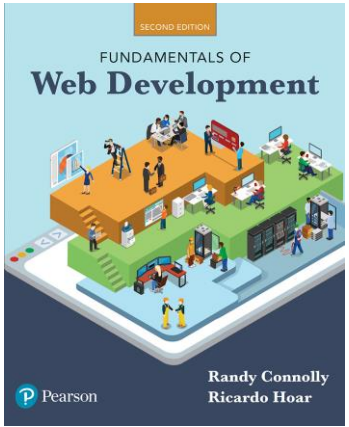
CSE 686 Internet Programming

Week 11: Introduction to JavaScript, Part 2: Using JavaScript

Edmund Yu, PhD
Associate Teaching Professor
esyu@syr.edu

March 28, 2018



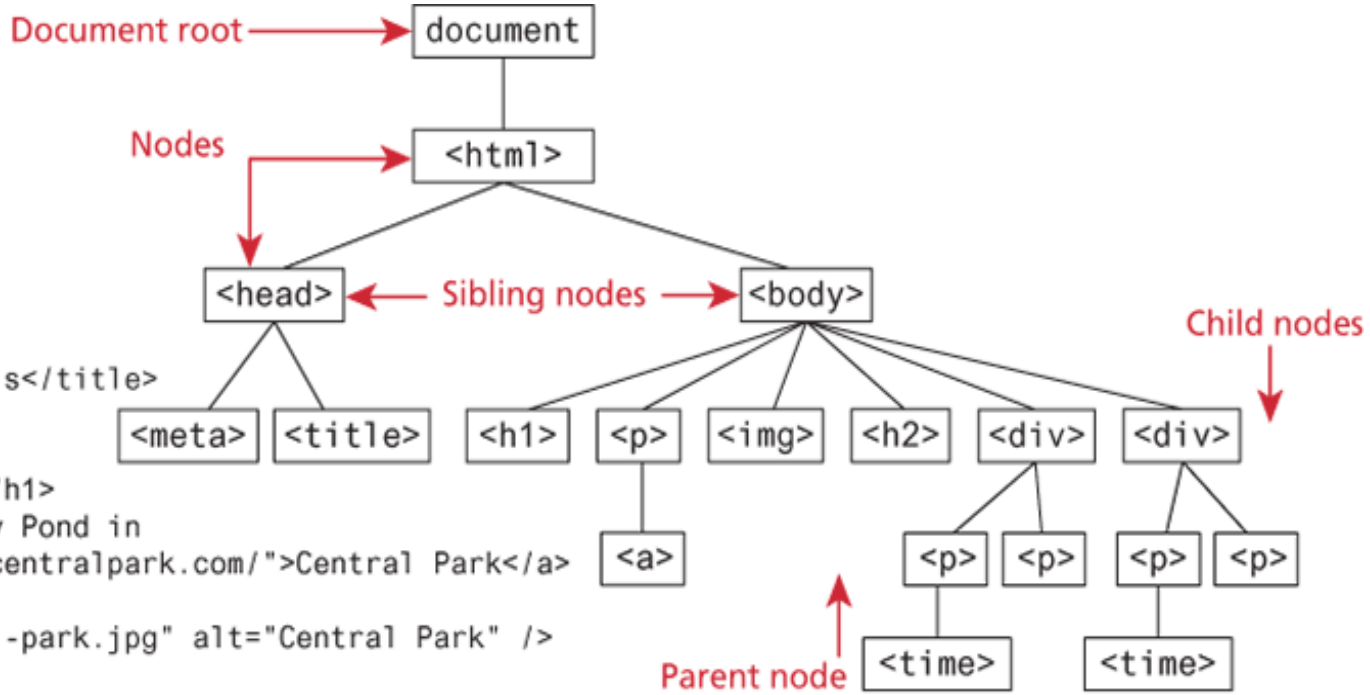
The Document Object Model (DOM)

- JavaScript is almost always used to interact with the HTML document in which it is contained. As such, there needs to be some way of programmatically accessing the elements and attributes within the HTML (API). → the Document Object Model (DOM).

- According to the W3C, the DOM is a:

  Platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.

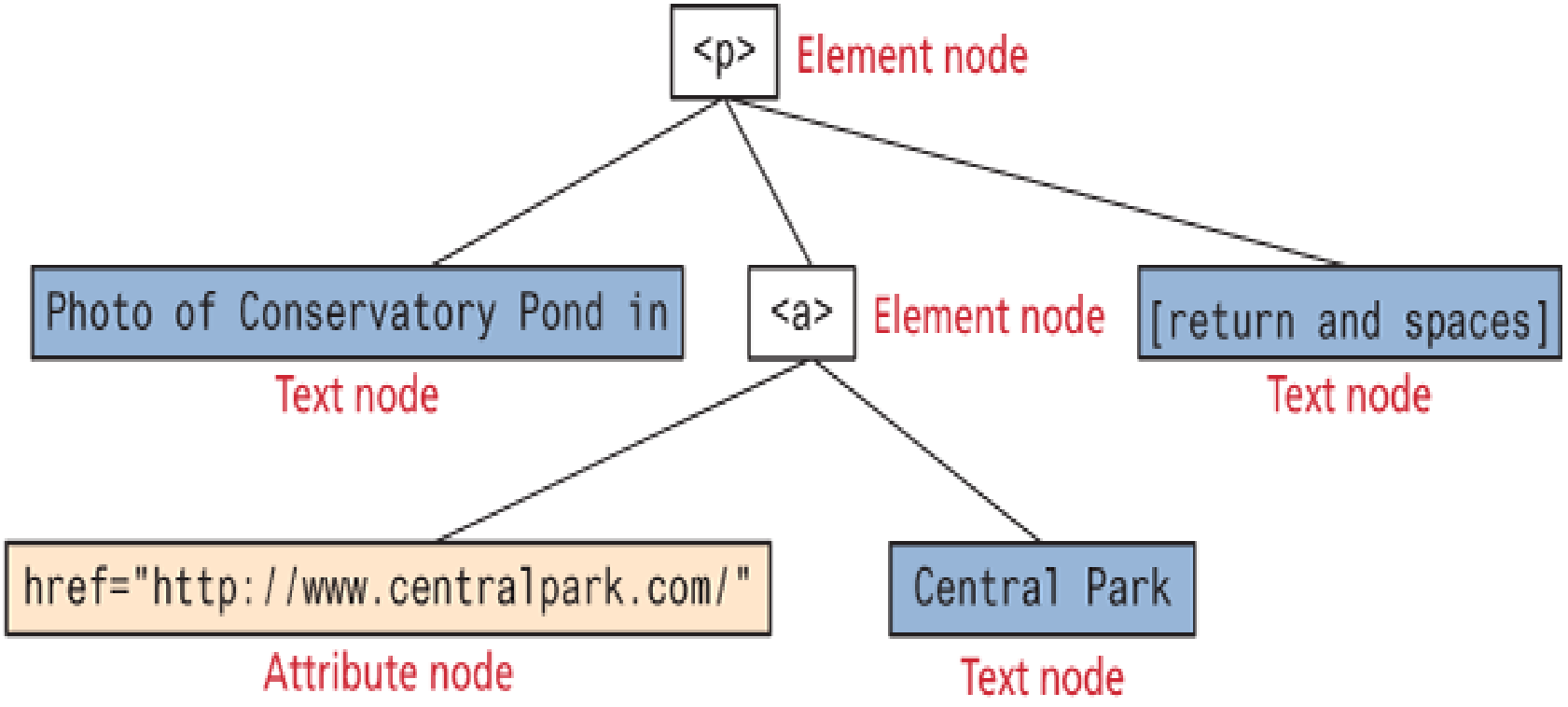
DOM Revisited




```
<html>
<head>
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>By Ricardo on <time>2016-05-23</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```

Nodes

```
<p>Photo of Conservatory Pond in
   <a href="http://www.centralpark.com/">Central Park</a>
</p>
```



Document Object

- The DOM document object is the root JavaScript object representing the entire HTML document

```
// retrieve the URL of the current page
var a = document.URL;
// retrieve the page encoding, for example ISO-8859-1
var b = document.inputEncoding;
```

- In addition to these properties, there are several essential methods you will use all the time. E.g. **document.write()**
- We can will group the methods into these three categories
  - **Selection methods**
  - Family manipulation methods
  - Event methods

JavaScript Output (3)

- **document.write**() Outputs the content directly to the HTML document.
  - This method can be a useful way to output markup content from within JavaScript.
  - This method is often used to output markup or to combine markup with JavaScript variables, as shown in the following example:

```
var name = "Randy";
document.write("<h1>Title</h1>");
// this uses the concatenate operator (+)
document.write("Hello " + name + " and welcome");
```

Run »

Result Size: 556 x 439

