CIS 4004: Web Based Information Technology Fall 2012

Introduction To JavaScript – Part 4 – More On Events

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cis4004/fall2012

Department of Electrical Engineering and Computer Science
University of Central Florida

JavaScript – Part 4 – More On Events

- To make your web application respond to user actions on the page, you need to do three things:
  - Decide which events should be monitored (listened for).
  - Set up the event handlers that trigger functions when events occur.
  - Write the functions that provide the appropriate responses to the events.
- As you've seen in the previous JavaScript notes and your latest project, an event is issued as the result of some specific activity – usually user activity, but sometimes browser activity such as a page load – and that you can handle the event with an event handler.

JavaScript – Part 4 – More On Events

- An event handler is always the name of the event preceded by "`on`"; for example, the event `click` is handled by the `onclick` event handler.
- The event handler causes a function to run, and the function provides the response to the event.
- The tables on the next tow pages lists some of the more commonly used event handlers. For a more complete listing see: http://www.w3.org/TR/DOM-Level-3-Events/

JavaScript – Part 4 – More On Events

Event Category	Event Triggered When...	Event Handler
Browser Events	Page completes loading	`onload`
	Page is removed from browser window	`onunload`
	JavaScript throws an error	`onerror`
Mouse Events	User clicks over an element	`onclick`
	User double-clicks over an element	`obdblclick`
	The mouse button is pressed down over an element	`onmousedown`
	The mouse button is released over an element	`onmouseup`
	The mouse pointer moves onto an element	`onmouseover`
	The mouse pointer leaves an element	`onmouseout`

JavaScript – Part 4 – More On Events

Event Category	Event Triggered When…	Event Handler
Keyboard Events	A key is pressed	`onkeydown`
	A key is released	`onkeyup`
	A key is pressed and released	`onkeypress`
Form Events	The element receives focus	`onfocus`
	The element loses focus	`onblur`
	The user selects type in text or text area field	`onselect`
	User submits a form	`onsubmit`
	User resets a form	`onreset`
	Field loses focus and content has changed since receiving focus	`onchange`

Inline Event Handlers

- An event handler can be utilized inline by attaching the event handler directly to an element, such as:

  ```
  <input type=“text” onblur=“doValidate()” />
  ```

  In this case, a form text field has the JavaScript function `doValidate()` associated with its `blur` event – the function will be called when the user moves the cursor out of the field by pressing Tab or clicks elsewhere. The function could then check if the user typed something in the field or not.

- While inline event handlers have been used for a number of years, they are not ideal as they mix the HTML and the JavaScript, and these should be separate. In a modern web application – in the interests of accessibility, maintainability, and reliability – you want to keep JavaScript and CSS out of your HTML markup.

The Handler As An Object Property

- The example below illustrates the way that we've mostly thus far have utilized the event handlers in our JavaScript examples.

```
var clickableImage = document.getElementById("dog_pic");

clickableImage.onclick = showLargeImage;
```

- In this example, first the object representing the HTML element with the `id` = "dog_pic" is assigned to the variable `clickableImage`. The event handler `onclick` is assigned as a property of the object, using a function name as the `onclick` property's value. The function `showLargeImage` will run when the user click on the element with the `id` = "dog_pic".

The Handler As An Object Property

- The technique shown on the previous page has the desirable property of keeping the JavaScript out of the markup since this would appear only in an external JavaScript file and not in the markup.

- However, there are a couple of rather serious drawbacks to this approach.

- First, only one event at a time can be assigned using this technique, because only one value can exist for a property at any given time. You can't assign another event to the `onclick` property without overwriting this one, and for the same reason, another event that was previously assigned is overridden by this one.

The Handler As An Object Property

- Second, when the user click on this element and the function is called, the function has to be hard-coded with the name of the object so that it knows which element to work on.

```
function showLargeImage() {
   thePicture = document.getElementById(“dog_pic”);
   //do something with the picture
}
```

- If you change the object that is the source of the event, you will also need to modify the function.

The Handler As An Object Property

- For the two reasons just explained, the "handler as an object property" technique is suitable for use only when you just want to assign one event to one object, such as running an initial `onload` function once the page is first loaded.
- This technique does not really provide a robust solution for use throughout a RIA (Rich Interface Application, i.e. web pages with user interaction often incorporating AJAX – later this semester), where events commonly get assigned and removed from objects as the application runs.
- In almost every such case, the best way to manage events is to use event listeners.

Event Listeners

- Event listeners were introduced with the DOM model and provide comprehensive event registration.

- An event listener does what its name suggests: After being attached to an object (a node in the DOM), it then listens patiently for its event to occur. When it "hears" its event, it then calls its associated function in the same manner as the " handler as an object property" method but with two important distinctions.

Event Listeners

- First, *an event listener passes an event object containing information about its triggering event to the function it calls*.
- Within the function, you can read this object's properties to determine the target element, the type of event that occurred – such as `click`, `focus`, `mousedown` – and other details about the event.
- This capability can reduce coding considerably, because you can write very flexible functions for key tasks, such as handling clicks, that provide variations in the response depending on the calling object and triggering event. Otherwise, you would have to write a separate, and probably very similar, function for every type of event you need to handle.

Event Listeners

- Second, *you can attach multiple event listeners to a single object.*
- As a result, you don't have to worry when adding one listener that you are overwriting another that was added earlier, as you would when assigning an event as an object property.
- Both W3C-compliant browsers and Microsoft browsers enable event handlers, they differ in how those handlers are attached to element and in the way they provide access to the event object.
- We'll focus on the W3C approach, which will be the de facto standard in the future. I'll show you both techniques as well as a work around that will enable the JavaScript to determine which browser the user is using.

Event Listeners – W3C Technique

- The W3C technique for adding/registering an event handler is the method `addEventListener()` which takes three arguments:
  - The first is the name of the event for which you are registering the handler.
  - The second is the function that will be called to handle the event.
  - The third is either "true" or "false". Typically, this will be false. When true is used this relates to event bubbling (covered later).
- An example is shown on the next page.

Event Listeners – W3C Technique

- Example:

```
emailField=document.getElementById("email");

emailField.addEventListener('focus', doHighlight, false);

email.Field.addEventListener('blur', doValidate, false);
```

