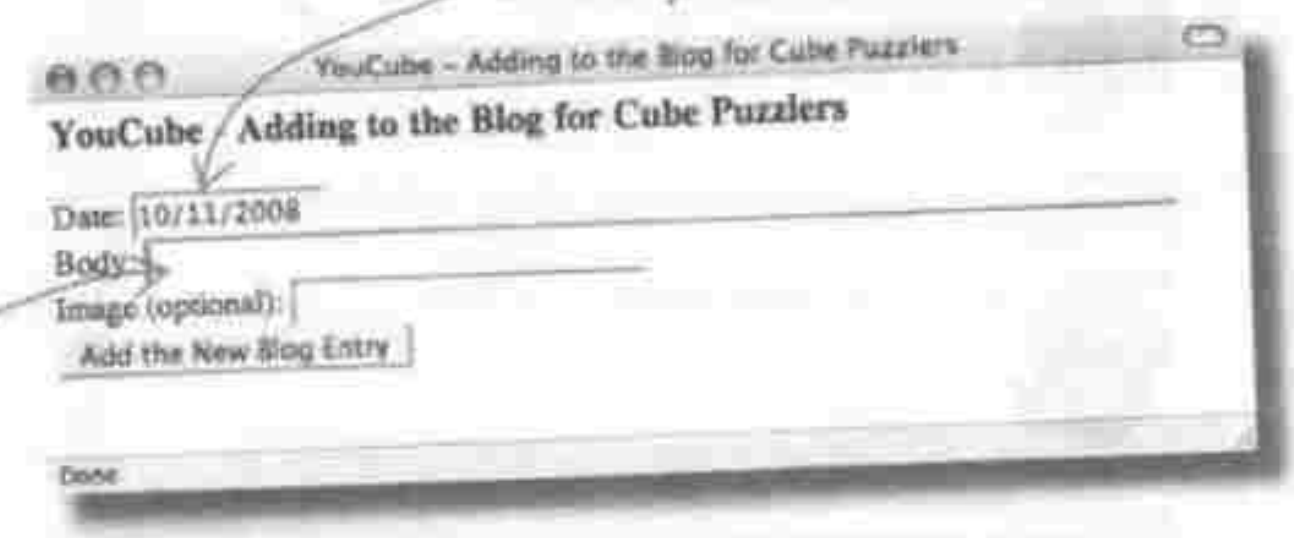
Você não se torna um faixa preta no quebra-cabeça de cubo em terceiro grau sem uma atenção crítica ao detalhe. Portanto, não é muito surpreendente que Ruby queira tornar a página de acréscimo do blog absolutamente perfeita. E ela aprendeu que os aplicativos Ajax são conhecidos por sua atenção ao detalhe em relação à utilização. Portanto, deseja fazer algumas melhorias na utilização de sua nova página para que fique no mesmo nível das páginas Web modernas.

Desejo maximizar minha eficiência da entrada do blog para que eu possa enviar mais rapidamente e espero que seja com mais freqüência.

Como maximizar a entrada de dados do blog YouCube

Como a grande das entradas do blog é criada no presente, Ruby acha que economizará uma preciosa digitação se o campo da data do formulário do blog for preenchido automaticamente com a data de hoje. E como ela usará a data atual para a maioria das entradas do blog, gostaria de colocar o foco de entrada no campo de corpo do formulário. Assim, ela pode começar a digitar em uma entrada do blog no momento em que a página abrir. Certamente, nenhuma dessas alterações é absolutamente crítica para o trabalho do blog e elas não estão diretamente relacionadas ao Ajax, mas melhoram muito o "clima" da página, que é muito parecido com o espírito do Ajax. Além do mais, ajudará assegurar que Ruby manterá o blog atualizado enviando regularmente.

Preencha automaticamente este campo com a data atual.



Define o foco da entrada aqui para que Ruby possa começar a fornecer o texto do blog imediatamente.

Campos de autopreenchimento para seus usuários

Se você lembra, o formato das datas do blog YouCube é MM/DD/AAAA, que significa que precisamos formatar a data atual no campo do formulário da data com autopreenchimento para o mesmo formato. Portanto, precisamos de algum código para formatar a data atual como MM/DD/AAAA.

Compartilhar o código comum é sempre uma boa idéia para evitar a duplicação.

Certamente, não queremos nenhum código duplicado por aí no YouCube que você tenha que manter em dois lugares diferentes, portanto compartilhar o código de formatação da data entre as duas páginas é uma ótima idéia. E há definitivamente um modo de armazenar o código em um lugar e, então, compartilhá-lo em qualquer página que precise dele.

Poder do cérebro

Como você compartilharia o código `shortFormat( )` em ambas as páginas YouCube?

Tarefa repetitiva? Que tal uma função?

Compartilhar o código JavaScript em diversas páginas Web envolve dividir o código em seu próprio arquivo, ou módulo, e, então, importar esse arquivo para cada uma das páginas. Já vimos isso feito com o objeto `AjaxRequest`, que é armazenado no arquivo ajax.js e, depois, importado com a seguinte linha de código:

A tag <script> familiar é usada para importar o código JavaScript que é armazenado em arquivos externos.

O nome do arquivo JavaScript é definido para o atributo src da tag <script>.

```
<script type="text/javascript" src="ajax.js"> </script>
```

O método `shortFormat( )` do objeto `Date` pode conseguir um objetivo parecido sendo colocado em um arquivo chamado date.js e, então, importado para cada uma das páginas Web do YouCube.

```
Date.prototype.shortFormat = function() {
  return (this.getMonth() + 1) + "/" + this.getDate() + "/" + this.getFullYear();
}
```

date.js

Uma tag `<script>` parecida foi usada para o código Ajax poder então ser utilizado em cada uma das páginas YouCube para importar o código do script armazenado em data.js.

