Web Development: Custom for Liberty Mutual

Version 1.0.0 January 2017



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Legal Information

Copyright (C) Descriptor Systems 2001, 2017

All Rights Reserved

All trademarks are owned by their respective companies

You can contact Descriptor Systems at:
http://www.descriptor.com
jbarnum@descriptor.com

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Web Development: Custom for Liberty Mutual

- Introductions
- Objectives

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introductions

- Please introduce yourself:
  - Name:
  - HTML Background:
  - CSS Background:
  - JavaScript Background:
  - Your goal for the course:

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Description

Description: This class is designed for students that have experience with basic HTML concepts that wish to learn about HTML Version 5, Cascading Style Sheets Version 3, JavaScript and AngularJS.

Prerequisites: Basic HTML background is assumed.

Audience: Business analysts, developers, managers and other people interested in learning how to create responsive Web pages.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Objectives

After taking this course, you will be able to:

- Understand the new features in HTML5
- Understand the new features in CSS3
- Write responsive Web sites that adapt to different screen sizes
- Write JavaScript code and access the HTML DOM
- Use the jQuery JavaScript library
- Use the AngularJS JavaScript library

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda

Day 1

Introduction to HTML5
Other HTML5 Feature Overview
CSS Fundamentals
CSS Box Model and Layouts
Introduction to CSS3
Designing Responsive Web Applications
Introduction to JavaScript

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

```
Day 2
-------

JavaScript Data Types and Assignment
JavaScript Operators
JavaScript Flow Control
JavaScript Events
JavaScript Objects
JavaScript Arrays
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 3

JavaScript Functions
JavaScript window and document Objects
JavaScript and CSS
Introduction to JSON
Introduction to jQuery
Introduction to REST
SpringMVC REST

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 4

Introduction to AngularJS
Additional AngularJS Topics
The Angular $http Service
AngularJS Filters

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 5

AngularJS Directives
AngularJS Forms
Testing JavaScript and AngularJS

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Logistics

- Schedule
  - Start time
  - Lunch
  - End time
  - Breaks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to HTML5

- What's in HTML5?
- Browser Support

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introducing HTML5

- HTML5 is the latest release of the **Hyper Text Markup Language** specification for Web pages
- HTML5 focuses on Web **applications**, not just Web pages
- Browser support for HTML5 features is growing but not yet universal

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML Version History at the W3C

- HTML (and HTTP) were invented by Tim Berners-Lee in 1990
- HTML2 was released in 1995
- HTML3.2 became a W3C Recommendation in early 1997
- HTML4.0 became a W3C Recommendation in late 1997
- XHTML1.0 became a W3C Recommendation in 2000
- HTML5 is a work in progress with a Recommendation target date in 2014

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What's New in HTML5

- HTML5 extends earlier versions with features geared toward Web applications, rather than just pages:
    - **Semantic markup** - HTML elements such as <nav> that impart more meaning than simple
    - **New Form Features** - HTML5 includes new form widgets types such as **date** and **email** and ability to mark fields as required
    - **Microdata** - A standard way to add metadata to pages
    - **Web storage** - A client-side storage mechanism that's an alternative to cookies
    - **Audio, video, canvas, SVG** - HTML5 facilities to display rich media and graphics
    - **Web messaging** - Communication between the server and client and within the client
    - **Drag and drop** - Drag and drop items such as files onto the browser

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML5 Specifications

- There are actually two groups and multiple specification documents that comprise HTML5
- The standards groups:
  - The World Wide Web Consortium (W3C)
  - The Web Hypertext Application Technology Working Group (WHATWG)
- The specification documents:
  - The W3C has an HTML5 draft spec at http://www.w3.org/html/wg/drafts/html/CR and separate documents for microdata, etc.
  - The WHATWG has a complete “living standard” at http://developers.whatwg.org
- The “living standard” is geared toward Web developers, while the W3C specifications are more formal and geared toward implementors

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What About XHTML?

- HTML5 supports two **serializations** or syntaxes - the **HTML serialization** and the **XML serialization**
- As in the past, XHTML5 is geared toward strict well-formedness so that other tools (e.g. XSLT) can generate and process HTML content

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The DOCTYPE

- Unlike in previous versions of HTML, the HTML5 DOCTYPE is very simple

```
<!DOCTYPE html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Root Element

- The **root** element of an HTML5 document *should* specify the **language code**

```
1    <html lang="en">
2
3      . . .
4
5    </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Head Element

- The **<head>** element acts as a container for document **metadata**, including the page's title, scripts and style sheets
- HTML5 assumes JavaScript for the scripting language and CSS for the style-sheet format so you don't need to specify it

```
1    <head>
2      <meta charset="utf-8">
3      <title>My Page</title>
4      <link rel="stylesheet" href="mystyles.css">
5      <script src="app.js"></script>
6    </head>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Semantic Elements

- In older versions of HTML, it's common to use and other elements with CSS style classes to establish the structure of a document
- HTML5 provides new **semantic** elements that provide a standard way to provide structure

```
1    <!-- HTML4 -->
2    <div class="header">
3    . . .
4    </div>
5    <div class="navigation">
6    . . .
7    </div>
8    <div class="main">
9    . . .
10   </div>
11   <div class="footer">
12   . . .
13   </div>
```

```
1    <!-- HTML5 -->
2    <header>
3    . . .
4    </header>
5    <nav>
6    . . .
7    </nav>
8    <article>
9    . . .
10   </article>
11   <footer>
12   . . .
13   </footer>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

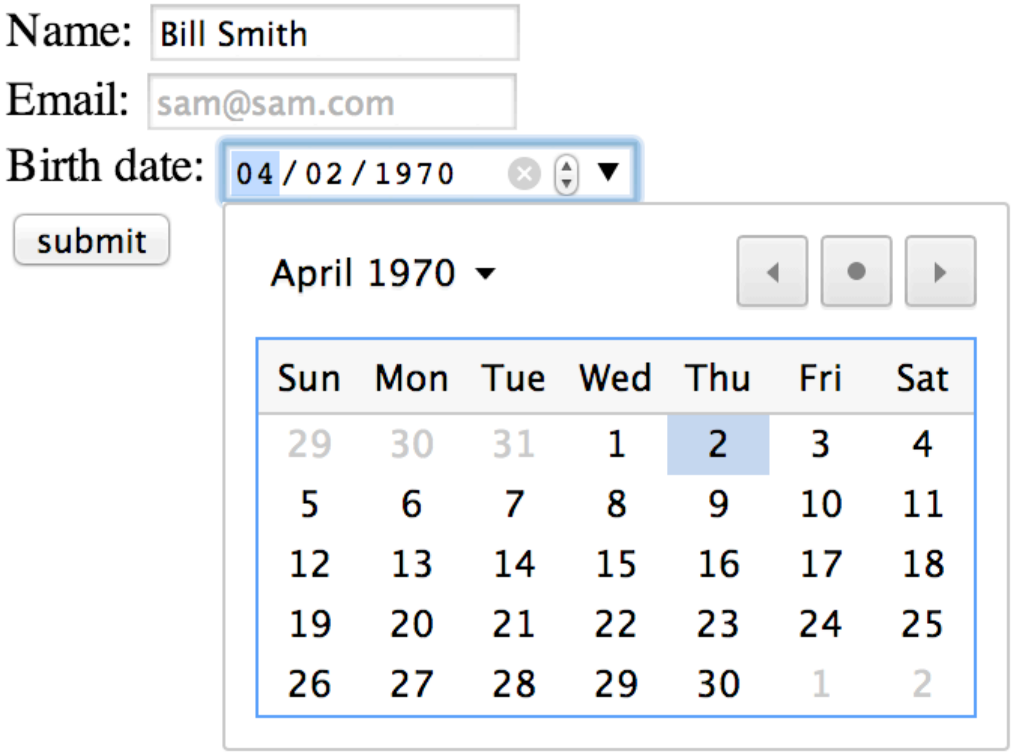
Semantic Elements, cont'd

- Semantic elements:
    - <header> <nav> <section> <article> <aside> <figure> <figcaption> <footer> <details> <summary> <mark> <time>
- The **section** element represents a generic section of a document or application
- The **article** element represents a complete, or self-contained, composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML5 Form Widgets

- HTML5 defines several new input-widget types as well as placeholder text and a required attribute



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML5 Form Widgets Example

```
1    <form>
2      <div>
3        Name: <input name="name" required autofocus>
4      </div>
5      <div>
6       Email: <input type="email" name="name" required
7          placeholder="sam@sam.com">
8      </div>
9      <div>
10       Birth date: <input type="date" name="date">
11     </div>
12     <div>
13       <input type="submit" value="submit">
14     </div>
15   </form>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

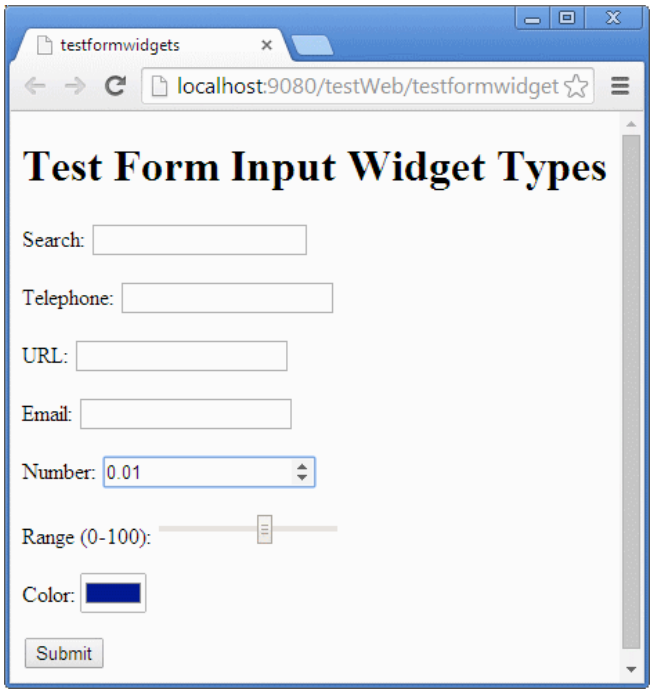
HTML5 Input Types

Type	Description
search	Like normal text, but browser can draw differently
tel	Browser can add spaces
url	Browsers might automatically escape special chars
email	Browsers might enforce syntax rules, e.g. require an '@'
datetime, date, time month, week, datetime-local	Browser can draw differently
number	Browser can draw differently and will enforce floating-point format
range	Browser can draw differently and enforce floating-point range
color	Browser can draw differently

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML5 Input Types, cont'd

inputtypes.html



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE HTML>
<html lang="en">
   <head>
     <title>testformwidgets</title>
   </head>
  <body>
    <h1>Test Form Input Widget Types</h1>
    <form>
      <p>
        Search: <input type="search">
      </p>
      <p>
        Telephone: <input type="tel">
      </p>
      <p>
        URL: <input type="url">
      </p>
      <p>
        Email: <input type="email">
      </p>
      <p>
        Number: <input type="number" min="0" step="0.01">
      </p>
      <p>
        Range (0-100): <input type="range" min="0"
           max="100" step="20" value="50">
      </p>
      <p>
        Color: <input type="color">
      </p>
      <p>
        <input type="submit">
      </p>
    </form>
  </body>
</html>
```

Other HTML5 Features

- Microdata
- Drag and drop
- Audio and video
- Canvas and SVG
- Web storage
- Web messaging: Server-side events, WebSockets
- XMLHTTPRequest object formal support

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other Related Technologies

- Though technically not part of HTML5, these are often associated with it:
    - CSS3
    - Geolocation
    - Web storage using the IndexedDB and File System APIs
    - WebGL
    - Node.js
    - jQuery

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Browser Support

- The technologies that comprise HTML5 have mixed support among popular browsers, but that support is rapidly growing
- There a few Web sites that track support:
    - http://caniuse.com
    - http://html5readiness.com
    - http://html5test.com
    - http://fmbip.com/litmus

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Supporting Older Browsers

- Until HTML5 becomes universal, applications using HTML5 will need to deal with back-level browsers
- You can use a JavaScript library to detect HTML5 features and/or provide back-level support
- Popular libraries include:
    - Remy Sharp's **HTML5 Shiv** (polyfill) library to provide back-level support for HTML5's new elements
    - **Modernizr** HTML-feature detection library (optionally includes the Shiv)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Modernizr

- Modernizr provides custom-generated versions that include only the features you need
- You can also choose between production and development versions
- To use, load the Modernizr script and add a *class* element to the document's <html> start tag

```
1    <html lang="en" class="no-js">
2      <head>
3        <title>My Page</title>
4        <meta charset="utf-8">
5        <script src="modernizr.js"></script>
6        <link rel="stylesheet" href="main.css">
7      </head>
8    . . .
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Page Source

```
1    <body>
2      <header>
3        <h1>My Header</h1>
4      </header>
5      <article>
6        <h1>My Article</h1>
7        <form>
8          <div>
9            Name: <input name="name" required autofocus>
10         </div>
11         <div>
12           Email: <input type="email" name="email" required
13            placeholder="sam@sam.com">
14        </div>
15        <div>
16         Birth date: <input type="date" name="date">
17       </div>
18       <div>
19         <input type="submit" value="submit">
20       </div>
21     </form>
22   </article>
23 </body>
```

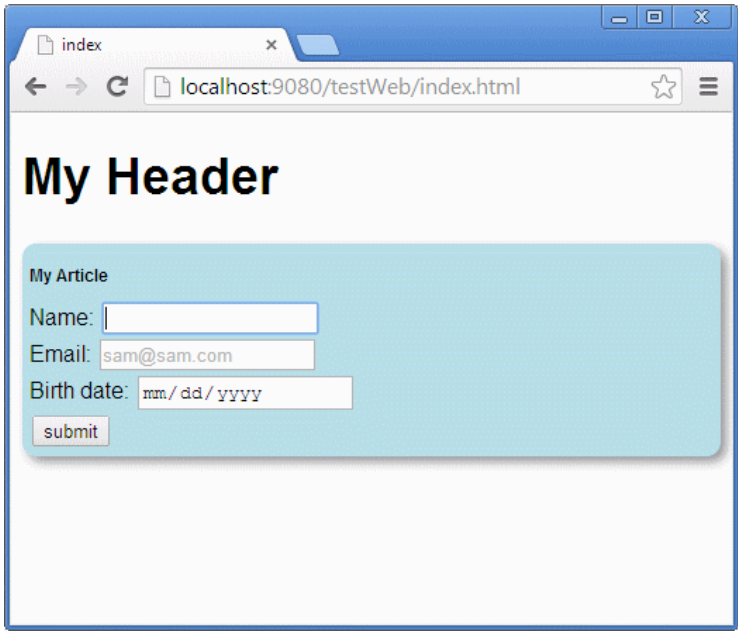

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Page CSS

```
1    body
2    {
3      font-size: 16px;
4      font-family: arial,helvetica,clean,sans-serif;
5    }
6    header h1 { font-size: 36px; margin-bottom: 25px; }
7    article
8    {
9      background: lightblue;
10     margin-bottom: 10px;
11     padding: 5px;
12     border-radius: 10px;
13     box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.5);
14   }
15   article h1 { font-size: 12px; }
```

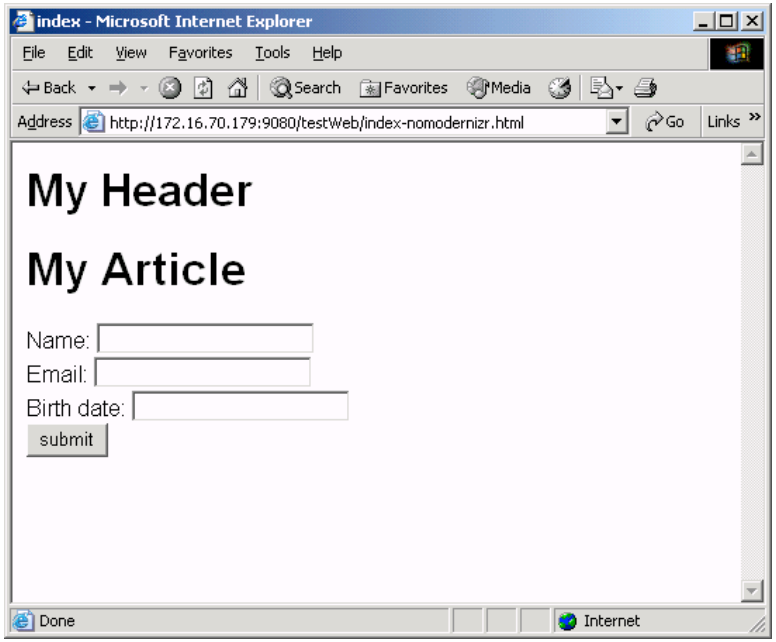

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Page in Modern Chrome



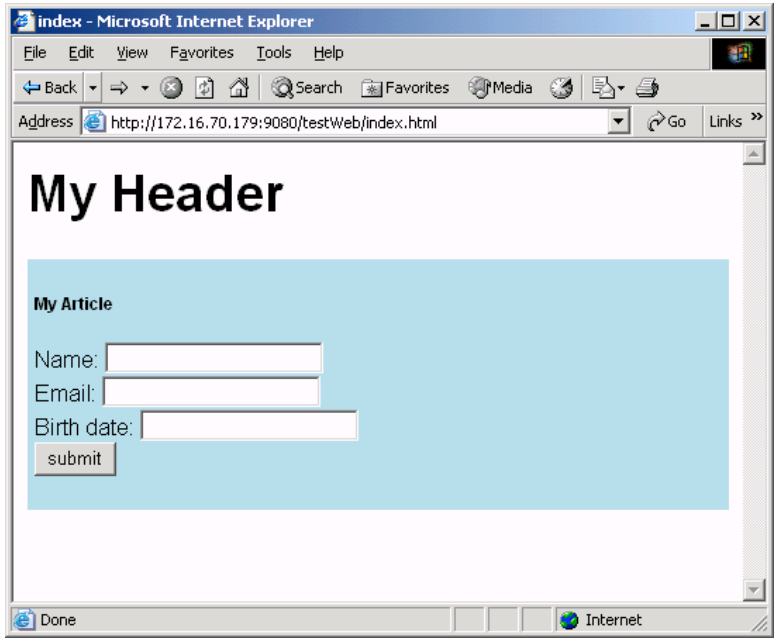
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Page in IE6, No Modernizr



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Page in IE6, With Modernizr



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- What the HTML5 technologies are
- About HTML5 specifications
- How to write simple HTML5 pages

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other HTML5 Feature Overview

- Microdata
- Web storage
- Video and audio
- Canvas
- Geolocation (not actually in HTML5 specs)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Microdata

- **Microdata** lets you add information about a web page to the page (metadata)
- This metadata is useful for search engines and other applications
- HTML5 provides a standard way to specify microdata

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Microdata Vocabularies

- A **vocabulary** is a set of name/values that describe "things" in Web pages, e.g. Person, Organization, Event and so forth
- There is a "standard" **vocabulary** that defines the HTML elements that you can use to define microdata that defines **itemscope**, **itemtype** and **itemprop**

test-microdata.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>About Penelope</title>
  </head>
  <body>
    <section itemscope itemtype="http://schema.org/Person">
      <h1>About Penelope</h1>
      <p>
        Penelope has been baking excellent pizza since 1945.
        She graduated with a PhD (Pizza-hurling Doctor) degree
        from the <span itemtype="alumniOf">
        Culinary Institute of the Midwest</span>
        in 1943.
      </p>
      <h2>Contact Info</h2>
      <div itemprop="name">Penelope Goodpie</div>
      <div itemscope itemtype="http://schema.org/PostalAddress">
          <div itemprop="streetAddress">123 Elm Street</div>
          <div>
             <span itemprop="addressLocality">Anytown</span>
             <span itemprop="addressRegion">IA</span>
             <span itemprop="postalCode">52402</span>
          </div>
      </div>
      <div itemprop="email">penelope@penelopepizza.com</div>
    </section>
  </body>
</html>
```

Statefulness: Issues with Cookies

- To maintain state across requests, applications historically have used **browser cookies**
- Though cookies have served the Web well, they have some shortcomings:
    - Browser limit cookie size, generally to 4KB max
    - Browsers send the cookie on all subsequent requests until the cookie expires

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Web Storage

- **Web storage** is an API for client-side storage that's widely adopted by browsers
- Most browsers limit Web storage to 5MB for each domain, and a domain cannot access storage from another domain
- Storage comes in two flavors:
    - **Local storage** - Data persists "forever"
    - **Session storage** - Data persists until browser closed

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Web Storage

- The Web Storage API provides methods for defining and retrieving data using key/value pairs

test-webstorage-page1.html

test-webstorage-page2.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

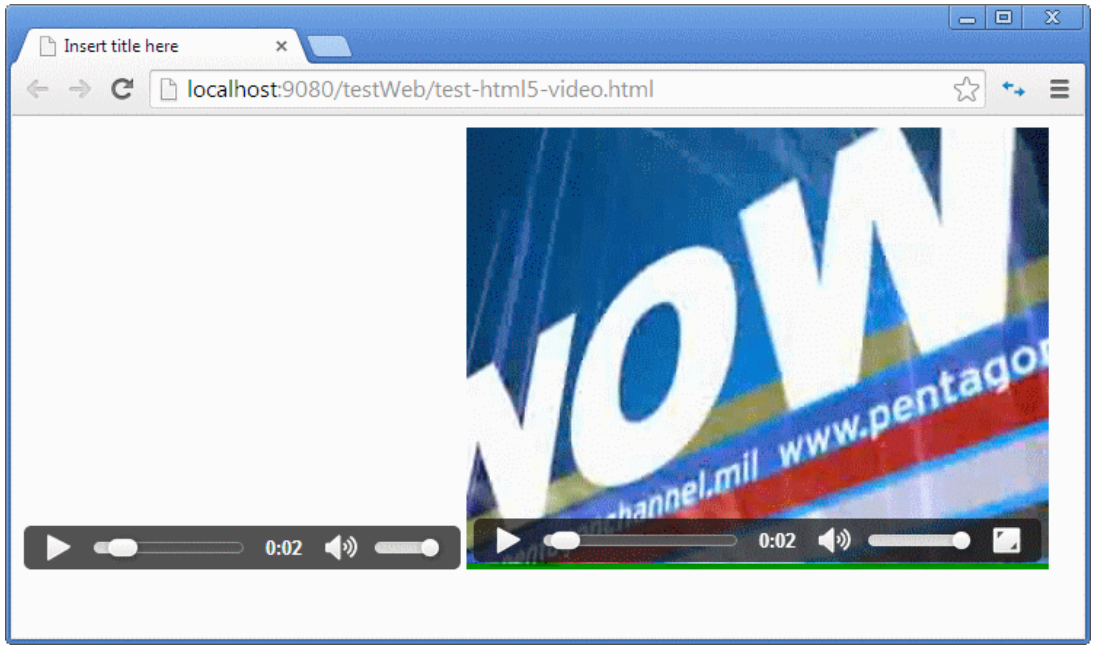
```
<!DOCTYPE html>
<html>
  <head>
    <title>Test Web Storage, Page 1</title>
    <script>
      function saveIt()
      {
        sessionStorage.setItem("mydata1", "Some good data");
        localStorage.setItem("mydata2", "Some more good data");
        return true;
      }
    </script>
  </head>
  <body>
    <form action="test-webstorage-page2.html"
      onsubmit="return saveIt();">
      <input type="submit" value="Next">
    </form>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test Web Storage, Page 2</title>
    <script>
      function getIt()
      {
        var d1 = sessionStorage.getItem("mydata1");
        alert("Session var: " + d1);
        var d2 = localStorage.getItem("mydata2");
        alert("local var: " + d2);
      }
    </script>
  </head>
  <body onload="getIt();">
  </body>
</html>
```

Audio and Video

- HTML5 provides standards-based audio and video playback using the <audio> and <video> tags
- Browser support for the tags is quite good, but support for all audio and video formats is not universal - you can list multiple sources and let the browser choose

test-audio-video.html



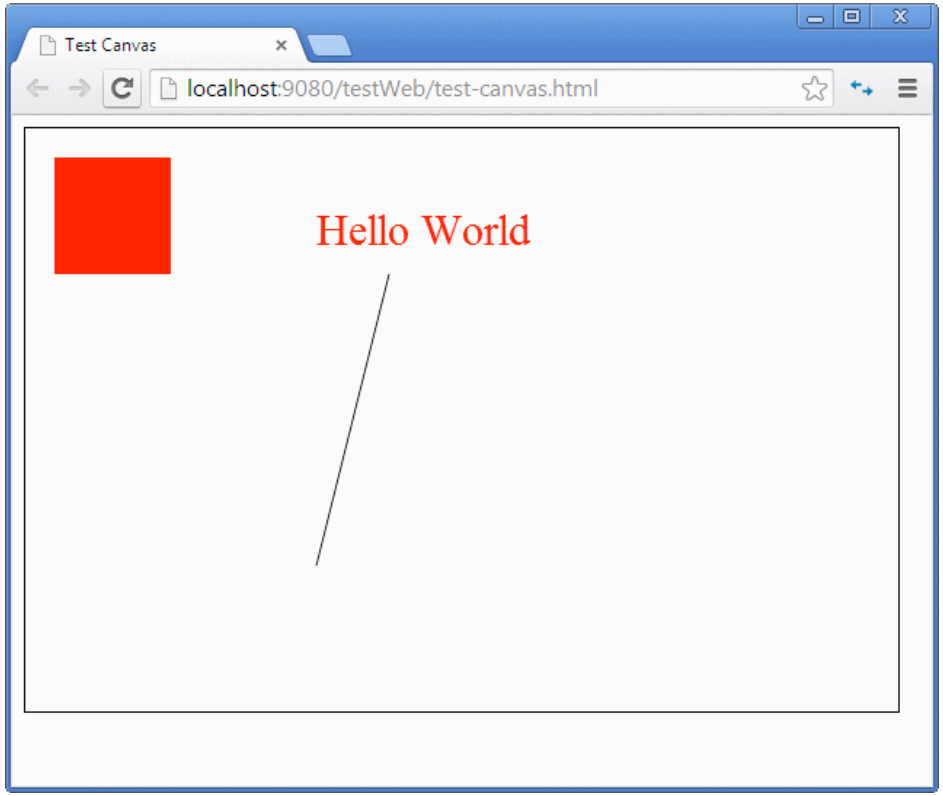
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test Audio/Video</title>
  </head>
  <body>
    <audio controls>
      <source src="ACDC_-_Back_In_Black-sample.ogg"
        type="audio/ogg">
      <source src="ACDC_-_Back_In_Black-sample.mp3"
        type="audio/mpeg">
      <p>Audio not supported fallback content</p>
    </audio>
    <video controls>
      <source src="Pentagon_News_Sample.ogg"
       type='video/ogg;'>
      <p>Video not supported fallback content</p>
    </video>
  </body>
</html>
```

Canvas

- The HTML5 **canvas** API defines a standard way for JavaScript programs to draw to a rectangle on a Web page

test-canvas.html



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test Canvas</title>
    <style>
      canvas {border: solid black 1px;}
    </style>
    <script>
      function drawIt()
      {
        var canvas = document.getElementById("mycanvas");
        var ctx = canvas.getContext("2d");

        ctx.fillStyle = "rgb(255, 0, 0)";

        ctx.fillRect(20, 20, 80, 80);

        ctx.moveTo(250, 100);
        ctx.lineTo(200, 300);
        ctx.stroke();

        ctx.font = "30px Times";
        ctx.fillText("Hello World",200,80);

      }
    </script>
  </head>
  <body onload="drawIt()">

    <canvas width="600" height="400" id="mycanvas">

      <p>No canvas support placeholder</p>
    </canvas>

  </body>
</html>
```

Geolocation

- The **geolocation** API defines a standard way for JavaScript programs to obtain positional information about the user
- Browsers generally require the the user to consent before providing the information

test-geolocation.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>Test Geolocation</title>
    <script>
      function findMe()
      {
         navigator.geolocation.getCurrentPosition(
            foundMe, error);
      }

      function foundMe(position)
      {
        var latitude = position.coords.latitude;
        var longitude = position.coords.longitude;
        document.write("Latitude: " + latitude + "<br>");
        document.write("Longitude: " + longitude);
      }

      function error(errorObj)
      {
        if(errorObj.code == 1) document.write("Denied");
        else if(errorObj.code == 2) document.write("Unavailable");
        else if(errorObj.code == 3) document.write("Timeout");
      }
    </script>
  </head>
  <body onload="findMe()">

  </body>
</html>
```

Chapter Summary

In this chapter, you learned:

- About HTML5 features, such as canvas, microdata, geolocation and Web storage

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

CSS Fundamentals

- Selectors
- Combining Selectors
- Inheritance and Cascade

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What are Cascading Style Sheets?

- Cascading style sheets (CSS) let you define **styles** that govern the appearance of elements in one or more HTML documents
- Major benefit: You can change the look of an entire Web site by modifying one file!

Style

```
h2
{
font-family: sans-serif;
color : Blue;
font-size : large;
font-weight : bold;
}
```

HTML

```
<h2>Motorcycles</h2>
<p>Indian, Harley</p>
<h2>Automobiles</h2>
<p>Ford, BMW</p>
```

Rendered HTML

Motorcycles

Indian, Harley

Automobiles

Ford, BMW

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

CSS History

- CSS Level 1 became a W3C recommendation in December 1966
- CSS Level 2 became a W3C recommendation in May 1998
- As of this writing (2014), CSS Level 3 (CSS3) is still a work in progress - it contains **modules**:
    - Media Queries (2012)
    - Namespaces (2011)
    - Selectors Level 3 (2011)
    - Color (2011)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Guidelines for Writing HTML for CSS

- Basic strategy: create content in HTML, style and layout using CSS
- Avoid old-style HTML tags like <font>, <b> and so forth
- Avoid using <table> for page layout
- Use and for defining content that will be styled

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Styles

- There are three places where you can define styles:
  - In-line within a tag
  - In a page's **head** tag (document-level)
  - In an external file
- Using an external file lets you share the style definitions among multiple HTML files

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Basic Style Syntax

- To define a style, you write a **selector** and enclose the style's properties within curly braces
- Terminate each property with a semicolon (not really required for last property, but recommended)

```
1    /* This is a CSS comment */
2    h2
3    {
4      font-family:sans-serif;
5      color:blue;
6      font-size:large;
7      font-weight:bold;
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Document-Level Styles

- HTML defines a **style** tag in which you can define styles for elements within the page
- The *style* tag must be nested within the *head* tag

```
1    <html>
2      <head>
3        <style>
4          h2
5          {
6            font-family: sans-serif;
7            color : Blue;
8            font-size : large;
9            font-weight : bold;
10         }
11         </style>
12       </head>
13       <body>
14         . . .
15       </body>
16   </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using External Style Sheets

- External style sheets reside in a separate file, typically with .css extension
- To reference the external style sheet, write a **link** tag within the *head* tag

my.css

```
h2
{
font-family: sans-serif;
color : Blue;
font-size : large;
font-weight : bold;
}
```

```
1    <html>
2      <head>
3        <link href="my.css"
4              rel="stylesheet">
5      </head>
6      <body>
7        . . .
8      </body>
9    </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Inline Styles

- Though less common, you can also define styles for single elements directly on the element in the HTML

```
1    <p style="color:purple;font-size:xx-large;">
2    This is some text.
3    </p>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Specifying Colors

- You can specify colors for elements:
  - Using hex (base16)
  - Using rgb() or rgba()
  - Using hsl() or hsla()
  - Using predefined color names

```
1    h1 {color:0xFF0000;}              <!-- red -->
2    h2 {color:rgb(255,0,0);}
3    h3 {color:rgb(255,0,0, .5);}      <!-- 50% opaque -->
4    h4 {color:hsl(0, 100%, 50%);}
5    h5 {color:red;}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Selectors

- **Selectors**, as their name implies, let you point to, or select parts of an HTML document so you can apply styles
- CSS defines the following selector categories:
    - The universal selector
    - Type selectors
    - Attribute selectors
    - Class selectors
    - ID selectors
    - Pseudo class and element selectors

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Universal Selector

- The **universal selector** (*) selects all elements (tags)

```
1    * {font-weight: bold;}  <!-- all elements -->
2
3    div * {color:red;}  <!-- all descendents of div -->
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Type Selectors

- **Tag selectors** (aka **element or type selectors**) let you specify individual HTML elements

```
1    h1 {color:red; font-family:serif;}
2    h2 {color:blue;}
3    h3 {color:green;}
4    h4 {color:yellow;}
5    h5 {color:black;}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Attribute Selectors

- **Attribute selectors** let you specify HTML elements that have *attributes*

```
1    /* draw border only on images with title */
2    img[title] {border:5px solid black;}
3    /* draw background green only for text widgets */
4    input[type="text"] {background-color:green;}
```

```
1    <img title="Figure One" src="test2.jpg">
2    <img src="test3.jpg">  <!-- no border -->
3    <form>
4      <input type="text" name="email">
5      <input type="submit" value="Click here">
6    </form>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Class Selectors

- **Class selectors** let you define *classes* of elements that share styling
- Class names always begin with the period (.) character

```
1    .myclass {color:red;}
2    p.intro {font-family:monospace;}
3    p.summary {font-family:sans-serif;}
```

```
1    <h1 class="myclass">My Title</h1>
2    <div class="myclass">My division</div>
3
4    <p class="intro">Vehicle List</p>
5    <p class="summary">All done!</p>
6    <p>Car, Motorcycle, Scooter</p>  <!-- no style -->
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

ID Selectors

- **ID selectors** let you apply styles based on the ID attribute on HTML elements
- CSS id names begin with the hash (#) character

```
1    #cool {color:blue;}
```

```
1    <p id="cool">This is cool text.</p>
2    <p>This is uncool text.</p>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Pseudo-Class and Psuedo-Element Selectors

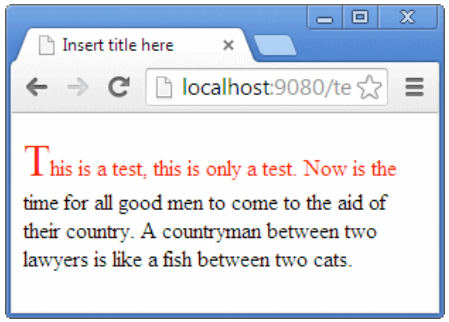
- CSS lets you apply styles to parts of an HTML document that are not elements, e.g. the first line in a paragraph
- Psuedo-class and element selectors include:
    - **a:link** - unvisited link text
    - **a:visited** - visitied link text
    - **a:hover** - link text as user mouses over
    - **:first-letter** - for "drop caps"
    - **:first-line** - style first line of a paragraph

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Pseudo-Class and Psuedo-Element Selectors, cont'd

```
1    p:first-line{color:red;}
2    p:first-letter{font-size: xx-large;}
```

```
1    <p>
2    This is a test, this is only a test.
3    Now is the time for all good men to
4    come to the aid of their country.
5    A countryman between two lawyers is
6    like a fish between two cats.
7    </p>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Grouping Selectors

- You can group selectors by separating them with commas so you can apply the same styling to multiple types

```
1    p, span {color:red;}
```

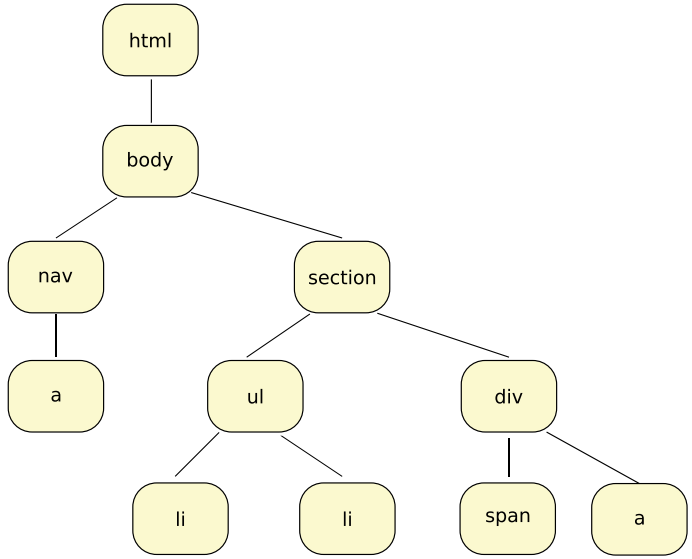
```
1    <p>
2    Now is the time for all good men to
3    come to the aid of their country.
4    </p>
5    <div>
6    A <span>countryman</span> between two lawyers is
7    like a fish between two cats.
8    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Document Tree

- Within an HTML document, you can nest elements within others, thus forming a tree

```
1    <html lang="en">
2      <body>
3        <nav>
4          <a href="home.html">Home</a>
5        </nav>
6        <section>
7          <ul>
8             <li>Item 1</li>
9             <li>Item 2</li>
10         </ul>
11         <div>
12           Some <span>text</span>
13           <a href="test.html">Test</a>
14         </div>
15       </section>
16     </body>
17   </html>
```

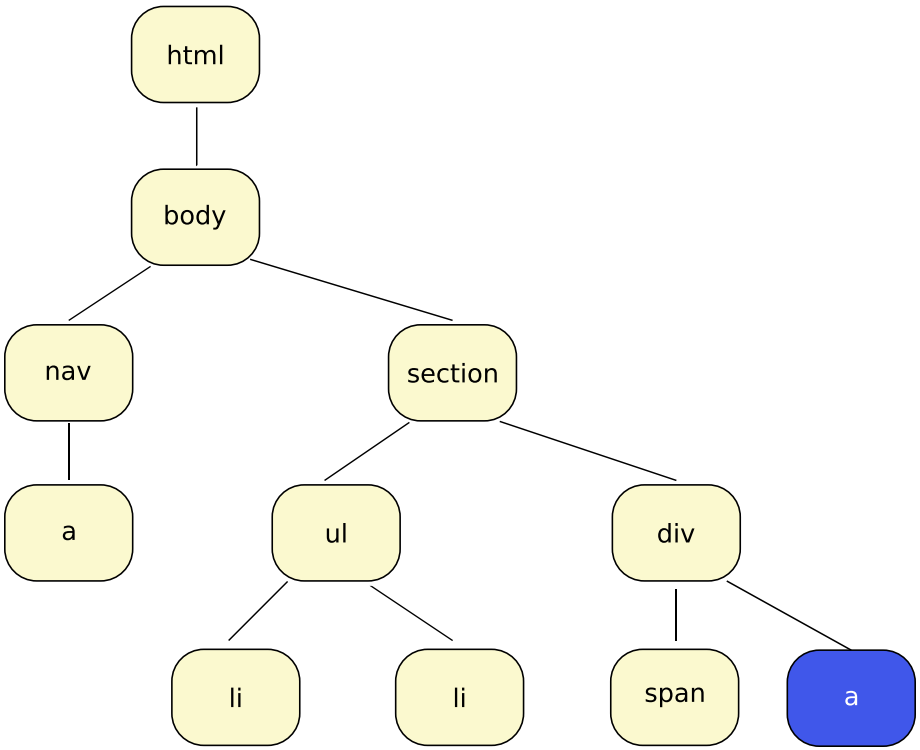


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Child Selectors

- You can specify an element's direct children using the > syntax

```
1     div > a {color:blue;}
```