Get the object

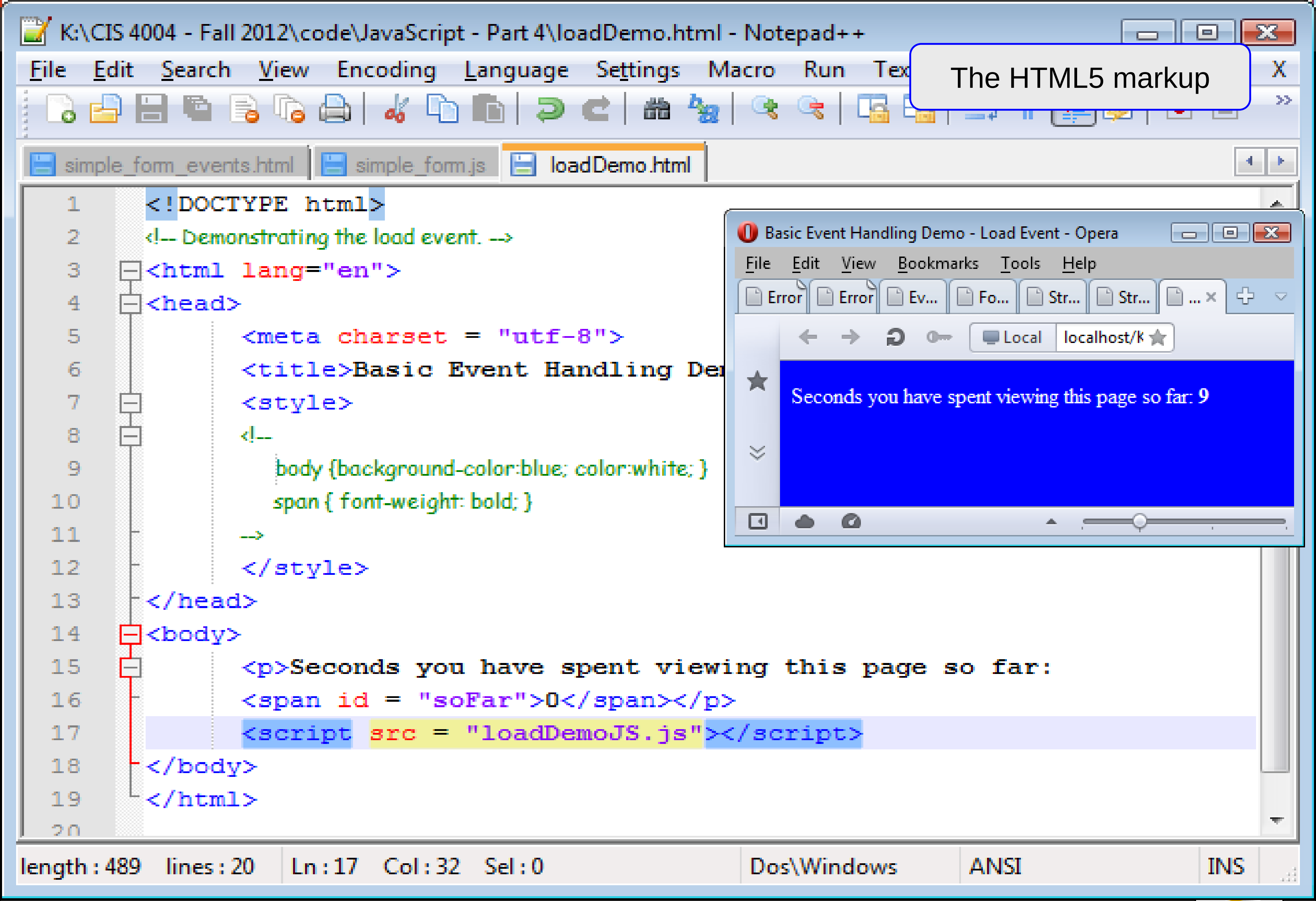
Add a focus listener

Add a blur listener

- The function `doHighlight` would be called when the cursor moves into the field, and the function `doValidate` would be called when the cursor moves out of the field.

- As many event listeners as you would like can be attached to an object in this fashion.

- The next two pages illustrate simple event handler registration.

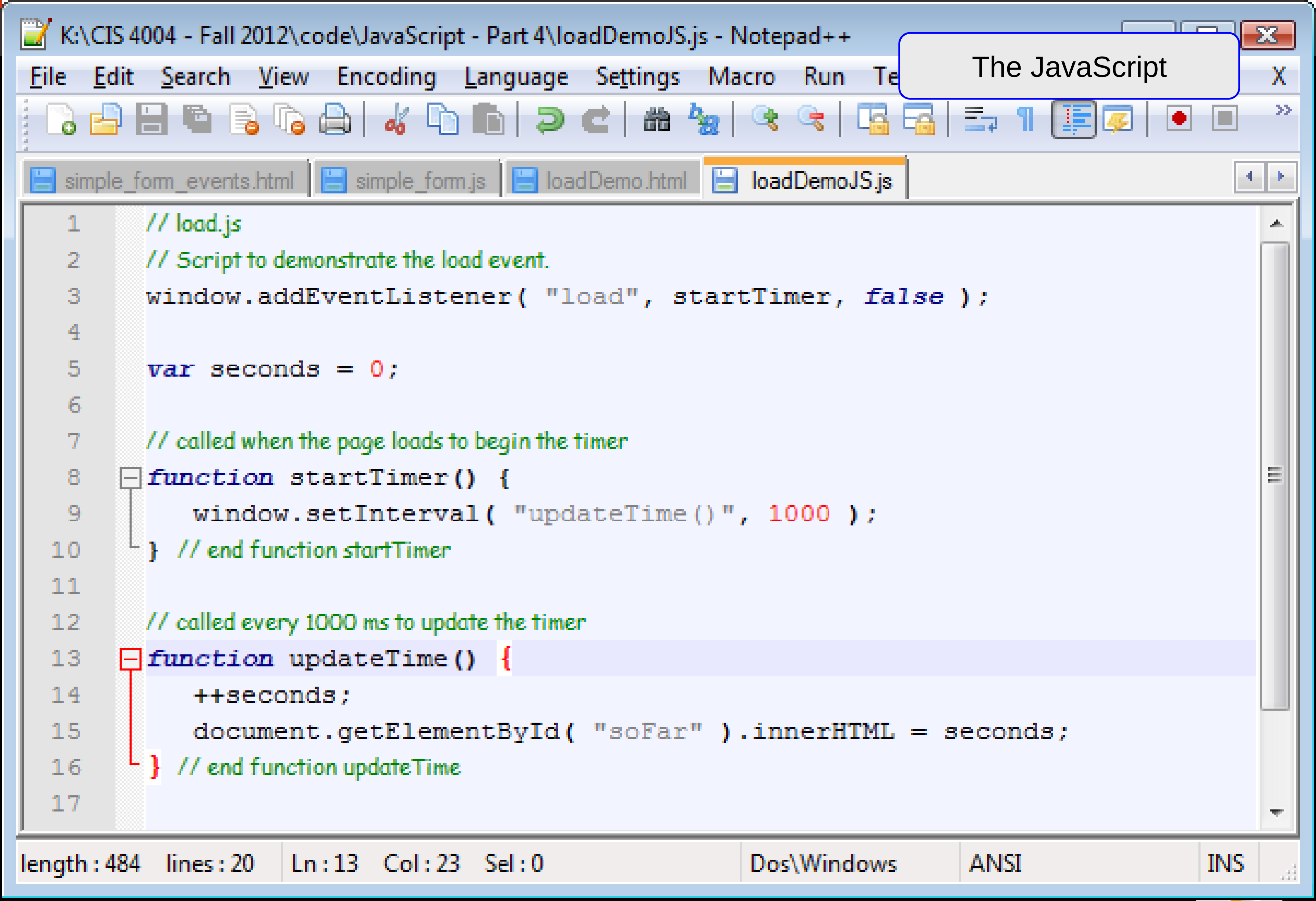

K:\CIS 4004 - Fall 2012\code\JavaScript - Part 4\loadDemo.html - Notepad++
File Edit Search View Encoding Language Settings Macro Run
simple_form_events.html
simple_form.js
loadDemo.html

The HTML5 markup

```
<!DOCTYPE html>
<!-- Demonstrating the load event. -->
<html lang="en">
<head>
        <meta charset = "utf-8">
        <title>Basic Event Handling Dem
        <style>
        <!--
           body {background-color:blue; color:white; }
           span { font-weight: bold; }
        -->
        </style>
</head>
<body>
        <p>Seconds you have spent viewing this page so far:
        <span id = "soFar">0</span></p>
        <script src = "loadDemoJS.js"></script>
</body>
</html>
```

length : 489 lines : 20 Ln : 17 Col : 32 Sel : 0 Dos\Windows ANSI INS
Basic Event Handling Demo - Load Event - Opera
File Edit View Bookmarks Tools Help
Error Error Ev... Fo... Str... Str... ...
Local localhost/k

Seconds you have spent viewing this page so far: 9


K:\CIS 4004 - Fall 2012\code\JavaScript - Part 4\loadDemoJS.js - Notepad++
File Edit Search View Encoding Language Settings Macro Run Te
simple_form_events.html
simple_form.js
loadDemo.html
loadDemoJS.js

The JavaScript

```javascript
// load.js
// Script to demonstrate the load event.
window.addEventListener( "load", startTimer, false );

var seconds = 0;

// called when the page loads to begin the timer
function startTimer() {
   window.setInterval( "updateTime()", 1000 );
} // end function startTimer

// called every 1000 ms to update the timer
function updateTime() {
   ++seconds;
   document.getElementById( "soFar" ).innerHTML = seconds;
} // end function updateTime

```

length : 484 lines : 20 Ln : 13 Col : 23 Sel : 0 Dos\Windows ANSI INS

Event Listeners – W3C Technique

- Event listeners can be removed (unregistered) in a similar manner by using the `removeEventListener` method.

- Example:

```
emailField=document.getElementById(“email”);

emailField.removeEventListener(‘focus’, doHighlight, false);

email.Field.removeEventListener(‘blur’, doValidate, false);
```

Get the object

Remove the focus listener

Remove the blur listener

Event Listeners – Microsoft Technique

- Microsoft's event registration model is slightly different than the W3C technique.