O mesmo código JavaScript é compartilhado em dois lugares graças ao armazenamento do código em um arquivo externo.

```
<script type="text/javascript" src="date.js"> </script>
```

Todo o conteúdo do arquivo date.js é importado com esta única tag <script>.

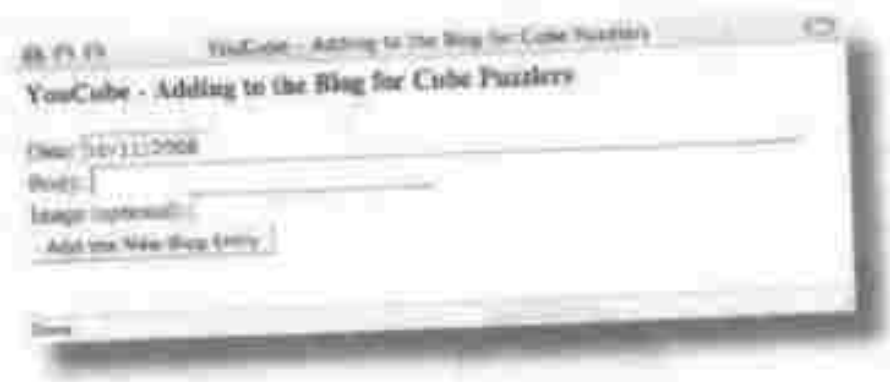
Quase nunca é uma má idéia dividir o código JavaScript reutilizável em seu próprio arquivo que pode ser compartilhado em mais de um lugar.

```
<div id="blog"></div>
```

O método shortFormat() formata as entradas do blog na página YouCube principal...

```
<input type="text" id="date" name="date" value="" size="10" />
```

...e também formata a data atual no campo da data da página de acréscimo do blog.



Aponte seu Lápis

Escreva o código para uma função denominada `initForm( )` que é chamada na sub-rotina de evento `onload` do script de acréscimo YouCube. A função deve inicializar o campo da data com a data atual e, então, definir o foco de entrada para o campo do corpo.

.....

.....

.....

.....

Aponte seu Lápis
Solução

Escreva o código para uma função denominada `initForm( )` que é chamada na sub-rotina de evento `onload` do script de acréscimo YouCube. A função deve inicializar o campo da data com a data atual e, então, definir o foco de entrada para o campo do corpo.

O campo da data é definido para a data atual.

```
function initForm( ) {

document.getElementById("date").value = (new Date( )).shortFormat( );

document.getElementById("body").focus( );

}
```

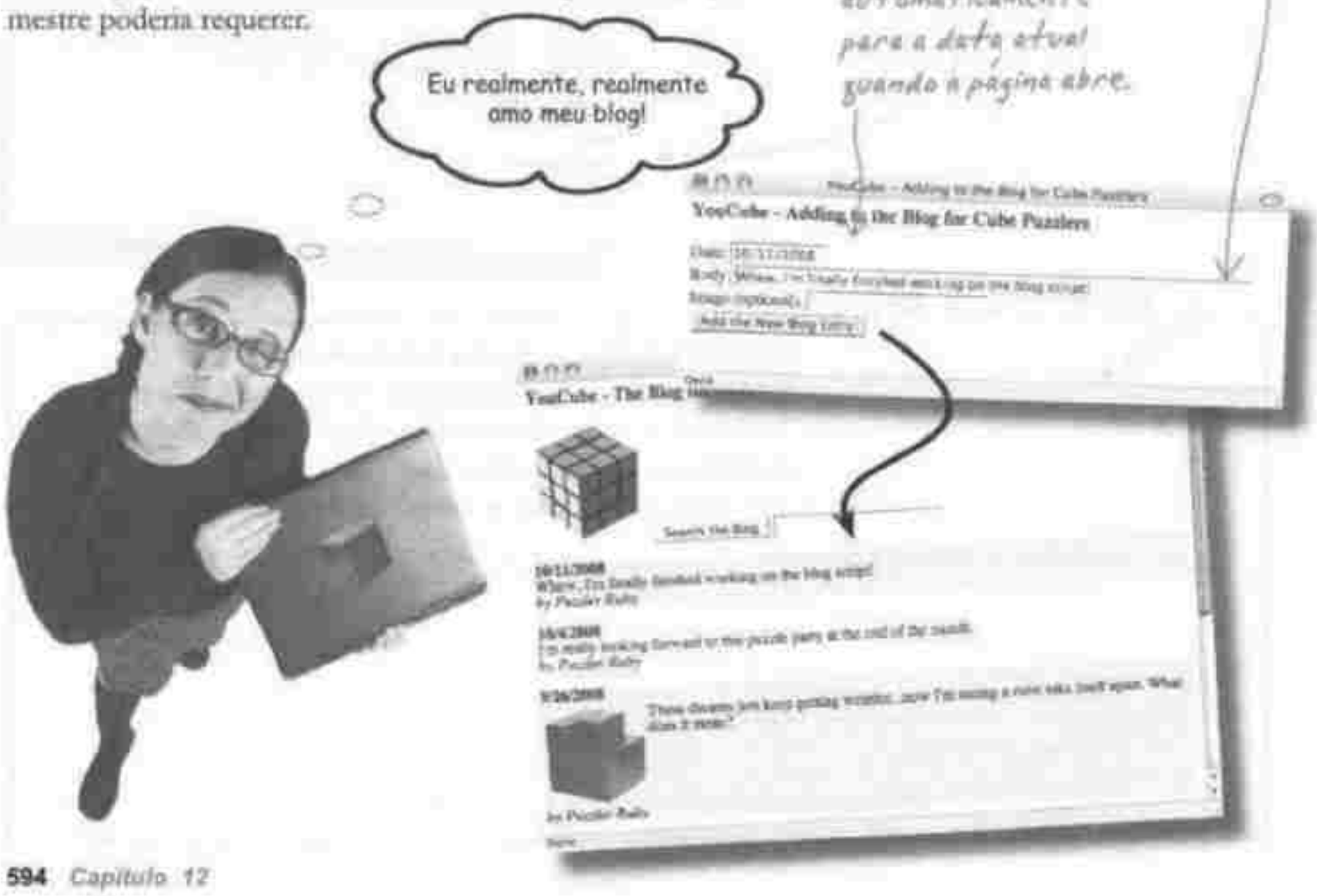
Define o foco de entrada para o campo do corpo.