```
<html>
<body>

<script>
var name = "Randy";
document.write("<h1>Title</h1>");
// this uses the concatenate operator (+)
document.write("Hello " + name + " and welcome");
</script>
</body>
</html>
```

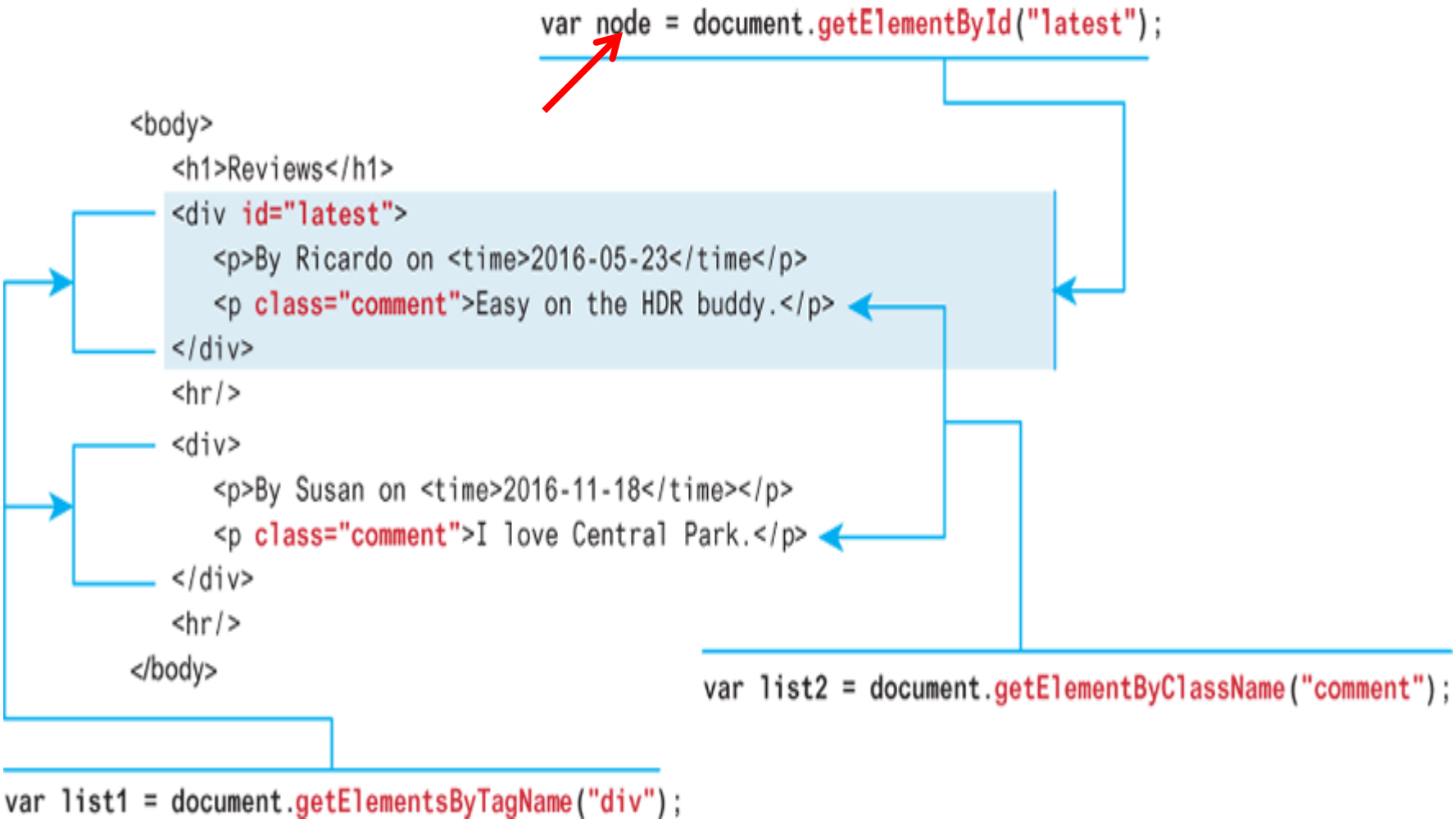
Title

Hello Randy and welcome

Table 9.2 Selection DOM Methods

Method	Description
`getElementById(id)`	Returns the single element node whose `id` attribute matches the passed `id`.
`getElementsByClassName(name)`	Returns a `NodeList` of elements whose class name matches the passed `name`.
`getElementsByTagName(name)`	Returns a `NodeList` of elements whose tag name matches the passed `name`.
`querySelector(selector)`	Returns the first element node that matches the passed CSS selector.
`querySelectorAll(selector)`	Returns a `NodeList` of elements that match the passed CSS selector.

getElement()


var node = document.getElementById("latest");
<body>
<h1>Reviews</h1>

By Ricardo on <time>2016-05-23</time>
Easy on the HDR buddy.

<hr/>

By Susan on <time>2016-11-18</time>
I love Central Park.

<hr/>
</body>
var list2 = document.getElementByClassName("comment");
var list1 = document.getElementsByTagName("div");

Table 9.3 Some Essential Element Node Properties

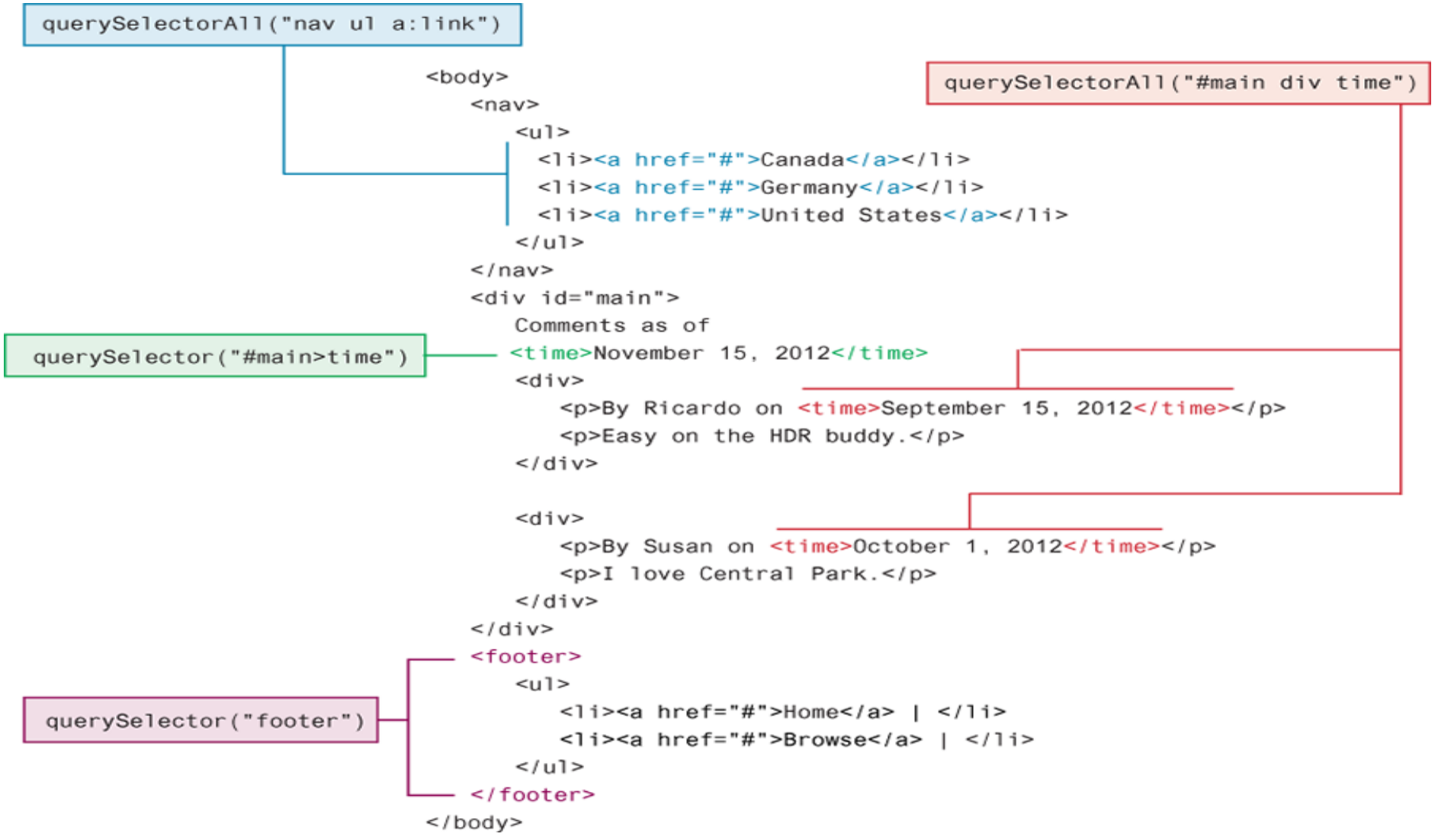
Property	Description
`classList`	A read-only list of CSS classes assigned to this element. This list has a variety of helper methods for manipulating this list.
`className`	The current value for the `class` attribute of this HTML element.
`id`	The current value for the `id` of this element.
`innerHTML`	Represents all the content (text and tags) of the element.
`style`	The style attribute of an element. This returns a `CSSStyleDeclaration` object that contains sub-properties that correspond to the various CSS properties.
`tagName`	The tag name for the element.

Properties and Methods

The following properties and methods can be used on all HTML elements:

Property / Method	Description
element.accessKey	Sets or returns the accesskey attribute of an element
element.addEventListener()	Attaches an event handler to the specified element
element.appendChild()	Adds a new child node, to an element, as the last child node
element.attributes	Returns a NamedNodeMap of an element's attributes
element.blur()	Removes focus from an element
element.childElementCount	Returns the number of child elements an element has
element.childNodes	Returns a collection of an element's child nodes (including text and comment nodes)
element.children	Returns a collection of an element's child element (excluding text and comment nodes)
element.classList	Returns the class name(s) of an element
element.className	Sets or returns the value of the class attribute of an element

querySelector and querySelectorAll



See also: https://www.w3schools.com/cssref/css_selectors.asp

Table 9.4 Some Specific HTML DOM Element Properties for Certain Tag Types

Property	Description	Tags
`href`	Used in `<a>` tags to specify the linking URL.	`a`
`name`	Used to identify a tag. Unlike `id` which is available to all tags, `name` is limited to certain form-related tags.	`a, input, textarea, form`
`src`	Links to an external URL that should be loaded into the page (as opposed to `href` which is a link to follow when clicked).	`img, input, iframe, script`
`value`	Provides access to the `value` attribute of input tags. Typically used to access the user's input into a form field.	`input, textarea, submit`

Listing 9.1 Accessing elements and their properties

```
<p id="here">hello <span>there</span></p>
<ul>
    <li>France</li>
    <li>Spain</li>
    <li>Thailand</li>
</ul>
<div id="main">
    <a href="somewhere.html"><img src="whatever.gif" class="thumb"></a>
</div>
<script>
    var node = document.getElementById("here");
    // outputs: hello <span>there</span>
    console.log(node.innerHTML);
    // outputs: hello there
    console.log(node.textContent);

    var items = document.getElementsByTagName("li");
    for (var i=0; i<items.length; i++) {
        // outputs: France, then Spain, then Thailand
        console.log(items[i].textContent);
    }

    var link = document.querySelector("#main a");
    // outputs: somewhere.html
    console.log(link.href);

    var img = document.querySelector("#main img");
    // outputs: whatever.gif
    console.log(img.src);
    // outputs: thumb
```

Modifying the DOM: Styles

```
<style>
    .box {
        margin: 2em; padding: 0;
        border: solid 1pt black;
    }
   .yellowish { background-color: #EFE63F; }
   .hide { display: none; }
</style>
<main>
   <div class="box">
    ...
   </div>
</main>
```