W3C: `emailField.addEventListener('focus', doHighlight, false);`

Microsoft: `emailField.attachEvent('onfocus', doHighlight);`

- Similary, Microsoft's event listener removal is also slightly different than the W3C technique.

W3C: `emailField.removeEventListener('focus', doHighlight, false);`

Microsoft: `emailField.detachEvent('onfocus', doHighlight);`

Adding Event Listeners

- For the time being, at least until IE either disappears or becomes W3C-compliant (not likely!), you will need to write your JavaScript to add event listeners in the correct format for the browser being used by your visitor.
- Fortunately, John Resig (the guy who developed jQuery) has written a couple of helper functions that will allow your JavaScript to determine the correct event model to use.
- The next two pages illustrate these two functions and I will also place them on the course web page for you to download and use. From a JavaScript perspective the functions are a little complex, so don’t worry if you don’t fully understand how they work. Remember that this is the beauty of “black boxing”.

John Resig's `addEvent` Helper Function

```
function addEvent( obj, type, fn ) {
  if ( obj.attachEvent ) {
    obj['e'+type+fn] = fn;
    obj[type+fn] = function(){obj['e'+type+fn]  ( 
window.event );}
    obj.attachEvent( 'on'+type, obj[type+fn] );
  } else
    obj.addEventListener(type, fn, false);
}
```

John Resig’s `removeEvent` Helper Function

```
function removeEvent( obj, type, fn ) {
  if ( obj.detachEvent ) {
    obj['e'+type+fn] = fn;
    obj.detachEvent( 'on'+type, obj[type+fn] );
    obj[type+fn] = null;
  } else
    obj.removeListener(type, fn, false);
}
```

Using John Resig's Helper Functions

- Black boxing means that you don't need to understand how John Resig's functions work, just know what they do and how to use them.
- If you want to add an event listener to the email field form in the previous example, all you would need to do is call the `addEvent` helper function like this:

```
addEvent(emailField, ‘focus’, doHighLight);
```

- The three arguments are the element, the event, and the function to call when the element receives that event. Resig's function then takes care of formatting the event registration appropriately for the browser on which it is running. I'll use Resig's functions from this point on.

The First Event: `load`

- Typically, the first thing you want JavaScript to do is set up the initial state of the page so its ready for use by the visitor.
- A very common part of this initialization process is to attach event listeners to the elements in the DOM that will respond to user actions, and you cannot do that until the DOM has loaded into the browser.
- For example, you might want to attach `blur` events to the text fields of a form so you can detect when the user click or tabs away from them. You can then immediately validate the text the user entered.
- To help you ensure that you are working with a DOM that actually exists, a load event is issued when the page is entirely loaded.

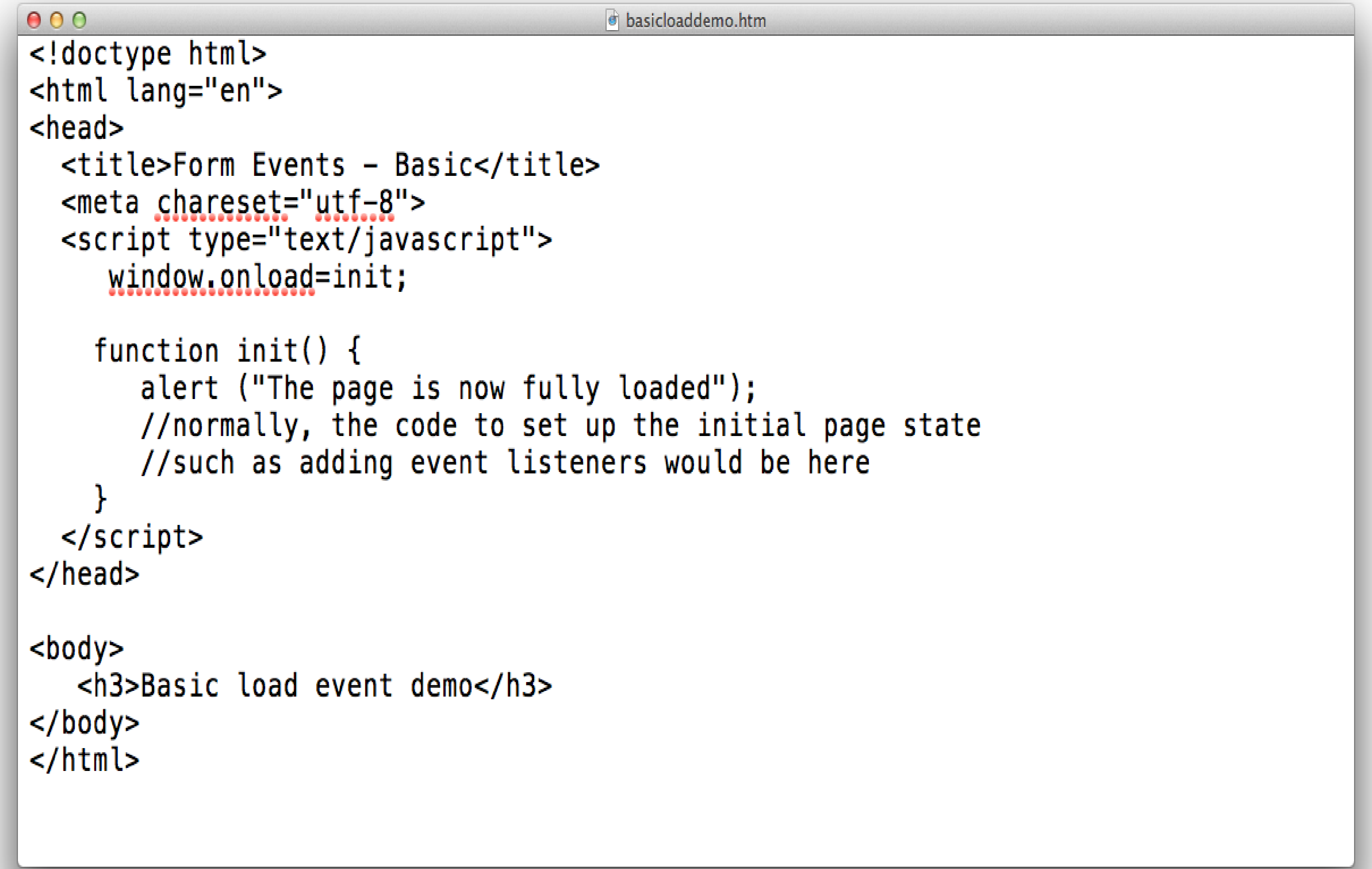
The First Event: `load`

- You can use the `onload` event handler to detect this event and trigger the JavaScript functions that will set up the page state for the user.

- The example on the next page illustrates this technique.

- Notice that in the JavaScript that the first line calls the `init` function; there are no parentheses after the `init` function name. You would normally add parentheses after a function name because you would want the function to run immediately at that point in the code.

- However, because you are setting up an event that will call the function at a later time, you don't do that here.