A produtividade do blog aumenta

Ruby finalmente conseguiu uma satisfação completa com seu blog YouCube. Seu blog é baseado em dados e amistoso graças ao Ajax e a um comprometimento implacável com o detalhe que somente uma solucionadora de quebra-cabeças mestre poderia requerer.

O foco de entrada é definido para o corpo quando a página é carregada pela primeira vez.

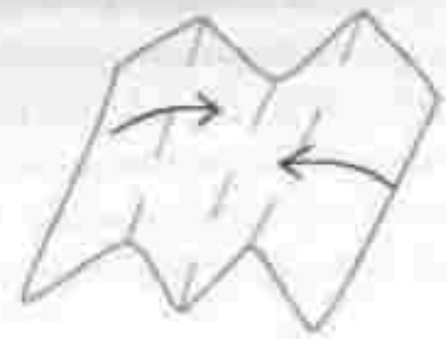
A data é definida automaticamente para a data atual quando a página abre.

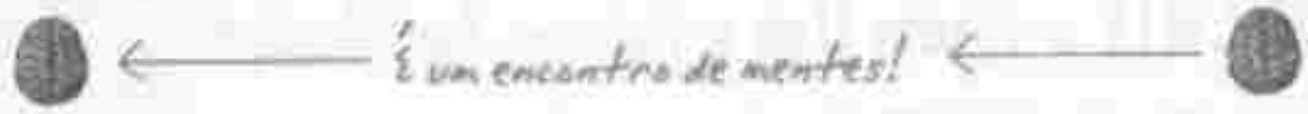


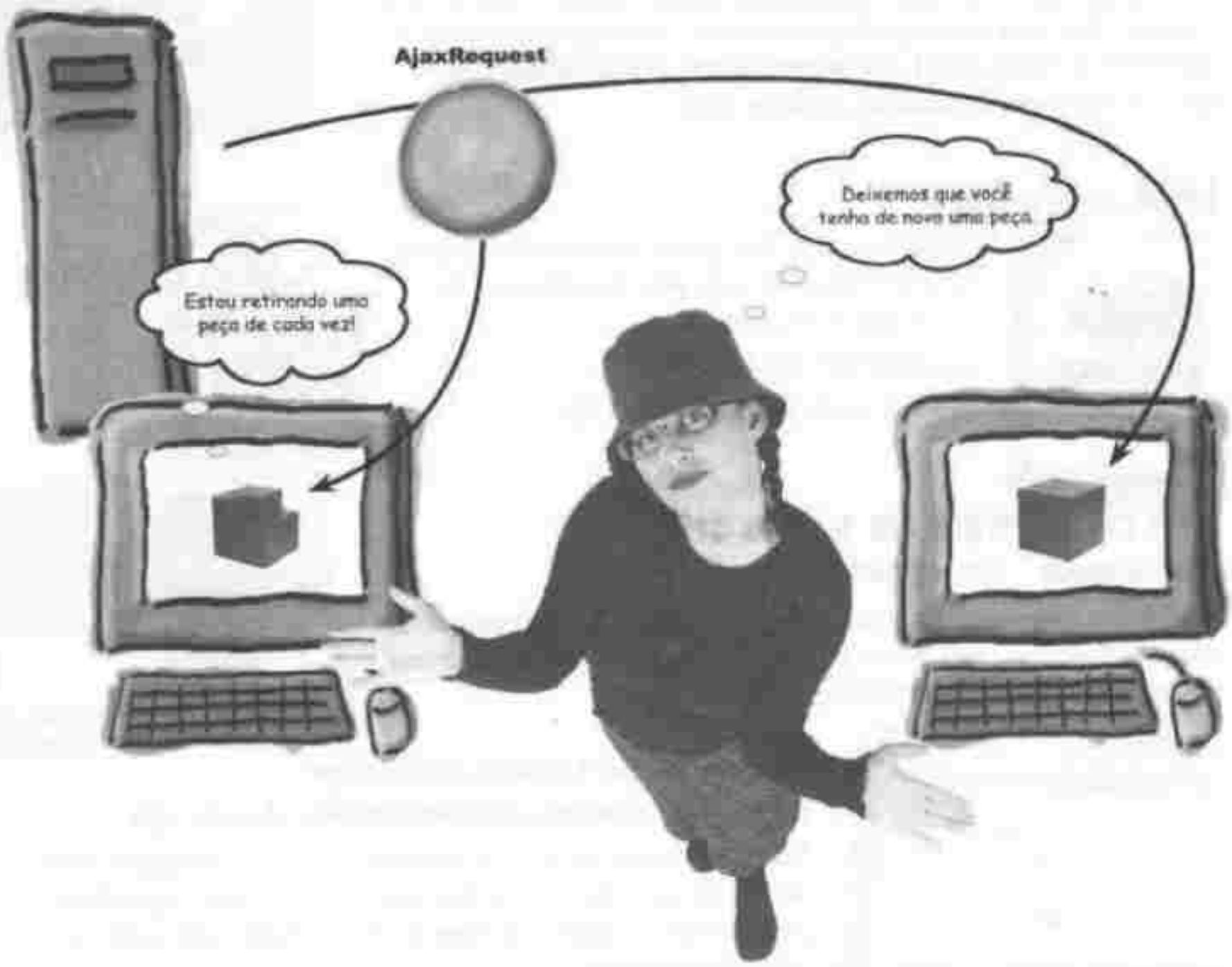
DOBRA DA PÁGINA

Dobre a página na vertical para alinhar os dois cérebros e resolver a charada.

O que o Ajax deu a Ruby?







O Ajax deu tantas coisas a Ruby que é difícil dizer. Primeiro, há dados dinâmicos, que tornaram o blog do cubo muito mais fácil. Agora, Ruby pode conversar sobre seus quebra-cabeças de cubo com facilidade.

Para onde ir agora?

Bem, você terminou o Use a Cabeça JavaScript e está pronto para continuar sua jornada para criar experiências do usuário interativas com o JavaScript e mais... porém, para onde deve ir em seguida? Eis algumas coisas nas quais pensamos que você pode estar interessado quando der seus próximos passos ao construir e ao criar aplicativos para a fantástica World Wide Web.

EXPERIMENTE o fórum Use a Cabeça JavaScript

Irritado com as expressões? Desarmado com os operadores? Ou você está apenas curioso para compartilhar sua última criação JavaScript com a comunidade Use a Cabeça? Faça uma visita ao fórum Use a Cabeça JavaScript no Head First Labs (***http://www.headfirstlabs.com***) e junte-se a uma das discussões... ou inicie uma!