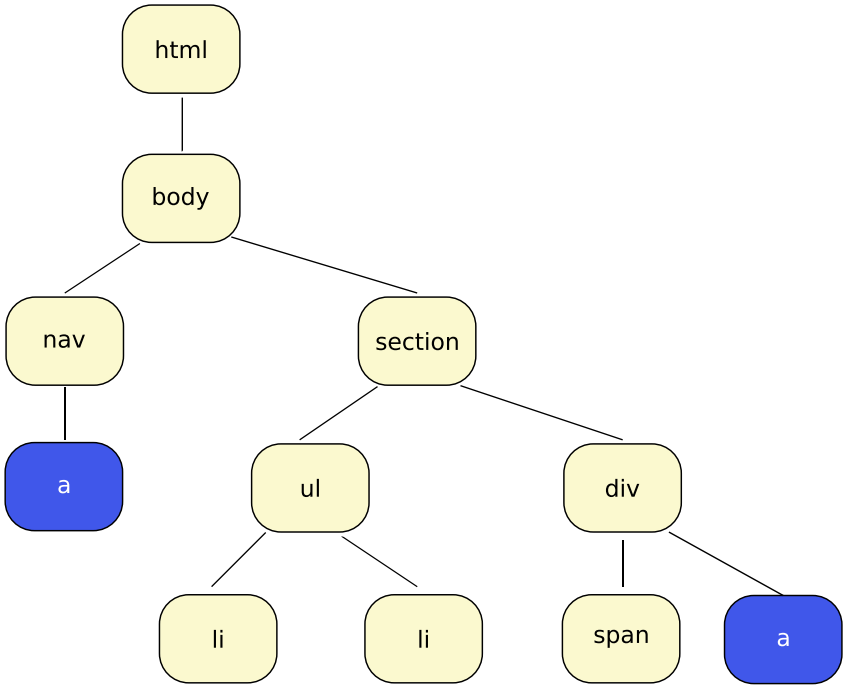


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Descendent Selectors

- You can specify multiple selectors separating with spaces to style based on the document's tree structure
- Unlike with child selectors, there can be many levels between the specified elements

```
1    body a {color:blue;}
```

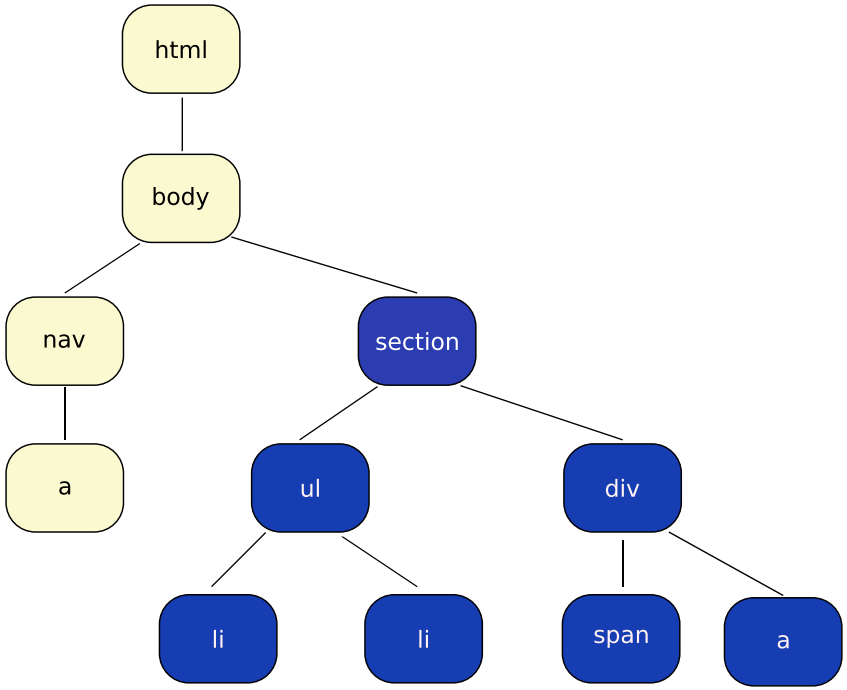


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Style Inheritance

- By default, descendent elements in the HTML tree inherit most styles attached to their ancestor(s)
- You can, of course, override them by writing more specific selectors

```
1    section {font-size: xx-large;}
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

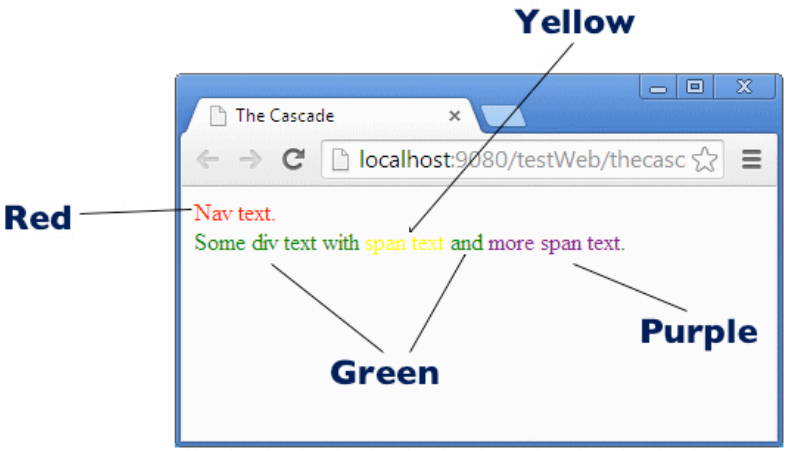
The Cascade

- Either via inheritance, or via an element mapped by multiple selectors, it's easy to have conflicts, where the same element has more than one competing style configured
- The **cascade** resolves these conflicts: the browser calculates a **specificity** score to determine which style wins
- In general, the more specific style declaration wins

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Cascade Example

thecascade.html



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>The Cascade</title>
    <style>
      body {color:red;}
      section {color:blue;}
      div {color:green;}
      span {color:yellow;}
    </style>
  </head>
  <body>
    <nav>
      Nav text.
    </nav>
    <section>
      <div>
        Some div text with <span>span text</span>
        and <span style="color:purple">more span text</span>.
      </div>
    </section>
  </body>
</html>
```

Text Properties

Property name	Description	Possible Values
color	Text color	#RRGGBB, red, blue, etc.
text-align	Alignment	left, center, right, justify,
text-indent	Indentation	Length in pixels, ems, etc.
text-decoration	Adds feature	none,underline,overline, line-through,blink




```
1    p.test-text
2    {
3      color : #9933FF;
4      text-align : left;
5      text-decoration : line-through;
6      text-indent : 10px;
7    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Font Properties

Property name	Description	Possible Values
font-family	Prioritized list	specific name,generic
font-size	Size	xx-small, small,medium large,xx-large, etc.
font-style	Italics?	normal,italic,oblique
font-weight	Boldness	normal,bold,bolder lighter, 100 to 900

Hello

```
1    p.test-fonts
2    {
3      font-family: Arial, Helvetica,
4        sans-serif;
5      font-size: xx-large;
6      font-style: oblique;
7      font-weight: bold;
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Background Properties

Property name	Description	Possible Values
background-color	Color	#RRGGBB, red, blue, etc.
background-image	Image	Reference to image file
background-position	Image position	x,y, top left, top center, etc.
background-repeat	Repeat image?	repeat,repeat-x,repeat-y no-repeat

```
1    p.test-background-image
2    {
3      background-image : url(Greenstone.gif);
4      background-repeat : repeat;
5    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a CSS Validator

- To check for syntax errors, you can use the W3C CSS validator:
- http://jigsaw.w3.org/css-validator/

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Reset Stylesheet

- Since different browsers have different default styling, some Web designers use a **reset** stylesheet that defines uniform properties across browsers
- There are several reset scripts available, including:
    - Eric Meyer's HTML4 reset: http://meyerweb.com/eric/tools/css/reset/
    - Nicolas Gallagher and Jonathan Neal's HTML5 reset: https://necolas.github.io/normalize.css/

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a CSS Preprocessor and/or Framework

- To make working with CSS more like a "real" programming language, there are **CSS preprocessors** that let you define variables and so forth:
    - SASS - http://sass-lang.com/
    - Less - http://lesscss.org/
- There are also some popular **CSS frameworks** that provide components (e.g. breadcrumbs), grids for fluid designs and so forth:
    - Twitter's Bootstrap - http://getbootstrap.com/

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to apply styles using selectors
- The different selectors that CSS provides
- How CSS resolves conflicting styles

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The CSS Box Model and Layouts

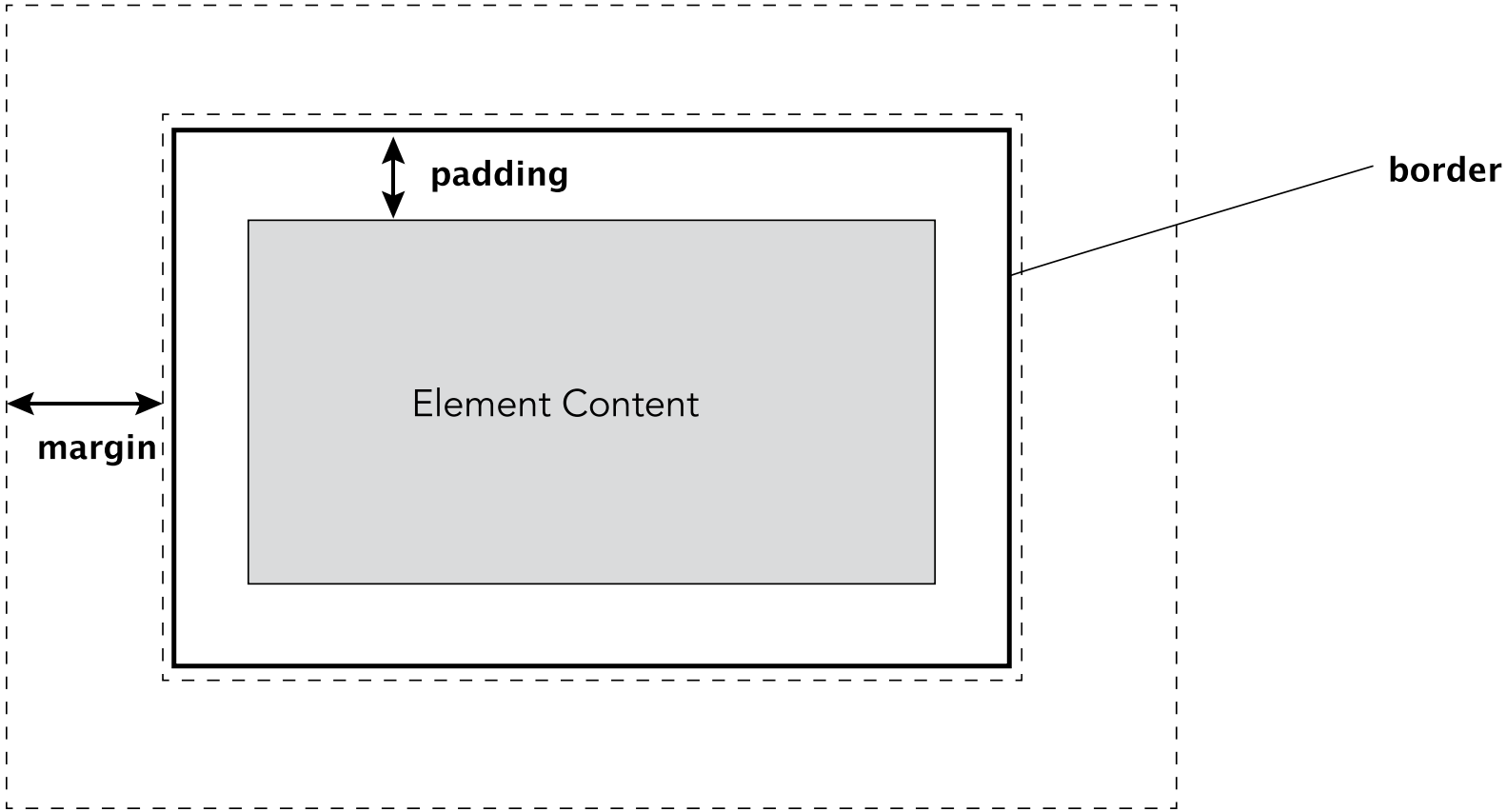
- Borders, Margins and Padding
- Floating Elements
- Fixed and Fluid Layouts

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is the Box Model?

- Browsers treat all HTML content elements as being contained in a box
- CSS provides control over the box's properties



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

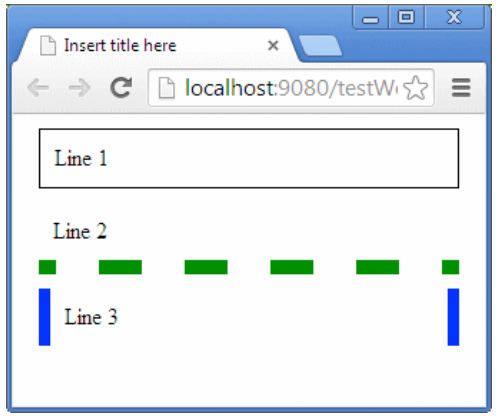
Borders

- The **border** is a line that the browser optionally draws around the element
- You can set the border individually for each of the four box sides or for the entire box
- You can specify width in pixels, ems, or keywords (thin, medium, thick)
- Border properties:
    - color
    - width
    - style

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Borders, cont'd

```
1     .line1
2     {
3       border: thin solid black; /* all sides */
4       padding: 10px; margin: 10px;
5     }
6     .line2
7     {
8       border-bottom: 10px dashed green;
9       padding: 10px; margin: 10px;
10    }
11    .line3
12    {
13      border-left: .5em solid blue;
14      border-right: .5em solid blue;
15      padding: 10px; margin: 10px;
16    }
```

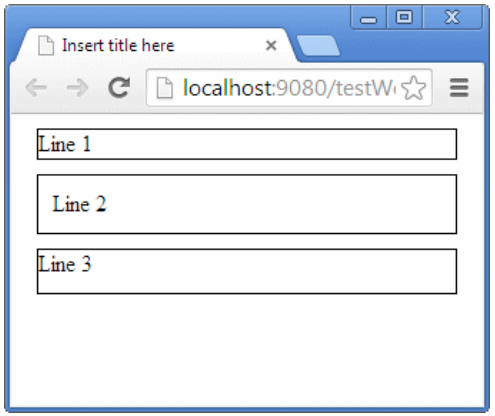


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Padding

- The **padding** is the amount of space between the actual content and the border
- You can set the padding individually for each box side or for the entire box using pixels, ems or percentages

```
1     .line1 {
2       padding: 0px;  /* all sides */
3       border: thin solid black; margin: 10px;
4     }
5     .line2 {
6       padding: 10px;  /* all sides */
7       border: thin solid black; margin: 10px;
8     }
9     .line3 {
10      padding-bottom: 10px;
11      border: thin solid black; margin: 10px;
12    }
```

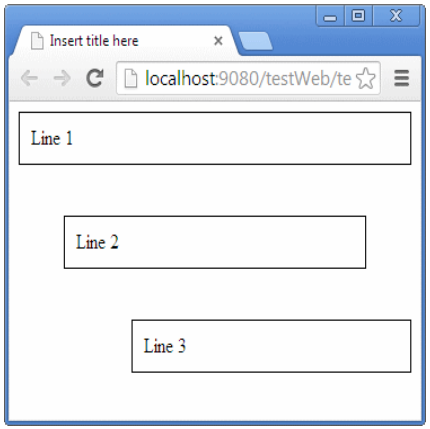


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Margins

- The **margin** is the space between each element's border
- You can set the margin individually for each box side or for the entire box using pixels, ems or percentages

```
1    .line1 {
2    margin: 0px;  /* all sides */
3    border: thin solid black; padding: 10px;
4    }
5    .line2 {
6     margin: 40px; /* all sides */
7    border: thin solid black; padding: 10px;
8    }
9    .line3 {
10   margin-left: 100px;
11   border: thin solid black; padding: 10px;
12   }
```



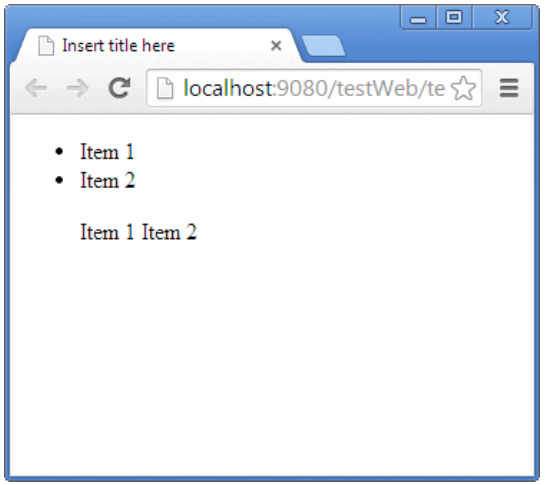
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Inline vs Block Styles

- Browsers draw page breaks after **block** elements, but not after **inline** elements
- By default, , <p>, <li> and so forth are block elements and , <td>, <a>, <img> and so forth are inline
- You can override the default by setting the **display** property

```
1     .secondlist {display: inline;}
```

```
1    <ul>
2     <li>Item 1</li>
3     <li>Item 2</li>
4    </ul>
5    <ul>
6     <li class="secondlist">Item 1</li>
7     <li class="secondlist">Item 2</li>
8    </ul>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Floating Elements

- Normally, the browser draws block elements from top to bottom in the Web page
- You can change that using the **float** property, which has the following values:
    - **left** - Move element to the left of the containing element, wrapping on the right
    - **right** - Move element to the right of the containing element, wrapping on left
    - **none** - No float
- You should always set the **width** property for floated elements, except perhaps for images, which have a native width

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Left Floating Elements

```
1    .leftfloat {float: left; margin-right: 20px;}
```

```
1    <div>
2      <img class="leftfloat" src="test2.png" width="100">
3      <p>
4        Now is the time for all good men to come to
5        the aid of their country. A countryman between
6        two lawyers is like a fish between two cats.
7      </p>
8    </div>
```

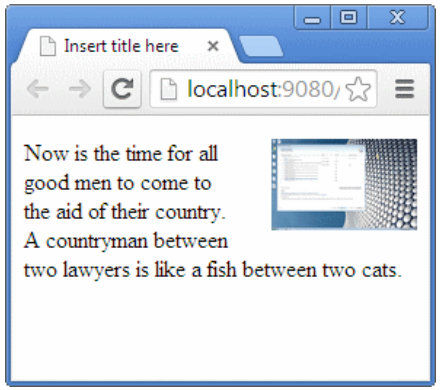


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Right Floating Elements

```
1    .rightfloat {float: right; margin-left: 20px;}
```

```
1    <div>
2      <img class="rightfloat" src="test2.png" width="100">
3      <p>
4        Now is the time for all good men to come to
5        the aid of their country. A countryman between
6        two lawyers is like a fish between two cats.
7      </p>
8    </div>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Floating Elements and Backgrounds

- Sometimes, you don’t want the background of an element "behind" the floated element to show through
- You can use the **overflow:hidden** property to prevent this

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Floating Elements and Backgrounds, cont'd

```
1     h1 {
2       border: solid thin black; background-color: gray;
3       margin: 10px;
4       overflow: hidden; /* omitted on left screenshot */
5     }
6     .sidebar {
7       float:right; border: thin black solid; width: 200px;
8       background-color: yellow;
9       margin-right: 30px; margin-left: 20px;
10    }
```





Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Clearing the Float

- Sometimes you want an element **not** to wrap around a floated element - consider a footer that should always appear at the bottom of the page
- You can use the **clear** property on the footer element so it doesn't wrap
- Values for the **clear** property:
    - **left** - element will drop below left-floats, but potentially wrap around right floats
    - **right** - element will drop below right-floats, but potentially wrap around left floats
    - **both** - element will drop below both left and right floats

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Clearing the Float Example

```
1    .floatleft {float:left; margin-right: 10px;}
2    footer {clear:both; margin-top: 20px;}
```

```
1    <section>
2      <img class="floatleft" src="test2.png" width="100px">
3      <p>
4        Now is the time for all good men to come to
5        the aid of their country.
6      </p>
7    </section>
8    <footer>
9      Copyright &copy; 2014, PGPP, Inc. All Rights Reserved.
10   </footer>
```





Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Web Page Layouts

- Using CSS, you can create Web page layouts in three basic categories:
    - Fixed width
    - Fluid
    - Responsive

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Fixed-Width Layouts

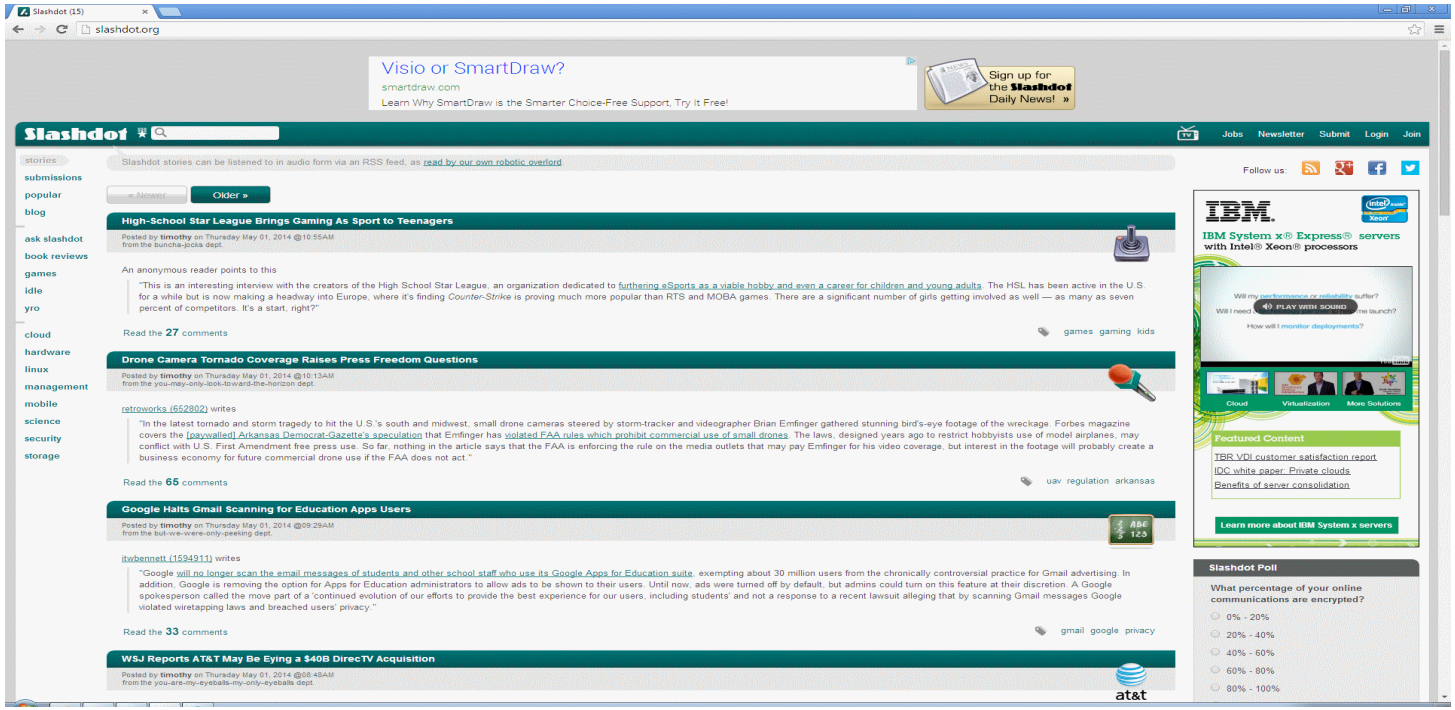
- Many designers coming from a print background prefer **fixed-width** designs
- With a fixed-width design, users with high-resolution screens see empty space on the sides; users with small screens may need to scroll both horizontally and vertically
- To create a fixed-width layout, use a hard-coded pixel value to set the maximum width of the layout



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Fluid Layouts

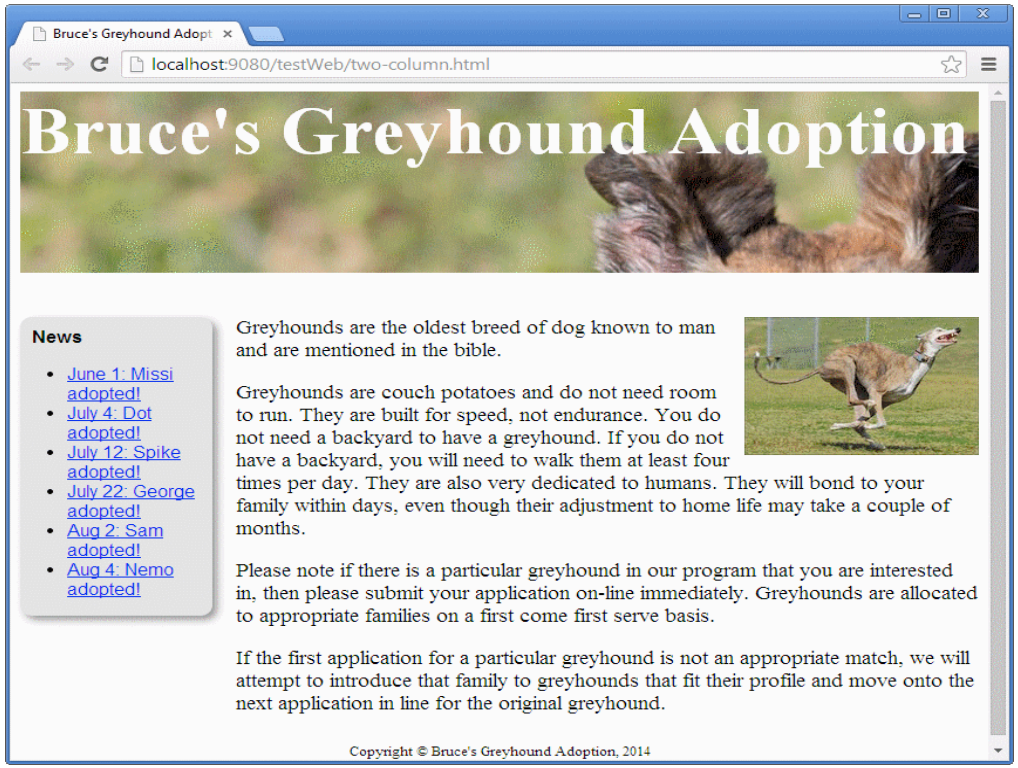
- With a **fluid** layout, the page expands to fill either high or low-resolution screens
- On wide, high-resolution screens, lines of text might get very long, making them hard to read
- To create a fluid layout, use percentages to set the widths of the layout and its sub-elements



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating a Two-Column, Fluid Layout

- A common layout is to have a header, a "main" content rectangle split into a sidebar and main column and then a footer
- The key is to such designs is to float the sidebar to the left or right



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Two Column Recipe

- Define a top-level element, e.g. HTML5 <section> that will hold your two columns
- Define the floating sidebar perhaps using a and give it a width, perhaps a percentage
- Define the main content block perhaps using a and set the margin on the side of the floating element to the same as the floating-element width
- Use a **clear:both** property on an element below the main content

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Two Column Example

two-column.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Bruce's Greyhound Adoption</title>
    <style>
      header h1
      {
        background-image: url(bruce.jpg);
        font-size:4em;
        margin-top: 5px;
        color: white;
        padding-bottom: 100px;
       }
      .sidebar
      {
        float: left;
        width:20%;
        padding-top: 10px;
        padding-bottom: 40px;
        padding-right: 10px;
        background-color: #dddddd;
        margin-right: 10px;
        border-radius: 10px;
        box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.5);
      }
      .main
      {
        margin-left: 22%;
        padding-left: 20px;
        font-size: larger;
      }
      footer
      {
        margin-top: 2em;
        text-align: center;
        font-size: smaller;
      }
      .sidebar h2
      {
        font-family: sans-serif;
        font-size: medium;
        margin-top: 0px;
        padding-left: 10px;
```

```
46        .sidebar li
47        {
48          font-family: sans-serif;
49        }
50        .main img
51        {
52          float:right;
53          margin-left: 10px;
54        }
55      </style>
56    </head>
57    <body>
58       <header>
59         <h1>Bruce's Greyhound Adoption</h1>
60       </header>
61       <nav>
62       </nav>
63       <section>
64         <div class="sidebar">
65           <h2>News</h2>
66           <ul>
67             <li>
68               <a href="missi.html">June 1: Missi adopted!</a>
69             </li>
70             <li>
71               <a href="dot.html">July 4: Dot adopted!</a>
72             </li>
73             <li>
74               <a href="spike.html">July 12: Spike adopted!</a>
75             </li>
76             <li>
77               <a href="george.html">July 22: George adopted!</a>
78             </li>
79             <li>
80               <a href="sam.html">Aug 2: Sam adopted!</a>
81             </li>
82             <li>
83               <a href="Nemo.html">Aug 4: Nemo adopted!</a>
84             </li>
85           </ul>
86         </div>
87         <div class="main">
88           <img src="greyhound.jpeg" width="200px;">
89           <p>
```

```
91              man and are mentioned in the bible.
92            </p>
93            <p>
94               Greyhounds are couch potatoes and do not need room to run.
95               They are built for speed, not endurance.  You do not need a
96               backyard to have a greyhound. If you do not have a backyard,
97               you will need to walk them at least four times per day.
98               They are also very dedicated to humans.  They will bond to
99               your family within days, even though their adjustment to
100              home life may take a couple of months.
101           </p>
102           <p>
103             Please note if there is a particular greyhound in our
104             program that you are interested in, then please submit
105             your application on-line immediately. Greyhounds are allocated
106             to appropriate families on a first come first serve basis.
107           </p>
108           <p>
109            If the first application for a particular greyhound is
110            not an appropriate match, we will attempt to introduce that
111            family to greyhounds that fit their profile and
112            move onto the next application in line for the original greyhound.
113           </p>
114         </div>
115         <div style="clear: both;"></div>
116       </section>
117       <footer>
118         Copyright &copy; Bruce's Greyhound Adoption, 2014
119       </footer>
120     </body>
121   </html>
```

Chapter Summary

In this chapter, you learned:

- About the CSS box model: borders, padding and margins
- About floating elements
- How to create fluid page layouts

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to CSS3

- What's New in CSS3?
- Rounded Corners
- Transitions, Transforms and Animations
- Intro to Media Queries

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What's New in CSS3?

- CSS3 is actually not an official standard, since the the W3C has split CSS into independently evolving **modules**: Selectors, Media Queries, Backgrounds and Borders, Box Model, etc.
- Nontheless, there are some widely accepted techniques that the community refers to as CSS3

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Browser Support

- As of this writing (mid 2014), browser support for these features is not universal, so your pages should not depend on them (degrade gracefully) if your user population has older browsers, especially Internet Explorer 9 or below
- But even among non-IE browsers, support is spotty, but in some cases, you can accomplish the same effect using **vendor prefixes**

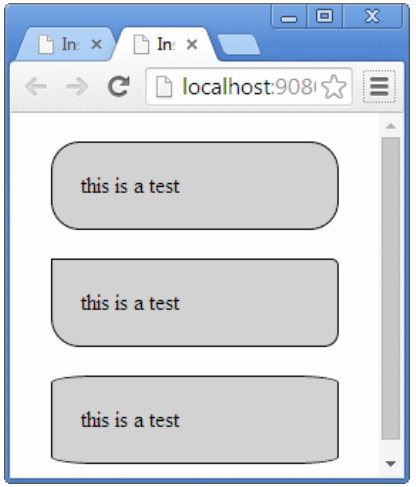
```
1    -webkit-transform:rotate(7deg); /* Chrome, Safari, etc. */
2
3    -moz-transform:rotate(7deg); /* Firefox */
4
5    -o-transform:rotate(7deg); /* Opera */
6
7    -ms-transform:rotate(7deg); /* IE */
8
9    transform:rotate(7deg); /* CSS standard syntax */
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Rounded Corners

- Prior to CSS3, developers typically used images to create boxes with rounded corners
- CSS3 provides the **border-radius** property in which you can specify a portion of a circle or ellipse for all corners or for each corner individually

```
1    .round1  { /* all corners circular */
2      border-radius: 20px;
3    }
4    .round2 { /* individual corners */
5      border-radius: 0px 5px 10px 20px;
6    }
7    .round3 { /* all corners elliptical */
8      border-radius: 30px/5px;
9    }
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Text Shadows

- Prior to CSS3, developers had to use browser-specific hacks to draw shadows
- CSS3 provides the **text-shadow** property that let you specify the shadow's position, **blur radius** and color

```
1    .shadow1 {
2      text-shadow: 4px 4px 2px gray;
3      border: 1px solid black;
4      font-size: 3em;
5      padding: 20px;
6      margin: 20px;
7    }
```




```
1    <div class="shadow1">
2      this is a test
3    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Box Shadows

- Prior to CSS3, developers had to use browser-specific hacks to draw shadows
- CSS3 provides the **box-shadow** property that let you specify the shadow's position, **blur radius** and color

```
1     .shadow2 {
2       box-shadow: -4px -4px 4px black;
3       border: 1px solid black;
4       font-size: 3em;
5       padding: 20px;
6       margin: 20px;
7     }
```

```
1     <div class="shadow2">
2       this is a test
3     </div>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transforms

- Though not yet universially supported by browsers, CSS lets you specify the **transform** property on elements to rotate, translate (move), scale and skew (distort)

```
1    .transform1 {
2      transform:rotate(7deg);
3      border: 1px solid black;
4      font-size: 3em;
5      padding: 20px;
6      margin: 20px;
7    }
```

```
1    <div class="transform1">
2      this is a test
3    </div>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transitions

- CSS3 **transitions** are animations between CSS properties over a fixed time period
- You can specify a **trigger** that causes the transition to begin

```
1    /* when user hovers mouse, change background color
2       to yellow over the course of 1 second */
3    .transition1 {
4      border: 1px solid black;
5      background-color: green;
6      font-size: 3em;
7      padding: 20px;
8      margin: 20px;
9      transition-property: background-color;
10     transition-duration: 1s;
11   }
12   .transition1:hover {
13     background-color: yellow;
14   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transitions, cont'd

```
1    /* on hover, rotate and change border width
2       over the course of 1 second */
3    .transition2
4    {
5      border: 1px solid black;
6      font-size: 3em;
7      padding: 20px;
8      margin: 20px;
9      transition-duration: 1s;
10     transition-property: all;
11   }
12   .transition2:hover
13   {
14     border-width: 10px;
15     -webkit-transform:rotate(7deg);
16     transform:rotate(7deg);
17   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Animations

- CSS3 **animations** are like transitions, but more powerful - you specify a set of **keyframes** that the browser draws to create the animation
- Animations are a bit more complicated to set up than transitions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Animation Keyframes

- A **keyframes rule** consists of keyframes that lets you specify a set of CSS properties - by defining multiple frames with different properties, you create the animation effect
- You assign a name to the keyframes rule that you will use to apply the animation

```
1    @keyframes fadeIn
2    {
3      from
4      {
5        opacity: 0;
6      }
7      to
8      {
9        opacity: 1;
10     }
11   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Applying the Animation

- After creating the keyframes, you apply the animation to some element or elements on the page using the **animation-name** and **animation-duration** properties
- By default, the animation starts when the page loads

```
1    .animation1
2    {
3      animation-name: fadeIn;
4      animation-duration: 3s;
5
6      border: 1px solid black;
7      background-color: green;
8      font-size: 3em;
9      padding: 20px; margin: 20px;
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Multiple Keyframes

- You can use percentages to define multiple keyframes

```
1    /* change an element's background and position */
2    @keyframes moveIt {
3      from {
4        background-color: blue;
5      }
6
7      50% {
8        background-color: magenta;
9        transform: translateX(1em);
10     }
11
12     to {
13       background-color: yellow;
14       transform: translateX(2em);
15     }
16   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Animation Example

animation.css

test-animations.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
@CHARSET "ISO-8859-1";

/* first animation */
@keyframes fadeIn
{
  from
  {
    opacity: 0;
  }
  to
  {
    opacity: 1;
  }
}

@-webkit-keyframes fadeIn
{
  from
  {
    opacity: 0;
  }
  to
  {
    opacity: 1;
  }
}

.animation1
{
  -webkit-animation-name: fadeIn;
  -webkit-animation-duration: 3s;

  animation-name: fadeIn;
  animation-duration: 3s;

  border: 1px solid black;
  background-color: green;
  font-size: 3em;
  padding: 20px;
  margin: 20px;
}

/* second animation */
@keyframes moveIt
```

```
  from
  {
    background-color: blue;
  }

  50%
  {
    background-color: magenta;
    transform: translateX(1em);
  }

  to
  {
    background-color: yellow;
    transform: translateX(2em);
  }
}

@-webkit-keyframes moveIt
{
  from
  {
    background-color: blue;
  }

  50%
  {
    -webkit-transform: translateX(1em);
    background-color: magenta;
  }

  to
  {
    -webkit-transform: translateX(2em);
    background-color: yellow;
  }
}

.animation2
{
  -webkit-animation-name: moveIt;
  -webkit-animation-duration: 3s;

  animation-name: moveIt;
```

```
91
92      width: 5em;
93      border: 1px solid black;
94      background-color: green;
95      font-size: 3em;
96      padding: 20px;
97      margin: 20px;
98    }
99
100   /* third animation */
101   @keyframes rotateIt
102   {
103     from
104     {
105     }
106
107     to
108     {
109       transform: rotate(45deg);
110     }
111   }
112
113   @-webkit-keyframes rotateIt
114   {
115     from
116     {
117     }
118
119     to
120     {
121       -webkit-transform: rotate(45deg);
122     }
123   }
124
125   .animation3
126   {
127     -webkit-animation-name: rotateIt;
128     -webkit-animation-duration: 3s;
129
130     animation-name: rotateIt;
131     animation-duration: 3s;
132
133     width: 5em;
134     border: 1px solid black;
```

```
136    padding: 20px;
137    margin: 20px;
138  }
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="ISO-8859-1">
    <title>Animations</title>
    <link rel="stylesheet" href="animation.css">
  </head>
<body>
  <div class="animation1">
    this is a test
  </div>
  <div class="animation2">
    this is a test
  </div>
  <div class="animation3">
    this is a test
  </div>
</body>
</html>
```

Introduction to Media Queries

- **Media queries** let you target CSS styles based on a the browsing device's capabilities such as width, height, resolution and so forth
- Using media queries is an important facet of designing **responsive Web pages** that adapt to different devices such as phones, tables and desktop computers

```
1    @media screen and (max-width: 1000px)
2    {
3      .media1 {background-color: red;}
4    }
5    @media screen and (max-width: 800px)
6    {
7      .media1 {background-color: blue;}
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About CSS3 features

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Designing Responsive Web Applications

- Flexible Grids
- Media Queries
- Minimizing Bandwidth

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Responsive Design?