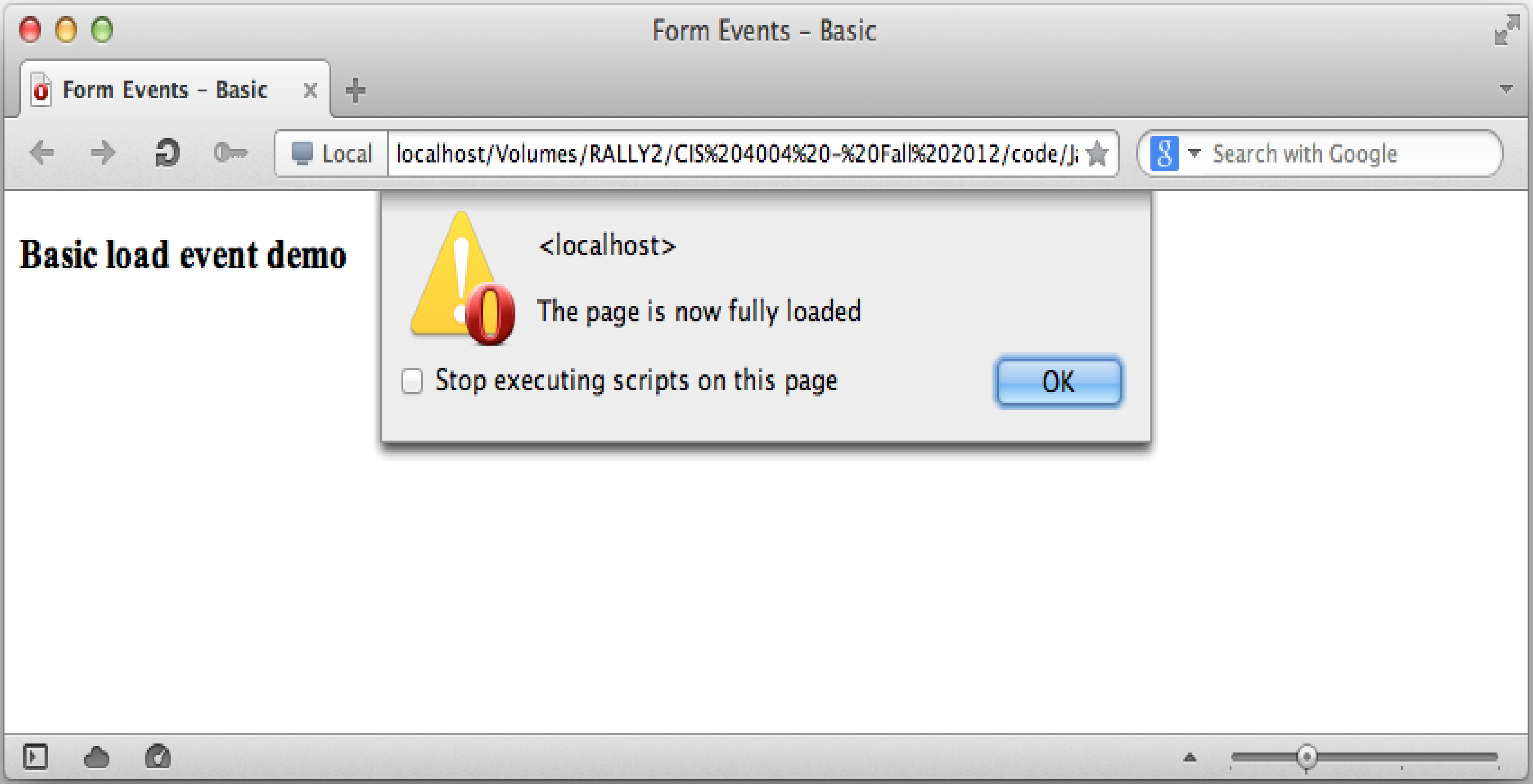


basicloaddemo.htm

```
<!doctype html>
<html lang="en">
<head>
  <title>Form Events - Basic</title>
  <meta chareset="utf-8">
  <script type="text/javascript">
     window.onload=init;

    function init() {
       alert ("The page is now fully loaded");
       //normally, the code to set up the initial page state
       //such as adding event listeners would be here
    }
  </script>
</head>

<body>
   <h3>Basic load event demo</h3>
</body>
</html>
```


Form Events - Basic

localhost/Volumes/RALLY2/CIS%204004%20-%20Fall%202012/code/J

Basic load event demo

<localhost>

The page is now fully loaded

The First Event: `load`

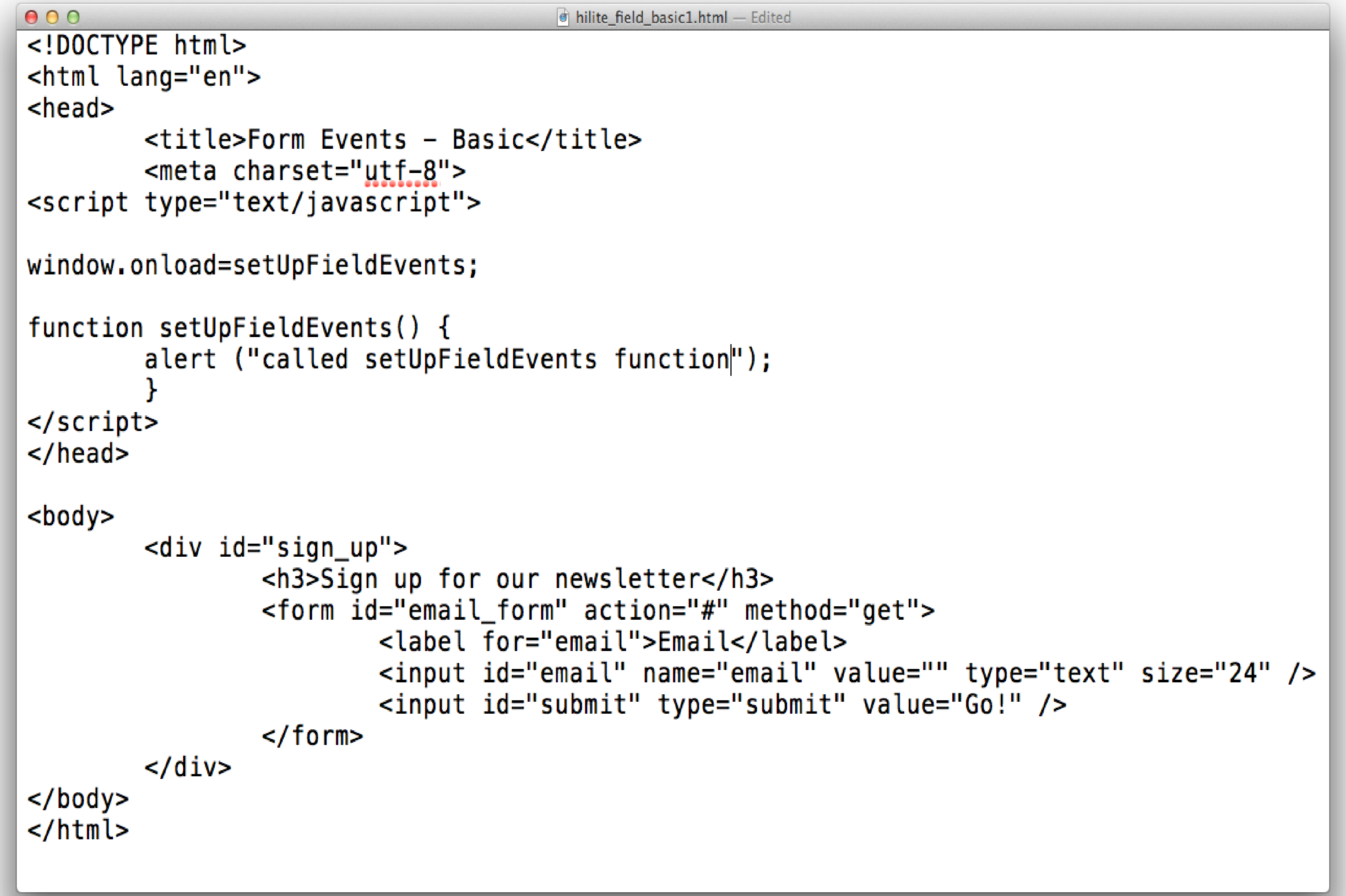
- If you wrote `window.onload= init();` the function would run immediately (setting the `onload` property to the result of the function) and not wait for the page load event to be sent.
- By omitting the parentheses when you assign the `init` function to the `onload` property, the function does not run immediately, instead, it runs when the load event occurs after the page is fully loaded.
- Also note that `onload` is a method of the `window` object, so you must always precede it with `window`, for it to work.
- Note too, that any JavaScript statement not enclosed in a function and just "loose" on the page runs as soon as it loads. For this reason, it's very unusual to place any JavaScript except the `onload` event assignment outside of a function.

Adding Event Listeners on Page Load

- After all of the previous discussion, we'll now look at a simple example of event listeners that are added to an element when the page loads.
- In this example case, when the `onload` event handler calls the `init` function, it will add event listeners to a text field.
- As a result of the functions called by these event listeners, the text field will highlight (its background will be set to green) when the field receives focus; it will unhighlight (the default white background be restored) when the focus is removed.
- We'll develop this example in a systematic manner which might help you with the techniques you can use in developing your own projects.