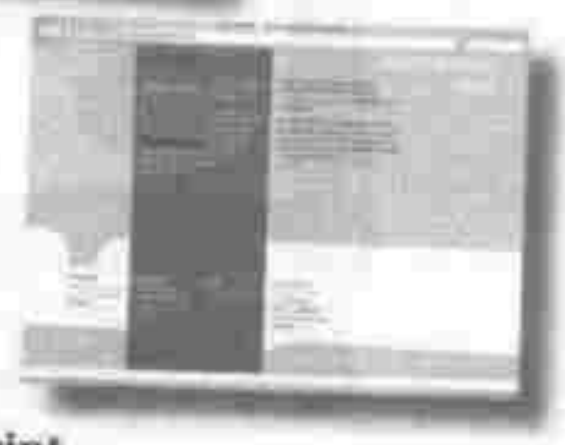
LEIA outro livro

Você viu o essencial, portanto esteja pronto para se aprofundar mais nos prós e nos contras do JavaScript mais avançado.

JavaScript: O Guia Definitivo

Livro de receitas JavaScript e DHTML

APRENDA mais em outros sites



Quirksmode ***www.quirksmode.org***

Infelizmente, algumas vezes navegadores diferentes têm seu próprio modo de fazer as coisas com o JavaScript. Descubra as inconsistências no navegador JavaScript no Quirksmode.

Referência do Mozilla JavaScript

http://developer.mozilla.org/en/docs/JavaScript

Não levará muito tempo até que você esteja se aventurando a sair do usual e precisando descobrir mais sobre os objetos JavaScript predefinidos. Explore cada recanto do JavaScript com a referência on-line do Mozilla.

Estrutura do protótipo JavaScript

http://www.prototypejs.org

Tentado em experimentar uma biblioteca de código reutilizável de terceiros para levar o JavaScript a um novo nível? A Prototype é uma das melhores e complemente gratuita!

Índice Remissivo

Símbolos

-- (operador de diminuição) 202
! (operador de negação) 164, 165, 218
!= (operador de desigualdade) 163
&& (operador AND) 217
|| (operador OR) 218
++ (operador de aumento) 202
< (operador menor que) 164
<= (operador menor que ou igual a) 164
= versus == 510, 532
== (operador de igualdade) 163
    versus = 164
> (operador maior que) 164
>= (operador maior que ou igual a) 164
.xml, extensão de arquivo 544
, tag 346, 362
<img>, tag 204
<input>, atributo desativado da tag 574
<script>, tag 13, 119, 592
, tag 302, 362
2D, Um foco no Mandango 236–237