- **Responsive design** is a set of principles using HTML and CSS3 to create Web sites that respond to different screen sizes
- The term "responsive design" was coined by Ethan Marcotte
- You can read his article at:

```
http://alistapart.com/article/responsive-web-design
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Principles of Responsive Designs

- Use flexible, relative, percentage-based measurements rather than fixed, pixel-based values
- Use fluid rather than fixed layouts
- Use media queries as necessary to customize layout for different screen widths
- Scale images to fit the screen size
- Minimize page size, especially images, for low-bandwidth devices

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Strategies for Responsive Designs

- Content is more important than navigation, sidebars, etc.
- Start with mobile first
- Progressive enhancement - start as simple as possible, then test with larger screen widths and add extras when it starts to look bad

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

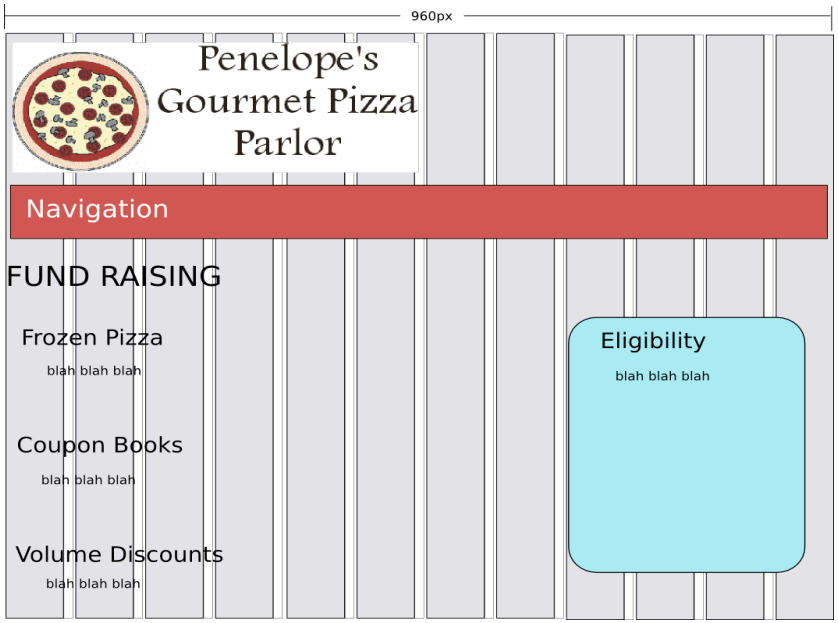
Alternatives to Responsive Designs

- Fixed-width (pixel-based) pages that look the same on as on the printed page
- Separate, complete Web sites for desktop browsers and small-screen devices (i.e. a "mobile" Web site)
- Device specific "apps" that duplicate Web functionality

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Relative Measurements

- Graphic artists typically create page mockups using Photoshop or other image-editing program
- For a responsive design, we need to translate fixed, pixel-based measurements into percentages
- show figure of a mockup with text and grids
- Traditionally, such mockups have a fixed 960px width



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Calculation

- Instead of using pixels, we can use a simple formula to convert to relative measurements such as percentages or **ems**:

$$\text{target} \div \text{context} = \text{result}$$

- We can apply this to:
  - Widths of s and other block elements
  - Font sizes
  - Margins and padding
- The key is determining the **context**, which usually is the enclosing element's wdith

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Applying the Formula: Font Size

- For proportional font size in **ems**, take the desired pixel height and divide it by the browser's default pixel height (16 pixels)
- It's a good idea to show the calculation as a comment
- For example, if the pixel-based mockup called for 24 pixel text:

```
1    h1
2    {
3      1.5em;   /* 24px / 16px */
4    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Applying the Formula: Block Elements

- For block-element widths and margin widths, the **context** is the containing element
- For padding width, the **context** is the element itself

```
1    /* containing element desired width: 900px */
2    /* desired width of sidebar: 548px */
3    .sidebar
4    {
5       float: left;
6       width: 60.888888888889%;          /* 548px / 900px */
7       margin-right: 2.444444444444%;    /* 22px / 900px */
8       padding-left: 0.0182481751824%; /* 10px / 548px */
9       padding-top: .625em;              /* 10px / 16px */
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Additional Responsive Concepts

- Users with small screens often don't have a mouse and must use fingers to "press" buttons - consider making navigation elements relatively larger
- Mobile users might be interested in different content - for example, a restaurant's desktop page might feature the food and beverage menus, while mobile page might feature operating hours

How can we tailor the page?

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What are Media Queries?

- A hallmark of responsive designs is tailoring appearance to screen size
- You can use CSS3 **media queries** to create conditional styles, typically based on screen width

```
1    @media screen and (max-width: 1000px)
2    {
3      .media1 {background-color: red;}
4    }
5    @media screen and (max-width: 800px)
6    {
7      .media1 {background-color: blue;}
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What Can You Query?

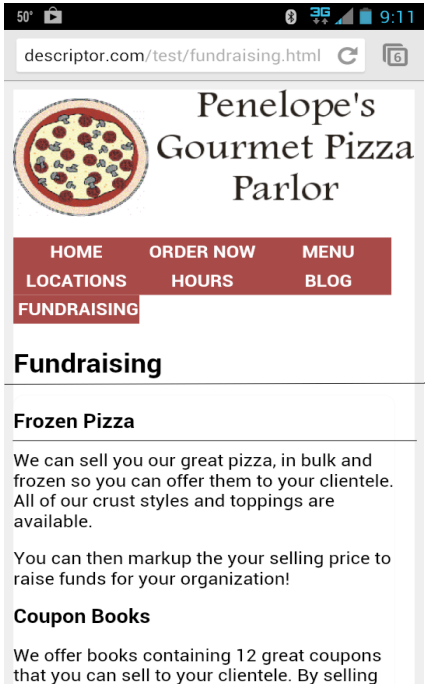
- CSS3 lets you query many **features**, but most responsive pages use **width** or **device-width**

Feature	Min/Max?	Description
width	Y	Viewport width
height	Y	Viewport height
device-width	Y	Screen width
device-height	Y	Screen height
orientation	N	landscape, portrait
aspect-ratio	Y	width/height
device-aspect-ratio	Y	device-width/device-height
color	Y	Number of color bits
color-index	Y	Color table entry count
monochrome	Y	Number of mono bits
resolution	Y	Pixel density, e.g. dpi
scan	N	progressive?
grid	N	grid, bitmap

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Viewport vs Screen Size

- Every device or screen has a physical, pixel-based width and height
- However, browsers, even on mobile devices, often restrict pages to a smaller rectangle known as the **viewport**
- Most designers use the viewport width in their media queries



device-width

width (viewport)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defeating Mobile Device Scaling

- Many browsers on small devices (especially) the iPhone, assume a fixed page size of 980x
- The browser thus scales content to that size, even for responsive designs - the user has to pinch scroll and zoom
- You can defeat this scaling by writing a **meta** tag in each page:

```
1    <meta name="viewport"
2       content="initial-scale=1.0,
3       width=device-width" />
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Media Query Breakpoints

- Responsive designers often target well-known common screen widths, for example, cell phone (400px), tablet (800px) and desktop (larger than 800ypx)
- You can use multiple media queries based on these values to define **breakpoints** for these common sizes

```
1    @media screen and (max-width: 400px)
2    {
3      .media1 {background-color: red;}
4    }
5    @media screen and (max-width: 800px)
6    {
7      .media1 {background-color: blue;}
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Minimizing Image Size

- Avoid large background images - tile smaller ones if possible
- You can instruct browsers to scale images to the page width:

```
1     img
2     {
3       max-width: 100%;
4     }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the principles of responsive design
- How to use fluid grid layouts, media queries and image scaling to create responsive pages

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaScript

- What is JavaScript?
- JavaScript Objects

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is JavaScript?

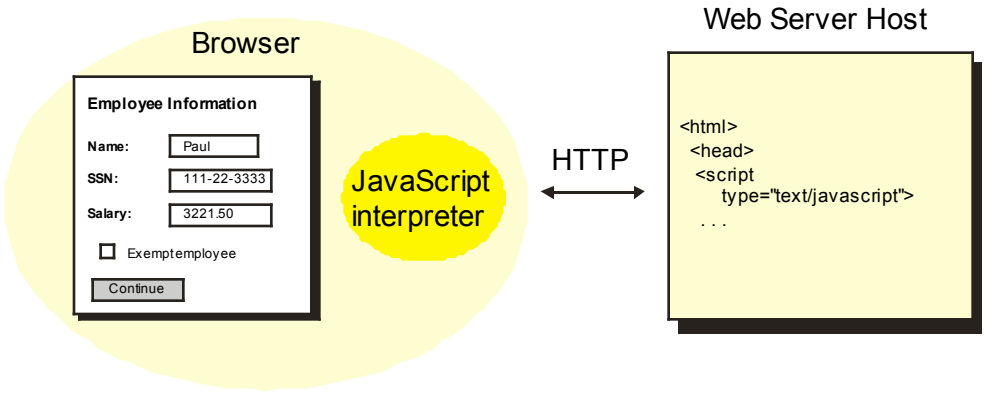
- JavaScript is a modern, object-oriented programming language
- JavaScript is considered to be a **weakly typed** language and supports **inheritance by prototype** and **functional programming**



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Where Does JavaScript Run?

- A very common environment for JavaScript is referred to as **client side** in which JavaScript runs within a Web browser
- You can use JavaScript in a browser to:
  - Validate forms
  - Expand/collapse page elements
  - Communicate with the server via Ajax
- JavaScript also can run on the server or in **embedded mode** within other applications



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript vs Java

- JavaScript is not really related to the Java programming language, except for a roughly similar syntax



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

History of JavaScript

- JavaScript was originally developed by Netscape in 1995 under the name **LiveScript**
- Microsoft added JavaScript support to Internet Explorer in version 3.0
- Netscape submitted the language to the European Computer Manufacturers Association (ECMA) - the standardized version is known as **ECMAScript**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

ECMAScript Versions

- **Version 1.0 1997** - First edition
- **Version 2.0 1998** - Minor changes
- **Version 3.0 1999** - Added regular expressions, exception handling etc.
- **Version 4.0** - Abandoned
- **Version 5.0 2009** - Strict mode, JSON support, etc.
- **Version 6.0 2015** - Focused on writing applications; includes *modules* and additional collection types

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Browser Support for ECMAScript Versions

- Most modern browsers support ECMAScript 3, but version 5 support is becoming universal, even on mobile browsers
- See http://kangax.github.com/es5-compat-table for a chart
- Also see http://caniuse.com/use-strict for ECMAScript 5 **strict mode** support

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript is Typically Interpreted

- Most JavaScript implementations (e.g. in a Web browser) **interpret** the JavaScript source code as it executes
- Thus, users must have access to the actual source code

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Four Categories of JavaScript Functionality

- **Core**: The common syntax and function
- **Client-side**: Implemented by Web browsers
- **Server-side**: Implemented by engines, for example, Node.js
- **Embedded**: Implemented by engines such as Rhino

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Syntax Overview

- JavaScript is cASe SENsitive
- JavaScript statements can span more than one line and should be terminated with a semicolon
- You define variables using the **var** keyword
- JavaScript supports various ways to add comments, including //
- You can create character strings delimiting with double-quotes or single quotes

```
1    var x = 12;
2
3    alert(x);  // display in a message box
4
5    // display in browser's developer tools console
6    console.log("x: " + x);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Try it Now!

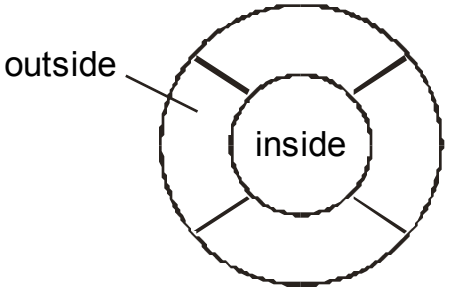
- Open a Web browser and in the URL entry field, type:

```
javascript:alert("Hello, world");
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is an Object?

- In object-oriented programming, an **object** is a self-contained chunk of **state** and **behavior**
- Objects respond to a stimulus (function call) often by working on the object's state
- For example, an "account" object might support a **debit()** behavior that reduces the object's **balance** state



Object Name	Sample State	Sample Behavior
lassie	weight	bark
account12	balance	debit
policy765	amount	payOut

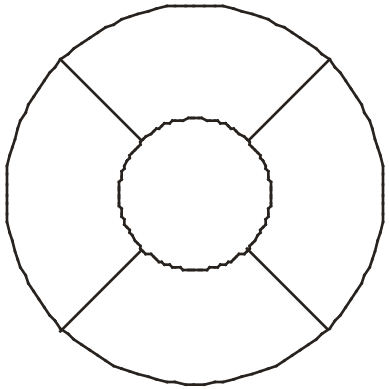
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

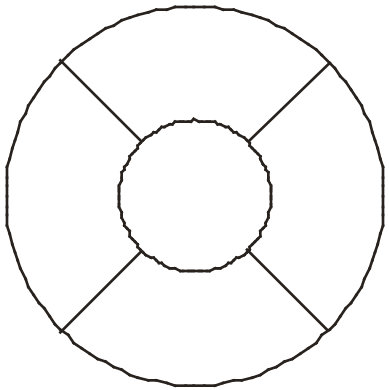
Object Terminology

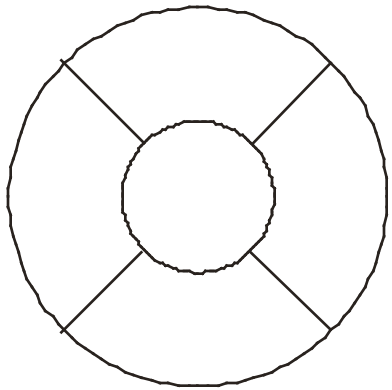
- An object's state is data - synonyms include **properties** and **attributes**
- An object's behavior is program logic provided by **methods** also known as **functions** or **subroutines**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Objects Exercise







Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Objects in JavaScript

- JavaScript itself provides several predefined objects that provide **properties** (state), **methods** (behavior) and **events**
- To access an object, you use the **dot** operator
- It's also possible to create your own custom JavaScript objects using **prototypes**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Client-Side JavaScript Objects

- Client-side JavaScript is runs in a browser - the browser vendor provides the JavaScript interpreter
- The two most important client-side JavaScript **objects** are:
    - **window** - represents the browser's window (or frame)
    - **document** - represents the page

```
1    // store last modified date in a variable 's'
2    var s = document.lastModified;
3
4    // print the contents of the browser window
5    window.print();
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Embedding JavaScript in HTML

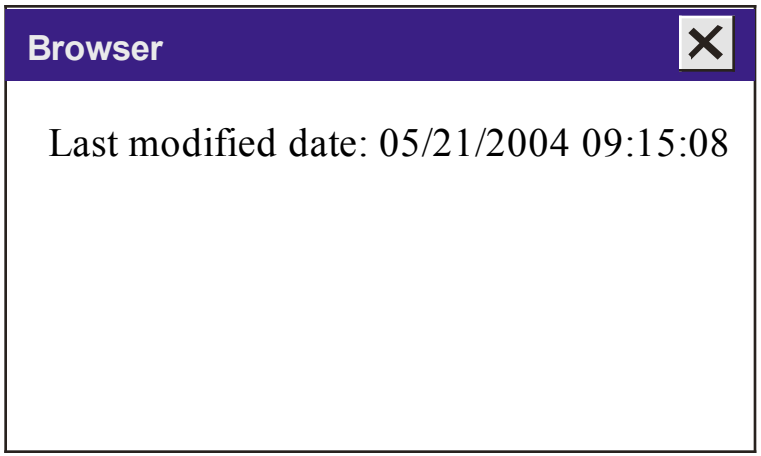
- There are four ways to embed JavaScript in HTML
    - **Inline** using the **script** tag
    - **External file** using the **src** attribute of a script tag
    - **Within an event handler** such as a button's **onclick** event
    - **Within a javascript: URL**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using the script Tag

- You can embed one or more **script** elements in a Web page's *head* or *body*

```
1     <html>
2       <body>
3         <script>
4           document.write ("Last modified date: ");
5           document.write (document.lastModified);
6         </script>
7       </body>
8     </html>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using an External File

- For reuse and for large complex scripts, you can place the script in its own file and reference from an HTML page
- The **src** attributes specifies the URL of the external script
- The script file can be located local to the HTML page or on an external Web server

```
1    file1.html
2    ----------
3
4    <html>
5     <body>
6      <script src="first.js">
7      </script>
8     </body>
9    </html>
```

```
1    first.js
2    ----------
3
4    document.write
5      ("Last modified date: ");
6    document.write
7      (document.lastModified);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Functions

- A **function** is a block of re-usable code that can accept parameters and return values
- It's common to define functions in the **head** and then call them from other scripts

```
1     <head>
2       <script>
3         function toCelsius(f)
4         {
5           return (5/9) * (f-32);
6         }
7       </script>
8       <script>
9          alert(toCelsius(-40));
10      </script>
11    </head>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Modifying Page Elements

- It's common in client-side JavaScript to respond to **events** and then modify something on the page
- The **document** object provides the **Document Object Model** (DOM) API

```
1    <div id="mydiv">Initial text</div>
2    <button onclick="changeIt()">Press Me</button>
3
4    <script>
5      function changeIt()
6      {
7        var theDiv = document.getElementById("mydiv");
8        theDiv.innerHTML = "New text";
9      }
10   </script>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Form Submission Event

- If a page defines a form with a submit button, you can write JavaScript that fires when the user submits the form - the JavaScript runs before the browser sends the request to the server

```
1    <script>
2      function confirm() {
3        var aNumber = Math.random();
4        console.log("aNumber: " + aNumber);
5        if(aNumber < 0.5) return true;
6        else return false;
7      }
8    </script>
9
10   <form onsubmit="return confirm()"
11     action="http://localhost/myserver">
12     <input type="submit">
13   </form>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- What JavaScript is, and what it is not
- The categories of JavaScript functionality
- How to write and embed simple scripts into HTML

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Data Types and Assignment

- Defining Variables
- Data Types

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Comments

- JavaScript supports two kinds of comments:
    - Single line comments
    - Block comments

```
1    // single-line comment
2
3    /* multi-line comment
4
5      all of this is not processed
6
7      neither is this
8
9    */
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Basics

- In general, JavaScript programs consist of **statements**, each of which is **optionally** terminated with a semicolon
- JavaScript is a **case-sensitive** language - most predifined keywords are in lower case

```
1    var x = 1;   // define a variable
2
3    alert(x);    // call a method (function)
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Variables

- **Variables** are locations in memory that hold a value
- You declare a variable using the **var** keyword
- Even though JavaScript supports data types, variables are **untyped**
- Datatypes include:
    - Primitives: **numbers**, **strings** and **booleans**
    - Objects: **Date**, **Math**, etc.

```
1    var count = 12;                  // number
2    var interestRate = 4.5;          // number
3    var name ="Paul Westerberg";     // string
4    var flag = false;                // boolean
5    var now = new Date();            // Date object
6
7    count = "abcd";   // change variable's type!
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Rules for Identifiers

- Any length combination of letters, digits and special characters: ('_ ', '$') but must begin with an letter or special character
- JavaScript is case sensitive!

Valid	Invalid
count	1account
interestRate	order-number
purchase_Number1	delivery date

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Reserved Words

break	this	private
case	throw	protected
catch	try	public
continue	typeof	static
debugger	var	yield
default	void	const
delete	while	
do	with	
else	class	
finally	enum	
for	export	
function	extends	
if	import	
in	super	
instanceof	implements	
new	interface	
return	let	
switch	package	

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Numbers

- Internally, JavaScript stores numbers as IEEE 64-bit floating-point values
- You can specify both integer and floating-point **literals** in expressions

```
1    var a = 12;      // integer literal
2    var b = 0xff;    // integer literal (hex)
3
4    var c = 2.3;     // floating-point literal
5
6    var d = 6.032e23; // floating-point literal
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Booleans

- The boolean data type allows only the literal values **true** and **false**
- The most common use of booleans is in flow control statements (e.g. the "if" statement)

```
1    var finished = false;
2    var ready = true;
3
4    var ok = confirm("Continue?");
5    if(ok)
6    {
7      // do something
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Truthy and Falsy

- JavaScript converts the following to **false** and are considered to be **falsy**:
    - undefined
    - null
    - 0
    - -0
    - NaN
    - ""
- All other values are considered to be **truthy**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Truthy and Falsy, cont'd

```
1    var a = "hello";
2    if (a)
3    {
4      // a is truthy, we will get here
5    }
6
7    var b = "";
8    if (b)
9    {
10     // b is falsy, won't get here
11   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Strings

- Strings hold sequences of characters
- To delimit a literal string, use single or double quotes
- Strings define a **length** property and several methods
- You can concatenate strings using the + operator

```
1    var s1 = "If Only You Were Lonely";
2    var index = s1.indexOf ('Only');
3
4    var s2 = s1 + " Not";
5    var count = s1.length;
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

String Literal Escape Codes

- JavaScript defines special **escape codes** for common non-printable characters
- You can initialize a character to any Unicode value using an integer constant

```
\b      Backspace
\t      Horizontal tab
\n      Linefeed
\f      Form feed
\r      Carriage return
\"      Double quote
\'      Single quote
\\      Backslash
\uxxxx  Hexadecimal Unicode code point
\xxx    Octal Unicode code point
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dates and Times

- JavaScript programs use **constructors** to create objects
- Core JavaScript provides a **Date** constructor to let you work with dates and times

```
1     var d1 = new Date();  // current date/time
2
3     var d2 = new Date(1993, 0, 18); // January 18, 1993
4
5     // January 18, 1993 6:10:20PM local
6     var d3 = new Date(1993, 0, 18, 18, 10, 20);
7
8     var year = d2.getFullYear();
9     var month = d2.getMonth();
10    var day = d2.getDate();
11    var dayOfWeek = d2.getDay(); // 0 - Sunday
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Scope of Variables

- JavaScript supports **global** and **local** variables
- Variables defined within a function (method) are **local** and can only be used by that method

```
1    var a = "My global variable";
2
3    function function1()
4    {
5      var b = "My local variable";
6
7      alert(a);  // access global
8    }
9
10   function function2()
11   {
12     alert(a);  // access global
13     alert(b);  // error - out of scope
14   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Quick Practice

- Define a variable and initialize it to your favorite whole number:
- Define a variable and initialize it to your favorite floating-point number:
- Define a variable and initialize it to your name:

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The rules for defining and initializing variables
- The JavaScript data types

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Operators

- Operator Types
- Assignment

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Operators

- Operators let you perform math, comparisons and so forth
- JavaScript provides 47 operators, which we can divide into categories:
    - Math
    - Equality and Relational
    - Assignment
    - Boolean Logical
    - Integer Bitwise
    - Shift

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Math Operators

- JavaScript defines math operators that work on numbers - note that the plus (+) operator also works on strings
- JavaScript converts operands to numbers or the special value **NaN**
- Division by zero results in the special value **Infinity**

```
+    Addition
-    Subtraction
*    Multiplication
/    Division
%    Remainder
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Math Operators, cont'd

```
1    var i1 = 12;
2    var d1 = 12.34;
3
4    alert(i1 + d1);  // 24.34
5
6    var i2 = 0;
7
8    alert(i1/i2);    // Infinity
9
10   var i3 = "123";
11
12   alert(i3 * 2);  // 246
13
14   var i4 = "abc";
15
16   alert(i4 * 2);  // NaN
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other Math Operations

- The Math object defines additional math operations beyond the standard math operators (e.g. trig functions)
- These **methods** are not intrinsic JavaScript syntax

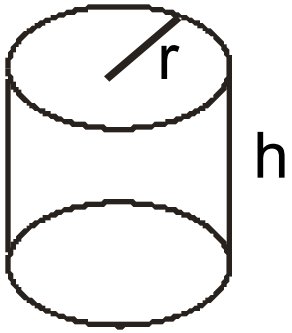
```
1    var d = Math.sin(Math.PI / 2);
2
3    alert(d);  // 1
4
5    var radius = 5.3;
6    var area = Math.pow(radius, 2) * Math.PI;
7
8    alert(area);  // 88.247
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Quick Practice

- The equation for the surface area of a cylinder is:

$$2\pi r^2 + 2\pi r h$$



- Complete this program fragment to calculate a cylinder's surface area:

```
1     var r = 12.0;
2     var h = 3.0;
3
4     var surfaceArea =
5
6     alert(surfaceArea);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Compound Assignment Operators

- JavaScript provides several compound assignment operators

```
*= /= %= += -= <<= >>= >>>= &= ^= |=
```

```
1    var i1 = 12;
2
3    i1 += 45; // same as i1 = i1 + 45;
4
5    alert(i1);  // 57
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Increment and Decrement Operators

- These shortcut operators come in two forms: **prefix** and **postfix**
- Postfix operators work after the expression is evaluated, prefix before

```
                    ++ --
```

```
1    var i1 = 12;
2    i1++; // same as i1 = i1 + 1;
3
4    alert(i1);  // 13
5
6    var i2 = --i1 * 3;
7
8    alert(i2); // 36
9
10   alert(i1); // 12
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Equality Operators

- Equality operators return a boolean result and are most often used in "if" statements or loops
- The **strict equality operator** === compares without performing a type conversion

```
                                == != ===
1     var i1 = 12;
2     var i2 = 14;
3
4     var b = i1 == i2;
5     alert(b);   // false
6
7     var x = "23";
8     var y = 23;
9
10    alert(x == y);  // true
11    alert(x === y); // false
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Comparison Operators

- The numeric comparison operators work on numbers and strings and return a boolean result

```
                < <= > >=
```

```
1     var i1 = 12;
2     var i2 = 14;
3
4     var b = i1 >= i2;
5     alert(b);  // false
6
7     var q = "abc";
8     var r = "abd";
9     alert(q < r); // true
10
11    var x = "Sam";
12    var y = "sam";
13    alert (x < y); // true
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Boolean Logical Operators

- You can use logical operators to build more complex testing expressions
- These operators work with booleans and "truthy" and "falsy" values

```
&& || !
```

```
1    var atHome = true;
2    var eatingDinner = false;
3
4    var result = atHome && eatingDinner;
5    alert(result);  // false
6
7    result = atHome || eatingDinner;
8    alert(result); // true
9
10   result = atHome && !eatingDinner;
11   alert(result); // true
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Conditional Operator

- The conditional operator is a shortcut for an if-else statement

```
? :
```

```
1    var i1 = 12;
2    var i2 = 14;
3
4    var min = ( i1 < i2 ) ? i1 : i2;
5    alert(min); // 12
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

String Concatenation

- You can use the + and += operators to concatenate strings

```
1    var s1 = "Hello";
2    var s2 = s1 + " world";
3
4    alert(s2); // Hello world
5
6    s1 += " Dolly";
7
8    alert(s1); // Hello Dolly
9
10   var s3 = s1 + 23;
11
12   alert(s3); // Hello Dolly23
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bitwise Operators

- You can use bitwise operators to test or manipulate bit patterns within integer values

```
& ^ | ~
```

```
1    var i1 = 18;
2    var i2 = i1 & 8;   // 10010 and 01000
3    var i3 = i1 | 8;   // 10010 or  01000
4    var i4 = i1 ^ 16;  // 10010 xor 10000
5    alert(i2);  // 0
6    alert(i3);  // 26
7    alert(i4);  // 2
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Shift Operators

- You can use the shift operators to move bits within integer values
- You can use shift operators to multiply and divide by powers of two

<< >> >>>

```
1    var i1 = 18;  // 10010
2    alert(i1 << 1); // 18 * 2 = 36
3    alert(i1 >> 2); // 18 / 4 = 4
4    var i2 = -18;
5    alert(i2 <<  1); // -18 * 2 = -36
6    alert(i2 >>  2); // -18 / 4 = -5
7    alert(i2 >>> 2); // 1073741819
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Type Conversion

- JavaScript is a weakly typed language and will often convert variables between types "on the fly" during operations
- Depending on the operator, data will be converted from string to number or vice versa

```
1    var a = 9 + "hello";   // "9hello" (string)
2    var b = "3" * "4";     // 12 (number)
3    var c = 3 * "4";       // 12 (number)
4    var d = 2 - "abc";     // NaN
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Quiz

- What is the value of z in each script?

```
A:
  var x = 2;
  var y = "4";
  var z = x * y;
B:
  x = 2;
  y = "4";
  z = x + y;
C:
  x = "2";
  y = "4";
  z = x + y;
D:
  x = true;
  y = "4";
  z = x + y;
E:
  x = true;
  y = 4;
  z = x + y;
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other Operators

in	Tests if an object contains a property
delete	Removes an object property or array element
instanceof	Tests if an object is in a class
typeof	Returns type name of a value
eval()	A function that dynamically executes code

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About all of the JavaScript operators
- How to create expressions that you can use in "if" statements and loops

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Flow Control

- Conditional Statements
- Looping

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Blocks

- Some JavaScript facilties, for example loops and functions, use **blocks** defined by a pair of curly braces
- You can nest blocks within other blocks

```
1    function myMethod(var x)
2    {
3       if (x == 12)
4       {
5          // do something interesting
6       }
7    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Conditional Statements

- Like all complete programming languages, JavaScript provides syntax for making decisions:
  - if
  - Conditional (ternary) operator

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The if Statement

- The *if* statement examines an arbitrarily complex boolean expression for "truthiness"

```
if ( <expr> )
{
  // statements
}
```

```
if ( <expr> )
  // one statement
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The if else Statement

- The *if else* statement examines an arbitrarily complex boolean expression

```
if ( <expr> )
{
  // statements
}
else
{
  // statements
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The if else if Statement

- The *if else if* statement examines an arbitrarily complex boolean expression

```
if ( <expr> )
{
  // statements
}
else if ( <expr> )
{
  // statements
}
else
{
  // statements
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The if else if Statement, cont'd

```
1    var a = Math.random();
2    var b = Math.random();
3    var c = Math.random();
4
5    if ( (a + b) > c )
6    {
7      alert( "Sum is greater" );
8    }
9    else if ( (a+b) == c )
10   {
11     alert ( "The same" );
12   }
13   else
14   {
15      alert( "Sum NOT greater" );
16   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Quick Practice

- Assume that a Web page has an input box with an id of **input** and that you can obtain the characters in the input box using the **value** property. Write code that examines the text entered and if the text is "sam", display an alert "Hello, sam". Otherwise display an alert "Who are you?".

```
var input = document.getElementById("input");
var text = input.value;
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The while Loop

- The *while* loop evaluates a boolean expression and stays in the loop as long as the express is "truthy"

```
while ( <expr> )
{
  // statements
}

do
{
  // statements
}
while ( <expr> );
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The while Loop, cont'd

```
1    var i = 0;
2    while ( i < 10 )
3    {
4      console.log( "in while: " + i );
5      i++;
6    }
7
8    do
9    {
10     console.log( "in do-while: " + i );
11     i++;
12   } while ( i < 10 );
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The for Loop

- The *for* loop is a convenient and concise form of the *while* loop

```
// form #1
for ( <init statement(s)>;<expr>;<exec statement(s)> )
{
  // statements
}


// form #2
for ( variable in object )
{
  // statements
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The for Loop, cont'd

```
1     for ( var i = 0; i < 10; i++ )
2     {
3       console.log( "for loop: " + i );
4     }
5
6     for ( var i = 10,j = 0; i > 0; i--, j++ )
7     {
8       console.log( "(" + i + "," +
9                                   j + ")" );
10    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The for Loop, cont'd

```
1    var i = 0;
2    for ( ; i < 10; )
3    {
4      console.log( "for loop: " + i );
5      i++;
6    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Breaking Out of a Loop

- Sometimes it's useful to exit a loop prematurely using the *break* statement

```
1    while ( true )
2    {
3      var i = Math.random();
4      console.log( i );
5      if ( i < 0.5 )
6        break;
7    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The switch-case Statement

- The *switch-case* statement lets you choose alternatives based on an expression's evaluation
- JavaScript uses the === operator to determine which case executes

```
switch ( expression )
{
  case expression:
    // statements
    break;
  case expression:
    // statements
    break;
  default:
    // statements
    break;
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The switch-case Statement, cont'd

```
1    // get a random number between 0-100 (inclusive)
2    var a = Math.floor(Math.random()*101);
3    console.log(a);
4
5    switch ( a )
6    {
7      case 1:
8        console.log( "Value is One!" );
9        break;
10
11     case 42:
12       console.log( "Value is 42!" );
13       break;
14
15     default:
16       console.log( "Value is something else" );
17   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The switch-case Statement, cont'd

```
1    function checkChar()
2    {
3      var input = document.getElementById("input");
4      var c = input.value;
5      switch ( c )
6      {
7        case 'd':
8        case 'D':
9          console.log(new Date());
10         break;
11
12       case 'q':
13       case 'Q':
14         alert("User wants to quit");
15         break;
16
17       default:
18         console.log("What the heck?");
19     }
20   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Quick Practice

- Write a loop that counts from 10 to 100 by fives and displays each value:

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of Java conditionals
- About looping contructs

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Events

- Registering for Events
- Event Bubbling

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaScript Events

- JavaScript defines several **events** that the browser triggers when the user interacts with page elements or for page lifecycle events
- You can connect JavaScript functions to an event
- Events comprise a string **name** and a **target**
- One way to register a handler function is to specify it as a so-called "on" attribute on a DOM element

```
1    <input type="text" onfocus="myfunction()">
2
3    <script>
4    function myfunction()
5    {
6      // respond to event
7    }
8    </script>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Common JavaScript Events

- Browsers define many event types, including:

Event	Description
submit	The user pressed a form's submit button
focus	The user tabbed to or from an input widget
blur	
click	The user clicked on a widget
change	The user changed the content of a widget

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Events Example

```
1    <html>
2      <head>
3        <script>
4           function confirmSubmit()
5           {
6              var ok = confirm("Really submit?");
7              return ok;
8           }
9        </script>
10     </head>
11     <body>
12       <form onsubmit="return confirmSubmit()">
13         <input type="Submit">
14       </form>
15     </body>
16   </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Javascript to Validate a Form

```
1    <html>
2      <head>
3        <script>
4           function validateMyForm()
5           {
6             var theForm = document.MyForm;
7             var email = theForm.email.value;
8             var ok = true;
9             if (email.length == 0)
10            {
11              ok = false;
12              alert ("Email must not be empty");
13            }
14            return ok;
15          }
16       </script>
17     </head>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Validating a Form, cont'd

```
18   <body>
19       <form action="http://localhost/test"
20             name="MyForm"
21             onsubmit="return validateMyForm()">
22         Email: <input type="text" name="email"><br>
23         <input type="Submit">
24       </form>
25     </body>
26   </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Calling addEventListener

- Another way to register an event handler is to call **addEventListener()** on the event target
- This keeps the HTML JavaScript-free

```
1    <input type="text" id="myinput">
2
3    <script>
4      // find the element
5      var myinput = document.getElementById("myinput");
6
7      // register event handler
8      myinput.addEventListener("focus", function(evt)
9      {
10       // respond to event
11     }, false);
12   </script>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Event Bubbling

- Some events, including DOM events, propagate, or "bubble up" the ancestor hierarchy
- This lets you attach a handler to a parent element, e.g. a form, and handle all events for its children
- The handler can use the event object's **target** property to determine which child fired the event

test-events.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test Events</title>
  </head>
  <body>
   <form id="myForm">
     <button id="button1">Button 1</button>
     <button id="button2">Button 2</button>
   </form>

   <script>
     var form = document.getElementById("myForm");

     // last parameter of false means this is a
     // "non-capturing" event handler
     form.addEventListener("click",
       function(evt)
       {
           // print reference to a button
           console.log("target: " + evt.target);

           // print reference to the form itself
           console.log("Current target: " + evt.currentTarget);

           var id = evt.target.id;
           alert("You clicked on " + id);