Adding Event Listeners on Page Load

- Step 1 in the development process is to ensure that the load event is triggering the function that will set up the event listeners.
- The markup for this example is shown on the next page, but the only significant element is the form input field.
- Notice that all I did was set up the `onload` event to trigger the function `setUpFieldEvents`. In order to ensure that the function is being called properly, I just used a JavaScript alert box to display. So I now know that the function is being triggered properly by the `onload` event.
- As with some of the other examples, I'm including the JavaScript in the markup file for ease of viewing here…normally it would be external to the markup.

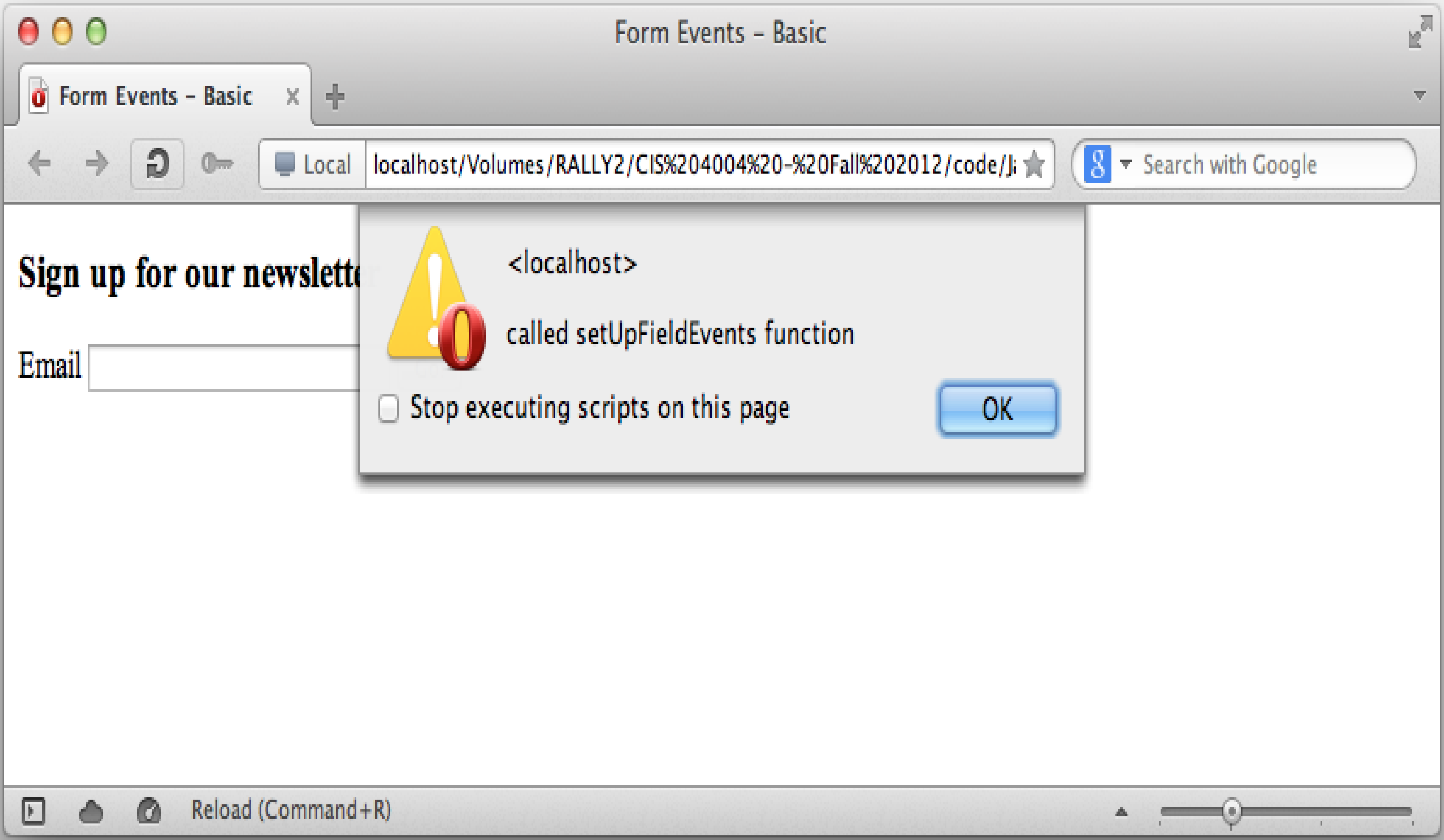

hilite_field_basic1.html — Edited

```
<!DOCTYPE html>
<html lang="en">
<head>
	<title>Form Events - Basic</title>
	<meta charset="utf-8">
<script type="text/javascript">

window.onload=setUpFieldEvents;

function setUpFieldEvents() {
	alert ("called setUpFieldEvents function");
	}
</script>
</head>

<body>
	<div id="sign_up">
		<h3>Sign up for our newsletter</h3>
		<form id="email_form" action="#" method="get">
			<label for="email">Email</label>
			<input id="email" name="email" value="" type="text" size="24" />
			<input id="submit" type="submit" value="Go!" />
		</form>
	</div>
</body>
</html>
```



Form Events - Basic

localhost/Volumes/RALLY2/CIS%204004%20-%20Fall%202012/code/J

Sign up for our newslett

Email

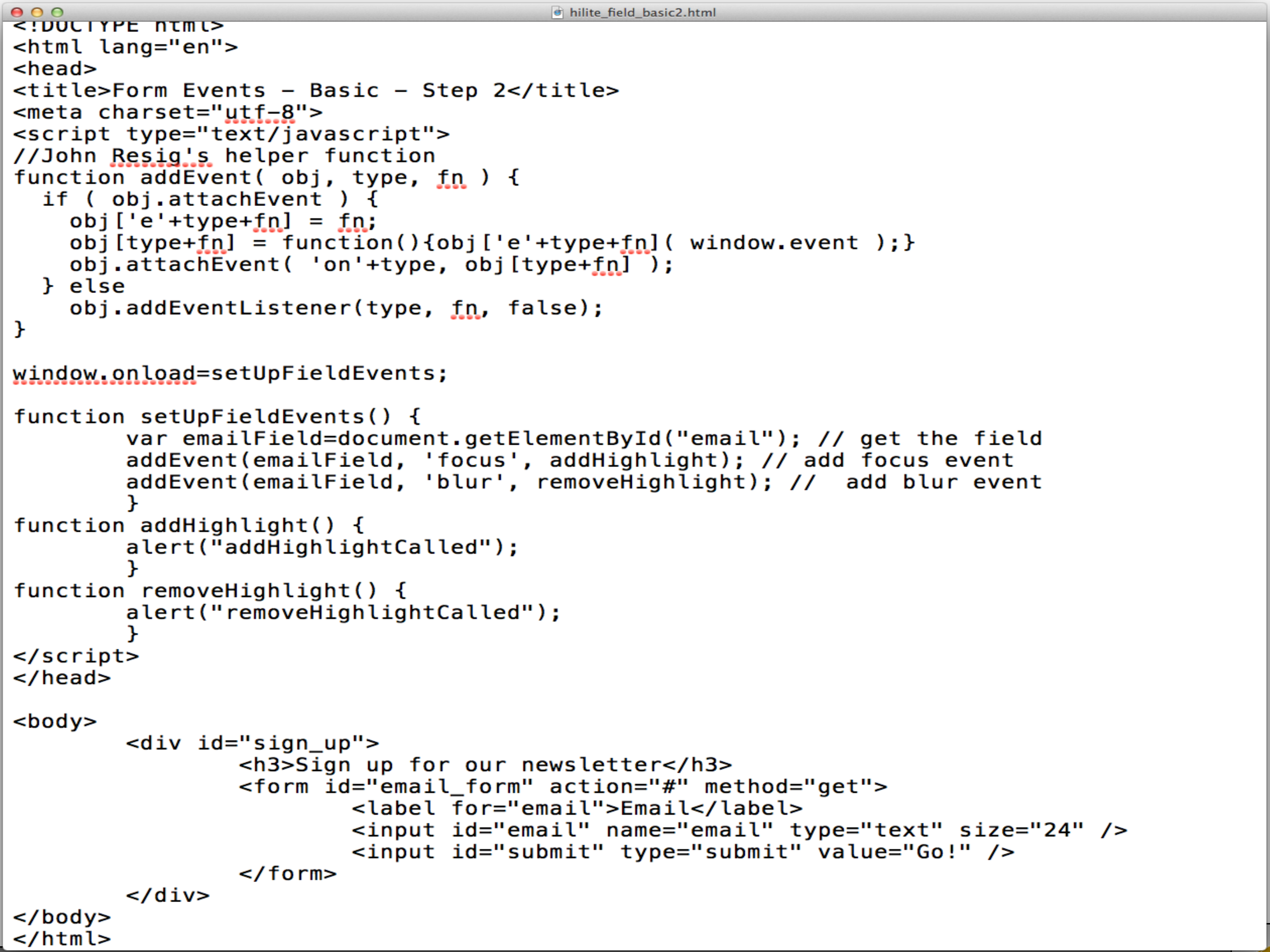
<localhost>

called setUpFieldEvents function

Stop executing scripts on this page

Adding Event Listeners on Page Load

- In step 2 we'll actually add the code to the `setUpFieldEvents` function that will add the event listeners. We'll use Resig's `addEvent` helper function to ensure that our page will render properly in any browser.
- The markup is shown on the next page.