A

ação 6–7
adição numérica 64, 66, 76
Ajax 541
    assíncrona, solicitação Ajax 560
    PHP 579–584
    solicitação/resposta 561
    YouCube, projeto550–551
AjaxRequest, objeto 556–561
    getResponseText( ), método 565
    postDataType 559
    url 559
ajuda, elemento do texto 309
ajuda, limpar mensagem 309
aleatórios, números 434–437
alert( ), função 19, 297
alerta, caixas 297, 314
    depurar 507–509
    vazias 162
aninhadas, instruções if/else 175
aninhados, loops 227
anônimas, funções 279
apóstrofos 21
    depurar 502, 532
appendChild( ), método 361
apresentação, separar funcionalidade, conteúdo e apresentação 270–271
argumentos 252
    alterar 260
armazenar dados
    constantes (veja constantes)
Array.sort( ), método 423
arrays 198
Arrays Expostos 200
aspas 21
    caracteres de escape 503, 511
assíncrona, solicitação Ajax 560
atributos em formulários 293
Aventura do Bonequinho, projeto 144–171
    árvore de decisão 152–153, 382–383, 387
    changeScene( ), função 157, 162, 367
    criar código HTML 384
    curScene, variável 146
    decisões em camadas 154
    id, atributo da tag 346
    if/else, instruções 154
    manipular estilos individuais 379
    substituir texto do nó com função 368–369
Aventura do Bonequinho, Um foco no código 351

B

Bannerocity, projeto 290–292
    dados, validação 302
    placeOrder( ), função 313–314
bidimensionais, arrays 231–234
    acessar dados data 233
Blog, objeto 404, 406, 408, 412–413
    ações 441
    classificar 472–473

signature, propriedade 460–465
blogSorter( ), classe 473–474
booleano, operador lógico 218
booleano, tipo de dados 35
booleanos, operadores
diferença entre operadores normal e lógico 219
botões, desativar 574–576
break, instruções 178, 179, 212

C

cabeça da página, acessar código a partir de 522–523
callback, funções 277, 281, 282
onload, sub-rotina do evento 283
callNum, variável
aparecer como NaN 526–527
camadas, decisões em 154
CamelCase 52, 62
caractere, classes 336
changeScene( ), função 157, 162, 367
chaves
ausentes 493–494
switch/case, instrução 178
checkWinner( ), função 504–505
classe, métodos 471
chamar 474
classe, métodos de instância possuídos por 471
classes
armazenar métodos 455
versus instâncias 452–453, 459
classificar 418–423
compare( ), função 421
className, propriedade do nó 379
clearInterval( ), função 97
cliente, janela 99–101
porcentagem do tamanho 103
width e height 104
clientes 86, 89
controlar 89
Navegador Exposto 98
cliente-servidor, interações 86
código postal, validação 310–311
expressões constantes 319
código, separação 270–271, 274, 283
Código, um foco em (projeto Mandango) 215
códigos postais 312
comentários 166, 167
código, dentro de 173
ponto-e-vírgula 173
comparação, operadores 164
compare( ), função 421
compostas, instruções 148, 149, 150
dentro de loops for 201
concatenar strings 64, 69, 76
const, palavra-chave 46
constantes 40, 46–47
definir explicitamente 49
quando usar 49
constantes, expressões 318–337
$ 320
? 322
\s 320
{min,máx} 329
{n} 322
barras (/ /) 320
e-mail, validação do endereço 334–337
quantificadores 322–323
construtores 402–403, 405, 459
conteúdo, separar funcionalidade, conteúdo e apresentação 270–271
contexto 169
continue, instruções 212
Conversas em volta da lareira
dados do formulário ruins 298–299
persistente, armazenamento de dados 114
XHTML 546
converter texto em número 65, 69
cookieEnabled, propriedade 126
cookies 88, 112–115
compartilhados nos navegadores 125
nomes 125
touchRock( ), função 122–123
createElement( ), método 383, 385
createTextNode( ), método 361, 371
CSS 6–7
interatividade 8
usar DOM para ajustar estilos CSS 372–373
CSS, classes de estilo 375
manipular estilos individuais 379

D

dados
  acessar dados do formulário 293
  não vazios, validação 300–301
  validação 294–296, 302
dados, tipos 35, 45
  definir explicitamente 49
dados, visibilidade 169
datas, validar 315–316, 329
Date, objeto 97, 410–411, 414
  acessar partes de uma data 417
  converter objetos em texto 415–417
  getFullYear( ), método 417
  shortFormat( ), método 592
decisão, árvore 152–153
  caminho 161
decisões 146
  complexas 218
depuradores 511
depurar 486–536
  = versus == 510, 532
  caracteres de escape 503, 532
  cola 505
  erro, console no navegador 491
  navegadores 492
  Safari 488
Depurar com Alerta 509
desativar código 527–529, 531
dinamicamente, redimensionar imagens 108–109
dinâmicas, decisões de texto 370
dinâmicos, dados 538–598
  Ajax (veja Ajax)
  XHTML 545
dispositivos móveis, gráficos com tamanho errado 99–101
dispositivos, texto e gráfico com tamanho errado 99–101
div versus span, elementos 371
Document Object Model (DOM) 352–392
  appendChild( ), método 361
  DOM, árvore 357
  hierárquica, árvore de nós 353
  nó superior 354
  propriedades 359
  usar DOM para ajustar estilos CSS 372–373
document.body.clientHeight 100
document.body.clientWidth 100
DOM, Blocos de construção 362
DOM
  extrair conteúdo de nós dos dados XML 566
Duncan's Donuts, projeto 55–84
  buscar dados nas páginas Web 71
  pesquisar entrada do usuário 78
duplicado, eliminar código 254–256

E

ELEMENTO, nó 354, 362
Elementos
  IDs 71
e-mail, validação do endereço 334
encapsulamento 472–473
erro, consoles 491
  observar variáveis 516
erros (veja depuração)
erros
  execução 524–525
escape, caracteres 503, 511, 532
  limite 511
escopo 169, 170
  global 171
estilo 6–7
estilo, classe 302
estilo, objeto 102–104
estrutura 6–7
evento, sub-rotinas 278
eventos 18
  de onde vêm 21
  ligar através de literais da função 279–280
exclusivos, identificadores 62
execução, erros 524–525