       }, false);
   </script>
  </body>
</html>
```

Issues with JavaScript Events

- Not all browsers support addEventListener(), especially older ones such as IE 8 and below
- The **event object** that you can optionally specify in the event-handler function varies across event types and browsers
- These issues sometimes cause developers to use a JavaScript library such as jQuery or the Dojo Toolkit - such libraries provide cross-browser support for events

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to register for and handle events
- About event bubbling

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Objects

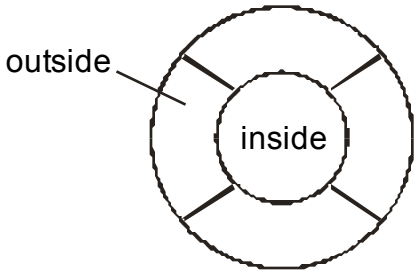
- What is an Object?
- Properties
- Prototypes
- Constructor Functions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is an Object?

- In classic object orientation, an **object** is a set of state (data) and behavior (code)
- JavaScript provides an interesting and powerful way for programs to work with objects
- JavaScript does not have **classes** in the same sense as other languages like Java, but does support inheriting state and behavior via a **prototype** mechanism

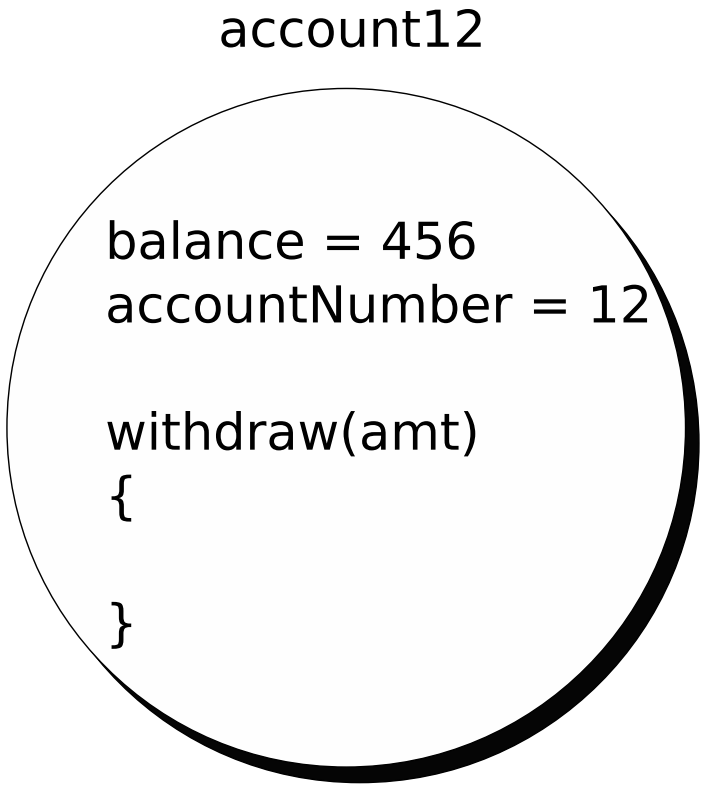


Object Name	Sample State	Sample Behavior
lassie	weight	bark
account12	balance	debit
policy765	amount	payOut

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Objects

- An **object** is a collection of **properties**, each of which has a name and a value
- One kind of property is a **method**, which is a function associated with the object



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Categories of Objects

- **Built-in** (native) objects such as arrays and functions
- **Host** objects such as those provided by a browser (*document*, *window* and so forth)
- **User-defined** objects created by programs

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What Are Properties?

- Properties consist of a name and value where the name is unique in the object
- Methods are really just a special case of a property - they are properties that associate a name with a set of statements
- Unlike in many languages, JavaScript lets you add/delete properties to objects dynamically

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating Objects

- JavaScript provides three ways to create new objects:
    - Using **object literal** syntax
    - Using the **new** operator
    - Using the ECMAScript v5 Object.create() function

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Object Literal Syntax

- JavaScript provides a special syntax so that you can create user-defined objects

```
1    var nothing = {};
2    var person =  {
3      name: "Sue",
4      age:  32
5    };
6    var book = {
7       title: "101 JavaScript Tricks",
8       author: {
9          name: "Paul Westerberg",
10         email: "paul@westerberg.com"
11      }
12   };
13   alert("Person name: " + person.name);
14   alert("Book author email: " + book.author.email);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using the new Operator

- You can use the **new** operator to create empty objects

```
1    var nothing = new Object();
2    var person = new Object();
3    person.name = "Sue";
4    person.age = 32;
5
6    var book = new Object();
7    book.title = "101 JavaScript Tricks";
8    book.author = new Object();
9    book.author.name = "Paul Westerberg";
10   book.author.email = "paul@westerberg.com";
11
12   alert("Person name: " + person.name);
13   alert("Book author email: " + book.author.email);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Objects Are Reference Types

- The value returned from creating an object is a **reference**, or pointer to the actual object
- You can define aliases to the object

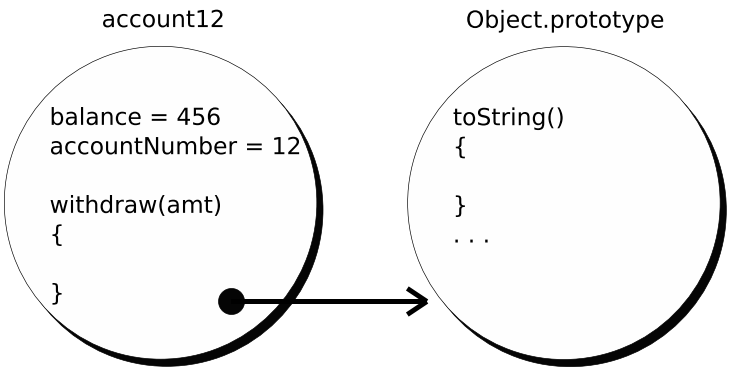
```
1    var sue =
2    {
3      name: "Sue",
4      age:  32
5    };
6
7    var sam = sue;
8    sam.age = 44;
9
10   alert("Sue age: " + sue.age); // Sue age: 44
11   alert("Sam age: " + sam.age); // Sam age: 44
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Prototype Inheritance

- Most JavaScript objects have another object chained to them - the **prototype**
- So far, the prototype has been **Object.prototype**
- Objects inherit properties/methods from their prototype - Object.prototype defines several methods like **hasOwnProperty()**, **toString()** and so forth

```
1    var nothing = new Object();
2    alert("nothing.toString(): " + nothing.toString());
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Working with Properties

- You can read or write properties using either the dot (.) or square bracket operators
- Using square brackets is more typing, but more powerful:
    - You can have property names that are JavaScript keywords (e.g. *for*) or contain spaces
    - You can use a string variable

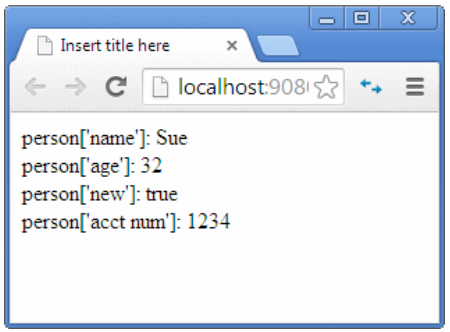
```
1    var person = new Object();
2    person.name = "Sue";
3    var myvar = "age";
4    person[myvar] = 32;
5    person['new'] = true;
6    person["acct num"] = 1234;
7    alert(person.age);  // 32
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The for-in Lop

- JavaScript provides a special looping syntax that lets you enumerate an object's properties

```
1     for (prop in person)
2     {
3       document.write("person['" + prop + "']: " +
4           person[prop] + "<br>");
5     }
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Global Object

- JavaScript provides a predefined, **global object** that contains global methods, properties, constructor functions and objects
- In browser-based JavaScript, the **window** object acts as the global object

```
1    alert("Hi #1");
2    window.alert("Hi #2");
3
4    alert("PI #1: " + Math.PI);
5    window.alert("PI #2: " + window.Math.PI);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deleting Properties

- JavaScript provides the **delete** operator that you can use to remove properties from objects

```
1    var sue =
2    {
3      name: "Sue",
4      age:  32
5    };
6
7    delete sue.age;  // remove age property
8
9    for (prop in sue)
10   {
11     document.write("sue['" + prop + "']: " +
12           sue[prop] + "<br>");
13   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Determining if a Property Exists

- You can use the **in** operator or the **hasOwnProperty()** method to check to see if an object has a given property
- The hasOwnProperty() method ignores inherited properties

```
1    alert(sue.hasOwnProperty("name"));  // true
2
3    alert("toString" in sue);           // true
4
5    alert(sue.hasOwnProperty("toString"));   // false
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Enumerating Properties

- We can treat objects as **hash tables**, adding and deleting properties on the fly
- The **for in** loop covered earlier does that, but also reports inherited properties and **methods**
- You can filter out all but non-method, "own" properties

```
1    for (prop in sue)
2    {
3      if(sue.hasOwnProperty(prop) &&
4         typeof sue[prop] !== "function")
5      {
6         document.write("sue['" + prop + "']: " +
7            sue[prop] + "<br>");
8      }
9    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Methods

- JavaScript **methods** are functions associated with an object
- Methods are really just a type of property - in JavaScript, you can pass functions around as if they were simple values
- Within a method, you use the **this** keyword to refer to the object's properties

```
1    var sue = {
2      name: "Sue",
3      age:  32
4    };
5
6    sue.ageInDogYears = function() {
7       return this.age * 7;
8    }
9
10   alert("Age in dog years: " + sue.ageInDogYears());
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Methods with Literal Syntax

- You can define methods using the literal syntax

```
1     var sue =
2     {
3       name: "Sue",
4       age:  32,
5       ageInDogYears: function()
6       {
7          return this.age;
8       }
9     };
10
11    alert("Age in dog years: " + sue.ageInDogYears());
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Method and Function Arguments

- It's useful to pass parameters to methods and functions
- Methods and functions can accept zero or more arguments

```
1     var sue =
2     {
3       name: "Sue",
4       age:  32,
5       ageMultiple: function(multiple)
6       {
7           return this.age * multiple;
8       }
9     };
10
11    document.write("Age multiple: " +
12       sue.ageMultiple(3));
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The this Keyword

- Within an object's methods, you use the **this** keyword to refer to its properties
- If you use **this** in a script outside of a method, or in a function not associated with an object, it's a synonym for the global object

```
1    alert("PI #1: " + Math.PI);
2    window.alert("PI #2: " + window.Math.PI);
3    this.alert("PI #3: " + this.Math.PI);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Constructor Functions

- The **new** operator invokes a **constructor function**
- Constructor functions and the **prototype** property let you create **classes** of related objects

```
1    // use a "native" constructor function
2    var birthday = new Date(1993, 0, 18);
3    document.write ("Birthday: " + birthday);
4
5    // use a user-defined constructor function
6    var sue = new Person("Sue", "Jones", 22, 120);
7    document.write("Age-weight: " + sue.ageWeight());
8
9    var bill = new Person("Bill", "Smith", 45, 185);
10   document.write("Age-weight: " + bill.ageWeight());
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

User-Defined Constructor Functions

```
1    // Person.js
2    function Person(firstname, lastname, age, weight)
3    {
4       this.firstname = firstname;
5       this.lastname = lastname;
6       this.age = age;
7       this.weight = weight;
8    }
9
10   Person.prototype =
11   {
12      ageWeight: function ()
13      {
14         return this.age * this.weight;
15      },
16      whatever:  function() {}
17   };
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How JavaScript implements object-oriented concepts
- How to create objects and add/remove properties
- How to use and define constructor methods

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Arrays

- What is an Array?
- Manipulating Arrays
- Arrays vs Objects

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is an Array?

- In JavaScript, an array is an ordered collection of values
- Arrays and objects are closely related - arrays are actually objects!
- If you use arrays with integer indexes, the indexes start at zero

0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	10	12	14	16	18

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating an Array

- You can create an array either using the square-bracket [] syntax or using the Array() constructor function

```
1    // empty array
2    var array1 = [];
3    // array with 10 integers
4    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
5    // array with 3 strings
6    var array3 = ["tom", "dick", "harry"];
7    // array with a mixture of types
8    var array4 = ["sue", 1, false];
9    // empty array
10   var array5 = new Array();
11   // array with space for 10 elements
12   var array6 = new Array(10);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Array Indexes

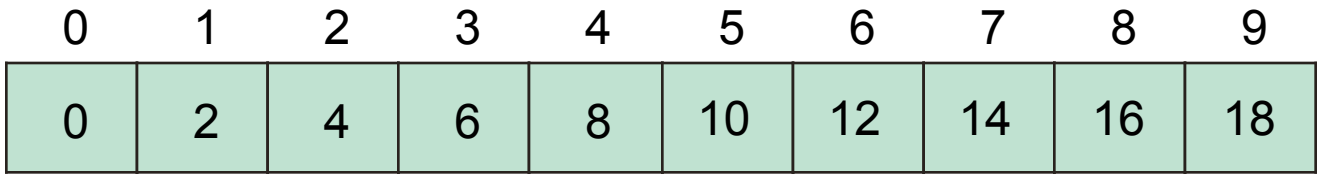
- There are basically two ways to treat an array:
  - Using consecutive non-negative integer indexes
  - Using arbitrary keys (arrays are objects, and thus can have properties)
- Using integer indexes is the "normal" way, and provides the best performance

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    array2[5] = 42;
3
4    var array7 = [];
5    array7['phone'] = "555-1212";
6    array7[-42] = 654;
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Iterating Arrays

- Arrays provide a **length** property that you can use to iterate the array




```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3    for(var i = 0; i < array2.length; i++)
4    {
5      document.write(array2[i] + " ");
6    }
7    // 0 2 4 6 8 10 12 14 16 18
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Appending Elements

- You can append elements by assigning using the **length** as an index or by calling the **push()** method

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3    array2[array2.length] = 20;
4    array2.push(22);
5
6    for(var i = 0; i < array2.length; i++)
7    {
8      document.write(array2[i] + " ");
9    }
10   // 0 2 4 6 8 10 12 14 16 18 20 22
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Inserting Elements at the Beginning

- You can use the **unshift()** method to insert one or more elements at the beginning of an array, moving existing elements back

```
1     var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3     array2.unshift(42);
4
5     for(var i = 0; i < array2.length; i++)
6     {
7       document.write(array2[i] + " ");
8     }
9     // 42 0 2 4 6 8 10 12 14 16 18
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deleting Array Elements

- Since arrays are objects, and array elements are properties, you can delete them in the same fashion as you can delete "normal" object properties
- If you delete in the middle, the array becomes **sparse**

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3    delete array2[3];
4
5    for(var i = 0; i < array2.length; i++)
6    {
7      document.write(array2[i] + " ");
8    }
9    // 0 2 4 undefined 8 10 12 14 16 18
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The splice() Method

- The **splice()** method is a powerful way to insert or remove elements from an array
- The method takes one to four arguments, and operates differently depending on how many you provide

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    var array3 = array2.splice(3);
3
4    document.write("array2: ");
5    for(var i = 0; i < array2.length; i++) {
6      document.write(array2[i] + " ");
7    }
8    document.write("<br>array3: ");
9    for(var i = 0; i < array3.length; i++) {
10     document.write(array3[i] + " ");
11   }
12   // array2: 0 2 4
13   // array3: 6 8 10 12 14 16 18
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using splice() to Insert

- By providing splice with a "delete count" of zero and additional elements, you can insert them into the array, moving existing elements back

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    var array3 = array2.splice(3, 0, 999, 888);
3
4    document.write("array2: ");
5    for(var i = 0; i < array2.length; i++)
6    {
7      document.write(array2[i] + " ");
8    }
9    document.write("<br>array3: ");
10   for(var i = 0; i < array3.length; i++)
11   {
12     document.write(array3[i] + " ");
13   }
14   // array2: 0 2 4 999 888 6 8 10 12 14 16 18
15   // array3:
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sparse Arrays

- You can create an array as **sparse**, or an array can become sparse if you delete elements in the middle
- Most JavaScript programs avoid sparse arrays

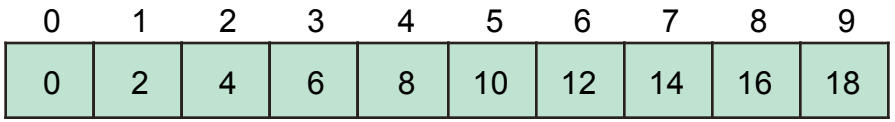
```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    delete array2[5];
3    //  0 2 4 6 8 undefined 12 14 16 18
4
5    var array8 = [];
6    array8[3] = 42;
7    // undefined undefined undefined 42
8
9    var array9 = new Array(3);
10   // undefined undefined undefined
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other Useful Array Methods

- **join()** - converts all elements to a string and returns it
- **reverse()** - used with indexed arrays, it reverses the element order
- **sort()** - sorts the array alphabetically or via a user-defined criteria
- **concat()** - creates a new array with additional elements appended to the original
- **slice()** - returns a subset of an array:

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    var array3 = array2.slice(4, 6);
3    // array3:  8 10
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Array-Like Objects

- Some objects that are not actually arrays can "impersonate" an array by providing a **length** property, so you can iterate them like arrays
- The DOM provided by Web browsers supplies several array-like objects

```
1    var divs = document.getElementsByTagName("div");
2
3    for(var i = 0; i < divs.length; i++)
4    {
5      alert("divs[" + i + "]=" + divs[i].innerHTML);
6    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Treating Strings as Arrays

- In many modern implementations, you can iterate strings as if they were arrays, thus avoiding having to explicitly call **charAt()**

```
1    var str = "You wished upon a star, " +
2      "but it turned into a plane";
3
4    for(var i = 0; i < str.length; i++)
5    {
6      document.write(" " + str[i]);
7    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The fundamentals of JavaScript arrays
- The differences between arrays and objects
- How to create, initialize and iterate arrays

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Functions

- Defining Functions
- Nesting Functions
- Closures

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Function?

- A **function** is a chunk of code that you define once but can call as many times as needed
- We refer to a function that is part of an object as a **method**
- Functions can accept **parameters** and optionally return a value

```
1    function sum(a, b)
2    {
3       return a+b;
4    }
5
6    var result = sum(5, 8);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Functions are Objects

- In JavaScript, functions are a type of **object**, and thus can be passed as parameters to another function or returned
- We can thus say that JavaScript functions are **first-class functions** as opposed to languages like Java (pre Java v8) in which functions (methods) can only be defined and invoked

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Functions

- JavaScript provides two syntaxes for defining functions:
    - Function declaration
    - Function definition expression
- The difference is that the **definition expression** syntax assigns the function object to a variable

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Declarations

- **Function declarations** let you define and name functions

```
1    // function declaration syntax
2    function function1(p1, p1)
3    {
4      alert(p1+p2);
5    }
6
7    function function2()
8    {
9      return "hi";
10   }
11
12   function1(2, 3);
13   var x = function2();
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Definition Expression

- **Function definition expressions** let you define a function, optionally name it and optionally invoke it immediately

```
1    // function definition expression
2    var function3 = function(p1) {return p1;}
3
4    // naming is useful for recursion
5    var f4 = function function4() {return function4};
6
7    function3(12);
8    var x = f4();    // references function4
9    // function4();  // not visible
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Definition Expression, cont'd

- You can use the function definition expression syntax to define and invoke **anonymous** functions on the fly

```
1    // define and invoke a function in one fell swoop
2    var f5 = function(){alert("hi");}();  // "hi"
3
4    // define and invoke an anonymous function
5    // note the extra set of parens
6    (function(p1){alert(p1);}(42));  // 42
7
8    // pass an anonymous function to another function
9    function function6(obj){alert(obj(72));}
10   function6(function(p1) {return p1;});  // 72
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Optional Arguments

- If a caller fails to provide an argument, JavaScript passes **undefined**
- You can use that implement optional function arguments

```
1    function myFunction(a, b, c)
2    {
3      b = b || 'bDefault';
4      c = c || 'cDefault';
5
6      console.log(a);
7      console.log(b);
8      console.log(c);
9    }
10
11   myFunction('a');
12   myFunction('a', 'b');
13   myFunction('a', 'b', 'c');
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Global Functions vs Methods

- Functions that are part of an object are known as **methods**
- JavaScript provides methods with an implicit **this** reference that refers to the object on which the method is running

```
1    var sue = {
2      name: "Sue",
3      age:  32
4    };
5
6    sue.ageInDogYears = function() {
7       return this.age * 7;
8    }
9
10   alert("Age in dog years: " + sue.ageInDogYears());
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Scope of Variables Review

- JavaScript supports **global** and **local** variables
- Variables defined within a function (method) are **local** and can only be used by that method

```
1    var a = "My global variable";
2
3    function function1()
4    {
5      var b = "My local variable";
6
7      alert(a);  // access global
8    }
9
10   function function2()
11   {
12     alert(a);  // access global
13     alert(b);  // error - out of scope
14   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Functions within Functions

- JavaScript programs can nest functions within other functions - the nested functions can access variables defined in the enclosing function

```
1     //  a nested function can access
2     //  variables defined by enclosing function
3       function f1() {
4          var x = 12;
5
6          function f2() {
7             alert(x);
8          }
9
10         f2();
11      }
12
13      f1();  // 12
14      f2();  // undefined
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Closures

- An enclosing function can return a nested function back to a caller, which can then invoke the nested function
- The nested function can still access local variables defined by the enclosing function, even if there's no call to the enclosing function - this is known as a **closure**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Closures, cont'd

```
1       function f1()
2       {
3          var x = 12;
4
5          function f2()
6          {
7             alert(x);
8          }
9
10         return f2;
11      }
12
13      // demonstrate a closure in action
14      var f2 = f1();
15      f2();              // 12
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Polluting the Global Namespace

- JavaScript defines global variables and functions as properties on the **global object**, which is the **window** object in browser-based JavaScript
- ECMAScript v5 and earlier provide no built-in support for handling conflicts if multiple JavaScript files or libraries use the same global names

```
1    // defines a global named myVar
2    <script src="libraryA.js"></script>
3
4    // also defines myVar, silently overwriting
5    // the global from libraryA
6    <script src="libraryB.js"></script>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Immediately Invoked Function Expressions

- To help alleviate global name pollution, you can define **immediately invoked function expressions** (IIFE)
- Using IIFEs, perhaps with closures, often organized into separate files, is a good practice in large-scale JavaScript development

```
1    // myfile.js
2
3    (function()
4    {
5      var myVar = 12;
6      var myFunction = function() {}
7      . . .
8    })();
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The fundamentals of JavaScript functions
- How to implement a **closure**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JavaScript window and document Objects

- Timers
- URLs and Navigation
- Dialog Boxes
- The Document Object Model (DOM)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JavaScript window Object

- The **window** object acts as the **global object** for JavaScript programs that run in a browser
- The window object provides many useful properties and methods

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Timers

- The **window.setTimeout()** method lets you start a **one-shot** timed event
- The **window.setInterval()** method lets you start a repeating event

```
1     var functionA = function()
2     {
3        alert("It's done!");
4        setInterval(functionB, 1000)
5     }
6
7     var functionB = function()
8     {
9        alert("I will annoy you!");
10    }
11
12    setTimeout(functionA, 3000);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

URLs and Navigation

- The window object defines a **window.location** property that contains the current browser URL
- You can modify the **location** object to cause the browser to navigate to a different URL

```
1    alert(window.location);
2
3    window.location="http://www.google.com";
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The navigator and screen Objects

- The **window.navigator** object provides information about the browser
- The **window.screen** object provides information about the user's display

```
   navigator Object
   ----------------
appName    Official browser name
appVersion Browser version
platform   Platform browser running on
userAgent  HTTP user-agent string
. . .

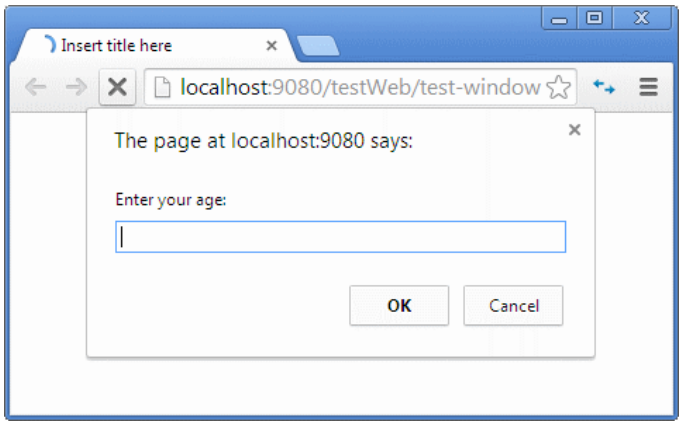
  screen Object
  -------------
width      Display pixel width
height     Display pixel height
pixelDepth Number of bits/pixel
. . .
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Functions for Dialog Boxes

- The window object provides three methods for displaying simple, **modal** dialog boxes:
  - **alert()** - displays a message
  - **confirm()** - displays a message and OK and Cancel buttons
  - **prompt()** - displays a message and an entry field

```
1    var txt = window.prompt("Enter your age:");
2    document.write("Prompt value: " + txt + "<br>");
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Opening and Closing Windows

- The window object provides the **open()** method so a script can open a new browser (popup) window
- The **close()** method lets a script close a previously opened window
- Use this capability for good, not evil!

```
1    window.open("test-arrays.html",
2       "newwindow",
3       "width=400,height=350,resizable=yes");
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JavaScript document Object

- The JavaScript **document** object lets scripts manipulate Web page elements using the **Document Object Model (DOM)** API
- The DOM is administered by the World Wide Web Consortium (W3C)
- The **document** object is a property on the **window** object

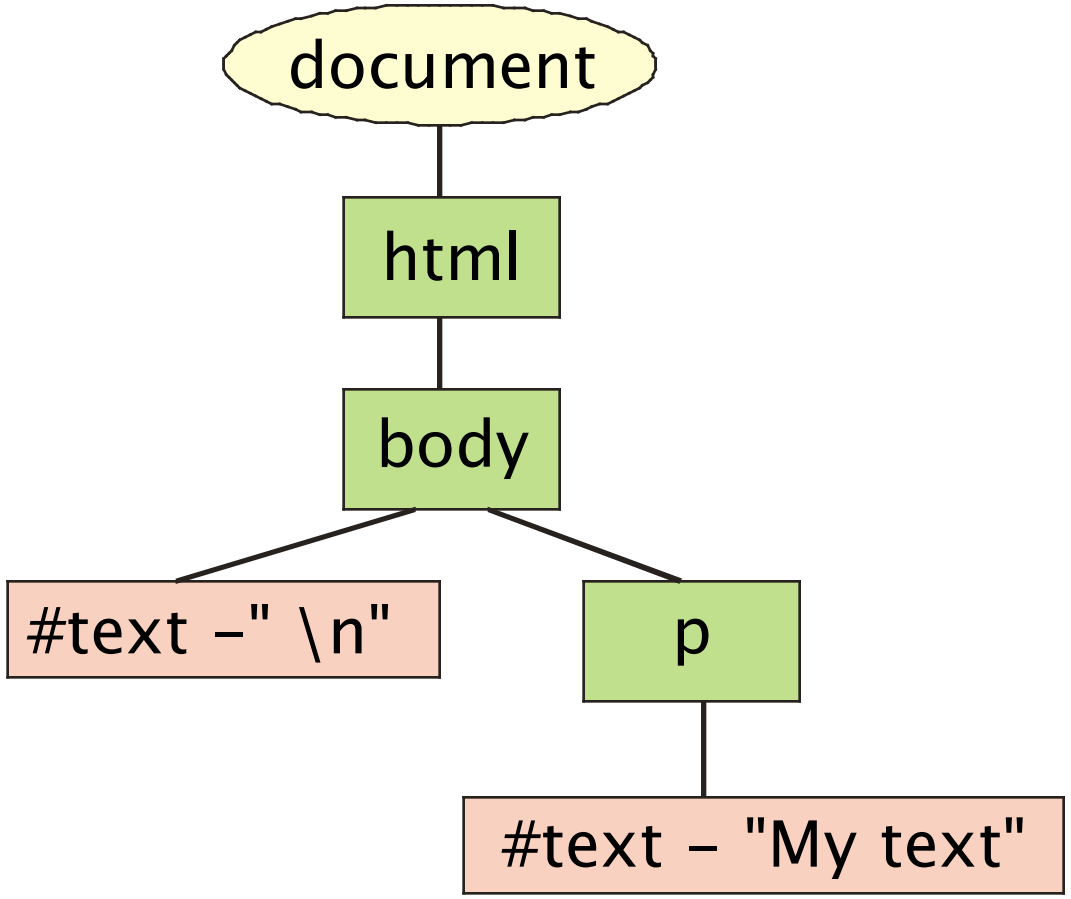


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The DOM Tree

- A well formed HTML document forms a tree, where all *nodes* except the root have a parent
- The DOM provides an API to traverse the tree, either up or down

```
<html>
  <body>
    <p>
      My text
    </p>
  </body>
</html>
```

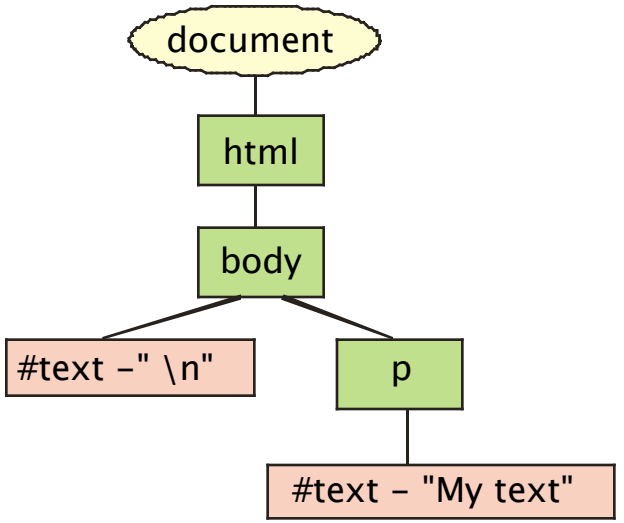


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Nodes

- The DOM API provides methods for traversing the tree and accessing nodes in the tree

```
1    var htmlEl = document.firstChild;
2    var bodyEl = htmlEl.lastChild;
3    var paraEl = bodyEl.childNodes[1];
4    var textNode = paraEl.firstChild.nodeValue;
5    var textNode1 = paraEl.innerHTML;
6
7    paraEl.firstChild.nodeValue="hello";
```

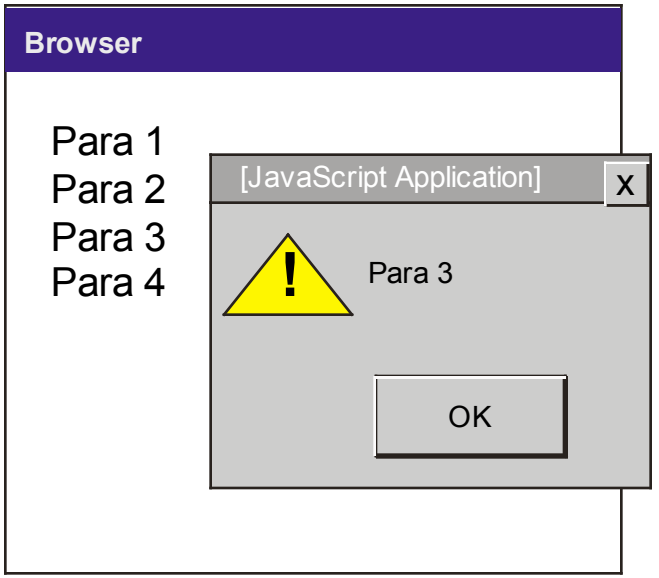


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Finding HTML Elements by ID

- HTML lets you assign unique IDs to individual elements

```
1    <html>
2      <head>
3        <script>
4            window.onload=function()
5            {
6              var p3 = document.getElementById("c");
7              alert(p3.innerHTML);
8            }
9        </script>
10     </head>
11     <body>
12       <p id="a">Para 1</p>
13       <p id="b">Para 2</p>
14       <p id="c">Para 3</p>
15       <p id="d">Para 4</p>
16     </body>
17   </html>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Finding Elements by Tag Name

- The document object lets you search the entire document to find all elements with a given tag name
- This call returns an **array-like** object that you can iterate

```
1    var paras = document.getElementsByTagName("p");
2
3    var theString = "";
4    for(var i = 0; i < paras.length; i++)
5      theString += paras[i].innerHTML;
6
7    alert(theString);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Finding Elements by CSS Class

- The document object lets you search the entire document to find all elements of a given CSS class
- This call returns an **array-like** object that you can iterate

```
1    var els =
2       document.getElementsByClassName("stylin");
3
4    var theString = "";
5    for(var i = 0; i < els.length; i++)
6      theString += els[i].innerHTML;
7
8    alert(theString);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Finding Elements using CSS Selectors

- The document object lets you search the entire document to find all elements that match a **CSS selector**
- This call returns an **array-like** object that you can iterate

```
1    // find all <p> elements that are direct
2    // children of the <body>
3    var els = document.querySelectorAll("body > p");
4
5    var theString = "";
6    for(var i = 0; i < els.length; i++)
7      theString += els[i].innerHTML;
8
9    alert(theString);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Element Content

- You can use the **innerHTML** property on DOM elements to read/write the element's contents
- If you modify it, the browser changes the displayed Web page

```
1     function changeIt()
2     {
3         var p2 = document.getElementById("b");
4         p2.innerHTML = "GOTCHA!";
5     }
6
7     setTimeout(changeIt, 5000);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Inserting Nodes

- The DOM provides methods to create nodes and insert them into the tree
- When you insert nodes, the browser changes the displayed Web page

```
1    function changeIt()
2    {
3      var body =
4         document.getElementsByTagName("body")[0];
5      var newpara = document.createElement("p");
6      newpara.innerHTML = "NEW PARA!";
7      body.appendChild(newpara);
8    }
9
10   setTimeout(changeIt, 5000);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Replacing or Deleting Nodes

- The DOM provides methods to remove and replace nodes
- When you delete or replace nodes, the browser changes the displayed Web page

```
1    function changeIt()
2    {
3      var body = document.getElementsByTagName("body")[0];
4      var p2 = document.getElementById("b");
5      body.removeChild(p2);
6
7      var newpara = document.createElement("p");
8      newpara.innerHTML = "NEW PARA!";
9      var p3 = document.getElementById("c");
10     body.replaceChild(newpara, p3);
11   }
12
13   setTimeout(changeIt, 5000);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to work with the JavaScript **window** object
- How to work with the JavaScript **document** object

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript and CSS

- CSS Overview
- Changing Styles Dynamically in a Script

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What are Cascading Style Sheets?

- Cascading style sheets (CSS) let you define **styles** that govern the appearance of elements in one or more HTML documents
- Major benefit: You can change the look of an entire Web site by modifying one file!