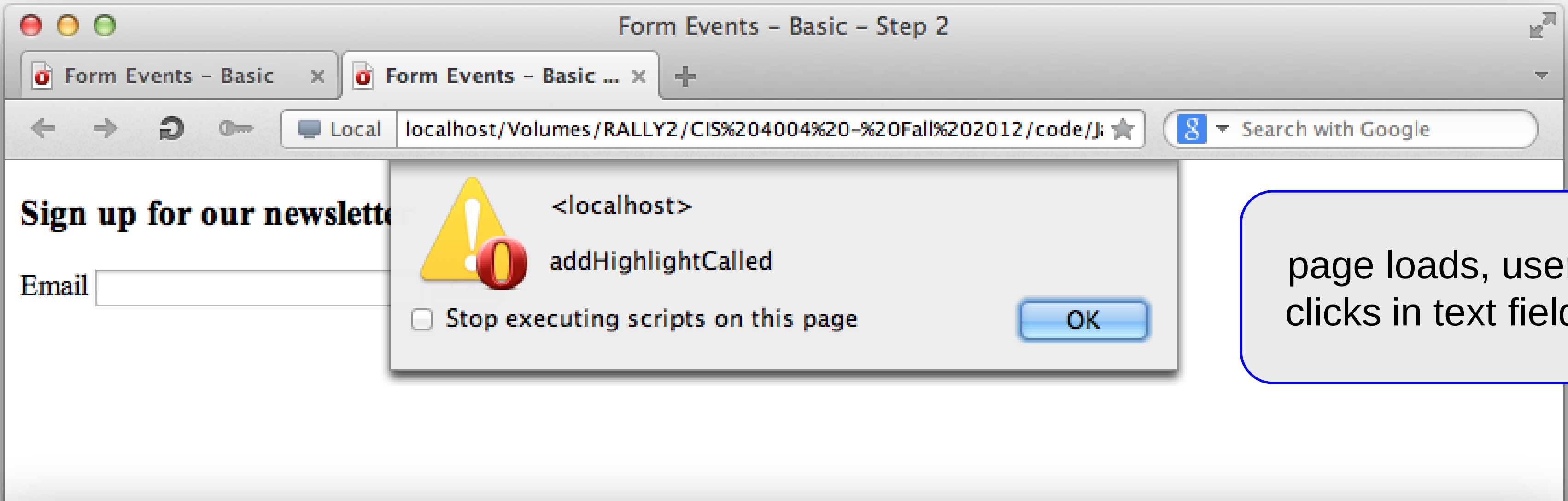
hilite_field_basic2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Form Events - Basic - Step 2</title>
<meta charset="utf-8">
<script type="text/javascript">
//John Resig's helper function
function addEvent( obj, type, fn ) {
  if ( obj.attachEvent ) {
    obj['e'+type+fn] = fn;
    obj[type+fn] = function(){obj['e'+type+fn]( window.event );}
    obj.attachEvent( 'on'+type, obj[type+fn] );
  } else
    obj.addEventListener(type, fn, false);
}

window.onload=setUpFieldEvents;

function setUpFieldEvents() {
        var emailField=document.getElementById("email"); // get the field
        addEvent(emailField, 'focus', addHighlight); // add focus event
        addEvent(emailField, 'blur', removeHighlight); //  add blur event
        }
function addHighlight() {
        alert("addHighlightCalled");
        }
function removeHighlight() {
        alert("removeHighlightCalled");
        }
</script>
</head>