F

filhos, nós 353, 363
findSear( ), função 206, 211, 258
Firefox, depurar 489–491
floor( ), método 436, 437
for, loops 192–193, 226–228
  código não sendo chamado 227
formulários 13, 292
  acessar dados do formulário 293

atributos 293
campos 294
gerar eventos 296
onchange, evento 295
validar comprimento dos dados 305–306
função, literais 279, 281, 282
evento, ligar através de 279–280
função, referências 278, 282
funcionalidade, separar funcionalidade, conteúdo e apresentação 270–271
funções 19, 243–288, 398
alert( ) 19, 297
callback (veja callback, funções)
como soluções de problemas 246
finalidade 249, 260
referir 273–275
validateRegEx( ) 326–327

G

GET, solicitação 554–555, 561
getElementById( ), método 71, 76, 101, 102, 347, 352, 358, 375
getElementsByTagName( ), método 358
getSeatStatus( ), função 267–268
getTemp( ), função 262
getText( ), Um foco na função 567
globais, variáveis 169–173
nível do script 173
global, escopo 171
greetUser( ), função 121

H

heat( ), função 251, 263
hierárquica, árvore de nós 353
HTML 6–7
acessar elementos 347–348, 522
size, atributo 309
HTTP, solicitação/resposta 561

I

id, atributo da tag 346
identificadores 50
exclusivos 62
letra maiúscula 52
IDs dos elementos 71
if, instruções 138, 150
aninhadas 155, 158
if/else, instruções 150
aninhadas 175
escolher entre duas coisas 141
importar scripts externos 119
indefinida, erro da variável 495–496
indefinido
versus não definido 497
indexados, arrays 199, 201, 202
infinitos, loops 201, 223
Mandango, projeto 210
inicializar dados 69
constantes 60–61
inicializar loops 191
initSeats( ), função 205, 248
innerHTML, propriedade 348, 350, 352, 364, 408
definir conteúdo 349
instâncias
propriedades da classe versus propriedades da instância 464, 466
interatividade
HTML e CSS 8
Internet Explorer, depurar 488, 491
interpretador 86, 89, 493–494
intervalo, temporizadores 93, 95, 97
parar 97
iRock, projeto
100 no cálculo do tamanho da imagem iRock 105
cookie, escrita 122–123
eventos 18
redimensionar janelas do navegador 106–107

J

JavaScript, código
chaves ausentes 493–494
eventos 18
tipos de dados 35
JavaScript, Saindo do forno
parseDonuts( ), função 79
JavaScript
interpretador (veja interpretador)

L

length, propriedade de uma string 306–307
letra maiúscula 52, 62, 69
  objetos 459
Lidar com respostas Ajax 568
locais, variáveis 169–173
lógicos, erros 510–511, 525
  script ruim 512–513
loops 189–242
  ação 191
  break, instruções 212, 213, 224
  contador de loops 213
  inicializar 191
  loops infinitos 201, 223
  while (veja while, loop)
lowerCamelCase 52, 62, 249

M

Mandango, projeto 194–220
  2D, Um foco no Mandango 236–237
  assento, inicialização 205
  Código, Um foco em 215
  findSeats( ), função 258
  mapear array para imagens de assento 204
  setSeat( ), função 257–258
marcação, linguagem 542
Math, objeto 434, 437
Math, Objeto Exposto 435
Mestre dos padrões 325
metacaracteres 320–321, 324
métodos
  armazenar em classes 455
  transformar funções em 441–442
milissegundos 95, 104

N

NaN (Not a Number) 62, 69
Não existem perguntas idiotas
  ! (negação, operador) 165
  Ajax 580
  ajuda, limpar mensagem 309
  arrays 234
  aspas e apóstrofos 21
  Blog, objeto 441
  campo do formulário, atributo size 309
  clearInterval( ), função 97
  conexão entre XML e Ajax 548
  cookies permanentes 125
  Date, objeto 414
  depuradores 511
  diferença entre operador booleano normal e operador lógico booleano 219
  erros tipográficos, depurar 497
  floor( ) versus round( ), métodos 437
  função, literais 281
  GET e POST, solicitações 561
  HTML, atributo size 309
  indefinido versus não definido 497
  interpretador 89
  letras maiúsculas e minúsculas 62
  Math, objeto 437
  NOT, operador 219
  parseInt( ), função e números decimais 76
  ponto-e-vírgula em comentários 173
  prototype, objetos 459
  segurança 13
  temporizadores 97
  tipos de dados 49
  variáveis shadow 531
  YouCube, script de acréscimo do blog 586
não vazio, validação 300–303
Navegador Exposto 98
navegador, objeto 126
navegadores 89
  cliente-servidor, interações 86
  cookies e segurança do navegador 124
  fechar navegador antes do temporizador expirar 97
  métrica 88
  sintaxe, erros 500
new, operador 405, 413
nós 353
  diversos filhos 360
  substituir texto do nó com função 368–369
NOT, operador 219
null 165
  argumentos da função ausentes 477
number, tipo de dados 35
número do telefone, validação 333