```
var node = document.querySelector("main div");
```

1. `node.className = "yellowish";`
   This replaces the existing class specification with this one. Thus the no longer has the box class

   Equivalent to: `<div class="yellowish">`

2. `node.classList.remove("yellowish");`
   `node.classList.add("box");`
   Removes the specified class specification and adds the box class

   Equivalent to: `<div class="">` `<div class="box">`

3. `node.classList.add("yellowish");`
   Adds a new class to the existing class specification

   Equivalent to: `<div class="box yellowish">`

4. `node.classList.toggle("hide");`
   If it isn't in the class specification, then add it

   Equivalent to: `<div class="box yellowish hide">`

5. `node.classList.toggle("hide");`
   If it is in the class specification, then remove it

   Equivalent to: `<div class="box yellowish">`

Modifying the DOM: Contents

document.getElementById("here").**innerHTML** = "foobar";

- Modify Listing 9-1 to do this.

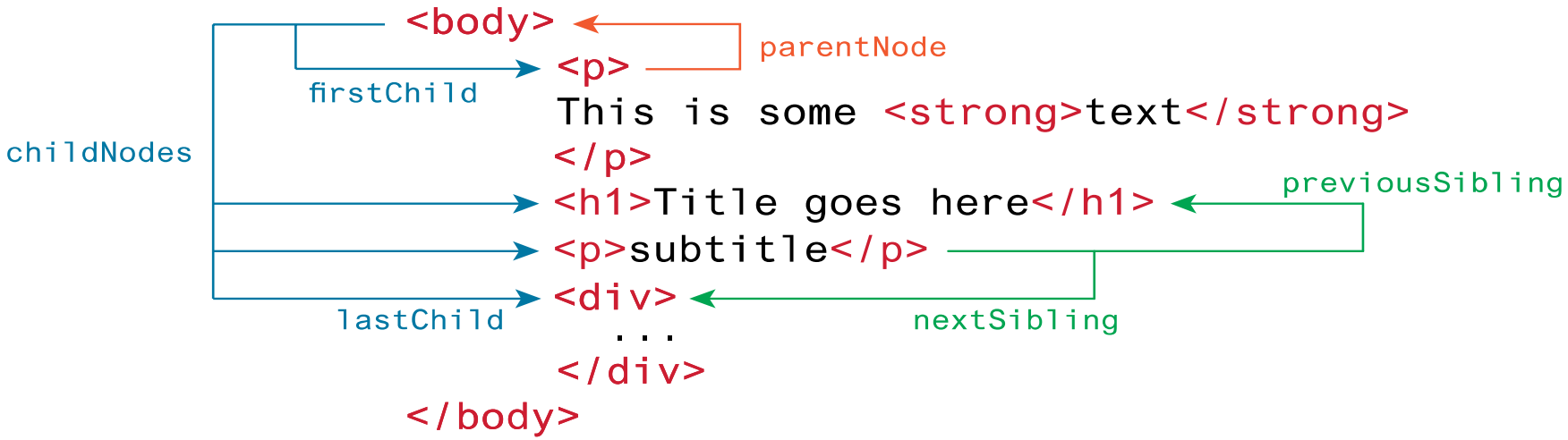
Document Object

- The DOM document object is the root JavaScript object representing the entire HTML document

```
// retrieve the URL of the current page
var a = document.URL;
// retrieve the page encoding, for example ISO-8859-1
var b = document.inputEncoding;
```

- In addition to these properties, there are several essential methods you will use all the time. E.g. **document.write()**
- We can will group the methods into these three categories
  - Selection methods
  - **Family manipulation methods**
  - Event methods

Family Relations



Listing 9.2 Dynamically creating elements

```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
</div>
<script>
// begin by creating two new nodes
var text = document.createTextNode("this is dynamic");
var p = document.createElement("p");
// add the text node to the <p> element node
p.appendChild(text);
// now add the new <p> element to the <div>
var first = document.getElementById("first");
first.appendChild(p);
</script>
```

JavaScript Reference

Overview

JavaScript

JS Array
JS Boolean
JS Date
JS Error
JS Global
JS JSON
JS Math
JS Number
JS Operators
JS RegExp
JS Statements
JS String

HTML DOM

DOM Attributes
DOM Console
DOM Document
DOM Elements
DOM Events

The createElement() method creates an Element Node with the specified name.

Tip: Use the createTextNode() method to create a text node.

Tip: After the element is created, use the *element*.appendChild() or *element*.insertBefore() method to insert it to the document.

Browser Support

Method					
createElement()	Yes	Yes	Yes	Yes	Yes

Syntax

```
document.createElement(nodename)
```

Parameter Values

Parameter	Type	Description
nodename	String	Required. The name of the element you want to create

Top Navigation
Modal Boxes
Progress Bars
Parallax
Login Form
HTML Includes
Google Maps
Range Sliders
Tooltips
Slideshow
Filter List
Sort List

Creating DOM Elements

1. Create a new text node

```
var text = document.createTextNode("this is dynamic");
```

"this is dynamic"

2. Create a new empty element

```
var p = document.createElement("p");
```

3. Add the text node to new element

```
p.appendChild(text);
```

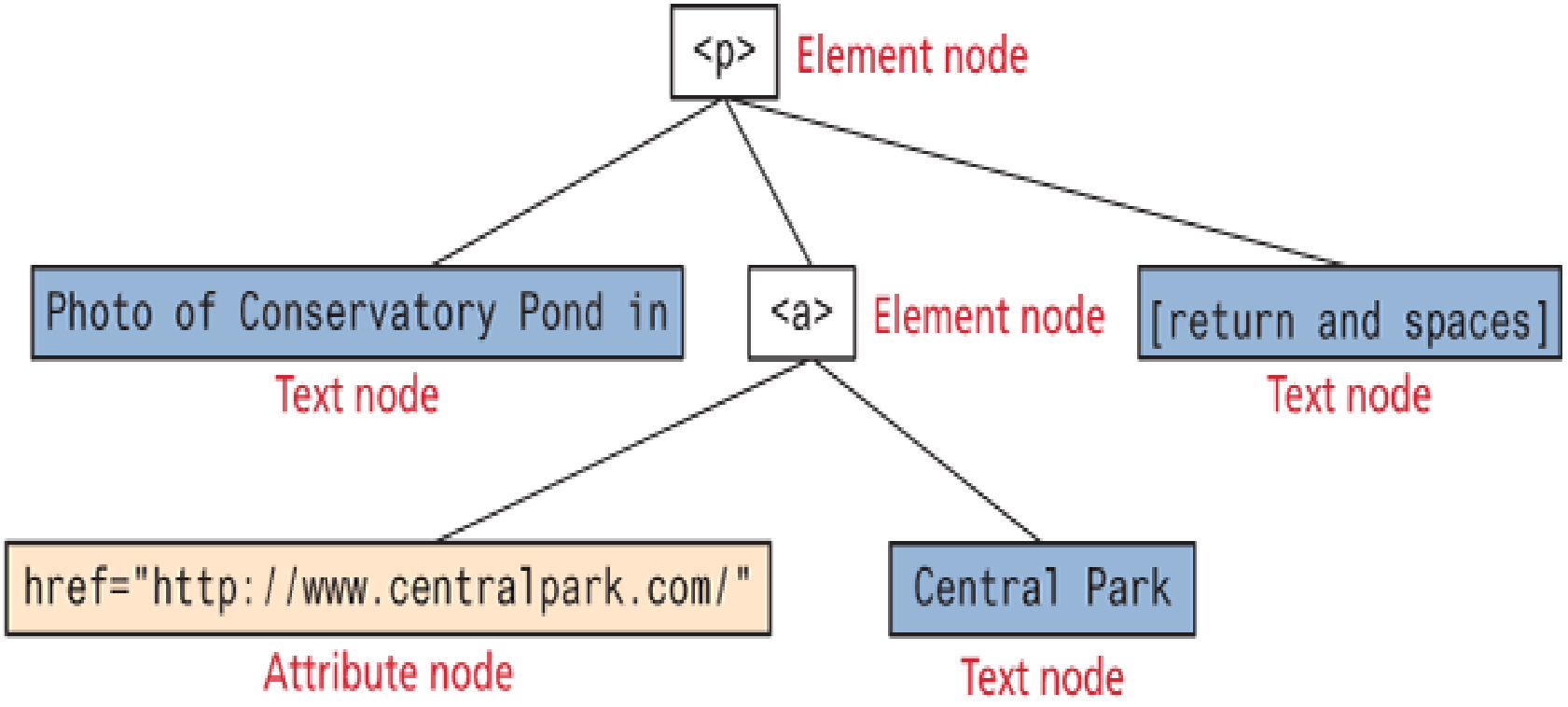
 "this is dynamic"

4. Add the element to the

```
var first = document.getElementById("first");
first.appendChild(p);
```

Text Nodes Revisited

```
<p>Photo of Conservatory Pond in
   <a href="http://www.centralpark.com/">Central Park</a>
</p>
```



Creating DOM Elements

4 Add the <p> element to the