<body>
        <div id="sign_up">
                <h3>Sign up for our newsletter</h3>
                <form id="email_form" action="#" method="get">
                        <label for="email">Email</label>
                        <input id="email" name="email" type="text" size="24" />
                        <input id="submit" type="submit" value="Go!" />
                </form>
        </div>
</body>
</html>
```



Form Events – Basic – Step 2

Sign up for our newslett

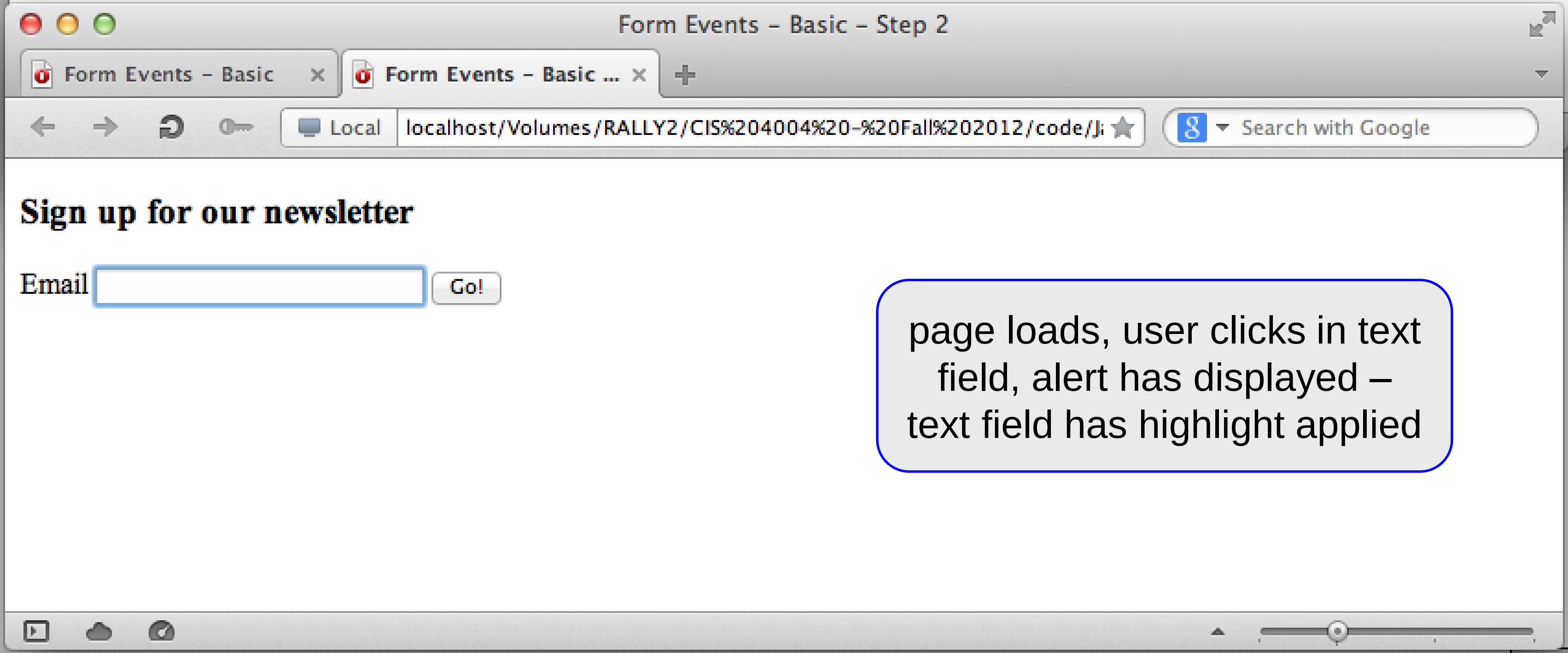
Email

<localhost>

addHighlightCalled

Stop executing scripts on this page

page loads, user clicks in text field

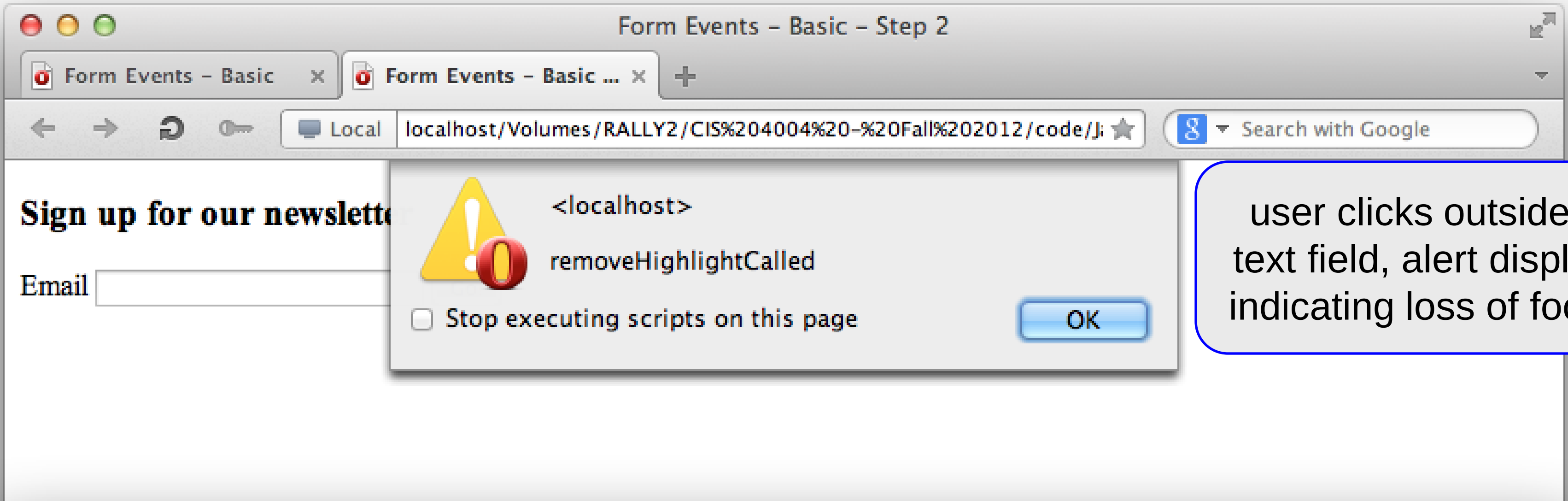


Form Events – Basic – Step 2

Sign up for our newsletter

Email

page loads, user clicks in text field, alert has displayed – text field has highlight applied



Form Events – Basic – Step 2

Form Events – Basic | Form Events – Basic ...

Local | localhost/Volumes/RALLY2/CIS%204004%20-%20Fall%202012/code/J

Search with Google

Sign up for our newslett

Email

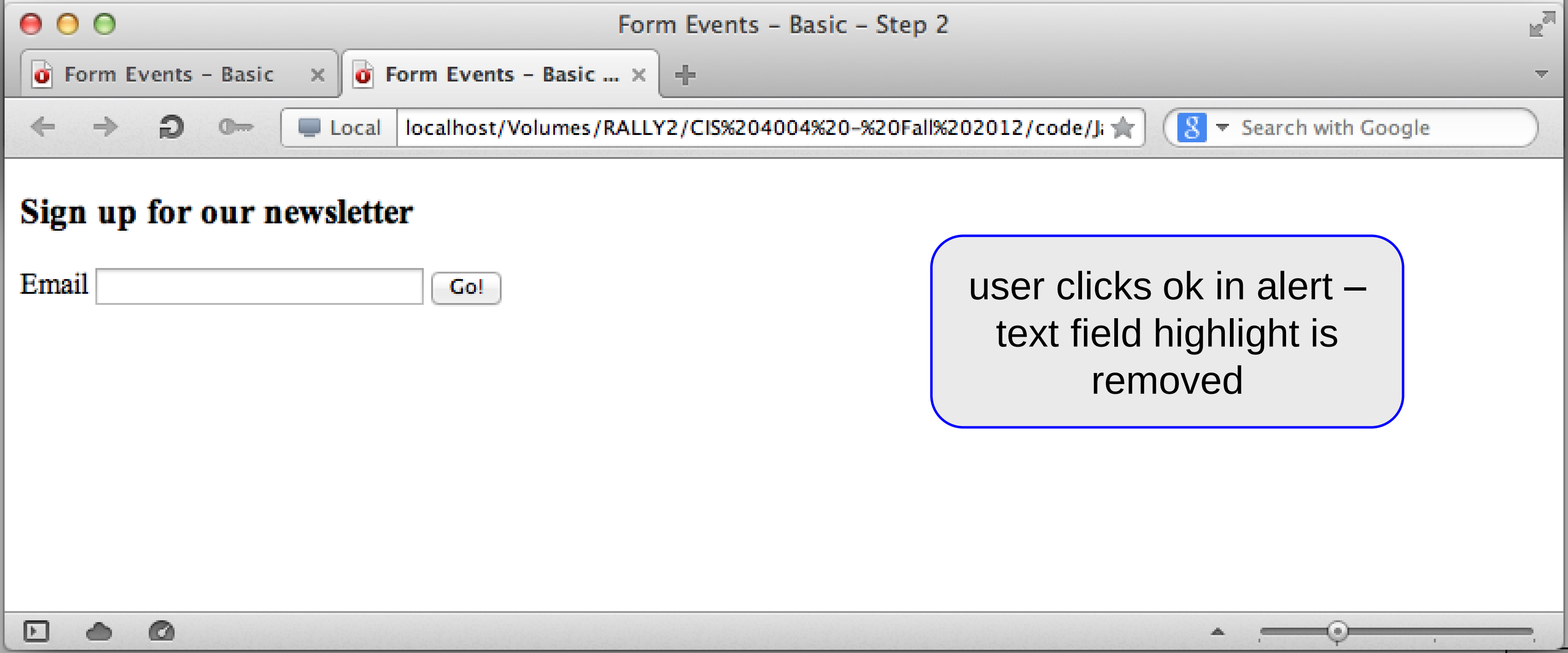
<localhost>

removeHighlightCalled

Stop executing scripts on this page

OK

user clicks outside of text field, alert displays indicating loss of focus.



Form Events – Basic – Step 2

Form Events – Basic | Form Events – Basic ...

Local | localhost/Volumes/RALLY2/CIS%204004%20-%20Fall%202012/code/J

Search with Google

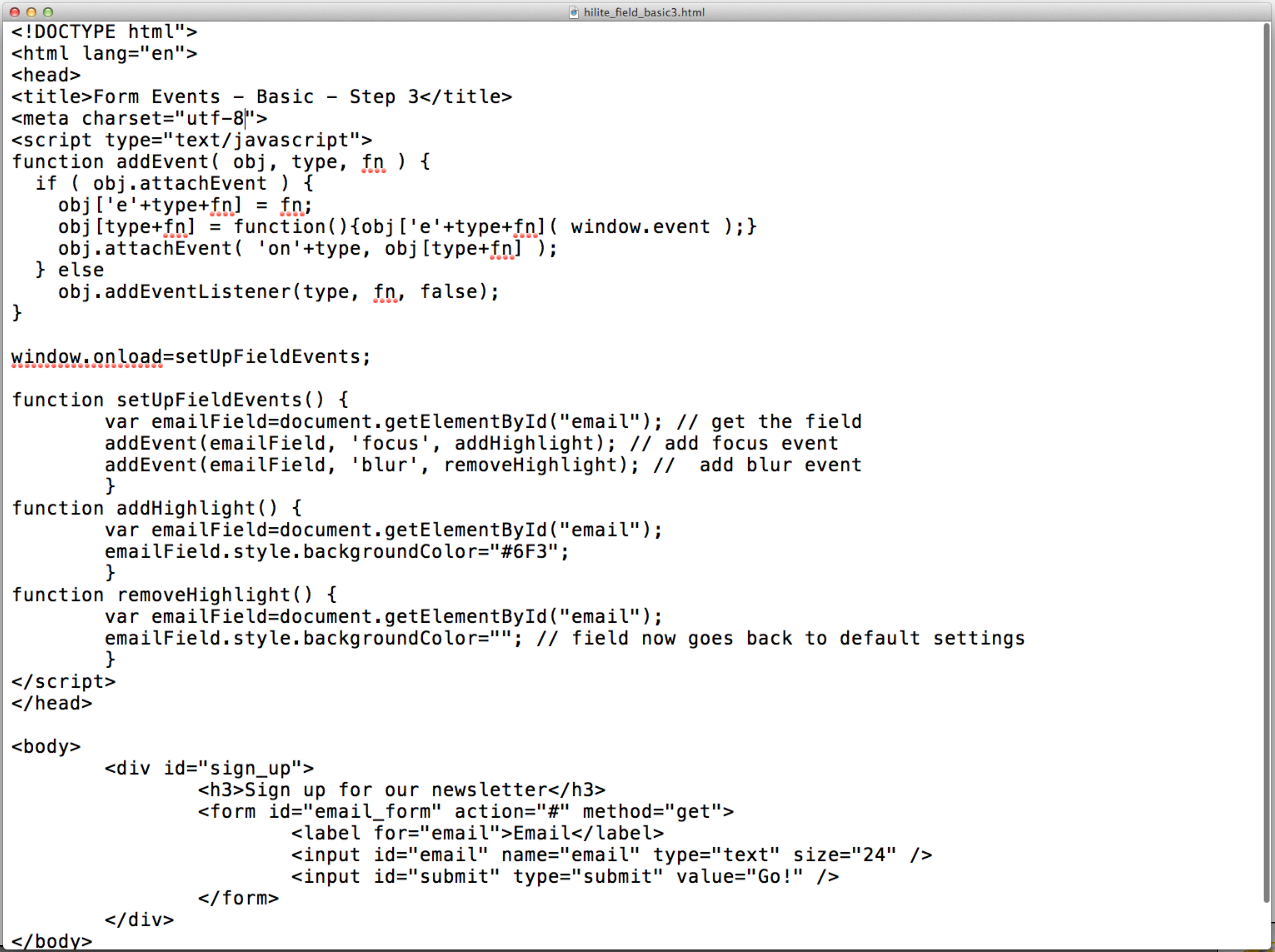
Sign up for our newsletter

Email Go!

user clicks ok in alert – text field highlight is removed

Adding Event Listeners on Page Load

- In step 3 we’ll replace the alert code in the functions with the actual highlighting that we originally intended.
- The markup is shown on the next page.

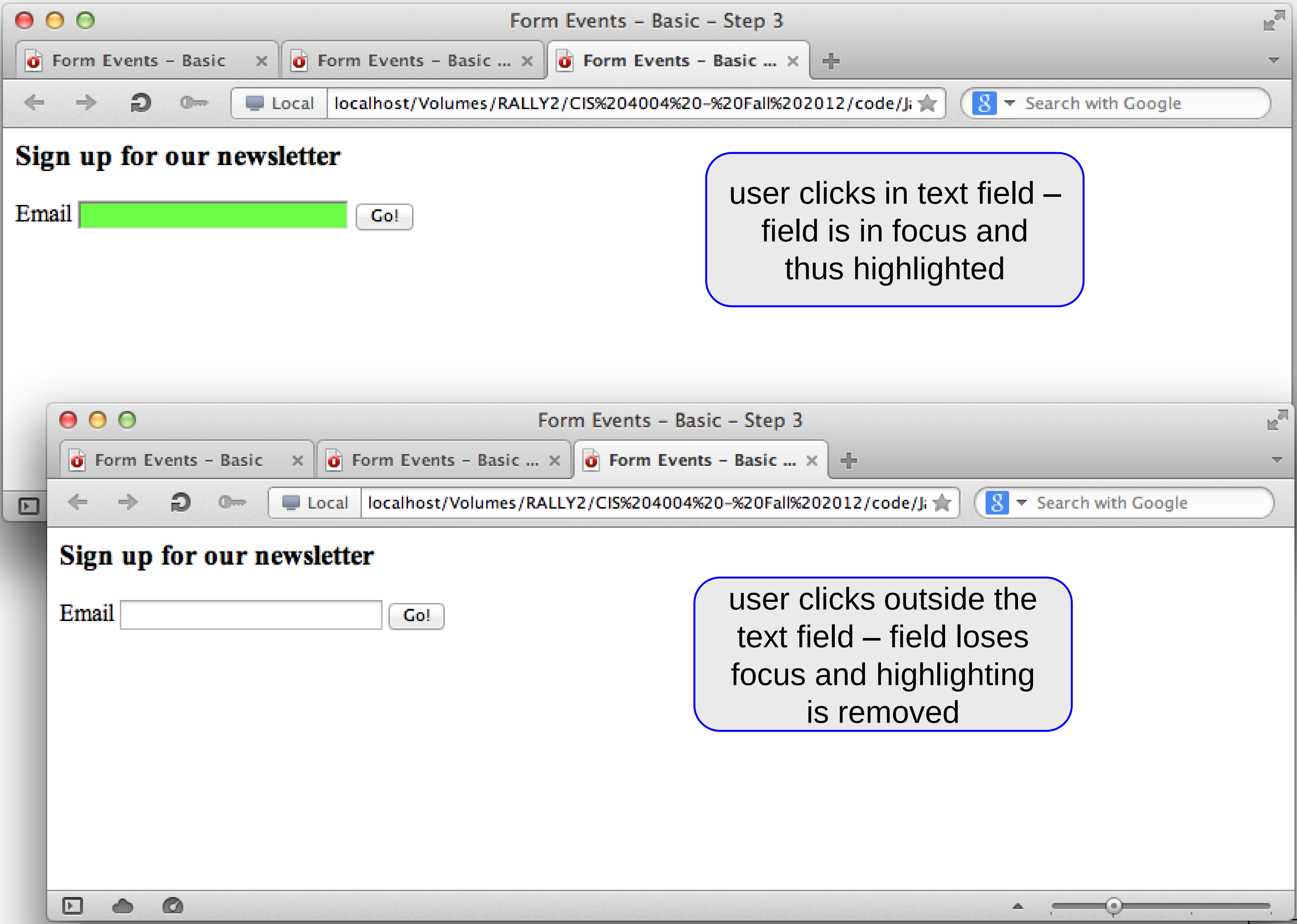

hilite_field_basic3.html