O

objeto, literais 413
objetos 395
    arrays como 420
    converter em texto 415–417
    dentro de objetos 413
    letra maiúscula 459
    new, operador 405, 413
    personalizados 400–401
    ponto, operador 396–397
    protótipo 456–457
    strings como 432
    toString( ), método 416
observação, variáveis 508, 515
    consoles de erro 516
onblur, evento 295–297
    dados do formulário ruins 298–299
onchange, evento 295
    dados do formulário ruins 298–299
onfocus, evento 295
onload, evento 20, 105, 278
onload, sub-rotina do evento 280, 283
onmouseout, evento 375
onmouseover, evento 375
onresize, evento 107–109
Opera, depurar 488
organizacional, objeto 434
orientada a objetos, construção (OOP) 460, 461, 465, 467, 470–475
orientado a objetos 461
orientado a objetos, script 399

P

página, problemas no carregamento 522–523
palavras-chaves 44
    letra maiúscula 46
parêntesis
    depurar 532
parseDonuts( ), função 79
parseFloat( ), função 65
parseInt( ), função 65, 77
    números decimais 76
PD[illegible] gráficos com tamanho errado 99–101
persistência 112–115
    clientes versus servidores 116
personalizados, objetos 450–484
pesquisáveis, objetos 424–432
PHP 579–584
    executar scripts 581–582
placeOrder( ), função 313–314
placeOrder( ), Um foco na função 314
ponto, operador 396–397
ponto-e-vírgula em comentários 173
    ponto-e-vírgula na instrução if 143
ponto-e-vírgula
    comentários 173
Pontos importantes
    AjaxRequest, objeto 560
    arrays bidimensionais 235
    break, instruções 224
    classificar 432
    cliente, janela 104
    createElement( ), método 385
    depurar no console de erro do navegador 498
    erros de sintaxe 511
    expressões constantes 330
    função, literais 282
    funções 21, 259
    globais versus locais, variáveis 173
    inicializar dados 69
    instruções compostas 150
    loops 202, 224
    objetos 399
    return, instrução 269
    this, palavra-chave 459
POST, solicitação 554–555, 561
predefinidas, funções 21
Preste atenção!
    = versus == 164
    códigos postais 312
    const, palavra-chave 46
    CSS, classes de estilo versus classes JavaScript 372
    expressões constantes, aplicar escape em caracteres especiais 336
    redimensionar imagens 109
    validar datas 329
problemas, dividir 244–245
prototype, objetos 456–457, 459, 468–469
pseudocódigo 158

Q

QI, script da calculadora 487–497
chaves ausentes 493–494
erros tipográficos, depurar 496
quantificadores 322–323

R

rádio, script de chamadas 498–499
caixas de alerta 507–508
checkWinner( ), função 504–505
depurar aspas 501–502
random( ), método 436
recarregar páginas, limpar variáveis 111
redimensionar imagens 99–109
dinamicamente 108–109
reload( ), método 97
repetição (veja loops)
replaceNodeText( ), função 369
Retorno Exposto 264
return, instrução 262–264, 269
reutilizáveis, tarefas 254–256
round( ), método 437

S

Safari, depurar 488
script 244
script, variáveis globais no nível 173
scripts, importar externos 119
segurança 13
sem continuação, temporizadores 93
servidores
armazenar dados em versus cookies 125
persistência 116
setInterval( ), função 95, 97, 104
setSeat( ), função 257–258
setTimeout( ), função 94, 104
shadow, variáveis 530–532
showSeatStatus( ), função 268–269, 272–273, 279
sintaxe 51
sintaxe, erros 500, 525
script ruim 512–513
solicitação, métodos 554
sort( ), método 420–421
span versus div, elementos 371
string, limite 502
String, objeto 413
charAt( ), método 426
indexOf( ), método 426–427
pesquisar em 427
toUpperCase( ), método 426
strings
caracteres de escape 503
concatenar 64, 69, 76
depurar aspas 501–502
vazias 72
substrings 426
Switch Exposta 180
switch/case, instruções 143, 177–180
break, instruções 178, 179
desvio padrão 178

T

tabulares, dados 233
tarefas reutilizáveis 254–256
temporizadores 88
compreender 92
retardo 94–96
sem continuação 93
text, tipo de dados 35
texto, converter em número 65
TEXTO, nó 354, 362
this, palavra-chave 300, 309, 375, 403, 405, 459
Blog, objeto 451
tipográficos, depurar erros 496, 497, 532
toString( ), método 416, 423
touchRock( ), função 24
cookie, escrita 122–123

U

unário, operador 219
URL, codificação 585
espaços 586
usuário, decisões 146
árvore de decisão 152–153
usuário, entrada 4–5
clara 77

V

validação dos dados (veja dados, validação)
validateDate( ), função 329
validateLength( ), função 306
validateNonEmpty( ), função 300–303, 329
validateRegEx( ), função 326–327
var, palavra-chave 44
variáveis 40
  Aventura do Bonequinho, revisitar 170
  definir explicitamente 49
  erro da variável indefinida 495–496
  globais 169–173
  locais 169–173
  NaN (Not a Number) 62
  quando usar 49
  tentar fazer referências a variável indefinida ou função 511
  variáveis locais e globais com mesmo nome 530–531
  versus funções 274
vazia, caixa de alerta 162
  resolver 163
vazias, strings 72
vírus 124

W

Web, cliente (veja clientes)
Web, padrões 365
Web, páginas
  elemento versus seu valor 76
  obter dados de 71
Web, script 244
while, loops 222–229
  código não sendo chamado 227
width e height, propriedades do estilo CSS 105

X

XHTML 545–546
XML
  Ajax 549
  conexão entre XML e Ajax 548
  DOM 549
  HTML, tag 572
  XHTML 545
XMLHttpRequest, objeto 552–556, 560
  abort( ) 552
  criar com menos problemas 556
  onreadystatechange 552
  readyState 552
  responseXML 552
  status 552