```
var first = document.getElementById("first");
first.appendChild(p);
```

```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
    <p>this is dynamic</p>
</div>
```

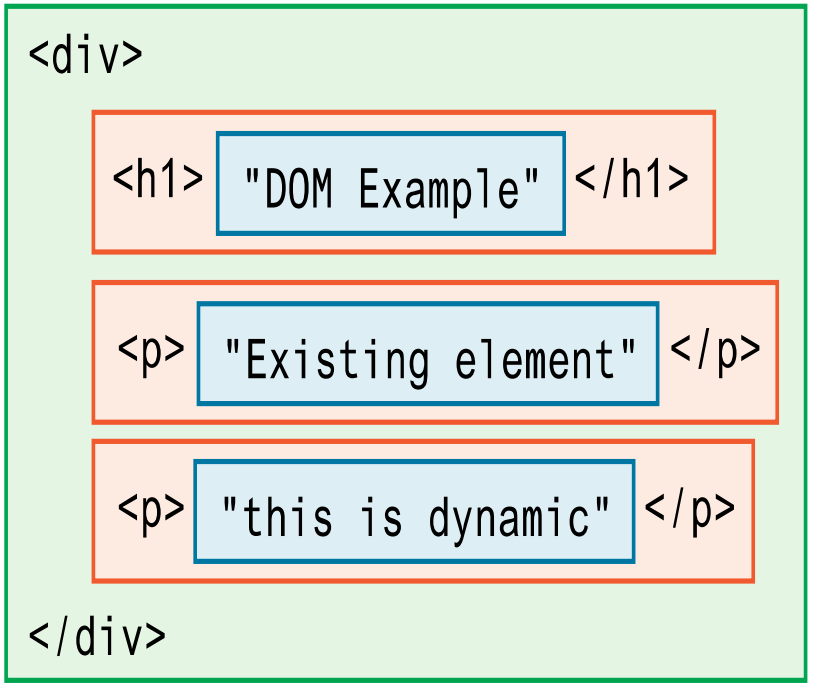


Table 9.5 DOM Manipulation Methods

Method	Description
`appendChild`	Adds a new child node to the end of the current node. `aParentNode.appendChild(newNode)`
`createAttribute`	Creates a new attribute node. `var newAttribute = document.createAttribute("name");`
`createElement`	Creates an HTML element node. `var newElement = document.createElement("tag");`
`createTextNode`	Creates a text node. `var newText = document.createTextNode("text content");`
`InsertBefore`	Inserts a new child node before a reference node in the current node. `aParentNode.insertBefore(newNode, referenceNode)`
`removeChild`	Removes a child from the current node. `aParentNode.removeChild(child)`
`replaceChild`	Replaces a child node with a different child. `aParentNode.replaceChild(newChild, oldChild)`

HTML DOM createAttribute() Method

‹ Document Object

Example

Create a class attribute, with the value "democlass", and insert it to an <h1> element:

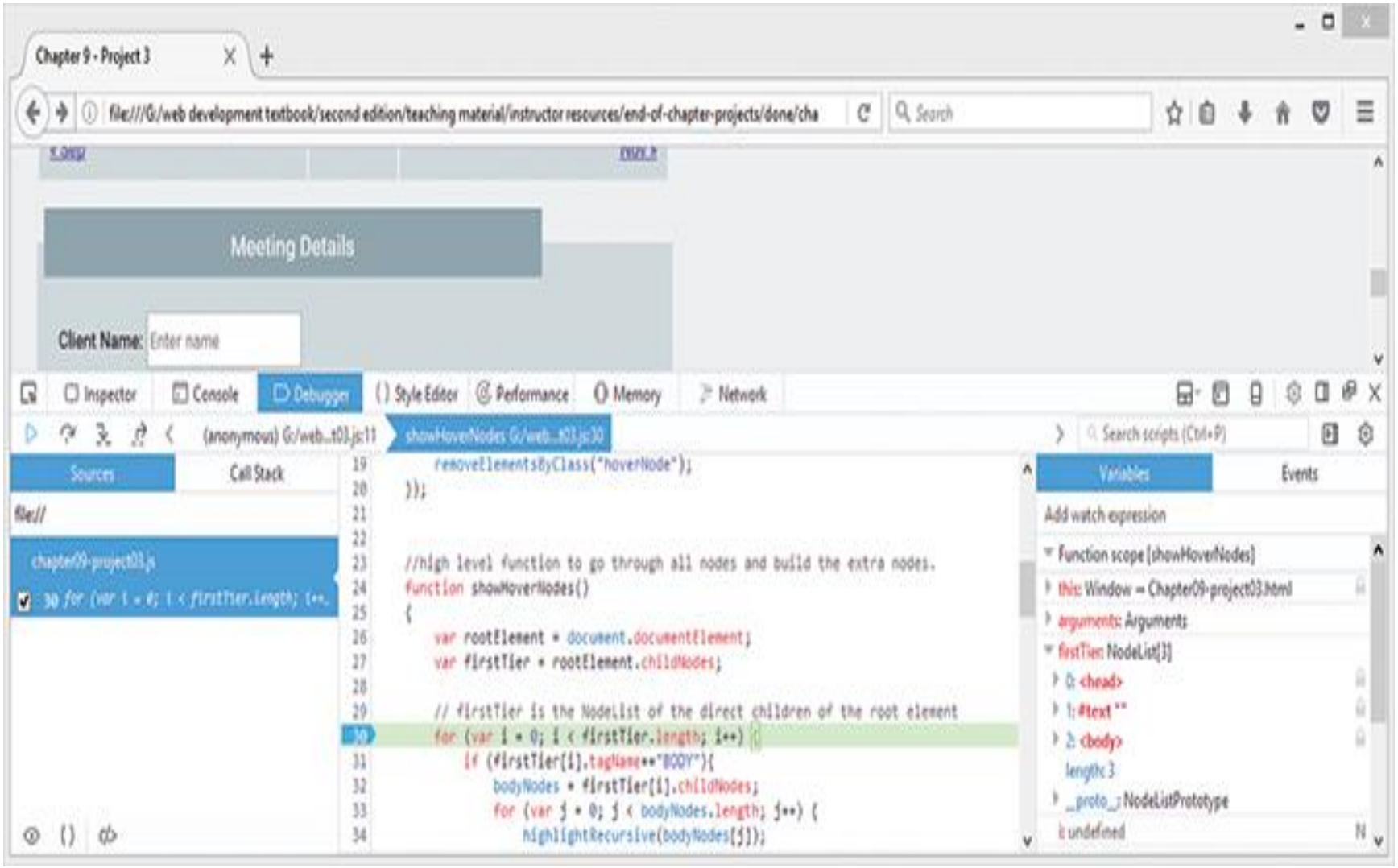
```
var h1 = document.getElementsByTagName("H1")[0];    // Get the first <h1> element in the document
var att = document.createAttribute("class");        // Create a "class" attribute
att.value = "democlass";                            // Set the value of the class attribute
h1.setAttributeNode(att);                           // Add the class attribute to <h1>
```

Before creating the attribute:

Hello World

After inserting the attribute:

Debugging within the FireFox browser



Chapter 9

file:///C:/Users/Chris/Desktop/686/FunWebDev2/0134481747_cl-268317/code-listings/chapter09/example.html

F12 Elements Console Debugger Network Performance Memory Emulation

Find (Ctrl+F)

example.html

Type to filter

- Local Storage
- Session Storage
- Cookies
- Cache
- C:
  - C:/Users/Chris/Desktop/686/FunWeb
    - example.html
    - example.js
    - styles.css
- code.jquery.com
  - jquery-3.2.1.min.js

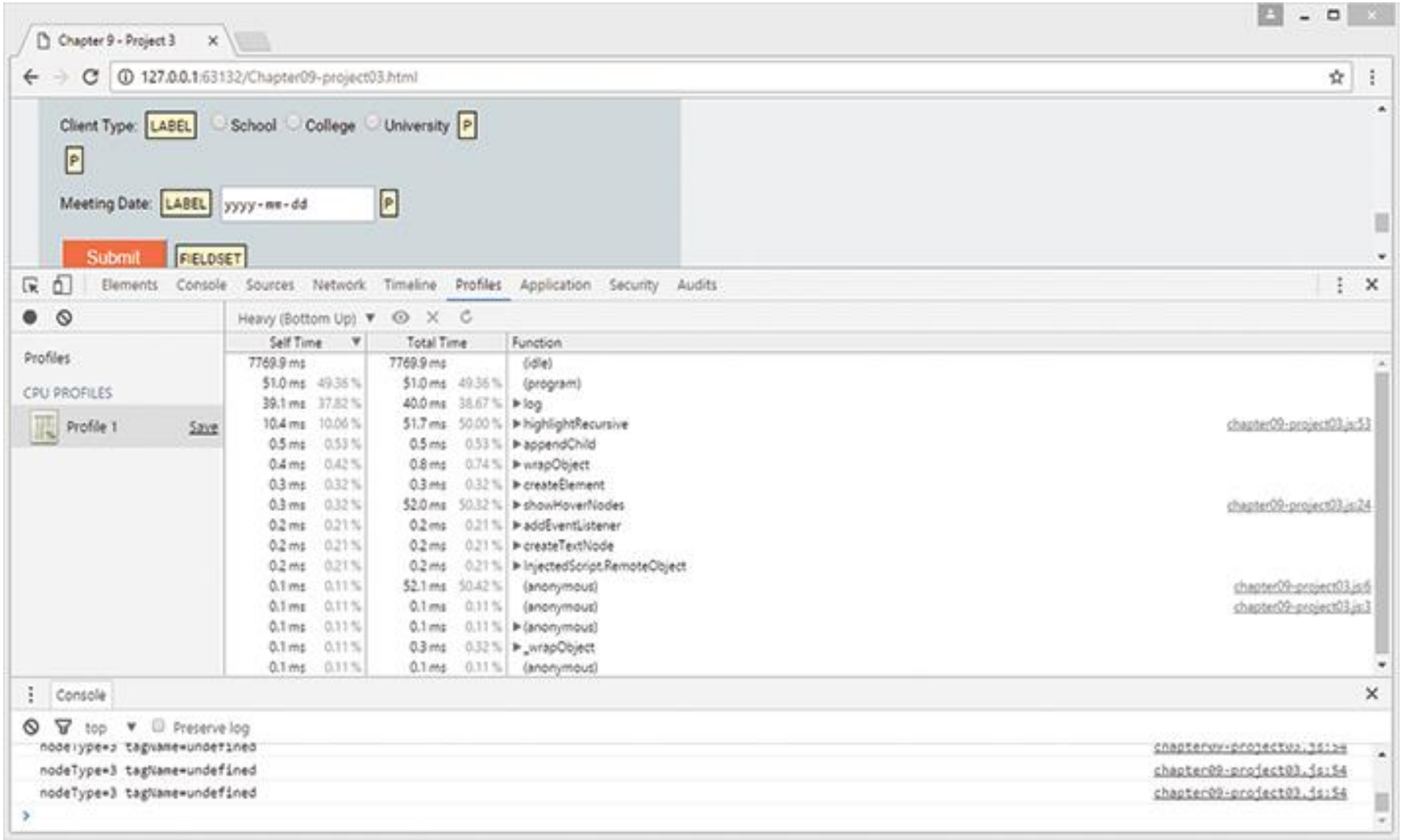
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4     <meta charset="utf-8">
5     <title>Chapter 9</title>
6     <script src="http://code.jquery.com/jquery-3.2.1.min.js" ></script>
7     <script src="example.js" ></script>
8     <link rel="stylesheet" href="styles.css">
9  </head>
10 <body>
11
12 <div id="main">
13    <img src="images/8711645510.jpg" id="mainImage" alt="main image" />
14    <p id="content">
15    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec convallis ullamcorper neque eget volutpat. Sed mi lectus, accumsan ac leo in, condimentum sollicitudin magna. Quisque vel ante c
      blandit nibh et euismod imperdiet. Pellentesque ligula mi, facilisis id tempor quis, dapibus eget diam. Vestibulum eget urna odio. Nullam in nulla felis. Donec rutrum efficitur vestibulum.
16    </p>
17    <button id="testButton">Hide</button>
18 </div>
19
20 </body>
21 </html>
```

Watches

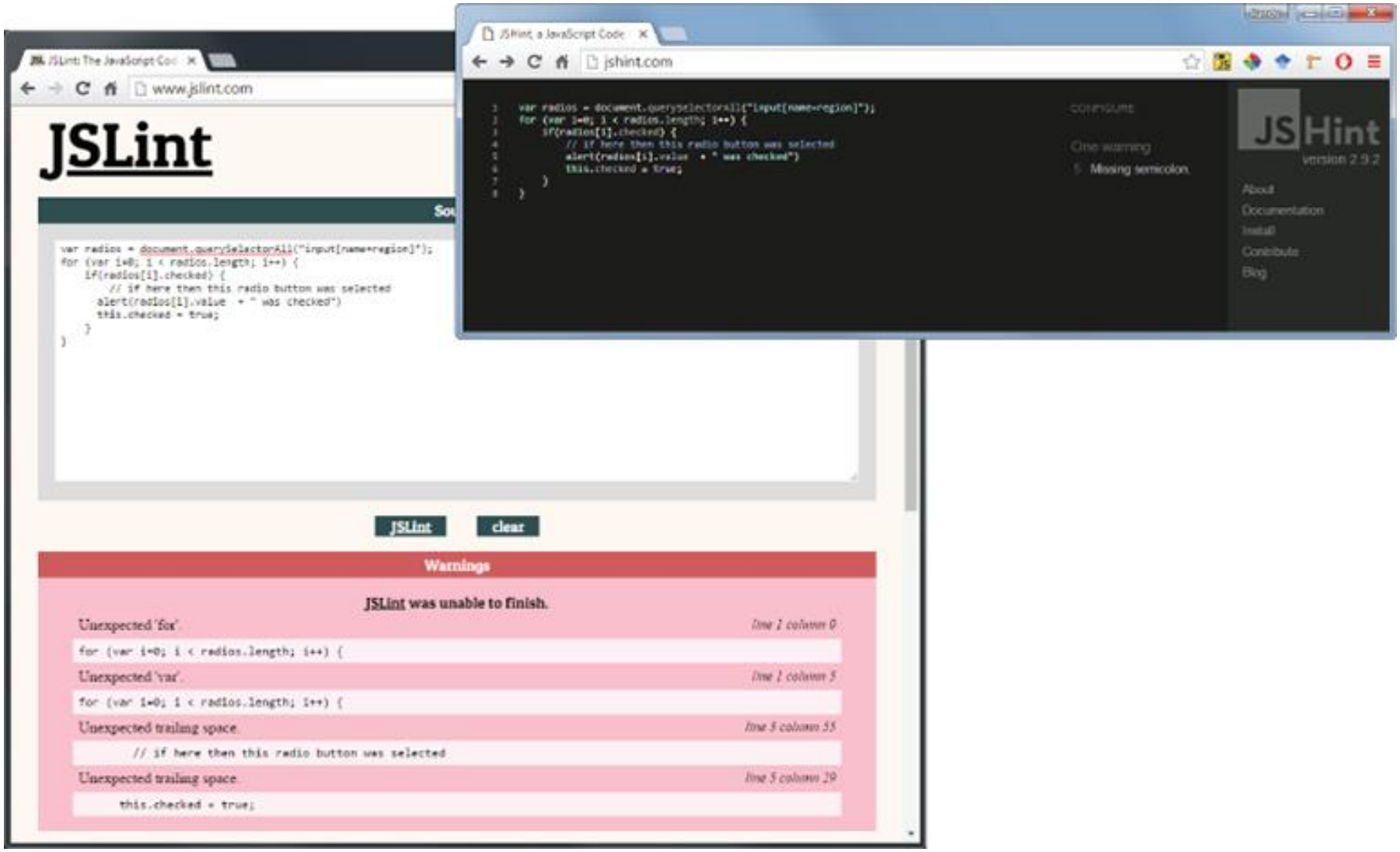
Add watch

Call stack Breakpoints DOM breakpoints

Performance Evaluation in Chrome



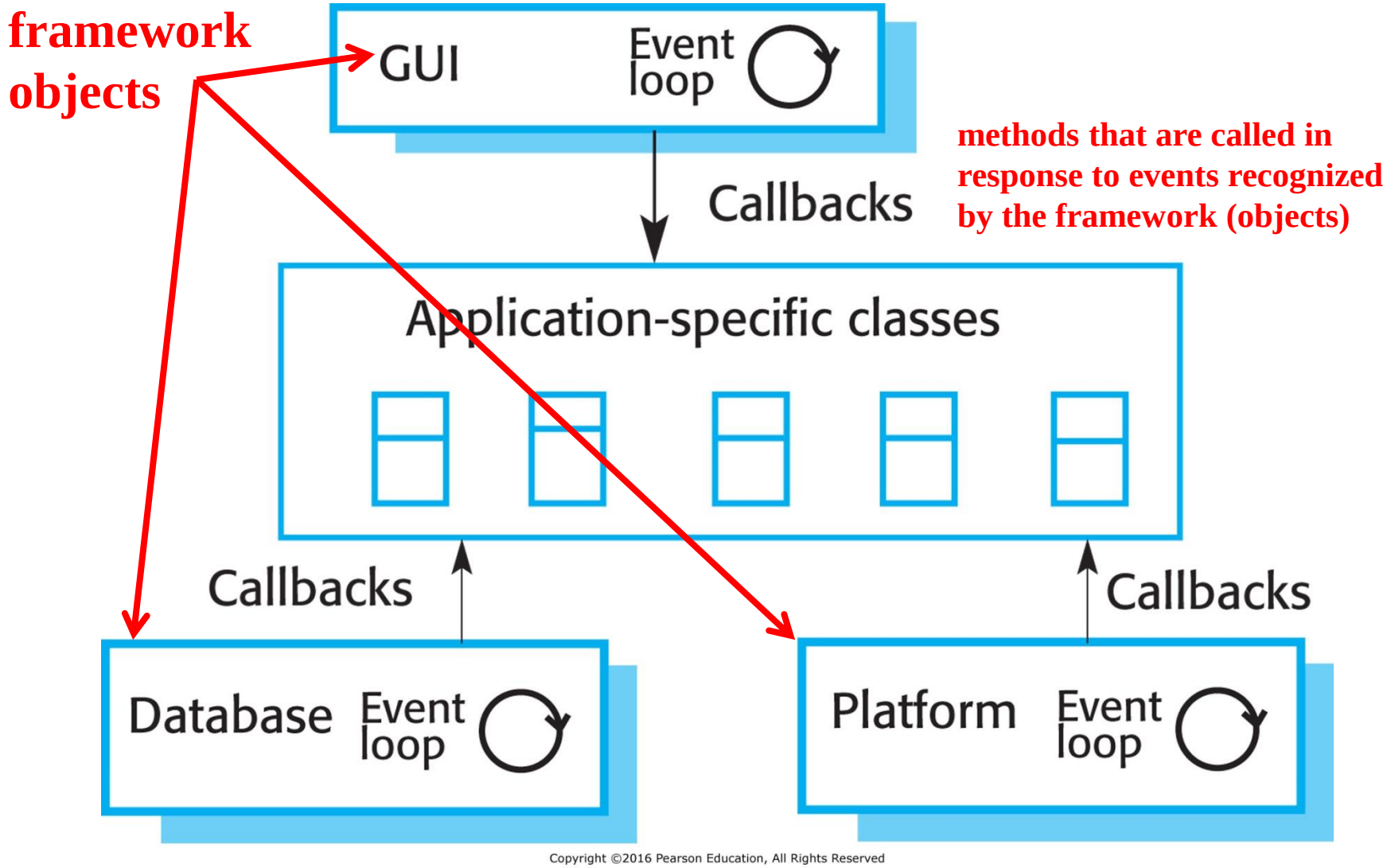
Linters


JSLint
www.jslint.com
var radios = document.querySelectorAll("input[name=region]");
for (var i=0; i < radios.length; i++) {
if(radios[i].checked) {
// if here then this radio button was selected
alert(radios[i].value + " was checked")
this.checked = true;
}
}
JSLint
clear
Warnings
JSLint was unable to finish.
Unexpected 'for'
line 1 column 0
for (var i=0; i < radios.length; i++) {
Unexpected 'var'.
line 1 column 5
for (var i=0; i < radios.length; i++) {
Unexpected trailing space.
line 5 column 55
// if here then this radio button was selected
Unexpected trailing space.
line 5 column 29
this.checked = true;
JSHint, a JavaScript Code
jshint.com
var radios = document.querySelectorAll("input[name=region]");
for (var i=0; i < radios.length; i++) {
if(radios[i].checked) {
// if here then this radio button was selected
alert(radios[i].value + " was checked")
this.checked = true;
}
}
configure
One warning
1 Missing semicolon.
JSHint
version 2.9.2
About
Documentation
Install
Contribute
Blog

Events

- JavaScript **event** is an action that can be detected by JavaScript
  - Many of them are initiated by user actions
  - Some are generated by the browser itself.
- We say that an event is **triggered** and then it is **handled** by JavaScript functions
- There are three main approaches to handling events in JavaScript:
  - Using inline hooks to handlers embedded within markup elements,
  - Attaching callbacks to event properties
  - Using event listeners

Inversion of Control in Frameworks



Schmidt, D. C., A. Gokhale, and B. Natarajan. 2004. "Leveraging Application Frameworks." *ACM Queue* 2 (5 (July/August)): 66–75. doi:10.1145/1016998.1017005.