```
<!DOCTYPE html">
<html lang="en">
<head>
<title>Form Events - Basic - Step 3</title>
<meta charset="utf-8">
<script type="text/javascript">
function addEvent( obj, type, fn ) {
  if ( obj.attachEvent ) {
    obj['e'+type+fn] = fn;
    obj[type+fn] = function(){obj['e'+type+fn]( window.event );}
    obj.attachEvent( 'on'+type, obj[type+fn] );
  } else
    obj.addEventListener(type, fn, false);
}

window.onload=setUpFieldEvents;

function setUpFieldEvents() {
        var emailField=document.getElementById("email"); // get the field
        addEvent(emailField, 'focus', addHighlight); // add focus event
        addEvent(emailField, 'blur', removeHighlight); //  add blur event
        }
function addHighlight() {
        var emailField=document.getElementById("email");
        emailField.style.backgroundColor="#6F3";
        }
function removeHighlight() {
        var emailField=document.getElementById("email");
        emailField.style.backgroundColor=""; // field now goes back to default settings
        }
</script>
</head>

<body>
        <div id="sign_up">
                <h3>Sign up for our newsletter</h3>
                <form id="email_form" action="#" method="get">
                        <label for="email">Email</label>
                        <input id="email" name="email" type="text" size="24" />
                        <input id="submit" type="submit" value="Go!" />
                </form>
        </div>
</body>
```



Form Events – Basic – Step 3

Form Events – Basic | Form Events – Basic ... | Form Events – Basic ...

Local localhost/Volumes/RALLY2/CIS%204004%20-%20Fall%202012/code/J

Search with Google

Sign up for our newsletter

Email [ ] Go!

user clicks in text field – field is in focus and thus highlighted

Form Events – Basic – Step 3

Form Events – Basic | Form Events – Basic ... | Form Events – Basic ...

Local localhost/Volumes/RALLY2/CIS%204004%20-%20Fall%202012/code/J

Search with Google

Sign up for our newsletter

Email [ ] Go!

user clicks outside the text field – field loses focus and highlighting is removed