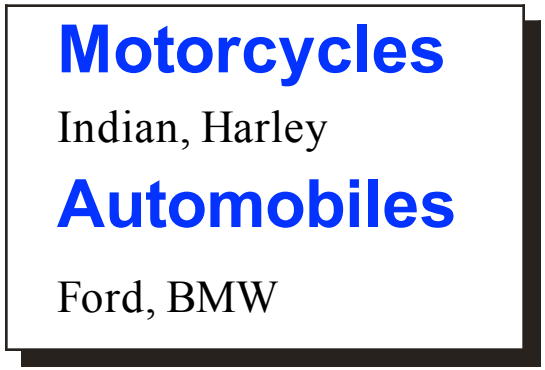
Style

```
h2
{
font-family: sans-serif;
color : Blue;
font-size : large;
font-weight : bold;
}
```

HTML

```
<h2>Motorcycles</h2>
<p>Indian, Harley</p>
<h2>Automobiles</h2>
<p>Ford, BMW</p>
```

Rendered HTML



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Styles

- There are three places where you can define styles:
  - In-line within a tag
  - In a page's **head** tag (document-level)
  - In an external file
- Using an external file lets you share the style definitions among multiple HTML files

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Basic Style Syntax

- To define a style, you write a **selector** and enclose the style's properties within curly braces
- Terminate each property with a semicolon (not really required for last property, but recommended)

```
1    /* This is a CSS comment */
2    h2
3    {
4      font-family:sans-serif;
5      color:blue;
6      font-size:large;
7      font-weight:bold;
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript and CSS

- You can write JavaScript that modifies HTML element styles dynamically
- This chapter covers how to modify **in-line** styles

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Changing an Element's CSS Class

- A **CSS class** lets you apply the same styles to different kinds of HTML elements
- One simple JavaScript/CSS technique is to change the CSS **class** of an element dynamically
- The DOM exposes the **className** attribute to let you manipulate an element's class

```
1    .class1 {color:red;}
2    .class2 {color:blue;}
3    . . .
4
5    <div class="class1>Div #1</div>
6    <p class="class1>Para #1</p>
7    <div class="class2">Div #2</div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Changing CSS Class Example

```
1    <style>
2      .class1 {color:red;}
3      .class2 {color:blue;}
4    </style>
5    <script>
6    function changeClass()
7    {
8      var theDiv = document.getElementById("div1");
9      theDiv.className = "class2";
10   }
11   </script>
12   . . .
13   <div id="div1" class="class1">
14      Div #1
15   </div>
16   <form>
17     <input type="button" value="Change style class"
18          onclick="changeClass();"><br>
19   </form>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The CSSStyleDeclaration Object

- To work on an element's individual CSS properties, JavaScript provides the **CSSStyleDeclaration** object, available via the DOM **style** property
- This object exposes CSS styles, some with slightly different names to meet JavaScript naming rules

```
1    var theDiv = document.getElementById("div1");
2    theDiv.style.color = "blue";
3    theDiv.style.fontSize = "24px";
4    . . .
5    <div id="div1" style="color:red">
6      Div #1
7    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Changing an Element's Style Example

```
1    <script>
2    function changeFontSizeStyle()
3    {
4      var theDiv = document.getElementById("div2");
5      theDiv.style.fontSize = "48px";
6    }
7    </script>
8    . . .
9    <div id="div2">
10      Div #2
11   </div>
12   <form>
13     <input type="button" value="Change font-size style"
14          onclick="changeFontSizeStyle();"><br>
15   </form>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Scripting Element Visibility and Display

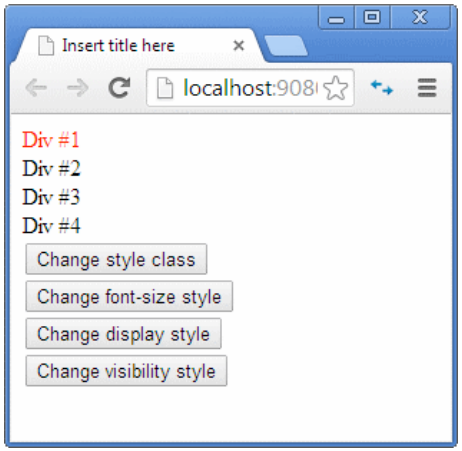
- CSS provides two different properties that control visibility
    - **visibility** - values: **hidden**, **visible**.
    - **display** - values: **none**, **block**, **inline** and so forth

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

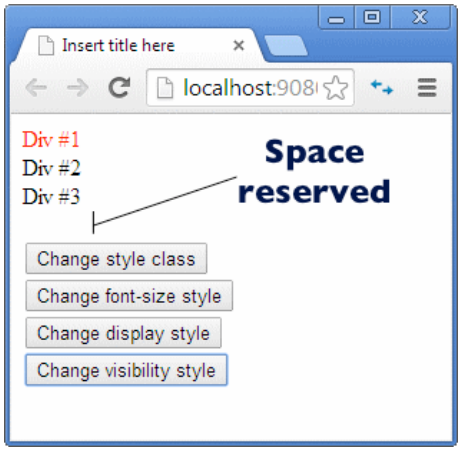
Changing the Visibility

- Setting **visibility** to *hidden* hides the element but keeps its space on the page

Before



After



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Changing the Visibility Example

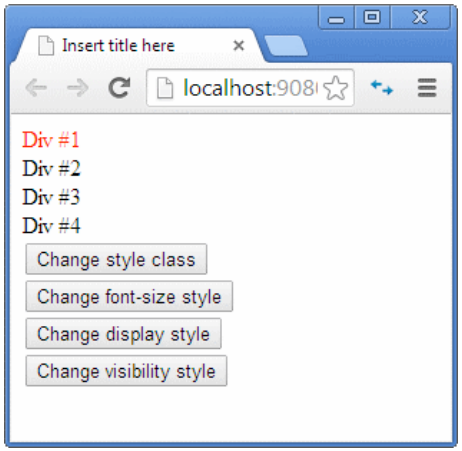
```
1     <script>
2     function changeVisibility()
3     {
4       var theDiv = document.getElementById("div4");
5       theDiv.style.visibility = "hidden";
6     }
7     </script>
8     . . .
9     <div id="div4">
10       Div #4
11    </div>
12    <form>
13      <input type="button" value="Change visibility style"
14           onclick="changeVisibility();"><br>
15    </form>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

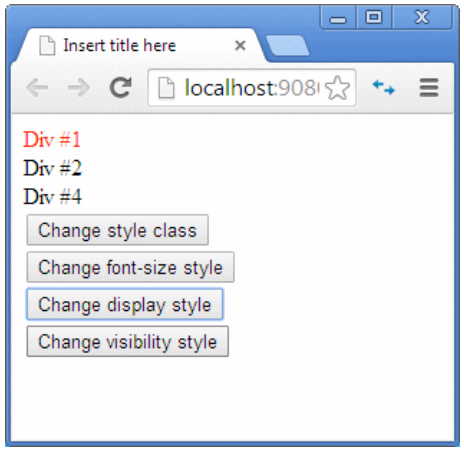
Changing the Display Style

- Setting **display** to *none* hides the element and reflows the page

Before



After



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Changing the Display Style Example

```
1     <script>
2     function changeDisplay()
3     {
4       var theDiv = document.getElementById("div3");
5       theDiv.style.display = "none";
6     }
7     </script>
8     . . .
9     <div id="div3">
10       Div #3
11    </div>
12    <form>
13      <input type="button" value="Change display style"
14           onclick="changeDisplay();"><br>
15    </form>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to manipulate CSS styles and classes from a JavaScript running in a browser

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JSON

- What is JSON?
- Using JSON in JavaScript
- Using JSON in Java

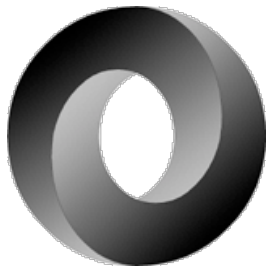
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is JSON?

- JSON is the **JavaScript Object Notation** and was originally syntax in JavaScript for initializing objects, but has now become a standard data format rivaling XML

```
{
  "id"  : 17,
  "name": "Sue Smith",
  "gpa":3.45
}
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Uses of JSON

- Natively in JavaScript for creating and initializing objects
- As a general-purpose data exchange format, especially for AJAX-based Web applications
- As a data format in RESTful Web services

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Basic JSON Syntax

- JSON has a simple, easy to read and understand syntax, both for humans and computers

```
1     -- sample object
2     {
3       "gpa"       : 3.33
4       "name"      : "Sue Smith",
5       "address"   :
6       {
7          "street":      "123 Elm",
8          "city"  :      "Marion"
9       }
10    }
11
12    -- sample array
13    [ "red", "green", "blue" ]
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using JSON in JavaScript

- JavaScript applications can use JSON to create and initialize objects and arrays

```
1    var rect1 =
2    {
3       upperLeft: { x: 2, y: 2 },
4       lowerRight: { x: 4, y: 5 }
5    };
6
7    var str = JSON.stringify(rect1);
8
9    var rect2 = JSON.parse(str);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using JSON in Java

- Java programs can easily parse or generate JSON strings:
    - Using basic Java String, regex and StringBuilder methods
    - Using a library, for example from www.json.org/java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using the json.org Java Library

```
1    package json;
2    import org.json.JSONObject;
3
4    public class TestJSON
5    {
6      public static void main(String[] args)
7        throws Exception
8      {
9        JSONObject rect = new JSONObject();
10       JSONObject upperLeft = new JSONObject();
11       JSONObject lowerRight = new JSONObject();
12       upperLeft.put("x", 2);
13       upperLeft.put("y", 2);
14       lowerRight.put("x", 4);
15       lowerRight.put("y", 4);
16       rect.put("upperLeft", upperLeft);
17       rect.put("lowerRight", lowerRight);
18       System.out.println(rect.toString());
19     }
20   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using JSON in a Java Web Application

- Web apps that use servlets/JSPs can use a JSON library to parse and generate JSON for AJAX or RESTful applications

```
1    public void doGet(...)
2    {
3       response.setContentType("application/json");
4       PrintWriter.out = response.getWriter();
5
6       JSONObject rect = new JSONObject();
7       . . .
8       out.print(rect.toString());
9    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of the JavaScript Object Notation
- How to use JSON in JavaScript and Java applications

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to jQuery

- What is jQuery?
- jQuery Fundamentals

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Do We Need a JavaScript Library?

- JavaScript is a powerful, object oriented language with relatively simple syntax but a complex API
- Since JavaScript is most often hosted by browsers, there are major incompatibilities between implementations
- Writing browser detection scripts is tedious and error prone
- Frameworks such as **jQuery**, **Prototype**, **Dojo** and similar aim to make JavaScript easier

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to jQuery

- **jQuery** is a browser-independent JavaScript library originally developed by John Resig, originally released in 2006
- jQuery is designed to make it easy to select (query) DOM elements for manipulation by a JavaScript program

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

jQuery Features

- An easy syntax for finding elements within the HTML document
- An cross-browser syntax for changing the DOM
- Provides Ajax support
- Defines a **plug in** architecture for writing extensions to jQuery

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

More Features

- Supports **method chaining** so you can execute a sequence of operations in one line of JavaScript
- Provides methods for working with all of the elements returned by a query so you don't need to iterate
- A small footprint - approximately 30KB, plus since jQuery is popular, there's a good chance that user's browsers will have a copy in cache

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Getting Started with jQuery

- Reference the jQuery library in a **script** tag in your document's **head**:
- You can download and reference a local copy of the jQuery.js file
- You can reference jQuery implementations hosted on the Web by jQuery, Microsoft, Google etc.
- Write jQuery queries in your scripts

```
1    <!-- local copy -->
2    <html>
3      <head>
4        <script src="jquery.js">
5        </script>
6      <head>
7    . . .
8    </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Basic jQuery Syntax

- jQuery uses **$(CSS selector)** as its basic syntax
- $() returns the **jQuery object** on which you can execute jQuery methods
- The jQuery object is an **array-like** object, so you can treat it like a normal array

```
1    var all = $("p");                  // all paragraphs
2    var myclass = $('p.myclass');      // all paragraphs with class
3    var anotherDiv = $('#mydiv');      // div(s) with ID "mydiv"
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Traversing Results of a Query

- The return from $() is a **jQuery object** which is an array-like object
- The jQuery object provides techniques to traverse the results, but there are subtle differences - some return a jQuery object, while others return standard DOM objects
- Explicitly traversing is not that common, since jQuery methods work on ALL of the results by default

```
1    alert($('p')[0]);      // returns a DOM HTMLParagraph object
2    alert($('p').get(0));  // returns a DOM HTMLParagraph object
3    alert($('p').first()); // returns a jQuery object
4    alert($('p').size());  // returns length using jQuery method
5    alert($('p').length);  // returns length using jQuery prop
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The jQuery ready() Function

- jQuery provides a function named **ready()** to which we can pass a user-written function - ready() will call the function when the browser has finished loading the DOM tree
- The ready() method provides a cross-browser technique so that we can execute scripts after DOM is loaded, but before the browser has downloaded page images

```
1    var function1 = function()
2    {
3      alert("Function1!");
4    }
5
6    $(document).ready(function1);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using an Anonymous ready() Function

- Instead of defining a named function, it's common to use an anonymous function, sometimes referred to as a **lambda function**:

```
1    $(document).ready(function()
2    {
3      alert("Lambda!");
4    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

jQuery Methods

- jQuery provides many **methods** for working on the returned array-like object
- In general, the methods operate on **all** of the returned elements and return the original jQuery object so you can chain calls:

```
$('p').removeClass("class1").addClass("class2");
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

jQuery Getters and Setters: Attributes

- jQuery provides methods that act as getters and setters - if you pass a single argument with a name, the method returns a value, but if you pass a name and a value, the method changes a property
- One such jQuery method is **attr()**, which lets you manipulate HTML element attributes

```
1    // query an attribute
2    alert($('#image1').attr("src"));
3
4    // change an attribute
5    $('#image1').attr("title", "A nice picture");
6    . . .
7    <img id="image1" src="greyhound.jpeg">
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

jQuery Getters and Setters: CSS Properties

- It's common for scripts to manipulate elements' CSS properties - jQuery provides the **css()** get/set method

```
1    // query a CSS property on a single element
2    alert($('p').first().css("font-weight"));
3
4    // modify a CSS property on all paragraphs
5    $('p').css("fontWeight", "bold");
6    . . .
7    <p>Para #1</p>
8    <p>Para #2</p>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

jQuery Getters and Setters: Element Content

- jQuery provides the **html** get/set method that let you manipulate HTML elements' HTML
- This is equivalent to using the DOM **innerHTML** property

```
1    // query an element's HTML
2    alert($('p').first().html());
3       // Some <strong>strong text</strong>
4
5    // modify an element's HTML
6    $('p').last().html("NEW TEXT");
7    . . .
8    <p>Some <strong>strong text</strong></p>
9    <p>Para #2</p>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Inserting or Deleting Elements

- jQuery provides easy-to-use methods so you can modify the document's structure

```
1    // remove first paragraph
2    $('p').first().remove();
3
4    // append a new paragraph after last
5    $('p').last().append("<p>NEW! IMPROVED!</p>");
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

jQuery Event Handlers

- jQuery makes it easy to attach **event-handling** functions to elements:

```
1    // when user tabs to any input box,
2    // thicken its border
3    $("input[type='text']").focus(function()
4    {
5      $(this).css("borderWidth", "4px");
6    });
7
8    // when user tabs away from any input box,
9    // reduce its border
10   $("input[type='text']").blur(function()
11   {
12     $(this).css("borderWidth", "1px");
13   });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of the jQuery library
- How jQuery lets you select, query, and modify the DOM tree

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to REST

- What is Representational State Transfer?
- Principles of REST
- Introduction to Java REST frameworks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is REST?

- **Representational State Transfer** is an architecture for distributed systems that emphasizes statelessness
- REST was introduced by Roy Fielding, who was one of the authors of HTTP

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

REST vs SOAP

- REST provides a straightforward, simple architecture for providing highly scalable services (note that the Web itself is RESTful)
- SOAP provides a more complex but perhaps more enterprise-class architecture that includes additional standards such as WSDL, WS-Security and so forth

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Principles of REST

- Every resource should have a unique ID
- Resources can have multiple representations
- Use links to connect resources
- Use standard HTTP methods
- Clients and servers communicate statelessly

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Resource IDs

- The Internet provides a resource-identification technique known as a **Uniform Resource Identifier** (URI), a.k.a URL
- RESTful applications use URIs to assign unique IDs to application resources
- Examples:
    - http://www.mydomain.com/students/5847
    - http://www.mydomain.com/students?id=44
    - http://www.mydomain.com/id=17

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Resources With Multiple Representations

- For flexibility, applications that provide data can support different data formats, e.g. HTML, XML, CSV, JSON
- Clients can then request the resource in the format they like best, perhaps by specifying an HTTP **accept** header

```
//XML representation
<student>
  <ID>17</ID>
  <name>Sue Smith</student>
  <gpa>3.45</gpa>
</student>

//JSON representation
{"student":{"id":17, "name":"Sue Smith, "gpa":3.45}}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Use Links to Connect Resources

- HTML supports **hyperlinking** so that resources can reference other resources
- You can also perform linking in other data formats such as XML

```
<student xmlns:xlink="http://www.w3.org/1999/xlink">
  <ID>17</ID>
  <name>Sue Smith</student>
  <gpa>3.45</gpa>
  <advisor
    xlink:href="http://www.mydomain.com/advisors/156"/>
</student>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Use Standard HTTP Methods

- HTTP supports **methods** that map to the four standard database Create, Retrieve, Update and Delete (CRUD) actions
- Note that only GET and POST can be sent directly from a browser so PUT, DELETE actions must be done programmatically (e.g. via AJAX)

HTTP Method	Database CRUD Operation
POST	CREATE
GET	SELECT (Retrieve)
PUT	UPDATE or CREATE
DELETE	DELETE

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Stateless Communication

- RESTful applications do **not** require the server to maintain communication state based on client requests
- This makes servers more scalable and allows the server to efficiently use techniques like caching, clustering and instance pooling

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

REST in Java

- JEE developers can use two basic approaches for writing RESTful services:
    - Write basic servlets
    - Use a REST framework
- Java RESTful frameworks include:
    - Restlets
    - JAX-WS REST
    - JAX-RS
    - Spring MVC REST

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Restlet?

- Restlet is a lightweight, open-source REST framework for Java
- Restlet applications can run within a servlet container or as standalone Java applications
- Restlet was created by Jerome Louvel in 2005

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Restlet Resource

```
1    public class HelloServerResource extends ServerResource
2    {
3        @Get
4        public String represent()
5        {
6            return "hello, world";
7        }
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-WS and REST

- JAX-WS is the standard Web services API for Java Enterprise Edition 5 and later
- JAX-WS provides support for creating Web services with no SOAP envelope, i.e. RESTful services

```
1    @WebServiceProvider(serviceName = "HWService")
2    @BindingType(value = HTTPBinding.HTTP_BINDING)
3    public class MyProvider implements Provider<Source>
4    {
5      public Source invoke(Source source)
6      {
7         String replyElement =
8            new String("<p>hello world</p>");
9         StreamSource reply = new StreamSource(
10           new StringReader(replyElement));
11        return reply;
12     }
13   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-RS

- JAX-RS is an API for providing RESTful services in Java
- JAX-RS went through the Java Community Process as JSR-311 and is a part of JEE Version 6



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A JAX-RS Service

- To use JAX-RS, you create **resource** classes, using annotations to configure them

```
1    @Path("/helloworld")
2    public class HelloWorldResource
3    {
4        @GET
5        @Produces("text/plain")
6        public String getMessage(
7                @DefaultValue("")
8                @QueryParam("name") String name)
9        {
10           return "Hello World, " + name;
11       }
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring MVC REST

- The popular Spring Framework's MVC module includes support for RESTful services

```
1    @Controller
2    @RequestMapping("rest")
3    public class StudentController
4    {
5      @RequestMapping(value = "/student/{id}",
6        method = RequestMethod.GET)
7        public @ResponseBody
8         Student getStudent(@PathVariable("id") int id)
9      {
10       Student s = findStudentByID(id);
11
12       return s;
13     }
14   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the fundamentals of representational state transfer
- The basics of Java REST frameworks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and REST

- Spring Support for REST
- Message Converters
- Annotations

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Support for REST

- Starting in Spring 3, Spring makes it easy to write SpringMVC **controllers** that work with REST resources
- If you are writing your application with Spring, than using Spring REST is easier and more natural than using JAX-RS

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

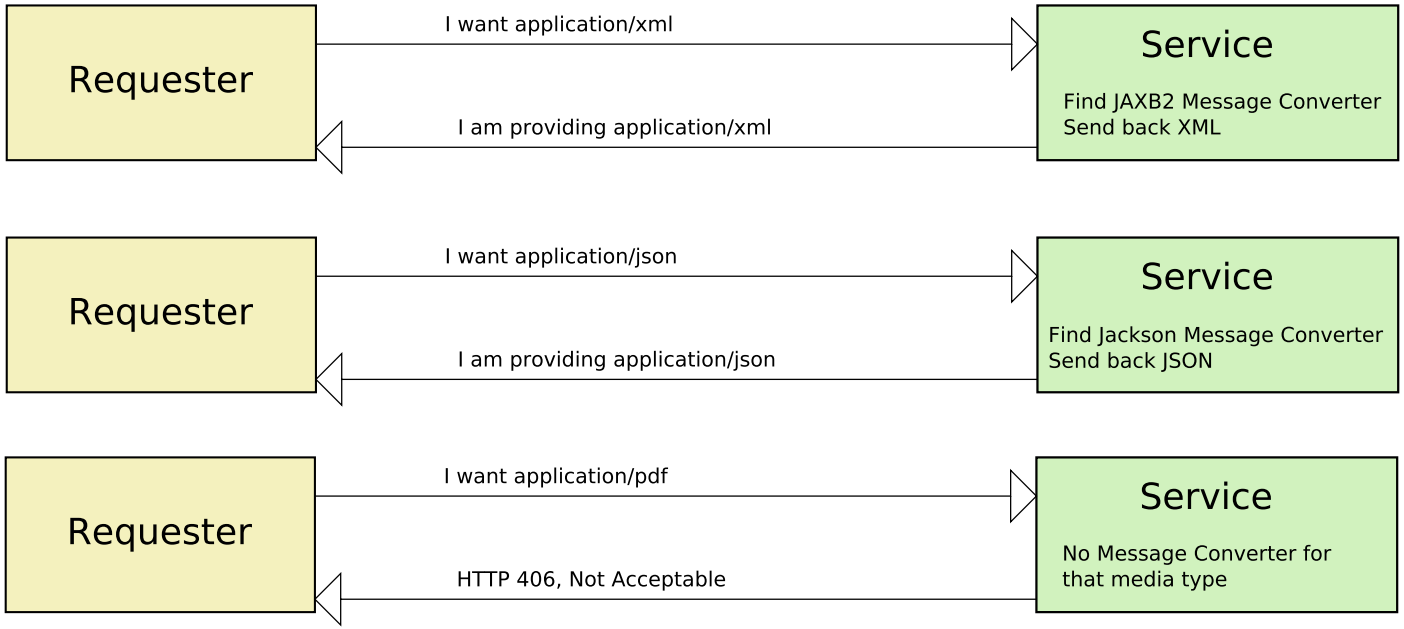
Message Converters

- Spring uses **message converters** to convert Java objects to/from a **resource representation**, including the following:
    - **SpringHttpMessageConverter** - transforms String objects to/from *text/plain*
    - **Jaxb2RootElementHttpMessageConverter** - transforms JAXB-annotated objects to/from *application/xml*
    - **MappingJacksonHttpMessageConverter** - transforms object to/from *application/json*

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation

- The client and server must agree on a data **representation**, which is usually a standard format such as text, XML or JSON using a **MediaType**
- Ultimately, the service decides on which representations it will provide



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Negotiation via HTTP Accept Header

- The HTTP specification allows for an **Accept** header, in which the user agent client provide a list of media types
- Spring parses the Accept header and then finds a message converter that can work with that media type

```
curl -H "Accept: application/xml, application/json"
 http://localhost:9080/TestSpringRESTWeb/rest/advisor/12
```

```
<!-- response -->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<advisor>
  <id>12</id>
  <name>Sue Joens</name>
  <salary>50000.0</salary>
</advisor>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring REST Annotations

- Spring provides several annotations to configure controllers to work with REST:
    - **@RequestMapping** - assigns a URL and an HTTP method to a processing method
    - **@ResponsStatus** - sets the HTTP status code
    - **@RequestBody** - transforms REST request into Java object(s)
    - **@ResponseBody** - transforms Java object into a REST response
    - **@PathVariable** - initializes a method parameter from part of the request URL
    - **@RequestParam** - initializes a method parameter from an HTTP *request parameter*

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple RESTful Controller

- This controller method returns either XML or JSON depending on the client's **Accept** header

```
1    @Controller
2    @RequestMapping("rest")
3    public class MyService
4    {
5      // respond to URLs like ..../rest/advisor/12
6      @RequestMapping(value = "/advisor/{id}",
7        method = RequestMethod.GET)
8      public @ResponseBody
9        Advisor getAdvisor(@PathVariable("id") int id)
10     {
11       Advisor a = findAdvisorByID(id);
12       if(a == null) throw new ResourceNotFoundException():
13       return a;
14     }
15   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Responding to POST Requests

- POST requests cause the REST service to create the resource and assign its RESTful URL

```
1    @RequestMapping(value = "/advisor",
2      method=RequestMethod.POST)
3    @ResponseStatus(HttpStatus.CREATED)
4    public @ResponseBody Advisor createAdvisor(
5      @RequestBody Advisor a, HttpServletResponse response)
6    {
7        int newID = insertAdvisor(a);
8        a.id = newID;
9
10       response.setHeader("Location", "/advisor/" + newID);
11       return a;
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Responding to PUT Requests

- PUT requests cause the REST service to update the RESTful resource

```
1    @RequestMapping(value = "/advisor/{id}",
2      method = RequestMethod.PUT)
3    public ResponseEntity<String> putAdvisor(
4      @PathVariable("id") long id, @RequestBody Advisor a)
5    {
6      ResponseEntity<String> retVal =
7        new ResponseEntity<String>(HttpStatus.NO_CONTENT);
8
9        boolean ok = updateAdvisor(a);
10
11       if(!ok)
12         retVal = new
13           ResponseEntity<String>(HttpStatus.NOT_FOUND);
14
15       return retVal;
16   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a Spring REST Application

- You configure a RESTful Spring application in the same fashion as for a "normal" Spring MVC application

```
1    <!-- springmvc-servlet.xml -->
2    <beans . . .>
3
4      <context:component-scan
5         base-package="com.goliath" />
6
7      <mvc:annotation-driven />
8    . . .
9    </beans>
```

MyConfig.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package com.goliath.config;
2
3    import org.springframework.context.annotation.ComponentScan;
4    import org.springframework.context.annotation.Configuration;
5    import org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;
6    import org.springframework.web.servlet.config.annotation.EnableWebMvc;
7    import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
8
9    @Configuration
10   @EnableWebMvc
11   @ComponentScan("com.goliath.services")
12   public class MyConfig extends WebMvcConfigurerAdapter
13   {
14       @Override
15       public void configureDefaultServletHandling(
16               DefaultServletHandlerConfigurer configurer)
17       {
18           configurer.enable();
19       }
20   }
```

Chapter Summary

In this chapter, you learned:

- How to use Spring MVC controllers, annotations and message converters to write RESTful CRUD services in Spring.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to AngularJS

- What is AngularJS?
- Data Binding
- Controllers and MVC
- AngularJS and Ajax

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is AngularJS?

- **AngularJS** is an Model-View-Controller (MVC) JavaScript framework for creating single-page Web applications
- AngularJS was originally developed as an internal Google project but is now a popular open-source project



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS vs jQuery

- **jQuery** is another popular JavaScript framework
- jQuery is all about manual DOM manipulation in scripts, while Angular is geared toward thinking about Web pages as **applications** with components and data binding
- AngularJS programmers typically don't need to write code to modify the DOM - Angular does that work for you





Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

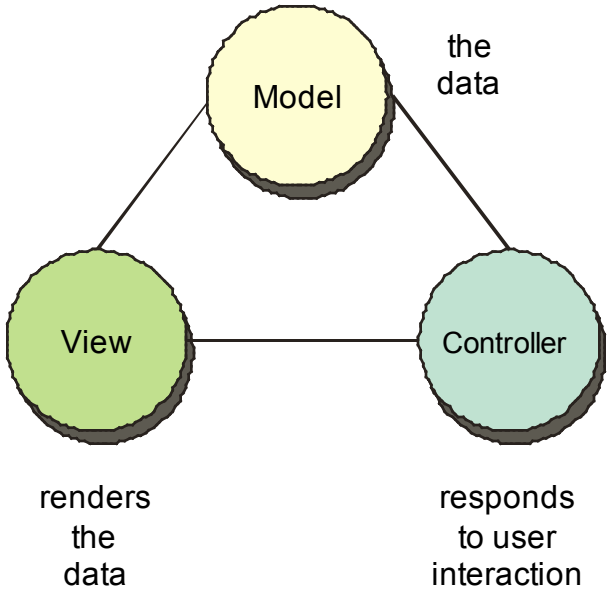
AngularJS Features

- MVC Architecture
- Two-Way Data Binding
- Easy Testing
- Dependency Injection

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is the MVC Architecture?

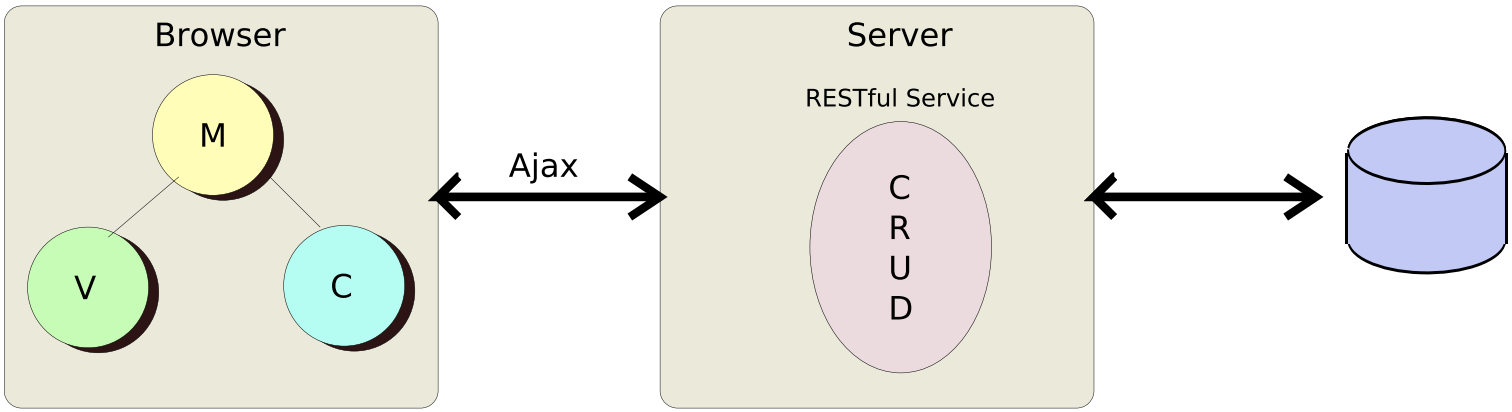
- MVC originated in the 1980s in Smalltalk as a technique to help build graphical user interfaces (GUIs) - AngularJS apps typically use some version of MVC
- MVC is effective since it decomposes applications into components that you can develop, debug and modify separately



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Typical AngularJS Architecture

- AngularJS promotes an architecture where the user interface is implemented totally on the client in JavaScript
- The client typically uses Ajax to request or update data on the server
- The server is typically organized as a series of RESTful services



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS Hello, World

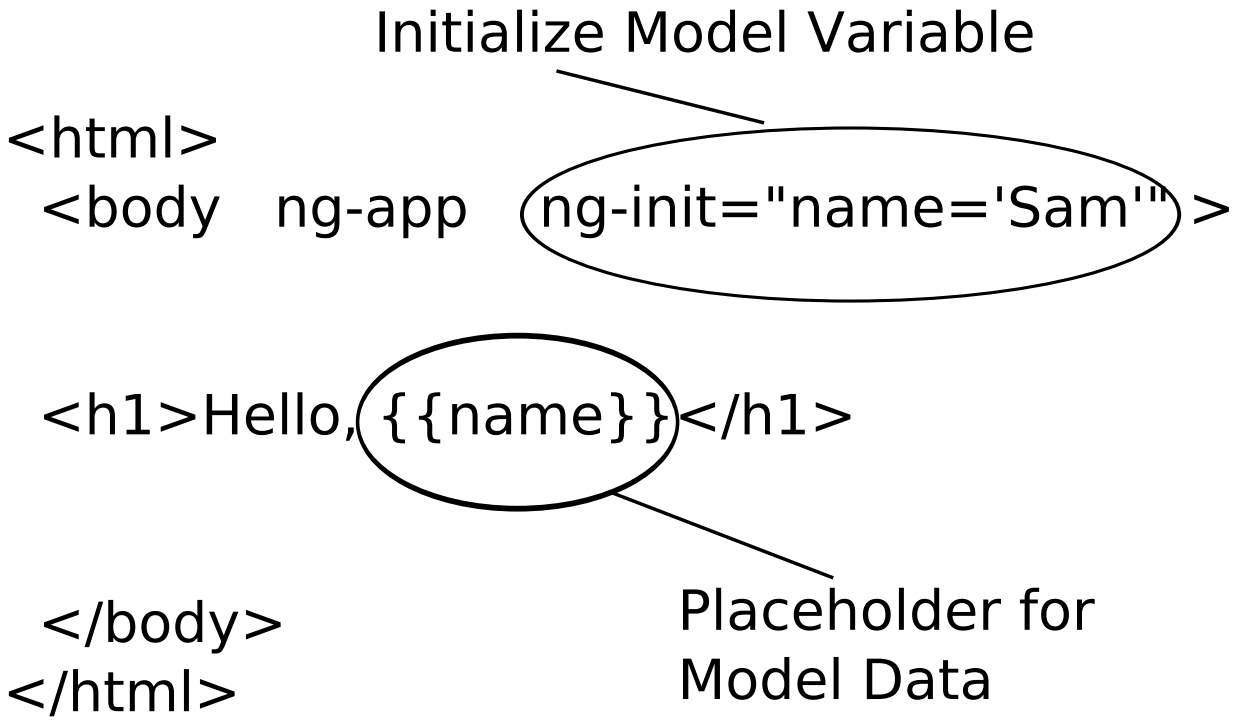
- A simple AngularJS application can use **directives** to define a variable in the **model** and an **AngularJS expression** to render it

```
1    <html>
2      <head>
3        <title>AngularJS Hello, World</title>
4        <script src="angular.js"></script>
5      </head>
6      <!-- ng-app bootstraps the app and creates
7           a container for the model. ng-init
8           intializes a model variable -->
9      <body ng-app ng-init="name='Sam'">
10       <h1>Hello, {{name}}</h1>
11     </body>
12   </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The View in AngularJS

- In AngularJS, we think of the HTML as being a template into which we insert dynamic content
- This is very similar in concept to JSPs used in JEE MVC Web applications



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Scope?

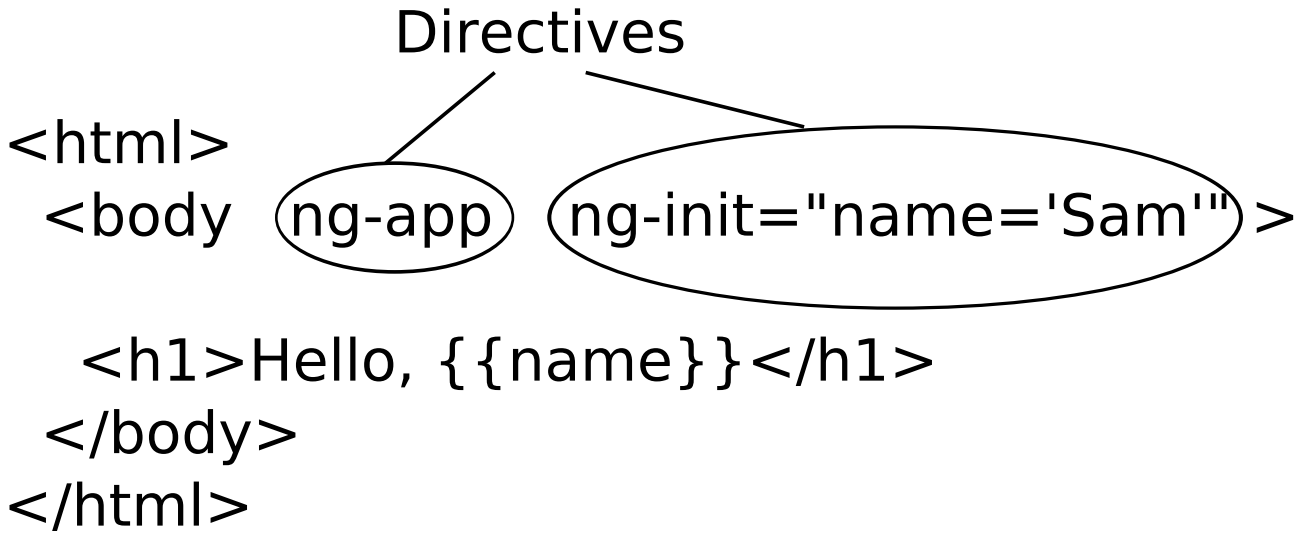
- An AngularJS **scope** is a container for model data and functionality (e.g. display logic)
- The **ng-app** directive bootstraps a **root scope** and **ng-init** lets you initialize it

```
1    <html>
2      <body ng-app ng-init="name='Sam'">
3        <h1>Hello, {{name}}</h1>
4      </body>
5    </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Directive?

- A **directive** is essentially just a JavaScript function that AngularJS applications use to extend HTML
- AngularJS supplies many directives (prefixed with **ng-**) and applications can create their own

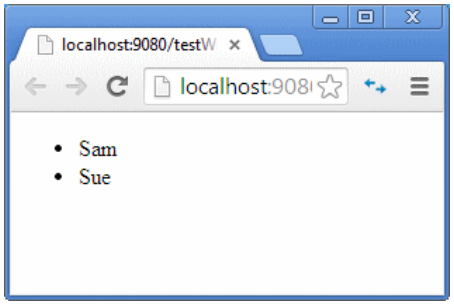


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Repeater

- A common Web-page job is to display repeating content, e.g. a list or table
- AngularJS provides the **ng-repeat** directive for that purpose

```
1    <html>
2      <head>
3        <script src="angular.js"></script>
4      </head>
5      <body ng-app ng-init="names = ['Sam','Sue']">
6        <ul>
7          <li ng-repeat="name in names">{{name}}</li>
8        </ul>
9      </body>
10   </html>
```

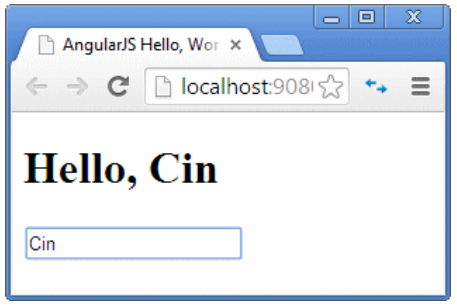


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Two-Way Data Binding

- The Hello, World example shown earlier demonstrating **binding** a placeholder to a model - it was a **one-way** binding since we just displayed the variable
- AngularJS supports **two-way** binding - we can bind a model property to an input box using the **ng-model** directive

```
1    <html>
2      <head>
3        <script src="angular.js"></script>
4      </head>
5      <body ng-app ng-init="name='Sam'">
6        <h1>Hello, {{name}}</h1>
7        <input type="text" name="name"
8           ng-model="name">
9      </body>
10   </html>
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Controller?

- For larger-scale applications, AngularJS provides the notion of a **controller** which is a JavaScript function that can define model variables and respond to user interaction

```
1    <html>
2      <head>
3        <script src="angular.js"></script>
4        <script>
5          var MyController = function($scope){
6             $scope.name="Sam";
7          };
8        </script>
9      </head>
10     <body ng-app>
11       <div ng-controller="MyController">
12         <h1>Hello, {{name}}</h1>
13         <input type="text" name="name" ng-model="name">
14       </div>
15     </body>
16   </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Controllers Define a Scope

- When you attach a controller to an element, AngularJS creates a scope and passes it to the controller's function, which can initialize the model
- Thus, this app doesn't need the **ng-init** directive

```
var MyController = function($scope)
{
  $scope.name="Sam";
};

<body ng-app>
 <div ng-controller="MyController">

   <h1>Hello, {{name}}</h1>
 </div>
</body>
```

AngularJS creates $scope, then calls controller function

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Module?