Inline Hooks

HTML document using the inline hooks

```
...
 <script type="text/javascript" src="inline.js"></script>
...
<form name='mainForm' onsubmit="validate(this);">
      <input name="name" type="text"
          onchange="check(this);"
          onfocus="highlight(this, true);"
          onblur="highlight(this, false);">
      <input name="email" type="text"
          onchange="check(this);"
          onfocus="highlight(this, true);"
          onblur="highlight(this, false);">
      <input type="submit"
          onclick="function (e) {
                      ...
                  }">
...
```

Notice that you can define an entire event handling function within the markup. This is NOT recommended!

inline.js

```
function validate(node) {
  ...
}
function check(node) {
  ...
}
function highlight(node) {
  ...
}
```

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an onclick attribute (with code), is added to a button element:

Example

COLOR PICKER

HOW TO
Tabs
Dropdowns
Accordions
Convert Weights
Animated Buttons
Side Navigation
Top Navigation

HTML Reference

HTML by Alphabet
HTML by Category
HTML Attributes
HTML Global Attributes
HTML Events
HTML Colors
HTML Canvas
HTML Audio/Video
HTML Character Sets
HTML Doctypes
HTML URL Encode
HTML Language Codes
HTML Country Codes
HTML Country Codes
HTTP Messages
HTTP Methods
PX to EM Converter
Keyboard Shortcuts

HTML Tags

<!-->
<!DOCTYPE>
<a>

All HTML Attributes

Attribute	Belongs to	Description
accept	<input>	Specifies the types of files that the server accepts (only for type="file")
accept-charset	<form>	Specifies the character encodings that are to be used for the form submission
accesskey	Global Attributes	Specifies a shortcut key to activate/focus an element
action	<form>	Specifies where to send the form-data when a form is submitted
align	Not supported in HTML 5.	Specifies the alignment according to surrounding elements. Use CSS instead
alt	<area>, <img>, <input>	Specifies an alternate text when the original element fails to display
async	<script>	Specifies that the script is executed asynchronously (only for external scripts)
autocomplete	<form>, <input>	Specifies whether the <form> or the <input> element should have autocomplete enabled

Event Properties

- The problem with inline hooks is that it does not separate content from behavior, and as a consequence is much more difficult to maintain. (tightly-coupled)
- The event property approach shown below is supported by all browsers and allows us to separate the specification of an event handler from the markup.

```
var myButton = document.getElementById('example');
myButton.onclick = alert('some message');
```

  - The first line creates a temporary variable for the HTML element that will trigger the event.
  - The second line attaches the button element's **onclick** event to the event handler (the alert() function in this case).

Event Listeners

- All modern browsers (i.e., from Internet Explorer 9 forward) support the event listener approach to handling events and it is the preferred approach since it is possible to assign multiple handlers to a given event.

```
var myButton = document.getElementById('example');
myButton.addEventListener('click', alert('some message'));
myButton.addEventListener('mouseout', funcName);
```

Table 9.7 Mouse Events in JavaScript

Event	Description
`click`	The mouse was clicked on an element
`dblclick`	The mouse was double clicked on an element
`mousedown`	The mouse was pressed down over an element
`mouseup`	The mouse was released over an element
`mouseover`	The mouse was moved (not clicked) over an element
`mouseout`	The mouse was moved off of an element
`mousemove`	The mouse was moved while over an element

Table 9.8 Keyboard Events in JavaScript

Event	Description
`keydown`	The user is pressing a key (this happens first)
`keypress`	The user presses a key (this happens after keydown)
`keyup`	The user releases a key that was down (this happens last)

Table 9.9 Form Events in JavaScript

Event	Description
`blur`	Triggered when a form element has lost focus (that is, control has moved to a different element), perhaps due to a click or Tab key press.
`change`	Some `<input>`, `<textarea>` or `<select>` field had their value change. This could mean the user typed something, or selected a new choice.
`focus`	Complementing the `blur` event, this is triggered when an element gets focus (the user clicks in the field or tabs to it).
`reset`	HTML forms have the ability to be reset. This event is triggered when that happens.
`select`	When the users selects some text. This is often used to try and prevent copy/paste.
`submit`	When the form is submitted this event is triggered. We can do some prevalidation of the form in JavaScript before sending the data on to the server.

Event Listeners

- If you want to do something more elaborate than just **alert()** when an event is triggered, the behavior would have to be encapsulated within a function, as shown below:

```
function displayTheDate() {
    var d = new Date();
    alert (“You clicked this on ”+ d.toString());
}

var element = document.getElementById('example’);
element.addEventListener('click', displayTheDate);
```

Event Listeners

- Function expressions are full-fledged objects that can be passed as a parameter to another function. Such a passed-in function is said to be a **callback function** and are commonly used in event-driven JavaScript programming.
- Also, we are interested in reducing the number of global identifiers when programming with JavaScript. As a result, we often will make use of **anonymous functions** as event handlers:

```
var element = document.getElementById('example’);
// now we are using an anonymous function as the handler
element.addEventListener('click', function() {
    var d = new Date();
    alert(“You clicked this on ”+ d.toString());
});
```

Event Objects

- When an event is triggered, the browser will construct an event object that contains information about the event.

- Your event handlers can access this event object simply by including it as a parameter to the callback function (by convention, this event object parameter is often named **e**).

Listing 9.7 Using the event object parameter

```
var div = document.querySelector('div#example');
div.addEventListener('click', function(e) {
    // find out where the user clicked
    var x =  e.clientX;
    var y =  e.clientY;
    // output the information for debugging purposes
    console.log(e.type + ' event triggered by ' +
e.target);
    console.log(' at location ' + x + ' ' + y);
    // …
});
```

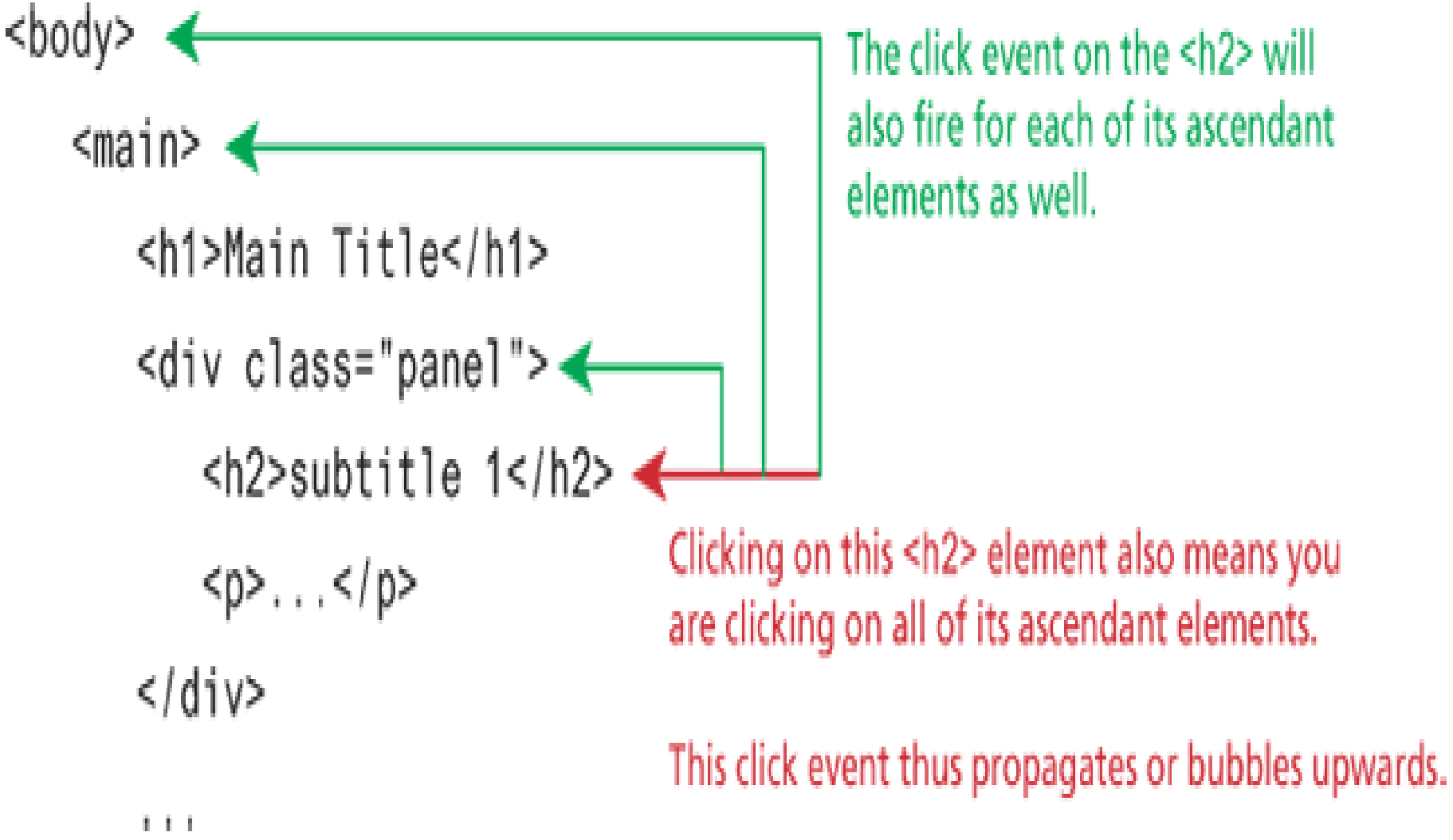
Table 9.6 Common Properties and Methods of the Event Object

Event	Description
`bubbles`	Indicates whether the event bubbles up through the DOM (see Dive Deeper section)
`cancelable`	Indicates whether the event can be cancelled
`target`	The object that generated (or dispatched) the event
`type`	The type of the event (see **Section 9.4** below)

Bubbles

- The bubbles property is a Boolean value.
- If an event's bubbles property is set to true then there must be an event handler in place to handle the event, or it will bubble up to its parent and trigger an event handler there.
- If the parent has no handler it continues to bubble up until it hits the document root, and then it goes away, unhandled.

Event Bubbling



Cancelable

- The Cancelable property is also a Boolean value that indicates whether or not the event can be cancelled.
- If an event is cancellable then the default action associated with it can be cancelled.
- A common example is a user clicking on a link.
- The default action is to follow the link and load the new page.

preventDefault

- A **cancelable** default action for an event can be stopped using the preventDefault() method as shown below:

```
function submitButtonClicked(e) {
    // prevent the submit action
    e.preventDefault();
    // now do other amazing things
    // …
}
```

- This is a common practice when you want to send data asynchronously, since the default action of a form submit click is to post to a new URL, which causes the browser to refresh the entire page.

Event Types

Mouse Events

- Mouse events are defined to capture a range of interactions driven by the mouse.
- These can be further categorized as **mouse click** and **mouse move** events.
- Table 9.7 (next slide) lists the possible events one can listen for from the mouse.
- Interestingly, many mouse events can be sent at a time. The user could be moving the mouse off of one and onto another in the same moment, triggering **mouseon** and **mouseout** events as well as the **mousemove** event. The Cancelable and Bubbles properties can be used to handle these complexities.