Y

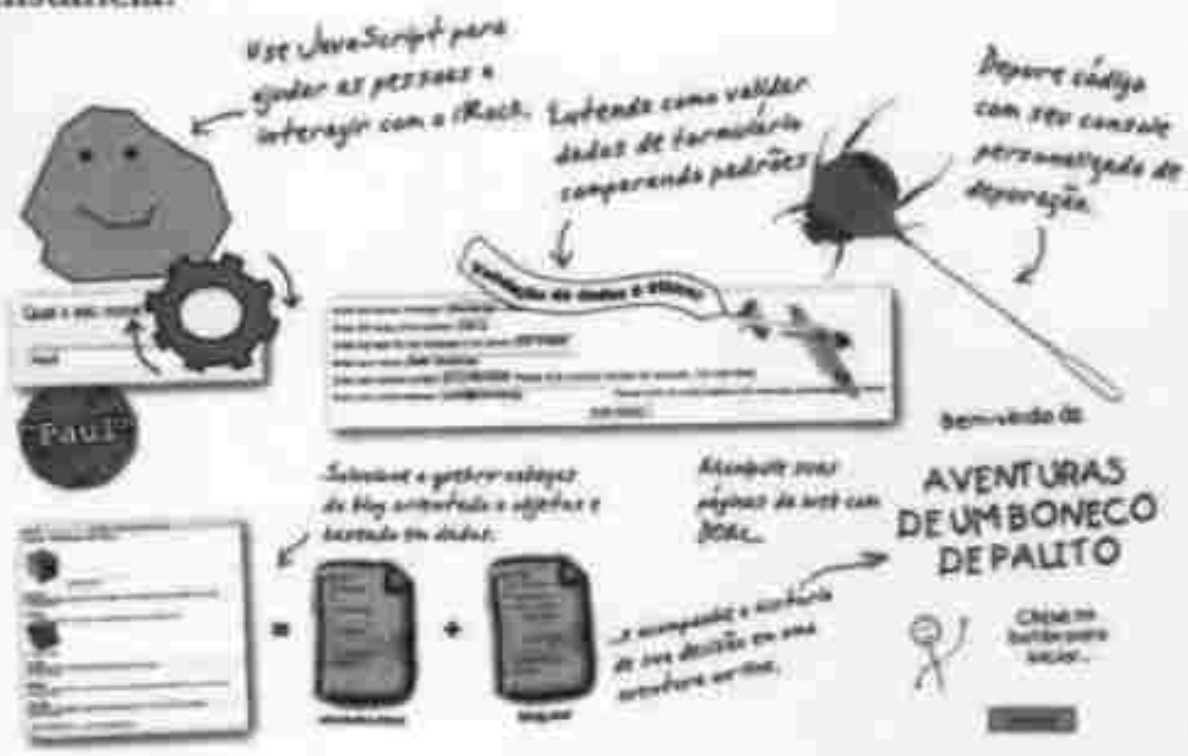
YouCube, projeto 399–448
  Add, botão 586
  adicionar nova entrada do blog 585
  adicionar propriedade signature à classe Blog 460–465
  Ajax, solicitação 563–564
  blog, classificar 418–423
  blog, script para acréscimo 586
  botões 573–576
  chamar métodos da classe 474
  classificar 472–473
  dados dinâmicos 538–539
  Data, método shortFormat( ) do objeto 592
  estender objetos 467–470
  handleRequest( ), função 569–570
  importar código JavaScript de arquivos externos 592–593
  innerHTML 408
  ordem das entradas do blog 408–410
  PHP 579–584
  preencher automaticamente dados do blog 590–591
  transformar funções em métodos 441–442
YouCube, Um foco em 407

Use a Cabeça! JavaScript

Programação Web/JavaScript

O que você aprenderá com este livro?

Você está pronto para dar um salto na arte de escrever páginas da web em HTML e CSS à criação de aplicações dinâmicas para web? Comece por aqui. *Use a Cabeça! JavaScript* é o seu passeio guiado para a emocionante e interativa criação de páginas da web. Criado para o seu cérebro, este livro abrange todos os fundamentos de JavaScript, das técnicas básicas de programação web, incluindo variáveis, funções e execução de loop, aos tópicos mais avançados como validação de formulário, manipulação de DOM, objetos personalizados, depuração – e até Ajax! Então, prepare-se... Os responsivos sites da web estão há algumas páginas de distância.



Por que este livro parece tão diferente?

Nós acreditamos que seu tempo é muito valioso para ser desperdiçado. Tendo como base a última pesquisa em neurobiologia, ciência cognitiva e teoria do aprendizado, *Use a Cabeça ! JavaScript* tem um visual rico, projetado na forma como seu cérebro funciona; não se trata de uma abordagem pesada que faz com que você caia em sono profundo.

"Prático, útil e bem explicado. Este livro realiza um excelente trabalho ao apresentar o JavaScript a um completo novato, e é outra prova do estilo de ensino do Use a Cabeça!. Além de outros livros de JavaScript, Use a Cabeça! JavaScript é excelente para o aprendizado, comparado a outras obras de referência do tamanho de uma lista telefônica."

—Alex Lee, estudante, Universidade de Houston

"Uma excelente opção para o desenvolvedor iniciante de JavaScript."

—Fletcher Moore, Desenvolvedor web & Designer, Instituto de Tecnologia da Geórgia

"Outra grande obra no clássico estilo 'Use a Cabeça!'."

—TW Scannell

ISBN 978-85-7608-213-2

O'REILLY®