- The controller in the last example was simply a global JavaScript function, but in more complex apps, it's better to partition controllers into **modules**

```
1    <script>
2     // a module is a container for controllers
3     // and other AngularJS artifacts
4     var myModule = angular.module('MyModule', []);
5
6     myModule.controller('MyController', function($scope)
7     {
8        $scope.name="Sam";
9     });
10   </script>
11   <body ng-app="MyModule">
12   . . .
13   </body>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS and Forms

- AngularJS provides the **ng-submit** directive, which lets you attach a controller handler function to a form

```
1    myModule.controller('MyController', function($scope) {
2       $scope.age = 42;
3
4       $scope.doSomething = function() {
5         alert("Form submit, age: " + $scope.age);
6       };
7    });
8    . . .
9    <div ng-controller="MyController">
10     <form ng-submit="doSomething()">
11       Age: <input type="text" ng-model="age">
12       <input type="submit" value="Do Something">
13     </form>
14   </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS and Ajax

- AngularJS provides the **$http** *service* that lets you issue HTTP GET, POST, UPDATE and DELETE requests

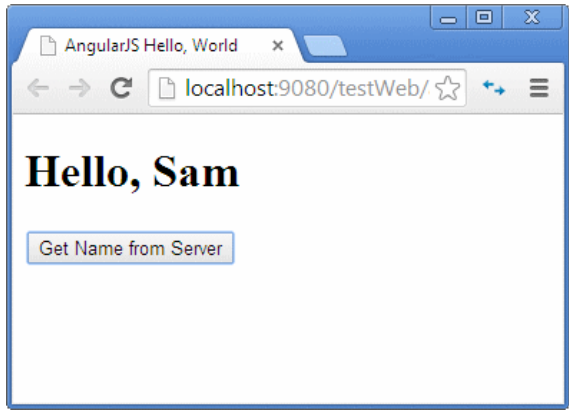
```
1    myModule.controller('MyController',
2       function($scope, $http) {
3     $scope.name="Placeholder";
4
5     $scope.getData = function() {
6       var response = $http.get("namedata.js");
7       response.success(function(data) {
8         $scope.name = data.name;
9       });
10
11      response.error(function() {
12        alert("Ajax error!");
13      });
14    };
15   });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Ajax Example

namedata.js

angular-ajax.html



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    {"name":"Sam"}
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>AngularJS Hello, World</title>
    <script src="angular.js"></script>
    <script>
      var myModule = angular.module('MyModule', []);

      myModule.controller('MyController', function($scope, $http)
      {
         $scope.name="Placeholder";

         // respond to form submit by making an Ajax request
         $scope.getData = function()
         {
           var response = $http.get("namedata.js");
           response.success(function(data)
           {
             $scope.name = data.name;
           });

           response.error(function(data, status, header, config)
           {
             alert("Ajax error!");
           });
         };
      });
    </script>
  </head>
  <!-- Need to specify module name on ng-app directive! -->
  <body ng-app="MyModule">
    <div ng-controller="MyController">
      <h1>Hello, {{name}}</h1>
      <form ng-submit="getData()">
        <input type="submit" value="Get Name from Server">
      </form>
    </div>
  </body>
</html>
```

Introduction to AngularJS 2

- AngularJS Version 2 is essentially a complete rewrite of the AngularJS code base, and will require significant changes to applications
- As of this writing, there is no concrete release date, but it's widely expected to be finished in 2016
- Major changes:
    - Target ECMAScript v6
    - Modular organization of features instead of a monolith
    - Mobile first design
    - Annotations

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The fundamentals of the AngularJS framework
- How use AngularJS data binding
- How to write modules and controllers
- How to access server data via Ajax

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Additional AngularJS Topics

- Modules
- Controller
- Services
- Directives
- Filters

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Module?

- A **module** acts as a container for controllers, services, directives and so forth
- You can attach a module to a DOM element using the **ng-app** directive

```
1    // index.html
2
3    <html ng-app="StudentInfo">
4    ...
5    </html>
```

```
1    // app.js
2    var myModule = angular.module('StudentInfo', []);
3
4    myModule.service("StudentModel", function($http)
5    {. . .});
6    myModule.controller("StudentController", function(StudentModel)
7    {. . .});
8    myModule.directive("studentEl", function()
9    {. . .});
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating a Module

- You create a new module, or retrieve an existing module by calling the **angular.module()** method
- The optional second argument lets you specify a list of **dependent modules** for the new module

```
1     // create a new module
2     var myModule = angular.module('myModule', []);
3
4     // retrieve an existing module's reference
5     var alsoMyModule = angular.module('myModule');
6
7     // explicitly pass a module reference to a new
8     // module
9     var anotherModule = angular.module('anotherMod',
10      ['myModule']);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using an IIFE

- To avoid polluting the global namespace, you can use an **immediately invoked function expression** to wrap your module

```
1    (function(angular)
2    {
3      var myModule = angular.module('myModule', []);
4
5      . . .
6
7    })(angular);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Controller?

- A **controller** is an object that can augment the AngularJS scope
- You can attach a controller to a DOM element using the **ng-controller** directive

```
1    // index.html
2
3    <section ng-controller="StudentController">
4      <div>{{currentStudent.name}}</div>
5    </html>
```

```
1    // app.js
2
3    myModule.controller("StudentController", function($scope)
4    {
5      $scope.currentStudent.name = "Sue";
6    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using $scope in a Controller (Or Not)

- When AngularJS first came out, apps used the predefined $scope variable to hold objects
- In AngularJS 1.1 or later, you can instead use the controller's reference for that purpose, i.e. **this**
- You must use the **as** syntax in HTML to assign a name to the controller

```
1    // app.js
2    myModule.controller("StudentController", function()
3    {
4      var sc = this;
5      sc.currentStudent.name = "Sue";
6    });
```

```
1    // index.html
2    <section ng-controller="StudentController as sc">
3      <div>{{sc.currentStudent.name}}</div>
4    </section>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Service?

- A **service** is an object that you can use to share code and data across your application
- Angular defines built-in services such as **$http** (for Ajax) and you can write your own
- Angular then can **inject** service references into the components that need them

```
1    // define a service
2    myModule.service('myService', function()
3    {
4        return 12;
5    });
6
7    // use a predefined service
8    var response = $http.get("namedata.js");
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Service As a Model

- You can define services that provide the data used by one or more controllers
- You can then use **dependency injection** to provide the service to each controller

```
1    myModule.service("StudentModel", function() {
2      this.getStudent = function() {
3        return {name:"Sue", studentID: 34, gpa: 3.44};
4      };
5    });
6
7    myModule.controller("StudentController",
8      function(StudentModel) {
9      var sc = this;
10     sc.currentStudent = StudentModel.getStudent();
11   });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Modules and Dependency Injection

- Modules can express dependencies on other modules and then use dependency injection to access components, e.g. services, defined in the dependent module

```
1    // module1.js
2    var myModule = angular.module('moduleOne', []);
3
4    myModule.service("myService", function() {
5    . . .
6    });
```

```
1    // module2.js
2    var myModule = angular.module('moduleTwo', ['moduleOne']);
3
4    myModule.controller("MyController", function(myService) {
5    . . .
6    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Minification

- Many JavaScript developers use a **minification** tool to reduce the size of scripts to improve performance
- The bad news is that Angular dependency injection depends on exact names, and minification changes (shortens) names

```
1    // uh-oh - what if minification name changes
2    // $http and StudentModel to something shorter?
3    myModule.controller("StudentController",
4      function($http, StudentModel)
5    {
6
7    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defeating Minification

- The trick is to specify the dependencies and the controller function itself in an **array** - minification tools will not alter array contents

```
1    myModule.controller("StudentController",
2      ['$http', 'StudentModel',
3         function($http, StudentModel)
4    {
5
6    }]);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Directive?

- A **directive** is an object that you can attach to a DOM element to add behavior or custom markup
- Angular provides many directives (e.g. **ngApp**) and you can create your own

```
1    <html>
2      <head>
3        <script src="angular.js"></script>
4      </head>
5      <body ng-app ng-init="names = ['Sam','Sue']">
6        <ul>
7          <li ng-repeat="name in names">{{name}}</li>
8        </ul>
9      </body>
10   </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Directive Name Matching

- In the HTML, you can specify directives using several syntaxes
- AngularJS **normalizes** the directive name and then attempts to find a matching directive
- The following syntaxes match some user-defined directive **myEl**:

```
1    <my-el></my-el>
2    <my_el></my_el>
3    <div my-el></div>
4    <div x-my-el></div>
5    <div data-my-el></div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Angular Expressions

- In the view, you can use **expressions** delimited with {{ }} as placeholders for model data
- Angular expressions are similar to normal JavaScript expressions - the main difference is that they evaluate against an Angular scope instead of the global **window** object
- You can use the **ngBind** directive as an alternative

```
1    <div x-ng-controller="StudentController as sc">
2      <div>Name: {{sc.currentStudent.name}}</div>
3      <div>GPA:
4        <span x-ng-bind="sc.currentStudent.gpa"></span>
5      </div>
6    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Filters

- In many cases, you want to limit or transform the model data while displaying it
- For example, display a subset of a domain-object collection or format a number as currency
- Angular supports **filters** for these purposes
- In the view, you use the pipe | symbol to invoke a filter and the colon : symbol to pass arguments

```
1    <div ng-controller="StudentController as sc">
2      <ul>
3        <li
4          ng-repeat="student in sc.students | orderBy:'gpa'">
5            Name: {{student.name}} GPA: {{student.gpa}}
6        </li>
7      </ul>
8    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Predefined Filters

- Angular provides several filters and you can create your own

Filter Name	Description
currency	Formats a number with two decimal places and currency symbol
date	Formats a date
lowercase	Changes a string's case
number	Formats a number
uppercase	Changes a string's case
json	Pretty prints JSON for debugging
filter	Subsets a collection based on a criteria
limitTo	Subsets a collection based on a count
orderBy	Orders a collection by a criteria

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Event Handling

- Native JavaScript programs handle events (e.g. click) by registering event handler functions
- In an Angular view, you can use directives instead

```
1    <div x-ng-controller="StudentController as sc">
2     <ul>
3      <li
4       x-ng-repeat="student in sc.students"
5       x-ng-click="sc.handleClick(student)">
6         Name: {{student.name}} GPA: {{student.gpa}}
7     </li>
8     </ul>
9    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Event Handling, cont'd

```
1    myModule.controller("StudentController",
2      function(StudentModel)
3    {
4      var sc = this;
5
6      sc.students = StudentModel.getStudents();
7
8      sc.handleClick = function(theStudent)
9      {
10       console.log("In handleClick, theStudent: " +
11         theStudent.name);
12     };
13   });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to define modules, controllers and directives
- About depdendency injection, filters and events

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Angular $http Service

- Introduction to Ajax
- Promises
- The $http Service

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is AJAX?

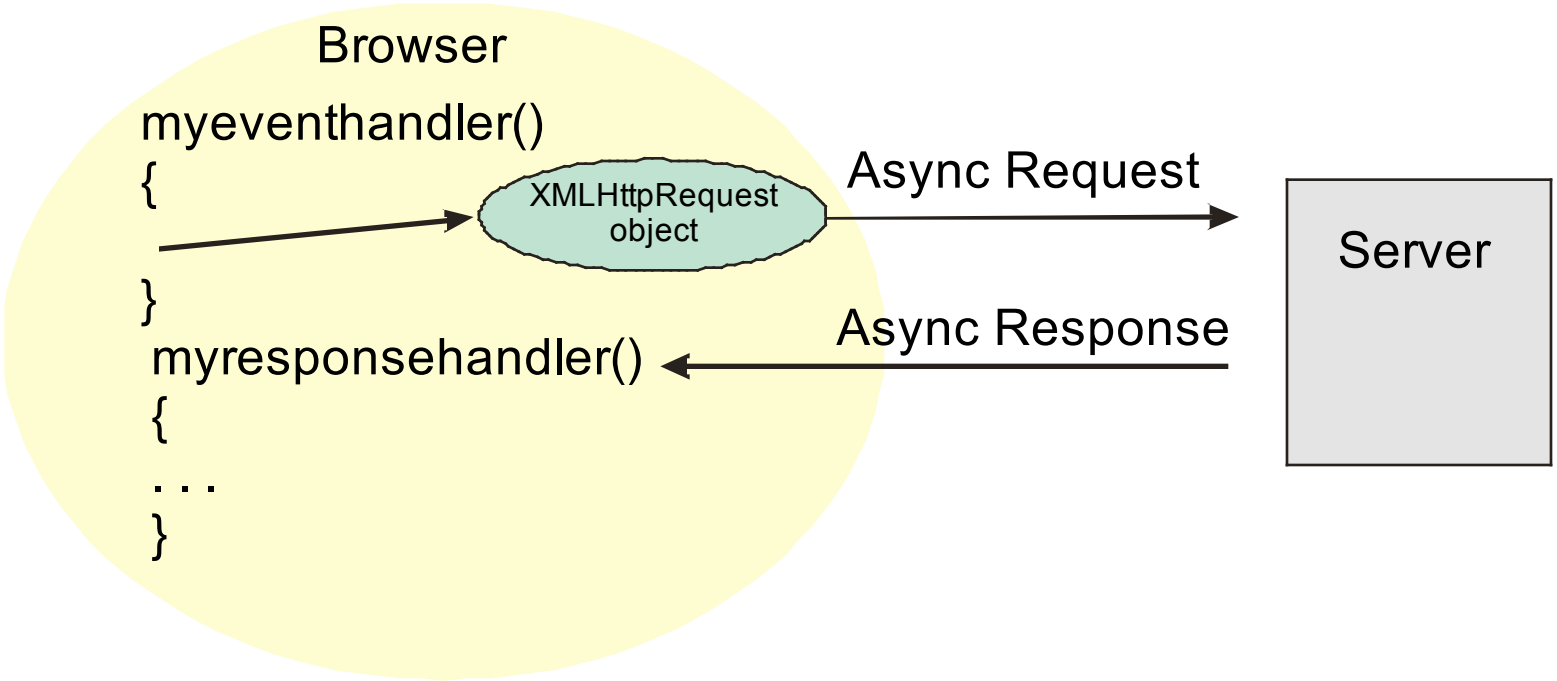
- AJAX (Asychronous JavaScript and XML) uses a combination of existing technologies:
  - JavaScript
  - The XMLHttpRequest Object
  - HTML Document Object Model
- AJAX lets you create Web applications with a higher degree of interactivity



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The XMLHttpRequest Object

- The **XMLHttpRequest** object was first implemented by Microsoft Internet Explorer, but is now a part of other browsers such as Firefox and Opera
- This object defines an API for making remote requests, either synchronously or asynchronously



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Standard JavaScript Ajax Example

standardAjax.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<html>
  <head>
    <title>Current Time and Date</title>
    <script>

      var req;

      function setDate(text)
      {
        var dateTxt=document.getElementById('dateTxt');
        dateTxt.firstChild.nodeValue=text;
      }

      function changeFmt(text)
      {
        var url = "GetDate?fmt=" + escape(text);
        //sendRequest (url, null, "GET");
        sendAsyncGetRequest(url);
      }

      function sendRequest(url,params,HttpMethod)
      {
        if (!HttpMethod)
        {
            HttpMethod="POST";
        }
        req=getXMLHttpRequest();
        if (req)
        {
            req.open(HttpMethod,url,true);
            req.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
            req.onreadystatechange = callback;
            req.send(params);
        }
    }

    function sendAsyncGetRequest(url)
    {
       req = getXMLHttpRequest();
       if (req)
       {
          req.open("GET", url, true);
          req.onreadystatechange = callback;
```

```
46          }
47      }
48
49      function callback()
50      {
51          if (req.readyState == 4)
52          {
53              if (req.status == 200)
54              {
55                  var text = req.responseText;
56                  setDate(text);
57              }
58              else
59              {
60                  alert("Problem: " + req.statusText);
61              }
62          }
63      }
64
65        function getXMLHttpRequest()
66        {
67          var xRequest=null;
68          if (window.XMLHttpRequest)
69          {
70              xRequest=new XMLHttpRequest();
71          }
72          else if (typeof ActiveXObject != "undefined")
73          {
74              xRequest=new ActiveXObject("Microsoft.XMLHTTP");
75          }
76          return xRequest;
77        }
78      </script>
79    </head>
80    <body>
81      <p>Choose format:</p>
82      <p><input type="radio" name="fmt" value="short"
83            onclick="changeFmt('short')">Short</p>
84       <p><input type="radio" name="fmt" value="medium"
85            onclick="changeFmt('medium')">Medium</p>
86       <p><input type="radio" name="fmt" value="long"
87            onclick="changeFmt('long')">Long</p>
88       <p id="dateTxt">No date</p>
89    </body>
```

Issues with Standard JavaScript Ajax

- Working with the XHR object directly is cumbersome, error-prone and requires a lot of code
- The XHR object itself is browser-specific, though modern browsers have converged on a common API

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS and Ajax

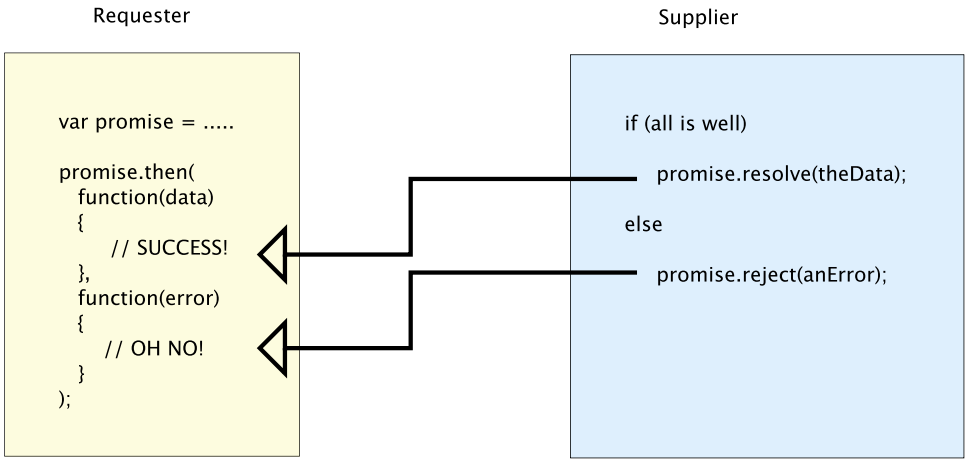
- AngularJS provides the **$http** *service* that lets you issue HTTP GET, POST, UPDATE and DELETE requests

```
1    myModule.controller('MyController',
2       function($scope, $http) {
3     $scope.name="Placeholder";
4
5     $scope.getData = function() {
6       var promise = $http.get("namedata.js");
7       promise.then(function(resp) {
8         $scope.name = resp.data.name;
9       },
10      function() {
11        alert("Ajax error!");
12      });
13    };
14   });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Promise?

- **Promises** is an emerging API for writing asynchronous code
- Many popular JavaScript frameworks, including AngularJS, implement this API
- A **promise** is an object that represents an asynchronous action
- The "supplier" calls **resolve()** or **reject()** to fulfill or reject the promise
- The "requester" calls **then()** to register one or two callbacks



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS and Promises

- AngularJS provides a **$q promise** service that makes it easier to work with asynchronous programming
- The **$http** service uses promises behind the scenes

```
1    // define the async request
2    var myAsyncWorker = $q.defer();
3
4    // retrieve the promise object
5    var promise = myAsyncWorker.promise;
6
7    // register success and error callbacks
8    promise.then(function(name)  {
9      $scope.name = name;
10   },
11   function(error) {
12     $scope.name = error;
13   });
```

testPromise.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html lang="en" ng-app="MyModule">
  <head>
    <script src="scripts/angular.min.js"></script>
    <script>
    var myModule = angular.module("MyModule", []);

    myModule.controller("MyController",
       function($scope, $timeout, $q)
    {
        $scope.name = "PLACEHOLDER";

        // define the async request
        var myAsyncWorker = $q.defer();

        // retrieve the promise object
        var promise = myAsyncWorker.promise;

        // register success and error callbacks
        promise.then(function(name) // OK
        {
            $scope.name = name;
        },
        function(error) // error
        {
            $scope.name = error;
        });

        // simulate async operation with timer
        $timeout(function()
        {
            var random = Math.random();
            console.log("Random number: " + random);

            if (random > 0.5)
                myAsyncWorker.resolve("Sue");
            else
                myAsyncWorker.reject("Bad boy");
        }, Math.random() * 2000);
    });
    </script>
  </head>
  <body>
    <div ng-controller="MyController">
```

```
46        </div>
47      </body>
48    </html>
```

The $http Service and Promises

- The $http service returns promise objects when you make Ajax requests
- By default, the service assumes the JSON data format

```
1    var response = $http.get("rest/students/12");
2
3    response.then(function(resp)
4    {
5        console.log(resp.data);
6    },
7    function(resp)
8    {
9        console.log(resp.statusText);
10   });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Promises in the View

- A controller can define a scoped variable and then reset it when the promise resolves

```
1    myModule.controller("MyCtrl", function($http)
2    {
3      var mc = this;
4      mc.myvar = "PLACEHOLDER";
5
6      var thePromise = $http.get("mydata.js");
7      thePromise.then(function(resp)
8      {
9        console.log("Success");
10       mc.myvar = resp.data;
11     });
12     console.log("Ctrl function end");
13   });
```

```
1    <div ng-controller="MyCtrl as mc">
2      <div x-ng-bind="mc.myvar"></div>
3    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

RESTful CRUD Service Overview

- Many applications use the four **CRUD** operations: **Create**, **Retrieve**, **Update** and **Delete**
- The REST community has come up with guidelines for how to implement CRUD using standard HTTP facilities
- The AngularJS **$http** service provides methods so can you invoke the four CRUD methods from a Web page

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking the C in CRUD

- The $http service provides the **post()** convenience method that you can use to create a RESTful resource

```
1     var theStudent = {name: "Joe", gpa:  3.44};
2
3     var promise = $http.post("rest/students", theStudent);
4     promise.then(function(resp)
5     {
6        console.log("New student ID: " + resp.data.studentId);
7        alert("New student's RESTful URI: "
8          + resp.headers("Location"));
9     },
10    function(resp)
11    {
12      alert("Ajax error, " + resp.status);
13    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking the R in CRUD

- The $http service provides the **get()** convenience method that you can use to retrieve a RESTful resource

```
1     var promise = $http.get("rest/students/12");
2
3     promise.then(function(resp)
4     {
5        console.log("Name: " + resp.data.name);
6        console.log("ID: " + resp.data.studentId);
7        console.log("GPA: " + resp.data.gpa);
8     },
9     function(resp)
10    {
11       alert("Ajax error: " + resp.status);
12    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking the U in CRUD

- The $http service provides the **put()** convenience method that you can use to update a RESTful resource

```
1     var theStudent = {studentId: 12, name: "Sam", gpa:  2.22};
2
3     var promise = $http.put("rest/students", theStudent);
4     promise.then(function(resp)
5     {
6        console.log("Student successfully updated");
7     },
8     function(resp)
9     {
10       alert("Update failed: " + resp.status);
11    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking the D in CRUD

- The $http service provides the **delete()** convenience method that you can use to delete a RESTful resource

```
1     var promise = $http.delete("rest/students/12");
2
3     promise.then(function(resp)
4     {
5        console.log("Student successfully deleted");
6     },
7     function(resp)
8     {
9         alert("Delete failed: " + resp.status);
10    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

The JAX-RS Service

Student.java

MyContextListener.java

StudentApplication.java

StudentService.java

AngularJS Clients

the-c-in-crud.html

the-r-in-crud.html

the-u-in-crud.html

the-d-in-crud.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package edu.bigcollege;

import java.io.Serializable;

import javax.xml.bind.annotation.XmlRootElement;

@SuppressWarnings("serial")
@XmlRootElement
public class Student implements Serializable
{
    private int studentId;
    private String name;
    private double gpa;

    public Student()
    {
    }

    public Student(int studentId, String name, double gpa)
    {
        super();
        this.studentId = studentId;
        this.name = name;
        this.gpa = gpa;
    }

    public int getStudentId()
    {
        return studentId;
    }

    public void setStudentId(int studentId)
    {
        this.studentId = studentId;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
```

```
46
47      public double getGpa()
48      {
49          return gpa;
50      }
51
52      public void setGpa(double gpa)
53      {
54          this.gpa = gpa;
55      }
56
57      @Override
58      public int hashCode()
59      {
60          final int prime = 31;
61          int result = 1;
62          result = prime * result + studentId;
63          return result;
64      }
65
66      @Override
67      public boolean equals(Object obj)
68      {
69          if (this == obj)
70              return true;
71          if (obj == null)
72              return false;
73          if (getClass() != obj.getClass())
74              return false;
75          Student other = (Student) obj;
76          if (studentId != other.studentId)
77              return false;
78          return true;
79      }
80
81      @Override
82      public String toString()
83      {
84          StringBuilder builder = new StringBuilder();
85          builder.append("Student [studentId=").append(studentId)
86                  .append(", name=").append(name).append(", gpa=").append(gpa)
87                  .append("]");
88          return builder.toString();
89      }
```

```java
package edu.bigcollege.listeners;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

import edu.bigcollege.Student;

@WebListener
public class MyContextListener implements ServletContextListener
{

    @Override
    public void contextDestroyed(ServletContextEvent ctx)
    {
    }

    @Override
    public void contextInitialized(ServletContextEvent ctx)
    {
        System.out.println("Context initialized");
        List<Student> students = new ArrayList<Student>();

        Student c = new Student();
        c.setGpa(3.44);
        c.setName("Sue Smith");
        c.setStudentId(12);
        students.add(c);

        c = new Student();
        c.setGpa(3.87);
        c.setName("Bill Jones");
        c.setStudentId(44);
        students.add(c);

        ctx.getServletContext().setAttribute("students", students);
    }

}
```

```java
package edu.bigcollege.service;

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/rest/")
public class StudentApplication extends Application
{
    @Override
    public Set<Class<?>> getClasses()
    {
        Set<Class<?>> s = new HashSet<Class<?>>();
        s.add(StudentService.class);

        return s;
    }
}
```

```
1   package edu.bigcollege.service;
2
3   import java.net.URI;
4   import java.net.URISyntaxException;
5   import java.util.List;
6   import java.util.Random;
7
8   import javax.servlet.ServletContext;
9   import javax.ws.rs.Consumes;
10  import javax.ws.rs.DELETE;
11  import javax.ws.rs.GET;
12  import javax.ws.rs.POST;
13  import javax.ws.rs.PUT;
14  import javax.ws.rs.Path;
15  import javax.ws.rs.PathParam;
16  import javax.ws.rs.Produces;
17  import javax.ws.rs.core.Context;
18  import javax.ws.rs.core.Response;
19  import javax.ws.rs.core.Response.ResponseBuilder;
20  import javax.ws.rs.core.Response.Status;
21
22  import edu.bigcollege.Student;
23
24  @Path("students")
25  public class StudentService
26  {
27      @Context
28      ServletContext servletContext;
29
30      @SuppressWarnings("unchecked")
31      @DELETE
32      @Path("{id}")
33      public Response delete(@PathParam("id") int id)
34      {
35          ResponseBuilder rb = null;
36
37          System.out.println("In delete: " + id);
38
39          Student found = findStudent(id);
40          if(found != null)
41          {
42              List<Student> students = (List<Student>) servletContext
43                      .getAttribute("students");
44
```

```
46              rb = Response.noContent();
47          }
48          else
49          {
50              rb = Response.status(Status.NOT_FOUND);
51          }
52
53          return rb.build();
54      }
55
56      @PUT
57      @Consumes({ "text/xml", "application/json" })
58      public Response update(Student s)
59      {
60          System.out.println("In update: " + s);
61          ResponseBuilder rb = null;
62
63          if(s != null)
64          {
65              Student found = findStudent(s.getStudentId());
66
67              if(found != null)
68              {
69                  found.setGpa(s.getGpa());
70                  found.setName(s.getName());
71                  rb = Response.noContent();
72              }
73              else
74              {
75                  rb = Response.status(Status.NOT_FOUND);
76              }
77          }
78          else
79          {
80              rb = Response.status(Status.CONFLICT);
81          }
82
83          return rb.build();
84      }
85
86
87      @SuppressWarnings("unchecked")
88      @POST
89      @Consumes({ "text/xml", "application/json" })
```

```
91      // representation in response body
92      @Produces({ "text/xml", "application/json" })
93      public Response insert(Student s)
94      {
95          System.out.println("In insert: " + s);
96          ResponseBuilder rb = null;
97
98          if(s != null)
99          {
100             List<Student> students = (List<Student>) servletContext
101                     .getAttribute("students");
102
103             Random r = new Random();
104             s.setStudentId(Math.abs(r.nextInt()));
105
106             students.add(s);
107
108             try
109             {
110                 rb = Response.created(new URI("/students/" +
111                     s.getStudentId()));
112                 // marshal the student into the response body
113                 rb.entity(s);
114             }
115             catch (URISyntaxException e)
116             {
117                 System.out.println("ERROR inserting: " +
118                   e.getMessage());
119                 rb = Response.serverError();
120             }
121         }
122         else
123         {
124             rb = Response.status(Status.BAD_REQUEST);
125         }
126
127         return rb.build();
128     }
129
130     @SuppressWarnings("unchecked")
131     @GET
132     @Path("{id}")
133     @Produces({ "text/xml", "application/json" })
134     public Response getStudent(@PathParam("id") int id)
```

```
136         Response ret = null;
137 
138         List<Student> students = (List<Student>) servletContext
139                 .getAttribute("students");
140 
141         Student theStudent = null;
142         for (Student c : students)
143         {
144             if (c.getStudentId() == id)
145             {
146                 theStudent = c;
147                 break;
148             }
149         }
150 
151         if (theStudent != null)
152         {
153             ret = Response.ok(theStudent).build();
154         }
155         else
156         {
157             ret = Response.status(Response.Status.NOT_FOUND).build();
158         }
159 
160         return ret;
161     }
162 
163     @SuppressWarnings("unchecked")
164     @GET
165     @Produces({ "text/xml", "application/json" })
166     public Response getAllStudents()
167     {
168         Response ret = null;
169 
170         List<Student> students = (List<Student>) servletContext
171                 .getAttribute("students");
172 
173         Student[] theStudents = new Student[students.size()];
174 
175         int i = 0;
176         for(Student s : students)
177             theStudents[i++] = s;
178 
179         ret = Response.ok(theStudents).build();
```

```
181         return ret;
182     }
183 
184     @SuppressWarnings("unchecked")
185     private Student findStudent(int id)
186     {
187         List<Student> students = (List<Student>) servletContext
188                 .getAttribute("students");
189 
190         Student found = null;
191         for(Student temp : students)
192         {
193             if(temp.getStudentId() == id)
194             {
195                 found = temp;
196                 break;
197             }
198         }
199         return found;
200     }
201 
202 }
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>AngularJS Create RESTful Client</title>
    <script src="scripts/vendor/angular.js"></script>
    <script>
      var myModule = angular.module('MyModule', []);

      myModule.controller('MyController', function($http)
      {
         var mc = this;

         mc.name="Name Placeholder";
         mc.gpa = "GPA Placeholder"
         mc.newId = "New ID Placeholder";
         mc.newURI = "New URI Placeholder"

         // respond to form submit by making an Ajax request
         mc.createStudent = function()
         {
           // create object from form data
           var theStudent =
           {
              name: mc.name,
              gpa:  mc.gpa
           };

           var promise = $http.post("rest/students", theStudent);
           promise.then(function(resp)
           {
                console.log("New student's RESTful URI: " +
                        resp.headers("Location"));

                mc.newId = resp.data.studentId;
                mc.newURI = resp.headers("Location");
           },
           function(resp)
           {
             alert("Ajax error! " + resp.statusText);
           });
         };
      });
    </script>
```

```
46    <body data-ng-app="MyModule">
47      <div data-ng-controller="MyController as mc">
48        <h1>Create Student</h1>
49        <form data-ng-submit="mc.createStudent()">
50          <div>
51             New student name: <input data-ng-model="mc.name" type="text">
52          </div>
53          <div>
54             New student GPA: <input data-ng-model="mc.gpa" type="number" min="0.0" max="4.0" step="0.01">
55          </div>
56          <input type="submit" value="Create Student on Server">
57        </form>
58        <div>
59          New student's ID: <span data-ng-bind="mc.newId">Placeholder</span>
60        </div>
61        <div>
62          New student's RESTful URI: <span data-ng-bind="mc.newURI">Placeholder</span>
63        </div>
64      </div>
65    </body>
66  </html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>AngularJS Retrieve RESTful Client</title>
    <script src="scripts/vendor/angular.js"></script>
    <script>
      var myModule = angular.module('MyModule', []);

      myModule.controller('MyController', function($http)
      {
         var mc = this;

         mc.name="Name Placeholder";
         mc.id = "ID Placeholder";
         mc.gpa = "GPA Placeholder"

         mc.getStudent = function()
         {
           var promise = $http.get("rest/students/12");
           promise.then(function(resp)
           {
             mc.name = resp.data.name;
             mc.id = resp.data.studentId;
             mc.gpa = resp.data.gpa;
           },
           function(resp)
           {
             alert("Ajax error! " + resp.statusText);
           });
         };

         mc.allStudents = [];

         mc.getAllStudents = function()
         {
             var promise = $http.get("rest/students");
             promise.then(function(resp)
             {
                 mc.allStudents = resp.data.student;
             },
             function(resp)
             {
               alert("Ajax error! " + resp.statusText);
```

```
46
47            };
48          });
49        </script>
50      </head>
51      <body data-ng-app="MyModule">
52        <h1>Get Student #12</h1>
53        <div data-ng-controller="MyController as mc">
54          <div>
55             Name: {{mc.name}}
56           </div>
57          <div>
58             ID: {{mc.id}}
59           </div>
60          <div>
61             GPA: {{mc.gpa}}
62           </div>
63          <form data-ng-submit="mc.getStudent()">
64            <input type="submit" value="Get Student #12 from Server">
65          </form>
66          <h1>All Student IDs</h1>
67          <form data-ng-submit="mc.getAllStudents()">
68            <input type="submit" value="Get All Student IDs from Server">
69          </form>
70          <ul>
71            <li data-ng-repeat = "student in mc.allStudents">{{student.studentId}}</li>
72          </ul>
73        </div>
74      </body>
75    </html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>AngularJS Update RESTful Client</title>
    <script src="scripts/vendor/angular.js"></script>
    <script>
      var myModule = angular.module('MyModule', []);

      myModule.controller('MyController', function($http)
      {
         var mc = this;

         mc.id = "ID Placeholder";
         mc.name="Name Placeholder";
         mc.gpa = "GPA Placeholder"

         mc.response = "Placeholder";

         // respond to form submit by making an Ajax request
         mc.updateStudent = function()
         {
           // create object from form data
           var theStudent =
           {
              studentId: mc.id,
              name: mc.name,
              gpa:  mc.gpa
           };

           var promise = $http.put("rest/students", theStudent);
           promise.then(function(resp)
           {
               mc.response = "Student successfully updated";
           },
           function(resp)
           {
               mc.response = "Update failed: " + resp.statusText;
           });
         };
      });
    </script>
  </head>
  <body data-ng-app="MyModule">
```

```
46        <h1>Update Student</h1>
47        <form data-ng-submit="mc.updateStudent()">
48          <div>
49             ID of student to update: <input data-ng-model="mc.id" type="number">
50          </div>
51          <div>
52             Student's new name: <input data-ng-model="mc.name" type="text">
53          </div>
54          <div>
55             Student's new GPA: <input data-ng-model="mc.gpa" type="number" min="0.0" max="4.0" step="0.01">
56          </div>
57          <input type="submit" value="Update Student on Server">
58        </form>
59        <div>
60          Response from service: <span data-ng-bind="mc.response">Placeholder</span>
61        </div>
62      </div>
63    </body>
64  </html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>AngularJS Delete RESTful Client</title>
    <script src="scripts/vendor/angular.js"></script>
    <script>
      var myModule = angular.module('MyModule', []);

      myModule.controller('MyController', function($http)
      {
         var mc = this;

         mc.id = "ID Placeholder";

         mc.response = "Placeholder";

         // respond to form submit by making an Ajax request
         mc.deleteStudent = function()
         {
           var promise = $http.delete("rest/students/" + mc.id);
           promise.then(function(resp)
           {
               mc.response = "Student successfully deleted";
           },
           function(resp)
           {
               mc.response = "Delete failed: " + resp.statusText;
           });
         };
      });
    </script>
  </head>
  <body data-ng-app="MyModule">
    <div data-ng-controller="MyController as mc">
      <h1>Delete Student</h1>
      <form data-ng-submit="mc.deleteStudent()">
        <div>
           ID of student to delete: <input data-ng-model="mc.id" type="number">
        </div>
        <input type="submit" value="Delete Student on Server">
      </form>
      <div>
        Response from service: <span data-ng-bind="mc.response">Placeholder</span>
```

```
46        </div>
47      </body>
48    </html>
```

Chapter Summary

In this chapter, you learned:

- How to use AngularJS's **$http** service to invoke RESTful services

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS Filters

- Using Filters in the View
- Using Filters in JavaScript
- Writing Custom Filters

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Filters

- In many cases, you want to limit or transform the model data while displaying it
- For example, display a subset of a domain-object collection or format a number as currency
- Angular supports **filters** for these purposes
- In the view, you use the pipe | symbol to invoke a filter and the colon : symbol to pass arguments

```
1    <div ng-controller="StudentController as sc">
2      <ul>
3        <li
4          ng-repeat="student in sc.students | orderBy:'gpa'">
5            Name: {{student.name}} GPA: {{student.gpa}}
6        </li>
7      </ul>
8    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Predefined Filters

- Angular provides several filters and you can create your own

```
Filter Name    Description
-----------    ------------
currency*      Formats a number with two decimal
               places and currency symbol
date*          Formats a date
lowercase*     Changes a string's case
number*        Formats a number
uppercase*     Changes a string's case
json*          Pretty prints JSON for debugging
filter**       Subsets a collection based on a criteria
limitTo**      Subsets a collection based on a count
orderBy**      Orders a collection by a criteria

* Formatting or transformation filter
** Array-based filter
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Formatting and Transformation Filters

- Formatting filters let the view control the display of model variables
- Transformation filters let the view change the case of string-based variables

```
1    Name: {{bsc.firstStudent.name | uppercase}}
2    GPA: {{bsc.firstStudent.gpa | number:2}}
3    Fees paid: {{bsc.firstStudent.fees | currency}}
4    Enroll date: {{bsc.firstStudent.enrollDate | date}}
5    Raw JSON: {{bsc.firstStudent | json}}
```

Name: SUE
GPA: 3.21
Fees paid: $21.20
Enroll date: Apr 23, 2012
Raw JSON: { "name": "Sue", "studentID": 34, "gpa": 3.214, "fees": 21.2, "enrollDate": "2012-04-23T18:25:43.511Z" }