Table 9.7 Mouse Events in JavaScript

Event	Description
`click`	The mouse was clicked on an element
`dblclick`	The mouse was double clicked on an element
`mousedown`	The mouse was pressed down over an element
`mouseup`	The mouse was released over an element
`mouseover`	The mouse was moved (not clicked) over an element
`mouseout`	The mouse was moved off of an element
`mousemove`	The mouse was moved while over an element

Keyboard Events

- **Keyboard events** are often overlooked by novice web developers, but are important tools for power users.

- **Table 9.8** (next slide) lists the possible keyboard events.

Table 9.8 Keyboard Events in JavaScript

Event	Description
`keydown`	The user is pressing a key (this happens first)
`keypress`	The user presses a key (this happens after keydown)
`keyup`	The user releases a key that was down (this happens last)

Keyboard Events

- These events are most useful within input fields. (See next slide)

```
<input type="text" id="key">
```

We could listen to key press events for this input box and echo each pressed key back to the user as shown in **Listing 9.9**.

Listing 9.9 Listener that hears and alerts key presses

```
document.getElementById("key").addEventListener("keydown",
  function (e) {
     var keyPressed=e.keyCode;
     // get the raw key code
     var character=String.fromCharCode(keyPressed);
     // convert to string
     alert("Key " + character + " was pressed");
});
```

Touch Events

- Touch events are a new category of events that can be triggered by devices with **touch screens**. The different events (e.g., touchstart, touchmove, and touchend) are analogous to some of the mouse events (mousedown, mousemove, and mouseup).
- Unfortunately, at the time of writing, touch events are only available by default in Chrome and iOS Safari.
- The user needs to enable touch events in Edge and FireFox. Edge does support Pointer (mouse + touch + pen) events, which is a new standard specification However, at the time of writing, pointer events are only supported in Edge.

Form Events

- Forms are the main means by which user input is collected and transmitted to the server.

- Table 9.9 (Next slide) lists the different form events.

Table 9.9 Form Events in JavaScript

Event	Description
`blur`	Triggered when a form element has lost focus (that is, control has moved to a different element), perhaps due to a click or Tab key press.
`change`	Some `<input>`, `<textarea>` or `<select>` field had their value change. This could mean the user typed something, or selected a new choice.
`focus`	Complementing the `blur` event, this is triggered when an element gets focus (the user clicks in the field or tabs to it).
`reset`	HTML forms have the ability to be reset. This event is triggered when that happens.
`select`	When the users selects some text. This is often used to try and prevent copy/paste.
`submit`	When the form is submitted this event is triggered. We can do some prevalidation of the form in JavaScript before sending the data on to the server.

Form Events

- The events triggered by forms allow us to do some timely processing in response to user input.
- The most common JavaScript listener for forms is the **submit** event.
- In Listing 9.10 (next slide), we listen for that event on a form with id **loginForm**.
  - If the password field (with id pw) is blank, we prevent submitting to the server using preventDefault() and alert the user. Otherwise we do nothing, which allows the default event to happen (submitting the form).

Listing 9.10 Catching the submit event and validating a password to not be blank

```
document.getElementById
("loginForm").addEventListener('submit',
   function(e) {
    var pass = document.getElementById
("pw").value;
    if (pass=="")  {
         alert ("enter a password");
         e.preventDefault();
    }
});
```

Frame Events

- Frame events (see Table 9.10, next slide) are the events related to the browser frame that contains your web page.
- The most important event is the **load** event, which tells us an object is loaded and therefore can be manipulated via the DOM.
- In fact, every nontrivial event listener you write requires that the HTML be fully loaded.

Table 9.10 Frame Events in JavaScript

Event	Description
`abort`	An object was stopped from loading
`error`	An object or image did not properly load
`load`	When a document or object has been loaded
`resize`	The document view was resized
`scroll`	The document view was scrolled
`unload`	The document has unloaded

Frame Events

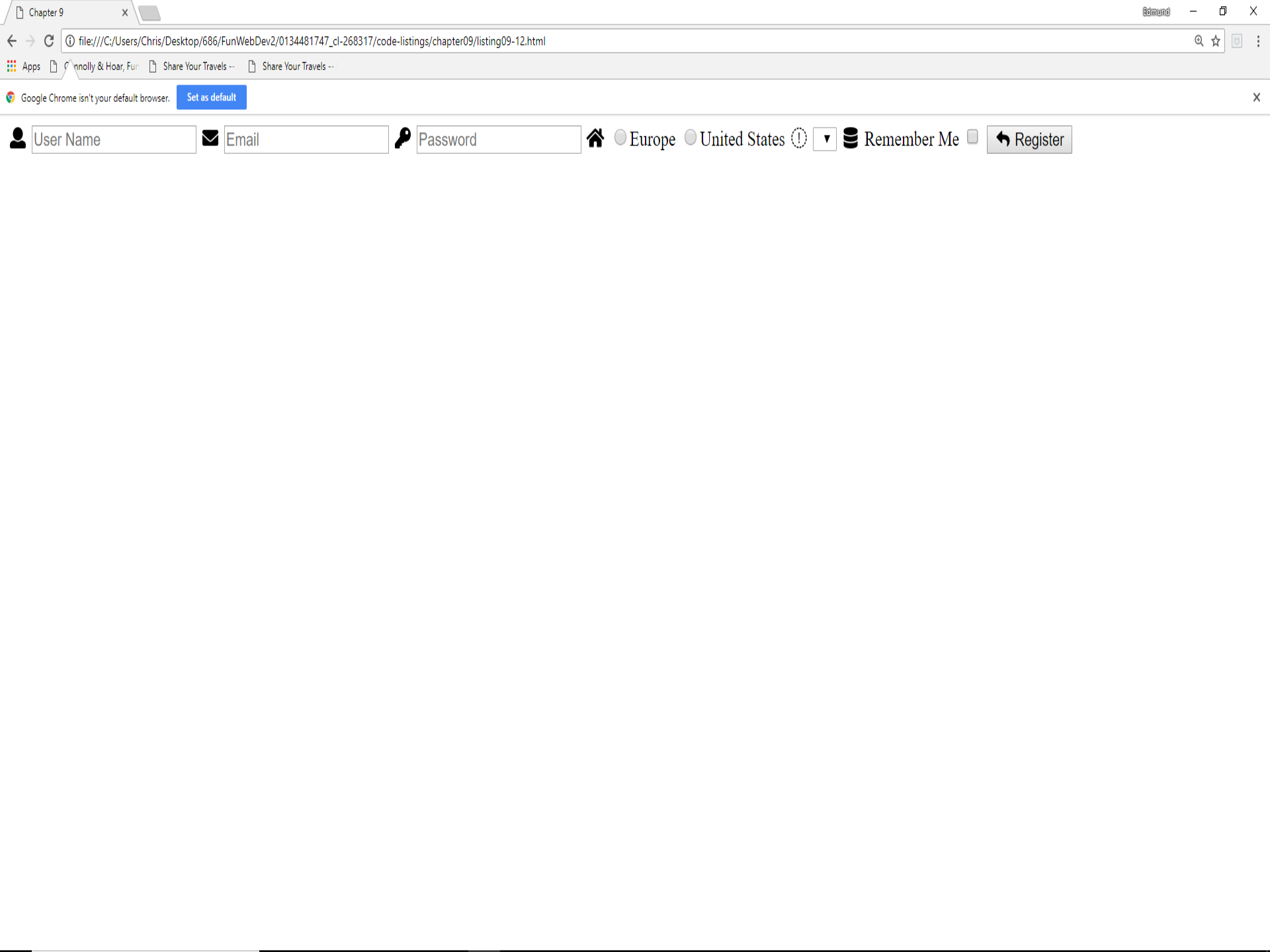
- As mentioned earlier, a problem can occur if the JavaScript tries to programmatically reference a DOM element that has not yet been loaded.
- If the code attempts to set up a listener on this not-yet-loaded element then an error will be triggered.
- For this reason, it is common practice to use the **load** event of the window object to trigger the execution of the rest of the page's scripts, as shown below:

```
window.addEventListener("load", function() {
  // the DOM can be safely manipulated within this function
  // …
});
```

HTML Forms Revisited

Forms Revisited

- We learned how to create HTML forms for data entry last week. I also mentioned on Monday that user input to the HTML forms should be validated.
- This week, we will take a look at how we can use JavaScript for this task.
- When working with forms in JavaScript, we are typically interested in three types of events:
  1. Movement between elements
  2. Data being changed within a form element
  3. The final submission of the form.

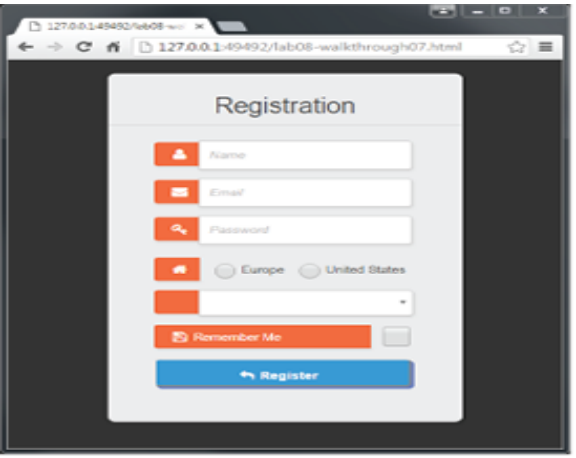

Chapter 9
Edmund
file:///C:/Users/Chris/Desktop/686/FunWebDev2/0134481747_cl-268317/code-listings/chapter09/listing09-12.html
Apps
Share Your Travels --
Share Your Travels --
Google Chrome isn't your default browser.
Set as default
User Name
Email
Password
Europe
United States
Remember Me
Register