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Array-Based Filters

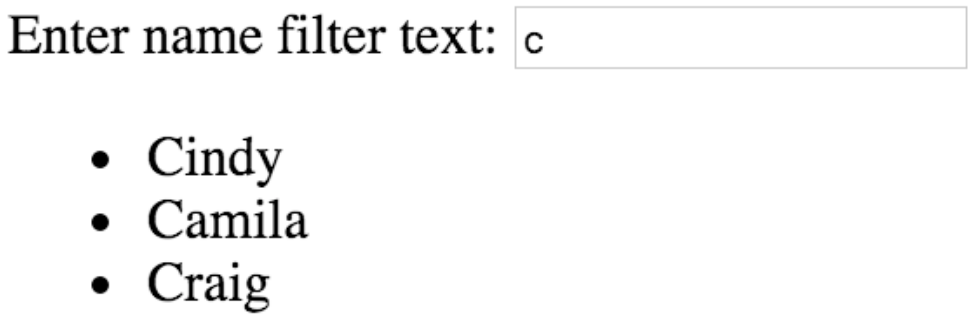
- Array-based filters work on arrays and let you subset the array or change ordering for display

```
1    <div ng-controller="StudentController as sc">
2      <ul>
3        <li
4          ng-repeat="student in sc.students | orderBy:'gpa'">
5            Name: {{student.name}} GPA: {{student.gpa}}
6        </li>
7      </ul>
8    </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The filter Filter

- The **filter** filter subsets an array based on a predicate and optional comparator
- The predicate can be a simple string, an object or a filtering function



filterFilterString.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html x-ng-app="BasicControllerModule">
  <head>
    <title>The "filter" Filter</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/student-service.js"></script>
    <script src="scripts/app/basic-controller.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="BasicStudentController as bsc">
      <div>
        <label for="filterText">Enter name filter text: </label>
        <input id="filterText" type="text" x-ng-model="bsc.filterText">
      </div>
      <ul>
        <li x-ng-repeat="aName in bsc.studentNames | filter:bsc.filterText">
          {{aName}}
        </li>
      </ul>
    </div>
  </body>
</html>
```

Filtering using a Filter Object

- To filter based on more than one criteria, you can pass the **filter** filter an object instead of a simple string

Enter name filter text: d
Enter GPA filter text: 2

Name	ID	GPA
Fred	41	2.431
Aida	49	2.593

filterFilterObject.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html x-ng-app="BasicControllerModule">
  <head>
    <title>The "filter" Filter ()</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/student-service.js"></script>
    <script src="scripts/app/basic-controller.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="BasicStudentController as bsc">
      <div>
        <label for="filterText">Enter name filter text: </label>
        <input id="filterText" type="text"
           x-ng-model="bsc.filterObject.name">
      </div>
      <div>
        <label for="filterText">Enter GPA filter text: </label>
        <input id="filterText" type="text"
          x-ng-model="bsc.filterObject.gpa">
      </div>
      <table style="margin-top: 1em;">
        <tr>
          <th>Name</th>
          <th>ID</th>
          <th>GPA</th>
        </tr>
        <tr
          x-ng-repeat="student in bsc.students | filter:bsc.filterObject">
            <td>{{student.name}}</td>
            <td>{{student.studentID}}</td>
            <td>{{student.gpa}}</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

The limitTo Filter

- The **limitTo** filter subsets an array or string by counting from either the beginning or end
- To subset from the beginning, specify a positive number, negative from the end

Enter count: 2

Limit from beginning

Limit from end

Multiplier: -1

Name	ID	GPA
Dubois	51	3.169
Craig	52	3.446

limitToFilter.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html x-ng-app="BasicControllerModule">
  <head>
    <title>The limitTo Filter</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/student-service.js"></script>
    <script src="scripts/app/basic-controller.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="BasicStudentController as bsc">
      <div>
        <label for="count">Enter count: </label>
        <input id="count" type="text" x-ng-model="bsc.count">
      </div>
      <div>
        <label for="beginning">
          <input id="beginning" type="radio" value="1"
              x-ng-model="bsc.multiplier">
          Limit from beginning
        </label>
      </div>
      <div>
        <label for="end">
          <input id="end" type="radio" value="-1"
              x-ng-model="bsc.multiplier">
          Limit from end
        </label>
      </div>
      <div>
        Multiplier: {{bsc.multiplier}}
      </div>
      <table style="margin-top: 1em;">
        <tr>
          <th>Name</th>
          <th>ID</th>
          <th>GPA</th>
        </tr>
        <tr
x-ng-repeat="student in bsc.students | limitTo:bsc.count * bsc.multiplier">
            <td>{{student.name}}</td>
            <td>{{student.studentID}}</td>
            <td>{{student.gpa}}</td>
        </tr>
```

```
46        </div>
47      </body>
48    </html>
```

The orderBy Filter

- The **orderBy** filter orders an array based on a predicate
- The predicate can be a string (expression), a function or an array of strings/functions
- You can optionally specify whether or not to reverse the ordering

Sort Field

- (•) Order by student name
- ( ) Order by student ID
- ( ) Order by GPA

Ascending or Descending

- ( ) Ascending
- (•) Descending

Order by: name, descending: true

Name	ID	GPA
Wilma	39	3.287
Tekla	50	2.338
Sue	34	3.214

orderByFilter.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html x-ng-app="BasicControllerModule">
  <head>
    <title>The orderBy Filter</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/student-service.js"></script>
    <script src="scripts/app/basic-controller.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="BasicStudentController as bsc">
        <fieldset>
          <legend>Sort Field</legend>
          <div>
            <label for="name">
              <input id="name" type="radio" value="name"
                  x-ng-model="bsc.orderBy">
              Order by student name
            </label>
          </div>
          <div>
            <label for="id">
              <input id="id" type="radio" value="studentID"
                  x-ng-model="bsc.orderBy">
              Order by student ID
            </label>
          </div>
          <div>
            <label for="gpa">
              <input id="gpa" type="radio" value="gpa"
                  x-ng-model="bsc.orderBy">
              Order by GPA
            </label>
          </div>
        </fieldset>
       <fieldset>
          <legend>Ascending or Descending</legend>
          <div>
            <label for="ascending">
              <input id="ascending" type="radio" x-ng-value="false"
                  x-ng-model="bsc.direction">
              Ascending
            </label>
          </div>
```

```
46              <label for="descending">
47                <input id="descending" type="radio" x-ng-value="true"
48                    x-ng-model="bsc.direction">
49                Descending
50              </label>
51            </div>
52        </fieldset>
53        <div>
54          Order by: {{bsc.orderBy}}, descending: {{bsc.direction}}
55        </div>
56        <table style="margin-top: 1em;">
57          <tr>
58            <th>Name</th>
59            <th>ID</th>
60            <th>GPA</th>
61          </tr>
62          <tr
63  x-ng-repeat="student in bsc.students | orderBy:bsc.orderBy:bsc.direction">
64              <td>{{student.name}}</td>
65              <td>{{student.studentID}}</td>
66              <td>{{student.gpa}}</td>
67          </tr>
68        </table>
69      </div>
70    </body>
71  </html>
```

Piping Multiple Filters

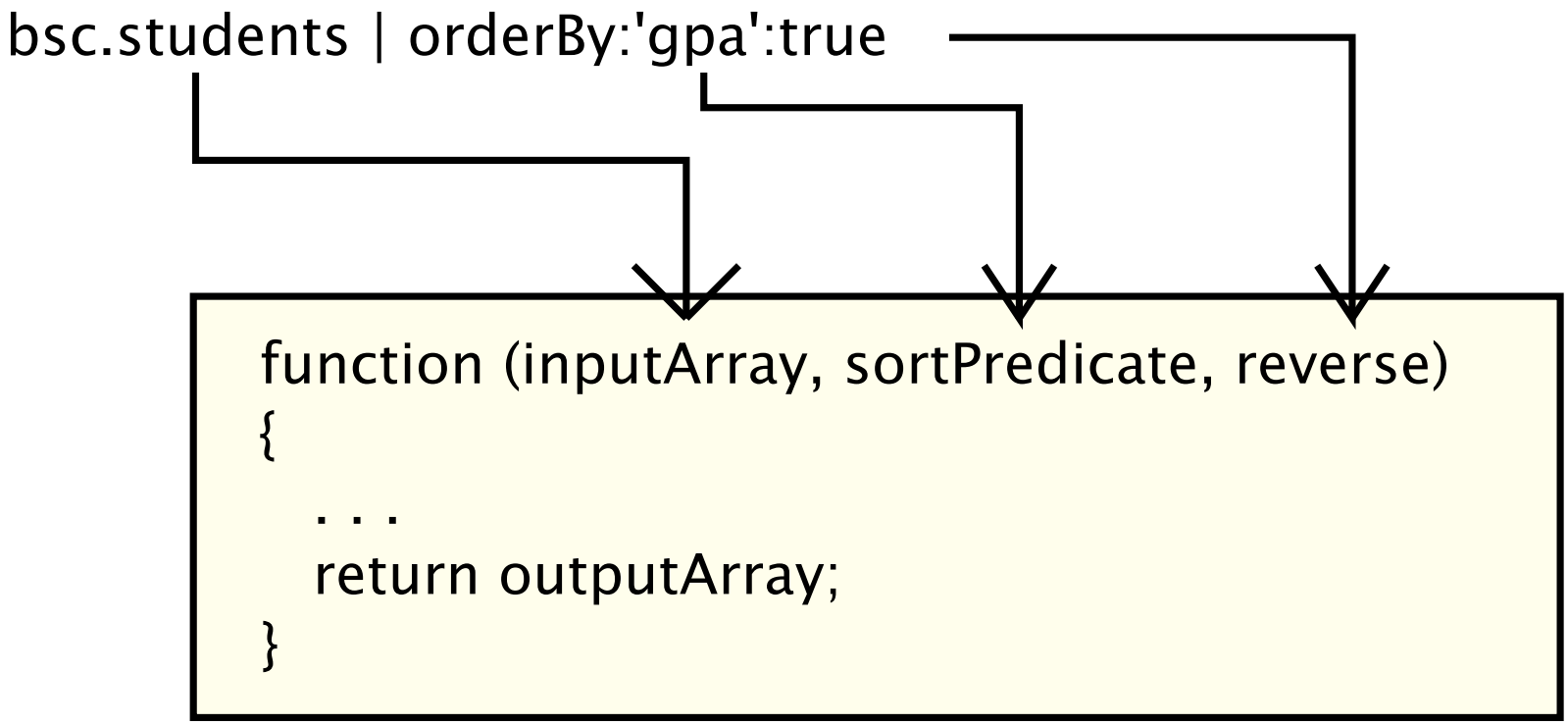
- In the view, you can use the pipe | symbol to connect the output of a filter to another

```
<tr
 ng-repeat="student in bsc.students | orderBy:'gpa' | limitTo:5">
     <td>{{student.name}}</td>
     <td>{{student.studentID}}</td>
     <td>{{student.gpa}}</td>
</tr>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Filters are Functions

- At the JavaScript level, a filter is a function with one required argument followed by optional arguments
- The required argument is either an array or string



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking Filters From JavaScript

- Instead of filtering in the view, you can run a filter in a controller or other Angular component
- You can gain access to the filter function using **dependency injection**
- Another approach is to use the $filter service

filteredController.html

filter-controller.js

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html x-ng-app="BasicControllerModule">
  <head>
    <title>A Filtered Controller</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/student-service.js"></script>
    <script src="scripts/app/filter-controller.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="BasicStudentController as bsc">
      <table>
        <tr>
          <th>Name</th>
          <th>ID</th>
          <th>GPA</th>
        </tr>
        <tr
          x-ng-repeat="student in bsc.students">
            <td>{{student.name}}</td>
            <td>{{student.studentID}}</td>
            <td>{{student.gpa}}</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

```
1    (function(angular)
2    {
3        var myModule = angular.module('BasicControllerModule',
4                [ 'StudentServiceModule' ]);
5
6        myModule.controller("BasicStudentController",
7                function(StudentModel, StudentModel, $filter, orderByFilter)
8        {
9            var bsc = this;
10
11   //       var theFilter = $filter("orderBy");
12           var theFilter = orderByFilter;
13
14           bsc.students =
15               theFilter(StudentModel.getStudents(), "gpa", true);
16
17       });
18
19   })(angular);
```

Writing a Custom Filter

- If the predefined filters don't meet your needs, you can register your own by calling a module's **filter()** function
- Registering assigns a name that you can use in the view
- You implement the filter by returning a function that accepts at least one argument and returns something

Name	ID	GPA
Sam	35	3.576
Sarah	38	3.985
Bessie	42	3.662
Julia	45	3.585
Camila	47	3.982
Ivy	48	3.785

good-student-filter.js

custom-filter-controller.js

customFilter.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
(function(angular)
{
    var myModule = angular.module('FiltersModule', []);

    myModule.filter("goodStudent", function()
    {
        return function(inputArray, threshold)
        {
            // threshold is optional - establish default
            if (!threshold)
                threshold = 3.0;

            // make sure it's typed as a number
            threshold = Number(threshold);

            var outputArray = [];

            // only return students whose GPA is at least
            // the threshold
            for (var i = 0; i < inputArray.length; i++)
            {
                if (inputArray[i].gpa >= threshold)
                    outputArray.push(inputArray[i]);
            }

            return outputArray;
        }
    });

})(angular);
```

```
1    (function(angular)
2    {
3        var myModule = angular.module('CustomFilterControllerModule',
4                ['StudentServiceModule', 'FiltersModule']);
5
6        myModule.controller("BasicStudentController",
7                function(StudentModel, StudentModel)
8        {
9            var bsc = this;
10
11           bsc.students = StudentModel.getStudents();
12       });
13
14   })(angular);
```

```
<!DOCTYPE html>
<html x-ng-app="CustomFilterControllerModule">
  <head>
    <title>Using a Custom Filter</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/student-service.js"></script>
    <script src="scripts/app/good-student-filter.js"></script>
    <script src="scripts/app/custom-filter-controller.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="BasicStudentController as bsc">
      <table>
        <tr>
          <th>Name</th>
          <th>ID</th>
          <th>GPA</th>
        </tr>
        <tr
          x-ng-repeat="student in bsc.students | goodStudent:3.5">
            <td>{{student.name}}</td>
            <td>{{student.studentID}}</td>
            <td>{{student.gpa}}</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

Chapter Summary

In this chapter, you learned:

- How to use filters, both in the view and from a controller
- How to code custom filters

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS Directives

- Directives as Reusable Components
- Isolate Scope
- Communicating Directives

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Directive?

- A **directive** is an object that you can attach to a DOM element to add behavior or custom markup
- Angular provides many directives (e.g. **ngApp**) and you can create your own

```
1    <html>
2      <head>
3        <script src="angular.js"></script>
4      </head>
5      <body ng-app ng-init="names = ['Sam','Sue']">
6        <ul>
7          <li ng-repeat="name in names">{{name}}</li>
8        </ul>
9      </body>
10   </html>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Directive Name Matching

- In the HTML, you can specify directives using several syntaxes
- AngularJS **normalizes** the directive name and then attempts to find a matching directive
- The following syntaxes match some user-defined directive **myEl**:

```
1    <my-el></my-el>
2    <my_el></my_el>
3    <div my-el></div>
4    <div x-my-el></div>
5    <div data-my-el></div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating Custom Directives

- You can create custom directives by calling module.directive() and specifying the directive's name and a function
- Typically, the function returns a **directive definition object** that configures the directive
- We can use custom directives to create reusable Web **components** so we can implement the **Don't Repeat Yourself** (DRY) principle

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Hello, World Directive

- This simple directive simply emits HTML markup
- We can use it in the view as either an element or an attribute

```
1    // hello-world.directive.js
2    helloModule.directive("helloSimple", function()
3    {
4      var ddo =
5      {
6        template: "<div>Hello, world!</div>"
7      };
8      return ddo;
9    });
```

```
1    <!-- UseSimpleHelloWorldDirective.html -->
2    <div x-hello-simple></div>
3    <hello-simple></hello-simple>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Directives as Attributes or Elements

- By default, directives work as either elements or attributes, but you can use **restrict** in the DDO
- The general rule of thumb is to use an element for directives that act as a **domain-specific language** (DSL) and attributes for those that decorate existing elements

```
1    helloModule.directive("helloAttribute", function() {
2      var ddo = {
3        restrict : "A",
4        template : "<div>Hello, world!</div>"
5      };
6      return ddo;
7    });
```

```
1    <div hello-attribute></div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Expressions in Markup Templates

- You can use {{ }} as normal within a directive's template markup

```
1    helloModule.directive("helloWithExpression", function()
2    {
3      var ddo =
4      {
5        template : "<div>Hello, {{name}}!</div>"
6      };
7      return ddo;
8    });
```

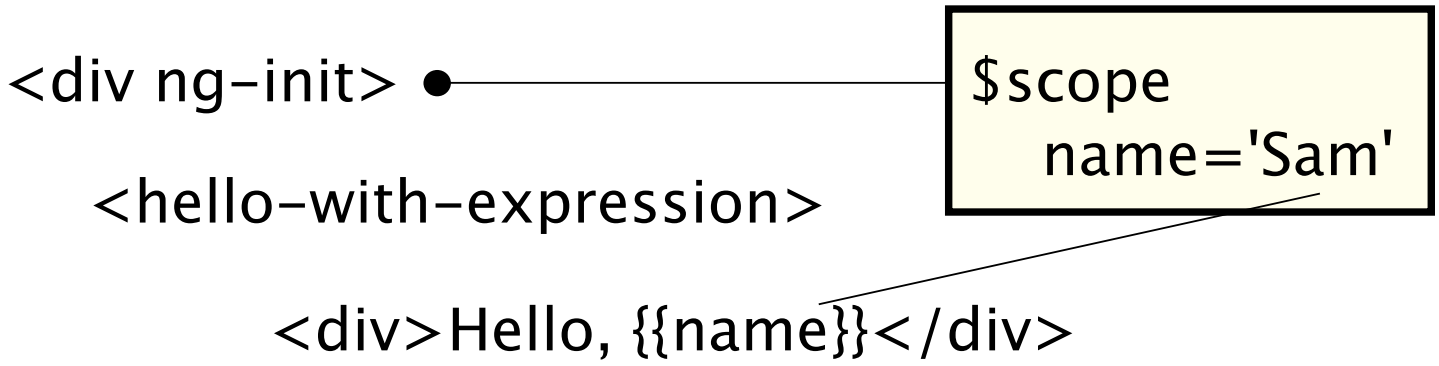
```
1    <div ng-init="name='Sam'">
2      <hello-with-expression></hello-with-expression>
3    </div>
```

- What would happen if we forget the **ng-init**?

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Directive Scopes

- By default, Angular evaluates template expressions against the current scope, which is typically defined by a controller on an enclosing element
- This is easy, but makes writing truly reusable directives difficult since they depend on an enclosing scope containing variables

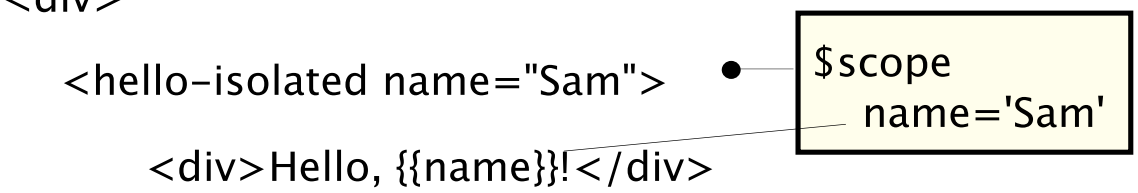


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Isolate Scope

- When you define a directive, you can configure an **isolate** scope which is not chained to any ancestor scope
- You can then pass **attributes** into the directive to populate the isolate scope

```
1    helloModule.directive("helloIsolated", function(){
2      var ddo = {
3        scope :  {
4           name: "@name"
5        },
6        template : "<div>Hello, {{name}}!</div>"
7      };
8      return ddo;
9    });
```



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Isolate Scope, cont'd

- To configure isolate scope, define **attributes** that the user of the directive provides
- Use @ for string-based attributes, = for other types, e. g. objects

isolate-scope.js

UseIsolateScope.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
(function(angular)
{
    var helloModule = angular.module("IsolateModule", []);

    helloModule.directive("isolateString", function()
    {
        var ddo =
        {
            restrict : "A",
            scope    :
            {
                abc: "@xyz"
            },
            template : "<div>Hello, {{abc}}!</div>"
        };

        return ddo;
    });

    helloModule.directive("isolateObject", function()
    {
        var ddo =
        {
            restrict : "A",
            scope    :
            {
                abc: "=xyz"
            },
            template : "<div>Hello, {{abc.fname}} {{abc.lname}}!</div>"
        };

        return ddo;
    });

})(angular);
```

```
<!DOCTYPE html>
<html x-ng-app="IsolateModule">
  <head>
    <title>Directive Isolate Scope</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/isolate-scope.js"></script>
  </head>
  <body x-ng-init="myobj={fname:'Bill', lname:'Jones'}">
    <div x-xyz="Sam" x-isolate-string></div>
    <div x-xyz="myobj" x-isolate-object></div>
  </body>
</html>
```

Working with the DOM

- Directives can work with the DOM by providing a **link** function
- Angular calls the link function during initialization, passing three arguments:
    - **scope** - The directive's scope (parent or isolate)
    - **element** - The element on which the directive resides, wrapped as a jQuery object
    - **attrs** - An object hash of the attributes defined on the directive's element

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Link Function Example

- This directive provides a reusable component that changes color every time the user clicks it

link-function.js

UseLinkFunction.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
(function(angular)
{
    var helloModule = angular.module("LinkFunctionModule", []);

    helloModule.directive("clickToggle", function()
    {
        var linkFunction = function(scope, element, attrs)
        {
            var toggled = false;
            var toggleColor = scope.toggleColor;
            var originalColor = element.css("color");
            if (!originalColor)
                originalColor = "black";

            element.on("click", function()
            {
                if(!toggled)
                {
                    toggled = true;
                    element.css("color", toggleColor);
                }
                else
                {
                    toggled = false;
                    element.css("color", originalColor);
                }
            });
        };

        var ddo =
        {
            restrict : "E",
            scope :
            {
                "toggleColor" : "@"
            },
            link: linkFunction
        };

        return ddo;
    });

})(angular);
```

```
<!DOCTYPE html>
<html x-ng-app="LinkFunctionModule">
  <head>
    <title>Directive Link Function</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/link-function.js"></script>
  </head>
  <body>
    <click-toggle toggle-color="red" style="color:green">
      Hello, world
    </click-toggle>
  </body>
</html>
```

Rendering a Directive's Child Nodes

- Using isolate scope, we can pass information into the directive as parameters, but we are limited to using **attributes**
- Sometimes, we need to pass large blocks of marked-up text into a directive, and that's not feasible using attributes
- A directive can use **transclusion** to splice the directive's child nodes into the directive's template

```
<clumsy-attribute-directive
  my-attr="We want a lot of text with markup here, but clumsy">
</clumsy-attribute-directive>

<nice-transclusion-directive>
  It's much nicer to pass a <b>lot</b> of text like this
</nice-transclusion-directive>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Transclusion Example

- To use transclusion in a directive, in the DDO, specify **transclude: true** and in the template, use **ng-transclude** to define the insertion point

```
1    myModule.directive("simpleTransclude", function() {
2      var ddo = {
3        restrict : "E",
4        transclude : true,
5        template :
6         "<h2>Simple Transclusion</h2><div ng-transclude></div>"
7      };
8      return ddo;
9    });
```

```
1    <simple-transclude>
2      Lotsa stuff including <div>markup</div>
3    </test-transclude>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transclusion and Scope

- If the transcluded text contains expressions, Angular evaluates them against the directive's **parent** scope

transclusion-scopes.js

transclusion-scope.html

UseTransclusionScope.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    (function(angular)
2    {
3        var transcludeModule = angular.module("TransclusionModule", []);
4
5        transcludeModule.directive("transclusionScope", function()
6        {
7            var ddo =
8            {
9                restrict    : "E",
10               scope       :
11               {
12                   myAttr: "@"
13               },
14               transclude  : true,
15               templateUrl : "transclusion-scope.html"
16           };
17
18           return ddo;
19       });
20
21       transcludeModule.controller("TranscludeController", function()
22       {
23           var tc = this;
24
25           this.myVariable = "My variable";
26       });
27
28   })(angular);
```

```
1    <h2>Transclusion Scope, myAttr: {{myAttr}}</h2>
2    <div ng-transclude></div>
```

```
<!DOCTYPE html>
<html x-ng-app="TransclusionModule">
  <head>
    <title>Directive Transclusion Scopes</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/transclusion-scopes.js"></script>
  </head>
  <body>
    <div ng-controller="TranscludeController as tc">
      <transclusion-scope my-attr="My Attribute">
        Here is the content of myVariable: {{tc.myVariable}}
      </transclusion-scope>
    </div>
  </body>
</html>
```

Controllers in Directives

- Directives can work together to implement **composite** reusable components, such as tabbed containers
- You can write a **controller** function within a directive so that you can consolidate the directive's logic and/or to communicate with other directives

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Directive Controller

- Adding a controller to a directive lets you define functions that you can invoke from other directives
- You can use the **require** option to configure the "child" directives and within them, use dependency injection to access the "parent" directive's controller

simple-directive-controller.js

UseSimpleController.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1   (function(angular)
2   {
3       var controllerModule =
4         angular.module("SimpleDirectiveControllerModule", []);
5
6       controllerModule.directive("simpleParent", function()
7       {
8           var ddo =
9           {
10              restrict : "E",
11              controller: function()
12              {
13                  this.myFunction = function()
14                  {
15                      console.log("In parent's myFunction");
16                  }
17              }
18          };
19
20          return ddo;
21      });
22
23      controllerModule.directive("simpleChild", function()
24      {
25          var ddo =
26          {
27              restrict : "A",
28              require  : "simpleParent",
29              link     : function(scope, element, attrs, simpleParent)
30              {
31                  simpleParent.myFunction();
32              }
33          };
34
35          return ddo;
36      });
37
38  })(angular);
```

```
<!DOCTYPE html>
<html x-ng-app="SimpleDirectiveControllerModule">
  <head>
    <title>Simple Directive Controller</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/simple-directive-controller.js"></script>
  </head>
  <body>
    <simple-parent simple-child></simple-parent>
  </body>
</html>
```

More on the require Option

- On a "child" directive, you can specify how Angular finds the "parent"

require Value	Description
myParent	Look only on current element; throw error if not found
?myParent	Same as above, but pass null if not found
^myParent	Look on current element or ancestors; throw error if not found
^^myParent	Do not look on current element, only ancestors; throw error if not found
?^myParent	Look on current element or ancestors; pass null if not found
?^^myParent	Do not look on current element, only ancestors; pass null if not found

container-directive.js UseContainerDirective.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
(function(angular)
{
    var controllerModule =
        angular.module("ContainerDirectiveModule", []);

    controllerModule.directive("myContainer", function()
    {
        var ddo =
        {
            restrict : "E",
            controller: ["$scope", function($scope)
            {
                var children = [];

                this.addChild = function(theChild)
                {
                    console.log("In addChild: " + theChild);
                    children.push(theChild);
                }
            }]
        };

        return ddo;
    });

    controllerModule.directive("myChild",
            function()
    {
        var ddo =
        {
            restrict : "E",
            require  : "^^myContainer",
            link     : function(scope, element, attrs, simpleParent)
            {
                simpleParent.addChild(this);
            },
            template : "<div ng-transclude></div>",
            transclude: true
        };

        return ddo;
    });

})(angular);
```

```
<!DOCTYPE html>
<html x-ng-app="ContainerDirectiveModule">
  <head>
    <title>Container Directive</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/container-directive.js"></script>
  </head>
  <body>
    <my-container>
      <my-child>Child 1</my-child>
      <my-child>Child 2</my-child>
    </my-container>
  </body>
</html>
```

Link vs Controller Functions

- A directive can define a **link** function, a **controller** function, or both
- Angular calls the link() function during page load as it "compiles" the page, so it's a good place to initialize the directive, e.g. to modify the DOM and register event handlers
- A controller is a more general-purpose holder for directive logic and you can use the **require** option to create communicating directives

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How directives act as reusable components
- How scopes work with directives, including isolate scope
- How to modify the DOM from a directive
- How to write directives that communicate with other directives

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS Forms

- Validation and CSS Styling
- Displaying Validation Error Messages
- Custom Validators

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Forms Review

- Angular provides the **ngSubmit** and **ngModel** directives so you can create and process forms in a controller

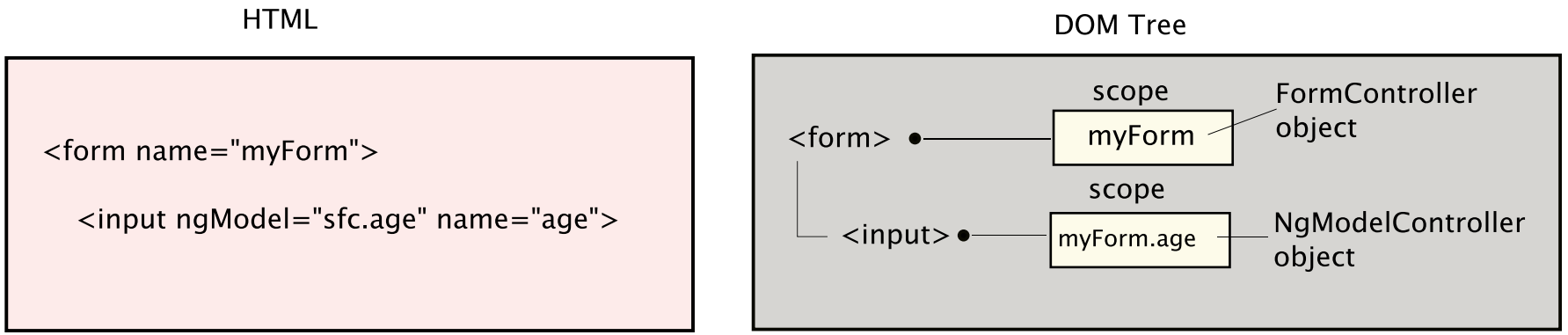
```
1    <div x-ng-controller="SimpleFormController as sfc">
2      <form x-ng-submit="sfc.mySubmit()" name="myForm">
3        Age: <input type="number" x-ng-model="sfc.age"
4           name="age">
5        <input type="submit">
6      </form>
7    </div>
```

```
1    myModule.controller("SimpleFormController", function() {
2      var sfc = this;
3      sfc.mySubmit = function() {
4         console.log("Age: " + age);
5      };
6    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Behind the Scenes

- Angular treats standard HTML elements such as **form** and **input** as Angular directives
- That makes it possible to handle validation of input fields in the "Angular way"
- <form> creates a **FormController** object and <input> (with ngModel) creates an **NgModelController** object



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

FormController Properties

- The FormController's properties show the current state of the form and its contained elements
- You can access these via Angular expressions or from JavaScript, say in a controller

```
Name         Description
----         -----------

$error       Object hash with any failed validations
$pristine    True if user hasn't interacted with form
$dirty       True if user has interacted with form
$invalid     True if at least one widget is invalid
$submitted   True if user has submitted form
```

```
<div>Form pristine? {{myForm.$pristine}}</div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

NgModelController Properties

- The NgModelController properties show the current state of the widget
- You can access these via Angular expressions or from JavaScript, say in a controller

```
Name         Description
----         -----------

$error       Object hash with any failed validations: key
             is validate name; value is true if validation
             failed
$validators  Object hash with any validators: key is validator
             name, value is validator function
$pristine    True if user hasn't interacted with widget
$dirty       True if user has interacted with widget
$invalid     True if at least one widget is invalid
```

```
<div>Age widgit pristine? {{myForm.age.$pristine}}</div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Form Validation

- Angular provides rich support for validation, including CSS styling and error-message display
- Angular's JavaScript-based validation runs on the client and does not obviate the need for server-side validation
- You should use **novalidate** on the form to disable native HTML5 validation

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Form Validation Directives

- Angular mimics standard HTML validation

```
Directive   Description
---------   -----------

required    Marks a field as required
ng-required Marks a field as required based on a condition
date        Field must contain a properly formatted date
email       Field must contain a properly formatted email address
number      Field must contain a number
url         Field must contain a properly formatted URL
pattern     Field validated against a regex
min-length  Field must have at least that many characters
max-length  Field must have at most that many characters
min, mix    Field must have a numeric value in the range
```

```
<input type="text" ng-model="sfc.ssn" name="ssn"
   id="ssn" required pattern="[0-9]{3}-[0-9]{2}-[0-9]{4}">
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Preventing Form Submission if Invalid

- You can assign the form a name and then conditionally disable the submit button

```
<input type="submit" x-ng-disabled="myForm.$invalid">
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

CSS Styling for Validation

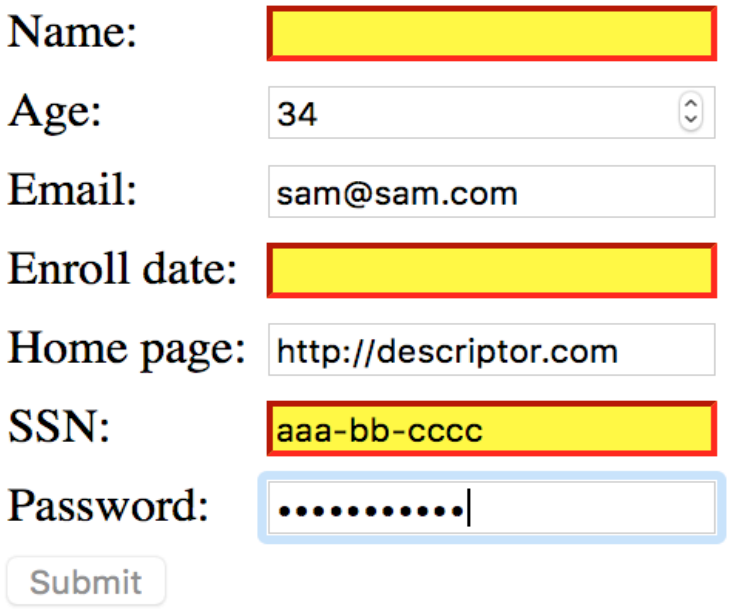
- Angular assigns CSS style classes to form elements based on their validation state
- You can write CSS rules to style "touched" elements that are invalid

CSS Class	Description
ng-valid	Set if the control is valid
ng-invalid	Set if the control is invalid
ng-pristine	Set if the control is pristine
ng-dirty	Set if the control is dirty
ng-touched	Set if control has been focused then lost focus

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

CSS Styling Example

```
1    /* main.css */
2    input.ng-invalid.ng-touched
3    {
4      border-color: red;
5      background-color: yellow
6    }
```



UseSimpleFormValidate.html

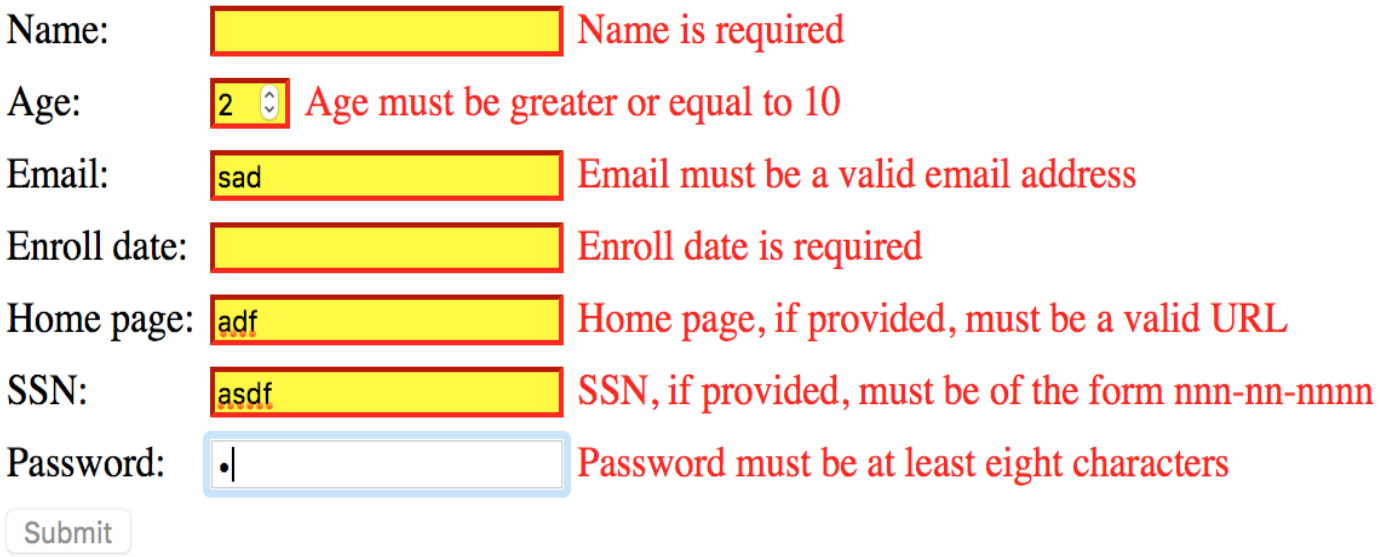
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html x-ng-app="SimpleFormValidateModule">
  <head>
    <title>Form (Basic Validation)</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/vendor/angular-messages.js"></script>
    <script src="scripts/app/simple-form-validate.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="SimpleFormController as sfc">
      <form x-ng-submit="sfc.mySubmit()" novalidate name="myForm">
        <table>
          <tr>
            <td>
              <label for="name">Name:</label>
            </td>
            <td>
              <input type="text" x-ng-model="sfc.name" name="name"
                id="name" required>
            </td>
          </tr>
          <tr>
            <td>
              <label for="age">Age:</label>
            </td>
            <td>
              <input type="number" x-ng-model="sfc.age" name="age"
                id="age" required>
            </td>
          </tr>
          <tr>
            <td>
              <label for="email">Email:</label>
            </td>
            <td>
              <input type="email" x-ng-model="sfc.email" name="email"
                id="email" required>
            </td>
          </tr>
          <tr>
            <td>
              <label for="enrollDate">Enroll date:</label>
            </td>
```

```
46                <input type="date" x-ng-model="sfc.enrollDate"
47                  name="enrollDate"
48                  id="enrollDate" required>
49              </td>
50            </tr>
51            <tr>
52              <td>
53                <label for="homePage">Home page:</label>
54              </td>
55              <td>
56                <input type="url" x-ng-model="sfc.homePage" name="homePage"
57                  id="homePage" required>
58              </td>
59            </tr>
60            <tr>
61              <td>
62                <label for="ssn">SSN:</label>
63              </td>
64              <td>
65                <input type="text" x-ng-model="sfc.ssn" name="ssn"
66                  id="ssn" required
67                  pattern="[0-9]{3}-[0-9]{2}-[0-9]{4}">
68              </td>
69            </tr>
70            <tr>
71              <td>
72                <label for="password">Password:</label>
73              </td>
74              <td>
75                <input type="password" x-ng-model="sfc.password"
76                  name="password"
77                  id="password" required x-ng-minlength="8">
78              </td>
79            </tr>
80          </table>
81          <input type="submit" x-ng-disabled="myForm.$invalid">
82        </form>
83      </div>
84    </body>
85  </html>
```