Listing 9.12 A basic HTML form

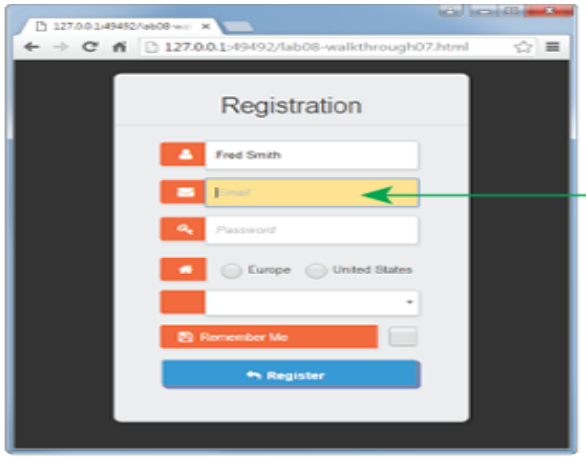
```
<form method="post" action="login.php" id="loginForm">
    <label class="icon" for="username"><i class="fa fa-user fa-fw">       </i></label>
    <input type="text" name="username" id="username" placeholder="User       Name" />
    <label class="icon" for="email"><i class="fa fa-envelope fa-fw">       </i></label>
    <input type="text" name="email" id="email" placeholder="Email" />

    <label class="icon" for="pass"><i class="fa fa-key fa-fw"></i>       </label>
    <input type="password" name="pass" id="pass" placeholder="Password" />
    <label class="icon" for="region"><i class="fa fa-home fa-fw"></i>       </label>
    <input type="radio" name="region" id="region" value="Europe"
       class="bigRadio"><span>Europe</span>
    <input type="radio" name="region" id="region" value="United States"
       class="bigRadio"><span>United States</span>
    <label class="icon" for="options">
       <i class="fa fa-fw" id="optLabel"></i></label>
    <select name="options" id="options"></select>
    <label class="icon" id="long" for="save">
       <i class="fa fa-database fa-fw"></i> Remember Me</label>
    <input type="checkbox" name="save" id="save" class="bigCheckBox" />
    <button type="submit" ><i class="fa fa-reply fa-fw"></i> Register       </button>
</form>
<div id="errors" class="hidden"></div>
```

Responding to Form Movement Events



How form appears when no controls have the focus



When a control has the focus, then change its background color

```
// This function is going to get called every time the focus or blur events are
// triggered in one of our form's input elements.
function setBackground(e) {
    if (e.type == "focus") {
        e.target.style.backgroundColor = "#FFE393";
    }
    else if (e.type == "blur") {
        e.target.style.backgroundColor = "white";
    }
}

// set up the event listeners only after the DOM is loaded
window.addEventListener("load", function() {
    var cssSelector = "input[type=text],input[type=password]";
    var fields = document.querySelectorAll(cssSelector);
    for (i=0; i<fields.length; i++)  {
        fields[i].addEventListener("focus", setBackground);
        fields[i].addEventListener("blur", setBackground);
    }
});
```

Here we use the `style` property instead of the `classList` property because of specificity conflicts (i.e., attribute selectors override class selectors).

Selects the fields that will change.

Assigns the `setBackground()` function to change the background color of the control depending upon whether it has the focus.

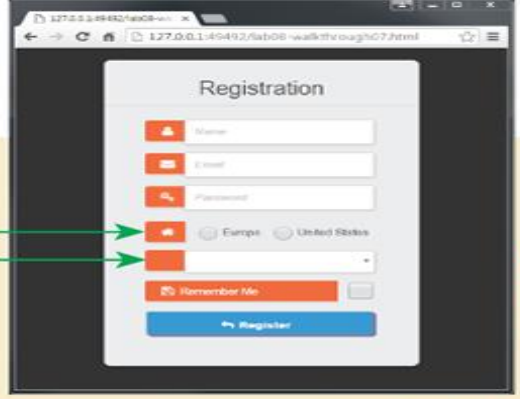
Table 9.8 Keyboard Events in JavaScript

Event	Description
`keydown`	The user is pressing a key (this happens first)
`keypress`	The user presses a key (this happens after keydown)
`keyup`	The user releases a key that was down (this happens last)

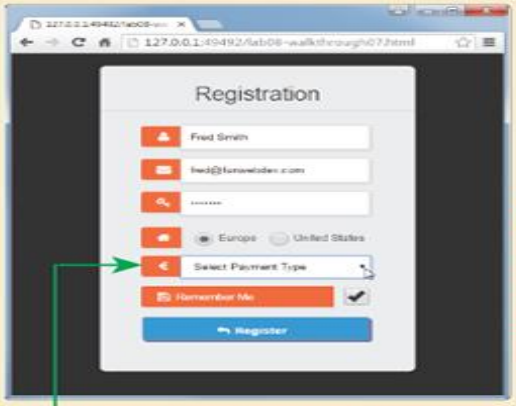
Table 9.9 Form Events in JavaScript

Event	Description
`blur`	Triggered when a form element has lost focus (that is, control has moved to a different element), perhaps due to a click or Tab key press.
`change`	Some `<input>`, `<textarea>` or `<select>` field had their value change. This could mean the user typed something, or selected a new choice.
`focus`	Complementing the `blur` event, this is triggered when an element gets focus (the user clicks in the field or tabs to it).
`reset`	HTML forms have the ability to be reset. This event is triggered when that happens.
`select`	When the users selects some text. This is often used to try and prevent copy/paste.
`submit`	When the form is submitted this event is triggered. We can do some prevalidation of the form in JavaScript before sending the data on to the server.

```
// depending on the state of the region radio buttons
// change the options of the select list
var label = document.getElementById("payLabel");
var select = document.getElementById("payment");
select.disabled = true;
var radios = document.querySelectorAll("input[name=region]");
// listen to each radio button
for (var i=0; i < radios.length; i++) {
    // whenever a radio button changes, modify the select
    // list as well as the label beside it
    radios[i].addEventListener("change",
        function (e) {
            select.disabled = false;
            select.innerHTML = "";
            addOption(select, "Select Payment Type" , "0");
            var choice = e.target.value;
            if (choice == "United States") {
                // display the dollar symbol
                label.classList.remove("fa-euro");
                label.classList.add("fa-dollar");
                addOption(select, "American Express" , "1");
                addOption(select, "Mastercard" , "2");
                addOption(select, "Visa" , "3");
            }
            else if (choice == "Europe") {
                // display the euro symbol
                label.classList.remove("fa-dollar");
                label.classList.add("fa-euro");
                addOption(select, "Bitcoin" , "4");
                addOption(select, "PayPal" , "5");
            }
        }
    );
}
function addOption(select, optionText, optionValue) {
    var opt = document.createElement('option');
    opt.appendChild( document.createTextNode(optionText) );
    opt.value = optionValue;
    select.appendChild(opt);
}
```

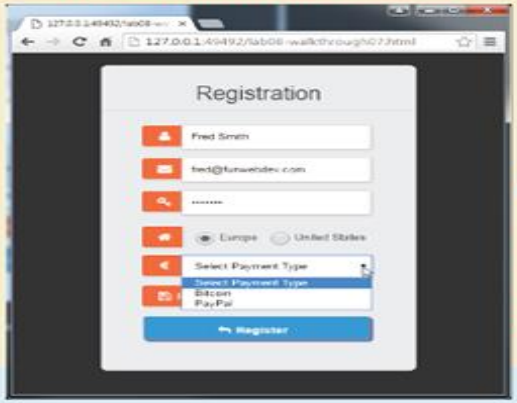


1 Initially the <select> list is disabled.



2 But when user changes a radio button, enable the select list, ...

3 ... change the icon in the label based on the radio button, ...



4 ... and then populate the list with appropriate option values,

Use the DOM functions from Section 9.2 to create a new <option> element, populate it with the appropriate text, and then add it to the <select> element.

Table 9.8 Keyboard Events in JavaScript

Event	Description
`keydown`	The user is pressing a key (this happens first)
`keypress`	The user presses a key (this happens after keydown)
`keyup`	The user releases a key that was down (this happens last)

Table 9.9 Form Events in JavaScript

Event	Description
`blur`	Triggered when a form element has lost focus (that is, control has moved to a different element), perhaps due to a click or Tab key press.
`change`	Some `<input>`, `<textarea>` or `<select>` field had their value change. This could mean the user typed something, or selected a new choice.
`focus`	Complementing the `blur` event, this is triggered when an element gets focus (the user clicks in the field or tabs to it).
`reset`	HTML forms have the ability to be reset. This event is triggered when that happens.
`select`	When the users selects some text. This is often used to try and prevent copy/paste.
`submit`	When the form is submitted this event is triggered. We can do some prevalidation of the form in JavaScript before sending the data on to the server.

Empty Field Validation

A common application of a client-side validation is to make sure the user entered something into a field (or selected a value). There's certainly no point sending a request to log in if the username was left blank, so why not prevent the request from working? The way to check for an empty field in JavaScript is to compare a value to both null and the empty string (“”), as shown in **Listing 9.13**.

Listing 9.13 A simple validation script to check for empty fields

```
document.getElementById("loginForm").addEventListener("submit",
  function(e) {
    var fieldValue = document.getElementById("username").value;
    if (fieldValue == null  ||  fieldValue == "") {
        // the field was empty. Stop form submission
        e.preventDefault();
        // Now tell the user something went wrong
        alert("you must enter a username");
    }
});
```

Listing 9.15 A function to test for a numeric value

```
function isNumeric(n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
}
```

More involved examples to validate email, phone numbers, or social security numbers would include checking for blank fields and making use of `isNumeric` and regular expressions as illustrated in **Chapter 15**.

9.5.4 Submitting Forms

Submitting a form using JavaScript requires having a node variable for the form element. Once the variable, say, `formExample` is acquired, one can simply call the `submit()` method:

```
var formExample = document.getElementById("loginForm");
formExample.submit();
```

This is often done in conjunction with calling `preventDefault()` on the `submit` event. This can be used to submit a form

JavaScript isNaN() Function

‹ JavaScript Global Functions

Example

Check whether a value is NaN:

```
isNaN(123) //false
isNaN(-1.23) //false
isNaN(5-2) //false
isNaN(0) //false
isNaN('123') //false
isNaN('Hello') //true
isNaN('2005/12/12') //true
isNaN('') //false
isNaN(true) //false
isNaN(undefined) //true
isNaN('NaN') //true
isNaN(NaN) //true
isNaN(0 / 0) //true
```

Try it Yourself »

JavaScript isFinite() Function

‹ JavaScript Global Functions

Example

Check whether a number is a finite, legal number:

```
var a = isFinite(123) + "<br>";
var b = isFinite(-1.23) + "<br>";
var c = isFinite(5-2) + "<br>";
var d = isFinite(0) + "<br>";
var e = isFinite("123") + "<br>";
var f = isFinite("Hello") + "<br>";
var g = isFinite("2005/12/12");

var res = a + b + c + d + e + f + g;
```

Try it Yourself »

Definition and Usage