Displaying Validation Error Messages

- So far, we have covered using CSS styling to indicate validation errors, but the user might not know WHY
- Angular provides two techniques to display error messages conditionally:
    - Use the ngShow attribute
    - Use the **ngMessages** directive



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using ngShow for Conditional Display

- The **ngShow** directive lets you display any element conditionally based on an Angular expression
- You can use it to display validation error messages only if an input widget contains invalid data

```
1    <div>
2      <input type="number" name="input"
3              ng-model="example.value"
4               min="0" max="99" required>
5      <span class="error"
6          ng-show="myForm.input.$error.required">
7          Required!
8      </span>
9      <span class="error"
10         ng-show="myForm.input.$error.number">
11         Not valid number!
12     </span>
13   </div>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The ngMessages Directive

- The **ngMessages** directive makes it easy to define and display error messages
- **ngMessages** is not part of the standard Angular distribution
- Recipe for using ngMessages:
    - Download and reference the angular-messages.js script
    - Declare a dependency on the **ngMessages** module in your "main" module
    - Use the **ngMessages** and **ngMessage** directives in your HTML

UseSimpleFormValidateWithMessages.html

simple-form-validate.js

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html x-ng-app="SimpleFormValidateModule">
  <head>
    <title>Form (Basic Validation)</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/vendor/angular-messages.js"></script>
    <script src="scripts/app/simple-form-validate.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="SimpleFormController as sfc">
      <form x-ng-submit="sfc.mySubmit()" novalidate name="myForm">
        <table>
          <tr>
            <td>
              <label for="name">Name:</label>
            </td>
            <td>
              <input type="text" x-ng-model="sfc.name" id="name"
                name="name" required>
              <span x-ng-messages="myForm.name.$error">
                <span x-ng-message="required" class="errorMsg">
                  Name is required
                </span>
              </span>
            </td>
          </tr>
          <tr>
            <td>
              <label for="age">Age:</label>
            </td>
            <td>
              <input type="number" x-ng-model="sfc.age" id="age"
                name="age" min="10" max="99" required>
              <span x-ng-messages="myForm.age.$error">
                <span x-ng-message="required" class="errorMsg">
                  Age is required
                </span>
                <span x-ng-message="min" class="errorMsg">
                  Age must be greater or equal to 10
                </span>
                <span x-ng-message="max" class="errorMsg">
                  Age must be less than 100
                </span>
```

```
46              </td>
47            </tr>
48            <tr>
49              <td>
50                <label for="email">Email:</label>
51              </td>
52              <td>
53                <input type="email" x-ng-model="sfc.email" id="email"
54                  name="email" required>
55                <span x-ng-messages="myForm.email.$error">
56                  <span x-ng-message="required" class="errorMsg">
57                    Email is required
58                  </span>
59                  <span x-ng-message="email" class="errorMsg">
60                    Email must be a valid email address
61                  </span>
62                </span>
63              </td>
64            </tr>
65            <tr>
66              <td>
67                <label for="enrollDate">Enroll date:</label>
68              </td>
69              <td>
70                <input type="date" x-ng-model="sfc.enrollDate"
71                  id="enrollDate" name="enrollDate" required>
72                <span x-ng-messages="myForm.enrollDate.$error">
73                  <span x-ng-message="required" class="errorMsg">
74                    Enroll date is required
75                  </span>
76                  <span x-ng-message="date" class="errorMsg">
77                    Enroll date must be a valid date
78                  </span>
79                </span>
80              </td>
81            </tr>
82            <tr>
83              <td>
84                <label for="homePage">Home page:</label>
85              </td>
86              <td>
87                <input type="url" x-ng-model="sfc.homePage"
88                  id="homePage" name="homePage">
89                <span x-ng-messages="myForm.homePage.$error">
```

```
                    Home page, if provided, must be a valid URL
                  </span>
                </span>
              </td>
            </tr>
            <tr>
              <td>
                <label for="ssn">SSN:</label>
              </td>
              <td>
                <input type="text" x-ng-model="sfc.ssn" name="ssn"
                  id="ssn" pattern="[0-9]{3}-[0-9]{2}-[0-9]{4}">
                <span x-ng-messages="myForm.ssn.$error">
                  <span x-ng-message="pattern" class="errorMsg">
                    SSN, if provided, must be of the form nnn-nn-nnnn
                  </span>
                </span>
              </td>
            </tr>
            <tr>
              <td>
                <label for="password">Password:</label>
              </td>
              <td>
                <input type="password" x-ng-model="sfc.password"
                  name="password"
                  id="password" required x-ng-minlength="8">
                <span x-ng-messages="myForm.password.$error">
                  <span x-ng-message="required" class="errorMsg">
                    Password is required
                  </span>
                  <span x-ng-message="minlength" class="errorMsg">
                    Password must be at least eight characters
                  </span>
                </span>
              </td>
            </tr>
          </table>
          <input type="submit" x-ng-disabled="myForm.$invalid">
        </form>
      </div>
    </body>
  </html>
```

```
1    (function(angular)
2    {
3        var myModule = angular.module('SimpleFormValidateModule',
4           ['ngMessages']);
5
6        myModule.controller("SimpleFormController", function()
7        {
8            var sfc = this;
9
10           sfc.mySubmit = function()
11           {
12               console.log("Name: " + sfc.name);
13               console.log("Age: " + sfc.age);
14           };
15       });
16
17   })(angular);
```

Writing a Custom Validator

- Angular provides rich validation support "out of the box", but it's possible to write your own validators by writing a **directive** with a link() function that:
    - Defines a directive restricted as an attribute
    - Requires ngModel (the "parent" directive)
    - Configures a function into the widget's $validators hash - the function examines the element's content and returns a boolean

UseCustomValidator.html

custom-validator.js

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html x-ng-app="CustomValidatorModule">
  <head>
    <title>Form (Custom Validator)</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/app/custom-validator.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="CustomValidatorController as cvc">
      <form x-ng-submit="cvc.mySubmit()" novalidate name="myForm">
        <table>
          <tr>
            <td>
              <label for="age">Number (must be prime):</label>
            </td>
            <td>
              <input type="number" x-ng-model="cvc.number" id="number"
                name="number" min="0" required prime>
              <span class="errorMsg"
                x-ng-show="myForm.number.$error.required">
                Number is required
              </span>
              <span class="errorMsg"
                x-ng-show="myForm.number.$error.prime">
                The number must be a prime number
              </span>
            </td>
          </tr>
        </table>
        <input type="submit" x-ng-disabled="myForm.$invalid">
      </form>
    </div>
  </body>
</html>
```

```
(function(angular)
{
    var myModule = angular.module('CustomValidatorModule',
            []);

    myModule.controller("CustomValidatorController", function()
    {
        var cvc = this;

        cvc.mySubmit = function()
        {
            console.log("Number: " + cvc.number);
        };
    });

    myModule.directive('prime', function()
    {
        // borrowed from:
        // http://www.javascripter.net/faq/numberisprime.htm
        function isPrime(n)
        {
            if (isNaN(n) || !isFinite(n) || n % 1 || n < 2)
                return false;
            if (n % 2 == 0)
                return (n == 2);
            if (n % 3 == 0)
                return (n == 3);
            var m = Math.sqrt(n);
            for (var i = 5; i <= m; i += 6)
            {
                if (n % i == 0)
                    return false;
                if (n % (i + 2) == 0)
                    return false;
            }
            return true;
        }

        var ddo =
        {
            restrict : 'A',
            require : 'ngModel',

            link : function(scope, element, attrs, modelCtrl)
```

```
46                  modelCtrl.$validators.prime =
47                  function(modelValue, viewValue)
48                  {
49                      return isPrime(viewValue);
50                  }
51              }
52          };
53
54          return ddo;
55      });
56
57  })(angular);
```

Chapter Summary

In this chapter, you learned:

- About Angular's form architecture
- How to configure validation and use CSS styling as well as display validation error messages
- How to write a custom validator directive

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing JavaScript and AngularJS

- Jasmine
- Karma
- Testing Angular Controllers, Filters and Directives

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Testing Issues

- Like programs in any language, JavaScript programs should have **unit tests** that test the basic function of functions and classes
- The notion of unit testing began with the Java **jUnit** framework, but has been adopted by other languages, including JavaScript

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Testing Tools

- There are several popular JavaScript tools to facilitate unit testing:
    - **Jasmine** - Unit testing framework
    - **QUnit** - Unit testing framework
    - **Mocha** - Unit testing framework
    - **Karma** - A test runner for many environments, including browsers and Node.js. It can integrate with other frameworks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Jasmine

- **Jasmine** is a unit testing framework formerly known as jsUnit
- The home page for Java is at jasmine.github.io



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Installing Jasmine

- Jasmine provides several installation options, including:
    - Standalone
    - Node.js
    - Ruby
    - Java (Maven)
- You can download the standalone installation of Jasmine from https://github.com/jasmine/jasmine/releases

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Jasmine Fundamentals

- Jasmine provides:
    - **describe()** - a function you can use to group related tests, i.e. a **test suite**
    - **it()** - a function that contains one or more tests, i.e. a **spec**
    - **expect()** - a function that execute an assertion
    - **matchers** - functions that act as assertions
- Jasmine provides several matchers and you can write your own

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Test

- Here's a simple test that verifies array sorting

```
1    // sortUsers() is our function we want to test
2    describe('sorting the list of users', function()
3    {
4      it('sorts in descending order by default', function()
5      {
6        var users = ['jack', 'igor', 'jeff'];
7        var sorted = sortUsers(users);
8        expect(sorted).toEqual(['jeff', 'jack', 'igor']);
9      });
10   });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Jasmine Built-in Matchers

Matcher	Description
toBe()	Compares using ===
toEqual()	Compares using ==
toMatch()	Matches a regex
toBeUndefined()	Compares against JavaScript 'undefined' value
toBeNull()	Compares against JavaScript null
toBeTruthy()	Checks for truthiness
toBeFalsy()	Checks for falsyness
toContain()	Determines if an item is in an array
toBeLessThan()	Numeric comparison
toBeGreaterThan()	Numeric comparison
toBeCloseTo()	Numeric comparison with a fudge factor
toThrow()	Nxpect an exception

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Jasmine Setup and Teardown Methods

- Within a describe() function, you can optionally provide **beforeAll()**, **beforeEach()**, **afterEach()** and **afterAll()** functions
- These help reduce redundancy in test suites

```
1    describe('sorting the list of users', function()
2    {
3      beforEach(function()
4      {
5         console.log("Initializing test data");
6         . . .
7      }
8
9      it('sorts in descending order by default', function()
10     {
11        ...
12     });
13   });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Test Runner HTML Page

- To run tests, you can write a testing Web page that includes your JavaScript code, your tests (specs) and the Jasmine framework
- Jasmine provides CSS to style test results for easy reading

```
1    <link rel="shortcut icon" type="image/png"
2       href="lib/jasmine-2.3.4/jasmine_favicon.png">
3    <link rel="stylesheet" href="lib/jasmine-2.3.4/jasmine.css">
4
5    <script src="lib/jasmine-2.3.4/jasmine.js"></script>
6    <script src="lib/jasmine-2.3.4/jasmine-html.js"></script>
7    <script src="lib/jasmine-2.3.4/boot.js"></script>
8
9    <!-- code to test -->
10   <script src="src/Player.js"></script>
11   <script src="src/Song.js"></script>
12
13   <!-- Jasmine test scripts (specs) -->
14   <script src="spec/SpecHelper.js"></script>
15   <script src="spec/PlayerSpec.js"></script>
```

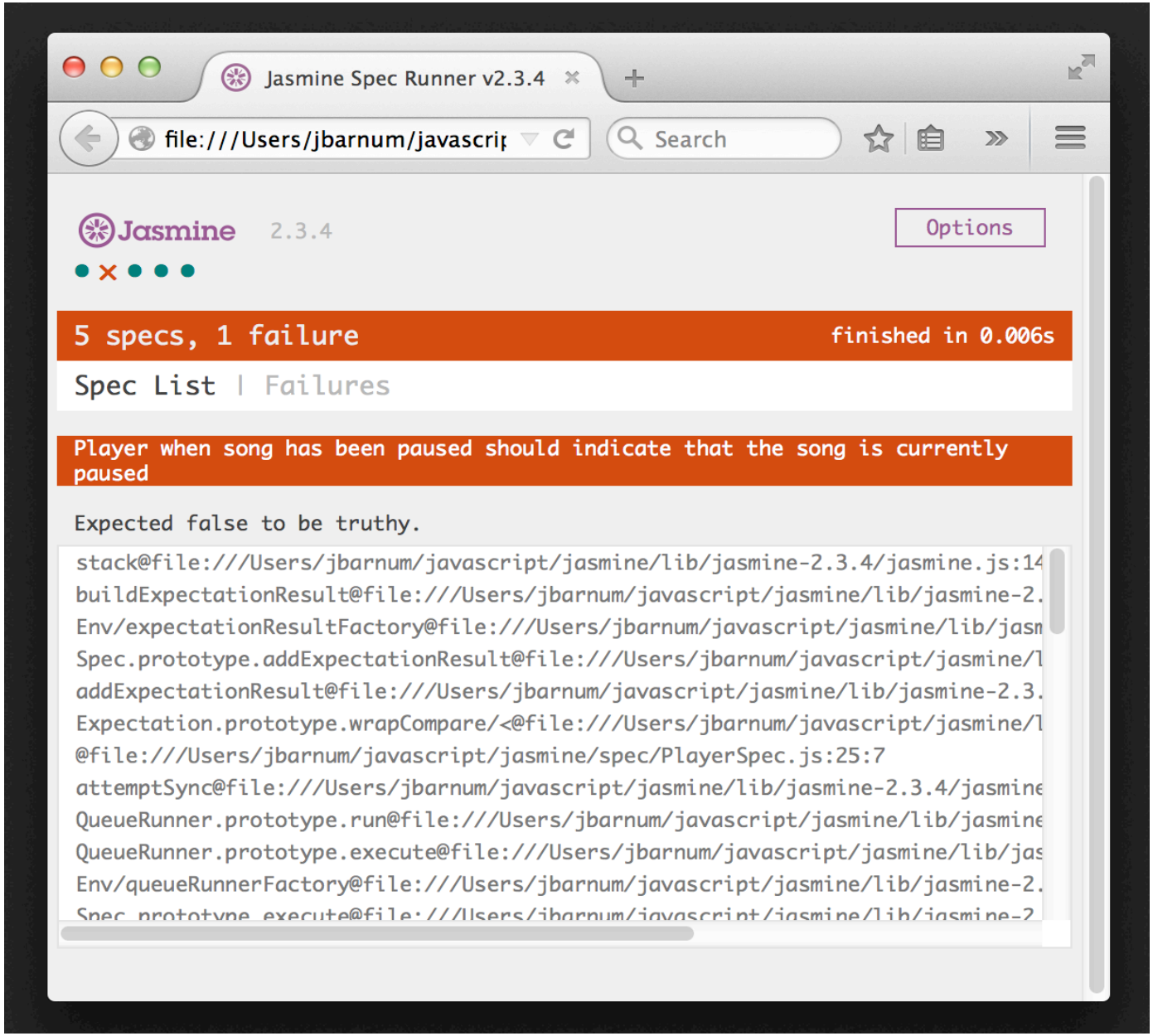

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Test Runner HTML Page, cont'd



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Test Runner HTML Page, cont'd



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Karma

- Karma is a **test runner** that makes it easy to run tests written in Jasmine or other unit-testing framework
- Karma runs your tests in various browsers or Node.js and reports pass/fail to the command line
- The home page for Karma is at: karma-runner.github.io



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Installing Karma

- Karma runs as a command-line tool using the **Node Package Manager** (npm)
- After installing Node.js, you can install Karma:

```
npm install karma --save-dev

sudo npm install -g karma-cli
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Karma Configuration File

- Before running Karma, you must create a configuration file that specifies where your tests are and which browsers you want to test under
- Karma provides an **init** script you can run to create the configuration file

```
karma init my.conf.js
```

my.conf.js

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1   // Karma configuration
2   // Generated on Wed Jun 17 2015 12:08:02 GMT-0700 (PDT)
3
4   module.exports = function(config) {
5     config.set({
6
7       // base path that will be used to resolve all patterns (eg. files, exclude)
8       basePath: '',
9
10
11      // frameworks to use
12      // available frameworks: https://npmjs.org/browse/keyword/karma-adapter
13      frameworks: ['jasmine'],
14
15
16      // list of files / patterns to load in the browser
17      files: [
18        '/Users/jbarnum/javascript/jasmine/**/*.js'
19      ],
20
21
22      // list of files to exclude
23      exclude: [
24      ],
25
26
27      // preprocess matching files before serving them to the browser
28      // available preprocessors: https://npmjs.org/browse/keyword/karma-preprocessor
29      preprocessors: {
30      },
31
32
33      // test results reporter to use
34      // possible values: 'dots', 'progress'
35      // available reporters: https://npmjs.org/browse/keyword/karma-reporter
36      reporters: ['progress'],
37
38
39      // web server port
40      port: 9876,
41
42
43      // enable / disable colors in the output (reporters and logs)
44      colors: true,
```

```
46
47      // level of logging
48      // possible values: config.LOG_DISABLE || config.LOG_ERROR || config.LOG_WARN || config.LOG_INFO || config.LOG_DEI
49      logLevel: config.LOG_INFO,
50
51
52      // enable / disable watching file and executing tests whenever any file changes
53      autoWatch: false,
54
55
56      // start these browsers
57      // available browser launchers: https://npmjs.org/browse/keyword/karma-launcher
58      browsers: ['Chrome'],
59
60
61      // Continuous Integration mode
62      // if true, Karma captures browsers, runs the tests and exits
63      singleRun: false
64    });
65  };
```

Karma and Browsers

- Karma supports many browsers for testing (referred to as **capturing** a browser)
- Karma will start the brower(s), run the tests, kill the browser and then report the pass/fail to the command line
- You can also start a browser manually for testing, even a browser running on a mobile device

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Running Karma Tests

- To run tests, from the command-line:

```
karma start my.conf.js
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing AngularJS Applications

- It's common to use Karma and Jasmine to test AngularJS artifacts such as **controllers** and **directives**
- You can use the **ngMocks** module, which provides the **module()** and **inject()** global functions
- Recipe for using ngMocks:
    - Load the angular-mocks.js script
    - Write Karma or Mocha unit tests that use module() to load the module under test and inject() to access the resources needed for the test

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing Controllers

- You can test if a controller correctly initializes its scope
- Recipe:
    - In a beforeEach block, use **module()** and then **inject()** to inject the Angular $controller service
    - Create an empty scope
    - Use the $controller service to locate and run the controller function
    - Assert that the scope is what you expect

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing a Controller

```
1     describe("MyController", function()
2     {
3       var $controller;
4
5       beforeEach(function()
6       {
7         // load the module under test
8         module("MyModule");
9
10        // inject $controller service - inject() will
11        // strip the underscores from name
12        inject(function(_$controller_)
13        {
14            $controller = _$controller_;
15        });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing a Controller, cont'd

```
17      describe("studentList", function()
18      {
19        it("should have 10 students", function()
20        {
21          // create empty scope
22          var scope = {};
23
24          // locate and run controller, passing scope
25          var mc = $controller('MyController as mc',
26          { $scope : scope });
27
28          // assert expectations
29          expect(scope.mc.students().length).toBe(10);
30      });
31    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing Custom Filters

- You can test if an array-based filter correctly removes elements from its input array
- Recipe:
    - In a beforeEach block, use **module()** and then **inject()** to inject the Angular $filter service
    - Use the $filter service to locate the filter
    - Create an input array, then run the filter function
    - Assert that the output array is as expected

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing a Filter

```
1     describe("goodStudent Filter", function()
2     {
3       var $filter;
4
5       var goodStudent;  // goodStudent filter
6
7       beforeEach(function()
8       {
9         module("MyModule");
10
11        inject(function(_$filter_)
12        {
13          $filter = _$filter_;
14        });
15
16        goodStudent = $filter("goodStudent");
17      });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing a Filter, cont'd

```
18      it("should filter students with GPA less than 3.0",
19      function()
20      {
21        var students = [
22          {name : "Sue", studentID : 11, gpa : 2.21},
23          {name : "Sam", studentID : 17, gpa : 3.57}
24        ];
25
26        // run the filter
27        var results = goodStudent(students);
28
29        // assert expectations
30        expect(results.length).toBe(1);
31        expect(results[0].name).toBe("Sam");
32      });
33    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing Directives

- You can test if a directive correctly modifies the DOM or if a validation directive correctly sets the $invalid flag
- Recipe:
    - In a beforeEach block, use **module()** and then **inject()** to inject the Angular $compile service and a $rootScope
    - Call $rootScope.$new() to create a new scope, then initialize it with anything expected by the directive
    - Call $compile() to create an Angular element from HTML containing the directive
    - Call $rootScope.$digest() to fire any watches
    - Assert expected results

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing a Directive

```
1     describe("bestStudent",function()
2     {
3       it("Replaces the carrying element with " +
4          "best student info",
5       function()
6       {
7         var scope = $rootScope.$new();
8
9         scope.students = [
10          {name : "Sam", gpa : 3.3},
11          {name : "Sue", gpa : 3.0} ];
12
13        var element = $compile(
14          "<best-student student-list='students'>" +
15          "</best-student>")(scope);
16
17        $rootScope.$digest();
```

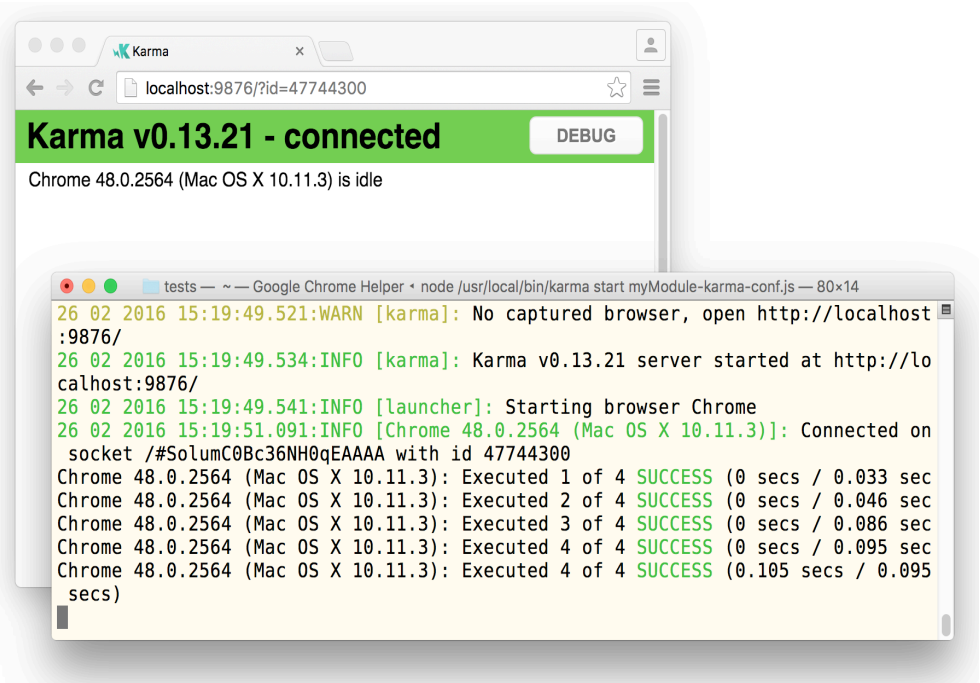

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Testing a Directive, cont'd

```
18
19        expect(element.html()).toContain(
20            "Best student: Sam, with GPA=3.3");
21      });
22    });
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example



myModule.js

Students.html

myModule-karma-conf.js

myModule-specs.js

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    (function(angular)
2    {
3        var myModule = angular.module("MyModule", [ 'ngMessages' ]);
4
5        myModule.service("myModel", function()
6        {
7            var students = [
8            {
9                name : "Sue",
10               studentID : 11,
11               gpa : 3.21
12           },
13           {
14               name : "Sam",
15               studentID : 17,
16               gpa : 3.57
17           },
18           {
19               name : "Cindy",
20               studentID : 59,
21               gpa : 3.35
22           },
23           {
24               name : "George",
25               studentID : 79,
26               gpa : 2.48
27           },
28           {
29               name : "Sarah",
30               studentID : 131,
31               gpa : 3.98
32           },
33           {
34               name : "Wilma",
35               studentID : 757,
36               gpa : 3.28
37           },
38           {
39               name : "Barney",
40               studentID : 631,
41               gpa : 3.09
42           },
43           {
44               name : "Fred",
```

```
46              gpa : 2.43
47          },
48          {
49              name : "Bessie",
50              studentID : 983,
51              gpa : 3.66
52          },
53          {
54              name : "Craig",
55              studentID : 911,
56              gpa : 3.44
57          } ]
58
59          this.getStudents = function()
60          {
61              return students;
62          };
63
64          this.addOrUpdateStudent = function(student)
65          {
66              var found = false;
67
68              for (var i = 0; i < students.length; i++)
69              {
70                  if (students[i].studentID == student.studentID)
71                  {
72                      students[i].name = student.name;
73                      students[i].gpa = student.gpa;
74                      found = true;
75                      break;
76                  }
77              }
78
79              if (!found)
80              {
81                  students.push(student);
82              }
83          }
84      });
85
86      myModule.filter("goodStudent", function()
87      {
88          return function(inputArray, threshold)
89          {
```

28 - 28B : myModule.js

```
91              if (!threshold)
92                  threshold = 3.0;
93
94              // make sure it's typed as a number
95              threshold = Number(threshold);
96
97              var outputArray = [];
98
99              // only return students whose GPA is at least
100             // the threshold
101             for (var i = 0; i < inputArray.length; i++)
102             {
103                 if (inputArray[i].gpa >= threshold)
104                     outputArray.push(inputArray[i]);
105             }
106
107             return outputArray;
108         }
109     });
110
111     myModule.directive("bestStudent", function()
112     {
113
114         var linkFunction = function(scope, element, attributes)
115         {
116             scope.theBest = {name: "no name"};
117
118             scope.$watch('studentList', function(studentList)
119             {
120                 if (studentList)
121                 {
122                     var bestGpa = 0;
123
124                     for(var i = 0; i < studentList.length; i++)
125                     {
126                         console.log(studentList[i]);
127                         if(studentList[i]. gpa > bestGpa)
128                         {
129                             bestGpa = studentList[i].gpa;
130                             scope.theBest = angular.copy(studentList[i]);
131                         }
132                     }
133                 }
134             });
```

```
136
137         var ddo =
138         {
139             restrict : "E",
140             scope :
141             {
142                 studentList : "="
143             },
144             link : linkFunction,
145             template : "<div>Best student: {{theBest.name}}, with GPA={{theBest.gpa}}</div>"
146         };
147
148         return ddo;
149     });
150
151     myModule.directive('prime', function()
152     {
153         // borrowed from:
154         // http://www.javascripter.net/faq/numberisprime.htm
155         function isPrime(n)
156         {
157             if (isNaN(n) || !isFinite(n) || n % 1 || n < 2)
158                 return false;
159             if (n % 2 == 0)
160                 return (n == 2);
161             if (n % 3 == 0)
162                 return (n == 3);
163             var m = Math.sqrt(n);
164             for (var i = 5; i <= m; i += 6)
165             {
166                 if (n % i == 0)
167                     return false;
168                 if (n % (i + 2) == 0)
169                     return false;
170             }
171             return true;
172         }
173
174         var ddo =
175         {
176             restrict : 'A',
177             require : 'ngModel',
178
179             link : function(scope, element, attrs, modelCtrl)
```

```
                modelCtrl.$validators.prime = function(modelValue, viewValue)
                {
                    return isPrime(viewValue);
                }
            }
        };

        return ddo;
    });

    myModule.controller("MyController", function(myModel, $filter,
            goodStudentFilter)
    {
        var sc = this;

        sc.currentStudent =
        {};

        sc.goodStudentsOnly = false;

        var theFilter = $filter("goodStudent");

        sc.students = function()
        {
            if (!sc.goodStudentsOnly)
            {
                return myModel.getStudents();
            }
            else
            {
                return theFilter(myModel.getStudents());
            }
        }

        sc.setCurrentStudent = function(student)
        {
            sc.currentStudent = angular.copy(student);
        }

        sc.addOrUpdateStudent = function()
        {
            myModel.addOrUpdateStudent(sc.currentStudent);
        }
    });
```

```
226  })(angular);
```

```
<!DOCTYPE html>
<html x-ng-app="MyModule">
  <head>
    <title>Students</title>
    <script src="scripts/vendor/angular.js"></script>
    <script src="scripts/vendor/angular-messages.js"></script>
    <script src="scripts/app/myModule.js"></script>
    <link href="styles/main.css" rel="stylesheet" ></link>
  </head>
  <body>
    <div x-ng-controller="MyController as mc">
      <best-student student-list="mc.students()"></best-student>
      <div>
      <form x-ng-submit="mc.addOrUpdateStudent()" novalidate name="myForm">
        <table>
          <tr>
            <td>
              <label for="studentID">Student ID:</label>
            </td>
            <td>
              <input type="number"
                x-ng-model="mc.currentStudent.studentID"
                id="studentID"
                name="studentID" required min="1" x-prime>
              <span x-ng-messages="myForm.studentID.$error">
                <span x-ng-message="required" class="errorMsg">
                  Student ID is required
                </span>
                <span x-ng-message="min" class="errorMsg">
                  Student ID must be greater or equal to 1
                </span>
                <span x-ng-message="prime" class="errorMsg">
                  The student ID must be a prime number
              </span>
              </span>
            </td>
          </tr>
          <tr>
            <td>
              <label for="name">Name:</label>
            </td>
            <td>
              <input type="text"
                x-ng-model="mc.currentStudent.name" id="name"
```

```
46              <span x-ng-messages="myForm.name.$error">
47                <span x-ng-message="required" class="errorMsg">
48                  Name is required
49                </span>
50              </span>
51            </td>
52          </tr>
53          <tr>
54            <td>
55              <label for="gpa">GPA:</label>
56            </td>
57            <td>
58              <input type="number"
59                x-ng-model="mc.currentStudent.gpa" id="gpa"
60                name="gpa" required min="0" max="4" step="0.01">
61              <span x-ng-messages="myForm.gpa.$error">
62                <span x-ng-message="required" class="errorMsg">
63                  GPA is required
64                </span>
65                <span x-ng-message="min" class="errorMsg">
66                  GPA must be greater or equal to 0
67                </span>
68                <span x-ng-message="max" class="errorMsg">
69                  GPA must be less than or equal to 4
70                </span>
71              </span>
72            </td>
73          </tr>
74        </table>
75        <input type="submit" value="Add or update"
76          x-ng-disabled="myForm.$invalid">
77      </form>
78      </div>
79      <div class="goodStudentCheck">
80        <input type="checkbox" id="goodStudents"
81          x-ng-model="mc.goodStudentsOnly">
82        <label for="goodStudents">Limit to honor roll students</label>
83      </div>
84      <div>
85        <table class="studentList">
86          <tr>
87            <td>Student ID</td>
88            <td>Name</td>
89            <td>GPA</td>
```

```
91            <tr x-ng-repeat="student in mc.students()"
92                x-ng-click="mc.setCurrentStudent(student)">
93              <td>{{student.studentID}}</td>
94              <td>{{student.name}}</td>
95              <td>{{student.gpa}}</td>
96            </tr>
97          </table>
98        </div>
99      </div>
100   </body>
101 </html>
```

```
1   // Karma configuration
2   // Generated on Fri Feb 26 2016 12:41:31 GMT-0600 (CST)
3
4   module.exports = function(config) {
5     config.set({
6
7       // base path that will be used to resolve all patterns (eg. files, exclude)
8       basePath: '',
9
10
11      // frameworks to use
12      // available frameworks: https://npmjs.org/browse/keyword/karma-adapter
13      frameworks: ['jasmine'],
14
15
16      // list of files / patterns to load in the browser
17      files: [
18        '../vendor/angular.js',
19        '../vendor/angular-messages.js',
20        '../vendor/angular-mocks.js',
21        '../app/myModule.js',
22        'myModule-specs.js'
23      ],
24
25
26      // list of files to exclude
27      exclude: [
28      ],
29
30
31      // preprocess matching files before serving them to the browser
32      // available preprocessors: https://npmjs.org/browse/keyword/karma-preprocessor
33      preprocessors: {
34      },
35
36
37      // test results reporter to use
38      // possible values: 'dots', 'progress'
39      // available reporters: https://npmjs.org/browse/keyword/karma-reporter
40      reporters: ['progress'],
41
42
43      // web server port
44      port: 9876,
```

```
46
47      // enable / disable colors in the output (reporters and logs)
48      colors: true,
49
50
51      // level of logging
52      // possible values: config.LOG_DISABLE || config.LOG_ERROR || config.LOG_WARN || config.LOG_INFO || config.LOG_DEI
53      logLevel: config.LOG_INFO,
54
55
56      // enable / disable watching file and executing tests whenever any file changes
57      autoWatch: true,
58
59
60      // start these browsers
61      // available browser launchers: https://npmjs.org/browse/keyword/karma-launcher
62      browsers: ['Chrome'],
63
64
65      // Continuous Integration mode
66      // if true, Karma captures browsers, runs the tests and exits
67      singleRun: false,
68
69      // Concurrency level
70      // how many browser should be started simultaneous
71      concurrency: Infinity
72    })
73  }
```

```
describe(
        "Testing components in myModule",
        function()
        {
            describe("MyController", function()
            {
                // reference to Angular's $controller service
                var $controller;

                // holds reference to the MyModel service
                var myModel;

                beforeEach(function()
                {
                    // load the module using Angular mock's module() function
                    module("MyModule");

                    // get a reference to Angular's $controller service
                    // using Angular mock's inject() function
                    // The inject() function will strip the underscores
                    // so we can name our variable $controller
                    inject(function(_$controller_)
                    {
                        $controller = _$controller_;
                    });

                    inject(function(_myModel_)
                    {
                        myModel = _myModel_;
                    });
                });

                describe("studentList", function()
                {
                    it("should have 10 students", function()
                    {
                        var scope =
                        {};

                        var mc = $controller('MyController as mc',
                        {
                            $scope : scope
                        });

```

```
46                  });
47              });
48          });
49
50          describe("goodStudent Filter", function()
51          {
52              // reference to Angular's $filter service
53              var $filter;
54
55              // reference to the goodStudent filter
56              var goodStudent;
57
58              beforeEach(function()
59              {
60                  module("MyModule");
61
62                  inject(function(_$filter_)
63                  {
64                      $filter = _$filter_;
65                  });
66
67                  goodStudent = $filter("goodStudent");
68              });
69
70              it("should filter students with GPA less than 3.0", function()
71              {
72                  var students = [
73                  {
74                      name : "Sue",
75                      studentID : 11,
76                      gpa : 2.21
77                  },
78                  {
79                      name : "Sam",
80                      studentID : 17,
81                      gpa : 3.57
82                  } ];
83
84                  var results = goodStudent(students);
85
86                  expect(results.length).toBe(1);
87                  expect(results[0].name).toBe("Sam");
88              });
89          });
```

```
91              describe("bestStudent Directive",
92              function()
93              {
94                  var $rootScope;
95                  var $compile;
96
97                  beforeEach(function()
98                  {
99                      module("MyModule");
100
101                     inject(function(_$compile_, _$rootScope_)
102                     {
103                         $rootScope = _$rootScope_;
104                         $compile = _$compile_;
105                     });
106                 });
107
108                 describe("bestStudent",
109                 function()
110                 {
111                     it("Replaces the carrying element with best student info",
112                     function()
113                     {
114                         var scope = $rootScope.$new();
115
116                         scope.students = [
117                         {
118                             name : "Sam",
119                             gpa : 3.3
120                         },
121                         {
122                             name : "Sue",
123                             gpa : 3.0
124                         } ];
125
126                         var element = $compile(
127                           "<best-student student-list='students'>" +
128                           " </best-student>")(scope);
129
130                         $rootScope.$digest();
131
132                         expect(element.html()).toContain(
133                            "Best student: Sam, with GPA=3.3");
134                     });
```

```
136             });
137
138             describe("prime Directive (validator)",
139             function()
140             {
141                 var $scope;
142                 var form;
143
144                 beforeEach(function()
145                 {
146                     module("MyModule");
147
148                     inject(function($compile, $rootScope)
149                     {
150                         $scope = $rootScope;
151
152                         // define an HTML snippet with an <input>
153                         // element containing the prime directive
154                         var element = angular
155                                 .element('<form name="form">'
156                                         + '<input ng-model="model.somenum" +
157                                         + "name='somenum' prime/>"
158                                         + '</form>');
159
160                         $scope.model =
161                         {
162                             somenum : null
163                         };
164
165                         // compile the element and invoke the
166                         // template function
167                         var templateFunction = $compile(element);
168                         templateFunction($rootScope);
169
170                         // save the form reference
171                         form = $scope.form;
172                     });
173                 });
174
175                 describe("prime", function()
176                 {
177                     it("should pass with a prime", function()
178                     {
179                         // simulate changing the text in the input
```

```
181                         form.somenum.$setViewValue('13');
182                         $scope.$digest();
183 
184                         expect(form.somenum.$valid).toBe(true);
185                     });
186                     it("should fail with a non-prime", function()
187                     {
188                         form.somenum.$setViewValue('12');
189                         $scope.$digest();
190                         expect(form.somenum.$valid).toBe(false);
191                     });
192                 });
193 
194             });
195         });
```

Chapter Summary

In this chapter, you learned:

- About popular JavaScript testing frameworks
- How to get started with Jasmine and Karma
- How to test Angular controllers, directives and filters

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum